Radial Basis Function networks for regression and classification

April 5, 2011

Outline

Classification and Regression

- Given a dataset (TRAINING SET) of input-target pairs:

$$D = \left\{\mathbf{x}^i; \mathbf{t}^i\right\}_{i=1,2,\ldots,N}$$

- the learning task is to predict the corresponding $\mathbf{t}$ to each new input vector $\mathbf{x} \notin D$
  1. **classification**: $\mathbf{t}^i$ are discrete (class label)
  2. **regression**: $\mathbf{t}^i$ continuous value
- Underlying relation between $\mathbf{x}$ and $\mathbf{t}$ given by:

$$\mathbf{t}^i = f\left(\mathbf{x}^i\right) + \epsilon^i$$

- GOAL: to approximate f with a parametric function $\mathbf{y}(\mathbf{x}; \mathbf{w})$
  - polinomial, ffw, etc.

Radial basis function exact interpolation

- Consider a mapping from an input space $X \subseteq \mathbb{R}^D$ to a target space $Y \subseteq \mathbb{R}$

$$f(\mathbf{x}) : \mathbf{x} \in X \subseteq \mathbb{R}^D \longrightarrow y \in Y \subseteq \mathbb{R}$$

- In general we do not know the function $f(\mathbf{x})$
- We only have a set of input-target pairs called *Training Set (TS)*

$$\begin{array}{ccc} \mathbf{x}^1 & \rightarrow & y^1 \\ \vdots & & \vdots \\ \mathbf{x}^N & \rightarrow & y^N \end{array}$$

Radial basis function exact interpolation

- The goal of exact interpolation is to find a function $h(\mathbf{x})$ such that:

$$h(\mathbf{x^i}) = t^i, i = 1, \ldots, N$$

- The radial basis function approach introduces a set of N basis functions (one for each data point) of the form:

$$\phi_n(\mathbf{x}) = \phi(\| \mathbf{x} - \mathbf{x}^n \|)$$

where $\phi(\cdot)$ is some non linear function and $\| \mathbf{x} - \mathbf{x}^n \|$ denote the Euclidean distance between the input $\mathbf{x}$ and the point $\mathbf{x}^n$

Form of the basis functions

- Gaussian: $\phi(x) = exp\left(-\frac{x^2}{2\sigma^2}\right)$
- $\phi(x) = \left(x^2 - \sigma^2\right)^{-\alpha}$, with $\alpha > 0$
- Thin-plate: $\phi(x) = x^2 ln(x)$
- ...

We will consider the case of Gaussian basis function:

$\phi_j(\mathbf{x}) = exp\left(-\frac{\|\mathbf{x}-\boldsymbol{\mu}_j\|}{2\sigma_j^2}\right)$

Radial basis function exact interpolation

- The output of the mapping is a linear combination of the basis functions:

$$h(\mathbf{x}) = \sum_{j=1}^{N} w_j \phi_j(\mathbf{x})$$

- Thus the interpolation condition can be expressed:

$$\sum_{j=1}^{N} w_j \Phi_{ij} = t^i \quad i = 1, 2, \ldots, N$$

Radial basis function exact interpolation

- Thus the interpolation condition can be expressed:

$$\sum_{j=1}^{N} w_j \Phi_{ij} = t^i \quad i = 1, 2, \ldots, N$$

- this condition can be rewritten in a matrix form: $\mathbf{\Phi w^T} = \mathbf{t}$

$$\begin{pmatrix} \Phi_{11} & \Phi_{12} & \ldots & \Phi_{1N} \\ \Phi_{21} & \Phi_{22} & \ldots & \Phi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_{N1} & \Phi_{N2} & \ldots & \Phi_{NN} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} = \begin{pmatrix} t^1 \\ t^2 \\ \vdots \\ t^N \end{pmatrix}$$

- $\mathbf{\Phi}$ is a matrix of dimension $N \times N$ of components $\Phi(i,j) = \phi_j(\mathbf{x}^i)$
- $\mathbf{w}$, $\mathbf{t}$ are vector of dimension N

Radial basis function exact interpolation

$$\mathbf{\Phi w}^{\mathbf{T}} = \mathbf{t}$$

- if $\mathbf{\Phi}$ is a non singular matrix the solution for the parameters can be found simply by:

$$\mathbf{w} = \mathbf{\Phi}^{-1}\mathbf{t}$$

MATLAB EXPERIMENTS

MATLAB EXPERIMENTS

Radial basis function exact interpolation

- This method can generalized for mapping to multidimensional output space $Y \subseteq \mathbb{R}^C$
- In this case each input vector $\mathbf{x}^n$ must be mapped exactly onto an output vector $\mathbf{t}^n$of components t_k^n with $k = 1, \ldots, C$
- Thus the interpolation condition can be written:

