Regression

6. Gradient descent

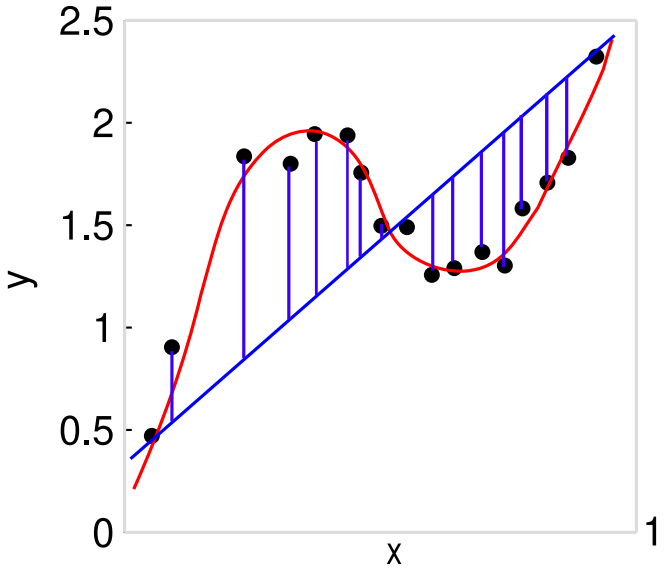
Olivier Sigaud

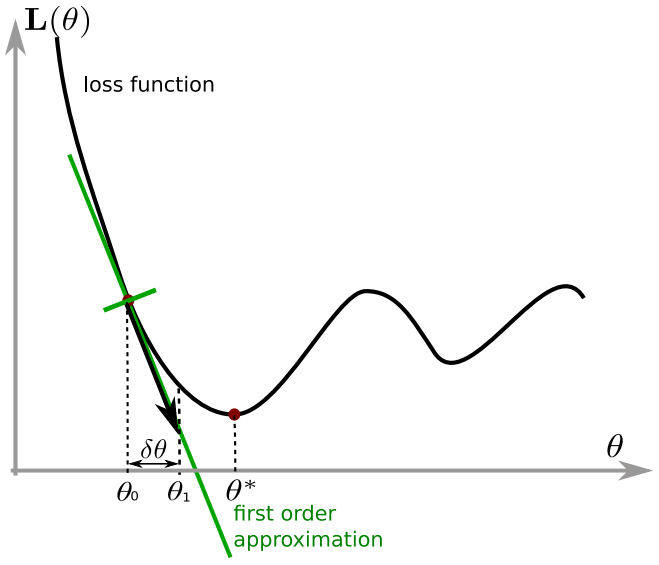
Sorbonne Université
http://people.isir.upmc.fr/sigaud

Outline

- In this class, we will see:
  - How regression can be cast as a gradient descent problem
  - What are the gradient descent properties independently from regression
  - What is the difference between batch and stochastic gradient descent
- More advanced methods will be explained in the next class

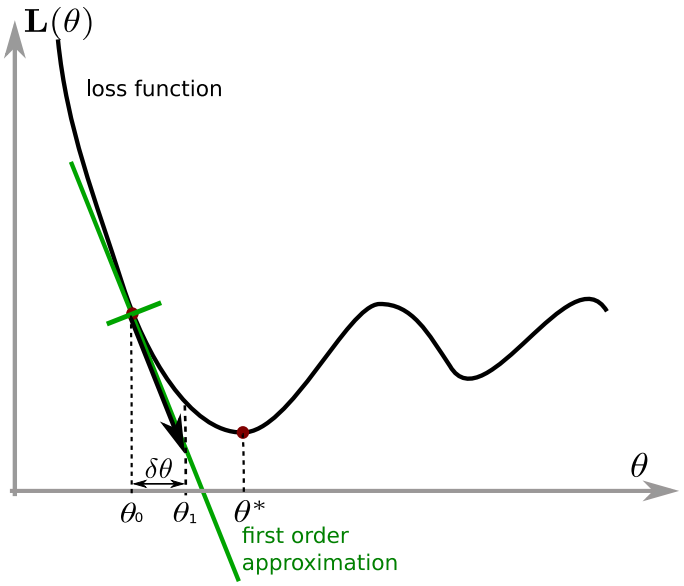
Regression through gradient descent





- We want to minimize a regression error (left)
- Thus find the minimum of a **loss function** $\mathbf{L}(\boldsymbol{\theta})$ (right)
- Instead of solving $\boldsymbol{\theta}^* = \min_{\boldsymbol{\theta}} \mathbf{L}(\boldsymbol{\theta})$ (immediate, but expensive)
- Rather perform local steps using an iterative approach
- The iteration equation is $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i + \delta\boldsymbol{\theta}_i$
- This is the essence of gradient descent

