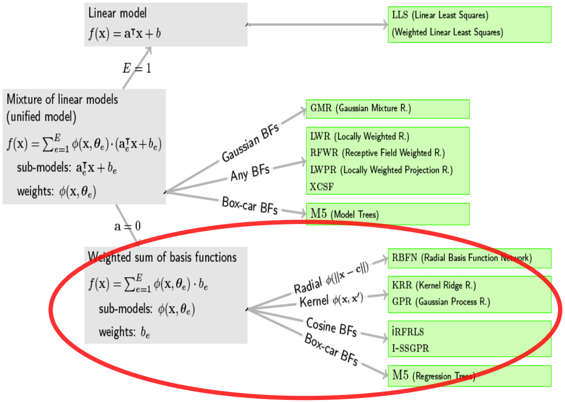
Regression

4. Batch non-linear projection methods

Olivier Sigaud

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

Reminder: Outline of methods

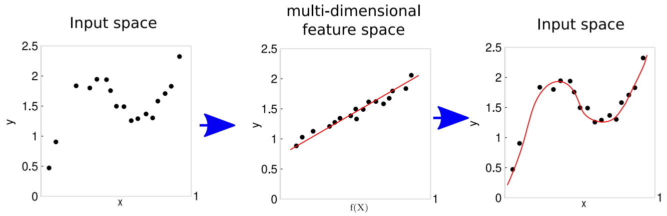


Linear model
$f(x) = a^\intercal x + b$
LLS (Linear Least Squares)
(Weighted Linear Least Squares)
$E = 1$
Mixture of linear models (unified model)
$f(x) = \sum_{e=1}^{E} \phi(x, \theta_e) \cdot (a_e^\intercal x + b_e)$
sub-models: $a_e^\intercal x + b_e$
weights: $\phi(x, \theta_e)$
Gaussian BFs
Any BFs
Box-car BFs
GMR (Gaussian Mixture R.)
LWR (Locally Weighted R.)
RFWR (Receptive Field Weighted R.)
LWPR (Locally Weighted Projection R.)
XCSF
M5 (Model Trees)
$a = 0$
Weighted sum of basis functions
$f(x) = \sum_{e=1}^{E} \phi(x, \theta_e) \cdot b_e$
sub-models: $\phi(x, \theta_e)$
weights: b_e
Radial $\phi(\|x - c\|)$
Kernel $\phi(x, x')$
Cosine BFs
Box-car BFs
RBFN (Radial Basis Function Network)
KRR (Kernel Ridge R.)
GPR (Gaussian Process R.)
iRFRLS
I-SSGPR
M5 (Regression Trees)

- Projecting the input space into a feature space using non-linear basis functions (shown with RBFNs)

Stulp, F. and Sigaud, O. (2015). Many regression algorithms, one unified model: A review. *Neural Networks*, 69:60–79.

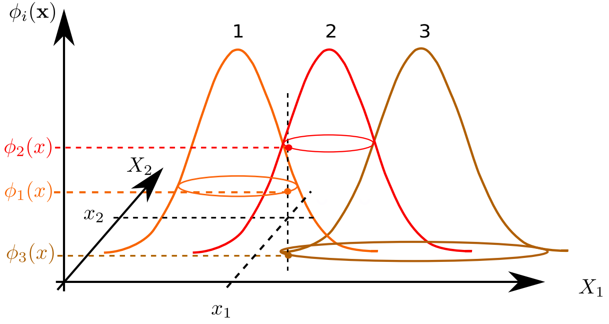
Basis Function Networks: general idea



- With linear regression, we look for $\hat{f}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$
- This is not general enough for non-linear functions
- More general form: $\hat{f}(\mathbf{x}) = \sum_{e=0}^{E} w_e \cdot \phi_{\boldsymbol{\theta}_e}(\mathbf{x})$ with $\phi_{\boldsymbol{\theta}_0}(\mathbf{x}) = 1$
- This can be seen as projecting the input to a different space...
- ... where the latent function is linear

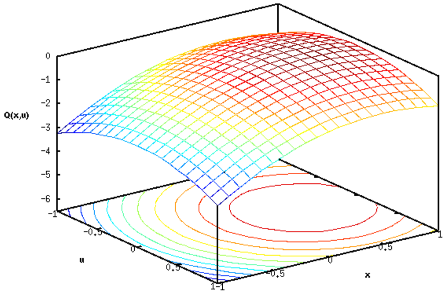
Bishop, C. M. (2007) *Pattern recognition and machine learning.* Springer Berlin/Heidelberg, Germany