$$\mathbf{\Phi W^T} = \mathbf{T}$$

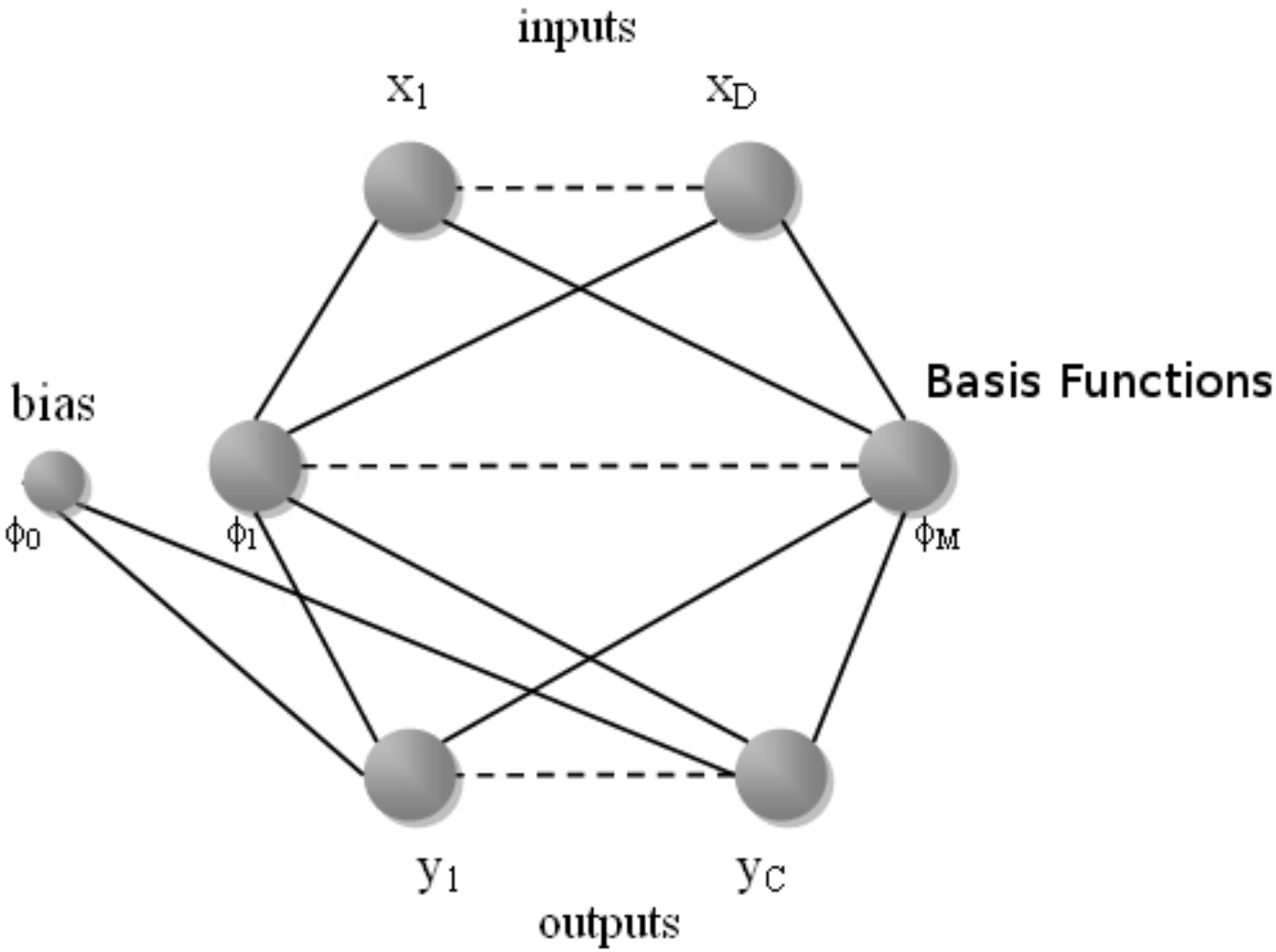
- Where $\mathbf{W}$ is a matrix of dimension $C \times N$ while $\mathbf{T}$ is a matrix of dimension $N \times C$
- To find the solution we must invert the matrix $\mathbf{\Phi}$ and perform a matrix product $\mathbf{\Phi}^{-1}\mathbf{T}$

Radial basis function networks

- We made the following changes to the previous model:
  1. The number M of basis functions $\phi_1(\mathbf{x}), \ldots \phi_M(\mathbf{x})$ is much less than N (number of data points);
  2. The centers $\boldsymbol{\mu}_j$ and the width σ_j of basis functions are determined during the training process;
  3. The bias parameters are included in the linear sum.

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}$$

Radial basis function networks



Radial basis function networks training

- Two stage training procedure:
  1. Unsupervised training to determine the parameters of the basis functions;
  2. By fixing the parameters of the basis function we determine the weights (w_{kj}) by Supervised training

Determining the weights of the network

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}$$

we can absorb the bias parameters into the weights to give:

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x})$$

where the function $\phi_0(\mathbf{x}) = 1$

Determining the weights of the network

- Consider a Training Set consisting of N data point and rbf network with M basis functions (internal nodes)
- We can construct the matrix $\mathbf{\Phi}$ of dimension $N \times (M+1)$ as follows:

$$\begin{pmatrix} \Phi_{10}\Phi_{11} & \Phi_{12} & \cdots & \Phi_{1M} \\ \Phi_{20}\Phi_{21} & \Phi_{22} & \cdots & \Phi_{2M} \\ \vdots\vdots & \vdots & \ddots & \vdots \\ \Phi_{N0}\Phi_{N1} & \Phi_{N2} & \cdots & \Phi_{NM} \end{pmatrix}$$

Determining the weights of the network

- The expression for the output of the rbf network

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x})$$

- can be expressed as:

$$\mathbf{Y} = \mathbf{\Phi}\mathbf{W}^T$$

- where $\mathbf{Y}$, $\mathbf{\Phi}$ and $\mathbf{W}$ are matrix of dimension $N \times C$, $N \times M$ and $C \times M$ respectively;

Determining the weights of the network

$$\mathbf{Y} = \mathbf{\Phi}\mathbf{W}^T$$

- We can find the parameters $\mathbf{W}$ by minimizing a suitable error function (e.g. sum-of-squares)

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{C}\{y_k(\mathbf{x}^n) - t_k^n\}^2$$

- The weights values are given by the solution of the linear equation:

$$\mathbf{\Phi}^T\mathbf{\Phi}\mathbf{W}^T = \mathbf{\Phi}^T\mathbf{T}$$

- The solution is $\mathbf{W}^T = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{T}$

Determining the weights of the network

$$
\begin{aligned}
E &= \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{C}\{y_k(\mathbf{x}^n) - t_k^n\}^2 \\
&= \frac{1}{2}\parallel \mathbf{\Phi}\mathbf{W}^T - \mathbf{T} \parallel^2
\end{aligned}
$$

- Deriving with respect to $\mathbf{W}$

$$
\begin{aligned}
\frac{\partial E}{\mathbf{W}} &= \mathbf{\Phi}^T\left(\mathbf{\Phi}\mathbf{W}^T - \mathbf{T}\right) \\
&= \mathbf{\Phi}^T\mathbf{\Phi}\mathbf{W}^T - \mathbf{\Phi}^T\mathbf{T}
\end{aligned}
$$

- Setting to zero the derivative and finding $\mathbf{W}$ we obtain: $\mathbf{W}^T = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{T}$

Determining the weights of the network

- It can be showed that the solution for $\mathbf{W}$ can be found using the Singular Value Decomposition (SVD)
  1. Decompose the matrix $\mathbf{\Phi} = \mathbf{U\Sigma V}^T$
  2. Compute $\mathbf{V\Sigma}^{-1}\mathbf{U}^T$
  3. $\mathbf{W} = \mathbf{V\Sigma}^{-1}\mathbf{U}^T\mathbf{T}$
- Where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrix of dimension $N \times N$ and $M \times M$ respectively and $\mathbf{\Sigma}$ is a matrix of dimension $N \times M$ with the singular value on the diagonal
- Note that $\mathbf{\Sigma}^{-1}$ denote a $M \times N$ matrix constructed as follows:

$$\mathbf{\Sigma}^{-1}(i,i) = \begin{cases} \frac{1}{\mathbf{\Sigma}(i,i)} & \text{if } \mathbf{\Sigma}(i,i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Determining the parameters of the basis functions

