Hyperparameter-tuning Cookbook

A guide for scikit-learn, PyTorch, river, and spotPython

Thomas Bartz-Beielstein

2023-06-08

Table of contents

Preface

The goal of hyperparameter tuning (or hyperparameter optimization) is to optimize the hyperparameters to improve the performance of the machine or deep learning model.

spotPython ("Sequential Parameter Optimization Toolbox in Python") is the Python version of the well-known hyperparameter tuner SPOT, which has been developed in the R programming environment for statistical analysis for over a decade. The related open-access book is available here: Hyperparameter Tuning for Machine and Deep Learning with R—A Practical Guide.

scikit-learn is a Python module for machine learning built on top of SciPy and is distributed under the 3-Clause BSD license. The project was started in 2007 by David Cournapeau as a Google Summer of Code project, and since then many volunteers have contributed.

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs.

River is a Python library for online machine learning. It is designed to be used in real-world environments, where not all data is available at once, but streaming in.

> ! Important: This book is still under development.

Citation

If this document has been useful to you and you wish to cite it in a scientific publication, please refer to the following paper:

```
@ARTICLE{bart23earxiv,
       author = {{Bartz-Beielstein}, Thomas},
        title = "{PyTorch Hyperparameter Tuning -- A Tutorial for spotPython}",
      journal = {arXiv e-prints},
     keywords = {Computer Science - Machine Learning, Computer Science - Artificial Intell
         year = 2023,
        month = may,
          eid = {arXiv:2305.11930},
        pages = {arXiv:2305.11930},
```

```
          doi = {10.48550/arXiv.2305.11930},
archivePrefix = {arXiv},
       eprint = {2305.11930},
 primaryClass = {cs.LG},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2023arXiv230511930B},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

1 Introduction: Hyperparameter Tuning

Hyperparameter tuning is an important, but often difficult and computationally intensive task. Changing the architecture of a neural network or the learning rate of an optimizer can have a significant impact on the performance.

The goal of hyperparameter tuning is to optimize the hyperparameters in a way that improves the performance of the machine learning or deep learning model. The simplest, but also most computationally expensive, approach uses manual search (or trial-and-error (Meignan et al. 2015)). Commonly encountered is simple random search, i.e., random and repeated selection of hyperparameters for evaluation, and lattice search ("grid search"). In addition, methods that perform directed search and other model-free algorithms, i.e., algorithms that do not explicitly rely on a model, e.g., evolution strategies (Bartz-Beielstein et al. 2014) or pattern search (Lewis, Torczon, and Trosset 2000) play an important role. Also, "hyperband", i.e., a multi-armed bandit strategy that dynamically allocates resources to a set of random configurations and uses successive bisections to stop configurations with poor performance (Li et al. 2016), is very common in hyperparameter tuning. The most sophisticated and efficient approaches are the Bayesian optimization and surrogate model based optimization methods, which are based on the optimization of cost functions determined by simulations or experiments.

We consider below a surrogate model based optimization-based hyperparameter tuning approach based on the Python version of the SPOT ("Sequential Parameter Optimization Toolbox") (Bartz-Beielstein, Lasarczyk, and Preuss 2005), which is suitable for situations where only limited resources are available. This may be due to limited availability and cost of hardware, or due to the fact that confidential data may only be processed locally, e.g., due to legal requirements. Furthermore, in our approach, the understanding of algorithms is seen as a key tool for enabling transparency and explainability. This can be enabled, for example, by quantifying the contribution of machine learning and deep learning components (nodes, layers, split decisions, activation functions, etc.). Understanding the importance of hyperparameters and the interactions between multiple hyperparameters plays a major role in the interpretability and explainability of machine learning models. SPOT provides statistical tools for understanding hyperparameters and their interactions. Last but not least, it should be noted that the SPOT software code is available in the open source `spotPython` package on github[1], allowing replicability of the results. This tutorial descries the Python variant of SPOT, which is called

[1] https://github.com/sequential-parameter-optimization

`spotPython`. The R implementation is described in Bartz et al. (2022). SPOT is an established open source software that has been maintained for more than 15 years (Bartz-Beielstein, Lasarczyk, and Preuss 2005) (Bartz et al. 2022).

This tutorial is structured as follows. The concept of the hyperparameter tuning software `spotPython` is described in Section 1.1. Chapter 19 describes the execution of the example from the tutorial "Hyperparameter Tuning with Ray Tune" (PyTorch 2023a). The integration of `spotPython` into the `PyTorch` training workflow is described in detail in the following sections. Section 19.1 describes the setup of the tuners. Section 19.3 describes the data loading. Section 19.4 describes the model to be tuned. The search space is introduced in Section 19.5. Optimizers are presented in Section 19.6. How to split the data in train, validation, and test sets is described in Section 19.7. The selection of the loss function and metrics is described in Section 19.8. @#sec-prepare-spot-call describes the preparation of the `spotPython` call. The objective function is described in Section 19.10. How to use results from previous runs and default hyperparameter configurations is described in Section 19.11. Starting the tuner is shown in Section 19.12. TensorBoard can be used to visualize the results as shown in Section 19.13. Results are discussed and explained in Section 19.14 Finally, Section 19.15 presents a summary and an outlook.

> **Note**
>
> The corresponding `.ipynb` notebook (Bartz-Beielstein 2023) is updated regularly and reflects updates and changes in the `spotPython` package. It can be downloaded from https://github.com/sequential-parameter-optimization/spotPython/blob/main/notebooks/14_spot_ray_hpt_torch_cifar10.ipynb.

1.1 The Hyperparameter Tuning Software SPOT

Surrogate model based optimization methods are common approaches in simulation and optimization. SPOT was developed because there is a great need for sound statistical analysis of simulation and optimization algorithms. SPOT includes methods for tuning based on classical regression and analysis of variance techniques. It presents tree-based models such as classification and regression trees and random forests as well as Bayesian optimization (Gaussian process models, also known as Kriging). Combinations of different meta-modeling approaches are possible. SPOT comes with a sophisticated surrogate model based optimization method, that can handle discrete and continuous inputs. Furthermore, any model implemented in `scikit-learn` can be used out-of-the-box as a surrogate in `spotPython`.

SPOT implements key techniques such as exploratory fitness landscape analysis and sensitivity analysis. It can be used to understand the performance of various algorithms, while simultaneously giving insights into their algorithmic behavior. In addition, SPOT can be used as an

optimizer and for automatic and interactive tuning. Details on SPOT and its use in practice are given by Bartz et al. (2022).

A typical hyperparameter tuning process with `spotPython` consists of the following steps:

1. Loading the data (training and test datasets), see Section 19.3.
2. Specification of the preprocessing model, see Section 19.4.1. This model is called `prep_model` ("preparation" or pre-processing). The information required for the hyperparameter tuning is stored in the dictionary `fun_control`. Thus, the information needed for the execution of the hyperparameter tuning is available in a readable form.
3. Selection of the machine learning or deep learning model to be tuned, see Section 19.4.2. This is called the `core_model`. Once the `core_model` is defined, then the associated hyperparameters are stored in the `fun_control` dictionary. First, the hyperparameters of the `core_model` are initialized with the default values of the `core_model`. As default values we use the default values contained in the `spotPython` package for the algorithms of the `torch` package.
4. Modification of the default values for the hyperparameters used in `core_model`, see Section 19.5.3.1. This step is optional.
    1. numeric parameters are modified by changing the bounds.
    2. categorical parameters are modified by changing the categories ("levels").
5. Selection of target function (loss function) for the optimizer, see Section 19.8.
6. Calling SPOT with the corresponding parameters, see Section 19.12. The results are stored in a dictionary and are available for further analysis.
7. Presentation, visualization and interpretation of the results, see Section 19.14.

1.2 Spot as an Optimizer

The `spot` loop consists of the following steps:

1. Init: Build initial design X
2. Evaluate initial design on real objective f: $y = f(X)$
3. Build surrogate: $S = S(X, y)$
4. Optimize on surrogate: $X_0 = \text{optimize}(S)$
5. Evaluate on real objective: $y_0 = f(X_0)$
6. Impute (Infill) new points: $X = X \cup X_0$, $y = y \cup y_0$.
7. Got 3.

Central Idea: Evaluation of the surrogate model `S` is much cheaper (or / and much faster) than running the real-world experiment f. We start with a small example.

1.3 Example: `Spot` and the Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

1.3.1 The Objective Function: Sphere

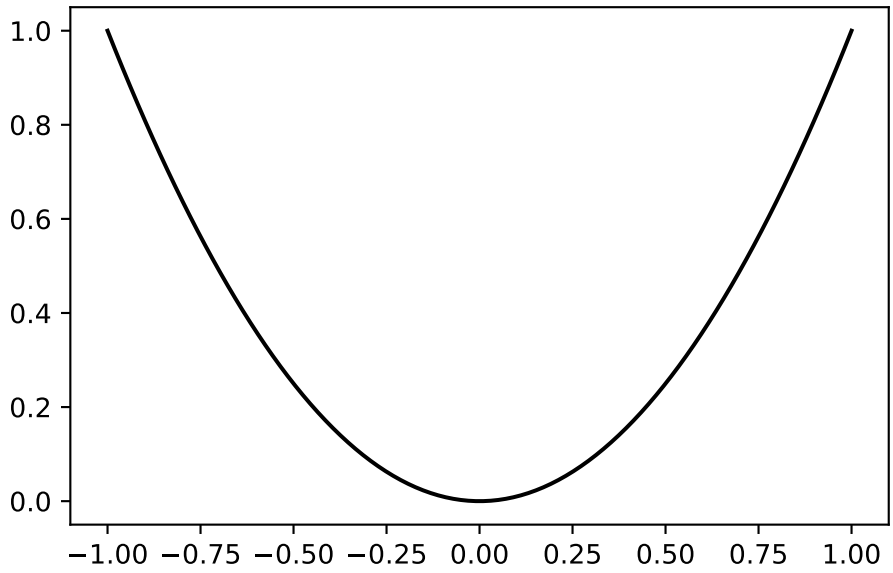
The `spotPython` package provides several classes of objective functions. We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x) = x^2$$

```
fun = analytical().fun_sphere
```

We can apply the function `fun` to input values and plot the result:

```
x = np.linspace(-1,1,100).reshape(-1,1)
y = fun(x)
plt.figure()
plt.plot(x, y, "k")
plt.show()
```




```
spot_0 = spot.Spot(fun=fun,
                   lower = np.array([-1]),
                   upper = np.array([1]))
```

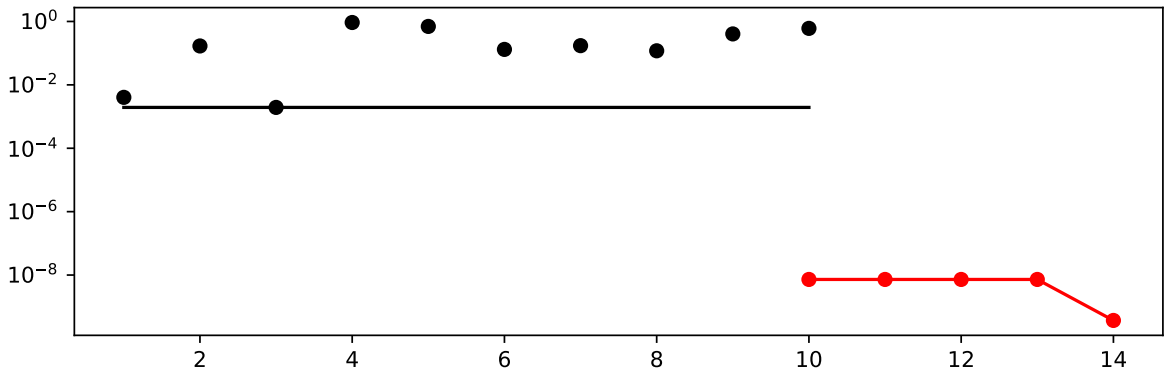
```
spot_0.run()
```

```
<spotPython.spot.spot.Spot at 0x14fa6f910>
```

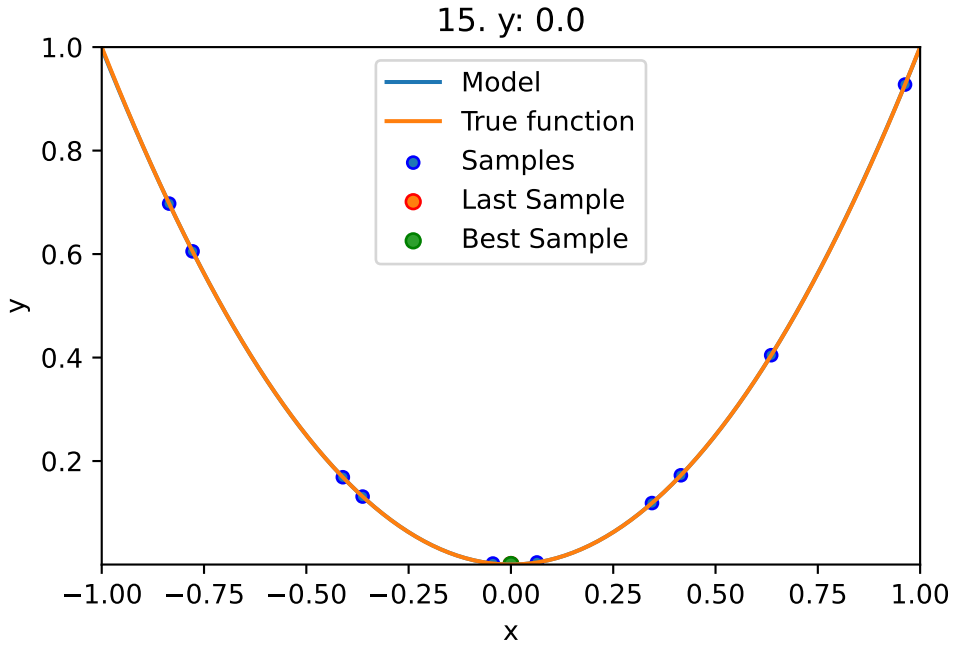
```
spot_0.print_results()
```

```
min y: 3.696886711914087e-10
x0: 1.922728975158508e-05

[['x0', 1.922728975158508e-05]]
```

```
spot_0.plot_progress(log_y=True)
```




```
spot_0.plot_model()
```



1.4 Spot Parameters: `fun_evals`, `init_size` and `show_models`

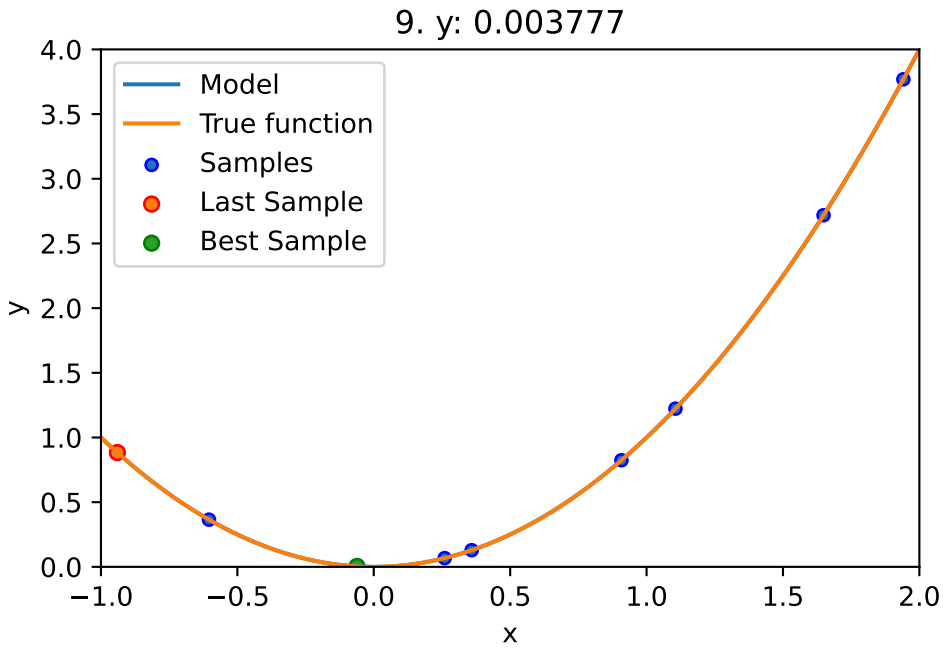
We will modify three parameters:

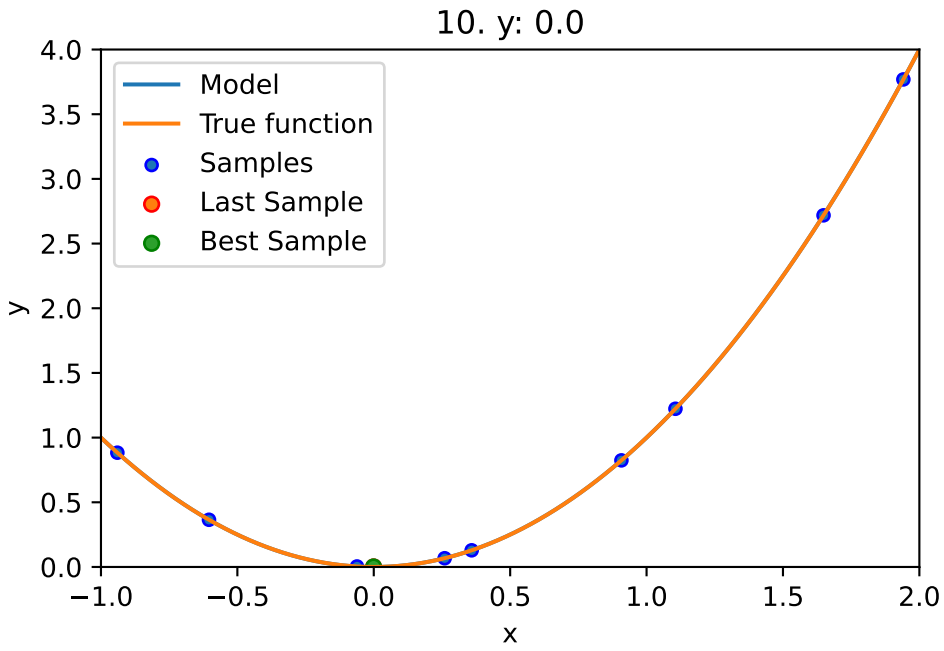
1. The number of function evaluations (`fun_evals`)
2. The size of the initial design (`init_size`)

3. The parameter `show_models`, which visualizes the search process for 1-dim functions.

The full list of the `Spot` parameters is shown in the Help System and in the notebook `spot_doc.ipynb`.

```
spot_1 = spot.Spot(fun=fun,
                   lower = np.array([-1]),
                   upper = np.array([2]),
                   fun_evals= 10,
                   seed=123,
                   show_models=True,
                   design_control={"init_size": 9})
spot_1.run()
```






```
<spotPython.spot.spot.Spot at 0x15faa3220>
```

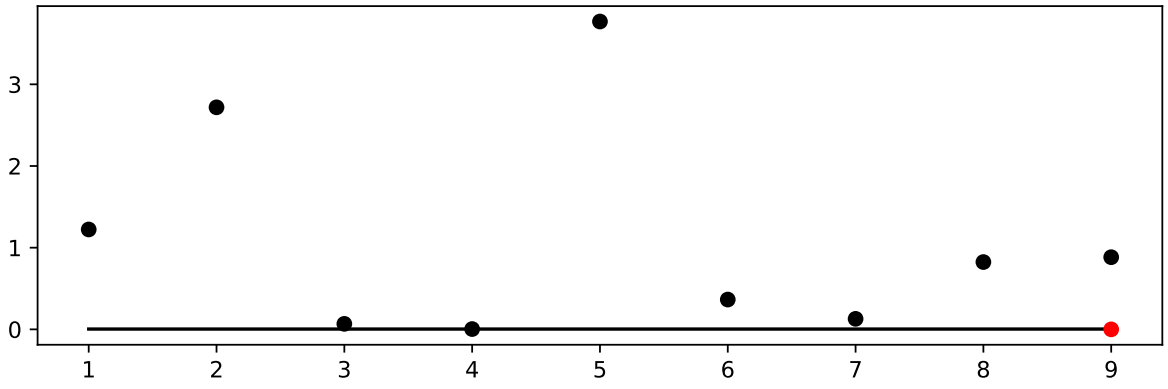
1.5 Print the Results

```
spot_1.print_results()
```

```
min y: 3.6779240309761575e-07
x0: -0.0006064589047063418

[['x0', -0.0006064589047063418]]
```

1.6 Show the Progress

```
spot_1.plot_progress()
```


3
2
1
0
1
2
3
4
5
6
7
8
9

2 Multi-dimensional Functions

This notebook illustrates how high-dimensional functions can be analyzed.

2.1 Example: `Spot` and the 3-dim Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
import pylab
from numpy import append, ndarray, multiply, isinf, linspace, meshgrid, ravel
from numpy import array
```

2.1.1 The Objective Function: 3-dim Sphere

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x) = \sum_{i}^{n} x_i^2$$

- Here we will use $n = 3$.

```
fun = analytical().fun_sphere
```

- The size of the `lower` bound vector determines the problem dimension.
- Here we will use `np.array([-1, -1, -1])`, i.e., a three-dim function.

- We will use three different `theta` values (one for each dimension), i.e., we set `surrogate_control={"n_theta": 3}`.

```
spot_3 = spot.Spot(fun=fun,
                   lower = -1.0*np.ones(3),
                   upper = np.ones(3),
                   var_name=["Pressure", "Temp", "Lambda"],
                   show_progress=True,
                   surrogate_control={"n_theta": 3})

spot_3.run()
```

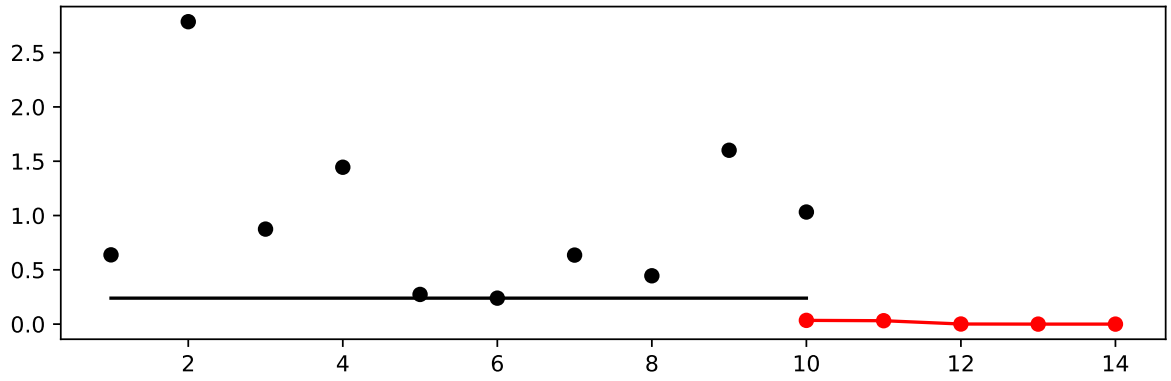
```
spotPython tuning: 0.03443344056467332 [#######---] 73.33%

spotPython tuning: 0.03134865993507926 [########--] 80.00%

spotPython tuning: 0.0009629342967936851 [#########-] 86.67%

spotPython tuning: 8.541951463966474e-05 [#########-] 93.33%

spotPython tuning: 6.285135731399678e-05 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x10635ca30>
```

2.1.2 Results

```
spot_3.print_results()
```

```
min y: 6.285135731399678e-05
Pressure: 0.005236109709736696
Temp: 0.0019572552655686714
Lambda: 0.005621713639718905

[['Pressure', 0.005236109709736696],
 ['Temp', 0.0019572552655686714],
 ['Lambda', 0.005621713639718905]]
```

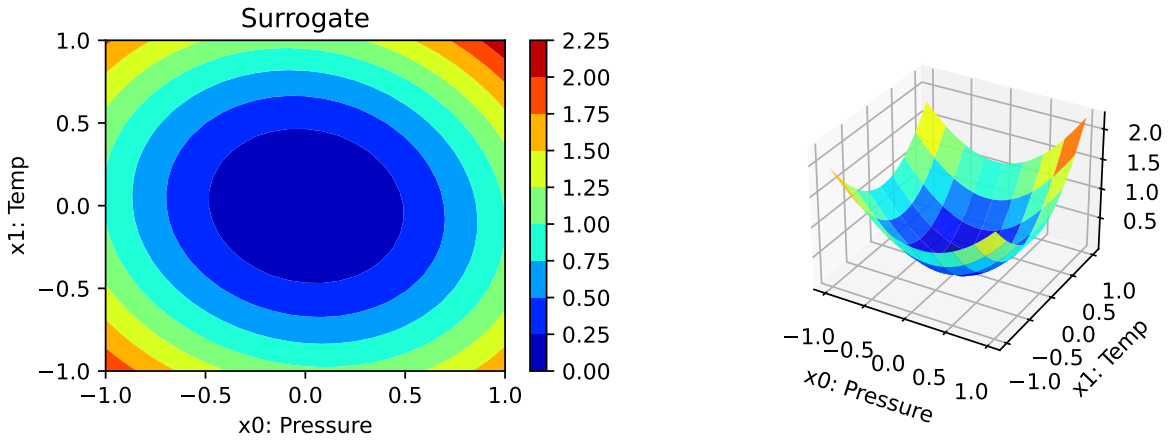
```
spot_3.plot_progress()
```



2.1.3 A Contour Plot

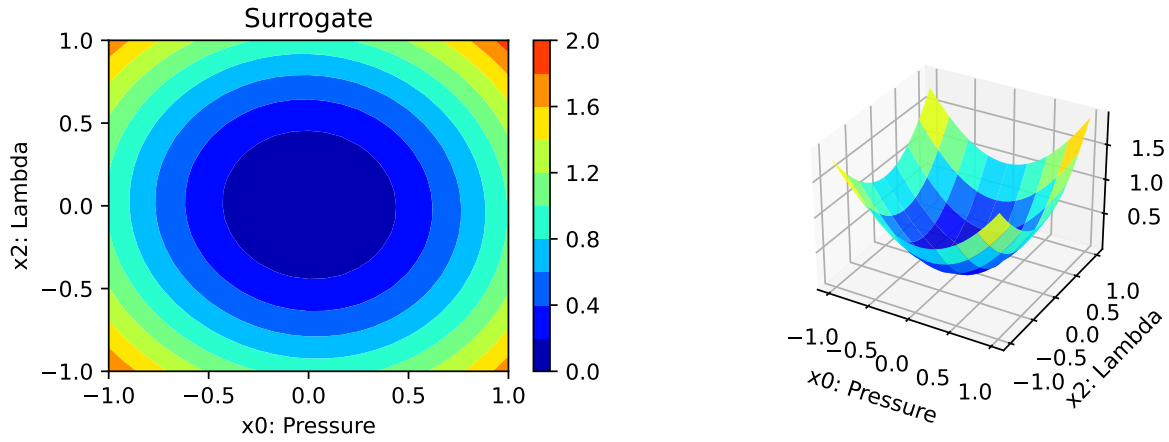
- We can select two dimensions, say $i = 0$ and $j = 1$, and generate a contour plot as follows.
  - Note: We have specified identical `min_z` and `max_z` values to generate comparable plots!

```
spot_3.plot_contour(i=0, j=1, min_z=0, max_z=2.25)
```



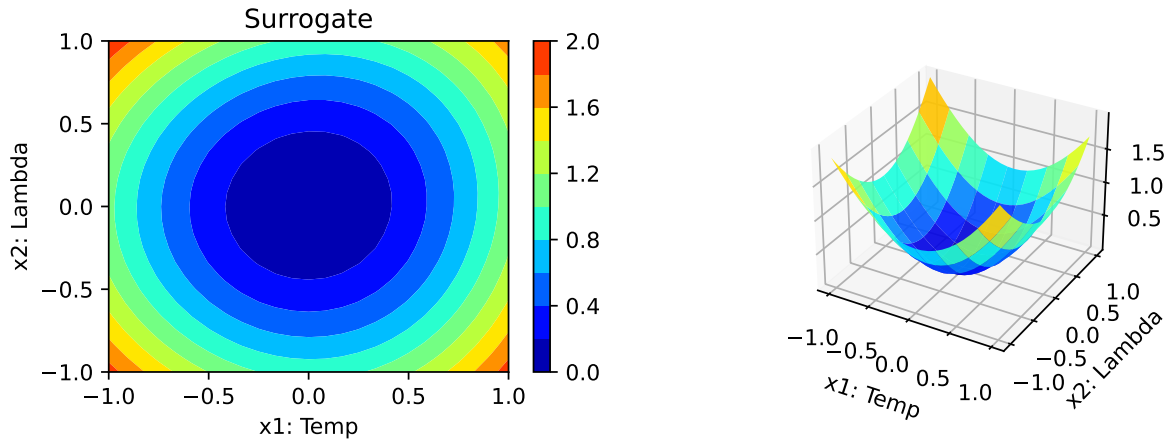
- In a similar manner, we can plot dimension $i = 0$ and $j = 2$:

```
spot_3.plot_contour(i=0, j=2, min_z=0, max_z=2.25)
```



- The final combination is $i = 1$ and $j = 2$:

```
spot_3.plot_contour(i=1, j=2, min_z=0, max_z=2.25)
```



- The three plots look very similar, because the `fun_sphere` is symmetric.
- This can also be seen from the variable importance:

```
spot_3.print_importance()
```

```
Pressure:  99.35185545837122
Temp:  99.99999999999999
```

```
Lambda:  94.31627052007231
```

```
[['Pressure', 99.35185545837122],
 ['Temp', 99.99999999999999],
 ['Lambda', 94.31627052007231]]
```

2.2 Conclusion

Based on this quick analysis, we can conclude that all three dimensions are equally important (as expected, because the analytical function is known).

2.3 Exercises

- Important:
  - Results from these exercises should be added to this document, i.e., you should submit an updated version of this notebook.
  - Please combine your results using this notebook.
  - Only one notebook from each group!
  - Presentation is based on this notebook. No addtional slides are required!
  - spotPython version `0.16.11` (or greater) is required

2.3.1 The Three Dimensional `fun_cubed`

- The input dimension is `3`. The search range is $-1 \leq x \leq 1$ for all dimensions.
- Generate contour plots
- Calculate the variable importance.
- Discuss the variable importance:
  - Are all variables equally important?
  - If not:
    - Which is the most important variable?
    - Which is the least important variable?

2.3.2 The Ten Dimensional `fun_wing_wt`

- The input dimension is `10`. The search range is $0 \leq x \leq 1$ for all dimensions.
- Calculate the variable importance.
- Discuss the variable importance:
  - Are all variables equally important?
  - If not:
    - Which is the most important variable?
    - Which is the least important variable?
  - Generate contour plots for the three most important variables. Do they confirm your selection?

2.3.3 The Three Dimensional `fun_runge`

- The input dimension is `3`. The search range is $-5 \leq x \leq 5$ for all dimensions.
- Generate contour plots
- Calculate the variable importance.
- Discuss the variable importance:
  - Are all variables equally important?
  - If not:
    - Which is the most important variable?
    - Which is the least important variable?

2.3.4 The Three Dimensional `fun_linear`

- The input dimension is `3`. The search range is $-5 \leq x \leq 5$ for all dimensions.
- Generate contour plots
- Calculate the variable importance.
- Discuss the variable importance:
  - Are all variables equally important?
  - If not:
    - Which is the most important variable?
    - Which is the least important variable?

3 Isotropic and Anisotropic Kriging

3.1 Example: Isotropic `Spot` Surrogate and the 2-dim Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

3.1.1 The Objective Function: 2-dim Sphere

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x, y) = x^2 + y^2$$

```
fun = analytical().fun_sphere
fun_control = {"sigma": 0,
               "seed": 123}
```

- The size of the `lower` bound vector determines the problem dimension.
- Here we will use `np.array([-1, -1])`, i.e., a two-dim function.

```
spot_2 = spot.Spot(fun=fun,
                   lower = np.array([-1, -1]),
                   upper = np.array([1, 1]))

spot_2.run()
```

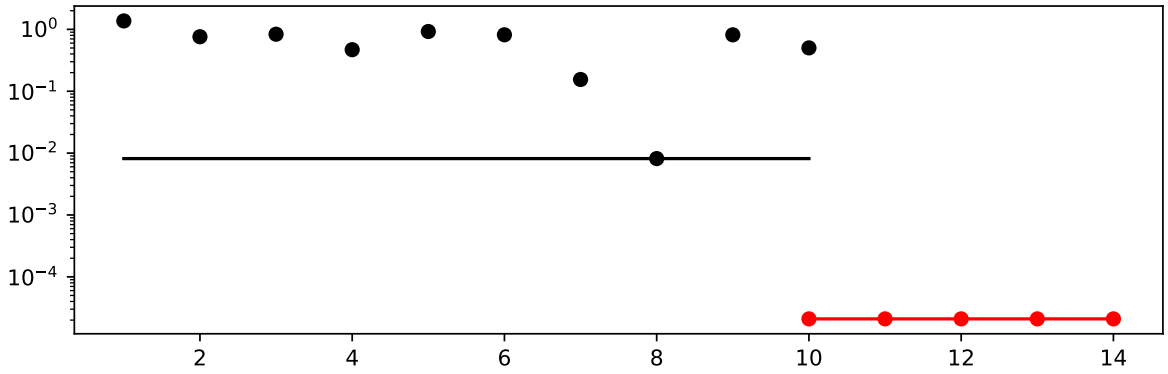
```
<spotPython.spot.spot.Spot at 0x2924d3a90>
```

3.1.2 Results

```
spot_2.print_results()
```

```
min y: 2.093282610941807e-05
x0: 0.0016055267473267492
x1: 0.00428428640184529

[['x0', 0.0016055267473267492], ['x1', 0.00428428640184529]]
```

```
spot_2.plot_progress(log_y=True)
```



3.2 Example With Anisotropic Kriging

- The default parameter setting of `spotPython`'s Kriging surrogate uses the same `theta` value for every dimension.
- This is referred to as "using an isotropic kernel".
- If different `theta` values are used for each dimension, then an anisotropic kernel is used
- To enable anisotropic models in `spotPython`, the number of `theta` values should be larger than one.
- We can use `surrogate_control={"n_theta": 2}` to enable this behavior (2 is the problem dimension).

```
spot_2_anisotropic = spot.Spot(fun=fun,
                    lower = np.array([-1, -1]),
                    upper = np.array([1, 1]),
                    surrogate_control={"n_theta": 2})
spot_2_anisotropic.run()
```

```
<spotPython.spot.spot.Spot at 0x29733b640>
```

3.2.1 Taking a Look at the `theta` Values

- We can check, whether one or several `theta` values were used.
- The `theta` values from the surrogate can be printed as follows:

```
spot_2_anisotropic.surrogate.theta
```

```
array([0.19447342, 0.30813872])
```

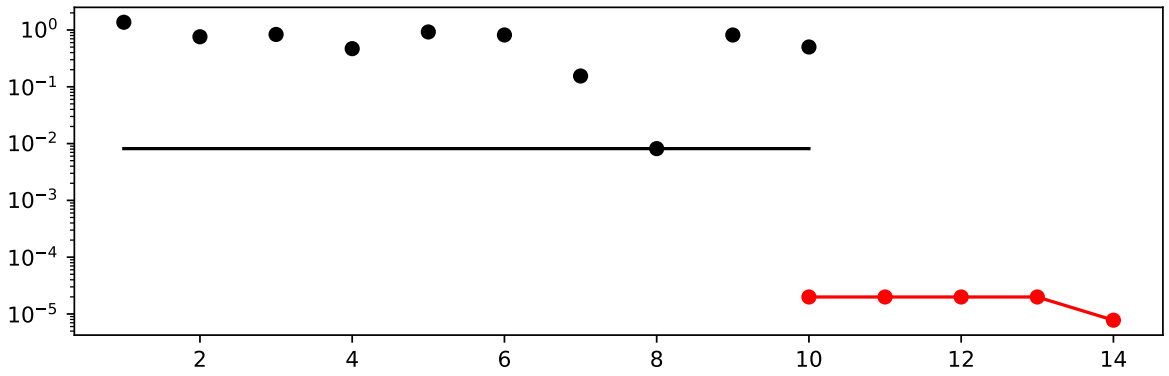
- Since the surrogate from the isotropic setting was stored as `spot_2`, we can also take a look at the `theta` value from this model:

```
spot_2.surrogate.theta
```

```
array([0.26287447])
```

- Next, the search progress of the optimization with the anisotropic model can be visualized:

```
spot_2_anisotropic.plot_progress(log_y=True)
```

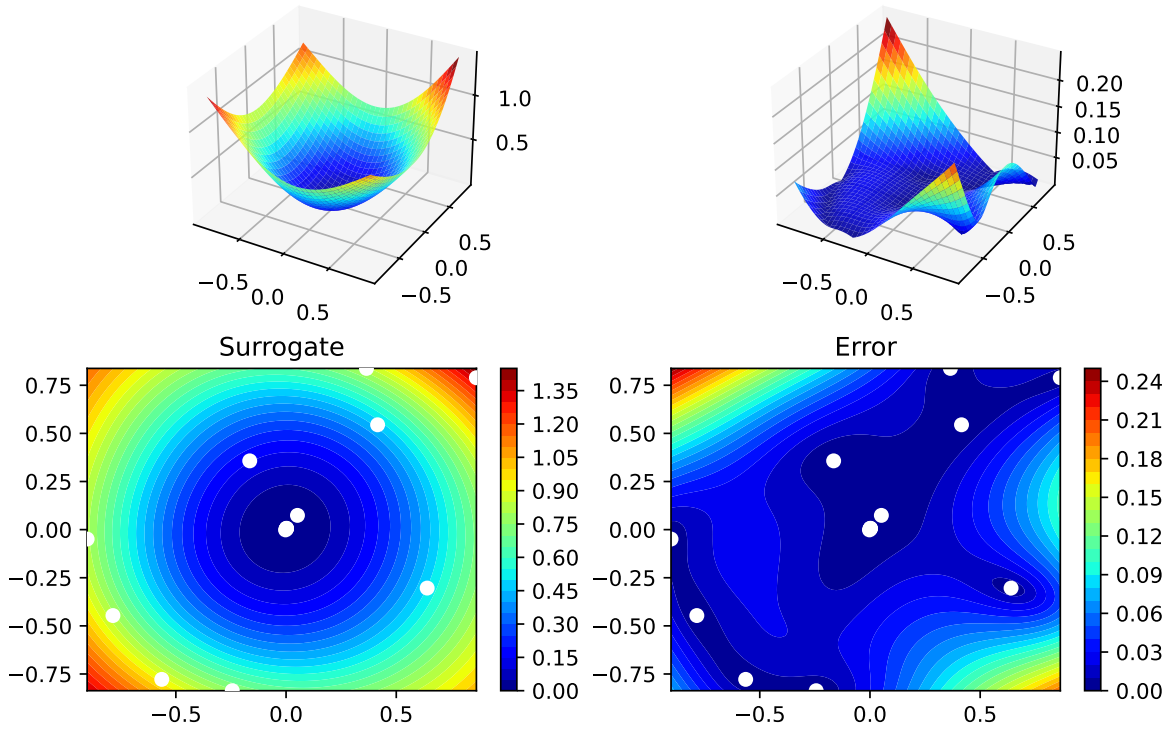


```
spot_2_anisotropic.print_results()
```

```
min y: 7.77061191821505e-06
x0: -0.0024488252797500764
x1: -0.0013318658594137815
```

```
[['x0', -0.0024488252797500764], ['x1', -0.0013318658594137815]]
```

```
spot_2_anisotropic.surrogate.plot()
```



4 Exercises

4.1 `fun_branin`

- Describe the function.
  - The input dimension is 2. The search range is $-5 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 15$.
- Compare the results from `spotPython` run a) with isotropic and b) anisotropic surrogate models.
- Modify the termination criterion: instead of the number of evaluations (which is specified via `fun_evals`), the time should be used as the termination criterion. This can be done as follows (`max_time=1` specifies a run time of one minute):

```
fun_evals=inf,
max_time=1,
```

4.2 `fun_sin_cos`

- Describe the function.
  - The input dimension is 2. The search range is $-2\pi \leq x_1 \leq 2\pi$ and $-2\pi \leq x_2 \leq 2\pi$.
- Compare the results from `spotPython` run a) with isotropic and b) anisotropic surrogate models.
- Modify the termination criterion (`max_time` instead of `fun_evals`) as described for `fun_branin`.

4.3 `fun_runge`

- Describe the function.
  - The input dimension is 2. The search range is $-5 \leq x_1 \leq 5$ and $-5 \leq x_2 \leq 5$.
- Compare the results from `spotPython` run a) with isotropic and b) anisotropic surrogate models.

- Modify the termination criterion (`max_time` instead of `fun_evals`) as described for `fun_branin`.

4.4 `fun_wingwt`

- Describe the function.
  - The input dimension is `10`. The search ranges are between 0 and 1 (values are mapped internally to their natural bounds).
- Compare the results from `spotPython` run a) with isotropic and b) anisotropic surrogate models.
- Modify the termination criterion (`max_time` instead of `fun_evals`) as described for `fun_branin`.

5 Using sklearn Surrogates in spotPython

This notebook explains how different surrogate models from `scikit-learn` can be used as surrogates in `spotPython` optimization runs.

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

5.1 Example: Branin Function with spotPython's Internal Kriging Surrogate

5.1.1 The Objective Function Branin

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula.
- Here we will use the Branin function:

  ```
  y = a * (x2 - b * x1**2 + c * x1 - r) ** 2 + s * (1 - t) * np.cos(x1) + s,
  where values of a, b, c, r, s and t are: a = 1, b = 5.1 / (4*pi**2),
  c = 5 / pi, r = 6, s = 10 and t = 1 / (8*pi).
  ```

- It has three global minima:

  ```
  f(x) = 0.397887 at (-pi, 12.275), (pi, 2.275), and (9.42478, 2.475).
  ```

```
from spotPython.fun.objectivefunctions import analytical
lower = np.array([-5,-0])
```

```
upper = np.array([10,15])
```

```
fun = analytical().fun_branin
```

5.1.2 Running the surrogate model based optimizer `Spot`:

```
spot_2 = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = 20,
                   max_time = inf,
                   seed=123,
                   design_control={"init_size": 10})
```

```
spot_2.run()
```

```
<spotPython.spot.spot.Spot at 0x10455d780>
```

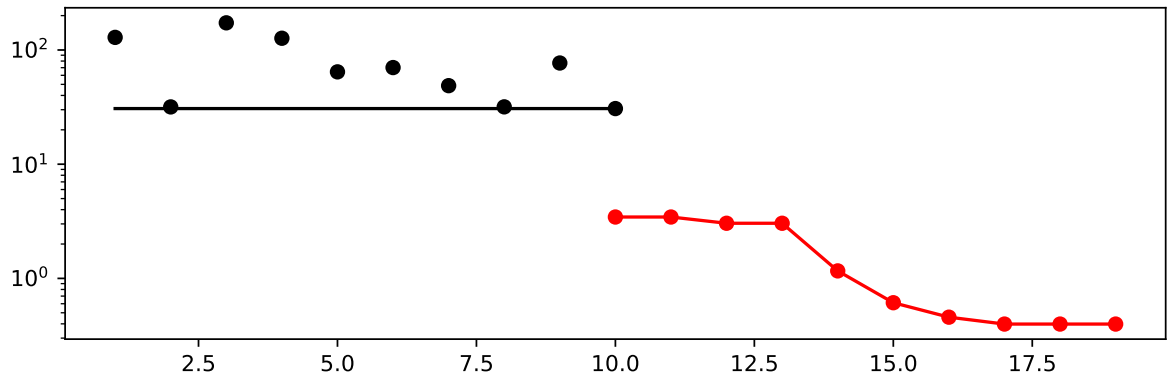
5.1.3 Print the Results

```
spot_2.print_results()
```

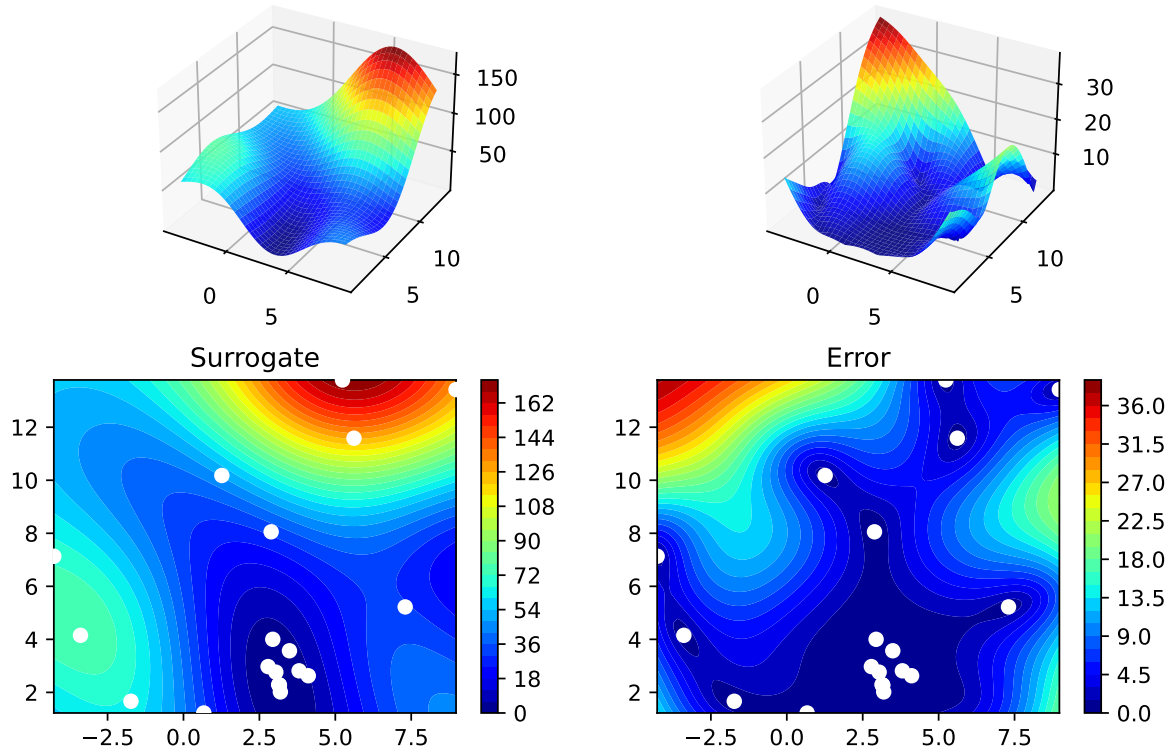
```
min y: 0.3982295132785083
x0: 3.135528626303215
x1: 2.2926027772585886

[['x0', 3.135528626303215], ['x1', 2.2926027772585886]]
```

5.1.4 Show the Progress and the Surrogate

```
spot_2.plot_progress(log_y=True)
```




```
spot_2.surrogate.plot()
```



5.2 Example: Using Surrogates From scikit-learn

- Default is the `spotPython` (i.e., the internal) `kriging` surrogate.

- It can be called explicitely and passed to `Spot`.

```
from spotPython.build.kriging import Kriging
S_0 = Kriging(name='kriging', seed=123)
```

- Alternatively, models from `scikit-learn` can be selected, e.g., Gaussian Process, RBFs, Regression Trees, etc.

```
# Needed for the sklearn surrogates:
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model
from sklearn import tree
import pandas as pd
```

- Here are some additional models that might be useful later:

```
S_Tree = DecisionTreeRegressor(random_state=0)
S_LM = linear_model.LinearRegression()
S_Ridge = linear_model.Ridge()
S_RF = RandomForestRegressor(max_depth=2, random_state=0)
```

5.2.1 GaussianProcessRegressor as a Surrogate

- To use a Gaussian Process model from `sklearn`, that is similar to `spotPython`'s `Kriging`, we can proceed as follows:

```
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
S_GP = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
```

- The scikit-learn GP model `S_GP` is selected for `Spot` as follows:

  `surrogate = S_GP`

- We can check the kind of surogate model with the command `isinstance`:

```
isinstance(S_GP, GaussianProcessRegressor)
```

```
True
```

```
isinstance(S_0, Kriging)
```

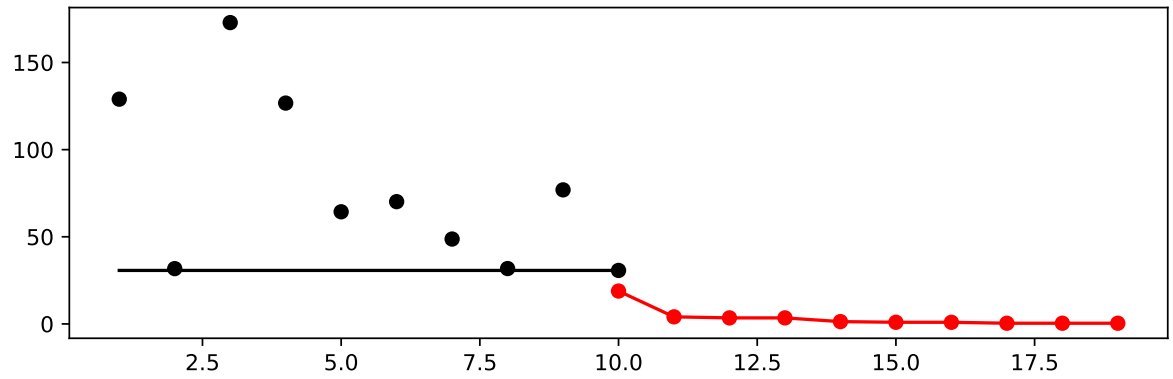
```
True
```

- Similar to the `Spot` run with the internal `Kriging` model, we can call the run with the `scikit-learn` surrogate:

```
fun = analytical(seed=123).fun_branin
spot_2_GP = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = 20,
                    seed=123,
                    design_control={"init_size": 10},
                    surrogate = S_GP)
spot_2_GP.run()
```

```
<spotPython.spot.spot.Spot at 0x163266ad0>
```

```
spot_2_GP.plot_progress()
```




```
spot_2_GP.print_results()
```

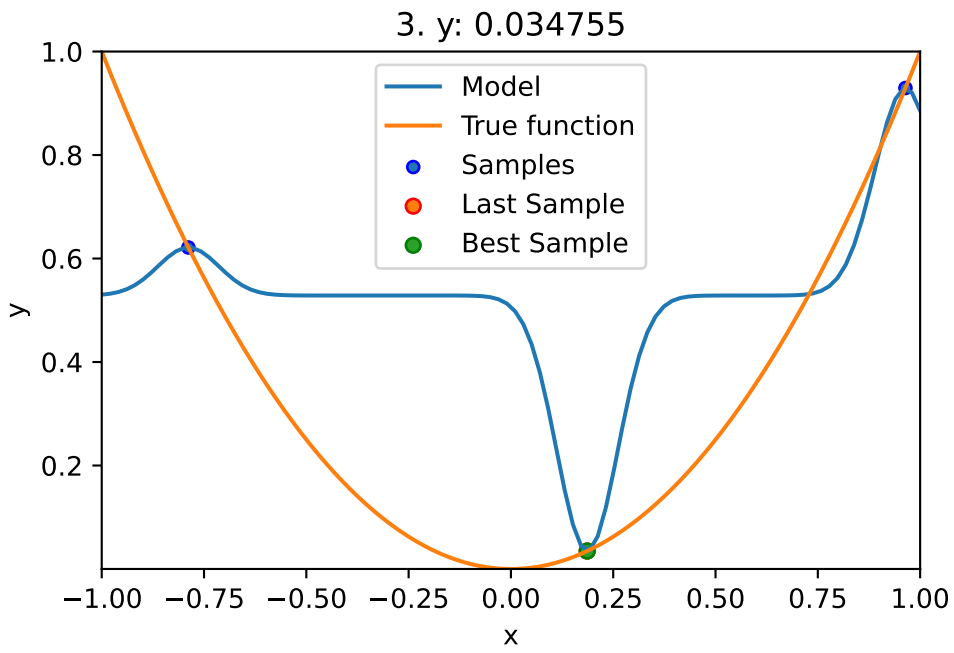
```
min y: 0.398231180851484
x0: 3.1491730713676622
x1: 2.27733849605692

[['x0', 3.1491730713676622], ['x1', 2.27733849605692]]
```

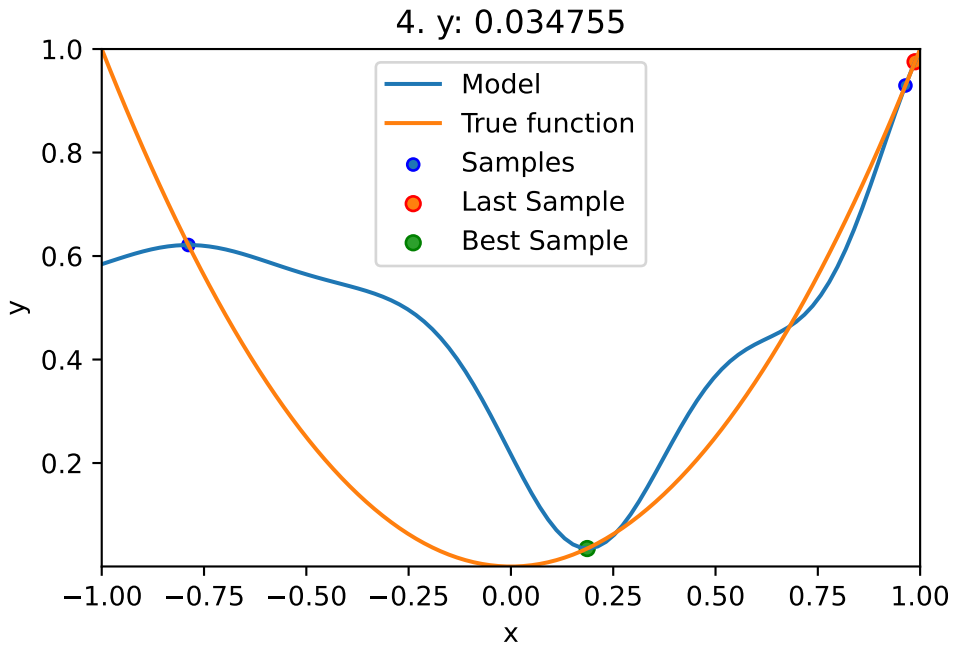
6 Example: One-dimensional Sphere Function With spotPython's Kriging

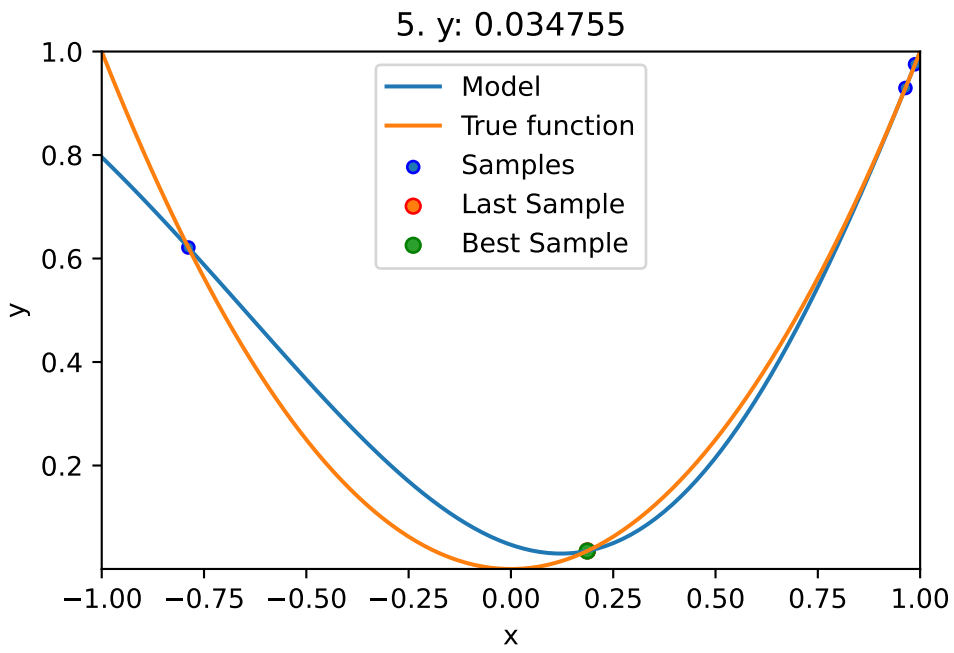
- In this example, we will use an one-dimensional function, which allows us to visualize the optimization process.
  - `show_models= True` is added to the argument list.

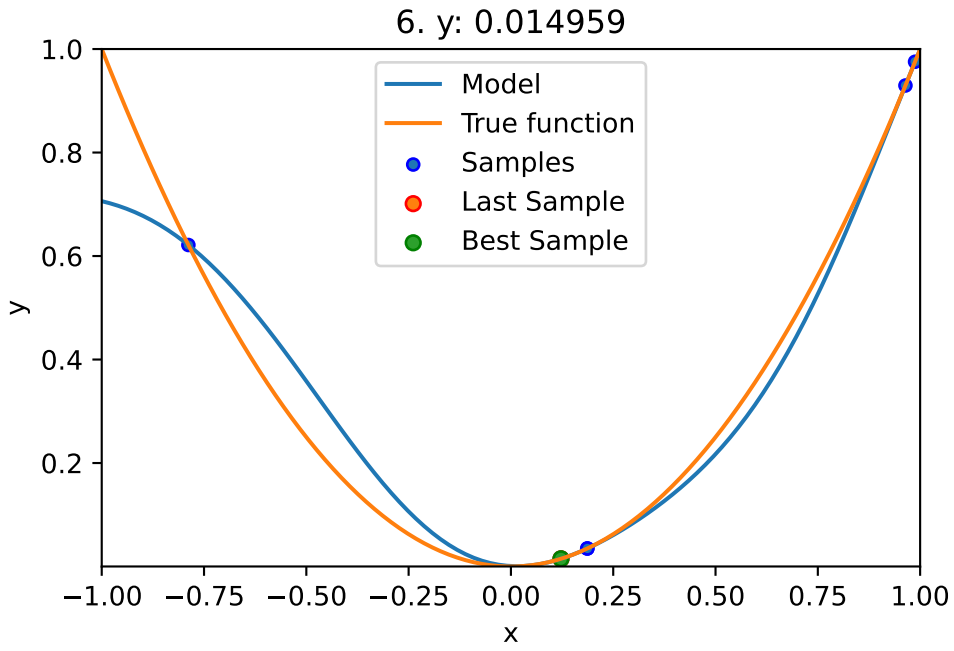
```
from spotPython.fun.objectivefunctions import analytical
lower = np.array([-1])
upper = np.array([1])
fun = analytical(seed=123).fun_sphere
```

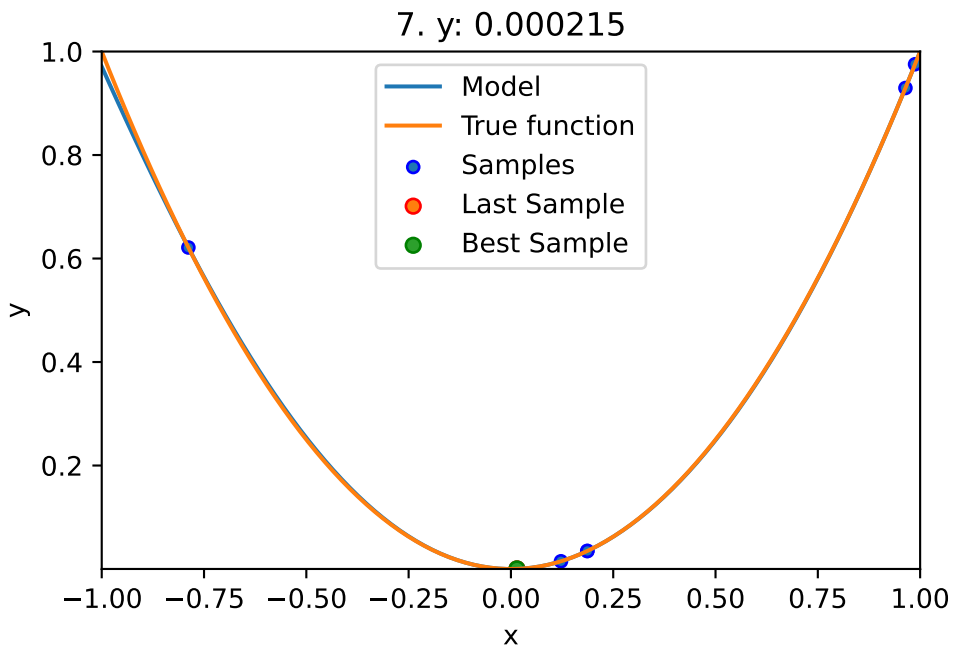
```
spot_1 = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = 10,
                   max_time = inf,
                   seed=123,
                   show_models= True,
                   tolerance_x = np.sqrt(np.spacing(1)),
                   design_control={"init_size": 3},)
spot_1.run()
```

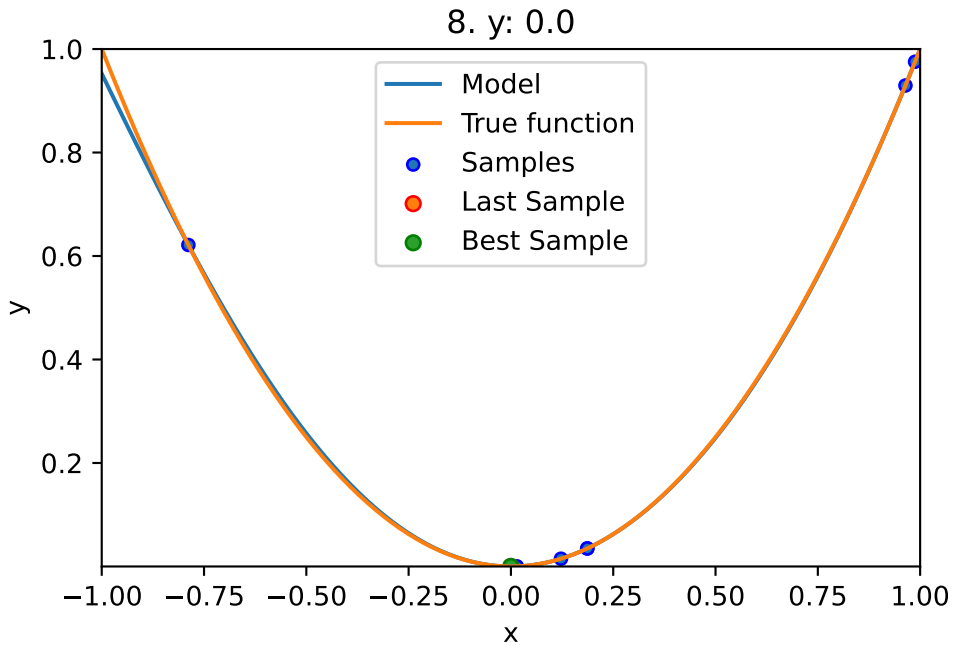

3. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

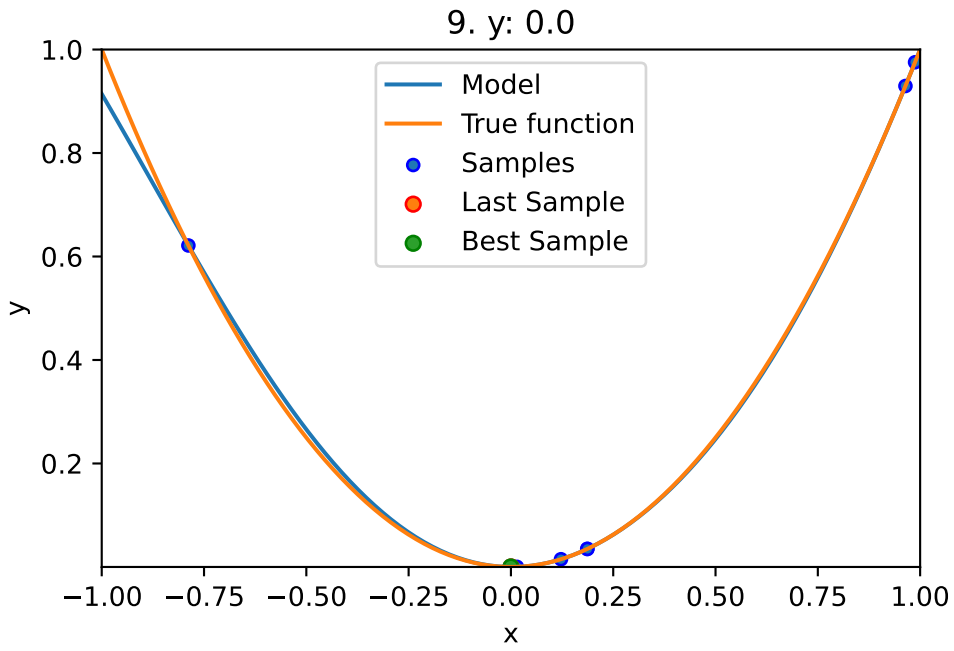

4. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

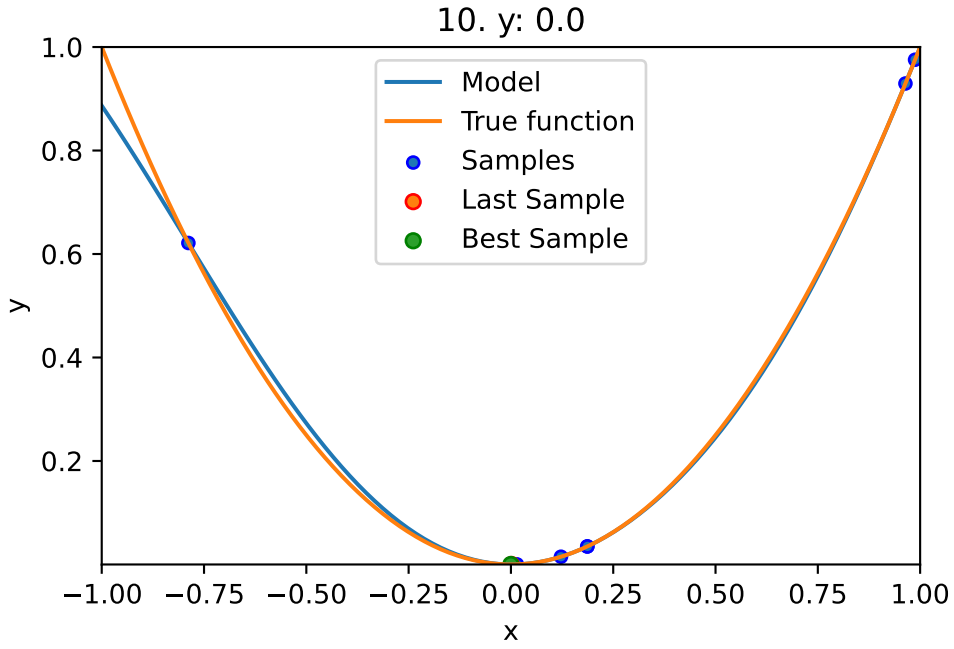

5. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


6. y: 0.014959
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


7. y: 0.000215
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


8. y: 0.0
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x





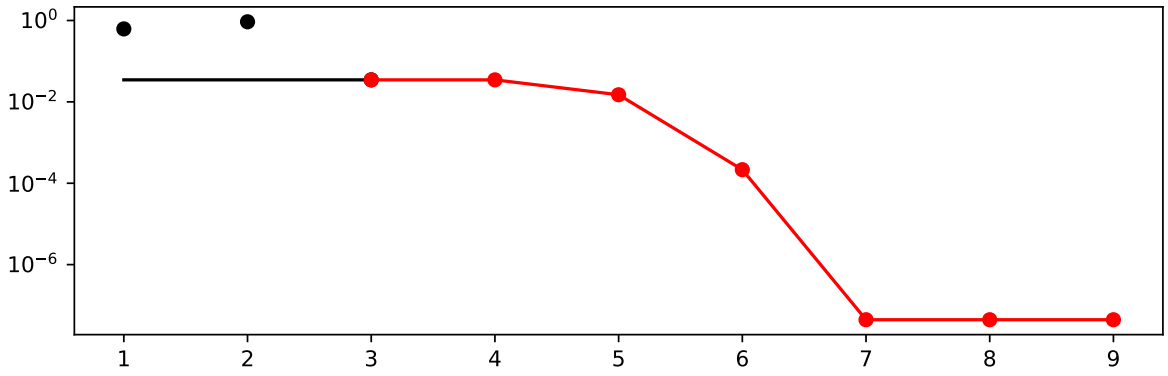

```
<spotPython.spot.spot.Spot at 0x163267610>
```

6.0.1 Results

```
spot_1.print_results()
```

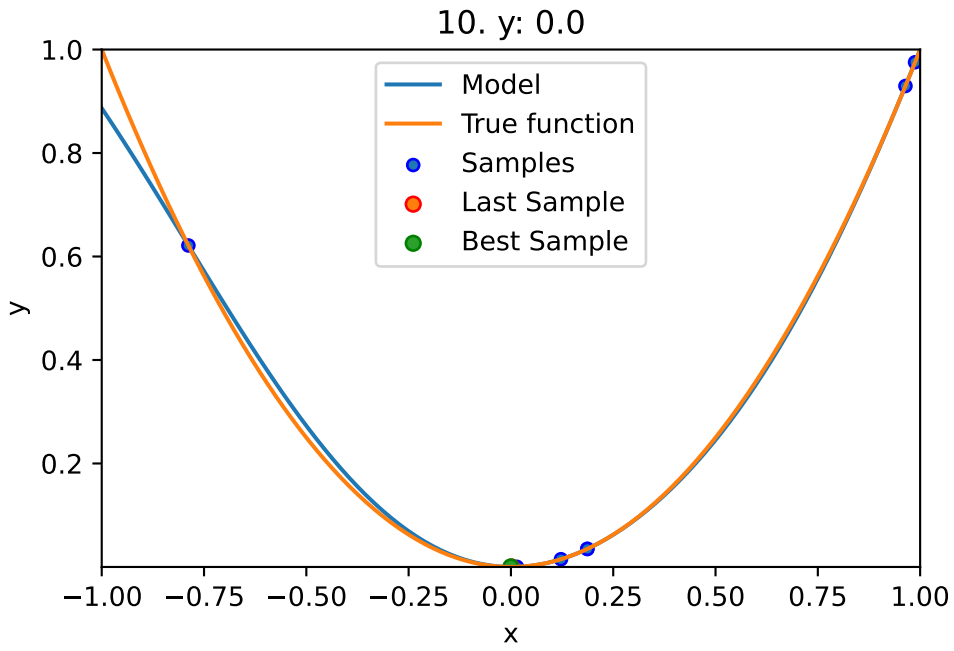
```
min y: 4.41925228274096e-08
x0: -0.00021022017702259125

[['x0', -0.00021022017702259125]]
```

```
spot_1.plot_progress(log_y=True)
```



- The method `plot_model` plots the final surrogate:

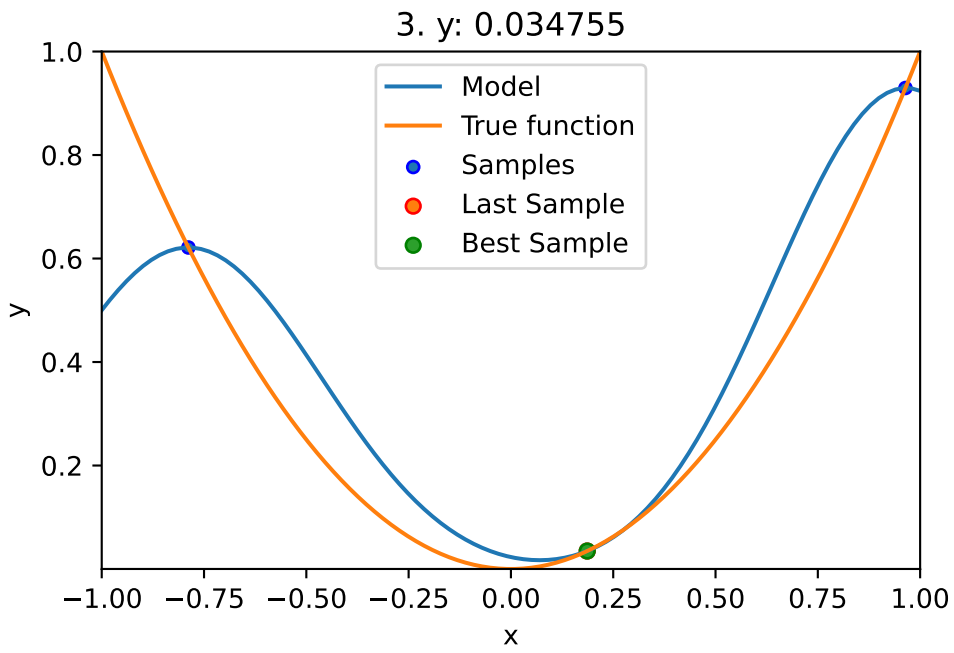
```
spot_1.plot_model()
```

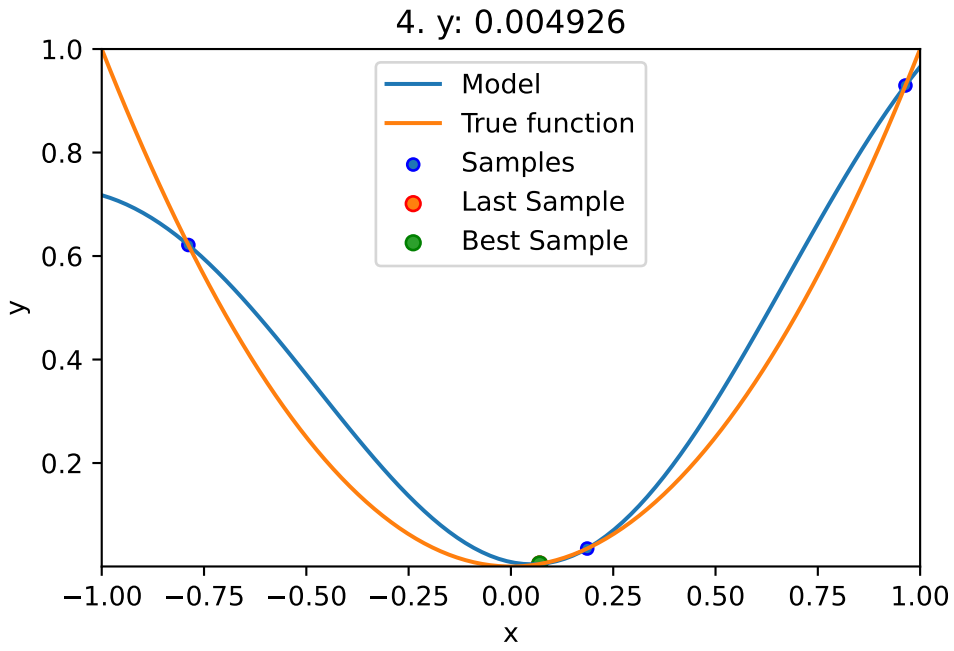


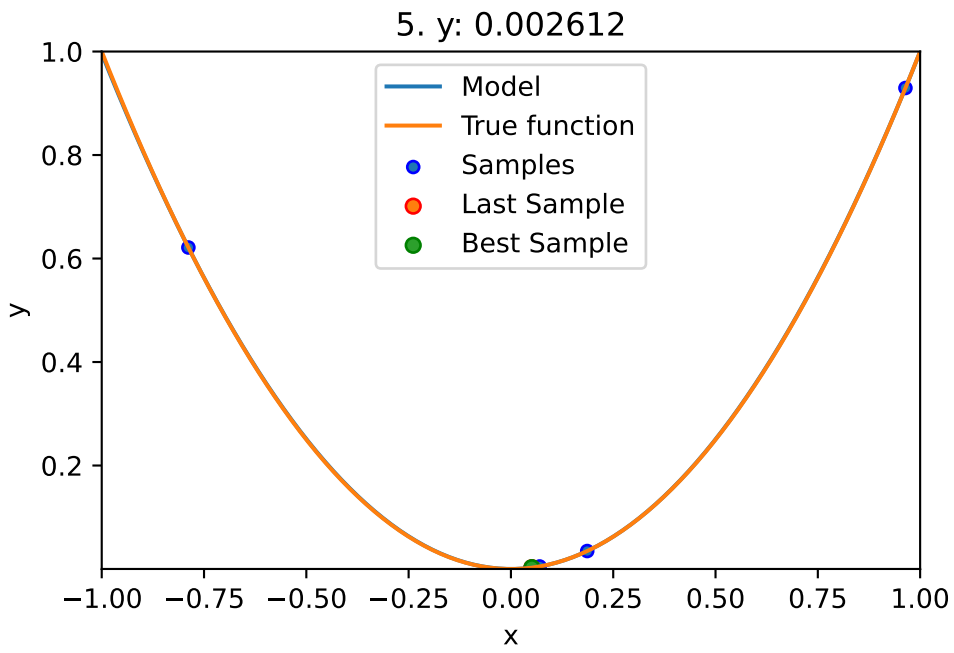
6.1 Example: `Sklearn` Model GaussianProcess

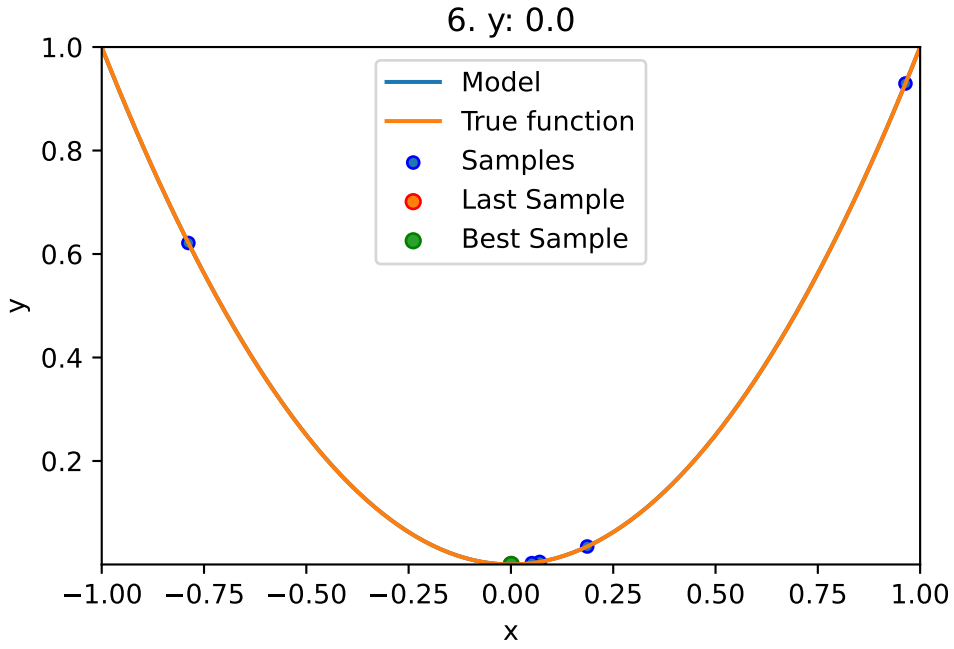
- This example visualizes the search process on the `GaussianProcessRegression` surrogate from `sklearn`.
- Therefore `surrogate = S_GP` is added to the argument list.

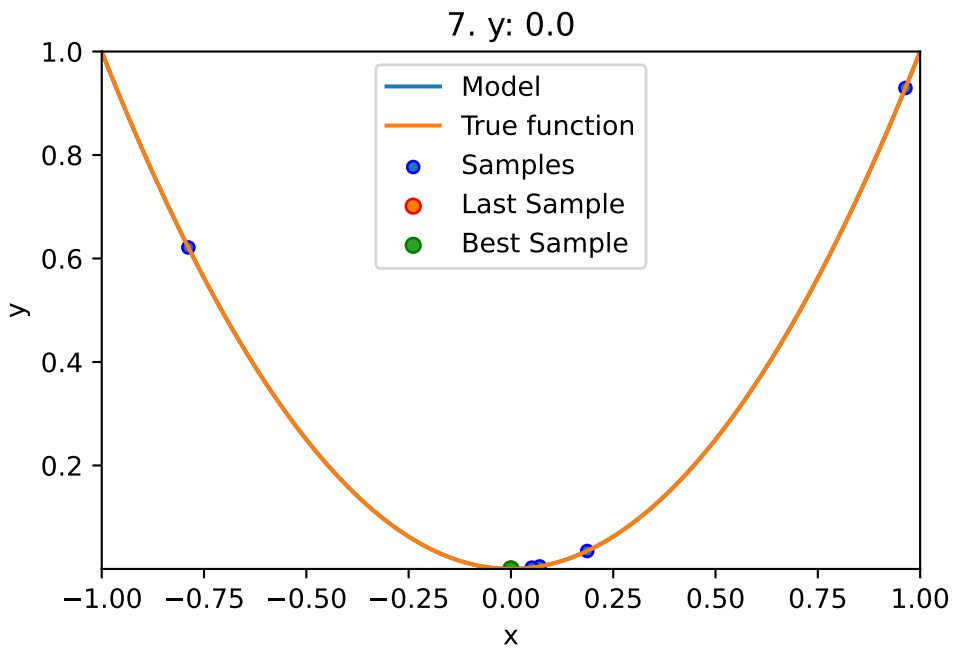
```
fun = analytical(seed=123).fun_sphere
spot_1_GP = spot.Spot(fun=fun,
                      lower = lower,
                      upper = upper,
                      fun_evals = 10,
                      max_time = inf,
                      seed=123,
                      show_models= True,
                      design_control={"init_size": 3},
                      surrogate = S_GP)
spot_1_GP.run()
```

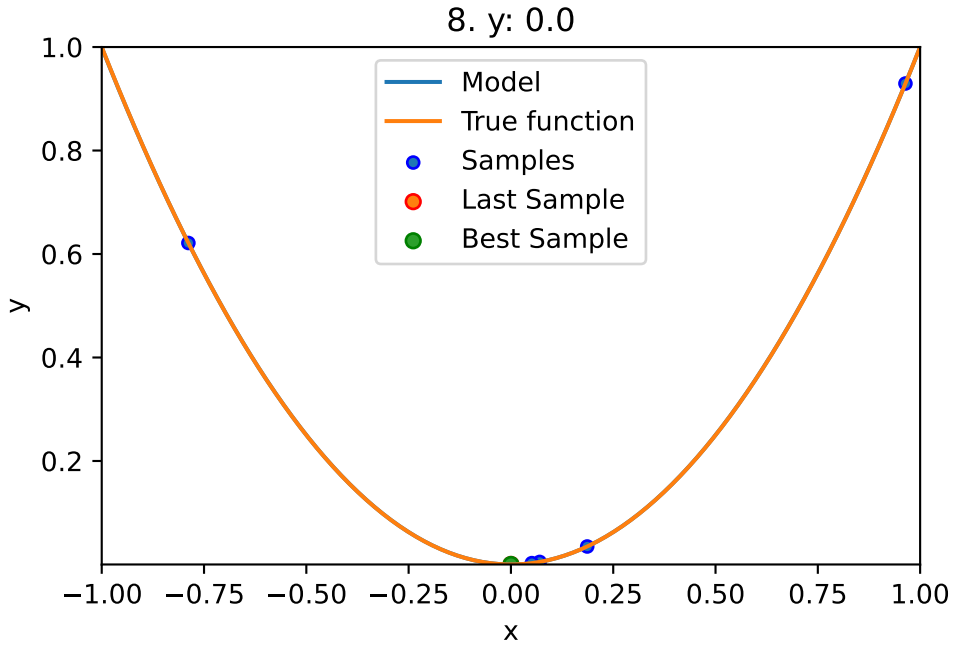

3. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

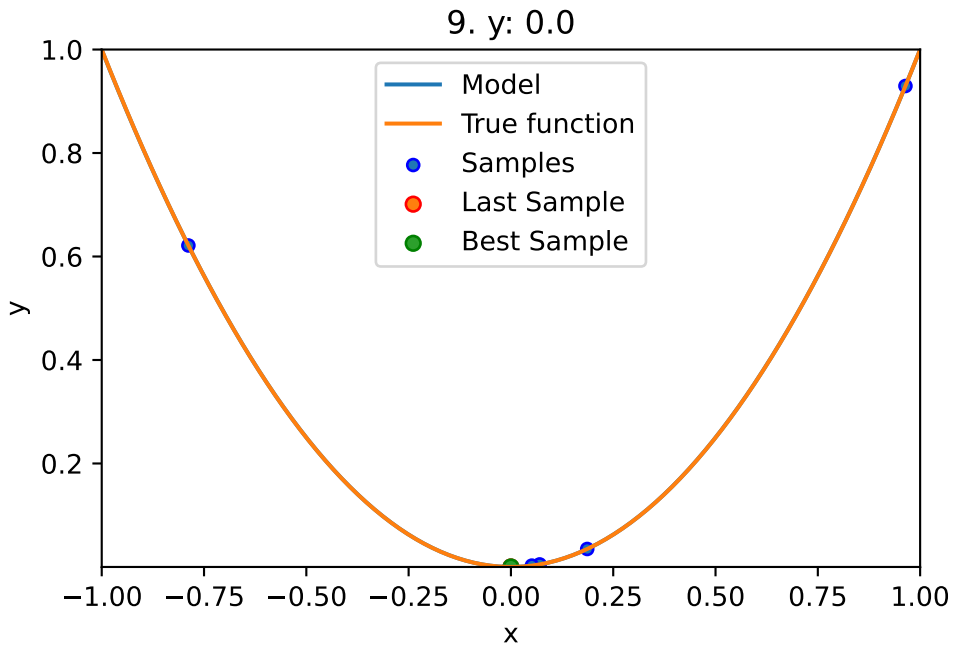

4. y: 0.004926
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

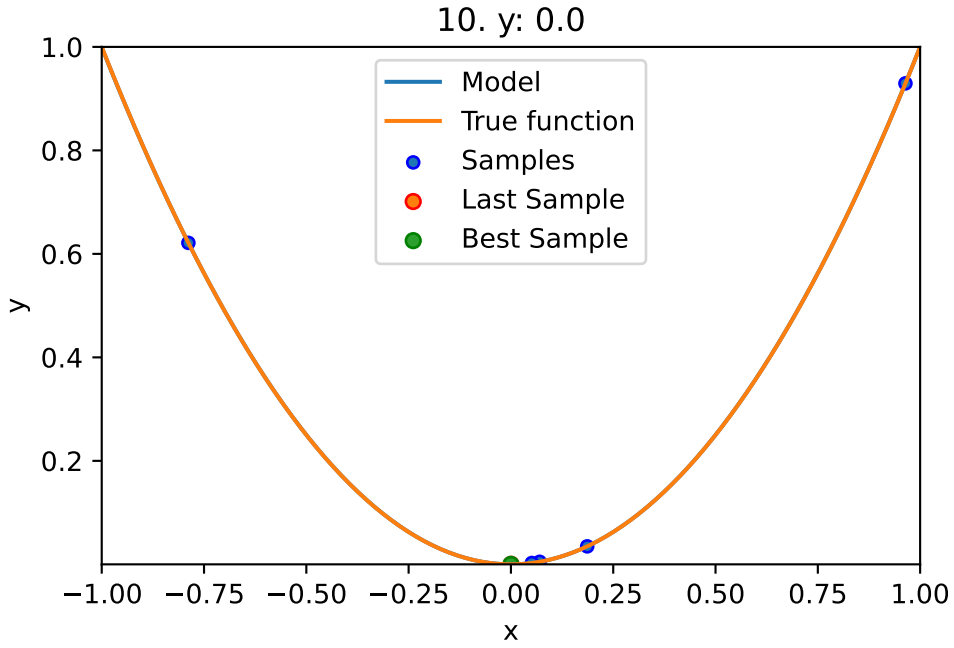

5. y: 0.002612
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


6. y: 0.0
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


7. y: 0.0
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


8. y: 0.0
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x



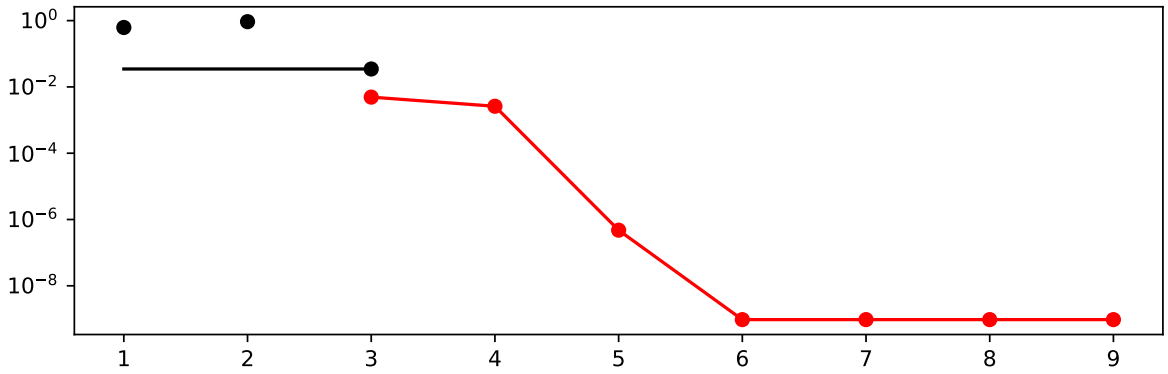



```
<spotPython.spot.spot.Spot at 0x28f3210f0>
```

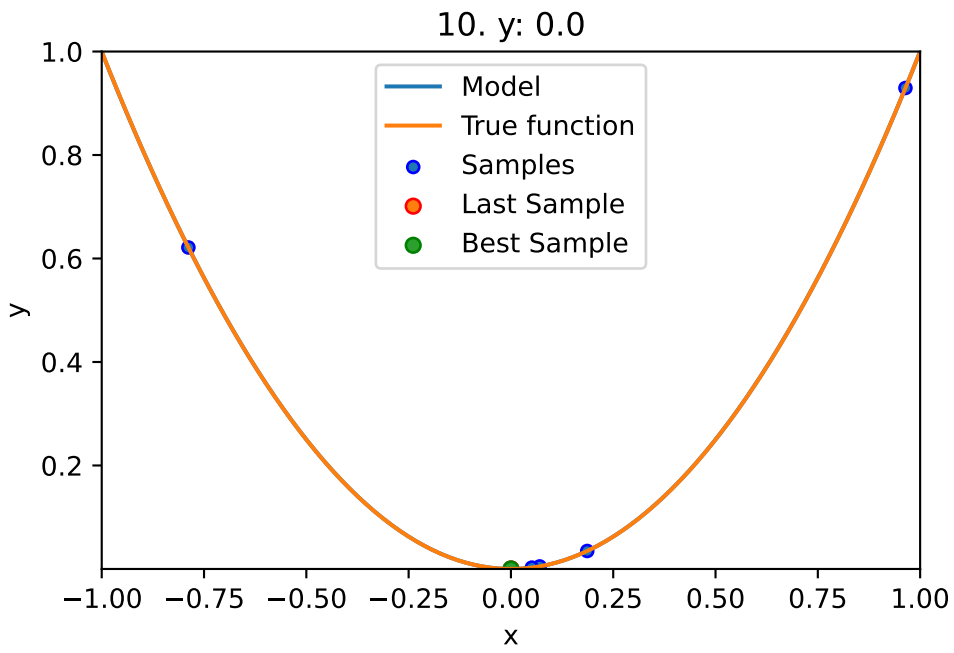
```
spot_1_GP.print_results()
```

```
min y: 9.548445362762409e-10
x0: 3.090055883436804e-05

[['x0', 3.090055883436804e-05]]
```

```
spot_1_GP.plot_progress(log_y=True)
```




```
spot_1_GP.plot_model()
```


10. y: 0.0
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

7 Exercises

- Important:
  - Results from these exercises should be added to this document, i.e., you should submit an updated version of this notebook.
  - Please combine your results using this notebook.
  - Only one notebook from each group!
  - Presentation is based on this notebook. No addtional slides are required!
  - spotPython version `0.16.11` (or greater) is required.

7.0.1 `DecisionTreeRegressor`

- Describe the surrogate model.
- Use the surrogate as the model for optimization.

7.0.2 `RandomForestRegressor`

- Describe the surrogate model.
- Use the surrogate as the model for optimization.

7.0.3 `linear_model.LinearRegression`

- Describe the surrogate model.
- Use the surrogate as the model for optimization.

7.0.4 `linear_model.Ridge`

- Describe the surrogate model.
- Use the surrogate as the model for optimization.

7.1 Exercise 2

- Compare the performance of the five different surrogates on both objective functions:
  - `spotPython`'s internal Kriging
  - `DecisionTreeRegressor`
  - `RandomForestRegressor`
  - `linear_model.LinearRegression`
  - `linear_model.Ridge`

8 Sequential Parameter Optimization: Using `scipy` Optimizers

This notebook describes how different optimizers form the `scipy optimize` package can be used on the surrogate. The optimization algorithms are available from https://docs.scipy.org/doc/scipy/reference/optimize.html

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
from scipy.optimize import dual_annealing
from scipy.optimize import basinhopping
import matplotlib.pyplot as plt
```

8.1 The Objective Function Branin

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula.
- Here we will use the Branin function. The 2-dim Branin function is

$$y = a * (x2 - b * x1 * *2 + c * x1 - r) * *2 + s * (1 - t) * cos(x1) + s,$$

  where values of a, b, c, r, s and t are: $a = 1, b = 5.1/(4 * pi * *2), c = 5/pi, r = 6, s = 10$ and $t = 1/(8 * pi)$.
- It has three global minima:

  $f(x) = 0.397887$ at $(-\pi, 12.275)$, $(\pi, 2.275)$, and $(9.42478, 2.475)$.

- Input Domain: This function is usually evaluated on the square x1 in [-5, 10] x x2 in [0, 15].

```
from spotPython.fun.objectivefunctions import analytical
lower = np.array([-5,-0])
upper = np.array([10,15])
```

```
fun = analytical(seed=123).fun_branin
```

8.2 The Optimizer

- Differential Evalution from the `scikit.optimize` package, see https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html#scipy.optimize.differential_evolution is the default optimizer for the search on the surrogate.
- Other optimiers that are available in `spotPython`:
  - `dual_annealing`
  - `direct`
  - `shgo`
  - `basinhopping`, see https://docs.scipy.org/doc/scipy/reference/optimize.html#global-optimization.
- These can be selected as follows:

  `surrogate_control = "model_optimizer": differential_evolution`
- We will use `differential_evolution`.
- The optimizer can use `1000` evaluations. This value will be passed to the `differential_evolution` method, which has the argument `maxiter` (int). It defines the maximum number of generations over which the entire differential evolution population is evolved, see https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html#scipy.optimize.differential_evolution

```
spot_de = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = 20,
                    max_time = inf,
                    seed=125,
                    noise=False,
```

```
                    show_models= False,
                    design_control={"init_size": 10},
                    surrogate_control={"n_theta": 2,
                                       "model_optimizer": differential_evolution,
                                       "model_fun_evals": 1000,
                                       })
spot_de.run()
```

```
<spotPython.spot.spot.Spot at 0x14f3f7eb0>
```

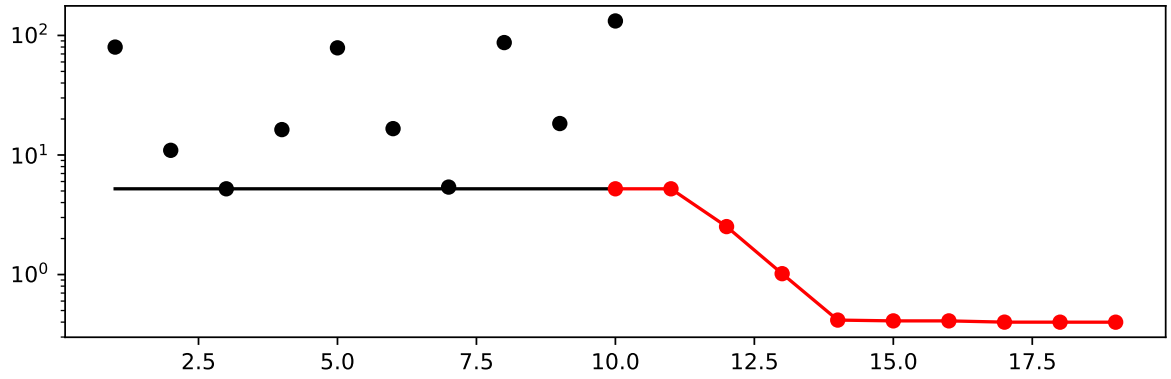
8.3 Print the Results

```
spot_de.print_results()
```

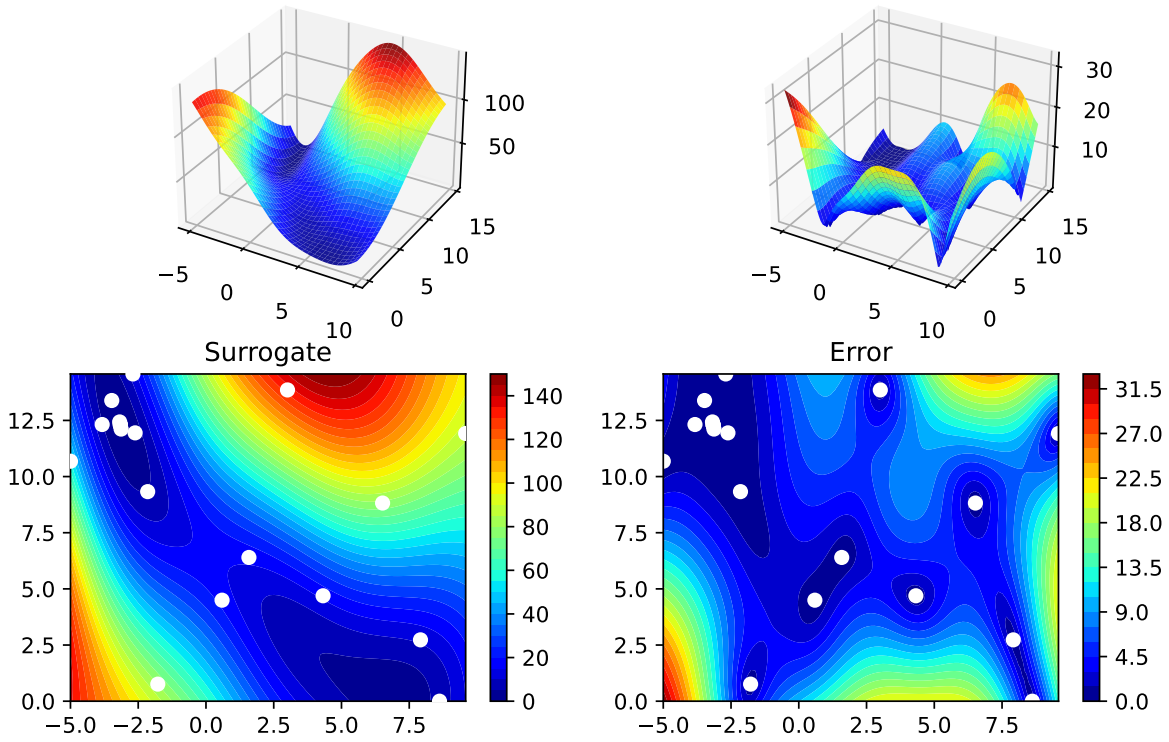
```
min y: 0.39951958110619046
x0: -3.1570201165683587
x1: 12.289980569430284

[['x0', -3.1570201165683587], ['x1', 12.289980569430284]]
```

8.4 Show the Progress

```
spot_de.plot_progress(log_y=True)
```



```
spot_de.surrogate.plot()
```



9 Exercises

9.1 `dual_annealing`

- Describe the optimization algorithm
- Use the algorithm as an optimizer on the surrogate

9.2 `direct`

- Describe the optimization algorithm
- Use the algorithm as an optimizer on the surrogate

9.3 `shgo`

- Describe the optimization algorithm
- Use the algorithm as an optimizer on the surrogate

9.4 `basinhopping`

- Describe the optimization algorithm
- Use the algorithm as an optimizer on the surrogate

9.5 Performance Comparison

Compare the performance and run time of the 5 different optimizers:

```
* `differential_evolution`
* `dual_annealing`
*  `direct`
* `shgo`
* `basinhopping`.
```

The Branin function has three global minima:

- $f(x) = 0.397887$ at
  - $(-\pi, 12.275)$,
  - $(\pi, 2.275)$, and
  - $(9.42478, 2.475)$.

- Which optima are found by the optimizers? Does the `seed` change this behavior?

10 Sequential Parameter Optimization: Gaussian Process Models

- This notebook analyzes differences between
  - the `Kriging` implementation in `spotPython` and
  - the `GaussianProcessRegressor` in `scikit-learn`.

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.design.spacefilling import spacefilling
from spotPython.spot import spot
from spotPython.build.kriging import Kriging
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
import math as m
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
```

10.1 Gaussian Processes Regression: Basic Introductory `scikit-learn` Example

- This is the example from scikit-learn: https://scikit-learn.org/stable/auto_examples/gaussian_process/pl
- After fitting our model, we see that the hyperparameters of the kernel have been optimized.
- Now, we will use our kernel to compute the mean prediction of the full dataset and plot the 95% confidence interval.