Understanding projection



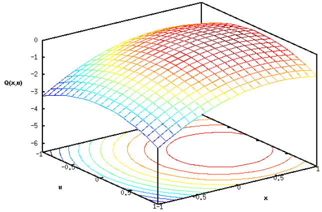
- The point $x = (x_1, x_2)$ is projected to $(\phi_1(x), \phi_2(x), \phi_3(x))$

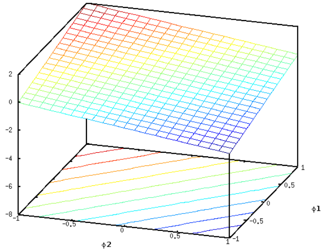
Regression with features: example



- The (unkown) function to be approximated is $f(x_1, x_2) = |x_1 - 2|^2 + 3.|x_2|^2 + 4$
- We define features $\phi_i(x_1, x_2)$ over (x_1, x_2)
- We look for $\boldsymbol{w}$ such that $\hat{f}(x_1, x_2) = \Sigma_i w_i \phi_i(x_1, x_2)$

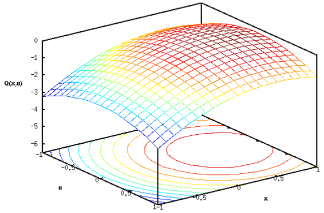
With poor features

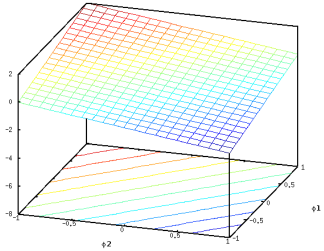




- If we take 3 feature functions $\phi_0(x_1, x_2) = 1$, $\phi_1(x_1, x_2) = x_1$ and $\phi_2(x_1, x_2) = x_2$
- We cannot do better than $\hat{f}(x_1, x_2) = w_1x_1 + w_2x_2 + c$
- Very poor linear approximation

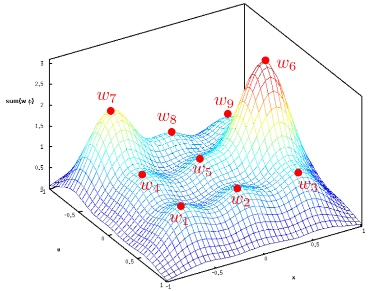
With good features





- If we take $\phi_0(x_1, x_2) = 1$, $\phi_1(x_1, x_2) = |x_1 - 2|^2$ and $\phi_2(x_1, x_2) = |x_2|^2$
- Then $\hat{f}(x_1, x_2) = w_0 + w_1|x_1 - 2|^2 + w_2|x_2|^2$
- If we take $w_0 = 4$, $w_1 = 1$ and $w_2 = 3$, we get exactly $\hat{f}(x_1, x_2) = |x_1 - 2|^2 + 3.|x_2|^2 + 4 = f(x_1, x_2)$
- Perfect approximation
- Finding good features is critical

Standard features: Gaussian basis functions



- The more features, the better the approximation
- ... but the more expensive the computation
- All the following algorithms use this structure
- In particular, we may use one kernel per known data point

Kernel Ridge Regression (KRR) = Kernel Regularised Least Squares (KRGLS)

- Define features with a kernel function $k(\mathbf{x}, \mathbf{x}_i)$ per point $\mathbf{x}_i$
- Define the Gram matrix as a kernel matrix:

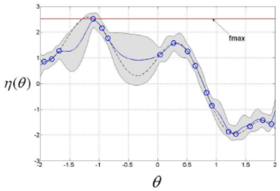
$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \tag{1}$$

- If we had an infinity of data points, the linear approximation in feature space would become perfect
- Intuition: the error is a function of the distance to data points
- Computing the weights is done with RR using

$$\boldsymbol{\theta}^* = (\lambda \mathbf{I} + \mathbf{K})^{-1} \mathbf{y}, \tag{2}$$

- Note that $\mathbf{K}$ is symmetric
- The kernel matrix $\mathbf{K}$ grows with the number of points (kernel expansion)
- The matrix inversion may become too expensive
- Solution: finite set of features (RBFNs), incremental methods

Gaussian Process Regression (GPR)