- We consider the case of Gaussian basis function:

$$\phi_j(\mathbf{x}) = exp\left(-\frac{\| \mathbf{x} - \boldsymbol{\mu}_j \|}{2\sigma_j^2}\right)$$

- Thus we have to determine the parameters $\boldsymbol{\mu}_j$ and σ_j for each basis function

Determining centers and width of the basis functions

- Different approach exists:
  - Subset of data points
  - Orthogonal least squares
  - **Clustering algorithms**
  - Gaussian mixture models

Clustering algorithms for selecting the parameters of the basis functions

- If we consider a clustering algorithm for which the number of clusters is predefined (e.g. K-Means):
  1. Set the number of cluster to M and run the clustering algorithm
  2. Set the centers of the basis functions equals to the centers of clusters
  3. Set the widths (variances) of the basis functions equals to the variances of clusters

How do we select the model complexity ?

- Choosing a very simple model may give rise to poor results ! (e.g. $M = 1$)
- Choosing a very complex model may give rise to over-fitting and thus poor generalization performace !
- One technique that is often used to control over-fitting is to still use a complex model but to add a penalty term to the error function in order to discourage the coefficients from reaching large values (*regularization*):

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{C}\{y_k(\mathbf{x}^n) - t_k^n\}^2 + \frac{\lambda}{2} \parallel \mathbf{w} \parallel^2$$

where λ is called regularization coefficient

- The solution is $\mathbf{W}^T = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\mathbf{\Phi}^T\mathbf{T}$

Determining the weights of the network

- It can be showed that the solution for $\mathbf{W}$ can be found using the Singular Value Decomposition (SVD)
  1. Decompose the matrix $\mathbf{\Phi} = \mathbf{U\Sigma V}^T$
  2. Compute $\mathbf{V\Sigma}^{-1}\mathbf{U}^T$
  3. $\mathbf{W} = \mathbf{V\Sigma}^{-1}\mathbf{U}^T\mathbf{T}$
- Where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrix of dimension $N \times N$ and $M \times M$ respectively and $\mathbf{\Sigma}$ is a matrix of dimension $N \times M$ with the singular value on the diagonal
- Note that $\mathbf{\Sigma}^{-1}$ denote a $M \times N$ matrix constructed as follows:

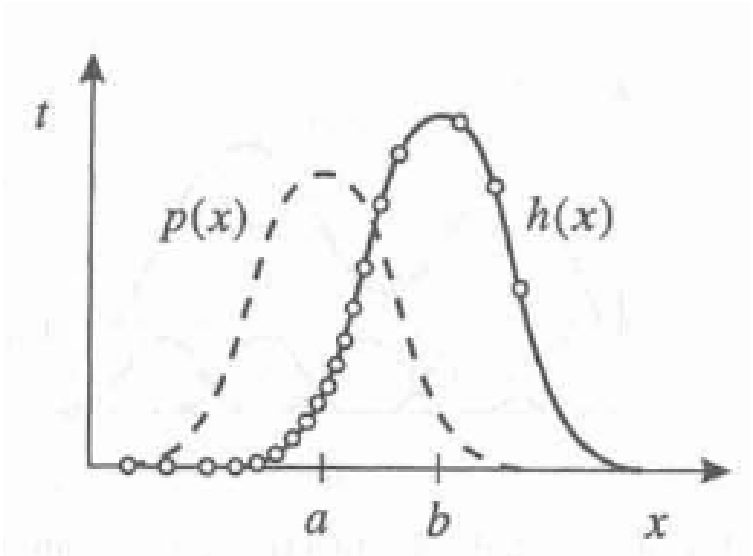
$$\mathbf{\Sigma}^{-1}(i,i) = \begin{cases} \frac{1}{\mathbf{\Sigma}(i,i)+\lambda} & \text{if } \mathbf{\Sigma}(i,i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

MATLAB EXPERIMENTS

MATLAB EXPERIMENTS

Supervised Learning

- The use of unsupervised techniques to determine the basis function parameters is not in general an optimal procedure in so far as the subsequent supervised training is concerned
- Indeed with unsupervided techniques the setting up of the basis functions takes no account of the target labels
- In order to obtain best result we should include the target data in the training procedure, that is we should perform supervised training



Supervised Learning

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj} exp\left(-\frac{\| \mathbf{x} - \boldsymbol{\mu}_j \|}{2\sigma_j^2}\right) + w_{k0}$$

- To find the parameters $\boldsymbol{\mu}_j$ and σ_j we should minimize the error function (e.g. sum-of-squares) with respect to these parameters
- This can be done by deriving the error function with respect to $\boldsymbol{\mu}_j$ and σ_j and make use of these derivatives in the *Gradient descent* optimization algorithm.

Supervised Learning

$$\frac{\partial E}{\partial \sigma_j} = \sum_n \sum_k \{y_k(\mathbf{x}^n) - t_k^n\} w_{kj} exp\left(-\frac{\| \mathbf{x}^n - \boldsymbol{\mu}_j \|^2}{2\sigma_j^2}\right) \frac{\| \mathbf{x}^n - \boldsymbol{\mu}_j \|^2}{\sigma_j^3}$$

$$\frac{\partial E}{\partial \mu_{ji}} = \sum_n \sum_k \{y_k(\mathbf{x}^n) - t_k^n\} w_{kj} exp\left(-\frac{\| \mathbf{x}^n - \boldsymbol{\mu}_j \|^2}{2\sigma_j^2}\right) \frac{(x_i^n - \mu_{ji})}{\sigma_j^2}$$

Gradient descent

- Differentiable error function E depending on parameters $\Theta = (\theta_1, \ldots, \theta_S)$

1. We began with some initial guess for Θ (e.g. random)
2. We update the parameters by moving a small distance in the Θ-space in the direction in which E decrease most rapidly $(-\nabla_\Theta E)$

$$\theta_j^{(\tau+1)} = \theta_j^{(\tau)} - \eta \left. \frac{\partial E}{\partial \theta_j} \right|_{\Theta^{(\tau)}}$$

- where η is called *learning rate* and is usually taken in the range $[0 \ldots 1]$