10.1.1 Train and Test Data

```
X = np.linspace(start=0, stop=10, num=1_000).reshape(-1, 1)
y = np.squeeze(X * np.sin(X))
rng = np.random.RandomState(1)
training_indices = rng.choice(np.arange(y.size), size=6, replace=False)
X_train, y_train = X[training_indices], y[training_indices]
```

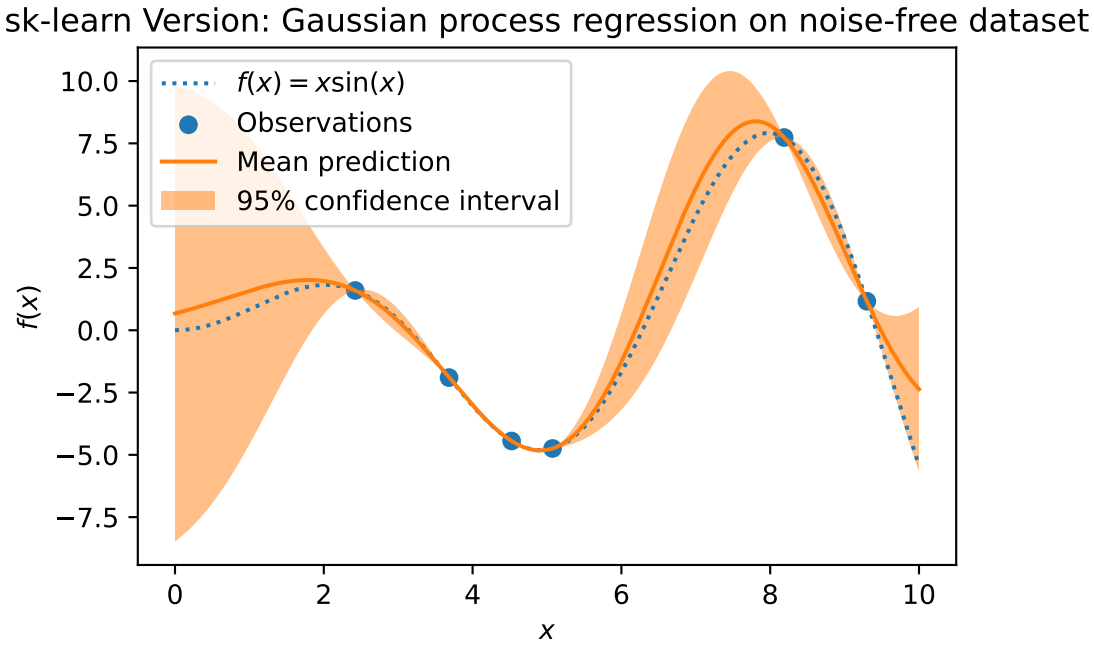
10.1.2 Building the Surrogate With `Sklearn`

- The model building with `sklearn` consisits of three steps:
    1. Instantiating the model, then
    2. fitting the model (using `fit`), and
    3. making predictions (using `predict`)

```
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
gaussian_process = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
gaussian_process.fit(X_train, y_train)
mean_prediction, std_prediction = gaussian_process.predict(X, return_std=True)
```

10.1.3 Plotting the `Sklearn`Model

```
plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
plt.fill_between(
    X.ravel(),
    mean_prediction - 1.96 * std_prediction,
    mean_prediction + 1.96 * std_prediction,
    alpha=0.5,
    label=r"95% confidence interval",
)
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("sk-learn Version: Gaussian process regression on noise-free dataset")
```



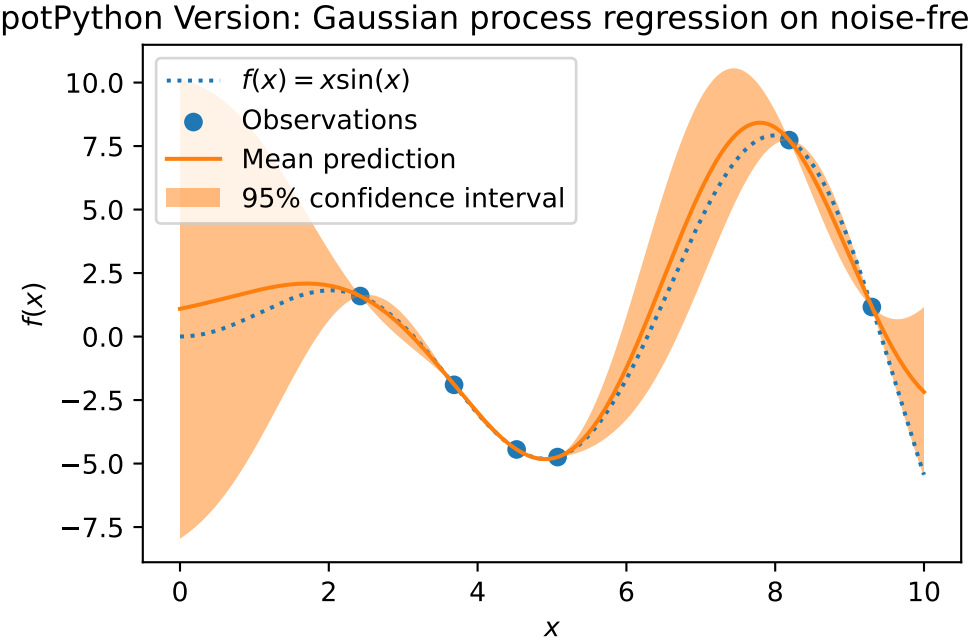
10.1.4 The `spotPython` Version

- The `spotPython` version is very similar:
  1. Instantiating the model, then
  2. fitting the model and
  3. making predictions (using `predict`).

```
S = Kriging(name='kriging',  seed=123, log_level=50, cod_type="norm")
S.fit(X_train, y_train)
S_mean_prediction, S_std_prediction, S_ei = S.predict(X, return_val="all")
```

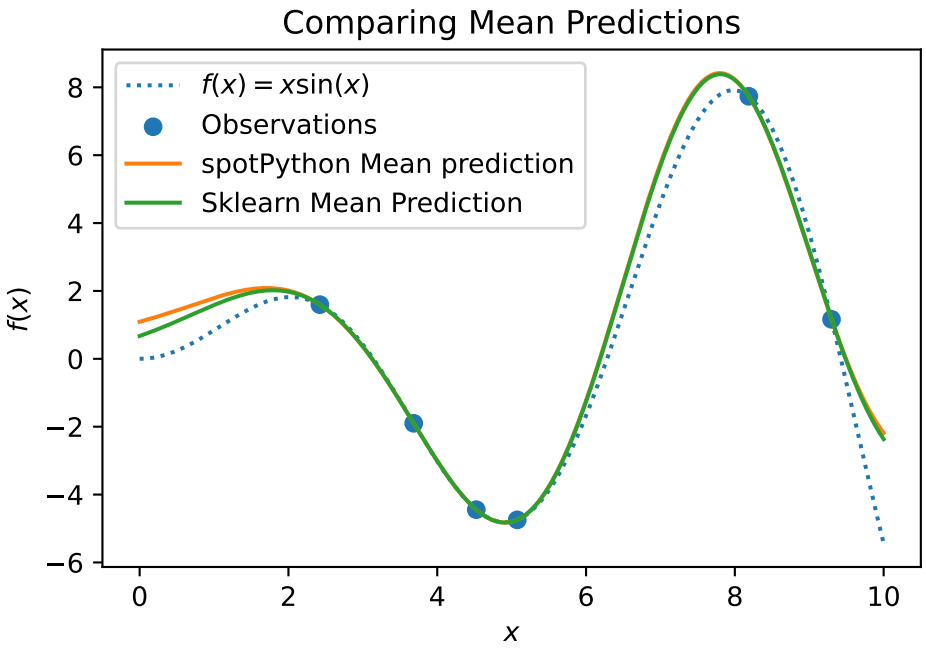
```
plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, S_mean_prediction, label="Mean prediction")
plt.fill_between(
    X.ravel(),
    S_mean_prediction - 1.96 * S_std_prediction,
    S_mean_prediction + 1.96 * S_std_prediction,
    alpha=0.5,
    label=r"95% confidence interval",
```

```
)
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("spotPython Version: Gaussian process regression on noise-free dataset")
```



10.1.5 Visualizing the Differences Between the `spotPython` and the `sklearn` Model Fits

```
plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, S_mean_prediction, label="spotPython Mean prediction")
plt.plot(X, mean_prediction, label="Sklearn Mean Prediction")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Comparing Mean Predictions")
```


Comparing Mean Predictions
$f(x) = x\sin(x)$
Observations
spotPython Mean prediction
Sklearn Mean Prediction
$f(x)$
x
8
6
4
2
0
−2
−4
−6
0
2
4
6
8
10

11 Exercises

11.1 Schonlau Example Function

- The Schonlau Example Function is based on sample points only (there is no analytical function description available):

```
X = np.linspace(start=0, stop=13, num=1_000).reshape(-1, 1)
X_train = np.array([1., 2., 3., 4., 12.]).reshape(-1,1)
y_train = np.array([0., -1.75, -2, -0.5, 5.])
```

- Describe the function.
- Compare the two models that were build using the `spotPython` and the `sklearn` surrogate.
- Note: Since there is no analytical function available, you might be interested in adding some points and describe the effects.

11.2 Forrester Example Function

- The Forrester Example Function is defined as follows:

  `f(x) = (6x- 2)^2 sin(12x-4) for x in [0,1].`

- Data points are generated as follows:

```
X = np.linspace(start=-0.5, stop=1.5, num=1_000).reshape(-1, 1)
X_train = np.array([0.0, 0.175, 0.225, 0.3, 0.35, 0.375, 0.5,1]).reshape(-1,1)
fun = analytical().fun_forrester
fun_control = {"sigma": 0.1,
               "seed": 123}
y = fun(X, fun_control=fun_control)
y_train = fun(X_train, fun_control=fun_control)
```

- Describe the function.

- Compare the two models that were build using the `spotPython` and the `sklearn` surrogate.
- Note: Modify the noise level (`"sigma"`), e.g., use a value of `0.2`, and compare the two models.

```
fun_control = {"sigma": 0.2}
```

11.3 `fun_runge` Function (1-dim)

- The Runge function is defined as follows:

  `f(x) = 1/ (1 + sum(x_i))^2`

- Data points are generated as follows:

```
gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_runge
fun_control = {"sigma": 0.025,
               "seed": 123}
X_train = gen.scipy_lhd(10, lower=lower, upper = upper).reshape(-1,1)
y_train = fun(X, fun_control=fun_control)
X = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
y = fun(X, fun_control=fun_control)
```

- Describe the function.
- Compare the two models that were build using the `spotPython` and the `sklearn` surrogate.
- Note: Modify the noise level (`"sigma"`), e.g., use a value of `0.05`, and compare the two models.

```
fun_control = {"sigma": 0.5}
```

11.4 `fun_cubed` (1-dim)

- The Cubed function is defined as follows:

  `np.sum(X[i]** 3)`

- Data points are generated as follows:

```
gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_cubed
fun_control = {"sigma": 0.025,
               "seed": 123}
X_train = gen.scipy_lhd(10, lower=lower, upper = upper).reshape(-1,1)
y_train = fun(X, fun_control=fun_control)
X = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
y = fun(X, fun_control=fun_control)
```

- Describe the function.
- Compare the two models that were build using the `spotPython` and the `sklearn` surrogate.
- Note: Modify the noise level (`"sigma"`), e.g., use a value of `0.05`, and compare the two models.

```
fun_control = {"sigma": 0.05}
```

11.5 The Effect of Noise

How does the behavior of the `spotPython` fit changes when the argument `noise` is set to `True`, i.e.,

```
S = Kriging(name='kriging',  seed=123, n_theta=1, noise=True)
```

is used?

12 Expected Improvement

12.1 Example: `Spot` and the 1-dim Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

12.1.1 The Objective Function: 1-dim Sphere

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x) = x^2$$

```
fun = analytical().fun_sphere
```

```
fun = analytical().fun_sphere
fun_control = {"sigma": 0,
               "seed": 123}
```

- The size of the `lower` bound vector determines the problem dimension.
- Here we will use `np.array([-1])`, i.e., a one-dim function.

```
spot_1 = spot.Spot(fun=fun,
                   lower = np.array([-1]),
                   upper = np.array([1]))
```

```
spot_1.run()
```

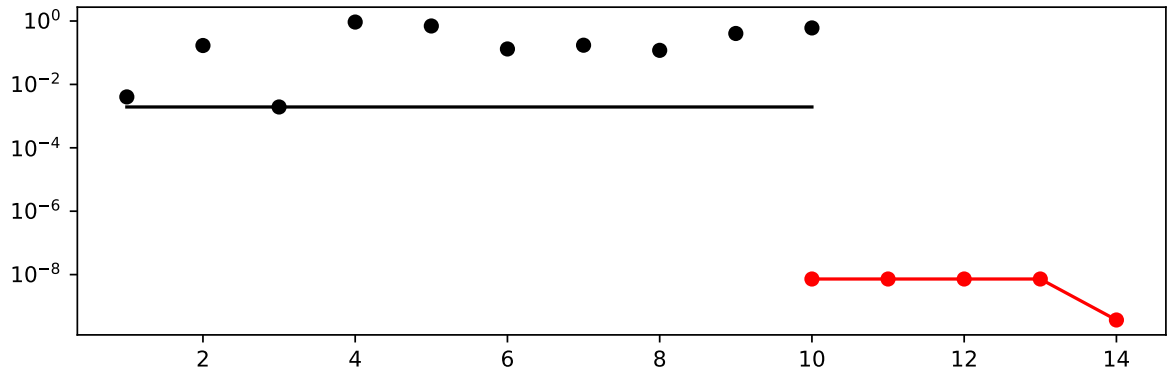
```
<spotPython.spot.spot.Spot at 0x17b8e38e0>
```

12.1.2 Results

```
spot_1.print_results()
```

```
min y: 3.696886711914087e-10
x0: 1.922728975158508e-05

[['x0', 1.922728975158508e-05]]
```

```
spot_1.plot_progress(log_y=True)
```

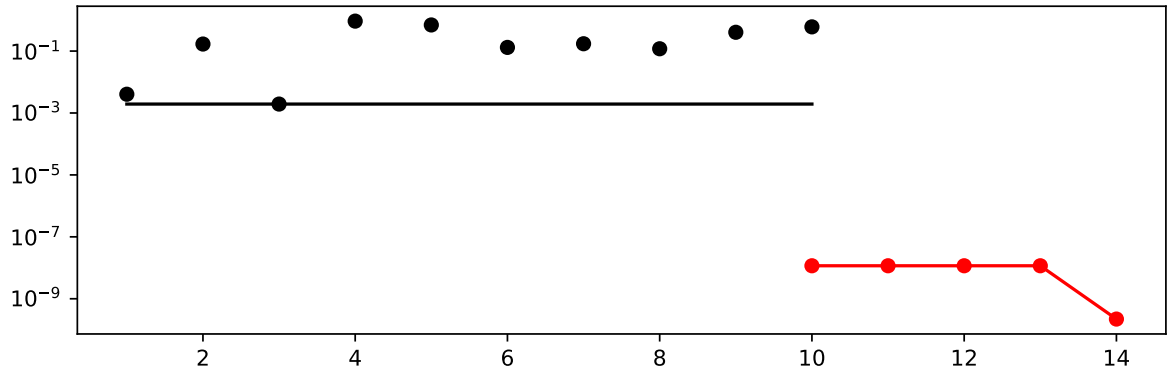


12.2 Same, but with EI as infill_criterion

```
spot_1_ei = spot.Spot(fun=fun,
                    lower = np.array([-1]),
                    upper = np.array([1]),
                    infill_criterion = "ei")
spot_1_ei.run()
```

```
<spotPython.spot.spot.Spot at 0x17f65e0b0>
```

```
spot_1_ei.plot_progress(log_y=True)
```




```
spot_1_ei.print_results()
```

```
min y: 2.207887258868953e-10
x0: 1.4858961130809088e-05

[['x0', 1.4858961130809088e-05]]
```

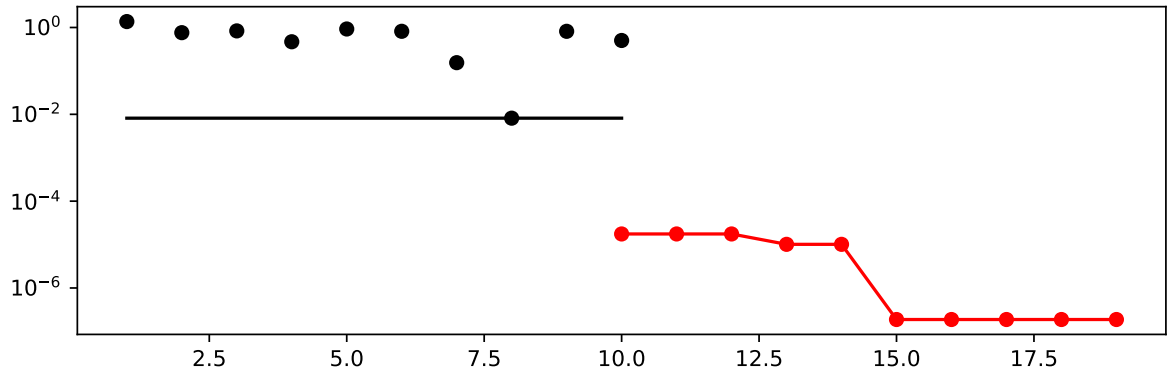
12.3 Non-isotropic Kriging

```
spot_2_ei_noniso = spot.Spot(fun=fun,
                    lower = np.array([-1, -1]),
                    upper = np.array([1, 1]),
                    fun_evals = 20,
                    fun_repeats = 1,
                    max_time = inf,
                    noise = False,
                    tolerance_x = np.sqrt(np.spacing(1)),
                    var_type=["num"],
                    infill_criterion = "ei",
                    n_points = 1,
                    seed=123,
                    log_level = 50,
                    show_models=True,
                    fun_control = fun_control,
```

```
                    design_control={"init_size": 10,
                                    "repeats": 1},
                    surrogate_control={"noise": False,
                                       "cod_type": "norm",
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": 2,
                                       "model_optimizer": differential_evolution,
                                       "model_fun_evals": 1000,
                                       })
spot_2_ei_noniso.run()
```

```
<spotPython.spot.spot.Spot at 0x17f7c2590>
```

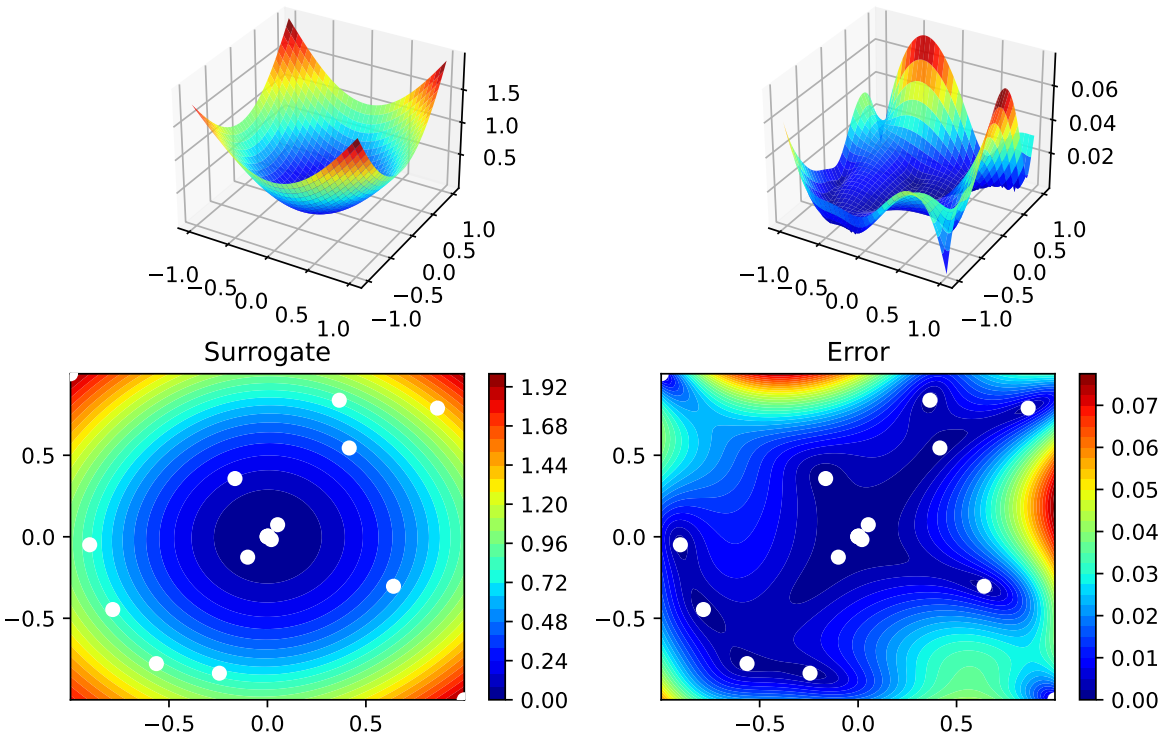
```
spot_2_ei_noniso.plot_progress(log_y=True)
```




```
spot_2_ei_noniso.print_results()
```

```
min y: 1.8779971830281702e-07
x0: -0.0002783721390529846
x1: 0.0003321274913371111

[['x0', -0.0002783721390529846], ['x1', 0.0003321274913371111]]
```

```
spot_2_ei_noniso.surrogate.plot()
```



12.4 Using `sklearn` Surrogates

12.4.1 The spot Loop

The `spot` loop consists of the following steps:

1. Init: Build initial design X
2. Evaluate initial design on real objective f: $y = f(X)$
3. Build surrogate: $S = S(X, y)$
4. Optimize on surrogate: $X_0 = \text{optimize}(S)$
5. Evaluate on real objective: $y_0 = f(X_0)$
6. Impute (Infill) new points: $X = X \cup X_0$, $y = y \cup y_0$.
7. Got 3.

The `spot` loop is implemented in `R` as follows:

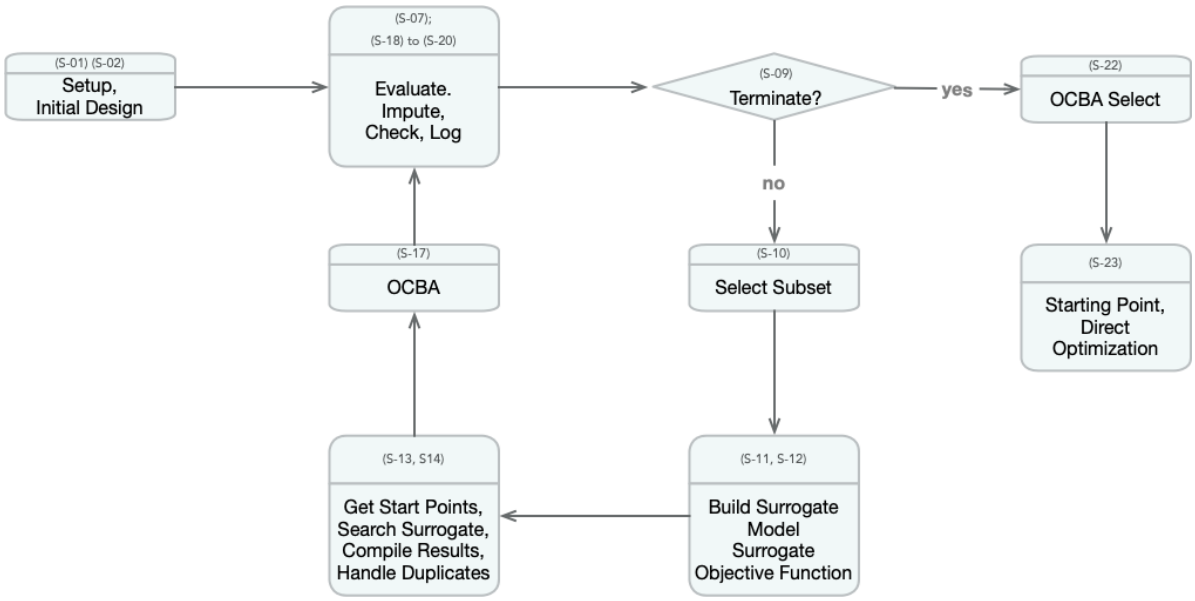


Figure 12.1: Visual representation of the model based search with SPOT. Taken from: Bartz-Beielstein, T., and Zaefferer, M. Hyperparameter tuning approaches. In Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide, E. Bartz, T. Bartz-Beielstein, M. Zaefferer, and O. Mersmann, Eds. Springer, 2022, ch. 4, pp. 67–114.

12.4.2 spot: The Initial Model

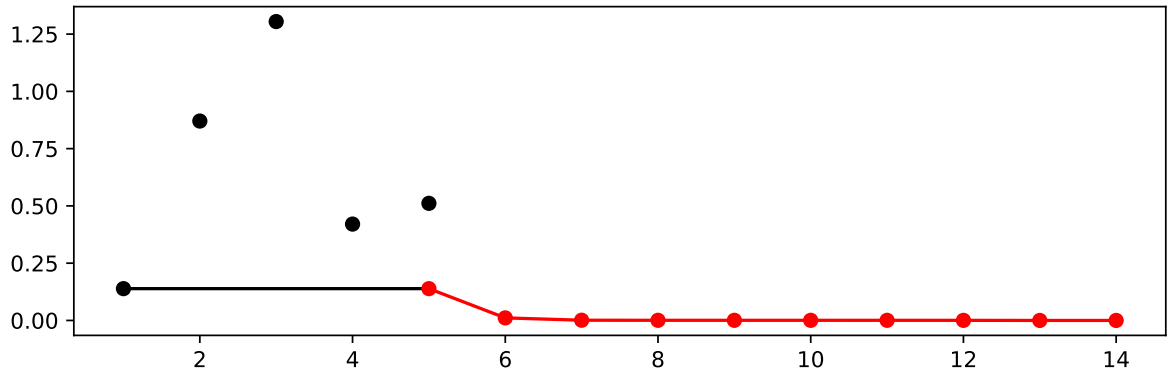
12.4.2.1 Example: Modifying the initial design size

This is the "Example: Modifying the initial design size" from Chapter 4.5.1 in [bart21i].

```
spot_ei = spot.Spot(fun=fun,
                    lower = np.array([-1,-1]),
                    upper= np.array([1,1]),
                    design_control={"init_size": 5})
spot_ei.run()
```

```
<spotPython.spot.spot.Spot at 0x2b513edd0>
```

```
spot_ei.plot_progress()
```




```
np.min(spot_1.y), np.min(spot_ei.y)
```

```
(3.696886711914087e-10, 1.7928640814182596e-05)
```

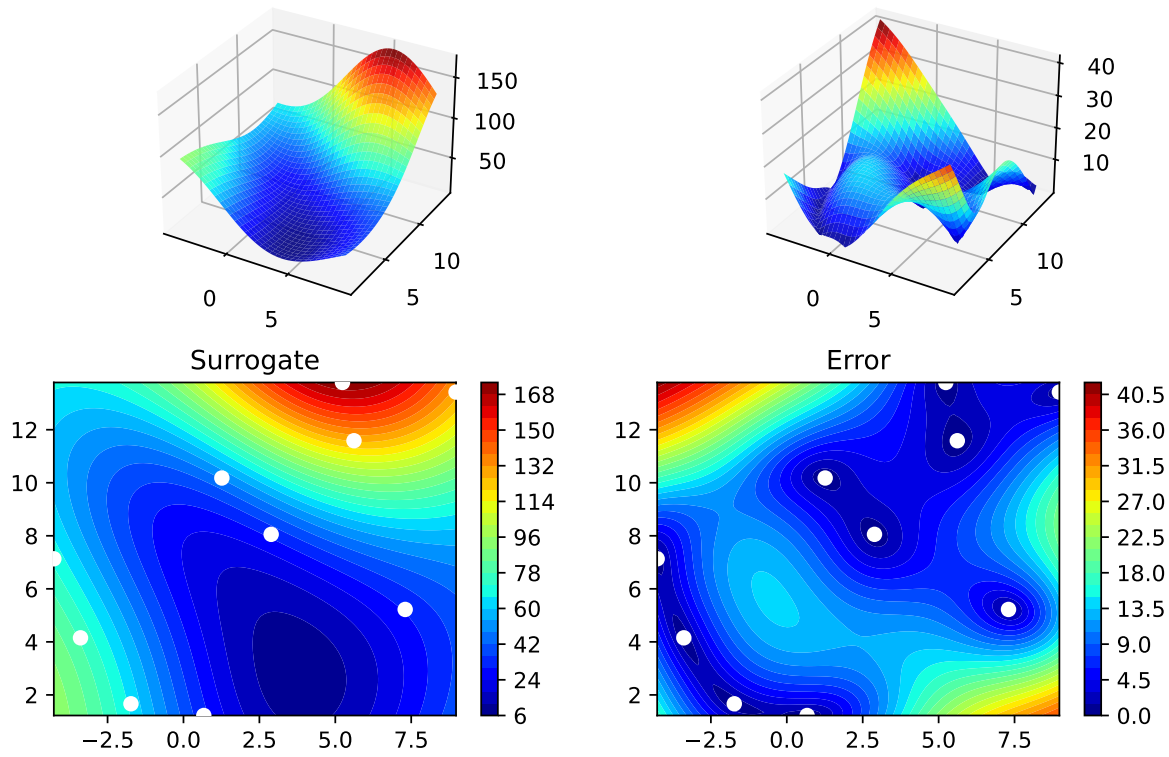
12.4.3 Init: Build Initial Design

```
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
from spotPython.fun.objectivefunctions import analytical
gen = spacefilling(2)
```

```
rng = np.random.RandomState(1)
lower = np.array([-5,-0])
upper = np.array([10,15])
fun = analytical().fun_branin
fun_control = {"sigma": 0,
               "seed": 123}

X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
print(y)
```

```
[[ 8.97647221 13.41926847]
 [ 0.66946019  1.22344228]
 [ 5.23614115 13.78185824]
 [ 5.6149825  11.5851384 ]
 [-1.72963184  1.66516096]
 [-4.26945568  7.1325531 ]
 [ 1.26363761 10.17935555]
 [ 2.88779942  8.05508969]
 [-3.39111089  4.15213772]
 [ 7.30131231  5.22275244]]
[128.95676449  31.73474356 172.89678121 126.71295908  64.34349975
  70.16178611  48.71407916  31.77322887  76.91788181  30.69410529]
```

```
S = Kriging(name='kriging',  seed=123)
S.fit(X, y)
S.plot()
```




```
gen = spacefilling(2, seed=123)
X0 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=345)
X1 = gen.scipy_lhd(3)
X2 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=123)
X3 = gen.scipy_lhd(3)
X0, X1, X2, X3
```

```
(array([[0.77254938, 0.31539299],
        [0.59321338, 0.93854273],
        [0.27469803, 0.3959685 ]]),
 array([[0.78373509, 0.86811887],
        [0.06692621, 0.6058029 ],
        [0.41374778, 0.00525456]]),
 array([[0.121357  , 0.69043832],
        [0.41906219, 0.32838498],
        [0.86742658, 0.52910374]]),
```

```
array([[0.77254938, 0.31539299],
       [0.59321338, 0.93854273],
       [0.27469803, 0.3959685 ]]))
```

12.4.4 Evaluate

12.4.5 Build Surrogate

12.4.6 A Simple Predictor

The code below shows how to use a simple model for prediction.

- Assume that only two (very costly) measurements are available:
  1. f(0) = 0.5
  2. f(2) = 2.5
- We are interested in the value at $x_0 = 1$, i.e., $f(x_0 = 1)$, but cannot run an additional, third experiment.

```
from sklearn import linear_model
X = np.array([[0], [2]])
y = np.array([0.5, 2.5])
S_lm = linear_model.LinearRegression()
S_lm = S_lm.fit(X, y)
X0 = np.array([[1]])
y0 = S_lm.predict(X0)
print(y0)
```

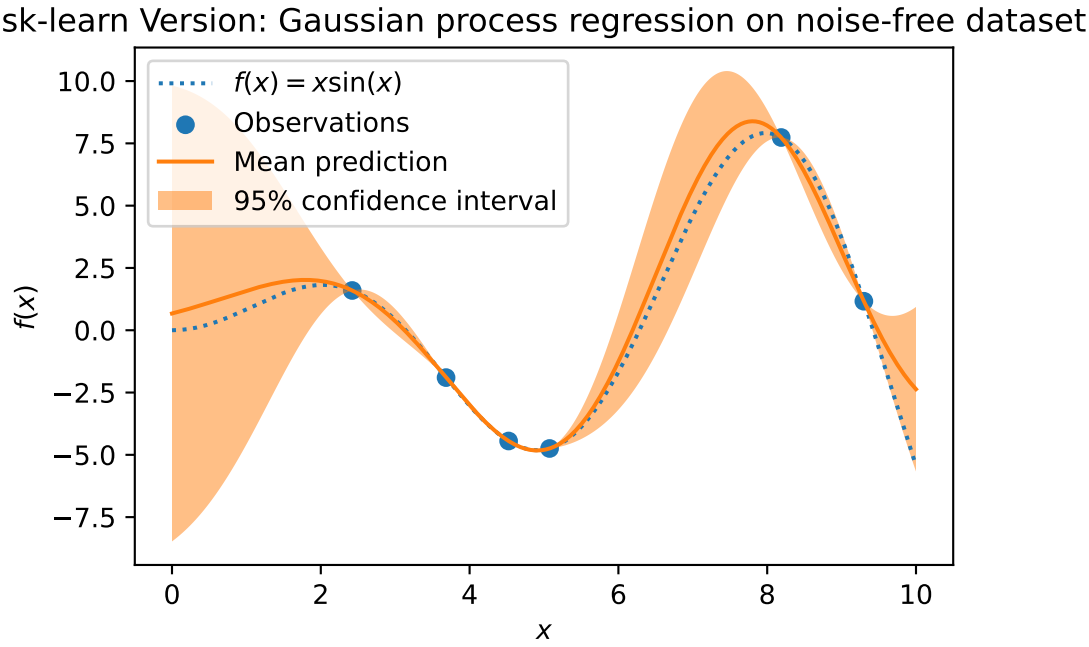
```
[1.5]
```

- Central Idea:
  - Evaluation of the surrogate model `S_lm` is much cheaper (or / and much faster) than running the real-world experiment f.

12.5 Gaussian Processes regression: basic introductory example

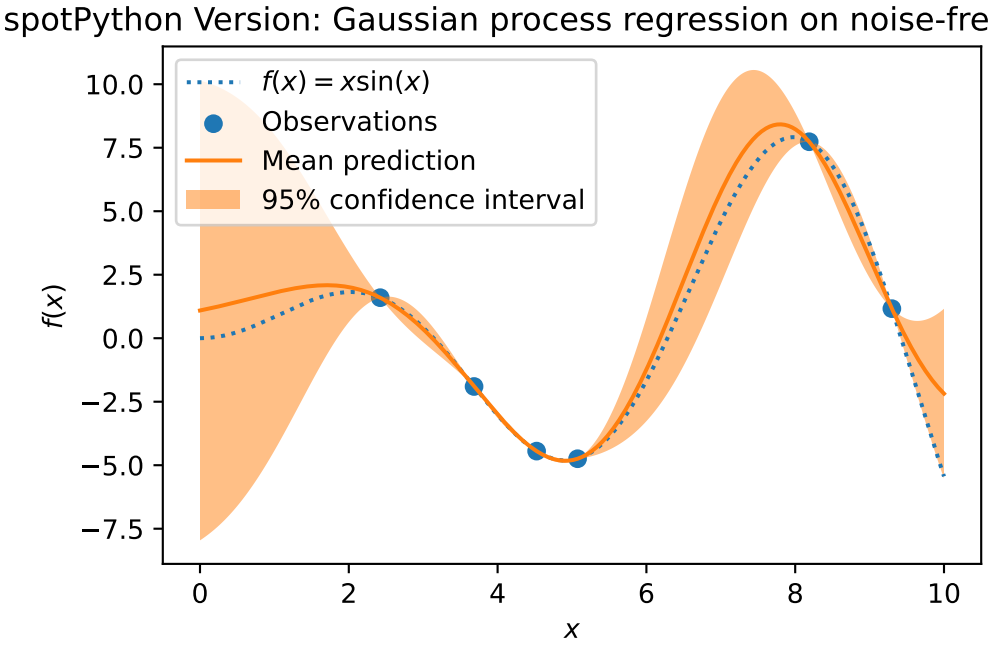
This example was taken from scikit-learn. After fitting our model, we see that the hyperparameters of the kernel have been optimized. Now, we will use our kernel to compute the mean prediction of the full dataset and plot the 95% confidence interval.

```
import numpy as np
import matplotlib.pyplot as plt
import math as m
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF

X = np.linspace(start=0, stop=10, num=1_000).reshape(-1, 1)
y = np.squeeze(X * np.sin(X))
rng = np.random.RandomState(1)
training_indices = rng.choice(np.arange(y.size), size=6, replace=False)
X_train, y_train = X[training_indices], y[training_indices]

kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
gaussian_process = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
gaussian_process.fit(X_train, y_train)
gaussian_process.kernel_

mean_prediction, std_prediction = gaussian_process.predict(X, return_std=True)

plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
plt.fill_between(
    X.ravel(),
    mean_prediction - 1.96 * std_prediction,
    mean_prediction + 1.96 * std_prediction,
    alpha=0.5,
    label=r"95% confidence interval",
)
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("sk-learn Version: Gaussian process regression on noise-free dataset")
```




```
from spotPython.build.kriging import Kriging
import numpy as np
import matplotlib.pyplot as plt
rng = np.random.RandomState(1)
X = np.linspace(start=0, stop=10, num=1_000).reshape(-1, 1)
y = np.squeeze(X * np.sin(X))
training_indices = rng.choice(np.arange(y.size), size=6, replace=False)
X_train, y_train = X[training_indices], y[training_indices]


S = Kriging(name='kriging',  seed=123, log_level=50, cod_type="norm")
S.fit(X_train, y_train)

mean_prediction, std_prediction, ei = S.predict(X, return_val="all")

std_prediction

plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
plt.fill_between(
```

```
    X.ravel(),
    mean_prediction - 1.96 * std_prediction,
    mean_prediction + 1.96 * std_prediction,
    alpha=0.5,
    label=r"95% confidence interval",
)
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("spotPython Version: Gaussian process regression on noise-free dataset")
```



12.6 The Surrogate: Using scikit-learn models

Default is the internal `kriging` surrogate.

```
S_0 = Kriging(name='kriging', seed=123)
```

Models from `scikit-learn` can be selected, e.g., Gaussian Process:

```
# Needed for the sklearn surrogates:
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model
from sklearn import tree
import pandas as pd
```

```
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
S_GP = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
```

- and many more:

```
S_Tree = DecisionTreeRegressor(random_state=0)
S_LM = linear_model.LinearRegression()
S_Ridge = linear_model.Ridge()
S_RF = RandomForestRegressor(max_depth=2, random_state=0)
```

- The scikit-learn GP model `S_GP` is selected.

```
S = S_GP
```

```
isinstance(S, GaussianProcessRegressor)
```

```
True
```

```
from spotPython.fun.objectivefunctions import analytical
fun = analytical().fun_branin
lower = np.array([-5,-0])
upper = np.array([10,15])
design_control={"init_size": 5}
surrogate_control={
            "infill_criterion": None,
            "n_points": 1,
        }
spot_GP = spot.Spot(fun=fun, lower = lower, upper= upper, surrogate=S,
                    fun_evals = 15, noise = False, log_level = 50,
                    design_control=design_control,
                    surrogate_control=surrogate_control)
```

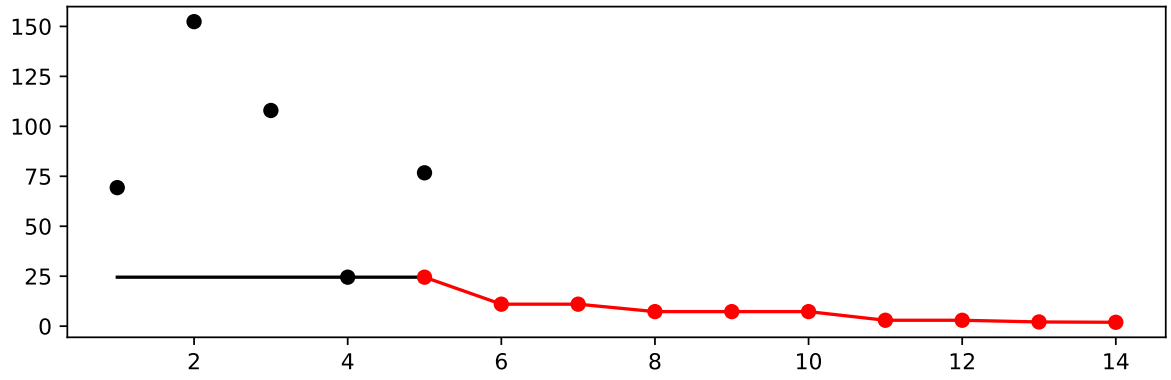
```
spot_GP.run()
```

```
<spotPython.spot.spot.Spot at 0x2b5240850>
```

```
spot_GP.y
```

```
array([ 69.32459936, 152.38491454, 107.92560483,  24.51465459,
        76.73500031,  86.30425961,  11.00307905,  16.11741963,
         7.28111644,  21.82327787,  10.96088904,   2.95189497,
         3.02909698,   2.10496168,   1.94316197])
```

```
spot_GP.plot_progress()
```




```
spot_GP.print_results()
```

```
min y: 1.9431619651857286
x0: 10.0
x1: 2.998341163759256

[['x0', 10.0], ['x1', 2.998341163759256]]
```

12.7 Additional Examples

```
# Needed for the sklearn surrogates:
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model
from sklearn import tree
import pandas as pd
```

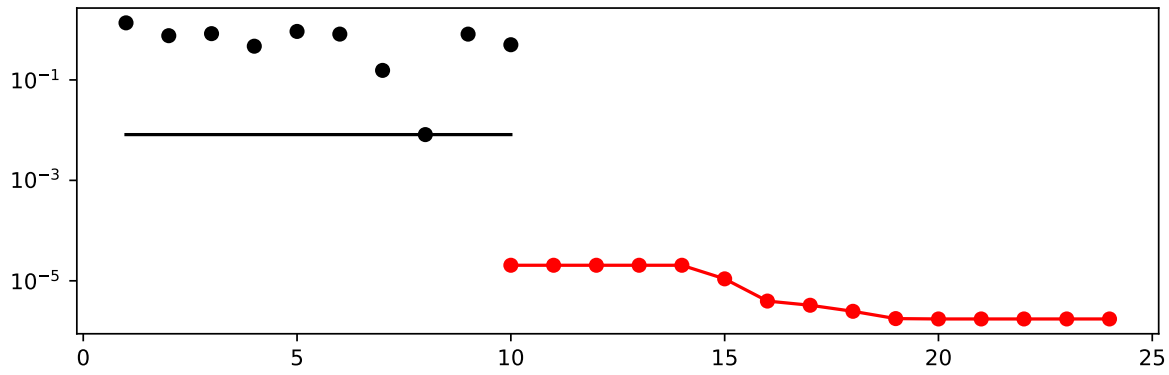
```
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
S_GP = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
```

```
from spotPython.build.kriging import Kriging
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot

S_K = Kriging(name='kriging',
              seed=123,
              log_level=50,
              infill_criterion = "y",
              n_theta=1,
              noise=False,
              cod_type="norm")
fun = analytical().fun_sphere
lower = np.array([-1,-1])
upper = np.array([1,1])

design_control={"init_size": 10}
surrogate_control={
            "n_points": 1,
        }
spot_S_K = spot.Spot(fun=fun,
                     lower = lower,
                     upper= upper,
                     surrogate=S_K,
                     fun_evals = 25,
                     noise = False,
                     log_level = 50,
```

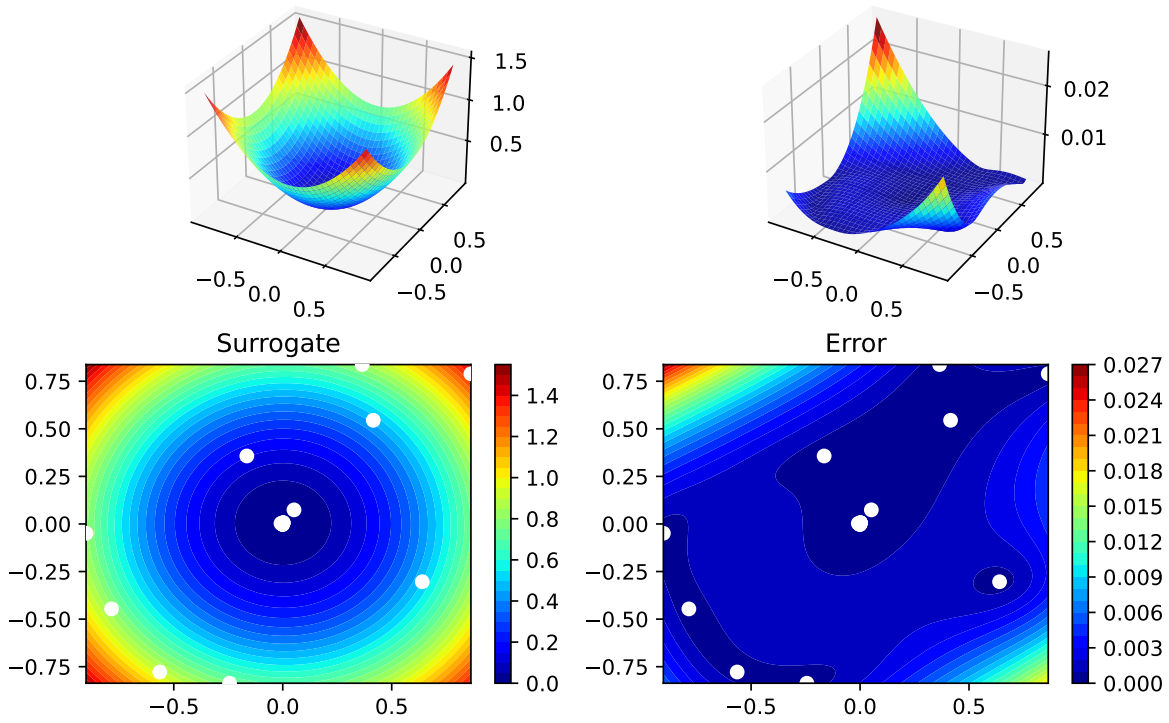
```
                    design_control=design_control,
                    surrogate_control=surrogate_control)

spot_S_K.run()
```

```
<spotPython.spot.spot.Spot at 0x17b9327a0>
```

```
spot_S_K.plot_progress(log_y=True)
```




```
spot_S_K.surrogate.plot()
```




```
spot_S_K.print_results()
```

```
min y: 1.7395335905335862e-06
x0: -0.0013044072412622557
x1: 0.0001950777780173277

[['x0', -0.0013044072412622557], ['x1', 0.0001950777780173277]]
```

12.7.1 Optimize on Surrogate

12.7.2 Evaluate on Real Objective

12.7.3 Impute / Infill new Points

12.8 Tests

```
import numpy as np
from spotPython.spot import spot
from spotPython.fun.objectivefunctions import analytical

fun_sphere = analytical().fun_sphere
spot_1 = spot.Spot(
    fun=fun_sphere,
    lower=np.array([-1, -1]),
    upper=np.array([1, 1]),
    n_points = 2
)

# (S-2) Initial Design:
spot_1.X = spot_1.design.scipy_lhd(
    spot_1.design_control["init_size"], lower=spot_1.lower, upper=spot_1.upper
)
print(spot_1.X)

# (S-3): Eval initial design:
spot_1.y = spot_1.fun(spot_1.X)
print(spot_1.y)

spot_1.surrogate.fit(spot_1.X, spot_1.y)
X0 = spot_1.suggest_new_X()
print(X0)
assert X0.size == spot_1.n_points * spot_1.k
```

```
[[ 0.86352963  0.7892358 ]
 [-0.24407197 -0.83687436]
 [ 0.36481882  0.8375811 ]
 [ 0.415331    0.54468512]
 [-0.56395091 -0.77797854]
 [-0.90259409 -0.04899292]]
```

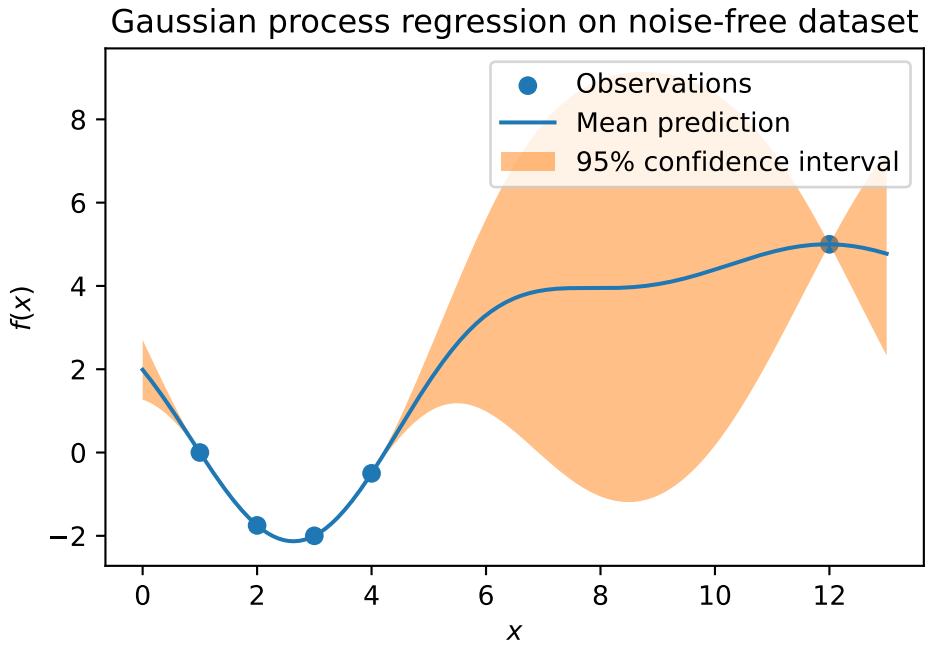
```
 [-0.16484832  0.35724741]
 [ 0.05170659  0.07401196]
 [-0.78548145 -0.44638164]
 [ 0.64017497 -0.30363301]]
[1.36857656 0.75992983 0.83463487 0.46918172 0.92329124 0.8170764
 0.15480068 0.00815134 0.81623768 0.502017  ]
[[0.00160553 0.00428429]
 [0.00160553 0.00428429]]
```

12.9 EI: The Famous Schonlau Example

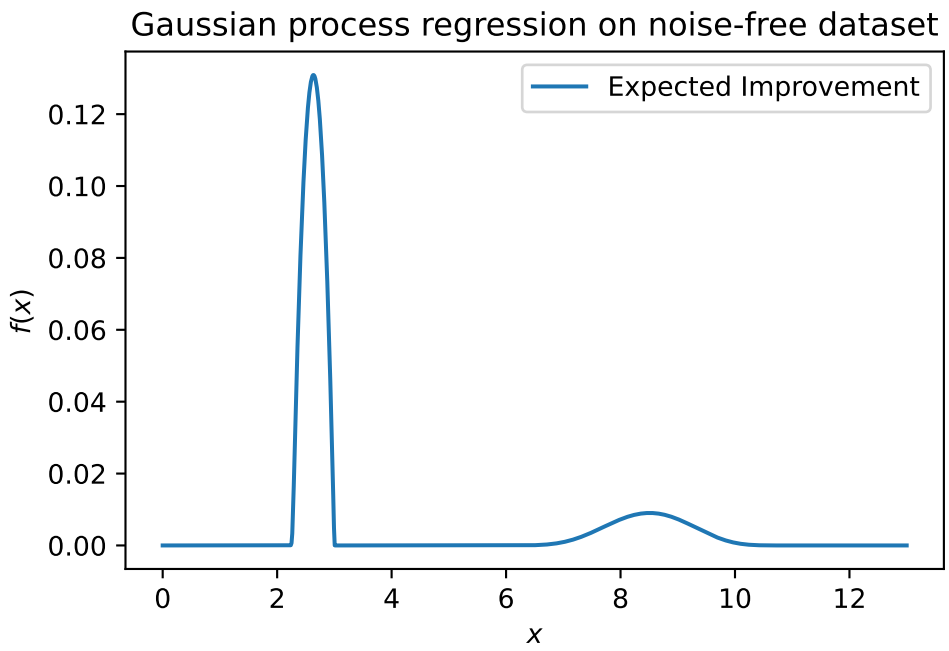
```
X_train0 = np.array([1, 2, 3, 4, 12]).reshape(-1,1)
X_train = np.linspace(start=0, stop=10, num=5).reshape(-1, 1)
```

```
from spotPython.build.kriging import Kriging
import numpy as np
import matplotlib.pyplot as plt

X_train = np.array([1., 2., 3., 4., 12.]).reshape(-1,1)
y_train = np.array([0., -1.75, -2, -0.5, 5.])

S = Kriging(name='kriging',  seed=123, log_level=50, n_theta=1, noise=False, cod_type="nor
S.fit(X_train, y_train)

X = np.linspace(start=0, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
if True:
    plt.fill_between(
        X.ravel(),
        mean_prediction - 2 * std_prediction,
        mean_prediction + 2 * std_prediction,
        alpha=0.5,
        label=r"95% confidence interval",
    )
plt.legend()
plt.xlabel("$x$")
```

```
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noise-free dataset")
```




```
#plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
# plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, -ei, label="Expected Improvement")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noise-free dataset")
```



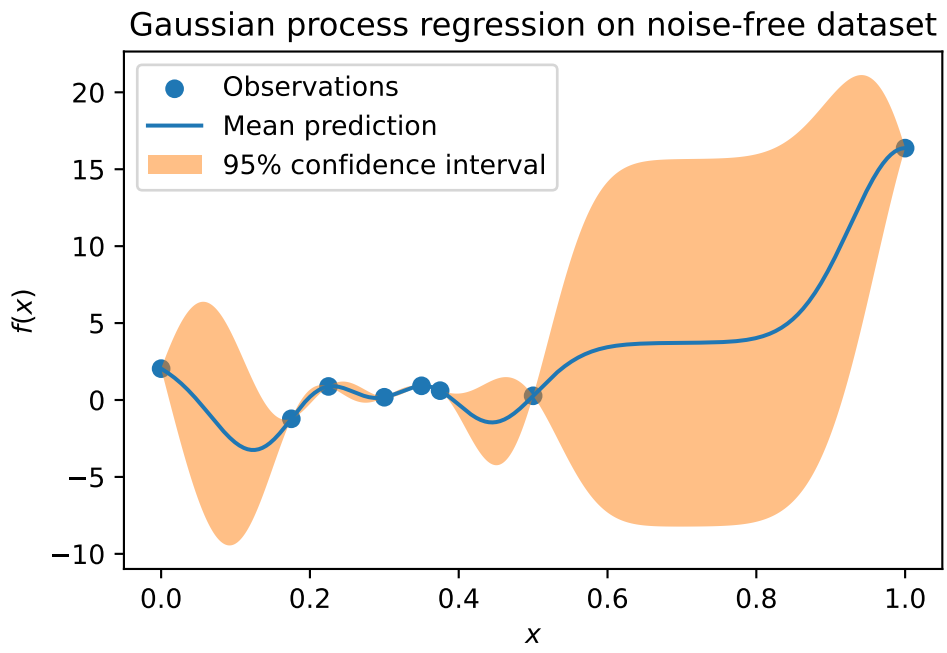

```
S.log
```

```
{'negLnLike': array([1.20788205]),
 'theta': array([1.09276]),
 'p': array([2.]),
 'Lambda': array([None], dtype=object)}
```

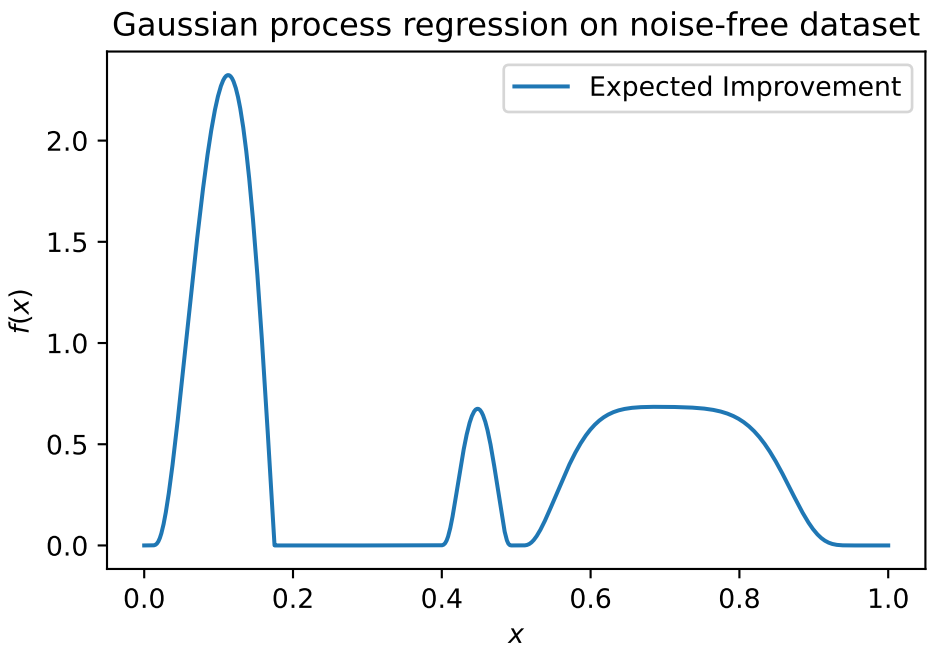
12.10 EI: The Forrester Example

```
from spotPython.build.kriging import Kriging
import numpy as np
import matplotlib.pyplot as plt
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot

# exact x locations are unknown:
X_train = np.array([0.0, 0.175, 0.225, 0.3, 0.35, 0.375, 0.5,1]).reshape(-1,1)
```

```
fun = analytical().fun_forrester
fun_control = {"sigma": 1.0,
               "seed": 123}
y_train = fun(X_train, fun_control=fun_control)

S = Kriging(name='kriging',  seed=123, log_level=50, n_theta=1, noise=False, cod_type="nor
S.fit(X_train, y_train)

X = np.linspace(start=0, stop=1, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
if True:
    plt.fill_between(
        X.ravel(),
        mean_prediction - 2 * std_prediction,
        mean_prediction + 2 * std_prediction,
        alpha=0.5,
        label=r"95% confidence interval",
    )
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noise-free dataset")
```




```
#plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
# plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, -ei, label="Expected Improvement")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noise-free dataset")
```

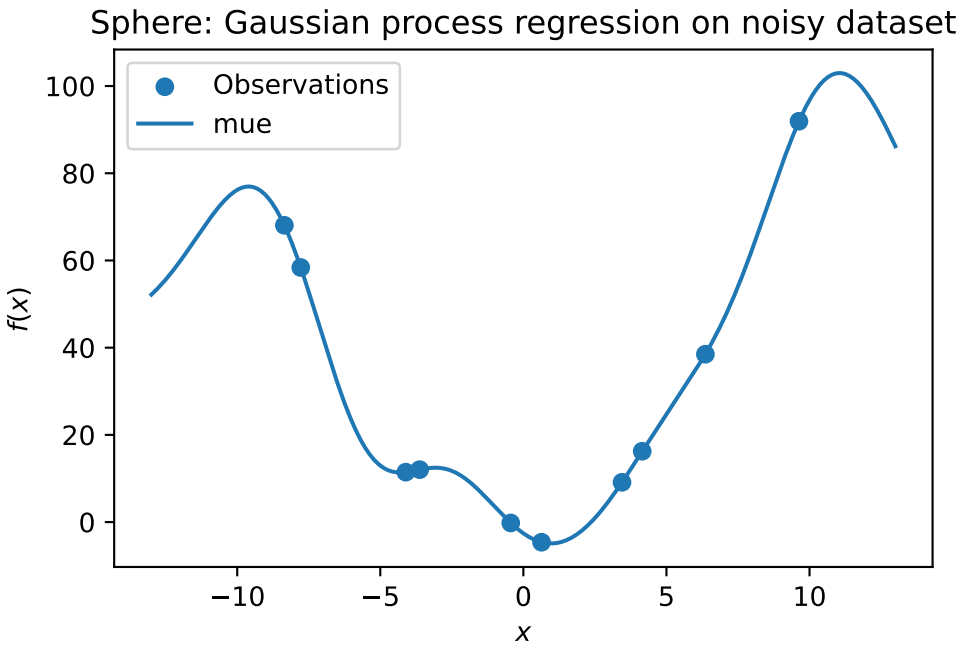


12.11 Noise

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_sphere
fun_control = {"sigma": 2,
               "seed": 125}
X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
```

```
print(y)
y.shape
X_train = X.reshape(-1,1)
y_train = y

S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

#plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression on noisy dataset")
```

```
[[ 0.63529627]
 [-4.10764204]
 [-0.44071975]
 [ 9.63125638]
 [-8.3518118 ]
 [-3.62418901]
 [ 4.15331   ]
 [ 3.4468512 ]
 [ 6.36049088]
 [-7.77978539]]
[-4.61635371 11.44873209 -0.19988024 91.92791676 68.05926244 12.02926818
 16.2470957   9.12729929 38.4987029  58.38469104]
```

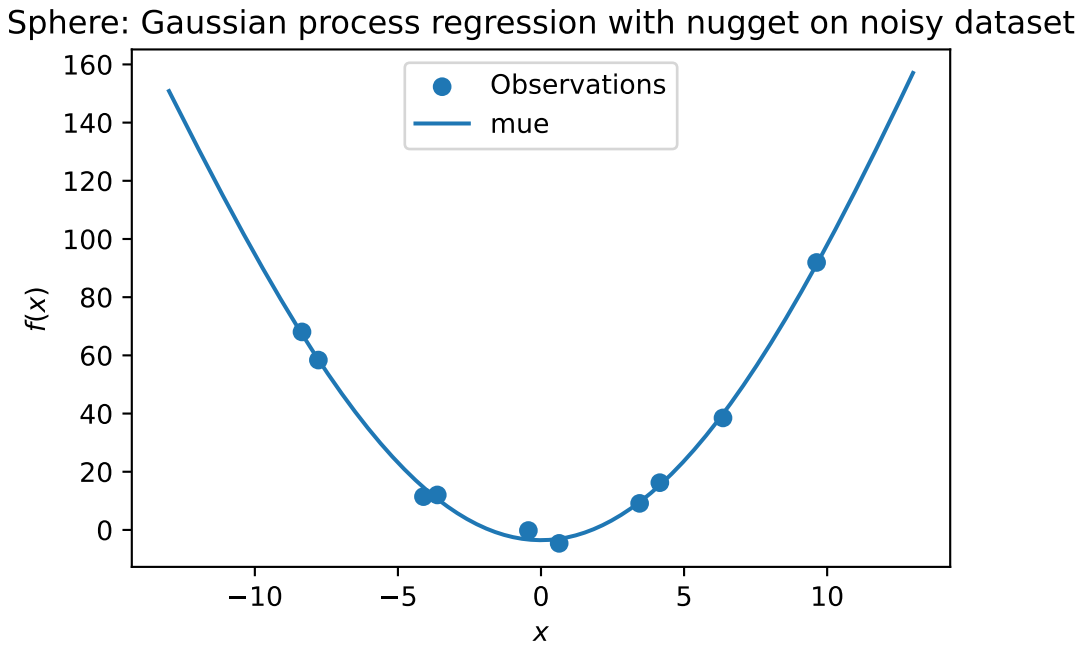



```
S.log
```

```
{'negLnLike': array([24.69806131]),
 'theta': array([1.31023943]),
 'p': array([2.]),
 'Lambda': array([None], dtype=object)}
```

```
S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=True)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

#plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
```

```
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression with nugget on noisy dataset")
```




```
S.log
```

```
{'negLnLike': array([22.14095646]),
 'theta': array([-0.32527397]),
 'p': array([2.]),
 'Lambda': array([9.08815007e-05])}
```

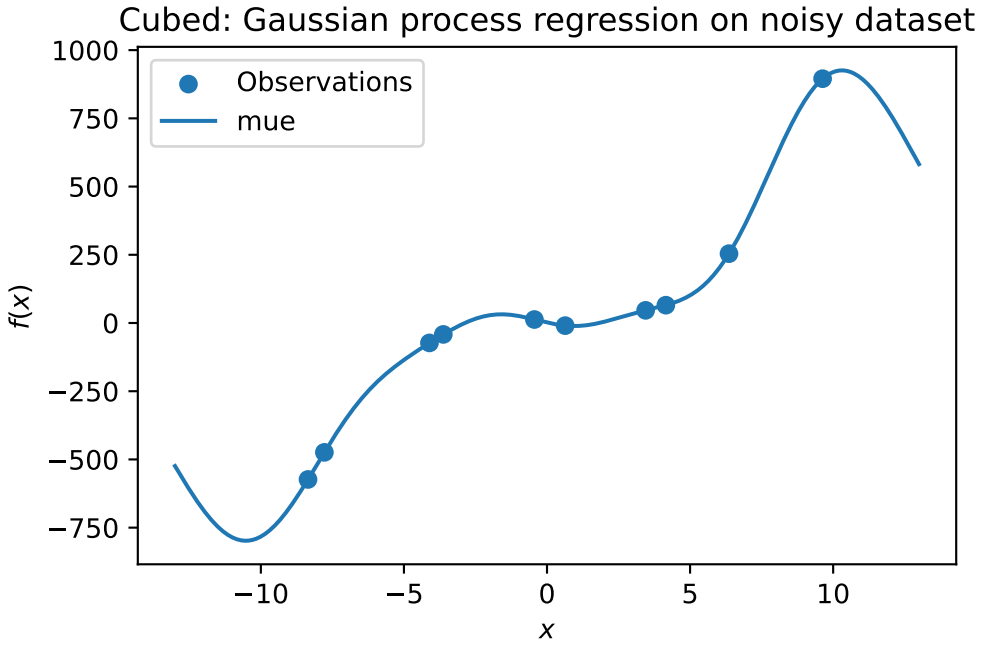
12.12 Cubic Function

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
```

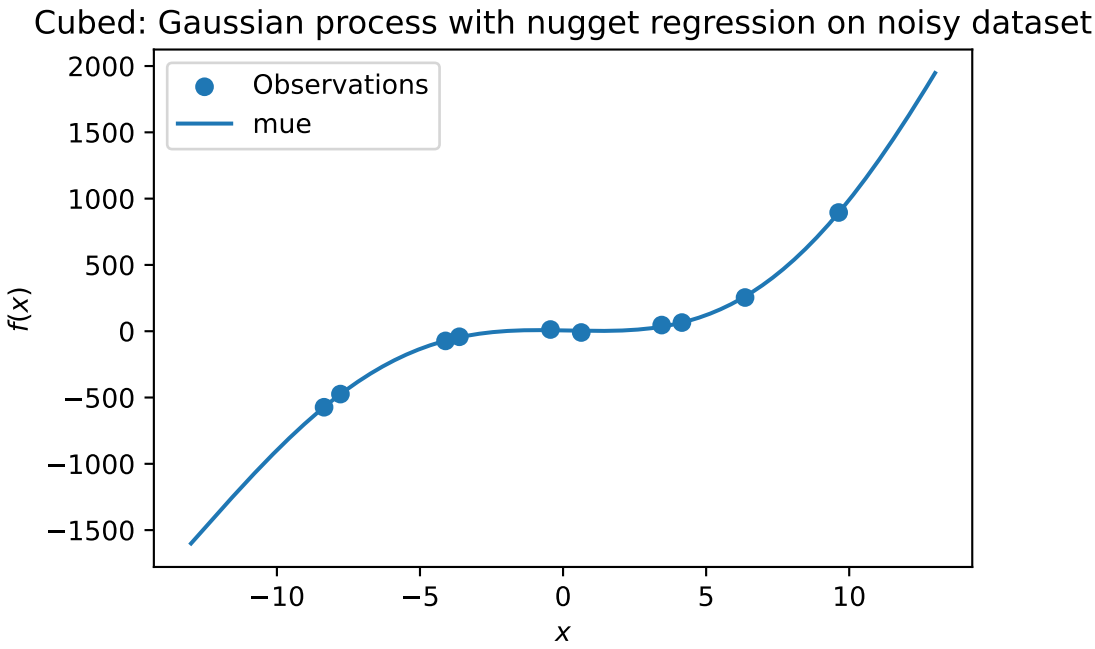
```
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_cubed
fun_control = {"sigma": 10,
               "seed": 123}

X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
print(y)
y.shape
X_train = X.reshape(-1,1)
y_train = y

S = Kriging(name='kriging',  seed=123, log_level=50, n_theta=1, noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Cubed: Gaussian process regression on noisy dataset")
```

```
[[ 0.63529627]
 [-4.10764204]
 [-0.44071975]
 [ 9.63125638]
 [-8.3518118 ]
 [-3.62418901]
 [ 4.15331   ]
 [ 3.4468512 ]
 [ 6.36049088]
```

```
 [-7.77978539]]
[  -9.63480707  -72.98497325   12.7936499   895.34567477 -573.35961837
  -41.83176425   65.27989461   46.37081417  254.1530734  -474.09587355]
```



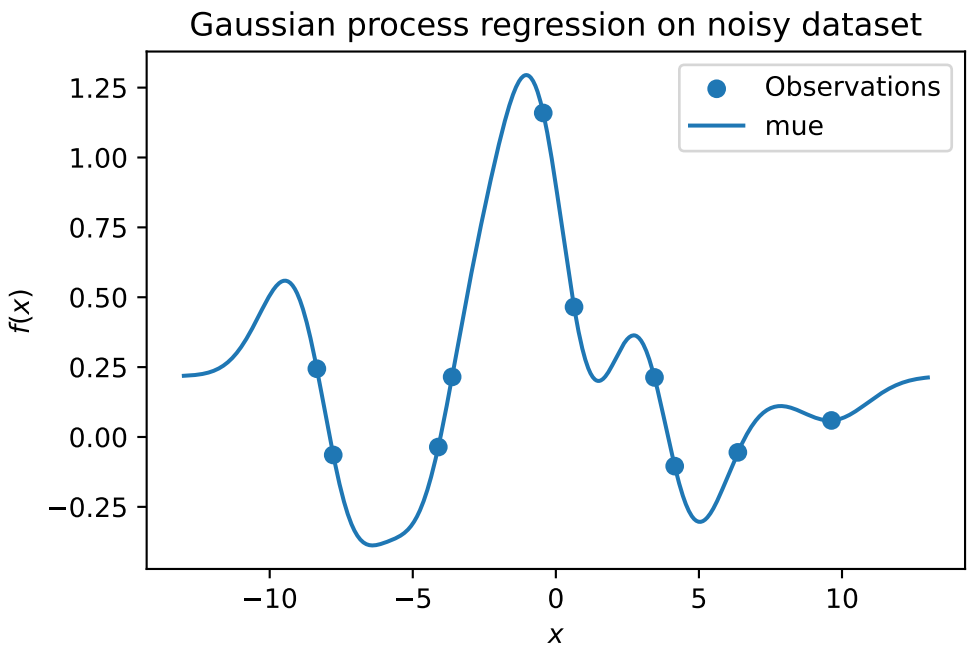

```
S = Kriging(name='kriging',  seed=123, log_level=0, n_theta=1, noise=True)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Cubed: Gaussian process with nugget regression on noisy dataset")
```



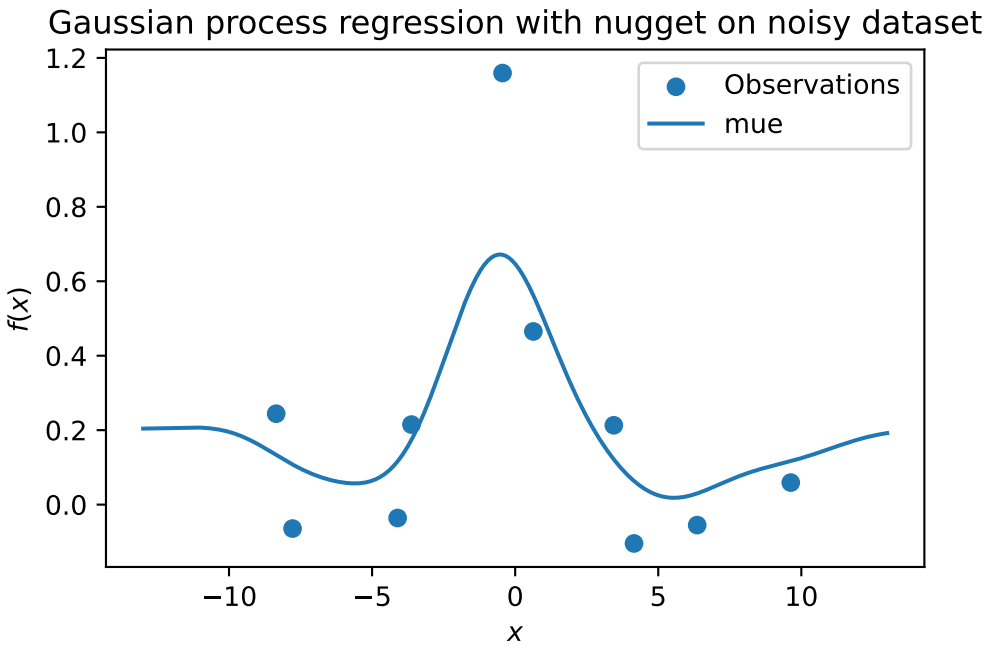

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_runge
fun_control = {"sigma": 0.25,
               "seed": 123}

X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
print(y)
y.shape
```

```
X_train = X.reshape(-1,1)
y_train = y

S = Kriging(name='kriging',  seed=123, log_level=50, n_theta=1, noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noisy dataset")
```

```
[[ 0.63529627]
 [-4.10764204]
 [-0.44071975]
 [ 9.63125638]
 [-8.3518118 ]
 [-3.62418901]
 [ 4.15331   ]
 [ 3.4468512 ]
 [ 6.36049088]
 [-7.77978539]]
[ 0.46517267 -0.03599548  1.15933822  0.05915901  0.24419145  0.21502359
 -0.10432134  0.21312309 -0.05502681 -0.06434374]
```




```
S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=True)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression with nugget on noisy dataset")
```



12.13 Factors

```
["num"] * 3
```

```
['num', 'num', 'num']
```

```
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
from spotPython.fun.objectivefunctions import analytical
import numpy as np
```

```
gen = spacefilling(2)
n = 30
rng = np.random.RandomState(1)
lower = np.array([-5,-0])
upper = np.array([10,15])
fun = analytical().fun_branin_factor
#fun = analytical(sigma=0).fun_sphere
```

```
X0 = gen.scipy_lhd(n, lower=lower, upper = upper)
X1 = np.random.randint(low=1, high=3, size=(n,))
X = np.c_[X0, X1]
y = fun(X)
S = Kriging(name='kriging',  seed=123, log_level=50, n_theta=3, noise=False, var_type=["nu
S.fit(X, y)
Sf = Kriging(name='kriging',  seed=123, log_level=50, n_theta=3, noise=False, var_type=["n
Sf.fit(X, y)
n = 50
X0 = gen.scipy_lhd(n, lower=lower, upper = upper)
X1 = np.random.randint(low=1, high=3, size=(n,))
X = np.c_[X0, X1]
y = fun(X)
s=np.sum(np.abs(S.predict(X)[0] - y))
sf=np.sum(np.abs(Sf.predict(X)[0] - y))
sf - s
```

```
128.19635925436705
```

```
# vars(S)
```

```
# vars(Sf)
```

13 Hyperparameter Tuning and Noise

This chapter demonstrates how noisy functions can be handled by `Spot`.

13.1 Example: `Spot` and the Noisy Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

13.1.1 The Objective Function: Noisy Sphere

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function with noise, i.e., a function that can be described by a (closed) formula:

$$f(x) = x^2 + \epsilon$$

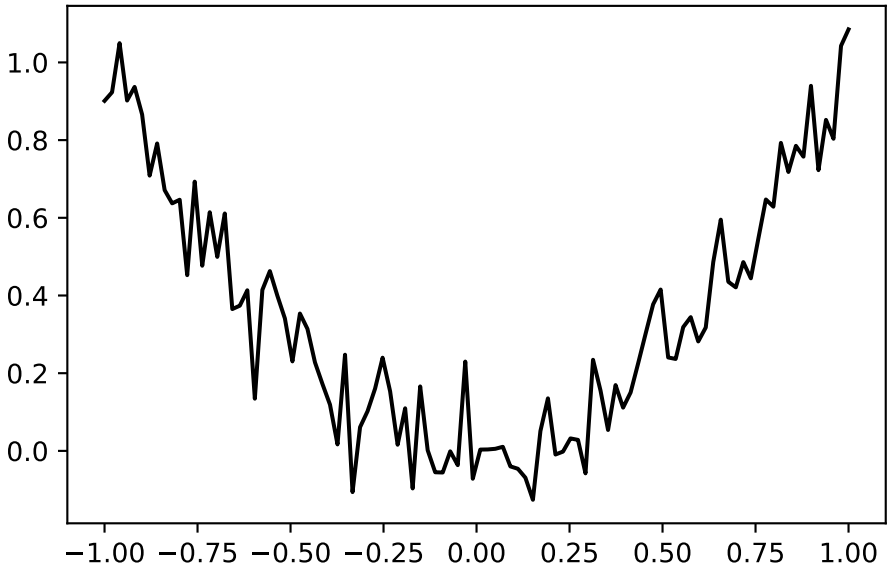
- Since `sigma` is set to `0.1`, noise is added to the function:

```
fun = analytical().fun_sphere
fun_control = {"sigma": 0.1,
               "seed": 123}
```

- A plot illustrates the noise:

```
x = np.linspace(-1,1,100).reshape(-1,1)
y = fun(x, fun_control=fun_control)
plt.figure()
```

```
plt.plot(x,y, "k")
plt.show()
```

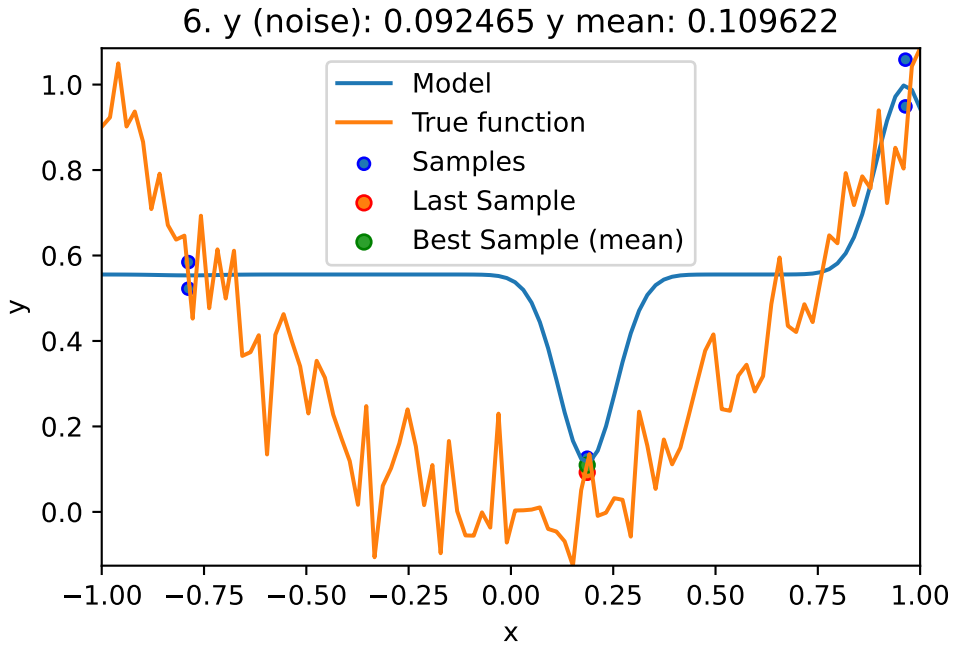


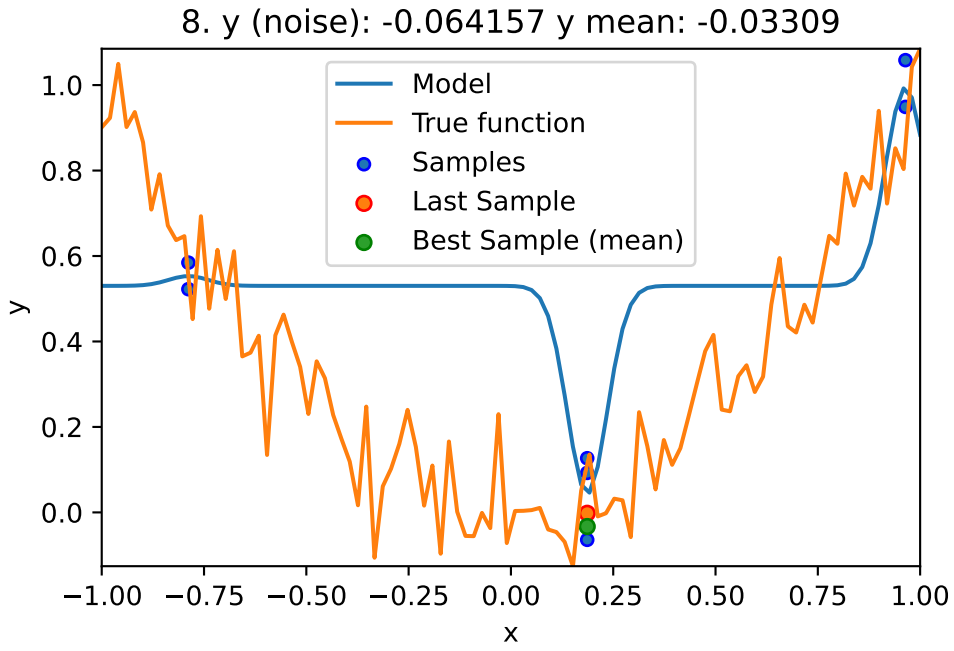
`Spot` is adopted as follows to cope with noisy functions:

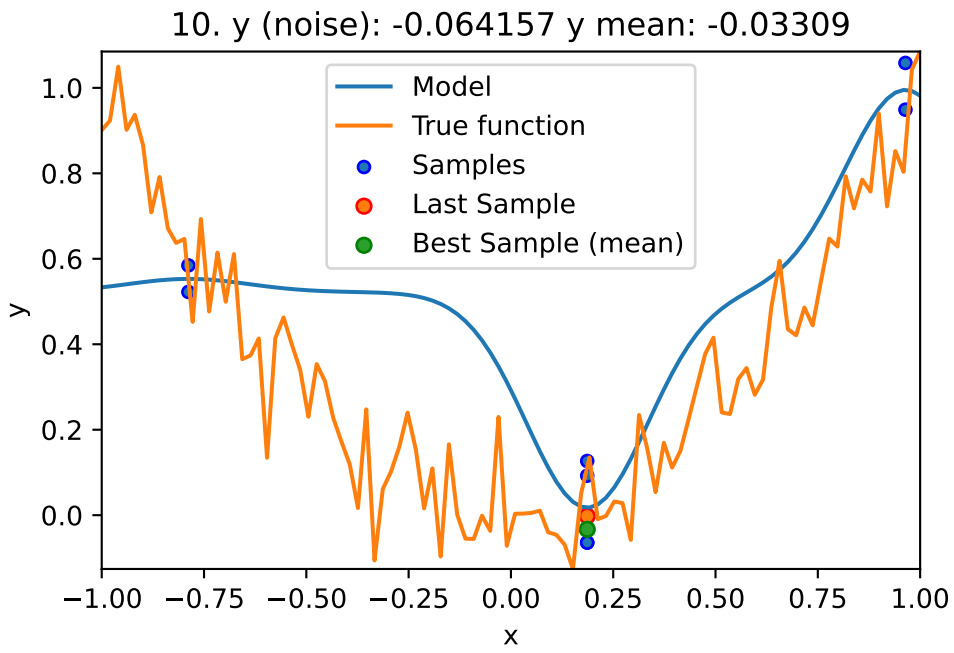
1. `fun_repeats` is set to a value larger than 1 (here: 2)
2. `noise` is set to `true`. Therefore, a nugget (`Lambda`) term is added to the correlation matrix
3. `init size` (of the `design_control` dictionary) is set to a value larger than 1 (here: 2)

```
spot_1_noisy = spot.Spot(fun=fun,
                    lower = np.array([-1]),
                    upper = np.array([1]),
                    fun_evals = 10,
                    fun_repeats = 2,
                    noise = True,
                    seed=123,
                    show_models=True,
                    fun_control = fun_control,
                    design_control={"init_size": 3,
                                    "repeats": 2},
                    surrogate_control={"noise": True})
```

```
spot_1_noisy.run()
```







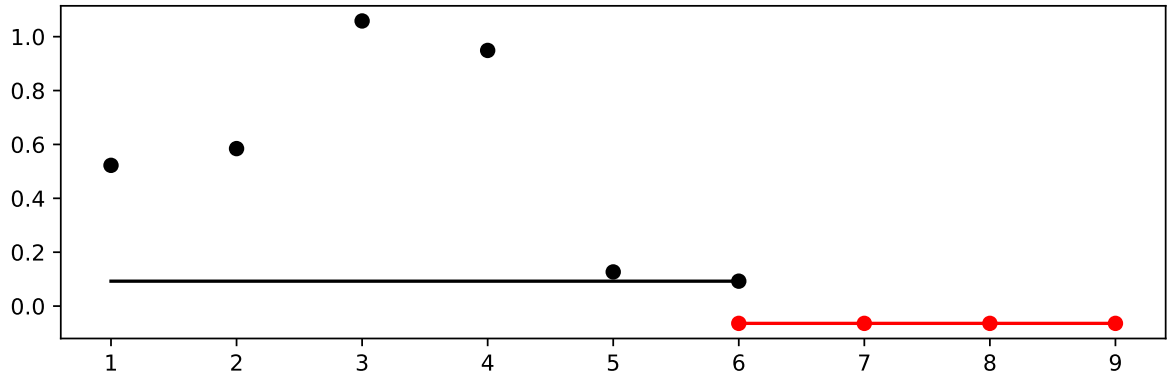

```
<spotPython.spot.spot.Spot at 0x15fbbf760>
```

13.2 Print the Results

```
spot_1_noisy.print_results()
```

```
min y: -0.06415721594238855
x0: 0.18642671238960512
min mean y: -0.03309048099839016
x0: 0.18642671238960512

[['x0', 0.18642671238960512], ['x0', 0.18642671238960512]]
```

```
spot_1_noisy.plot_progress(log_y=False)
```



13.3 Noise and Surrogates: The Nugget Effect

13.3.1 The Noisy Sphere

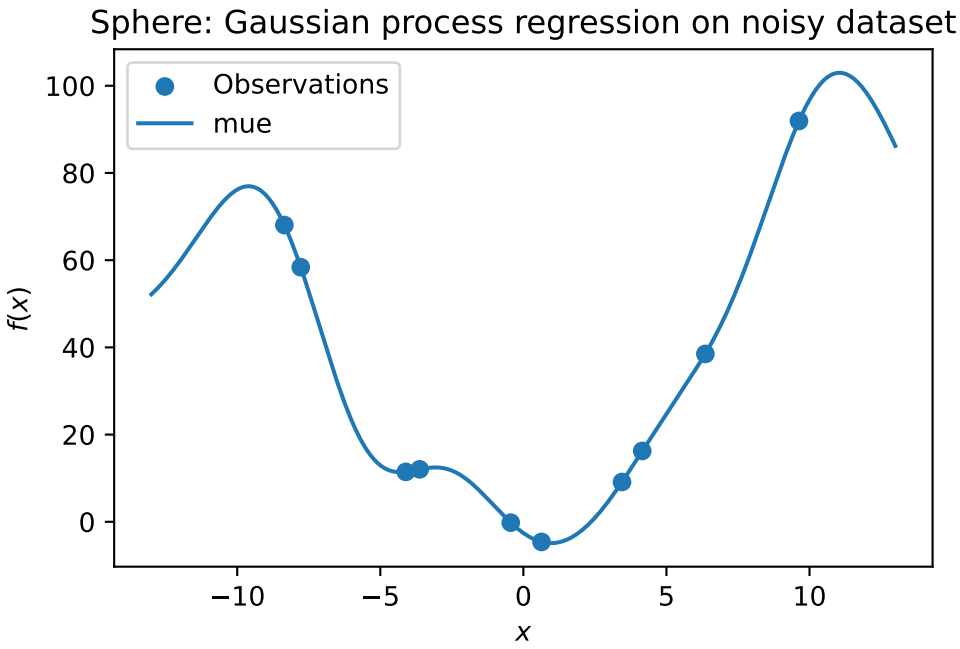
13.3.1.1 The Data

- We prepare some data first:

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_sphere
fun_control = {"sigma": 2,
               "seed": 125}
X = gen.scipy_lhd(10, lower=lower, upper = upper)
y = fun(X, fun_control=fun_control)
X_train = X.reshape(-1,1)
y_train = y
```

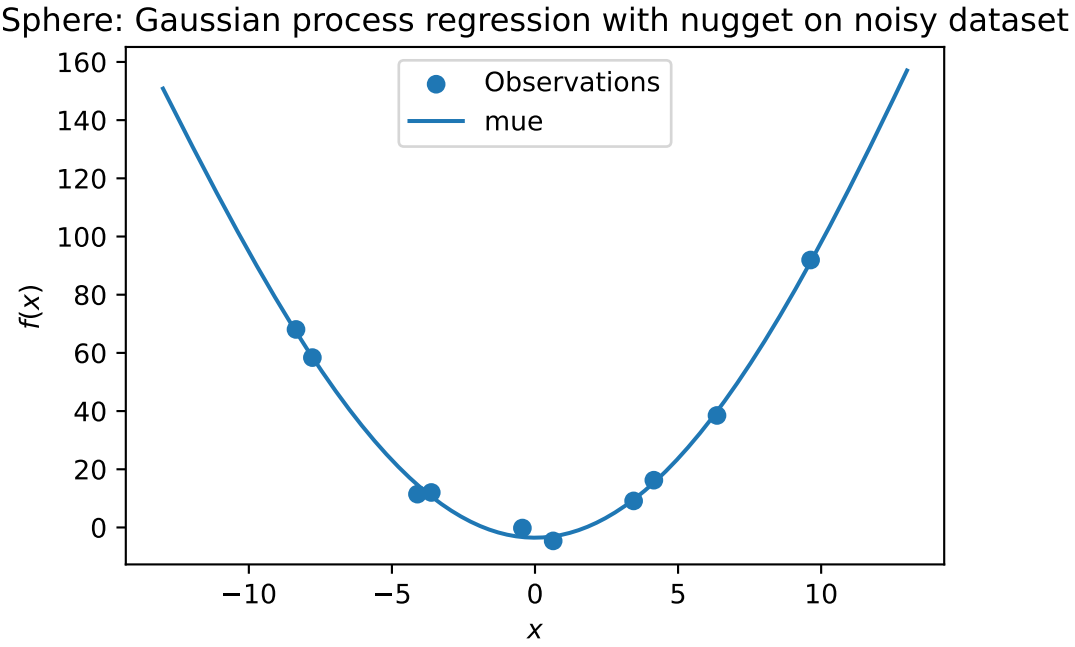
- A surrogate without nugget is fitted to these data:

```
S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression on noisy dataset")
```



- In comparison to the surrogate without nugget, we fit a surrogate with nugget to the data:

```
S_nug = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=True)
S_nug.fit(X_train, y_train)
X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S_nug.predict(X_axis, return_val="all")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression with nugget on noisy dataset")
```



- The value of the nugget term can be extracted from the model as follows:

```
S.Lambda
```

```
S_nug.Lambda
```

```
9.088150066416743e-05
```

- We see:
  - the first model `S` has no nugget,
  - whereas the second model has a nugget value (`Lambda`) larger than zero.

13.4 Exercises

13.4.1 Noisy `fun_cubed`

- Analyse the effect of noise on the `fun_cubed` function with the following settings:

```
fun = analytical().fun_cubed
fun_control = {"sigma": 10,
               "seed": 123}
lower = np.array([-10])
upper = np.array([10])
```

13.4.2 `fun_runge`

- Analyse the effect of noise on the `fun_runge` function with the following settings:

```
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_runge
fun_control = {"sigma": 0.25,
               "seed": 123}
```

13.4.3 `fun_forrester`

- Analyse the effect of noise on the `fun_forrester` function with the following settings:

```
lower = np.array([0])
upper = np.array([1])
fun = analytical().fun_forrester
fun_control = {"sigma": 5,
               "seed": 123}
```

13.4.4 `fun_xsin`

- Analyse the effect of noise on the `fun_xsin` function with the following settings:

```
lower = np.array([-1.])
upper = np.array([1.])
fun = analytical().fun_xsin
fun_control = {"sigma": 0.5,
               "seed": 123}
```

14 Handling Noise: Optimal Computational Budget Allocation in Spot

This notebook demonstrates how noisy functions can be handled with OCBA by Spot.

