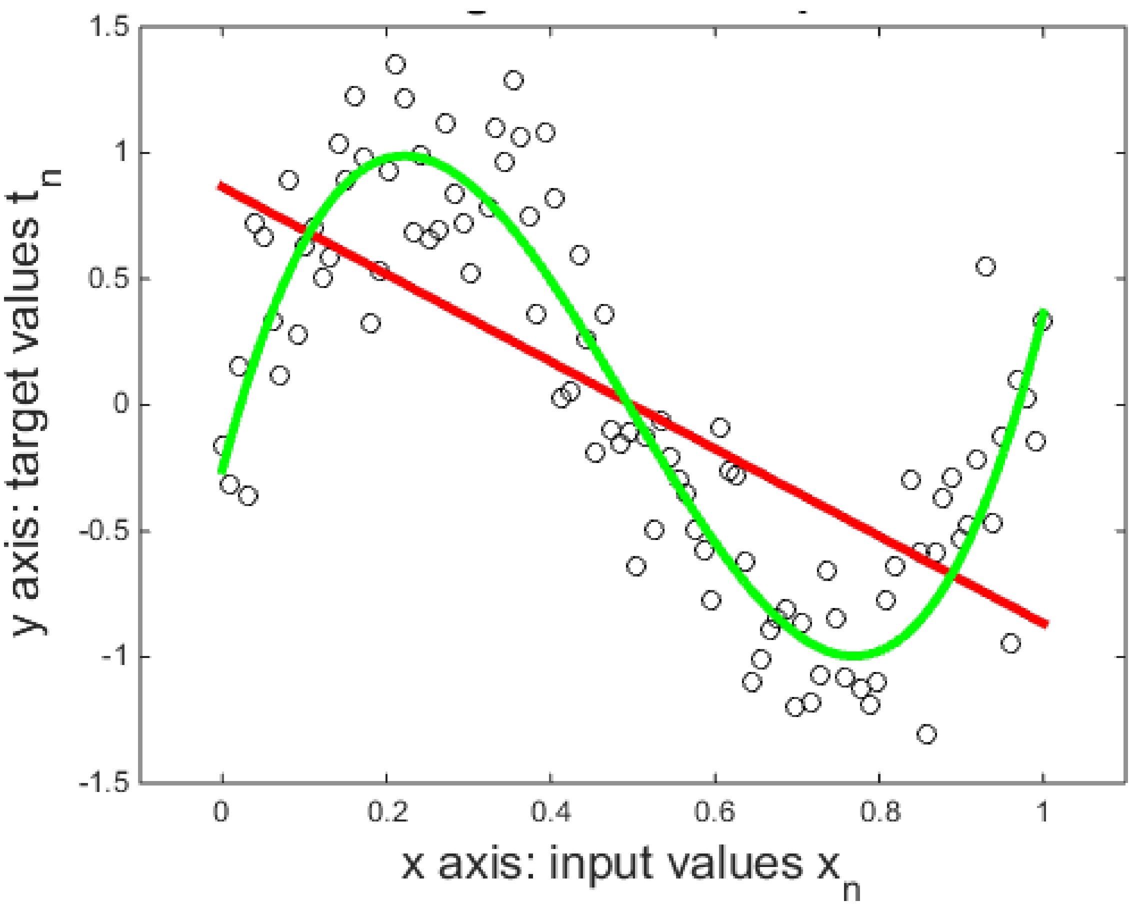
Linear Models for Regression

CSE 4309 – Machine Learning
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

The Regression Problem

- Training data: A set of input-output pairs: $\{(x_n, t_n)\}$
  - x_n is the n-th training input.
  - t_n is the target output for x_n.
- Goal: learn a function y(x), that can predict the target value *t* for a new input *x*.
- So far, this is the standard definition of a generic supervised learning problem.
- What differentiates regression problems is that the target outputs come from a **continuous** space.

A Regression Example



- circles:
  training data.
- red curve:
  one possible solution: a line.
- green curve:
  another possible solution: a cubic polynomial.

Linear Models for Regression

$$y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j \varphi_j(x)$$

- Functions φ_j are called **basis functions**.
  - You must decide what these functions should be, before you start training.
  - They can be any functions you want.
- Parameters w_j are **weights**. They are real numbers.
  - The goal of linear regression is to estimate these weights.
  - The output of training is the values of these weights.

The Dummy φ_0 Function

$$y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j \varphi_j(x)$$

- To simplify notation, we define a "dummy" basis function $\varphi_0(x) = 1$.
- Then, the above formula becomes:

$$y(x, w) = \sum_{j=0}^{M-1} w_j \varphi_j(x)$$

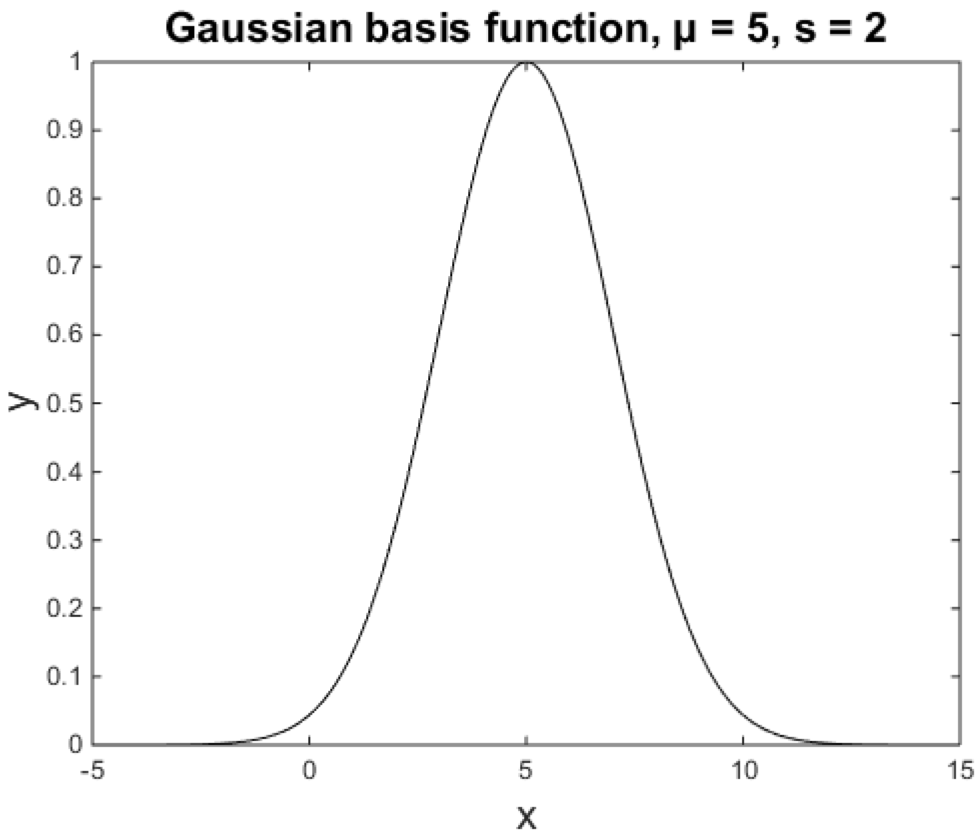
Dot Product Version

$$y(x, w) = \sum_{j=0}^{M-1} w_j \varphi_j(x) = \boldsymbol{w}^T \varphi(x)$$

- **w** is a **column vector** of weights: $\mathrm{w} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_{M-1} \end{bmatrix}$
- $\boldsymbol{w}^T$ is the transpose of **w**: $\boldsymbol{w}^T = [w_0, w_1, \dots, w_{M-1}]$.
- $\varphi(x)$ is a column vector: $\varphi(x) = \begin{bmatrix} \varphi_0(x) \\ \varphi_1(x) \\ \dots \\ \varphi_{M-1}(x) \end{bmatrix}$