Gradient descent iteration

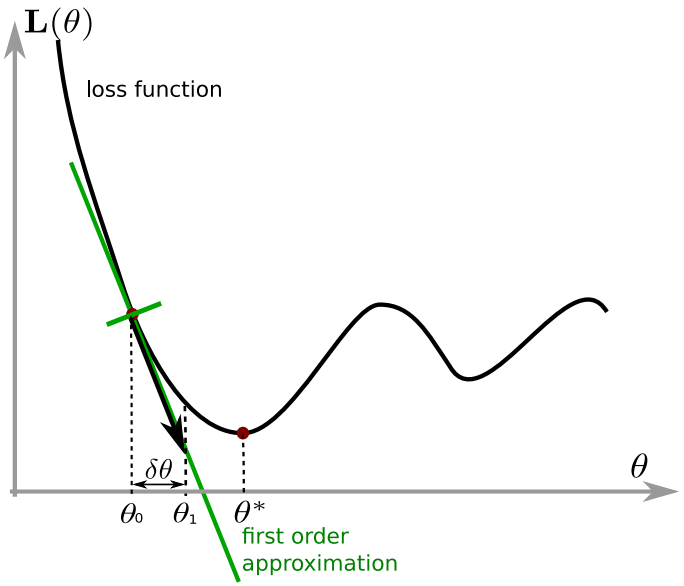


- Now we want to minimize $\mathbf{L}(\boldsymbol{\theta}_i + \delta\boldsymbol{\theta}_i)$ over $\delta\boldsymbol{\theta}_i$.
- How to choose $\delta\boldsymbol{\theta}_i$?

First order approximation of the loss

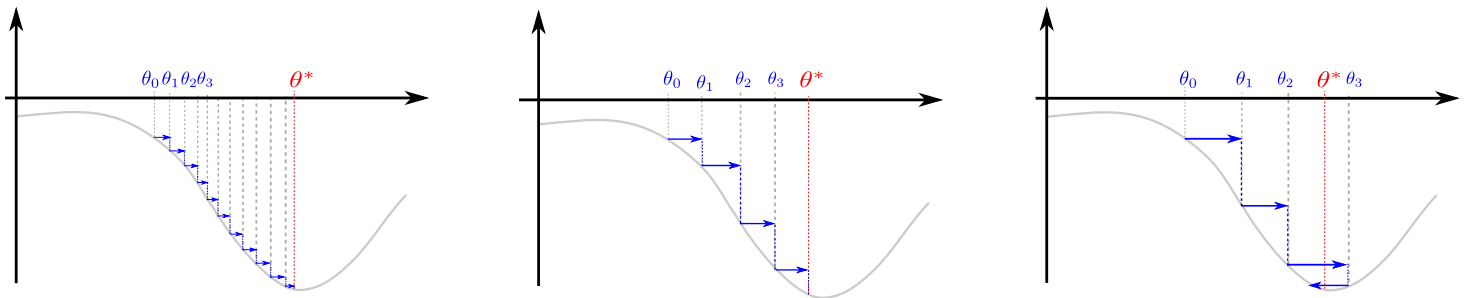
- The optimum on $\delta\boldsymbol{\theta}$ is reached when $\frac{\partial \mathbf{L}(\boldsymbol{\theta}+\delta\boldsymbol{\theta})}{\partial \delta\boldsymbol{\theta}} = 0$
- $\mathbf{L}(\boldsymbol{\theta}+\delta\boldsymbol{\theta})$ can be approximated at the first order as $\mathbf{L}(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})^T\delta\boldsymbol{\theta} + \nu\delta\boldsymbol{\theta}^T\delta\boldsymbol{\theta} +$ higher order terms
- $\frac{\partial \mathbf{L}(\boldsymbol{\theta}+\delta\boldsymbol{\theta})}{\partial \delta\boldsymbol{\theta}} \sim \nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})^T\delta\boldsymbol{\theta} + 2\nu\delta\boldsymbol{\theta}$
- Thus $\delta\boldsymbol{\theta}^* = -\frac{1}{2\nu}\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})$
- We rewrite it $\delta\boldsymbol{\theta}^* = -\alpha\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})$
- And the iteration rule is $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i - \alpha_i\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})$
- Thus the steepest descent direction is given by the first order derivative