- Predicting y for a novel input $\mathbf{x}$ is done by assuming that the novel output y is sampled from a multi-variate Gaussian.
- Information for some $\mathbf{x}$ removes uncertainty in its neighborhood using some kernel-related *covariance function* $\mathbf{k}(\mathbf{x}, \mathbf{X})$
- The best estimate for y is the mean $\overline{y} = \mathbf{k}(\mathbf{x}, \mathbf{X})\mathbf{K}^{-1}\mathbf{y}$
- The variance in y is $var(y) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x}, \mathbf{X})\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}, \mathbf{X})^{\intercal}$

Ebden, M. (2008). Gaussian processes for regression: A quick introduction. Technical report, Department of Engineering Science, University of Oxford

GPR $\sim$ KRR

- When computing the mean $\overline{y}$, $\mathbf{K}$ and $\mathbf{y}$ depend only on the training data, not the novel input $\mathbf{x}$.
- Therefore, $\mathbf{K}^{-1}\mathbf{y}$ can be compacted into one weight vector, which does not depend on the query $\mathbf{x}$.
- We call this vector $\boldsymbol{\theta}^*$ and we get $\boldsymbol{\theta}^* = \mathbf{K}^{-1}\mathbf{y}$,
- We can rewrite $\overline{y}$ as follows:

$$\begin{aligned} \overline{y} &= \mathbf{k}(\mathbf{x}, \mathbf{X})\mathbf{K}^{-1}\mathbf{y} \\ &= [k(\mathbf{x}, \mathbf{x}_1) \ldots k(\mathbf{x}, \mathbf{x}_N)] \cdot \boldsymbol{\theta}^* \\ &= \sum_{n=1}^{N} \boldsymbol{\theta}^*_n \cdot k(\mathbf{x}, \mathbf{x}_n). \end{aligned} \tag{3}$$

- The mean of GPR is the same weighted sum of basis functions as in KRR
- KRR computes a regularized version of the weights computed by GPR, with an additional regularization parameter λ.
- See the tutorial paper for details

Stulp, F. and Sigaud, O. (2015). Many regression algorithms, one unified model: A review. *Neural Networks*, 69:60–79.

Radial Basis Function Networks: definition and solution

- Radial Basis Functions versus Kernels (Gaussians $\phi(\mathbf{x}, \boldsymbol{\theta}_e) = e^{-\frac{1}{2}(\mathbf{x}-\mu_e)^T \boldsymbol{\Sigma}_e^{-1}(\mathbf{x}-\mu_e)}$ are both)
- We define a set of E basis functions (often Gaussian)

$$\hat{f}(\mathbf{x}) = \sum_{e=1}^{E} w_e \cdot \phi(\mathbf{x}, \boldsymbol{\theta}_e) \tag{4}$$

$$= \boldsymbol{\theta}^{\intercal} \cdot \phi(\mathbf{x}). \tag{5}$$

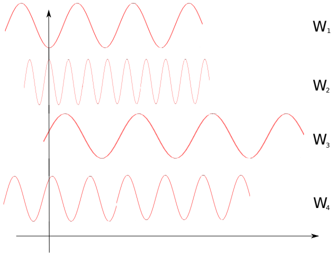
- We also define the *Gram matrix*

$$\mathbf{G} = \begin{bmatrix} \phi(\mathbf{x}_1, \boldsymbol{\theta}_1) & \phi(\mathbf{x}_1, \boldsymbol{\theta}_2) & \cdots & \phi(\mathbf{x}_1, \boldsymbol{\theta}_E) \\ \phi(\mathbf{x}_2, \boldsymbol{\theta}_1) & \phi(\mathbf{x}_2, \boldsymbol{\theta}_2) & \cdots & \phi(\mathbf{x}_2, \boldsymbol{\theta}_E) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_N, \boldsymbol{\theta}_1) & \phi(\mathbf{x}_N, \boldsymbol{\theta}_2) & \cdots & \phi(\mathbf{x}_N, \boldsymbol{\theta}_E) \end{bmatrix} \tag{6}$$

- and we get the least squares solution

$$\boldsymbol{\theta}^* = (\mathbf{G}^{\intercal}\mathbf{G})^{-1}\mathbf{G}^{\intercal}\mathbf{y}. \tag{7}$$

Incremental Receptive Fields Regularized Least Squares



- Approximate the function through its (approximate) Fourier transform using random features $z_k(X_i) = \frac{\sqrt{2}}{\sqrt{D}} cos(\omega_k^T X_i + b_k)$, with $\omega_k \sim \mathcal{N}(0, 2\gamma I)$ and $b_k \sim \mathcal{U}(0, 2\pi)$.
- As RBFNs, but with K cosinus features $\rightarrow$ global versus local
- Provides a strong grip against over-fitting (ignoring the high frequencies)
- In practice, efficient for large enough K, and easy to tune
- I-SSGPR: same tricks based on GPR

Gijsberts, A. & Metta, G. (2011) "Incremental learning of robot dynamics using random features." In *IEEE International Conference on Robotics and Automation* (pp. 951–956).