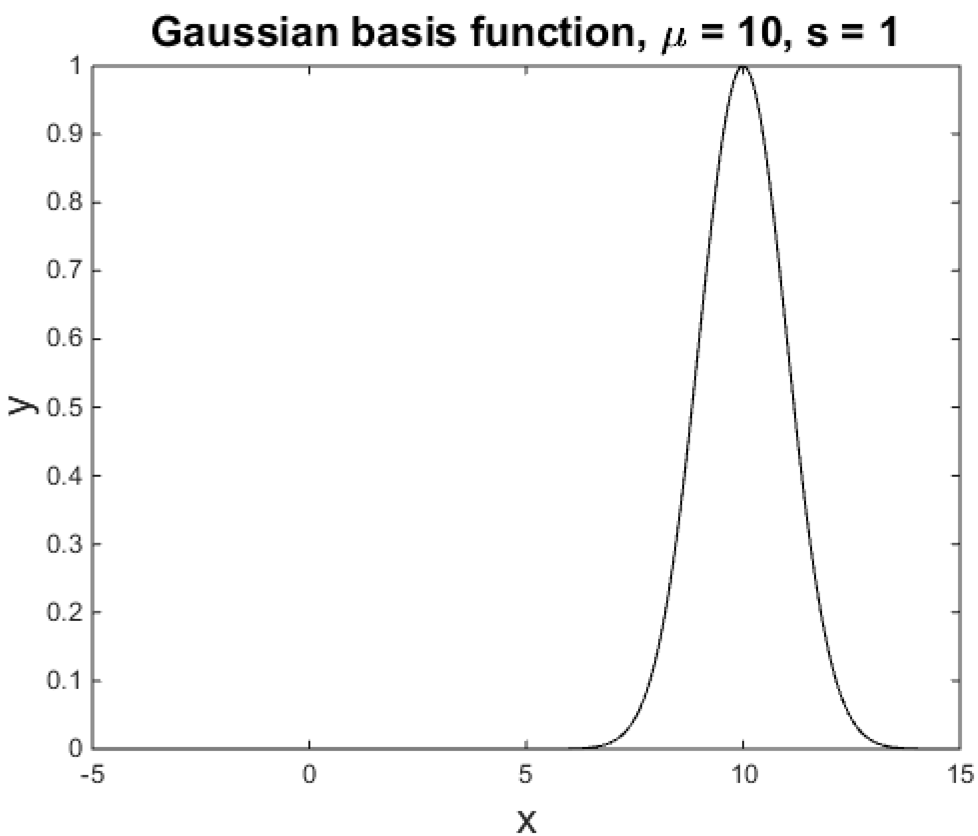
Common Choices for Basis Functions

- Gaussian basis functions: $\varphi_j(x) = e^{-\frac{(x-\mu_j)^2}{2s^2}}$



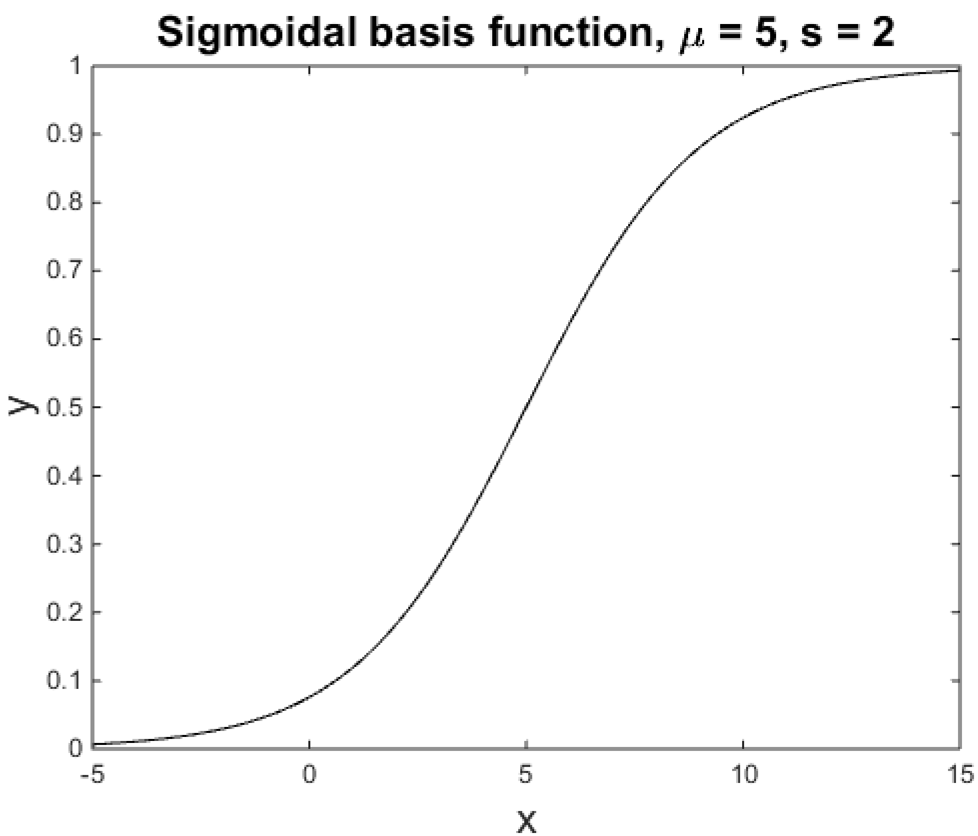
Common Choices for Basis Functions

- Gaussian basis functions: $\varphi_j(x) = e^{-\frac{(x-\mu_j)^2}{2s^2}}$



Common Choices for Basis Functions

- Sigmoidal basis functions: $\varphi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$

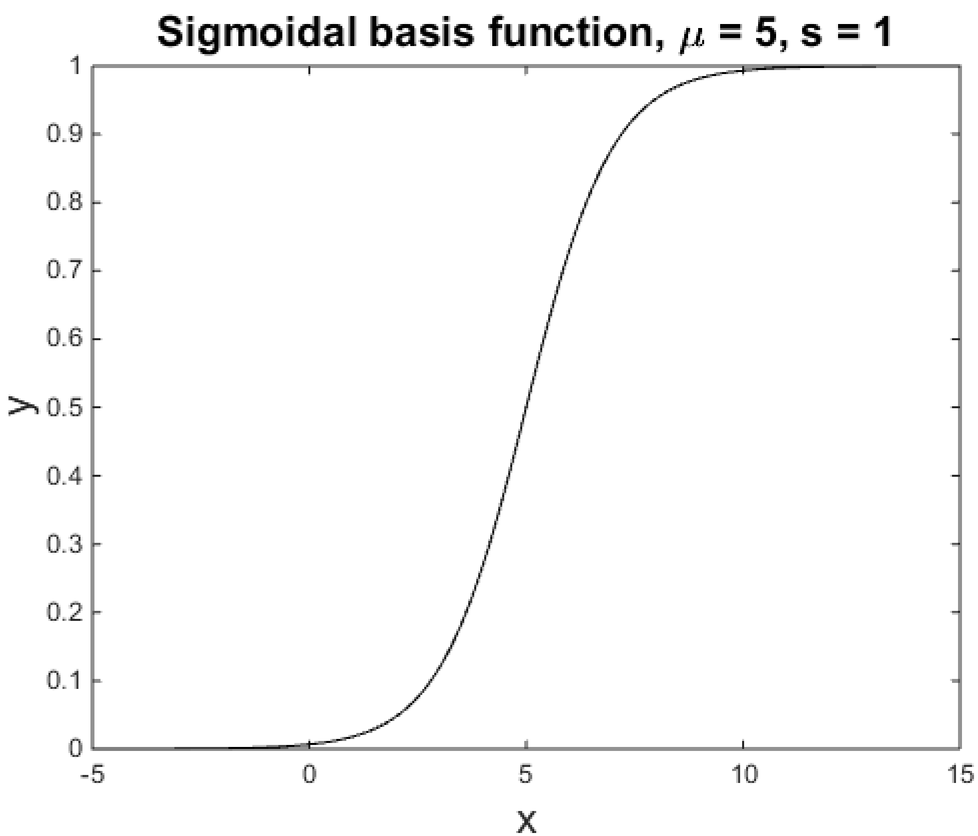


$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

$\sigma(a)$ is called the **logistic sigmoid** function. We will see it again in neural networks.

Common Choices for Basis Functions

- Sigmoidal basis functions: $\varphi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$



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

$\sigma(a)$ is called the **logistic sigmoid** function. We will see it again in neural networks.

Common Choices for Basis Functions

- Polynomial basis functions.
- For example: powers of x: $\varphi_j(x) = x^j$
- If the basis functions are powers of x, then the regression process fits a polynomial to the data.
- In other words, the regression process estimates the parameters w_j of a polynomial of degree M-1:

$$y(x, \boldsymbol{w}) = \sum_{j=0}^{M-1} w_j x^j$$

Global Versus Local Functions

- Polynomial basis functions are **global functions**.
  - Their values are far from zero for most of the input range from $-\infty$ to $+\infty$.
  - Changing a single weight w_j affects the output y(x, w) in the entire input range.
- Gaussian functions are **local functions**.
  - Their values are practically (but not mathematically) zero for most of the input range from $-\infty$ to $+\infty$.
  - Changing a single weight w_j practically does not affect the output y(x, w) except in a specific small interval.
- It is often easier to fit data with local basis functions.
  - Each basis function fits a small region of the input space.

Linear Versus Nonlinear Functions

- A linear function y(x, **w**) produces an output that depends linearly on **both** x and **w**.
- Note that polynomial, Gaussian, and sigmoidal basis functions are **nonlinear**.
- If we use nonlinear basis functions, then the regression process produces a function y(x, **w**) which is:
  - Linear to **w**.
  - Nonlinear to x.
- It is important to remember: <u>**linear regression can be used to estimate nonlinear functions of x**</u>.
  - It is called **linear** regression because y is linear to **w**, NOT because y is linear to x.

Solving Regression Problems

- There are different methods for solving regression problems.
- We will study two approaches:
  - Least squares: find the weights **w** that minimize the squared error.
  - Regularized least squares: find the weights **w** that minimize the squared error, using some hand-picked regularization parameter λ.

The Gaussian Noise Assumption

- Suppose that we want to find the **most likely** solution.
  - We want to find the weights **w** that maximize the likelihood of the training data.
- In order to do that, we need to make an additional assumption about the process that generates outputs based on inputs.
- A common approach is to assume a Gaussian noise model.

$$t = y(x, w) + \varepsilon$$

- In words, t is generated by computing y(x, **w**) and then adding some noise.
- The noise ε is a random variable from a zero-mean Gaussian distribution.

The Gaussian Noise Assumption

$$t = y(x, \boldsymbol{w}) + \varepsilon$$

- The noise ε is a random variable from a zero-mean Gaussian distribution.
- Therefore: $p(t|x, \boldsymbol{w}, \beta) = \mathcal{N}(t|y(x, \boldsymbol{w}), \beta^{-1})$
  - In the above equation, β is called the **precision** of the Gaussian, and is defined as the inverse of the variance: $\beta = \frac{1}{\sigma^2}$
  - The likelihood that input x leads to output t is a Gaussian, whose mean is y(x, **w**), and its variance is β^{-1}.

Finding the Most Likely Solution

- What is the value of **w** that is most likely given the data?
- Suppose we have a set of training inputs: $X = \{x_1, \dots, x_N\}$
- We also have a set of corresponding outputs: $\boldsymbol{t} = \{t_1, \dots, t_N\}$
- We assume that training inputs are independent of each other.
- We assume that outputs are conditionally independent of each other, given their inputs and noise parameter β.
- Then:

$$p(w|X, \beta, \boldsymbol{t}) = \frac{p(\boldsymbol{t}|X, \boldsymbol{w}, \beta) * p(\boldsymbol{w}|X, \beta)}{p(\boldsymbol{t}|X, \beta)}$$

Finding the Most Likely Solution

$$p(\boldsymbol{w}|X, \beta, \boldsymbol{t}) = \frac{p(\boldsymbol{t}|X, \boldsymbol{w}, \beta) * p(\boldsymbol{w}|X, \beta)}{p(\boldsymbol{t}|X, \beta)}$$

- We assume that, given X and β, all values of **w** are equally likely.
- Then, $\frac{p(\boldsymbol{w}|X,\beta)}{p(\boldsymbol{t}|X, \beta)}$ is a constant that does not depend on **w**.
- Therefore, finding the **w** that maximizes $p(\boldsymbol{w}|X, \beta, \boldsymbol{t})$ is the same as finding the **w** that maximizes $p(\boldsymbol{t}|X, \boldsymbol{w}, \beta)$.
- $p(\boldsymbol{t}|X, \boldsymbol{w}, \beta)$ is the **likelihood** of the training data.
- So, to find the most likely answer **w**, we must find the value of **w** that maximizes the likelihood of the training data.

Likelihood of the Training Data

- What is the probability of the training data given **w**?
- We assume that outputs are **<u>conditionally independent</u>** of each other, given their inputs and noise parameter β.
- Then:

$$p(\boldsymbol{t}|X, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n | y(x_n, \boldsymbol{w}), \beta^{-1})$$

Likelihood of the Training Data

$$p(\boldsymbol{t}|X, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n | y(x_n, \boldsymbol{w}), \beta^{-1})$$

- Remember that, using dot product notation:
  $\mathrm{y}(x_n, \boldsymbol{w}) = \boldsymbol{w}^T \varphi(x_n)$
- Therefore:

$$p(\boldsymbol{t}|X, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n | \boldsymbol{w}^T \varphi(x_n), \beta^{-1})$$

- Our goal is to find the weights w that maximize $p(\boldsymbol{t}|X, \boldsymbol{w}, \beta)$.