Gradient descent iteration



- The first order local derivative $\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_i}$ gives the right direction (move left or right)
- But minimizing the first order derivative is not lower bounded
- We need a step size α to determine how far to go

Tuning the step size

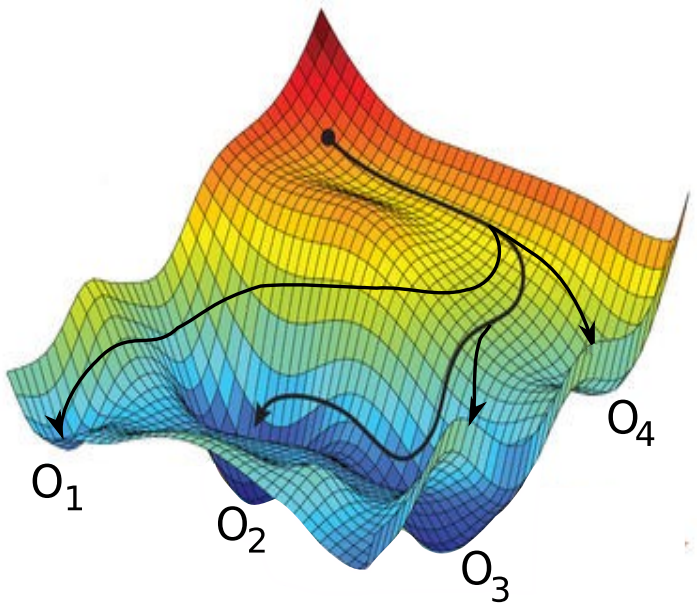


(a) (b) (c)

- (a) α_i are too small, (b) α_i are adequate, (c) α_i are too large
- If too small, too many steps. If too large, may miss a local optimum
- Line search: iterate to find the best step size (used e.g. in TRPO)

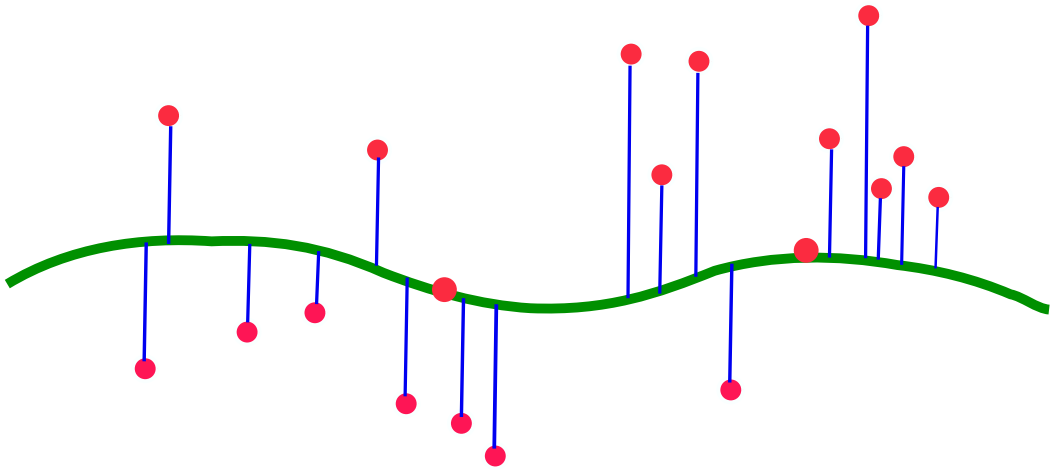
Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015) Trust region policy optimization. *CoRR, abs/1502.05477*

