Machine Learning 101b

Jan Peters
Gerhard Neumann
Michael Lutter

Purpose of this Lecture

- How can we define such features for general machine learning problems?
- Can we avoid or automate the feature specification?
- Familiarize you with non-parametric models

Content of this Lecture

Constructing Basis Functions

- Radial Basis Function Networks
- Deep Learning

Non-Parametric Approaches

- Locally Weighted Regression
- Kernel Methods

What we did so far…

- Models that are linear in the parameters: $y = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\theta}$
- **Parameter Estimation in Regression**
    - **Least Squares** ~ Maximum Likelihood estimation (ML; Frequentist)
    - **Ridge Regression** ~ Maximum a Posteriori estimation (MAP; Bayesian)
- **Full Bayesian Regression** integrates out the parameters when predicting
    - State dependent uncertainty

However, for most problems good features are not easy to find

What to do when you don’t know the features?

In most real applications, we know **some** good features.

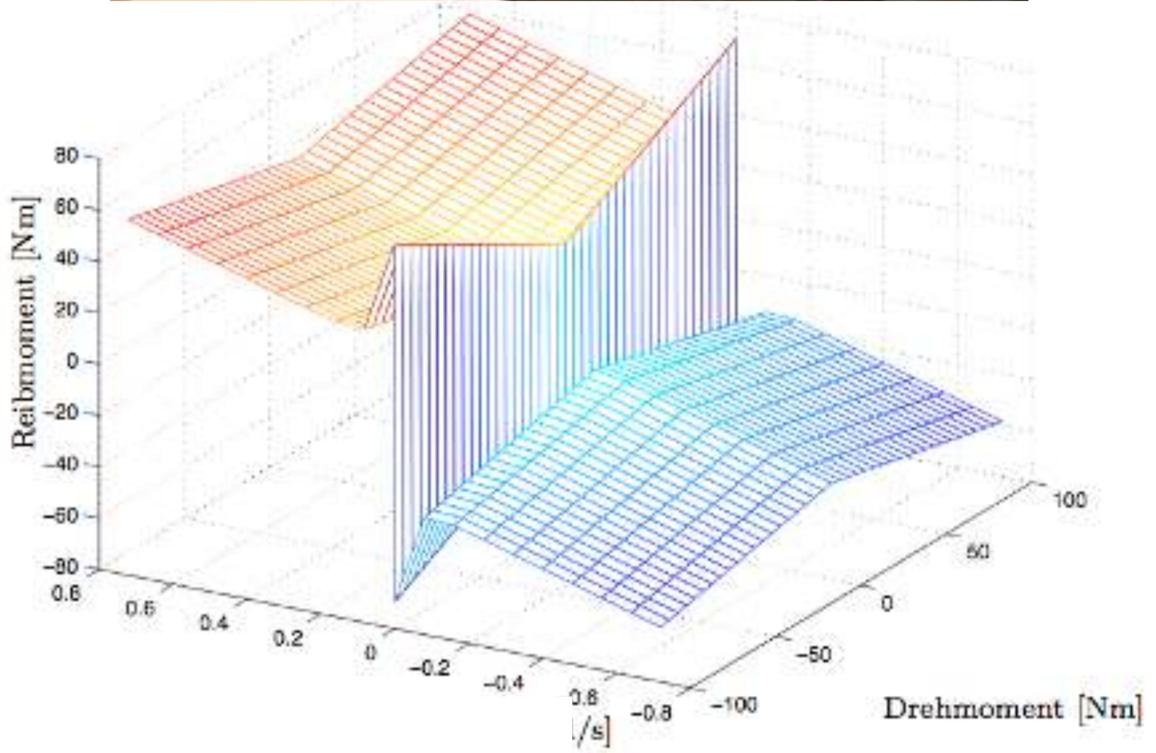
However, we almost certainly **don’t know all** features we need.

Example: Rigid body dynamics

- Friction has no good features
- Unknown dynamics causes huge problems (requires more state variables).

There may also be way too many features!

Hand-crafted features are almost never enough...



Can we avoid having to find good features?

Yes, we can!

We need to find machine learning approaches that **generate the features automatically** from data.

- **Type 1**: *Automatic Basis Function Construction* constructs basis functions from data.
- **Type 2**: *Non-Parametric Regression* look at data locally and interpolate with similar data.
- **Type 3**: *Kernel Regression* finds the features implicitly by going into *function space* using a *kernel*

Type 1: Construct Basis Functions from Data

Classical idea behind “neural networks”

- Multi-Layer Perceptrons (see Machine Learning: Statistical Approaches)

- Radial Basis Function Networks

Radial Basis Function Network

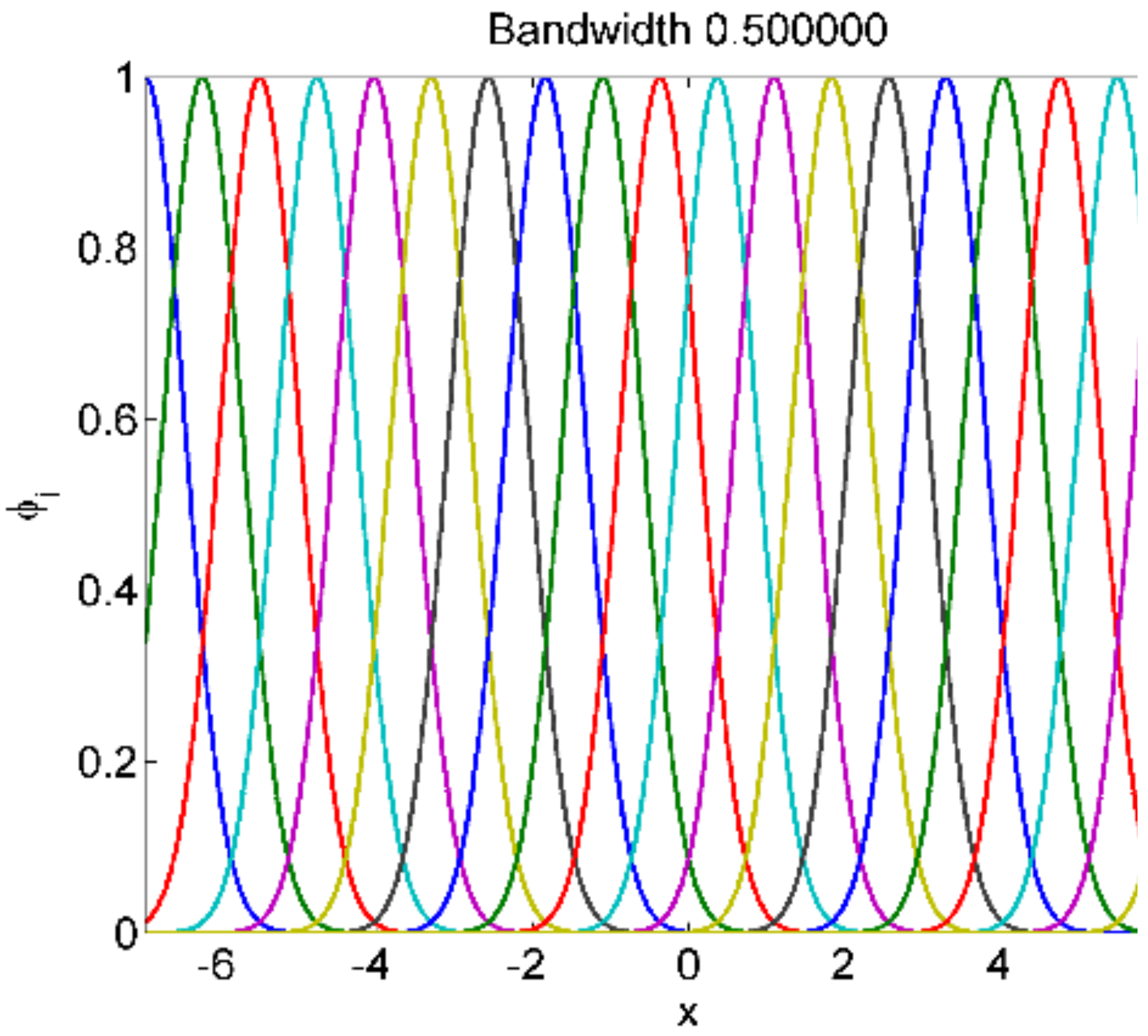
Assume a smoothness prior and obtain the cost function

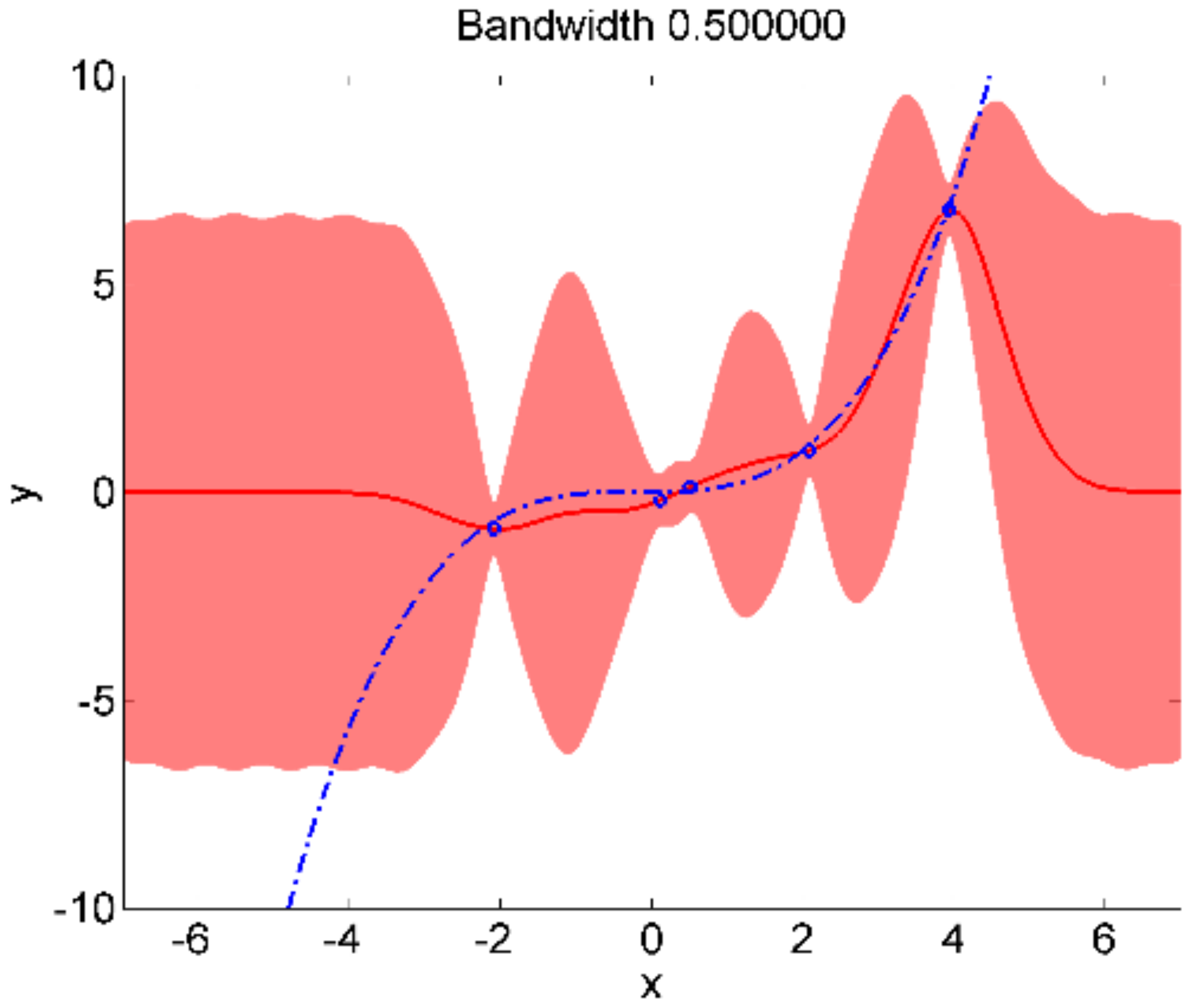
$$J = \frac{1}{2}\sum_{i=1}^{N}\left[(y_i - \mathbf{f}_\theta(\mathbf{x}_i))^2 + \left\|\frac{d^2}{dx^2}\mathbf{f}_\theta(\mathbf{x}_i)\right\|^2\right]$$

This prior yields radial basis functions as features:

$$\begin{aligned} f_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \sum_i \theta_i \exp\left(\frac{-||\boldsymbol{x} - \boldsymbol{\mu}_i||^2}{2l^2}\right) \\ &= \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\theta}, \quad \text{with } \phi_i(\boldsymbol{x}) = \exp\left(\frac{-||\boldsymbol{x} - \boldsymbol{\mu}_i||^2}{2l^2}\right) \end{aligned}$$