Model complexity and PCA

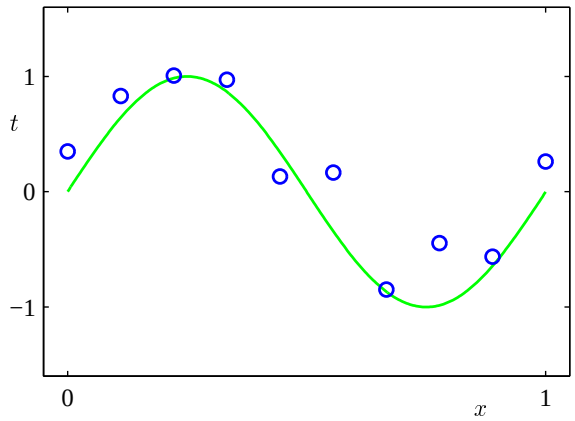
April 5, 2011

Outline

A simple regression problem

We introduce some key concept by means of a toy example of regression problem

- In this toy example we know the regression function $u(x) = \sin(2\pi x)$
- The Training Set comprises 10 input data points $\{x_i\}_{i=1}^{10}$ spaced uniformly in the range $[0, 1]$ with the corresponding target value $\{t_i = u(x_i) + \epsilon_i\}_{i=1}^{10}$ where ϵ_i is small random noise (drawn from a gaussian distribution)



A simple regression problem

To approximate the "unknown" function $u(x)$ by means of the function $y(x, \mathbf{w})$:

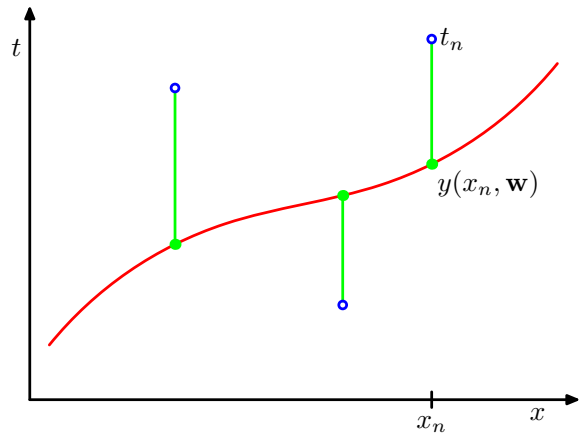
- choose the function *model* (linear, polynomial, neural network, ...)
- determine the parameters $\mathbf{w}$ of the model by the *learning* algorithm (usually this step involve):
  - choosing of an *error function;*
    - for a given value of $\tilde{\mathbf{w}}$ the error function measures the misfit between the function $y(\mathbf{x}, \tilde{\mathbf{w}})$ and the training data;
  - *Learning Algorithm*: is the procedure able to select the parameters $\mathbf{w}$ that minimizing the error function.

A first model: Polynomial

- Polynomial Model:

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j \text{ with } M \in \mathbb{N}$$

- Sum of squares Error Function: $E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$



- Thus we have to select the value of M and then we have to determine the values of the parameters $\mathbf{w}$.

A second model: Neural Network

- in the case of single input and output, identity output function we can write:

$$y(x, \mathbf{w}) = \sum_{j=0}^{M} w_j^{(2)} \phi_j(x) \text{ with } M \in \mathbb{N}$$

$$\phi_j(x) = g\left(w_j^{(1)} x\right)$$

where $g(\cdot)$ is some non linear function (e.g. sigmoid, radial basis function RBF networks)

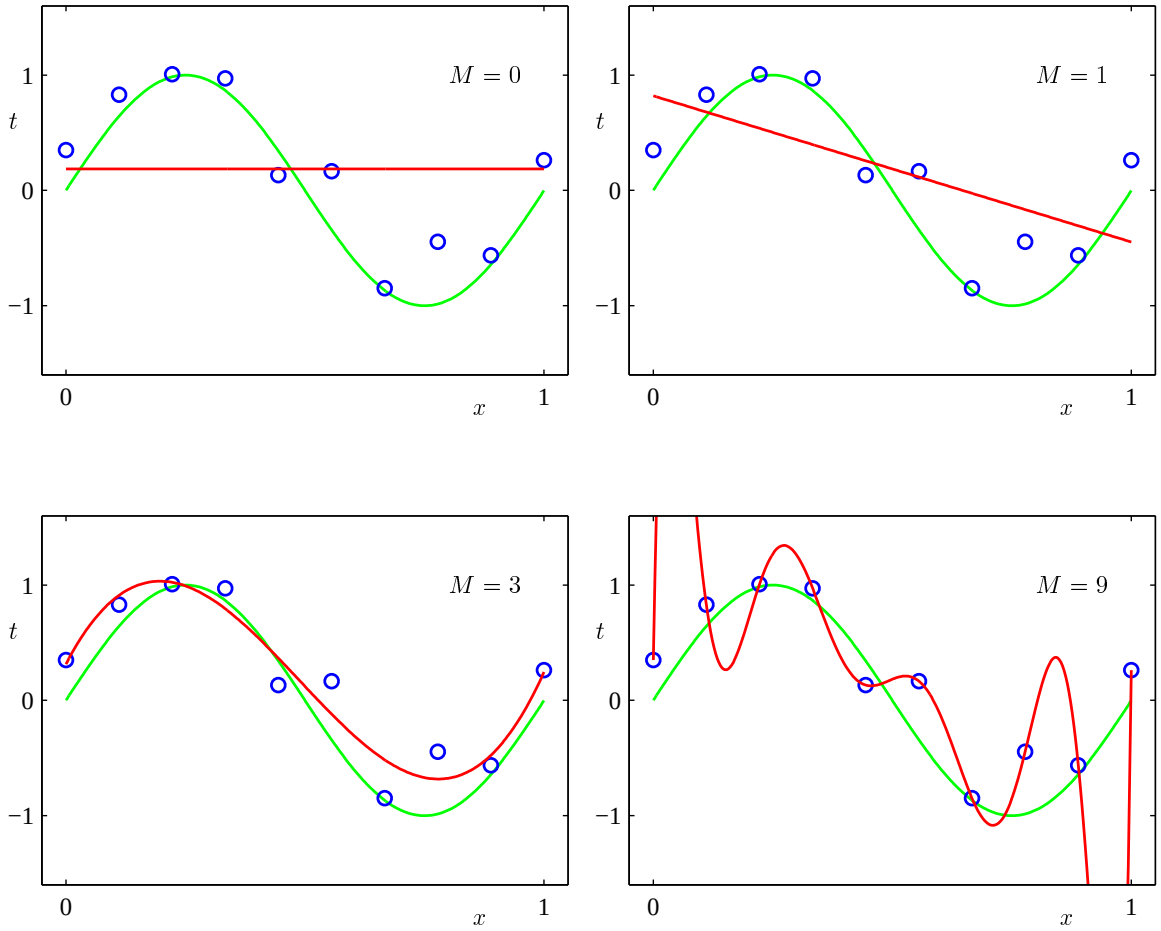
- Thus we have to select the value of M and then we have to determine the values of the parameters $\mathbf{w}$.