Local Optima



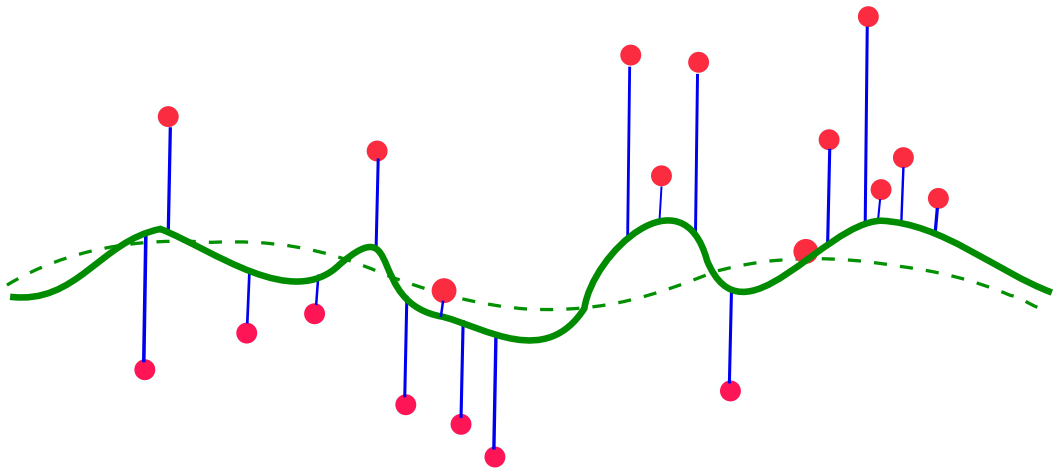
- Gradient descent is a local improvement approach where, at each step, we follow the steepest descent direction
- But we don't know where is the optimum we are targetting
- Unless the cost function is convex in the parameter space, gradient descent can end-up in different local optima depending on the starting point or noise
- Anything that basically goes down will end up somewhere low!

Batch regression through gradient descent



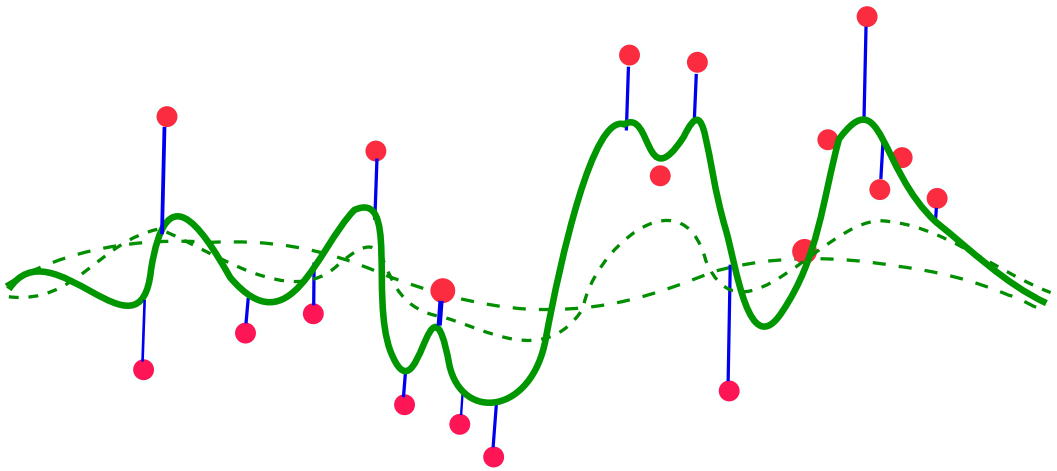
- The loss is the squared sum of all errors $\mathbf{L}(\boldsymbol{\theta}) = \|\mathbf{y} - \hat{f}(\mathbf{X})\|^2$
- The weights of the regressor $\hat{f}$ are updated so as to decrease all errors on average
- The maths make it so that the regressor is corrected more where it makes larger errors

Batch regression through gradient descent



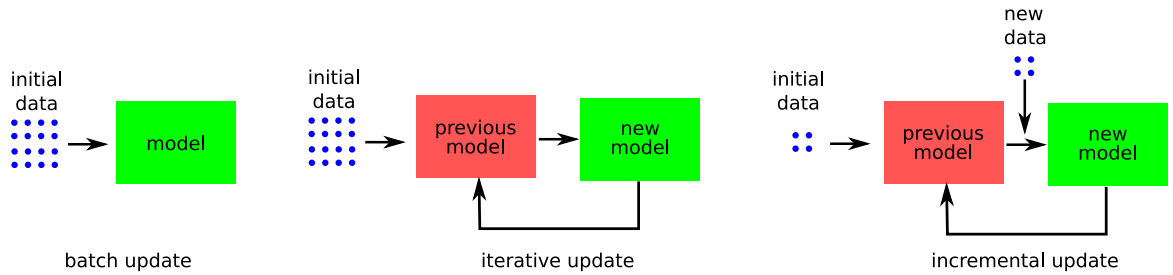
- The loss is the squared sum of all errors $\mathbf{L}(\boldsymbol{\theta}) = \|\mathbf{y} - \hat{f}(\mathbf{X})\|^2$
- The weights of the regressor $\hat{f}$ are updated so as to decrease all errors on average
- The maths make it so that the regressor is corrected more where it makes larger errors

