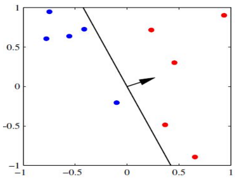
EECS 545: Machine Learning
Lecture 2. Linear Regression (Part 1)

Honglak Lee

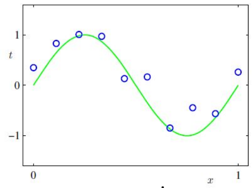
1/13/2025

Supervised Learning

- Goal:
  - Given data X in feature space and the labels Y
  - Learn to predict Y from X
- Labels could be discrete or continuous
  - Discrete-valued labels: classification
  - Continuous-valued labels: regression (today's topic)



classification



regression

Overview of Topics

- Linear Regression
  - Objective function
  - Vectorization
  - Computing gradient
  - Batch gradient vs. Stochastic Gradient
  - Closed form solution

Notation

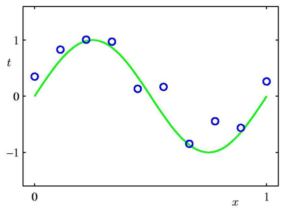
In this lecture, we will use the following notation:

- $\mathbf{x} \in \mathbb{R}^D$: data (scalar or vector)
- $\phi(\mathbf{x}) \in \mathbb{R}^M$: features for $\mathbf{x}$ (vector)
- $\phi_j(\mathbf{x}) \in \mathbb{R}$: j-th feature for $\mathbf{x}$ (scalar)
- $y \in \mathbb{R}$: continuous-valued label (i.e.,target value)

- $\mathbf{x}^{(n)}$: denotes the n-th training example.
- $y^{(n)}$: denotes the n-th training label.

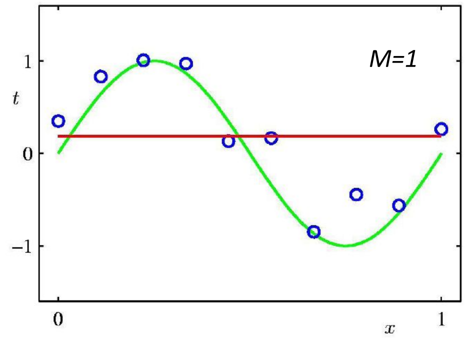
Linear regression (with 1D inputs)

- Consider the 1D case (e.g. D=1)
- Given a set of observation $\{x^{(1)} \dots x^{(N)}\}$
- and corresponding target values $\{y^{(1)} \dots y^{(N)}\}$
- We want to learn a function $h(x, \mathbf{w}) \approx y$ to predict future values

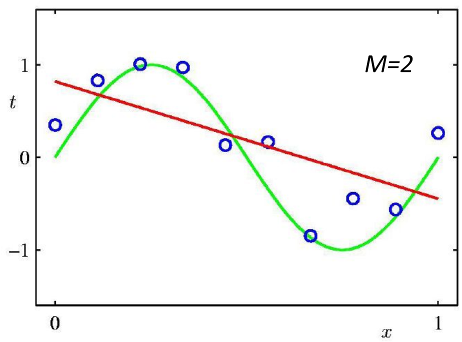


$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_{M-1} x^{M-1} = \sum_{j=0}^{M-1} w_j x^j$$

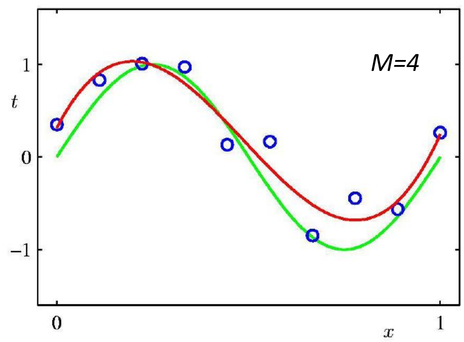
0th Order Polynomial



1st Order Polynomial



3rd Order Polynomial



Linear Regression (general case)

$$h(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

- The function $h(\mathbf{x}, \mathbf{w})$ is linear in parameters $\mathbf{w}$.
  - Goal: Find the best value for the weights $\mathbf{w}$.
- For simplicity, add a *bias term (constant function):*

$$h(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$$

$$\phi_0(\mathbf{x}) = 1$$

where $\mathbf{w} = (w_0, \ldots, w_{M-1})^\top$

$$\phi(x) = (\phi_0(\mathbf{x}), \ldots, \phi_{M-1}(\mathbf{x}))^\top$$

($\mathbf{w}$ and $\phi(\mathbf{x})$ are column vectors)