Example: Radial Basis Function Features





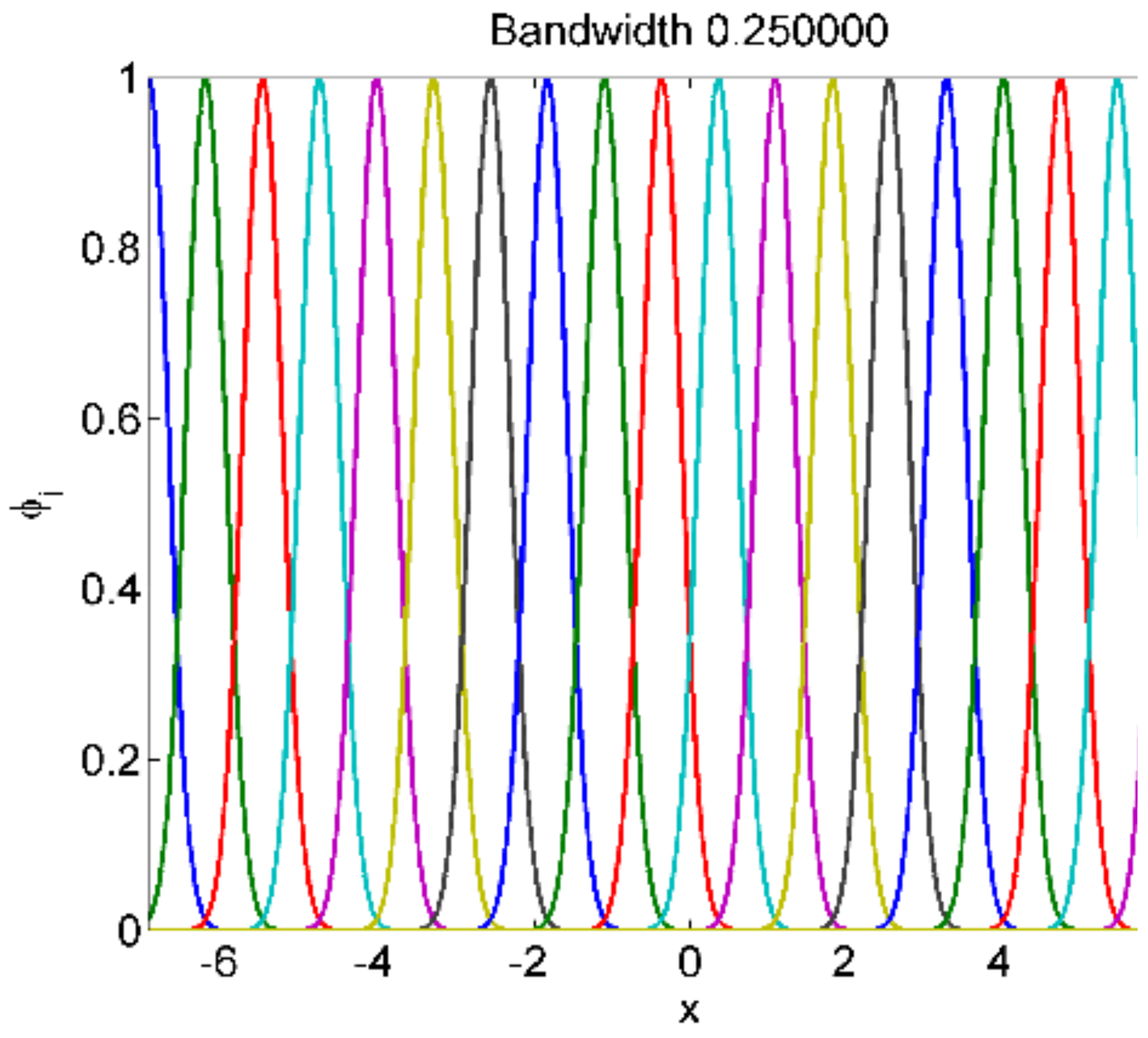
Radial Basis Functions Hyperparameters

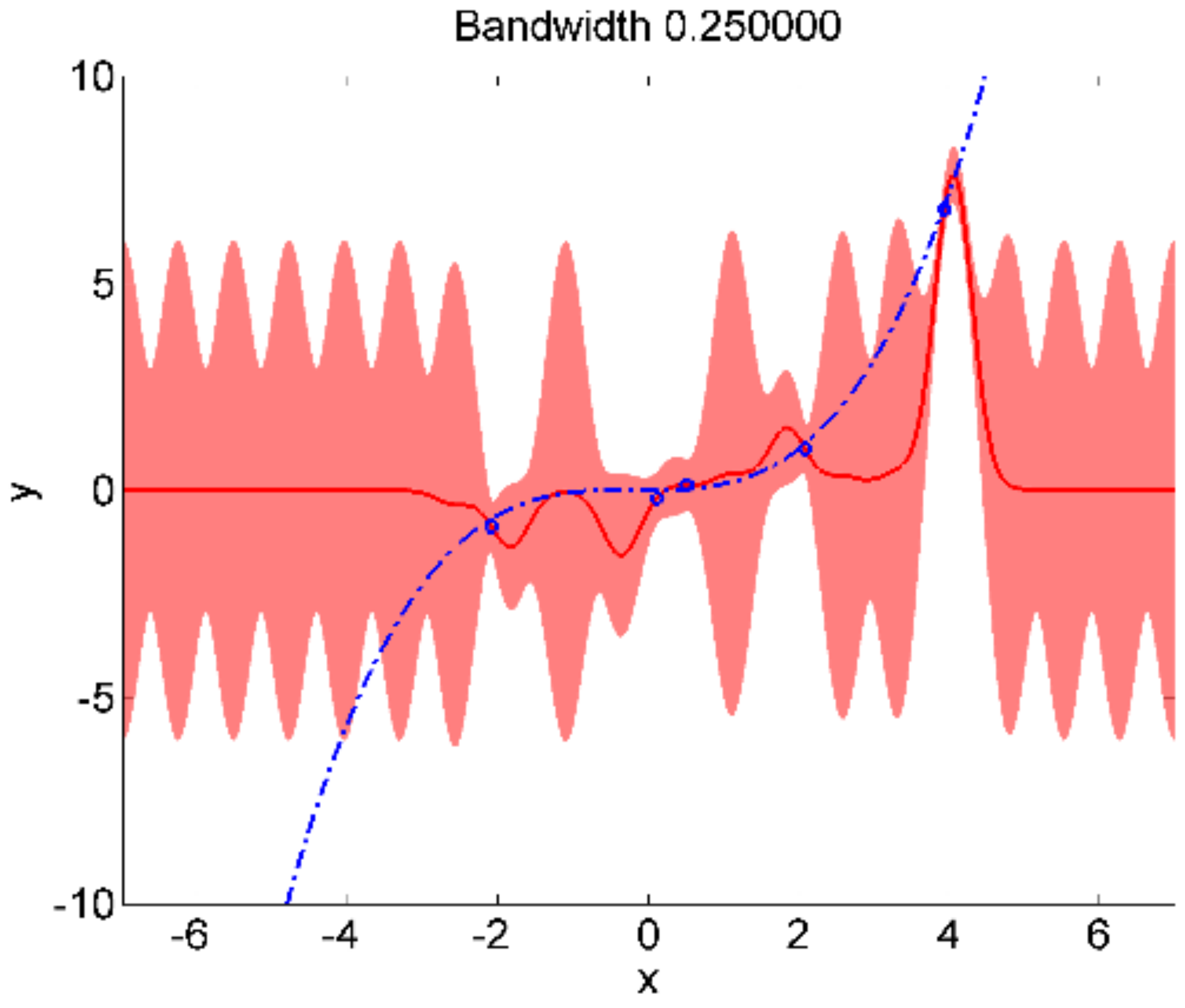
Let's look again at

$$\phi_i(\boldsymbol{x}) = \exp\left(\frac{-||\boldsymbol{x} - \boldsymbol{\mu}_i||^2}{2l^2}\right)$$

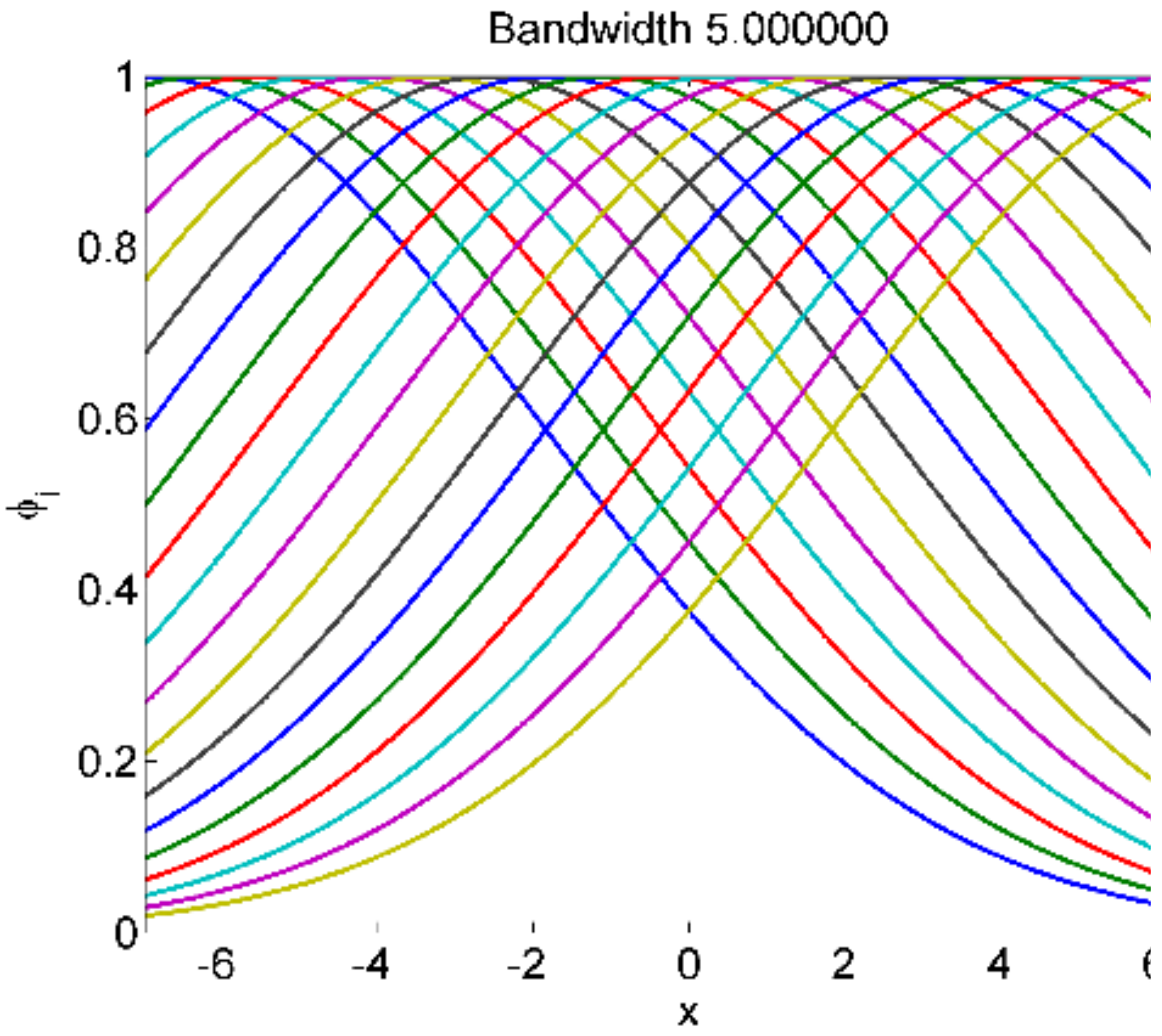
- How do I find the width *l* of the basis functions or the centers $\boldsymbol{\mu}_i$?
- Linear regression? Nope: not linear in *l or mu* !
- We need to optimize this width on the training set
- We can do that by gradient descent: Write down a loss function, take the derivative w.r.t. *l,* and use an algorithm for non-convex optimization

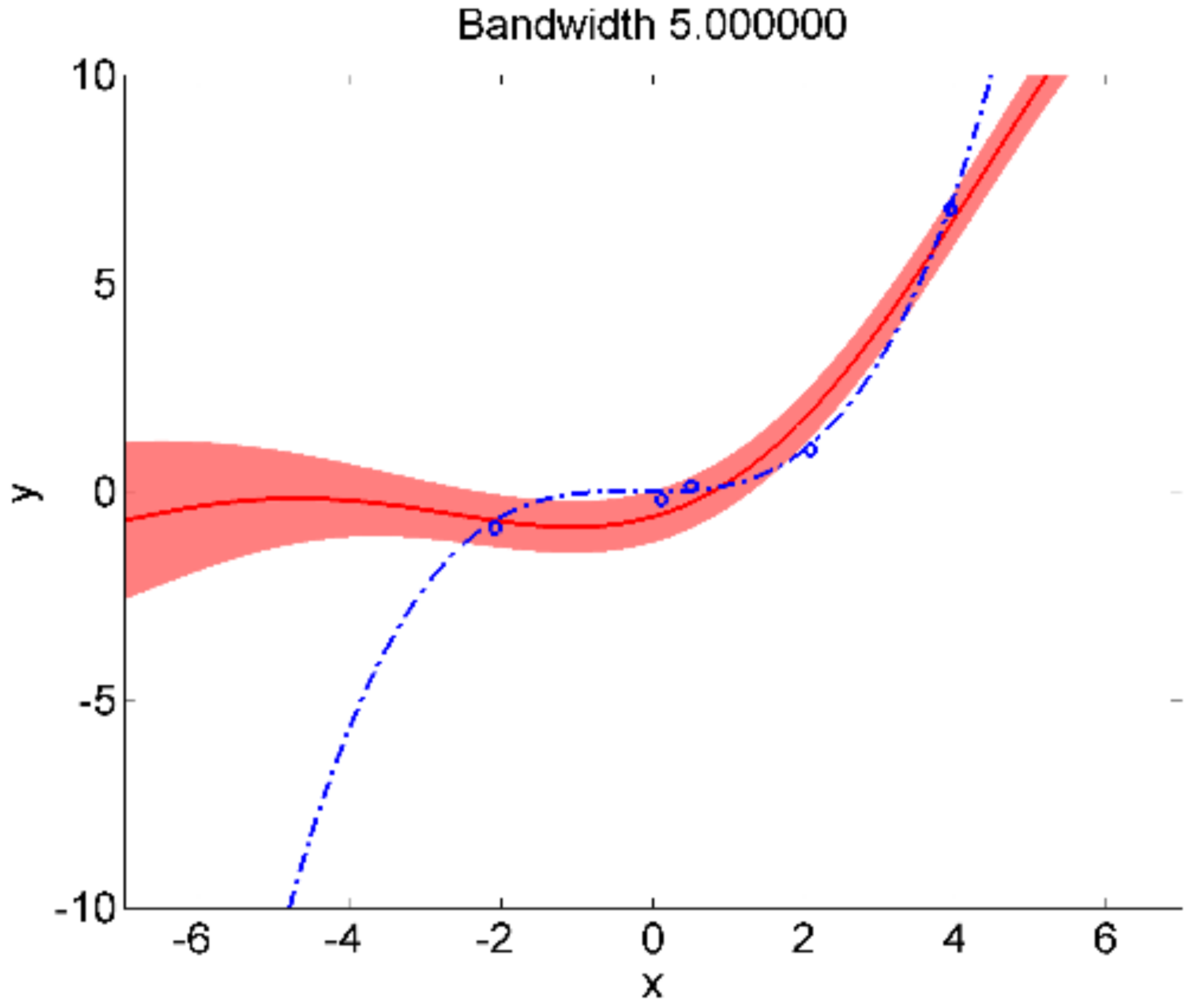
Example: Bandwidth too small





Example: Bandwidth too large





Content of this Lecture

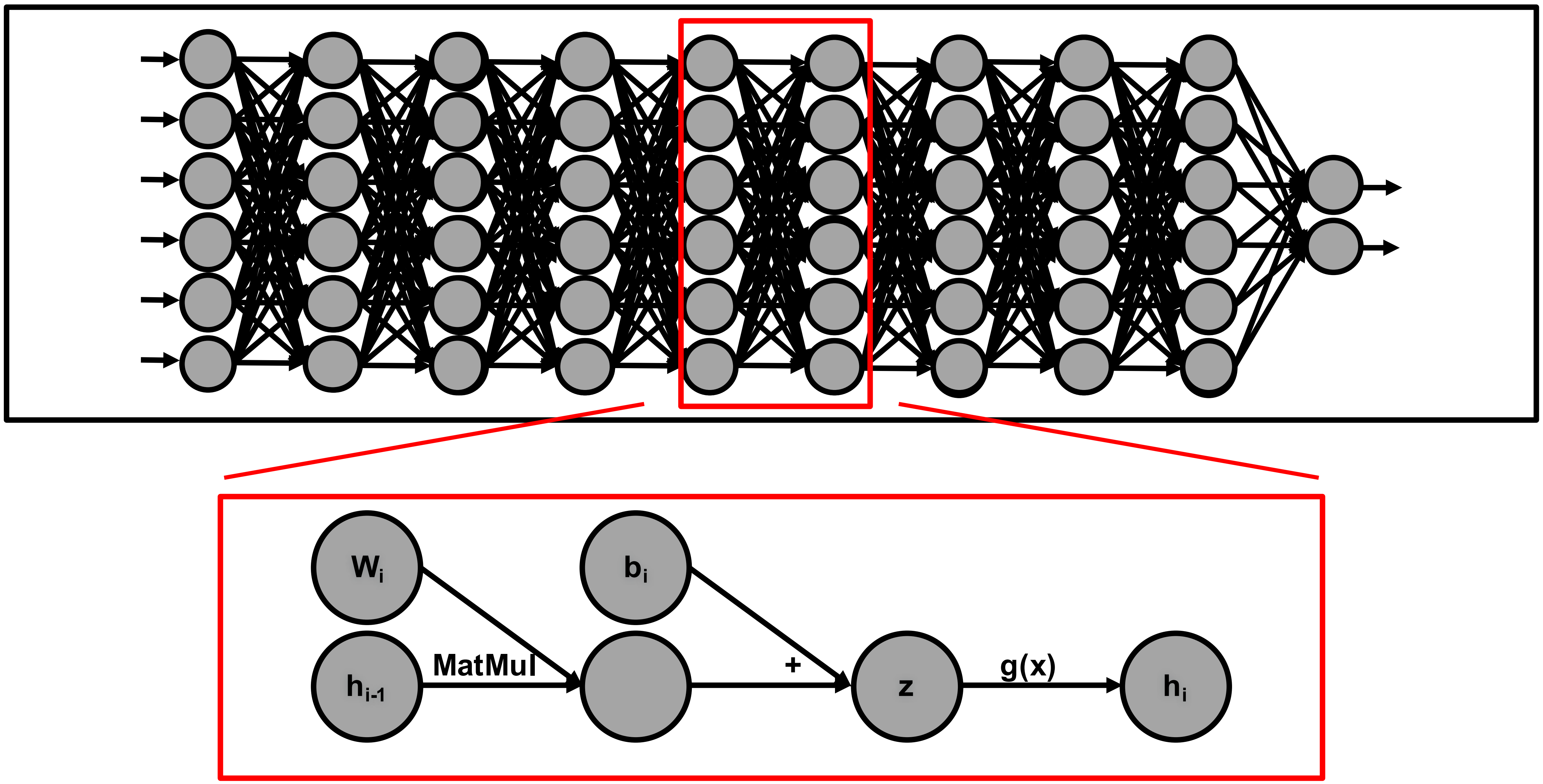
Constructing Basis Functions

- Radial Basis Function Networks
- Deep Learning

Non-Parametric Approaches

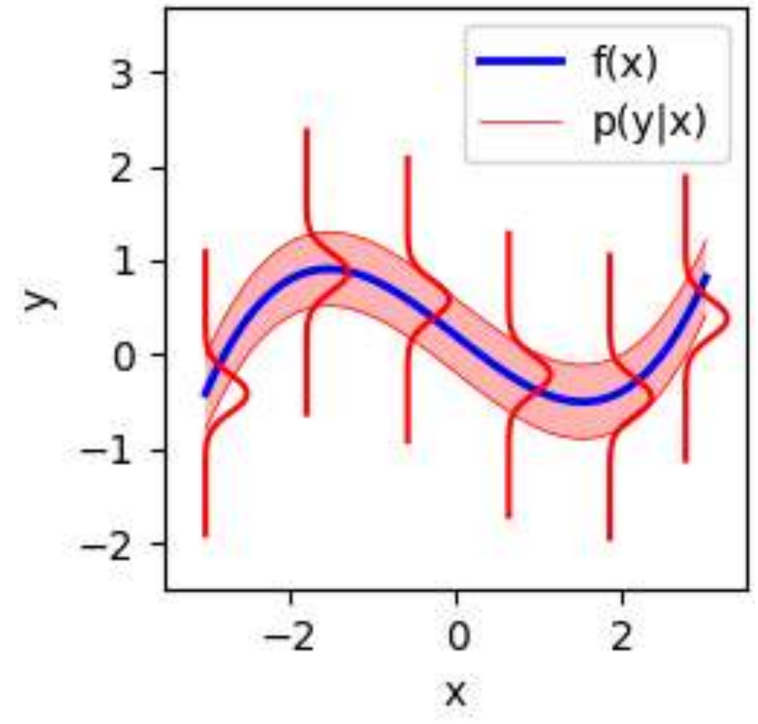
- Locally Weighted Regression
- Kernel Methods