Log Likelihood of the Training Data

$$p(\boldsymbol{t}|X, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\boldsymbol{w}^T \varphi(x_n), \beta^{-1})$$

- Maximizing $p(\boldsymbol{t}|X, \boldsymbol{w}, \beta)$ is the same as maximizing $\ln(p(\boldsymbol{t}|X, w, \beta))$, where ln is the natural logarithm.

$$\ln\big(p(\boldsymbol{t}|X, \boldsymbol{w}, \beta)\big) = \sum_{n=1}^{N} \ln(\mathcal{N}(t_n|\boldsymbol{w}^T \varphi(x_n), \beta^{-1}))$$

Log Likelihood of the Training Data

$$\ln\left(p(\boldsymbol{t}|X, \boldsymbol{w}, \beta)\right) = \sum_{n=1}^{N} \ln(\mathcal{N}(t_n|\boldsymbol{w}^T \varphi(x_n), \beta^{-1}))$$

$$= \sum_{n=1}^{N} \ln\left(\frac{1}{\sqrt{\beta^{-1} 2\pi}}\, e^{-\frac{\left(t_n - \boldsymbol{w}^T \varphi(x_n)\right)^2}{2\beta^{-1}}}\right)$$

$$= \sum_{n=1}^{N} \ln\left(\frac{\sqrt{\beta}}{\sqrt{2\pi}}\, e^{-\frac{\beta\left(t_n - \boldsymbol{w}^T \varphi(x_n)\right)^2}{2}}\right)$$

$$= \sum_{n=1}^{N} \ln\left(\frac{\sqrt{\beta}}{\sqrt{2\pi}}\right) + \sum_{n=1}^{N} \ln\left(e^{-\frac{\beta\left(t_n - \boldsymbol{w}^T \varphi(x_n)\right)^2}{2}}\right)$$

Log Likelihood of the Training Data

$$\ln\big(p(\boldsymbol{t}|X,\boldsymbol{w},\beta)\big)$$
$$= \sum_{n=1}^{N} \ln\left(\frac{\sqrt{\beta}}{\sqrt{2\pi}}\right) + \sum_{n=1}^{N} \ln\left(e^{-\frac{\beta\left(t_n - \boldsymbol{w}^T\varphi(x_n)\right)^2}{2}} \right)$$

$$= \mathrm{N}\ln\left(\frac{\sqrt{\beta}}{\sqrt{2\pi}}\right) + \sum_{n=1}^{N} -\frac{\beta(t_n - \boldsymbol{w}^T\varphi(x_n))^2}{2}$$

- Note that $\mathrm{N}\ln\left(\frac{\sqrt{\beta}}{\sqrt{2\pi}}\right)$ is independent of **w**.
- Therefore, to maximize $\ln\big(p(\boldsymbol{t}|X,\boldsymbol{w},\beta)\big)$ we must maximize

$$\textstyle\sum_{n=1}^{N} - \frac{\beta\left(t_n - \boldsymbol{w}^T\varphi(x_n)\right)^2}{2}$$

Log Likelihood and Sum-of-Squares Error

- To maximize $\ln\big(p(\boldsymbol{t}|X, \boldsymbol{w}, \beta)\big)$ we must maximize:

$$\sum_{n=1}^{N} -\frac{\beta(t_n - \boldsymbol{w}^T\varphi(x_n))^2}{2}$$

- Remember that the sum-of-squares error is defined in the textbook as: $E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - \boldsymbol{w}^T\varphi(x_n))^2$
- Therefore, we want to **maximize** $-\beta E_D(\boldsymbol{w})$, which is the same as saying that we want to **minimize** $E_D(\boldsymbol{w})$.
- Therefore, we have proven that: **the w that maximizes the likelihood of the training data is the same w that minimizes the sum-of-squares error**.

Minimizing Sum-of-Squares Error

- We want to find the w that minimizes:

$$E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[(t_n - \boldsymbol{w}^T\varphi(x_n))^2]$$

- $E_D(\boldsymbol{w})$ is a function mapping an M-dimensional vector to a real number.
- Remember from calculus: to minimize any such function:
  - Compute the gradient vector $\nabla E_D(\boldsymbol{w})$.
    This gradient is an M-dimensional **row** vector.
  - Finding values of **w** that solve equation $\nabla E_D(\boldsymbol{w}) = 0$.
  - These values of **w** can be possible maxima or minima.

Minimizing Sum-of-Squares Error

- We want to find the w that minimizes:

$$E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[(t_n - \boldsymbol{w}^T\varphi(x_n))^2]$$

- To calculate $\nabla E_d(\boldsymbol{w})$: First, let's calculate the gradient $\nabla E_{D,n}(\boldsymbol{w})$ of function:

$$E_{D,n}(\boldsymbol{w}) = (t_n - \boldsymbol{w}^T\varphi(x_n))^2$$

- $E_{D,n}(\boldsymbol{w})$ is the composition $f \circ g(\boldsymbol{w})$ of:

$$f(x) = x^2$$
$$g(\boldsymbol{w}) = t_n - \boldsymbol{w}^T\varphi(x_n)$$

- According to the chain rule: $(f \circ g)' = f'(g) * g'$.

Minimizing Sum-of-Squares Error

- $E_{D,n}(\boldsymbol{w}) = (t_n - \boldsymbol{w}^T \varphi(x_n))^2$

- $E_{D,n}(\boldsymbol{w})$ is the composition $f \circ g(\boldsymbol{w})$ of:

$$f(x) = x^2$$
$$g(\boldsymbol{w}) = t_n - \boldsymbol{w}^T \varphi(x_n)$$

- According to the chain rule: $(f \circ g)' = f'(g) * g'$

$$f'(x) = 2x$$

$$g'(\boldsymbol{w}) = -\varphi(x_n)^T$$

$$f'(g(\boldsymbol{w})) * g'(\boldsymbol{w}) = 2\big(t_n - \boldsymbol{w}^T \varphi(x_n)\big) * (-\varphi(x_n)^T)$$

Minimizing Sum-of-Squares Error

- $E_{D,n}(\boldsymbol{w}) = (t_n - \boldsymbol{w}^T \varphi(x_n))^2$
- $\nabla E_{D,n}(\boldsymbol{w}) = -2\left(t_n - \boldsymbol{w}^T \varphi(x_n)\right) * \varphi(x_n)^T$
- $E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[(t_n - \boldsymbol{w}^T \varphi(x_n))^2] = \frac{1}{2}\sum_{n=1}^{N} E_{D,n}(\boldsymbol{w})$
- Therefore:

$$\nabla E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N} \nabla E_{D,n}(\boldsymbol{w})$$

$$= \frac{1}{2}\sum_{n=1}^{N}\left[-2\left(t_n - \boldsymbol{w}^T \varphi(x_n)\right) * \varphi(x_n)^T\right]$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left[-2\big(t_n - \boldsymbol{w}^T\varphi(x_n)\big) * \varphi(x_n)^T\right]$$

$$= \sum_{n=1}^{N}[(\boldsymbol{w}^T\varphi(x_n) - t_n) * \varphi(x_n)^T]$$

$$= \left(\boldsymbol{w}^T\sum_{n=1}^{N}(\varphi(x_n)\varphi(x_n)^T)\right) - \left(\sum_{n=1}^{N}(t_n\varphi(x_n)^T)\right)$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = \left(\boldsymbol{w}^T \sum_{n=1}^{N} (\varphi(x_n)\varphi(x_n)^T) \right) - \left(\sum_{n=1}^{N} (t_n \varphi(x_n)^T) \right)$$

- We want to solve equation $\nabla E_D(\boldsymbol{w}) = 0$.
- We can simplify expressions using vector and matrix notation:

$$\Phi = \begin{bmatrix} \varphi_0(x_1), \dots, \varphi_{M-1}(x_1) \\ \varphi_0(x_2), \dots, \varphi_{M-1}(x_2) \\ \dots \\ \varphi_0(x_N), \dots, \varphi_{M-1}(x_N) \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \dots \\ t_N \end{bmatrix}$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = \left(\boldsymbol{w}^T \sum_{n=1}^{N} \left(\varphi(x_n)\varphi(x_n)^T \right) \right) - \left(\sum_{n=1}^{N} \left(t_n \varphi(x_n)^T \right) \right)$$

- We want to solve equation $\nabla E_D(\boldsymbol{w}) = 0$.
- We can simplify expressions using vector and matrix notation:

$$\boldsymbol{\Phi} = \begin{bmatrix} \varphi_0(x_1), \ldots, \varphi_{M-1}(x_1) \\ \varphi_0(x_2), \ldots, \varphi_{M-1}(x_2) \\ \ldots \\ \varphi_0(x_N), \ldots, \varphi_{M-1}(x_N) \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \ldots \\ t_N \end{bmatrix}$$

Minimizing Sum-of-Squares Error

$$\sum_{n=1}^{N}(\varphi(x_n)\varphi(x_n)^T) = \sum_{n=1}^{N}\left(\begin{bmatrix}\varphi_0(x_n)\\ \dots\\ \varphi_{M-1}(x_n)\end{bmatrix} * [\varphi_0(x_n), \dots, \varphi_{M-1}(x_n)]\right)$$

$$= \sum_{n=1}^{N}\left(\begin{bmatrix}\varphi_0(x_n)^2, \varphi_0(x_n)\varphi_1(x_n), \dots, \varphi_0(x_n)\varphi_{M-1}(x_n)\\ \varphi_0(x_n)\varphi_1(x_n), \varphi_1(x_n)^2, \dots, \varphi_1(x_n)\varphi_{M-1}(x_n)\\ \dots\\ \varphi_0(x_n)\varphi_{M-1}(x_n), \varphi_1(x_n)\varphi_{M-1}(x_n), \dots, \varphi_{M-1}(x_n)^2\end{bmatrix}\right)$$