Outline

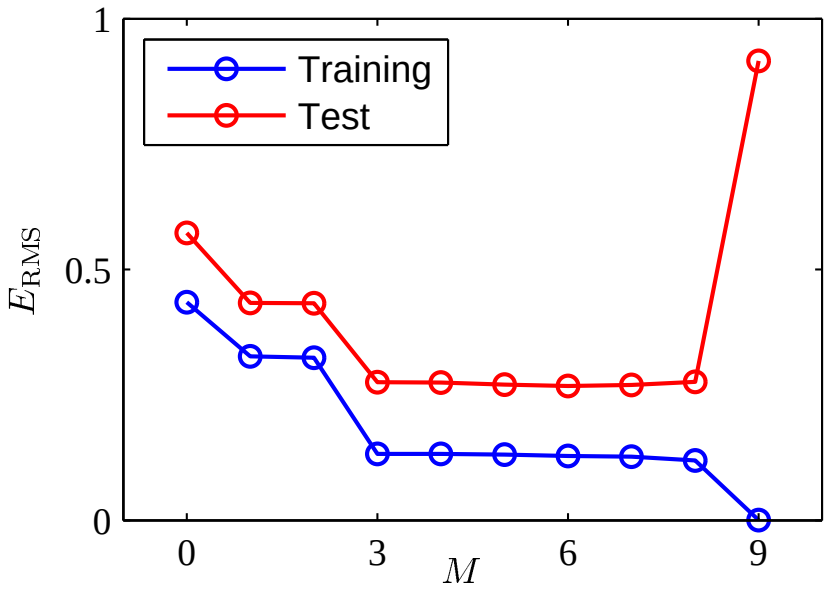
Determining the parameters **w**

- Choosen the function model (polinomial, neural network, and so on), chosen the related value of M techniques exist to determine the parameters values $\mathbf{w}$:
  - **Maximum Likelihood**;
  - Bayesian approach;
  - ...
- We indicate with $\mathbf{w}^*$ the values of the parameters that minimize the error function for a given model (in our case identified by the value of M)

What does it happens when we change the model complexity ?



The Over-fitting Problem



- Measuring the generalization performance on the *Test Set*
- *Root Mean Square* error function $E_{RMS} = \sqrt{2E(\mathbf{w}^m)/N}$
- We choose M that give the best generalization performance (that is the minumum error on the test set)

What does it happens to **w** when we change the model complexity ?

	$M=0$	$M=1$	$M=6$	$M=9$
w_0^*	0.19	0.28	0.31	0.35
w_1^*		−1.27	7.99	232.37
w_2^*			−25.43	−5321.83
w_3^*			17.37	45868.31
w_4^*				−231639.30
w_5^*				640042.26
w_6^*				−1061800.52
w_7^*				1042400.18
w_8^*				−557682.99
w_9^*				125201.43

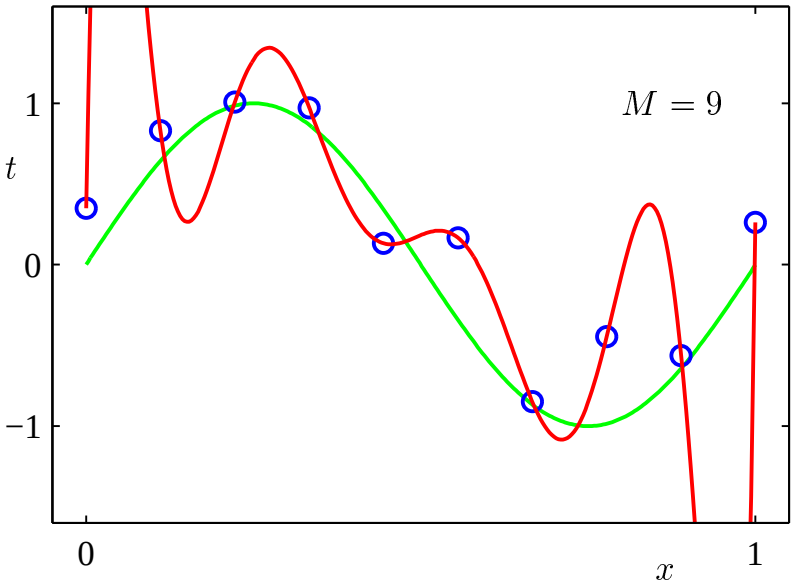
How do we select the model complexity ?

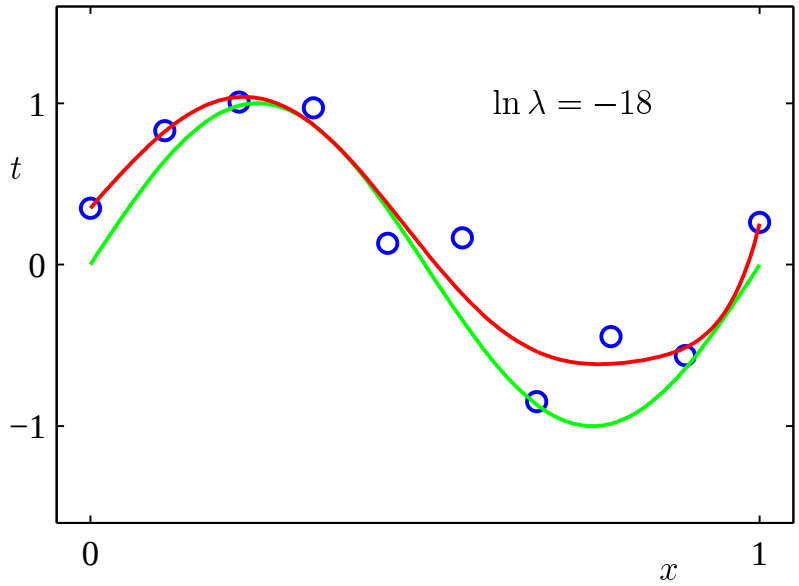
- Choosing a very simple model may give rise to poor results ! (e.g. $M = 1$ we are using a linear model)
- Choosing a very complex model may give rise to over-fitting and thus poor generalization performace !
- One technique that is often used to control over-fitting is to still use a complex model but to add a penalty term to the error function in order to discourage the coefficients from reaching large values (*regularization*):