Computational Graphs



Output Neuron Types

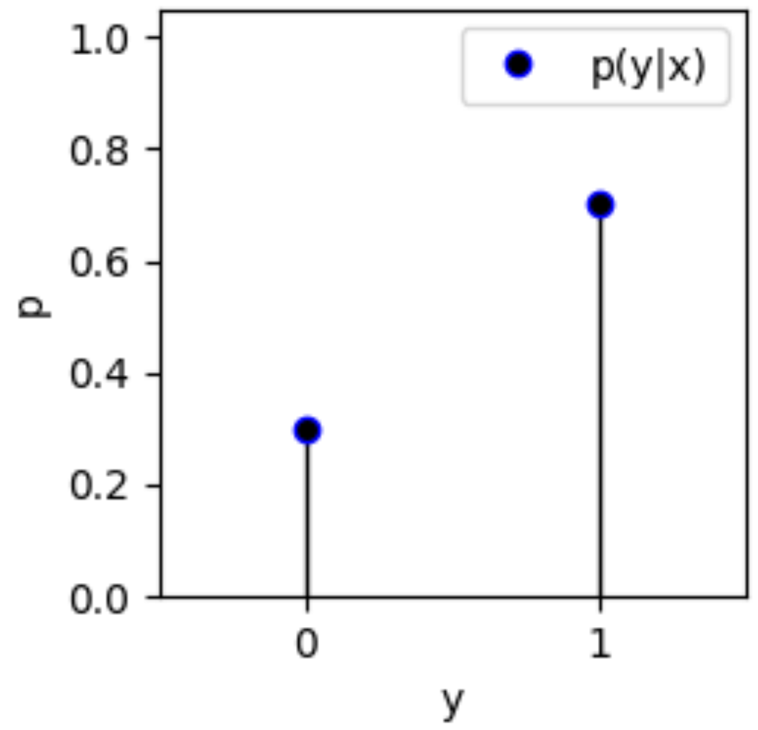
Linear Neuron



$g(\boldsymbol{z}_i) = \boldsymbol{z}_i$

$p(\boldsymbol{y} \mid \boldsymbol{z}) = N(\boldsymbol{y} - \boldsymbol{z}, \boldsymbol{I})$

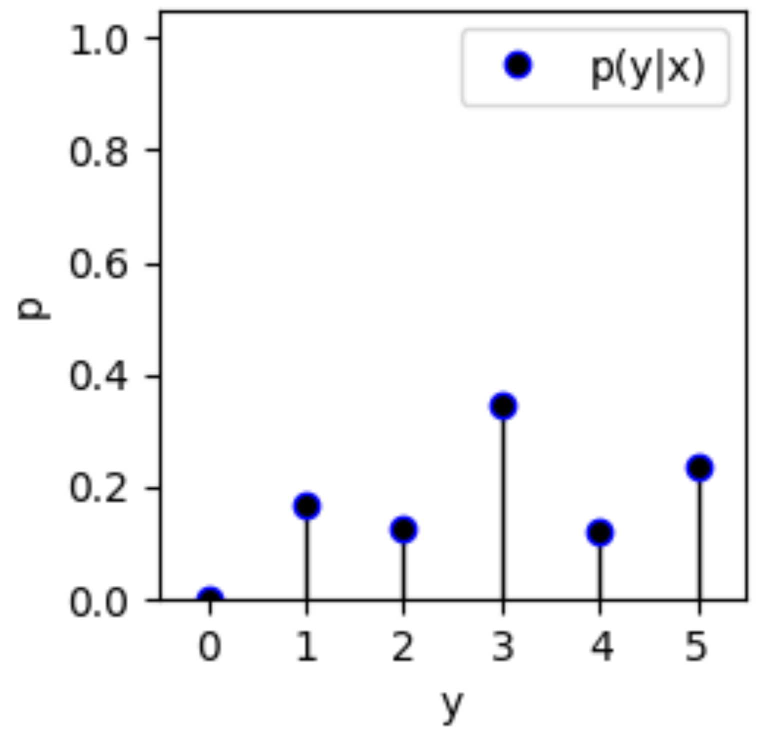
Sigmoid Neuron



$g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$

$p(y \mid z) = \sigma((2y - 1)z)$

Softmax Neuron

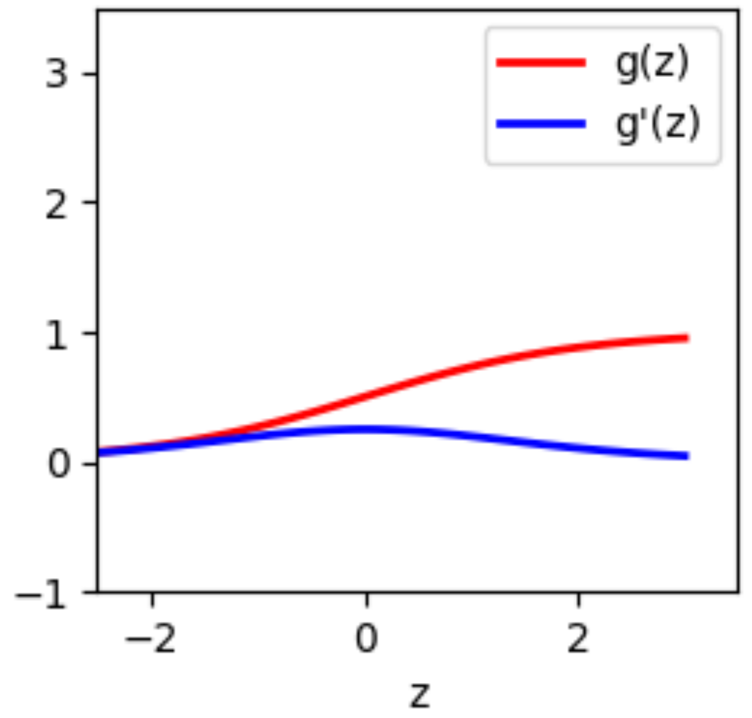


$g(\boldsymbol{z}_i) = \frac{\exp \boldsymbol{z}_i}{\sum_j \exp z_j}$

$p(y = i \mid \boldsymbol{z}) = g(\boldsymbol{z}_i)$

Hidden Neuron Types

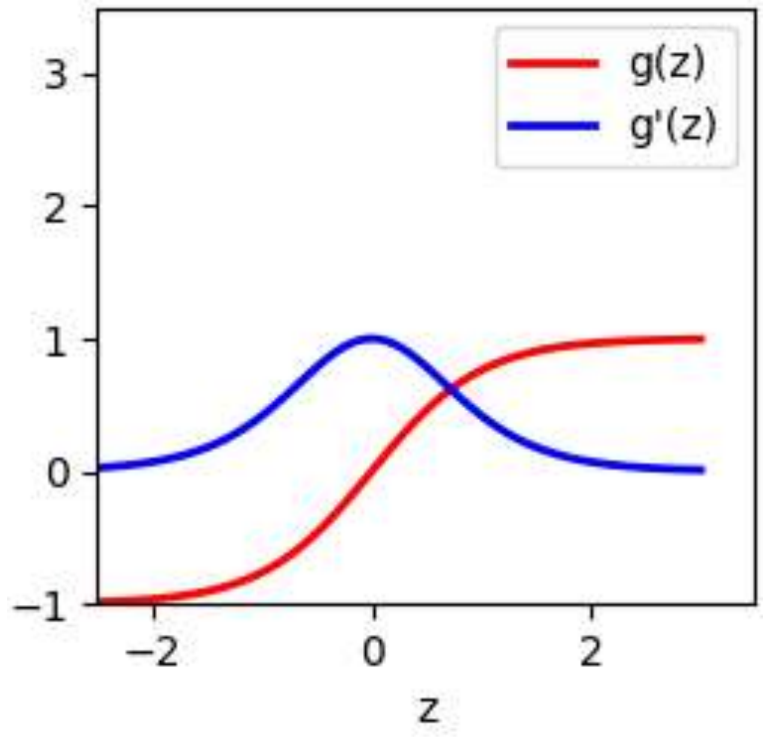
Sigmoid Neuron



$g(\boldsymbol{z}_i) = \sigma(\boldsymbol{z}_i) = \frac{1}{1+e^{-z_i}}$

$g'(\boldsymbol{z}_i) = \sigma(\boldsymbol{z}_i)\left(1 - \sigma(\boldsymbol{z}_i)\right)$

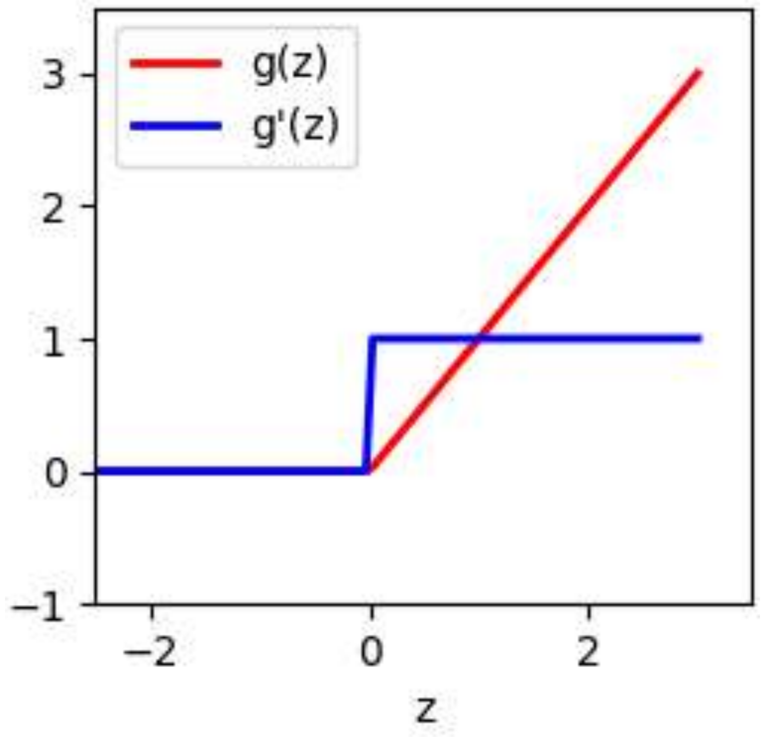
Tanh Neuron



$g(\boldsymbol{z}_i) = \tanh(\boldsymbol{z}_i)$

$g'(\boldsymbol{z}_i) = 1 - \tanh(\boldsymbol{z}_i)^2$

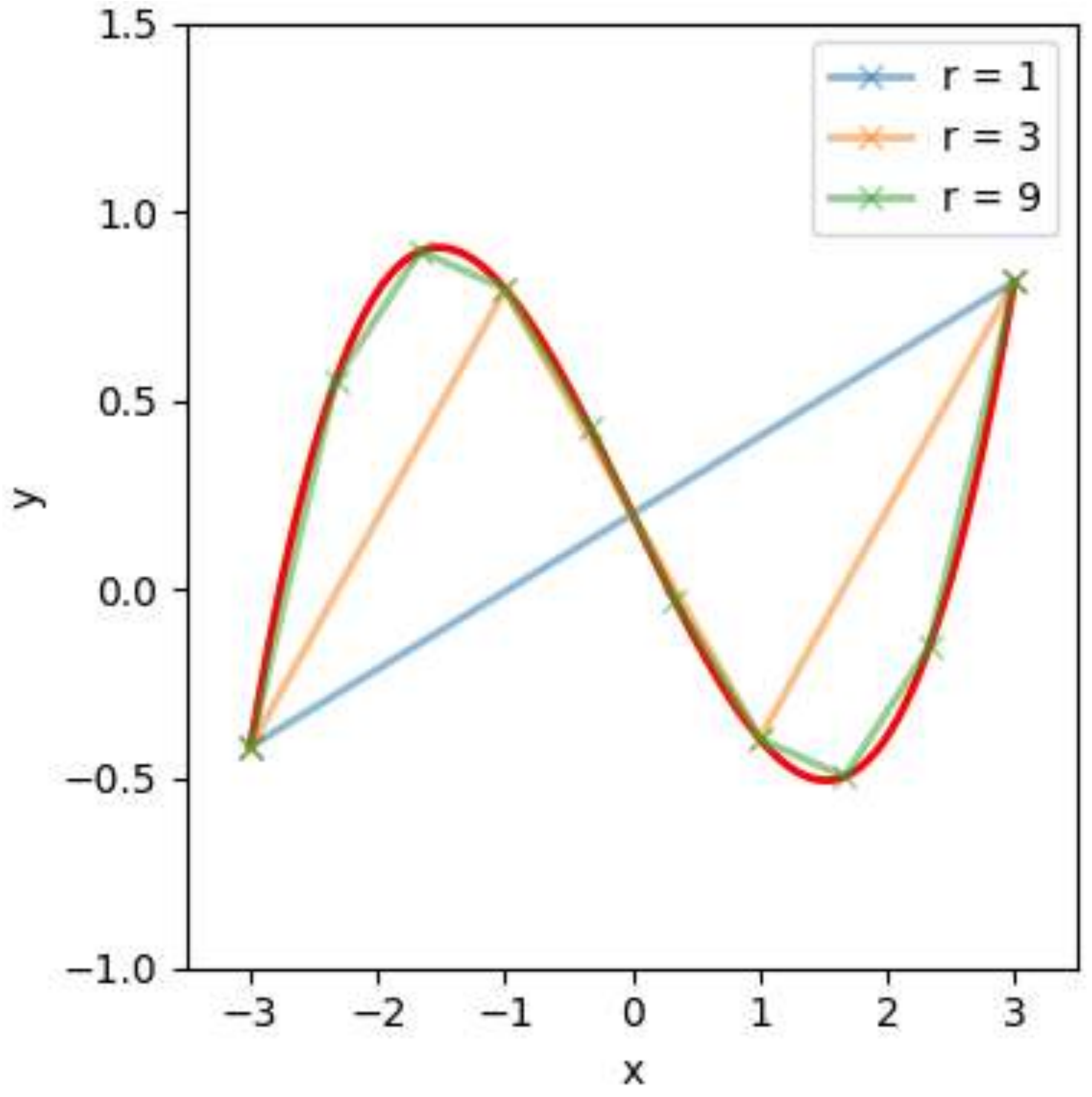
ReLu Neuron



$g(\boldsymbol{z}_i) = \max(\boldsymbol{0}, \boldsymbol{z}_i)$

$g'(\boldsymbol{z}_i) = \begin{cases} 1, & \boldsymbol{z}_i \geq 0 \\ 0, & \boldsymbol{z}_i < 0 \end{cases}$