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

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = \left(\boldsymbol{w}^T \sum_{n=1}^{N} \left(\varphi(x_n)\varphi(x_n)^T\right)\right) - \left(\sum_{n=1}^{N} \left(t_n \varphi(x_n)^T\right)\right)$$

$$\boldsymbol{\Phi} = \begin{bmatrix} \varphi_0(x_1), \ldots, \varphi_{M-1}(x_1) \\ \varphi_0(x_2), \ldots, \varphi_{M-1}(x_2) \\ \ldots \\ \varphi_0(x_N), \ldots, \varphi_{M-1}(x_N) \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \ldots \\ t_N \end{bmatrix}$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = (\boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi}) - \left(\sum_{n=1}^{N} (t_n \varphi(x_n)^T) \right)$$

$$\boldsymbol{\Phi} = \begin{bmatrix} \varphi_0(x_1), \dots, \varphi_{M-1}(x_1) \\ \varphi_0(x_2), \dots, \varphi_{M-1}(x_2) \\ \dots \\ \varphi_0(x_N), \dots, \varphi_{M-1}(x_N) \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \dots \\ t_N \end{bmatrix}$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = (\boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi}) - \left(\sum_{n=1}^{N} (t_n \varphi(x_n)^T) \right)$$

$$\boldsymbol{\Phi} = \begin{bmatrix} \varphi_0(x_1), \dots, \varphi_{M-1}(x_1) \\ \varphi_0(x_2), \dots, \varphi_{M-1}(x_2) \\ \dots \\ \varphi_0(x_N), \dots, \varphi_{M-1}(x_N) \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \dots \\ t_N \end{bmatrix}$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = \left(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{\Phi}^{T} * \boldsymbol{\Phi}\right) - (\boldsymbol{t}^{\,T}\boldsymbol{\Phi})$$

$$\Phi = \begin{bmatrix} \varphi_0(x_1), \dots, \varphi_{M-1}(x_1) \\ \varphi_0(x_2), \dots, \varphi_{M-1}(x_2) \\ \dots \\ \varphi_0(x_N), \dots, \varphi_{M-1}(x_N) \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \dots \\ t_N \end{bmatrix}$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = (\boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi}) - (\boldsymbol{t}^T \boldsymbol{\Phi})$$

$$\nabla E_D(\boldsymbol{w}) = 0 \Rightarrow \boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi} = \boldsymbol{t}^T \boldsymbol{\Phi}$$

Multiplying both sides by $(\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}$

$$\Rightarrow \boldsymbol{w}^T = (\boldsymbol{t}^T \boldsymbol{\Phi}) * (\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = (\boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi}) - (\boldsymbol{t}^T \boldsymbol{\Phi})$$

$$\nabla E_D(\boldsymbol{w}) = 0 \Rightarrow \boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi} = \boldsymbol{t}^T \boldsymbol{\Phi}$$

Transposing both sides

$$\Rightarrow \boldsymbol{w}^T = (\boldsymbol{t}^T \boldsymbol{\Phi}) * (\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}$$

$$\Rightarrow \boldsymbol{w} = [(\boldsymbol{t}^T \boldsymbol{\Phi}) * (\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}]^T$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = (\boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi}) - (\boldsymbol{t}^T \boldsymbol{\Phi})$$

$$\nabla E_D(\boldsymbol{w}) = 0 \Rightarrow \boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi} = \boldsymbol{t}^T \boldsymbol{\Phi}$$

Using rule:
$(AB)^T = B^T A^T$

$$\Rightarrow \boldsymbol{w}^T = (\boldsymbol{t}^T \boldsymbol{\Phi}) * (\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}$$

$$\Rightarrow \boldsymbol{w} = [(\boldsymbol{t}^T \boldsymbol{\Phi}) * (\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}]^T$$

$$\Rightarrow \boldsymbol{w} = [(\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}]^T (\boldsymbol{\Phi}^T \mathbf{t})$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = (\boldsymbol{w}^T\boldsymbol{\Phi}^T * \boldsymbol{\Phi}) - (\boldsymbol{t}^T\boldsymbol{\Phi})$$

$$\nabla E_D(\boldsymbol{w}) = 0 \Rightarrow \boldsymbol{w}^T\boldsymbol{\Phi}^T * \boldsymbol{\Phi} = \boldsymbol{t}^T\boldsymbol{\Phi}$$

Using rule:
$(A^T)^{-1} = (A^{-1})^T$

$$\Rightarrow \boldsymbol{w}^T = (\boldsymbol{t}^T\boldsymbol{\Phi}) * (\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}$$

$$\Rightarrow \boldsymbol{w} = [(\boldsymbol{t}^T\boldsymbol{\Phi}) * (\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}]^T$$

$$\Rightarrow \boldsymbol{w} = [(\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}]^T(\boldsymbol{\Phi}^T\mathbf{t})$$

$$\Rightarrow \boldsymbol{w} = [(\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^T]^{-1}(\boldsymbol{\Phi}^T\mathbf{t})$$

Minimizing Sum-of-Squares Error

$$\nabla E_D(\boldsymbol{w}) = (\boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi}) - (\boldsymbol{t}^T \boldsymbol{\Phi})$$

$$\nabla E_D(\boldsymbol{w}) = 0 \Rightarrow \boldsymbol{w}^T \boldsymbol{\Phi}^T * \boldsymbol{\Phi} = \boldsymbol{t}^T \boldsymbol{\Phi}$$

Using rule:
$(AB)^T = B^T A^T$

$$\Rightarrow \boldsymbol{w}^T = (\boldsymbol{t}^T \boldsymbol{\Phi}) * (\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}$$

$$\Rightarrow \boldsymbol{w} = [(\boldsymbol{t}^T \boldsymbol{\Phi}) * (\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}]^T$$

$$\Rightarrow \boldsymbol{w} = [(\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^{-1}]^T (\boldsymbol{\Phi}^T \mathbf{t})$$
$$\Rightarrow \boldsymbol{w} = [(\boldsymbol{\Phi}^T * \boldsymbol{\Phi})^T]^{-1} (\boldsymbol{\Phi}^T \mathbf{t})$$
$$\Rightarrow \boldsymbol{w} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \mathbf{t}$$

Notation: $\boldsymbol{w}_{ML}$

- From the previous slides, we got the formula

$$\boldsymbol{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$

- We denote this value of **w** as $\boldsymbol{w}_{ML}$, because it is the **maximum likelihood** estimate of **w**.
  - In other words, $\boldsymbol{w}_{ML}$ is the most likely value of **w** given the data.
- So, we rewrite the formula as:

$$\boldsymbol{w}_{ML} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$

Solving for β

- Remember the Gaussian Noise model:

$$p(t|x, \boldsymbol{w}, \beta) = \mathcal{N}(t|y(x, \boldsymbol{w}), \beta^{-1})$$

- In the above equation, β is the **precision** of the Gaussian, and is defined as the inverse of the variance: $\beta = \frac{1}{\sigma^2}$

- Given our estimate $\boldsymbol{w}_{ML}$, we can also estimate the precision β and the variance σ^2:

$$(\sigma_{ML})^2 = \frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{[t_n - (\boldsymbol{w}_{ML})^T\varphi(x_n)]^2\}$$

Least Squares Solution - Summary

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_{M-1} \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \dots \\ t_N \end{bmatrix} \quad \Phi = \begin{bmatrix} \varphi_0(x_1), \dots, \varphi_{M-1}(x_1) \\ \varphi_0(x_2), \dots, \varphi_{M-1}(x_2) \\ \dots \\ \varphi_0(x_N), \dots, \varphi_{M-1}(x_N) \end{bmatrix}$$

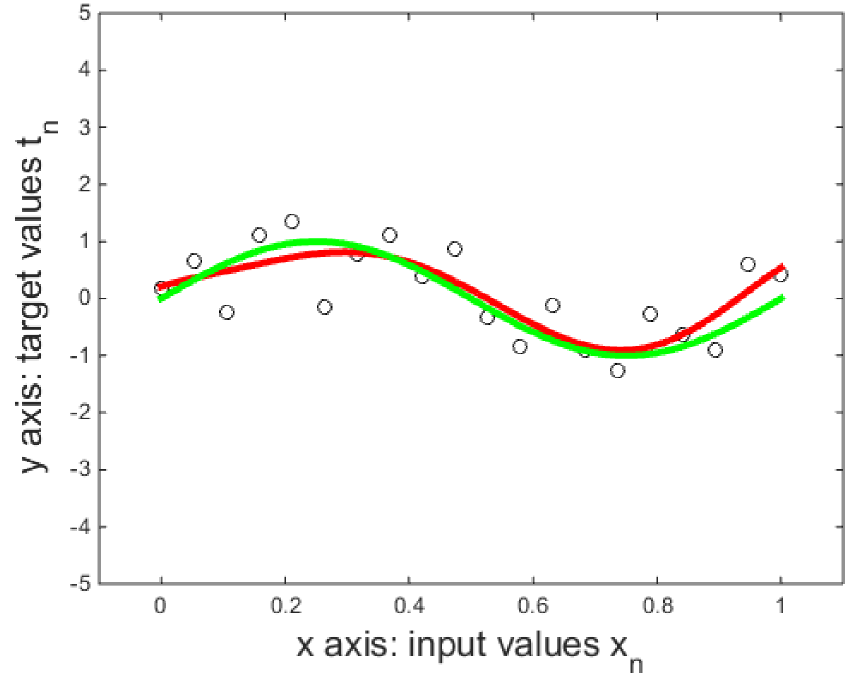
- Given the above notation:

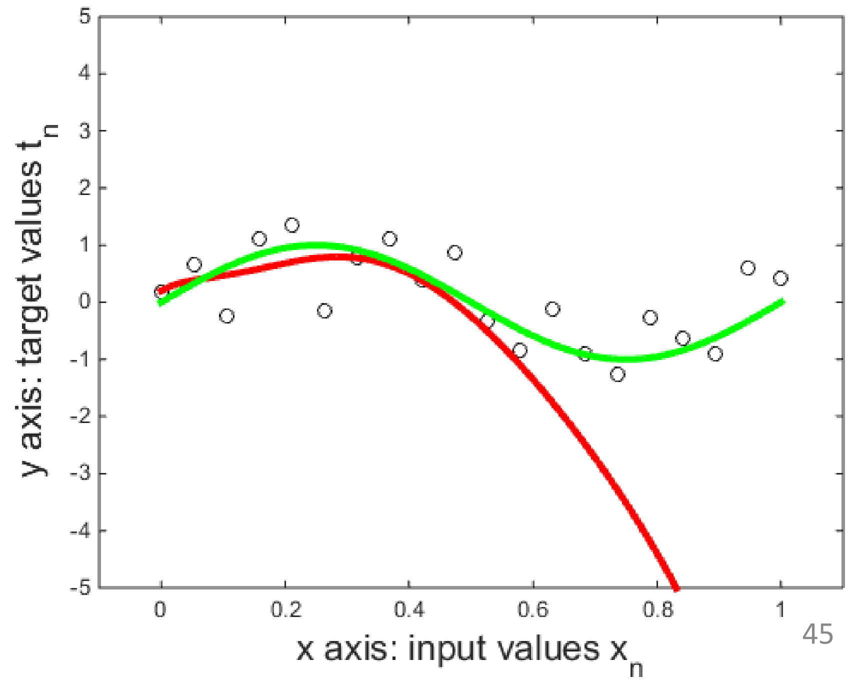
$$\boldsymbol{w}_{ML} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$

$$(\sigma_{ML})^2 = \frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n=1}^{N} \{[t_n - (\boldsymbol{w}_{ML})^T \varphi(x_n)]^2\}$$