14.1 Example: Spot, OCBA, and the Noisy Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

14.1.1 The Objective Function: Noisy Sphere

The spotPython package provides several classes of objective functions. We will use an analytical objective function with noise, i.e., a function that can be described by a (closed) formula:

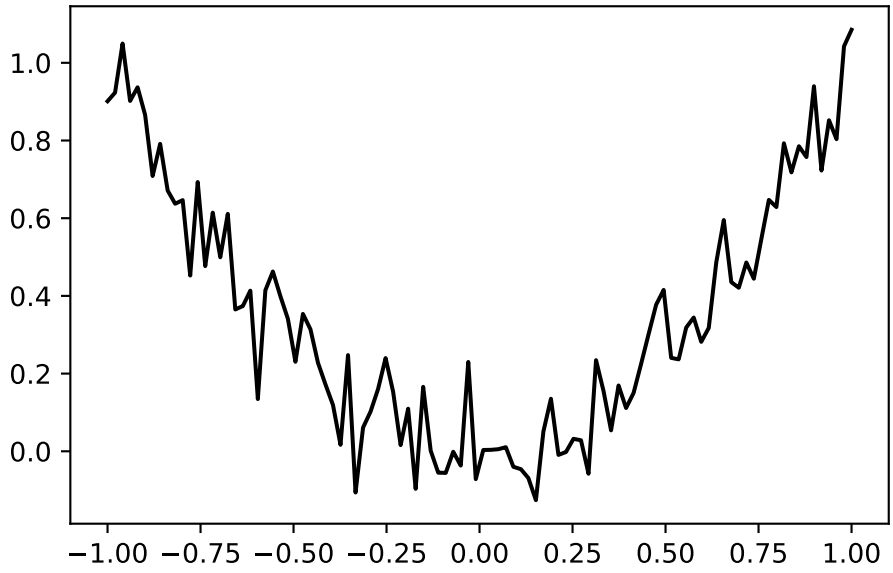
$$f(x) = x^2 + \epsilon$$

Since sigma is set to 0.1, noise is added to the function:

```
fun = analytical().fun_sphere
fun_control = {"sigma": 0.1,
               "seed": 123}
```

A plot illustrates the noise:

```
x = np.linspace(-1,1,100).reshape(-1,1)
y = fun(x, fun_control=fun_control)
plt.figure()
plt.plot(x,y, "k")
plt.show()
```



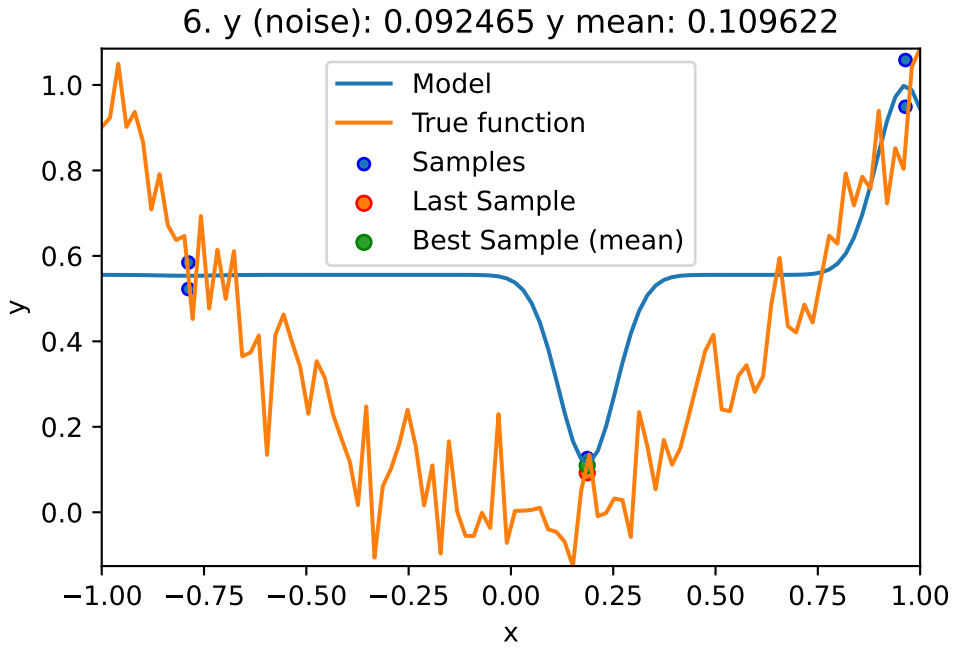
`Spot` is adopted as follows to cope with noisy functions:

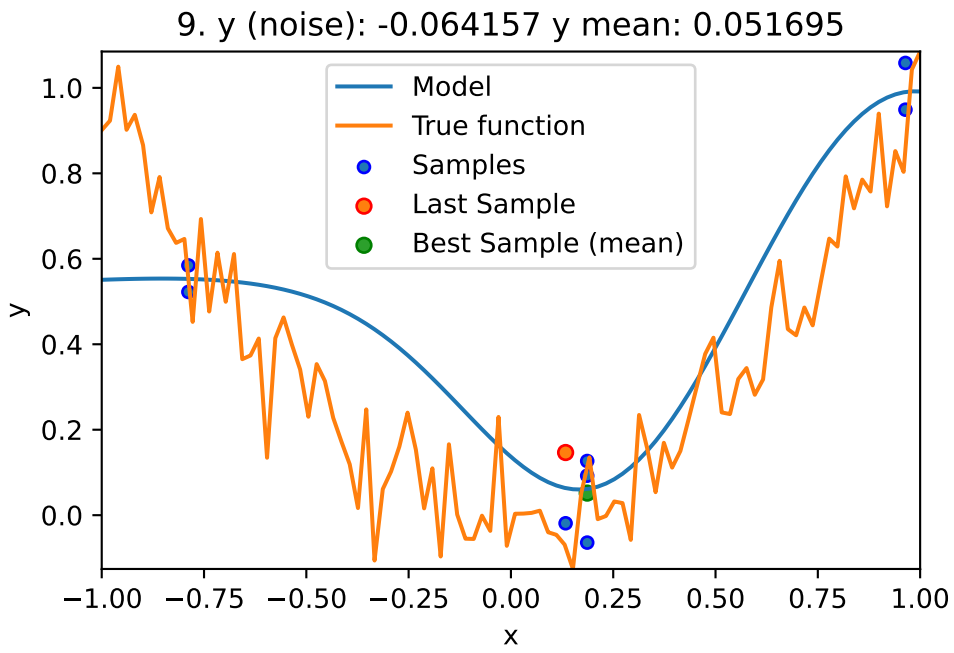
1. `fun_repeats` is set to a value larger than 1 (here: 2)
2. `noise` is set to `true`. Therefore, a nugget (`Lambda`) term is added to the correlation matrix
3. `init size` (of the `design_control` dictionary) is set to a value larger than 1 (here: 2)

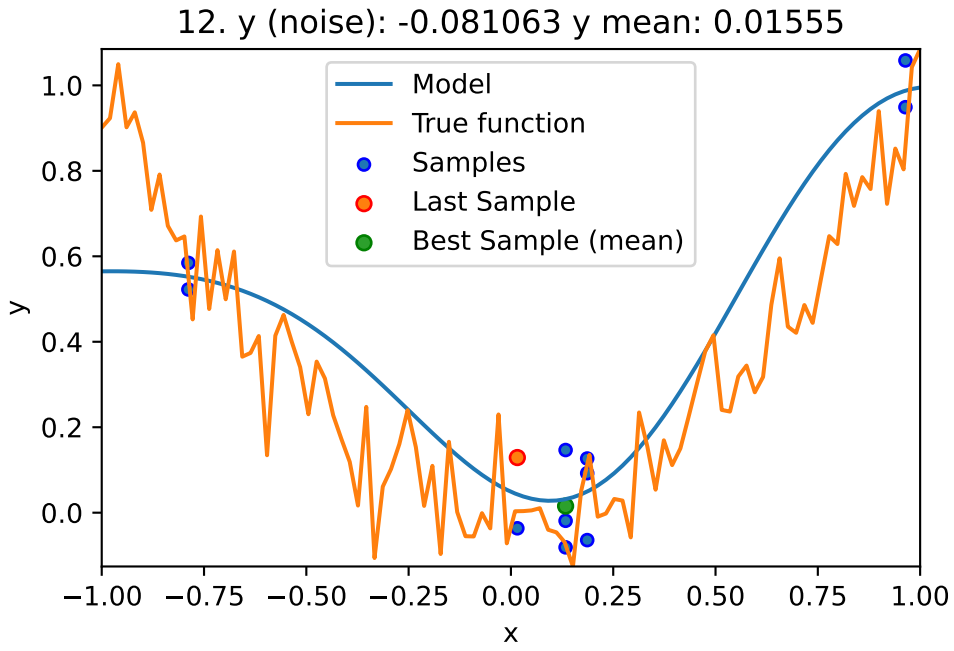
```
spot_1_noisy = spot.Spot(fun=fun,
                    lower = np.array([-1]),
                    upper = np.array([1]),
                    fun_evals = 50,
                    fun_repeats = 2,
                    infill_criterion="ei",
                    noise = True,
                    tolerance_x=0.0,
                    ocba_delta = 1,
                    seed=123,
```

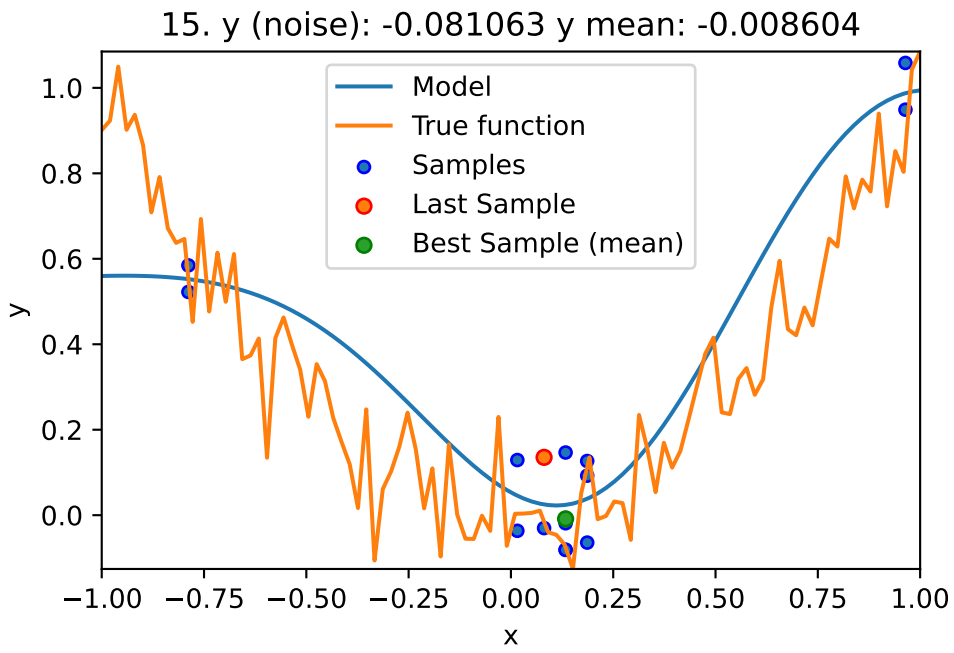
```
                    show_models=True,
                    fun_control = fun_control,
                    design_control={"init_size": 3,
                                    "repeats": 2},
                    surrogate_control={"noise": True})
```

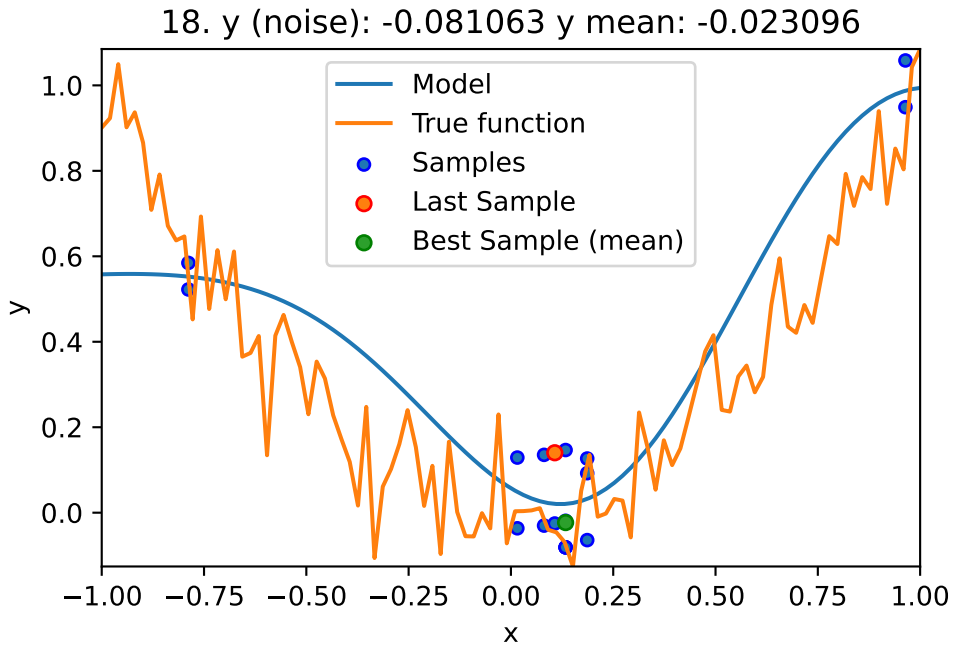
```
spot_1_noisy.run()
```

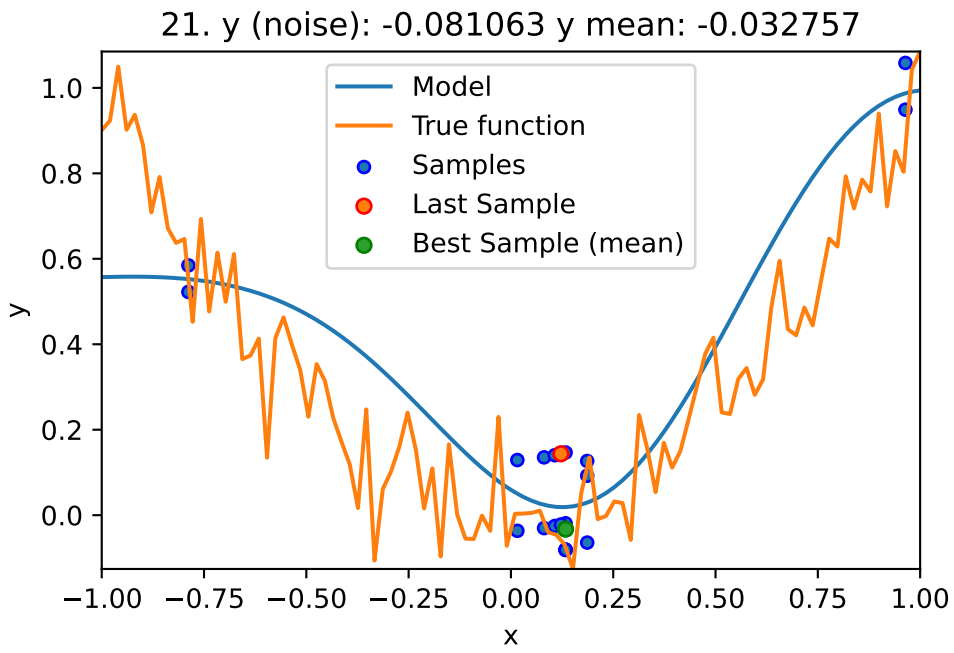


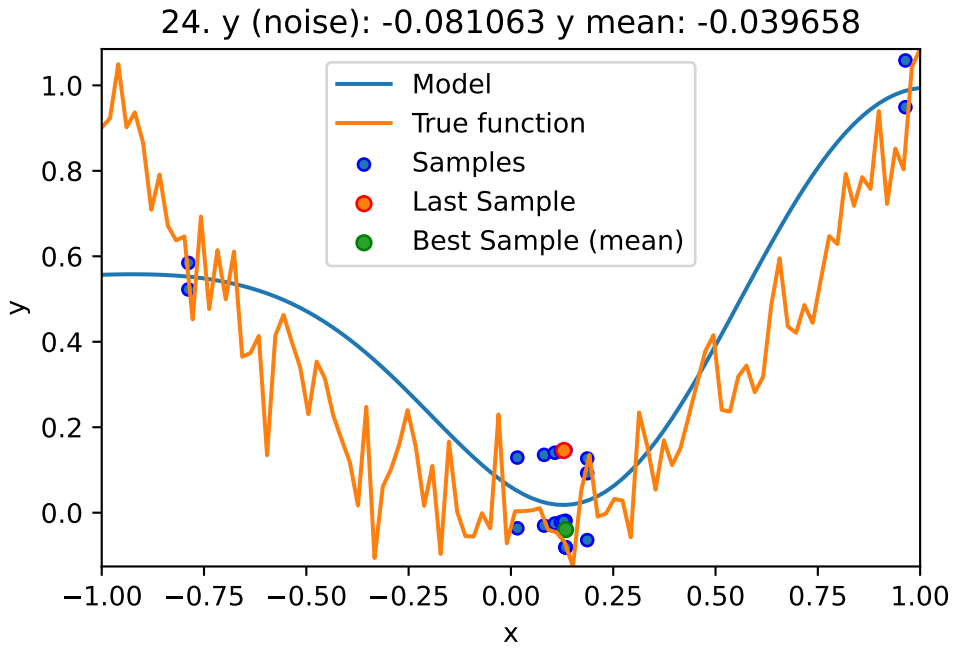

9. y (noise): -0.064157 y mean: 0.051695
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

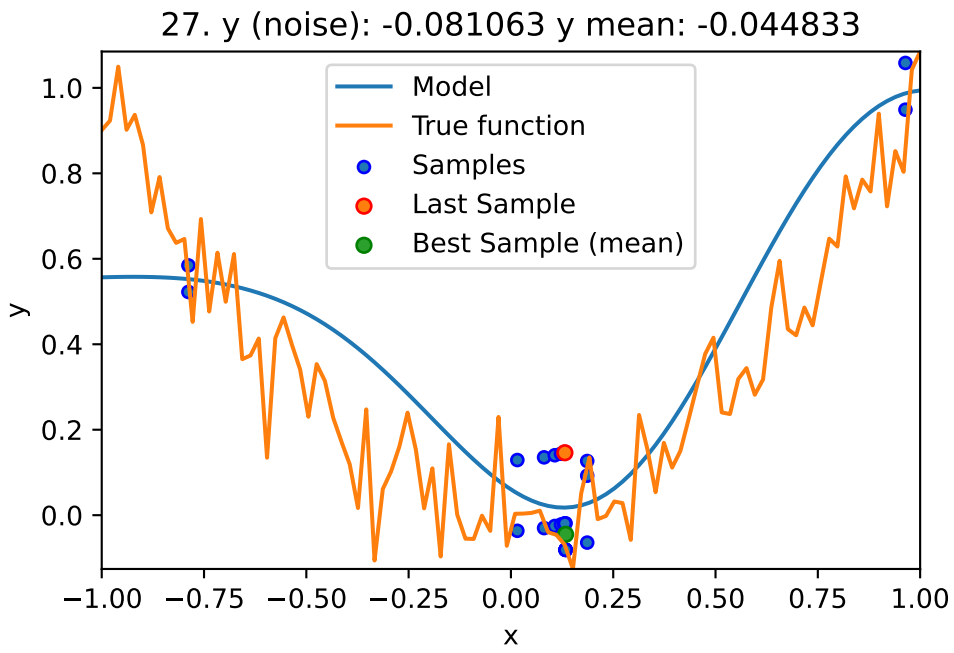

12. y (noise): -0.081063 y mean: 0.01555
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

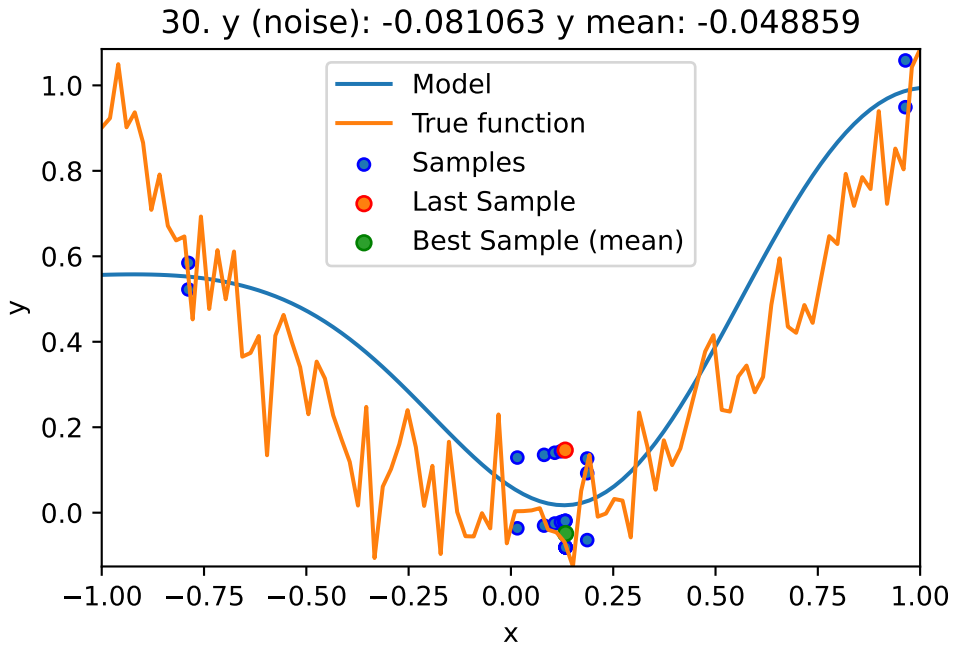

15. y (noise): -0.081063 y mean: -0.008604
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

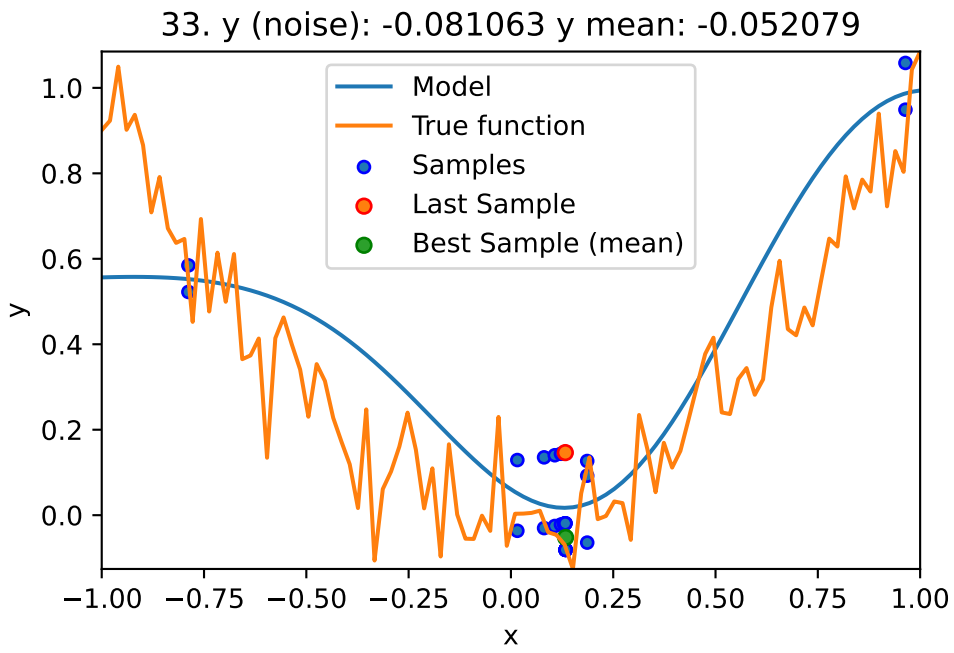

18. y (noise): -0.081063 y mean: -0.023096
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

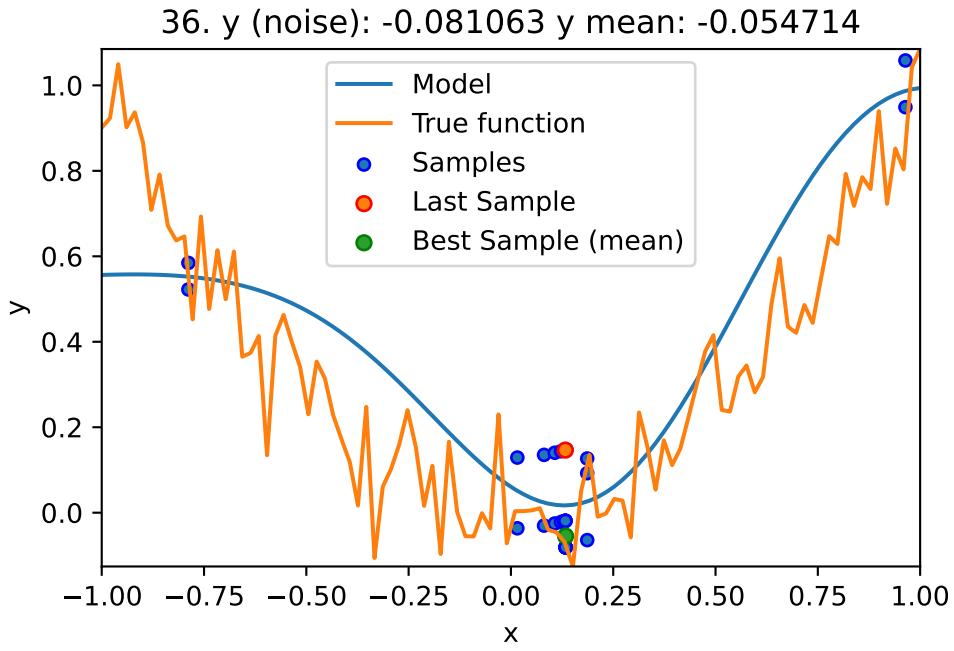

21. y (noise): -0.081063 y mean: -0.032757
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

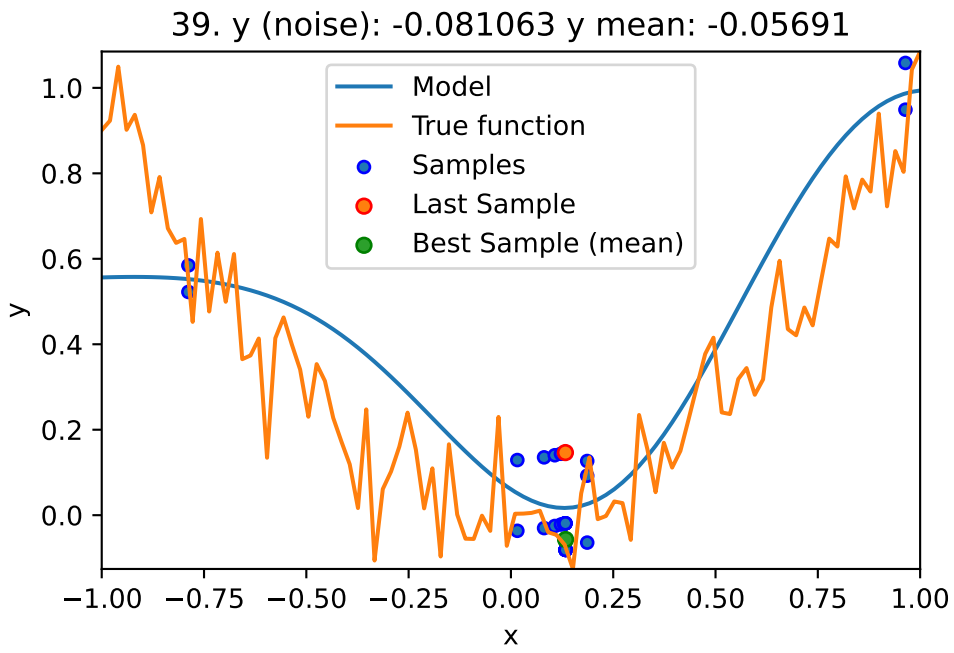

24. y (noise): -0.081063 y mean: -0.039658
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

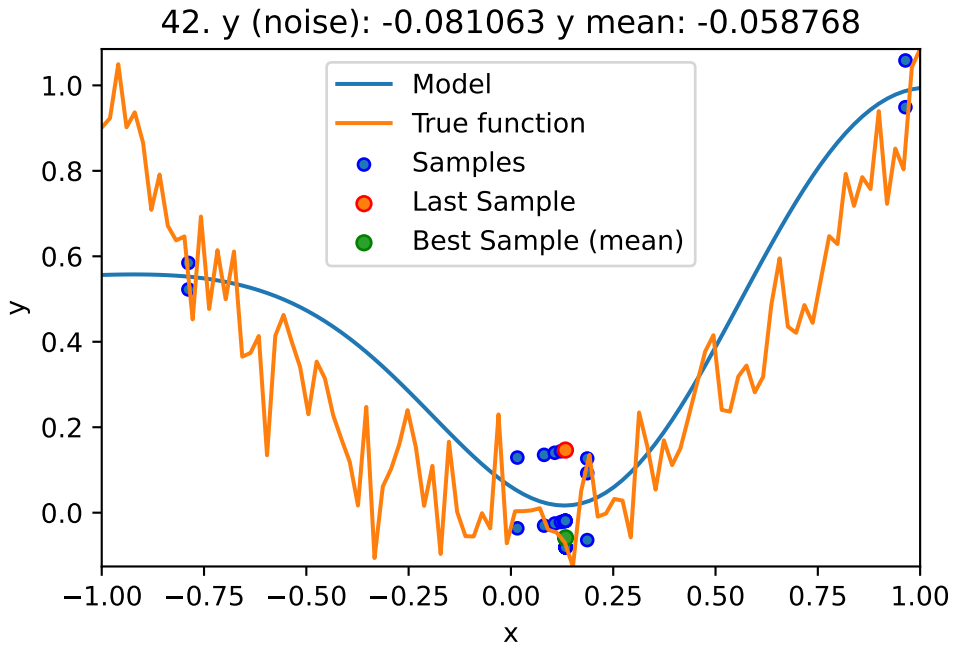

27. y (noise): -0.081063 y mean: -0.044833
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

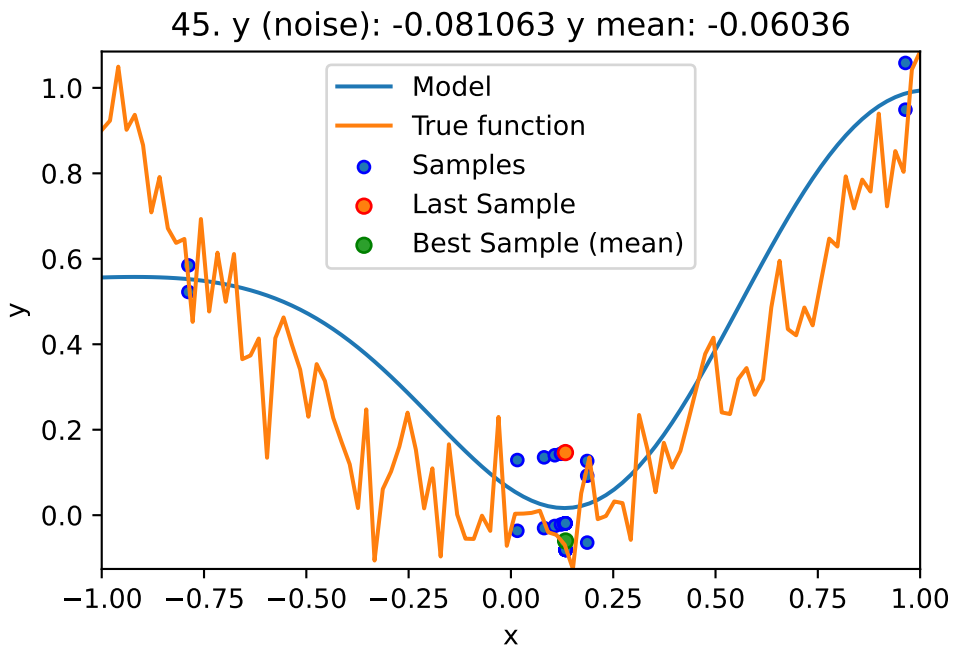

30. y (noise): -0.081063 y mean: -0.048859
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

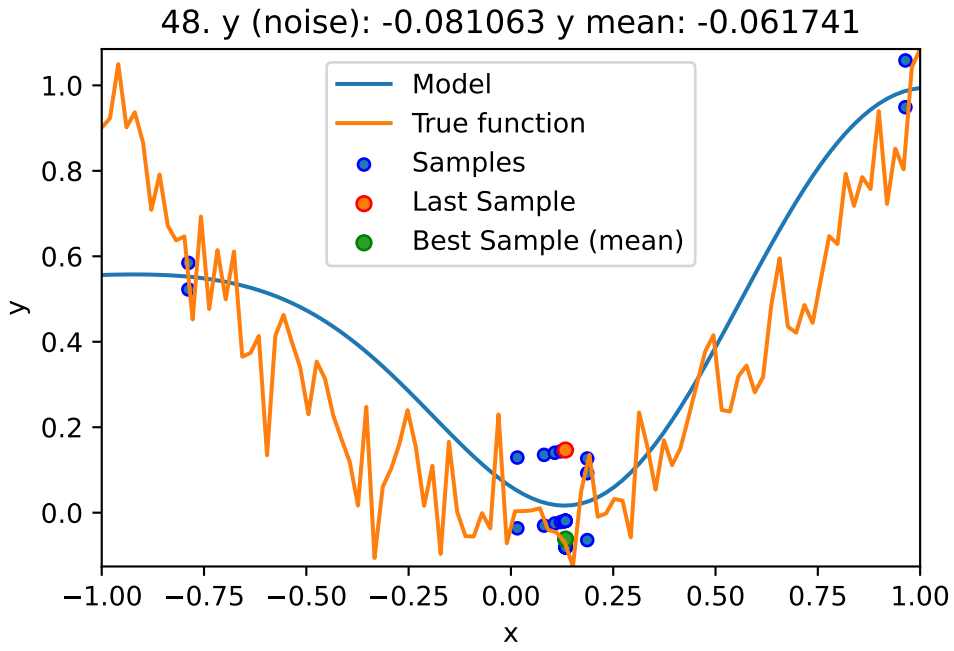

33. y (noise): -0.081063 y mean: -0.052079
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

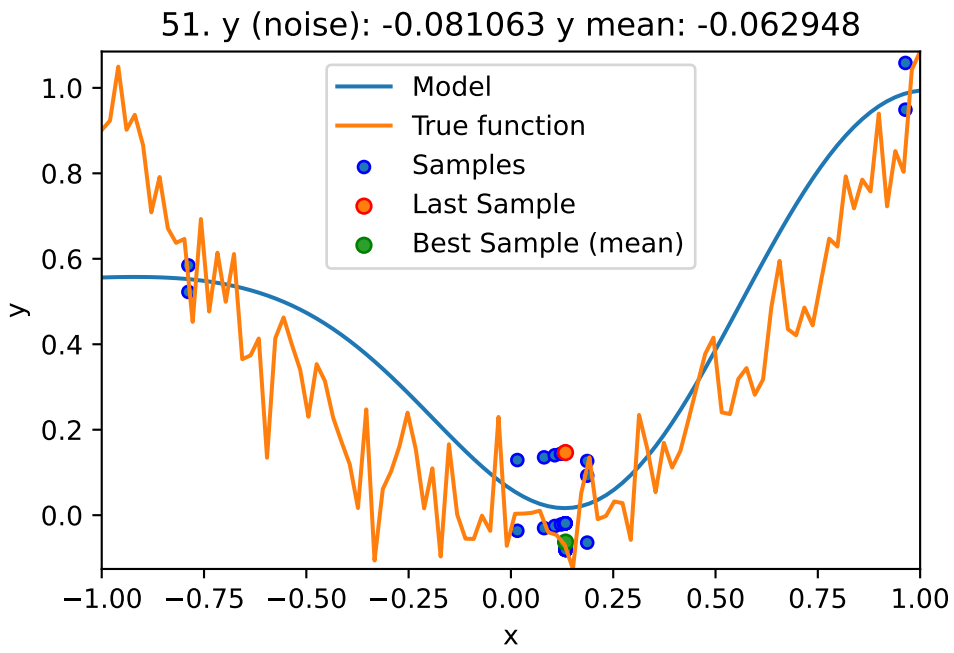

36. y (noise): -0.081063 y mean: -0.054714
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


39. y (noise): -0.081063 y mean: -0.05691
Model
True function
Samples
Last Sample
Best Sample (mean)
1.0
0.8
0.6
0.4
0.2
0.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


42. y (noise): -0.081063 y mean: -0.058768
Model
True function
Samples
Last Sample
Best Sample (mean)
1.0
0.8
0.6
0.4
0.2
0.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


45. y (noise): -0.081063 y mean: -0.06036
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


48. y (noise): -0.081063 y mean: -0.061741
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



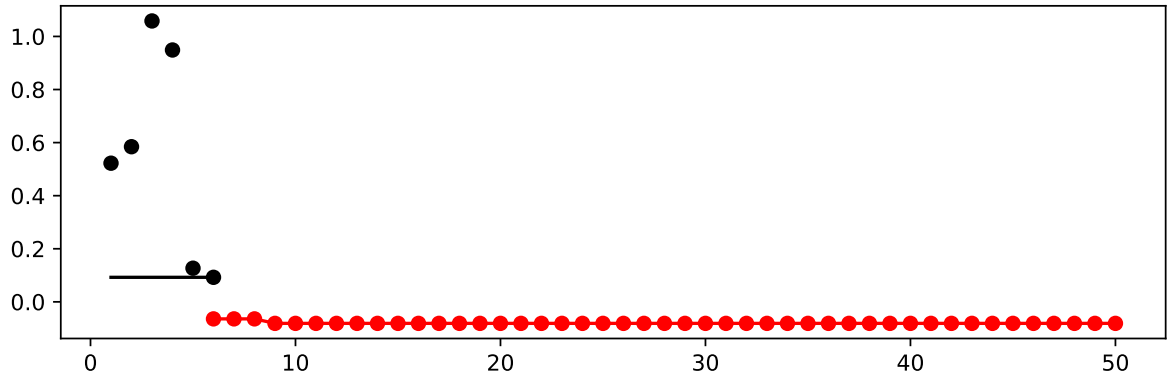

```
<spotPython.spot.spot.Spot at 0x17fdeb760>
```

14.2 Print the Results

```
spot_1_noisy.print_results()
```

```
min y: -0.08106318979661208
x0: 0.1335999447536301
min mean y: -0.06294830660588041
x0: 0.1335999447536301

[['x0', 0.1335999447536301], ['x0', 0.1335999447536301]]
```

```
spot_1_noisy.plot_progress(log_y=False)
```



14.3 Noise and Surrogates: The Nugget Effect

14.3.1 The Noisy Sphere

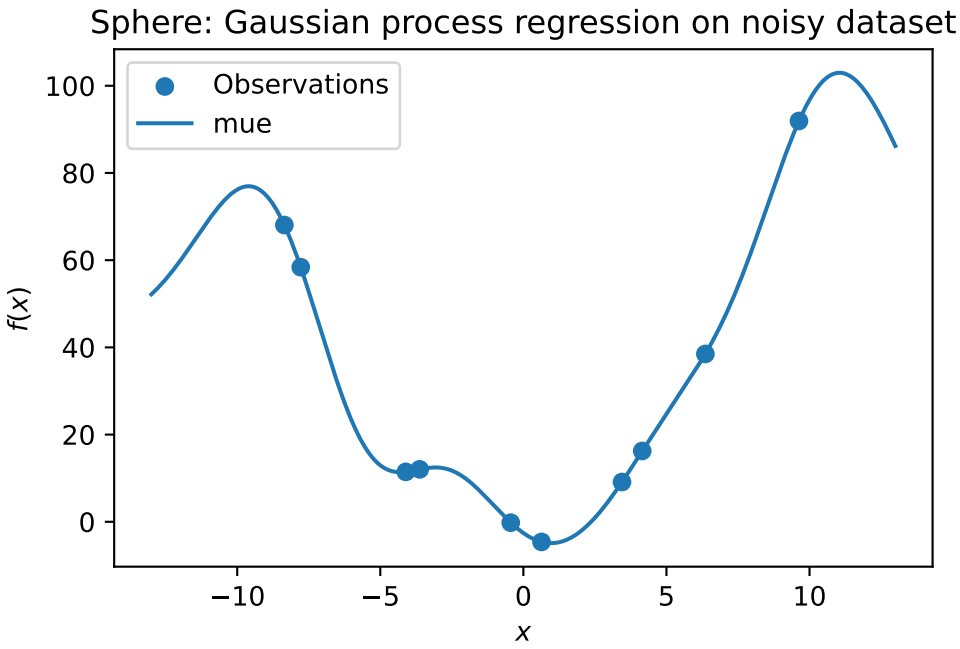
14.3.1.1 The Data

We prepare some data first:

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_sphere
fun_control = {"sigma": 2,
               "seed": 125}
X = gen.scipy_lhd(10, lower=lower, upper = upper)
y = fun(X, fun_control=fun_control)
X_train = X.reshape(-1,1)
y_train = y
```

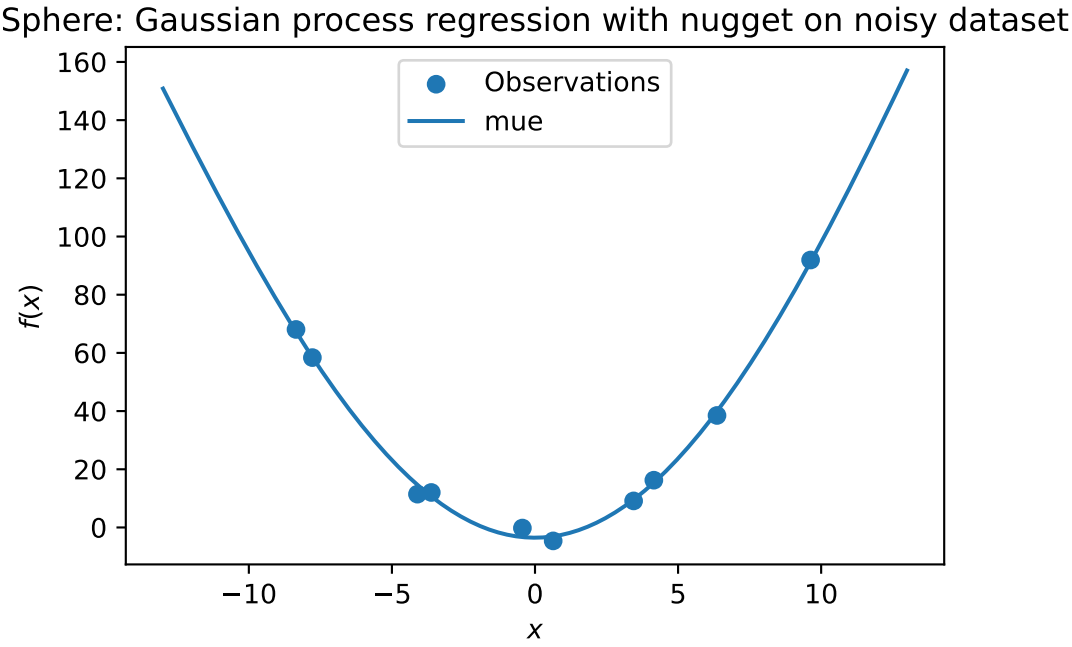
A surrogate without nugget is fitted to these data:

```
S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression on noisy dataset")
```



In comparison to the surrogate without nugget, we fit a surrogate with nugget to the data:

```
S_nug = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=True)
S_nug.fit(X_train, y_train)
X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S_nug.predict(X_axis, return_val="all")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression with nugget on noisy dataset")
```



The value of the nugget term can be extracted from the model as follows:

```
S.Lambda
```

```
S_nug.Lambda
```

```
9.088150066416743e-05
```

We see:

- the first model `S` has no nugget,
- whereas the second model has a nugget value (`Lambda`) larger than zero.

14.4 Exercises

14.4.1 Noisy `fun_cubed`

Analyse the effect of noise on the `fun_cubed` function with the following settings:

```
fun = analytical().fun_cubed
fun_control = {"sigma": 10,
               "seed": 123}
lower = np.array([-10])
upper = np.array([10])
```

14.4.2 `fun_runge`

Analyse the effect of noise on the `fun_runge` function with the following settings:

```
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_runge
fun_control = {"sigma": 0.25,
               "seed": 123}
```

14.4.3 `fun_forrester`

Analyse the effect of noise on the `fun_forrester` function with the following settings:

```
lower = np.array([0])
upper = np.array([1])
fun = analytical().fun_forrester
fun_control = {"sigma": 5,
               "seed": 123}
```

14.4.4 `fun_xsin`

Analyse the effect of noise on the `fun_xsin` function with the following settings:

```
lower = np.array([-1.])
upper = np.array([1.])
fun = analytical().fun_xsin
fun_control = {"sigma": 0.5,
               "seed": 123}
```

```
spot_1_noisy.mean_y.shape[0]
```

```
18
```

15 Hyperparameter Tuning: sklearn

```
MAX_TIME = 1 # Time in minutes. Counter starts, after the initial design is evaluated. So,
INIT_SIZE = 5 # Initial number of designs to evaluate, before the surrogate is build.
```

```
import pickle
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '10-sklearn' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_
experiment_name = experiment_name.replace(':', '-')
experiment_name
```

```
'10-sklearn_bartz09_1min_5init_2023-06-14_23-38-27'
```

This notebook exemplifies hyperparameter tuning with SPOT (spotPython). The hyperparameter software SPOT was developed in R (statistical programming language), see Open Access book "Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide", available here: https://link.springer.com/book/10.1007/978-981-19-5170-1.

```
pip list | grep  "spot[RiverPython]"
```

```
spotPython                0.2.29
spotRiver                 0.0.93

Note: you may need to restart the kernel to use updated packages.
```

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

```
from tabulate import tabulate
import warnings
import numpy as np
from math import inf
import pandas as pd

from scipy.optimize import differential_evolution

import matplotlib.pyplot as plt

from sklearn.preprocessing import OneHotEncoder , MinMaxScaler, StandardScaler
from sklearn.preprocessing import OrdinalEncoder
from sklearn.linear_model import RidgeCV
from sklearn.pipeline import make_pipeline , Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.model_selection import cross_validate
from sklearn.datasets import fetch_openml
from sklearn.metrics import mean_absolute_error, accuracy_score, roc_curve, roc_auc_score,
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import make_regression
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.linear_model import RidgeCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import ElasticNet

warnings.filterwarnings("ignore")

from spotPython.spot import spot
from spotPython.hyperparameters.values import (
    add_core_model_to_fun_control,
    assign_values,
```

```
    convert_keys,
    get_bound_values,
    get_default_hyperparameters_for_core_model,
    get_default_values,
    get_dict_with_levels_and_types,
    get_values_from_dict,
    get_var_name,
    get_var_type,
    iterate_dict_values,
    modify_hyper_parameter_levels,
    modify_hyper_parameter_bounds,
    replace_levels_with_positions,
    return_conf_list_from_var_dict,
    get_one_core_model_from_X,
    transform_hyper_parameter_values,
    get_dict_with_levels_and_types,
    convert_keys,
    iterate_dict_values,
    get_one_sklearn_model_from_X
)

from spotPython.utils.convert import class_for_name
from spotPython.utils.eda import (
    get_stars,
    gen_design_table)
from spotPython.utils.transform import transform_hyper_parameter_values
from spotPython.utils.convert import get_Xy_from_df
from spotPython.plot.validation import plot_cv_predictions, plot_roc, plot_confusion_matri
from spotPython.utils.init import fun_control_init

from spotPython.data.sklearn_hyper_dict import SklearnHyperDict
from spotPython.fun.hypersklearn import HyperSklearn
from spotPython.utils.metrics import mapk, apk
```

15.1 Step 1: Initialization of the Empty `fun_control` Dictionary

```
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/10_spot_hpt_sklearn_classification")
```

15.2 Step 2: Load Data (Classification)

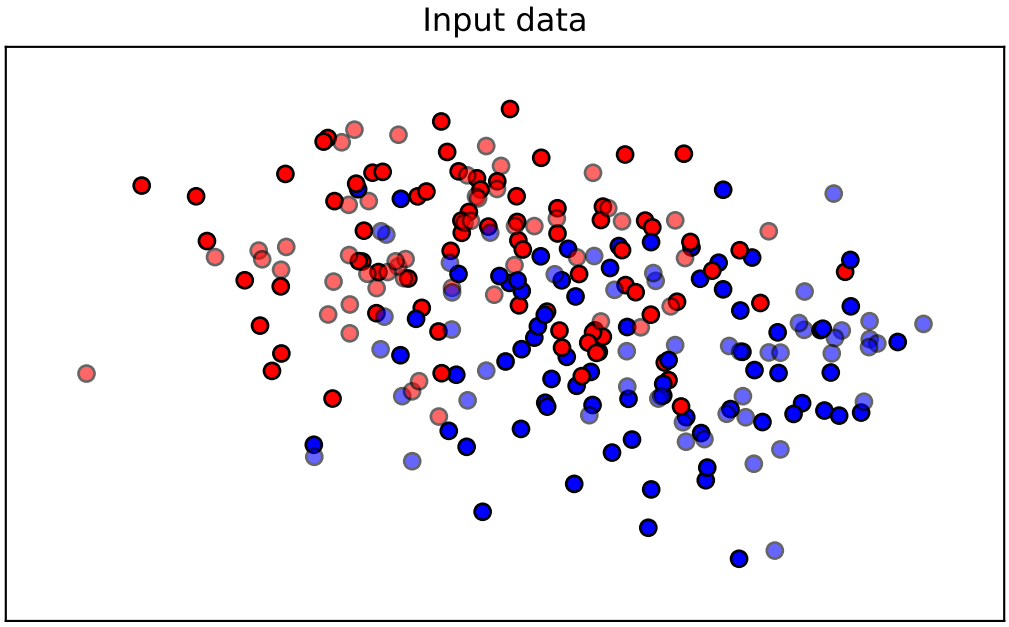
Randomly generate classification data.

```
n_features = 2
n_samples = 250
target_column = "y"
ds =  make_moons(n_samples, noise=0.5, random_state=0)
X, y = ds
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42
)
train = pd.DataFrame(np.hstack((X_train, y_train.reshape(-1, 1))))
test = pd.DataFrame(np.hstack((X_test, y_test.reshape(-1, 1))))
train.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
test.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
train.head()
```

	x1	x2	y
0	1.083978	-1.246111	1.0
1	0.074916	0.868104	0.0
2	-1.668535	0.751752	0.0
3	1.286597	1.454165	0.0
4	1.387021	0.448355	1.0

```
from matplotlib.colors import ListedColormap
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

# just plot the dataset first
cm = plt.cm.RdBu
cm_bright = ListedColormap(["#FF0000", "#0000FF"])
ax = plt.subplot(1, 1, 1)
ax.set_title("Input data")
# Plot the training points
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, edgecolors="k")
# Plot the testing points
ax.scatter(
    X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6, edgecolors="k"
)
```

```
ax.set_xlim(x_min, x_max)
ax.set_ylim(y_min, y_max)
ax.set_xticks(())
ax.set_yticks(())

plt.tight_layout()
plt.show()
```




```
n_samples = len(train)
# add the dataset to the fun_control
fun_control.update({"data": None, # dataset,
                "train": train,
                "test": test,
                "n_samples": n_samples,
                "target_column": target_column})
```

15.3 Step 3: Specification of the Preprocessing Model

Data preprocesssing can be very simple, e.g., you can ignore it. Then you would choose the `prep_model` "None":

```
prep_model = None
fun_control.update({"prep_model": prep_model})
```

A default approach for numerical data is the `StandardScaler` (mean 0, variance 1). This can be selected as follows:

```
prep_model = StandardScaler()
fun_control.update({"prep_model": prep_model})
```

Even more complicated pre-processing steps are possible, e.g., the follwing pipeline:

```
# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
#         transformers=[
#             ("categorical", one_hot_encoder, categorical_columns),
#         ],
#         remainder=StandardScaler(),
#     )
```

15.4 Step 4: Select `algorithm` and `core_model_hyper_dict`

The selection of the algorithm (ML model) that should be tuned is done by specifying the its name from the `sklearn` implementation. For example, the `SVC` support vector machine classifier is selected as follows:

```
# core_model  = RidgeCV
# core_model = GradientBoostingRegressor
# core_model = ElasticNet
# core_model = RandomForestClassifier
core_model = SVC
# core_model = LogisticRegression
# core_model = KNeighborsClassifier
# core_model = GradientBoostingClassifier
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                fun_control=fun_control,
                                hyper_dict=SklearnHyperDict,
                                filename=None)
```

Now `fun_control` has the information from the JSON file:

```
"SVC":
{
    "C": {
        "type": "float",
        "default": 1.0,
        "transform": "None",
        "lower": 0.1,
        "upper": 10.0},
    "kernel": {
        "levels": ["linear", "poly", "rbf", "sigmoid"],
        "type": "factor",
        "default": "rbf",
        "transform": "None",
        "core_model_parameter_type": "str",
        "lower": 0,
        "upper": 3},
    "degree": {
        "type": "int",
        "default": 3,
        "transform": "None",
        "lower": 3,
        "upper": 3},
    "gamma": {
        "levels": ["scale", "auto"],
        "type": "factor",
        "default": "scale",
        "transform": "None",
        "core_model_parameter_type": "str",
        "lower": 0,
        "upper": 1},
    "coef0": {
        "type": "float",
        "default": 0.0,
        "transform": "None",
        "lower": 0.0,
        "upper": 0.0},
    "shrinking": {
        "levels": [0, 1],
        "type": "factor",
        "default": 0,
        "transform": "None",
```

```
            "core_model_parameter_type": "bool",
            "lower": 0,
            "upper": 1},
        "probability": {
            "levels": [0, 1],
            "type": "factor",
            "default": 0,
            "transform": "None",
            "core_model_parameter_type": "bool",
            "lower": 0,
            "upper": 1},
        "tol": {
            "type": "float",
            "default": 1e-3,
            "transform": "None",
            "lower": 1e-4,
            "upper": 1e-2},
        "cache_size": {
            "type": "float",
            "default": 200,
            "transform": "None",
            "lower": 100,
            "upper": 400},
        "break_ties": {
            "levels": [0, 1],
            "type": "factor",
            "default": 0,
            "transform": "None",
            "core_model_parameter_type": "bool",
            "lower": 0,
            "upper": 1}
    }
```

15.5 Step 5: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

15.5.1 Modify hyperparameter of type factor

Factors can be modified with the `modify_hyper_parameter_levels` function. For example, to exclude the `sigmoid` kernel from the tuning, the `kernel` hyperparameter of the `SVC` model

can be modified as follows:

```
fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["linear", "poly", "rbf
fun_control["core_model_hyper_dict"]["kernel"]
```

```
{'levels': ['linear', 'poly', 'rbf'],
 'type': 'factor',
 'default': 'rbf',
 'transform': 'None',
 'core_model_parameter_type': 'str',
 'lower': 0,
 'upper': 2}
```

15.5.2 Modify hyperparameter of type numeric and integer (boolean)

Numeric and boolean values can be modified using the `modify_hyper_parameter_bounds` method. For example, to change the `tol` hyperparameter of the `SVC` model to the interval [1e-3, 1e-2], the following code can be used:

```
fun_control = modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
# fun_control = modify_hyper_parameter_bounds(fun_control, "min_samples_split", bounds=[3,
#fun_control = modify_hyper_parameter_bounds(fun_control, "merit_preprune", bounds=[0, 0])
fun_control["core_model_hyper_dict"]["tol"]
```

```
{'type': 'float',
 'default': 0.001,
 'transform': 'None',
 'lower': 0.001,
 'upper': 0.01}
```

15.6 Step 6: Selection of the Objective (Loss) Function

There are two metrics:

```
1. `metric_river` is used for the river based evaluation via `eval_oml_iter_progressive`.
2. `metric_sklearn` is used for the sklearn based evaluation.
```

```
fun = HyperSklearn(seed=123, log_level=50).fun_sklearn
# metric_sklearn = roc_auc_score
# weights = -1.0
metric_sklearn = log_loss
weights = 1.0
# k = None
# custom_metric = mapk

fun_control.update({
                "horizon": None,
                "oml_grace_period": None,
                "weights": weights,
                "step": None,
                "log_level": 50,
                "weight_coeff": None,
                "metric_river": None,
                "metric_sklearn": metric_sklearn,
                # "metric_params": {"k": k},
                })
```

15.6.1 Predict Classes or Class Probabilities

If the key `"predict_proba"` is set to `True`, the class probabilities are predicted. `False` is the default, i.e., the classes are predicted.

```
fun_control.update({
                "predict_proba": False,
                })
```

15.7 Step 7: Calling the SPOT Function

15.7.1 Prepare the SPOT Parameters

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

```
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})
```

```
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

```
print(gen_design_table(fun_control))
```

```
| name        | type   | default   |   lower |   upper | transform   |
|-------------|--------|-----------|---------|---------|-------------|
| C           | float  | 1.0       |   0.1   |   10    | None        |
| kernel      | factor | rbf       |   0     |    2    | None        |
| degree      | int    | 3         |   3     |    3    | None        |
| gamma       | factor | scale     |   0     |    1    | None        |
| coef0       | float  | 0.0       |   0     |    0    | None        |
| shrinking   | factor | 0         |   0     |    1    | None        |
| probability | factor | 0         |   0     |    1    | None        |
| tol         | float  | 0.001     |   0.001 |    0.01 | None        |
| cache_size  | float  | 200.0     | 100     |  400    | None        |
| break_ties  | factor | 0         |   0     |    1    | None        |
```

15.7.2 Run the `Spot` Optimizer

- Run SPOT for approx. x mins (`max_time`).
- Note: the run takes longer, because the evaluation time of initial design (here: `initi_size`, 20 points) is not considered.

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=SklearnHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
X_start
```

```
array([[1.e+00, 2.e+00, 3.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 1.e-03,
        2.e+02, 0.e+00]])
```

```
spot_tuner = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = inf,
                    fun_repeats = 1,
                    max_time = MAX_TIME,
```

```
                    noise = False,
                    tolerance_x = np.sqrt(np.spacing(1)),
                    var_type = var_type,
                    var_name = var_name,
                    infill_criterion = "y",
                    n_points = 1,
                    seed=123,
                    log_level = 50,
                    show_models= False,
                    show_progress= True,
                    fun_control = fun_control,
                    design_control={"init_size": INIT_SIZE,
                                    "repeats": 1},
                    surrogate_control={"noise": True,
                                       "cod_type": "norm",
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": len(var_name),
                                       "model_optimizer": differential_evolution,
                                       "model_fun_evals": 10_000,
                                       "log_level": 50
                                       })
spot_tuner.run(X_start=X_start)
```

```
spotPython tuning: 5.691103166702708 [----------] 2.79%

spotPython tuning: 5.691103166702708 [----------] 4.63%

spotPython tuning: 5.691103166702708 [#---------] 6.16%

spotPython tuning: 5.691103166702708 [#---------] 7.65%

spotPython tuning: 5.691103166702708 [#---------] 9.06%

spotPython tuning: 5.691103166702708 [#---------] 11.44%

spotPython tuning: 5.691103166702708 [#---------] 13.78%

spotPython tuning: 5.691103166702708 [##--------] 16.01%
```

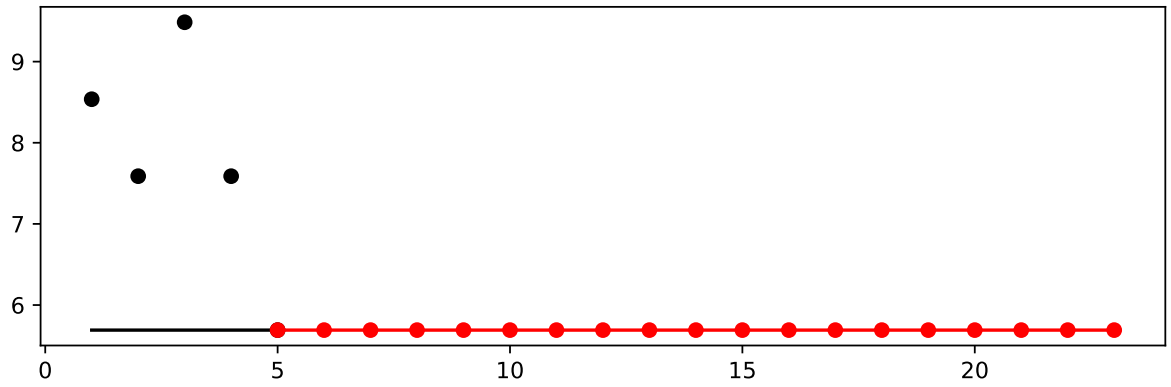
```
spotPython tuning: 5.691103166702708 [##--------] 18.31%

spotPython tuning: 5.691103166702708 [##--------] 20.56%

spotPython tuning: 5.691103166702708 [##--------] 23.02%

spotPython tuning: 5.691103166702708 [###-------] 25.66%

spotPython tuning: 5.691103166702708 [###-------] 34.08%

spotPython tuning: 5.691103166702708 [#####-----] 45.98%

spotPython tuning: 5.691103166702708 [######----] 58.64%

spotPython tuning: 5.691103166702708 [#######---] 71.85%

spotPython tuning: 5.691103166702708 [########--] 84.92%

spotPython tuning: 5.691103166702708 [##########] 97.68%

spotPython tuning: 5.691103166702708 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2c5832a40>
```

15.7.3 Results

```
SAVE = False
LOAD = False

if SAVE:
    result_file_name = "res_" + experiment_name + ".pkl"
    with open(result_file_name, 'wb') as f:
        pickle.dump(spot_tuner, f)

if LOAD:
    result_file_name = "res_ch10-friedman-hpt-0_maans03_60min_20init_1K_2023-04-14_10-11-1
    with open(result_file_name, 'rb') as f:
        spot_tuner =  pickle.load(f)
```

- Show the Progress of the hyperparameter tuning:

```
spot_tuner.plot_progress(log_y=False, filename="./figures/" + experiment_name+"_progress.p
```



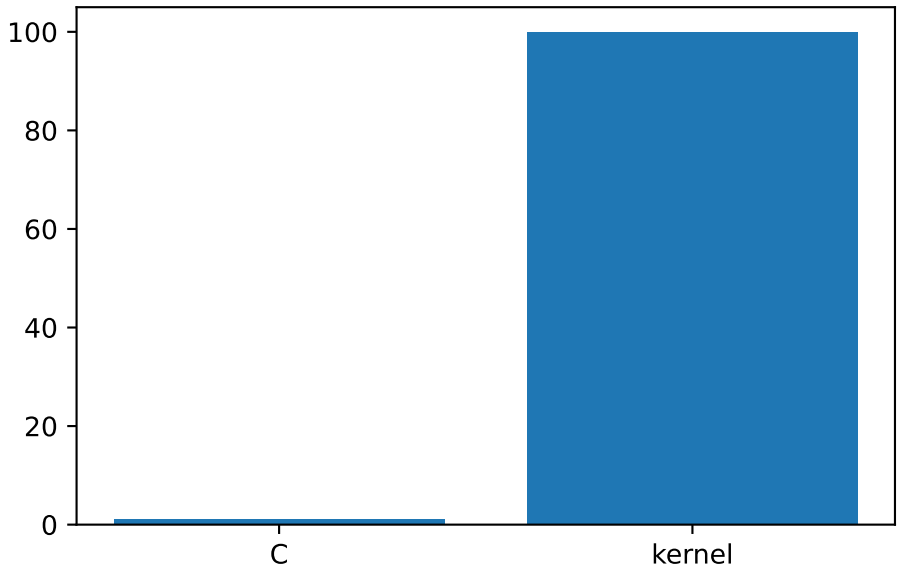
Print the results.

```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
C	float	1.0	0.1	10.0	3.6280771109650245	None
kernel	factor	rbf	0.0	2.0	1.0	None
degree	int	3	3.0	3.0	3.0	None
gamma	factor	scale	0.0	1.0	0.0	None
coef0	float	0.0	0.0	0.0	0.0	None
shrinking	factor	0	0.0	1.0	1.0	None
probability	factor	0	0.0	1.0	0.0	None
tol	float	0.001	0.001	0.01	0.006642600916881275	None
cache_size	float	200.0	100.0	400.0	202.03372626175258	None
break_ties	factor	0	0.0	1.0	1.0	None

15.8 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```



15.9 Get Default Hyperparameters

```
values_default = get_default_values(fun_control)
values_default = transform_hyper_parameter_values(fun_control=fun_control, hyper_parameter
values_default
```

```
{'C': 1.0,
 'kernel': 'rbf',
 'degree': 3,
 'gamma': 'scale',
 'coef0': 0.0,
 'shrinking': 0,
 'probability': 0,
 'tol': 0.001,
 'cache_size': 200.0,
 'break_ties': 0}
```

```
model_default = make_pipeline(fun_control["prep_model"], fun_control["core_model"](**value
model_default
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
```

```
                ('svc',
                 SVC(break_ties=0, cache_size=200.0, probability=0,
                     shrinking=0))])
```

15.10 Get SPOT Results

```
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
print(X)
```

```
[[3.62807711e+00 1.00000000e+00 3.00000000e+00 0.00000000e+00
  0.00000000e+00 1.00000000e+00 0.00000000e+00 6.64260092e-03
  2.02033726e+02 1.00000000e+00]]
```

```
v_dict = assign_values(X, fun_control["var_name"])
return_conf_list_from_var_dict(var_dict=v_dict, fun_control=fun_control)
```

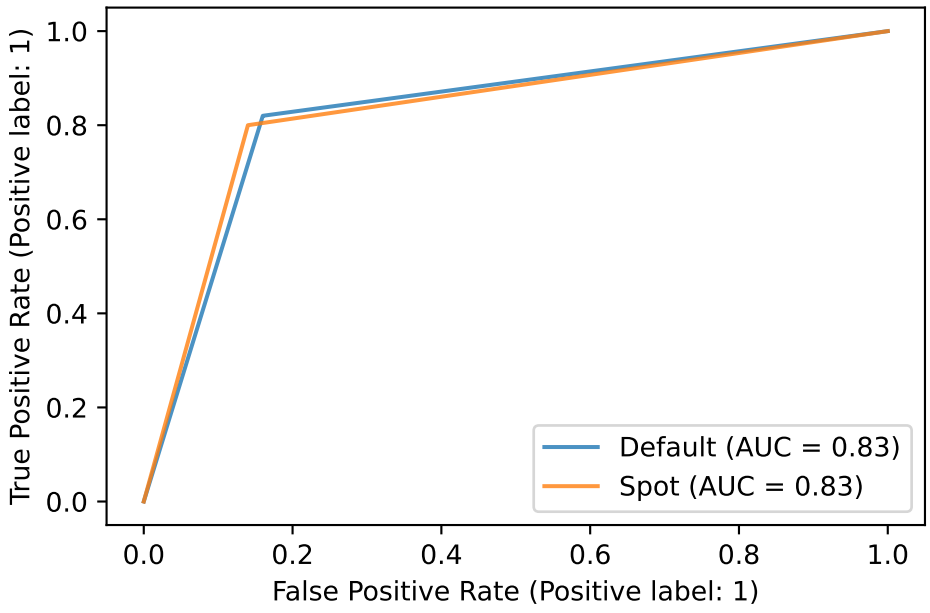
```
[{'C': 3.6280771109650245,
  'kernel': 'poly',
  'degree': 3,
  'gamma': 'scale',
  'coef0': 0.0,
  'shrinking': 1,
  'probability': 0,
  'tol': 0.006642600916881275,
  'cache_size': 202.03372626175258,
  'break_ties': 1}]
```

```
model_spot = get_one_sklearn_model_from_X(X, fun_control)
model_spot
```

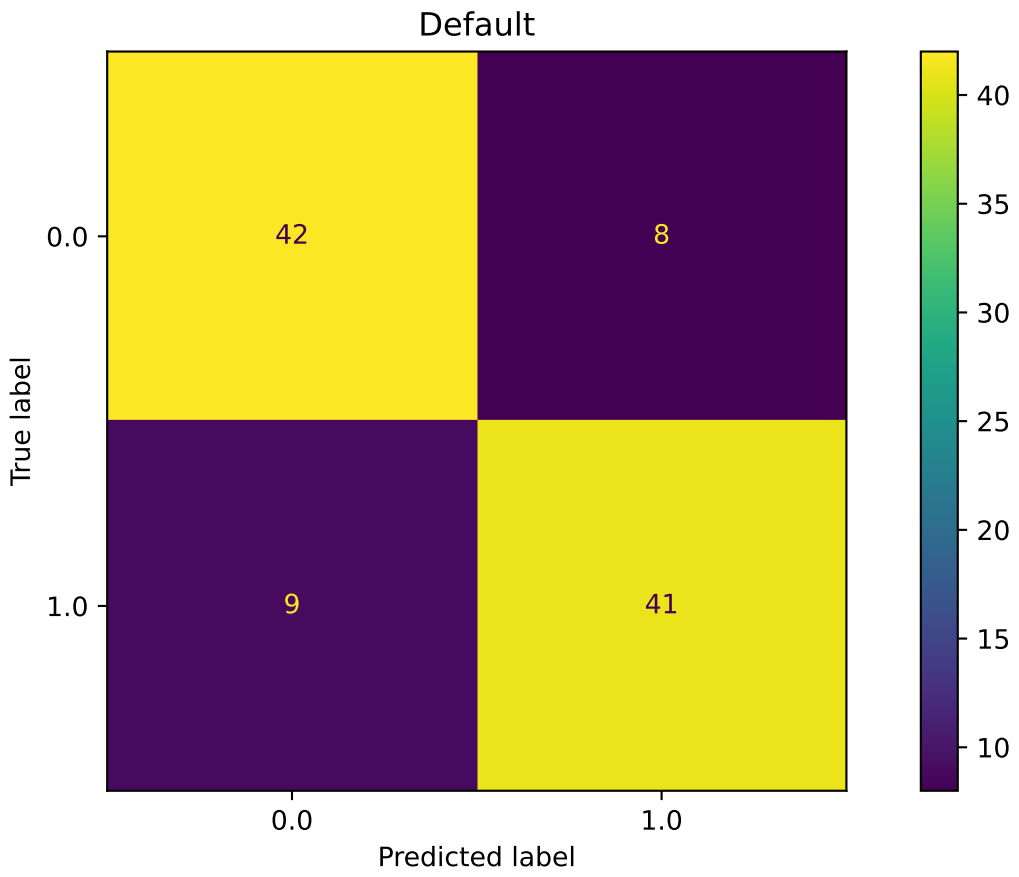
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc',
                 SVC(C=3.6280771109650245, break_ties=1,
                     cache_size=202.03372626175258, kernel='poly',
                     probability=0, shrinking=1, tol=0.006642600916881275))])
```

15.11 Plot: Compare Predictions

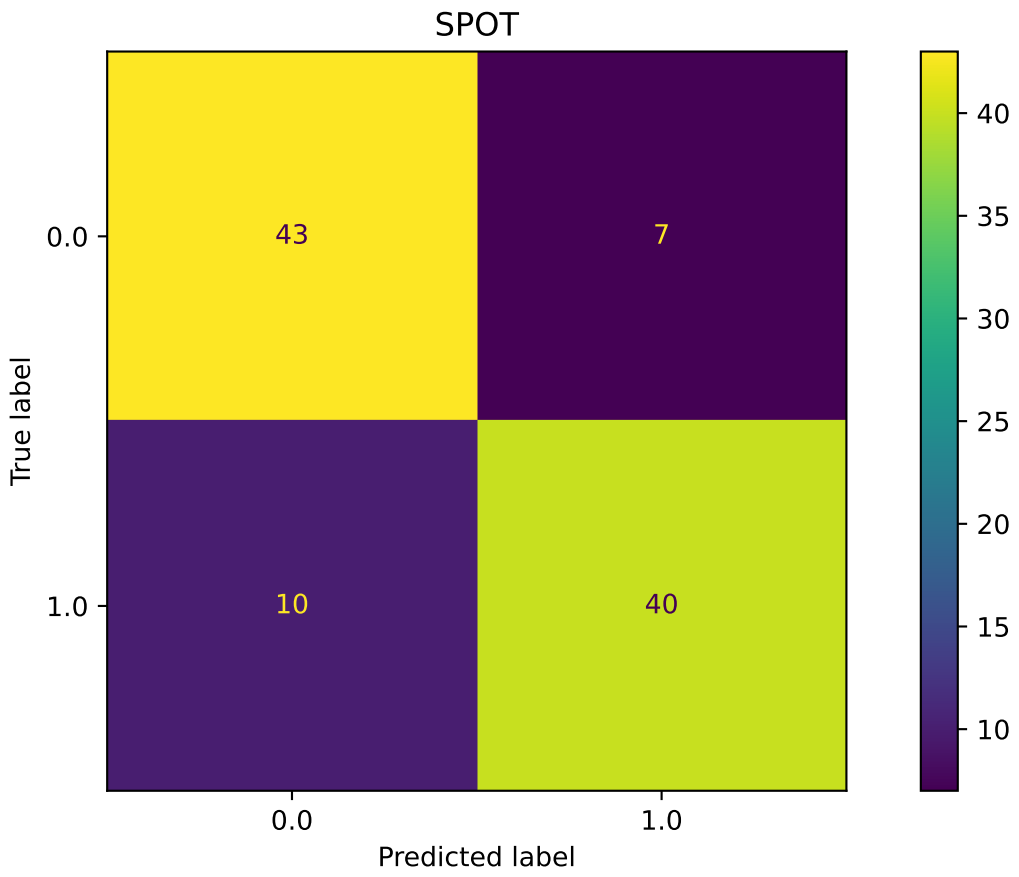
```
plot_roc([model_default, model_spot], fun_control, model_names=["Default", "Spot"])
```




```
plot_confusion_matrix(model_default, fun_control, title = "Default")
```




```
plot_confusion_matrix(model_spot, fun_control, title="SPOT")
```



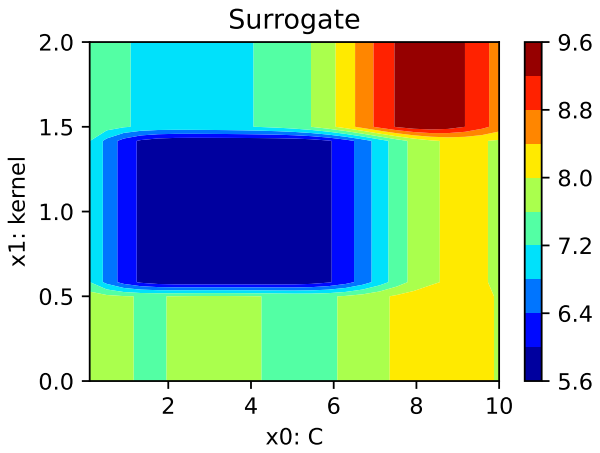

```
min(spot_tuner.y), max(spot_tuner.y)
```

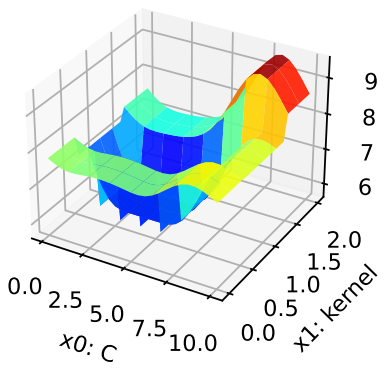
```
(5.691103166702708, 9.485171944504513)
```

15.12 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
C:  1.1399176173997725
kernel:  100.0
```





15.13 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

15.14 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

16 Hyperparameter Tuning: PyTorch With fashionMNIST Data Using Hold-out Data Sets

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = None # "cpu" # "cuda:0"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

```
mps
```

```
import pickle
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '11-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SI
experiment_name = experiment_name.replace(':', '-')
experiment_name
```

```
'11-torch_bartz09_1min_5init_2023-06-15_00-01-08'
```

This notebook exemplifies hyperparameter tuning with SPOT (spotPython). The hyperparameter software SPOT was developed in R (statistical programming language), see Open Access book "Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide", available here: https://link.springer.com/book/10.1007/978-981-19-5170-1.

```
pip list | grep  "spot[RiverPython]"
```

```
spotPython                0.2.29
spotRiver                 0.0.93

Note: you may need to restart the kernel to use updated packages.
```

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

```
from tabulate import tabulate
import copy
import warnings
import numbers
import json
import calendar
import math
import datetime as dt
import numpy as np
from math import inf
import pandas as pd

from scipy.optimize import differential_evolution

import matplotlib.pyplot as plt

import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
from functools import partial
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import random_split
import torchvision
import torchvision.transforms as transforms

from spotPython.spot import spot
from spotPython.hyperparameters.values import (
    add_core_model_to_fun_control,
    assign_values,
```

```
    convert_keys,
    get_bound_values,
    get_default_hyperparameters_for_core_model,
    get_default_values,
    get_dict_with_levels_and_types,
    get_values_from_dict,
    get_var_name,
    get_var_type,
    iterate_dict_values,
    modify_hyper_parameter_levels,
    modify_hyper_parameter_bounds,
    replace_levels_with_positions,
    return_conf_list_from_var_dict,
    get_one_core_model_from_X,
    transform_hyper_parameter_values,
    get_dict_with_levels_and_types,
    convert_keys,
    iterate_dict_values,
)

from spotPython.torch.traintest import evaluate_cv, evaluate_hold_out
from spotPython.utils.convert import class_for_name
from spotPython.utils.eda import (
    get_stars,
    gen_design_table)
from spotPython.utils.transform import transform_hyper_parameter_values

from spotPython.utils.convert import get_Xy_from_df
from spotPython.utils.init import fun_control_init
from spotPython.plot.validation import plot_cv_predictions, plot_roc, plot_confusion_matri

from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.fun.hypertorch import HyperTorch

warnings.filterwarnings("ignore")

# Neural Net specific imports:
from spotPython.torch.netfashionMNIST import Net_fashionMNIST
```

```
print(torch.__version__)
# Check that MPS is available
```

```
if not torch.backends.mps.is_available():
    if not torch.backends.mps.is_built():
        print("MPS not available because the current PyTorch install was not "
              "built with MPS enabled.")
    else:
        print("MPS not available because the current MacOS version is not 12.3+ "
              "and/or you do not have an MPS-enabled device on this machine.")

else:
    mps_device = torch.device("mps")
    print("MPS device: ", mps_device)
```

```
2.0.1
MPS device:  mps
```

16.1 Step 1: Initialization of the Empty `fun_control` Dictionary

```
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/11_spot_hpt_torch_fashion_mnist",
    device=DEVICE)
```

16.2 Step 2: Load fashionMNIST Data

```
def load_data(data_dir="./data"):
    # Download training data from open datasets.
    training_data = datasets.FashionMNIST(
        root=data_dir,
        train=True,
        download=True,
        transform=ToTensor(),
    )
    # Download test data from open datasets.
    test_data = datasets.FashionMNIST(
        root=data_dir,
        train=False,
        download=True,
        transform=ToTensor(),
```

```
    )
    return training_data, test_data
```

```
train, test = load_data()
train.data.shape, test.data.shape
```

```
(torch.Size([60000, 28, 28]), torch.Size([10000, 28, 28]))
```

```
n_samples = len(train)
# add the dataset to the fun_control
fun_control.update({"data": None,
                "train": train,
                "test": test,
                "n_samples": n_samples,
                "target_column": None})
```

16.3 Step 3: Specification of the Preprocessing Model

```
# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
#         transformers=[
#             ("categorical", one_hot_encoder, categorical_columns),
#         ],
#         remainder=StandardScaler(),
#     )
prep_model = None
fun_control.update({"prep_model": prep_model})
```

16.4 Step 4: Select `algorithm` and `core_model_hyper_dict`

`spotPython` implements a class which is similar to the class described in the PyTorch tutorial. The class is called `Net_fashionMNIST` and is implemented in the file `netcifar10.py`. The class is imported here.

Note: In addition to the class Net from the PyTorch tutorial, the class Net_CIFAR10 has additional attributes, namely:

- learning rate (`lr`),
- batchsize (`batch_size`),
- epochs (`epochs`), and
- k_folds (`k_folds`).

Further attributes can be easily added to the class, e.g., `optimizer` or `loss_function`.

```
core_model = Net_fashionMNIST
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                fun_control=fun_control,
                                hyper_dict=TorchHyperDict,
                                filename=None)
```

16.5 Step 5: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

16.5.1 Modify hyperparameter of type factor

```
# fun_control = modify_hyper_parameter_levels(fun_control, "leaf_model", ["LinearRegressio
# fun_control["core_model_hyper_dict"]
```

16.5.2 Modify hyperparameter of type numeric and integer (boolean)

```
# fun_control = modify_hyper_parameter_bounds(fun_control, "delta", bounds=[1e-10, 1e-6])
# fun_control = modify_hyper_parameter_bounds(fun_control, "min_samples_split", bounds=[3,
#fun_control = modify_hyper_parameter_bounds(fun_control, "merit_preprune", bounds=[0, 0])
# fun_control["core_model_hyper_dict"]
fun_control = modify_hyper_parameter_bounds(fun_control, "k_folds", bounds=[0, 0])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 2])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 3])
```

16.6 Step 6: Selection of the Objective (Loss) Function

```
from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({"loss_function": loss_function})
```

In addition to the loss functions, `spotPython` provides access to a large number of metrics.

- The key `"metric_sklearn"` is used for metrics that follow the `scikit-learn` conventions.
- The key `"river_metric"` is used for the river based evaluation (Montiel et al. 2021) via `eval_oml_iter_progressive`, and
- the key `"metric_torch"` is used for the metrics from `TorchMetrics`.

`TorchMetrics` is a collection of more than 90 PyTorch metrics[1].

Because the `PyTorch` tutorial uses the accuracy as metric, we use the same metric here. Currently, accuracy is computed in the tutorial's example code. We will use `TorchMetrics` instead, because it offers more flexibilty, e.g., it can be used for regression and classification. Furthermore, `TorchMetrics` offers the following advantages:

- A standardized interface to increase reproducibility
- Reduces Boilerplate
- Distributed-training compatible
- Rigorously tested
- Automatic accumulation over batches
- Automatic synchronization between multiple devices

Therefore, we set

```
import torchmetrics
metric_torch = torchmetrics.Accuracy(task="multiclass", num_classes=10).to(fun_control["de
fun_control.update({"metric_torch": metric_torch})
```

> **Minimization and maximization:**
>
> `spotPython` performs minimization by default. If accuracy should be maximized, then the objective function has to be multiplied by -1. Therefore, `weights` is set to -1 in this case.

[1] https://torchmetrics.readthedocs.io/en/latest/.

```
fun = HyperTorch(seed=123, log_level=50).fun_torch
loss_function = CrossEntropyLoss()
weights = 1.0
shuffle = True
eval = "train_hold_out"
show_batch_interval = 100_000
path="torch_model.pt"

fun_control.update({
                "data_dir": None,
                "checkpoint_dir": None,
                "horizon": None,
                "oml_grace_period": None,
                "weights": weights,
                "step": None,
                "log_level": 50,
                "weight_coeff": None,
                "metric_river": None,
                "metric_sklearn": None,
                "loss_function": loss_function,
                "shuffle": shuffle,
                "eval": eval,
                "show_batch_interval": show_batch_interval,
                "path": path,
                })
```

```
fun_control
```

```
{'data': None,
 'train': Dataset FashionMNIST
     Number of datapoints: 60000
     Root location: ./data
     Split: Train
     StandardTransform
 Transform: ToTensor(),
 'test': Dataset FashionMNIST
     Number of datapoints: 10000
     Root location: ./data
     Split: Test
     StandardTransform
 Transform: ToTensor(),
```

```
'loss_function': CrossEntropyLoss(),
'metric_sklearn': None,
'metric_river': None,
'metric_torch': MulticlassAccuracy(),
'metric_params': {},
'prep_model': None,
'n_samples': 60000,
'target_column': None,
'shuffle': True,
'eval': 'train_hold_out',
'k_folds': None,
'optimizer': None,
'device': 'mps',
'show_batch_interval': 100000,
'path': 'torch_model.pt',
'task': 'classification',
'save_model': False,
'weights': 1.0,
'writer': <torch.utils.tensorboard.writer.SummaryWriter at 0x162e67190>,
'core_model': spotPython.torch.netfashionMNIST.Net_fashionMNIST,
'core_model_hyper_dict': {'l1': {'type': 'int',
  'default': 5,
  'transform': 'transform_power_2_int',
  'lower': 2,
  'upper': 9},
 'l2': {'type': 'int',
  'default': 5,
  'transform': 'transform_power_2_int',
  'lower': 2,
  'upper': 9},
 'lr_mult': {'type': 'float',
  'default': 1.0,
  'transform': 'None',
  'lower': 0.1,
  'upper': 10.0},
 'batch_size': {'type': 'int',
  'default': 4,
  'transform': 'transform_power_2_int',
  'lower': 1,
  'upper': 4},
 'epochs': {'type': 'int',
  'default': 3,
  'transform': 'transform_power_2_int',
```

```
  'lower': 2,
  'upper': 3},
 'k_folds': {'type': 'int',
  'default': 1,
  'transform': 'None',
  'lower': 0,
  'upper': 0},
 'patience': {'type': 'int',
  'default': 5,
  'transform': 'None',
  'lower': 2,
  'upper': 2},
 'optimizer': {'levels': ['Adadelta',
    'Adagrad',
    'Adam',
    'AdamW',
    'SparseAdam',
    'Adamax',
    'ASGD',
    'NAdam',
    'RAdam',
    'RMSprop',
    'Rprop',
    'SGD'],
  'type': 'factor',
  'default': 'SGD',
  'transform': 'None',
  'core_model_parameter_type': 'str',
  'lower': 0,
  'upper': 12},
 'sgd_momentum': {'type': 'float',
  'default': 0.0,
  'transform': 'None',
  'lower': 0.0,
  'upper': 1.0}},
'data_dir': None,
'checkpoint_dir': None,
'horizon': None,
'oml_grace_period': None,
'step': None,
'log_level': 50,
'weight_coeff': None}
```

16.7 Step 7: Calling the SPOT Function

16.7.1 Prepare the SPOT Parameters

Get types and variable names as well as lower and upper bounds for the hyperparameters.

```
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})

lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

```
print(gen_design_table(fun_control))
```

```
| name         | type   | default   |   lower |   upper | transform             |
|--------------|--------|-----------|---------|---------|-----------------------|
| l1           | int    | 5         |     2   |       9 | transform_power_2_int |
| l2           | int    | 5         |     2   |       9 | transform_power_2_int |
| lr_mult      | float  | 1.0       |     0.1 |      10 | None                  |
| batch_size   | int    | 4         |     1   |       4 | transform_power_2_int |
| epochs       | int    | 3         |     2   |       3 | transform_power_2_int |
| k_folds      | int    | 1         |     0   |       0 | None                  |
| patience     | int    | 5         |     2   |       2 | None                  |
| optimizer    | factor | SGD       |     0   |      12 | None                  |
| sgd_momentum | float  | 0.0       |     0   |       1 | None                  |
```

16.7.2 Run the `Spot` Optimizer

- Run SPOT for approx. x mins (`max_time`).
- Note: the run takes longer, because the evaluation time of initial design (here: `initi_size`, 20 points) is not considered.

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=TorchHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
X_start
```

```
array([[ 5.,  5.,  1.,  4.,  3.,  1.,  5., 11.,  0.]])
```

```
spot_tuner = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = inf,
                    fun_repeats = 1,
                    max_time = MAX_TIME,
                    noise = False,
                    tolerance_x = np.sqrt(np.spacing(1)),
                    var_type = var_type,
                    var_name = var_name,
                    infill_criterion = "y",
                    n_points = 1,
                    seed=123,
                    log_level = 50,
                    show_models= False,
                    show_progress= True,
                    fun_control = fun_control,
                    design_control={"init_size": INIT_SIZE,
                                    "repeats": 1},
                    surrogate_control={"noise": True,
                                       "cod_type": "norm",
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": len(var_name),
                                       "model_optimizer": differential_evolution,
                                       "model_fun_evals": 10_000,
                                       "log_level": 50
                                       })
spot_tuner.run(X_start=X_start)
```

```
config: {'l1': 16, 'l2': 32, 'lr_mult': 9.563687451910228, 'batch_size': 8, 'epochs': 8, 'k_
Epoch: 1

Loss on hold-out set: 0.5639299386562779
Accuracy on hold-out set: 0.8086666666666666
MulticlassAccuracy value on hold-out data: 0.8086666464805603
Epoch: 2

Loss on hold-out set: 0.5653582727587005
Accuracy on hold-out set: 0.8198333333333333
```

```
MulticlassAccuracy value on hold-out data: 0.8198333382606506
Epoch: 3

Loss on hold-out set: 0.5918368185905662
Accuracy on hold-out set: 0.82375
MulticlassAccuracy value on hold-out data: 0.8237500190734863
Early stopping at epoch 2
Returned to Spot: Validation loss: 0.5918368185905662
--------------------------------------------

config: {'l1': 128, 'l2': 32, 'lr_mult': 6.258012467639852, 'batch_size': 2, 'epochs': 4, 'k
Epoch: 1

Loss on hold-out set: 0.5757737813411148
Accuracy on hold-out set: 0.83775
MulticlassAccuracy value on hold-out data: 0.8377500176429749
Epoch: 2

Loss on hold-out set: 0.6058747537216007
Accuracy on hold-out set: 0.85975
MulticlassAccuracy value on hold-out data: 0.859749972820282
Epoch: 3

Loss on hold-out set: 0.5740763965449877
Accuracy on hold-out set: 0.856875
MulticlassAccuracy value on hold-out data: 0.8568750023841858
Epoch: 4

Loss on hold-out set: 0.6059362746353876
Accuracy on hold-out set: 0.87025
MulticlassAccuracy value on hold-out data: 0.8702499866485596
Returned to Spot: Validation loss: 0.6059362746353876
--------------------------------------------

config: {'l1': 256, 'l2': 256, 'lr_mult': 0.2437336281201693, 'batch_size': 16, 'epochs': 8,
Epoch: 1

Loss on hold-out set: 0.5067713373502095
Accuracy on hold-out set: 0.8195833333333333
MulticlassAccuracy value on hold-out data: 0.8195833563804626
Epoch: 2
```

```
Loss on hold-out set: 0.46309622772037984
Accuracy on hold-out set: 0.8365
MulticlassAccuracy value on hold-out data: 0.8364999890327454
Epoch: 3

Loss on hold-out set: 0.44287427786861855
Accuracy on hold-out set: 0.8460833333333333
MulticlassAccuracy value on hold-out data: 0.8460833430290222
Epoch: 4

Loss on hold-out set: 0.4336258042678237
Accuracy on hold-out set: 0.8489166666666667
MulticlassAccuracy value on hold-out data: 0.8489166498184204
Epoch: 5

Loss on hold-out set: 0.41878673816348116
Accuracy on hold-out set: 0.8535833333333334
MulticlassAccuracy value on hold-out data: 0.8535833358764648
Epoch: 6

Loss on hold-out set: 0.4121393164371451
Accuracy on hold-out set: 0.8559166666666667
MulticlassAccuracy value on hold-out data: 0.8559166789054871
Epoch: 7

Loss on hold-out set: 0.40916690373172365
Accuracy on hold-out set: 0.8587916666666666
MulticlassAccuracy value on hold-out data: 0.8587916493415833
Epoch: 8

Loss on hold-out set: 0.401622335669895
Accuracy on hold-out set: 0.8594583333333333
MulticlassAccuracy value on hold-out data: 0.859458327293396
Returned to Spot: Validation loss: 0.401622335669895
--------------------------------------------

config: {'l1': 64, 'l2': 8, 'lr_mult': 2.906205211581667, 'batch_size': 8, 'epochs': 4, 'k_f
Epoch: 1
```

```
Loss on hold-out set: 0.9739701749483745
Accuracy on hold-out set: 0.6530416666666666
MulticlassAccuracy value on hold-out data: 0.653041660785675
Epoch: 2

Loss on hold-out set: 0.793643853550156
Accuracy on hold-out set: 0.7045416666666666
MulticlassAccuracy value on hold-out data: 0.7045416831970215
Epoch: 3

Loss on hold-out set: 0.7214714847852787
Accuracy on hold-out set: 0.7467916666666666
MulticlassAccuracy value on hold-out data: 0.746791660785675
Epoch: 4

Loss on hold-out set: 0.6667192413906257
Accuracy on hold-out set: 0.7648333333333334
MulticlassAccuracy value on hold-out data: 0.7648333311080933
Returned to Spot: Validation loss: 0.6667192413906257
--------------------------------------------

config: {'l1': 4, 'l2': 128, 'lr_mult': 4.224097306355747, 'batch_size': 4, 'epochs': 8, 'k_
Epoch: 1

Loss on hold-out set: 1.132752470350538
Accuracy on hold-out set: 0.56825
MulticlassAccuracy value on hold-out data: 0.5682500004768372
Epoch: 2

Loss on hold-out set: 1.1316382856442748
Accuracy on hold-out set: 0.5524583333333334
MulticlassAccuracy value on hold-out data: 0.5524583458900452
Epoch: 3

Loss on hold-out set: 1.1668552164934227
Accuracy on hold-out set: 0.5403333333333333
MulticlassAccuracy value on hold-out data: 0.5403333306312561
Epoch: 4
```

```
Loss on hold-out set: 1.165837463103846
Accuracy on hold-out set: 0.564875
MulticlassAccuracy value on hold-out data: 0.5648750066757202
Early stopping at epoch 3
Returned to Spot: Validation loss: 1.165837463103846
--------------------------------------------


config: {'l1': 256, 'l2': 256, 'lr_mult': 0.39410343836663486, 'batch_size': 16, 'epochs': 8
Epoch: 1

Loss on hold-out set: 0.44941737756629785
Accuracy on hold-out set: 0.8405
MulticlassAccuracy value on hold-out data: 0.840499997138977
Epoch: 2

Loss on hold-out set: 0.4162547228510181
Accuracy on hold-out set: 0.8530416666666667
MulticlassAccuracy value on hold-out data: 0.8530416488647461
Epoch: 3

Loss on hold-out set: 0.39927290502749385
Accuracy on hold-out set: 0.859625
MulticlassAccuracy value on hold-out data: 0.859624981880188
Epoch: 4

Loss on hold-out set: 0.38514728893650074
Accuracy on hold-out set: 0.8624583333333333
MulticlassAccuracy value on hold-out data: 0.862458348274231
Epoch: 5

Loss on hold-out set: 0.3713996725200365
Accuracy on hold-out set: 0.8677083333333333
MulticlassAccuracy value on hold-out data: 0.8677083253860474
Epoch: 6

Loss on hold-out set: 0.3682736954626938
Accuracy on hold-out set: 0.8685
MulticlassAccuracy value on hold-out data: 0.8684999942779541
Epoch: 7
```

```
Loss on hold-out set: 0.3636407248607526
Accuracy on hold-out set: 0.8705416666666667
MulticlassAccuracy value on hold-out data: 0.8705416917800903
Epoch: 8

Loss on hold-out set: 0.35213999955169856
Accuracy on hold-out set: 0.8750416666666667
MulticlassAccuracy value on hold-out data: 0.875041663646698
Returned to Spot: Validation loss: 0.35213999955169856
-----------------------------------------------
spotPython tuning: 0.35213999955169856 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2b2edd990>
```

16.7.3 Results

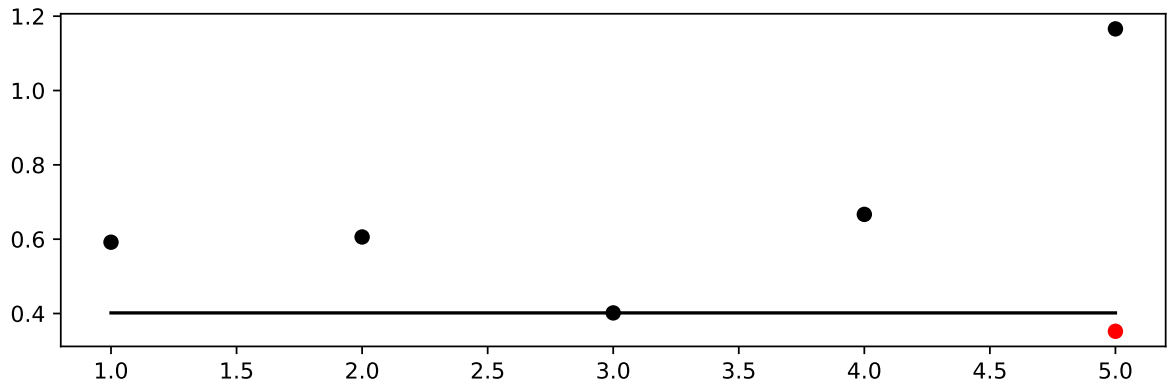
```
SAVE = False
LOAD = False

if SAVE:
    result_file_name = "res_" + experiment_name + ".pkl"
    with open(result_file_name, 'wb') as f:
        pickle.dump(spot_tuner, f)

if LOAD:
    result_file_name = "res_ch10-friedman-hpt-0_maans03_60min_20init_1K_2023-04-14_10-11-1
    with open(result_file_name, 'rb') as f:
        spot_tuner =  pickle.load(f)
```

- Show the Progress of the hyperparameter tuning:

```
spot_tuner.y
```

```
array([0.59183682, 0.60593627, 0.40162234, 0.66671924, 1.16583746,
       0.35214   ])
```

```
spot_tuner.plot_progress(log_y=False, filename="./figures/" + experiment_name+"_progress.p
```



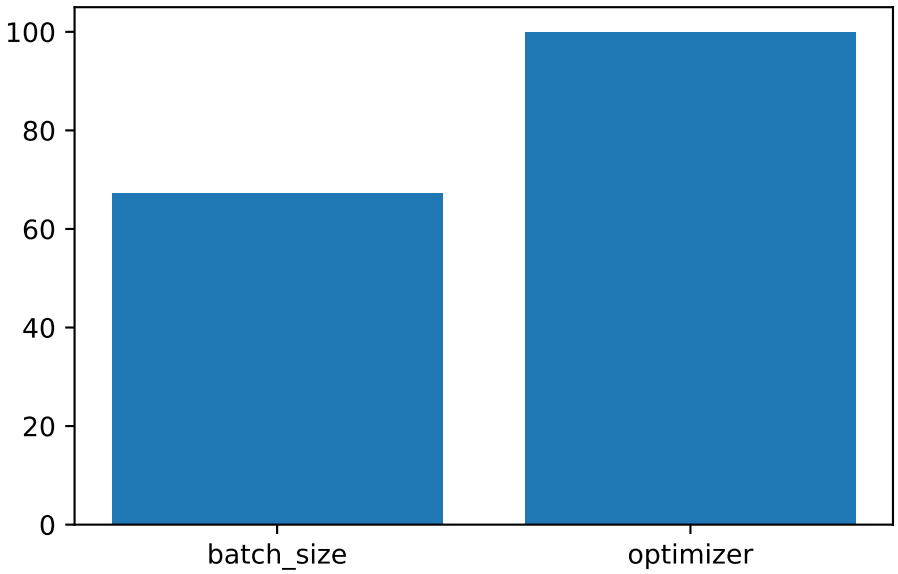
Print the results

```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

```
| name          | type    | default    |   lower |   upper |                tuned | transform
|---------------|---------|------------|---------|---------|----------------------|-------------
| l1            | int     | 5          |     2.0 |     9.0 |                  8.0 | transform_pow
| l2            | int     | 5          |     2.0 |     9.0 |                  8.0 | transform_pow
| lr_mult       | float   | 1.0        |     0.1 |    10.0 | 0.39410343836663486 | None
| batch_size    | int     | 4          |     1.0 |     4.0 |                  4.0 | transform_pow
| epochs        | int     | 3          |     2.0 |     3.0 |                  3.0 | transform_pow
| k_folds       | int     | 1          |     0.0 |     0.0 |                  0.0 | None
| patience      | int     | 5          |     2.0 |     2.0 |                  2.0 | None
| optimizer     | factor  | SGD        |     0.0 |    12.0 |                  1.0 | None
| sgd_momentum  | float   | 0.0        |     0.0 |     1.0 | 0.15594817237081737 | None
```

16.8 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```



16.9 Get SPOT Results

The architecture of the `spotPython` model can be obtained by the following code:

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_fashionMNIST(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=256, bias=True)
    (3): ReLU()
    (4): Linear(in_features=256, out_features=10, bias=True)
  )
)
```

16.10 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
fc = fun_control
fc.update({"core_model_hyper_dict":
    hyper_dict[fun_control["core_model"].__name__]})
model_default = get_one_core_model_from_X(X_start, fun_control=fc)
model_default
```

```
Net_fashionMNIST(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=32, bias=True)
    (3): ReLU()
    (4): Linear(in_features=32, out_features=10, bias=True)
  )
)
```

16.11 Evaluation of the Default and the Tuned Architectures

The method `train_tuned` takes a model architecture without trained weights and trains this model with the train data. The train data is split into train and validation data. The validation data is used for early stopping. The trained model weights are saved as a dictionary.

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
    )
train_tuned(net=model_default, train_dataset=train, shuffle=True,
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        device = fun_control["device"],
        show_batch_interval=1_000_000,
        path=None,
        task=fun_control["task"],)

test_tuned(net=model_default, test_dataset=test,
```

```
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        shuffle=False,
        device = fun_control["device"],
        task=fun_control["task"],)
```

```
Epoch: 1

Loss on hold-out set: 1.9906350915431976
Accuracy on hold-out set: 0.24491666666666667
MulticlassAccuracy value on hold-out data: 0.2449166625738144
Epoch: 2

Loss on hold-out set: 1.4683835213184357
Accuracy on hold-out set: 0.5572083333333333
MulticlassAccuracy value on hold-out data: 0.5572083592414856
Epoch: 3

Loss on hold-out set: 1.1979320629437764
Accuracy on hold-out set: 0.6045416666666666
MulticlassAccuracy value on hold-out data: 0.6045416593551636
Epoch: 4

Loss on hold-out set: 1.0542242929935455
Accuracy on hold-out set: 0.62825
MulticlassAccuracy value on hold-out data: 0.628250002861023
Epoch: 5

Loss on hold-out set: 0.9663734111785889
Accuracy on hold-out set: 0.6489583333333333
MulticlassAccuracy value on hold-out data: 0.6489583253860474
Epoch: 6

Loss on hold-out set: 0.9055797074437142
Accuracy on hold-out set: 0.6669583333333333
MulticlassAccuracy value on hold-out data: 0.6669583320617676
Epoch: 7
```

```
Loss on hold-out set: 0.862669857263565
Accuracy on hold-out set: 0.6820416666666667
MulticlassAccuracy value on hold-out data: 0.6820416450500488
Epoch: 8

Loss on hold-out set: 0.8289879405697187
Accuracy on hold-out set: 0.6920833333333334
MulticlassAccuracy value on hold-out data: 0.6920833587646484
Returned to Spot: Validation loss: 0.8289879405697187
------------------------------------------------


Loss on hold-out set: 0.8460753546714783
Accuracy on hold-out set: 0.6787
MulticlassAccuracy value on hold-out data: 0.6786999702453613
Final evaluation: Validation loss: 0.8460753546714783
Final evaluation: Validation metric: 0.6786999702453613
------------------------------------------------


(0.8460753546714783, nan, tensor(0.6787, device='mps:0'))
```

The following code trains the model `model_spot`. If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be saved to this file.

```
train_tuned(net=model_spot, train_dataset=train,
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        shuffle=True,
        device = fun_control["device"],
        path=None,
        task=fun_control["task"],)
#| echo: true
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"],
            task=fun_control["task"],)
```

```
Epoch: 1
```

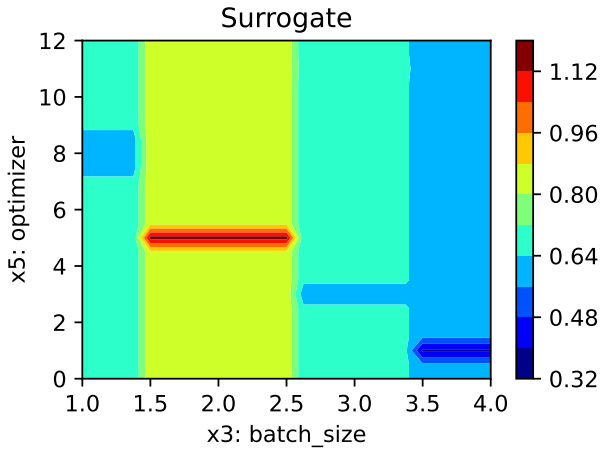
```
Loss on hold-out set: 0.4794034645060698
Accuracy on hold-out set: 0.8305416666666666
MulticlassAccuracy value on hold-out data: 0.8305416703224182
Epoch: 2

Loss on hold-out set: 0.4259570139373342
Accuracy on hold-out set: 0.8504166666666667
MulticlassAccuracy value on hold-out data: 0.8504166603088379
Epoch: 3

Loss on hold-out set: 0.4010279770568013
Accuracy on hold-out set: 0.8564583333333333
MulticlassAccuracy value on hold-out data: 0.856458306312561
Epoch: 4

Loss on hold-out set: 0.38146660766998924
Accuracy on hold-out set: 0.8628333333333333
MulticlassAccuracy value on hold-out data: 0.8628333210945129
Epoch: 5

Loss on hold-out set: 0.37652173261592786
Accuracy on hold-out set: 0.8648333333333333
MulticlassAccuracy value on hold-out data: 0.8648333549499512
Epoch: 6

Loss on hold-out set: 0.3684048171291749
Accuracy on hold-out set: 0.8678333333333333
MulticlassAccuracy value on hold-out data: 0.8678333163261414
Epoch: 7

Loss on hold-out set: 0.3692953255151709
Accuracy on hold-out set: 0.8680833333333333
MulticlassAccuracy value on hold-out data: 0.8680833578109741
Epoch: 8

Loss on hold-out set: 0.3551399592893819
Accuracy on hold-out set: 0.8725833333333334
MulticlassAccuracy value on hold-out data: 0.8725833296775818
Returned to Spot: Validation loss: 0.3551399592893819
-----------------------------------------------
```

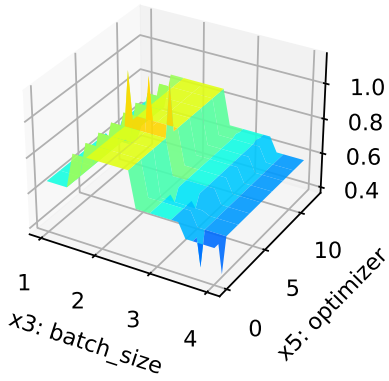
```
Loss on hold-out set: 0.39007223044633865
Accuracy on hold-out set: 0.8651
MulticlassAccuracy value on hold-out data: 0.8651000261306763
Final evaluation: Validation loss: 0.39007223044633865
Final evaluation: Validation metric: 0.8651000261306763
------------------------------------------------

(0.39007223044633865, nan, tensor(0.8651, device='mps:0'))
```

16.12 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
batch_size:  67.35725241508129
optimizer:  100.0
```





16.13 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html
```

16.14 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

17 Hyperparameter Tuning: PyTorch wth cifar10 Data

In this tutorial, we will show how `spotPython` can be integrated into the `PyTorch` training workflow.

This document refers to the following software versions:

- `python`: 3.10.10
- `torch`: 2.0.1
- `torchvision`: 0.15.0
- `spotPython`: 0.2.29

`spotPython` can be installed via pip. Alternatively, the source code can be downloaded from gitHub: https://github.com/sequential-parameter-optimization/spotPython.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of `spotPython` from gitHub.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

17.1 Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = None # "cpu" # "cuda:0"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

```
mps
```

```
24-torch_bartz09_1min_5init_2023-06-15_02-25-00
```

17.2 Initialization of the `fun_control` Dictionary

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process. This dictionary is called `fun_control` and is initialized with the function `fun_control_init`. The function `fun_control_init` returns a skeleton dictionary. The dictionary is filled with the required information for the hyperparameter tuning process. It stores the hyperparameter tuning settings, e.g., the deep learning network architecture that should be tuned, the classification (or regression) problem, and the data that is used for the tuning. The dictionary is used as an input for the SPOT function.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/12_spot_hpt_torch_cifar10",
    device=DEVICE)
```

```
import torch
print(torch.__version__)
# Check that MPS is available
if not torch.backends.mps.is_available():
    if not torch.backends.mps.is_built():
        print("MPS not available because the current PyTorch install was not "
              "built with MPS enabled.")
    else:
        print("MPS not available because the current MacOS version is not 12.3+ "
              "and/or you do not have an MPS-enabled device on this machine.")

else:
    mps_device = torch.device("mps")
    print("MPS device: ", mps_device)
```

```
2.0.1
MPS device:  mps
```

17.3 PyTorch Data Loading

17.4 1. Load Data Cifar10 Data

```
from torchvision import datasets, transforms
import torchvision
def load_data(data_dir="./data"):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    trainset = torchvision.datasets.CIFAR10(
        root=data_dir, train=True, download=True, transform=transform)

    testset = torchvision.datasets.CIFAR10(
        root=data_dir, train=False, download=True, transform=transform)

    return trainset, testset
train, test = load_data()
train.data.shape, test.data.shape
n_samples = len(train)
```

```
Files already downloaded and verified
```

```
Files already downloaded and verified
```

- Since this works fine, we can add the data loading to the `fun_control` dictionary:

```
# add the dataset to the fun_control
fun_control.update({"data": None, # dataset,
                "train": train,
                "test": test,
                "n_samples": n_samples,
                "target_column": None})
```

17.5 Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling

of the data or for the one-hot encoding of categorical variables. The preprocessing model is called `prep_model` ("preparation" or pre-processing) and includes steps that are not subject to the hyperparameter tuning process. The preprocessing model is specified in the `fun_control` dictionary. The preprocessing model can be implemented as a `sklearn` pipeline. The following code shows a typical preprocessing pipeline:

```
# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
#         transformers=[
#             ("categorical", one_hot_encoder, categorical_columns),
#         ],
#         remainder=StandardScaler(),
#     )
prep_model = None
fun_control.update({"prep_model": prep_model})
```

17.6 Select `algorithm` and `core_model_hyper_dict`

17.6.1 Implementing a Configurable Neural Network With spotPython

`spotPython` includes the `Net_CIFAR10` class which is implemented in the file `netcifar10.py`. The class is imported here.

17.6.1.1 The `Net_Core` class

`Net_lin_reg` inherits from the class `Net_Core` which is implemented in the file `netcore.py`. It implements the additional attributes that are common to all neural network models. The `Net_Core` class is implemented in the file `netcore.py`. It implements hyperparameters as attributes, that are not used by the `core_model`, e.g.:

- optimizer (`optimizer`),
- learning rate (`lr`),
- batch size (`batch_size`),
- epochs (`epochs`),
- k_folds (`k_folds`), and
- early stopping criterion "patience" (`patience`).

Users can add further attributes to the class. The class `Net_Core` is shown below.

```
from torch import nn


class Net_Core(nn.Module):
    def __init__(self, lr_mult, batch_size, epochs, k_folds, patience,
    optimizer, sgd_momentum):
        super(Net_Core, self).__init__()
        self.lr_mult = lr_mult
        self.batch_size = batch_size
        self.epochs = epochs
        self.k_folds = k_folds
        self.patience = patience
        self.optimizer = optimizer
        self.sgd_momentum = sgd_momentum
```

:::{.callout-note}

We see that the class `Net_lin_reg` has additional attributes and does not inherit from `nn` directly. It adds an additional class, `Net_core`, that takes care of additional attributes that are common to all neural network models, e.g., the learning rate multiplier `lr_mult` or the batch size `batch_size`.

```
from spotPython.torch.netcifar10 import Net_CIFAR10
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
core_model = Net_CIFAR10
fun_control = add_core_model_to_fun_control(core_model=core_model,
                               fun_control=fun_control,
                               hyper_dict=TorchHyperDict,
                               filename=None)
```

17.7 The Search Space

17.7.1 Configuring the Search Space With spotPython

17.7.1.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

`spotPython` uses `JSON` files for the specification of the hyperparameters. The `JSON` file for the `core_model` is called `torch_hyper_dict.json`. The corresponding entries for the `Net_CIFAR10` class are shown below.

spotPython can handle numerical, boolean, and categorical hyperparameters. They can be specified in the JSON file in a similar way as the numerical hyperparameters as shown below. Each entry in the JSON file represents one hyperparameter with the following structure: type, default, transform, lower, and upper.

```
"factor_hyperparameter": {
    "levels": ["A", "B", "C"],
    "type": "factor",
    "default": "B",
    "transform": "None",
    "core_model_parameter_type": "str",
    "lower": 0,
    "upper": 2},
```

The corresponding entries for the Net_CIFAR10 class are shown below.

```
    "Net_CIFAR10":
    {
        "l1": {
            "type": "int",
            "default": 5,
            "transform": "transform_power_2_int",
            "lower": 2,
            "upper": 9},
        "l2": {
            "type": "int",
            "default": 5,
            "transform": "transform_power_2_int",
            "lower": 2,
            "upper": 9},
        "lr_mult": {
            "type": "float",
            "default": 1.0,
            "transform": "None",
            "lower": 0.1,
            "upper": 10.0},
        "batch_size": {
            "type": "int",
            "default": 4,
            "transform": "transform_power_2_int",
            "lower": 1,
            "upper": 4},
```

```
        "epochs": {
            "type": "int",
            "default": 3,
            "transform": "transform_power_2_int",
            "lower": 3,
            "upper": 4},
        "k_folds": {
            "type": "int",
            "default": 1,
            "transform": "None",
            "lower": 1,
            "upper": 1},
        "patience": {
            "type": "int",
            "default": 5,
            "transform": "None",
            "lower": 2,
            "upper": 10
        },
        "optimizer": {
            "levels": ["Adadelta", "Adagrad", "Adam", "AdamW", "SparseAdam", "Adamax", "AS
            "type": "factor",
            "default": "SGD",
            "transform": "None",
            "class_name": "torch.optim",
            "core_model_parameter_type": "str",
            "lower": 0,
            "upper": 12},
        "sgd_momentum": {
            "type": "float",
            "default": 0.0,
            "transform": "None",
            "lower": 0.0,
            "upper": 1.0}
    },
```

17.8 Modifying the Hyperparameters

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python

source code. These functions are described in the following.

17.8.1 Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

After specifying the model, the corresponding hyperparameters, their types and bounds are loaded from the `JSON` file `torch_hyper_dict.json`. After loading, the user can modify the hyperparameters, e.g., the bounds. `spotPython` provides a simple rule for de-activating hyperparameters: If the lower and the upper bound are set to identical values, the hyperparameter is de-activated. This is useful for the hyperparameter tuning, because it allows to specify a hyperparameter in the `JSON` file, but to de-activate it in the `fun_control` dictionary. This is done in the next step.

17.8.2 Modify Hyperparameters of Type numeric and integer (boolean)

17.9 4. Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

After specifying the model, the corresponding hyperparameters, their types and bounds are loaded from the `JSON` file `torch_hyper_dict.json`. After loading, the user can modify the hyperparameters, e.g., the bounds. `spotPython` provides a clever rule for de-activating hyperparameters. If the lower and the upper bound are set to identical values, the hyperparameter is de-activated. This is useful for the hyperparameter tuning, because it allows to specify a hyperparameter in the `JSON` file, but to de-activate it in the `fun_control` dictionary. This is done in the next step.

17.9.1 Modify hyperparameter of type numeric and integer (boolean)

The hyperparameter `k_folds` is not used, it is de-activated here by setting the lower and upper bound to the same value.

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
# fun_control = modify_hyper_parameter_bounds(fun_control, "delta", bounds=[1e-10, 1e-6])
# fun_control = modify_hyper_parameter_bounds(fun_control, "min_samples_split", bounds=[3,
#fun_control = modify_hyper_parameter_bounds(fun_control, "merit_preprune", bounds=[0, 0])
# fun_control["core_model_hyper_dict"]
fun_control = modify_hyper_parameter_bounds(fun_control, "k_folds", bounds=[2, 2])
```

17.9.2 Modify hyperparameter of type factor

In a similar manner as for the numerical hyperparameters, the categorical hyperparameters can be modified. For example, the hyperparameter `leaf_model` is de-activated here by choosing only one value `"LinearRegression"`.

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam"])
# fun_control = modify_hyper_parameter_levels(fun_control, "leaf_model", ["LinearRegressio
# fun_control["core_model_hyper_dict"]
```

17.9.3 Optimizers

Optimizers can be selected as described in Section 20.7.4.

> **A note on the learning rate**
>
> `spotPython` provides a multiplier for the default learning rates, `lr_mult`, because optimizers use different learning rates. Using a multiplier for the learning rates might enable a simultaneous tuning of the learning rates for all optimizers. However, this is not recommended, because the learning rates are not comparable across optimizers. Therefore, we recommend fixing the learning rate for all optimizers if multiple optimizers are used. This can be done by setting the lower and upper bounds of the learning rate multiplier to the same value as shown below.

Thus, the learning rate, which affects the `SGD` optimizer, will be set to a fixed value. We choose the default value of `1e-3` for the learning rate, because it is used in other `PyTorch` examples (it is also the default value used by `spotPython` as defined in the `optimizer_handler()` method). We recommend tuning the learning rate later, when a reduced set of optimizers is fixed. Here, we will demonstrate how to select in a screening phase the optimizers that should be used for the hyperparameter tuning.

For the same reason, we will fix the `sgd_momentum` to `0.9`.

```
fun_control = modify_hyper_parameter_bounds(fun_control,
    "lr_mult", bounds=[1e-3, 1e-3])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "sgd_momentum", bounds=[0.9, 0.9])
```

17.10 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set and
2. the loss function (and a metric).

These are described in Section 20.8.

The loss function is specified by the key `"loss_function"`. We will use CrossEntropy loss for the multiclass-classification task.

```
from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({"loss_function": loss_function})
```

In addition to the loss functions, `spotPython` provides access to a large number of metrics.

- The key `"metric_sklearn"` is used for metrics that follow the `scikit-learn` conventions.
- The key `"river_metric"` is used for the river based evaluation (Montiel et al. 2021) via `eval_oml_iter_progressive`, and
- the key `"metric_torch"` is used for the metrics from `TorchMetrics`.

`TorchMetrics` is a collection of more than 90 PyTorch metrics[1].

A description can be found in Section 20.8.2.