Basis Functions

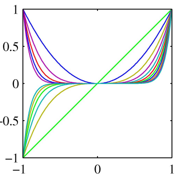
- The basis functions $\phi_j(\mathbf{x})$ doesn't need to be linear

$$\phi_j(x) = x^j$$

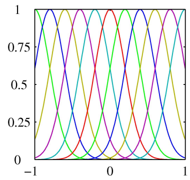
$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$

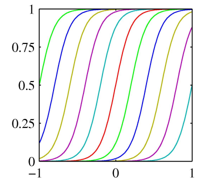
$$\sigma(a) = \frac{1}{1+\exp(-a)}$$



polynomial

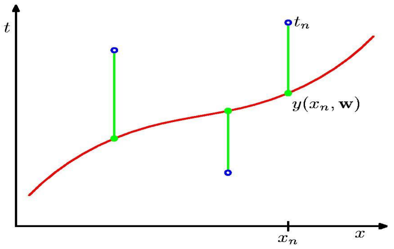


Gaussian



Sigmoid

Objective: Sum-of-Squares Error Function

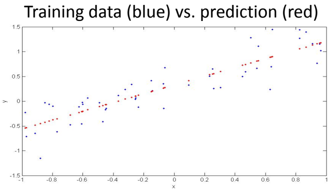


$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{h(\mathbf{x}^{(n)}, \mathbf{w}) - y^{(n)}\right\}^2$$

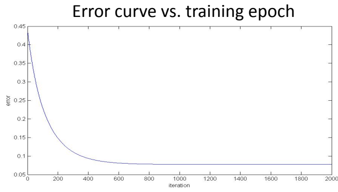
We want to find w that minimizes $E(\mathbf{w})$ over the training data.

Linear regression via gradient descent (illustration)

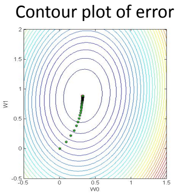
Training data (blue) vs. prediction (red)



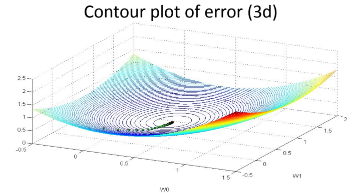
Error curve vs. training epoch



Contour plot of error



Contour plot of error (3d)



Least squares problem

- Objective function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

- Gradient

$$\begin{aligned}\frac{\partial E(w)}{\partial w_k} &= \frac{\partial}{\partial w_k}\frac{1}{2}\sum_{n=1}^{N}\left(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)}\right)^2 \\ &= \sum_{n=1}^{N}\left[\left(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)}\right)\frac{\partial}{\partial w_k}\left(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)}\right)\right] \\ &= \sum_{n=1}^{N}\left(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)}\right)\phi_k(\mathbf{x}^{(n)})\end{aligned}$$

Concatenate each component of the gradient:

$$\frac{\partial E(w)}{\partial w_k} = \sum_{n=1}^{N}\left(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)}\right)\phi_k(\mathbf{x}^{(n)})$$

We get a vectorized form of the gradient:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \qquad \phi(\mathbf{x}^{(n)}) =$$

$$\begin{aligned}\begin{bmatrix}\frac{\partial}{\partial w_0}E(\mathbf{w}) \\ \frac{\partial}{\partial w_1}E(\mathbf{w}) \\ \vdots \\ \vdots \\ \frac{\partial}{\partial w_{M-1}}E(\mathbf{w})\end{bmatrix} &= \sum_{n=1}^{N}\left(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)}\right)\begin{bmatrix}\phi_0(\mathbf{x}^{(n)}) \\ \phi_1(\mathbf{x}^{(n)}) \\ \vdots \\ \vdots \\ \phi_{M-1}(\mathbf{x}^{(n)})\end{bmatrix} \\ &= \sum_{n=1}^{N}\left(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)}\right)\phi(\mathbf{x}^{(n)}) \\ &= \sum_{n=1}^{N}\left(\mathbf{w}^\top\phi(\mathbf{x}^{(n)}) - y^{(n)}\right)\phi(\mathbf{x}^{(n)})\end{aligned}$$