Numerical Issues

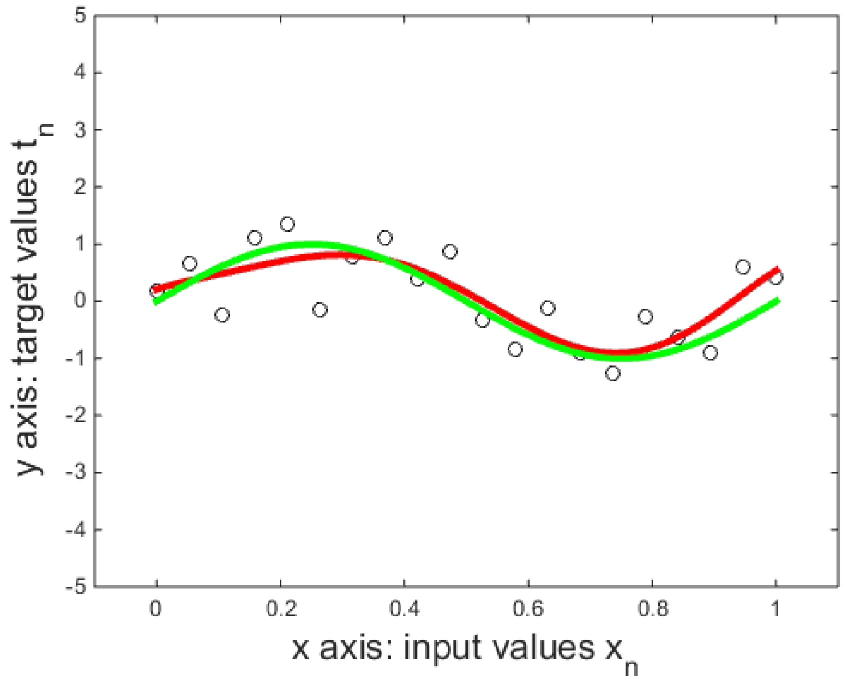
- Black circles: training data.
- Green curve: true function $y(x)=\sin(2\pi x)$.
- Red curve:
  - Top figure: Fitted 7-degree polynomial.
  - Bottom figure: Fitted 8-degree polynomial.
- Do you see anything strange?

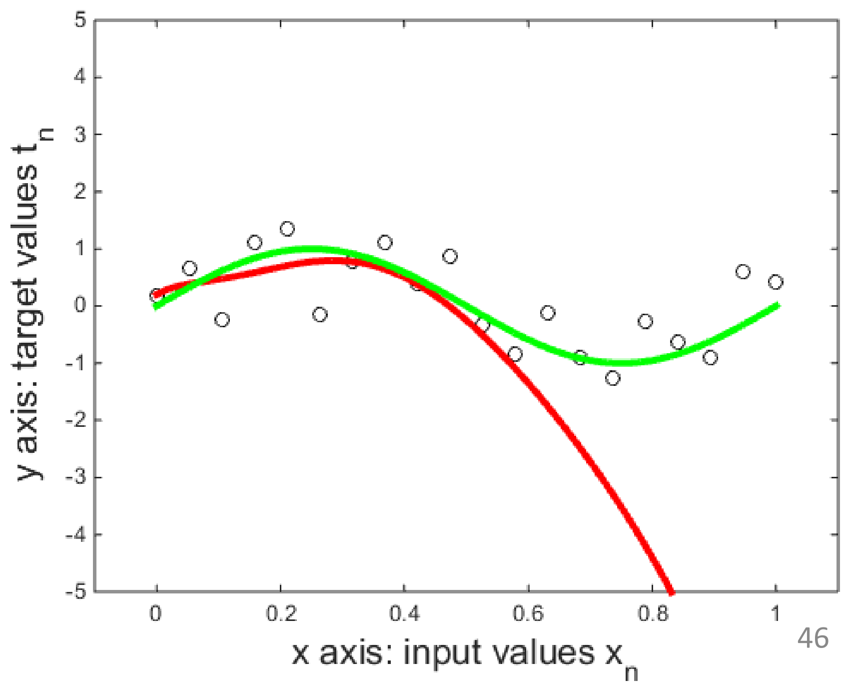




Numerical Issues

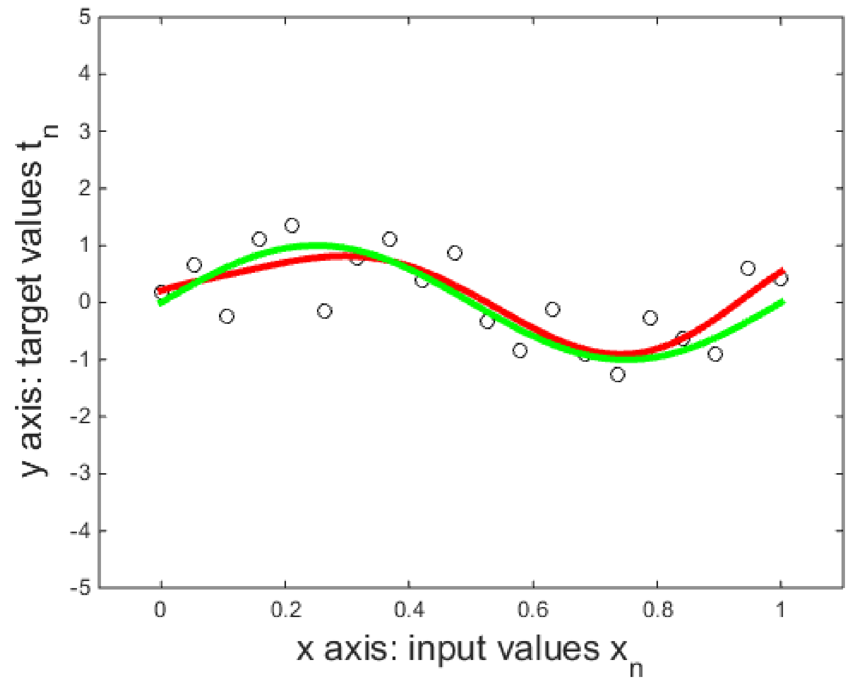
- The 8-degree polynomial fits the data worse than the 7-degree polynomial.
- Is this possible?

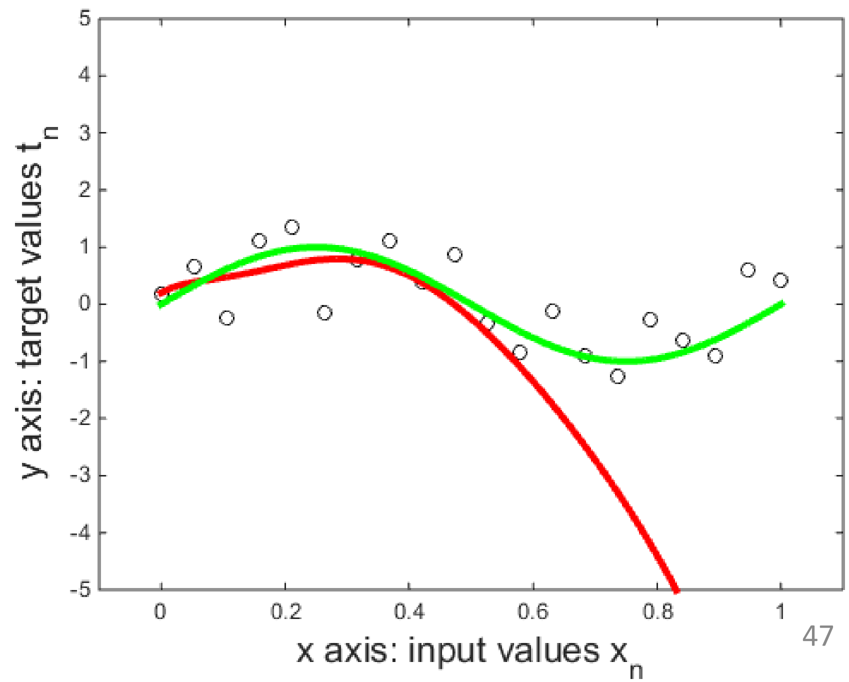




Numerical Issues

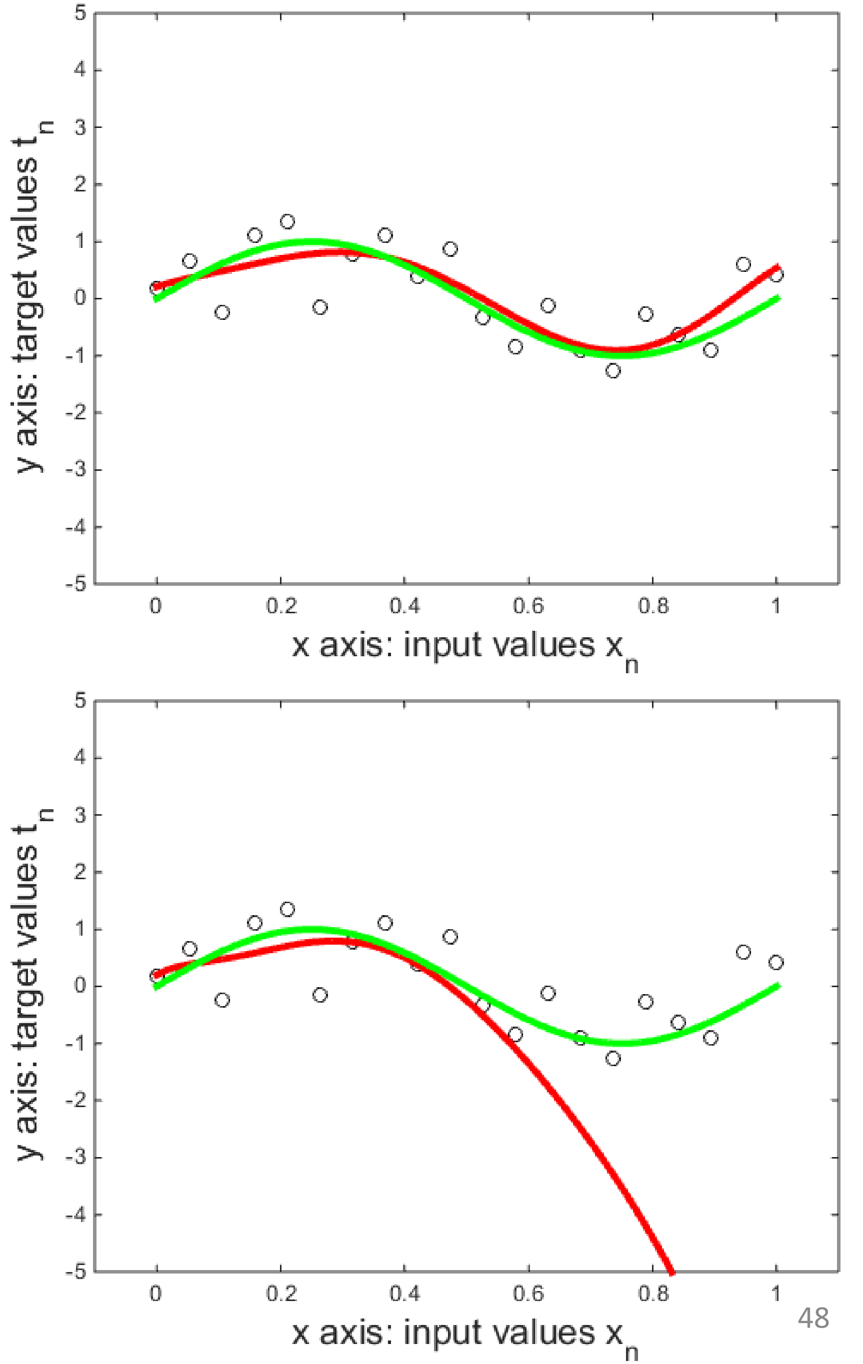
- The 8-degree polynomial fits the data worse than the 7-degree polynomial.
- Mathematically, this is impossible.
- 8-degree polynomials are a superset of 7-degree polynomials.
- Thus, the best-fitting 8-degree polynomial cannot fit the data worse than the best-fitting 7-degree polynomial.