Universal Approximation Theorem



$$O\left(\binom{n}{d}^{d(l-1)} n^d\right)$$

n = Number of Neurons per Laye
l = Number of Hidden Layers
d = Number of Inputs

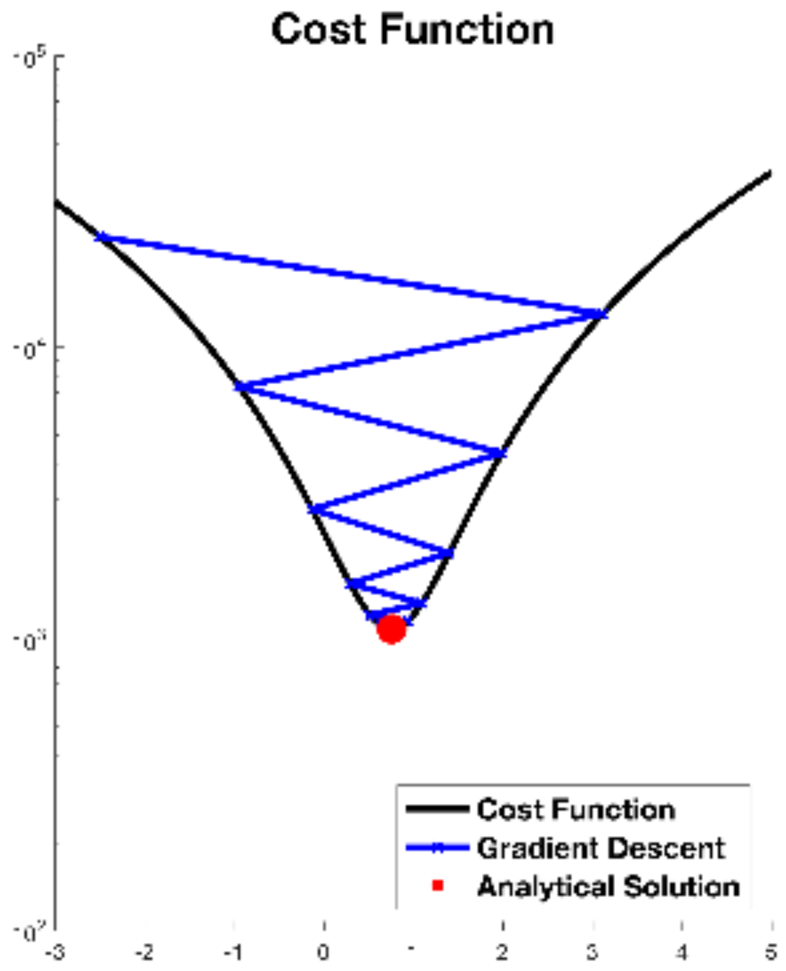
$$O\left(\binom{n}{1}^{1(1-1)} n^1\right) = O(n) \qquad l = 1,\ d = 1$$

$$O\left(\binom{n}{1}^{1(2-1)} n^1\right) = O(n^2) \qquad l = 2,\ d = 1$$

$$O\left(\binom{n}{1}^{1(k-1)} n^1\right) = O(n^k) \qquad l = k,\ d = 1$$

Kurt Hornik et. al., “Multilayer feedforward networks are universal approximators”, 1989
Guido Montufar et.al., “On the Number of Linear Regions of Deep Neural Networks”, 2014

Gradient Descent



Optimization Objective:

$$\theta^* = \operatorname{argmin} J(\theta)$$
$$\theta_{i+1} = \theta_i^{\theta} + \Delta\theta_i = \theta_i - \alpha \, \nabla_{\theta_i} J(\theta)$$

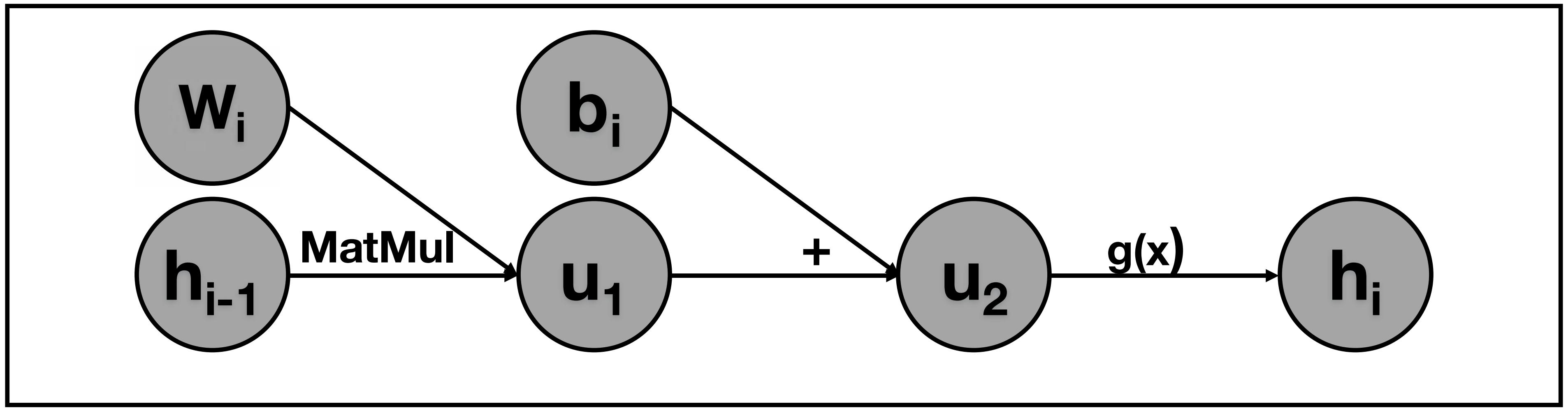
Cost Functions:

$J(\theta) = \underset{p_d}{E}\{|y - f(x,\theta)|_1\}$ → Median of $p(y \mid z)$

$J(\theta) = \underset{p_d}{E}\{|y - f(x,\theta)|_2\}$ → Mean of $p(y \mid z)$

$$J(\theta) = \underset{p_d}{E}\{-\log(p_m(y \mid x,\theta))\}$$

Backpropagation



$\boldsymbol{u}_0 = \boldsymbol{h}_{i-1}$

$\frac{d}{d\boldsymbol{u}_0}\boldsymbol{u}_1 = \boldsymbol{W}_i^T$

$\frac{d}{dW_i}\boldsymbol{u}_1 = [\boldsymbol{u}_0 \quad \dots \quad \boldsymbol{u}_0]$

$\boldsymbol{u}_1 = \boldsymbol{W}_i^T\boldsymbol{u}_0$

$\frac{d}{d\boldsymbol{u}_1}\boldsymbol{u}_2 = \boldsymbol{I}$

$\frac{d}{d\boldsymbol{b}_i}\boldsymbol{u}_2 = \boldsymbol{I}$

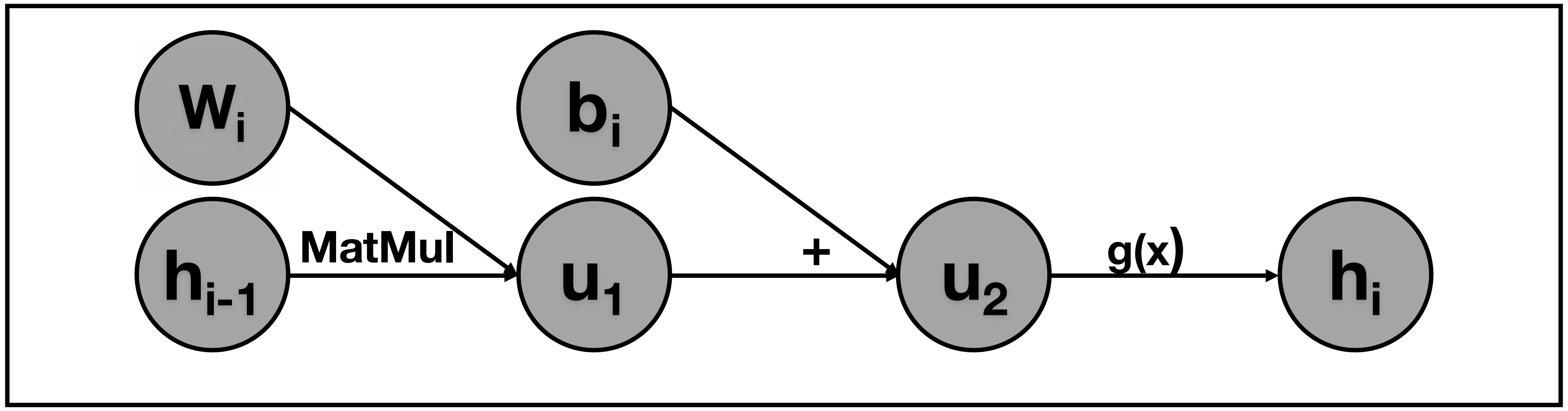
$\boldsymbol{u}_2 = \boldsymbol{u}_1 + \boldsymbol{b}_i$

$\frac{d}{d\boldsymbol{u}_2}\boldsymbol{u}_3 = \boldsymbol{g}'(\boldsymbol{u}_2)$

$\boldsymbol{u}_3 = g(\boldsymbol{u}_2) = \boldsymbol{h}_i$

Slides by Michael Lutter

Backpropagation



$$\nabla_{\boldsymbol{b}_i} J(\theta) = \frac{d\boldsymbol{u}_2}{d\boldsymbol{b}_i}\frac{d\boldsymbol{u}_3}{d\boldsymbol{u}_2} \odot \nabla J_{\boldsymbol{u}_3} = \boldsymbol{I}\ \boldsymbol{g}'(\boldsymbol{u}_2) \odot \nabla J_{\boldsymbol{u}_3}$$

$$\nabla_{\boldsymbol{W}_i} J(\theta) = \frac{d\boldsymbol{u}_1}{d\boldsymbol{W}_1}\frac{d\boldsymbol{u}_2}{d\boldsymbol{u}_1}\frac{d\boldsymbol{u}_3}{d\boldsymbol{W}_i} \odot \nabla_{\boldsymbol{u}_3} J = (\boldsymbol{g}'(\boldsymbol{u}_2) \odot \nabla J_{\boldsymbol{u}_3})\ \boldsymbol{u}_{\boldsymbol{0}}^T$$

$$\nabla_{\boldsymbol{u}_0} J(\theta) = \frac{d\boldsymbol{u}_1}{d\boldsymbol{u}_0}\frac{d\boldsymbol{u}_2}{d\boldsymbol{u}_1}\frac{d\boldsymbol{u}_3}{d\boldsymbol{u}_2} \odot \nabla J_{\boldsymbol{u}_3} = \boldsymbol{W}_i^T\ \boldsymbol{g}'(\boldsymbol{u}_2) \odot \nabla J_{\boldsymbol{u}_3}$$

Slides by Michael Lutter

Status Quo – Image Classification

MNIST

10	classes
70k	Images
0.20 %	Human Performance
0.21 %	**Best Performance**

CIFAR 10

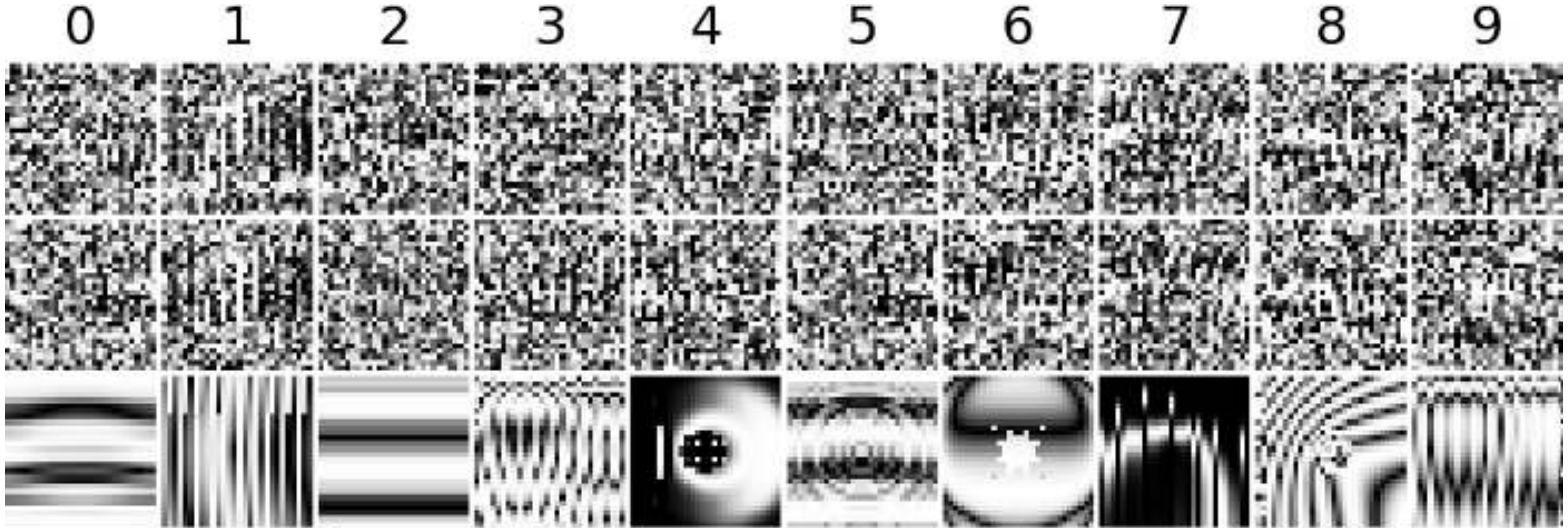
10	classes
60k	Images
6.00 %	Human Performance
4.41 %	**Best Performance**

Imagenet

1000	classes
1200k	Images
5.10 %	Human Performance
4.80 %	**Best Performance**

Slides by Michael Lutter

Status Quo – Image Classification



Anh Nguyen et.al., “Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images”, 2015

Content of this Lecture

Constructing Basis Functions

- Radial Basis Function Networks
- Deep Multi-Layer Perceptrons

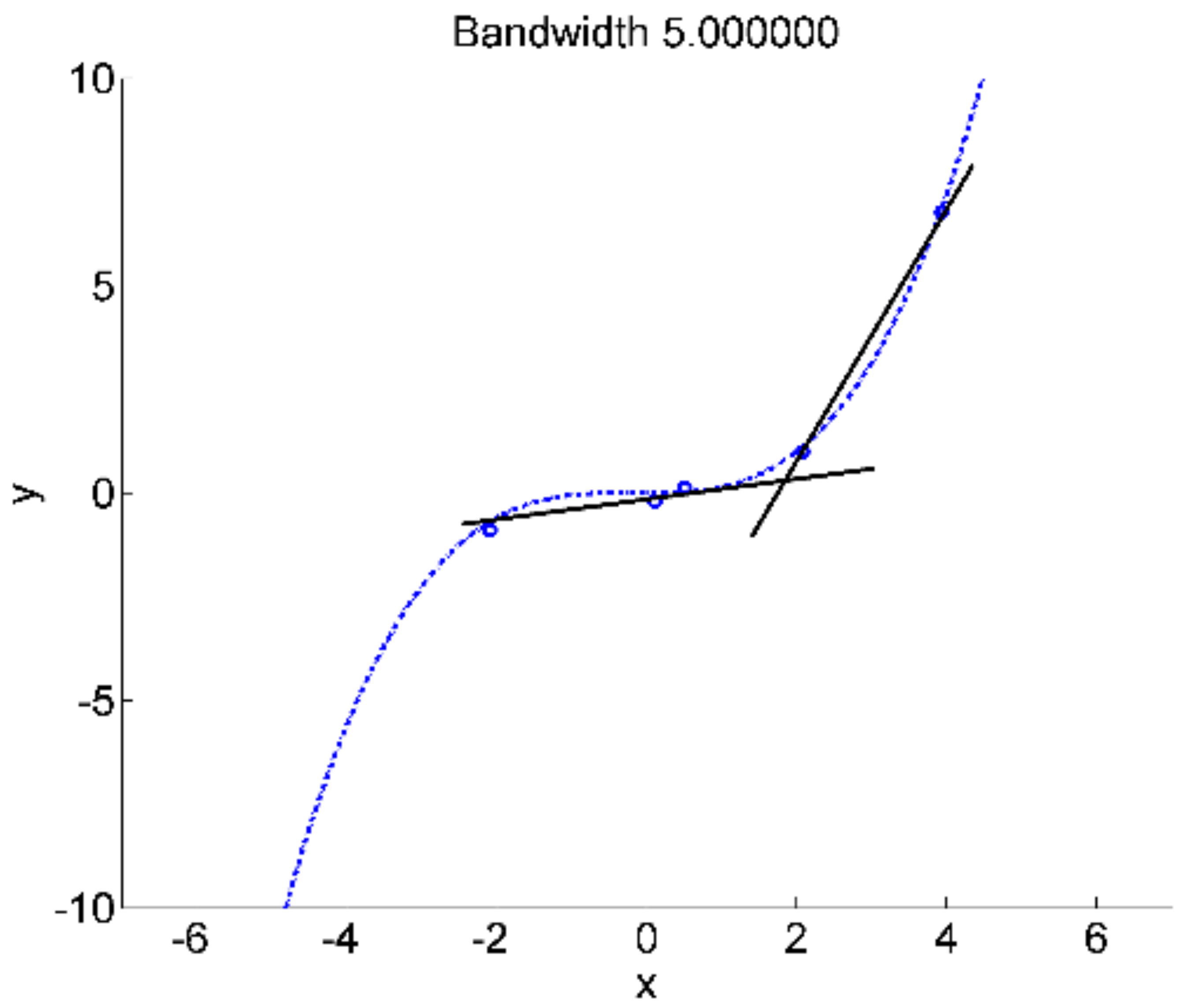
Non-Parametric Approaches

- Locally Weighted Regression
- Kernel Methods

Type 2: Non-Parametric Regression

- If you choose to have one feature/basis function per sample, you have a "**non-parametric method**" $\Rightarrow$ Don't need to select the number of bases
- Non-parametric means
  - infinitely many parameters not no parameters
  - expressiveness of the model depends on the number of data points
  - No predetermined "parametric" form necessary
    - (e.g., "5th-degree polynomial")
- One of them is **locally-weighted linear regression**...