```
import torchmetrics
metric_torch = torchmetrics.Accuracy(task="multiclass", num_classes=10).to(fun_control["de
fun_control.update({"metric_torch": metric_torch})
```

17.11 Calling the SPOT Function

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` generates a design table as follows:

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

[1] https://torchmetrics.readthedocs.io/en/latest/.

name	type	default	lower	upper	transform
l1	int	5	2	9	transform_power_2_int
l2	int	5	2	9	transform_power_2_int
lr_mult	float	1.0	0.001	0.001	None
batch_size	int	4	1	4	transform_power_2_int
epochs	int	3	3	4	transform_power_2_int
k_folds	int	1	2	2	None
patience	int	5	2	10	None
optimizer	factor	SGD	0	0	None
sgd_momentum	float	0.0	0.9	0.9	None

This allows to check if all information is available and if the information is correct.

The objective function `fun_torch` is selected next. It implements an interface from `PyTorch`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch
```

The `spotPython` hyperparameter tuning is started by calling the `Spot` function. Here, we will run the tuner for approximately 30 minutes (`max_time`). Note: the initial design is always evaluated in the `spotPython` run. As a consequence, the run may take longer than specified by `max_time`, because the evaluation time of initial design (here: `init_size`, 10 points) is performed independently of `max_time`.

```
import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = inf,
                    fun_repeats = 1,
                    max_time = MAX_TIME,
                    noise = False,
                    tolerance_x = np.sqrt(np.spacing(1)),
                    var_type = var_type,
                    var_name = var_name,
                    infill_criterion = "y",
                    n_points = 1,
                    seed=123,
```

```
                    log_level = 50,
                    show_models= False,
                    show_progress= True,
                    fun_control = fun_control,
                    design_control={"init_size": INIT_SIZE,
                                    "repeats": 1},
                    surrogate_control={"noise": True,
                                       "cod_type": "norm",
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": len(var_name),
                                       "model_fun_evals": 10_000,
                                       "log_level": 50
                                       })
spot_tuner.run(X_start=X_start)
```

```
config: {'l1': 128, 'l2': 8, 'lr_mult': 0.001, 'batch_size': 16, 'epochs': 16, 'k_folds': 2,
Epoch: 1

Loss on hold-out set: 2.311683391952515
Accuracy on hold-out set: 0.0998
MulticlassAccuracy value on hold-out data: 0.0997999981045723
Epoch: 2

Loss on hold-out set: 2.311245431137085
Accuracy on hold-out set: 0.09795
MulticlassAccuracy value on hold-out data: 0.09794999659061432
Epoch: 3

Loss on hold-out set: 2.310761882019043
Accuracy on hold-out set: 0.09735
MulticlassAccuracy value on hold-out data: 0.09735000133514404
Epoch: 4

Loss on hold-out set: 2.3100209629058837
Accuracy on hold-out set: 0.0926
MulticlassAccuracy value on hold-out data: 0.09260000288486481
Epoch: 5
```

```
Loss on hold-out set: 2.309134296989441
Accuracy on hold-out set: 0.09245
MulticlassAccuracy value on hold-out data: 0.09245000034570694
Epoch: 6

Loss on hold-out set: 2.308236855506897
Accuracy on hold-out set: 0.0953
MulticlassAccuracy value on hold-out data: 0.09529999643564224
Epoch: 7

Loss on hold-out set: 2.307350242614746
Accuracy on hold-out set: 0.09625
MulticlassAccuracy value on hold-out data: 0.09624999761581421
Epoch: 8

Loss on hold-out set: 2.306451375961304
Accuracy on hold-out set: 0.0971
MulticlassAccuracy value on hold-out data: 0.09709999710321426
Epoch: 9

Loss on hold-out set: 2.3055309701919557
Accuracy on hold-out set: 0.09795
MulticlassAccuracy value on hold-out data: 0.09794999659061432
Epoch: 10

Loss on hold-out set: 2.3045993661880493
Accuracy on hold-out set: 0.09785
MulticlassAccuracy value on hold-out data: 0.097850002348423
Epoch: 11

Loss on hold-out set: 2.303634782791138
Accuracy on hold-out set: 0.0979
MulticlassAccuracy value on hold-out data: 0.09790000319480896
Epoch: 12

Loss on hold-out set: 2.3026232616424562
Accuracy on hold-out set: 0.0983
MulticlassAccuracy value on hold-out data: 0.09830000251531601
Epoch: 13
```

```
Loss on hold-out set: 2.3015770915985105
Accuracy on hold-out set: 0.0993
MulticlassAccuracy value on hold-out data: 0.09929999709129333
Epoch: 14

Loss on hold-out set: 2.300471157836914
Accuracy on hold-out set: 0.1
MulticlassAccuracy value on hold-out data: 0.10000000149011612
Epoch: 15

Loss on hold-out set: 2.2991993629455565
Accuracy on hold-out set: 0.10025
MulticlassAccuracy value on hold-out data: 0.1002499982714653
Epoch: 16

Loss on hold-out set: 2.2978011751174927
Accuracy on hold-out set: 0.10155
MulticlassAccuracy value on hold-out data: 0.10154999792575836
Returned to Spot: Validation loss: 2.2978011751174927
--------------------------------------------

config: {'l1': 16, 'l2': 16, 'lr_mult': 0.001, 'batch_size': 8, 'epochs': 8, 'k_folds': 2, '
Epoch: 1

Loss on hold-out set: 2.3132267066001893
Accuracy on hold-out set: 0.09125
MulticlassAccuracy value on hold-out data: 0.09125000238418579
Epoch: 2

Loss on hold-out set: 2.3107895686149598
Accuracy on hold-out set: 0.0944
MulticlassAccuracy value on hold-out data: 0.09440000355243683
Epoch: 3

Loss on hold-out set: 2.308547576713562
Accuracy on hold-out set: 0.0952
MulticlassAccuracy value on hold-out data: 0.09520000219345093
Epoch: 4
```

```
Loss on hold-out set: 2.305959084892273
Accuracy on hold-out set: 0.09545
MulticlassAccuracy value on hold-out data: 0.09544999897480011
Epoch: 5

Loss on hold-out set: 2.3028558511734007
Accuracy on hold-out set: 0.09585
MulticlassAccuracy value on hold-out data: 0.09584999829530716
Epoch: 6

Loss on hold-out set: 2.299030938720703
Accuracy on hold-out set: 0.106
MulticlassAccuracy value on hold-out data: 0.10599999874830246
Epoch: 7

Loss on hold-out set: 2.2942890069007875
Accuracy on hold-out set: 0.14105
MulticlassAccuracy value on hold-out data: 0.14104999601840973
Epoch: 8

Loss on hold-out set: 2.288422750377655
Accuracy on hold-out set: 0.15335
MulticlassAccuracy value on hold-out data: 0.15334999561309814
Returned to Spot: Validation loss: 2.288422750377655
--------------------------------------------

config: {'l1': 256, 'l2': 128, 'lr_mult': 0.001, 'batch_size': 2, 'epochs': 16, 'k_folds': 2
Epoch: 1

Loss on hold-out set: 2.2911675258874893
Accuracy on hold-out set: 0.11515
MulticlassAccuracy value on hold-out data: 0.11514999717473984
Epoch: 2

Loss on hold-out set: 2.2604329010009767
Accuracy on hold-out set: 0.1692
MulticlassAccuracy value on hold-out data: 0.16920000314712524
Epoch: 3
```

```
Loss on hold-out set: 2.2155130308508872
Accuracy on hold-out set: 0.22425
MulticlassAccuracy value on hold-out data: 0.22425000369548798
Epoch: 4

Loss on hold-out set: 2.1646438235402106
Accuracy on hold-out set: 0.24075
MulticlassAccuracy value on hold-out data: 0.2407499998807907
Epoch: 5

Loss on hold-out set: 2.1181887595176696
Accuracy on hold-out set: 0.25075
MulticlassAccuracy value on hold-out data: 0.25075000524520874
Epoch: 6

Loss on hold-out set: 2.0841842572927476
Accuracy on hold-out set: 0.2604
MulticlassAccuracy value on hold-out data: 0.2603999972343445
Epoch: 7

Loss on hold-out set: 2.059577731692791
Accuracy on hold-out set: 0.268
MulticlassAccuracy value on hold-out data: 0.2680000066757202
Epoch: 8

Loss on hold-out set: 2.039384551268816
Accuracy on hold-out set: 0.2771
MulticlassAccuracy value on hold-out data: 0.27709999680519104
Epoch: 9

Loss on hold-out set: 2.0205896859884263
Accuracy on hold-out set: 0.2854
MulticlassAccuracy value on hold-out data: 0.28540000319480896
Epoch: 10

Loss on hold-out set: 2.0014112600386142
Accuracy on hold-out set: 0.293
MulticlassAccuracy value on hold-out data: 0.2930000126361847
Epoch: 11
```

```
Loss on hold-out set: 1.98119487208426
Accuracy on hold-out set: 0.3015
MulticlassAccuracy value on hold-out data: 0.30149999260902405
Epoch: 12

Loss on hold-out set: 1.9605134097576142
Accuracy on hold-out set: 0.30825
MulticlassAccuracy value on hold-out data: 0.3082500100135803
Epoch: 13

Loss on hold-out set: 1.9401178082287311
Accuracy on hold-out set: 0.31155
MulticlassAccuracy value on hold-out data: 0.31154999136924744
Epoch: 14

Loss on hold-out set: 1.9207591419935226
Accuracy on hold-out set: 0.31625
MulticlassAccuracy value on hold-out data: 0.3162499964237213
Epoch: 15

Loss on hold-out set: 1.9031057552337647
Accuracy on hold-out set: 0.32135
MulticlassAccuracy value on hold-out data: 0.32135000824928284
Epoch: 16

Loss on hold-out set: 1.88704205583632
Accuracy on hold-out set: 0.32765
MulticlassAccuracy value on hold-out data: 0.3276500105857849
Returned to Spot: Validation loss: 1.88704205583632
-----------------------------------------------

config: {'l1': 8, 'l2': 32, 'lr_mult': 0.001, 'batch_size': 4, 'epochs': 8, 'k_folds': 2, 'pa
Epoch: 1

Loss on hold-out set: 2.3067592100143433
Accuracy on hold-out set: 0.1034
MulticlassAccuracy value on hold-out data: 0.10339999943971634
Epoch: 2
```

```
Loss on hold-out set: 2.302479398679733
Accuracy on hold-out set: 0.1038
MulticlassAccuracy value on hold-out data: 0.10379999876022339
Epoch: 3

Loss on hold-out set: 2.299649449634552
Accuracy on hold-out set: 0.1036
MulticlassAccuracy value on hold-out data: 0.10360000282526016
Epoch: 4

Loss on hold-out set: 2.2961894056797028
Accuracy on hold-out set: 0.10285
MulticlassAccuracy value on hold-out data: 0.10284999758005142
Epoch: 5

Loss on hold-out set: 2.291740715122223
Accuracy on hold-out set: 0.10055
MulticlassAccuracy value on hold-out data: 0.10055000334978104
Epoch: 6

Loss on hold-out set: 2.2863934195041655
Accuracy on hold-out set: 0.1067
MulticlassAccuracy value on hold-out data: 0.10670000314712524
Epoch: 7

Loss on hold-out set: 2.2807027538776397
Accuracy on hold-out set: 0.12785
MulticlassAccuracy value on hold-out data: 0.1278499960899353
Epoch: 8

Loss on hold-out set: 2.274674406194687
Accuracy on hold-out set: 0.15445
MulticlassAccuracy value on hold-out data: 0.15444999933242798
Returned to Spot: Validation loss: 2.274674406194687
--------------------------------------------

config: {'l1': 64, 'l2': 512, 'lr_mult': 0.001, 'batch_size': 8, 'epochs': 16, 'k_folds': 2,
Epoch: 1
```

```
Loss on hold-out set: 2.2979142548561096
Accuracy on hold-out set: 0.1438
MulticlassAccuracy value on hold-out data: 0.14380000531673431
Epoch: 2

Loss on hold-out set: 2.2909610679626464
Accuracy on hold-out set: 0.1322
MulticlassAccuracy value on hold-out data: 0.13220000267028809
Epoch: 3

Loss on hold-out set: 2.279294453239441
Accuracy on hold-out set: 0.13775
MulticlassAccuracy value on hold-out data: 0.13774999976158142
Epoch: 4

Loss on hold-out set: 2.2606463893890383
Accuracy on hold-out set: 0.1589
MulticlassAccuracy value on hold-out data: 0.15889999270439148
Epoch: 5

Loss on hold-out set: 2.23454817199707
Accuracy on hold-out set: 0.1822
MulticlassAccuracy value on hold-out data: 0.18219999969005585
Epoch: 6

Loss on hold-out set: 2.2028478468894956
Accuracy on hold-out set: 0.2088
MulticlassAccuracy value on hold-out data: 0.20880000293254852
Epoch: 7

Loss on hold-out set: 2.1700289571285247
Accuracy on hold-out set: 0.22545
MulticlassAccuracy value on hold-out data: 0.22544999420642853
Epoch: 8

Loss on hold-out set: 2.140118355131149
Accuracy on hold-out set: 0.23595
MulticlassAccuracy value on hold-out data: 0.23594999313354492
Epoch: 9
```

```
Loss on hold-out set: 2.1146386970996858
Accuracy on hold-out set: 0.2466
MulticlassAccuracy value on hold-out data: 0.24660000205039978
Epoch: 10

Loss on hold-out set: 2.09286047244072
Accuracy on hold-out set: 0.25815
MulticlassAccuracy value on hold-out data: 0.25815001130104065
Epoch: 11

Loss on hold-out set: 2.073692308855057
Accuracy on hold-out set: 0.26495
MulticlassAccuracy value on hold-out data: 0.2649500072002411
Epoch: 12

Loss on hold-out set: 2.056339925813675
Accuracy on hold-out set: 0.27055
MulticlassAccuracy value on hold-out data: 0.270550012588501
Epoch: 13

Loss on hold-out set: 2.0405236787319185
Accuracy on hold-out set: 0.27545
MulticlassAccuracy value on hold-out data: 0.2754499912261963
Epoch: 14

Loss on hold-out set: 2.0261499420642854
Accuracy on hold-out set: 0.28065
MulticlassAccuracy value on hold-out data: 0.28064998984336853
Epoch: 15

Loss on hold-out set: 2.0132086304187773
Accuracy on hold-out set: 0.2828
MulticlassAccuracy value on hold-out data: 0.28279998898506165
Epoch: 16

Loss on hold-out set: 2.0016519038677214
Accuracy on hold-out set: 0.2865
MulticlassAccuracy value on hold-out data: 0.2865000069141388
Returned to Spot: Validation loss: 2.0016519038677214
--------------------------------------------
```

```
config: {'l1': 512, 'l2': 128, 'lr_mult': 0.001, 'batch_size': 2, 'epochs': 16, 'k_folds': 2
Epoch: 1

Loss on hold-out set: 2.2806100524663924
Accuracy on hold-out set: 0.14545
MulticlassAccuracy value on hold-out data: 0.14544999599456787
Epoch: 2

Loss on hold-out set: 2.2248466302752496
Accuracy on hold-out set: 0.2111
MulticlassAccuracy value on hold-out data: 0.2110999971628189
Epoch: 3

Loss on hold-out set: 2.1533449834227563
Accuracy on hold-out set: 0.233
MulticlassAccuracy value on hold-out data: 0.2329999953508377
Epoch: 4

Loss on hold-out set: 2.1005360376358033
Accuracy on hold-out set: 0.248
MulticlassAccuracy value on hold-out data: 0.24799999594688416
Epoch: 5

Loss on hold-out set: 2.0676645808160306
Accuracy on hold-out set: 0.25705
MulticlassAccuracy value on hold-out data: 0.2570500075817108
Epoch: 6

Loss on hold-out set: 2.0427548134982585
Accuracy on hold-out set: 0.26755
MulticlassAccuracy value on hold-out data: 0.267549991607666
Epoch: 7

Loss on hold-out set: 2.019639823168516
Accuracy on hold-out set: 0.27805
MulticlassAccuracy value on hold-out data: 0.2780500054359436
Epoch: 8
```

```
Loss on hold-out set: 1.996830719834566
Accuracy on hold-out set: 0.28725
MulticlassAccuracy value on hold-out data: 0.28725001215934753
Epoch: 9

Loss on hold-out set: 1.9743597572863103
Accuracy on hold-out set: 0.29465
MulticlassAccuracy value on hold-out data: 0.29464998841285706
Epoch: 10

Loss on hold-out set: 1.952907062649727
Accuracy on hold-out set: 0.3026
MulticlassAccuracy value on hold-out data: 0.3025999963283539
Epoch: 11

Loss on hold-out set: 1.9331943237364293
Accuracy on hold-out set: 0.3103
MulticlassAccuracy value on hold-out data: 0.31029999256134033
Epoch: 12

Loss on hold-out set: 1.9156081081062555
Accuracy on hold-out set: 0.3156
MulticlassAccuracy value on hold-out data: 0.3156000077724457
Epoch: 13

Loss on hold-out set: 1.9000840349942445
Accuracy on hold-out set: 0.32185
MulticlassAccuracy value on hold-out data: 0.3218500018119812
Epoch: 14

Loss on hold-out set: 1.886150071644783
Accuracy on hold-out set: 0.32655
MulticlassAccuracy value on hold-out data: 0.3265500068664551
Epoch: 15

Loss on hold-out set: 1.8733494440227747
Accuracy on hold-out set: 0.3322
MulticlassAccuracy value on hold-out data: 0.33219999074935913
Epoch: 16
```

```
Loss on hold-out set: 1.8613311507493258
Accuracy on hold-out set: 0.33635
MulticlassAccuracy value on hold-out data: 0.3363499939441681
Returned to Spot: Validation loss: 1.8613311507493258
-----------------------------------------------


spotPython tuning: 1.8613311507493258 [##########] 100.00% Done...


<spotPython.spot.spot.Spot at 0x2b67fbb20>
```

18 Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard.

18.1 Tensorboard: Start Tensorboard

Start TensorBoard through the command line to visualize data you logged. Specify the root log directory as used in `fun_control = fun_control_init(task="regression", tensorboard_path="runs/24_spot_torch_regression")` as the `tensorboard_path`. The argument logdir points to directory where TensorBoard will look to find event files that it can display. TensorBoard will recursively walk the directory structure rooted at logdir, looking for *.tfevents.* files.

```
tensorboard --logdir=runs
```

Go to the URL it provides or to http://localhost:6006/. The following figures show some screenshots of Tensorboard.

18.1.1 Results

```
SAVE = False
LOAD = False

if SAVE:
    result_file_name = "res_" + experiment_name + ".pkl"
    with open(result_file_name, 'wb') as f:
        pickle.dump(spot_tuner, f)

if LOAD:
    result_file_name = "ADD THE NAME here, e.g.: res_ch10-friedman-hpt-0_maans03_60min_20i
    with open(result_file_name, 'rb') as f:
        spot_tuner =  pickle.load(f)
```

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from Figure 20.4.

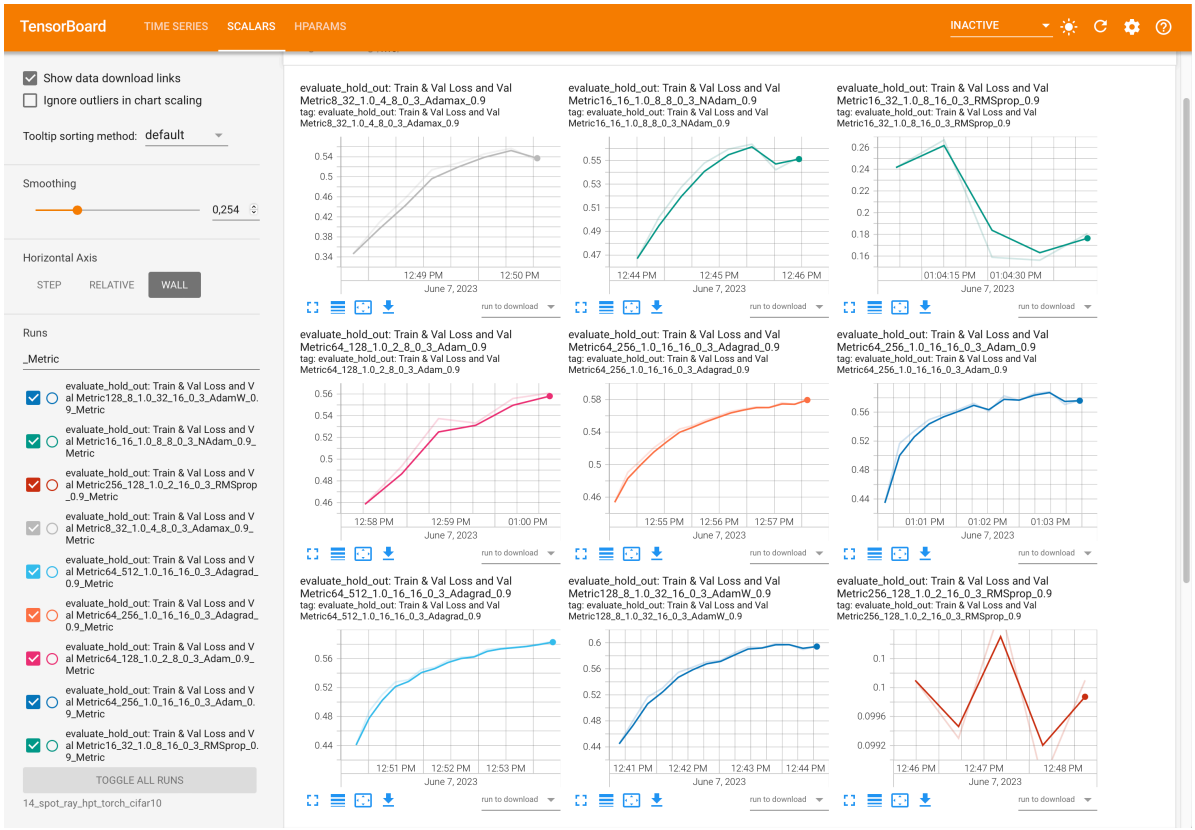


Figure 18.1: Tensorboard

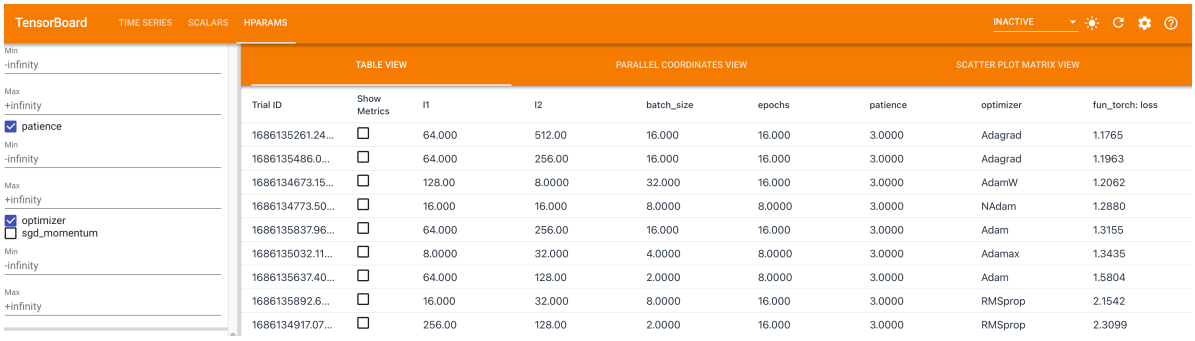
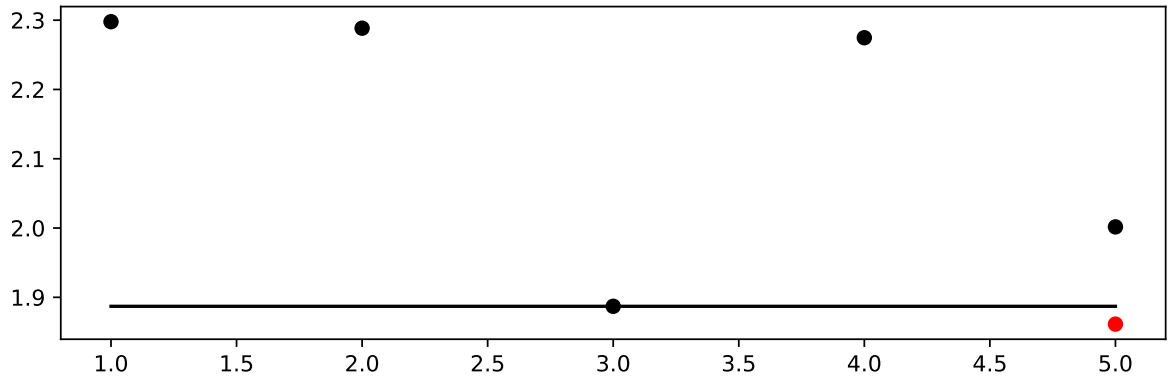


Figure 18.2: Tensorboard

```
spot_tuner.plot_progress(log_y=False, filename="./figures/" + experiment_name+"_progress.p
```



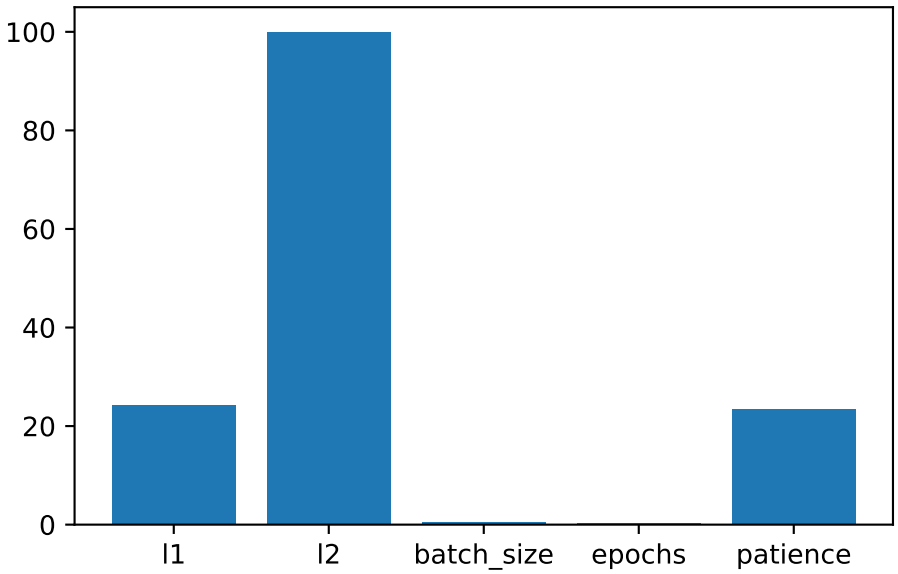
- Print the results

```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

```
| name          | type    | default   |   lower |   upper |   tuned | transform             |
|---------------|---------|-----------|---------|---------|---------|-----------------------|--
| l1            | int     | 5         |     2.0 |     9.0 |     9.0 | transform_power_2_int |
| l2            | int     | 5         |     2.0 |     9.0 |     7.0 | transform_power_2_int |
| lr_mult       | float   | 1.0       |   0.001 |   0.001 |   0.001 | None                  |
| batch_size    | int     | 4         |     1.0 |     4.0 |     1.0 | transform_power_2_int |
| epochs        | int     | 3         |     3.0 |     4.0 |     4.0 | transform_power_2_int |
| k_folds       | int     | 1         |     2.0 |     2.0 |     2.0 | None                  |
| patience      | int     | 5         |     2.0 |    10.0 |     9.0 | None                  |
| optimizer     | factor  | SGD       |     0.0 |     0.0 |     0.0 | None                  |
| sgd_momentum  | float   | 0.0       |     0.9 |     0.9 |     0.9 | None                  |
```

To visualize the most important hyperparameters, `spotPython` provides the function `plot_importance`. The following code generates the importance plot from Figure 20.5.

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```



18.2 Get the Tuned Architecture

The architecture of the `spotPython` model can be obtained by the following code:

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_CIFAR10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=10, bias=True)
)
```

18.3 Evaluation of the Tuned Architecture

The method `train_tuned` takes a model architecture without trained weights and trains this model with the train data. The train data is split into train and validation data. The validation

data is used for early stopping. The trained model weights are saved as a dictionary.

The following code trains the model `model_spot`. If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be saved to this file.

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
    )
train_tuned(net=model_spot, train_dataset=train,
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        shuffle=True,
        device = fun_control["device"],
        path=None,
        task=fun_control["task"],)
```

```
Epoch: 1

Batch: 10000. Batch Size: 2. Training Loss (running): 2.297

Loss on hold-out set: 2.27858503241539
Accuracy on hold-out set: 0.18805
MulticlassAccuracy value on hold-out data: 0.18805000185966492
Epoch: 2

Batch: 10000. Batch Size: 2. Training Loss (running): 2.258

Loss on hold-out set: 2.207171935904026
Accuracy on hold-out set: 0.20975
MulticlassAccuracy value on hold-out data: 0.2097499966621399
Epoch: 3

Batch: 10000. Batch Size: 2. Training Loss (running): 2.168

Loss on hold-out set: 2.1080658203959466
Accuracy on hold-out set: 0.244
MulticlassAccuracy value on hold-out data: 0.24400000274181366
Epoch: 4
```

```
Batch: 10000. Batch Size: 2. Training Loss (running): 2.071

Loss on hold-out set: 2.0400698545455933
Accuracy on hold-out set: 0.26945
MulticlassAccuracy value on hold-out data: 0.26945000886917114
Epoch: 5

Batch: 10000. Batch Size: 2. Training Loss (running): 2.013

Loss on hold-out set: 1.9983556761682033
Accuracy on hold-out set: 0.2832
MulticlassAccuracy value on hold-out data: 0.2831999957561493
Epoch: 6

Batch: 10000. Batch Size: 2. Training Loss (running): 1.977

Loss on hold-out set: 1.9717768412292003
Accuracy on hold-out set: 0.2929
MulticlassAccuracy value on hold-out data: 0.2928999960422516
Epoch: 7

Batch: 10000. Batch Size: 2. Training Loss (running): 1.947

Loss on hold-out set: 1.9547227626144885
Accuracy on hold-out set: 0.2993
MulticlassAccuracy value on hold-out data: 0.2992999851703644
Epoch: 8

Batch: 10000. Batch Size: 2. Training Loss (running): 1.925

Loss on hold-out set: 1.9387803271770476
Accuracy on hold-out set: 0.30765
MulticlassAccuracy value on hold-out data: 0.30764999985694885
Epoch: 9

Batch: 10000. Batch Size: 2. Training Loss (running): 1.922
```

```
Loss on hold-out set: 1.9254024437367916
Accuracy on hold-out set: 0.31145
MulticlassAccuracy value on hold-out data: 0.3114500045776367
Epoch: 10

Batch: 10000. Batch Size: 2. Training Loss (running): 1.912

Loss on hold-out set: 1.9138733855485917
Accuracy on hold-out set: 0.31475
MulticlassAccuracy value on hold-out data: 0.31474998593330383
Epoch: 11

Batch: 10000. Batch Size: 2. Training Loss (running): 1.888

Loss on hold-out set: 1.9002381176412106
Accuracy on hold-out set: 0.31965
MulticlassAccuracy value on hold-out data: 0.31964999437332153
Epoch: 12

Batch: 10000. Batch Size: 2. Training Loss (running): 1.873

Loss on hold-out set: 1.889250943905115
Accuracy on hold-out set: 0.3251
MulticlassAccuracy value on hold-out data: 0.32510000467300415
Epoch: 13

Batch: 10000. Batch Size: 2. Training Loss (running): 1.866

Loss on hold-out set: 1.8762503922969103
Accuracy on hold-out set: 0.3271
MulticlassAccuracy value on hold-out data: 0.32710000872612
Epoch: 14

Batch: 10000. Batch Size: 2. Training Loss (running): 1.857

Loss on hold-out set: 1.8643989986136555
Accuracy on hold-out set: 0.33415
MulticlassAccuracy value on hold-out data: 0.3341499865055084
Epoch: 15
```

```
Batch: 10000. Batch Size: 2. Training Loss (running): 1.841


Loss on hold-out set: 1.8540192962527275
Accuracy on hold-out set: 0.33615
MulticlassAccuracy value on hold-out data: 0.33614999055862427
Epoch: 16


Batch: 10000. Batch Size: 2. Training Loss (running): 1.818


Loss on hold-out set: 1.8393264235019684
Accuracy on hold-out set: 0.3412
MulticlassAccuracy value on hold-out data: 0.34119999408721924
Returned to Spot: Validation loss: 1.8393264235019684
------------------------------------------------
```

If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be loaded from this file.

```
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"],
            task=fun_control["task"],)
```

```
Loss on hold-out set: 1.8219649546802044
Accuracy on hold-out set: 0.3508
MulticlassAccuracy value on hold-out data: 0.3508000075817108
Final evaluation: Validation loss: 1.8219649546802044
Final evaluation: Validation metric: 0.3508000075817108
------------------------------------------------


(1.8219649546802044, nan, tensor(0.3508, device='mps:0'))
```

18.4 Cross-validated Evaluations

```
from spotPython.torch.traintest import evaluate_cv
# modify k-kolds:
setattr(model_spot, "k_folds",  10)
evaluate_cv(net=model_spot,
            dataset=fun_control["data"],
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            task=fun_control["task"],
            writer=fun_control["writer"],
            writerId="model_spot_cv",
            device = fun_control["device"])
```

```
Error in Net_Core. Call to evaluate_cv() failed. err=TypeError("Expected sequence or array-li
```

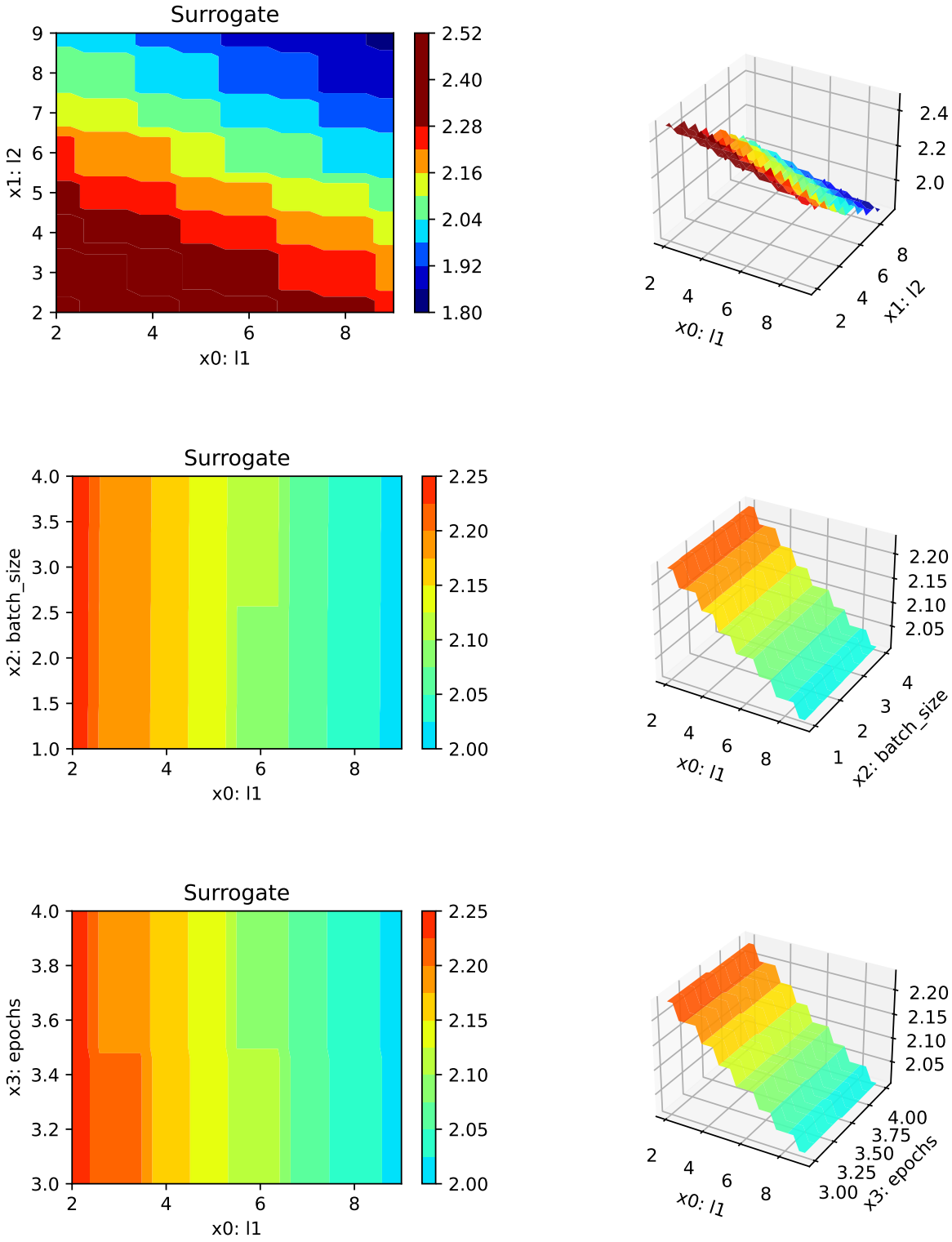
```
(nan, nan, nan)
```

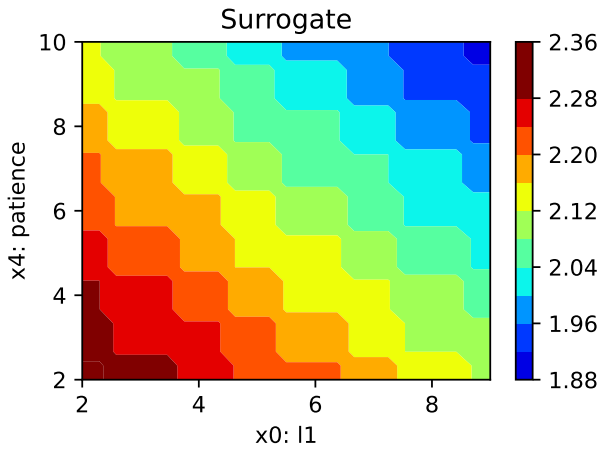
18.5 Detailed Hyperparameter Plots

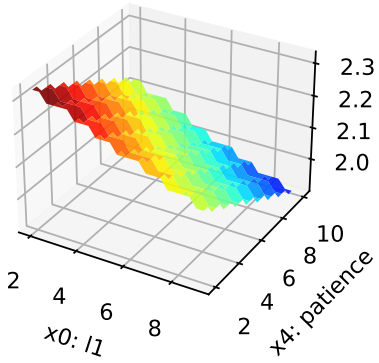
The contour plot in this section visualize the interactions of the two most important hyperparameters, `l1`, and `epochs` of the surrogate model used to optimize the hyperparameters. Since some of these hyperparameters take fatorial or integer values, sometimes step-like fitness landcapes (or response surfaces) are generated. SPOT draws the interactions of the main hyperparameters by default. It is also possible to visualize all interactions. For this, again refer to the notebook (Bartz-Beielstein 2023).

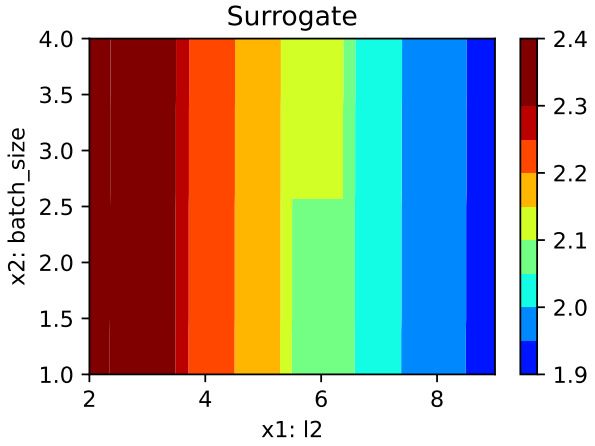
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

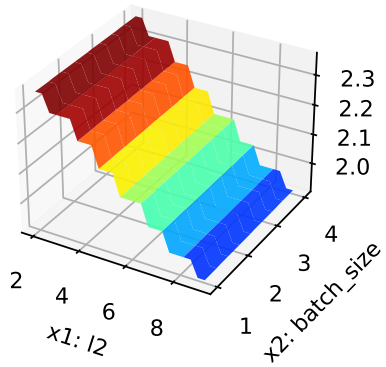
```
l1:  24.211834812180225
l2:  100.0
batch_size:  0.4290185240059406
epochs:  0.28030661323990286
patience:  23.42272897285452
```

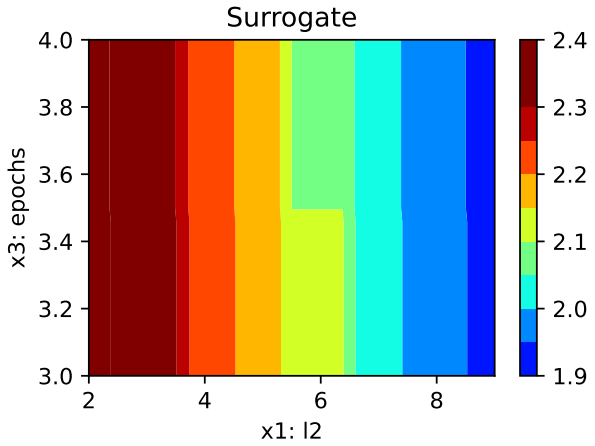

Surrogate
x0: l1
x1: l2
Surrogate
x0: l1
x2: batch_size
Surrogate
x0: l1
x3: epochs

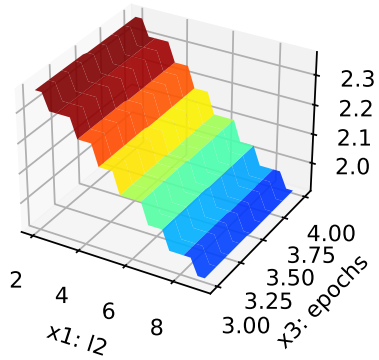

Surrogate
x4: patience
x0: l1
2.36
2.28
2.20
2.12
2.04
1.96
1.88

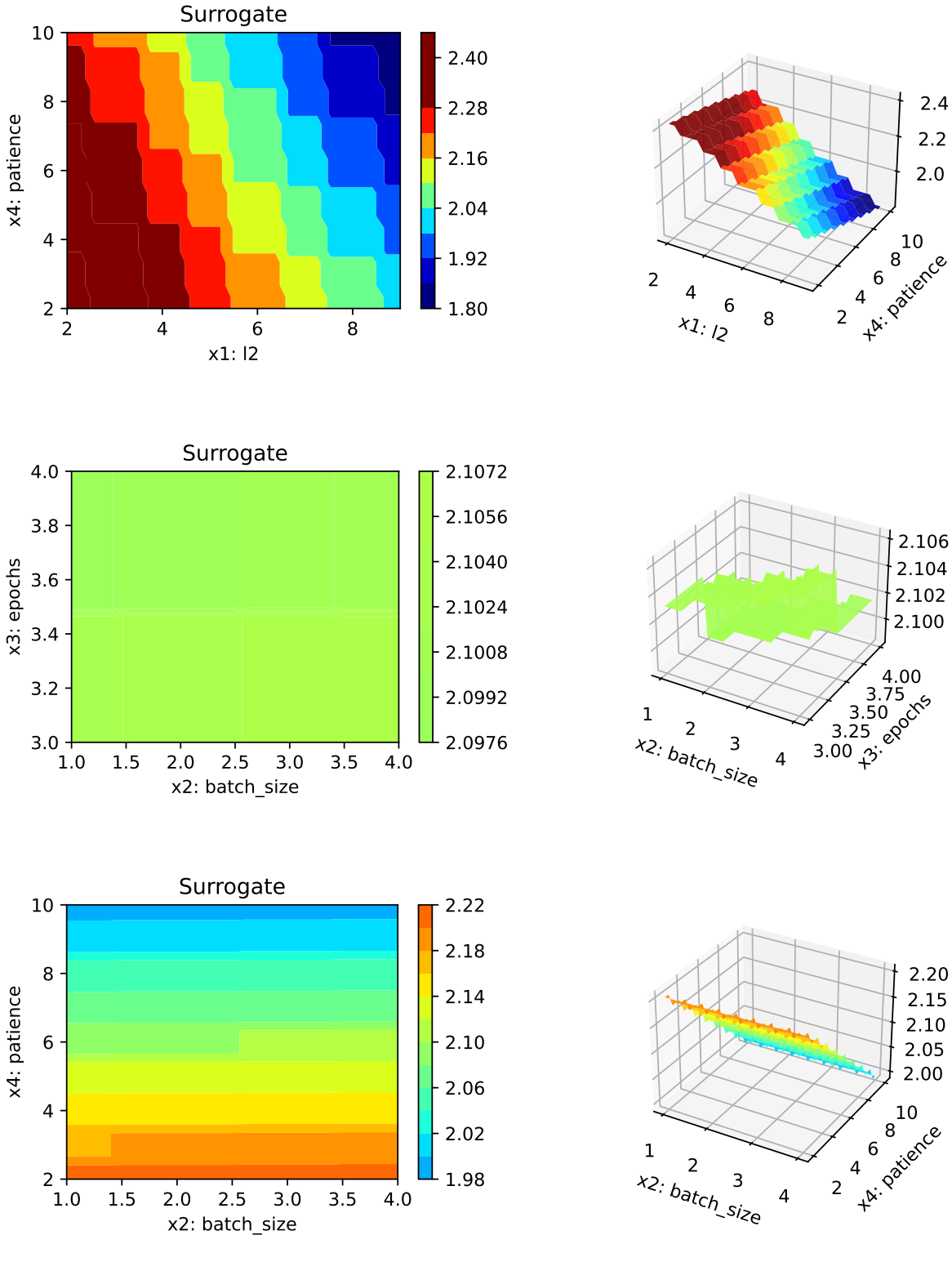

x0: l1
x4: patience

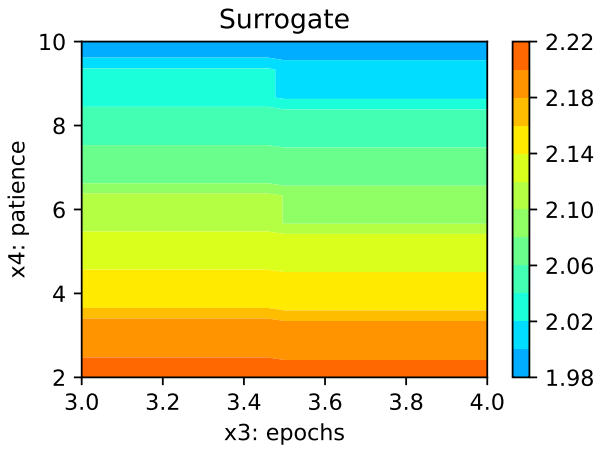

Surrogate
x2: batch_size
x1: l2
2.4
2.3
2.2
2.1
2.0
1.9

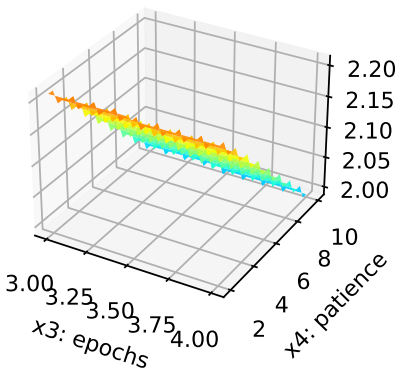

x1: l2
x2: batch_size


Surrogate
x3: epochs
x1: l2
2.4
2.3
2.2
2.1
2.0
1.9


x1: l2
x3: epochs


Surrogate
x4: patience
x1: l2
2.40
2.28
2.16
2.04
1.92
1.80
x1: l2
x4: patience
Surrogate
x3: epochs
x2: batch_size
2.1072
2.1056
2.1040
2.1024
2.1008
2.0992
2.0976
x2: batch_size
x3: epochs
Surrogate
x4: patience
x2: batch_size
2.22
2.18
2.14
2.10
2.06
2.02
1.98
x2: batch_size
x4: patience





18.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

18.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

19 Hyperparameter Tuning for PyTorch With spotPython

In this tutorial, we will show how spotPython can be integrated into the PyTorch training workflow. It is based on the tutorial "Hyperparameter Tuning with Ray Tune" from the PyTorch documentation (PyTorch 2023a), which is an extension of the tutorial "Training a Classifier" (PyTorch 2023b) for training a CIFAR10 image classifier.

This document refers to the following software versions:

- python: 3.10.10
- torch: 2.0.1
- torchvision: 0.15.0
- spotPython: 0.2.29

spotPython can be installed via pip[1].

```
!pip install spotPython
```

Results that refer to the Ray Tune package are taken from https://PyTorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html[2].

19.1 Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = None # "cpu" # "cuda:0"
```

[1] Alternatively, the source code can be downloaded from gitHub: https://github.com/sequential-parameter-optimization/spotPython.

[2] We were not able to install Ray Tune on our system. Therefore, we used the results from the PyTorch tutorial.

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

```
mps
```

```
14-torch_bartz09_1min_5init_2023-06-15_05-22-11
```

19.2 Initialization of the `fun_control` Dictionary

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process. This dictionary is called `fun_control` and is initialized with the function `fun_control_init`. The function `fun_control_init` returns a skeleton dictionary. The dictionary is filled with the required information for the hyperparameter tuning process. It stores the hyperparameter tuning settings, e.g., the deep learning network architecture that should be tuned, the classification (or regression) problem, and the data that is used for the tuning. The dictionary is used as an input for the SPOT function.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/14_spot_ray_hpt_torch_cifar10",
    device=DEVICE,)
```

19.3 Data Loading

The data loading process is implemented in the same manner as described in the Section "Data loaders" in PyTorch (2023a). The data loaders are wrapped into the function `load_data_cifar10` which is identical to the function `load_data` in PyTorch (2023a). A global data directory is used, which allows sharing the data directory between different trials. The method `load_data_cifar10` is part of the `spotPython` package and can be imported from `spotPython.data.torchdata`.

In the following step, the test and train data are added to the dictionary `fun_control`.

```
from spotPython.data.torchdata import load_data_cifar10
train, test = load_data_cifar10()
n_samples = len(train)
# add the dataset to the fun_control
fun_control.update({
```

```
    "train": train,
    "test": test,
    "n_samples": n_samples})
```

```
Files already downloaded and verified
```

```
Files already downloaded and verified
```

19.4 The Model (Algorithm) to be Tuned

19.4.1 Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables. The preprocessing model is called `prep_model` ("preparation" or pre-processing) and includes steps that are not subject to the hyperparameter tuning process. The preprocessing model is specified in the `fun_control` dictionary. The preprocessing model can be implemented as a `sklearn` pipeline. The following code shows a typical preprocessing pipeline:

```
categorical_columns = ["cities", "colors"]
one_hot_encoder = OneHotEncoder(handle_unknown="ignore",
                                    sparse_output=False)
prep_model = ColumnTransformer(
        transformers=[
            ("categorical", one_hot_encoder, categorical_columns),
         ],
         remainder=StandardScaler(),
     )
```

Because the Ray Tune (`ray[tune]`) hyperparameter tuning as described in PyTorch (2023a) does not use a preprocessing model, the preprocessing model is set to `None` here.

```
prep_model = None
fun_control.update({"prep_model": prep_model})
```

19.4.2 Select `algorithm` and `core_model_hyper_dict`

The same neural network model as implemented in the section "Configurable neural network" of the `PyTorch` tutorial (PyTorch 2023a) is used here. We will show the implementation from PyTorch (2023a) in Section 19.4.2.1 first, before the extended implementation with `spotPython` is shown in Section 19.4.2.2.

19.4.2.1 Implementing a Configurable Neural Network With Ray Tune

We used the same hyperparameters that are implemented as configurable in the `PyTorch` tutorial. We specify the layer sizes, namely `l1` and `l2`, of the fully connected layers:

```
class Net(nn.Module):
    def __init__(self, l1=120, l2=84):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, l1)
        self.fc2 = nn.Linear(l1, l2)
        self.fc3 = nn.Linear(l2, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

The learning rate, i.e., `lr`, of the optimizer is made configurable, too:

```
optimizer = optim.SGD(net.parameters(), lr=config["lr"], momentum=0.9)
```

19.4.2.2 Implementing a Configurable Neural Network With spotPython

`spotPython` implements a class which is similar to the class described in the `PyTorch` tutorial. The class is called `Net_CIFAR10` and is implemented in the file `netcifar10.py`.

```
from torch import nn
import torch.nn.functional as F
import spotPython.torch.netcore as netcore


class Net_CIFAR10(netcore.Net_Core):
    def __init__(self, l1, l2, lr_mult, batch_size, epochs, k_folds, patience,
    optimizer, sgd_momentum):
        super(Net_CIFAR10, self).__init__(
            lr_mult=lr_mult,
            batch_size=batch_size,
            epochs=epochs,
            k_folds=k_folds,
            patience=patience,
            optimizer=optimizer,
            sgd_momentum=sgd_momentum,
        )
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, l1)
        self.fc2 = nn.Linear(l1, l2)
        self.fc3 = nn.Linear(l2, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

19.4.3 The `Net_Core` class

`Net_CIFAR10` inherits from the class `Net_Core` which is implemented in the file `netcore.py`. It implements the additional attributes that are common to all neural network models. The `Net_Core` class is implemented in the file `netcore.py`. It implements hyperparameters as attributes, that are not used by the `core_model`, e.g.:

- optimizer (`optimizer`),
- learning rate (`lr`),

- batch size (`batch_size`),
- epochs (`epochs`),
- k_folds (`k_folds`), and
- early stopping criterion "patience" (`patience`).

Users can add further attributes to the class. The class `Net_Core` is shown below.

```
from torch import nn


class Net_Core(nn.Module):
    def __init__(self, lr_mult, batch_size, epochs, k_folds, patience,
        optimizer, sgd_momentum):
        super(Net_Core, self).__init__()
        self.lr_mult = lr_mult
        self.batch_size = batch_size
        self.epochs = epochs
        self.k_folds = k_folds
        self.patience = patience
        self.optimizer = optimizer
        self.sgd_momentum = sgd_momentum
```

19.4.4 Comparison of the Approach Described in the PyTorch Tutorial With spotPython

Comparing the class `Net` from the `PyTorch` tutorial and the class `Net_CIFAR10` from `spotPython`, we see that the class `Net_CIFAR10` has additional attributes and does not inherit from `nn` directly. It adds an additional class, `Net_core`, that takes care of additional attributes that are common to all neural network models, e.g., the learning rate multiplier `lr_mult` or the batch size `batch_size`.

`spotPython`'s `core_model` implements an instance of the `Net_CIFAR10` class. In addition to the basic neural network model, the `core_model` can use these additional attributes. `spotPython` provides methods for handling these additional attributes to guarantee 100% compatibility with the `PyTorch` classes. The method `add_core_model_to_fun_control` adds the hyperparameters and additional attributes to the `fun_control` dictionary. The method is shown below.

```
from spotPython.torch.netcifar10 import Net_CIFAR10
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
core_model = Net_CIFAR10
```

```
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                fun_control=fun_control,
                                hyper_dict=TorchHyperDict,
                                filename=None)
```

19.5 The Search Space: Hyperparameters

In Section 19.5.1, we first describe how to configure the search space with `ray[tune]` (as shown in PyTorch (2023a)) and then how to configure the search space with `spotPython` in Section 19.5.2.

19.5.1 Configuring the Search Space With Ray Tune

Ray Tune's search space can be configured as follows (PyTorch 2023a):

```
config = {
    "l1": tune.sample_from(lambda _: 2**np.random.randint(2, 9)),
    "l2": tune.sample_from(lambda _: 2**np.random.randint(2, 9)),
    "lr": tune.loguniform(1e-4, 1e-1),
    "batch_size": tune.choice([2, 4, 8, 16])
}
```

The `tune.sample_from()` function enables the user to define sample methods to obtain hyperparameters. In this example, the `l1` and `l2` parameters should be powers of 2 between 4 and 256, so either 4, 8, 16, 32, 64, 128, or 256. The `lr` (learning rate) should be uniformly sampled between 0.0001 and 0.1. Lastly, the batch size is a choice between 2, 4, 8, and 16.

At each trial, `ray[tune]` will randomly sample a combination of parameters from these search spaces. It will then train a number of models in parallel and find the best performing one among these. `ray[tune]` uses the `ASHAScheduler` which will terminate bad performing trials early.

19.5.2 Configuring the Search Space With spotPython

19.5.2.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

`spotPython` uses `JSON` files for the specification of the hyperparameters. Users can specify their individual `JSON` files, or they can use the `JSON` files provided by `spotPython`. The `JSON` file for the `core_model` is called `torch_hyper_dict.json`.

In contrast to `ray[tune]`, `spotPython` can handle numerical, boolean, and categorical hyperparameters. They can be specified in the `JSON` file in a similar way as the numerical hyperparameters as shown below. Each entry in the `JSON` file represents one hyperparameter with the following structure: `type`, `default`, `transform`, `lower`, and `upper`.

```
"factor_hyperparameter": {
    "levels": ["A", "B", "C"],
    "type": "factor",
    "default": "B",
    "transform": "None",
    "core_model_parameter_type": "str",
    "lower": 0,
    "upper": 2},
```

The corresponding entries for the `Net_CIFAR10` class are shown below.

```
{"Net_CIFAR10":
    {
        "l1": {
            "type": "int",
            "default": 5,
            "transform": "transform_power_2_int",
            "lower": 2,
            "upper": 9},
        "l2": {
            "type": "int",
            "default": 5,
            "transform": "transform_power_2_int",
            "lower": 2,
            "upper": 9},
        "lr_mult": {
            "type": "float",
            "default": 1.0,
            "transform": "None",
            "lower": 0.1,
            "upper": 10},
        "batch_size": {
            "type": "int",
            "default": 4,
            "transform": "transform_power_2_int",
            "lower": 1,
            "upper": 4},
```

```
        "epochs": {
            "type": "int",
            "default": 3,
            "transform": "transform_power_2_int",
            "lower": 1,
            "upper": 4},
        "k_folds": {
            "type": "int",
            "default": 2,
            "transform": "None",
            "lower": 2,
            "upper": 3},
        "patience": {
            "type": "int",
            "default": 5,
            "transform": "None",
            "lower": 2,
            "upper": 10},
        "optimizer": {
            "levels": ["Adadelta",
                       "Adagrad",
                       "Adam",
                       "AdamW",
                       "SparseAdam",
                       "Adamax",
                       "ASGD",
                       "LBFGS",
                       "NAdam",
                       "RAdam",
                       "RMSprop",
                       "Rprop",
                       "SGD"],
            "type": "factor",
            "default": "SGD",
            "transform": "None",
            "class_name": "torch.optim",
            "core_model_parameter_type": "str",
            "lower": 0,
            "upper": 12},
        "sgd_momentum": {
            "type": "float",
```

```
            "default": 0.0,
            "transform": "None",
            "lower": 0.0,
            "upper": 1.0}
    }
}
```

19.5.3 Modifying the Hyperparameters

Ray tune (PyTorch 2023a) does not provide a way to change the specified hyperparameters without re-compilation. However, `spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions are described in the following.

19.5.3.1 Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

After specifying the model, the corresponding hyperparameters, their types and bounds are loaded from the `JSON` file `torch_hyper_dict.json`. After loading, the user can modify the hyperparameters, e.g., the bounds. `spotPython` provides a simple rule for de-activating hyperparameters: If the lower and the upper bound are set to identical values, the hyperparameter is de-activated. This is useful for the hyperparameter tuning, because it allows to specify a hyperparameter in the `JSON` file, but to de-activate it in the `fun_control` dictionary. This is done in the next step.

19.5.3.2 Modify Hyperparameters of Type numeric and integer (boolean)

Since the hyperparameter `k_folds` is not used in the `PyTorch` tutorial, it is de-activated here by setting the lower and upper bound to the same value. Note, `k_folds` is of type "integer".

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
fun_control = modify_hyper_parameter_bounds(fun_control,
    "batch_size", bounds=[1, 5])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "k_folds", bounds=[0, 0])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "patience", bounds=[3, 3])
```

19.5.3.3 Modify Hyperparameter of Type factor

In a similar manner as for the numerical hyperparameters, the categorical hyperparameters can be modified. New configurations can be chosen by adding or deleting levels. For example, the hyperparameter `optimizer` can be re-configured as follows:

In the following setting, two optimizers (`"SGD"` and `"Adam"`) will be compared during the `spotPython` hyperparameter tuning. The hyperparameter `optimizer` is active.

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control,
     "optimizer", ["SGD", "Adam"])
```

The hyperparameter `optimizer` can be de-activated by choosing only one value (level), here: `"SGD"`.

```
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["SGD"])
```

As discussed in Section 19.6, there are some issues with the LBFGS optimizer. Therefore, the usage of the LBFGS optimizer is not deactivated in `spotPython` by default. However, the LBFGS optimizer can be activated by adding it to the list of optimizers. `Rprop` was removed, because it does perform very poorly (as some pre-tests have shown). However, it can also be activated by adding it to the list of optimizers. Since `SparseAdam` does not support dense gradients, `Adam` was used instead. Therefore, there are 10 default optimizers:

```
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer",
    ["Adadelta", "Adagrad", "Adam", "AdamW", "Adamax", "ASGD",
    "NAdam", "RAdam", "RMSprop", "SGD"])
```

19.6 Optimizers

Table 20.1 shows some of the optimizers available in `PyTorch`:

a denotes (0.9,0.999), b (0.5,1.2), and c (1e-6, 50), respectively. R denotes `required, but unspecified`. "m" denotes `momentum`, "w_d" `weight_decay`, "d" `dampening`, "n" `nesterov`, "r" `rho`, "l_s" `learning rate for scaling delta`, "l_d" `lr_decay`, "b" `betas`, "l" `lambd`, "a" `alpha`, "m_d" for `momentum_decay`, "e" `etas`, and "s_s" for `step_sizes`.

Table 19.1: Optimizers available in PyTorch (selection). The default values are shown in the table.

Optimizer	lr	m	w_d	d	n	r	l_s	l_d	b	l	a	m_d	e	s_s
Adadelta	-	-	0.	-	-	0.9	1.	-	-	-	-	-	-	-
Adagrad	1e-2	-	0.	-	-	-	-	0.	-	-	-	-	-	-
Adam	1e-3	-	0.	-	-	-	-	-	a	-	-	-	-	-
AdamW	1e-3	-	1e-2	-	-	-	-	-	a	-	-	-	-	-
SparseAdam	1e-3	-	-	-	-	-	-	-	a	-	-	-	-	-
Adamax	2e-3	-	0.	-	-	-	-	-	a	-	-	-	-	-
ASGD	1e-2	.9	0.	-	F	-	-	-	-	1e-4	.75	-	-	-
LBFGS	1.	-	-	-	-	-	-	-	-	-	-	-	-	-
NAdam	2e-3	-	0.	-	-	-	-	-	a	-	-	0	-	-
RAdam	1e-3	-	0.	-	-	-	-	-	a	-	-	-	-	-
RMSprop	1e-2	0.	0.	-	-	-	-	-	a	-	-	-	-	-
Rprop	1e-2	-	-	-	-	-	-	-	-	-	b	c	-	-
SGD	R	0.	0.	0.	F	-	-	-	-	-	-	-	-	-

`spotPython` implements an `optimization` handler that maps the optimizer names to the corresponding `PyTorch` optimizers.

i A note on LBFGS

We recommend deactivating `PyTorch`'s LBFGS optimizer, because it does not perform very well. The `PyTorch` documentation, see https://pytorch.org/docs/stable/generated/torch.optim.LBFGS.html#torch.optim.LBFGS, states:

> This is a very memory intensive optimizer (it requires additional `param_bytes * (history_size + 1)` bytes). If it doesn't fit in memory try reducing the history size, or use a different algorithm.

Furthermore, the LBFGS optimizer is not compatible with the `PyTorch` tutorial. The reason is that the LBFGS optimizer requires the `closure` function, which is not implemented in the `PyTorch` tutorial. Therefore, the `LBFGS` optimizer is recommended here. Since there are ten optimizers in the portfolio, it is not recommended tuning the hyperparameters that effect one single optimizer only.

i A note on the learning rate

`spotPython` provides a multiplier for the default learning rates, `lr_mult`, because optimizers use different learning rates. Using a multiplier for the learning rates might enable

a simultaneous tuning of the learning rates for all optimizers. However, this is not recommended, because the learning rates are not comparable across optimizers. Therefore, we recommend fixing the learning rate for all optimizers if multiple optimizers are used. This can be done by setting the lower and upper bounds of the learning rate multiplier to the same value as shown below.
Thus, the learning rate, which affects the `SGD` optimizer, will be set to a fixed value. We choose the default value of `1e-3` for the learning rate, because it is used in other `PyTorch` examples (it is also the default value used by `spotPython` as defined in the `optimizer_handler()` method). We recommend tuning the learning rate later, when a reduced set of optimizers is fixed. Here, we will demonstrate how to select in a screening phase the optimizers that should be used for the hyperparameter tuning.

For the same reason, we will fix the `sgd_momentum` to `0.9`.

```
fun_control = modify_hyper_parameter_bounds(fun_control,
    "lr_mult", bounds=[1.0, 1.0])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "sgd_momentum", bounds=[0.9, 0.9])
```

19.7 Evaluation: Data Splitting

The evaluation procedure requires the specification of the way how the data is split into a train and a test set and the loss function (and a metric). As a default, `spotPython` provides a standard hold-out data split and cross validation.

19.7.1 Hold-out Data Split

If a hold-out data split is used, the data will be partitioned into a training, a validation, and a test data set. The split depends on the setting of the `eval` parameter. If `eval` is set to `train_hold_out`, one data set, usually the original training data set, is split into a new training and a validation data set. The training data set is used for training the model. The validation data set is used for the evaluation of the hyperparameter configuration and early stopping to prevent overfitting. In this case, the original test data set is not used.

`spotPython` returns the hyperparameters of the machine learning and deep learning models, e.g., number of layers, learning rate, or optimizer, but not the model weights. Therefore, after the SPOT run is finished, the corresponding model with the optimized architecture has to be trained again with the best hyperparameter configuration. The training is performed on the training data set. The test data set is used for the final evaluation of the model.

Summarizing, the following splits are performed in the hold-out setting:

1. Run `spotPython` with `eval` set to `train_hold_out` to determine the best hyperparameter configuration.
2. Train the model with the best hyperparameter configuration ("architecture") on the training data set: `train_tuned(model_spot, train, "model_spot.pt")`.
3. Test the model on the test data: `test_tuned(model_spot, test, "model_spot.pt")`

These steps will be exemplified in the following sections. In addition to this `hold-out` setting, `spotPython` provides another hold-out setting, where an explicit test data is specified by the user that will be used as the validation set. To choose this option, the `eval` parameter is set to `test_hold_out`. In this case, the training data set is used for the model training. Then, the explicitly defined test data set is used for the evaluation of the hyperparameter configuration (the validation).

19.7.2 Cross-Validation

The cross validation setting is used by setting the `eval` parameter to `train_cv` or `test_cv`. In both cases, the data set is split into k folds. The model is trained on $k-1$ folds and evaluated on the remaining fold. This is repeated k times, so that each fold is used exactly once for evaluation. The final evaluation is performed on the test data set. The cross validation setting is useful for small data sets, because it allows to use all data for training and evaluation. However, it is computationally expensive, because the model has to be trained k times.

Combinations of the above settings are possible, e.g., cross validation can be used for training and hold-out for evaluation or *vice versa*. Also, cross validation can be used for training and testing. Because cross validation is not used in the `PyTorch` tutorial (PyTorch 2023a), it is not considered further here.

19.7.3 Overview of the Evaluation Settings

19.7.3.1 Settings for the Hyperparameter Tuning

An overview of the training evaluations is shown in Table 20.2. `"train_cv"` and `"test_cv"` use `sklearn.model_selection.KFold()` internally. More details on the data splitting are provided in Section 24.14 (in the Appendix).

Table 19.2: Overview of the evaluation settings.

`eval`	`train`	`test`	function	comment
`"train_hold_out"`	✓		`train_one_epoch()`, `validate_one_epoch()` for early stopping	splits the `train` data set internally

eval	train	test	function	comment
"test_hold_out"	✓	✓	train_one_epoch(), validate_one_epoch() for early stopping	use the test data set for validate_one_epoch()
"train_cv"	✓		evaluate_cv(net, train)	CV using the train data set
"test_cv"		✓	evaluate_cv(net, test)	CV using the test data set . Identical to "train_cv", uses only test data.

19.7.3.2 Settings for the Final Evaluation of the Tuned Architecture

19.7.3.2.1 Training of the Tuned Architecture

`train_tuned(model, train)`: train the model with the best hyperparameter configuration (or simply the default) on the training data set. It splits the `train`data into new `train` and `validation` sets using `create_train_val_data_loaders()`, which calls `torch.utils.data.random_split()` internally. Currently, 60% of the data is used for training and 40% for validation. The `train` data is used for training the model with `train_hold_out()`. The `validation` data is used for early stopping using `validate_fold_or_hold_out()` on the `validation` data set.

19.7.3.2.2 Testing of the Tuned Architecture

`test_tuned(model, test)`: test the model on the test data set. No data splitting is performed. The (trained) model is evaluated using the `validate_fold_or_hold_out()` function. Note: During training, `shuffle` is set to `True`, whereas during testing, `shuffle` is set to `False`.

Section 24.14.1.4 describes the final evaluation of the tuned architecture.

19.8 Evaluation: Loss Functions and Metrics

The key `"loss_function"` specifies the loss function which is used during the optimization. There are several different loss functions under `PyTorch`'s `nn` package. For example, a simple loss is `MSELoss`, which computes the mean-squared error between the output and the target. In this tutorial we will use `CrossEntropyLoss`, because it is also used in the `PyTorch` tutorial.

```
from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({"loss_function": loss_function})
```

In addition to the loss functions, `spotPython` provides access to a large number of metrics.

- The key `"metric_sklearn"` is used for metrics that follow the `scikit-learn` conventions.
- The key `"river_metric"` is used for the river based evaluation (Montiel et al. 2021) via `eval_oml_iter_progressive`, and
- the key `"metric_torch"` is used for the metrics from `TorchMetrics`.

`TorchMetrics` is a collection of more than 90 PyTorch metrics, see https://torchmetrics.readthedocs.io/en/latest/. Because the `PyTorch` tutorial uses the accuracy as metric, we use the same metric here. Currently, accuracy is computed in the tutorial's example code. We will use `TorchMetrics` instead, because it offers more flexibilty, e.g., it can be used for regression and classification. Furthermore, `TorchMetrics` offers the following advantages:

```
* A standardized interface to increase reproducibility
* Reduces Boilerplate
* Distributed-training compatible
* Rigorously tested
* Automatic accumulation over batches
* Automatic synchronization between multiple devices
```

Therefore, we set

```
import torchmetrics
metric_torch = torchmetrics.Accuracy(task="multiclass", num_classes=10).to(fun_control["de
fun_control.update({"metric_torch": metric_torch})
```

```
loss_function = CrossEntropyLoss()
weights = 1.0
shuffle = True
eval = "train_hold_out"
show_batch_interval = 100_000
path="torch_model.pt"

fun_control.update({
                "data_dir": None,
                "checkpoint_dir": None,
                "horizon": None,
```

```
                "oml_grace_period": None,
                "weights": weights,
                "step": None,
                "log_level": 50,
                "weight_coeff": None,
                "metric_river": None,
                "metric_sklearn": None,
                "loss_function": loss_function,
                "shuffle": shuffle,
                "eval": eval,
                "show_batch_interval": show_batch_interval,
                "path": path,
                })
```

19.9 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`.

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` generates a design table as follows:

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

```
| name          | type   | default   |   lower |   upper | transform             |
|---------------|--------|-----------|---------|---------|-----------------------|
| l1            | int    | 5         |     2   |     9   | transform_power_2_int |
| l2            | int    | 5         |     2   |     9   | transform_power_2_int |
| lr_mult       | float  | 1.0       |     1   |     1   | None                  |
| batch_size    | int    | 4         |     1   |     5   | transform_power_2_int |
| epochs        | int    | 3         |     3   |     4   | transform_power_2_int |
| k_folds       | int    | 1         |     0   |     0   | None                  |
| patience      | int    | 5         |     3   |     3   | None                  |
| optimizer     | factor | SGD       |     0   |     9   | None                  |
| sgd_momentum | float  | 0.0       |     0.9 |     0.9 | None                  |
```

This allows to check if all information is available and if the information is correct. Table 20.3 shows the experimental design for the hyperparameter tuning. The table shows the hyperparameters, their types, default values, lower and upper bounds, and the transformation function.

The transformation function is used to transform the hyperparameter values from the unit hypercube to the original domain. The transformation function is applied to the hyperparameter values before the evaluation of the objective function. Hyperparameter transformations are shown in the column "transform", e.g., the `l1` default is `5`, which results in the value $2^5 = 32$ for the network, because the transformation `transform_power_2_int` was selected in the `JSON` file. The default value of the `batch_size` is set to `4`, which results in a batch size of $2^4 = 16$.

Table 19.3: Experimental design for the hyperparameter tuning.

name	type	default	lower	upper	transform
l1	int	5	2	9	transform_power_2_int
l2	int	5	2	9	transform_power_2_int
lr_mult	float	1.0	1	1	None
batch_size	int	4	1	5	transform_power_2_int
epochs	int	3	3	4	transform_power_2_int
k_folds	int	1	0	0	None
patience	int	5	3	3	None
optimizer	factor	SGD	0	9	None
sgd_momentum	float	0.0	0.9	0.9	None

19.10 The Objective Function `fun_torch`

The objective function `fun_torch` is selected next. It implements an interface from `PyTorch`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch
```

19.11 Using Default Hyperparameters or Results from Previous Runs

We add the default setting to the initial design:

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=TorchHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
```

19.12 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function. Here, we will run the tuner for approximately 30 minutes (`max_time`). Note: the initial design is always evaluated in the `spotPython` run. As a consequence, the run may take longer than specified by `max_time`, because the evaluation time of initial design (here: `init_size`, 10 points) is performed independently of `max_time`.

```
from spotPython.spot import spot
from math import inf
import numpy as np
spot_tuner = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = inf,
                    fun_repeats = 1,
                    max_time = MAX_TIME,
                    noise = False,
                    tolerance_x = np.sqrt(np.spacing(1)),
                    var_type = var_type,
                    var_name = var_name,
                    infill_criterion = "y",
                    n_points = 1,
                    seed=123,
                    log_level = 50,
                    show_models= False,
                    show_progress= True,
                    fun_control = fun_control,
                    design_control={"init_size": INIT_SIZE,
                                    "repeats": 1},
                    surrogate_control={"noise": True,
                                       "cod_type": "norm",
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": len(var_name),
                                       "model_fun_evals": 10_000,
                                       "log_level": 50
                                       })
spot_tuner.run(X_start=X_start)
```

```
config: {'l1': 128, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 32, 'epochs': 16, 'k_folds': 0, '
```

```
Epoch: 1

Loss on hold-out set: 1.6038289865493773
Accuracy on hold-out set: 0.4022
MulticlassAccuracy value on hold-out data: 0.40220001339912415
Epoch: 2

Loss on hold-out set: 1.498838716506958
Accuracy on hold-out set: 0.4431
MulticlassAccuracy value on hold-out data: 0.4431000053882599
Epoch: 3

Loss on hold-out set: 1.3894483339309693
Accuracy on hold-out set: 0.49365
MulticlassAccuracy value on hold-out data: 0.49364998936653137
Epoch: 4

Loss on hold-out set: 1.3450864872932433
Accuracy on hold-out set: 0.51415
MulticlassAccuracy value on hold-out data: 0.5141500234603882
Epoch: 5

Loss on hold-out set: 1.3017903809547424
Accuracy on hold-out set: 0.5305
MulticlassAccuracy value on hold-out data: 0.5304999947547913
Epoch: 6

Loss on hold-out set: 1.2768837705612182
Accuracy on hold-out set: 0.54365
MulticlassAccuracy value on hold-out data: 0.5436499714851379
Epoch: 7

Loss on hold-out set: 1.2734394570350647
Accuracy on hold-out set: 0.5444
MulticlassAccuracy value on hold-out data: 0.5443999767303467
Epoch: 8

Loss on hold-out set: 1.2504254244804383
Accuracy on hold-out set: 0.55855
MulticlassAccuracy value on hold-out data: 0.5585500001907349
Epoch: 9
```

```
Loss on hold-out set: 1.220351642036438
Accuracy on hold-out set: 0.56715
MulticlassAccuracy value on hold-out data: 0.5671499967575073
Epoch: 10

Loss on hold-out set: 1.1836971961975098
Accuracy on hold-out set: 0.5852
MulticlassAccuracy value on hold-out data: 0.5852000117301941
Epoch: 11

Loss on hold-out set: 1.1999144332885743
Accuracy on hold-out set: 0.5816
MulticlassAccuracy value on hold-out data: 0.58160001039505
Epoch: 12

Loss on hold-out set: 1.204782396697998
Accuracy on hold-out set: 0.5824
MulticlassAccuracy value on hold-out data: 0.5824000239372253
Epoch: 13

Loss on hold-out set: 1.2046439470291137
Accuracy on hold-out set: 0.5873
MulticlassAccuracy value on hold-out data: 0.5873000025749207
Early stopping at epoch 12
Returned to Spot: Validation loss: 1.2046439470291137
------------------------------------------------

config: {'l1': 16, 'l2': 16, 'lr_mult': 1.0, 'batch_size': 8, 'epochs': 8, 'k_folds': 0, 'pat
Epoch: 1

Loss on hold-out set: 1.483202227115631
Accuracy on hold-out set: 0.45675
MulticlassAccuracy value on hold-out data: 0.4567500054836273
Epoch: 2

Loss on hold-out set: 1.3854034468054772
Accuracy on hold-out set: 0.4962
MulticlassAccuracy value on hold-out data: 0.49619999527931213
Epoch: 3
```

```
Loss on hold-out set: 1.3463982392311096
Accuracy on hold-out set: 0.5171
MulticlassAccuracy value on hold-out data: 0.5170999765396118
Epoch: 4

Loss on hold-out set: 1.3535124581933022
Accuracy on hold-out set: 0.52575
MulticlassAccuracy value on hold-out data: 0.5257499814033508
Epoch: 5

Loss on hold-out set: 1.2804110110759734
Accuracy on hold-out set: 0.538
MulticlassAccuracy value on hold-out data: 0.5379999876022339
Epoch: 6

Loss on hold-out set: 1.2948389030873775
Accuracy on hold-out set: 0.5443
MulticlassAccuracy value on hold-out data: 0.5443000197410583
Epoch: 7

Loss on hold-out set: 1.3230243331670761
Accuracy on hold-out set: 0.53665
MulticlassAccuracy value on hold-out data: 0.5366500020027161
Epoch: 8

Loss on hold-out set: 1.2906903485715389
Accuracy on hold-out set: 0.5474
MulticlassAccuracy value on hold-out data: 0.5473999977111816
Early stopping at epoch 7
Returned to Spot: Validation loss: 1.2906903485715389
--------------------------------------------

config: {'l1': 256, 'l2': 128, 'lr_mult': 1.0, 'batch_size': 2, 'epochs': 16, 'k_folds': 0,
Epoch: 1

Loss on hold-out set: 2.306265900039673
Accuracy on hold-out set: 0.1002
MulticlassAccuracy value on hold-out data: 0.10019999742507935
Epoch: 2
```

```
Loss on hold-out set: 2.309000279545784
Accuracy on hold-out set: 0.09955
MulticlassAccuracy value on hold-out data: 0.09955000132322311
Epoch: 3

Loss on hold-out set: 2.310344591188431
Accuracy on hold-out set: 0.1012
MulticlassAccuracy value on hold-out data: 0.10119999945163727
Epoch: 4

Loss on hold-out set: 2.3086429764032363
Accuracy on hold-out set: 0.09755
MulticlassAccuracy value on hold-out data: 0.09754999727010727
Early stopping at epoch 3
Returned to Spot: Validation loss: 2.3086429764032363
---------------------------------------------

config: {'l1': 8, 'l2': 32, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pati
Epoch: 1

Loss on hold-out set: 1.6333317642867564
Accuracy on hold-out set: 0.39355
MulticlassAccuracy value on hold-out data: 0.39355000853538513
Epoch: 2

Loss on hold-out set: 1.4752094827115536
Accuracy on hold-out set: 0.4596
MulticlassAccuracy value on hold-out data: 0.4596000015735626
Epoch: 3

Loss on hold-out set: 1.45965206592381
Accuracy on hold-out set: 0.47145
MulticlassAccuracy value on hold-out data: 0.47145000100135803
Epoch: 4

Loss on hold-out set: 1.3052383463710546
Accuracy on hold-out set: 0.53305
MulticlassAccuracy value on hold-out data: 0.533050000667572
Epoch: 5
```

```
Loss on hold-out set: 1.2835813677877188
Accuracy on hold-out set: 0.5442
MulticlassAccuracy value on hold-out data: 0.5442000031471252
Epoch: 6

Loss on hold-out set: 1.2875813015773891
Accuracy on hold-out set: 0.546
MulticlassAccuracy value on hold-out data: 0.5460000038146973
Epoch: 7

Loss on hold-out set: 1.249462442008406
Accuracy on hold-out set: 0.5676
MulticlassAccuracy value on hold-out data: 0.5676000118255615
Epoch: 8

Loss on hold-out set: 1.262561587954685
Accuracy on hold-out set: 0.56785
MulticlassAccuracy value on hold-out data: 0.5678499937057495
Returned to Spot: Validation loss: 1.262561587954685
---------------------------------------------

config: {'l1': 64, 'l2': 512, 'lr_mult': 1.0, 'batch_size': 16, 'epochs': 16, 'k_folds': 0,
Epoch: 1

Loss on hold-out set: 1.449402775812149
Accuracy on hold-out set: 0.47055
MulticlassAccuracy value on hold-out data: 0.470550000667572
Epoch: 2

Loss on hold-out set: 1.379134748840332
Accuracy on hold-out set: 0.50145
MulticlassAccuracy value on hold-out data: 0.5014500021934509
Epoch: 3

Loss on hold-out set: 1.3415317151069641
Accuracy on hold-out set: 0.5189
MulticlassAccuracy value on hold-out data: 0.5188999772071838
Epoch: 4
```

```
Loss on hold-out set: 1.300934012556076
Accuracy on hold-out set: 0.53545
MulticlassAccuracy value on hold-out data: 0.5354499816894531
Epoch: 5

Loss on hold-out set: 1.2737155021190643
Accuracy on hold-out set: 0.5482
MulticlassAccuracy value on hold-out data: 0.5482000112533569
Epoch: 6

Loss on hold-out set: 1.256284383916855
Accuracy on hold-out set: 0.5537
MulticlassAccuracy value on hold-out data: 0.5536999702453613
Epoch: 7

Loss on hold-out set: 1.2514449563503265
Accuracy on hold-out set: 0.5597
MulticlassAccuracy value on hold-out data: 0.5597000122070312
Epoch: 8

Loss on hold-out set: 1.2354318824291228
Accuracy on hold-out set: 0.5673
MulticlassAccuracy value on hold-out data: 0.567300021648407
Epoch: 9

Loss on hold-out set: 1.2261272450447083
Accuracy on hold-out set: 0.5686
MulticlassAccuracy value on hold-out data: 0.5685999989509583
Epoch: 10

Loss on hold-out set: 1.219343085861206
Accuracy on hold-out set: 0.5722
MulticlassAccuracy value on hold-out data: 0.5722000002861023
Epoch: 11

Loss on hold-out set: 1.2112840285778046
Accuracy on hold-out set: 0.5756
MulticlassAccuracy value on hold-out data: 0.5756000280380249
Epoch: 12
```

```
Loss on hold-out set: 1.2114751480579375
Accuracy on hold-out set: 0.57375
MulticlassAccuracy value on hold-out data: 0.5737500190734863
Epoch: 13

Loss on hold-out set: 1.196377786564827
Accuracy on hold-out set: 0.583
MulticlassAccuracy value on hold-out data: 0.5830000042915344
Epoch: 14

Loss on hold-out set: 1.1923975789546966
Accuracy on hold-out set: 0.58265
MulticlassAccuracy value on hold-out data: 0.5826500058174133
Epoch: 15

Loss on hold-out set: 1.1893289243221283
Accuracy on hold-out set: 0.58795
MulticlassAccuracy value on hold-out data: 0.5879499912261963
Epoch: 16

Loss on hold-out set: 1.189708170723915
Accuracy on hold-out set: 0.5873
MulticlassAccuracy value on hold-out data: 0.5873000025749207
Returned to Spot: Validation loss: 1.189708170723915
---------------------------------------------


config: {'l1': 64, 'l2': 256, 'lr_mult': 1.0, 'batch_size': 16, 'epochs': 16, 'k_folds': 0,
Epoch: 1

Loss on hold-out set: 1.5391039041519166
Accuracy on hold-out set: 0.43405
MulticlassAccuracy value on hold-out data: 0.4340499937534332
Epoch: 2

Loss on hold-out set: 1.4534715387821198
Accuracy on hold-out set: 0.46835
MulticlassAccuracy value on hold-out data: 0.46834999322891235
Epoch: 3
```

```
Loss on hold-out set: 1.4147080455303191
Accuracy on hold-out set: 0.4828
MulticlassAccuracy value on hold-out data: 0.4828000068664551
Epoch: 4

Loss on hold-out set: 1.386229646921158
Accuracy on hold-out set: 0.49735
MulticlassAccuracy value on hold-out data: 0.4973500072956085
Epoch: 5

Loss on hold-out set: 1.384547208070755
Accuracy on hold-out set: 0.5003
MulticlassAccuracy value on hold-out data: 0.5002999901771545
Epoch: 6

Loss on hold-out set: 1.3526852039813995
Accuracy on hold-out set: 0.51115
MulticlassAccuracy value on hold-out data: 0.5111500024795532
Epoch: 7

Loss on hold-out set: 1.3264264718055725
Accuracy on hold-out set: 0.5192
MulticlassAccuracy value on hold-out data: 0.5192000269889832
Epoch: 8

Loss on hold-out set: 1.3130590114593506
Accuracy on hold-out set: 0.5291
MulticlassAccuracy value on hold-out data: 0.5291000008583069
Epoch: 9

Loss on hold-out set: 1.2845707306385041
Accuracy on hold-out set: 0.5416
MulticlassAccuracy value on hold-out data: 0.5415999889373779
Epoch: 10

Loss on hold-out set: 1.2755087090492248
Accuracy on hold-out set: 0.54685
MulticlassAccuracy value on hold-out data: 0.5468500256538391
Epoch: 11
```

```
Loss on hold-out set: 1.2637518959999083
Accuracy on hold-out set: 0.5506
MulticlassAccuracy value on hold-out data: 0.550599992275238
Epoch: 12

Loss on hold-out set: 1.2682413181304932
Accuracy on hold-out set: 0.55195
MulticlassAccuracy value on hold-out data: 0.5519499778747559
Epoch: 13

Loss on hold-out set: 1.2595609003067016
Accuracy on hold-out set: 0.55495
MulticlassAccuracy value on hold-out data: 0.5549499988555908
Epoch: 14

Loss on hold-out set: 1.2419652947664261
Accuracy on hold-out set: 0.55995
MulticlassAccuracy value on hold-out data: 0.5599499940872192
Epoch: 15

Loss on hold-out set: 1.2320137884140014
Accuracy on hold-out set: 0.56515
MulticlassAccuracy value on hold-out data: 0.5651500225067139
Epoch: 16

Loss on hold-out set: 1.2336790138959886
Accuracy on hold-out set: 0.5673
MulticlassAccuracy value on hold-out data: 0.567300021648407
Returned to Spot: Validation loss: 1.2336790138959886
-----------------------------------------------
spotPython tuning: 1.189708170723915 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2c6ed2da0>
```

During the run, the following output is shown:

```
config: {'l1': 128, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 32,
    'epochs': 16, 'k_folds': 0, 'patience': 3,
    'optimizer': 'AdamW', 'sgd_momentum': 0.9}
Epoch: 1
```

```
Loss on hold-out set: 1.5143253986358642
Accuracy on hold-out set: 0.4447
MulticlassAccuracy value on hold-out data: 0.4447000026702881
Epoch: 2
...
Epoch: 15
Loss on hold-out set: 1.2061678514480592
Accuracy on hold-out set: 0.59505
MulticlassAccuracy value on hold-out data: 0.5950499773025513
Early stopping at epoch 14
Returned to Spot: Validation loss: 1.2061678514480592
------------------------------------------------
```

19.13 Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard.

19.13.1 Tensorboard: Start Tensorboard

Start TensorBoard through the command line to visualize data you logged. Specify the root log directory as used in `fun_control = fun_control_init(task="regression", tensorboard_path="runs/24_spot_torch_regression")` as the `tensorboard_path`. The argument logdir points to directory where TensorBoard will look to find event files that it can display. TensorBoard will recursively walk the directory structure rooted at logdir, looking for *.tfevents.* files.

```
tensorboard --logdir=runs
```

Go to the URL it provides or to http://localhost:6006/. The following figures show some screenshots of Tensorboard.

19.13.2 Saving the State of the Notebook

The state of the notebook can be saved and reloaded as follows:

```
import pickle
SAVE = False
LOAD = False
```

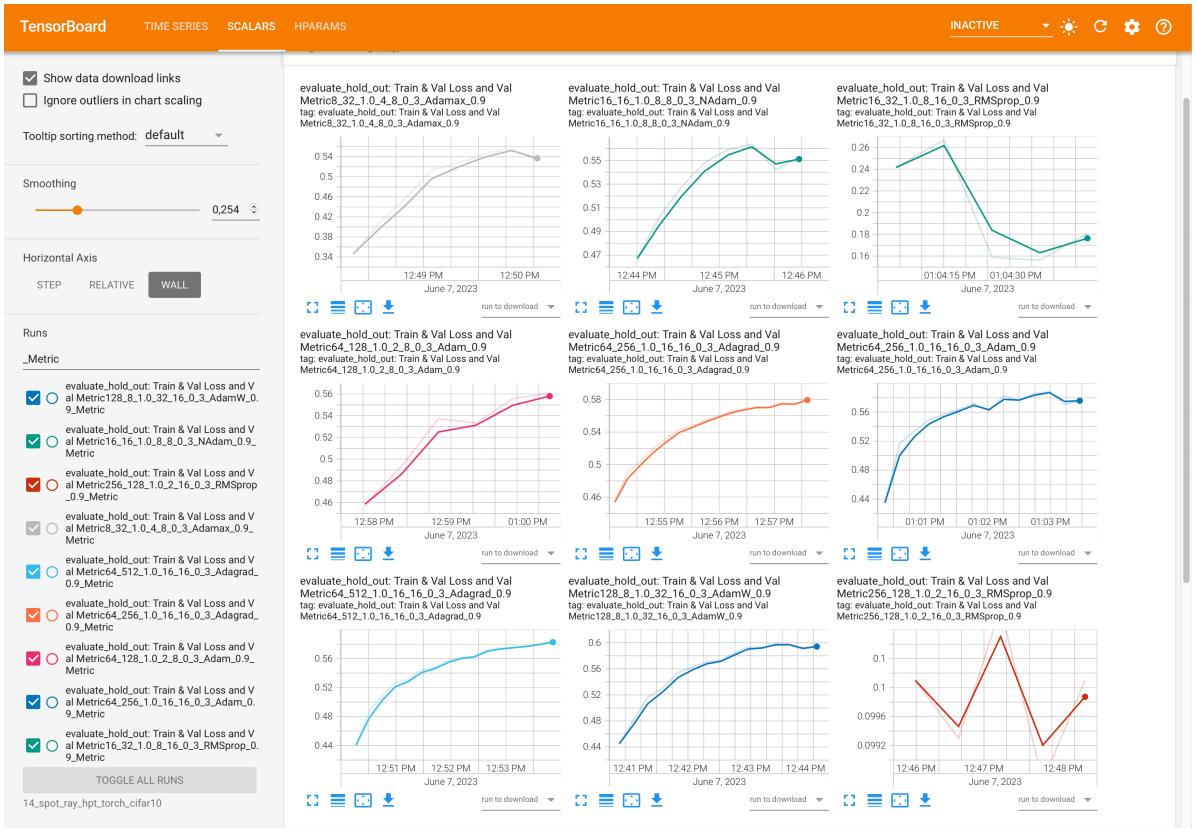


Figure 19.1: Tensorboard

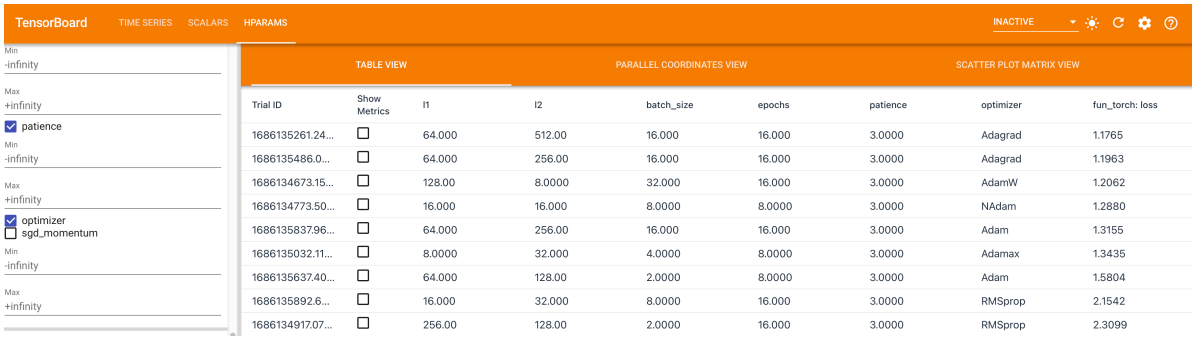


Figure 19.2: Tensorboard

```
if SAVE:
    result_file_name = "res_" + experiment_name + ".pkl"
    with open(result_file_name, 'wb') as f:
        pickle.dump(spot_tuner, f)

if LOAD:
    result_file_name = "add_the_name_of_the_result_file_here.pkl"
    with open(result_file_name, 'rb') as f:
        spot_tuner =  pickle.load(f)
```

19.14 Results

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from Figure 20.4.

```
spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")
```

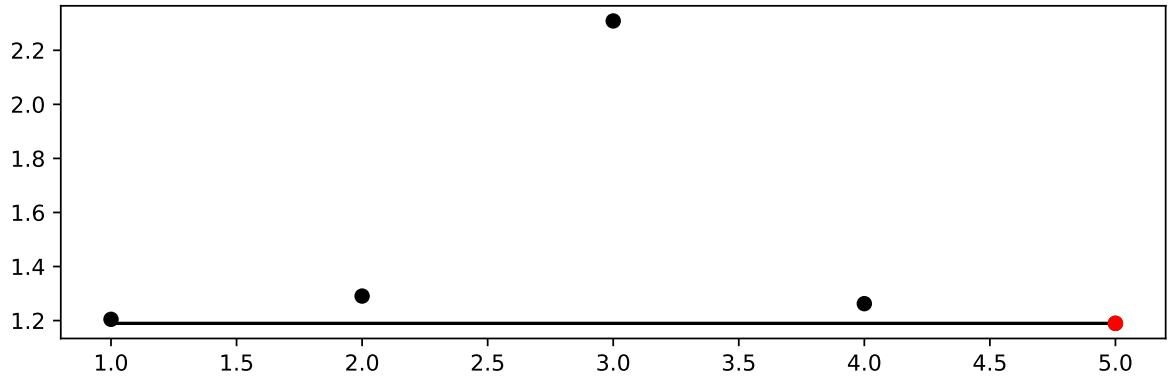


Figure 19.3: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

Figure 20.4 shows a typical behaviour that can be observed in many hyperparameter studies (Bartz et al. 2022): the largest improvement is obtained during the evaluation of the initial design. The surrogate model based optimization-optimization with the surrogate refines the results. Figure 20.4 also illustrates one major difference between `ray[tune]` as used in PyTorch (2023a) and `spotPython`: the `ray[tune]` uses a random search and will generate results similar to the *black* dots, whereas `spotPython` uses a surrogate model based optimization and presents results represented by *red* dots in Figure 20.4. The surrogate model based optimization is

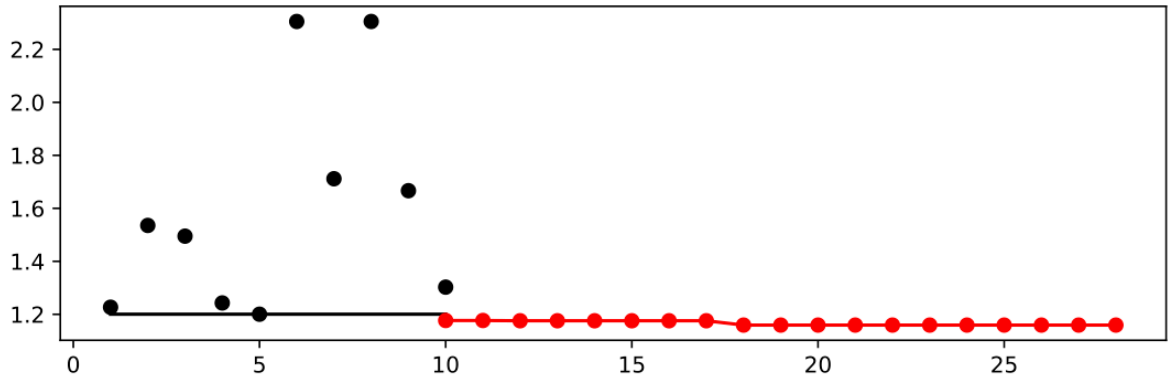


Figure 19.4: Progress plot. `Black` dots denote results from the initial design. `Red` dots illustrate the improvement found by the surrogate model based optimization (surrogate model based optimization).

considered to be more efficient than a random search, because the surrogate model guides the search towards promising regions in the hyperparameter space.

In addition to the improved ("optimized") hyperparameter values, `spotPython` allows a statistical analysis, e.g., a sensitivity analysis, of the results. We can print the results of the hyperparameter tuning, see Table 20.4. The table shows the hyperparameters, their types, default values, lower and upper bounds, and the transformation function. The column "tuned" shows the tuned values. The column "importance" shows the importance of the hyperparameters. The column "stars" shows the importance of the hyperparameters in stars. The importance is computed by the SPOT software.

```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

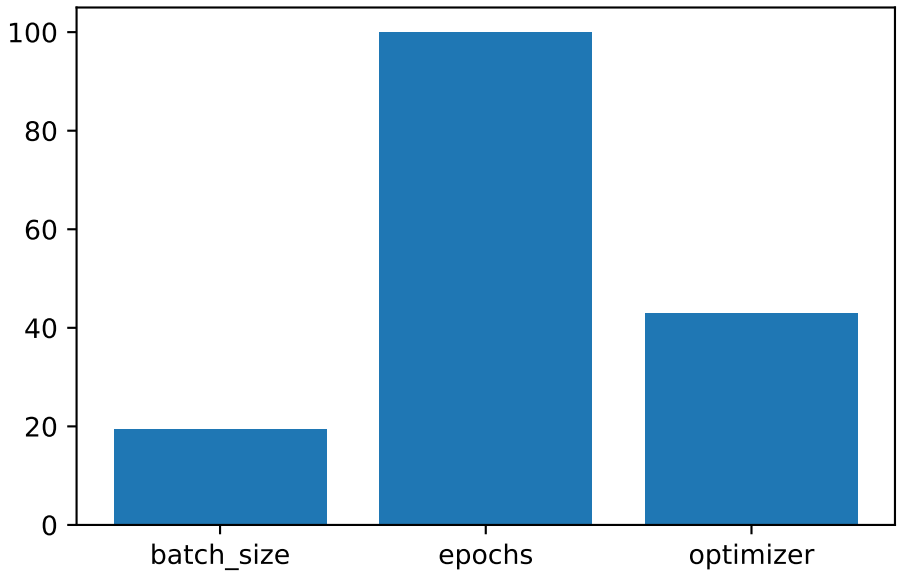
name	type	default	lower	upper	tuned	transform	
l1	int	5	2.0	9.0	6.0	transform_power_2_int	
l2	int	5	2.0	9.0	9.0	transform_power_2_int	
lr_mult	float	1.0	1.0	1.0	1.0	None	
batch_size	int	4	1.0	5.0	4.0	transform_power_2_int	
epochs	int	3	3.0	4.0	4.0	transform_power_2_int	
k_folds	int	1	0.0	0.0	0.0	None	
patience	int	5	3.0	3.0	3.0	None	
optimizer	factor	SGD	0.0	9.0	1.0	None	
sgd_momentum	float	0.0	0.9	0.9	0.9	None	

Table 19.4: Results of the hyperparameter tuning.

name	type	default	lower	upper	tuned	transform	importance	stars
l1	int	5	2.0	9.0	7.0	pow_2_int	100.00	***
l2	int	5	2.0	9.0	3.0	pow_2_int	96.29	***
lr_mult	float	1.0	0.1	10.0	0.1	None	0.00	
batchsize	int	4	1.0	5.0	4.0	pow_2_int	0.00	
epochs	int	3	3.0	4.0	4.0	pow_2_int	4.18	*
k_folds	int	2	0.0	0.0	0.0	None	0.00	
patience	int	5	3.0	3.0	3.0	None	0.00	
optimizer	factor	SGD	0.0	9.0	3.0	None	0.16	.

To visualize the most important hyperparameters, `spotPython` provides the function `plot_importance`. The following code generates the importance plot from Figure 20.5.

```
spot_tuner.plot_importance(threshold=0.025,
    filename="./figures/" + experiment_name+"_importance.png")
```



19.14.1 Get SPOT Results

The architecture of the `spotPython` model can be obtained by the following code:

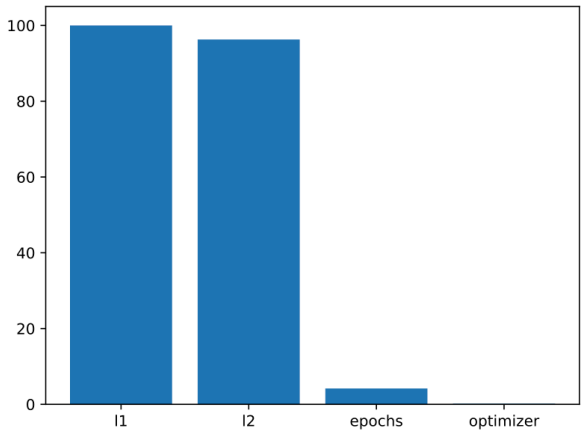


Figure 19.5: Variable importance

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_CIFAR10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=10, bias=True)
)
```

First, the numerical representation of the hyperparameters are obtained, i.e., the numpy array `X` is generated. This array is then used to generate the model `model_spot` by the function `get_one_core_model_from_X`. The model `model_spot` has the following architecture:

```
Net_CIFAR10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=10, bias=True)
```

```
)
```

19.14.2 Get Default Hyperparameters

In a similar manner as in Section 19.14.1, the default hyperparameters can be obtained.