Numerical Issues

- The 8-degree polynomial fits the data worse than the 7-degree polynomial.
- Mathematically, this is impossible.
- Cause of the problem: numerical issues.
- Computing the inverse of $\Phi^T \Phi$ may produce a result that is far from the correct one.



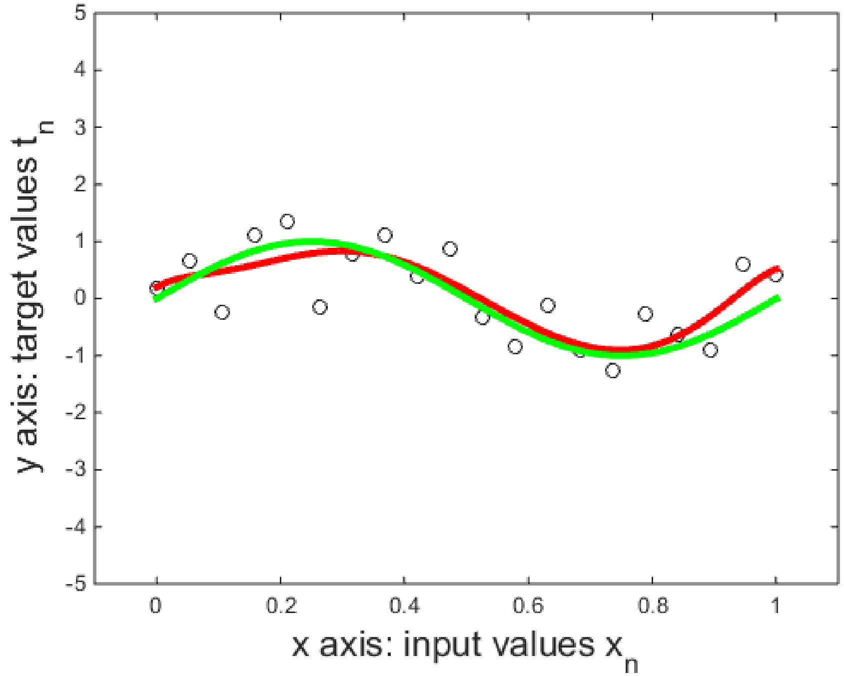
Numerical Issues

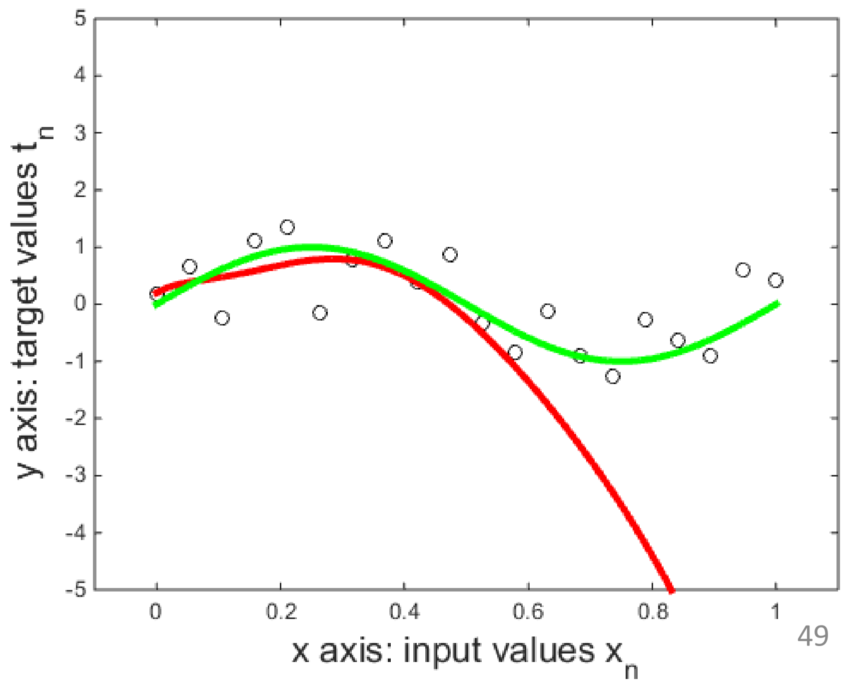
- Work-around in Matlab:
- `inv(phi' * phi)`

leads to the incorrect 8-degree polynomial fit below.

- `pinv(phi' * phi)`

leads to the correct 8-degree polynomial fit above (almost identical to the 7-degree result).





Sequential Learning

- The $\boldsymbol{w}_{ML}$ estimate was obtained by processing the training data as a **batch**.
  - We use all the data at once to estimate $\boldsymbol{w}_{ML}$.
- However, batch processing can be computationally expensive for large datasets.
  - It involves matrix multiplications of MxN matrices.
- An alternative is **sequential learning**.
  - This is also called **online learning**.
- In this scenario:
  - We first, somehow, get an initial estimate $\boldsymbol{w}^{(0)}$.
  - Then, we observe training examples, one by one.
  - Every time we observe a new training example, we update the estimate.

Sequential Learning

- We first, somehow, get an initial estimate $\boldsymbol{w}^{(0)}$.
  - That can be obtained, for example, by computing $\boldsymbol{w}_{ML}$ using the first few training examples as a batch.
  - Or, we initialize **w** to a random value.
- Then, we observe training examples, one by one.
- When we observe the n^th training example, we update the estimate from $\boldsymbol{w}^{(\tau)}$ to $\boldsymbol{w}^{(\tau+1)}$.
- Remember that the n^th training example contributes to the overall error $E_D(\boldsymbol{w})$ a term $E_{D,n}(\boldsymbol{w})$ defined as:

$$E_{D,n}(\boldsymbol{w}) = (t_n - \boldsymbol{w}^T \varphi(x_n))^2$$

Sequential Learning

- The n^th^ training example contributes to the overall error $E_D(\boldsymbol{w})$ a term $E_{D,n}(\boldsymbol{w})$ defined as:

$$E_{D,n}(\boldsymbol{w}) = (t_n - \boldsymbol{w}^T \varphi(x_n))^2$$

- When we observe the n^th^ training example, we update the estimate from $w^{(\tau)}$ to $w^{(\tau+1)}$ as follows:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_{D,n}(\boldsymbol{w}^{(\tau)}) = \boldsymbol{w}^{(\tau)} + \eta(t_n - \boldsymbol{w}^{(\tau)T} \varphi_n)\varphi_n$$

- η is called the **learning rate**. It is picked manually.
- This whole process is called **stochastic gradient descent.**

Sequential Learning - Intuition

- When we observe the n^{th} training example, we update the estimate from $w^{(\tau)}$ to $w^{(\tau+1)}$ as follows:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_{D,n}(\boldsymbol{w}^{(\tau)}) = \boldsymbol{w}^{(\tau)} + \eta(t_n - \boldsymbol{w}^{(\tau)T} \varphi_n)\varphi_n$$

- What is the intuition behind this update?
- The gradient $\nabla E_{D,n}(\boldsymbol{w})$ is a vector that points in the direction where $E_{D,n}(\boldsymbol{w})$ increases.
- Therefore, subtracting a very small amount of $\nabla E_{D,n}(\boldsymbol{w})$ from $\boldsymbol{w}$ should make $E_{D,n}(\boldsymbol{w})$ a little bit smaller.