Batch Gradient Descent

- Given data $(\mathbf{x}, \mathbf{y})$ and an initial $\mathbf{w}$
  - Repeat until convergence:

$$\mathbf{w} := \mathbf{w} - \eta\nabla_{\mathbf{w}} E(\mathbf{w})$$

where

$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}) &= \sum_{n=1}^{N}\left(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)}\right)\phi(\mathbf{x}^{(n)}) \\ &= \sum_{n=1}^{N}\left(\mathbf{w}^\top\phi(\mathbf{x}^{(n)}) - y^{(n)}\right)\phi(\mathbf{x}^{(n)})\end{aligned}$$

Stochastic Gradient Descent

- Main idea: instead of computing batch gradient (over entire training data), just compute gradient for individual example and update
- Repeat until convergence
  - for n=1,…,N

$$\mathbf{w} := \mathbf{w} - \eta\nabla_{\mathbf{w}} E(\mathbf{w}|\mathbf{x}^{(n)})$$

where

$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}|\mathbf{x}^{(n)}) &= \left(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)}\right)\phi(\mathbf{x}^{(n)}) \\ &= \left(\mathbf{w}^\top\phi(\mathbf{x}^{(n)}) - y^{(n)}\right)\phi(\mathbf{x}^{(n)})\end{aligned}$$

Note: Typically the learning rate is gradually decreased as training time (t) goes on: e.g., $\eta_t \propto \frac{1}{t}$ or $\eta_t = \eta_1\frac{1}{(1+(t-1)/\tau)}$

Stochastic Gradient Descent

- Repeat until convergence:
  - for n=1,…,N

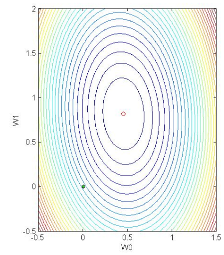
$$\mathbf{w} := \mathbf{w} - \eta \nabla_{\mathbf{w}} E(\mathbf{w}|\mathbf{x}^{(n)})$$

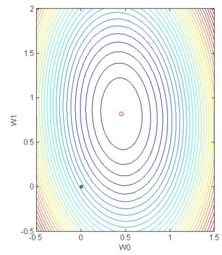
Note: Typically the learning rate is gradually decreased as training time (t) goes on:
e.g., $\eta_t \propto \frac{1}{t}$ or $\eta_t = \eta_1 \frac{1}{(1+(t-1)/\tau)}$

where
$$\nabla_{\mathbf{w}} E(\mathbf{w}|\mathbf{x}^{(n)}) = \left(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)} \right) \phi(\mathbf{x}^{(n)})$$
$$= \left(\mathbf{w}^\top \phi(\mathbf{x}^{(n)}) - y^{(n)} \right) \phi(\mathbf{x}^{(n)})$$

- Implementation tips in practice:
  - For each step of gradient computation in SGD, a small number of samples ("minibatch") may be used for computing the gradient instead of just one sample. Then we iterate this over the entire dataset with multiple epochs until convergence.

Batch gradient vs. Stochastic gradient





Closed form solution

- Main idea:
  - Compute gradient and set gradient to 0. (condition for optimal solution)
  - Solve the equation in a closed form
- The objective function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(\sum_{j=0}^{M-1} w_j \phi_j \left(\mathbf{x}^{(n)} \right) - y^{(n)} \right)^2$$

- We will derive the gradient from matrix calculus

Closed form solution

- Objective function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(\sum_{j=0}^{M-1} w_j \phi_j \left(\mathbf{x}^{(n)} \right) - y^{(n)} \right)^2$$
$$= \frac{1}{2} \sum_{n=1}^{N} \left(\mathbf{w}^\top \phi(\mathbf{x}^{(n)}) - y^{(n)} \right)^2$$
$$= \frac{1}{2} \sum_{n=1}^{N} \left(\mathbf{w}^\top \phi(\mathbf{x}^{(n)}) \right)^2 - \sum_{n=1}^{N} y^{(n)} \mathbf{w}^\top \phi(\mathbf{x}^{(n)}) + \frac{1}{2} \sum_{n=1}^{N} (y^{(n)})^2$$
$$= \frac{1}{2} \mathbf{w}^\top \Phi^\top \Phi \mathbf{w} - \mathbf{w}^\top \Phi^\top \mathbf{y} + \frac{1}{2} \mathbf{y}^\top \mathbf{y}$$