```
# fun_control was modified, we generate a new one with the original
# default hyperparameters
from spotPython.hyperparameters.values import get_one_core_model_from_X
fc = fun_control
fc.update({"core_model_hyper_dict":
    hyper_dict[fun_control["core_model"].__name__]})
model_default = get_one_core_model_from_X(X_start, fun_control=fc)
```

The corresponding default model has the following architecture:

```
Net_CIFAR10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=10, bias=True)
)
```

19.14.3 Evaluation of the Default Architecture

The method `train_tuned` takes a model architecture without trained weights and trains this model with the train data. The train data is split into train and validation data. The validation data is used for early stopping. The trained model weights are saved as a dictionary.

This evaluation is similar to the final evaluation in PyTorch (2023a).

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
    )
train_tuned(net=model_default, train_dataset=train, shuffle=True,
        loss_function=fun_control["loss_function"],
```

```
        metric=fun_control["metric_torch"],
        device = fun_control["device"], show_batch_interval=1_000_000,
        path=None,
        task=fun_control["task"],)

test_tuned(net=model_default, test_dataset=test,
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        shuffle=False,
        device = fun_control["device"],
        task=fun_control["task"],)
```

```
Epoch: 1

Loss on hold-out set: 2.3047729780197144
Accuracy on hold-out set: 0.0996
MulticlassAccuracy value on hold-out data: 0.09960000216960907
Epoch: 2

Loss on hold-out set: 2.3011977392196656
Accuracy on hold-out set: 0.11595
MulticlassAccuracy value on hold-out data: 0.11595000326633453
Epoch: 3

Loss on hold-out set: 2.295659527397156
Accuracy on hold-out set: 0.12105
MulticlassAccuracy value on hold-out data: 0.12105000019073486
Epoch: 4

Loss on hold-out set: 2.280763189315796
Accuracy on hold-out set: 0.1454
MulticlassAccuracy value on hold-out data: 0.1454000025987625
Epoch: 5

Loss on hold-out set: 2.2387226348876954
Accuracy on hold-out set: 0.1774
MulticlassAccuracy value on hold-out data: 0.17739999294281006
Epoch: 6
```

```
Loss on hold-out set: 2.180402205657959
Accuracy on hold-out set: 0.19825
MulticlassAccuracy value on hold-out data: 0.19824999570846558
Epoch: 7

Loss on hold-out set: 2.1250165114402773
Accuracy on hold-out set: 0.21235
MulticlassAccuracy value on hold-out data: 0.212349995970726
Epoch: 8

Loss on hold-out set: 2.0814302061080934
Accuracy on hold-out set: 0.23505
MulticlassAccuracy value on hold-out data: 0.2350499927997589
Returned to Spot: Validation loss: 2.0814302061080934
------------------------------------------------

Loss on hold-out set: 2.0777481615066526
Accuracy on hold-out set: 0.2385
MulticlassAccuracy value on hold-out data: 0.23849999904632568
Final evaluation: Validation loss: 2.0777481615066526
Final evaluation: Validation metric: 0.23849999904632568
------------------------------------------------

(2.0777481615066526, nan, tensor(0.2385, device='mps:0'))
```

19.14.4 Evaluation of the Tuned Architecture

The following code trains the model `model_spot`. If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be saved to this file.

```
train_tuned(net=model_spot, train_dataset=train,
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        shuffle=True,
        device = fun_control["device"],
        path=None,
        task=fun_control["task"],)
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
```

```
            device = fun_control["device"],
            task=fun_control["task"],)
```

```
Epoch: 1

Loss on hold-out set: 1.496268241214752
Accuracy on hold-out set: 0.4528
MulticlassAccuracy value on hold-out data: 0.4528000056743622
Epoch: 2

Loss on hold-out set: 1.4125423771381378
Accuracy on hold-out set: 0.4894
MulticlassAccuracy value on hold-out data: 0.4893999993801117
Epoch: 3

Loss on hold-out set: 1.3443015516757966
Accuracy on hold-out set: 0.5158
MulticlassAccuracy value on hold-out data: 0.5157999992370605
Epoch: 4

Loss on hold-out set: 1.3306156158447267
Accuracy on hold-out set: 0.5227
MulticlassAccuracy value on hold-out data: 0.5227000117301941
Epoch: 5

Loss on hold-out set: 1.2999932603597641
Accuracy on hold-out set: 0.5363
MulticlassAccuracy value on hold-out data: 0.536300003528595
Epoch: 6

Loss on hold-out set: 1.2721322926044465
Accuracy on hold-out set: 0.5479
MulticlassAccuracy value on hold-out data: 0.5479000210762024
Epoch: 7

Loss on hold-out set: 1.261376016139984
Accuracy on hold-out set: 0.5502
MulticlassAccuracy value on hold-out data: 0.5501999855041504
Epoch: 8
```

```
Loss on hold-out set: 1.2475526552677154
Accuracy on hold-out set: 0.55845
MulticlassAccuracy value on hold-out data: 0.5584499835968018
Epoch: 9

Loss on hold-out set: 1.2304288831233978
Accuracy on hold-out set: 0.56205
MulticlassAccuracy value on hold-out data: 0.5620499849319458
Epoch: 10

Loss on hold-out set: 1.2232957232952117
Accuracy on hold-out set: 0.5648
MulticlassAccuracy value on hold-out data: 0.5648000240325928
Epoch: 11

Loss on hold-out set: 1.2127183192253113
Accuracy on hold-out set: 0.5697
MulticlassAccuracy value on hold-out data: 0.5697000026702881
Epoch: 12

Loss on hold-out set: 1.2200033745765686
Accuracy on hold-out set: 0.56995
MulticlassAccuracy value on hold-out data: 0.5699499845504761
Epoch: 13

Loss on hold-out set: 1.1972633383274078
Accuracy on hold-out set: 0.57715
MulticlassAccuracy value on hold-out data: 0.5771499872207642
Epoch: 14

Loss on hold-out set: 1.1911635332107544
Accuracy on hold-out set: 0.57825
MulticlassAccuracy value on hold-out data: 0.578249990940094
Epoch: 15

Loss on hold-out set: 1.2071927551269532
Accuracy on hold-out set: 0.575
MulticlassAccuracy value on hold-out data: 0.574999988079071
Epoch: 16
```

```
Loss on hold-out set: 1.1877539900302887
Accuracy on hold-out set: 0.58015
MulticlassAccuracy value on hold-out data: 0.5801500082015991
Returned to Spot: Validation loss: 1.1877539900302887
------------------------------------------------


Loss on hold-out set: 1.185979939031601
Accuracy on hold-out set: 0.5799
MulticlassAccuracy value on hold-out data: 0.5799000263214111
Final evaluation: Validation loss: 1.185979939031601
Final evaluation: Validation metric: 0.5799000263214111
------------------------------------------------


(1.185979939031601, nan, tensor(0.5799, device='mps:0'))
```

These runs will generate output similar to the following:

```
Loss on hold-out set: 1.2267619131326675
Accuracy on hold-out set: 0.58955
Early stopping at epoch 13
```

19.14.5 Comparison with Default Hyperparameters and Ray Tune

Table 20.5 shows the loss and accuracy of the default model, the model with the hyperparameters from SPOT, and the model with the hyperparameters from `ray[tune]`. The table shows a comparison of the loss and accuracy of the default model, the model with the hyperparameters from SPOT, and the model with the hyperparameters from `ray[tune]`. `ray[tune]` only shows the validation loss, because training loss is not reported by `ray[tune]`

Table 19.5: Comparison.

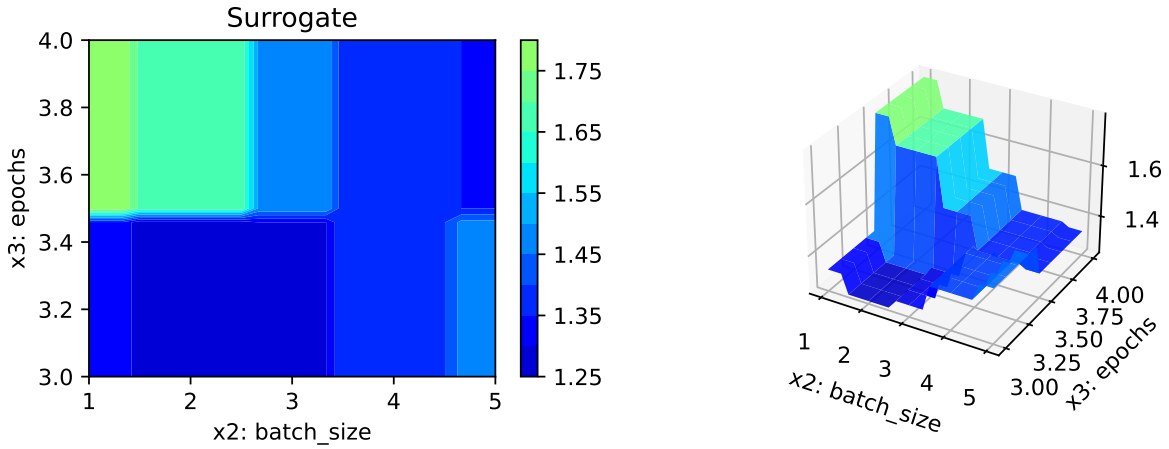
Model	Validation Loss	Validation Accuracy	Loss	Accuracy
Default	2.1221	0.2452	2.1182	0.2425
`spotPython`	1.2268	0.5896	1.2426	0.5957
`ray[tune]`	1.1815	0.5836	-	0.5806

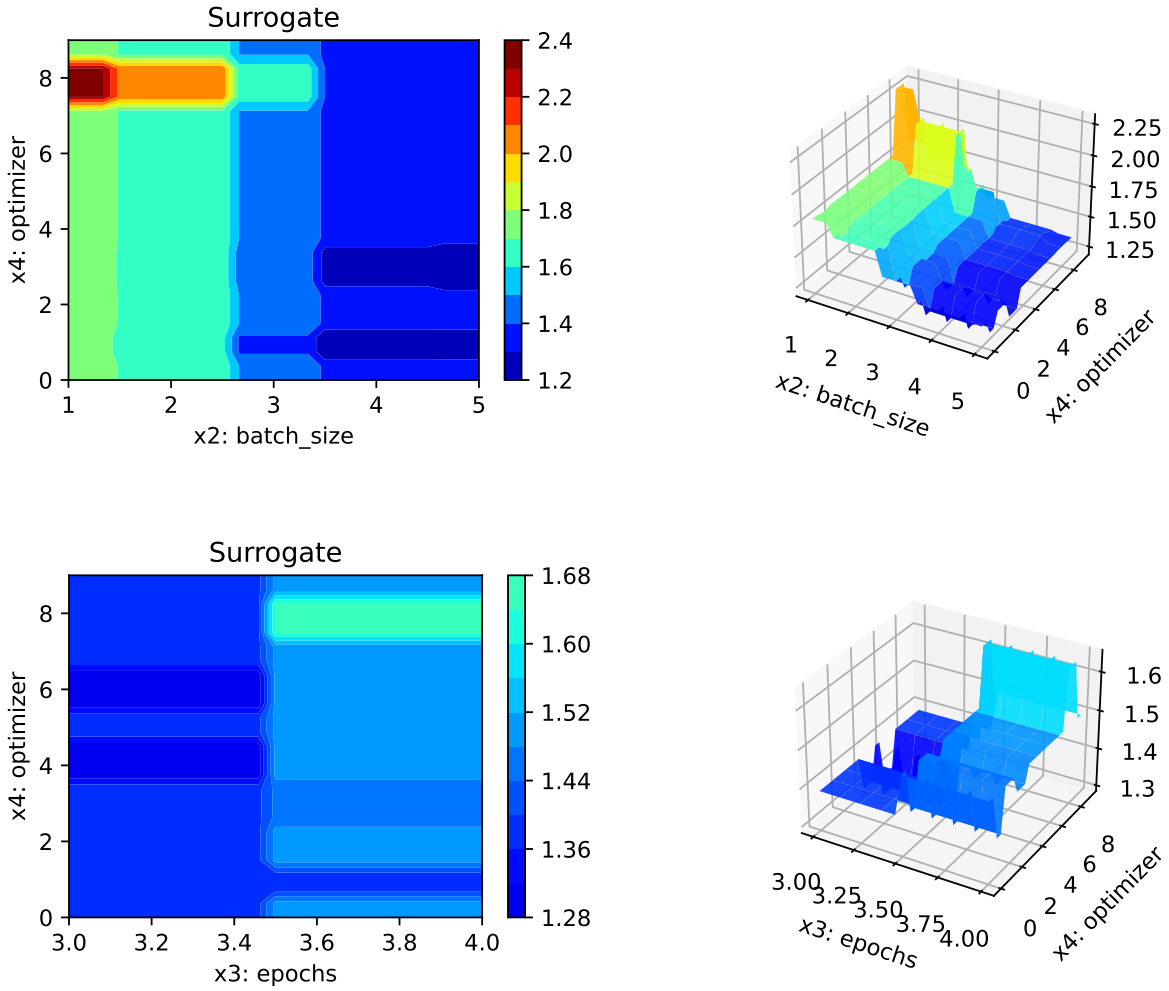
19.14.6 Detailed Hyperparameter Plots

The contour plots in this section visualize the interactions of the three most important hyperparameters, `l1`, `l2`, and `epochs`, and `optimizer` of the surrogate model used to optimize the hyperparameters. Since some of these hyperparameters take fatorial or integer values, sometimes step-like fitness landcapes (or response surfaces) are generated. SPOT draws the interactions of the main hyperparameters by default. It is also possible to visualize all interactions. For this, again refer to the notebook (Bartz-Beielstein 2023).

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
batch_size:  19.37438240435625
epochs:  100.0
optimizer:  42.90076243000858
```





The figures (Figure 19.6 to Figure 19.11) show the contour plots of the loss as a function of the hyperparameters. These plots are very helpful for benchmark studies and for understanding neural networks. `spotPython` provides additional tools for a visual inspection of the results and give valuable insights into the hyperparameter tuning process. This is especially useful for model explainability, transparency, and trustworthiness. In addition to the contour plots, Figure 20.7 shows the parallel plot of the hyperparameters.

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

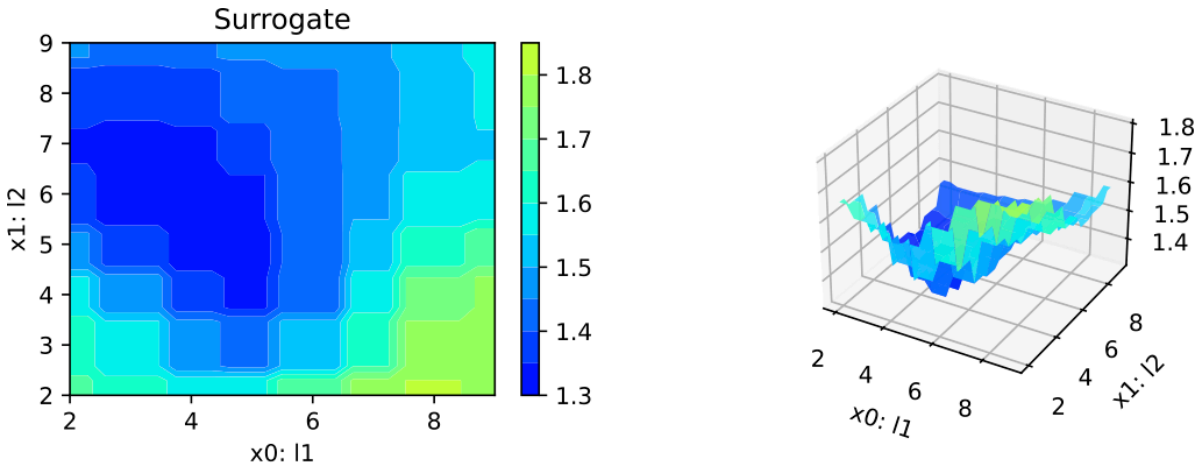


Figure 19.6: Contour plot of the loss as a function of `l1` and `l2`, i.e., the number of neurons in the layers.

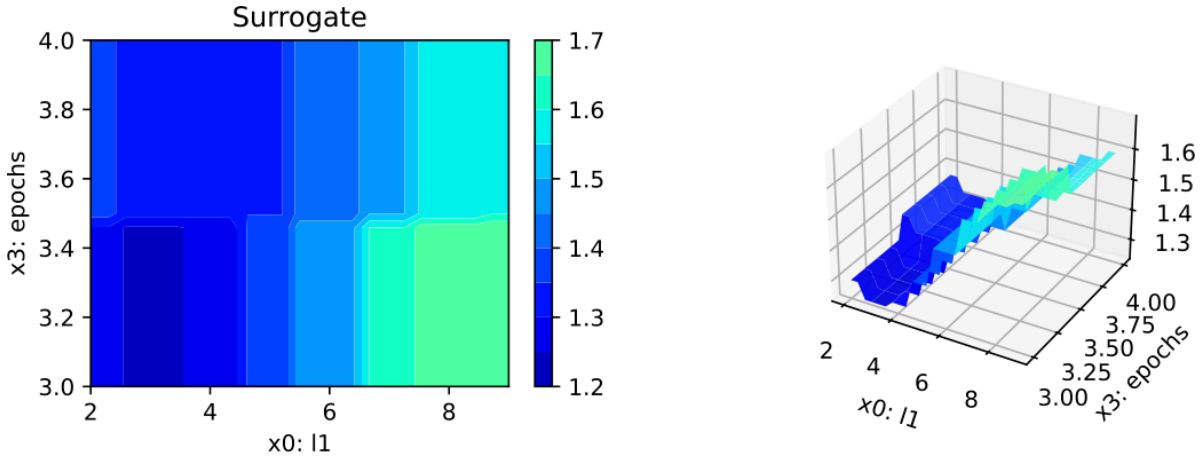


Figure 19.7: Contour plot of the loss as a function of the number of epochs and the neurons in layer `l1`.

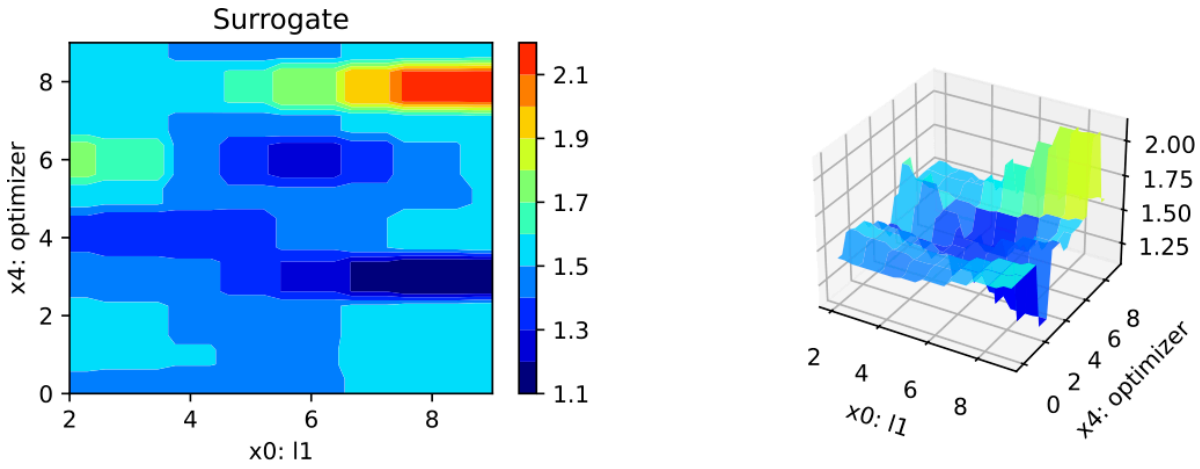


Figure 19.8: Contour plot of the loss as a function of the optimizer and the neurons in layer `l1`.

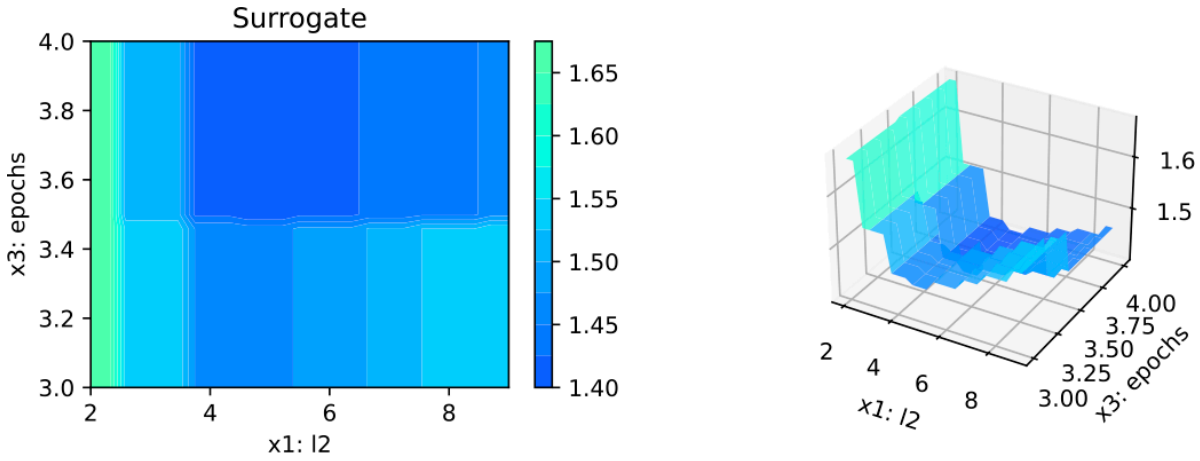


Figure 19.9: Contour plot of the loss as a function of the number of epochs and the neurons in layer `l2`.

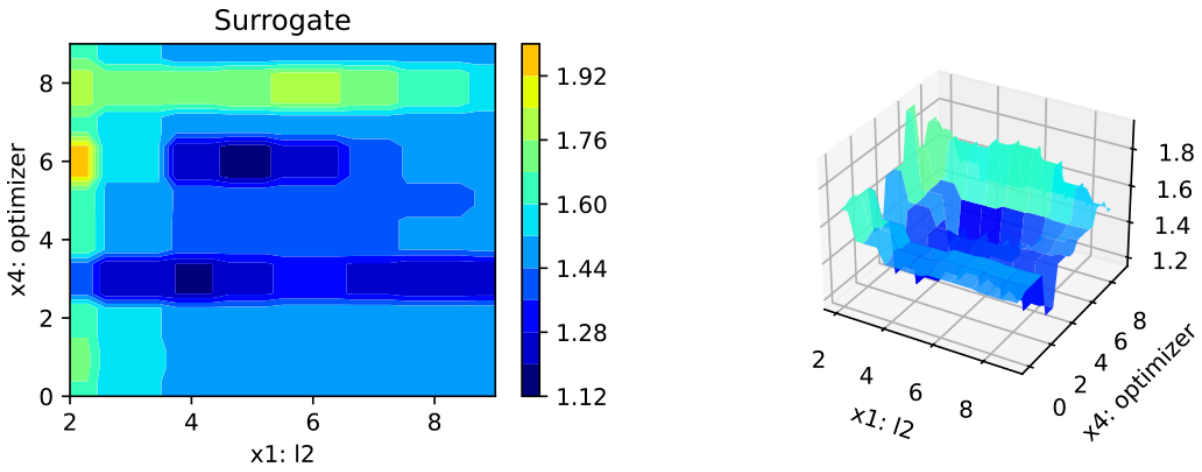


Figure 19.10: Contour plot of the loss as a function of the optimizer and the neurons in layer 12.

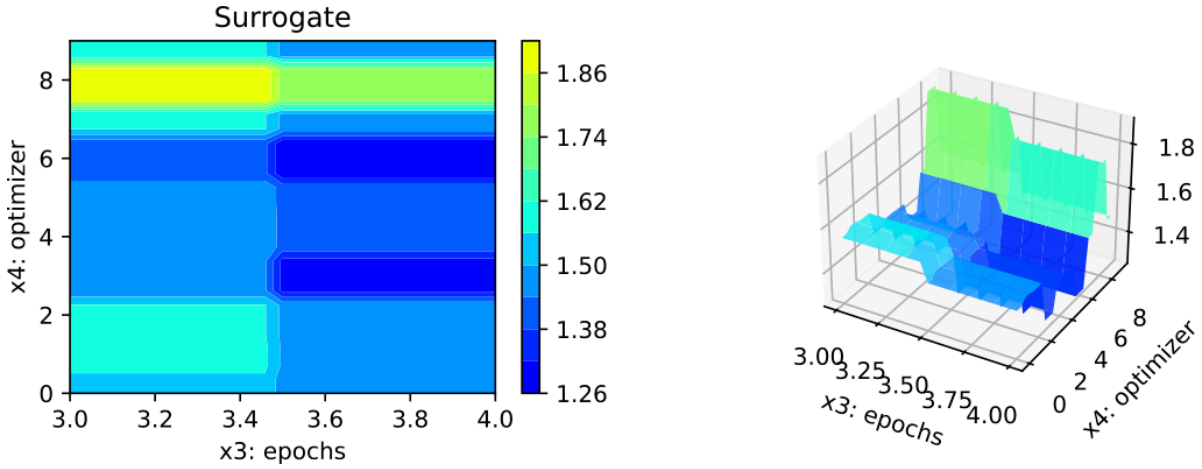


Figure 19.11: Contour plot of the loss as a function of the optimizer and the number of epochs.

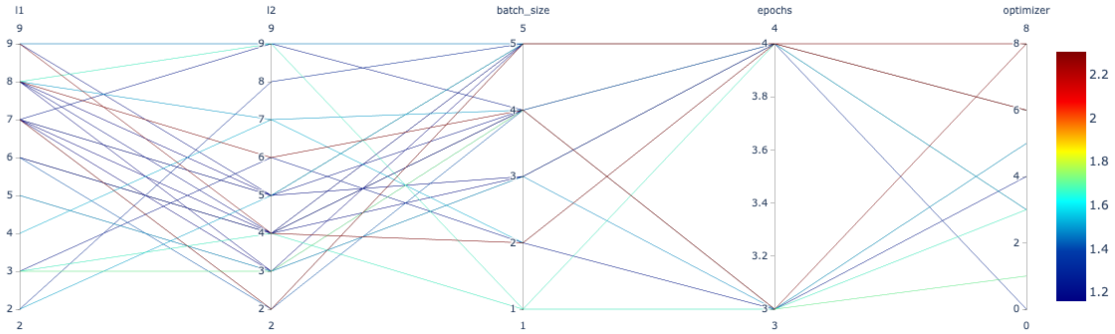


Figure 19.12: Parallel plot

19.15 Summary and Outlook

This tutorial presents the hyperparameter tuning open source software `spotPython` for `PyTorch`. To show its basic features, a comparison with the "official" `PyTorch` hyperparameter tuning tutorial (PyTorch 2023a) is presented. Some of the advantages of `spotPython` are:

- Numerical and categorical hyperparameters.
- Powerful surrogate models.
- Flexible approach and easy to use.
- Simple JSON files for the specification of the hyperparameters.
- Extension of default and user specified network classes.
- Noise handling techniques.
- Interaction with `tensorboard`.

Currently, only rudimentary parallel and distributed neural network training is possible, but these capabilities will be extended in the future. The next version of `spotPython` will also include a more detailed documentation and more examples.

> **!** Important
>
> Important: This tutorial does not present a complete benchmarking study (Bartz-Beielstein et al. 2020). The results are only preliminary and highly dependent on the local configuration (hard- and software). Our goal is to provide a first impression of the performance of the hyperparameter tuning package `spotPython`. To demonstrate its capabilities, a quick comparison with `ray[tune]` was performed. `ray[tune]` was chosen, because it is presented as "an industry standard tool for distributed hyperparameter

tuning." The results should be interpreted with care.

19.16 Appendix

19.16.1 Sample Output From Ray Tune's Run

The output from `ray[tune]` could look like this (PyTorch 2023b):

```
Number of trials: 10 (10 TERMINATED)
------+------+-------------+--------------+---------+------------+--------------------+
|   l1 |   l2 |          lr |   batch_size |    loss |   accuracy | training_iteration |
+------+------+-------------+--------------+---------+------------+--------------------|
|   64 |    4 | 0.00011629  |            2 | 1.87273 |     0.244  |                  2 |
|   32 |   64 | 0.000339763 |            8 | 1.23603 |     0.567  |                  8 |
|    8 |   16 | 0.00276249  |           16 | 1.1815  |     0.5836 |                 10 |
|    4 |   64 | 0.000648721 |            4 | 1.31131 |     0.5224 |                  8 |
|   32 |   16 | 0.000340753 |            8 | 1.26454 |     0.5444 |                  8 |
|    8 |    4 | 0.000699775 |            8 | 1.99594 |     0.1983 |                  2 |
|  256 |    8 | 0.0839654   |           16 | 2.3119  |     0.0993 |                  1 |
|   16 |  128 | 0.0758154   |           16 | 2.33575 |     0.1327 |                  1 |
|   16 |    8 | 0.0763312   |           16 | 2.31129 |     0.1042 |                  4 |
|  128 |   16 | 0.000124903 |            4 | 2.26917 |     0.1945 |                  1 |
+------+------+-------------+--------------+---------+------------+--------------------+
Best trial config: {'l1': 8, 'l2': 16, 'lr': 0.00276249, 'batch_size': 16, 'data_dir': '..
Best trial final validation loss: 1.181501
Best trial final validation accuracy: 0.5836
Best trial test set accuracy: 0.5806
```

20 Hyperparameter Tuning for PyTorch With spotPython: Regression

In this tutorial, we will show how spotPython can be integrated into the PyTorch training workflow.

This document refers to the following software versions:

- python: 3.10.10
- torch: 2.0.1
- torchvision: 0.15.0
- spotPython: 0.2.29

spotPython can be installed via pip. Alternatively, the source code can be downloaded from gitHub: https://github.com/sequential-parameter-optimization/spotPython.

```
!pip install spotPython
```

20.1 Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = None # "cpu" # "cuda:0"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

```
mps
```

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '24-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SI
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

```
24-torch_bartz09_1min_5init_2023-06-15_06-01-31
```

20.2 Initialization of the `fun_control` Dictionary

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process. This dictionary is called `fun_control` and is initialized with the function `fun_control_init`. The function `fun_control_init` returns a skeleton dictionary. The dictionary is filled with the required information for the hyperparameter tuning process. It stores the hyperparameter tuning settings, e.g., the deep learning network architecture that should be tuned, the classification (or regression) problem, and the data that is used for the tuning. The dictionary is used as an input for the SPOT function.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="regression",
 tensorboard_path="runs/24_spot_torch_regression",
 device=DEVICE)
```

20.3 PyTorch Data Loading

```
# Create dataset
import pandas as pd
import numpy as np
from sklearn import datasets as sklearn_datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

```
X, y = sklearn_datasets.make_regression(
    n_samples=1000, n_features=10, noise=1, random_state=123)
y = y.reshape(-1, 1)

# Normalize the data
X_scaler = MinMaxScaler()
X_scaled = X_scaler.fit_transform(X)
y_scaler = MinMaxScaler()
y_scaled = y_scaler.fit_transform(y)

# combine the features and target into a single dataframe named train_df
train_df = pd.DataFrame(np.hstack((X_scaled, y_scaled)))

target_column = "y"
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column,
    axis=1),
    train_df[target_column],
    random_state=42,
    test_size=0.25)
trainset = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
testset = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
trainset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
testset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
print(trainset.shape)
print(testset.shape)
```

```
(1000, 11)
(750, 11)
(250, 11)
```

```
import torch
from spotPython.torch.dataframedataset import DataFrameDataset
dtype_x = torch.float32
dtype_y = torch.float32
train_df = DataFrameDataset(train_df, target_column=target_column,
    dtype_x=dtype_x, dtype_y=dtype_y)
train = DataFrameDataset(trainset, target_column=target_column,
```

```
    dtype_x=dtype_x, dtype_y=dtype_y)
test = DataFrameDataset(testset, target_column=target_column,
    dtype_x=dtype_x, dtype_y=dtype_y)
n_samples = len(train)
```

- Now we can test the data loading:

```
from spotPython.torch.traintest import create_train_val_data_loaders
trainloader, testloader = create_train_val_data_loaders(train, 2, True, 0)
for i, data in enumerate(trainloader, 0):
    inputs, labels = data
    print(inputs.shape)
    print(labels.shape)
    print(inputs)
    print(labels)
    break
```

```
torch.Size([2, 10])
torch.Size([2])
tensor([[0.4509, 0.5118, 0.4808, 0.5563, 0.4139, 0.4307, 0.5563, 0.6938, 0.5978,
         0.4371],
        [0.3344, 0.6868, 0.6351, 0.2967, 0.5846, 0.5621, 0.4318, 0.3901, 0.4451,
         0.5040]])
tensor([0.4984, 0.5386])
```

- Since this works fine, we can add the data loading to the `fun_control` dictionary:

```
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
                "train": train,
                "test": test,
                "n_samples": n_samples,
                "target_column": target_column,})
```

20.4 Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables. The preprocessing model is called `prep_model` ("preparation" or pre-processing) and includes steps that are not subject to

the hyperparameter tuning process. The preprocessing model is specified in the `fun_control` dictionary. The preprocessing model can be implemented as a `sklearn` pipeline. The following code shows a typical preprocessing pipeline:

```
categorical_columns = ["cities", "colors"]
one_hot_encoder = OneHotEncoder(handle_unknown="ignore",
                                    sparse_output=False)
prep_model = ColumnTransformer(
        transformers=[
             ("categorical", one_hot_encoder, categorical_columns),
         ],
         remainder=StandardScaler(),
     )
```

```
fun_control.update({"prep_model": None})
```

20.5 Select `algorithm` and `core_model_hyper_dict`

20.5.1 Implementing a Configurable Neural Network With spotPython

`spotPython` includes the `Net_lin_reg` class which is implemented in the file `netregression.py`.

```
from torch import nn
import spotPython.torch.netcore as netcore


class Net_lin_reg(netcore.Net_Core):
    def __init__(
        self, _L_in, _L_out, l1, dropout_prob, lr_mult,
        batch_size, epochs, k_folds, patience, optimizer,
        sgd_momentum
    ):
        super(Net_lin_reg, self).__init__(
            lr_mult=lr_mult,
            batch_size=batch_size,
            epochs=epochs,
            k_folds=k_folds,
            patience=patience,
            optimizer=optimizer,
            sgd_momentum=sgd_momentum,
```

```
        )
        l2 = max(l1 // 2, 4)
        self.fc1 = nn.Linear(_L_in, l1)
        self.fc2 = nn.Linear(l1, l2)
        self.fc3 = nn.Linear(l2, _L_out)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)
        self.dropout1 = nn.Dropout(p=dropout_prob)
        self.dropout2 = nn.Dropout(p=dropout_prob / 2)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return x
```

20.5.1.1 The `Net_Core` class

`Net_lin_reg` inherits from the class `Net_Core` which is implemented in the file `netcore.py`. It implements the additional attributes that are common to all neural network models. The `Net_Core` class is implemented in the file `netcore.py`. It implements hyperparameters as attributes, that are not used by the `core_model`, e.g.:

- optimizer (`optimizer`),
- learning rate (`lr`),
- batch size (`batch_size`),
- epochs (`epochs`),
- k_folds (`k_folds`), and
- early stopping criterion "patience" (`patience`).

Users can add further attributes to the class. The class `Net_Core` is shown below.

```
from torch import nn


class Net_Core(nn.Module):
    def __init__(self, lr_mult, batch_size, epochs, k_folds, patience,
```

```
    optimizer, sgd_momentum):
        super(Net_Core, self).__init__()
        self.lr_mult = lr_mult
        self.batch_size = batch_size
        self.epochs = epochs
        self.k_folds = k_folds
        self.patience = patience
        self.optimizer = optimizer
        self.sgd_momentum = sgd_momentum
```

We see that the class `Net_lin_reg` has additional attributes and does not inherit from `nn` directly. It adds an additional class, `Net_core`, that takes care of additional attributes that are common to all neural network models, e.g., the learning rate multiplier `lr_mult` or the batch size `batch_size`.

`spotPython`'s `core_model` implements an instance of the `Net_lin_reg` class. In addition to the basic neural network model, the `core_model` can use these additional attributes. `spotPython` provides methods for handling these additional attributes to guarantee 100% compatibility with the `PyTorch` classes. The method `add_core_model_to_fun_control` adds the hyperparameters and additional attributes to the `fun_control` dictionary. The method is shown below.

```
from spotPython.torch.netregression import Net_lin_reg
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
core_model = Net_lin_reg
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                fun_control=fun_control,
                                hyper_dict=TorchHyperDict,
                                filename=None)
```

20.6 The Search Space

20.6.1 Configuring the Search Space With spotPython

20.6.1.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

`spotPython` uses `JSON` files for the specification of the hyperparameters. Users can specify their individual `JSON` files, or they can use the `JSON` files provided by `spotPython`. The `JSON` file for the `core_model` is called `torch_hyper_dict.json`.

spotPython can handle numerical, boolean, and categorical hyperparameters. They can be specified in the JSON file in a similar way as the numerical hyperparameters as shown below. Each entry in the JSON file represents one hyperparameter with the following structure: type, default, transform, lower, and upper.

```
"factor_hyperparameter": {
    "levels": ["A", "B", "C"],
    "type": "factor",
    "default": "B",
    "transform": "None",
    "core_model_parameter_type": "str",
    "lower": 0,
    "upper": 2},
```

The corresponding entries for the Net_lin_reg class are shown below.

```
    "Net_lin_reg":
    {
        "_L_in": {
            "type": "int",
            "default": 10,
            "transform": "None",
            "lower": 10,
            "upper": 10},
        "_L_out": {
            "type": "int",
            "default": 1,
            "transform": "None",
            "lower": 1,
            "upper": 1},
        "l1": {
            "type": "int",
            "default": 3,
            "transform": "transform_power_2_int",
            "lower": 3,
            "upper": 8},
        "dropout_prob": {
            "type": "float",
            "default": 0.01,
            "transform": "None",
            "lower": 0.0,
            "upper": 0.9},
```

```
        "lr_mult": {
            "type": "float",
            "default": 1.0,
            "transform": "None",
            "lower": 0.1,
            "upper": 10.0},
        "batch_size": {
            "type": "int",
            "default": 4,
            "transform": "transform_power_2_int",
            "lower": 1,
            "upper": 4},
        "epochs": {
            "type": "int",
            "default": 4,
            "transform": "transform_power_2_int",
            "lower": 4,
            "upper": 9},
        "k_folds": {
            "type": "int",
            "default": 1,
            "transform": "None",
            "lower": 1,
            "upper": 1},
        "patience": {
            "type": "int",
            "default": 2,
            "transform": "transform_power_2_int",
            "lower": 1,
            "upper": 5
        },
        "optimizer": {
            "levels": ["Adadelta",
                       "Adagrad",
                       "Adam",
                       "AdamW",
                       "SparseAdam",
                       "Adamax",
                       "ASGD",
                       "NAdam",
                       "RAdam",
```

```
                    "RMSprop",
                    "Rprop",
                    "SGD"],
            "type": "factor",
            "default": "SGD",
            "transform": "None",
            "class_name": "torch.optim",
            "core_model_parameter_type": "str",
            "lower": 0,
            "upper": 12},
        "sgd_momentum": {
            "type": "float",
            "default": 0.0,
            "transform": "None",
            "lower": 0.0,
            "upper": 1.0}
    },
```

20.7 Modifying the Hyperparameters

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions are described in the following.

20.7.1 Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

After specifying the model, the corresponding hyperparameters, their types and bounds are loaded from the `JSON` file `torch_hyper_dict.json`. After loading, the user can modify the hyperparameters, e.g., the bounds. `spotPython` provides a simple rule for de-activating hyperparameters: If the lower and the upper bound are set to identical values, the hyperparameter is de-activated. This is useful for the hyperparameter tuning, because it allows to specify a hyperparameter in the `JSON` file, but to de-activate it in the `fun_control` dictionary. This is done in the next step.

20.7.2 Modify Hyperparameters of Type numeric and integer (boolean)

```
# modify the hyperparameter levels

from spotPython.hyperparameters.values import modify_hyper_parameter_bounds

fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 16])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[3, 7])
```

20.7.3 Modify Hyperparameter of Type factor

In a similar manner as for the numerical hyperparameters, the categorical hyperparameters can be modified. New configurations can be chosen by adding or deleting levels. For example, the hyperparameter `optimizer` can be re-configured as follows:

In the following setting, two optimizers (`"SGD"` and `"Adam"`) will be compared during the `spotPython` hyperparameter tuning. The hyperparameter `optimizer` is active.

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer",
    ["SGD", "Adam"])
```

The hyperparameter `optimizer` can be de-activated by choosing only one value (level), here: `"SGD"`.

```
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["SGD"])
```

As discussed in Section 20.7.4, there are some issues with the LBFGS optimizer. Therefore, the usage of the LBFGS optimizer is not deactivated in `spotPython` by default. However, the LBFGS optimizer can be activated by adding it to the list of optimizers. `Rprop` was removed, because it does perform very poorly (as some pre-tests have shown). However, it can also be activated by adding it to the list of optimizers. Since `SparseAdam` does not support dense gradients, `Adam` was used instead. Therefore, there are 10 default optimizers:

```
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer",
    ["Adadelta", "Adagrad", "Adam", "AdamW", "Adamax", "ASGD", "NAdam"])


fun_control.update({
               "_L_in": n_features,
               "_L_out": 1,})
```

20.7.4 Optimizers

Table 20.1 shows some of the optimizers available in `PyTorch`:

Table 20.1: Optimizers available in PyTorch (selection). "mom" denotes `momentum`, "weight" `weight_decay`, "damp" `dampening`, "nest" `nesterov`, "lr_sc" `learning rate for scaling delta`, "mom_dec" for `momentum_decay`, and "step_s" for `step_sizes`. The default values are shown in the table.

Optimizer	lr	mom	weight	damp	nest	rho	lr_sc	lr_dec	betas	lambd	alpha	mom_dec	decay	step_s
Adadelta	-	-	0.	-	-	0.9	1.0	-	-	-	-	-	-	-
Adagrad	1e-2	-	0.	-	-	-	-	0.	-	-	-	-	-	-
Adam	1e-3	-	0.	-	-	-	-	-	(0.9,0.999)	-	-	-	-	-
AdamW	1e-3	-	1e-2	-	-	-	-	-	(0.9,0.999)	-	-	-	-	-
SparseAdam	1e-3	-	-	-	-	-	-	-	(0.9,0.999)	-	-	-	-	-
Adamax	2e-3	-	0.	-	-	-	-	-	(0.9, 0.999)	-	-	-	-	-
ASGD	1e-2	0.9	0.	-	False	-	-	-	-	1e-4	0.75	-	-	-
LBFGS	1.	-	-	-	-	-	-	-	-	-	-	-	-	-
NAdam	2e-3	-	0.	-	-	-	-	-	(0.9,0.999)	-	0	-	-	-
RAdam	1e-3	-	0.	-	-	-	-	-	(0.9,0.999)	-	-	-	-	-
RMSprop	1e-2	0.	0.	-	-	-	-	-	(0.9,0.999)	-	-	-	-	-
Rprop	1e-2	-	-	-	-	-	-	-	-	-	(0.5,1.2)	(1e-6, 50)	-	-
SGD	required	0.	0.	0.	False	-	-	-	-	-	-	-	-	-

`spotPython` implements an `optimization` handler that maps the optimizer names to the corresponding `PyTorch` optimizers.

> **A note on LBFGS**
>
> We recommend deactivating `PyTorch`'s LBFGS optimizer, because it does not perform very well. The `PyTorch` documentation, see https://pytorch.org/docs/stable/generated/torch.optim.LBFGS.html#torch.optim.LBFGS, states:
>
> > This is a very memory intensive optimizer (it requires additional `param_bytes * (history_size + 1)` bytes). If it doesn't fit in memory try reducing the history size, or use a different algorithm.
>
> Furthermore, the LBFGS optimizer is not compatible with the `PyTorch` tutorial. The reason is that the LBFGS optimizer requires the `closure` function, which is not implemented in the `PyTorch` tutorial. Therefore, the `LBFGS` optimizer is recommended here.

Since there are 10 optimizers in the portfolio, it is not recommended tuning the hyperparameters that effect one single optimizer only.

> **A note on the learning rate**
>
> `spotPython` provides a multiplier for the default learning rates, `lr_mult`, because optimizers use different learning rates. Using a multiplier for the learning rates might enable a simultaneous tuning of the learning rates for all optimizers. However, this is not recommended, because the learning rates are not comparable across optimizers. Therefore, we recommend fixing the learning rate for all optimizers if multiple optimizers are used. This can be done by setting the lower and upper bounds of the learning rate multiplier to the same value as shown below.

Thus, the learning rate, which affects the `SGD` optimizer, will be set to a fixed value. We choose the default value of `1e-3` for the learning rate, because it is used in other `PyTorch` examples (it is also the default value used by `spotPython` as defined in the `optimizer_handler()` method). We recommend tuning the learning rate later, when a reduced set of optimizers is fixed. Here, we will demonstrate how to select in a screening phase the optimizers that should be used for the hyperparameter tuning.

For the same reason, we will fix the `sgd_momentum` to `0.9`.

```
fun_control = modify_hyper_parameter_bounds(fun_control,
    "lr_mult", bounds=[1e-3, 1e-3])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "sgd_momentum", bounds=[0.9, 0.9])
```

20.8 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set and
2. the loss function (and a metric).

20.8.1 Hold-out Data Split and Cross-Validation

As a default, `spotPython` provides a standard hold-out data split and cross validation.

20.8.1.1 Hold-out Data Split

If a hold-out data split is used, the data will be partitioned into a training, a validation, and a test data set. The split depends on the setting of the `eval` parameter. If `eval` is set to `train_hold_out`, one data set, usually the original training data set, is split into a new training and a validation data set. The training data set is used for training the model. The validation data set is used for the evaluation of the hyperparameter configuration and early stopping to prevent overfitting. In this case, the original test data set is not used. The following splits are performed in the hold-out setting: $\{\text{train}_0, \text{test}\} \rightarrow \{\text{train}_1, \text{validation}_1, \text{test}\}$, where $\text{train}_1 \cup \text{validation}_1 = \text{train}_0$.

> **Note**
>
> `spotPython` returns the hyperparameters of the machine learning and deep learning models, e.g., number of layers, learning rate, or optimizer, but not the model weights. Therefore, after the SPOT run is finished, the corresponding model with the optimized architecture has to be trained again with the best hyperparameter configuration. The training is performed on the training data set. The test data set is used for the final evaluation of the model.
> Summarizing, the following splits are performed in the hold-out setting:
>
> 1. Run `spotPython` with `eval` set to `train_hold_out` to determine the best hyperparameter configuration.
> 2. Train the model with the best hyperparameter configuration ("architecture") on the training data set:
>   - `train_tuned(model_spot, train, "model_spot.pt")`.
> 3. Test the model on the test data:
>   - `test_tuned(model_spot, test, "model_spot.pt")`
>
> These steps will be exemplified in the following sections.

In addition to this `hold-out` setting, `spotPython` provides another hold-out setting, where an explicit test data is specified by the user that will be used as the validation set. To choose this option, the `eval` parameter is set to `test_hold_out`. In this case, the training data set is used for the model training. Then, the explicitly defined test data set is used for the evaluation of the hyperparameter configuration (the validation).

20.8.1.2 Cross-Validation

The cross validation setting is used by setting the `eval` parameter to `train_cv` or `test_cv`. In both cases, the data set is split into k folds. The model is trained on $k-1$ folds and evaluated on the remaining fold. This is repeated k times, so that each fold is used exactly once for evaluation. The final evaluation is performed on the test data set. The cross validation setting is useful for small data sets, because it allows to use all data for training and evaluation. However, it is computationally expensive, because the model has to be trained k times.

> **Note**
>
> Combinations of the above settings are possible, e.g., cross validation can be used for training and hold-out for evaluation or *vice versa*. Also, cross validation can be used for training and testing. Because cross validation is not used in the `PyTorch` tutorial (PyTorch 2023a), it is not considered further here.

20.8.1.3 Overview of the Evaluation Settings

20.8.1.3.1 Settings for the Hyperparameter Tuning

Table 20.2 provides an overview of the training evaluations.

Table 20.2: Overview of the evaluation settings.

`eval`	`train`	`test`	function	comment
`"train_hold_out"`	✓		`train_one_epoch()`, `validate_one_epoch()` for early stopping	splits the `train` data set internally
`"test_hold_out"`	✓	✓	`train_one_epoch()`, `validate_one_epoch()` for early stopping	use the `test data set` for `validate_one_epoch()`
`"train_cv"`	✓		`evaluate_cv(net, train)`	CV using the `train` data set

eval	train	test	function	comment
"test_cv"		✓	evaluate_cv(net, test)	CV using the test data set . Identical to "train_cv", uses only test data.

- "train_cv" and "test_cv" use sklearn.model_selection.KFold() internally.

20.8.1.4 Settings for the Final Evaluation of the Tuned Architecture

20.8.1.4.1 Training of the Tuned Architecture

train_tuned(model, train): train the model with the best hyperparameter configuration (or simply the default) on the training data set. It splits the traindata into new train and validation sets using create_train_val_data_loaders(), which calls torch.utils.data.random_split() internally. Currently, 60% of the data is used for training and 40% for validation. The train data is used for training the model with train_one_epoch(). The validation data is used for early stopping using validate_one_epoch() on the validation data set.

20.8.1.4.2 Testing of the Tuned Architecture

test_tuned(model, test): test the model on the test data set. No data splitting is performed. The (trained) model is evaluated using the validate_one_epoch() function.

Note: During training, shuffle is set to True, whereas during testing, shuffle is set to False.

20.8.2 Loss Functions and Metrics

The key "loss_function" specifies the loss function which is used during the optimization. There are several different loss functions under PyTorch's nn package. For example, a simple loss is MSELoss, which computes the mean-squared error between the output and the target. In this tutorial we will use CrossEntropyLoss, because it is also used in the PyTorch tutorial.

20.8.2.1 Loss Function

The loss function is specified by the key "loss_function". We will use MSE loss for the regression task.

```
from torch.nn import MSELoss
loss_torch = MSELoss()
fun_control.update({"loss_function": loss_torch})
```

In addition to the loss functions, `spotPython` provides access to a large number of metrics.

- The key `"metric_sklearn"` is used for metrics that follow the `scikit-learn` conventions.
- The key `"river_metric"` is used for the river based evaluation (Montiel et al. 2021) via `eval_oml_iter_progressive`, and
- the key `"metric_torch"` is used for the metrics from `TorchMetrics`.

`TorchMetrics` is a collection of more than 90 PyTorch metrics[1].

```
from torchmetrics import MeanAbsoluteError
metric_torch = MeanAbsoluteError(device=fun_control["device"])
fun_control.update({"metric_torch": metric_torch})
```

20.9 Calling the SPOT Function

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` generates a design table as follows:

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

[1] https://torchmetrics.readthedocs.io/en/latest/.

```
| name          | type   | default   |   lower |   upper | transform             |
|---------------|--------|-----------|---------|---------|-----------------------|
| _L_in         | int    | 10        |  10     |  10     | None                  |
| _L_out        | int    | 1         |   1     |   1     | None                  |
| l1            | int    | 3         |   3     |   8     | transform_power_2_int |
| dropout_prob  | float  | 0.01      |   0     |   0.9   | None                  |
| lr_mult       | float  | 1.0       |   0.001 |   0.001 | None                  |
| batch_size    | int    | 4         |   1     |   4     | transform_power_2_int |
| epochs        | int    | 4         |   2     |  16     | transform_power_2_int |
| k_folds       | int    | 1         |   1     |   1     | None                  |
| patience      | int    | 2         |   3     |   7     | transform_power_2_int |
| optimizer     | factor | SGD       |   0     |   6     | None                  |
| sgd_momentum  | float  | 0.0       |   0.9   |   0.9   | None                  |
```

This allows to check if all information is available and if the information is correct. Table 20.3 shows the experimental design for the hyperparameter tuning. Hyperparameter transformations are shown in the column "transform", e.g., the `l1` default is `5`, which results in the value $2^5 = 32$ for the network, because the transformation `transform_power_2_int` was selected in the `JSON` file. The default value of the `batch_size` is set to `4`, which results in a batch size of $2^4 = 16$.

Table 20.3: Experimental design for the hyperparameter tuning. The table shows the hyperparameters, their types, default values, lower and upper bounds, and the transformation function. The transformation function is used to transform the hyperparameter values from the unit hypercube to the original domain. The transformation function is applied to the hyperparameter values before the evaluation of the objective function.

name	type	default	lower	upper	transform
_L_in	int	10	10	10	None
_L_out	int	1	1	1	None
l1	int	3	3	8	transform_power_2_int
dropout_prob	float	0.01	0	0.9	None
lr_mult	float	1.0	0.001	0.001	None
batch_size	int	4	1	4	transform_power_2_int
epochs	int	4	2	16	transform_power_2_int
k_folds	int	1	1	1	None
patience	int	2	3	7	transform_power_2_int
optimizer	factor	SGD	0	6	None
sgd_momentum	float	0.0	0.9	0.9	None

The objective function `fun_torch` is selected next. It implements an interface from `PyTorch`'s

training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch
```

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=TorchHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
```

```
fun_control.update({
               "device": "cpu",
               })
```