Example: Locally Linear Solutions



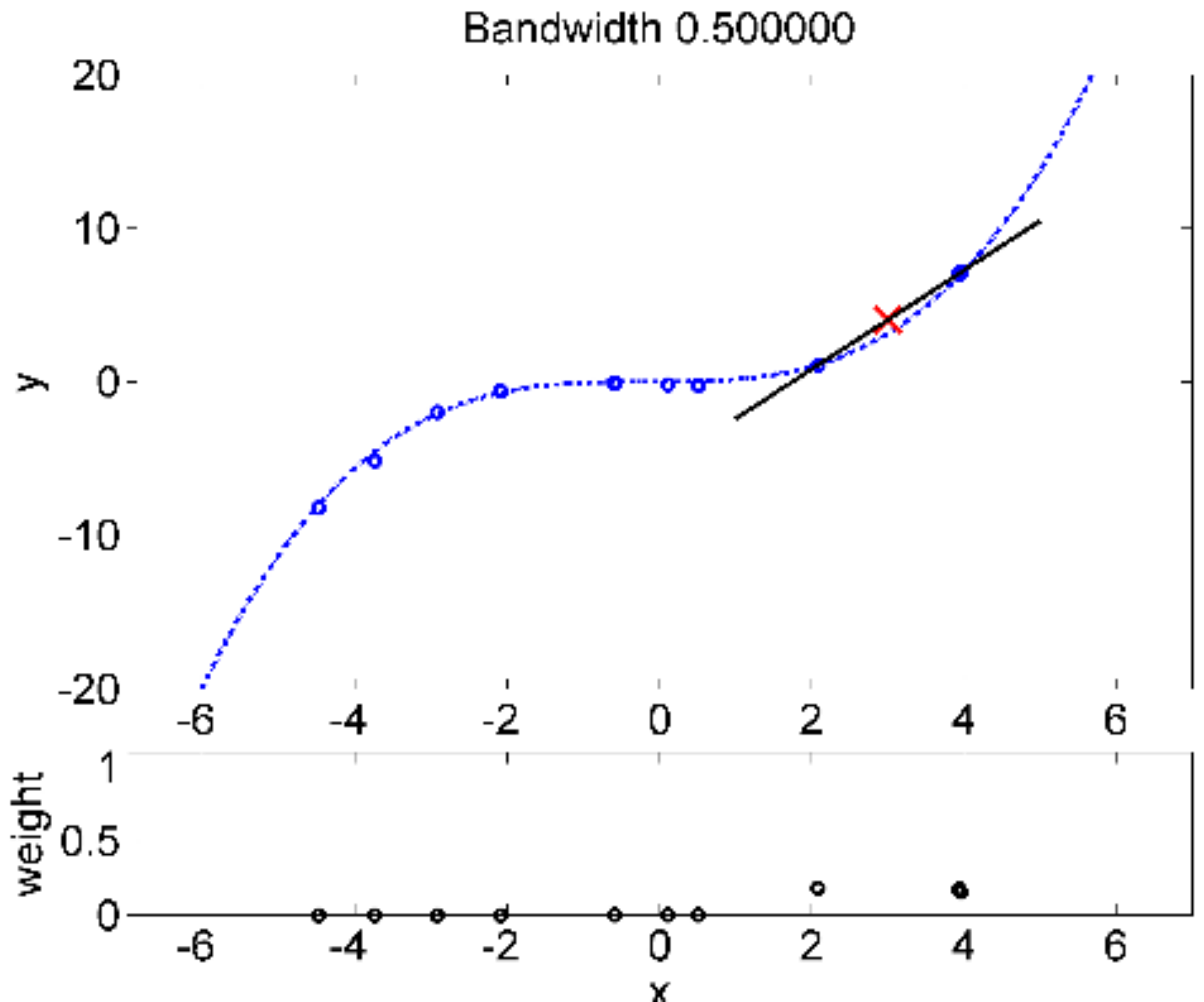
Locally all data is linear!

Locally all data is linear ...

so why don't we take the neighboring data points to predict the solution?

- Use higher importance or weighting of neighboring data points
- For each query point $\boldsymbol{x}$, weight training points $\boldsymbol{x}_i$ by

$$w_i(\boldsymbol{x}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{x}_i||^2}{2l^2}\right) \ldots \text{squarred exponential weighting}$$



Weighted Linear Regression

Weighted cost function

$$J = \frac{1}{2}\sum_{i=1}^{N} w_i(\mathbf{x})(y_i - \mathbf{f}_\theta(\mathbf{x}_i))^2, \qquad w_i(\boldsymbol{x}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{x}_i||^2}{2l^2}\right)$$

The function is linear in x

$$f_\theta(\boldsymbol{x}) = \boldsymbol{\theta}^T \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix} = \boldsymbol{\theta}^T \tilde{\boldsymbol{x}}$$

In matrix form with $W = \mathrm{diag}(w_1, w_2, w_3, \ldots, w_n)$:

$$J = \frac{1}{2}(\tilde{\boldsymbol{X}}\boldsymbol{\theta} - \boldsymbol{y})^T \boldsymbol{W} (\tilde{\boldsymbol{X}}\boldsymbol{\theta} - \boldsymbol{y})$$

Weighted Linear Regression

The solution to this problem: weighted pseudo inverse

$$\boldsymbol{\theta} = (\tilde{\boldsymbol{X}}^T \boldsymbol{W} \tilde{\boldsymbol{X}})^{-1} \tilde{\boldsymbol{X}}^T \boldsymbol{W} \boldsymbol{y}$$

- **W** can be large - don’t implement it like this...
- Dismiss data points with small weights / use bsxfun

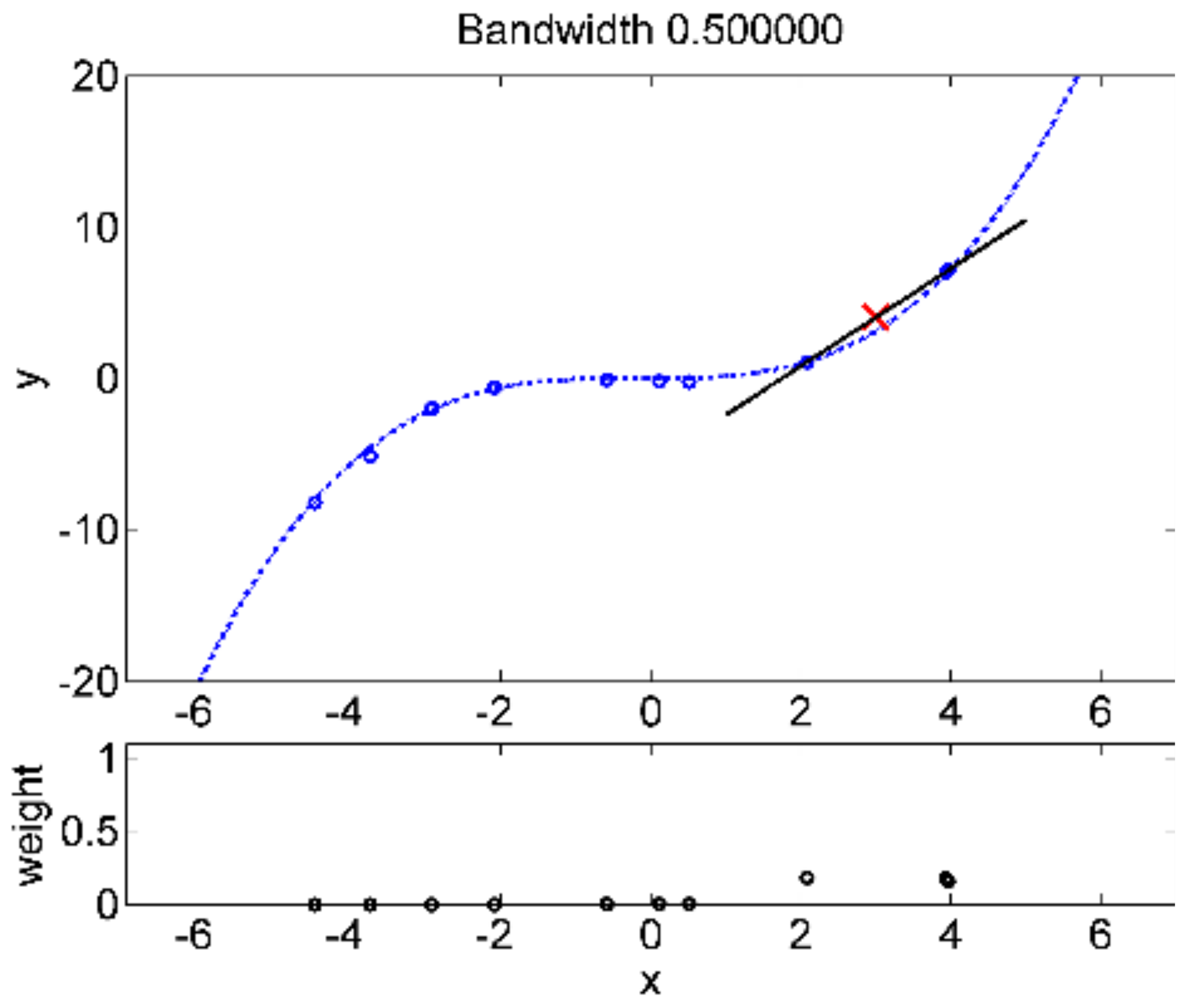
Local Ridge Regression:

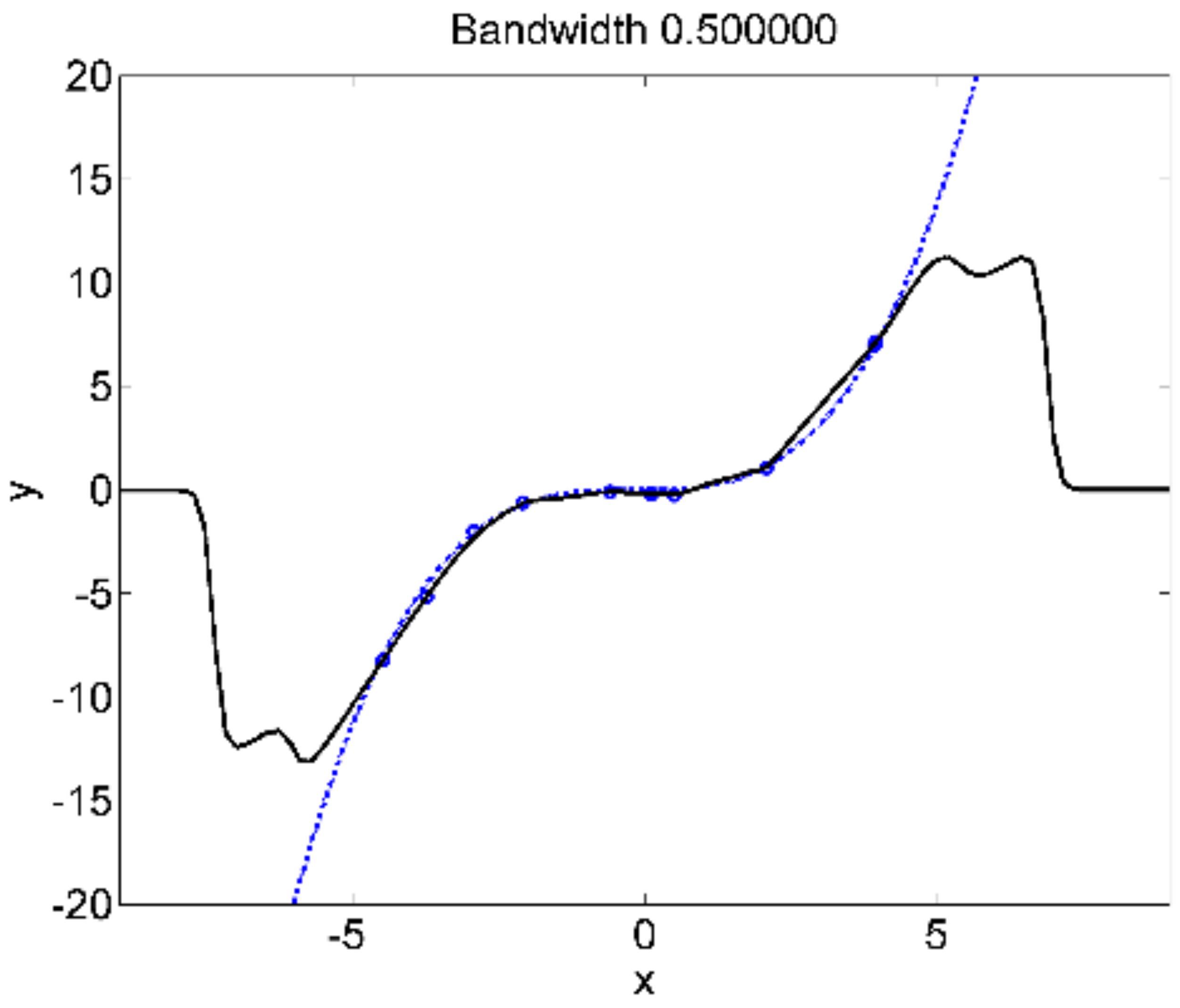
$$\boldsymbol{\theta} = (\tilde{\boldsymbol{X}}^T \boldsymbol{W} \tilde{\boldsymbol{X}} + \sigma^2 \boldsymbol{I})^{-1} \tilde{\boldsymbol{X}}^T \boldsymbol{W} \boldsymbol{y}$$

Advantages: Fast(real-time capable), scales(lots of data), interpolates linearly(useful in control)
Disadvantage: Tuning is not easy