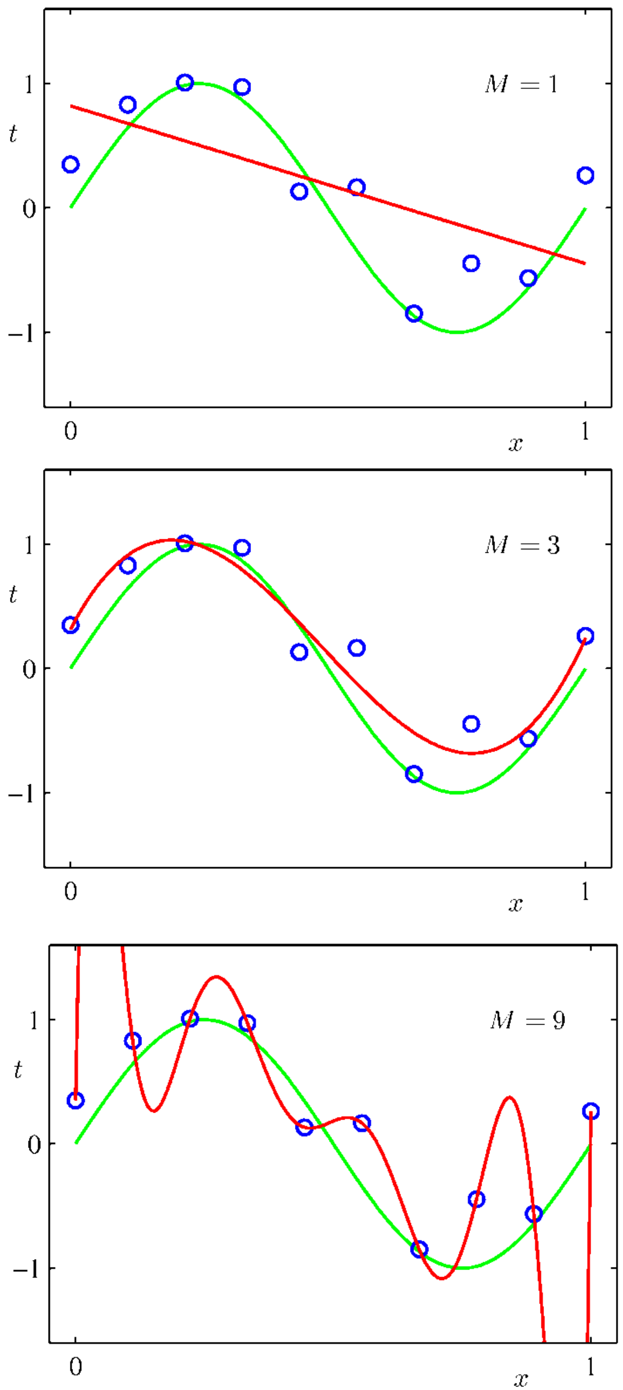
Sequential Learning - Intuition

- When we observe the n^th training example, we update the estimate from $w^{(\tau)}$ to $w^{(\tau+1)}$ as follows:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_{D,n}(\boldsymbol{w}^{(\tau)}) = \boldsymbol{w}^{(\tau)} + \eta(t_n - \boldsymbol{w}^{(\tau)T} \varphi_n)\varphi_n$$

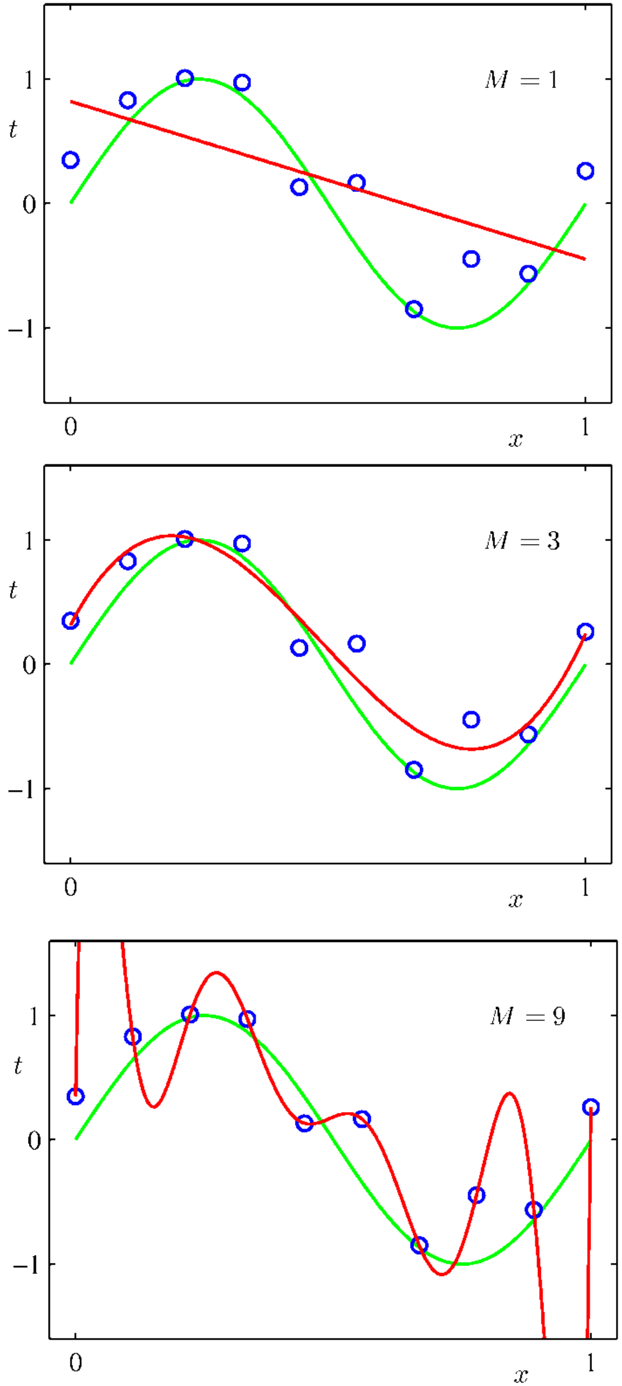
- Choosing a good value for η is important.
  - If η is too small, the minimization may happen too slowly and require too many training examples.
  - If η is large, the update may overfit the most recent training example, and overall **w** may fluctuate too much from one update to the next.
- Unfortunately, picking a good η is more of an art than a science, and involves trial-and-error.

Regularization

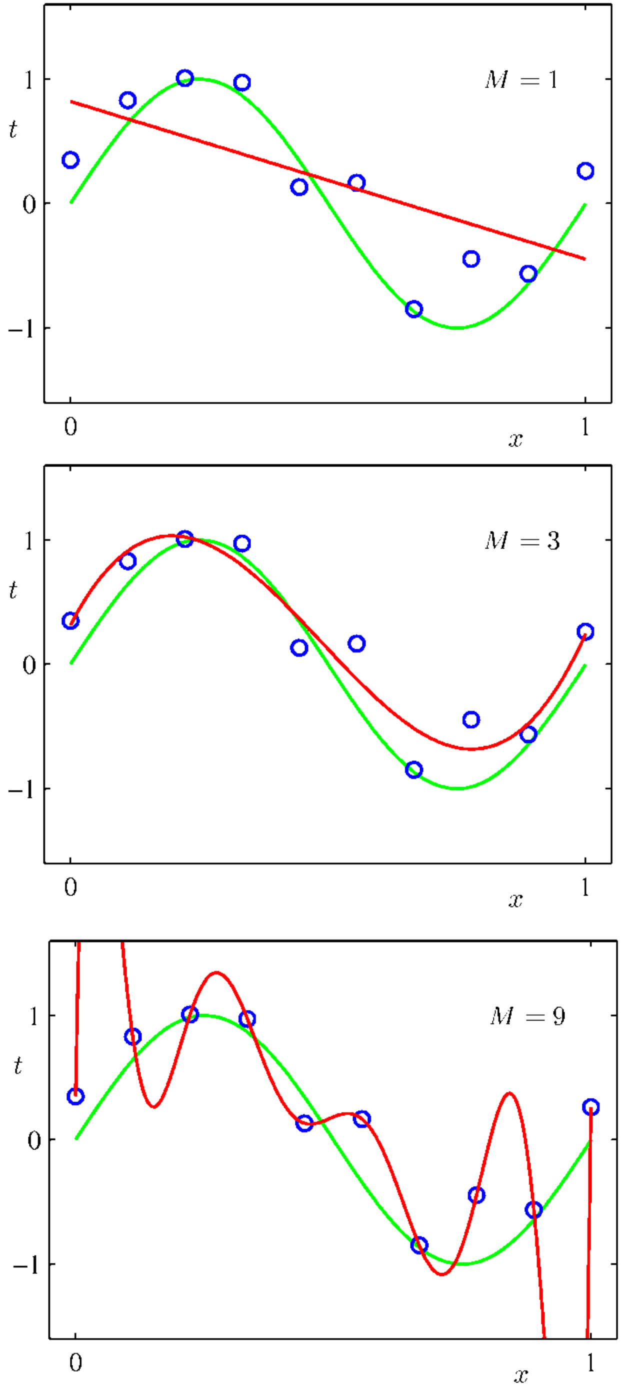


- Here we have a training set of 10 points (shown as blue circles).
- In green, we see the function that was used to generate these points (noise was added to that function).
  - It is a sinusoidal function.
- In red we see the best-fitting polynomials of degree 1, 3, 9.
- Interestingly, degree 3 matches the data well.
  - It would not match as well if we included points from more periods of the sinusoidal wave.

Regularization



- The solution for degree 9 suffers severely from overfitting.



Regularization

- We note that overfitting leads to very large magnitudes of parameters.

	Degree 1	Degree 3	Degree 9
w_0	0.82	0.31	0.35
w_1	-1.27	7.99	232.37
w_2		-25.43	-5321.83
w_3		17.37	48568.31
w_4			-231639.30
w_5			640042.26
w_6			-1061800.52
w_7			1042400.18
w_8			-557682.99
w_9			125201.43

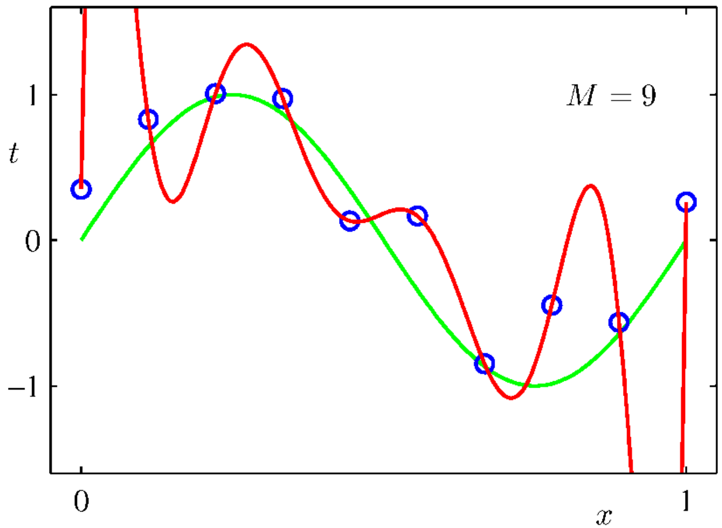
Regularization

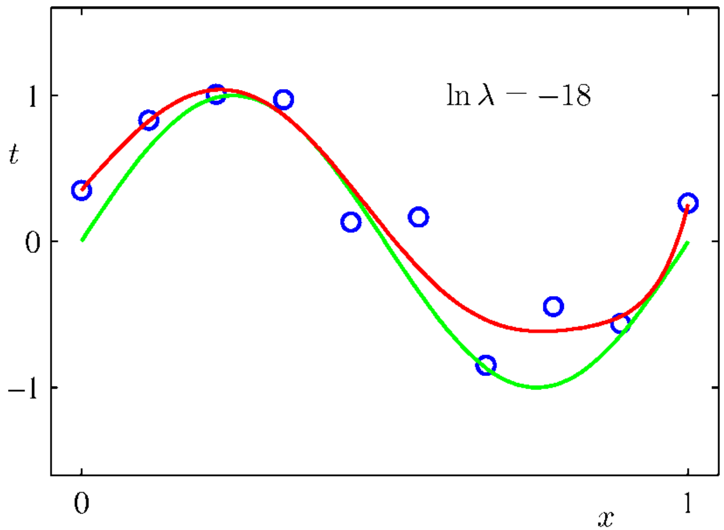
- If we are confident that large magnitudes of polynomial parameters are due to overfitting, we can penalize them in the error function:

$$\left(\sum_{n}(t_n - y(x_n, w))^2\right) + \lambda||w||^2$$

- The blue part is the sum-of-squares error that we saw before.
- The red part is what is called a **regularization term**.
- $||w||^2$ is the sum of squares of the parameters w_i.
- λ is a parameter that you have to specify.
  - It controls how much you penalize large $||w||^2$ values.