The `spotPython` hyperparameter tuning is started by calling the `Spot` function. Here, we will run the tuner for approximately 30 minutes (`max_time`). Note: the initial design is always evaluated in the `spotPython` run. As a consequence, the run may take longer than specified by `max_time`, because the evaluation time of initial design (here: `init_size`, 10 points) is performed independently of `max_time`.

```
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = inf,
                   fun_repeats = 1,
                   max_time = MAX_TIME,
                   noise = False,
                   tolerance_x = np.sqrt(np.spacing(1)),
                   var_type = var_type,
                   var_name = var_name,
                   infill_criterion = "y",
                   n_points = 1,
                   seed=123,
                   log_level = 50,
                   show_models= False,
                   show_progress= True,
                   fun_control = fun_control,
                   design_control={"init_size": INIT_SIZE,
                                   "repeats": 1},
                   surrogate_control={"noise": True,
```

```
                                    "cod_type": "norm",
                                    "min_theta": -4,
                                    "max_theta": 3,
                                    "n_theta": len(var_name),
                                    "model_fun_evals": 10_000,
                                    "log_level": 50
                                    })
spot_tuner.run(X_start=X_start)
```

```
config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7103122166156, 'lr_mult': 0.0

Epoch: 1
Loss on hold-out set: 0.27737906952699026
MeanAbsoluteError value on hold-out data: 0.4912404417991638
Epoch: 2
Loss on hold-out set: 0.2705897197127342
MeanAbsoluteError value on hold-out data: 0.48202332854270935
Epoch: 3
Loss on hold-out set: 0.2712238362431526
MeanAbsoluteError value on hold-out data: 0.48382359743118286
Epoch: 4

Loss on hold-out set: 0.2779320180416107
MeanAbsoluteError value on hold-out data: 0.49331846833229065
Epoch: 5
Loss on hold-out set: 0.2699290296435356
MeanAbsoluteError value on hold-out data: 0.48607102036476135
Epoch: 6
Loss on hold-out set: 0.2737545285622279
MeanAbsoluteError value on hold-out data: 0.4855026304721832
Epoch: 7
Loss on hold-out set: 0.25806730419397356
MeanAbsoluteError value on hold-out data: 0.4727768003940582
Epoch: 8
Loss on hold-out set: 0.24748795409997304
MeanAbsoluteError value on hold-out data: 0.46238577365875244
Epoch: 9

Loss on hold-out set: 0.2536972798903783
MeanAbsoluteError value on hold-out data: 0.46582168340682983
```

```
Epoch: 10
Loss on hold-out set: 0.2521917387843132
MeanAbsoluteError value on hold-out data: 0.46241840720176697
Epoch: 11
Loss on hold-out set: 0.2450722716252009
MeanAbsoluteError value on hold-out data: 0.4587111473083496
Epoch: 12
Loss on hold-out set: 0.24045813217759132
MeanAbsoluteError value on hold-out data: 0.4553874135017395
Epoch: 13

Loss on hold-out set: 0.2403079240520795
MeanAbsoluteError value on hold-out data: 0.4536590576171875
Epoch: 14
Loss on hold-out set: 0.24449915210405984
MeanAbsoluteError value on hold-out data: 0.4555387496948242
Epoch: 15
Loss on hold-out set: 0.24063367625077567
MeanAbsoluteError value on hold-out data: 0.45135629177093506
Epoch: 16
Loss on hold-out set: 0.22750168919563293
MeanAbsoluteError value on hold-out data: 0.4356045424938202
Epoch: 17

Loss on hold-out set: 0.23727207948764165
MeanAbsoluteError value on hold-out data: 0.448472797870636
Epoch: 18
Loss on hold-out set: 0.23073945050438247
MeanAbsoluteError value on hold-out data: 0.4410744309425354
Epoch: 19
Loss on hold-out set: 0.22165440306067466
MeanAbsoluteError value on hold-out data: 0.4354499876499176
Epoch: 20
Loss on hold-out set: 0.22623526046673456
MeanAbsoluteError value on hold-out data: 0.43641287088394165
Epoch: 21
Loss on hold-out set: 0.229056775867939
MeanAbsoluteError value on hold-out data: 0.4362049996852875
Epoch: 22

Loss on hold-out set: 0.22721160585681596
MeanAbsoluteError value on hold-out data: 0.43596065044403076
```

```
Epoch: 23
Loss on hold-out set: 0.21990032409628232
MeanAbsoluteError value on hold-out data: 0.42741939425468445
Epoch: 24
Loss on hold-out set: 0.2206366824110349
MeanAbsoluteError value on hold-out data: 0.4288545846939087
Epoch: 25
Loss on hold-out set: 0.22006802196304004
MeanAbsoluteError value on hold-out data: 0.4237593710422516
Epoch: 26
Loss on hold-out set: 0.21346615913013617
MeanAbsoluteError value on hold-out data: 0.42017126083374023
Epoch: 27

Loss on hold-out set: 0.21200790842374165
MeanAbsoluteError value on hold-out data: 0.41983115673065186
Epoch: 28
Loss on hold-out set: 0.2098051281273365
MeanAbsoluteError value on hold-out data: 0.418117493391037
Epoch: 29
Loss on hold-out set: 0.19744210541248322
MeanAbsoluteError value on hold-out data: 0.40562331676483154
Epoch: 30
Loss on hold-out set: 0.20144305899739265
MeanAbsoluteError value on hold-out data: 0.4073893129825592
Epoch: 31

Loss on hold-out set: 0.2055955464641253
MeanAbsoluteError value on hold-out data: 0.41088494658470154
Epoch: 32
Loss on hold-out set: 0.1922033777832985
MeanAbsoluteError value on hold-out data: 0.39671993255615234
Epoch: 33
Loss on hold-out set: 0.19681458309292793
MeanAbsoluteError value on hold-out data: 0.40227168798446655
Epoch: 34
Loss on hold-out set: 0.1934041026731332
MeanAbsoluteError value on hold-out data: 0.3988163471221924
Epoch: 35
Loss on hold-out set: 0.19059295912583668
MeanAbsoluteError value on hold-out data: 0.39460936188697815
```

```
Epoch: 36
Loss on hold-out set: 0.1871507304906845
MeanAbsoluteError value on hold-out data: 0.38783013820648193
Epoch: 37
Loss on hold-out set: 0.17830978284279506
MeanAbsoluteError value on hold-out data: 0.38025036454200745
Epoch: 38
Loss on hold-out set: 0.18173522209127743
MeanAbsoluteError value on hold-out data: 0.3817526400089264
Epoch: 39

Loss on hold-out set: 0.18025696923335394
MeanAbsoluteError value on hold-out data: 0.38219502568244934
Epoch: 40
Loss on hold-out set: 0.17406498298048972
MeanAbsoluteError value on hold-out data: 0.375672310590744
Epoch: 41
Loss on hold-out set: 0.17278824066122372
MeanAbsoluteError value on hold-out data: 0.3731611967086792
Epoch: 42
Loss on hold-out set: 0.16155908152461051
MeanAbsoluteError value on hold-out data: 0.36028921604156494
Epoch: 43

Loss on hold-out set: 0.17067683296899
MeanAbsoluteError value on hold-out data: 0.3666785657405853
Epoch: 44
Loss on hold-out set: 0.16950856293241182
MeanAbsoluteError value on hold-out data: 0.36785975098609924
Epoch: 45
Loss on hold-out set: 0.15350337187449137
MeanAbsoluteError value on hold-out data: 0.3494012951850891
Epoch: 46
Loss on hold-out set: 0.1756304343789816
MeanAbsoluteError value on hold-out data: 0.37046146392822266
Epoch: 47
Loss on hold-out set: 0.15879406755169231
MeanAbsoluteError value on hold-out data: 0.3540625274181366
Epoch: 48

Loss on hold-out set: 0.16097397059202195
MeanAbsoluteError value on hold-out data: 0.3569835126399994
```

```
Epoch: 49
Loss on hold-out set: 0.1659414391219616
MeanAbsoluteError value on hold-out data: 0.36259201169013977
Epoch: 50
Loss on hold-out set: 0.15444167755544186
MeanAbsoluteError value on hold-out data: 0.34542039036750793
Epoch: 51
Loss on hold-out set: 0.15915436933437982
MeanAbsoluteError value on hold-out data: 0.35322046279907227
Epoch: 52

Loss on hold-out set: 0.15414643745869397
MeanAbsoluteError value on hold-out data: 0.3467097878456116
Epoch: 53
Loss on hold-out set: 0.1526827088991801
MeanAbsoluteError value on hold-out data: 0.34680479764938354
Epoch: 54
Loss on hold-out set: 0.14936665962139764
MeanAbsoluteError value on hold-out data: 0.34166067838668823
Epoch: 55
Loss on hold-out set: 0.14407242432236672
MeanAbsoluteError value on hold-out data: 0.338095486164093
Epoch: 56

Loss on hold-out set: 0.1436948707451423
MeanAbsoluteError value on hold-out data: 0.3357619345188141
Epoch: 57
Loss on hold-out set: 0.1414409562945366
MeanAbsoluteError value on hold-out data: 0.32925474643707275
Epoch: 58
Loss on hold-out set: 0.13535741612315177
MeanAbsoluteError value on hold-out data: 0.31960976123809814
Epoch: 59
Loss on hold-out set: 0.13991815388202666
MeanAbsoluteError value on hold-out data: 0.3246578872203827
Epoch: 60

Loss on hold-out set: 0.13742430248608192
MeanAbsoluteError value on hold-out data: 0.3267269730567932
Epoch: 61
Loss on hold-out set: 0.13230000071227552
MeanAbsoluteError value on hold-out data: 0.3179420828819275
```

```
Epoch: 62
Loss on hold-out set: 0.1349130884433786
MeanAbsoluteError value on hold-out data: 0.32005539536476135
Epoch: 63
Loss on hold-out set: 0.13697888659934201
MeanAbsoluteError value on hold-out data: 0.3281429708003998
Epoch: 64
Loss on hold-out set: 0.1262283267825842
MeanAbsoluteError value on hold-out data: 0.3103582262992859
Epoch: 65

Loss on hold-out set: 0.11606008067727089
MeanAbsoluteError value on hold-out data: 0.29626303911209106
Epoch: 66
Loss on hold-out set: 0.1310575549180309
MeanAbsoluteError value on hold-out data: 0.31181034445762634
Epoch: 67
Loss on hold-out set: 0.12121988291541735
MeanAbsoluteError value on hold-out data: 0.3035274147987366
Epoch: 68
Loss on hold-out set: 0.12290513683731358
MeanAbsoluteError value on hold-out data: 0.30045464634895325
Epoch: 69

Loss on hold-out set: 0.11914017781615258
MeanAbsoluteError value on hold-out data: 0.2982252836227417
Epoch: 70
Loss on hold-out set: 0.11229815497994423
MeanAbsoluteError value on hold-out data: 0.29574766755104065
Epoch: 71
Loss on hold-out set: 0.11817136923472087
MeanAbsoluteError value on hold-out data: 0.29896360635757446
Epoch: 72
Loss on hold-out set: 0.116539318288366
MeanAbsoluteError value on hold-out data: 0.28645533323287964
Epoch: 73
Loss on hold-out set: 0.1169979212184747
MeanAbsoluteError value on hold-out data: 0.29432106018066406
Epoch: 74

Loss on hold-out set: 0.11237254635741313
MeanAbsoluteError value on hold-out data: 0.28294435143470764
```

```
Epoch: 75
Loss on hold-out set: 0.11764826965828737
MeanAbsoluteError value on hold-out data: 0.28731557726860046
Epoch: 76
Loss on hold-out set: 0.10680029721309742
MeanAbsoluteError value on hold-out data: 0.2777889668941498
Epoch: 77
Loss on hold-out set: 0.10956076132754484
MeanAbsoluteError value on hold-out data: 0.28175005316734314
Epoch: 78

Loss on hold-out set: 0.10687698734303315
MeanAbsoluteError value on hold-out data: 0.28305375576019287
Epoch: 79
Loss on hold-out set: 0.10335739893217881
MeanAbsoluteError value on hold-out data: 0.272903710603714
Epoch: 80
Loss on hold-out set: 0.10294156635800998
MeanAbsoluteError value on hold-out data: 0.27492091059684753
Epoch: 81
Loss on hold-out set: 0.09751483085254828
MeanAbsoluteError value on hold-out data: 0.2668132483959198
Epoch: 82
Loss on hold-out set: 0.09993632023533185
MeanAbsoluteError value on hold-out data: 0.26961028575897217
Epoch: 83

Loss on hold-out set: 0.09823991889754931
MeanAbsoluteError value on hold-out data: 0.26715710759162903
Epoch: 84
Loss on hold-out set: 0.10314046184221903
MeanAbsoluteError value on hold-out data: 0.27066707611083984
Epoch: 85
Loss on hold-out set: 0.09300706140697003
MeanAbsoluteError value on hold-out data: 0.25282737612724304
Epoch: 86
Loss on hold-out set: 0.10007947017749151
MeanAbsoluteError value on hold-out data: 0.2636696398258209
Epoch: 87
Loss on hold-out set: 0.0970798992489775
MeanAbsoluteError value on hold-out data: 0.25882309675216675
Epoch: 88
```

```
Loss on hold-out set: 0.08809828010698159
MeanAbsoluteError value on hold-out data: 0.24777106940746307
Epoch: 89
Loss on hold-out set: 0.09572610796739657
MeanAbsoluteError value on hold-out data: 0.25937050580978394
Epoch: 90
Loss on hold-out set: 0.08622985689590375
MeanAbsoluteError value on hold-out data: 0.2491781860589981
Epoch: 91
Loss on hold-out set: 0.08625348379214605
MeanAbsoluteError value on hold-out data: 0.24286547303199768
Epoch: 92
Loss on hold-out set: 0.08302420347929002
MeanAbsoluteError value on hold-out data: 0.24284344911575317
Epoch: 93

Loss on hold-out set: 0.09518804257114728
MeanAbsoluteError value on hold-out data: 0.25627589225769043
Epoch: 94
Loss on hold-out set: 0.08226981355498235
MeanAbsoluteError value on hold-out data: 0.23424015939235687
Epoch: 95
Loss on hold-out set: 0.08426237776875496
MeanAbsoluteError value on hold-out data: 0.24335713684558868
Epoch: 96
Loss on hold-out set: 0.08422323422506452
MeanAbsoluteError value on hold-out data: 0.24398508667945862
Epoch: 97
Loss on hold-out set: 0.08199748041729132
MeanAbsoluteError value on hold-out data: 0.2396218329668045
Epoch: 98

Loss on hold-out set: 0.08658086739480496
MeanAbsoluteError value on hold-out data: 0.2419225573539734
Epoch: 99
Loss on hold-out set: 0.07791185316940148
MeanAbsoluteError value on hold-out data: 0.22935229539871216
Epoch: 100
Loss on hold-out set: 0.0803685793094337
MeanAbsoluteError value on hold-out data: 0.23190602660179138
Epoch: 101
Loss on hold-out set: 0.07605655052388707
```

```
MeanAbsoluteError value on hold-out data: 0.23193766176700592
Epoch: 102
Loss on hold-out set: 0.07649150760223468
MeanAbsoluteError value on hold-out data: 0.23191498219966888
Epoch: 103

Loss on hold-out set: 0.08397958517695467
MeanAbsoluteError value on hold-out data: 0.2359907329082489
Epoch: 104
Loss on hold-out set: 0.077111623895665
MeanAbsoluteError value on hold-out data: 0.22659635543823242
Epoch: 105
Loss on hold-out set: 0.07276829863588015
MeanAbsoluteError value on hold-out data: 0.22255243360996246
Epoch: 106
Loss on hold-out set: 0.07879400879144668
MeanAbsoluteError value on hold-out data: 0.22746025025844574
Epoch: 107
Loss on hold-out set: 0.07630424397687117
MeanAbsoluteError value on hold-out data: 0.22751638293266296
Epoch: 108

Loss on hold-out set: 0.0792791124433279
MeanAbsoluteError value on hold-out data: 0.22953328490257263
Epoch: 109
Loss on hold-out set: 0.07158532097004354
MeanAbsoluteError value on hold-out data: 0.21731793880462646
Epoch: 110
Loss on hold-out set: 0.07947624495873849
MeanAbsoluteError value on hold-out data: 0.22985562682151794
Epoch: 111
Loss on hold-out set: 0.06677537812540929
MeanAbsoluteError value on hold-out data: 0.2093600630760193
Epoch: 112
Loss on hold-out set: 0.071478276476264
MeanAbsoluteError value on hold-out data: 0.21746325492858887
Epoch: 113

Loss on hold-out set: 0.06990106524899602
MeanAbsoluteError value on hold-out data: 0.21337765455245972
Epoch: 114
Loss on hold-out set: 0.0718095480899016
```

```
MeanAbsoluteError value on hold-out data: 0.2193876951932907
Epoch: 115
Loss on hold-out set: 0.07332947704941034
MeanAbsoluteError value on hold-out data: 0.21875900030136108
Epoch: 116
Loss on hold-out set: 0.06843494043995936
MeanAbsoluteError value on hold-out data: 0.21536873281002045
Epoch: 117

Loss on hold-out set: 0.06432858354101578
MeanAbsoluteError value on hold-out data: 0.20941367745399475
Epoch: 118
Loss on hold-out set: 0.06334384606064608
MeanAbsoluteError value on hold-out data: 0.2012060284614563
Epoch: 119
Loss on hold-out set: 0.06748024961600702
MeanAbsoluteError value on hold-out data: 0.21410349011421204
Epoch: 120
Loss on hold-out set: 0.06333447171996037
MeanAbsoluteError value on hold-out data: 0.20676039159297943
Epoch: 121
Loss on hold-out set: 0.06561410998925567
MeanAbsoluteError value on hold-out data: 0.20094646513462067
Epoch: 122

Loss on hold-out set: 0.06104587489118179
MeanAbsoluteError value on hold-out data: 0.1982167661190033
Epoch: 123
Loss on hold-out set: 0.06032568318148454
MeanAbsoluteError value on hold-out data: 0.19743193686008453
Epoch: 124
Loss on hold-out set: 0.06031870715475331
MeanAbsoluteError value on hold-out data: 0.19929742813110352
Epoch: 125
Loss on hold-out set: 0.06106611478763322
MeanAbsoluteError value on hold-out data: 0.19731612503528595
Epoch: 126
Loss on hold-out set: 0.06027580550250908
MeanAbsoluteError value on hold-out data: 0.19914090633392334
Epoch: 127

Loss on hold-out set: 0.06149007200573881
```

```
MeanAbsoluteError value on hold-out data: 0.20501650869846344
Epoch: 128
Loss on hold-out set: 0.063074473614494
MeanAbsoluteError value on hold-out data: 0.20795948803424835
Returned to Spot: Validation loss: 0.063074473614494
-----------------------------------------------

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.13021660839652088, 'lr_mult':
Epoch: 1
Loss on hold-out set: 0.2639324388613826
MeanAbsoluteError value on hold-out data: 0.48437023162841797
Epoch: 2
Loss on hold-out set: 0.2634713130169793
MeanAbsoluteError value on hold-out data: 0.48457300662994385
Epoch: 3
Loss on hold-out set: 0.2615967547815097
MeanAbsoluteError value on hold-out data: 0.4829813539981842
Epoch: 4
Loss on hold-out set: 0.2636269372152655
MeanAbsoluteError value on hold-out data: 0.48377859592437744
Epoch: 5
Loss on hold-out set: 0.2616237929384959
MeanAbsoluteError value on hold-out data: 0.4832559823989868
Epoch: 6
Loss on hold-out set: 0.26511593672790024
MeanAbsoluteError value on hold-out data: 0.4848185181617737
Epoch: 7
Loss on hold-out set: 0.2597030505145851
MeanAbsoluteError value on hold-out data: 0.4807102382183075
Epoch: 8

Loss on hold-out set: 0.2616487841464971
MeanAbsoluteError value on hold-out data: 0.4830818772315979
Epoch: 9
Loss on hold-out set: 0.25892038662966926
MeanAbsoluteError value on hold-out data: 0.4800698757171631
Epoch: 10
Loss on hold-out set: 0.2629739438232623
MeanAbsoluteError value on hold-out data: 0.484463095664978
Epoch: 11
Loss on hold-out set: 0.2594735308697349
MeanAbsoluteError value on hold-out data: 0.48006561398506165
```

```
Epoch: 12
Loss on hold-out set: 0.25823199650958967
MeanAbsoluteError value on hold-out data: 0.4787323474884033
Epoch: 13
Loss on hold-out set: 0.259520694025253
MeanAbsoluteError value on hold-out data: 0.4807213842868805
Epoch: 14
Loss on hold-out set: 0.2604811173912726
MeanAbsoluteError value on hold-out data: 0.4809702932834625
Epoch: 15
Loss on hold-out set: 0.2615290307684949
MeanAbsoluteError value on hold-out data: 0.48231154680252075
Epoch: 16
Loss on hold-out set: 0.26101045349710866
MeanAbsoluteError value on hold-out data: 0.48096781969070435
Epoch: 17
Loss on hold-out set: 0.26143188343236323
MeanAbsoluteError value on hold-out data: 0.4823537766933441
Epoch: 18
Loss on hold-out set: 0.2584050168332301
MeanAbsoluteError value on hold-out data: 0.47964608669281006
Epoch: 19

Loss on hold-out set: 0.2598672973874368
MeanAbsoluteError value on hold-out data: 0.48101556301116943
Epoch: 20
Loss on hold-out set: 0.26087434256547376
MeanAbsoluteError value on hold-out data: 0.4817075729370117
Epoch: 21
Loss on hold-out set: 0.2585053026284042
MeanAbsoluteError value on hold-out data: 0.47889816761016846
Epoch: 22
Loss on hold-out set: 0.2567489982435578
MeanAbsoluteError value on hold-out data: 0.47830936312675476
Epoch: 23
Loss on hold-out set: 0.2583339492741384
MeanAbsoluteError value on hold-out data: 0.4790361523628235
Epoch: 24
Loss on hold-out set: 0.2599623207198946
MeanAbsoluteError value on hold-out data: 0.48015183210372925
Epoch: 25
Loss on hold-out set: 0.25819750816414233
```

```
MeanAbsoluteError value on hold-out data: 0.47915253043174744
Epoch: 26
Loss on hold-out set: 0.25960876322106313
MeanAbsoluteError value on hold-out data: 0.4795624315738678
Epoch: 27
Loss on hold-out set: 0.2588841438685593
MeanAbsoluteError value on hold-out data: 0.4798726439476013
Epoch: 28
Loss on hold-out set: 0.25832232911335795
MeanAbsoluteError value on hold-out data: 0.4789051413536072
Epoch: 29
Loss on hold-out set: 0.2571862175276405
MeanAbsoluteError value on hold-out data: 0.47781163454055786
Epoch: 30

Loss on hold-out set: 0.25970564429697235
MeanAbsoluteError value on hold-out data: 0.48076915740966797
Epoch: 31
Loss on hold-out set: 0.2577187771859922
MeanAbsoluteError value on hold-out data: 0.4786440134048462
Epoch: 32
Loss on hold-out set: 0.25728018856362295
MeanAbsoluteError value on hold-out data: 0.4789002537727356
Epoch: 33
Loss on hold-out set: 0.25411353358312655
MeanAbsoluteError value on hold-out data: 0.4757460057735443
Epoch: 34
Loss on hold-out set: 0.258507801121787
MeanAbsoluteError value on hold-out data: 0.4785971939563751
Epoch: 35
Loss on hold-out set: 0.25700664383016136
MeanAbsoluteError value on hold-out data: 0.47689148783683777
Epoch: 36
Loss on hold-out set: 0.25779773961556585
MeanAbsoluteError value on hold-out data: 0.47821298241615295
Epoch: 37
Loss on hold-out set: 0.25674891530683164
MeanAbsoluteError value on hold-out data: 0.4778323471546173
Epoch: 38
Loss on hold-out set: 0.26013636040060145
MeanAbsoluteError value on hold-out data: 0.48007726669311523
Epoch: 39
```

```
Loss on hold-out set: 0.2558904735273437
MeanAbsoluteError value on hold-out data: 0.4762612283229828
Epoch: 40
Loss on hold-out set: 0.255133505910635
MeanAbsoluteError value on hold-out data: 0.4760981798171997
Epoch: 41

Loss on hold-out set: 0.25687148520036746
MeanAbsoluteError value on hold-out data: 0.4777144491672516
Epoch: 42
Loss on hold-out set: 0.2545916316540618
MeanAbsoluteError value on hold-out data: 0.4755186438560486
Epoch: 43
Loss on hold-out set: 0.2575070107061612
MeanAbsoluteError value on hold-out data: 0.47882914543151855
Epoch: 44
Loss on hold-out set: 0.25672067133219617
MeanAbsoluteError value on hold-out data: 0.47856059670448303
Epoch: 45
Loss on hold-out set: 0.25691268298971026
MeanAbsoluteError value on hold-out data: 0.4777621328830719
Epoch: 46
Loss on hold-out set: 0.25455863616968455
MeanAbsoluteError value on hold-out data: 0.47596898674964905
Epoch: 47
Loss on hold-out set: 0.2576640189478272
MeanAbsoluteError value on hold-out data: 0.478261262178421
Epoch: 48
Loss on hold-out set: 0.25672328727025734
MeanAbsoluteError value on hold-out data: 0.477157860994339
Epoch: 49
Loss on hold-out set: 0.2564741208365089
MeanAbsoluteError value on hold-out data: 0.4769105017185211
Epoch: 50
Loss on hold-out set: 0.25835686118194934
MeanAbsoluteError value on hold-out data: 0.47822967171669006
Epoch: 51
Loss on hold-out set: 0.2561713962962753
MeanAbsoluteError value on hold-out data: 0.476950466632843
Epoch: 52

Loss on hold-out set: 0.2540683942405801
```

```
MeanAbsoluteError value on hold-out data: 0.4754332900047302
Epoch: 53
Loss on hold-out set: 0.25647347028318207
MeanAbsoluteError value on hold-out data: 0.47709524631500244
Epoch: 54
Loss on hold-out set: 0.2542493410016361
MeanAbsoluteError value on hold-out data: 0.47522130608558655
Epoch: 55
Loss on hold-out set: 0.2557508221974498
MeanAbsoluteError value on hold-out data: 0.4765438735485077
Epoch: 56
Loss on hold-out set: 0.25533787729708773
MeanAbsoluteError value on hold-out data: 0.4753926694393158
Epoch: 57
Loss on hold-out set: 0.25564748342884214
MeanAbsoluteError value on hold-out data: 0.47617849707603455
Epoch: 58
Loss on hold-out set: 0.25544626383404984
MeanAbsoluteError value on hold-out data: 0.4763234853744507
Epoch: 59
Loss on hold-out set: 0.2542192079126835
MeanAbsoluteError value on hold-out data: 0.47420451045036316
Epoch: 60
Loss on hold-out set: 0.25364009036045326
MeanAbsoluteError value on hold-out data: 0.4743504226207733
Epoch: 61
Loss on hold-out set: 0.25670364165776655
MeanAbsoluteError value on hold-out data: 0.4761832058429718
Epoch: 62
Loss on hold-out set: 0.2565439298356834
MeanAbsoluteError value on hold-out data: 0.47704941034317017
Epoch: 63

Loss on hold-out set: 0.25452713844807523
MeanAbsoluteError value on hold-out data: 0.4761911630630493
Epoch: 64
Loss on hold-out set: 0.2556167999772649
MeanAbsoluteError value on hold-out data: 0.47669315338134766
Epoch: 65
Loss on hold-out set: 0.25678834632823344
MeanAbsoluteError value on hold-out data: 0.47747012972831726
Epoch: 66
```

```
Loss on hold-out set: 0.2543791318802457
MeanAbsoluteError value on hold-out data: 0.47490519285202026
Epoch: 67
Loss on hold-out set: 0.2562155162817554
MeanAbsoluteError value on hold-out data: 0.4769498109817505
Epoch: 68
Loss on hold-out set: 0.25427782437519025
MeanAbsoluteError value on hold-out data: 0.47597742080688477
Epoch: 69
Loss on hold-out set: 0.25524479895830154
MeanAbsoluteError value on hold-out data: 0.4759594798088074
Epoch: 70
Loss on hold-out set: 0.25414470014603513
MeanAbsoluteError value on hold-out data: 0.4748913049697876
Epoch: 71
Loss on hold-out set: 0.2536251758666415
MeanAbsoluteError value on hold-out data: 0.47502100467681885
Epoch: 72
Loss on hold-out set: 0.2569921799004078
MeanAbsoluteError value on hold-out data: 0.47662830352783203
Epoch: 73
Loss on hold-out set: 0.25334662414695086
MeanAbsoluteError value on hold-out data: 0.47425323724746704
Epoch: 74

Loss on hold-out set: 0.25198443626102646
MeanAbsoluteError value on hold-out data: 0.47304779291152954
Epoch: 75
Loss on hold-out set: 0.2517001891606732
MeanAbsoluteError value on hold-out data: 0.4726329445838928
Epoch: 76
Loss on hold-out set: 0.2556682378053665
MeanAbsoluteError value on hold-out data: 0.4768317639827728
Epoch: 77
Loss on hold-out set: 0.2533158275641893
MeanAbsoluteError value on hold-out data: 0.4743044674396515
Epoch: 78
Loss on hold-out set: 0.251286835066582
MeanAbsoluteError value on hold-out data: 0.47277531027793884
Epoch: 79
Loss on hold-out set: 0.25475589421234635
MeanAbsoluteError value on hold-out data: 0.4745804965496063
```

```
Epoch: 80
Loss on hold-out set: 0.2555964089145786
MeanAbsoluteError value on hold-out data: 0.4765945374965668
Epoch: 81
Loss on hold-out set: 0.2560101914170541
MeanAbsoluteError value on hold-out data: 0.476261705160141
Epoch: 82
Loss on hold-out set: 0.25842080422137914
MeanAbsoluteError value on hold-out data: 0.47894740104675293
Epoch: 83
Loss on hold-out set: 0.2544900320078197
MeanAbsoluteError value on hold-out data: 0.4754592776298523
Epoch: 84
Loss on hold-out set: 0.2548627714185338
MeanAbsoluteError value on hold-out data: 0.4755135178565979
Epoch: 85

Loss on hold-out set: 0.2536848728594027
MeanAbsoluteError value on hold-out data: 0.47415396571159363
Epoch: 86
Loss on hold-out set: 0.2519437453072322
MeanAbsoluteError value on hold-out data: 0.4729183316230774
Epoch: 87
Loss on hold-out set: 0.2523815067190873
MeanAbsoluteError value on hold-out data: 0.472848504781723
Epoch: 88
Loss on hold-out set: 0.2525838762521744
MeanAbsoluteError value on hold-out data: 0.4732861816883087
Epoch: 89
Loss on hold-out set: 0.25406626298239354
MeanAbsoluteError value on hold-out data: 0.4748392701148987
Epoch: 90
Loss on hold-out set: 0.25238682015946035
MeanAbsoluteError value on hold-out data: 0.47254693508148193
Epoch: 91
Loss on hold-out set: 0.2510310666341531
MeanAbsoluteError value on hold-out data: 0.47108379006385803
Epoch: 92
Loss on hold-out set: 0.2544778299174811
MeanAbsoluteError value on hold-out data: 0.47467726469039917
Epoch: 93
Loss on hold-out set: 0.2497213925970228
```

```
MeanAbsoluteError value on hold-out data: 0.4703372120857239
Epoch: 94
Loss on hold-out set: 0.2554997078290111
MeanAbsoluteError value on hold-out data: 0.47617894411087036
Epoch: 95
Loss on hold-out set: 0.25440571849283417
MeanAbsoluteError value on hold-out data: 0.4754246473312378
Epoch: 96

Loss on hold-out set: 0.25286322459578514
MeanAbsoluteError value on hold-out data: 0.47304046154022217
Epoch: 97
Loss on hold-out set: 0.25384739864813655
MeanAbsoluteError value on hold-out data: 0.47442224621772766
Epoch: 98
Loss on hold-out set: 0.2527647636046535
MeanAbsoluteError value on hold-out data: 0.4730164110660553
Epoch: 99
Loss on hold-out set: 0.2539777653781991
MeanAbsoluteError value on hold-out data: 0.4752943515777588
Epoch: 100
Loss on hold-out set: 0.2532887296064904
MeanAbsoluteError value on hold-out data: 0.47383585572242737
Epoch: 101
Loss on hold-out set: 0.2542617538649785
MeanAbsoluteError value on hold-out data: 0.4745626449584961
Epoch: 102
Loss on hold-out set: 0.2515762001276016
MeanAbsoluteError value on hold-out data: 0.472001850605011
Epoch: 103
Loss on hold-out set: 0.2531537588097547
MeanAbsoluteError value on hold-out data: 0.47385847568511963
Epoch: 104
Loss on hold-out set: 0.2522578749217485
MeanAbsoluteError value on hold-out data: 0.4720723330974579
Epoch: 105
Loss on hold-out set: 0.25430856782354805
MeanAbsoluteError value on hold-out data: 0.47495225071907043
Epoch: 106
Loss on hold-out set: 0.25104664599424914
MeanAbsoluteError value on hold-out data: 0.4721246361732483
Epoch: 107
```

```
Loss on hold-out set: 0.25290914370041145
MeanAbsoluteError value on hold-out data: 0.4733128249645233
Epoch: 108
Loss on hold-out set: 0.25129237692607076
MeanAbsoluteError value on hold-out data: 0.4720877707004547
Epoch: 109
Loss on hold-out set: 0.25214939780141177
MeanAbsoluteError value on hold-out data: 0.4721713364124298
Epoch: 110
Loss on hold-out set: 0.2526190941663165
MeanAbsoluteError value on hold-out data: 0.47329187393188477
Epoch: 111
Loss on hold-out set: 0.25005390749950157
MeanAbsoluteError value on hold-out data: 0.4706025719642639
Epoch: 112
Loss on hold-out set: 0.25348836026693644
MeanAbsoluteError value on hold-out data: 0.47290369868278503
Epoch: 113
Loss on hold-out set: 0.2502643383647266
MeanAbsoluteError value on hold-out data: 0.46998292207717896
Epoch: 114
Loss on hold-out set: 0.25207282171437617
MeanAbsoluteError value on hold-out data: 0.4723030626773834
Epoch: 115
Loss on hold-out set: 0.2497393859452323
MeanAbsoluteError value on hold-out data: 0.47130605578422546
Epoch: 116
Loss on hold-out set: 0.2493544662077176
MeanAbsoluteError value on hold-out data: 0.4690370261669159
Epoch: 117
Loss on hold-out set: 0.2492006017189277
MeanAbsoluteError value on hold-out data: 0.46966317296028137
Epoch: 118

Loss on hold-out set: 0.25284665528880923
MeanAbsoluteError value on hold-out data: 0.472145140171051
Epoch: 119
Loss on hold-out set: 0.2516872949506107
MeanAbsoluteError value on hold-out data: 0.47227123379707336
Epoch: 120
Loss on hold-out set: 0.2520520616518824
MeanAbsoluteError value on hold-out data: 0.4712560474872589
```

```
Epoch: 121
Loss on hold-out set: 0.24825991945047127
MeanAbsoluteError value on hold-out data: 0.46940627694129944
Epoch: 122
Loss on hold-out set: 0.24908085873252467
MeanAbsoluteError value on hold-out data: 0.46942585706710815
Epoch: 123
Loss on hold-out set: 0.2505586021824887
MeanAbsoluteError value on hold-out data: 0.4712493419647217
Epoch: 124
Loss on hold-out set: 0.2500702132912059
MeanAbsoluteError value on hold-out data: 0.47146284580230713
Epoch: 125
Loss on hold-out set: 0.2538038417696953
MeanAbsoluteError value on hold-out data: 0.47454580664634705
Epoch: 126
Loss on hold-out set: 0.2510569842630311
MeanAbsoluteError value on hold-out data: 0.471213161945343
Epoch: 127
Loss on hold-out set: 0.25039818804515035
MeanAbsoluteError value on hold-out data: 0.4704435169696808
Epoch: 128
Loss on hold-out set: 0.25111483449214383
MeanAbsoluteError value on hold-out data: 0.4708660840988159
Epoch: 129

Loss on hold-out set: 0.24966155149434743
MeanAbsoluteError value on hold-out data: 0.46918532252311707
Epoch: 130
Loss on hold-out set: 0.2480180165485332
MeanAbsoluteError value on hold-out data: 0.46894001960754395
Epoch: 131
Loss on hold-out set: 0.25132074697237267
MeanAbsoluteError value on hold-out data: 0.47110503911972046
Epoch: 132
Loss on hold-out set: 0.250156007707119
MeanAbsoluteError value on hold-out data: 0.4708087742328644
Epoch: 133
Loss on hold-out set: 0.2492036433204224
MeanAbsoluteError value on hold-out data: 0.469899445772171
Epoch: 134
Loss on hold-out set: 0.25316990362970454
```

```
MeanAbsoluteError value on hold-out data: 0.47345101833343506
Epoch: 135
Loss on hold-out set: 0.2492561673647479
MeanAbsoluteError value on hold-out data: 0.47087278962135315
Epoch: 136
Loss on hold-out set: 0.25011383840127993
MeanAbsoluteError value on hold-out data: 0.4702186584472656
Epoch: 137
Loss on hold-out set: 0.2524487250729611
MeanAbsoluteError value on hold-out data: 0.47184276580810547
Epoch: 138
Loss on hold-out set: 0.24910633030690646
MeanAbsoluteError value on hold-out data: 0.4700329601764679
Epoch: 139
Loss on hold-out set: 0.25102422092305987
MeanAbsoluteError value on hold-out data: 0.4720405638217926
Epoch: 140

Loss on hold-out set: 0.25132038640348536
MeanAbsoluteError value on hold-out data: 0.4706709384918213
Epoch: 141
Loss on hold-out set: 0.25252487353588404
MeanAbsoluteError value on hold-out data: 0.4721527695655823
Epoch: 142
Loss on hold-out set: 0.24768604122494398
MeanAbsoluteError value on hold-out data: 0.4679553210735321
Epoch: 143
Loss on hold-out set: 0.25308330259040784
MeanAbsoluteError value on hold-out data: 0.473336786031723
Epoch: 144
Loss on hold-out set: 0.25046989870698827
MeanAbsoluteError value on hold-out data: 0.4703955054283142
Epoch: 145
Loss on hold-out set: 0.24876310676336288
MeanAbsoluteError value on hold-out data: 0.4695795774459839
Epoch: 146
Loss on hold-out set: 0.24773496016860008
MeanAbsoluteError value on hold-out data: 0.46823379397392273
Epoch: 147
Loss on hold-out set: 0.2486464522386852
MeanAbsoluteError value on hold-out data: 0.46927276253700256
Epoch: 148
```

```
Loss on hold-out set: 0.2507334171157134
MeanAbsoluteError value on hold-out data: 0.4709569811820984
Epoch: 149
Loss on hold-out set: 0.2505361675039718
MeanAbsoluteError value on hold-out data: 0.47094473242759705
Epoch: 150
Loss on hold-out set: 0.24854824221447894
MeanAbsoluteError value on hold-out data: 0.4691561460494995
Epoch: 151

Loss on hold-out set: 0.2499728514567802
MeanAbsoluteError value on hold-out data: 0.470831036567688
Epoch: 152
Loss on hold-out set: 0.25167662826807874
MeanAbsoluteError value on hold-out data: 0.4720374047756195
Epoch: 153
Loss on hold-out set: 0.2501774798882635
MeanAbsoluteError value on hold-out data: 0.47011062502861023
Epoch: 154
Loss on hold-out set: 0.25003368897657646
MeanAbsoluteError value on hold-out data: 0.47045597434043884
Epoch: 155
Loss on hold-out set: 0.24931652922379344
MeanAbsoluteError value on hold-out data: 0.4699564576148987
Epoch: 156
Loss on hold-out set: 0.24879876681064306
MeanAbsoluteError value on hold-out data: 0.46938928961753845
Epoch: 157
Loss on hold-out set: 0.253004171934567
MeanAbsoluteError value on hold-out data: 0.47364017367362976
Epoch: 158
Loss on hold-out set: 0.247976101542774
MeanAbsoluteError value on hold-out data: 0.46950140595436096
Epoch: 159
Loss on hold-out set: 0.24829007430296196
MeanAbsoluteError value on hold-out data: 0.47109508514404297
Epoch: 160
Loss on hold-out set: 0.24809451754155912
MeanAbsoluteError value on hold-out data: 0.46830567717552185
Epoch: 161
Loss on hold-out set: 0.2486489523006113
MeanAbsoluteError value on hold-out data: 0.46825140714645386
Epoch: 162
```

```
Loss on hold-out set: 0.2509186963893865
MeanAbsoluteError value on hold-out data: 0.4719502627849579
Epoch: 163
Loss on hold-out set: 0.24995998979399078
MeanAbsoluteError value on hold-out data: 0.47014662623405457
Epoch: 164
Loss on hold-out set: 0.2476694842702464
MeanAbsoluteError value on hold-out data: 0.46823111176490784
Epoch: 165
Loss on hold-out set: 0.2467340987763907
MeanAbsoluteError value on hold-out data: 0.467565655708313
Epoch: 166
Loss on hold-out set: 0.24743181074920453
MeanAbsoluteError value on hold-out data: 0.46868768334388733
Epoch: 167
Loss on hold-out set: 0.24582919301955322
MeanAbsoluteError value on hold-out data: 0.46632710099220276
Epoch: 168
Loss on hold-out set: 0.25001639166944906
MeanAbsoluteError value on hold-out data: 0.4711545407772064
Epoch: 169
Loss on hold-out set: 0.24820444870151973
MeanAbsoluteError value on hold-out data: 0.468553364276886
Epoch: 170
Loss on hold-out set: 0.2504987232387066
MeanAbsoluteError value on hold-out data: 0.47029435634613037
Epoch: 171
Loss on hold-out set: 0.25067069244227913
MeanAbsoluteError value on hold-out data: 0.4715162217617035
Epoch: 172
Loss on hold-out set: 0.25119324557875333
MeanAbsoluteError value on hold-out data: 0.4710591733455658
Epoch: 173

Loss on hold-out set: 0.25043149978706714
MeanAbsoluteError value on hold-out data: 0.46974319219589233
Epoch: 174
Loss on hold-out set: 0.24826062274606606
MeanAbsoluteError value on hold-out data: 0.4690980613231659
Epoch: 175
Loss on hold-out set: 0.2494680107031998
MeanAbsoluteError value on hold-out data: 0.47063201665878296
```

```
Epoch: 176
Loss on hold-out set: 0.24763487514696622
MeanAbsoluteError value on hold-out data: 0.4667342007160187
Epoch: 177
Loss on hold-out set: 0.24941481963584297
MeanAbsoluteError value on hold-out data: 0.469950407743454
Epoch: 178
Loss on hold-out set: 0.24969592710074626
MeanAbsoluteError value on hold-out data: 0.47086602449417114
Epoch: 179
Loss on hold-out set: 0.249105974443649
MeanAbsoluteError value on hold-out data: 0.4701312184333801
Epoch: 180
Loss on hold-out set: 0.2502563636946051
MeanAbsoluteError value on hold-out data: 0.47084692120552063
Epoch: 181
Loss on hold-out set: 0.24708930657882439
MeanAbsoluteError value on hold-out data: 0.46770232915878296
Epoch: 182
Loss on hold-out set: 0.24739218013066994
MeanAbsoluteError value on hold-out data: 0.4675018787384033
Epoch: 183
Loss on hold-out set: 0.24651349433942846
MeanAbsoluteError value on hold-out data: 0.4673866927623749
Epoch: 184

Loss on hold-out set: 0.24970728531479836
MeanAbsoluteError value on hold-out data: 0.47052305936813354
Epoch: 185
Loss on hold-out set: 0.2475321498748503
MeanAbsoluteError value on hold-out data: 0.46738117933273315
Epoch: 186
Loss on hold-out set: 0.25085219054629926
MeanAbsoluteError value on hold-out data: 0.47023701667785645
Epoch: 187
Loss on hold-out set: 0.24622805749899462
MeanAbsoluteError value on hold-out data: 0.465495765209198
Epoch: 188
Loss on hold-out set: 0.24938182336719414
MeanAbsoluteError value on hold-out data: 0.46966874599456787
Epoch: 189
Loss on hold-out set: 0.24868144741968104
```

```
MeanAbsoluteError value on hold-out data: 0.46972304582595825
Epoch: 190
Loss on hold-out set: 0.24937471512116885
MeanAbsoluteError value on hold-out data: 0.4708900451660156
Epoch: 191
Loss on hold-out set: 0.2481945898187788
MeanAbsoluteError value on hold-out data: 0.4679047763347626
Epoch: 192
Loss on hold-out set: 0.24990777749764292
MeanAbsoluteError value on hold-out data: 0.46978825330734253
Epoch: 193
Loss on hold-out set: 0.2459958470181415
MeanAbsoluteError value on hold-out data: 0.46648797392845154
Epoch: 194
Loss on hold-out set: 0.2497328780591488
MeanAbsoluteError value on hold-out data: 0.46971970796585083
Epoch: 195

Loss on hold-out set: 0.24931129460272036
MeanAbsoluteError value on hold-out data: 0.47030431032180786
Epoch: 196
Loss on hold-out set: 0.24789543449878693
MeanAbsoluteError value on hold-out data: 0.46804139018058777
Epoch: 197
Loss on hold-out set: 0.2474577848456408
MeanAbsoluteError value on hold-out data: 0.468581885099411
Epoch: 198
Loss on hold-out set: 0.24651564520440603
MeanAbsoluteError value on hold-out data: 0.46757543087005615
Epoch: 199
Loss on hold-out set: 0.24711999493209938
MeanAbsoluteError value on hold-out data: 0.46750447154045105
Epoch: 200
Loss on hold-out set: 0.24864113977865168
MeanAbsoluteError value on hold-out data: 0.46781447529792786
Epoch: 201
Loss on hold-out set: 0.24479066109970996
MeanAbsoluteError value on hold-out data: 0.4656831920146942
Epoch: 202
Loss on hold-out set: 0.2472605226855529
MeanAbsoluteError value on hold-out data: 0.46733158826828003
Epoch: 203
```

```
Loss on hold-out set: 0.2461922702036406
MeanAbsoluteError value on hold-out data: 0.4662102162837982
Epoch: 204
Loss on hold-out set: 0.2487387631676699
MeanAbsoluteError value on hold-out data: 0.46951374411582947
Epoch: 205
Loss on hold-out set: 0.24731961755376114
MeanAbsoluteError value on hold-out data: 0.46669870615005493
Epoch: 206

Loss on hold-out set: 0.24771223158428543
MeanAbsoluteError value on hold-out data: 0.467905730009079
Epoch: 207
Loss on hold-out set: 0.24761517581186795
MeanAbsoluteError value on hold-out data: 0.4673415720462799
Epoch: 208
Loss on hold-out set: 0.24851316527316444
MeanAbsoluteError value on hold-out data: 0.4687729775905609
Epoch: 209
Loss on hold-out set: 0.24716849350615552
MeanAbsoluteError value on hold-out data: 0.4666658043861389
Epoch: 210
Loss on hold-out set: 0.24875513660280327
MeanAbsoluteError value on hold-out data: 0.469266802072525
Epoch: 211
Loss on hold-out set: 0.24656895450071284
MeanAbsoluteError value on hold-out data: 0.467636376619339
Epoch: 212
Loss on hold-out set: 0.24575007804914525
MeanAbsoluteError value on hold-out data: 0.4673366844654083
Epoch: 213
Loss on hold-out set: 0.24619686034949204
MeanAbsoluteError value on hold-out data: 0.46573832631111145
Epoch: 214
Loss on hold-out set: 0.2474566347112781
MeanAbsoluteError value on hold-out data: 0.4674990773200989
Epoch: 215
Loss on hold-out set: 0.24723461191905172
MeanAbsoluteError value on hold-out data: 0.4675959646701813
Epoch: 216
Loss on hold-out set: 0.24697656890279368
MeanAbsoluteError value on hold-out data: 0.4669192433357239
Epoch: 217
```

```
Loss on hold-out set: 0.2476490316422362
MeanAbsoluteError value on hold-out data: 0.4680917263031006
Epoch: 218
Loss on hold-out set: 0.2461376154893323
MeanAbsoluteError value on hold-out data: 0.46601155400276184
Epoch: 219
Loss on hold-out set: 0.24887445294543317
MeanAbsoluteError value on hold-out data: 0.4693630337715149
Epoch: 220
Loss on hold-out set: 0.24808124158727496
MeanAbsoluteError value on hold-out data: 0.4683052599430084
Epoch: 221
Loss on hold-out set: 0.24555340899448647
MeanAbsoluteError value on hold-out data: 0.465567946434021
Epoch: 222
Loss on hold-out set: 0.24713439886507235
MeanAbsoluteError value on hold-out data: 0.4670725464820862
Epoch: 223
Loss on hold-out set: 0.24664217449332537
MeanAbsoluteError value on hold-out data: 0.4672316014766693
Epoch: 224
Loss on hold-out set: 0.2449465325396312
MeanAbsoluteError value on hold-out data: 0.46520477533340454
Epoch: 225
Loss on hold-out set: 0.24480917285147466
MeanAbsoluteError value on hold-out data: 0.46512746810913086
Epoch: 226
Loss on hold-out set: 0.2456732709941111
MeanAbsoluteError value on hold-out data: 0.46647801995277405
Epoch: 227
Loss on hold-out set: 0.2445081464554134
MeanAbsoluteError value on hold-out data: 0.4640544652938843
Epoch: 228

Loss on hold-out set: 0.24418610746138975
MeanAbsoluteError value on hold-out data: 0.46499741077423096
Epoch: 229
Loss on hold-out set: 0.24716269028814217
MeanAbsoluteError value on hold-out data: 0.4673776626586914
Epoch: 230
Loss on hold-out set: 0.2455567546973103
MeanAbsoluteError value on hold-out data: 0.4651850759983063
```

```
Epoch: 231
Loss on hold-out set: 0.2464640962057992
MeanAbsoluteError value on hold-out data: 0.46677130460739136
Epoch: 232
Loss on hold-out set: 0.2461458034813404
MeanAbsoluteError value on hold-out data: 0.466819703578949
Epoch: 233
Loss on hold-out set: 0.24743731476758657
MeanAbsoluteError value on hold-out data: 0.4675538241863251
Epoch: 234
Loss on hold-out set: 0.24741719819997487
MeanAbsoluteError value on hold-out data: 0.46759694814682007
Epoch: 235
Loss on hold-out set: 0.24516146904543826
MeanAbsoluteError value on hold-out data: 0.4652509093284607
Epoch: 236
Loss on hold-out set: 0.24336823017189377
MeanAbsoluteError value on hold-out data: 0.4633767306804657
Epoch: 237
Loss on hold-out set: 0.2451292497939185
MeanAbsoluteError value on hold-out data: 0.46548470854759216
Epoch: 238
Loss on hold-out set: 0.2453589200189239
MeanAbsoluteError value on hold-out data: 0.46647879481315613
Epoch: 239

Loss on hold-out set: 0.24631016328930855
MeanAbsoluteError value on hold-out data: 0.4658817946910858
Epoch: 240
Loss on hold-out set: 0.24720481959612747
MeanAbsoluteError value on hold-out data: 0.46689388155937195
Epoch: 241
Loss on hold-out set: 0.24826032844813248
MeanAbsoluteError value on hold-out data: 0.4689670205116272
Epoch: 242
Loss on hold-out set: 0.24735001494225703
MeanAbsoluteError value on hold-out data: 0.46759700775146484
Epoch: 243
Loss on hold-out set: 0.24412610440662033
MeanAbsoluteError value on hold-out data: 0.4648672044277191
Epoch: 244
Loss on hold-out set: 0.2471577660425713
```

```
MeanAbsoluteError value on hold-out data: 0.4671221971511841
Epoch: 245
Loss on hold-out set: 0.24631983139797262
MeanAbsoluteError value on hold-out data: 0.46615782380104065
Epoch: 246
Loss on hold-out set: 0.24480388293925084
MeanAbsoluteError value on hold-out data: 0.4650172293186188
Epoch: 247
Loss on hold-out set: 0.2449100066564585
MeanAbsoluteError value on hold-out data: 0.4650602638721466
Epoch: 248
Loss on hold-out set: 0.24201442811049914
MeanAbsoluteError value on hold-out data: 0.4635554552078247
Epoch: 249
Loss on hold-out set: 0.2479942997819499
MeanAbsoluteError value on hold-out data: 0.4673757553100586
Epoch: 250

Loss on hold-out set: 0.24486574805096575
MeanAbsoluteError value on hold-out data: 0.4653209447860718
Epoch: 251
Loss on hold-out set: 0.24696274572297147
MeanAbsoluteError value on hold-out data: 0.46676141023635864
Epoch: 252
Loss on hold-out set: 0.24732062906811111
MeanAbsoluteError value on hold-out data: 0.46714773774147034
Epoch: 253
Loss on hold-out set: 0.24304035286370076
MeanAbsoluteError value on hold-out data: 0.4639478921890259
Epoch: 254
Loss on hold-out set: 0.24717276955121442
MeanAbsoluteError value on hold-out data: 0.46758508682250977
Epoch: 255
Loss on hold-out set: 0.2448041674337889
MeanAbsoluteError value on hold-out data: 0.46552687883377075
Epoch: 256
Loss on hold-out set: 0.24729000345656746
MeanAbsoluteError value on hold-out data: 0.4666655361652374
Returned to Spot: Validation loss: 0.24729000345656746
--------------------------------------------

config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.5015226665924828, 'lr_mult': 0
```

```
Epoch: 1
Loss on hold-out set: 0.1410671915662916
MeanAbsoluteError value on hold-out data: 0.33079585433006287
Epoch: 2
Loss on hold-out set: 0.13478896570833107
MeanAbsoluteError value on hold-out data: 0.32388177514076233
Epoch: 3
Loss on hold-out set: 0.13313535365619159
MeanAbsoluteError value on hold-out data: 0.3198830783367157
Epoch: 4
Loss on hold-out set: 0.13608358800411224
MeanAbsoluteError value on hold-out data: 0.3237777352333069
Epoch: 5
Loss on hold-out set: 0.13522770491085553
MeanAbsoluteError value on hold-out data: 0.3203718364238739
Epoch: 6
Loss on hold-out set: 0.13449430387271077
MeanAbsoluteError value on hold-out data: 0.3178567886352539
Epoch: 7

Loss on hold-out set: 0.14063767186905207
MeanAbsoluteError value on hold-out data: 0.33063411712646484
Epoch: 8
Loss on hold-out set: 0.13634374580885233
MeanAbsoluteError value on hold-out data: 0.32196930050849915
Epoch: 9
Loss on hold-out set: 0.13618269249012596
MeanAbsoluteError value on hold-out data: 0.322646826505661
Epoch: 10
Loss on hold-out set: 0.13274679372185155
MeanAbsoluteError value on hold-out data: 0.32192984223365784
Epoch: 11
Loss on hold-out set: 0.1278246856833759
MeanAbsoluteError value on hold-out data: 0.31691375374794006
Epoch: 12
Loss on hold-out set: 0.13197797144714155
MeanAbsoluteError value on hold-out data: 0.31842032074928284
Epoch: 13
Loss on hold-out set: 0.12618987654384814
MeanAbsoluteError value on hold-out data: 0.3145260810852051
Epoch: 14
Loss on hold-out set: 0.12221943253749296
```

```
MeanAbsoluteError value on hold-out data: 0.30518999695777893
Epoch: 15
Loss on hold-out set: 0.125905132215274
MeanAbsoluteError value on hold-out data: 0.314395934343338
Epoch: 16
Loss on hold-out set: 0.1347815464985998
MeanAbsoluteError value on hold-out data: 0.32415515184402466
Epoch: 17
Loss on hold-out set: 0.1327485755870217
MeanAbsoluteError value on hold-out data: 0.3203687369823456
Epoch: 18
Loss on hold-out set: 0.13381165266036987
MeanAbsoluteError value on hold-out data: 0.3217594623565674
Epoch: 19
Loss on hold-out set: 0.12663408800175316
MeanAbsoluteError value on hold-out data: 0.3114473223686218
Epoch: 20
Loss on hold-out set: 0.13032408725274236
MeanAbsoluteError value on hold-out data: 0.31565845012664795
Epoch: 21
Loss on hold-out set: 0.12484943827516154
MeanAbsoluteError value on hold-out data: 0.3114683926105499
Epoch: 22
Loss on hold-out set: 0.13155165824450946
MeanAbsoluteError value on hold-out data: 0.3157272934913635
Early stopping at epoch 21
Returned to Spot: Validation loss: 0.13155165824450946
--------------------------------------------

config: {'_L_in': 10, '_L_out': 1, 'l1': 16, 'dropout_prob': 0.26673029336651144, 'lr_mult':
Epoch: 1


Loss on hold-out set: 0.06604599869368892
MeanAbsoluteError value on hold-out data: 0.21497230231761932
Epoch: 2
Loss on hold-out set: 0.065460244487775
MeanAbsoluteError value on hold-out data: 0.2134026288986206
Epoch: 3
Loss on hold-out set: 0.06532088608333939
MeanAbsoluteError value on hold-out data: 0.21321798861026764
Epoch: 4
Loss on hold-out set: 0.06537821251702935
```

```
MeanAbsoluteError value on hold-out data: 0.21401390433311462
Epoch: 5
Loss on hold-out set: 0.06554910599401123
MeanAbsoluteError value on hold-out data: 0.2142275720834732
Epoch: 6
Loss on hold-out set: 0.06540152827571881
MeanAbsoluteError value on hold-out data: 0.21331757307052612
Epoch: 7
Loss on hold-out set: 0.06447050284202162
MeanAbsoluteError value on hold-out data: 0.21216575801372528
Epoch: 8
Loss on hold-out set: 0.06494946733705308
MeanAbsoluteError value on hold-out data: 0.21260829269886017
Epoch: 9
Loss on hold-out set: 0.06466433861734051
MeanAbsoluteError value on hold-out data: 0.2122795283794403
Epoch: 10
Loss on hold-out set: 0.06472473380793083
MeanAbsoluteError value on hold-out data: 0.21245244145393372
Epoch: 11

Loss on hold-out set: 0.06527537377061028
MeanAbsoluteError value on hold-out data: 0.21350309252738953
Epoch: 12
Loss on hold-out set: 0.06468347013977013
MeanAbsoluteError value on hold-out data: 0.21257586777210236
Epoch: 13
Loss on hold-out set: 0.06531407987993014
MeanAbsoluteError value on hold-out data: 0.21257083117961884
Epoch: 14
Loss on hold-out set: 0.06517007103876064
MeanAbsoluteError value on hold-out data: 0.2126644402742386
Epoch: 15
Loss on hold-out set: 0.06520682264511522
MeanAbsoluteError value on hold-out data: 0.21321836113929749
Epoch: 16
Loss on hold-out set: 0.06471111434266756
MeanAbsoluteError value on hold-out data: 0.2117253690958023
Returned to Spot: Validation loss: 0.06471111434266756
-----------------------------------------------

config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.8973189149831583, 'lr_mult':
```

```
Epoch: 1
Loss on hold-out set: 0.22404156485882898
MeanAbsoluteError value on hold-out data: 0.412637323141098
Epoch: 2

Loss on hold-out set: 0.22776694978276887
MeanAbsoluteError value on hold-out data: 0.41466423869132996
Epoch: 3
Loss on hold-out set: 0.2016386019748946
MeanAbsoluteError value on hold-out data: 0.3934824764728546
Epoch: 4
Loss on hold-out set: 0.20567138587357475
MeanAbsoluteError value on hold-out data: 0.39237910509109497
Epoch: 5

Loss on hold-out set: 0.2171087021753192
MeanAbsoluteError value on hold-out data: 0.3973276913166046
Epoch: 6
Loss on hold-out set: 0.20122253215561312
MeanAbsoluteError value on hold-out data: 0.38454562425613403
Epoch: 7
Loss on hold-out set: 0.2190321727267777
MeanAbsoluteError value on hold-out data: 0.400615394115448
Epoch: 8

Loss on hold-out set: 0.21349189672308663
MeanAbsoluteError value on hold-out data: 0.3994046449661255
Epoch: 9
Loss on hold-out set: 0.21194477202331957
MeanAbsoluteError value on hold-out data: 0.40040576457977295
Epoch: 10
Loss on hold-out set: 0.20910393917312226
MeanAbsoluteError value on hold-out data: 0.4006097614765167
Epoch: 11

Loss on hold-out set: 0.21250962369764845
MeanAbsoluteError value on hold-out data: 0.40513014793395996
Epoch: 12
Loss on hold-out set: 0.2000627984229747
MeanAbsoluteError value on hold-out data: 0.3830963671207428
Epoch: 13
```

```
Loss on hold-out set: 0.1847788740694523
MeanAbsoluteError value on hold-out data: 0.36858317255973816
Epoch: 14

Loss on hold-out set: 0.17953441518785743
MeanAbsoluteError value on hold-out data: 0.362382709980011
Epoch: 15
Loss on hold-out set: 0.20048382246556382
MeanAbsoluteError value on hold-out data: 0.38669121265411377
Epoch: 16
Loss on hold-out set: 0.19411724289238919
MeanAbsoluteError value on hold-out data: 0.3836238384246826
Epoch: 17

Loss on hold-out set: 0.2028149135170194
MeanAbsoluteError value on hold-out data: 0.3824140131473541
Epoch: 18
Loss on hold-out set: 0.18551118124858476
MeanAbsoluteError value on hold-out data: 0.37134724855422974
Epoch: 19
Loss on hold-out set: 0.1635328991509353
MeanAbsoluteError value on hold-out data: 0.33950963616371155
Epoch: 20

Loss on hold-out set: 0.19748675830507031
MeanAbsoluteError value on hold-out data: 0.3834604322910309
Epoch: 21
Loss on hold-out set: 0.17135605891099354
MeanAbsoluteError value on hold-out data: 0.35077640414237976
Epoch: 22
Loss on hold-out set: 0.1801271416308979
MeanAbsoluteError value on hold-out data: 0.3569149672985077
Epoch: 23

Loss on hold-out set: 0.19234128168822887
MeanAbsoluteError value on hold-out data: 0.37854012846946716
Epoch: 24
Loss on hold-out set: 0.17234558017963234
MeanAbsoluteError value on hold-out data: 0.34840574860572815
Epoch: 25
Loss on hold-out set: 0.15814285522326826
```

```
MeanAbsoluteError value on hold-out data: 0.3363674283027649
Epoch: 26

Loss on hold-out set: 0.17979837934486567
MeanAbsoluteError value on hold-out data: 0.36273932456970215
Epoch: 27
Loss on hold-out set: 0.15909611174991975
MeanAbsoluteError value on hold-out data: 0.3417477607727051
Epoch: 28
Loss on hold-out set: 0.17337580399820582
MeanAbsoluteError value on hold-out data: 0.35184669494628906
Epoch: 29

Loss on hold-out set: 0.1687521711839751
MeanAbsoluteError value on hold-out data: 0.3471502661705017
Epoch: 30
Loss on hold-out set: 0.1732332039876686
MeanAbsoluteError value on hold-out data: 0.3467940390110016
Epoch: 31
Loss on hold-out set: 0.16189115307138613
MeanAbsoluteError value on hold-out data: 0.33892822265625
Epoch: 32

Loss on hold-out set: 0.16924173519636193
MeanAbsoluteError value on hold-out data: 0.347928524017334
Epoch: 33
Loss on hold-out set: 0.1583472756606837
MeanAbsoluteError value on hold-out data: 0.33969950675964355
Epoch: 34
Loss on hold-out set: 0.16944172215027115
MeanAbsoluteError value on hold-out data: 0.35305243730545044
Epoch: 35

Loss on hold-out set: 0.1578676366613945
MeanAbsoluteError value on hold-out data: 0.330411821603775
Epoch: 36
Loss on hold-out set: 0.16565377144686258
MeanAbsoluteError value on hold-out data: 0.3450448215007782
Epoch: 37
Loss on hold-out set: 0.15041746285627597
MeanAbsoluteError value on hold-out data: 0.3235618472099304
Epoch: 38
```

```
Loss on hold-out set: 0.16742136531664678
MeanAbsoluteError value on hold-out data: 0.34965983033180237
Epoch: 39
Loss on hold-out set: 0.15540596014199157
MeanAbsoluteError value on hold-out data: 0.33668458461761475
Epoch: 40
Loss on hold-out set: 0.1538784485589713
MeanAbsoluteError value on hold-out data: 0.3339499235153198
Epoch: 41

Loss on hold-out set: 0.16377454174954134
MeanAbsoluteError value on hold-out data: 0.3346950113773346
Epoch: 42
Loss on hold-out set: 0.15585448413000752
MeanAbsoluteError value on hold-out data: 0.34216389060020447
Epoch: 43
Loss on hold-out set: 0.1465975175627197
MeanAbsoluteError value on hold-out data: 0.3178761899471283
Epoch: 44

Loss on hold-out set: 0.152155380093803
MeanAbsoluteError value on hold-out data: 0.3271729648113251
Epoch: 45
Loss on hold-out set: 0.14098388589249225
MeanAbsoluteError value on hold-out data: 0.3098580539226532
Epoch: 46
Loss on hold-out set: 0.15275696955170134
MeanAbsoluteError value on hold-out data: 0.32055115699768066
Epoch: 47

Loss on hold-out set: 0.16835569199873135
MeanAbsoluteError value on hold-out data: 0.34751448035240173
Epoch: 48
Loss on hold-out set: 0.14503629238189508
MeanAbsoluteError value on hold-out data: 0.32180628180503845
Epoch: 49
Loss on hold-out set: 0.12559846231558672
MeanAbsoluteError value on hold-out data: 0.294494092464447
Epoch: 50

Loss on hold-out set: 0.1308622001344338
```

```
MeanAbsoluteError value on hold-out data: 0.29798075556755066
Epoch: 51
Loss on hold-out set: 0.1432399746403098
MeanAbsoluteError value on hold-out data: 0.321303129196167
Epoch: 52
Loss on hold-out set: 0.1338730143965222
MeanAbsoluteError value on hold-out data: 0.3093910217285156
Epoch: 53

Loss on hold-out set: 0.1374383661112127
MeanAbsoluteError value on hold-out data: 0.31263259053230286
Epoch: 54
Loss on hold-out set: 0.134709902285443
MeanAbsoluteError value on hold-out data: 0.30623894929885864
Epoch: 55
Loss on hold-out set: 0.13250004183365188
MeanAbsoluteError value on hold-out data: 0.30750060081481934
Epoch: 56

Loss on hold-out set: 0.13722650951502147
MeanAbsoluteError value on hold-out data: 0.31358590722084045
Epoch: 57
Loss on hold-out set: 0.14126891683864717
MeanAbsoluteError value on hold-out data: 0.31877458095550537
Epoch: 58
Loss on hold-out set: 0.12187117731664329
MeanAbsoluteError value on hold-out data: 0.288463830947876
Epoch: 59

Loss on hold-out set: 0.12237007032303761
MeanAbsoluteError value on hold-out data: 0.2885819673538208
Epoch: 60
Loss on hold-out set: 0.13949372503906488
MeanAbsoluteError value on hold-out data: 0.31758567690849304
Epoch: 61
Loss on hold-out set: 0.12927211046775483
MeanAbsoluteError value on hold-out data: 0.30522266030311584
Epoch: 62

Loss on hold-out set: 0.14353746491484343
MeanAbsoluteError value on hold-out data: 0.31941336393356323
```

```
Epoch: 63
Loss on hold-out set: 0.13998679891383897
MeanAbsoluteError value on hold-out data: 0.3134731650352478
Epoch: 64
Loss on hold-out set: 0.12784535079825823
MeanAbsoluteError value on hold-out data: 0.2948179543018341
Epoch: 65

Loss on hold-out set: 0.13743032383034007
MeanAbsoluteError value on hold-out data: 0.3066257834434509
Epoch: 66
Loss on hold-out set: 0.13390130570565817
MeanAbsoluteError value on hold-out data: 0.3054691553115845
Epoch: 67
Loss on hold-out set: 0.13803479260609797
MeanAbsoluteError value on hold-out data: 0.31143468618392944
Epoch: 68

Loss on hold-out set: 0.12503444292601973
MeanAbsoluteError value on hold-out data: 0.29563766717910767
Epoch: 69
Loss on hold-out set: 0.11757809984730556
MeanAbsoluteError value on hold-out data: 0.2812201976776123
Epoch: 70
Loss on hold-out set: 0.13828418103978038
MeanAbsoluteError value on hold-out data: 0.31446170806884766
Epoch: 71

Loss on hold-out set: 0.13190906826988794
MeanAbsoluteError value on hold-out data: 0.3055327832698822
Epoch: 72
Loss on hold-out set: 0.11537868549736838
MeanAbsoluteError value on hold-out data: 0.2829883098602295
Epoch: 73
Loss on hold-out set: 0.12645787791038554
MeanAbsoluteError value on hold-out data: 0.2988964021205902
Epoch: 74

Loss on hold-out set: 0.11892737013675893
MeanAbsoluteError value on hold-out data: 0.2895407974720001
Epoch: 75
```

```
Loss on hold-out set: 0.11811398725704446
MeanAbsoluteError value on hold-out data: 0.28719276189804077
Epoch: 76
Loss on hold-out set: 0.122326161005379
MeanAbsoluteError value on hold-out data: 0.3001905679702759
Epoch: 77

Loss on hold-out set: 0.11973346952038506
MeanAbsoluteError value on hold-out data: 0.28488388657569885
Epoch: 78
Loss on hold-out set: 0.13061620773289784
MeanAbsoluteError value on hold-out data: 0.29863572120666504
Epoch: 79
Loss on hold-out set: 0.11208983293695686
MeanAbsoluteError value on hold-out data: 0.2760583758354187
Epoch: 80

Loss on hold-out set: 0.11998382431590775
MeanAbsoluteError value on hold-out data: 0.2920362651348114
Epoch: 81
Loss on hold-out set: 0.11321465723439662
MeanAbsoluteError value on hold-out data: 0.2807960510253906
Epoch: 82
Loss on hold-out set: 0.12125698070817938
MeanAbsoluteError value on hold-out data: 0.2904837429523468
Epoch: 83

Loss on hold-out set: 0.12422018694206297
MeanAbsoluteError value on hold-out data: 0.2912091314792633
Epoch: 84
Loss on hold-out set: 0.12065764325981339
MeanAbsoluteError value on hold-out data: 0.2844049632549286
Epoch: 85
Loss on hold-out set: 0.1226528528041672
MeanAbsoluteError value on hold-out data: 0.29117703437805176
Epoch: 86

Loss on hold-out set: 0.12191949461664384
MeanAbsoluteError value on hold-out data: 0.28771260380744934
Epoch: 87
Loss on hold-out set: 0.11318143405020237
```

```
MeanAbsoluteError value on hold-out data: 0.2806585133075714
Epoch: 88
Loss on hold-out set: 0.1272610796900699
MeanAbsoluteError value on hold-out data: 0.3017241358757019
Epoch: 89

Loss on hold-out set: 0.11953043696548168
MeanAbsoluteError value on hold-out data: 0.28103554248809814
Epoch: 90
Loss on hold-out set: 0.11300939704912404
MeanAbsoluteError value on hold-out data: 0.27412909269332886
Epoch: 91
Loss on hold-out set: 0.1157905940579561
MeanAbsoluteError value on hold-out data: 0.27436867356300354
Epoch: 92

Loss on hold-out set: 0.10579982205793688
MeanAbsoluteError value on hold-out data: 0.2588527798652649
Epoch: 93
Loss on hold-out set: 0.11635770030940572
MeanAbsoluteError value on hold-out data: 0.27819153666496277
Epoch: 94
Loss on hold-out set: 0.11328297316290749
MeanAbsoluteError value on hold-out data: 0.2775765657424927
Epoch: 95

Loss on hold-out set: 0.11794726703471194
MeanAbsoluteError value on hold-out data: 0.2824779152870178
Epoch: 96
Loss on hold-out set: 0.11397721773789575
MeanAbsoluteError value on hold-out data: 0.27995675802230835
Epoch: 97
Loss on hold-out set: 0.10400607630610466
MeanAbsoluteError value on hold-out data: 0.2609563171863556
Epoch: 98

Loss on hold-out set: 0.11228992450710697
MeanAbsoluteError value on hold-out data: 0.27215757966041565
Epoch: 99
Loss on hold-out set: 0.11649084342643619
MeanAbsoluteError value on hold-out data: 0.278969943523407
```

```
Epoch: 100
Loss on hold-out set: 0.11824412230712672
MeanAbsoluteError value on hold-out data: 0.2803994119167328
Epoch: 101

Loss on hold-out set: 0.11750103653854846
MeanAbsoluteError value on hold-out data: 0.2768651843070984
Epoch: 102
Loss on hold-out set: 0.0939032159394507
MeanAbsoluteError value on hold-out data: 0.25298836827278137
Epoch: 103
Loss on hold-out set: 0.11752992547388809
MeanAbsoluteError value on hold-out data: 0.28222203254699707
Epoch: 104

Loss on hold-out set: 0.11279095479287207
MeanAbsoluteError value on hold-out data: 0.2750902771949768
Epoch: 105
Loss on hold-out set: 0.09544096622033976
MeanAbsoluteError value on hold-out data: 0.2588951885700226
Epoch: 106
Loss on hold-out set: 0.09943254936486483
MeanAbsoluteError value on hold-out data: 0.25387299060821533
Epoch: 107

Loss on hold-out set: 0.09526281107178268
MeanAbsoluteError value on hold-out data: 0.2543632388114929
Epoch: 108
Loss on hold-out set: 0.10156229859179196
MeanAbsoluteError value on hold-out data: 0.2595876455307007
Epoch: 109
Loss on hold-out set: 0.1066277161935189
MeanAbsoluteError value on hold-out data: 0.2704138457775116
Epoch: 110

Loss on hold-out set: 0.10989889729418792
MeanAbsoluteError value on hold-out data: 0.2692160904407501
Epoch: 111
Loss on hold-out set: 0.10330435901104162
MeanAbsoluteError value on hold-out data: 0.265007346868515
Epoch: 112
```

```
Loss on hold-out set: 0.11210621985994901
MeanAbsoluteError value on hold-out data: 0.27691566944122314
Epoch: 113

Loss on hold-out set: 0.11704418513819595
MeanAbsoluteError value on hold-out data: 0.2787482738494873
Epoch: 114
Loss on hold-out set: 0.09966777502500918
MeanAbsoluteError value on hold-out data: 0.24953605234622955
Epoch: 115
Loss on hold-out set: 0.10665488181092465
MeanAbsoluteError value on hold-out data: 0.26766934990882874
Epoch: 116

Loss on hold-out set: 0.10509142223978415
MeanAbsoluteError value on hold-out data: 0.26527416706085205
Epoch: 117
Loss on hold-out set: 0.10165416267467663
MeanAbsoluteError value on hold-out data: 0.2666526138782501
Epoch: 118
Loss on hold-out set: 0.10044858965169018
MeanAbsoluteError value on hold-out data: 0.2595323920249939
Epoch: 119

Loss on hold-out set: 0.10022886843536981
MeanAbsoluteError value on hold-out data: 0.2555849552154541
Epoch: 120
Loss on hold-out set: 0.10455682050669565
MeanAbsoluteError value on hold-out data: 0.261140376329422
Epoch: 121
Loss on hold-out set: 0.11374881892154615
MeanAbsoluteError value on hold-out data: 0.2683655321598053
Epoch: 122

Loss on hold-out set: 0.10166444218106335
MeanAbsoluteError value on hold-out data: 0.2493475079536438
Epoch: 123
Loss on hold-out set: 0.10813046961169069
MeanAbsoluteError value on hold-out data: 0.2604410946369171
Epoch: 124
Loss on hold-out set: 0.1054120683427512
```

```
MeanAbsoluteError value on hold-out data: 0.2688944637775421
Epoch: 125

Loss on hold-out set: 0.10533409979281715
MeanAbsoluteError value on hold-out data: 0.2617444396018982
Epoch: 126
Loss on hold-out set: 0.11213329801801591
MeanAbsoluteError value on hold-out data: 0.26898500323295593
Epoch: 127
Loss on hold-out set: 0.10047917034011335
MeanAbsoluteError value on hold-out data: 0.2557356655597687
Epoch: 128

Loss on hold-out set: 0.09892099501856137
MeanAbsoluteError value on hold-out data: 0.25566866993904114
Epoch: 129
Loss on hold-out set: 0.10305958282396507
MeanAbsoluteError value on hold-out data: 0.260862797498703
Epoch: 130
Loss on hold-out set: 0.10715558507169286
MeanAbsoluteError value on hold-out data: 0.27106529474258423
Epoch: 131

Loss on hold-out set: 0.10615401924044515
MeanAbsoluteError value on hold-out data: 0.2672101557254791
Epoch: 132
Loss on hold-out set: 0.11169146517398379
MeanAbsoluteError value on hold-out data: 0.2734561860561371
Epoch: 133
Loss on hold-out set: 0.10682077177489797
MeanAbsoluteError value on hold-out data: 0.26618921756744385
Epoch: 134

Loss on hold-out set: 0.11410828244324268
MeanAbsoluteError value on hold-out data: 0.26617422699928284
Early stopping at epoch 133
Returned to Spot: Validation loss: 0.11410828244324268
--------------------------------------------

config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7126337500646852, 'lr_mult':
Epoch: 1
```

```
Loss on hold-out set: 0.39507770975430806
MeanAbsoluteError value on hold-out data: 0.5988824963569641
Epoch: 2

Loss on hold-out set: 0.3953638184070587
MeanAbsoluteError value on hold-out data: 0.6000440716743469
Epoch: 3
Loss on hold-out set: 0.39339575270811716
MeanAbsoluteError value on hold-out data: 0.5933763384819031
Epoch: 4
Loss on hold-out set: 0.38936877886454263
MeanAbsoluteError value on hold-out data: 0.5936213731765747
Epoch: 5
Loss on hold-out set: 0.4061631727218628
MeanAbsoluteError value on hold-out data: 0.6082262396812439
Epoch: 6
Loss on hold-out set: 0.38503593623638155
MeanAbsoluteError value on hold-out data: 0.5886045098304749
Epoch: 7

Loss on hold-out set: 0.36551714460055035
MeanAbsoluteError value on hold-out data: 0.5703389048576355
Epoch: 8
Loss on hold-out set: 0.3915012530485789
MeanAbsoluteError value on hold-out data: 0.5938528776168823
Epoch: 9
Loss on hold-out set: 0.37498868107795713
MeanAbsoluteError value on hold-out data: 0.5796413421630859
Epoch: 10
Loss on hold-out set: 0.3811343896389008
MeanAbsoluteError value on hold-out data: 0.583713710308075
Epoch: 11
Loss on hold-out set: 0.3619476447502772
MeanAbsoluteError value on hold-out data: 0.5676211714744568
Epoch: 12

Loss on hold-out set: 0.34596339166164397
MeanAbsoluteError value on hold-out data: 0.5547571778297424
Epoch: 13
Loss on hold-out set: 0.357585124373436
MeanAbsoluteError value on hold-out data: 0.5659917593002319
Epoch: 14
```

```
Loss on hold-out set: 0.3578070358435313
MeanAbsoluteError value on hold-out data: 0.5639902353286743
Epoch: 15
Loss on hold-out set: 0.35575617174307506
MeanAbsoluteError value on hold-out data: 0.5646409392356873
Epoch: 16
Loss on hold-out set: 0.3483573363224665
MeanAbsoluteError value on hold-out data: 0.5578354597091675
Epoch: 17

Loss on hold-out set: 0.34386360357205076
MeanAbsoluteError value on hold-out data: 0.5521500110626221
Epoch: 18
Loss on hold-out set: 0.34377329995234807
MeanAbsoluteError value on hold-out data: 0.5532888174057007
Epoch: 19
Loss on hold-out set: 0.3386105686426163
MeanAbsoluteError value on hold-out data: 0.5507266521453857
Epoch: 20
Loss on hold-out set: 0.33882727881272634
MeanAbsoluteError value on hold-out data: 0.5485994219779968
Epoch: 21
Loss on hold-out set: 0.33881931712230046
MeanAbsoluteError value on hold-out data: 0.5497338175773621
Epoch: 22

Loss on hold-out set: 0.31970092296600344
MeanAbsoluteError value on hold-out data: 0.5314518809318542
Epoch: 23
Loss on hold-out set: 0.32202995002269746
MeanAbsoluteError value on hold-out data: 0.5355226397514343
Epoch: 24
Loss on hold-out set: 0.3161576551198959
MeanAbsoluteError value on hold-out data: 0.528671383857727
Epoch: 25
Loss on hold-out set: 0.31669414361317955
MeanAbsoluteError value on hold-out data: 0.5296686887741089
Epoch: 26
Loss on hold-out set: 0.3170734539628029
MeanAbsoluteError value on hold-out data: 0.5248983502388
Epoch: 27
```

```
Loss on hold-out set: 0.31139625281095507
MeanAbsoluteError value on hold-out data: 0.5237025022506714
Epoch: 28
Loss on hold-out set: 0.3097134663661321
MeanAbsoluteError value on hold-out data: 0.5206657648086548
Epoch: 29
Loss on hold-out set: 0.30071684410174687
MeanAbsoluteError value on hold-out data: 0.5123667120933533
Epoch: 30
Loss on hold-out set: 0.3066506376862526
MeanAbsoluteError value on hold-out data: 0.5195916295051575
Epoch: 31
Loss on hold-out set: 0.3001344390710195
MeanAbsoluteError value on hold-out data: 0.5124717950820923
Epoch: 32

Loss on hold-out set: 0.2969919590155284
MeanAbsoluteError value on hold-out data: 0.5099086761474609
Epoch: 33
Loss on hold-out set: 0.29387757231791817
MeanAbsoluteError value on hold-out data: 0.5103693604469299
Epoch: 34
Loss on hold-out set: 0.2931195126970609
MeanAbsoluteError value on hold-out data: 0.5066169500350952
Epoch: 35
Loss on hold-out set: 0.2844792874654134
MeanAbsoluteError value on hold-out data: 0.4987899661064148
Epoch: 36
Loss on hold-out set: 0.27497516562541324
MeanAbsoluteError value on hold-out data: 0.48493948578834534
Epoch: 37

Loss on hold-out set: 0.28363817046085993
MeanAbsoluteError value on hold-out data: 0.4964101016521454
Epoch: 38
Loss on hold-out set: 0.26357666711012523
MeanAbsoluteError value on hold-out data: 0.4778674244880676
Epoch: 39
Loss on hold-out set: 0.27476423382759096
MeanAbsoluteError value on hold-out data: 0.4911145567893982
Epoch: 40
Loss on hold-out set: 0.26378942469755806
```

```
MeanAbsoluteError value on hold-out data: 0.47696441411972046
Epoch: 41
Loss on hold-out set: 0.2683885767062505
MeanAbsoluteError value on hold-out data: 0.48366135358810425
Epoch: 42

Loss on hold-out set: 0.2720457008481026
MeanAbsoluteError value on hold-out data: 0.48550283908843994
Epoch: 43
Loss on hold-out set: 0.2652823080619176
MeanAbsoluteError value on hold-out data: 0.4788719713687897
Epoch: 44
Loss on hold-out set: 0.2572959613800049
MeanAbsoluteError value on hold-out data: 0.4716303050518036
Epoch: 45
Loss on hold-out set: 0.24736659983793893
MeanAbsoluteError value on hold-out data: 0.4574461877346039
Epoch: 46
Loss on hold-out set: 0.25581152309974037
MeanAbsoluteError value on hold-out data: 0.46561023592948914
Epoch: 47

Loss on hold-out set: 0.262367257575194
MeanAbsoluteError value on hold-out data: 0.47566258907318115
Epoch: 48
Loss on hold-out set: 0.2584783970316251
MeanAbsoluteError value on hold-out data: 0.4673479199409485
Epoch: 49
Loss on hold-out set: 0.2346836085120837
MeanAbsoluteError value on hold-out data: 0.4460025131702423
Epoch: 50
Loss on hold-out set: 0.23957746942838032
MeanAbsoluteError value on hold-out data: 0.4497544467449188
Epoch: 51
Loss on hold-out set: 0.23353636582692464
MeanAbsoluteError value on hold-out data: 0.443226158618927
Epoch: 52

Loss on hold-out set: 0.24561989764372508
MeanAbsoluteError value on hold-out data: 0.45198485255241394
Epoch: 53
Loss on hold-out set: 0.2315611899892489
```

```
MeanAbsoluteError value on hold-out data: 0.4446796178817749
Epoch: 54
Loss on hold-out set: 0.23580748334527016
MeanAbsoluteError value on hold-out data: 0.44304147362709045
Epoch: 55
Loss on hold-out set: 0.2220337184270223
MeanAbsoluteError value on hold-out data: 0.43313100934028625
Epoch: 56
Loss on hold-out set: 0.22745558696488538
MeanAbsoluteError value on hold-out data: 0.43342310190200806
Epoch: 57

Loss on hold-out set: 0.22346554681658745
MeanAbsoluteError value on hold-out data: 0.42995911836624146
Epoch: 58
Loss on hold-out set: 0.21600619663794834
MeanAbsoluteError value on hold-out data: 0.42415520548820496
Epoch: 59
Loss on hold-out set: 0.20966709367930889
MeanAbsoluteError value on hold-out data: 0.41753530502319336
Epoch: 60
Loss on hold-out set: 0.21662049233913422
MeanAbsoluteError value on hold-out data: 0.4283024072647095
Epoch: 61
Loss on hold-out set: 0.20301929334799448
MeanAbsoluteError value on hold-out data: 0.4080101251602173
Epoch: 62

Loss on hold-out set: 0.2103757188717524
MeanAbsoluteError value on hold-out data: 0.41945120692253113
Epoch: 63
Loss on hold-out set: 0.2107362014055252
MeanAbsoluteError value on hold-out data: 0.41602474451065063
Epoch: 64
Loss on hold-out set: 0.19931005333860716
MeanAbsoluteError value on hold-out data: 0.40372198820114136
Epoch: 65
Loss on hold-out set: 0.20577074776093165
MeanAbsoluteError value on hold-out data: 0.41163837909698486
Epoch: 66
Loss on hold-out set: 0.1952419151365757
MeanAbsoluteError value on hold-out data: 0.39635008573532104
```

```
Epoch: 67

Loss on hold-out set: 0.198982549905777
MeanAbsoluteError value on hold-out data: 0.4025513529777527
Epoch: 68
Loss on hold-out set: 0.1951264231900374
MeanAbsoluteError value on hold-out data: 0.40006735920906067
Epoch: 69
Loss on hold-out set: 0.18599100778500238
MeanAbsoluteError value on hold-out data: 0.38837358355522156
Epoch: 70
Loss on hold-out set: 0.19120520676175753
MeanAbsoluteError value on hold-out data: 0.39187607169151306
Epoch: 71
Loss on hold-out set: 0.18290302157402039
MeanAbsoluteError value on hold-out data: 0.3839729130268097
Epoch: 72

Loss on hold-out set: 0.1857138243317604
MeanAbsoluteError value on hold-out data: 0.39125680923461914
Epoch: 73
Loss on hold-out set: 0.19192776918411256
MeanAbsoluteError value on hold-out data: 0.3944156765937805
Epoch: 74
Loss on hold-out set: 0.17396517500281333
MeanAbsoluteError value on hold-out data: 0.3715391755104065
Epoch: 75
Loss on hold-out set: 0.18293635323643684
MeanAbsoluteError value on hold-out data: 0.3813750147819519
Epoch: 76
Loss on hold-out set: 0.17536496261755624
MeanAbsoluteError value on hold-out data: 0.37312933802604675
Epoch: 77

Loss on hold-out set: 0.16755012720823287
MeanAbsoluteError value on hold-out data: 0.3654266595840454
Epoch: 78
Loss on hold-out set: 0.16316596454630294
MeanAbsoluteError value on hold-out data: 0.35796451568603516
Epoch: 79
Loss on hold-out set: 0.1610545065999031
MeanAbsoluteError value on hold-out data: 0.35598450899124146
```

```
Epoch: 80
Loss on hold-out set: 0.16552235985795657
MeanAbsoluteError value on hold-out data: 0.3627857267856598
Epoch: 81
Loss on hold-out set: 0.16544341755410036
MeanAbsoluteError value on hold-out data: 0.3618793487548828
Epoch: 82

Loss on hold-out set: 0.16545906310280165
MeanAbsoluteError value on hold-out data: 0.35980167984962463
Epoch: 83
Loss on hold-out set: 0.1655131431668997
MeanAbsoluteError value on hold-out data: 0.35789981484413147
Epoch: 84
Loss on hold-out set: 0.15733464618523915
MeanAbsoluteError value on hold-out data: 0.3473641574382782
Epoch: 85
Loss on hold-out set: 0.15509653925895692
MeanAbsoluteError value on hold-out data: 0.3463479280471802
Epoch: 86
Loss on hold-out set: 0.15905273497104644
MeanAbsoluteError value on hold-out data: 0.35169973969459534
Epoch: 87

Loss on hold-out set: 0.15284153665105502
MeanAbsoluteError value on hold-out data: 0.34255534410476685
Epoch: 88
Loss on hold-out set: 0.14979159245888393
MeanAbsoluteError value on hold-out data: 0.34528324007987976
Epoch: 89
Loss on hold-out set: 0.12897902145981788
MeanAbsoluteError value on hold-out data: 0.309967964887619
Epoch: 90
Loss on hold-out set: 0.1537624402095874
MeanAbsoluteError value on hold-out data: 0.3440585434436798
Epoch: 91
Loss on hold-out set: 0.1404809188346068
MeanAbsoluteError value on hold-out data: 0.32490015029907227
Epoch: 92

Loss on hold-out set: 0.1404284318163991
MeanAbsoluteError value on hold-out data: 0.3234095573425293
```

```
Epoch: 93
Loss on hold-out set: 0.14431197355190914
MeanAbsoluteError value on hold-out data: 0.32872939109802246
Epoch: 94
Loss on hold-out set: 0.13744967530171076
MeanAbsoluteError value on hold-out data: 0.3210950791835785
Epoch: 95
Loss on hold-out set: 0.13585215260585148
MeanAbsoluteError value on hold-out data: 0.31810805201530457
Epoch: 96
Loss on hold-out set: 0.1294621142745018
MeanAbsoluteError value on hold-out data: 0.3081085681915283
Epoch: 97

Loss on hold-out set: 0.12832578378419082
MeanAbsoluteError value on hold-out data: 0.3091585636138916
Epoch: 98
Loss on hold-out set: 0.12553194468220075
MeanAbsoluteError value on hold-out data: 0.30347609519958496
Epoch: 99
Loss on hold-out set: 0.13188533440232278
MeanAbsoluteError value on hold-out data: 0.3185291588306427
Epoch: 100
Loss on hold-out set: 0.12567064516246318
MeanAbsoluteError value on hold-out data: 0.3023618161678314
Epoch: 101
Loss on hold-out set: 0.12284717850387096
MeanAbsoluteError value on hold-out data: 0.2947157323360443
Epoch: 102

Loss on hold-out set: 0.11157410716017087
MeanAbsoluteError value on hold-out data: 0.282491534948349
Epoch: 103
Loss on hold-out set: 0.12312864898393551
MeanAbsoluteError value on hold-out data: 0.30094513297080994
Epoch: 104
Loss on hold-out set: 0.11923171895245711
MeanAbsoluteError value on hold-out data: 0.2957533597946167
Epoch: 105
Loss on hold-out set: 0.13442558298508325
MeanAbsoluteError value on hold-out data: 0.3153665065765381
Epoch: 106
```

```
Loss on hold-out set: 0.12007300267616908
MeanAbsoluteError value on hold-out data: 0.3023703396320343
Epoch: 107

Loss on hold-out set: 0.11845777896543344
MeanAbsoluteError value on hold-out data: 0.29478919506073
Epoch: 108
Loss on hold-out set: 0.1131841224928697
MeanAbsoluteError value on hold-out data: 0.28657981753349304
Epoch: 109
Loss on hold-out set: 0.11413929036508004
MeanAbsoluteError value on hold-out data: 0.2853958308696747
Epoch: 110
Loss on hold-out set: 0.10373070277273655
MeanAbsoluteError value on hold-out data: 0.2735512852668762
Epoch: 111
Loss on hold-out set: 0.10639240418871244
MeanAbsoluteError value on hold-out data: 0.2748465836048126
Epoch: 112

Loss on hold-out set: 0.11808696274956067
MeanAbsoluteError value on hold-out data: 0.2890934646129608
Epoch: 113
Loss on hold-out set: 0.10969670712947846
MeanAbsoluteError value on hold-out data: 0.27951523661613464
Epoch: 114
Loss on hold-out set: 0.10457116559147835
MeanAbsoluteError value on hold-out data: 0.27555590867996216
Epoch: 115
Loss on hold-out set: 0.10677940184871355
MeanAbsoluteError value on hold-out data: 0.27495723962783813
Epoch: 116
Loss on hold-out set: 0.11050479340056578
MeanAbsoluteError value on hold-out data: 0.27954211831092834
Epoch: 117

Loss on hold-out set: 0.11399000232418378
MeanAbsoluteError value on hold-out data: 0.2868494689464569
Epoch: 118
Loss on hold-out set: 0.09560877848416567
MeanAbsoluteError value on hold-out data: 0.251302033662796
Epoch: 119
```

```
Loss on hold-out set: 0.10096088401973248
MeanAbsoluteError value on hold-out data: 0.2615692615509033
Epoch: 120
Loss on hold-out set: 0.0919797881382207
MeanAbsoluteError value on hold-out data: 0.25472602248191833
Epoch: 121
Loss on hold-out set: 0.08838441662490368
MeanAbsoluteError value on hold-out data: 0.25093695521354675
Epoch: 122

Loss on hold-out set: 0.10004650426407655
MeanAbsoluteError value on hold-out data: 0.2609286308288574
Epoch: 123
Loss on hold-out set: 0.09790375120937825
MeanAbsoluteError value on hold-out data: 0.26356449723243713
Epoch: 124
Loss on hold-out set: 0.0927121948140363
MeanAbsoluteError value on hold-out data: 0.25226202607154846
Epoch: 125
Loss on hold-out set: 0.09025182058724264
MeanAbsoluteError value on hold-out data: 0.24684815108776093
Epoch: 126
Loss on hold-out set: 0.08278048088153203
MeanAbsoluteError value on hold-out data: 0.2395344078540802
Epoch: 127

Loss on hold-out set: 0.10135412208425502
MeanAbsoluteError value on hold-out data: 0.26062676310539246
Epoch: 128
Loss on hold-out set: 0.0848882817166547
MeanAbsoluteError value on hold-out data: 0.2392461597919464
Returned to Spot: Validation loss: 0.0848882817166547
-----------------------------------------------
spotPython tuning: 0.063074473614494 [#---------] 10.49%


config: {'_L_in': 10, '_L_out': 1, 'l1': 16, 'dropout_prob': 0.27059736028723735, 'lr_mult':
Epoch: 1
Loss on hold-out set: 0.2323484697232121
MeanAbsoluteError value on hold-out data: 0.44617167115211487
Epoch: 2
Loss on hold-out set: 0.23447634671863757
```

```
MeanAbsoluteError value on hold-out data: 0.44901469349861145
Epoch: 3
Loss on hold-out set: 0.2276258113745012
MeanAbsoluteError value on hold-out data: 0.44261983036994934
Epoch: 4
Loss on hold-out set: 0.2356239380805116
MeanAbsoluteError value on hold-out data: 0.44985097646713257
Epoch: 5
Loss on hold-out set: 0.23152991501908554
MeanAbsoluteError value on hold-out data: 0.44581034779548645
Epoch: 6
Loss on hold-out set: 0.23582428929052854
MeanAbsoluteError value on hold-out data: 0.452985018491745
Epoch: 7
Loss on hold-out set: 0.23175337244021266
MeanAbsoluteError value on hold-out data: 0.446553498506546
Epoch: 8
Loss on hold-out set: 0.2318451984932548
MeanAbsoluteError value on hold-out data: 0.4460378885269165
Epoch: 9
Loss on hold-out set: 0.22692673516116643
MeanAbsoluteError value on hold-out data: 0.4405686557292938
Epoch: 10

Loss on hold-out set: 0.22708810375709282
MeanAbsoluteError value on hold-out data: 0.4404347836971283
Epoch: 11
Loss on hold-out set: 0.23013235961920336
MeanAbsoluteError value on hold-out data: 0.4438473880290985
Epoch: 12
Loss on hold-out set: 0.22557749383543668
MeanAbsoluteError value on hold-out data: 0.43861815333366394
Epoch: 13
Loss on hold-out set: 0.22802465075725004
MeanAbsoluteError value on hold-out data: 0.44421127438545227
Epoch: 14
Loss on hold-out set: 0.22840528758732895
MeanAbsoluteError value on hold-out data: 0.442875474691391
Epoch: 15
Loss on hold-out set: 0.22813615006835838
MeanAbsoluteError value on hold-out data: 0.4444199204444885
Epoch: 16
```

```
Loss on hold-out set: 0.22894723262441785
MeanAbsoluteError value on hold-out data: 0.4450794458389282
Returned to Spot: Validation loss: 0.22894723262441785
-----------------------------------------------
spotPython tuning: 0.063074473614494 [#---------] 11.33%


config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7112761417997722, 'lr_mult':
Epoch: 1
Loss on hold-out set: 0.19884953051805496
MeanAbsoluteError value on hold-out data: 0.3964498043060303
Epoch: 2
Loss on hold-out set: 0.2061307960251967
MeanAbsoluteError value on hold-out data: 0.4055088758468628
Epoch: 3
Loss on hold-out set: 0.19159703413645426
MeanAbsoluteError value on hold-out data: 0.39277195930480957
Epoch: 4
Loss on hold-out set: 0.18293502529462177
MeanAbsoluteError value on hold-out data: 0.38221514225006104
Epoch: 5

Loss on hold-out set: 0.1921521436671416
MeanAbsoluteError value on hold-out data: 0.39612025022506714
Epoch: 6
Loss on hold-out set: 0.17544900134205818
MeanAbsoluteError value on hold-out data: 0.37620291113853455
Epoch: 7
Loss on hold-out set: 0.1732803456981977
MeanAbsoluteError value on hold-out data: 0.37326452136039734
Epoch: 8
Loss on hold-out set: 0.17587070738275845
MeanAbsoluteError value on hold-out data: 0.37606099247932434
Epoch: 9
Loss on hold-out set: 0.18563113962610564
MeanAbsoluteError value on hold-out data: 0.3834579288959503
Epoch: 10

Loss on hold-out set: 0.178071276396513
MeanAbsoluteError value on hold-out data: 0.37103089690208435
Epoch: 11
Loss on hold-out set: 0.1693351165453593
```

```
MeanAbsoluteError value on hold-out data: 0.37157773971557617
Epoch: 12
Loss on hold-out set: 0.16759385347366332
MeanAbsoluteError value on hold-out data: 0.36221933364868164
Epoch: 13
Loss on hold-out set: 0.17495800216992696
MeanAbsoluteError value on hold-out data: 0.3704519271850586
Epoch: 14
Loss on hold-out set: 0.15799742855131627
MeanAbsoluteError value on hold-out data: 0.35376593470573425
Epoch: 15

Loss on hold-out set: 0.1620107626914978
MeanAbsoluteError value on hold-out data: 0.36044058203697205
Epoch: 16
Loss on hold-out set: 0.15722023467222848
MeanAbsoluteError value on hold-out data: 0.353779137134552
Epoch: 17
Loss on hold-out set: 0.15804068356752396
MeanAbsoluteError value on hold-out data: 0.3468295931816101
Epoch: 18
Loss on hold-out set: 0.16384361788630486
MeanAbsoluteError value on hold-out data: 0.3547916114330292
Epoch: 19
Loss on hold-out set: 0.15893174161513646
MeanAbsoluteError value on hold-out data: 0.3544120788574219
Epoch: 20

Loss on hold-out set: 0.1503339332838853
MeanAbsoluteError value on hold-out data: 0.33928048610687256
Epoch: 21
Loss on hold-out set: 0.1492849018673102
MeanAbsoluteError value on hold-out data: 0.34274008870124817
Epoch: 22
Loss on hold-out set: 0.14599705005685487
MeanAbsoluteError value on hold-out data: 0.3348415493965149
Epoch: 23
Loss on hold-out set: 0.14661807025472323
MeanAbsoluteError value on hold-out data: 0.33497393131256104
Epoch: 24
Loss on hold-out set: 0.14729427695274352
MeanAbsoluteError value on hold-out data: 0.33410176634788513
```

```
Epoch: 25

Loss on hold-out set: 0.1460249412059784
MeanAbsoluteError value on hold-out data: 0.3334328830242157
Epoch: 26
Loss on hold-out set: 0.1430066951860984
MeanAbsoluteError value on hold-out data: 0.33403706550598145
Epoch: 27
Loss on hold-out set: 0.1426254318157832
MeanAbsoluteError value on hold-out data: 0.3289873003959656
Epoch: 28
Loss on hold-out set: 0.13507191772262256
MeanAbsoluteError value on hold-out data: 0.32317131757736206
Epoch: 29
Loss on hold-out set: 0.13217110315958658
MeanAbsoluteError value on hold-out data: 0.31831124424934387
Epoch: 30

Loss on hold-out set: 0.12710754122585058
MeanAbsoluteError value on hold-out data: 0.3060861825942993
Epoch: 31
Loss on hold-out set: 0.12546489076068004
MeanAbsoluteError value on hold-out data: 0.3078380525112152
Epoch: 32
Loss on hold-out set: 0.13357673625151317
MeanAbsoluteError value on hold-out data: 0.3187636435031891
Epoch: 33
Loss on hold-out set: 0.13098478933175406
MeanAbsoluteError value on hold-out data: 0.3129400908946991
Epoch: 34
Loss on hold-out set: 0.12248824293414752
MeanAbsoluteError value on hold-out data: 0.3040026128292084
Epoch: 35

Loss on hold-out set: 0.1325142823283871
MeanAbsoluteError value on hold-out data: 0.3197743892669678
Epoch: 36
Loss on hold-out set: 0.12200007356703281
MeanAbsoluteError value on hold-out data: 0.30403247475624084
Epoch: 37
Loss on hold-out set: 0.12704870889584224
MeanAbsoluteError value on hold-out data: 0.3094150424003601
```

```
Epoch: 38
Loss on hold-out set: 0.12120759544273217
MeanAbsoluteError value on hold-out data: 0.2996427118778229
Epoch: 39
Loss on hold-out set: 0.11852605313062668
MeanAbsoluteError value on hold-out data: 0.2968824505805969
Epoch: 40

Loss on hold-out set: 0.11607860170304775
MeanAbsoluteError value on hold-out data: 0.29226014018058777
Epoch: 41
Loss on hold-out set: 0.11009105407943329
MeanAbsoluteError value on hold-out data: 0.28366315364837646
Epoch: 42
Loss on hold-out set: 0.10729422084987164
MeanAbsoluteError value on hold-out data: 0.2814805209636688
Epoch: 43
Loss on hold-out set: 0.11049179661087692
MeanAbsoluteError value on hold-out data: 0.2846699655056
Epoch: 44
Loss on hold-out set: 0.1080995467553536
MeanAbsoluteError value on hold-out data: 0.2743377685546875
Epoch: 45

Loss on hold-out set: 0.11330131004254024
MeanAbsoluteError value on hold-out data: 0.28311148285865784
Epoch: 46
Loss on hold-out set: 0.11003661148250103
MeanAbsoluteError value on hold-out data: 0.27961617708206177
Epoch: 47
Loss on hold-out set: 0.11241017242272695
MeanAbsoluteError value on hold-out data: 0.28350141644477844
Epoch: 48
Loss on hold-out set: 0.09861218266189098
MeanAbsoluteError value on hold-out data: 0.26371508836746216
Epoch: 49
Loss on hold-out set: 0.09953342581788699
MeanAbsoluteError value on hold-out data: 0.26712730526924133
Epoch: 50

Loss on hold-out set: 0.09979852405066292
MeanAbsoluteError value on hold-out data: 0.2673349976539612
```

```
Epoch: 51
Loss on hold-out set: 0.09964208278184136
MeanAbsoluteError value on hold-out data: 0.2568577229976654
Epoch: 52
Loss on hold-out set: 0.09045088007425268
MeanAbsoluteError value on hold-out data: 0.257308691740036
Epoch: 53
Loss on hold-out set: 0.09243312734489639
MeanAbsoluteError value on hold-out data: 0.2556573152542114
Epoch: 54
Loss on hold-out set: 0.09745575634141763
MeanAbsoluteError value on hold-out data: 0.26239320635795593
Epoch: 55

Loss on hold-out set: 0.09503113396465779
MeanAbsoluteError value on hold-out data: 0.25415846705436707
Epoch: 56
Loss on hold-out set: 0.09214671264092128
MeanAbsoluteError value on hold-out data: 0.2565299868583679
Epoch: 57
Loss on hold-out set: 0.09393365501115719
MeanAbsoluteError value on hold-out data: 0.25862619280815125
Epoch: 58
Loss on hold-out set: 0.09306338773419459
MeanAbsoluteError value on hold-out data: 0.25443536043167114
Epoch: 59
Loss on hold-out set: 0.0835562530780832
MeanAbsoluteError value on hold-out data: 0.24088795483112335
Epoch: 60

Loss on hold-out set: 0.0868484183649222
MeanAbsoluteError value on hold-out data: 0.2502835988998413
Epoch: 61
Loss on hold-out set: 0.08844985221823057
MeanAbsoluteError value on hold-out data: 0.25208431482315063
Epoch: 62
Loss on hold-out set: 0.08563402321189642
MeanAbsoluteError value on hold-out data: 0.24109038710594177
Epoch: 63
Loss on hold-out set: 0.0822727590923508
MeanAbsoluteError value on hold-out data: 0.24017538130283356
Epoch: 64
```

```
Loss on hold-out set: 0.08360667581359546
MeanAbsoluteError value on hold-out data: 0.24379438161849976
Epoch: 65

Loss on hold-out set: 0.07547835439443588
MeanAbsoluteError value on hold-out data: 0.22758041322231293
Epoch: 66
Loss on hold-out set: 0.076917410120368
MeanAbsoluteError value on hold-out data: 0.2296087145805359
Epoch: 67
Loss on hold-out set: 0.08410916571194926
MeanAbsoluteError value on hold-out data: 0.23821032047271729
Epoch: 68
Loss on hold-out set: 0.07932173011203607
MeanAbsoluteError value on hold-out data: 0.23269571363925934
Epoch: 69
Loss on hold-out set: 0.07537223263333241
MeanAbsoluteError value on hold-out data: 0.2246152013540268
Epoch: 70

Loss on hold-out set: 0.07794367529451847
MeanAbsoluteError value on hold-out data: 0.2351241260766983
Epoch: 71
Loss on hold-out set: 0.07993815435096621
MeanAbsoluteError value on hold-out data: 0.23204778134822845
Epoch: 72
Loss on hold-out set: 0.07535223400841157
MeanAbsoluteError value on hold-out data: 0.22259266674518585
Epoch: 73
Loss on hold-out set: 0.07409557123668492
MeanAbsoluteError value on hold-out data: 0.22099721431732178
Epoch: 74
Loss on hold-out set: 0.07094336457550526
MeanAbsoluteError value on hold-out data: 0.2225976586341858
Epoch: 75

Loss on hold-out set: 0.07981918339927992
MeanAbsoluteError value on hold-out data: 0.2299015074968338
Epoch: 76
Loss on hold-out set: 0.07025228018561999
MeanAbsoluteError value on hold-out data: 0.21866463124752045
Epoch: 77
```

```
Loss on hold-out set: 0.07337113842989007
MeanAbsoluteError value on hold-out data: 0.21997156739234924
Epoch: 78
Loss on hold-out set: 0.06599894162267446
MeanAbsoluteError value on hold-out data: 0.20799137651920319
Epoch: 79
Loss on hold-out set: 0.06625080538292726
MeanAbsoluteError value on hold-out data: 0.21423476934432983
Epoch: 80

Loss on hold-out set: 0.06308392386262615
MeanAbsoluteError value on hold-out data: 0.2073647677898407
Epoch: 81
Loss on hold-out set: 0.06994831291958689
MeanAbsoluteError value on hold-out data: 0.2129209041595459
Epoch: 82
Loss on hold-out set: 0.07177561726421118
MeanAbsoluteError value on hold-out data: 0.22234635055065155
Epoch: 83
Loss on hold-out set: 0.07065899837141236
MeanAbsoluteError value on hold-out data: 0.21766400337219238
Epoch: 84
Loss on hold-out set: 0.06860992979568739
MeanAbsoluteError value on hold-out data: 0.21229049563407898

Epoch: 85
Loss on hold-out set: 0.06714078649568062
MeanAbsoluteError value on hold-out data: 0.21379099786281586
Epoch: 86
Loss on hold-out set: 0.06563440727690856
MeanAbsoluteError value on hold-out data: 0.2072865515947342
Epoch: 87
Loss on hold-out set: 0.06268777408947547
MeanAbsoluteError value on hold-out data: 0.2016163170337677
Epoch: 88
Loss on hold-out set: 0.06647988620214164
MeanAbsoluteError value on hold-out data: 0.2113489955663681

Epoch: 89
Loss on hold-out set: 0.062262841754903396
MeanAbsoluteError value on hold-out data: 0.20844724774360657
Epoch: 90
```

```
Loss on hold-out set: 0.06268599266807239
MeanAbsoluteError value on hold-out data: 0.20432989299297333
Epoch: 91
Loss on hold-out set: 0.05856219927469889
MeanAbsoluteError value on hold-out data: 0.19689512252807617
Epoch: 92
Loss on hold-out set: 0.05911588766612112
MeanAbsoluteError value on hold-out data: 0.20235855877399445
Epoch: 93

Loss on hold-out set: 0.05684368140374621
MeanAbsoluteError value on hold-out data: 0.18799032270908356
Epoch: 94
Loss on hold-out set: 0.06513360555594166
MeanAbsoluteError value on hold-out data: 0.20837992429733276
Epoch: 95
Loss on hold-out set: 0.05740601465106011
MeanAbsoluteError value on hold-out data: 0.19307281076908112
Epoch: 96
Loss on hold-out set: 0.059231638486186663
MeanAbsoluteError value on hold-out data: 0.20002895593643188
Epoch: 97
Loss on hold-out set: 0.05618767047921817
MeanAbsoluteError value on hold-out data: 0.1930500715970993
Epoch: 98

Loss on hold-out set: 0.05332934205420315
MeanAbsoluteError value on hold-out data: 0.18701905012130737
Epoch: 99
Loss on hold-out set: 0.05473091966783007
MeanAbsoluteError value on hold-out data: 0.19038981199264526
Epoch: 100
Loss on hold-out set: 0.05191120636028548
MeanAbsoluteError value on hold-out data: 0.18725188076496124
Epoch: 101
Loss on hold-out set: 0.05793134836480021
MeanAbsoluteError value on hold-out data: 0.19263723492622375
Epoch: 102
Loss on hold-out set: 0.0540062690153718
MeanAbsoluteError value on hold-out data: 0.1869107186794281
Epoch: 103
```

```
Loss on hold-out set: 0.04950447903946042
MeanAbsoluteError value on hold-out data: 0.17735467851161957
Epoch: 104
Loss on hold-out set: 0.05291481090709567
MeanAbsoluteError value on hold-out data: 0.1854318380355835
Epoch: 105
Loss on hold-out set: 0.055706309210509064
MeanAbsoluteError value on hold-out data: 0.19061823189258575
Epoch: 106
Loss on hold-out set: 0.055637425600240625
MeanAbsoluteError value on hold-out data: 0.18939940631389618
Epoch: 107
Loss on hold-out set: 0.055092927850782875
MeanAbsoluteError value on hold-out data: 0.1925634890794754
Epoch: 108

Loss on hold-out set: 0.05393709988643726
MeanAbsoluteError value on hold-out data: 0.1809026598930359
Epoch: 109
Loss on hold-out set: 0.05871706864486138
MeanAbsoluteError value on hold-out data: 0.19786082208156586
Epoch: 110
Loss on hold-out set: 0.05863720334445437
MeanAbsoluteError value on hold-out data: 0.19687682390213013
Epoch: 111
Loss on hold-out set: 0.05192665280463795
MeanAbsoluteError value on hold-out data: 0.18031160533428192
Epoch: 112
Loss on hold-out set: 0.05044209911177556
MeanAbsoluteError value on hold-out data: 0.18155232071876526

Epoch: 113
Loss on hold-out set: 0.06132638225952784
MeanAbsoluteError value on hold-out data: 0.19898121058940887
Epoch: 114
Loss on hold-out set: 0.05342992429931959
MeanAbsoluteError value on hold-out data: 0.18557007610797882
Epoch: 115
Loss on hold-out set: 0.05263446789234877
MeanAbsoluteError value on hold-out data: 0.1865229308605194
Epoch: 116
Loss on hold-out set: 0.05111423779278994
```

```
MeanAbsoluteError value on hold-out data: 0.18401555716991425
Epoch: 117

Loss on hold-out set: 0.05958604438075175
MeanAbsoluteError value on hold-out data: 0.19749748706817627
Epoch: 118
Loss on hold-out set: 0.04823327540419996
MeanAbsoluteError value on hold-out data: 0.17182926833629608
Epoch: 119
Loss on hold-out set: 0.05123234694202741
MeanAbsoluteError value on hold-out data: 0.18186572194099426
Epoch: 120
Loss on hold-out set: 0.04918001433213552
MeanAbsoluteError value on hold-out data: 0.17959079146385193
Epoch: 121
Loss on hold-out set: 0.048532408587634565
MeanAbsoluteError value on hold-out data: 0.18022902309894562
Epoch: 122

Loss on hold-out set: 0.05208494819079836
MeanAbsoluteError value on hold-out data: 0.18349510431289673
Epoch: 123
Loss on hold-out set: 0.047289955491820974
MeanAbsoluteError value on hold-out data: 0.17992612719535828
Epoch: 124
Loss on hold-out set: 0.049321671227614085
MeanAbsoluteError value on hold-out data: 0.1776355504989624
Epoch: 125
Loss on hold-out set: 0.048682038085535166
MeanAbsoluteError value on hold-out data: 0.17510510981082916
Epoch: 126
Loss on hold-out set: 0.047216447962758444
MeanAbsoluteError value on hold-out data: 0.17804080247879028
Epoch: 127

Loss on hold-out set: 0.04938874273560941
MeanAbsoluteError value on hold-out data: 0.1815052479505539
Epoch: 128
Loss on hold-out set: 0.04639910859366258
MeanAbsoluteError value on hold-out data: 0.17332109808921814
Returned to Spot: Validation loss: 0.04639910859366258
-----------------------------------------------
```

```
spotPython tuning: 0.04639910859366258 [##--------] 22.11%


config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7384173495571664, 'lr_mult':


Epoch: 1
Loss on hold-out set: 0.45527631024519605
MeanAbsoluteError value on hold-out data: 0.629140317440033
Epoch: 2
Loss on hold-out set: 0.4448426284392675
MeanAbsoluteError value on hold-out data: 0.627829909324646
Epoch: 3
Loss on hold-out set: 0.4183800055583318
MeanAbsoluteError value on hold-out data: 0.5992152094841003
Epoch: 4
Loss on hold-out set: 0.4388718934853872
MeanAbsoluteError value on hold-out data: 0.6140342950820923
Epoch: 5

Loss on hold-out set: 0.43975292642911273
MeanAbsoluteError value on hold-out data: 0.6205071210861206
Epoch: 6
Loss on hold-out set: 0.4097070248921712
MeanAbsoluteError value on hold-out data: 0.5955525040626526
Epoch: 7
Loss on hold-out set: 0.40890683313210807
MeanAbsoluteError value on hold-out data: 0.5998004078865051
Epoch: 8
Loss on hold-out set: 0.4102619783083598
MeanAbsoluteError value on hold-out data: 0.5931419134140015
Epoch: 9
Loss on hold-out set: 0.38767369935909907
MeanAbsoluteError value on hold-out data: 0.5775493383407593
Epoch: 10

Loss on hold-out set: 0.4060617202520371
MeanAbsoluteError value on hold-out data: 0.5921320915222168
Epoch: 11
Loss on hold-out set: 0.3721655536691348
MeanAbsoluteError value on hold-out data: 0.5722568035125732
```

```
Epoch: 12
Loss on hold-out set: 0.36280858993530274
MeanAbsoluteError value on hold-out data: 0.5578823685646057
Epoch: 13
Loss on hold-out set: 0.38783051192760465
MeanAbsoluteError value on hold-out data: 0.5756087899208069
Epoch: 14
Loss on hold-out set: 0.3823117671410243
MeanAbsoluteError value on hold-out data: 0.5722373723983765
Epoch: 15

Loss on hold-out set: 0.35494314471880595
MeanAbsoluteError value on hold-out data: 0.5477345585823059
Epoch: 16
Loss on hold-out set: 0.3509993120034536
MeanAbsoluteError value on hold-out data: 0.5505962371826172
Epoch: 17
Loss on hold-out set: 0.375661740899086
MeanAbsoluteError value on hold-out data: 0.5721865892410278
Epoch: 18
Loss on hold-out set: 0.37970457375049593
MeanAbsoluteError value on hold-out data: 0.568191409111023
Epoch: 19
Loss on hold-out set: 0.3648752955595652
MeanAbsoluteError value on hold-out data: 0.5568380951881409
Epoch: 20

Loss on hold-out set: 0.35649665534496305
MeanAbsoluteError value on hold-out data: 0.5544812083244324
Epoch: 21
Loss on hold-out set: 0.33892221877972284
MeanAbsoluteError value on hold-out data: 0.5349441766738892
Epoch: 22
Loss on hold-out set: 0.33623027553160983
MeanAbsoluteError value on hold-out data: 0.5333371758460999
Epoch: 23
Loss on hold-out set: 0.3203277287632227
MeanAbsoluteError value on hold-out data: 0.5119890570640564
Epoch: 24
Loss on hold-out set: 0.32151335626840594
MeanAbsoluteError value on hold-out data: 0.516357958316803
Epoch: 25
```

```
Loss on hold-out set: 0.3157925768693288
MeanAbsoluteError value on hold-out data: 0.5194802284240723
Epoch: 26
Loss on hold-out set: 0.34404603511095044
MeanAbsoluteError value on hold-out data: 0.5430979132652283
Epoch: 27
Loss on hold-out set: 0.33008826275666553
MeanAbsoluteError value on hold-out data: 0.526513934135437
Epoch: 28
Loss on hold-out set: 0.3228203006585439
MeanAbsoluteError value on hold-out data: 0.5223667621612549
Epoch: 29
Loss on hold-out set: 0.3070026060938835
MeanAbsoluteError value on hold-out data: 0.5032530426979065
Epoch: 30

Loss on hold-out set: 0.2990411035219828
MeanAbsoluteError value on hold-out data: 0.5006046295166016
Epoch: 31
Loss on hold-out set: 0.30098104109366736
MeanAbsoluteError value on hold-out data: 0.5041744112968445
Epoch: 32
Loss on hold-out set: 0.3173143946627776
MeanAbsoluteError value on hold-out data: 0.5107991695404053
Epoch: 33
Loss on hold-out set: 0.27731694767872495
MeanAbsoluteError value on hold-out data: 0.4723749160766016
Epoch: 34
Loss on hold-out set: 0.2897105230887731
MeanAbsoluteError value on hold-out data: 0.4906061887741089
Epoch: 35

Loss on hold-out set: 0.28225571711858116
MeanAbsoluteError value on hold-out data: 0.47769245505332947
Epoch: 36
Loss on hold-out set: 0.280241640607516
MeanAbsoluteError value on hold-out data: 0.4747069180011749
Epoch: 37
Loss on hold-out set: 0.26948808421691256
MeanAbsoluteError value on hold-out data: 0.4697422683238983
Epoch: 38
Loss on hold-out set: 0.270139994819959
```

```
MeanAbsoluteError value on hold-out data: 0.4689478576183319
Epoch: 39
Loss on hold-out set: 0.270493657986323
MeanAbsoluteError value on hold-out data: 0.4782830774784088
Epoch: 40

Loss on hold-out set: 0.2698291861017545
MeanAbsoluteError value on hold-out data: 0.4741959571838379
Epoch: 41
Loss on hold-out set: 0.24705043599009513
MeanAbsoluteError value on hold-out data: 0.4447568655014038
Epoch: 42
Loss on hold-out set: 0.2342171460390091
MeanAbsoluteError value on hold-out data: 0.42764317989349365
Epoch: 43
Loss on hold-out set: 0.24928770581881204
MeanAbsoluteError value on hold-out data: 0.4480302929878235
Epoch: 44

Loss on hold-out set: 0.23928461362918219
MeanAbsoluteError value on hold-out data: 0.435867577791214
Epoch: 45
Loss on hold-out set: 0.24591269940137864
MeanAbsoluteError value on hold-out data: 0.44985997676849365
Epoch: 46
Loss on hold-out set: 0.2363617326815923
MeanAbsoluteError value on hold-out data: 0.43433091044425964
Epoch: 47
Loss on hold-out set: 0.2475802824894587
MeanAbsoluteError value on hold-out data: 0.4360943138599396
Epoch: 48
Loss on hold-out set: 0.23467126453916232
MeanAbsoluteError value on hold-out data: 0.42883092164993286
Epoch: 49

Loss on hold-out set: 0.233955034861962
MeanAbsoluteError value on hold-out data: 0.4328742027282715
Epoch: 50
Loss on hold-out set: 0.22453522562980652
MeanAbsoluteError value on hold-out data: 0.4133330285549164
Epoch: 51
Loss on hold-out set: 0.21710970227917034
```

```
MeanAbsoluteError value on hold-out data: 0.42004770040512085
Epoch: 52
Loss on hold-out set: 0.2198191216091315
MeanAbsoluteError value on hold-out data: 0.41269025206565857
Epoch: 53
Loss on hold-out set: 0.2220701342821121
MeanAbsoluteError value on hold-out data: 0.4190760552883148
Epoch: 54

Loss on hold-out set: 0.2264922122657299
MeanAbsoluteError value on hold-out data: 0.42179563641548157
Epoch: 55
Loss on hold-out set: 0.21733828658858936
MeanAbsoluteError value on hold-out data: 0.41376349329948425
Epoch: 56
Loss on hold-out set: 0.2084229916334152
MeanAbsoluteError value on hold-out data: 0.401915967464447
Epoch: 57
Loss on hold-out set: 0.2020088283220927
MeanAbsoluteError value on hold-out data: 0.39198243618011475
Epoch: 58

Loss on hold-out set: 0.20942022209366162
MeanAbsoluteError value on hold-out data: 0.4059727191925049
Epoch: 59
Loss on hold-out set: 0.1964935710032781
MeanAbsoluteError value on hold-out data: 0.38819143176078796
Epoch: 60
Loss on hold-out set: 0.1896145276228587
MeanAbsoluteError value on hold-out data: 0.378763347864151
Epoch: 61
Loss on hold-out set: 0.18881956989566484
MeanAbsoluteError value on hold-out data: 0.3805134892463684
Epoch: 62

Loss on hold-out set: 0.18855410059293112
MeanAbsoluteError value on hold-out data: 0.37789785861968994
Epoch: 63
Loss on hold-out set: 0.186182425369819
MeanAbsoluteError value on hold-out data: 0.37386614084243774
Epoch: 64
Loss on hold-out set: 0.16666620552539826
```

```
MeanAbsoluteError value on hold-out data: 0.3516756296157837
Epoch: 65
Loss on hold-out set: 0.1847467502951622
MeanAbsoluteError value on hold-out data: 0.37751102447509766
Epoch: 66
Loss on hold-out set: 0.1692993335922559
MeanAbsoluteError value on hold-out data: 0.3576069176197052
Epoch: 67

Loss on hold-out set: 0.17689398715893428
MeanAbsoluteError value on hold-out data: 0.35995474457740784
Epoch: 68
Loss on hold-out set: 0.18588505225876967
MeanAbsoluteError value on hold-out data: 0.3706034719944
Epoch: 69
Loss on hold-out set: 0.16740913274387517
MeanAbsoluteError value on hold-out data: 0.34518951177597046
Epoch: 70
Loss on hold-out set: 0.1655419887105624
MeanAbsoluteError value on hold-out data: 0.3533257246017456
Epoch: 71
Loss on hold-out set: 0.17042002404729525
MeanAbsoluteError value on hold-out data: 0.35140475630760193
Epoch: 72

Loss on hold-out set: 0.15655692397927246
MeanAbsoluteError value on hold-out data: 0.3333028256893158
Epoch: 73
Loss on hold-out set: 0.1610123293598493
MeanAbsoluteError value on hold-out data: 0.3361779451370239
Epoch: 74
Loss on hold-out set: 0.14988909780979157
MeanAbsoluteError value on hold-out data: 0.3317258358001709
Epoch: 75
Loss on hold-out set: 0.1567149386058251
MeanAbsoluteError value on hold-out data: 0.33861497044563293
Epoch: 76
Loss on hold-out set: 0.15707411472996077
MeanAbsoluteError value on hold-out data: 0.3400817811489105
Epoch: 77

Loss on hold-out set: 0.15243841382364431
```

```
MeanAbsoluteError value on hold-out data: 0.3338886797428131
Epoch: 78
Loss on hold-out set: 0.1552959192295869
MeanAbsoluteError value on hold-out data: 0.3368751108646393
Epoch: 79
Loss on hold-out set: 0.16038350348671276
MeanAbsoluteError value on hold-out data: 0.3403923213481903
Epoch: 80
Loss on hold-out set: 0.13245410699397325
MeanAbsoluteError value on hold-out data: 0.30506691336631775
Epoch: 81
Loss on hold-out set: 0.13505938557287056
MeanAbsoluteError value on hold-out data: 0.31024596095085144
Epoch: 82

Loss on hold-out set: 0.13544613444556794
MeanAbsoluteError value on hold-out data: 0.3088962137699127
Epoch: 83
Loss on hold-out set: 0.13764356895039478
MeanAbsoluteError value on hold-out data: 0.31219804286956787
Epoch: 84
Loss on hold-out set: 0.1306554144869248
MeanAbsoluteError value on hold-out data: 0.3042640686035156
Epoch: 85
Loss on hold-out set: 0.12448016421248516
MeanAbsoluteError value on hold-out data: 0.29987025260925293
Epoch: 86

Loss on hold-out set: 0.1364940921589732
MeanAbsoluteError value on hold-out data: 0.3097861111164093
Epoch: 87
Loss on hold-out set: 0.12196354385465384
MeanAbsoluteError value on hold-out data: 0.29111942648887634
Epoch: 88
Loss on hold-out set: 0.12214496886978547
MeanAbsoluteError value on hold-out data: 0.2943097949028015
Epoch: 89
Loss on hold-out set: 0.13419622046252092
MeanAbsoluteError value on hold-out data: 0.30828845500946045
Epoch: 90
Loss on hold-out set: 0.12958096874256927
MeanAbsoluteError value on hold-out data: 0.3036196231842041
```

```
Epoch: 91

Loss on hold-out set: 0.13047171864658594
MeanAbsoluteError value on hold-out data: 0.299602210521698
Epoch: 92
Loss on hold-out set: 0.13013865207632383
MeanAbsoluteError value on hold-out data: 0.30680856108665466
Epoch: 93
Loss on hold-out set: 0.124781855220596
MeanAbsoluteError value on hold-out data: 0.29297125339508057
Epoch: 94
Loss on hold-out set: 0.10514580654601256
MeanAbsoluteError value on hold-out data: 0.2657950222492218
Epoch: 95
Loss on hold-out set: 0.12362907661745946
MeanAbsoluteError value on hold-out data: 0.29290610551834106
Epoch: 96

Loss on hold-out set: 0.11904967337846756
MeanAbsoluteError value on hold-out data: 0.2880531847476959
Epoch: 97
Loss on hold-out set: 0.11060909713928899
MeanAbsoluteError value on hold-out data: 0.2744829058647156
Epoch: 98
Loss on hold-out set: 0.11864176943898201
MeanAbsoluteError value on hold-out data: 0.28487399220466614
Epoch: 99
Loss on hold-out set: 0.10995991714298725
MeanAbsoluteError value on hold-out data: 0.27471011877059937
Epoch: 100
Loss on hold-out set: 0.11937080776318908
MeanAbsoluteError value on hold-out data: 0.2910013198852539
Epoch: 101

Loss on hold-out set: 0.11812335376938184
MeanAbsoluteError value on hold-out data: 0.28868019580841064
Epoch: 102
Loss on hold-out set: 0.10860014515618484
MeanAbsoluteError value on hold-out data: 0.26667192578315735
Epoch: 103
Loss on hold-out set: 0.10801799442619085
MeanAbsoluteError value on hold-out data: 0.2668939530849457
```

```
Epoch: 104
Loss on hold-out set: 0.11973672643303872
MeanAbsoluteError value on hold-out data: 0.29072457551956177
Epoch: 105

Loss on hold-out set: 0.10000308209409316
MeanAbsoluteError value on hold-out data: 0.2601052522659302
Epoch: 106
Loss on hold-out set: 0.09323087993388375
MeanAbsoluteError value on hold-out data: 0.24999986588954926
Epoch: 107
Loss on hold-out set: 0.10900106299668551
MeanAbsoluteError value on hold-out data: 0.2740049660205841
Epoch: 108
Loss on hold-out set: 0.11179569127659003
MeanAbsoluteError value on hold-out data: 0.2739800214767456
Epoch: 109
Loss on hold-out set: 0.0997035872315367
MeanAbsoluteError value on hold-out data: 0.25560295581817627
Epoch: 110

Loss on hold-out set: 0.1059795144572854
MeanAbsoluteError value on hold-out data: 0.25833848118782043
Epoch: 111
Loss on hold-out set: 0.09905201068768898
MeanAbsoluteError value on hold-out data: 0.26084017753601074
Epoch: 112
Loss on hold-out set: 0.10014197620252768
MeanAbsoluteError value on hold-out data: 0.25984764099121094
Epoch: 113
Loss on hold-out set: 0.09424603419999282
MeanAbsoluteError value on hold-out data: 0.24911847710609436
Epoch: 114
Loss on hold-out set: 0.10141485162079335
MeanAbsoluteError value on hold-out data: 0.2559027373790741
Epoch: 115

Loss on hold-out set: 0.0924946150990824
MeanAbsoluteError value on hold-out data: 0.24871772527694702
Epoch: 116
Loss on hold-out set: 0.09035151497771343
MeanAbsoluteError value on hold-out data: 0.24196238815784454
```

```
Epoch: 117
Loss on hold-out set: 0.09662960336854061
MeanAbsoluteError value on hold-out data: 0.25432026386260986
Epoch: 118
Loss on hold-out set: 0.10426713429391384
MeanAbsoluteError value on hold-out data: 0.26339611411094666
Epoch: 119
Loss on hold-out set: 0.09173825849468509
MeanAbsoluteError value on hold-out data: 0.24543273448944092
Epoch: 120

Loss on hold-out set: 0.09198710290094217
MeanAbsoluteError value on hold-out data: 0.24291735887527466
Epoch: 121
Loss on hold-out set: 0.09198215438673894
MeanAbsoluteError value on hold-out data: 0.23999308049678802
Epoch: 122
Loss on hold-out set: 0.09777610786259175
MeanAbsoluteError value on hold-out data: 0.25733721256256104
Epoch: 123
Loss on hold-out set: 0.0785861844321092
MeanAbsoluteError value on hold-out data: 0.22858935594558716
Epoch: 124
Loss on hold-out set: 0.08204124719525377
MeanAbsoluteError value on hold-out data: 0.2325047254562378
Epoch: 125

Loss on hold-out set: 0.09375863761524669
MeanAbsoluteError value on hold-out data: 0.2518348693847656
Epoch: 126
Loss on hold-out set: 0.09193931301900496
MeanAbsoluteError value on hold-out data: 0.24960222840309143
Epoch: 127
Loss on hold-out set: 0.08403404330834746
MeanAbsoluteError value on hold-out data: 0.23277750611305237
Epoch: 128
Loss on hold-out set: 0.0850067349212865
MeanAbsoluteError value on hold-out data: 0.22967255115509033
Returned to Spot: Validation loss: 0.0850067349212865
---------------------------------------------

spotPython tuning: 0.04639910859366258 [###-------] 33.06%
```

```
config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7127779991790958, 'lr_mult':
Epoch: 1
Loss on hold-out set: 0.2594366767009099
MeanAbsoluteError value on hold-out data: 0.4637209177017212
Epoch: 2

Loss on hold-out set: 0.2634414581457774
MeanAbsoluteError value on hold-out data: 0.4661996066570282
Epoch: 3
Loss on hold-out set: 0.24490316977103552
MeanAbsoluteError value on hold-out data: 0.45433729887008667
Epoch: 4

Loss on hold-out set: 0.2462012713154157
MeanAbsoluteError value on hold-out data: 0.4541053771972656
Epoch: 5
Loss on hold-out set: 0.23569323102633158
MeanAbsoluteError value on hold-out data: 0.43935126066207886
Epoch: 6
Loss on hold-out set: 0.2435258959730466
MeanAbsoluteError value on hold-out data: 0.4523331820964813
Epoch: 7

Loss on hold-out set: 0.23863106826941172
MeanAbsoluteError value on hold-out data: 0.4429166615009308
Epoch: 8
Loss on hold-out set: 0.2327429818113645
MeanAbsoluteError value on hold-out data: 0.44275137782096863
Epoch: 9

Loss on hold-out set: 0.23483260124921798
MeanAbsoluteError value on hold-out data: 0.4424296021461487
Epoch: 10
Loss on hold-out set: 0.23004452407360076
MeanAbsoluteError value on hold-out data: 0.43610209226608276
Epoch: 11
Loss on hold-out set: 0.22535803417364755
MeanAbsoluteError value on hold-out data: 0.42963236570358276
```

```
Epoch: 12
Loss on hold-out set: 0.22041525478164356
MeanAbsoluteError value on hold-out data: 0.4179801046848297
Epoch: 13
Loss on hold-out set: 0.21483287046353022
MeanAbsoluteError value on hold-out data: 0.41867589950561523

Epoch: 14
Loss on hold-out set: 0.21500397101044655
MeanAbsoluteError value on hold-out data: 0.41776660084724426
Epoch: 15
Loss on hold-out set: 0.2106473597884178
MeanAbsoluteError value on hold-out data: 0.4133099913597107
Epoch: 16

Loss on hold-out set: 0.20837476442257563
MeanAbsoluteError value on hold-out data: 0.4092746376991272
Epoch: 17
Loss on hold-out set: 0.20944500754276912
MeanAbsoluteError value on hold-out data: 0.4128493070602417
Epoch: 18

Loss on hold-out set: 0.19697490667303402
MeanAbsoluteError value on hold-out data: 0.395319402217865
Epoch: 19
Loss on hold-out set: 0.20808862636486689
MeanAbsoluteError value on hold-out data: 0.40932026505470276
Epoch: 20
Loss on hold-out set: 0.206798337996006
MeanAbsoluteError value on hold-out data: 0.4066202938556671
Epoch: 21

Loss on hold-out set: 0.19411883915464084
MeanAbsoluteError value on hold-out data: 0.3954356908798218
Epoch: 22
Loss on hold-out set: 0.18120040426651637
MeanAbsoluteError value on hold-out data: 0.37774354219436646
Epoch: 23

Loss on hold-out set: 0.2033400951574246
MeanAbsoluteError value on hold-out data: 0.39930006861686707
```

```
Epoch: 24
Loss on hold-out set: 0.1901332516471545
MeanAbsoluteError value on hold-out data: 0.3930417001247406
Epoch: 25
Loss on hold-out set: 0.18291841581463814
MeanAbsoluteError value on hold-out data: 0.38028496503829956
Epoch: 26

Loss on hold-out set: 0.1778881218781074
MeanAbsoluteError value on hold-out data: 0.37627553939819336
Epoch: 27
Loss on hold-out set: 0.1772528338432312
MeanAbsoluteError value on hold-out data: 0.3765186667442322
Epoch: 28

Loss on hold-out set: 0.1812676373620828
MeanAbsoluteError value on hold-out data: 0.3771665394306183
Epoch: 29
Loss on hold-out set: 0.1764606953660647
MeanAbsoluteError value on hold-out data: 0.376229465007782
Epoch: 30
Loss on hold-out set: 0.17948614751299222
MeanAbsoluteError value on hold-out data: 0.3737820088863373

Epoch: 31
Loss on hold-out set: 0.15822886273264886
MeanAbsoluteError value on hold-out data: 0.3521953225135803
Epoch: 32
Loss on hold-out set: 0.16952462104459604
MeanAbsoluteError value on hold-out data: 0.36028003692626953
Epoch: 33

Loss on hold-out set: 0.1589691990117232
MeanAbsoluteError value on hold-out data: 0.3545733094215393
Epoch: 34
Loss on hold-out set: 0.16350217781340082
MeanAbsoluteError value on hold-out data: 0.3587236702442169
Epoch: 35

Loss on hold-out set: 0.1603325439989567
MeanAbsoluteError value on hold-out data: 0.352169930934906
```

```
Epoch: 36
Loss on hold-out set: 0.16063254920144876
MeanAbsoluteError value on hold-out data: 0.3500418961048126
Epoch: 37
Loss on hold-out set: 0.1498755366106828
MeanAbsoluteError value on hold-out data: 0.3444063663482666
Epoch: 38

Loss on hold-out set: 0.14771428120632965
MeanAbsoluteError value on hold-out data: 0.33736681938171387
Epoch: 39
Loss on hold-out set: 0.15499310915668804
MeanAbsoluteError value on hold-out data: 0.3370697796344757
Epoch: 40

Loss on hold-out set: 0.14847538659969967
MeanAbsoluteError value on hold-out data: 0.33918458223342896
Epoch: 41
Loss on hold-out set: 0.14542863237361114
MeanAbsoluteError value on hold-out data: 0.3350461423397064
Epoch: 42
Loss on hold-out set: 0.14271238030865788
MeanAbsoluteError value on hold-out data: 0.3259514570236206
Epoch: 43

Loss on hold-out set: 0.13524411554137866
MeanAbsoluteError value on hold-out data: 0.31750714778900146
Epoch: 44
Loss on hold-out set: 0.1359652682642142
MeanAbsoluteError value on hold-out data: 0.32043200731277466
Epoch: 45

Loss on hold-out set: 0.14689689626296362
MeanAbsoluteError value on hold-out data: 0.3286217749118805
Epoch: 46
Loss on hold-out set: 0.13733561739325523
MeanAbsoluteError value on hold-out data: 0.32179728150367737
Epoch: 47
Loss on hold-out set: 0.13606099724769594
MeanAbsoluteError value on hold-out data: 0.32020050287246704
Epoch: 48
```

```
Loss on hold-out set: 0.13351751312613488
MeanAbsoluteError value on hold-out data: 0.3175945281982422
Epoch: 49
Loss on hold-out set: 0.13306795199712118
MeanAbsoluteError value on hold-out data: 0.3177848160266876
Epoch: 50

Loss on hold-out set: 0.13538302128513655
MeanAbsoluteError value on hold-out data: 0.3163302540779114
Epoch: 51
Loss on hold-out set: 0.12417131021618844
MeanAbsoluteError value on hold-out data: 0.3093986213207245
Epoch: 52
Loss on hold-out set: 0.1305441726744175
MeanAbsoluteError value on hold-out data: 0.30948564410209656
Epoch: 53

Loss on hold-out set: 0.11430580037335555
MeanAbsoluteError value on hold-out data: 0.28448623418807983
Epoch: 54
Loss on hold-out set: 0.118356843739748
MeanAbsoluteError value on hold-out data: 0.2890438139438629
Epoch: 55

Loss on hold-out set: 0.11924651518464088
MeanAbsoluteError value on hold-out data: 0.29544925689697266
Epoch: 56
Loss on hold-out set: 0.11480729647912086
MeanAbsoluteError value on hold-out data: 0.287399023771286
Epoch: 57
Loss on hold-out set: 0.11326847002531092
MeanAbsoluteError value on hold-out data: 0.28367483615875244
Epoch: 58

Loss on hold-out set: 0.11999072377880414
MeanAbsoluteError value on hold-out data: 0.2923401892185211
Epoch: 59
Loss on hold-out set: 0.1112199442088604
MeanAbsoluteError value on hold-out data: 0.2787458300590515
Epoch: 60
```

```
Loss on hold-out set: 0.10813958176722129
MeanAbsoluteError value on hold-out data: 0.27857837080955505
Epoch: 61
Loss on hold-out set: 0.10305735672513644
MeanAbsoluteError value on hold-out data: 0.2740250527858734
Epoch: 62
Loss on hold-out set: 0.1060432065029939
MeanAbsoluteError value on hold-out data: 0.27888041734695435
Epoch: 63

Loss on hold-out set: 0.10918688354392847
MeanAbsoluteError value on hold-out data: 0.2782539129257202
Epoch: 64
Loss on hold-out set: 0.09762860322992006
MeanAbsoluteError value on hold-out data: 0.2624378502368927
Epoch: 65

Loss on hold-out set: 0.10548085909336806
MeanAbsoluteError value on hold-out data: 0.27123546600341797
Epoch: 66
Loss on hold-out set: 0.1055884137749672
MeanAbsoluteError value on hold-out data: 0.27592459321022034
Epoch: 67
Loss on hold-out set: 0.10074084233492613
MeanAbsoluteError value on hold-out data: 0.2679758667945862
Epoch: 68

Loss on hold-out set: 0.09970588746170203
MeanAbsoluteError value on hold-out data: 0.26630470156669617
Epoch: 69
Loss on hold-out set: 0.09725817400341233
MeanAbsoluteError value on hold-out data: 0.25815561413764954
Epoch: 70

Loss on hold-out set: 0.09658900702993076
MeanAbsoluteError value on hold-out data: 0.26041653752326965
Epoch: 71
Loss on hold-out set: 0.09252015430480241
MeanAbsoluteError value on hold-out data: 0.25368717312812805
Epoch: 72
Loss on hold-out set: 0.09055833009382089
```

```
MeanAbsoluteError value on hold-out data: 0.2544904351234436
Epoch: 73

Loss on hold-out set: 0.09461871720850468
MeanAbsoluteError value on hold-out data: 0.259045273065567
Epoch: 74
Loss on hold-out set: 0.09031278791526953
MeanAbsoluteError value on hold-out data: 0.25373998284339905
Epoch: 75

Loss on hold-out set: 0.08995508751521508
MeanAbsoluteError value on hold-out data: 0.24881704151630402
Epoch: 76
Loss on hold-out set: 0.08897231246034304
MeanAbsoluteError value on hold-out data: 0.24953705072402954
Epoch: 77
Loss on hold-out set: 0.09186178315741321
MeanAbsoluteError value on hold-out data: 0.24444709718227386
Epoch: 78

Loss on hold-out set: 0.08468218344574173
MeanAbsoluteError value on hold-out data: 0.2424493432044983
Epoch: 79
Loss on hold-out set: 0.08393251985311508
MeanAbsoluteError value on hold-out data: 0.24407686293125153
Epoch: 80

Loss on hold-out set: 0.08196112391849358
MeanAbsoluteError value on hold-out data: 0.23579168319702148
Epoch: 81
Loss on hold-out set: 0.08582518395036459
MeanAbsoluteError value on hold-out data: 0.2474440187215805
Epoch: 82
Loss on hold-out set: 0.08238590677579244
MeanAbsoluteError value on hold-out data: 0.23706050217151642
Epoch: 83

Loss on hold-out set: 0.08490397156526645
MeanAbsoluteError value on hold-out data: 0.2350594848394394
Epoch: 84
Loss on hold-out set: 0.07470413010567427
```

```
MeanAbsoluteError value on hold-out data: 0.22822481393814087
Epoch: 85

Loss on hold-out set: 0.08025044437497854
MeanAbsoluteError value on hold-out data: 0.23252052068710327
Epoch: 86
Loss on hold-out set: 0.08123596301923196
MeanAbsoluteError value on hold-out data: 0.23313270509243011
Epoch: 87
Loss on hold-out set: 0.07685203830401102
MeanAbsoluteError value on hold-out data: 0.22685931622982025
Epoch: 88

Loss on hold-out set: 0.07800888904680808
MeanAbsoluteError value on hold-out data: 0.23235385119915009
Epoch: 89
Loss on hold-out set: 0.07442690281818311
MeanAbsoluteError value on hold-out data: 0.22727666795253754
Epoch: 90

Loss on hold-out set: 0.07561041830107569
MeanAbsoluteError value on hold-out data: 0.22379806637763977
Epoch: 91
Loss on hold-out set: 0.07478302032997211
MeanAbsoluteError value on hold-out data: 0.22379636764526367
Epoch: 92

Loss on hold-out set: 0.07091487204345565
MeanAbsoluteError value on hold-out data: 0.21760544180870056
Epoch: 93
Loss on hold-out set: 0.07260728001594544
MeanAbsoluteError value on hold-out data: 0.2184818536043167
Epoch: 94

Loss on hold-out set: 0.07028157185142239
MeanAbsoluteError value on hold-out data: 0.21329711377620697
Epoch: 95
Loss on hold-out set: 0.0716581254079938
MeanAbsoluteError value on hold-out data: 0.22127263247966766
Epoch: 96
```

```
Loss on hold-out set: 0.06670805980761846
MeanAbsoluteError value on hold-out data: 0.20967276394367218
Epoch: 97
Loss on hold-out set: 0.07199232848982016
MeanAbsoluteError value on hold-out data: 0.21996378898620605
Epoch: 98
Loss on hold-out set: 0.07098617265932262
MeanAbsoluteError value on hold-out data: 0.21461723744869232


Epoch: 99
Loss on hold-out set: 0.06271548236409823
MeanAbsoluteError value on hold-out data: 0.20475123822689056
Epoch: 100
Loss on hold-out set: 0.07239534086858233
MeanAbsoluteError value on hold-out data: 0.22008812427520752
Epoch: 101

Loss on hold-out set: 0.06539048079711696
MeanAbsoluteError value on hold-out data: 0.20256203413009644
Epoch: 102
Loss on hold-out set: 0.06726211549093326
MeanAbsoluteError value on hold-out data: 0.2085704654455185
Epoch: 103

Loss on hold-out set: 0.07155567219480873
MeanAbsoluteError value on hold-out data: 0.22174443304538727
Epoch: 104
Loss on hold-out set: 0.06810664355754853
MeanAbsoluteError value on hold-out data: 0.21365799009799957
Epoch: 105

Loss on hold-out set: 0.06317790612578392
MeanAbsoluteError value on hold-out data: 0.20819629728794098
Epoch: 106
Loss on hold-out set: 0.05947210389655083
MeanAbsoluteError value on hold-out data: 0.19844135642051697
Epoch: 107

Loss on hold-out set: 0.06413032041862607
MeanAbsoluteError value on hold-out data: 0.203513965010643
Epoch: 108
```

```
Loss on hold-out set: 0.06399431275824706
MeanAbsoluteError value on hold-out data: 0.19963321089744568
Epoch: 109

Loss on hold-out set: 0.059206965770572426
MeanAbsoluteError value on hold-out data: 0.19178280234336853
Epoch: 110
Loss on hold-out set: 0.06533711303025484
MeanAbsoluteError value on hold-out data: 0.20705968141555786
Epoch: 111

Loss on hold-out set: 0.05748963096179068
MeanAbsoluteError value on hold-out data: 0.1939716339111328
Epoch: 112
Loss on hold-out set: 0.06194800183176994
MeanAbsoluteError value on hold-out data: 0.20368489623069763
Epoch: 113

Loss on hold-out set: 0.05668996856858333
MeanAbsoluteError value on hold-out data: 0.19035464525222778
Epoch: 114
Loss on hold-out set: 0.07048153767517458
MeanAbsoluteError value on hold-out data: 0.21315127611160278
Epoch: 115

Loss on hold-out set: 0.062321512922644616
MeanAbsoluteError value on hold-out data: 0.2026681900024414
Epoch: 116
Loss on hold-out set: 0.059988689248760545
MeanAbsoluteError value on hold-out data: 0.19391782581806183
Epoch: 117
Loss on hold-out set: 0.06160474150441587
MeanAbsoluteError value on hold-out data: 0.20133192837238312
Epoch: 118

Loss on hold-out set: 0.06325789050509532
MeanAbsoluteError value on hold-out data: 0.2006843537092209
Epoch: 119
Loss on hold-out set: 0.05380929737662276
MeanAbsoluteError value on hold-out data: 0.18676181137561798
Epoch: 120
```

```
Loss on hold-out set: 0.06430875630428394
MeanAbsoluteError value on hold-out data: 0.20928873121738434
Epoch: 121
Loss on hold-out set: 0.061504287521044414
MeanAbsoluteError value on hold-out data: 0.2004118114709854
Epoch: 122

Loss on hold-out set: 0.061940945672492184
MeanAbsoluteError value on hold-out data: 0.20075736939907074
Epoch: 123
Loss on hold-out set: 0.05204645606999596
MeanAbsoluteError value on hold-out data: 0.18280340731143951
Epoch: 124

Loss on hold-out set: 0.061093960657405355
MeanAbsoluteError value on hold-out data: 0.20176702737808228
Epoch: 125
Loss on hold-out set: 0.06415652576833963
MeanAbsoluteError value on hold-out data: 0.20064489543437958
Epoch: 126
Loss on hold-out set: 0.06270615529889861
MeanAbsoluteError value on hold-out data: 0.20079083740711212
Epoch: 127

Loss on hold-out set: 0.05699838100001216
MeanAbsoluteError value on hold-out data: 0.18828390538692474
Epoch: 128
Loss on hold-out set: 0.056333617120981214
MeanAbsoluteError value on hold-out data: 0.1945248693227768
Returned to Spot: Validation loss: 0.056333617120981214
-----------------------------------------------

spotPython tuning: 0.04639910859366258 [####------] 44.04%


config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7087442992740111, 'lr_mult':
Epoch: 1
Loss on hold-out set: 0.48561254918575286
MeanAbsoluteError value on hold-out data: 0.6601287126541138
Epoch: 2
```

```
Loss on hold-out set: 0.4656550657749176
MeanAbsoluteError value on hold-out data: 0.6477956771850586
Epoch: 3
Loss on hold-out set: 0.4744591559966405
MeanAbsoluteError value on hold-out data: 0.6527643799781799
Epoch: 4

Loss on hold-out set: 0.46610934992631275
MeanAbsoluteError value on hold-out data: 0.6476074457168579
Epoch: 5
Loss on hold-out set: 0.46418376088142393
MeanAbsoluteError value on hold-out data: 0.6432791352272034
Epoch: 6

Loss on hold-out set: 0.4475542465845744
MeanAbsoluteError value on hold-out data: 0.6314013600349426
Epoch: 7
Loss on hold-out set: 0.44813569088776906
MeanAbsoluteError value on hold-out data: 0.6325330138206482
Epoch: 8

Loss on hold-out set: 0.44018202940622964
MeanAbsoluteError value on hold-out data: 0.6264689564704895
Epoch: 9
Loss on hold-out set: 0.4488838291168213
MeanAbsoluteError value on hold-out data: 0.6344067454338074
Epoch: 10
Loss on hold-out set: 0.443587534626325
MeanAbsoluteError value on hold-out data: 0.6277550458908081

Epoch: 11
Loss on hold-out set: 0.4223937809467316
MeanAbsoluteError value on hold-out data: 0.61029052734375
Epoch: 12
Loss on hold-out set: 0.42037948409716286
MeanAbsoluteError value on hold-out data: 0.6140298247337341
Epoch: 13

Loss on hold-out set: 0.432662800749143
MeanAbsoluteError value on hold-out data: 0.6201430559158325
Epoch: 14
```

```
Loss on hold-out set: 0.4116482440630595
MeanAbsoluteError value on hold-out data: 0.6061309576034546
Epoch: 15

Loss on hold-out set: 0.408054526646932
MeanAbsoluteError value on hold-out data: 0.6031426787376404
Epoch: 16
Loss on hold-out set: 0.4097358254591624
MeanAbsoluteError value on hold-out data: 0.6039685010910034
Epoch: 17
Loss on hold-out set: 0.4039056448141734
MeanAbsoluteError value on hold-out data: 0.5986093878746033
Epoch: 18

Loss on hold-out set: 0.3994675276676814
MeanAbsoluteError value on hold-out data: 0.5952192544937134
Epoch: 19
Loss on hold-out set: 0.38536679089069364
MeanAbsoluteError value on hold-out data: 0.5833777189254761
Epoch: 20

Loss on hold-out set: 0.3965980331103007
MeanAbsoluteError value on hold-out data: 0.5906690955162048
Epoch: 21
Loss on hold-out set: 0.3888008274634679
MeanAbsoluteError value on hold-out data: 0.5838396549224854
Epoch: 22
Loss on hold-out set: 0.38990020791689556
MeanAbsoluteError value on hold-out data: 0.588996946811676
Epoch: 23

Loss on hold-out set: 0.3893149075905482
MeanAbsoluteError value on hold-out data: 0.585660457611084
Epoch: 24

Loss on hold-out set: 0.3898914460341136
MeanAbsoluteError value on hold-out data: 0.5872249603271484
Epoch: 25
Loss on hold-out set: 0.39218292335669197
MeanAbsoluteError value on hold-out data: 0.5901839733123779
Epoch: 26
```

```
Loss on hold-out set: 0.36677463352680206
MeanAbsoluteError value on hold-out data: 0.5680286288261414
Epoch: 27

Loss on hold-out set: 0.35343380630016324
MeanAbsoluteError value on hold-out data: 0.5551128387451172
Epoch: 28
Loss on hold-out set: 0.371598649819692
MeanAbsoluteError value on hold-out data: 0.5709776878356934
Epoch: 29

Loss on hold-out set: 0.3601035432020823
MeanAbsoluteError value on hold-out data: 0.5592615008354187
Epoch: 30
Loss on hold-out set: 0.3600912647445997
MeanAbsoluteError value on hold-out data: 0.5606124401092529
Epoch: 31
Loss on hold-out set: 0.354334244231383
MeanAbsoluteError value on hold-out data: 0.5513344407081604
Epoch: 32

Loss on hold-out set: 0.3510493596394857
MeanAbsoluteError value on hold-out data: 0.5457513332366943
Epoch: 33

Loss on hold-out set: 0.35266880253950755
MeanAbsoluteError value on hold-out data: 0.5513311624526978
Epoch: 34
Loss on hold-out set: 0.3366212906440099
MeanAbsoluteError value on hold-out data: 0.5413623452186584
Epoch: 35
Loss on hold-out set: 0.3347357122103373
MeanAbsoluteError value on hold-out data: 0.5364261865615845
Epoch: 36
Loss on hold-out set: 0.3419606159130732
MeanAbsoluteError value on hold-out data: 0.5466994643211365
Epoch: 37

Loss on hold-out set: 0.30459636976321536
MeanAbsoluteError value on hold-out data: 0.509712815284729
Epoch: 38
```

```
Loss on hold-out set: 0.32169000307718915
MeanAbsoluteError value on hold-out data: 0.5264512896537781
Epoch: 39
Loss on hold-out set: 0.3031363226970037
MeanAbsoluteError value on hold-out data: 0.5057992935180664
Epoch: 40
Loss on hold-out set: 0.31460481146971386
MeanAbsoluteError value on hold-out data: 0.5210326313972473
Epoch: 41
Loss on hold-out set: 0.309381942152977
MeanAbsoluteError value on hold-out data: 0.5100197792053223
Epoch: 42

Loss on hold-out set: 0.3165831819176674
MeanAbsoluteError value on hold-out data: 0.5205982327461243

Epoch: 43
Loss on hold-out set: 0.29639031141996386
MeanAbsoluteError value on hold-out data: 0.5013554692268372
Epoch: 44
Loss on hold-out set: 0.2990839429696401
MeanAbsoluteError value on hold-out data: 0.5066534280776978
Epoch: 45
Loss on hold-out set: 0.3052319144209226
MeanAbsoluteError value on hold-out data: 0.5049403309822083
Epoch: 46
Loss on hold-out set: 0.2850462825099627
MeanAbsoluteError value on hold-out data: 0.4941045045852661
Epoch: 47

Loss on hold-out set: 0.2959007411201795
MeanAbsoluteError value on hold-out data: 0.501236617565155
Epoch: 48
Loss on hold-out set: 0.29566189885139466
MeanAbsoluteError value on hold-out data: 0.5019702315330505
Epoch: 49
Loss on hold-out set: 0.28752041310071946
MeanAbsoluteError value on hold-out data: 0.4903942942619324
Epoch: 50
Loss on hold-out set: 0.2782616032163302
MeanAbsoluteError value on hold-out data: 0.4837821424007416
Epoch: 51
```

```
Loss on hold-out set: 0.27364939322074255
MeanAbsoluteError value on hold-out data: 0.4792232811450958
Epoch: 52

Loss on hold-out set: 0.2783351317048073
MeanAbsoluteError value on hold-out data: 0.4843922555446625
Epoch: 53
Loss on hold-out set: 0.2721445022026698
MeanAbsoluteError value on hold-out data: 0.47381526231765747
Epoch: 54
Loss on hold-out set: 0.2657054643829664
MeanAbsoluteError value on hold-out data: 0.4693619906902313
Epoch: 55
Loss on hold-out set: 0.2612919816871484
MeanAbsoluteError value on hold-out data: 0.461309552192688
Epoch: 56
Loss on hold-out set: 0.2617003772656123
MeanAbsoluteError value on hold-out data: 0.4660097658634186
Epoch: 57

Loss on hold-out set: 0.2570703918735186
MeanAbsoluteError value on hold-out data: 0.45806387066841125
Epoch: 58
Loss on hold-out set: 0.24625082661708195
MeanAbsoluteError value on hold-out data: 0.445707768201828
Epoch: 59
Loss on hold-out set: 0.2543429962793986
MeanAbsoluteError value on hold-out data: 0.4583108127117157
Epoch: 60
Loss on hold-out set: 0.234558980067571
MeanAbsoluteError value on hold-out data: 0.434333860874176
Epoch: 61
Loss on hold-out set: 0.23638063435753187
MeanAbsoluteError value on hold-out data: 0.4366798996925354
Epoch: 62

Loss on hold-out set: 0.23662312294046084
MeanAbsoluteError value on hold-out data: 0.43695124983787537
Epoch: 63
Loss on hold-out set: 0.2272473399837812
MeanAbsoluteError value on hold-out data: 0.42728230357170105
Epoch: 64
```

```
Loss on hold-out set: 0.24544611155986787
MeanAbsoluteError value on hold-out data: 0.4464091360569
Epoch: 65
Loss on hold-out set: 0.23742025171717007
MeanAbsoluteError value on hold-out data: 0.43393921852111816
Epoch: 66
Loss on hold-out set: 0.21893515666325888
MeanAbsoluteError value on hold-out data: 0.4211944043636322
Epoch: 67

Loss on hold-out set: 0.2172196701169014
MeanAbsoluteError value on hold-out data: 0.41612136363983154
Epoch: 68
Loss on hold-out set: 0.23972858503460884
MeanAbsoluteError value on hold-out data: 0.44158807396888733
Epoch: 69
Loss on hold-out set: 0.22817183127005894
MeanAbsoluteError value on hold-out data: 0.4244648218154907
Epoch: 70
Loss on hold-out set: 0.21475031758348148
MeanAbsoluteError value on hold-out data: 0.41735443472862244
Epoch: 71
Loss on hold-out set: 0.23023893440763155
MeanAbsoluteError value on hold-out data: 0.42645901441574097
Epoch: 72

Loss on hold-out set: 0.22222234467665355
MeanAbsoluteError value on hold-out data: 0.41484442353248596
Epoch: 73
Loss on hold-out set: 0.20694712728261946
MeanAbsoluteError value on hold-out data: 0.4049626886844635
Epoch: 74
Loss on hold-out set: 0.20730023955305418
MeanAbsoluteError value on hold-out data: 0.40583714842796326
Epoch: 75
Loss on hold-out set: 0.2030915956199169
MeanAbsoluteError value on hold-out data: 0.3955698013305664
Epoch: 76
Loss on hold-out set: 0.2073092842598756
MeanAbsoluteError value on hold-out data: 0.4020952582359314
Epoch: 77
```

```
Loss on hold-out set: 0.1950391329328219
MeanAbsoluteError value on hold-out data: 0.39538514614105225
Epoch: 78
Loss on hold-out set: 0.18789053628842037
MeanAbsoluteError value on hold-out data: 0.37994515895843506
Epoch: 79
Loss on hold-out set: 0.2039595887561639
MeanAbsoluteError value on hold-out data: 0.39826667308807373
Epoch: 80
Loss on hold-out set: 0.18734791701038678
MeanAbsoluteError value on hold-out data: 0.37373805046081543
Epoch: 81
Loss on hold-out set: 0.19782180666923524
MeanAbsoluteError value on hold-out data: 0.3880494236946106
Epoch: 82

Loss on hold-out set: 0.1857390302916368
MeanAbsoluteError value on hold-out data: 0.3801935315132141
Epoch: 83
Loss on hold-out set: 0.18689914186795553
MeanAbsoluteError value on hold-out data: 0.3800090253353119
Epoch: 84
Loss on hold-out set: 0.19670138376454513
MeanAbsoluteError value on hold-out data: 0.38787680864334106
Epoch: 85
Loss on hold-out set: 0.1810042082766692
MeanAbsoluteError value on hold-out data: 0.3657298684120178
Epoch: 86
Loss on hold-out set: 0.1814559511343638
MeanAbsoluteError value on hold-out data: 0.377408504486084
Epoch: 87

Loss on hold-out set: 0.1549686481555303
MeanAbsoluteError value on hold-out data: 0.3384236693382263
Epoch: 88
Loss on hold-out set: 0.1624056357393662
MeanAbsoluteError value on hold-out data: 0.3532196581363678
Epoch: 89
Loss on hold-out set: 0.1651784204443296
MeanAbsoluteError value on hold-out data: 0.3535352051258087
Epoch: 90
Loss on hold-out set: 0.16074113825956982
```

```
MeanAbsoluteError value on hold-out data: 0.3500306308269501
Epoch: 91
Loss on hold-out set: 0.16538906916975976
MeanAbsoluteError value on hold-out data: 0.35273540019989014
Epoch: 92

Loss on hold-out set: 0.15986453801393508
MeanAbsoluteError value on hold-out data: 0.3466022312641144
Epoch: 93
Loss on hold-out set: 0.16151619454224905
MeanAbsoluteError value on hold-out data: 0.34851163625717163
Epoch: 94
Loss on hold-out set: 0.16013856550057728
MeanAbsoluteError value on hold-out data: 0.3505953252315521
Epoch: 95
Loss on hold-out set: 0.17031641284624735
MeanAbsoluteError value on hold-out data: 0.35273200273513794
Epoch: 96
Loss on hold-out set: 0.1610667654623588
MeanAbsoluteError value on hold-out data: 0.34774383902549744
Epoch: 97

Loss on hold-out set: 0.15737468615174294
MeanAbsoluteError value on hold-out data: 0.3407305181026459
Epoch: 98
Loss on hold-out set: 0.15003782498339813
MeanAbsoluteError value on hold-out data: 0.3376207649707794
Epoch: 99
Loss on hold-out set: 0.1452639573191603
MeanAbsoluteError value on hold-out data: 0.3244631290435791
Epoch: 100
Loss on hold-out set: 0.14429672572761773
MeanAbsoluteError value on hold-out data: 0.31780630350112915
Epoch: 101
Loss on hold-out set: 0.148374398847421
MeanAbsoluteError value on hold-out data: 0.32975083589553833
Epoch: 102

Loss on hold-out set: 0.14137463885049026
MeanAbsoluteError value on hold-out data: 0.3130667507648468
Epoch: 103
Loss on hold-out set: 0.13797326361139614
```

```
MeanAbsoluteError value on hold-out data: 0.3147241771221161
Epoch: 104
Loss on hold-out set: 0.14572436101734637
MeanAbsoluteError value on hold-out data: 0.32244873046875
Epoch: 105
Loss on hold-out set: 0.12937366797278324
MeanAbsoluteError value on hold-out data: 0.3012970983982086
Epoch: 106
Loss on hold-out set: 0.13862112037837504
MeanAbsoluteError value on hold-out data: 0.31787025928497314
Epoch: 107

Loss on hold-out set: 0.12379354012509187
MeanAbsoluteError value on hold-out data: 0.2996107339859009
Epoch: 108
Loss on hold-out set: 0.1294882393380006
MeanAbsoluteError value on hold-out data: 0.2996329665184021
Epoch: 109
Loss on hold-out set: 0.1271201322476069
MeanAbsoluteError value on hold-out data: 0.29808762669563293
Epoch: 110
Loss on hold-out set: 0.1239952409764131
MeanAbsoluteError value on hold-out data: 0.2985267639160156
Epoch: 111
Loss on hold-out set: 0.12405722753455241
MeanAbsoluteError value on hold-out data: 0.29403576254844666
Epoch: 112

Loss on hold-out set: 0.13400781589870653
MeanAbsoluteError value on hold-out data: 0.3109174370765686
Epoch: 113
Loss on hold-out set: 0.11935180731117725
MeanAbsoluteError value on hold-out data: 0.2871973216533661
Epoch: 114
Loss on hold-out set: 0.12135477307563027
MeanAbsoluteError value on hold-out data: 0.29509711265563965
Epoch: 115
Loss on hold-out set: 0.12359298615405957
MeanAbsoluteError value on hold-out data: 0.2957366704940796
Epoch: 116
Loss on hold-out set: 0.12840510169665018
MeanAbsoluteError value on hold-out data: 0.3028215765953064
```

```
Epoch: 117

Loss on hold-out set: 0.11454165605207284
MeanAbsoluteError value on hold-out data: 0.27664852142333984
Epoch: 118
Loss on hold-out set: 0.12137173257768154
MeanAbsoluteError value on hold-out data: 0.2905774712562561
Epoch: 119
Loss on hold-out set: 0.12344041674397886
MeanAbsoluteError value on hold-out data: 0.29096513986587524
Epoch: 120
Loss on hold-out set: 0.11066098705555003
MeanAbsoluteError value on hold-out data: 0.2755657434463501
Epoch: 121
Loss on hold-out set: 0.10315026662002007
MeanAbsoluteError value on hold-out data: 0.26064828038215637
Epoch: 122

Loss on hold-out set: 0.10014842649300894
MeanAbsoluteError value on hold-out data: 0.26006174087524414
Epoch: 123
Loss on hold-out set: 0.11516868448505799
MeanAbsoluteError value on hold-out data: 0.2794298529624939
Epoch: 124
Loss on hold-out set: 0.12050640799105167
MeanAbsoluteError value on hold-out data: 0.2928982377052307
Epoch: 125
Loss on hold-out set: 0.10905172591408094
MeanAbsoluteError value on hold-out data: 0.26785433292388916
Epoch: 126
Loss on hold-out set: 0.1142100529000163
MeanAbsoluteError value on hold-out data: 0.2760927081108093
Epoch: 127

Loss on hold-out set: 0.09675813965499401
MeanAbsoluteError value on hold-out data: 0.2633838355541229
Epoch: 128
Loss on hold-out set: 0.10447014401356379
MeanAbsoluteError value on hold-out data: 0.2691121995449066
Returned to Spot: Validation loss: 0.10447014401356379
---------------------------------------------
spotPython tuning: 0.04639910859366258 [#####-----] 54.94%
```

```
config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7217006614431124, 'lr_mult':
Epoch: 1
Loss on hold-out set: 0.2749998732407888
MeanAbsoluteError value on hold-out data: 0.4777847230434418
Epoch: 2
Loss on hold-out set: 0.29012555847565336
MeanAbsoluteError value on hold-out data: 0.49393269419670105
Epoch: 3
Loss on hold-out set: 0.27703566789627077
MeanAbsoluteError value on hold-out data: 0.47577905654907227
Epoch: 4

Loss on hold-out set: 0.2778885096311569
MeanAbsoluteError value on hold-out data: 0.47838059067726135
Epoch: 5
Loss on hold-out set: 0.2631704606612523
MeanAbsoluteError value on hold-out data: 0.4691464602947235
Epoch: 6
Loss on hold-out set: 0.2503685250878334
MeanAbsoluteError value on hold-out data: 0.45640936493873596
Epoch: 7
Loss on hold-out set: 0.24824609890580177
MeanAbsoluteError value on hold-out data: 0.45247742533683777
Epoch: 8
Loss on hold-out set: 0.2515953130523364
MeanAbsoluteError value on hold-out data: 0.45379048585891724

Epoch: 9
Loss on hold-out set: 0.25489505772789317
MeanAbsoluteError value on hold-out data: 0.4573937952518463
Epoch: 10
Loss on hold-out set: 0.25225471913814546
MeanAbsoluteError value on hold-out data: 0.4494582712650299
Epoch: 11
Loss on hold-out set: 0.2622668895125389
MeanAbsoluteError value on hold-out data: 0.46777501702308655
Epoch: 12

Loss on hold-out set: 0.2517642778158188
MeanAbsoluteError value on hold-out data: 0.4535439610481262
Epoch: 13
```

```
Loss on hold-out set: 0.23847479298710822
MeanAbsoluteError value on hold-out data: 0.4396958351135254
Epoch: 14
Loss on hold-out set: 0.2347253974278768
MeanAbsoluteError value on hold-out data: 0.4330016076564789
Epoch: 15
Loss on hold-out set: 0.23898951202630997
MeanAbsoluteError value on hold-out data: 0.43525949120521545
Epoch: 16

Loss on hold-out set: 0.2393980121612549
MeanAbsoluteError value on hold-out data: 0.4400302767753601
Epoch: 17
Loss on hold-out set: 0.23413503681619963
MeanAbsoluteError value on hold-out data: 0.4364916980266571
Epoch: 18
Loss on hold-out set: 0.2227270803352197
MeanAbsoluteError value on hold-out data: 0.42235511541366577
Epoch: 19
Loss on hold-out set: 0.2227867563565572
MeanAbsoluteError value on hold-out data: 0.42455390095710754
Epoch: 20
Loss on hold-out set: 0.2172207439939181
MeanAbsoluteError value on hold-out data: 0.41399773955345154
Epoch: 21

Loss on hold-out set: 0.21894272699952125
MeanAbsoluteError value on hold-out data: 0.4159725308418274
Epoch: 22
Loss on hold-out set: 0.2125512816508611
MeanAbsoluteError value on hold-out data: 0.4130830764770508
Epoch: 23
Loss on hold-out set: 0.21149157643318175
MeanAbsoluteError value on hold-out data: 0.4070804715156555
Epoch: 24
Loss on hold-out set: 0.22145173341035843
MeanAbsoluteError value on hold-out data: 0.41431668400764465
Epoch: 25
Loss on hold-out set: 0.21325780396660168
MeanAbsoluteError value on hold-out data: 0.4101986587047577
Epoch: 26
```

```
Loss on hold-out set: 0.2028936039408048
MeanAbsoluteError value on hold-out data: 0.4008003771305084
Epoch: 27
Loss on hold-out set: 0.20008720842500527
MeanAbsoluteError value on hold-out data: 0.3905985951423645
Epoch: 28
Loss on hold-out set: 0.20479173069198928
MeanAbsoluteError value on hold-out data: 0.39966729283332825
Epoch: 29
Loss on hold-out set: 0.19329876725872358
MeanAbsoluteError value on hold-out data: 0.3880835473537445
Epoch: 30

Loss on hold-out set: 0.17898577372233074
MeanAbsoluteError value on hold-out data: 0.3702908456325531
Epoch: 31
Loss on hold-out set: 0.1950500469903151
MeanAbsoluteError value on hold-out data: 0.39004865288734436
Epoch: 32
Loss on hold-out set: 0.20385997826854388
MeanAbsoluteError value on hold-out data: 0.3929128646850586
Epoch: 33
Loss on hold-out set: 0.19689367192486923
MeanAbsoluteError value on hold-out data: 0.3899540901184082
Epoch: 34

Loss on hold-out set: 0.18552611662695806
MeanAbsoluteError value on hold-out data: 0.3783356547355652
Epoch: 35
Loss on hold-out set: 0.18016687278946242
MeanAbsoluteError value on hold-out data: 0.37648341059684753
Epoch: 36
Loss on hold-out set: 0.18312092373768488
MeanAbsoluteError value on hold-out data: 0.37554362416267395
Epoch: 37
Loss on hold-out set: 0.17864166110754012
MeanAbsoluteError value on hold-out data: 0.37370485067367554
Epoch: 38

Loss on hold-out set: 0.18615861604611078
MeanAbsoluteError value on hold-out data: 0.37253648042678833
Epoch: 39
```

```
Loss on hold-out set: 0.18396560102701187
MeanAbsoluteError value on hold-out data: 0.3761656880378723
Epoch: 40
Loss on hold-out set: 0.16766173188885053
MeanAbsoluteError value on hold-out data: 0.3558332324028015
Epoch: 41
Loss on hold-out set: 0.17414283173779646
MeanAbsoluteError value on hold-out data: 0.36179280281066895
Epoch: 42
Loss on hold-out set: 0.16137918924291927
MeanAbsoluteError value on hold-out data: 0.3511313498020172
Epoch: 43

Loss on hold-out set: 0.16680396780371665
MeanAbsoluteError value on hold-out data: 0.35658395290374756
Epoch: 44
Loss on hold-out set: 0.16306020493308704
MeanAbsoluteError value on hold-out data: 0.3497214615345001
Epoch: 45
Loss on hold-out set: 0.17674616823593775
MeanAbsoluteError value on hold-out data: 0.3631138205528259
Epoch: 46
Loss on hold-out set: 0.16183376401662827
MeanAbsoluteError value on hold-out data: 0.34636884927749634
Epoch: 47
Loss on hold-out set: 0.16476230132083097
MeanAbsoluteError value on hold-out data: 0.35290291905403137
Epoch: 48

Loss on hold-out set: 0.158155849725008
MeanAbsoluteError value on hold-out data: 0.34415724873542786
Epoch: 49
Loss on hold-out set: 0.14665243425716956
MeanAbsoluteError value on hold-out data: 0.3326846957206726
Epoch: 50
Loss on hold-out set: 0.16000844654937585
MeanAbsoluteError value on hold-out data: 0.3431061804294586
Epoch: 51
Loss on hold-out set: 0.14103725743790468
MeanAbsoluteError value on hold-out data: 0.32216140627861023
Epoch: 52
Loss on hold-out set: 0.14262318583826225
```

```
MeanAbsoluteError value on hold-out data: 0.3259238004684448
Epoch: 53

Loss on hold-out set: 0.1432200588285923
MeanAbsoluteError value on hold-out data: 0.3211232125759125
Epoch: 54
Loss on hold-out set: 0.1338152250647545
MeanAbsoluteError value on hold-out data: 0.3155871629714966
Epoch: 55
Loss on hold-out set: 0.14209277888139088
MeanAbsoluteError value on hold-out data: 0.32253971695899963
Epoch: 56
Loss on hold-out set: 0.13948264725506307
MeanAbsoluteError value on hold-out data: 0.31719353795051575
Epoch: 57
Loss on hold-out set: 0.1295783001681169
MeanAbsoluteError value on hold-out data: 0.3034621477127075
Epoch: 58

Loss on hold-out set: 0.11757516544312238
MeanAbsoluteError value on hold-out data: 0.2847636938095093
Epoch: 59
Loss on hold-out set: 0.13233126835276685
MeanAbsoluteError value on hold-out data: 0.3073923587799072
Epoch: 60
Loss on hold-out set: 0.13176445926229158
MeanAbsoluteError value on hold-out data: 0.3075374960899353
Epoch: 61
Loss on hold-out set: 0.129004273985823
MeanAbsoluteError value on hold-out data: 0.30586516857147217
Epoch: 62
Loss on hold-out set: 0.12751921852429707
MeanAbsoluteError value on hold-out data: 0.3050995171070099
Epoch: 63

Loss on hold-out set: 0.11981345010300477
MeanAbsoluteError value on hold-out data: 0.28743335604667664
Epoch: 64
Loss on hold-out set: 0.12182914394885301
MeanAbsoluteError value on hold-out data: 0.29395028948783875
Epoch: 65
Loss on hold-out set: 0.12687747446199257
```

```
MeanAbsoluteError value on hold-out data: 0.29693153500556946
Epoch: 66
Loss on hold-out set: 0.12963441948095958
MeanAbsoluteError value on hold-out data: 0.304794579744339
Epoch: 67
Loss on hold-out set: 0.12351002298295498
MeanAbsoluteError value on hold-out data: 0.2980046272277832
Epoch: 68

Loss on hold-out set: 0.12114430957163373
MeanAbsoluteError value on hold-out data: 0.2952110171318054
Epoch: 69
Loss on hold-out set: 0.11499649081379175
MeanAbsoluteError value on hold-out data: 0.283300518989563
Epoch: 70
Loss on hold-out set: 0.10249377254396677
MeanAbsoluteError value on hold-out data: 0.26825690269470215
Epoch: 71
Loss on hold-out set: 0.12227390391131242
MeanAbsoluteError value on hold-out data: 0.2895278036594391
Epoch: 72
Loss on hold-out set: 0.11367014569540819
MeanAbsoluteError value on hold-out data: 0.27626046538352966
Epoch: 73

Loss on hold-out set: 0.12541239999234677
MeanAbsoluteError value on hold-out data: 0.2977859079837799
Epoch: 74
Loss on hold-out set: 0.11445950090885162
MeanAbsoluteError value on hold-out data: 0.28719452023506165
Epoch: 75
Loss on hold-out set: 0.10691655195007721
MeanAbsoluteError value on hold-out data: 0.2730734050273895
Epoch: 76
Loss on hold-out set: 0.1029083164781332
MeanAbsoluteError value on hold-out data: 0.26806628704071045
Epoch: 77
Loss on hold-out set: 0.10034175977110862
MeanAbsoluteError value on hold-out data: 0.2651735842227936

Epoch: 78
Loss on hold-out set: 0.10862618632614612
```

```
MeanAbsoluteError value on hold-out data: 0.26916834712028503
Epoch: 79
Loss on hold-out set: 0.10471301631381115
MeanAbsoluteError value on hold-out data: 0.2662200927734375
Epoch: 80
Loss on hold-out set: 0.10570808747783303
MeanAbsoluteError value on hold-out data: 0.271659255027771
Epoch: 81
Loss on hold-out set: 0.10954865664243699
MeanAbsoluteError value on hold-out data: 0.27525925636291504
Epoch: 82

Loss on hold-out set: 0.10498854036132495
MeanAbsoluteError value on hold-out data: 0.267935574054718
Epoch: 83
Loss on hold-out set: 0.09493146458019813
MeanAbsoluteError value on hold-out data: 0.25365665555000305
Epoch: 84
Loss on hold-out set: 0.09799711077163617
MeanAbsoluteError value on hold-out data: 0.25571075081825256
Epoch: 85
Loss on hold-out set: 0.08924267564589779
MeanAbsoluteError value on hold-out data: 0.24343086779117584
Epoch: 86
Loss on hold-out set: 0.09742830416808526
MeanAbsoluteError value on hold-out data: 0.2548725605010986 3
Epoch: 87

Loss on hold-out set: 0.0857781778027614
MeanAbsoluteError value on hold-out data: 0.23771697282791138
Epoch: 88
Loss on hold-out set: 0.10281246551622947
MeanAbsoluteError value on hold-out data: 0.26665568351745605
Epoch: 89
Loss on hold-out set: 0.10031385984271765
MeanAbsoluteError value on hold-out data: 0.25505560636520386
Epoch: 90
Loss on hold-out set: 0.10080106943069647
MeanAbsoluteError value on hold-out data: 0.2605421841144562
Epoch: 91
Loss on hold-out set: 0.08467132012049357
MeanAbsoluteError value on hold-out data: 0.23678570985794067
```

```
Epoch: 92

Loss on hold-out set: 0.10081282428155343
MeanAbsoluteError value on hold-out data: 0.2579290568828583
Epoch: 93
Loss on hold-out set: 0.09510040907810131
MeanAbsoluteError value on hold-out data: 0.24561573565006256
Epoch: 94
Loss on hold-out set: 0.08296716560299199
MeanAbsoluteError value on hold-out data: 0.23045453429222107
Epoch: 95
Loss on hold-out set: 0.0768975382298231
MeanAbsoluteError value on hold-out data: 0.22755546867847443
Epoch: 96
Loss on hold-out set: 0.09256326210995515
MeanAbsoluteError value on hold-out data: 0.24979451298713684
Epoch: 97

Loss on hold-out set: 0.08735459256917238
MeanAbsoluteError value on hold-out data: 0.2426442950963974
Epoch: 98
Loss on hold-out set: 0.09365049454073111
MeanAbsoluteError value on hold-out data: 0.2547400891780853
Epoch: 99
Loss on hold-out set: 0.08538635020454724
MeanAbsoluteError value on hold-out data: 0.23815752565860748
Epoch: 100
Loss on hold-out set: 0.0860448230492572
MeanAbsoluteError value on hold-out data: 0.23401273787021637
Epoch: 101
Loss on hold-out set: 0.07946378159647187
MeanAbsoluteError value on hold-out data: 0.22249041497707367
Epoch: 102

Loss on hold-out set: 0.07766171127557754
MeanAbsoluteError value on hold-out data: 0.22456812858581543
Epoch: 103
Loss on hold-out set: 0.07862454018555581
MeanAbsoluteError value on hold-out data: 0.22503143548965454
Epoch: 104
Loss on hold-out set: 0.08382669956733783
MeanAbsoluteError value on hold-out data: 0.23343126475811005
```

```
Epoch: 105
Loss on hold-out set: 0.08394127051035563
MeanAbsoluteError value on hold-out data: 0.23809915781021118
Epoch: 106
Loss on hold-out set: 0.08525303989648819
MeanAbsoluteError value on hold-out data: 0.233628511428833
Epoch: 107

Loss on hold-out set: 0.0813175085062782
MeanAbsoluteError value on hold-out data: 0.2346118986606598
Epoch: 108
Loss on hold-out set: 0.07776480212807656
MeanAbsoluteError value on hold-out data: 0.22580955922603607
Epoch: 109
Loss on hold-out set: 0.07643542454888423
MeanAbsoluteError value on hold-out data: 0.22704976797103882
Epoch: 110
Loss on hold-out set: 0.07252391708393892
MeanAbsoluteError value on hold-out data: 0.21296945214271545
Epoch: 111
Loss on hold-out set: 0.08424869023263454
MeanAbsoluteError value on hold-out data: 0.23687294125556946
Epoch: 112

Loss on hold-out set: 0.08023075049122175
MeanAbsoluteError value on hold-out data: 0.2283974438905716
Epoch: 113
Loss on hold-out set: 0.08010881123753885
MeanAbsoluteError value on hold-out data: 0.22856535017490387
Epoch: 114
Loss on hold-out set: 0.0791993955677996
MeanAbsoluteError value on hold-out data: 0.2247948944568634
Epoch: 115
Loss on hold-out set: 0.07531818947754801
MeanAbsoluteError value on hold-out data: 0.2251155823469162
Epoch: 116
Loss on hold-out set: 0.06733480046813686
MeanAbsoluteError value on hold-out data: 0.20943869650363922
Epoch: 117

Loss on hold-out set: 0.07447376097242038
MeanAbsoluteError value on hold-out data: 0.2200024127960205
```

```
Epoch: 118
Loss on hold-out set: 0.07177606638520956
MeanAbsoluteError value on hold-out data: 0.22084002196788788
Epoch: 119
Loss on hold-out set: 0.07494089080020785
MeanAbsoluteError value on hold-out data: 0.2165650874376297
Epoch: 120
Loss on hold-out set: 0.06818573777874311
MeanAbsoluteError value on hold-out data: 0.20755527913570404
Epoch: 121
Loss on hold-out set: 0.0819714124004046
MeanAbsoluteError value on hold-out data: 0.23130978643894196
Epoch: 122

Loss on hold-out set: 0.0741637799764673
MeanAbsoluteError value on hold-out data: 0.21778269112110138
Epoch: 123
Loss on hold-out set: 0.07815128008524577
MeanAbsoluteError value on hold-out data: 0.2246340662240982
Epoch: 124
Loss on hold-out set: 0.07981298763304949
MeanAbsoluteError value on hold-out data: 0.23145070672035217
Epoch: 125
Loss on hold-out set: 0.07115739398247872
MeanAbsoluteError value on hold-out data: 0.21342216432094574
Epoch: 126
Loss on hold-out set: 0.07625834198668599
MeanAbsoluteError value on hold-out data: 0.2260652631521225
Epoch: 127

Loss on hold-out set: 0.07313340834031502
MeanAbsoluteError value on hold-out data: 0.21780794858932495
Epoch: 128
Loss on hold-out set: 0.06748241989562909
MeanAbsoluteError value on hold-out data: 0.21120470762252808
Returned to Spot: Validation loss: 0.06748241989562909
-----------------------------------------------
spotPython tuning: 0.04639910859366258 [#######---] 65.91%


config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7200696116104506, 'lr_mult':
Epoch: 1
```

```
Loss on hold-out set: 0.24690172870953878
MeanAbsoluteError value on hold-out data: 0.4493919312953949
Epoch: 2
Loss on hold-out set: 0.23881064007679623
MeanAbsoluteError value on hold-out data: 0.4485740065574646
Epoch: 3
Loss on hold-out set: 0.24828021109104156
MeanAbsoluteError value on hold-out data: 0.45661258697509766
Epoch: 4

Loss on hold-out set: 0.23415555372834207
MeanAbsoluteError value on hold-out data: 0.44435104727745056
Epoch: 5
Loss on hold-out set: 0.23841003487507503
MeanAbsoluteError value on hold-out data: 0.4448983669281006
Epoch: 6
Loss on hold-out set: 0.24081891228755314
MeanAbsoluteError value on hold-out data: 0.44984135031700134
Epoch: 7
Loss on hold-out set: 0.22069094702601433
MeanAbsoluteError value on hold-out data: 0.4252682030200958
Epoch: 8
Loss on hold-out set: 0.22493098119894664
MeanAbsoluteError value on hold-out data: 0.42946502566337585
Epoch: 9

Loss on hold-out set: 0.2322356680035591
MeanAbsoluteError value on hold-out data: 0.4407205581665039
Epoch: 10
Loss on hold-out set: 0.21699334224065145
MeanAbsoluteError value on hold-out data: 0.42264524102211
Epoch: 11
Loss on hold-out set: 0.2232296753923098
MeanAbsoluteError value on hold-out data: 0.42585256695747375
Epoch: 12
Loss on hold-out set: 0.23009982640544574
MeanAbsoluteError value on hold-out data: 0.4334544837474823
Epoch: 13
Loss on hold-out set: 0.21782589942216873
MeanAbsoluteError value on hold-out data: 0.42252853512763977
Epoch: 14
```

```
Loss on hold-out set: 0.2083112131555875
MeanAbsoluteError value on hold-out data: 0.4143624007701874
Epoch: 15
Loss on hold-out set: 0.2130969914793968
MeanAbsoluteError value on hold-out data: 0.41666290163993835
Epoch: 16
Loss on hold-out set: 0.20900539845228194
MeanAbsoluteError value on hold-out data: 0.4112340211868286
Epoch: 17
Loss on hold-out set: 0.20169205263257026
MeanAbsoluteError value on hold-out data: 0.4030660390853882
Epoch: 18
Loss on hold-out set: 0.19784447362025578
MeanAbsoluteError value on hold-out data: 0.39618760347366333
Epoch: 19

Loss on hold-out set: 0.19767539391915004
MeanAbsoluteError value on hold-out data: 0.400494784116745
Epoch: 20
Loss on hold-out set: 0.21553875813881557
MeanAbsoluteError value on hold-out data: 0.4195266366004944
Epoch: 21
Loss on hold-out set: 0.19432139292359352
MeanAbsoluteError value on hold-out data: 0.3968161344528198
Epoch: 22
Loss on hold-out set: 0.18820034782091777
MeanAbsoluteError value on hold-out data: 0.3873283863067627
Epoch: 23
Loss on hold-out set: 0.1923510479927063
MeanAbsoluteError value on hold-out data: 0.3894788324832916
Epoch: 24

Loss on hold-out set: 0.1775541550417741
MeanAbsoluteError value on hold-out data: 0.37529850006103516
Epoch: 25
Loss on hold-out set: 0.18907605955998102
MeanAbsoluteError value on hold-out data: 0.3908468782901764
Epoch: 26
Loss on hold-out set: 0.17705620934565863
MeanAbsoluteError value on hold-out data: 0.37272197008132935
Epoch: 27
Loss on hold-out set: 0.17319050307075182
```

```
MeanAbsoluteError value on hold-out data: 0.3703584671020508
Epoch: 28
Loss on hold-out set: 0.1804600651562214
MeanAbsoluteError value on hold-out data: 0.37603989243507385
Epoch: 29

Loss on hold-out set: 0.17795878812670707
MeanAbsoluteError value on hold-out data: 0.37180978059768677
Epoch: 30
Loss on hold-out set: 0.17713411236802737
MeanAbsoluteError value on hold-out data: 0.373231440782547
Epoch: 31
Loss on hold-out set: 0.17537335311373076
MeanAbsoluteError value on hold-out data: 0.37497007846832275
Epoch: 32
Loss on hold-out set: 0.17051245254774888
MeanAbsoluteError value on hold-out data: 0.36349496245384216
Epoch: 33
Loss on hold-out set: 0.16857285474737485
MeanAbsoluteError value on hold-out data: 0.36455756425857544
Epoch: 34

Loss on hold-out set: 0.17980014085769652
MeanAbsoluteError value on hold-out data: 0.3793528974056244
Epoch: 35
Loss on hold-out set: 0.1659100780884425
MeanAbsoluteError value on hold-out data: 0.36020487546920776
Epoch: 36
Loss on hold-out set: 0.16647550294796626
MeanAbsoluteError value on hold-out data: 0.3633655905723572
Epoch: 37
Loss on hold-out set: 0.16150875916083654
MeanAbsoluteError value on hold-out data: 0.35579684376716614
Epoch: 38
Loss on hold-out set: 0.1502569007749359
MeanAbsoluteError value on hold-out data: 0.34060513973236084
Epoch: 39

Loss on hold-out set: 0.15219504569967587
MeanAbsoluteError value on hold-out data: 0.34433069825172424
Epoch: 40
Loss on hold-out set: 0.1519869338472684
```

```
MeanAbsoluteError value on hold-out data: 0.34330105781555176
Epoch: 41
Loss on hold-out set: 0.147127068862319
MeanAbsoluteError value on hold-out data: 0.33807653188705444
Epoch: 42
Loss on hold-out set: 0.1464157287031412
MeanAbsoluteError value on hold-out data: 0.3332354426383972
Epoch: 43
Loss on hold-out set: 0.14773187426229317
MeanAbsoluteError value on hold-out data: 0.33715084195137024
Epoch: 44

Loss on hold-out set: 0.15014834543069203
MeanAbsoluteError value on hold-out data: 0.3393550217151642
Epoch: 45
Loss on hold-out set: 0.14058227611084778
MeanAbsoluteError value on hold-out data: 0.32890328764915466
Epoch: 46
Loss on hold-out set: 0.1421006009976069
MeanAbsoluteError value on hold-out data: 0.32760751247406006
Epoch: 47
Loss on hold-out set: 0.14742230564355852
MeanAbsoluteError value on hold-out data: 0.3306167721748352
Epoch: 48
Loss on hold-out set: 0.13793550016979375
MeanAbsoluteError value on hold-out data: 0.3218837380409241
Epoch: 49

Loss on hold-out set: 0.14364162993927795
MeanAbsoluteError value on hold-out data: 0.32878491282463074
Epoch: 50
Loss on hold-out set: 0.13512991319100062
MeanAbsoluteError value on hold-out data: 0.319368451833725
Epoch: 51
Loss on hold-out set: 0.1283390757193168
MeanAbsoluteError value on hold-out data: 0.31130415201187134
Epoch: 52
Loss on hold-out set: 0.13458373074730237
MeanAbsoluteError value on hold-out data: 0.31207558512687683
Epoch: 53
Loss on hold-out set: 0.12280236574510733
MeanAbsoluteError value on hold-out data: 0.30096176266670227
```

```
Epoch: 54

Loss on hold-out set: 0.12024895208577314
MeanAbsoluteError value on hold-out data: 0.2991490364074707
Epoch: 55
Loss on hold-out set: 0.12813983084013064
MeanAbsoluteError value on hold-out data: 0.3068625032901764
Epoch: 56
Loss on hold-out set: 0.13431818969547749
MeanAbsoluteError value on hold-out data: 0.3146470785140991
Epoch: 57
Loss on hold-out set: 0.1259693632523219
MeanAbsoluteError value on hold-out data: 0.30501481890678406
Epoch: 58
Loss on hold-out set: 0.12374541215598583
MeanAbsoluteError value on hold-out data: 0.2994668185710907
Epoch: 59

Loss on hold-out set: 0.12180762569109599
MeanAbsoluteError value on hold-out data: 0.30189263820648193
Epoch: 60
Loss on hold-out set: 0.11599340913196404
MeanAbsoluteError value on hold-out data: 0.292724609375
Epoch: 61
Loss on hold-out set: 0.11788541659712791
MeanAbsoluteError value on hold-out data: 0.2940880060195923
Epoch: 62
Loss on hold-out set: 0.10971415852506955
MeanAbsoluteError value on hold-out data: 0.2818000912666321
Epoch: 63
Loss on hold-out set: 0.11100626240173976
MeanAbsoluteError value on hold-out data: 0.281169056892395
Epoch: 64

Loss on hold-out set: 0.11337102822959423
MeanAbsoluteError value on hold-out data: 0.2870216965675354
Epoch: 65
Loss on hold-out set: 0.1089573651055495
MeanAbsoluteError value on hold-out data: 0.28029829263687134
Epoch: 66
Loss on hold-out set: 0.11870053599278133
MeanAbsoluteError value on hold-out data: 0.2918497622013092
```

```
Epoch: 67
Loss on hold-out set: 0.11823911773661773
MeanAbsoluteError value on hold-out data: 0.29205840826034546
Epoch: 68

Loss on hold-out set: 0.11234368457148472
MeanAbsoluteError value on hold-out data: 0.2833210825920105
Epoch: 69
Loss on hold-out set: 0.10589860908687115
MeanAbsoluteError value on hold-out data: 0.272217333316803
Epoch: 70
Loss on hold-out set: 0.10620105753342311
MeanAbsoluteError value on hold-out data: 0.2760888934135437
Epoch: 71
Loss on hold-out set: 0.10147707497080168
MeanAbsoluteError value on hold-out data: 0.27039414644241333
Epoch: 72
Loss on hold-out set: 0.09816706580420335
MeanAbsoluteError value on hold-out data: 0.2605816125869751
Epoch: 73

Loss on hold-out set: 0.10581743724644184
MeanAbsoluteError value on hold-out data: 0.27233344316482544
Epoch: 74
Loss on hold-out set: 0.10530426702151696
MeanAbsoluteError value on hold-out data: 0.27163365483283997
Epoch: 75
Loss on hold-out set: 0.09518033641080062
MeanAbsoluteError value on hold-out data: 0.26298394799232483
Epoch: 76
Loss on hold-out set: 0.1007942275951306
MeanAbsoluteError value on hold-out data: 0.2621598243713379
Epoch: 77
Loss on hold-out set: 0.09349867954539756
MeanAbsoluteError value on hold-out data: 0.252341091632843
Epoch: 78

Loss on hold-out set: 0.09570204315086206
MeanAbsoluteError value on hold-out data: 0.2586529850959778
Epoch: 79
Loss on hold-out set: 0.09700682292381922
MeanAbsoluteError value on hold-out data: 0.2624250650405884
```

```
Epoch: 80
Loss on hold-out set: 0.09622910336901744
MeanAbsoluteError value on hold-out data: 0.25536590814590454
Epoch: 81
Loss on hold-out set: 0.09172550359119971
MeanAbsoluteError value on hold-out data: 0.25320208072662354
Epoch: 82

Loss on hold-out set: 0.08731577481453617
MeanAbsoluteError value on hold-out data: 0.24866046011447906
Epoch: 83
Loss on hold-out set: 0.08774630712655683
MeanAbsoluteError value on hold-out data: 0.24937957525253296
Epoch: 84
Loss on hold-out set: 0.09306466029336055
MeanAbsoluteError value on hold-out data: 0.2575649619102478
Epoch: 85
Loss on hold-out set: 0.08299748758474985
MeanAbsoluteError value on hold-out data: 0.23860114812850952
Epoch: 86

Loss on hold-out set: 0.0867322556860745
MeanAbsoluteError value on hold-out data: 0.23956863582134247
Epoch: 87
Loss on hold-out set: 0.08671760333081087
MeanAbsoluteError value on hold-out data: 0.24570032954216003
Epoch: 88
Loss on hold-out set: 0.08381839949637651
MeanAbsoluteError value on hold-out data: 0.24381111562252045
Epoch: 89
Loss on hold-out set: 0.08340655067935586
MeanAbsoluteError value on hold-out data: 0.23701316118240356
Epoch: 90

Loss on hold-out set: 0.07518481706579526
MeanAbsoluteError value on hold-out data: 0.2239668220281601
Epoch: 91
Loss on hold-out set: 0.08416543673723936
MeanAbsoluteError value on hold-out data: 0.2427976280450821
Epoch: 92
Loss on hold-out set: 0.08424525239194433
MeanAbsoluteError value on hold-out data: 0.23555834591388702
```

```
Epoch: 93
Loss on hold-out set: 0.08288785566886266
MeanAbsoluteError value on hold-out data: 0.23802699148654938
Epoch: 94
Loss on hold-out set: 0.07856455349673827
MeanAbsoluteError value on hold-out data: 0.2282823920249939
Epoch: 95

Loss on hold-out set: 0.07712075905874372
MeanAbsoluteError value on hold-out data: 0.22761215269565582
Epoch: 96
Loss on hold-out set: 0.07288030816862981
MeanAbsoluteError value on hold-out data: 0.22001513838768005
Epoch: 97
Loss on hold-out set: 0.07490386160711447
MeanAbsoluteError value on hold-out data: 0.22689765691757202
Epoch: 98
Loss on hold-out set: 0.07354807083184521
MeanAbsoluteError value on hold-out data: 0.22398974001407623
Epoch: 99

Loss on hold-out set: 0.07902316708738605
MeanAbsoluteError value on hold-out data: 0.2295057624578476
Epoch: 100
Loss on hold-out set: 0.07246085127194722
MeanAbsoluteError value on hold-out data: 0.21781733632087708
Epoch: 101
Loss on hold-out set: 0.07695352494716644
MeanAbsoluteError value on hold-out data: 0.227823406457901
Epoch: 102
Loss on hold-out set: 0.0749712073430419
MeanAbsoluteError value on hold-out data: 0.21951203048229218
Epoch: 103
Loss on hold-out set: 0.07173584404091041
MeanAbsoluteError value on hold-out data: 0.22801589965820312
Epoch: 104

Loss on hold-out set: 0.07550964480265974
MeanAbsoluteError value on hold-out data: 0.22719630599021912
Epoch: 105
Loss on hold-out set: 0.07605928141623736
MeanAbsoluteError value on hold-out data: 0.22991475462913513
```

```
Epoch: 106
Loss on hold-out set: 0.07833170892049869
MeanAbsoluteError value on hold-out data: 0.2307119220495224
Epoch: 107
Loss on hold-out set: 0.06676061619073152
MeanAbsoluteError value on hold-out data: 0.21023333072662354
Epoch: 108
Loss on hold-out set: 0.07064086282004912
MeanAbsoluteError value on hold-out data: 0.21588511765003204
Epoch: 109

Loss on hold-out set: 0.071833143979311
MeanAbsoluteError value on hold-out data: 0.21709363162517548
Epoch: 110
Loss on hold-out set: 0.07821209038297336
MeanAbsoluteError value on hold-out data: 0.2268393486738205
Epoch: 111
Loss on hold-out set: 0.07108532644808292
MeanAbsoluteError value on hold-out data: 0.22168205678462982
Epoch: 112
Loss on hold-out set: 0.06376442007099589
MeanAbsoluteError value on hold-out data: 0.20517870783805847
Epoch: 113

Loss on hold-out set: 0.06755603929360708
MeanAbsoluteError value on hold-out data: 0.20871929824352264
Epoch: 114
Loss on hold-out set: 0.06310827136660616
MeanAbsoluteError value on hold-out data: 0.20804645121097565
Epoch: 115
Loss on hold-out set: 0.07020261930301785
MeanAbsoluteError value on hold-out data: 0.21523508429527283
Epoch: 116
Loss on hold-out set: 0.07279949120556314
MeanAbsoluteError value on hold-out data: 0.2225387543439865
Epoch: 117
Loss on hold-out set: 0.06394167938580116
MeanAbsoluteError value on hold-out data: 0.20832468569278717
Epoch: 118

Loss on hold-out set: 0.061620774952073895
MeanAbsoluteError value on hold-out data: 0.2022363692522049
```

```
Epoch: 119
Loss on hold-out set: 0.06913819909095764
MeanAbsoluteError value on hold-out data: 0.21384872496128082
Epoch: 120
Loss on hold-out set: 0.06924613582591216
MeanAbsoluteError value on hold-out data: 0.20649167895317078
Epoch: 121
Loss on hold-out set: 0.05724332317709923
MeanAbsoluteError value on hold-out data: 0.19180570542812347
Epoch: 122
Loss on hold-out set: 0.069077972608308
MeanAbsoluteError value on hold-out data: 0.20859163999557495
Epoch: 123

Loss on hold-out set: 0.07209661601421734
MeanAbsoluteError value on hold-out data: 0.21499928832054138
Epoch: 124
Loss on hold-out set: 0.0598234390343229
MeanAbsoluteError value on hold-out data: 0.19426333904266357
Epoch: 125
Loss on hold-out set: 0.061000792315850656
MeanAbsoluteError value on hold-out data: 0.19545377790927887
Epoch: 126
Loss on hold-out set: 0.06357110558698574
MeanAbsoluteError value on hold-out data: 0.2079509049654007
Epoch: 127
Loss on hold-out set: 0.06169838111847639
MeanAbsoluteError value on hold-out data: 0.19998981058597565
Epoch: 128

Loss on hold-out set: 0.055079013680418336
MeanAbsoluteError value on hold-out data: 0.1915697157382965
Returned to Spot: Validation loss: 0.055079013680418336
-----------------------------------------------
spotPython tuning: 0.04639910859366258 [########--] 76.82%


config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7092110561836277, 'lr_mult':

Epoch: 1
Loss on hold-out set: 0.12286341118315856
```

```
MeanAbsoluteError value on hold-out data: 0.3059696555137634
Epoch: 2
Loss on hold-out set: 0.11466911320885022
MeanAbsoluteError value on hold-out data: 0.29247942566871643
Epoch: 3
Loss on hold-out set: 0.11531769971052806
MeanAbsoluteError value on hold-out data: 0.2945963144302368
Epoch: 4

Loss on hold-out set: 0.11719298221170903
MeanAbsoluteError value on hold-out data: 0.29594331979751587
Epoch: 5
Loss on hold-out set: 0.1149714257568121
MeanAbsoluteError value on hold-out data: 0.2933576703071594
Epoch: 6
Loss on hold-out set: 0.11693104159707825
MeanAbsoluteError value on hold-out data: 0.2969513237476349
Epoch: 7
Loss on hold-out set: 0.11202905483543873
MeanAbsoluteError value on hold-out data: 0.29071149230003357
Epoch: 8
Loss on hold-out set: 0.11087797263016304
MeanAbsoluteError value on hold-out data: 0.2867862582206726
Epoch: 9

Loss on hold-out set: 0.11450676957766215
MeanAbsoluteError value on hold-out data: 0.2968813180923462
Epoch: 10
Loss on hold-out set: 0.10900429231735567
MeanAbsoluteError value on hold-out data: 0.27959761023521423
Epoch: 11
Loss on hold-out set: 0.10541116550564766
MeanAbsoluteError value on hold-out data: 0.2862006425857544
Epoch: 12
Loss on hold-out set: 0.10161975515385469
MeanAbsoluteError value on hold-out data: 0.27523118257522583
Epoch: 13
Loss on hold-out set: 0.10875034251560768
MeanAbsoluteError value on hold-out data: 0.2798278331756592
Epoch: 14

Loss on hold-out set: 0.09929934557527303
```

```
MeanAbsoluteError value on hold-out data: 0.2682306170463562
Epoch: 15
Loss on hold-out set: 0.10664718988041083
MeanAbsoluteError value on hold-out data: 0.28116920590400696
Epoch: 16
Loss on hold-out set: 0.09738580351074537
MeanAbsoluteError value on hold-out data: 0.2593713700771332
Epoch: 17
Loss on hold-out set: 0.10047151324649652
MeanAbsoluteError value on hold-out data: 0.2725110650062561
Epoch: 18
Loss on hold-out set: 0.098810680223008
MeanAbsoluteError value on hold-out data: 0.26857253909111023
Epoch: 19

Loss on hold-out set: 0.09502189839879671
MeanAbsoluteError value on hold-out data: 0.2591228783130646
Epoch: 20
Loss on hold-out set: 0.09166792264208197
MeanAbsoluteError value on hold-out data: 0.256495863199234
Epoch: 21
Loss on hold-out set: 0.09986504493902126
MeanAbsoluteError value on hold-out data: 0.2690580189228058
Epoch: 22
Loss on hold-out set: 0.1031165468879044
MeanAbsoluteError value on hold-out data: 0.2758598327636719
Epoch: 23
Loss on hold-out set: 0.08988616275290648
MeanAbsoluteError value on hold-out data: 0.25401771068573
Epoch: 24

Loss on hold-out set: 0.08769656447072824
MeanAbsoluteError value on hold-out data: 0.2539735436439514
Epoch: 25
Loss on hold-out set: 0.08777967389672994
MeanAbsoluteError value on hold-out data: 0.24701906740665436
Epoch: 26
Loss on hold-out set: 0.08341400745014349
MeanAbsoluteError value on hold-out data: 0.24365699291229248
Epoch: 27
Loss on hold-out set: 0.08671841345727443
MeanAbsoluteError value on hold-out data: 0.2475455403327942
```

```
Epoch: 28
Loss on hold-out set: 0.08950890844066937
MeanAbsoluteError value on hold-out data: 0.25532281398773193
Epoch: 29

Loss on hold-out set: 0.08355513645956913
MeanAbsoluteError value on hold-out data: 0.2446780651807785
Epoch: 30
Loss on hold-out set: 0.0872903169070681
MeanAbsoluteError value on hold-out data: 0.2489856481552124
Epoch: 31
Loss on hold-out set: 0.07779089372605086
MeanAbsoluteError value on hold-out data: 0.2374095916748047
Epoch: 32
Loss on hold-out set: 0.07766892392188311
MeanAbsoluteError value on hold-out data: 0.23580558598041534
Epoch: 33
Loss on hold-out set: 0.07777289897203446
MeanAbsoluteError value on hold-out data: 0.23309959471225739
Epoch: 34

Loss on hold-out set: 0.07331880026186506
MeanAbsoluteError value on hold-out data: 0.2246531993150711
Epoch: 35
Loss on hold-out set: 0.07563724299271901
MeanAbsoluteError value on hold-out data: 0.2351987510919571
Epoch: 36
Loss on hold-out set: 0.07339710364739101
MeanAbsoluteError value on hold-out data: 0.22635801136493683
Epoch: 37
Loss on hold-out set: 0.07808756545186042
MeanAbsoluteError value on hold-out data: 0.2352152019739151
Epoch: 38
Loss on hold-out set: 0.07248032918820778
MeanAbsoluteError value on hold-out data: 0.2231505811214447
Epoch: 39

Loss on hold-out set: 0.0755367919554313
MeanAbsoluteError value on hold-out data: 0.22959640622138977
Epoch: 40
Loss on hold-out set: 0.07366671770811081
MeanAbsoluteError value on hold-out data: 0.22396138310432434
```

```
Epoch: 41
Loss on hold-out set: 0.06521295331418514
MeanAbsoluteError value on hold-out data: 0.21268326044082642
Epoch: 42
Loss on hold-out set: 0.07062450930476188
MeanAbsoluteError value on hold-out data: 0.22320103645324707
Epoch: 43
Loss on hold-out set: 0.07362819656108817
MeanAbsoluteError value on hold-out data: 0.22002094984054565
Epoch: 44

Loss on hold-out set: 0.07133747400095065
MeanAbsoluteError value on hold-out data: 0.22123652696609497
Epoch: 45
Loss on hold-out set: 0.06822700931380192
MeanAbsoluteError value on hold-out data: 0.2206994593143463
Epoch: 46
Loss on hold-out set: 0.06369926217322548
MeanAbsoluteError value on hold-out data: 0.20508961379528046
Epoch: 47
Loss on hold-out set: 0.06850846266994874
MeanAbsoluteError value on hold-out data: 0.21459892392158508
Epoch: 48
Loss on hold-out set: 0.06611997126291196
MeanAbsoluteError value on hold-out data: 0.21106141805648804
Epoch: 49

Loss on hold-out set: 0.06417390127976735
MeanAbsoluteError value on hold-out data: 0.2060673087835312
Epoch: 50
Loss on hold-out set: 0.06773479552318652
MeanAbsoluteError value on hold-out data: 0.21676106750965118
Epoch: 51
Loss on hold-out set: 0.05905349717785915
MeanAbsoluteError value on hold-out data: 0.2064208835363388
Epoch: 52
Loss on hold-out set: 0.06706768838999172
MeanAbsoluteError value on hold-out data: 0.21566317975521088
Epoch: 53
Loss on hold-out set: 0.07534244135022164
MeanAbsoluteError value on hold-out data: 0.2328977733850479
Epoch: 54
```

```
Loss on hold-out set: 0.0629405497883757
MeanAbsoluteError value on hold-out data: 0.2031998634338379
Epoch: 55
Loss on hold-out set: 0.054149524023135505
MeanAbsoluteError value on hold-out data: 0.18786296248435974
Epoch: 56
Loss on hold-out set: 0.062181535720204316
MeanAbsoluteError value on hold-out data: 0.20430846512317657
Epoch: 57
Loss on hold-out set: 0.060720928125083444
MeanAbsoluteError value on hold-out data: 0.19971856474876404
Epoch: 58
Loss on hold-out set: 0.06321180203308661
MeanAbsoluteError value on hold-out data: 0.20740817487239838
Epoch: 59

Loss on hold-out set: 0.06062436663856109
MeanAbsoluteError value on hold-out data: 0.19997812807559967
Epoch: 60
Loss on hold-out set: 0.06277590402712424
MeanAbsoluteError value on hold-out data: 0.20463895797729492
Epoch: 61
Loss on hold-out set: 0.06131170082837343
MeanAbsoluteError value on hold-out data: 0.20558930933475494
Epoch: 62
Loss on hold-out set: 0.05470236967007319
MeanAbsoluteError value on hold-out data: 0.19298779964447021
Epoch: 63
Loss on hold-out set: 0.059327118806540964
MeanAbsoluteError value on hold-out data: 0.1968153864145279
Epoch: 64

Loss on hold-out set: 0.061344026513397695
MeanAbsoluteError value on hold-out data: 0.20160531997680664
Epoch: 65
Loss on hold-out set: 0.05518076268645624
MeanAbsoluteError value on hold-out data: 0.18924438953399658
Epoch: 66
Loss on hold-out set: 0.053893439397215845
MeanAbsoluteError value on hold-out data: 0.18789078295230865
Epoch: 67
Loss on hold-out set: 0.05700304035097361
```

```
MeanAbsoluteError value on hold-out data: 0.19770072400569916
Epoch: 68
Loss on hold-out set: 0.05836493181064725
MeanAbsoluteError value on hold-out data: 0.19548681378364563
Epoch: 69

Loss on hold-out set: 0.058047964113454024
MeanAbsoluteError value on hold-out data: 0.1965089738368988
Epoch: 70
Loss on hold-out set: 0.0559702375655373
MeanAbsoluteError value on hold-out data: 0.19079281389713287
Epoch: 71
Loss on hold-out set: 0.04956521661952138
MeanAbsoluteError value on hold-out data: 0.17983151972293854
Epoch: 72
Loss on hold-out set: 0.05633196409791708
MeanAbsoluteError value on hold-out data: 0.19315966963768005
Epoch: 73
Loss on hold-out set: 0.054159226516882576
MeanAbsoluteError value on hold-out data: 0.19038048386573792
Epoch: 74

Loss on hold-out set: 0.05348325526341796
MeanAbsoluteError value on hold-out data: 0.18646767735481262
Epoch: 75
Loss on hold-out set: 0.060948677994310856
MeanAbsoluteError value on hold-out data: 0.1944063901901245
Epoch: 76
Loss on hold-out set: 0.0575682780581216
MeanAbsoluteError value on hold-out data: 0.1926807165145874
Epoch: 77
Loss on hold-out set: 0.05281103514134884
MeanAbsoluteError value on hold-out data: 0.18576139211654663
Epoch: 78
Loss on hold-out set: 0.05461700015390913
MeanAbsoluteError value on hold-out data: 0.189850851893425
Epoch: 79

Loss on hold-out set: 0.05436396688843767
MeanAbsoluteError value on hold-out data: 0.18884752690792084
Epoch: 80
Loss on hold-out set: 0.04807945214211941
```

```
MeanAbsoluteError value on hold-out data: 0.17714187502861023
Epoch: 81
Loss on hold-out set: 0.05210892155766487
MeanAbsoluteError value on hold-out data: 0.180259570479393
Epoch: 82
Loss on hold-out set: 0.051330716752757626
MeanAbsoluteError value on hold-out data: 0.1775350123643875
Epoch: 83
Loss on hold-out set: 0.056243406906723976
MeanAbsoluteError value on hold-out data: 0.19821956753730774
Epoch: 84

Loss on hold-out set: 0.05356542516499758
MeanAbsoluteError value on hold-out data: 0.18789581954479218
Epoch: 85
Loss on hold-out set: 0.058348053737233084
MeanAbsoluteError value on hold-out data: 0.191201850771904
Epoch: 86
Loss on hold-out set: 0.05457960548189779
MeanAbsoluteError value on hold-out data: 0.18755561113357544
Epoch: 87
Loss on hold-out set: 0.05158109641323487
MeanAbsoluteError value on hold-out data: 0.18200558423995972
Epoch: 88
Loss on hold-out set: 0.048326104714845615
MeanAbsoluteError value on hold-out data: 0.17733976244926453
Epoch: 89

Loss on hold-out set: 0.05273409237464269
MeanAbsoluteError value on hold-out data: 0.18366678059101105
Epoch: 90
Loss on hold-out set: 0.050014740275219086
MeanAbsoluteError value on hold-out data: 0.18401309847831726
Epoch: 91
Loss on hold-out set: 0.05035032843550046
MeanAbsoluteError value on hold-out data: 0.18106114864349365
Epoch: 92
Loss on hold-out set: 0.052229866640021404
MeanAbsoluteError value on hold-out data: 0.18489019572734833
Epoch: 93
Loss on hold-out set: 0.04279772737373908
MeanAbsoluteError value on hold-out data: 0.16285666823387146
```

```
Epoch: 94

Loss on hold-out set: 0.04409949530847371
MeanAbsoluteError value on hold-out data: 0.17201778292655945
Epoch: 95
Loss on hold-out set: 0.04647642247999708
MeanAbsoluteError value on hold-out data: 0.1749236136674881
Epoch: 96
Loss on hold-out set: 0.05129593464856346
MeanAbsoluteError value on hold-out data: 0.18301968276500702
Epoch: 97
Loss on hold-out set: 0.05083345650384823
MeanAbsoluteError value on hold-out data: 0.1846304088830948
Epoch: 98
Loss on hold-out set: 0.04398206919431687
MeanAbsoluteError value on hold-out data: 0.1680206060409546
Epoch: 99

Loss on hold-out set: 0.04777846717275679
MeanAbsoluteError value on hold-out data: 0.1743806153535843
Epoch: 100
Loss on hold-out set: 0.05534113086294383
MeanAbsoluteError value on hold-out data: 0.18875665962696075
Epoch: 101
Loss on hold-out set: 0.0486111234066387
MeanAbsoluteError value on hold-out data: 0.17853280901908875
Epoch: 102
Loss on hold-out set: 0.05148247616675993
MeanAbsoluteError value on hold-out data: 0.18050967156887054
Epoch: 103
Loss on hold-out set: 0.04281619309990977
MeanAbsoluteError value on hold-out data: 0.1669241338968277
Epoch: 104

Loss on hold-out set: 0.04990214374537269
MeanAbsoluteError value on hold-out data: 0.17840449512004852
Epoch: 105
Loss on hold-out set: 0.04485986603423953
MeanAbsoluteError value on hold-out data: 0.17092247307300568
Epoch: 106
Loss on hold-out set: 0.04207291005800168
MeanAbsoluteError value on hold-out data: 0.16475600004196167
```

```
Epoch: 107
Loss on hold-out set: 0.042767651837008695
MeanAbsoluteError value on hold-out data: 0.16629943251609802
Epoch: 108
Loss on hold-out set: 0.04313744634700318
MeanAbsoluteError value on hold-out data: 0.16913120448589325
Epoch: 109

Loss on hold-out set: 0.04661897163838148
MeanAbsoluteError value on hold-out data: 0.1743515133857727
Epoch: 110
Loss on hold-out set: 0.047544334270060065
MeanAbsoluteError value on hold-out data: 0.1707562506198883
Epoch: 111
Loss on hold-out set: 0.04464111010233561
MeanAbsoluteError value on hold-out data: 0.17291301488876343
Epoch: 112
Loss on hold-out set: 0.04812764296929042
MeanAbsoluteError value on hold-out data: 0.17644740641117096
Epoch: 113
Loss on hold-out set: 0.04336316682281904
MeanAbsoluteError value on hold-out data: 0.16640347242355347
Epoch: 114

Loss on hold-out set: 0.04368558365851641
MeanAbsoluteError value on hold-out data: 0.16962930560112
Epoch: 115
Loss on hold-out set: 0.047486834606776634
MeanAbsoluteError value on hold-out data: 0.17711012065410614
Epoch: 116
Loss on hold-out set: 0.04824012912809849
MeanAbsoluteError value on hold-out data: 0.17903578281402588
Epoch: 117
Loss on hold-out set: 0.04511911359305183
MeanAbsoluteError value on hold-out data: 0.16991247236728668
Epoch: 118

Loss on hold-out set: 0.045086632082238794
MeanAbsoluteError value on hold-out data: 0.17297524213790894
Epoch: 119
Loss on hold-out set: 0.04380826192907989
MeanAbsoluteError value on hold-out data: 0.1656121164560318
```

```
Epoch: 120
Loss on hold-out set: 0.04826211493462324
MeanAbsoluteError value on hold-out data: 0.17462821304798126
Epoch: 121
Loss on hold-out set: 0.049946609611312545
MeanAbsoluteError value on hold-out data: 0.17752189934253693
Epoch: 122

Loss on hold-out set: 0.04143751903437078
MeanAbsoluteError value on hold-out data: 0.15749084949493408
Epoch: 123
Loss on hold-out set: 0.04816438486178716
MeanAbsoluteError value on hold-out data: 0.18186721205711365
Epoch: 124
Loss on hold-out set: 0.046949107165758806
MeanAbsoluteError value on hold-out data: 0.17528849840164185
Epoch: 125
Loss on hold-out set: 0.04735182526948241
MeanAbsoluteError value on hold-out data: 0.17583443224430084
Epoch: 126

Loss on hold-out set: 0.045463941792647046
MeanAbsoluteError value on hold-out data: 0.17165595293045044
Epoch: 127
Loss on hold-out set: 0.046769995242357254
MeanAbsoluteError value on hold-out data: 0.17542260885238647
Epoch: 128
Loss on hold-out set: 0.03844679653334121
MeanAbsoluteError value on hold-out data: 0.15682654082775116
Returned to Spot: Validation loss: 0.03844679653334121
-----------------------------------------------
spotPython tuning: 0.03844679653334121 [#########-] 87.87%


config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7056230478240827, 'lr_mult':
Epoch: 1
Loss on hold-out set: 0.2855568004647891
MeanAbsoluteError value on hold-out data: 0.49800291657447815
Epoch: 2
Loss on hold-out set: 0.28591975927352903
MeanAbsoluteError value on hold-out data: 0.4977775514125824
Epoch: 3
```

```
Loss on hold-out set: 0.2889871002237002
MeanAbsoluteError value on hold-out data: 0.5044019818305969
Epoch: 4

Loss on hold-out set: 0.28915584882100426
MeanAbsoluteError value on hold-out data: 0.501556932926178
Epoch: 5
Loss on hold-out set: 0.27497903774182003
MeanAbsoluteError value on hold-out data: 0.4892980456352234
Epoch: 6
Loss on hold-out set: 0.28281236002842586
MeanAbsoluteError value on hold-out data: 0.4968753457069397
Epoch: 7
Loss on hold-out set: 0.27615631421407066
MeanAbsoluteError value on hold-out data: 0.4892692565917969
Epoch: 8

Loss on hold-out set: 0.27624026983976363
MeanAbsoluteError value on hold-out data: 0.4904369115829468
Epoch: 9
Loss on hold-out set: 0.2670411217212677
MeanAbsoluteError value on hold-out data: 0.47787344455718994
Epoch: 10
Loss on hold-out set: 0.2629938717683156
MeanAbsoluteError value on hold-out data: 0.47509530186653137
Epoch: 11
Loss on hold-out set: 0.27119260330994927
MeanAbsoluteError value on hold-out data: 0.48415058851242065
Epoch: 12
Loss on hold-out set: 0.26451801935831704
MeanAbsoluteError value on hold-out data: 0.47869986295700073
Epoch: 13

Loss on hold-out set: 0.2620774105191231
MeanAbsoluteError value on hold-out data: 0.47643357515335083
Epoch: 14
Loss on hold-out set: 0.26409510046243667
MeanAbsoluteError value on hold-out data: 0.47531449794769287
Epoch: 15
Loss on hold-out set: 0.24951520025730134
MeanAbsoluteError value on hold-out data: 0.46503692865371704
Epoch: 16
```

```
Loss on hold-out set: 0.25821930249532066
MeanAbsoluteError value on hold-out data: 0.4684925973415375
Epoch: 17
Loss on hold-out set: 0.2525474368532499
MeanAbsoluteError value on hold-out data: 0.4671626687049866
Epoch: 18

Loss on hold-out set: 0.25200520108143487
MeanAbsoluteError value on hold-out data: 0.46493661403656006
Epoch: 19
Loss on hold-out set: 0.2482808029651642
MeanAbsoluteError value on hold-out data: 0.4576668441295624
Epoch: 20
Loss on hold-out set: 0.23982814460992813
MeanAbsoluteError value on hold-out data: 0.45322316884994507
Epoch: 21
Loss on hold-out set: 0.2330217792590459
MeanAbsoluteError value on hold-out data: 0.4459172189235687
Epoch: 22
Loss on hold-out set: 0.24169275199373563
MeanAbsoluteError value on hold-out data: 0.4498368203639984
Epoch: 23

Loss on hold-out set: 0.2326607236266136
MeanAbsoluteError value on hold-out data: 0.4433330297470093
Epoch: 24
Loss on hold-out set: 0.2352370063463847
MeanAbsoluteError value on hold-out data: 0.44610345363616943
Epoch: 25
Loss on hold-out set: 0.22985230018695196
MeanAbsoluteError value on hold-out data: 0.4413178563117981
Epoch: 26
Loss on hold-out set: 0.23917585571606953
MeanAbsoluteError value on hold-out data: 0.448739618062973
Epoch: 27
Loss on hold-out set: 0.23335665285587312
MeanAbsoluteError value on hold-out data: 0.4428660571575165
Epoch: 28

Loss on hold-out set: 0.2349590077996254
MeanAbsoluteError value on hold-out data: 0.44522085785865784
Epoch: 29
```

```
Loss on hold-out set: 0.2191560254494349
MeanAbsoluteError value on hold-out data: 0.4315059781074524
Epoch: 30
Loss on hold-out set: 0.21356808602809907
MeanAbsoluteError value on hold-out data: 0.42262792587280273
Epoch: 31
Loss on hold-out set: 0.21370641102393467
MeanAbsoluteError value on hold-out data: 0.4247739911079407
Epoch: 32

Loss on hold-out set: 0.22564464911818505
MeanAbsoluteError value on hold-out data: 0.4331310987472534
Epoch: 33
Loss on hold-out set: 0.20809950917959213
MeanAbsoluteError value on hold-out data: 0.4159935414791107
Epoch: 34
Loss on hold-out set: 0.2025195882221063
MeanAbsoluteError value on hold-out data: 0.410992294549942
Epoch: 35
Loss on hold-out set: 0.20952180668711662
MeanAbsoluteError value on hold-out data: 0.4092848300933838
Epoch: 36
Loss on hold-out set: 0.2081537232796351
MeanAbsoluteError value on hold-out data: 0.4170719385147095
Epoch: 37

Loss on hold-out set: 0.21304965322216352
MeanAbsoluteError value on hold-out data: 0.41782456636428833
Epoch: 38
Loss on hold-out set: 0.21236986150344214
MeanAbsoluteError value on hold-out data: 0.41708904504776
Epoch: 39
Loss on hold-out set: 0.1978098734219869
MeanAbsoluteError value on hold-out data: 0.40416353940963745
Epoch: 40
Loss on hold-out set: 0.19991717477639517
MeanAbsoluteError value on hold-out data: 0.4071972966194153
Epoch: 41
Loss on hold-out set: 0.20073663413524628
MeanAbsoluteError value on hold-out data: 0.40646374225616455
Epoch: 42
```

```
Loss on hold-out set: 0.19932649210095404
MeanAbsoluteError value on hold-out data: 0.4032769203186035
Epoch: 43
Loss on hold-out set: 0.19221839313705763
MeanAbsoluteError value on hold-out data: 0.39711451530456543
Epoch: 44
Loss on hold-out set: 0.1904057299097379
MeanAbsoluteError value on hold-out data: 0.3917444050312042
Epoch: 45
Loss on hold-out set: 0.19464682122071583
MeanAbsoluteError value on hold-out data: 0.3999422788619995
Epoch: 46
Loss on hold-out set: 0.18069197888175648
MeanAbsoluteError value on hold-out data: 0.3825882077217102
Epoch: 47

Loss on hold-out set: 0.18363085647424063
MeanAbsoluteError value on hold-out data: 0.3841559886932373
Epoch: 48
Loss on hold-out set: 0.1835610447327296
MeanAbsoluteError value on hold-out data: 0.3851867616176605
Epoch: 49
Loss on hold-out set: 0.17698139098783333
MeanAbsoluteError value on hold-out data: 0.37876206636428833
Epoch: 50
Loss on hold-out set: 0.17303862313429513
MeanAbsoluteError value on hold-out data: 0.3707846403121948
Epoch: 51

Loss on hold-out set: 0.17960032830635705
MeanAbsoluteError value on hold-out data: 0.38155698776245117
Epoch: 52
Loss on hold-out set: 0.1707546642422676
MeanAbsoluteError value on hold-out data: 0.37144899368286133
Epoch: 53
Loss on hold-out set: 0.1727485677599907
MeanAbsoluteError value on hold-out data: 0.3690151870250702
Epoch: 54
Loss on hold-out set: 0.17831180060903232
MeanAbsoluteError value on hold-out data: 0.3776220679283142
Epoch: 55
Loss on hold-out set: 0.16663383096456527
```

```
MeanAbsoluteError value on hold-out data: 0.3624953329563141
Epoch: 56

Loss on hold-out set: 0.16672115775446097
MeanAbsoluteError value on hold-out data: 0.36443355679512024
Epoch: 57
Loss on hold-out set: 0.16060447628299396
MeanAbsoluteError value on hold-out data: 0.354129433631897
Epoch: 58
Loss on hold-out set: 0.1584121482570966
MeanAbsoluteError value on hold-out data: 0.3516997992992401
Epoch: 59
Loss on hold-out set: 0.16101721848050754
MeanAbsoluteError value on hold-out data: 0.3549184203147888
Epoch: 60
Loss on hold-out set: 0.1538843030234178
MeanAbsoluteError value on hold-out data: 0.3453381061553955
Epoch: 61

Loss on hold-out set: 0.15682731007536252
MeanAbsoluteError value on hold-out data: 0.35163190960884094
Epoch: 62
Loss on hold-out set: 0.15722716656823954
MeanAbsoluteError value on hold-out data: 0.35157909989356995
Epoch: 63
Loss on hold-out set: 0.1546450964361429
MeanAbsoluteError value on hold-out data: 0.3444656431674957
Epoch: 64
Loss on hold-out set: 0.15028763716419538
MeanAbsoluteError value on hold-out data: 0.34631767868995667
Epoch: 65
Loss on hold-out set: 0.15452587261795997
MeanAbsoluteError value on hold-out data: 0.3486153781414032
Epoch: 66

Loss on hold-out set: 0.14743220706780752
MeanAbsoluteError value on hold-out data: 0.3356161117553711
Epoch: 67
Loss on hold-out set: 0.15154028033216793
MeanAbsoluteError value on hold-out data: 0.3439098596572876
Epoch: 68
Loss on hold-out set: 0.14527688513199488
```

```
MeanAbsoluteError value on hold-out data: 0.3366928994655609
Epoch: 69
Loss on hold-out set: 0.13672053078810373
MeanAbsoluteError value on hold-out data: 0.3267894983291626
Epoch: 70
Loss on hold-out set: 0.139720459630092
MeanAbsoluteError value on hold-out data: 0.3259495496749878
Epoch: 71

Loss on hold-out set: 0.1449330744644006
MeanAbsoluteError value on hold-out data: 0.3374074697494507
Epoch: 72
Loss on hold-out set: 0.13594753585755825
MeanAbsoluteError value on hold-out data: 0.3206258714199066
Epoch: 73
Loss on hold-out set: 0.12806232213974
MeanAbsoluteError value on hold-out data: 0.3091526925563812
Epoch: 74
Loss on hold-out set: 0.12416628800332546
MeanAbsoluteError value on hold-out data: 0.3056424856185913
Epoch: 75
Loss on hold-out set: 0.13064577934642632
MeanAbsoluteError value on hold-out data: 0.3142428696155548
Epoch: 76

Loss on hold-out set: 0.13038267247378826
MeanAbsoluteError value on hold-out data: 0.3146244287490845
Epoch: 77
Loss on hold-out set: 0.13286069214344023
MeanAbsoluteError value on hold-out data: 0.3152148127555847
Epoch: 78
Loss on hold-out set: 0.12452767478923003
MeanAbsoluteError value on hold-out data: 0.3049267530441284
Epoch: 79
Loss on hold-out set: 0.11982161531845728
MeanAbsoluteError value on hold-out data: 0.29914000630378723
Epoch: 80
Loss on hold-out set: 0.12575064269204936
MeanAbsoluteError value on hold-out data: 0.3038315773010254
Epoch: 81

Loss on hold-out set: 0.11778266752759615
```

```
MeanAbsoluteError value on hold-out data: 0.29570531845092773
Epoch: 82
Loss on hold-out set: 0.1071585230777661
MeanAbsoluteError value on hold-out data: 0.28233614563941956
Epoch: 83
Loss on hold-out set: 0.1220165707791845
MeanAbsoluteError value on hold-out data: 0.3030920624732971
Epoch: 84
Loss on hold-out set: 0.11022455288718144
MeanAbsoluteError value on hold-out data: 0.2832968235015869
Epoch: 85
Loss on hold-out set: 0.1140541053811709
MeanAbsoluteError value on hold-out data: 0.28540223836898804
Epoch: 86

Loss on hold-out set: 0.10382449521372715
MeanAbsoluteError value on hold-out data: 0.273856520652771
Epoch: 87
Loss on hold-out set: 0.10674035172909498
MeanAbsoluteError value on hold-out data: 0.27956724166870117
Epoch: 88
Loss on hold-out set: 0.11222860946009557
MeanAbsoluteError value on hold-out data: 0.2838582992553711
Epoch: 89
Loss on hold-out set: 0.11415852745374044
MeanAbsoluteError value on hold-out data: 0.28223705291748047
Epoch: 90
Loss on hold-out set: 0.11248233375449976
MeanAbsoluteError value on hold-out data: 0.2916194796562195
Epoch: 91

Loss on hold-out set: 0.10250938509901365
MeanAbsoluteError value on hold-out data: 0.2738005518913269
Epoch: 92
Loss on hold-out set: 0.11293531566858292
MeanAbsoluteError value on hold-out data: 0.2874675393104553
Epoch: 93
Loss on hold-out set: 0.09894892214486996
MeanAbsoluteError value on hold-out data: 0.2734355032444
Epoch: 94
Loss on hold-out set: 0.10264713811377685
MeanAbsoluteError value on hold-out data: 0.27138325572013855
```

```
Epoch: 95

Loss on hold-out set: 0.10303326570739349
MeanAbsoluteError value on hold-out data: 0.2710043489933014
Epoch: 96
Loss on hold-out set: 0.09782499705751736
MeanAbsoluteError value on hold-out data: 0.261691153049469
Epoch: 97
Loss on hold-out set: 0.09337651278823614
MeanAbsoluteError value on hold-out data: 0.2551042437553406
Epoch: 98
Loss on hold-out set: 0.10152716026951869
MeanAbsoluteError value on hold-out data: 0.2646147608757019
Epoch: 99

Loss on hold-out set: 0.09080571204423904
MeanAbsoluteError value on hold-out data: 0.25116589665412903
Epoch: 100
Loss on hold-out set: 0.09740077987313271
MeanAbsoluteError value on hold-out data: 0.2611486315727234
Epoch: 101
Loss on hold-out set: 0.09009337484836578
MeanAbsoluteError value on hold-out data: 0.2535862624645233
Epoch: 102
Loss on hold-out set: 0.09459882696469625
MeanAbsoluteError value on hold-out data: 0.25809937715530396
Epoch: 103
Loss on hold-out set: 0.09371371277918418
MeanAbsoluteError value on hold-out data: 0.25427916646003723
Epoch: 104

Loss on hold-out set: 0.08671346571296454
MeanAbsoluteError value on hold-out data: 0.2433534562587738
Epoch: 105
Loss on hold-out set: 0.09063399215228855
MeanAbsoluteError value on hold-out data: 0.25487300753593445
Epoch: 106
Loss on hold-out set: 0.08952697611103455
MeanAbsoluteError value on hold-out data: 0.2516169548034668
Epoch: 107
Loss on hold-out set: 0.08393238477408886
MeanAbsoluteError value on hold-out data: 0.23893827199935913
```

```
Epoch: 108
Loss on hold-out set: 0.08618610047424834
MeanAbsoluteError value on hold-out data: 0.24530169367790222
Epoch: 109

Loss on hold-out set: 0.09010802884896596
MeanAbsoluteError value on hold-out data: 0.25245821475982666
Epoch: 110
Loss on hold-out set: 0.08167808663720887
MeanAbsoluteError value on hold-out data: 0.23557284474372864
Epoch: 111
Loss on hold-out set: 0.08617871544013421
MeanAbsoluteError value on hold-out data: 0.2448195070028305
Epoch: 112
Loss on hold-out set: 0.08135507445782424
MeanAbsoluteError value on hold-out data: 0.23707786202430725
Epoch: 113
Loss on hold-out set: 0.07569836135332783
MeanAbsoluteError value on hold-out data: 0.22417035698890686
Epoch: 114

Loss on hold-out set: 0.08103695775227
MeanAbsoluteError value on hold-out data: 0.236375629901886
Epoch: 115
Loss on hold-out set: 0.07959404510756334
MeanAbsoluteError value on hold-out data: 0.2308730036020279
Epoch: 116
Loss on hold-out set: 0.08116187448302904
MeanAbsoluteError value on hold-out data: 0.23997926712036133
Epoch: 117
Loss on hold-out set: 0.08102831594645978
MeanAbsoluteError value on hold-out data: 0.23373329639434814
Epoch: 118
Loss on hold-out set: 0.0843167885641257
MeanAbsoluteError value on hold-out data: 0.2444375902414322
Epoch: 119

Loss on hold-out set: 0.07717153321641188
MeanAbsoluteError value on hold-out data: 0.22724319994449615
Epoch: 120
Loss on hold-out set: 0.07604003101587295
MeanAbsoluteError value on hold-out data: 0.22651351988315582
```

```
Epoch: 121
Loss on hold-out set: 0.07229931344588597
MeanAbsoluteError value on hold-out data: 0.22354260087013245
Epoch: 122
Loss on hold-out set: 0.07608460087950031
MeanAbsoluteError value on hold-out data: 0.22852635383605957
Epoch: 123
Loss on hold-out set: 0.07819877876589695
MeanAbsoluteError value on hold-out data: 0.22654388844966888
Epoch: 124

Loss on hold-out set: 0.0706503570266068
MeanAbsoluteError value on hold-out data: 0.21976345777511597
Epoch: 125
Loss on hold-out set: 0.06669251896440982
MeanAbsoluteError value on hold-out data: 0.20637916028499603
Epoch: 126
Loss on hold-out set: 0.078494198344027
MeanAbsoluteError value on hold-out data: 0.22732622921466827
Epoch: 127
Loss on hold-out set: 0.06956530353364845
MeanAbsoluteError value on hold-out data: 0.21797680854797363
Epoch: 128
Loss on hold-out set: 0.07388384866838654
MeanAbsoluteError value on hold-out data: 0.2190914899110794
Returned to Spot: Validation loss: 0.07388384866838654
-----------------------------------------------

spotPython tuning: 0.03844679653334121 [##########] 98.82%


config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.7070829290488606, 'lr_mult':
Epoch: 1

Loss on hold-out set: 0.27073621918757756
MeanAbsoluteError value on hold-out data: 0.48282063007354736
Epoch: 2
Loss on hold-out set: 0.2607798490921656
MeanAbsoluteError value on hold-out data: 0.47567427158355713
Epoch: 3
Loss on hold-out set: 0.270518544614315
```

```
MeanAbsoluteError value on hold-out data: 0.47839635610580444
Epoch: 4

Loss on hold-out set: 0.26582812289396923
MeanAbsoluteError value on hold-out data: 0.4789690673351288
Epoch: 5
Loss on hold-out set: 0.25763459752003354
MeanAbsoluteError value on hold-out data: 0.46965357661247253
Epoch: 6

Loss on hold-out set: 0.2550575744112333
MeanAbsoluteError value on hold-out data: 0.46552327275276184
Epoch: 7
Loss on hold-out set: 0.25204906553030015
MeanAbsoluteError value on hold-out data: 0.4620039463043213
Epoch: 8
Loss on hold-out set: 0.24789320439100265
MeanAbsoluteError value on hold-out data: 0.4550989270210266
Epoch: 9

Loss on hold-out set: 0.23677833115061125
MeanAbsoluteError value on hold-out data: 0.44604671001434326
Epoch: 10
Loss on hold-out set: 0.2500999888777733
MeanAbsoluteError value on hold-out data: 0.45837950706481934
Epoch: 11

Loss on hold-out set: 0.23875862161318462
MeanAbsoluteError value on hold-out data: 0.44624581933021545
Epoch: 12
Loss on hold-out set: 0.2361627263824145
MeanAbsoluteError value on hold-out data: 0.44595327973365784
Epoch: 13
Loss on hold-out set: 0.24896054436763126
MeanAbsoluteError value on hold-out data: 0.45785048604011536
Epoch: 14

Loss on hold-out set: 0.21813895603020986
MeanAbsoluteError value on hold-out data: 0.43028804659843445
Epoch: 15
Loss on hold-out set: 0.22209316780169805
```

```
MeanAbsoluteError value on hold-out data: 0.43185725808143616
Epoch: 16

Loss on hold-out set: 0.23064883023500443
MeanAbsoluteError value on hold-out data: 0.43769654631614685
Epoch: 17
Loss on hold-out set: 0.2271346405148506
MeanAbsoluteError value on hold-out data: 0.4352818429470062
Epoch: 18
Loss on hold-out set: 0.21787761787573495
MeanAbsoluteError value on hold-out data: 0.42574888467788696
Epoch: 19

Loss on hold-out set: 0.21957904249429702
MeanAbsoluteError value on hold-out data: 0.4260520040988922
Epoch: 20
Loss on hold-out set: 0.22084309667348861
MeanAbsoluteError value on hold-out data: 0.43105027079582214
Epoch: 21

Loss on hold-out set: 0.21518585274616878
MeanAbsoluteError value on hold-out data: 0.42453476786613464
Epoch: 22
Loss on hold-out set: 0.20976350873708724
MeanAbsoluteError value on hold-out data: 0.41487473249435425
Epoch: 23
Loss on hold-out set: 0.21812545359134675
MeanAbsoluteError value on hold-out data: 0.42518070340156555
Epoch: 24

Loss on hold-out set: 0.20238308906555175
MeanAbsoluteError value on hold-out data: 0.4053115248680115
Epoch: 25
Loss on hold-out set: 0.20360988825559617
MeanAbsoluteError value on hold-out data: 0.40972334146499634
Epoch: 26

Loss on hold-out set: 0.19797303269306818
MeanAbsoluteError value on hold-out data: 0.4030969738960266
Epoch: 27
Loss on hold-out set: 0.20228065262238185
```

```
MeanAbsoluteError value on hold-out data: 0.4023883640766144
Epoch: 28
Loss on hold-out set: 0.19414071728785834
MeanAbsoluteError value on hold-out data: 0.39840903878211975
Epoch: 29

Loss on hold-out set: 0.19061160758137702
MeanAbsoluteError value on hold-out data: 0.3954952359199524
Epoch: 30
Loss on hold-out set: 0.18939676523208618
MeanAbsoluteError value on hold-out data: 0.39113104343414307
Epoch: 31

Loss on hold-out set: 0.18340360432863234
MeanAbsoluteError value on hold-out data: 0.38184618949890137
Epoch: 32
Loss on hold-out set: 0.18153285587827364
MeanAbsoluteError value on hold-out data: 0.3861355185508728
Epoch: 33
Loss on hold-out set: 0.19309844916065533
MeanAbsoluteError value on hold-out data: 0.3950936496257782
Epoch: 34

Loss on hold-out set: 0.18337199439605076
MeanAbsoluteError value on hold-out data: 0.37867242097854614
Epoch: 35
Loss on hold-out set: 0.1910575427611669
MeanAbsoluteError value on hold-out data: 0.38862112164497375
Epoch: 36

Loss on hold-out set: 0.1697915747265021
MeanAbsoluteError value on hold-out data: 0.36521193385124207
Epoch: 37
Loss on hold-out set: 0.1739890039463838
MeanAbsoluteError value on hold-out data: 0.3702552616596222
Epoch: 38
Loss on hold-out set: 0.17356987697382767
MeanAbsoluteError value on hold-out data: 0.3703102469444275

Epoch: 39
Loss on hold-out set: 0.16308098102609317
```

```
MeanAbsoluteError value on hold-out data: 0.35156622529029846
Epoch: 40

Loss on hold-out set: 0.1643588125705719
MeanAbsoluteError value on hold-out data: 0.36103910207748413
Epoch: 41
Loss on hold-out set: 0.17294575144847235
MeanAbsoluteError value on hold-out data: 0.36951273679733276
Epoch: 42

Loss on hold-out set: 0.16147819290558496
MeanAbsoluteError value on hold-out data: 0.35537511110305786
Epoch: 43
Loss on hold-out set: 0.16130249651769796
MeanAbsoluteError value on hold-out data: 0.3501095473766327
Epoch: 44

Loss on hold-out set: 0.16684266770879427
MeanAbsoluteError value on hold-out data: 0.35309264063835144
Epoch: 45
Loss on hold-out set: 0.16460441132386525
MeanAbsoluteError value on hold-out data: 0.35625597834587097
Epoch: 46
Loss on hold-out set: 0.16051479722062748
MeanAbsoluteError value on hold-out data: 0.3510599136352539
Epoch: 47

Loss on hold-out set: 0.1550102278838555
MeanAbsoluteError value on hold-out data: 0.3444261848926544
Epoch: 48
Loss on hold-out set: 0.14732319662968318
MeanAbsoluteError value on hold-out data: 0.3357709050178528
Epoch: 49

Loss on hold-out set: 0.1492652009924253
MeanAbsoluteError value on hold-out data: 0.3312600255012512
Epoch: 50
Loss on hold-out set: 0.14678569737821817
MeanAbsoluteError value on hold-out data: 0.3297043442726135
Epoch: 51
Loss on hold-out set: 0.13955925521751245
```

```
MeanAbsoluteError value on hold-out data: 0.31792372465133667
Epoch: 52

Loss on hold-out set: 0.15597787228723367
MeanAbsoluteError value on hold-out data: 0.3455964922904968
Epoch: 53
Loss on hold-out set: 0.14547496035695076
MeanAbsoluteError value on hold-out data: 0.33359190821647644
Epoch: 54

Loss on hold-out set: 0.13533826619386674
MeanAbsoluteError value on hold-out data: 0.31966501474380493
Epoch: 55
Loss on hold-out set: 0.13359090507030488
MeanAbsoluteError value on hold-out data: 0.31941837072372437
Epoch: 56

Loss on hold-out set: 0.12784856452296178
MeanAbsoluteError value on hold-out data: 0.3095194101333618
Epoch: 57
Loss on hold-out set: 0.12716343969106675
MeanAbsoluteError value on hold-out data: 0.3092189133167267
Epoch: 58

Loss on hold-out set: 0.13698666721582411
MeanAbsoluteError value on hold-out data: 0.32053470611572266
Epoch: 59
Loss on hold-out set: 0.12828104774157206
MeanAbsoluteError value on hold-out data: 0.30857598781585693
Epoch: 60

Loss on hold-out set: 0.1251206853489081
MeanAbsoluteError value on hold-out data: 0.30793872475624084
Epoch: 61
Loss on hold-out set: 0.1233909505357345
MeanAbsoluteError value on hold-out data: 0.29835745692253113
Epoch: 62
Loss on hold-out set: 0.12675812949736914
MeanAbsoluteError value on hold-out data: 0.3042847812175751
```

```
Epoch: 63
Loss on hold-out set: 0.135279194402198
MeanAbsoluteError value on hold-out data: 0.31760919094085693
Epoch: 64
Loss on hold-out set: 0.12625780038225154
MeanAbsoluteError value on hold-out data: 0.30322694778442383
Epoch: 65

Loss on hold-out set: 0.12068669830759367
MeanAbsoluteError value on hold-out data: 0.2991059124469757
Epoch: 66
Loss on hold-out set: 0.11589586921036243
MeanAbsoluteError value on hold-out data: 0.2887963652610779
Epoch: 67

Loss on hold-out set: 0.12032065232594807
MeanAbsoluteError value on hold-out data: 0.3000119924545288
Epoch: 68
Loss on hold-out set: 0.1045224440842867
MeanAbsoluteError value on hold-out data: 0.2725294232368469
Epoch: 69
Loss on hold-out set: 0.1177869468430678
MeanAbsoluteError value on hold-out data: 0.28780004382133484
Epoch: 70

Loss on hold-out set: 0.10868079314629236
MeanAbsoluteError value on hold-out data: 0.28398987650871277
Epoch: 71
Loss on hold-out set: 0.12014225450654825
MeanAbsoluteError value on hold-out data: 0.2936997413635254

Epoch: 72
Loss on hold-out set: 0.11266564269860585
MeanAbsoluteError value on hold-out data: 0.27863773703575134
Epoch: 73
Loss on hold-out set: 0.10589885175228118
MeanAbsoluteError value on hold-out data: 0.2753128707408905
Epoch: 74
Loss on hold-out set: 0.10425809668997923
MeanAbsoluteError value on hold-out data: 0.27451491355895996
Epoch: 75
```

```
Loss on hold-out set: 0.10129679583013057
MeanAbsoluteError value on hold-out data: 0.26481541991233826
Epoch: 76

Loss on hold-out set: 0.09406725297371546
MeanAbsoluteError value on hold-out data: 0.2587411105632782
Epoch: 77
Loss on hold-out set: 0.0920359951009353
MeanAbsoluteError value on hold-out data: 0.25298941135406494
Epoch: 78
Loss on hold-out set: 0.10534620396792889
MeanAbsoluteError value on hold-out data: 0.268971711397171
Epoch: 79
Loss on hold-out set: 0.09126916043460369
MeanAbsoluteError value on hold-out data: 0.25580790638923645
Epoch: 80

Loss on hold-out set: 0.0990159802014629
MeanAbsoluteError value on hold-out data: 0.26006874442100525
Epoch: 81

Loss on hold-out set: 0.09099946796894073
MeanAbsoluteError value on hold-out data: 0.2485739141702652
Epoch: 82
Loss on hold-out set: 0.09140227946142356
MeanAbsoluteError value on hold-out data: 0.2529680132865906
Epoch: 83
Loss on hold-out set: 0.0914291945969065
MeanAbsoluteError value on hold-out data: 0.2447642832994461
Epoch: 84
Loss on hold-out set: 0.08953079885492722
MeanAbsoluteError value on hold-out data: 0.24832715094089508

Epoch: 85

Loss on hold-out set: 0.09775808670868476
MeanAbsoluteError value on hold-out data: 0.26087716221809387
Epoch: 86
Loss on hold-out set: 0.09279012485096852
MeanAbsoluteError value on hold-out data: 0.25501659512519836
Epoch: 87
```

```
Loss on hold-out set: 0.09598833685119947
MeanAbsoluteError value on hold-out data: 0.2513434588909149
Epoch: 88
Loss on hold-out set: 0.08724160401771466
MeanAbsoluteError value on hold-out data: 0.24591723084449768
Epoch: 89

Loss on hold-out set: 0.08148571600516637
MeanAbsoluteError value on hold-out data: 0.2341916412115097
Epoch: 90

Loss on hold-out set: 0.08543281363944212
MeanAbsoluteError value on hold-out data: 0.24649803340435028
Epoch: 91
Loss on hold-out set: 0.09230637556562821
MeanAbsoluteError value on hold-out data: 0.2510582208633423
Epoch: 92
Loss on hold-out set: 0.08442105946441492
MeanAbsoluteError value on hold-out data: 0.23841117322444916
Epoch: 93
Loss on hold-out set: 0.08608772682026028
MeanAbsoluteError value on hold-out data: 0.23699018359184265
Epoch: 94

Loss on hold-out set: 0.07927829574793577
MeanAbsoluteError value on hold-out data: 0.22874987125396729
Epoch: 95

Loss on hold-out set: 0.08313771745190024
MeanAbsoluteError value on hold-out data: 0.23734474182128906
Epoch: 96
Loss on hold-out set: 0.08570974297821522
MeanAbsoluteError value on hold-out data: 0.2342057079076767
Epoch: 97
Loss on hold-out set: 0.07475231604029735
MeanAbsoluteError value on hold-out data: 0.22381934523582458
Epoch: 98
Loss on hold-out set: 0.07509713983163238
MeanAbsoluteError value on hold-out data: 0.2185281366109848
Epoch: 99
```

```
Loss on hold-out set: 0.07304948754608631
MeanAbsoluteError value on hold-out data: 0.22217343747615814
Epoch: 100

Loss on hold-out set: 0.08063007537275553
MeanAbsoluteError value on hold-out data: 0.23091496527194977
Epoch: 101
Loss on hold-out set: 0.07252568958948057
MeanAbsoluteError value on hold-out data: 0.2223917692899704
Epoch: 102
Loss on hold-out set: 0.07036214905480544
MeanAbsoluteError value on hold-out data: 0.21749891340732574
Epoch: 103
Loss on hold-out set: 0.07742892334858577
MeanAbsoluteError value on hold-out data: 0.22611884772777557

Epoch: 104
Loss on hold-out set: 0.07280709127585093

MeanAbsoluteError value on hold-out data: 0.22229944169521332
Epoch: 105
Loss on hold-out set: 0.07515895444899798
MeanAbsoluteError value on hold-out data: 0.2238183617591858
Epoch: 106
Loss on hold-out set: 0.06901379813129703
MeanAbsoluteError value on hold-out data: 0.21153022348880768
Epoch: 107
Loss on hold-out set: 0.07350987494302293
MeanAbsoluteError value on hold-out data: 0.22189968824386597
Epoch: 108

Loss on hold-out set: 0.06752455016598105
MeanAbsoluteError value on hold-out data: 0.2117907702922821
Epoch: 109

Loss on hold-out set: 0.07428961316744487
MeanAbsoluteError value on hold-out data: 0.22754524648189545
Epoch: 110
Loss on hold-out set: 0.07294555510083835
MeanAbsoluteError value on hold-out data: 0.21832670271396637
Epoch: 111
```

```
Loss on hold-out set: 0.07005944536688427
MeanAbsoluteError value on hold-out data: 0.2179068773984909
Epoch: 112
Loss on hold-out set: 0.06480843268334865
MeanAbsoluteError value on hold-out data: 0.20801930129528046

Epoch: 113
Loss on hold-out set: 0.06911984119564295
MeanAbsoluteError value on hold-out data: 0.2151695191860199
Epoch: 114

Loss on hold-out set: 0.07690279225508372
MeanAbsoluteError value on hold-out data: 0.21847178041934967
Epoch: 115
Loss on hold-out set: 0.06818282596766949
MeanAbsoluteError value on hold-out data: 0.20556940138339996
Epoch: 116
Loss on hold-out set: 0.06721098711093267
MeanAbsoluteError value on hold-out data: 0.21006496250629425
Epoch: 117

Loss on hold-out set: 0.06295676534374554
MeanAbsoluteError value on hold-out data: 0.20516666769981384
Epoch: 118
Loss on hold-out set: 0.06402240188481907
MeanAbsoluteError value on hold-out data: 0.20442819595336914
Epoch: 119

Loss on hold-out set: 0.06416687419948479
MeanAbsoluteError value on hold-out data: 0.20541247725486755
Epoch: 120
Loss on hold-out set: 0.0709463690655927
MeanAbsoluteError value on hold-out data: 0.2174854278564453
Epoch: 121
Loss on hold-out set: 0.06617773943270246
MeanAbsoluteError value on hold-out data: 0.20376434922218323
Epoch: 122

Loss on hold-out set: 0.05902654009560744
MeanAbsoluteError value on hold-out data: 0.19532251358032227
Epoch: 123
```

```
Loss on hold-out set: 0.06665845606476069
MeanAbsoluteError value on hold-out data: 0.20749269425868988
Epoch: 124

Loss on hold-out set: 0.06723904980967442
MeanAbsoluteError value on hold-out data: 0.210086852312088
Epoch: 125
Loss on hold-out set: 0.06626325105006496
MeanAbsoluteError value on hold-out data: 0.21229451894760132
Epoch: 126
Loss on hold-out set: 0.06416067981937279
MeanAbsoluteError value on hold-out data: 0.2047276645898819
Epoch: 127

Loss on hold-out set: 0.06119773506497343
MeanAbsoluteError value on hold-out data: 0.19840262830257416
Epoch: 128
Loss on hold-out set: 0.057446112440278134
MeanAbsoluteError value on hold-out data: 0.19243063032627106
Returned to Spot: Validation loss: 0.057446112440278134
-----------------------------------------------

spotPython tuning: 0.03844679653334121 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2c1cd7130>
```