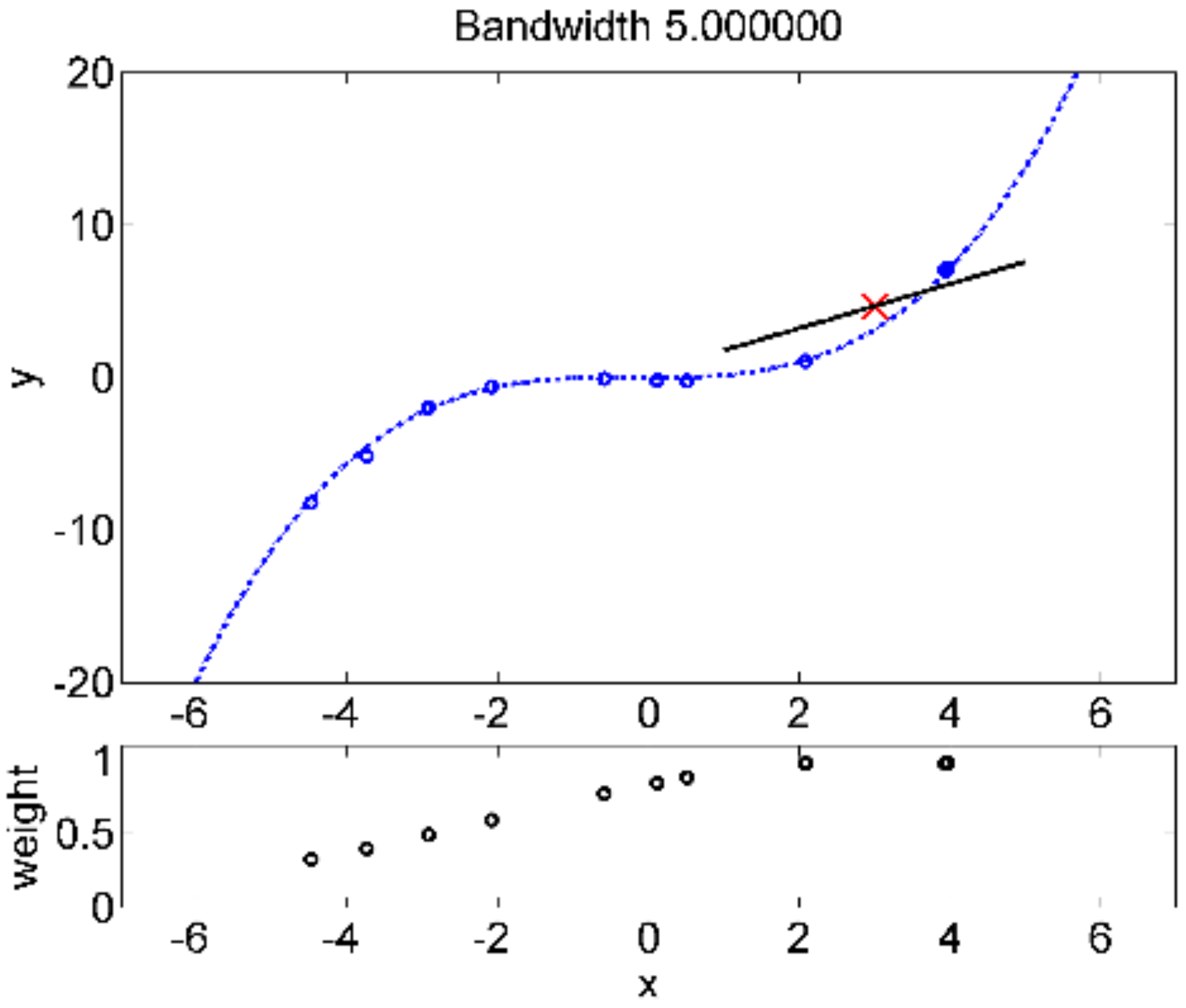
Frequently method of choice for control problems!

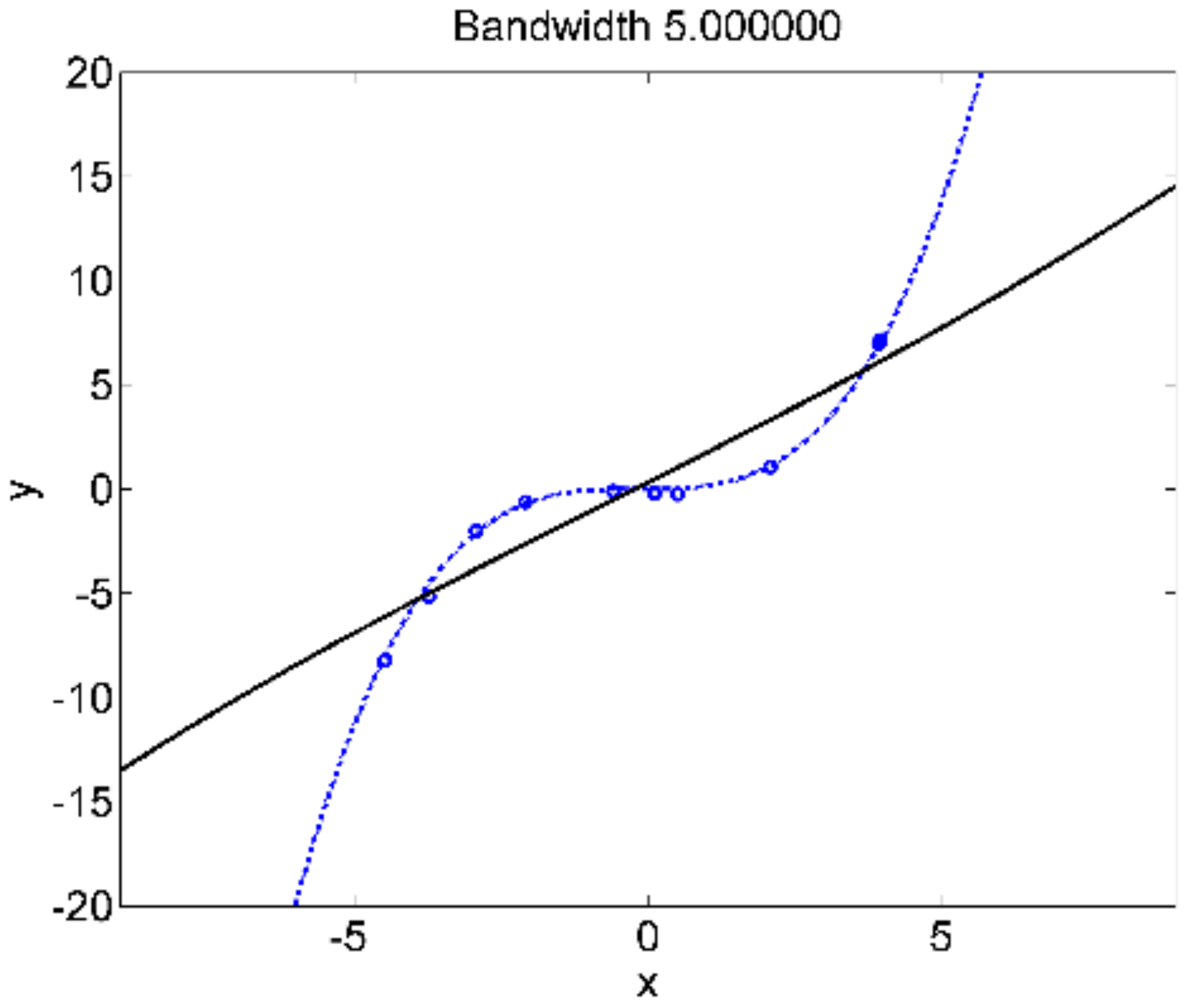
Solution with Locally-Weighted Regression



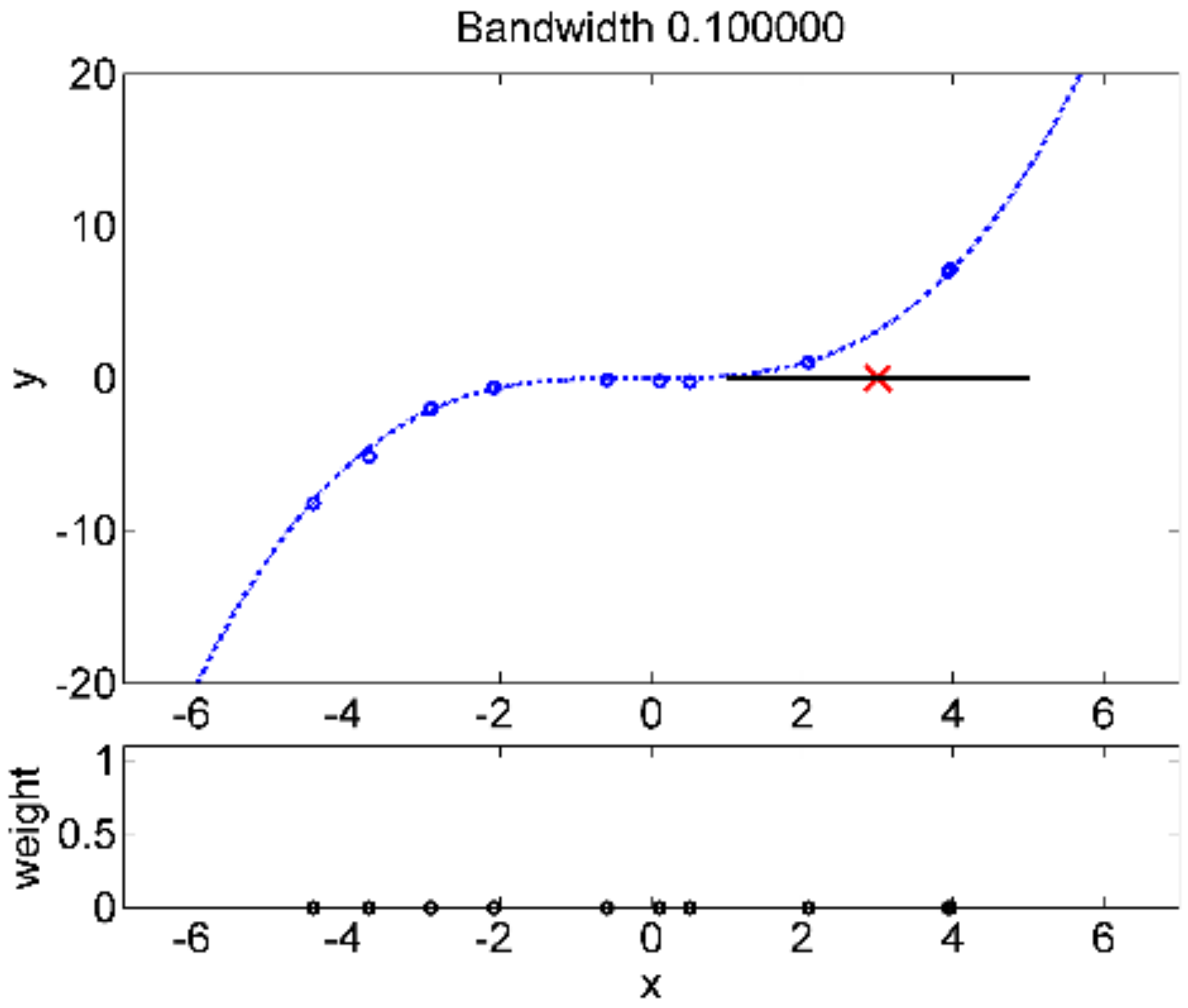


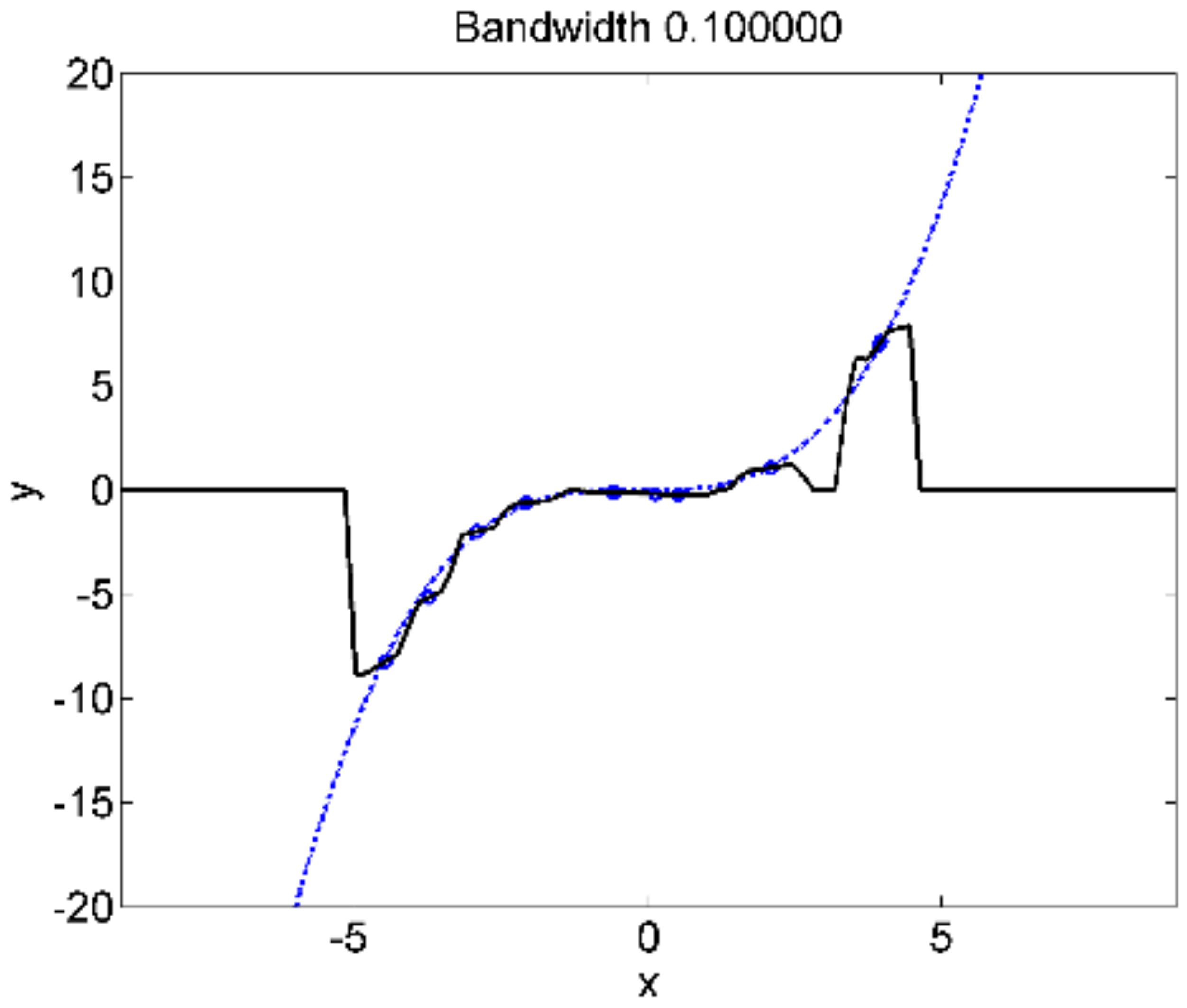
Solution with Locally-Weighted Regression