Least Square Projection Methods: summary of computations

- Linear case

$$\boldsymbol{\theta}^* = (\mathbf{X}^\intercal\mathbf{X})^{-1}\mathbf{X}^\intercal\mathbf{y} \qquad (LS) \qquad (8)$$

$$\boldsymbol{\theta}^* = (\lambda\mathbf{I} + \mathbf{X}^\intercal\mathbf{X})^{-1}\mathbf{X}^\intercal\mathbf{y}. \qquad (RLS) \qquad (9)$$

- Gram matrix case

$$\boldsymbol{\theta}^* = (\mathbf{G}^\intercal\mathbf{G})^{-1}\mathbf{G}^\intercal\mathbf{y} \qquad (RBFN) \qquad (10)$$

- Kernel matrix case

$$\boldsymbol{\theta}^* = \mathbf{K}^{-1}\mathbf{y}, \qquad (GPR) \qquad (11)$$

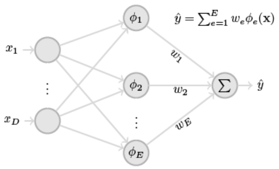
$$\boldsymbol{\theta}^* = (\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{y}. \qquad (KRR) \qquad (12)$$

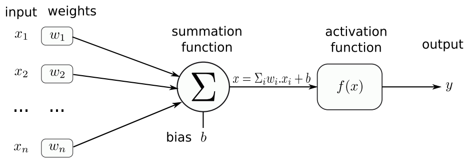
Least Square Projection Methods: summary of approaches

Algorithm	Regularized?	Number of BFs?	Features?
RBFN	No	E	RBFs
KRR	Yes	N	kernels
GPR	No	N	kernels
iRFRLS	Yes	E	cosine
I-SSGPR	Yes	E	cosine

Table: Design of all weigthed basis function algorithms.

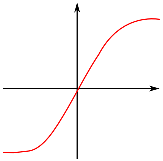
The case of (feedforward) neural networks

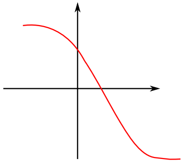


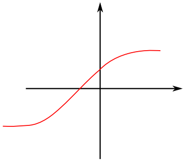


- The activation function is non local (sigmoid, ReLu, LeakyReLu...) vs Gaussians
- Weights of output layer: regression
- Weight of intermediate layer(s): tuning basis functions
- Shares the same structure as all basis function networks
- Sigmoids instead of Gaussians: better split of space in high dimensions

Regression with neural networks: discovering features

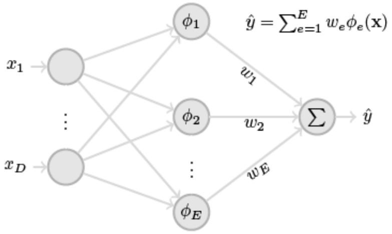

$w = 1, b = 0$


$w = -1, b = 1$


$w = 1/2, b = -1$

- The backprop algo tunes both kinds of weights
- Discovers the adequate features by itself
- Deep versus shallow: get more tunable features with less parameters
- Cannot be performed batch, see incremental methods (Classes 5, 6 and 7)

Regression with neural networks: variants



- If only the weights at the last layer are tuned, still defines a linear architecture (Extreme Learning Machine)
- Stochastic optimization of intermediate weights, linear regression on output weights?