During the run, the following output is shown:

```
config: {'_L_in': 10, '_L_out': 1, 'l1': 64, 'dropout_prob': 0.4475780541539,
    'lr_mult': 0.001, 'batch_size': 16, 'epochs': 512, 'k_folds': 1,
    'patience': 32,'optimizer': 'Adagrad', 'sgd_momentum': 0.9}
Epoch: 1
...
Epoch: 7002
Loss on hold-out set: 1.6959798782529844e-05
MeanAbsoluteError value on hold-out data: 0.0018855303060263395
Epoch: 7003
Loss on hold-out set: 1.6984027051769603e-05
MeanAbsoluteError value on hold-out data: 0.001883985591121018
Early stopping at epoch 7002
Returned to Spot: Validation loss: 1.6984027051769603e-05
```

20.10 Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard.

20.10.1 Tensorboard: Start Tensorboard

Start TensorBoard through the command line to visualize data you logged. Specify the root log directory as used in `fun_control = fun_control_init(task="regression", tensorboard_path="runs/24_spot_torch_regression")` as the `tensorboard_path`. The argument logdir points to directory where TensorBoard will look to find event files that it can display. TensorBoard will recursively walk the directory structure rooted at logdir, looking for *.tfevents.* files.

tensorboard –logdir=runs

Go to the URL it provides OR to http://localhost:6006/.

The following figures show some screenshots of Tensorboard.

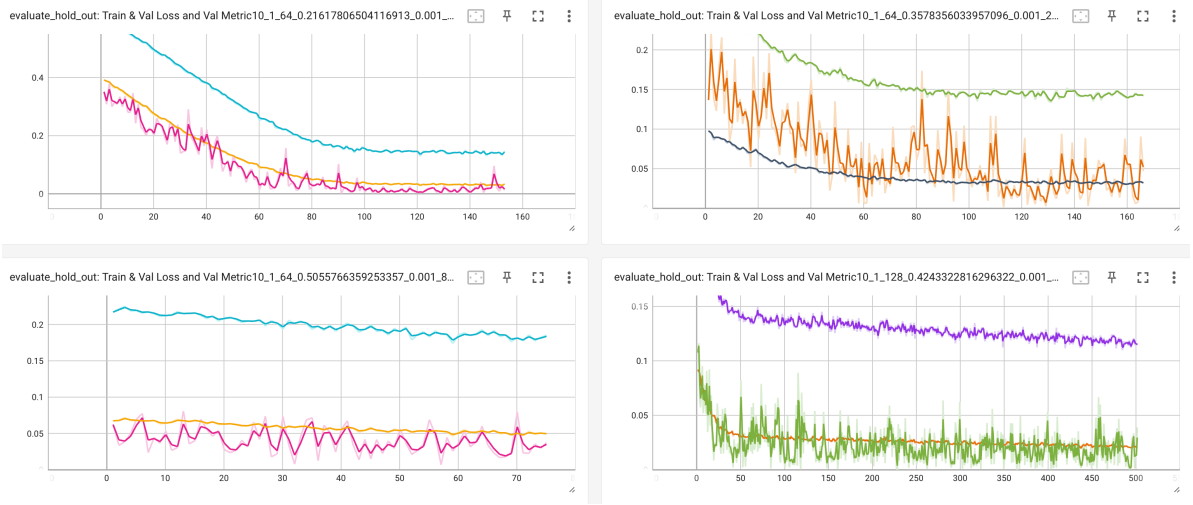


Figure 20.1: Tensorboard

20.11 Results

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from Figure 20.4.