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x^n;\mathbf{w}) - t^n\}^2 + \frac{\lambda}{2} \parallel \mathbf{w} \parallel^2$$

where λ is called regularization coefficient

Using Regularization

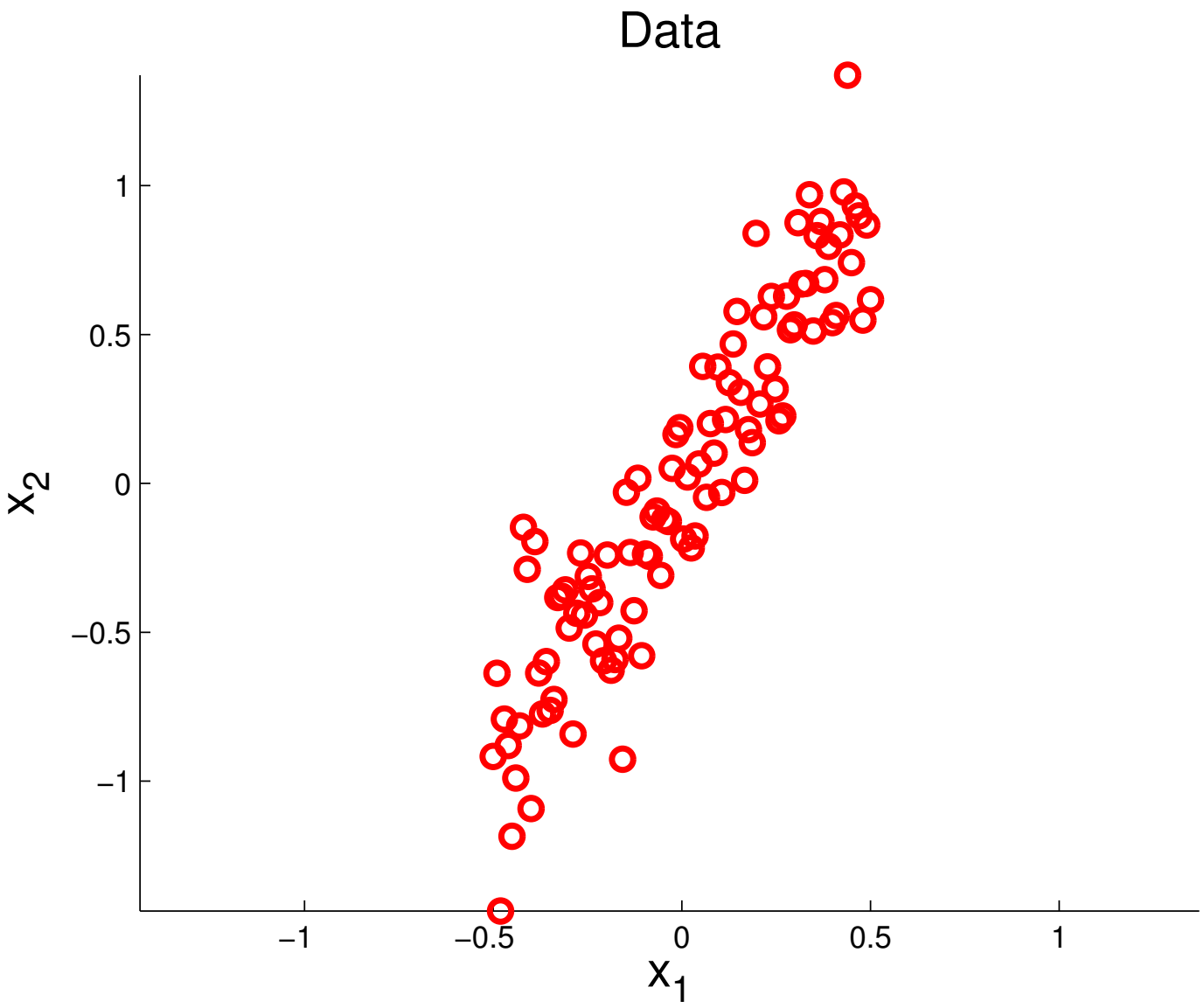




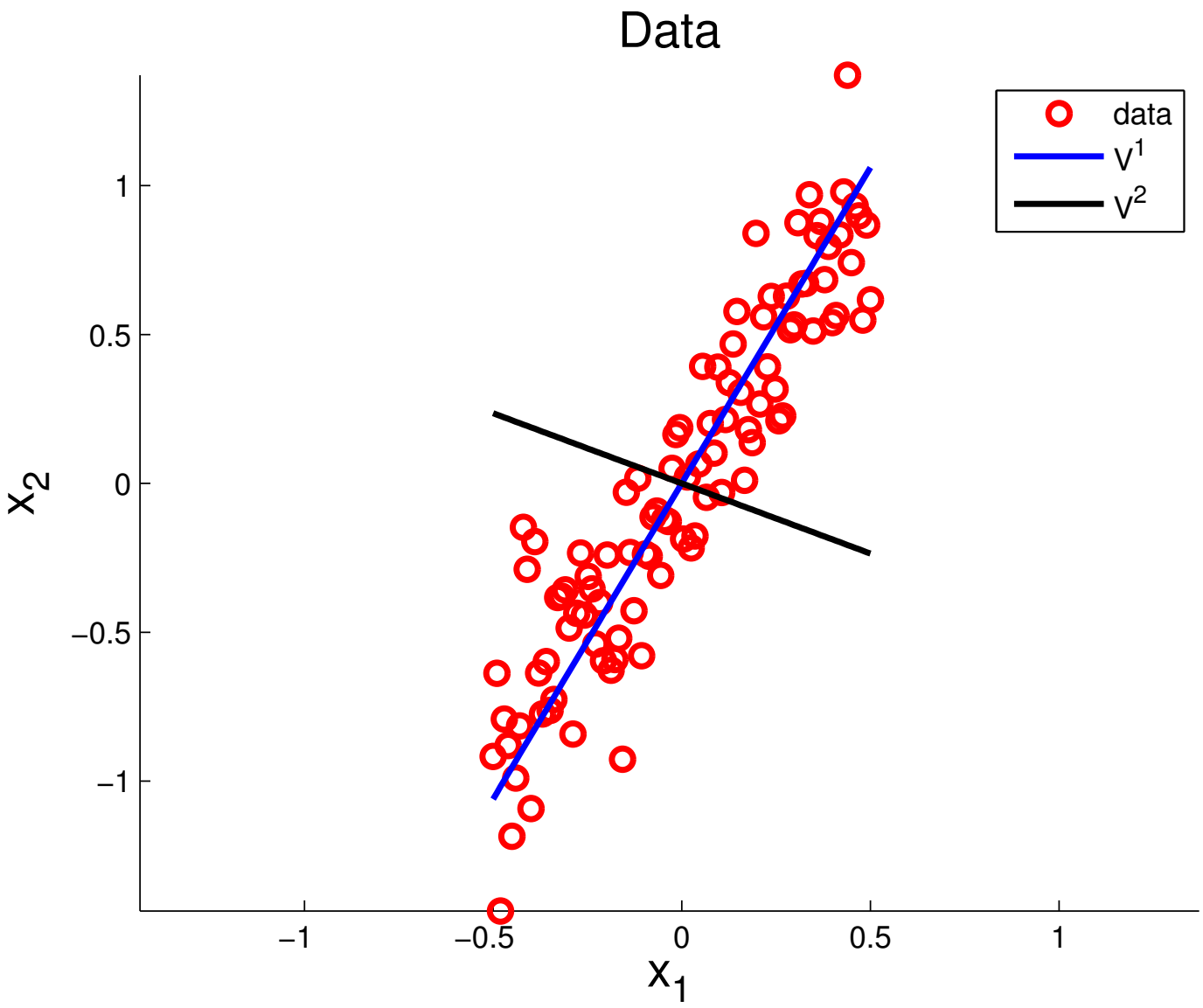
MATLAB EXPERIMENTS

MATLAB EXPERIMENTS

Dimensionality Reduction: 2D example

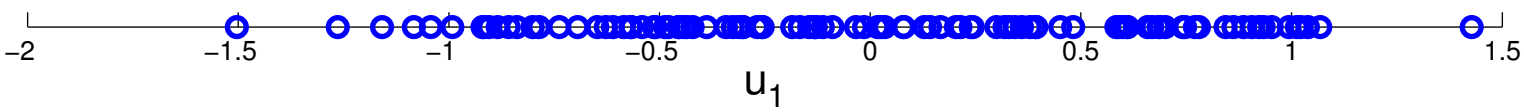


Dimensionality Reduction: 2D example

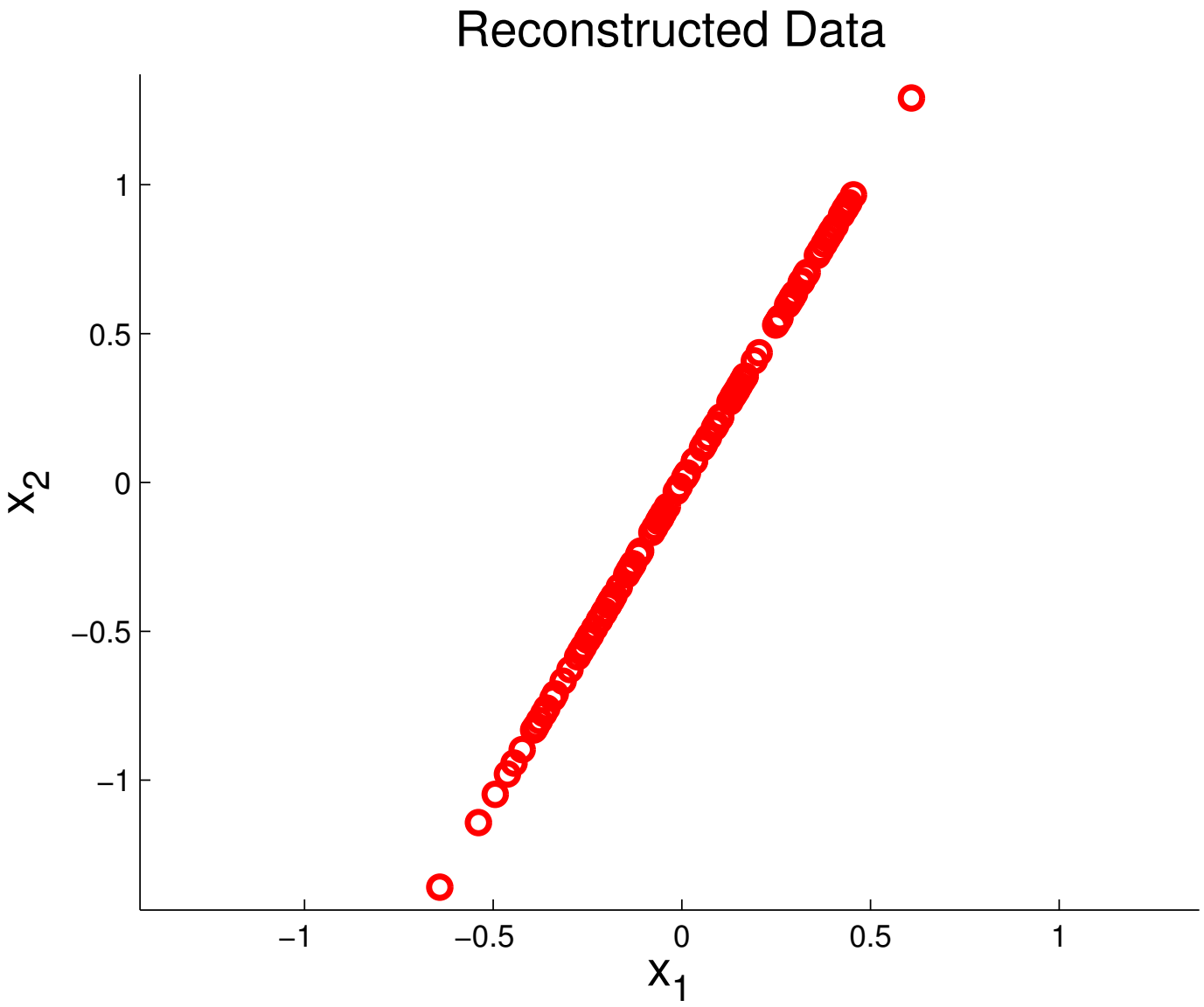


Dimensionality Reduction: 2D example

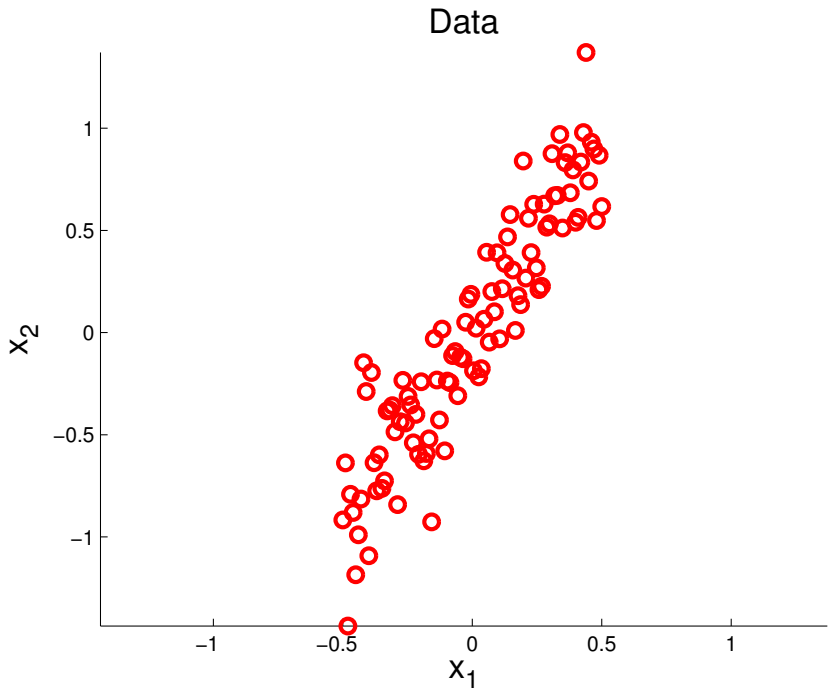
Projected data onto $\mathbf{V}^1$

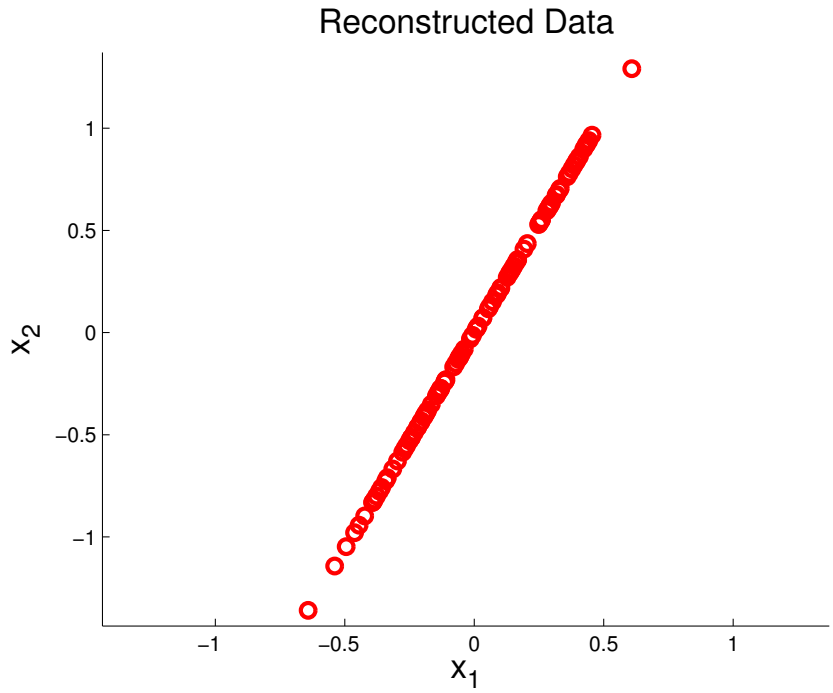


Dimensionality Reduction: 2D example



Dimensionality Reduction: 2D example





- Reconstruction error: 0.18 (RMS error)

Principal Component Analysis

- Consider a data set of N observations $\{\mathbf{x}^n\}_{n=1,\ldots,N}$ where $\mathbf{x}^n \in \mathbb{R}^d$