Solution with Locally-Weighted Regression



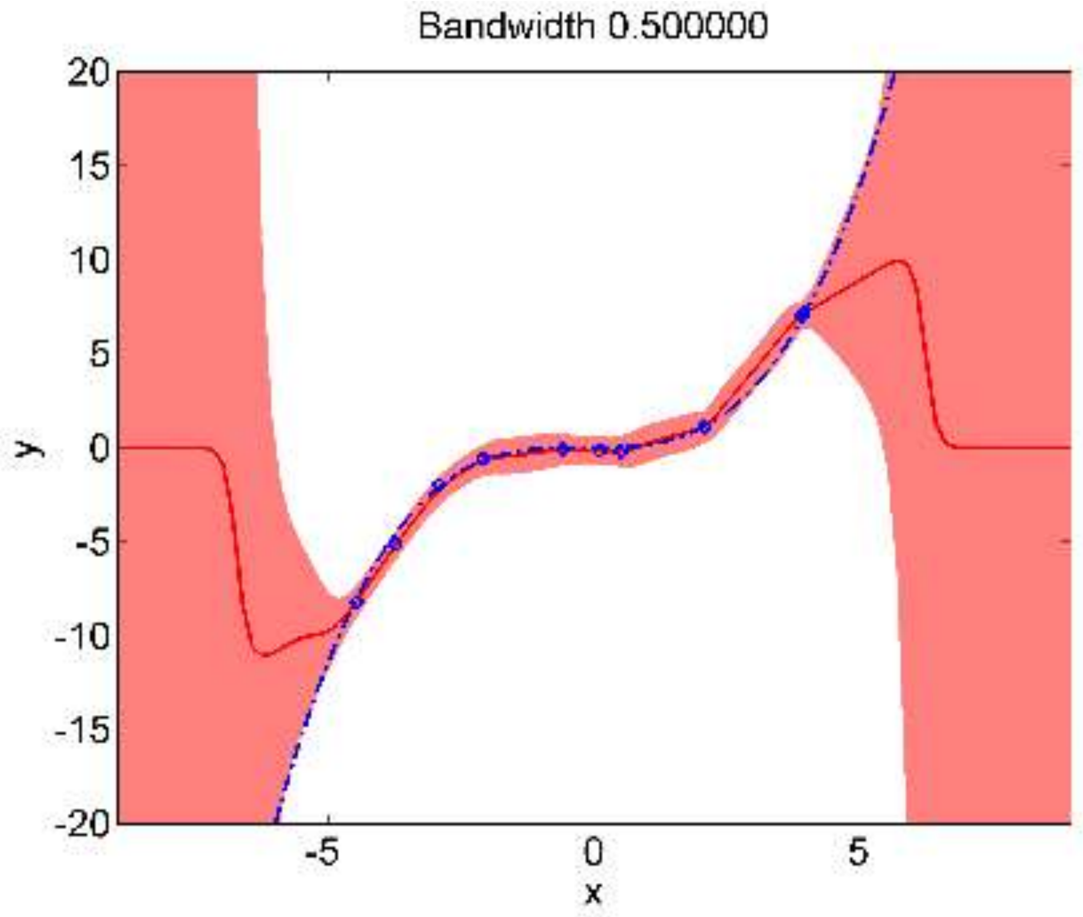


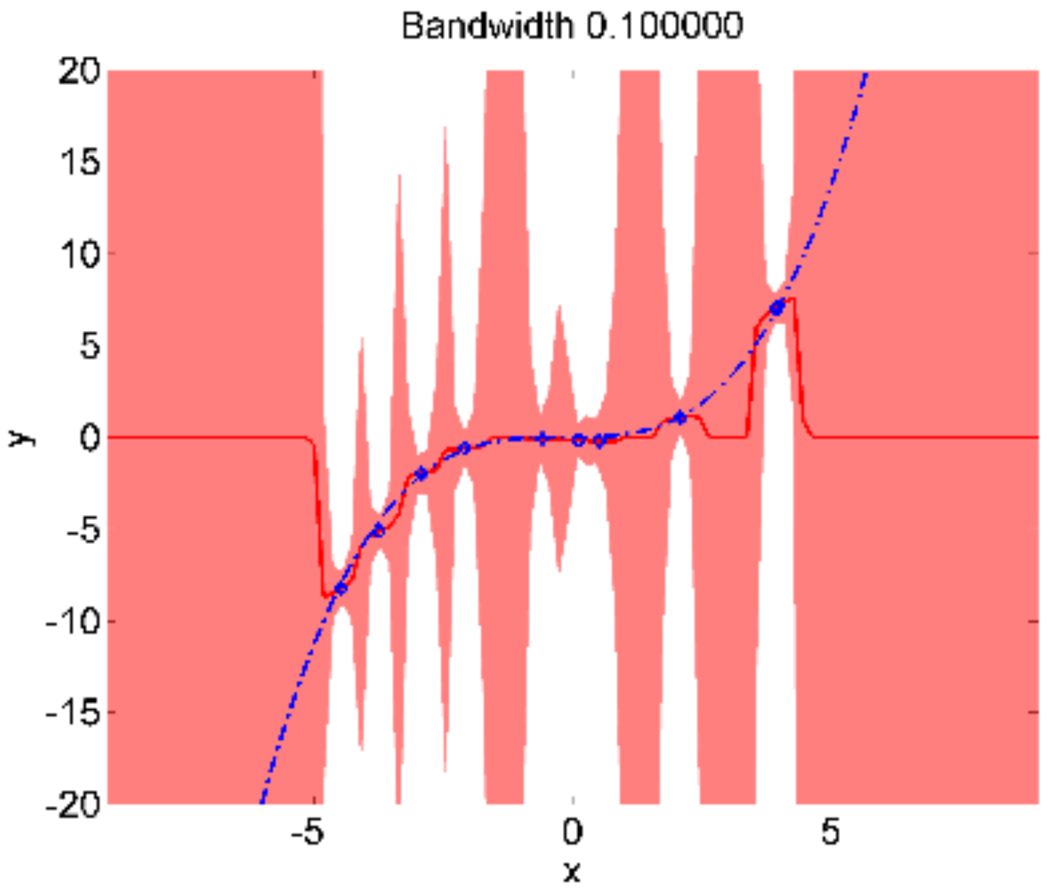
Weighted Linear Regression

Locally Weighted Bayesian Linear Regression

$$p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{W}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$$

$$\boldsymbol{\Sigma}_N = (\tilde{\boldsymbol{X}}^T \boldsymbol{W} \tilde{\boldsymbol{X}} + \sigma^2 \lambda \boldsymbol{I})^{-1} \qquad \boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N \tilde{\boldsymbol{X}} \boldsymbol{W} \boldsymbol{y}$$





Content of this Lecture

Constructing Basis Functions

- ➡ Radial Basis Function Networks

Non-Parametric Approaches

- ➡ Locally Weighted Regression
- ➡ Kernel Methods

Type 3: Kernel Methods

Kernel methods rely on the 'kernel trick'

➡ It is sufficient to evaluate the scalar product between two samples in feature space, called kernel

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \phi(\boldsymbol{x}_1)^T \phi(\boldsymbol{x}_2)$$

Why is this useful?

- Kernels are easier to design than features
- The feature space can be possibly infinite dimensional.
- We just need to be able to compute the scalar product

Type 3: Kernel Methods

Example: One RBF feature at every position c

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \phi(\boldsymbol{x}_1)^T \phi(\boldsymbol{x}_2) = \int \exp\left(-\frac{||\boldsymbol{x}_1 - \boldsymbol{c}||^2}{2l^2}\right) \exp\left(-\frac{||\boldsymbol{x}_2 - \boldsymbol{c}||^2}{2l^2}\right) d\boldsymbol{c}$$

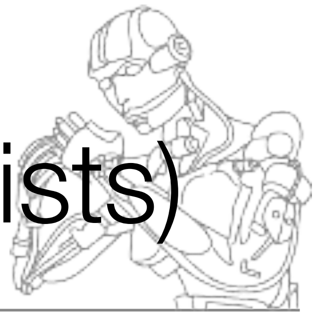
$$= \exp\left(-\frac{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2}{4l^2}\right)$$

➡ Reduces to an RBF feature at each sample

General conditions for kernels

- symmetric: $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\boldsymbol{x}_2, \boldsymbol{x}_1)$
- positive definite…

Kernel Ridge Regression (Kernels for Freqentists)



Look at the predictions with the MAP/RR estimator (linear regression) again:

$$y(\boldsymbol{x}_*) = \phi(\boldsymbol{x}_*)^T \boldsymbol{\theta} = \phi(\boldsymbol{x}_*)^T (\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}^T \boldsymbol{y}$$

Even more general, the **Woodbury identity for matrix inversion** yields

$$\boxed{(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{I}_D)^{-1} \boldsymbol{\Phi}^T} = \boxed{\boldsymbol{\Phi}^T (\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda \boldsymbol{I}_N)^{-1}}$$

This yields

$$y(\boldsymbol{x}_*) = \phi(\boldsymbol{x}_*)^T \boxed{\underbrace{(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{I}_D)^{-1}}_{D \times D} \boldsymbol{\Phi}^T} \boldsymbol{y}$$

$$= \phi(\boldsymbol{x}_*)^T \boxed{\boldsymbol{\Phi}^T \underbrace{(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda \boldsymbol{I}_N)^{-1}}_{N \times N}} \boldsymbol{y}$$

Equivalent solution to ridge regression
Why is this potentially useful?

Kernel Ridge Regression (Kernels for Freqentists)

Let’s say, we have an inner product for our features:

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_i)$$

We can use this to rewrite

$$y = \phi(x)^T \Phi (\Phi\Phi^T + \lambda^{-1} I)^{-1} Y$$

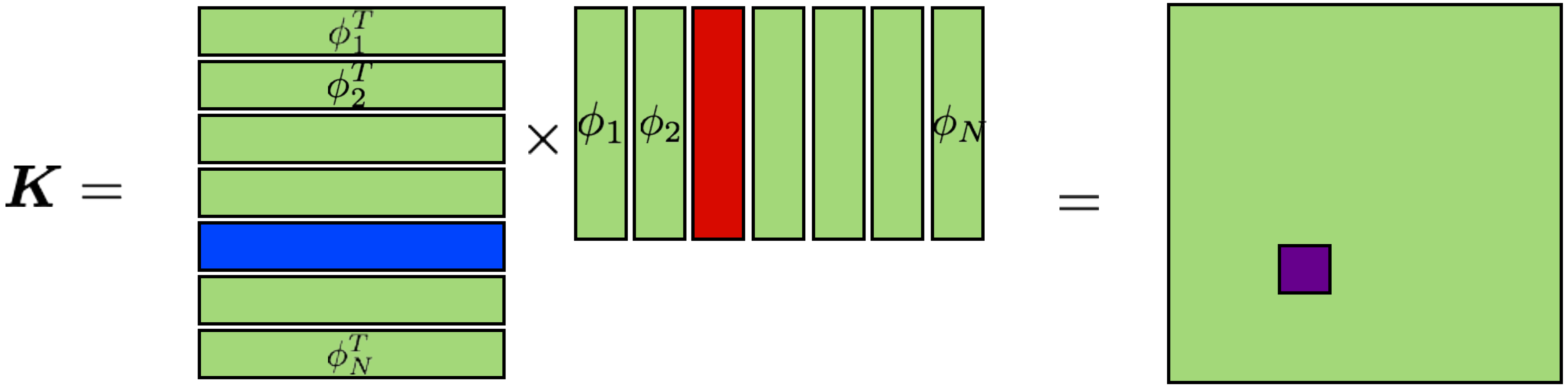
Into

$$y = k(x)(K + \lambda^{-1} I)^{-1} Y$$

This a “Kernelization” of regression! But why is this a good idea?

Kernels are scalar products in feature space!

$K_{ij} = \lambda^{-1}\boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{\phi}(\boldsymbol{x}_j) = k(\boldsymbol{x}_1, \boldsymbol{x}_2)$ scalar products in feature space



Kernels can measure the similarity between data points in feature space without evaluating or explicitly knowing all features!

Particularly useful when there are more features than data points as, e.g., in images…

Bayesian Linear Regression revisited

We have:

Data-Likelihood: $p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{X}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{\Phi\theta}, \sigma^2\boldsymbol{I})$

Prior: $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{0}, \lambda^{-1}\boldsymbol{I})$

If we integrate out the weights, we get

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int \mathcal{N}(\boldsymbol{y}|\boldsymbol{\Phi\theta}, \sigma^2\boldsymbol{I})\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{0}, \lambda^{-1}\boldsymbol{I})d\boldsymbol{\theta}$$

$$= \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \sigma^2\boldsymbol{I} + \lambda^{-1}\boldsymbol{\Phi\Phi}^T)$$

Defines a multivariate Gaussian distribution over the samples

➡ Samples are correlated as the marginalized weight vector is the same for each sample

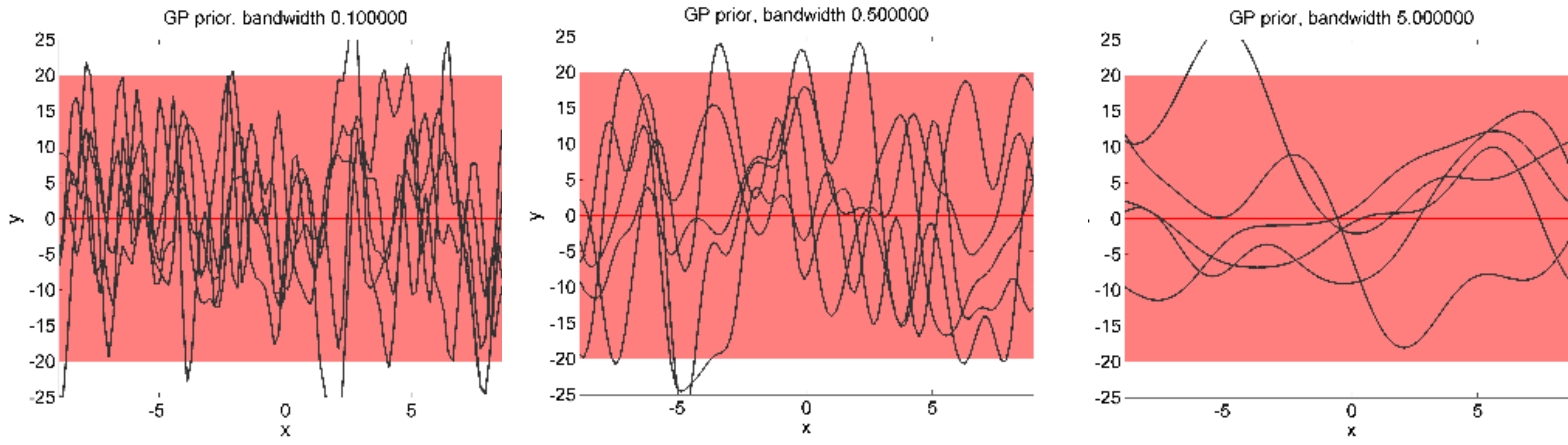
Bayesian Kernel Regression: Gaussian Processes (GPs)

Replace the features in Bayesian Linear Regression by a Kernel and you obtain:

$$p(\boldsymbol{y}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \sigma^2\boldsymbol{I} + \boldsymbol{K}) \text{ with } \boldsymbol{K} = \lambda^{-1}\boldsymbol{\Phi}\boldsymbol{\Phi}^T$$

This method is called a Gaussian Process $\mathcal{GP}(\mathbf{0}, \boldsymbol{K})$
with covariance function *k*

Sampling from the GP-Prior


GP prior. bandwidth 0.100000
GP prior. bandwidth 0.500000
GP prior. bandwidth 5.000000
y
x
25
20
15
10
5
0
-5
-10
-15
-20
-25
-5
0
5

The kernel bandwidth of the exponential kernel is a prior on the smoothness on the function

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp\left(-\frac{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2}{4l^2}\right)$$

GP-Posterior

Now we observe a data set given by $\boldsymbol{y}$ and $\boldsymbol{X}$ and we want to predict y^* for $\boldsymbol{x}^*$

- We can write down the GP prior for the concatenated data

$$p\left(\begin{bmatrix}\boldsymbol{y}\\ y_*\end{bmatrix} \middle| \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{x}_*\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{y}\\ y_*\end{bmatrix} \middle| 0, \begin{bmatrix}\boldsymbol{K} & \boldsymbol{k}(\boldsymbol{X},\boldsymbol{x}_*)\\ \boldsymbol{k}(\boldsymbol{x}_*,\boldsymbol{X}) & k(\boldsymbol{x}_*,\boldsymbol{x}_*)\end{bmatrix} + \sigma^2\boldsymbol{I}\right)$$

- We get the GP-posterior by Gaussian conditioning (see refresher)

$$p(y_*|\boldsymbol{y},\boldsymbol{X},\boldsymbol{x}_*) = \mathcal{N}(\boldsymbol{y}_*|\mu(\boldsymbol{x}_*),\sigma^2(\boldsymbol{x}_*))$$

GP-Posterior

$$p(y_*|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{x}_*) = \mathcal{N}(\boldsymbol{y}_*|\mu(\boldsymbol{x}_*), \sigma^2(\boldsymbol{x}_*))$$