Batch regression through gradient descent



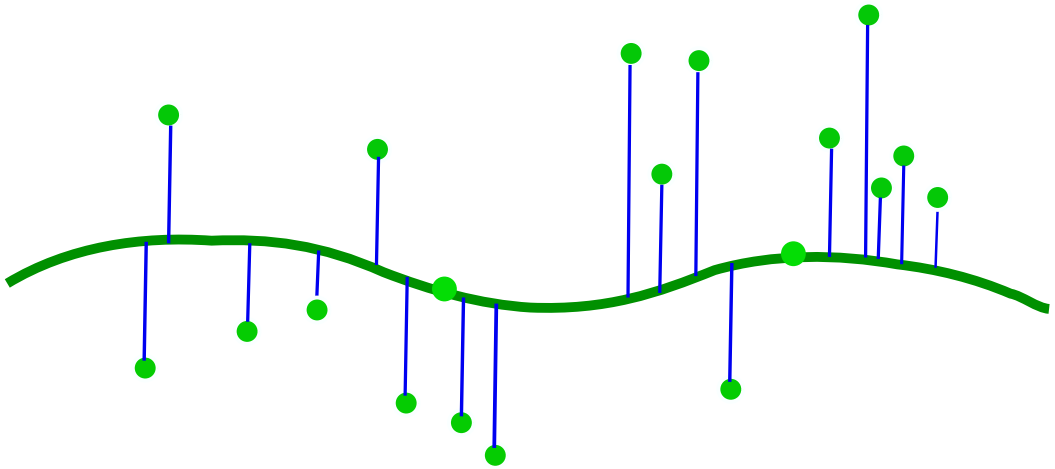
- The loss is the squared sum of all errors $\mathbf{L}(\boldsymbol{\theta}) = \|\mathbf{y} - \hat{f}(\mathbf{X})\|^2$
- The weights of the regressor $\hat{f}$ are updated so as to decrease all errors on average
- The maths make it so that the regressor is corrected more where it makes larger errors

Reminder: Batch, Iterative, Incremental



- In regression, the cost function or loss is the regression error (e.g. $\mathbf{L}(\boldsymbol{\theta}) = \|\mathbf{y} - \hat{f}(\mathbf{X})\|^2$)
- The model function $\hat{f}(\mathbf{x})$ is known, so its derivative $\nabla_{\boldsymbol{\theta}} \hat{f}(\mathbf{x})$ can be computed analytically
- The values $\mathbf{y}$ and $\mathbf{X}$ are known too
- Hence batch gradient descent is only iterative: no need for additional data
- Stochastic gradient descent is a specific case of incremental ($\mathbf{y}$ and $\mathbf{X}$ change for each mini-batch)
- The new datapoints are taken randomly from the batch

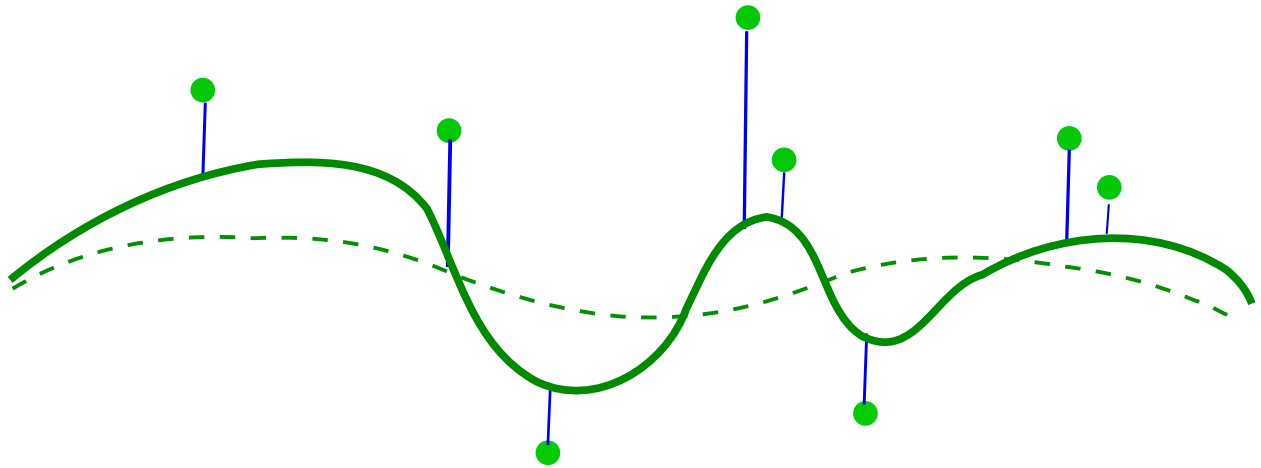
Stochastic gradient descent (SGD)