- Objective: to project the data onto a $k < d$ dimensional space while maximizing the variance of the projected data
- $k = 1$, vector $\mathbf{u} \in \mathbb{R}^d$
- $\mathbf{u}\mathbf{x}^n$ projection of the n-th point
- $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}^n$ sample mean of the data
- $\mathbf{u}\bar{\mathbf{x}}$ projected sample mean

Principal Component Analysis

- Variance of projected data:

$$\frac{1}{N}\sum_{n=1}^{N}\left(\mathbf{u}^T\mathbf{x}^n - \mathbf{u}^T\bar{\mathbf{x}}\right)^2 = \mathbf{u}^T S\mathbf{u}$$

- where S is:

$$S = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}^n - \bar{\mathbf{x}})(\mathbf{x}^n - \bar{\mathbf{x}})$$

- Optimization problem:

$$\arg\max_{\mathbf{u}} \mathbf{u}^T S\mathbf{u} \quad s.t. \quad \| \mathbf{u} \|^2 = 1$$

Principal Component Analysis

- Using Lagrange multiplier:

$$\mathbf{u}^T S\mathbf{u} + \lambda(1 - \mathbf{u}^T\mathbf{u})$$

- setting the deriving with respect to $\mathbf{u}$ equal to zero (Eigenvalue problem):

$$S\mathbf{u} = \lambda\mathbf{u}$$

- multiplying both sides by $\mathbf{u}^T$ (note that $\mathbf{u}^T\mathbf{u} = 1$)

$$\mathbf{u}^T S\mathbf{u} = \lambda$$

So the variance is maximized when we choose u equal to the eigenvector having the largest eigenvalue λ of S

PCA Algorithm

1: **function** PCA(X, k)
2: $X_{mean} \leftarrow$computeMean(X)
3: $X_c \leftarrow X - X_{mean}$ ▷ Centering data
4: $Cov_X \leftarrow X_c^T X_c$ ▷ Computing covariance matrix
5: $[U\ Lambda] \leftarrow$diagonalize(Cov_X) ▷ Eigenvectors, eigenvalues
6: $U \leftarrow$sortDescending($U, Lambda$) ▷ Sorting eigenvectors
7: $U_k \leftarrow$getFirstKcomponents(U, k) ▷ First k eigenvectors
8: $Y \leftarrow X_c U_k$ ▷ Projecting data
9: **return** Y
10: **end function**