Same solution as in Kernel Ridge Regression

Predictive mean

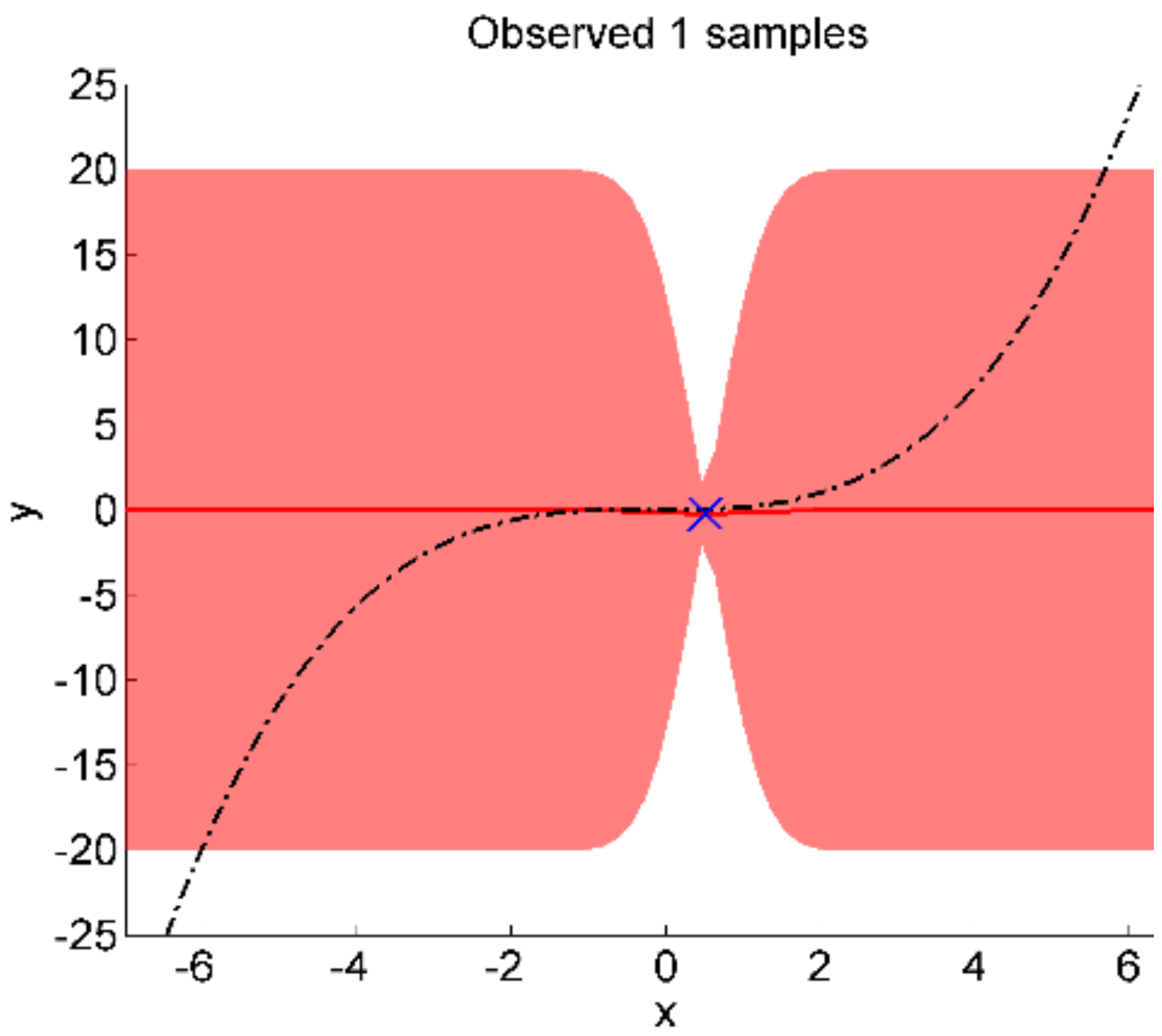
$$\mu(\boldsymbol{x}_*) = k(\boldsymbol{x}_*, \boldsymbol{X})(\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y}$$

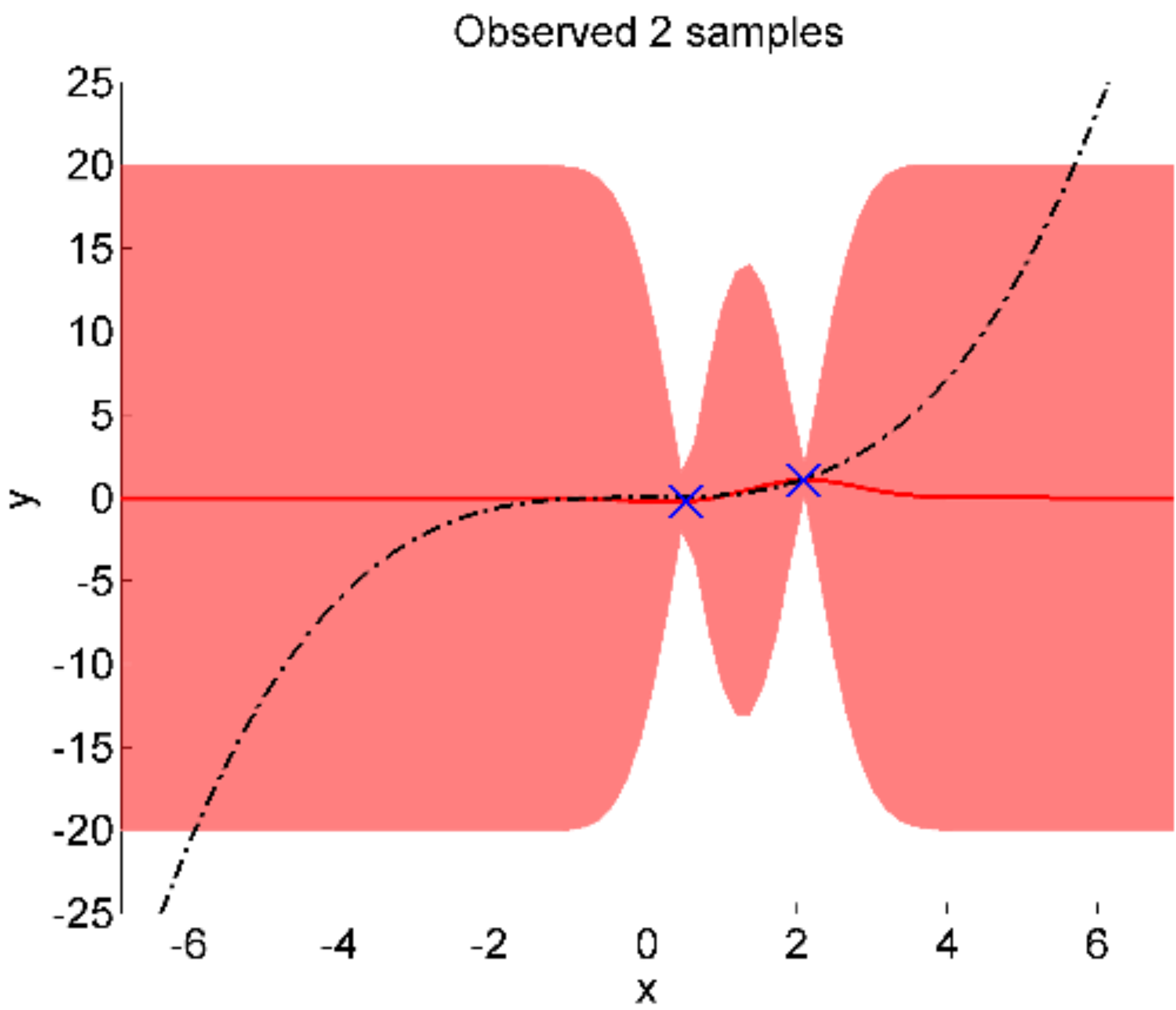
Predictive variance

$$\begin{aligned}\sigma^2(\boldsymbol{x}_*) =& k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 \\ & - \boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{X})(\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*)\end{aligned}$$

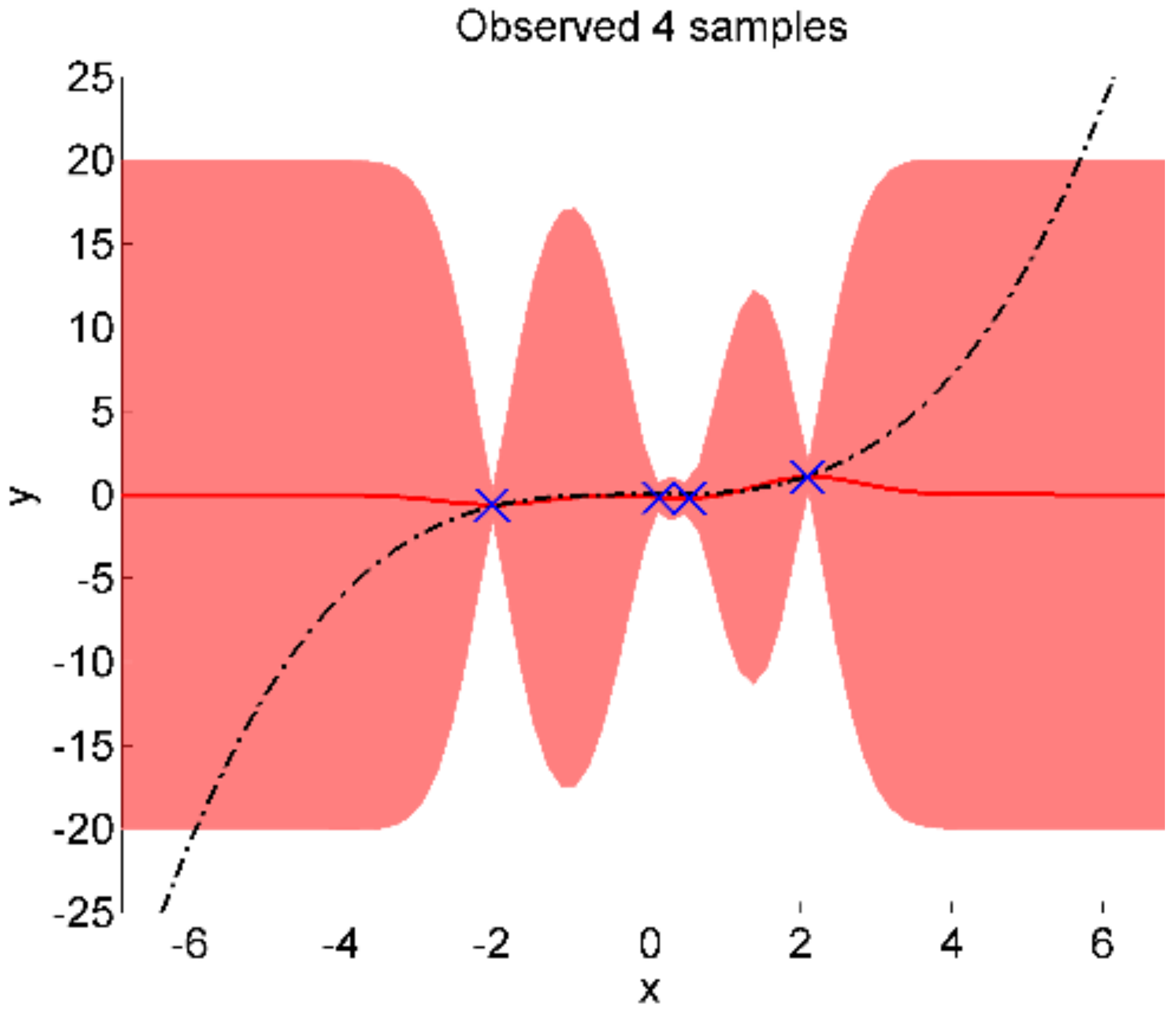
GPs = Kernel Ridge Regression + Knowledge on your Uncertainty!

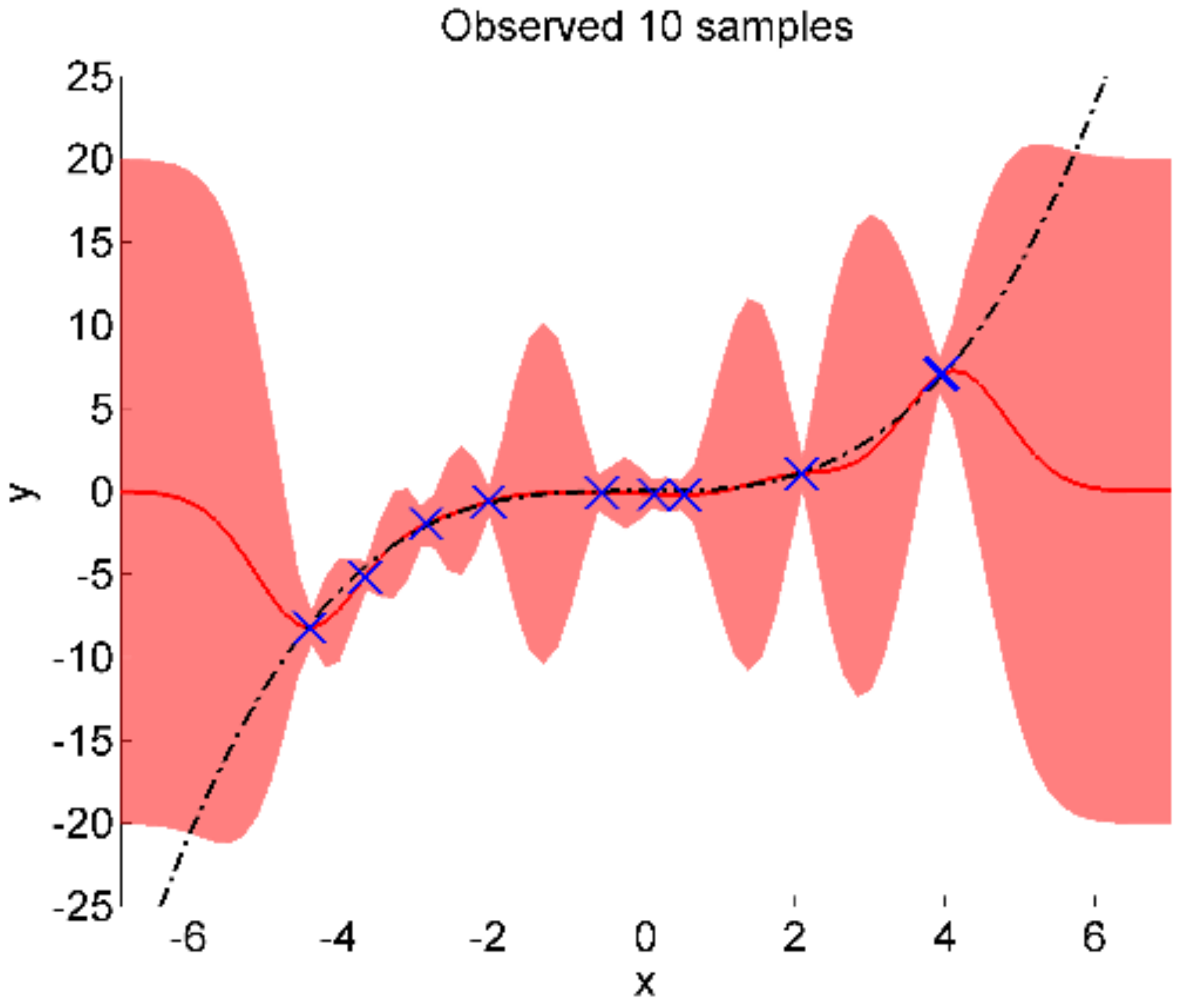
GP-Posterior





GP-Posterior





Gaussian Processes

A key advantage over Kernel Ridge Regression

Optimization of Hyper-Parameters (All learning becomes optimization)

- The parameters of the kernel are called hyper-parameters
- Cross validation or maximization of marginal log-likelihood

GPs vs. Bayesian Linear Regression:

- GP is Kernel Ridge Regression with uncertainty! Same mean!
- GPs is kernelized Bayesian Linear Regression
- Kernels are often easier to use than features!

GPs are currently the gold standard for regression!

… if you have few data points!

Summary

- You should have a really good overview of machine learning by now.
- You should remember the following **regression methods**
  - Least-Squares Regression / Ridge Regression
  - Bayesian Regression
  - Radial-Basis Function Regression
  - Tiny insight into Deep Neural Networks
  - Locally-Weighted Linear Regression
  - Kernel Ridge Regression and GPs
- You should know how to **choose the right method** for a regression problem