- Batch gradient descent is expensive when the batch is large
- We can decrease the cost by using several mini-batches (extra gain with parallel computations)
- In SGD, each mini-batch gives a different (inaccurate) estimate of the local gradient
- Combining several inaccurate estimates can help being accurate
- Actually, being inaccurate is no big deal (see Class 7)
- Note that SGD converges more slowly. Use it only for large batches.

Bottou, L. (2012) Stochastic gradient descent tricks. *Neural networks: Tricks of the trade*, pp. 421–436. Springer

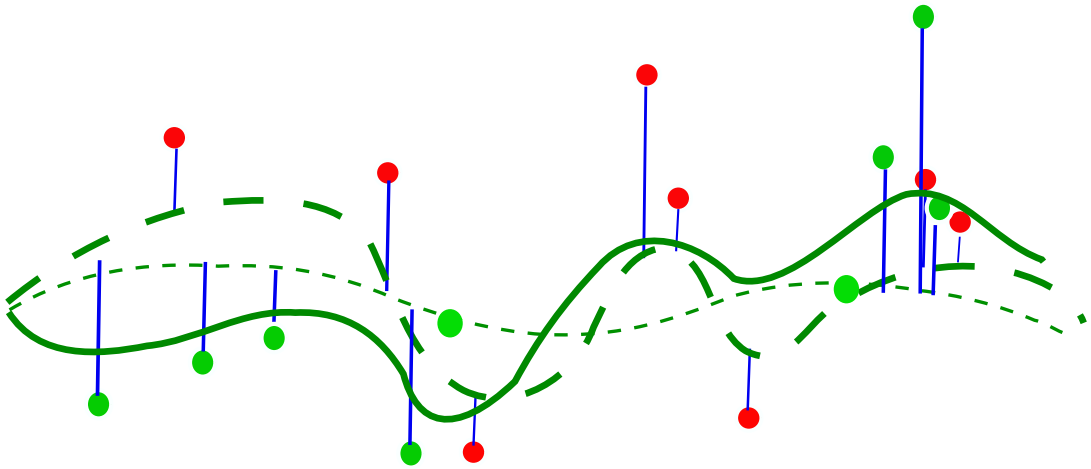
Stochastic gradient descent (SGD)



- Batch gradient descent is expensive when the batch is large
- We can decrease the cost by using several mini-batches (extra gain with parallel computations)
- In SGD, each mini-batch gives a different (inaccurate) estimate of the local gradient
- Combining several inaccurate estimates can help being accurate
- Actually, being inaccurate is no big deal (see Class 7)
- Note that SGD converges more slowly. Use it only for large batches.

Bottou, L. (2012) Stochastic gradient descent tricks. *Neural networks: Tricks of the trade*, pp. 421–436. Springer

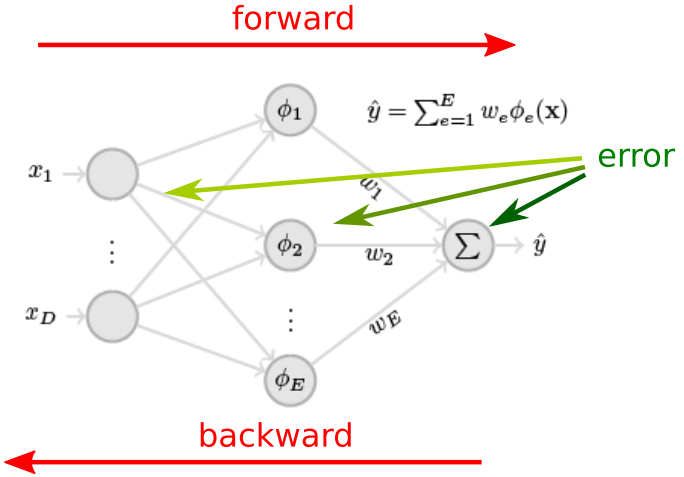
Stochastic gradient descent (SGD)