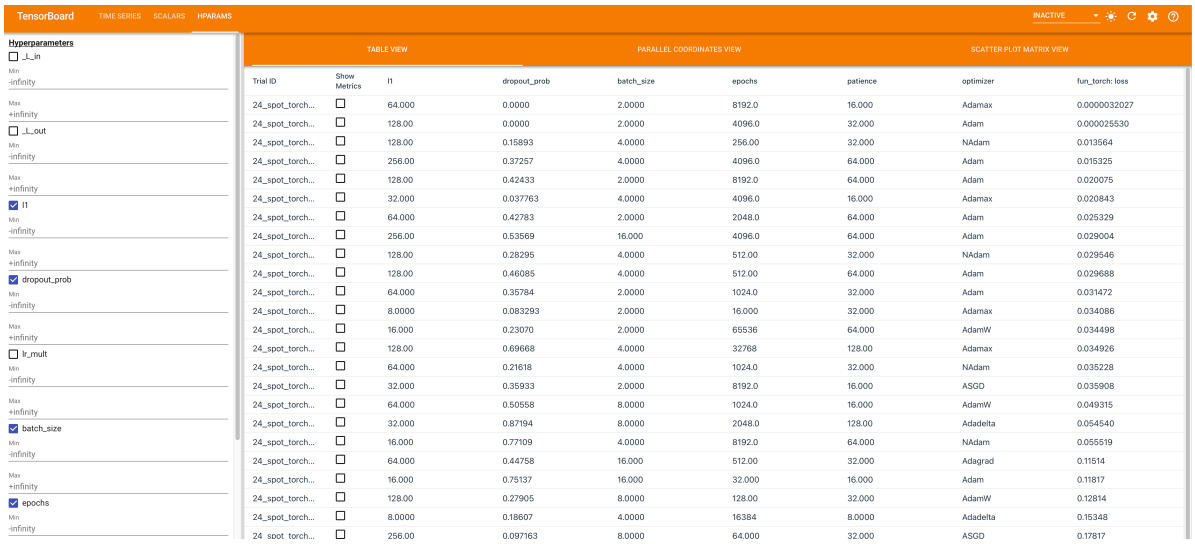


Figure 20.2: Tensorboard

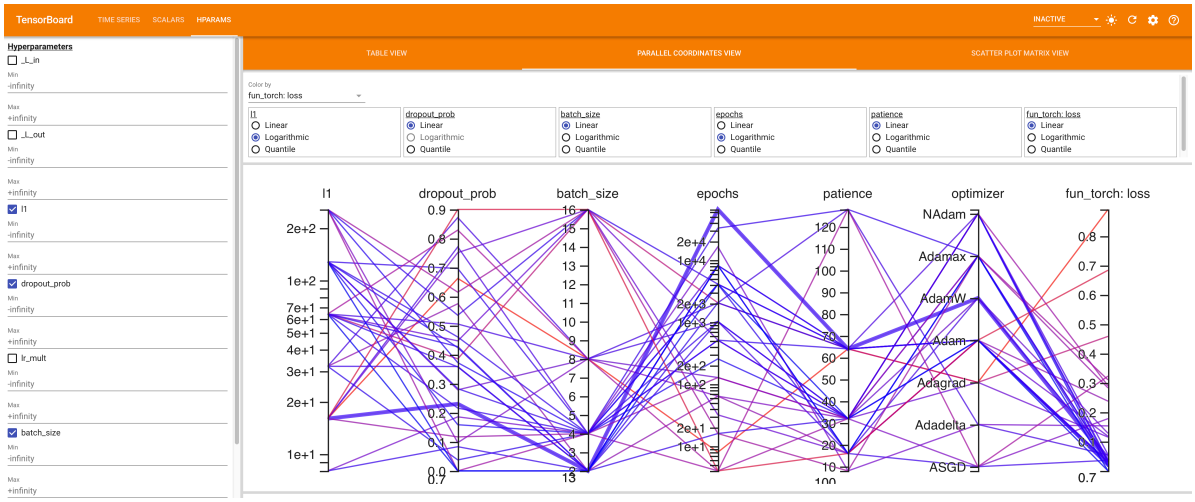


Figure 20.3: Tensorboard

```
spot_tuner.plot_progress(log_y=False, filename="./figures/" + experiment_name+"_progress.p
```

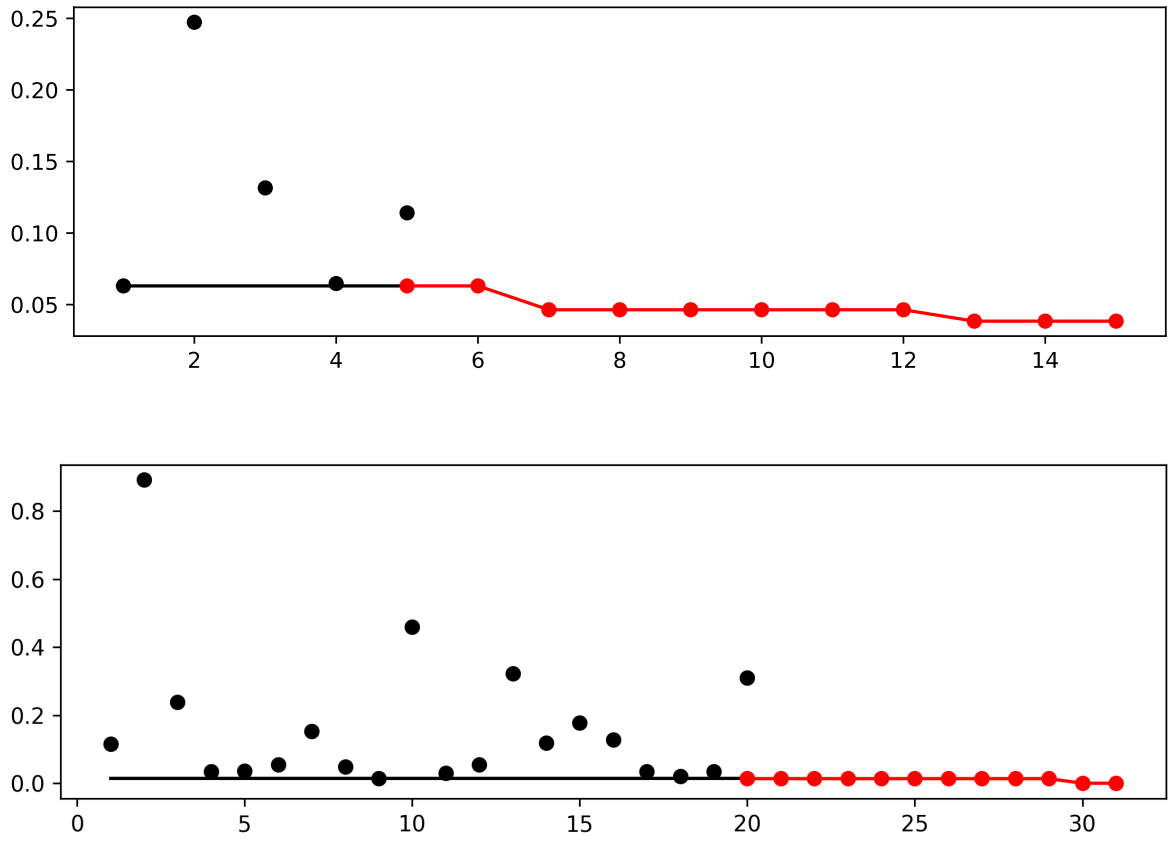


Figure 20.4: Progress plot. `Black` dots denote results from the initial design. `Red` dots illustrate the improvement found by the surrogate model based optimization (surrogate model based optimization).

Figure 20.4 shows a typical behaviour that can be observed in many hyperparameter studies (Bartz et al. 2022): the largest improvement is obtained during the evaluation of the initial design. The surrogate model based optimization-optimization with the surrogate refines the results. Figure 20.4 also illustrates one major difference between `ray[tune]` as used in PyTorch (2023a) and `spotPython`: the `ray[tune]` uses a random search and will generate results similar to the *black* dots, whereas `spotPython` uses a surrogate model based optimization and presents results represented by *red* dots in Figure 20.4. The surrogate model based optimization is considered to be more efficient than a random search, because the surrogate model guides the search towards promising regions in the hyperparameter space.

In addition to the improved ("optimized") hyperparameter values, `spotPython` allows a statistical analysis, e.g., a sensitivity analysis, of the results. We can print the results of the

hyperparameter tuning, see Table 20.4.

```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

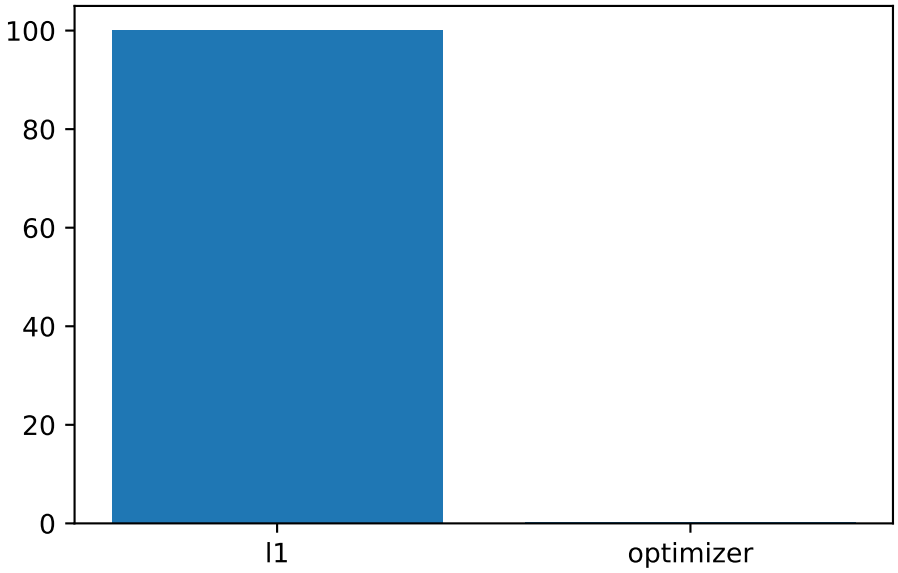
```
| name          | type   | default   |   lower |   upper |               tuned | transform
|---------------|--------|-----------|---------|---------|---------------------|--------------
| _L_in         | int    | 10        |    10.0 |    10.0 |                10.0 | None
| _L_out        | int    | 1         |     1.0 |     1.0 |                 1.0 | None
| l1            | int    | 3         |     3.0 |     8.0 |                 7.0 | transform_powe
| dropout_prob | float  | 0.01      |     0.0 |     0.9 | 0.7092110561836277 | None
| lr_mult       | float  | 1.0       |   0.001 |   0.001 |               0.001 | None
| batch_size    | int    | 4         |     1.0 |     4.0 |                 2.0 | transform_powe
| epochs        | int    | 4         |     2.0 |    16.0 |                 7.0 | transform_powe
| k_folds       | int    | 1         |     1.0 |     1.0 |                 1.0 | None
| patience      | int    | 2         |     3.0 |     7.0 |                 7.0 | transform_powe
| optimizer     | factor | SGD       |     0.0 |     6.0 |                 3.0 | None
| sgd_momentum  | float  | 0.0       |     0.9 |     0.9 |                 0.9 | None
```

Table 20.4: Results of the hyperparameter tuning. The table shows the hyperparameters, their types, default values, lower and upper bounds, and the transformation function. The column "tuned" shows the tuned values. The column "importance" shows the importance of the hyperparameters. The column "stars" shows the importance of the hyperparameters in stars. The importance is computed by the SPOT software.

name	type	default	lower	upper	tuned	transform	importance	stars
_L_in	int	10	10.0	10.0	10.0	None	0.00	
_L_out	int	1	1.0	1.0	1.0	None	0.00	
l1	int	3	3.0	8.0	6.0	power_2_int	1.42	*
drop_p	float	0.01	0.0	0.9	0.0	None	0.00	
lr_mult	float	1.0	0.001	0.001	0.001	None	0.00	
batch_s	int	4	1.0	4.0	1.0	power_2_int	0.01	
epochs	int	4	2.0	16.0	13.0	power_2_int	100.00	***
k_folds	int	1	1.0	1.0	1.0	None	0.00	
patience	int	2	3.0	7.0	4.0	power_2_int	0.00	
optim	factor	SGD	0.0	6.0	4.0	None	0.00	
sgd_mom	float	0.0	0.9	0.9	0.9	None	0.00	

To visualize the most important hyperparameters, `spotPython` provides the function `plot_importance`. The following code generates the importance plot from Figure 20.5.

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```



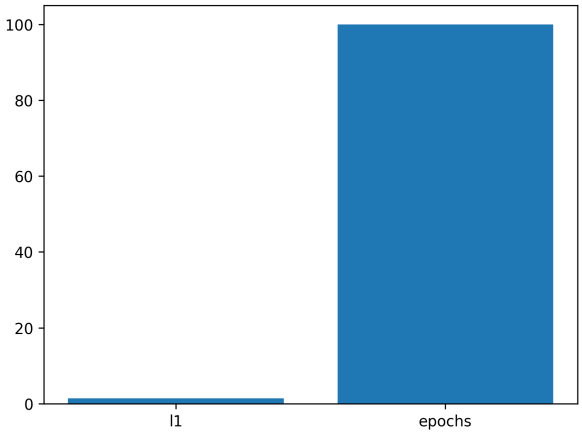


Figure 20.5: Variable importance

20.12 Get the Tuned Architecture

The architecture of the `spotPython` model can be obtained by the following code:

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_lin_reg(
  (fc1): Linear(in_features=10, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=1, bias=True)
  (relu): ReLU()
  (softmax): Softmax(dim=1)
  (dropout1): Dropout(p=0.7092110561836277, inplace=False)
  (dropout2): Dropout(p=0.35460552809181384, inplace=False)
)
```

First, the numerical representation of the hyperparameters are obtained, i.e., the numpy array `X` is generated. This array is then used to generate the model `model_spot` by the function `get_one_core_model_from_X`. The model `model_spot` has the following architecture:

```
Net_lin_reg(
  (fc1): Linear(in_features=10, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=1, bias=True)
  (relu): ReLU()
  (softmax): Softmax(dim=1)
  (dropout1): Dropout(p=0.0, inplace=False)
  (dropout2): Dropout(p=0.0, inplace=False)
)
```

20.13 Evaluation of the Tuned Architecture

The method `train_tuned` takes a model architecture without trained weights and trains this model with the train data. The train data is split into train and validation data. The validation data is used for early stopping. The trained model weights are saved as a dictionary.

The following code trains the model `model_spot`. If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be saved to this file.

```
from spotPython.torch.traintest import (
    train_tuned,
```

```
    test_tuned,
    )
train_tuned(net=model_spot, train_dataset=train,
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        shuffle=True,
        device = DEVICE,
        path=None,
        task=fun_control["task"],)
```

```
Epoch: 1

Error in Net_Core. Call to evaluate_hold_out() failed. err=RuntimeError('Encountered differe
Returned to Spot: Validation loss: nan
---------------------------------------------
```

```
Epoch: 1
Loss on hold-out set: 0.17853929138431945
MeanAbsoluteError value on hold-out data: 0.3907899856567383
Epoch: 2
Loss on hold-out set: 0.17439044278115035
MeanAbsoluteError value on hold-out data: 0.38570401072502136
```

If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be loaded from this file.

```
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = DEVICE,
            task=fun_control["task"],)
```

```
Error in Net_Core. Call to test_tuned() failed. err=RuntimeError('Encountered different devi
Final evaluation: Validation loss: nan
Final evaluation: Validation metric: nan
---------------------------------------------

(nan, nan, nan)
```

```
Loss on hold-out set: 1.85966069472272e-05
MeanAbsoluteError value on hold-out data: 0.0021022311411798
Final evaluation: Validation loss: 1.85966069472272e-05
Final evaluation: Validation metric: 0.0021022311411798
------------------------------------------------
(1.85966069472272e-05, nan, tensor(0.0021))
```

20.14 Cross-validated Evaluations

```
from spotPython.torch.traintest import evaluate_cv
# modify k-kolds:
setattr(model_spot, "k_folds",  10)
evaluate_cv(net=model_spot,
            dataset=fun_control["data"],
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            task=fun_control["task"],
            writer=fun_control["writer"],
            writerId="model_spot_cv", device=DEVICE)
```

```
Fold: 1
Epoch: 1

Error in Net_Core. Call to evaluate_cv() failed. err=RuntimeError('Encountered different devi

(nan, nan, nan)
```

```
Fold: 1
Epoch: 1
Loss on hold-out set: 0.36993918985128404
MeanAbsoluteError value on hold-out data: 0.5827060341835022
Epoch: 2
Loss on hold-out set: 0.3583159705996513

(0.0027241395250238156, nan, tensor(0.0147))
```

Table 20.5 shows the loss and meric value (MAE) of the model with the tuned hyperparameters from SPOT.

Table 20.5: Comparison of the loss and metric values.

Model	Loss	Metric (MAE)
`Validation`	1.8597e-05	0.0021
`10-fold CV`	0.00272	0.0147

20.15 Detailed Hyperparameter Plots

The contour plot in this section visualize the interactions of the two most important hyperparameters, `l1`, and `epochs` of the surrogate model used to optimize the hyperparameters. Since some of these hyperparameters take fatorial or integer values, sometimes step-like fitness landcapes (or response surfaces) are generated. SPOT draws the interactions of the main hyperparameters by default. It is also possible to visualize all interactions. For this, again refer to the notebook (Bartz-Beielstein 2023).

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
l1:  100.00000000000001
optimizer:  0.19192749069412157
```

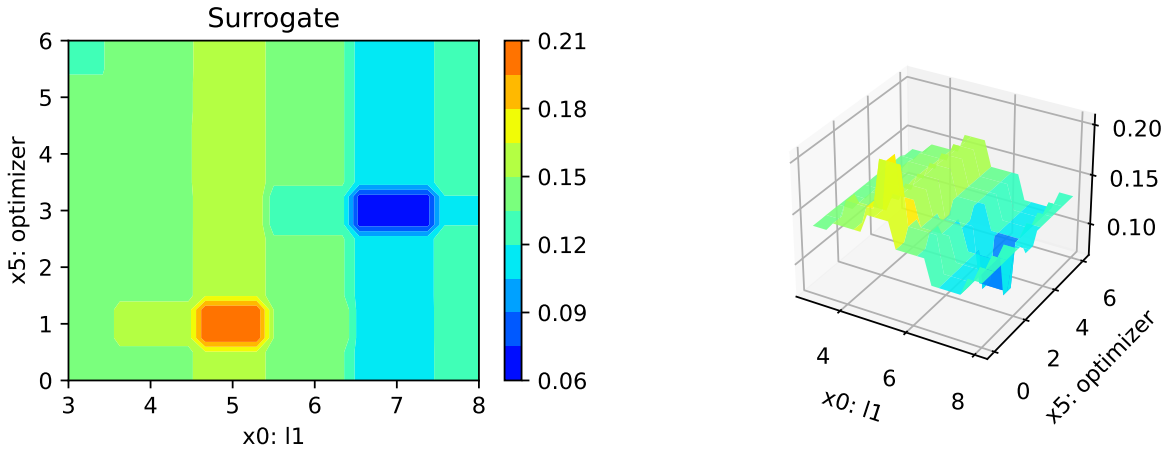


Figure 20.6 shows a contour plot of the loss as a function of the hyperparameters. These plots are very helpful for benchmark studies and for understanding neural networks. `spotPython` provides additional tools for a visual inspection of the results and give valuable insights into the

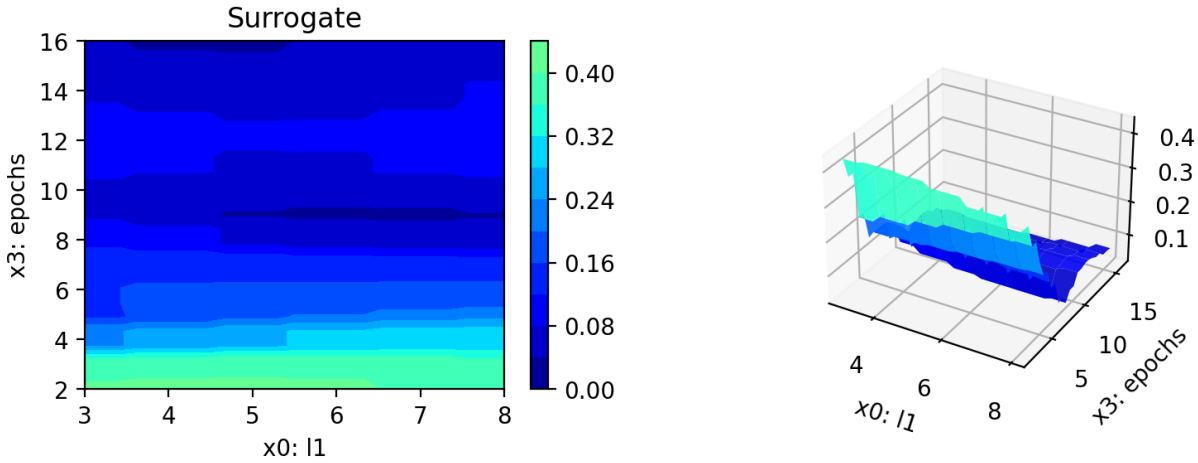


Figure 20.6: Contour plot of the loss as a function of `epochs` and `l1`, i.e., the number of neurons in the layers.

hyperparameter tuning process. This is especially useful for model explainability, transparency, and trustworthiness. In addition to the contour plots, Figure 20.7 shows the parallel plot of the hyperparameters.

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

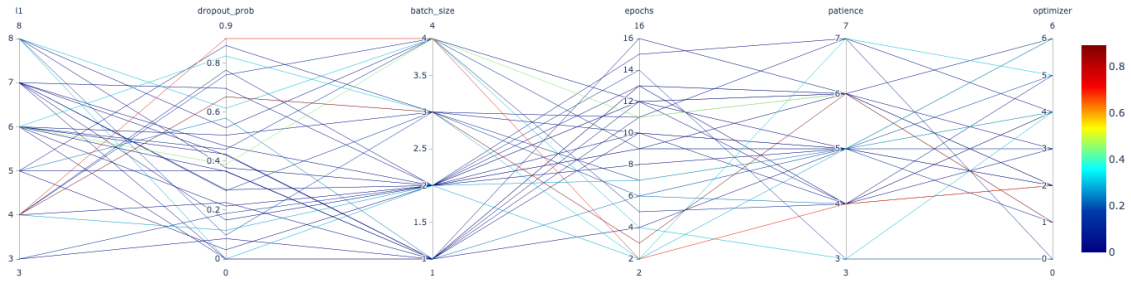


Figure 20.7: Parallel plot

20.16 Summary and Outlook

This tutorial presents the hyperparameter tuning open source software `spotPython` for `PyTorch`. Some of the advantages of `spotPython` are:

- Numerical and categorical hyperparameters.
- Powerful surrogate models.
- Flexible approach and easy to use.
- Simple JSON files for the specification of the hyperparameters.
- Extension of default and user specified network classes.
- Noise handling techniques.
- Online visualization of the hyperparameter tuning process with `tensorboard`.

Currently, only rudimentary parallel and distributed neural network training is possible, but these capabilities will be extended in the future. The next version of `spotPython` will also include a more detailed documentation and more examples.

> **!** Important
>
> Important: This tutorial does not present a complete benchmarking study (Bartz-Beielstein et al. 2020). The results are only preliminary and highly dependent on the local configuration (hard- and software). Our goal is to provide a first impression of the performance of the hyperparameter tuning package `spotPython`. The results should be interpreted with care.

21 Hyperparameter Tuning: VBDP

In this tutorial, we will show how `spotPython` can be integrated into the `PyTorch` training workflow.

This document refers to the following software versions:

- `python`: 3.10.10
- `torch`: 2.0.1
- `torchvision`: 0.15.0
- `spotPython`: 0.2.29

`spotPython` can be installed via pip. Alternatively, the source code can be downloaded from gitHub: https://github.com/sequential-parameter-optimization/spotPython.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of `spotPython` from gitHub.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

21.1 Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = None # "cpu" # "cuda:0"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
```

```
print(DEVICE)
```

```
mps
```

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '25-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SI
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

```
25-torch_bartz09_1min_5init_2023-06-15_06-05-52
```

21.2 Initialization of the `fun_control` Dictionary

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process. This dictionary is called `fun_control` and is initialized with the function `fun_control_init`. The function `fun_control_init` returns a skeleton dictionary. The dictionary is filled with the required information for the hyperparameter tuning process. It stores the hyperparameter tuning settings, e.g., the deep learning network architecture that should be tuned, the classification (or regression) problem, and the data that is used for the tuning. The dictionary is used as an input for the SPOT function.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/25_spot_torch_vbdp",
    device=DEVICE)
```

```
import torch
print(torch.__version__)
# Check that MPS is available
if not torch.backends.mps.is_available():
```

```
    if not torch.backends.mps.is_built():
        print("MPS not available because the current PyTorch install was not "
              "built with MPS enabled.")
    else:
        print("MPS not available because the current MacOS version is not 12.3+ "
              "and/or you do not have an MPS-enabled device on this machine.")

else:
    mps_device = torch.device("mps")
    print("MPS device: ", mps_device)
```

```
2.0.1
MPS device:  mps
```

22 PyTorch Data Loading

22.1 1. Load VBDP Data

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
train_df = pd.read_csv('./data/VBDP/train.csv')
# remove the id column
train_df = train_df.drop(columns=['id'])
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# # Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.head()

# convert all entries to int for faster processing
train_df = train_df.astype(int)
```

```
from spotPython.data.vbdp import combine_features
df_new = train_df.copy()
# save the target column using "target_column" as the column name
target = train_df[target_column]
# remove the target column
df_new = df_new.drop(columns=[target_column])
train_df = combine_features(df_new)
# add the target column back
train_df[target_column] = target
train_df.head()
```

	sudden_fever	headache	mouth_bleed	nose_bleed	muscle_pain	joint_pain	vomiting	rash	dia
0	1	1	0	1	1	1	1	0	1
1	0	0	0	0	0	0	1	0	1

	sudden_fever	headache	mouth_bleed	nose_bleed	muscle_pain	joint_pain	vomiting	rash	dia
2	0	1	1	1	0	1	1	1	1
3	0	0	1	1	1	1	0	1	0
4	0	0	0	0	0	0	0	0	1

- feature engineering: 6112 features

```
from sklearn.model_selection import train_test_split
import numpy as np

n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column, axis=1),
                                                    random_state=42,
                                                    test_size=0.25,
                                                    stratify=train_df[target_column])
trainset = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
testset = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
trainset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
testset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
print(trainset.shape)
print(testset.shape)
```

```
(707, 6113)
(530, 6113)
(177, 6113)
```

```
from sklearn.model_selection import train_test_split
from spotPython.torch.dataframedataset import DataFrameDataset
dtype_x = torch.float32
dtype_y = torch.long
train_df = DataFrameDataset(train_df, target_column=target_column, dtype_x=dtype_x, dtype_
train = DataFrameDataset(trainset, target_column=target_column, dtype_x=dtype_x, dtype_y=d
test = DataFrameDataset(testset, target_column=target_column, dtype_x=dtype_x, dtype_y=dty
n_samples = len(train)
```

```
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
                "train": train,
                "test": test,
                "n_samples": n_samples,
                "target_column": target_column})
```

22.2 Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables. The preprocessing model is called `prep_model` ("preparation" or pre-processing) and includes steps that are not subject to the hyperparameter tuning process. The preprocessing model is specified in the `fun_control` dictionary. The preprocessing model can be implemented as a `sklearn` pipeline. The following code shows a typical preprocessing pipeline:

```
# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
#         transformers=[
#             ("categorical", one_hot_encoder, categorical_columns),
#         ],
#         remainder=StandardScaler(),
#     )
prep_model = None
fun_control.update({"prep_model": prep_model})
```

22.3 Select `algorithm` and `core_model_hyper_dict`

22.3.1 Implementing a Configurable Neural Network With spotPython

`spotPython` includes the `Net_vbdp` class which is implemented in the file `netvbdp.py`. The class is imported here. `Net_vbdp` inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see **?@sec-the-net-core-class-24**.

23 add the nn model to the fun_control dictionary

```
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
from spotPython.torch.netvbdp import Net_vbdp
fun_control = add_core_model_to_fun_control(core_model=Net_vbdp,
                                fun_control=fun_control,
                                hyper_dict=TorchHyperDict)
```

23.1 Modifying the Hyperparameters

Small number of epochs for demonstration purposes

- `epochs` is set to 2 and 3 for demonstration purposes. These values are too small for a real application.

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds

fun_control = modify_hyper_parameter_bounds(fun_control, "_L0", bounds=[n_features, n_feat
fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[6, 13])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 2])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 6])
fun_control = modify_hyper_parameter_bounds(fun_control, "lr_mult", bounds=[1e-3, 1e-3])
fun_control = modify_hyper_parameter_bounds(fun_control, "sgd_momentum", bounds=[0.9, 0.9]
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer",["Adam", "AdamW", "Ad
# fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam"])
# fun_control = modify_hyper_parameter_levels(fun_control, "leaf_model", ["LinearRegressio
# fun_control["core_model_hyper_dict"]
```

```
fun_control = modify_hyper_parameter_bounds(fun_control,
    "lr_mult", bounds=[1e-3, 1e-3])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "sgd_momentum", bounds=[0.9, 0.9])
```

23.2 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set and
2. the loss function (and a metric).

These are described in Section 20.8.

The loss function is specified by the key `"loss_function"`. We will use CrossEntropy loss for the multiclass-classification task.

```
from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({"loss_function": loss_function})
```

23.2.1 Metric

```
from spotPython.torch.mapk import MAPK
import torch
mapk = MAPK(k=2)
target = torch.tensor([0, 1, 2, 2])
preds = torch.tensor(
    [
        [0.5, 0.2, 0.2],  # 0 is in top 2
        [0.3, 0.4, 0.2],  # 1 is in top 2
        [0.2, 0.4, 0.3],  # 2 is in top 2
        [0.7, 0.2, 0.1],  # 2 isn't in top 2
    ]
)
mapk.update(preds, target)
print(mapk.compute()) # tensor(0.6250)
```

```
tensor(0.6250)
```

```
from spotPython.torch.mapk import MAPK
import torchmetrics
metric_torch = MAPK(k=3)
fun_control.update({"metric_torch": metric_torch})
```

23.3 Calling the SPOT Function

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` generates a design table as follows:

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

```
| name         | type   | default   |    lower |    upper | transform             |
|--------------|--------|-----------|----------|----------|-----------------------|
| _L0          | int    | 64        | 6112     | 6112     | None                  |
| l1           | int    | 8         |    6     |   13     | transform_power_2_int |
| dropout_prob | float  | 0.01      |    0     |    0.9   | None                  |
| lr_mult      | float  | 1.0       |    0.001 |    0.001 | None                  |
| batch_size   | int    | 4         |    1     |    4     | transform_power_2_int |
| epochs       | int    | 4         |    2     |    2     | transform_power_2_int |
| k_folds      | int    | 1         |    1     |    1     | None                  |
| patience     | int    | 2         |    2     |    6     | transform_power_2_int |
| optimizer    | factor | SGD       |    0     |    3     | None                  |
| sgd_momentum | float  | 0.0       |    0.9   |    0.9   | None                  |
```

This allows to check if all information is available and if the information is correct.

The objective function `fun_torch` is selected next. It implements an interface from `PyTorch`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch
```

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=TorchHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
```

```
import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = inf,
                    fun_repeats = 1,
                    max_time = MAX_TIME,
                    noise = False,
                    tolerance_x = np.sqrt(np.spacing(1)),
                    var_type = var_type,
                    var_name = var_name,
                    infill_criterion = "y",
                    n_points = 1,
                    seed=123,
                    log_level = 50,
                    show_models= False,
                    show_progress= True,
                    fun_control = fun_control,
                    design_control={"init_size": INIT_SIZE,
                                    "repeats": 1},
                    surrogate_control={"noise": True,
                                       "cod_type": "norm",
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": len(var_name),
                                       "model_fun_evals": 10_000,
                                       "log_level": 50
                                       })
spot_tuner.run(X_start=X_start)
```

```
config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.17031221661559992, 'lr_mult': 0.001, 'ba
Epoch: 1

Loss on hold-out set: 2.3979967832565308
Accuracy on hold-out set: 0.12264150943396226
MAPK value on hold-out data: 0.174107164144516
Epoch: 2

Loss on hold-out set: 2.3978747640337263
Accuracy on hold-out set: 0.10849056603773585
MAPK value on hold-out data: 0.1703868955373764
Epoch: 3

Loss on hold-out set: 2.3978579895836964
Accuracy on hold-out set: 0.11320754716981132
MAPK value on hold-out data: 0.1696428507566452
Epoch: 4

Loss on hold-out set: 2.3978683607918874
Accuracy on hold-out set: 0.12264150943396226
MAPK value on hold-out data: 0.1778273731470108
Returned to Spot: Validation loss: 2.3978683607918874
-----------------------------------------------

config: {'_L0': 6112, 'l1': 256, 'dropout_prob': 0.19379790035512987, 'lr_mult': 0.001, 'bat
Epoch: 1

Loss on hold-out set: 2.3977124955919056
Accuracy on hold-out set: 0.09433962264150944
MAPK value on hold-out data: 0.18132717907428741
Epoch: 2

Loss on hold-out set: 2.3977297941843667
Accuracy on hold-out set: 0.09433962264150944
MAPK value on hold-out data: 0.18209877610206604
Epoch: 3

Loss on hold-out set: 2.3977693804988154
Accuracy on hold-out set: 0.09433962264150944
MAPK value on hold-out data: 0.16512344777584076
Epoch: 4
```

```
Loss on hold-out set: 2.397717140339039
Accuracy on hold-out set: 0.09433962264150944
MAPK value on hold-out data: 0.17978394031524658
Returned to Spot: Validation loss: 2.397717140339039
--------------------------------------------

config: {'_L0': 6112, 'l1': 4096, 'dropout_prob': 0.6759063718076167, 'lr_mult': 0.001, 'bat
Epoch: 1

Loss on hold-out set: 2.3978329672003693
Accuracy on hold-out set: 0.10377358490566038
MAPK value on hold-out data: 0.17610064148902893
Epoch: 2

Loss on hold-out set: 2.3975654930438637
Accuracy on hold-out set: 0.09433962264150944
MAPK value on hold-out data: 0.18946537375450134
Epoch: 3

Loss on hold-out set: 2.3975395234125965
Accuracy on hold-out set: 0.10377358490566038
MAPK value on hold-out data: 0.20676098763942719
Epoch: 4

Loss on hold-out set: 2.3971761285134083
Accuracy on hold-out set: 0.1179245283018868
MAPK value on hold-out data: 0.21933957934379578
Returned to Spot: Validation loss: 2.3971761285134083
--------------------------------------------

config: {'_L0': 6112, 'l1': 128, 'dropout_prob': 0.37306669346546995, 'lr_mult': 0.001, 'bat
Epoch: 1

Loss on hold-out set: 2.39852715438267
Accuracy on hold-out set: 0.06132075471698113
MAPK value on hold-out data: 0.14465411007404327
Epoch: 2

Loss on hold-out set: 2.3985299074424886
Accuracy on hold-out set: 0.06132075471698113
MAPK value on hold-out data: 0.14465411007404327
Epoch: 3
```

```
Loss on hold-out set: 2.3985011397667653
Accuracy on hold-out set: 0.06132075471698113
MAPK value on hold-out data: 0.14465411007404327
Epoch: 4

Loss on hold-out set: 2.3984982157653234
Accuracy on hold-out set: 0.06132075471698113
MAPK value on hold-out data: 0.1462264209985733
Returned to Spot: Validation loss: 2.3984982157653234
--------------------------------------------

config: {'_L0': 6112, 'l1': 1024, 'dropout_prob': 0.870137281216666, 'lr_mult': 0.001, 'batch
Epoch: 1

Loss on hold-out set: 2.398311314759431
Accuracy on hold-out set: 0.06132075471698113
MAPK value on hold-out data: 0.15123456716537476
Epoch: 2

Loss on hold-out set: 2.3981534286781594
Accuracy on hold-out set: 0.0660377358490566
MAPK value on hold-out data: 0.14660494029521942
Epoch: 3

Loss on hold-out set: 2.3982028696272106
Accuracy on hold-out set: 0.08490566037735849
MAPK value on hold-out data: 0.17206791043281555
Epoch: 4

Loss on hold-out set: 2.397975241696393
Accuracy on hold-out set: 0.06132075471698113
MAPK value on hold-out data: 0.15046298503875732
Returned to Spot: Validation loss: 2.397975241696393
--------------------------------------------

config: {'_L0': 6112, 'l1': 512, 'dropout_prob': 0.6758807490545228, 'lr_mult': 0.001, 'batch
Epoch: 1

Loss on hold-out set: 2.397592297306767
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.17283953726291656
Epoch: 2
```

```
Loss on hold-out set: 2.3976424005296497
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.17129629850387573
Epoch: 3

Loss on hold-out set: 2.397704327547992
Accuracy on hold-out set: 0.08490566037735849
MAPK value on hold-out data: 0.14429013431072235
Epoch: 4

Loss on hold-out set: 2.3976081477271185
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.16203702986240387
Returned to Spot: Validation loss: 2.3976081477271185
--------------------------------------------
spotPython tuning: 2.3971761285134083 [----------] 3.23%


config: {'_L0': 6112, 'l1': 128, 'dropout_prob': 0.6758405299575269, 'lr_mult': 0.001, 'batch
Epoch: 1

Loss on hold-out set: 2.3988353801223465
Accuracy on hold-out set: 0.08490566037735849
MAPK value on hold-out data: 0.15566039085388184
Epoch: 2

Loss on hold-out set: 2.39876715192255
Accuracy on hold-out set: 0.07547169811320754
MAPK value on hold-out data: 0.1533019095659256
Epoch: 3

Loss on hold-out set: 2.398924026849135
Accuracy on hold-out set: 0.07075471698113207
MAPK value on hold-out data: 0.14779873192310333
Epoch: 4

Loss on hold-out set: 2.3989389887395896
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.15880504250526428
Returned to Spot: Validation loss: 2.3989389887395896
--------------------------------------------
spotPython tuning: 2.3971761285134083 [#---------] 8.87%
```

```
config: {'_L0': 6112, 'l1': 4096, 'dropout_prob': 0.4132005099912892, 'lr_mult': 0.001, 'bat
Epoch: 1

Loss on hold-out set: 2.3978574455909007
Accuracy on hold-out set: 0.10377358490566038
MAPK value on hold-out data: 0.19575469195842743
Epoch: 2

Loss on hold-out set: 2.3975029104160814
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.19182389974594116
Epoch: 3

Loss on hold-out set: 2.397014966550863
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.18238994479179382
Epoch: 4

Loss on hold-out set: 2.3966288926466457
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.18553458154201508
Returned to Spot: Validation loss: 2.3966288926466457
-----------------------------------------------
spotPython tuning: 2.3966288926466457 [###-------] 26.92%


config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.19070372685884734, 'lr_mult': 0.001, 'ba
Epoch: 1

Loss on hold-out set: 2.3974582901540793
Accuracy on hold-out set: 0.10849056603773585
MAPK value on hold-out data: 0.18317610025405884
Epoch: 2

Loss on hold-out set: 2.39713295000904
Accuracy on hold-out set: 0.1320754716981132
MAPK value on hold-out data: 0.20440252125263214
Epoch: 3
```

```
Loss on hold-out set: 2.3967361720103137
Accuracy on hold-out set: 0.1320754716981132
MAPK value on hold-out data: 0.21226415038108826
Epoch: 4

Loss on hold-out set: 2.3962184168257803
Accuracy on hold-out set: 0.1320754716981132
MAPK value on hold-out data: 0.23191823065280914
Returned to Spot: Validation loss: 2.3962184168257803
-----------------------------------------------
spotPython tuning: 2.3962184168257803 [####------] 38.89%


config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.0, 'lr_mult': 0.001, 'batch_size': 2, 'ep
Epoch: 1

Loss on hold-out set: 2.3977733827986807
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.16273586452007294
Epoch: 2

Loss on hold-out set: 2.397623988817323
Accuracy on hold-out set: 0.1179245283018868
MAPK value on hold-out data: 0.17688679695129395
Epoch: 3

Loss on hold-out set: 2.3974878293163373
Accuracy on hold-out set: 0.12735849056603774
MAPK value on hold-out data: 0.18238994479179382
Epoch: 4

Loss on hold-out set: 2.3973523760741613
Accuracy on hold-out set: 0.12735849056603774
MAPK value on hold-out data: 0.18946541845798492
Returned to Spot: Validation loss: 2.3973523760741613
-----------------------------------------------
spotPython tuning: 2.3962184168257803 [#####-----] 50.10%


config: {'_L0': 6112, 'l1': 64, 'dropout_prob': 0.0, 'lr_mult': 0.001, 'batch_size': 2, 'epo
Epoch: 1
```

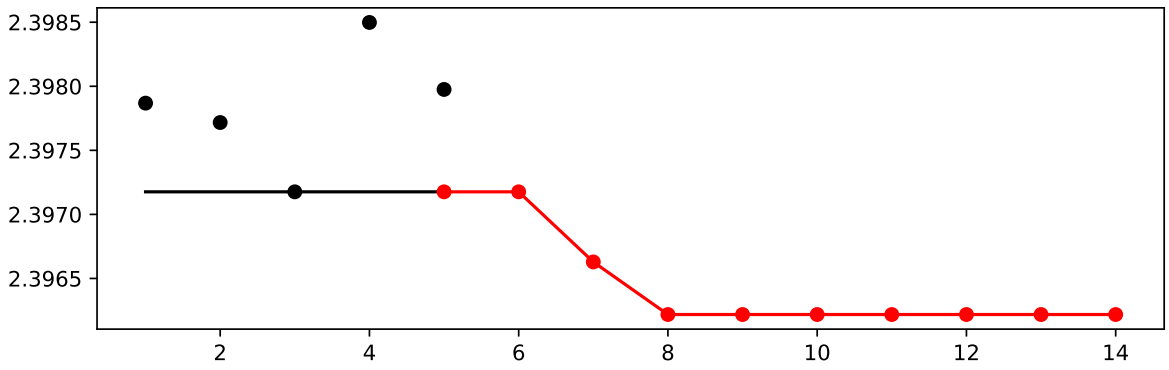
```
Loss on hold-out set: 2.3975883924736165
Accuracy on hold-out set: 0.1179245283018868
MAPK value on hold-out data: 0.17924530804157257
Epoch: 2

Loss on hold-out set: 2.3975256051657334
Accuracy on hold-out set: 0.1179245283018868
MAPK value on hold-out data: 0.17924530804157257
Epoch: 3

Loss on hold-out set: 2.397454376490611
Accuracy on hold-out set: 0.1179245283018868
MAPK value on hold-out data: 0.17924530804157257
Epoch: 4

Loss on hold-out set: 2.397371346095823
Accuracy on hold-out set: 0.1179245283018868
MAPK value on hold-out data: 0.17924530804157257
Returned to Spot: Validation loss: 2.397371346095823
-----------------------------------------------
spotPython tuning: 2.3962184168257803 [######----] 60.06%


config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.07351665686388897, 'lr_mult': 0.001, 'ba
Epoch: 1

Loss on hold-out set: 2.3979607865495502
Accuracy on hold-out set: 0.09433962264150944
MAPK value on hold-out data: 0.15172958374023438
Epoch: 2

Loss on hold-out set: 2.397653244576364
Accuracy on hold-out set: 0.1320754716981132
MAPK value on hold-out data: 0.17767296731472015
Epoch: 3

Loss on hold-out set: 2.3972324267873226
Accuracy on hold-out set: 0.13679245283018868
MAPK value on hold-out data: 0.18632075190544128
Epoch: 4
```

```
Loss on hold-out set: 2.3966896129104325
Accuracy on hold-out set: 0.1320754716981132
MAPK value on hold-out data: 0.19025158882141113
Returned to Spot: Validation loss: 2.3966896129104325
------------------------------------------------
spotPython tuning: 2.3962184168257803 [#######---] 72.09%


config: {'_L0': 6112, 'l1': 256, 'dropout_prob': 0.1378243435952057, 'lr_mult': 0.001, 'batch
Epoch: 1

Loss on hold-out set: 2.398113345200161
Accuracy on hold-out set: 0.08018867924528301
MAPK value on hold-out data: 0.15880505740642548
Epoch: 2

Loss on hold-out set: 2.398097260942999
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.1674528419971466
Epoch: 3

Loss on hold-out set: 2.397974470876298
Accuracy on hold-out set: 0.09433962264150944
MAPK value on hold-out data: 0.17059749364852905
Epoch: 4

Loss on hold-out set: 2.39794618903466
Accuracy on hold-out set: 0.07547169811320754
MAPK value on hold-out data: 0.1674528419971466
Returned to Spot: Validation loss: 2.39794618903466
------------------------------------------------
spotPython tuning: 2.3962184168257803 [########--] 82.24%


config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.2119462127828773, 'lr_mult': 0.001, 'batc
Epoch: 1

Loss on hold-out set: 2.3979113124451548
Accuracy on hold-out set: 0.0660377358490566
MAPK value on hold-out data: 0.14465412497520447
Epoch: 2
```

```
Loss on hold-out set: 2.3977622221101007
Accuracy on hold-out set: 0.08490566037735849
MAPK value on hold-out data: 0.15566039085388184
Epoch: 3

Loss on hold-out set: 2.3976518095664257
Accuracy on hold-out set: 0.10849056603773585
MAPK value on hold-out data: 0.1737421303987503
Epoch: 4

Loss on hold-out set: 2.397615504714678
Accuracy on hold-out set: 0.08490566037735849
MAPK value on hold-out data: 0.16352200508117676
Returned to Spot: Validation loss: 2.397615504714678
---------------------------------------------
spotPython tuning: 2.3962184168257803 [#########-] 93.89%


config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.21919111995999893, 'lr_mult': 0.001, 'ba
Epoch: 1

Loss on hold-out set: 2.397657515867701
Accuracy on hold-out set: 0.12264150943396226
MAPK value on hold-out data: 0.16823901236057281
Epoch: 2

Loss on hold-out set: 2.3973958447294414
Accuracy on hold-out set: 0.09905660377358491
MAPK value on hold-out data: 0.15487425029277802
Epoch: 3

Loss on hold-out set: 2.3971469537267147
Accuracy on hold-out set: 0.07547169811320754
MAPK value on hold-out data: 0.14386793971061707
Epoch: 4

Loss on hold-out set: 2.3966928518043376
Accuracy on hold-out set: 0.08962264150943396
MAPK value on hold-out data: 0.1603773534297943
Returned to Spot: Validation loss: 2.3966928518043376
---------------------------------------------
spotPython tuning: 2.3962184168257803 [##########] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x2d4627490>
```

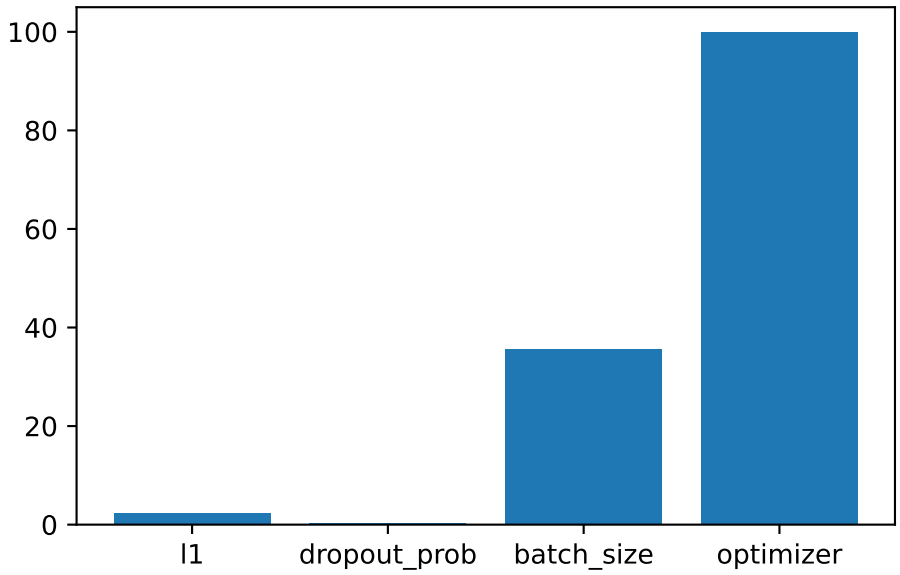
```
spot_tuner.plot_progress(log_y=False, filename="./figures/" + experiment_name+"_progress.p
```




```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
_L0	int	64	6112.0	6112.0	6112.0	None
l1	int	8	6.0	13.0	11.0	transform_pow
dropout_prob	float	0.01	0.0	0.9	0.19070372685884734	None
lr_mult	float	1.0	0.001	0.001	0.001	None
batch_size	int	4	1.0	4.0	1.0	transform_pow
epochs	int	4	2.0	2.0	2.0	transform_pow
k_folds	int	1	1.0	1.0	1.0	None
patience	int	2	2.0	6.0	4.0	transform_pow
optimizer	factor	SGD	0.0	3.0	3.0	None
sgd_momentum	float	0.0	0.9	0.9	0.9	None

```
spot_tuner.plot_importance()
```



23.4 Get the Tuned Architecture

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_vbdp(
  (fc1): Linear(in_features=6112, out_features=2048, bias=True)
  (fc2): Linear(in_features=2048, out_features=1024, bias=True)
  (fc3): Linear(in_features=1024, out_features=512, bias=True)
  (fc4): Linear(in_features=512, out_features=256, bias=True)
  (fc5): Linear(in_features=256, out_features=11, bias=True)
  (relu): ReLU()
  (softmax): Softmax(dim=1)
  (dropout1): Dropout(p=0.19070372685884734, inplace=False)
  (dropout2): Dropout(p=0.09535186342942367, inplace=False)
)
```

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
```

```
    )
train_tuned(net=model_spot, train_dataset=train,
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        shuffle=True,
        device = fun_control["device"],
        path=None,
        task=fun_control["task"],)
```

```
Epoch: 1

Loss on hold-out set: 2.3971693560762226
Accuracy on hold-out set: 0.12735849056603774
MAPK value on hold-out data: 0.2154088169336319
Epoch: 2

Loss on hold-out set: 2.3968633480791777
Accuracy on hold-out set: 0.12735849056603774
MAPK value on hold-out data: 0.22955970466136932
Epoch: 3

Loss on hold-out set: 2.3964038682433793
Accuracy on hold-out set: 0.1320754716981132
MAPK value on hold-out data: 0.25786158442497253
Epoch: 4

Loss on hold-out set: 2.3960499313642396
Accuracy on hold-out set: 0.14150943396226415
MAPK value on hold-out data: 0.2814464271068573
Returned to Spot: Validation loss: 2.3960499313642396
-----------------------------------------------
```

```
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"],
            task=fun_control["task"],)
```

```
Loss on hold-out set: 2.3960790098383185
Accuracy on hold-out set: 0.14124293785310735
MAPK value on hold-out data: 0.2584269940853119
Final evaluation: Validation loss: 2.3960790098383185
Final evaluation: Validation metric: 0.2584269940853119
---------------------------------------------


(2.3960790098383185, nan, tensor(0.2584))
```

23.5 Cross-validated Evaluations

- This is the evaluation that will be used in the comparison (evaluatecv has to be updated before, to get metric vlaues!):

```
from spotPython.torch.traintest import evaluate_cv
# modify k-kolds:
setattr(model_spot, "k_folds",  10)
df_eval, df_preds, df_metrics = evaluate_cv(net=model_spot, dataset=fun_control["data"], l
```

```
Fold: 1
Epoch: 1


Loss on hold-out set: 2.3975848489337497
Accuracy on hold-out set: 0.07042253521126761
MAPK value on hold-out data: 0.13425926864147186
Epoch: 2


Loss on hold-out set: 2.3972585002581277
Accuracy on hold-out set: 0.07042253521126761
MAPK value on hold-out data: 0.12962962687015533
Epoch: 3


Loss on hold-out set: 2.396460539764828
Accuracy on hold-out set: 0.09859154929577464
MAPK value on hold-out data: 0.14814816415309906
Epoch: 4
```

```
Loss on hold-out set: 2.3947698540157742
Accuracy on hold-out set: 0.08450704225352113
MAPK value on hold-out data: 0.20601852238178253
Fold: 2
Epoch: 1

Loss on hold-out set: 2.397291640440623
Accuracy on hold-out set: 0.11267605633802817
MAPK value on hold-out data: 0.16898146271705627
Epoch: 2

Loss on hold-out set: 2.3956269356939526
Accuracy on hold-out set: 0.15492957746478872
MAPK value on hold-out data: 0.23842591047286987
Epoch: 3

Loss on hold-out set: 2.3932102587487964
Accuracy on hold-out set: 0.18309859154929578
MAPK value on hold-out data: 0.2638888657093048
Epoch: 4

Loss on hold-out set: 2.389800343248579
Accuracy on hold-out set: 0.18309859154929578
MAPK value on hold-out data: 0.25
Fold: 3
Epoch: 1

Loss on hold-out set: 2.397821161482069
Accuracy on hold-out set: 0.08450704225352113
MAPK value on hold-out data: 0.13657407462596893
Epoch: 2

Loss on hold-out set: 2.397137224674225
Accuracy on hold-out set: 0.11267605633802817
MAPK value on hold-out data: 0.16435183584690094
Epoch: 3

Loss on hold-out set: 2.396116210354699
Accuracy on hold-out set: 0.11267605633802817
MAPK value on hold-out data: 0.17592592537403107
Epoch: 4
```

```
Loss on hold-out set: 2.3942671683099537
Accuracy on hold-out set: 0.1267605633802817
MAPK value on hold-out data: 0.21759259700775146
Fold: 4
Epoch: 1

Loss on hold-out set: 2.3973388539420233
Accuracy on hold-out set: 0.1267605633802817
MAPK value on hold-out data: 0.215277761220932
Epoch: 2

Loss on hold-out set: 2.39675509929657
Accuracy on hold-out set: 0.15492957746478872
MAPK value on hold-out data: 0.23379628360271454
Epoch: 3

Loss on hold-out set: 2.395556834008959
Accuracy on hold-out set: 0.18309859154929578
MAPK value on hold-out data: 0.2916666567325592
Epoch: 4

Loss on hold-out set: 2.3943753838539124
Accuracy on hold-out set: 0.14084507042253522
MAPK value on hold-out data: 0.28009259700775146
Fold: 5
Epoch: 1

Loss on hold-out set: 2.397038506137
Accuracy on hold-out set: 0.14084507042253522
MAPK value on hold-out data: 0.21064814925193787
Epoch: 2

Loss on hold-out set: 2.3966268168555365
Accuracy on hold-out set: 0.14084507042253522
MAPK value on hold-out data: 0.201388880610466
Epoch: 3

Loss on hold-out set: 2.39588944779502
Accuracy on hold-out set: 0.11267605633802817
MAPK value on hold-out data: 0.1805555373430252
Epoch: 4
```

```
Loss on hold-out set: 2.3948974278238087
Accuracy on hold-out set: 0.1267605633802817
MAPK value on hold-out data: 0.1736111044883728
Fold: 6
Epoch: 1

Loss on hold-out set: 2.397380643420749
Accuracy on hold-out set: 0.08450704225352113
MAPK value on hold-out data: 0.18518516421318054
Epoch: 2

Loss on hold-out set: 2.396389517519209
Accuracy on hold-out set: 0.08450704225352113
MAPK value on hold-out data: 0.19675925374031067
Epoch: 3

Loss on hold-out set: 2.3953447341918945
Accuracy on hold-out set: 0.08450704225352113
MAPK value on hold-out data: 0.21296292543411255
Epoch: 4

Loss on hold-out set: 2.3928520017200046
Accuracy on hold-out set: 0.09859154929577464
MAPK value on hold-out data: 0.24074074625968933
Fold: 7
Epoch: 1

Loss on hold-out set: 2.39737108680937
Accuracy on hold-out set: 0.08450704225352113
MAPK value on hold-out data: 0.1805555522441864
Epoch: 2

Loss on hold-out set: 2.3965147932370505
Accuracy on hold-out set: 0.09859154929577464
MAPK value on hold-out data: 0.1944444328546524
Epoch: 3

Loss on hold-out set: 2.395145058631897
Accuracy on hold-out set: 0.09859154929577464
MAPK value on hold-out data: 0.17592592537403107
Epoch: 4
```

```
Loss on hold-out set: 2.393059617943234
Accuracy on hold-out set: 0.09859154929577464
MAPK value on hold-out data: 0.17592591047286987
Fold: 8
Epoch: 1

Loss on hold-out set: 2.3975562504359655
Accuracy on hold-out set: 0.07142857142857142
MAPK value on hold-out data: 0.14761903882026672
Epoch: 2

Loss on hold-out set: 2.3964658805302212
Accuracy on hold-out set: 0.12857142857142856
MAPK value on hold-out data: 0.21190476417541504
Epoch: 3

Loss on hold-out set: 2.3951609747750418
Accuracy on hold-out set: 0.21428571428571427
MAPK value on hold-out data: 0.2690476179122925
Epoch: 4

Loss on hold-out set: 2.3927474498748778
Accuracy on hold-out set: 0.21428571428571427
MAPK value on hold-out data: 0.2571428716182709
Fold: 9
Epoch: 1

Loss on hold-out set: 2.397097260611398
Accuracy on hold-out set: 0.15714285714285714
MAPK value on hold-out data: 0.23095236718654633
Epoch: 2

Loss on hold-out set: 2.396096120561872
Accuracy on hold-out set: 0.11428571428571428
MAPK value on hold-out data: 0.22142858803272247
Epoch: 3

Loss on hold-out set: 2.394495119367327
Accuracy on hold-out set: 0.08571428571428572
MAPK value on hold-out data: 0.16428570449352264
Epoch: 4
```

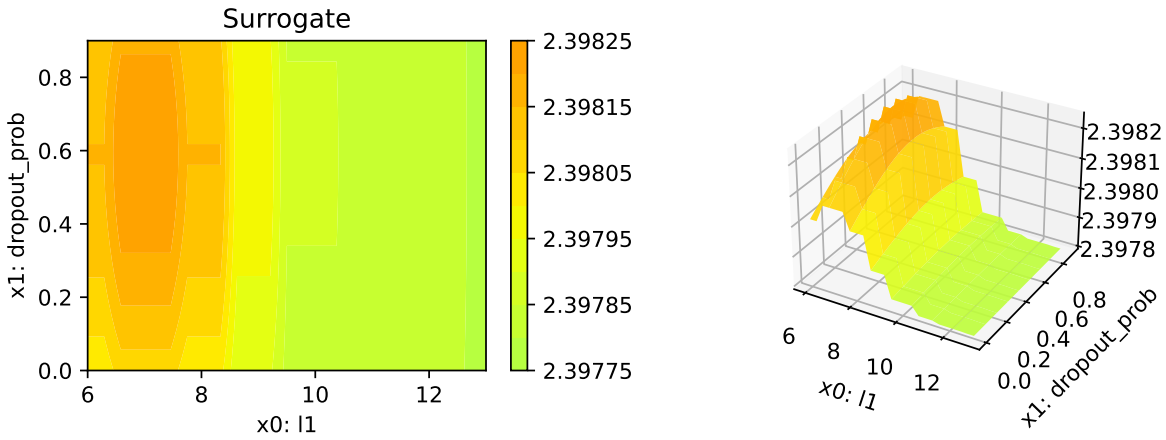
```
Loss on hold-out set: 2.392536074774606
Accuracy on hold-out set: 0.07142857142857142
MAPK value on hold-out data: 0.1666666567325592
Fold: 10
Epoch: 1

Loss on hold-out set: 2.396234117235456
Accuracy on hold-out set: 0.2
MAPK value on hold-out data: 0.3142857253551483
Epoch: 2

Loss on hold-out set: 2.3949395315987725
Accuracy on hold-out set: 0.24285714285714285
MAPK value on hold-out data: 0.31190478801727295
Epoch: 3

Loss on hold-out set: 2.3928964955466134
Accuracy on hold-out set: 0.2
MAPK value on hold-out data: 0.29523810744285583
Epoch: 4

Loss on hold-out set: 2.3897350651877267
Accuracy on hold-out set: 0.17142857142857143
MAPK value on hold-out data: 0.2904761731624603
```

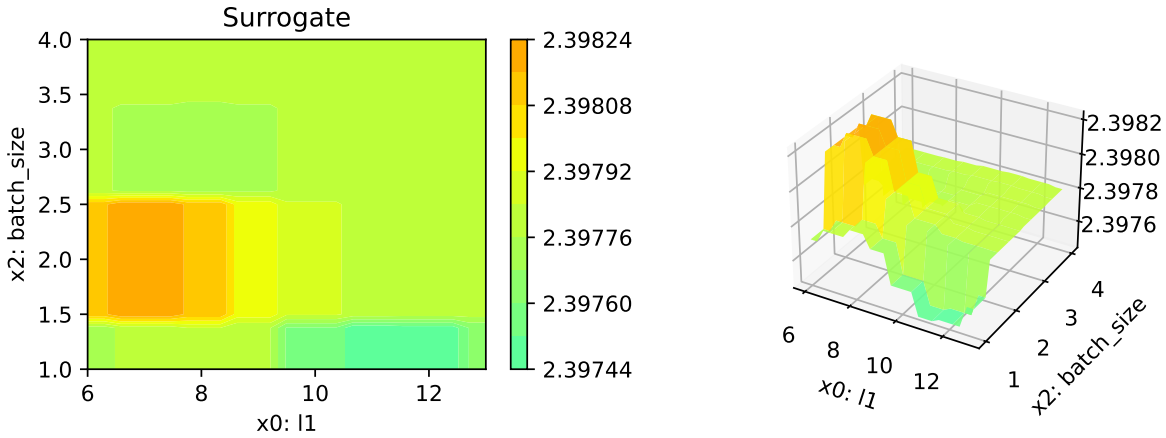
```
metric_name = type(fun_control["metric_torch"]).__name__
print(f"loss: {df_eval}, metric_name: {df_metrics}")
```

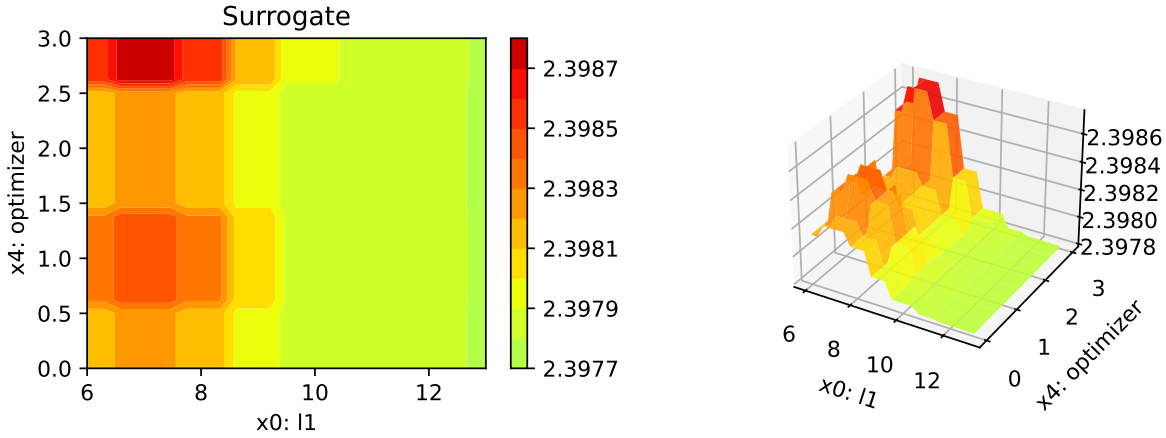
```
loss: 2.392904038675248, metric_name: 0.2258267104625702
```

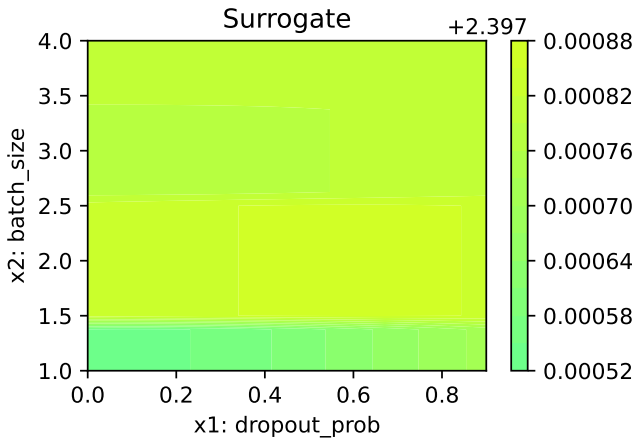
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

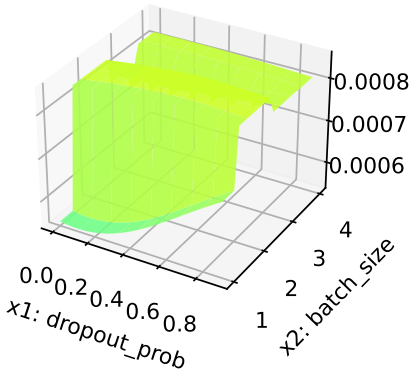
```
l1:  2.312572177956851
dropout_prob:  0.30018847721848785
batch_size:  35.69316590457506
optimizer:  100.0
```

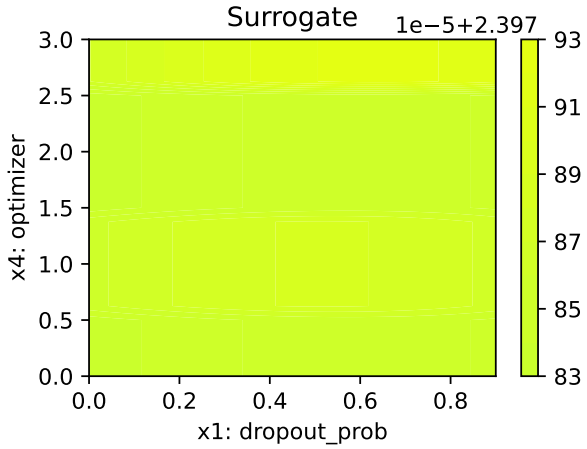

Surrogate
x1: dropout_prob
x0: l1
2.39825
2.39815
2.39805
2.39795
2.39785
2.39775
x0: l1
x1: dropout_prob
2.3982
2.3981
2.3980
2.3979
2.3978

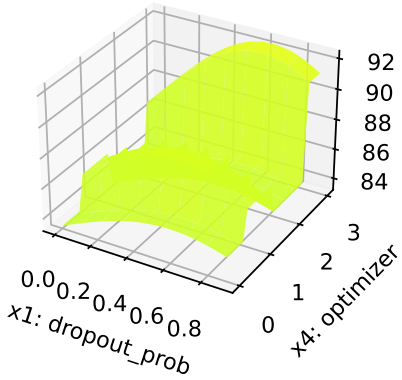

Surrogate
x2: batch_size
x0: l1
2.39824
2.39808
2.39792
2.39776
2.39760
2.39744
x0: l1
x2: batch_size
2.3982
2.3980
2.3978
2.3976

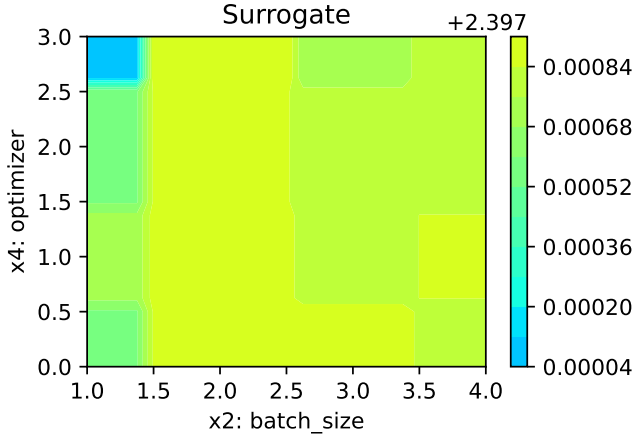

Surrogate
x4: optimizer
x0: l1
2.3987
2.3985
2.3983
2.3981
2.3979
2.3977
x0: l1
x4: optimizer
2.3986
2.3984
2.3982
2.3980
2.3978

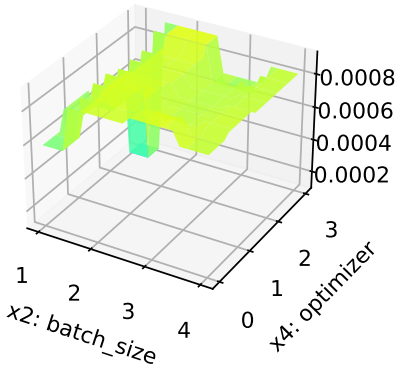

Surrogate
+2.397
x2: batch_size
x1: dropout_prob
0.00088
0.00082
0.00076
0.00070
0.00064
0.00058
0.00052


0.0008
0.0007
0.0006
x1: dropout_prob
x2: batch_size


Surrogate
1e−5+2.397
x4: optimizer
x1: dropout_prob
93
91
89
87
85
83


92
90
88
86
84
x1: dropout_prob
x4: optimizer


Surrogate
+2.397
x4: optimizer
x2: batch_size
0.00084
0.00068
0.00052
0.00036
0.00020
0.00004


0.0008
0.0006
0.0004
0.0002
x2: batch_size
x4: optimizer

23.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

```
# close tensorbaoard writer
if fun_control["writer"] is not None:
    fun_control["writer"].close()
```

23.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

24 Documentation of the Sequential Parameter Optimization

This document describes the `Spot` features.

24.1 Example: spot

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

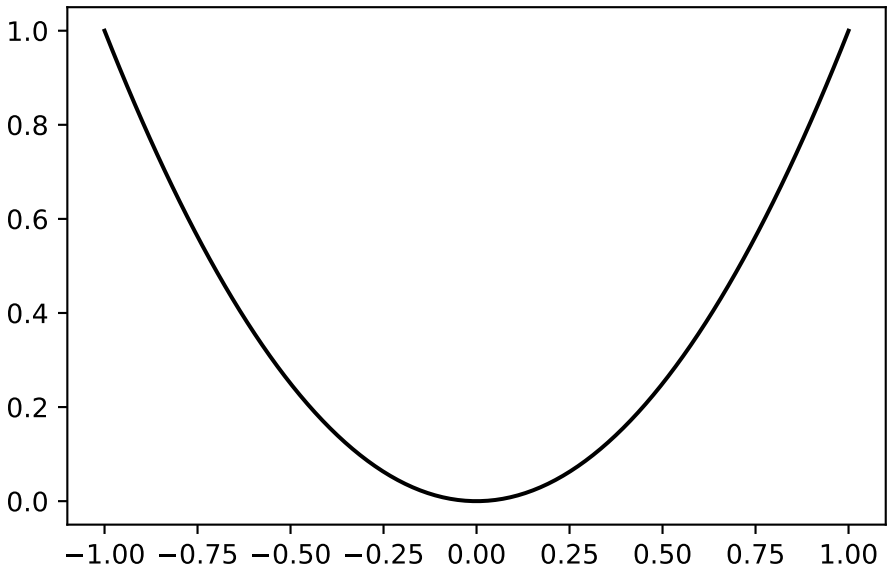
24.1.1 The Objective Function

The `spotPython` package provides several classes of objective functions. We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x) = x^2$$

```
fun = analytical().fun_sphere
```

```
x = np.linspace(-1,1,100).reshape(-1,1)
y = fun(x)
plt.figure()
plt.plot(x,y, "k")
plt.show()
```




```
spot_1 = spot.Spot(fun=fun,
                   lower = np.array([-10]),
                   upper = np.array([100]),
                   fun_evals = 7,
                   fun_repeats = 1,
                   max_time = inf,
                   noise = False,
                   tolerance_x = np.sqrt(np.spacing(1)),
                   var_type=["num"],
                   infill_criterion = "y",
                   n_points = 1,
                   seed=123,
                   log_level = 50,
                   show_models=True,
                   fun_control = {},
                   design_control={"init_size": 5,
                                   "repeats": 1},
                   surrogate_control={"noise": False,
                                      "cod_type": "norm",
                                      "min_theta": -4,
                                      "max_theta": 3,
                                      "n_theta": 1,
                                      "model_optimizer": differential_evolution,
                                      "model_fun_evals": 1000,
```

```
                                                    })
```

`spot`'s `__init__` method sets the control parameters. There are two parameter groups:

1. external parameters can be specified by the user
2. internal parameters, which are handled by `spot`.

24.1.2 External Parameters

external parameter	type	description	default	mandatory
`fun`	object	objective function		yes
`lower`	array	lower bound		yes
`upper`	array	upper bound		yes
`fun_evals`	int	number of function evaluations	15	no
`fun_evals`	int	number of function evaluations	15	no
`fun_control`	dict	noise etc.	{}	n
`max_time`	int	max run time budget	`inf`	no
`noise`	bool	if repeated evaluations of `fun` results in different values, then `noise` should be set to `True`.	`False`	no

external parameter	type	description	default	mandatory
`tolerance_x`	float	tolerance for new x solutions. Minimum distance of new solutions, generated by `suggest_new_X`, to already existing solutions. If zero (which is the default), every new solution is accepted.	`0`	no
`var_type`	list	list of type information, can be either `"num"` or `"factor"`	`["num"]`	no
`infill_criterion`	string	Can be `"y"`, `"s"`, `"ei"` (negative expected improvement), or `"all"`	`"y"`	no
`n_points`	int	number of infill points	1	no
`seed`	int	initial seed. If `Spot.run()` is called twice, different results will be generated. To reproduce results, the `seed` can be used.	`123`	no

external parameter	type	description	default	mandatory
`log_level`	int	log level with the following settings: `NOTSET` (`0`), `DEBUG` (`10`: Detailed information, typically of interest only when diagnosing problems.), `INFO` (`20`: Confirmation that things are working as expected.), `WARNING` (`30`: An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.), `ERROR` (`40`: Due to a more serious problem, the software has not been able to perform some function.), and `CRITICAL` (`50`: A serious error, indicating that the program itself may be unable to continue running.)	`50`	no

external parameter	type	description	default	mandatory
`show_models`	bool	Plot model. Currently only 1-dim functions are supported	`False`	no
`design`	object	experimental design	`None`	no
`design_control`	dict	control parameters	see below	no
`surrogate`		surrogate model	`kriging`	no
`surrogate_control`	dict	control parameters	see below	no
`optimizer`	object	optimizer	see below	no
`optimizer_control`	dict	control parameters	see below	no

- Besides these single parameters, the following parameter dictionaries can be specified by the user:
  - `fun_control`
  - `design_control`
  - `surrogate_control`
  - `optimizer_control`

24.2 The `fun_control` Dictionary

external parameter	type	description	default	mandatory
`sigma`	float	noise: standard deviation	`0`	yes
`seed`	int	seed for rng	`124`	yes

24.3 The `design_control` Dictionary

external parameter	type	description	default	mandatory
`init_size`	int	initial sample size	`10`	yes

external parameter	type	description	default	mandatory
`repeats`	int	number of repeats of the initial sammples	`1`	yes

24.4 The `surrogate_control` Dictionary

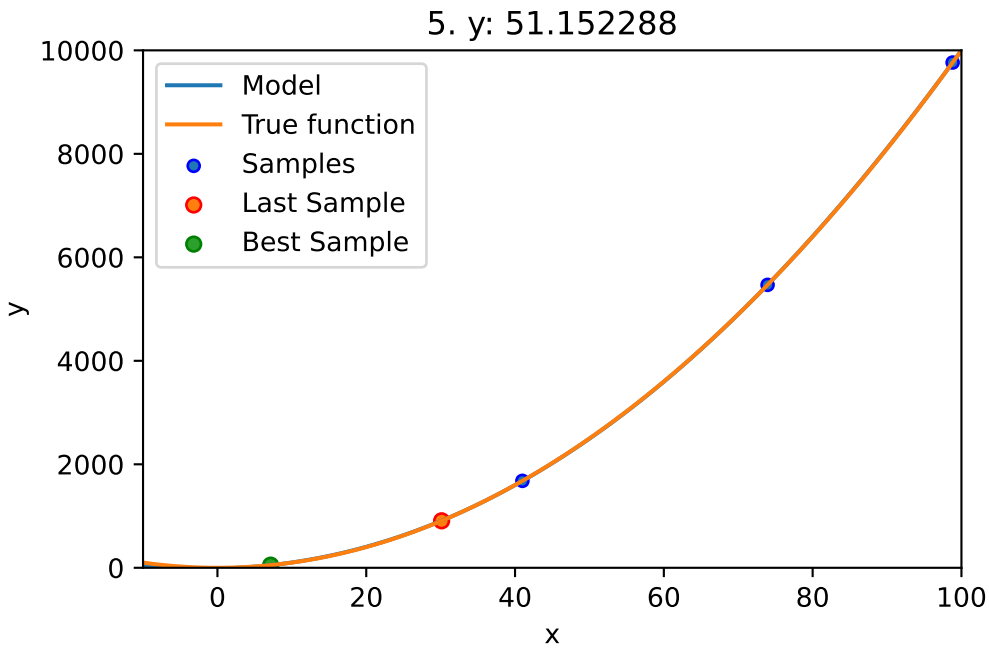
external parameter	type	description	default	mandatory
`noise_model_optimizer`	object	optimizer	`differential_evolution`	no
`model_fun_evals`				
`min_theta`			`-3.`	
`max_theta`			`3.`	
`n_theta`			`1`	
`n_p`			`1`	
`optim_p`			`False`	
`cod_type`			`"norm"`	
`var_type`				
`use_cod_y`	bool		`False`	

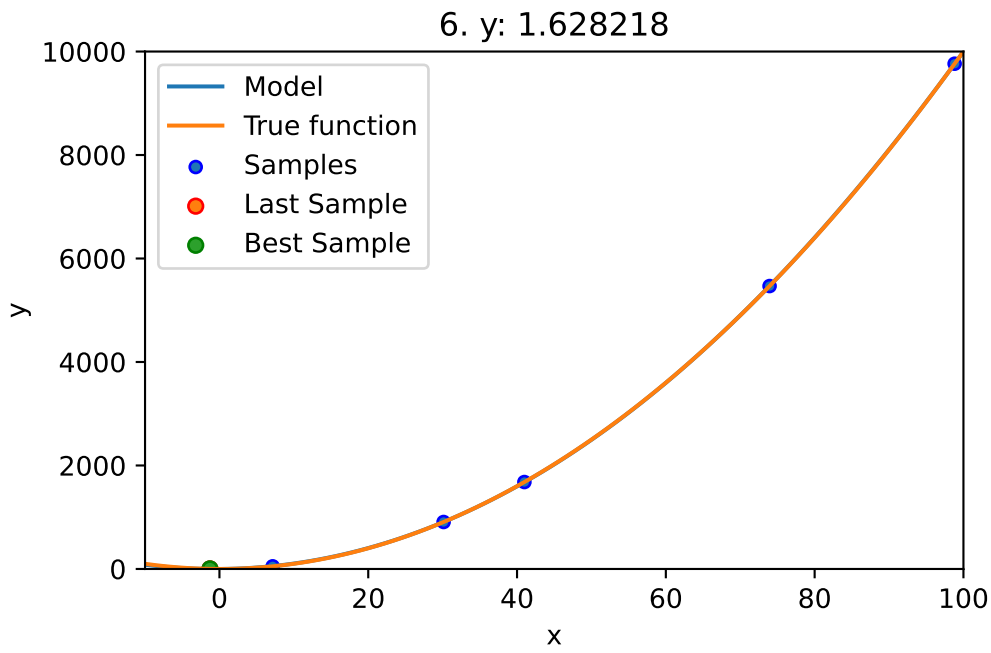
24.5 The `optimizer_control` Dictionary

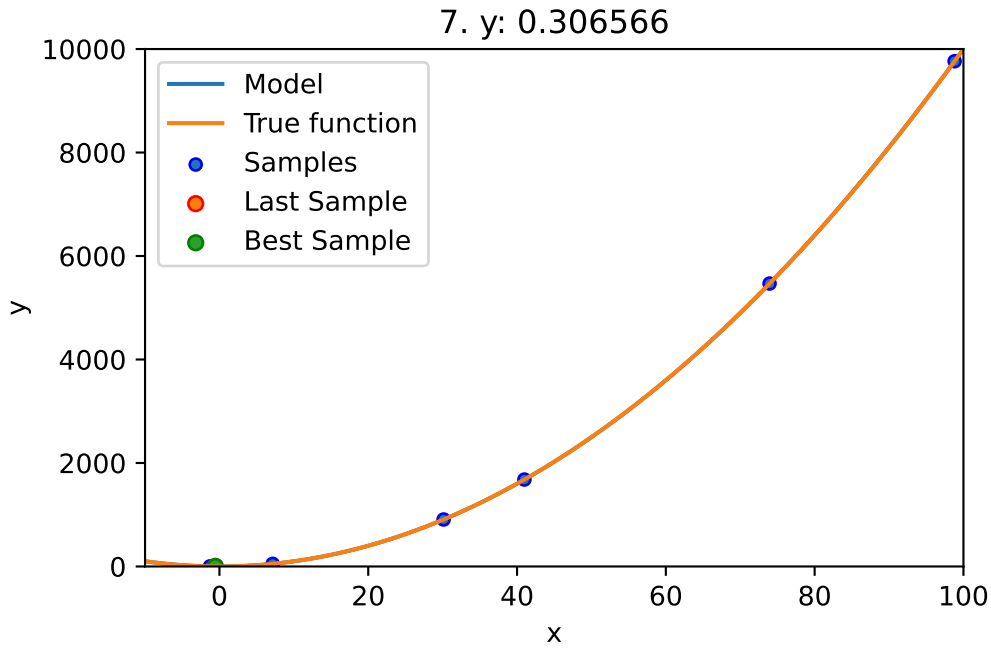
external parameter	type	description	default	mandatory
`max_iter`	int	max number of iterations. Note: these are the cheap evaluations on the surrogate.	`1000`	no

24.6 Run

```
spot_1.run()
```








```
<spotPython.spot.spot.Spot at 0x16ae8a920>
```

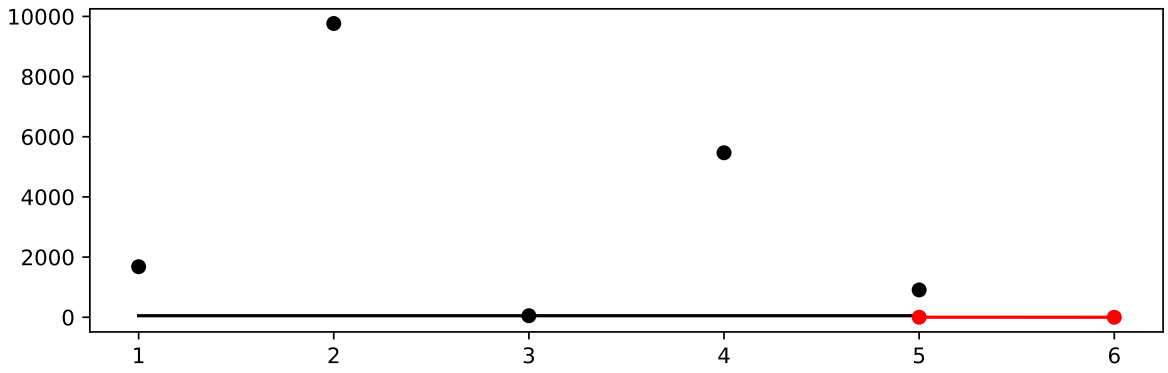
24.7 Print the Results

```
spot_1.print_results()
```

```
min y: 0.30656551286610595
x0: -0.5536835855126157

[['x0', -0.5536835855126157]]
```

24.8 Show the Progress

```
spot_1.plot_progress()
```

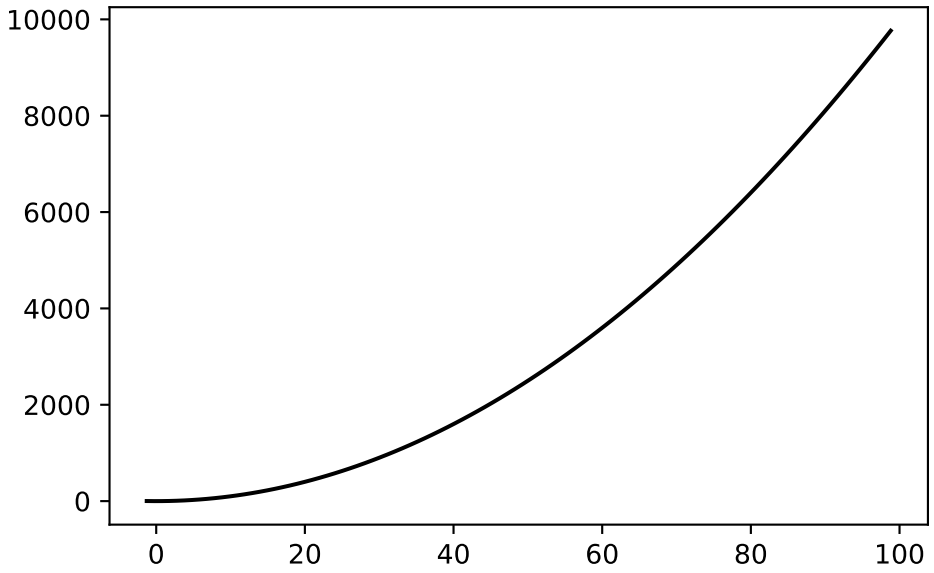


24.9 Visualize the Surrogate

- The plot method of the `kriging` surrogate is used.
- Note: the plot uses the interval defined by the ranges of the natural variables.

```
spot_1.surrogate.plot()
```

```
<Figure size 2700x1800 with 0 Axes>
```



24.10 Init: Build Initial Design

```
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
from spotPython.fun.objectivefunctions import analytical
gen = spacefilling(2)
rng = np.random.RandomState(1)
lower = np.array([-5,-0])
upper = np.array([10,15])
fun = analytical().fun_branin
fun_control = {"sigma": 0,
               "seed": 123}

X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
print(y)
```

```
[[ 8.97647221 13.41926847]
 [ 0.66946019  1.22344228]
 [ 5.23614115 13.78185824]
 [ 5.6149825  11.5851384 ]
```

```
 [-1.72963184  1.66516096]
 [-4.26945568  7.1325531 ]
 [ 1.26363761 10.17935555]
 [ 2.88779942  8.05508969]
 [-3.39111089  4.15213772]
 [ 7.30131231  5.22275244]]
[128.95676449  31.73474356 172.89678121 126.71295908  64.34349975
  70.16178611  48.71407916  31.77322887  76.91788181  30.69410529]
```

24.11 Replicability

Seed

```
gen = spacefilling(2, seed=123)
X0 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=345)
X1 = gen.scipy_lhd(3)
X2 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=123)
X3 = gen.scipy_lhd(3)
X0, X1, X2, X3
```

```
(array([[0.77254938, 0.31539299],
        [0.59321338, 0.93854273],
        [0.27469803, 0.3959685 ]]),
 array([[0.78373509, 0.86811887],
        [0.06692621, 0.6058029 ],
        [0.41374778, 0.00525456]]),
 array([[0.121357  , 0.69043832],
        [0.41906219, 0.32838498],
        [0.86742658, 0.52910374]]),
 array([[0.77254938, 0.31539299],
        [0.59321338, 0.93854273],
        [0.27469803, 0.3959685 ]]))
```

24.12 Surrogates

24.12.1 A Simple Predictor

The code below shows how to use a simple model for prediction. Assume that only two (very costly) measurements are available:

1. f(0) = 0.5
2. f(2) = 2.5

We are interested in the value at $x_0 = 1$, i.e., $f(x_0 = 1)$, but cannot run an additional, third experiment.

```
from sklearn import linear_model
X = np.array([[0], [2]])
y = np.array([0.5, 2.5])
S_lm = linear_model.LinearRegression()
S_lm = S_lm.fit(X, y)
X0 = np.array([[1]])
y0 = S_lm.predict(X0)
print(y0)
```

```
[1.5]
```

Central Idea: Evaluation of the surrogate model `S_lm` is much cheaper (or / and much faster) than running the real-world experiment f.

24.13 Demo/Test: Objective Function Fails

SPOT expects `np.nan` values from failed objective function values. These are handled. Note: SPOT's counter considers only successful executions of the objective function.

```
import numpy as np
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
import numpy as np
from math import inf
# number of initial points:
ni = 20
# number of points
n = 30
```

```
fun = analytical().fun_random_error
lower = np.array([-1])
upper = np.array([1])
design_control={"init_size": ni}

spot_1 = spot.Spot(fun=fun,
            lower = lower,
            upper= upper,
            fun_evals = n,
            show_progress=False,
            design_control=design_control,)
spot_1.run()
# To check whether the run was successfully completed,
# we compare the number of evaluated points to the specified
# number of points.
assert spot_1.y.shape[0] == n
```

```
[ 0.53176481 -0.9053821  -0.02203599 -0.21843718  0.78240941 -0.58120945
 -0.3923345   0.67234256  0.31802454         nan -0.75129705  0.97550354
         nan  0.0786237          nan  0.23700598 -0.49274073 -0.82319082
 -0.17991251  0.1481835 ]
[-1.]

[-0.47259301]

[0.95541987]

[0.17335968]
[-0.58552368]

[-0.20126111]
[-0.60100809]

[-0.97897336]

[-0.2748985]
[0.8359486]

[0.99035591]
[0.01641232]
```

```
[0.5629346]
```

24.14 PyTorch: Detailed Description of the Data Splitting

24.14.1 Description of the "train_hold_out" Setting

The "train_hold_out" setting is used by default. It uses the loss function specfied in fun_control and the metric specified in fun_control.

1. First, the method HyperTorch().fun_torch is called.
2. fun_torc(), which is implemented in the file hypertorch.py, calls evaluate_hold_out() as follows:

```
df_eval, _ = evaluate_hold_out(
    model,
    train_dataset=fun_control["train"],
    shuffle=self.fun_control["shuffle"],
    loss_function=self.fun_control["loss_function"],
    metric=self.fun_control["metric_torch"],
    device=self.fun_control["device"],
    show_batch_interval=self.fun_control["show_batch_interval"],
    path=self.fun_control["path"],
    task=self.fun_control["task"],
    writer=self.fun_control["writer"],
    writerId=config_id,
)
```

Note: Only the data set fun_control["train"] is used for training and validation. It is used in evaluate_hold_out as follows:

```
trainloader, valloader = create_train_val_data_loaders(
                dataset=train_dataset, batch_size=batch_size_instance, shuffle=shuffle
            )
```

create_train_val_data_loaders() splits the train_dataset into trainloader and valloader using torch.utils.data.random_split() as follows:

```
def create_train_val_data_loaders(dataset, batch_size, shuffle, num_workers=0):
    test_abs = int(len(dataset) * 0.6)
    train_subset, val_subset = random_split(dataset, [test_abs, len(dataset) - test_abs])
    trainloader = torch.utils.data.DataLoader(
        train_subset, batch_size=int(batch_size), shuffle=shuffle, num_workers=num_workers
    )
    valloader = torch.utils.data.DataLoader(
```

```
        val_subset, batch_size=int(batch_size), shuffle=shuffle, num_workers=num_workers
    )
    return trainloader, valloader
```

The optimizer is set up as follows:

```
optimizer_instance = net.optimizer
lr_mult_instance = net.lr_mult
sgd_momentum_instance = net.sgd_momentum
optimizer = optimizer_handler(
    optimizer_name=optimizer_instance,
    params=net.parameters(),
    lr_mult=lr_mult_instance,
    sgd_momentum=sgd_momentum_instance,
)
```

3. `evaluate_hold_out()` sets the `net` attributes such as `epochs`, `batch_size`, `optimizer`, and `patience`. For each epoch, the methods `train_one_epoch()` and `validate_one_epoch()` are called, the former for training and the latter for validation and early stopping. The validation loss from the last epoch (not the best validation loss) is returned from `evaluate_hold_out`.
4. The method `train_one_epoch()` is implemented as follows:

```
def train_one_epoch(
    net,
    trainloader,
    batch_size,
    loss_function,
    optimizer,
    device,
    show_batch_interval=10_000,
    task=None,
):
    running_loss = 0.0
    epoch_steps = 0
    for batch_nr, data in enumerate(trainloader, 0):
        input, target = data
        input, target = input.to(device), target.to(device)
        optimizer.zero_grad()
        output = net(input)
        if task == "regression":
```

```
            target = target.unsqueeze(1)
            if target.shape == output.shape:
                loss = loss_function(output, target)
            else:
                raise ValueError(f"Shapes of target and output do not match:
                 {target.shape} vs {output.shape}")
        elif task == "classification":
            loss = loss_function(output, target)
        else:
            raise ValueError(f"Unknown task: {task}")
        loss.backward()
        torch.nn.utils.clip_grad_norm_(net.parameters(), max_norm=1.0)
        optimizer.step()
        running_loss += loss.item()
        epoch_steps += 1
        if batch_nr % show_batch_interval == (show_batch_interval - 1):
            print(
                "Batch: %5d. Batch Size: %d. Training Loss (running): %.3f"
                % (batch_nr + 1, int(batch_size), running_loss / epoch_steps)
            )
            running_loss = 0.0
    return loss.item()
```

5. The method `validate_one_epoch()` is implemented as follows:

```
def validate_one_epoch(net, valloader, loss_function, metric, device, task):
    val_loss = 0.0
    val_steps = 0
    total = 0
    correct = 0
    metric.reset()
    for i, data in enumerate(valloader, 0):
        # get batches
        with torch.no_grad():
            input, target = data
            input, target = input.to(device), target.to(device)
            output = net(input)
            # print(f"target: {target}")
            # print(f"output: {output}")
            if task == "regression":
                target = target.unsqueeze(1)
```

```
                if target.shape == output.shape:
                    loss = loss_function(output, target)
                else:
                    raise ValueError(f"Shapes of target and output
                        do not match: {target.shape} vs {output.shape}")
                metric_value = metric.update(output, target)
            elif task == "classification":
                loss = loss_function(output, target)
                metric_value = metric.update(output, target)
                _, predicted = torch.max(output.data, 1)
                total += target.size(0)
                correct += (predicted == target).sum().item()
            else:
                raise ValueError(f"Unknown task: {task}")
            val_loss += loss.cpu().numpy()
            val_steps += 1
    loss = val_loss / val_steps
    print(f"Loss on hold-out set: {loss}")
    if task == "classification":
        accuracy = correct / total
        print(f"Accuracy on hold-out set: {accuracy}")
    # metric on all batches using custom accumulation
    metric_value = metric.compute()
    metric_name = type(metric).__name__
    print(f"{metric_name} value on hold-out data: {metric_value}")
    return metric_value, loss
```

24.14.1.1 Description of the `"test_hold_out"` Setting

It uses the loss function specfied in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torc()` calls `spotPython.torch.traintest.evaluate_hold_out()` similar to the `"train_hold_out"` setting with one exception: It passes an additional `test` data set to `evaluate_hold_out()` as follows:

```
test_dataset=fun_control["test"]
```

`evaluate_hold_out()` calls `create_train_test_data_loaders` instead of `create_train_val_data_loaders`: The two data sets are used in `create_train_test_data_loaders` as follows:

```
def create_train_test_data_loaders(dataset, batch_size, shuffle, test_dataset,
        num_workers=0):
    trainloader = torch.utils.data.DataLoader(
        dataset, batch_size=int(batch_size), shuffle=shuffle,
        num_workers=num_workers
    )
    testloader = torch.utils.data.DataLoader(
        test_dataset, batch_size=int(batch_size), shuffle=shuffle,
        num_workers=num_workers
    )
    return trainloader, testloader
```

3. The following steps are identical to the `"train_hold_out"` setting. Only a different data loader is used for testing.

24.14.1.2 Detailed Description of the `"train_cv"` Setting

It uses the loss function specfied in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torc()` calls `spotPython.torch.traintest.evaluate_cv()` as follows (Note: Only the data set `fun_control["train"]` is used for CV.):

```
df_eval, _ = evaluate_cv(
    model,
    dataset=fun_control["train"],
    shuffle=self.fun_control["shuffle"],
    device=self.fun_control["device"],
    show_batch_interval=self.fun_control["show_batch_interval"],
    task=self.fun_control["task"],
    writer=self.fun_control["writer"],
    writerId=config_id,
)
```

3. In ‘evaluate_cv(), the following steps are performed: The optimizer is set up as follows:

```
optimizer_instance = net.optimizer
lr_instance = net.lr
sgd_momentum_instance = net.sgd_momentum
optimizer = optimizer_handler(optimizer_name=optimizer_instance,
     params=net.parameters(), lr_mult=lr_mult_instance)
```

`evaluate_cv()` sets the `net` attributes such as `epochs`, `batch_size`, `optimizer`, and `patience`. CV is implemented as follows:

```
def evaluate_cv(
    net,
    dataset,
    shuffle=False,
    loss_function=None,
    num_workers=0,
    device=None,
    show_batch_interval=10_000,
    metric=None,
    path=None,
    task=None,
    writer=None,
    writerId=None,
):
    lr_mult_instance = net.lr_mult
    epochs_instance = net.epochs
    batch_size_instance = net.batch_size
    k_folds_instance = net.k_folds
    optimizer_instance = net.optimizer
    patience_instance = net.patience
    sgd_momentum_instance = net.sgd_momentum
    removed_attributes, net = get_removed_attributes_and_base_net(net)
    metric_values = {}
    loss_values = {}
    try:
        device = getDevice(device=device)
        if torch.cuda.is_available():
            device = "cuda:0"
            if torch.cuda.device_count() > 1:
                print("We will use", torch.cuda.device_count(), "GPUs!")
                net = nn.DataParallel(net)
        net.to(device)
        optimizer = optimizer_handler(
            optimizer_name=optimizer_instance,
            params=net.parameters(),
            lr_mult=lr_mult_instance,
            sgd_momentum=sgd_momentum_instance,
        )
        kfold = KFold(n_splits=k_folds_instance, shuffle=shuffle)
```

```
        for fold, (train_ids, val_ids) in enumerate(kfold.split(dataset)):
            print(f"Fold: {fold + 1}")
            train_subsampler = torch.utils.data.SubsetRandomSampler(train_ids)
            val_subsampler = torch.utils.data.SubsetRandomSampler(val_ids)
            trainloader = torch.utils.data.DataLoader(
                dataset, batch_size=batch_size_instance,
                sampler=train_subsampler, num_workers=num_workers
            )
            valloader = torch.utils.data.DataLoader(
                dataset, batch_size=batch_size_instance,
                sampler=val_subsampler, num_workers=num_workers
            )
            # each fold starts with new weights:
            reset_weights(net)
            # Early stopping parameters
            best_val_loss = float("inf")
            counter = 0
            for epoch in range(epochs_instance):
                print(f"Epoch: {epoch + 1}")
                # training loss from one epoch:
                training_loss = train_one_epoch(
                    net=net,
                    trainloader=trainloader,
                    batch_size=batch_size_instance,
                    loss_function=loss_function,
                    optimizer=optimizer,
                    device=device,
                    show_batch_interval=show_batch_interval,
                    task=task,
                )
                # Early stopping check. Calculate validation loss from one epoch:
                metric_values[fold], loss_values[fold] = validate_one_epoch(
                    net, valloader=valloader, loss_function=loss_function,
                    metric=metric, device=device, task=task
                )
                # Log the running loss averaged per batch
                metric_name = "Metric"
                if metric is None:
                    metric_name = type(metric).__name__
                    print(f"{metric_name} value on hold-out data:
                        {metric_values[fold]}")
```

```
                if writer is not None:
                    writer.add_scalars(
                        "evaluate_cv fold:" + str(fold + 1) +
                        ". Train & Val Loss and Val Metric" + writerId,
                        {"Train loss": training_loss, "Val loss":
                        loss_values[fold], metric_name: metric_values[fold]},
                        epoch + 1,
                    )
                    writer.flush()
                if loss_values[fold] < best_val_loss:
                    best_val_loss = loss_values[fold]
                    counter = 0
                    # save model:
                    if path is not None:
                        torch.save(net.state_dict(), path)
                else:
                    counter += 1
                    if counter >= patience_instance:
                        print(f"Early stopping at epoch {epoch}")
                        break
        df_eval = sum(loss_values.values()) / len(loss_values.values())
        df_metrics = sum(metric_values.values()) / len(metric_values.values())
        df_preds = np.nan
    except Exception as err:
        print(f"Error in Net_Core. Call to evaluate_cv() failed. {err=},
            {type(err)=}")
        df_eval = np.nan
        df_preds = np.nan
    add_attributes(net, removed_attributes)
    if writer is not None:
        metric_name = "Metric"
        if metric is None:
            metric_name = type(metric).__name__
        writer.add_scalars(
            "CV: Val Loss and Val Metric" + writerId,
            {"CV-loss": df_eval, metric_name: df_metrics},
            epoch + 1,
        )
        writer.flush()
    return df_eval, df_preds, df_metrics
```

4. The method `train_fold()` is implemented as shown above.

5. The method `validate_one_epoch()` is implemented as shown above. In contrast to the hold-out setting, it is called for each of the k folds. The results are stored in a dictionaries `metric_values` and `loss_values`. The results are averaged over the k folds and returned as `df_eval`.

24.14.1.3 Detailed Description of the `"test_cv"` Setting

It uses the loss function specfied in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torc()` calls `spotPython.torch.traintest.evaluate_cv()` as follows:

```
df_eval, _ = evaluate_cv(
    model,
    dataset=fun_control["test"],
    shuffle=self.fun_control["shuffle"],
    device=self.fun_control["device"],
    show_batch_interval=self.fun_control["show_batch_interval"],
    task=self.fun_control["task"],
    writer=self.fun_control["writer"],
    writerId=config_id,
)
```

Note: The data set `fun_control["test"]` is used for CV. The rest is the same as for the `"train_cv"` setting.

24.14.1.4 Detailed Description of the Final Model Training and Evaluation

There are two methods that can be used for the final evaluation of a Pytorch model:

1. `"train_tuned` and
2. `"test_tuned"`.

`train_tuned()` is just a wrapper to `evaluate_hold_out` using the `train` data set. It is implemented as follows:

```
def train_tuned(
    net,
    train_dataset,
    shuffle,
    loss_function,
    metric,
```

```
    device=None,
    show_batch_interval=10_000,
    path=None,
    task=None,
    writer=None,
):
    evaluate_hold_out(
        net=net,
        train_dataset=train_dataset,
        shuffle=shuffle,
        test_dataset=None,
        loss_function=loss_function,
        metric=metric,
        device=device,
        show_batch_interval=show_batch_interval,
        path=path,
        task=task,
        writer=writer,
    )
```

The `test_tuned()` procedure is implemented as follows:

```
def test_tuned(net, shuffle, test_dataset=None, loss_function=None,
    metric=None, device=None, path=None, task=None):
    batch_size_instance = net.batch_size
    removed_attributes, net = get_removed_attributes_and_base_net(net)
    if path is not None:
        net.load_state_dict(torch.load(path))
        net.eval()
    try:
        device = getDevice(device=device)
        if torch.cuda.is_available():
            device = "cuda:0"
            if torch.cuda.device_count() > 1:
                print("We will use", torch.cuda.device_count(), "GPUs!")
                net = nn.DataParallel(net)
        net.to(device)
        valloader = torch.utils.data.DataLoader(
            test_dataset, batch_size=int(batch_size_instance),
            shuffle=shuffle,
            num_workers=0
        )
```

```
        metric_value, loss = validate_one_epoch(
            net, valloader=valloader, loss_function=loss_function,
            metric=metric, device=device, task=task
        )
        df_eval = loss
        df_metric = metric_value
        df_preds = np.nan
    except Exception as err:
        print(f"Error in Net_Core. Call to test_tuned() failed. {err=},
            {type(err)=}")
        df_eval = np.nan
        df_metric = np.nan
        df_preds = np.nan
    add_attributes(net, removed_attributes)
    print(f"Final evaluation: Validation loss: {df_eval}")
    print(f"Final evaluation: Validation metric: {df_metric}")
    print("------------------------------------------------")
    return df_eval, df_preds, df_metric
```

References

Bartz, Eva, Thomas Bartz-Beielstein, Martin Zaefferer, and Olaf Mersmann, eds. 2022. *Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide*. Springer.

Bartz-Beielstein, Thomas. 2023. "PyTorch Hyperparameter Tuning with SPOT: Comparison with Ray Tuner and Default Hyperparameters on CIFAR10." https://github.com/sequential-parameter-optimization/spotPython/blob/main/notebooks/14_spot_ray_hpt_torch_cifar10.ipynb.

Bartz-Beielstein, Thomas, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. 2014. "Evolutionary Algorithms." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4 (3): 178–95.

Bartz-Beielstein, Thomas, Carola Doerr, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, et al. 2020. "Benchmarking in Optimization: Best Practice and Open Issues." arXiv. https://arxiv.org/abs/2007.03488.

Bartz-Beielstein, Thomas, Christian Lasarczyk, and Mike Preuss. 2005. "Sequential Parameter Optimization." In *Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland*, edited by B McKay et al., 773–80. Piscataway NJ: IEEE Press.

Lewis, R M, V Torczon, and M W Trosset. 2000. "Direct search methods: Then and now." *Journal of Computational and Applied Mathematics* 124 (1–2): 191–207.

Li, Lisha, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2016. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization." *arXiv e-Prints*, March, arXiv:1603.06560.

Meignan, David, Sigrid Knust, Jean-Marc Frayet, Gilles Pesant, and Nicolas Gaud. 2015. "A Review and Taxonomy of Interactive Optimization Methods in Operations Research." *ACM Transactions on Interactive Intelligent Systems*, September.

Montiel, Jacob, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, et al. 2021. "River: Machine Learning for Streaming Data in Python."

PyTorch. 2023a. "Hyperparameter Tuning with Ray Tune." https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html.

———. 2023b. "Training a Classifier." https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html.