Huang, G.-B., Zhou, H., Ding, X., & Zhang, R. (2012) Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529

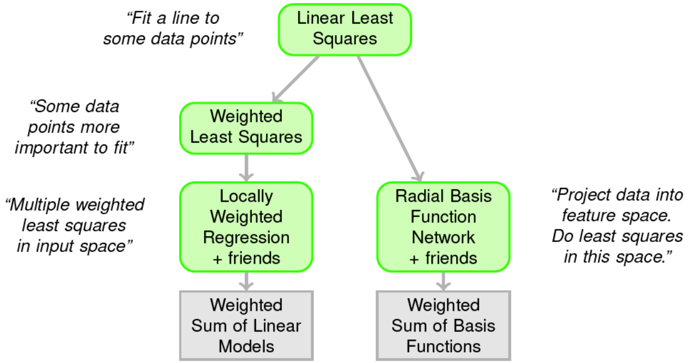
LWR versus RBFNs

$$\hat{f}(\mathbf{x}) = \sum_{e=1}^{E} \phi(\mathbf{x}, \boldsymbol{\theta}_e) \cdot (b_e + \mathbf{a}_e^\intercal \mathbf{x}) \quad (13)$$

$$\hat{f}(\mathbf{x}) = \sum_{e=1}^{E} \phi(\mathbf{x}, \boldsymbol{\theta}_e) \cdot w_e, \quad (14)$$

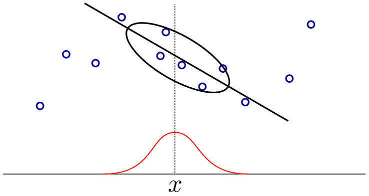
- Eq. (14) is a special case of (13) with $\mathbf{a}_e = \mathbf{0}$ and $b_e = w_e$.
- RBFNs: performs one LS computation in a projected space
- LWR: performs many LS computation in local domains

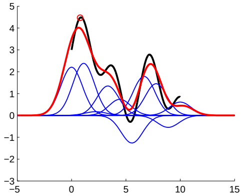
Wrap-up


"Fit a line to some data points"
Linear Least Squares
"Some data points more important to fit"
Weighted Least Squares
"Multiple weighted least squares in input space"
Locally Weighted Regression + friends
Radial Basis Function Network + friends
"Project data into feature space. Do least squares in this space."
Weighted Sum of Linear Models
Weighted Sum of Basis Functions

- Image taken from Freek Stulp's IROS 2018 Tutorial

Take home messages for robot model learning

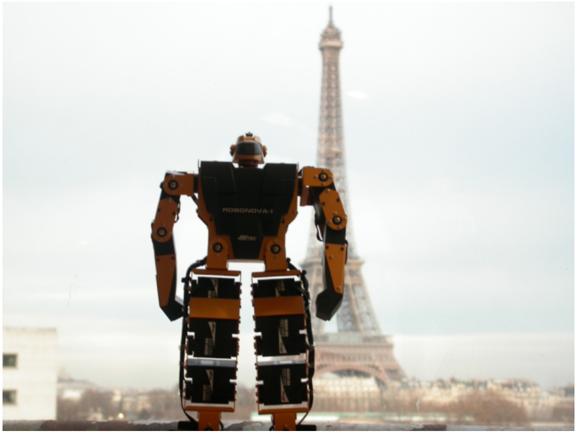




- Mixture of linear models vs Basis Function Networks
- Neural networks: tuning the features
- ISSGPR: easy tuning, no over-fitting
- LWPR: PLS, fast implementation, the reference method
- XCSF: distinguish Gaussian weights space and linear models space
- GMR: few features, the richest representation

Sigaud, O. , Salaün, C. and Padois, V. (2011) "On-line regression algorithms for learning mechanical models of robots: a survey," *Robotics and Autonomous Systems*, 59:1115-1129

Any question?



Send mail to: `Olivier.Sigaud@upmc.fr`

Bishop, C. M.
Pattern recognition and machine learning.
Springer Berlin/Heidelberg, Germany, 2007.

Ebden, M.
Gaussian processes for regression: A quick introduction.
Technical report, Department on Engineering Science, University of Oxford, 2008.

Gijsberts, A. and Metta, G.
Incremental learning of robot dynamics using random features.
In *IEEE International Conference on Robotics and Automation*, pp. 951–956, 2011.

Huang, G.-B., Zhou, H., Ding, X., and Zhang, R.
Extreme learning machine for regression and multiclass classification.
IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 42(2):513–529, 2012.

Sigaud, O., Salaün, C., and Padois, V.
On-line regression algorithms for learning mechanical models of robots: a survey.
Robotics and Autonomous Systems, 59(12):1115–1129, December 2011.

Stulp, F. and Sigaud, O.
Many regression algorithms, one unified model: A review.
Neural Networks, 69:60–79, 2015.