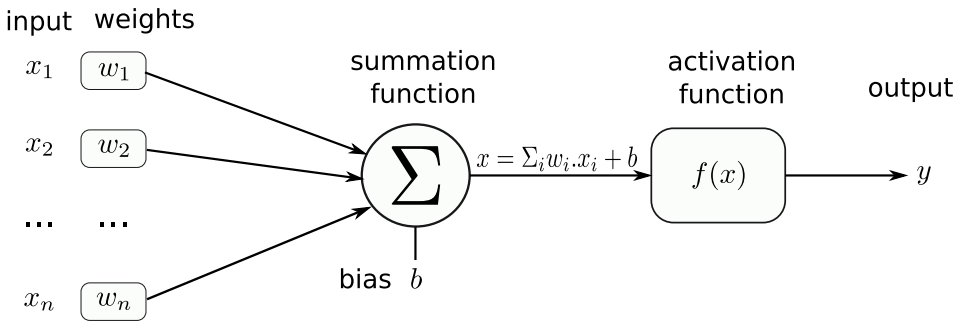


- Batch gradient descent is expensive when the batch is large
- We can decrease the cost by using several mini-batches (extra gain with parallel computations)
- In SGD, each mini-batch gives a different (inaccurate) estimate of the local gradient
- Combining several inaccurate estimates can help being accurate
- Actually, being inaccurate is no big deal (see Class 7)
- Note that SGD converges more slowly. Use it only for large batches.

Bottou, L. (2012) Stochastic gradient descent tricks. *Neural networks: Tricks of the trade*, pp. 421–436. Springer

Gradient descent over multilayer neural networks





- The parameters $\boldsymbol{\theta}$ are the weights w_{ij} and biases b_i
- In $\nabla_{\boldsymbol{\theta}}(\mathbf{L}(\boldsymbol{\theta}))$, the errors at a layer depends on the errors at the next layers
- Gradient computation cannot be performed in a single step
- Compute the gradient with the gradient backpropagation algorithm (backprop)
- It is technical an a little tedious to code for each specific network structure
- TensorFlow, pytorch and their ancestors (theano, caffe...) are built to provide gradient backpropagation for any tensor structure

Nielsen, M. A. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, 2015

http://neuralnetworksanddeeplearning.com/chap2.html

Creating an neural network in pytorch

```
class NeuralNetwork(nn.Module):

  def __init__(self, l1, l2, l3, l4, out, learning_rate):
    super(NeuralNetwork, self).__init__()
    self.relu = nn.ReLU()
    self.sigmoid = nn.Sigmoid()
    self.fc1 = nn.Linear(l1, l2)
    self.fc2 = nn.Linear(l2, l3)
    self.fc3 = nn.Linear(l3, l4)
    self.fc4 = nn.Linear(l4, out)
    self.optimizer = th.optim.Adam(self.parameters(), lr=learning_rate)

  def f(self, x):
      input = th.from_numpy(x).float()
      hidden1 = self.sigmoid(self.fc1(input))
      hidden2 = self.sigmoid(self.fc2(hidden1))
      hidden3 = self.sigmoid(self.fc3(hidden2))
      output = rescale(self.sigmoid(self.fc4(hidden3)))
      return output
```

- Adam does better than SGD
- f() is often called forward()
- Other functions not shown

Gradient descent in pytorch

- Computing the loss over a batch

```
for i in range(max_iter):
  output = model.f(xt)
  loss = func.mse_loss(output, yt)
  model.update(loss)
```

- Applying gradient descent

```
def update(self, loss) -> None:
    """
    Apply a loss to a network using gradient backpropagation
    :param loss: the applied loss
    :return: nothing
    """
    self.optimizer.zero_grad()
    loss.sum().backward()
    self.optimizer.step()
```

- The backprop in one line!

Any question?

Send mail to: Olivier.Sigaud@upmc.fr

Bottou, L.

Stochastic gradient descent tricks.
In *Neural networks: Tricks of the trade*, pp. 421–436. Springer, 2012.

Nielsen, M. A.

Neural networks and deep learning, volume 25.
Determination press San Francisco, CA, 2015.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P.

Trust region policy optimization.
In Bach, F. R. and Blei, D. M. (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 1889–1897. JMLR.org, 2015.
URL `http://proceedings.mlr.press/v37/schulman15.html`.