Regularization





	$\lambda = 0$	$\lambda = e^{-18}$
w_0	0.35	0.35
w_1	232.37	4.74
w_2	-5321.83	-0.77
w_3	48568.31	-31.97
w_4	-231639.30	-3.89
w_5	640042.26	55.28
w_6	-1061800.52	41.32
w_7	1042400.18	-45.95
w_8	-557682.99	-91.53
w_9	125201.43	72.68

A small λ solves the overfitting problem in this case.

Regularized Least Squares

- Formula for standard sum-of-squares error $E_D(\boldsymbol{w})$:

$$E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[(t_n - \boldsymbol{w}^T\varphi(x_n))^2]$$

- Formula for regularized sum-of-squares error:

$$\tilde{E}_D(\boldsymbol{w}) = \left\{\frac{1}{2}\sum_{n=1}^{N}[(t_n - \boldsymbol{w}^T\varphi(x_n))^2]\right\} + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

Solving Regularized Least Squares

$$\tilde{E}_D(\boldsymbol{w}) = \left\{ \frac{1}{2} \sum_{n=1}^{N} [(t_n - \boldsymbol{w}^T \varphi(x_n))^2] \right\} + \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{w}$$

- We skip the derivation, but the value of **w** that minimizes $\tilde{E}_D(w)$ is:

$$\boldsymbol{w} = (\lambda \mathrm{I} + \Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$

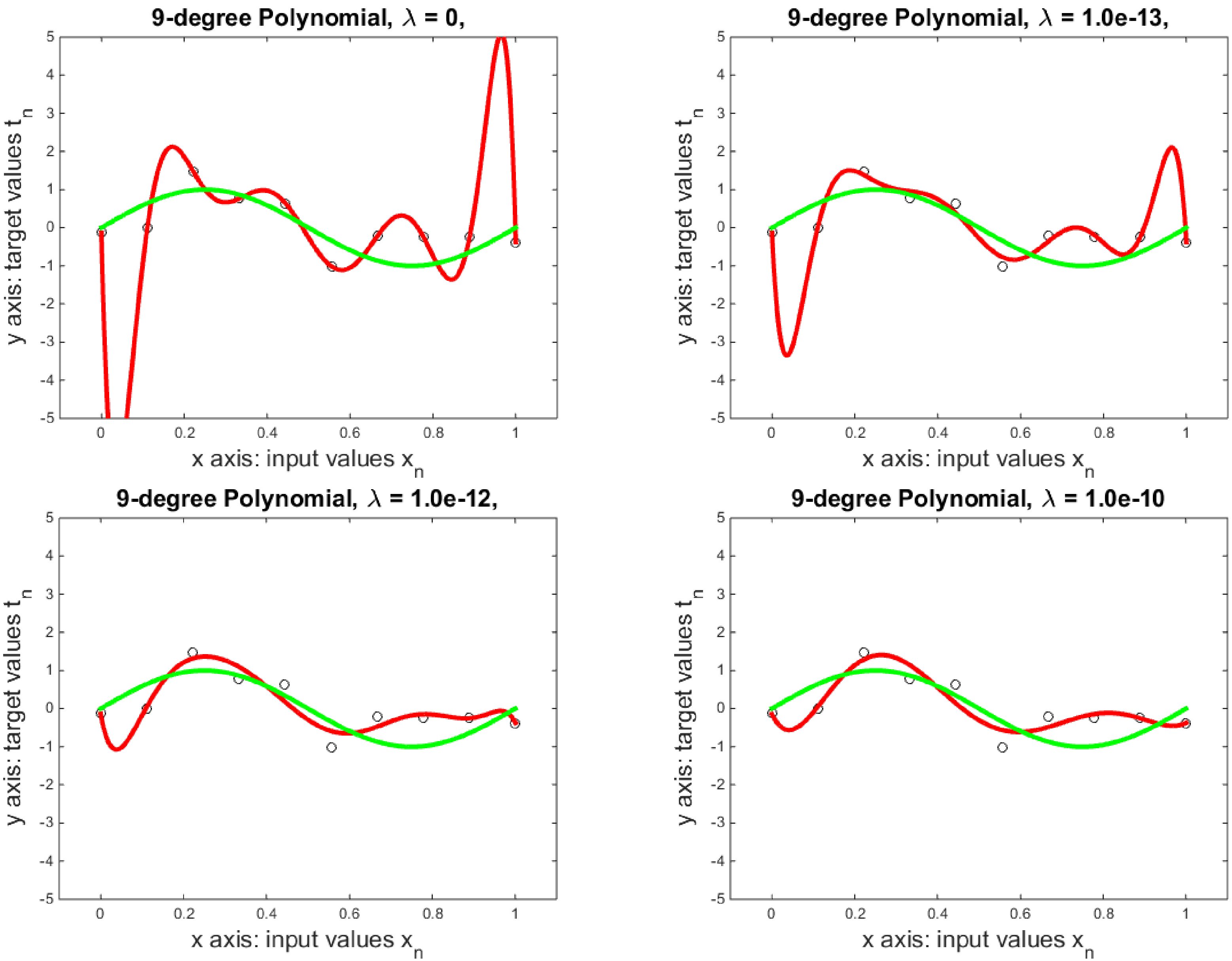
- In the above equation, I is the MxM identity matrix.

Why Use Regularization?

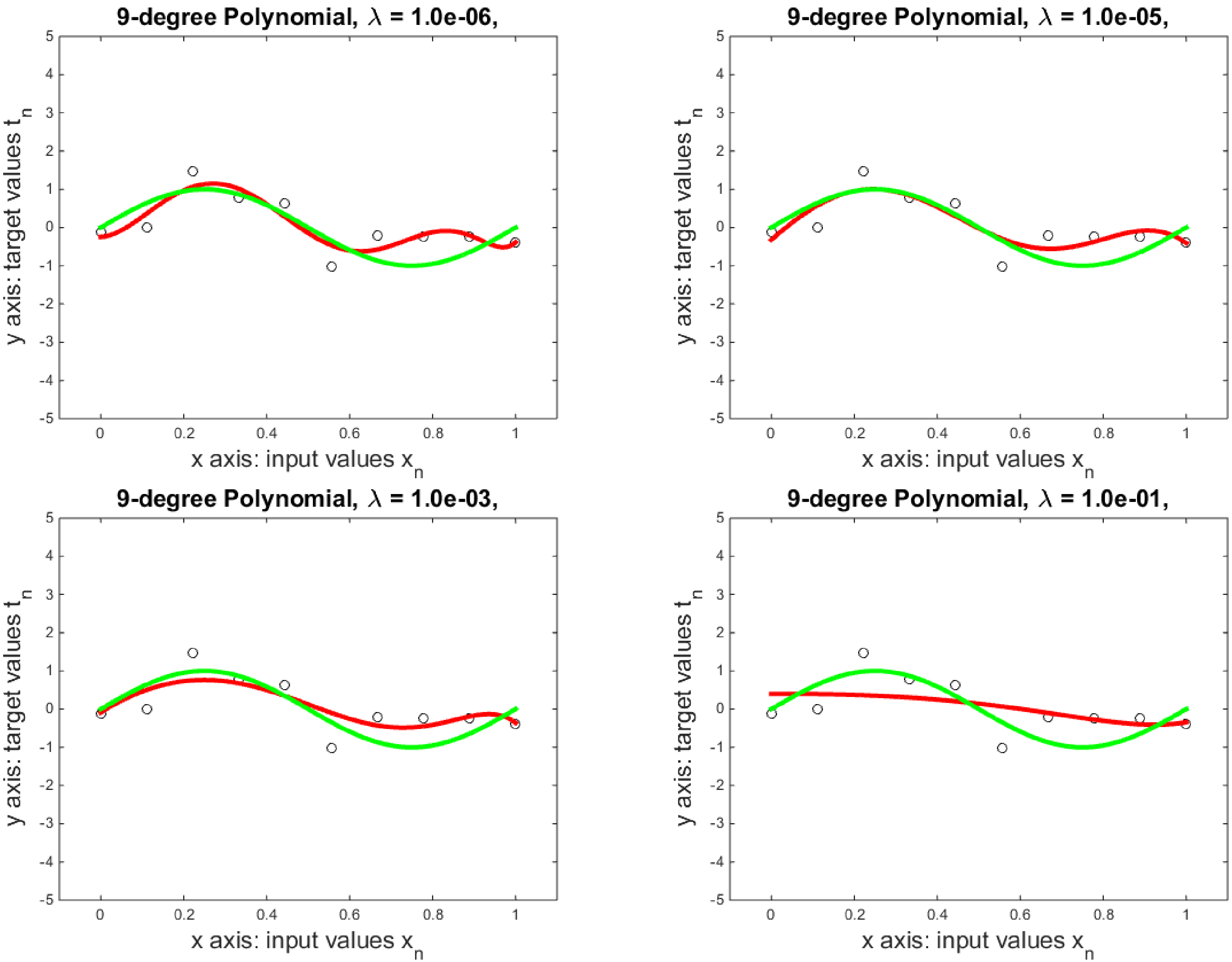
$$\tilde{E}_D(\boldsymbol{w}) = \left\{ \frac{1}{2} \sum_{n=1}^{N} \left[\left(t_n - \boldsymbol{w}^T \varphi(x_n) \right)^2 \right] \right\} + \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{w}$$

- We use regularization when we know, in advance, that some solutions should be preferred over other solutions.
- Then, we add to the error function a term that penalizes undesirable solutions.
- In the linear regression case: people know (from past experience) that high values of weights are indicative of overfitting.
- So, for each high value w_i we add to the error a penalty $(w_i)^2$.

Impact of Parameter λ



Impact of Parameter λ



Impact of Parameter λ

- As the previous figures show:
  - Low values of λ lead to polynomials whose values fluctuate more and more rapidly.
    - This can lead to increased overfitting.
  - High values of λ lead to flatter and flatter polynomials, that look more and more like straight lines.
    - This can lead to increased underfitting, or not fitting the data sufficiently.