- Trick: vectorization (by defining data matrix)

The data matrix

- The design matrix is an NxM matrix, applying
  - the **M** basis functions (columns)
  - to **N** data points (rows)

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}^{(1)}) & \phi_1(\mathbf{x}^{(1)}) & \dots & \phi_{M-1}(\mathbf{x}^{(1)}) \\ \phi_0(\mathbf{x}^{(2)}) & \phi_1(\mathbf{x}^{(2)}) & \dots & \phi_{M-1}(\mathbf{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}^{(N)}) & \phi_1(\mathbf{x}^{(N)}) & \dots & \phi_{M-1}(\mathbf{x}^{(N)}) \end{pmatrix}$$

$$\Phi \mathbf{w} \approx \mathbf{y}$$

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}^{(1)}) & \phi_1(\mathbf{x}^{(1)}) & \dots & \phi_{M-1}(\mathbf{x}^{(1)}) \\ \phi_0(\mathbf{x}^{(2)}) & \phi_1(\mathbf{x}^{(2)}) & \dots & \phi_{M-1}(\mathbf{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}^{(N)}) & \phi_1(\mathbf{x}^{(N)}) & \dots & \phi_{M-1}(\mathbf{x}^{(N)}) \end{pmatrix}$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(\sum_{j=0}^{M-1} w_j \phi_j \left(\mathbf{x}^{(n)} \right) - y^{(n)} \right)^2$$
$$= \frac{1}{2} \sum_{n=1}^{N} \left(\mathbf{w}^\top \phi(\mathbf{x}^{(n)}) - y^{(n)} \right)^2$$
$$= \frac{1}{2} \sum_{n=1}^{N} \left(\mathbf{w}^\top \phi(\mathbf{x}^{(n)}) \right)^2 - \sum_{n=1}^{N} y^{(n)} \mathbf{w}^\top \phi(\mathbf{x}^{(n)}) + \frac{1}{2} \sum_{n=1}^{N} (y^{(n)})^2$$
$$= \frac{1}{2} \mathbf{w}^\top \Phi^\top \Phi \mathbf{w} - \mathbf{w}^\top \Phi^\top \mathbf{y} + \frac{1}{2} \mathbf{y}^\top \mathbf{y}$$

Useful trick: Matrix Calculus

- Idea so far:
  - Compute gradient and set gradient to **0** (condition for optimal solution)
  - Solve the equation in a closed form using matrix calculus
- Need to compute the first derivative in matrix form

Matrix calculus: The Gradient

- Suppose that $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ is a function that takes as an input matrix **A** of size [m x n] and returns a real value (scalar).
- Then the gradient of f with respect to $A \in \mathbb{R}^{m \times n}$ is the matrix of partial derivatives, defined as:

$$\nabla_A f(A) \in \mathbb{R}^{m \times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

$$(\nabla_A f(A))_{ij} = \frac{\partial f(A)}{\partial A_{ij}}$$

Matrix calculus: The Gradient

Note that the size of $\nabla_A f(A)$ is always the same as the size of A. So if, in particular, A is just a vector $x \in \mathbb{R}^n$, then

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

- $\nabla_x (f(x) + g(x)) = \nabla_x f(x) + \nabla_x g(x)$.
- For $t \in \mathbb{R}$, $\nabla_x (t\, f(x)) = t \nabla_x f(x)$.

Gradient of Linear Functions

- Linear function: $f(\mathbf{x}) = \sum_{i=1}^{n} b_i x_i = \mathbf{b}^\top \mathbf{x}$
- Gradient: $\frac{\partial f(\mathbf{x})}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_{i=1}^{n} b_i x_i = b_k$
- Compact form: $\nabla_{\mathbf{x}} f(\mathbf{x}) = \mathbf{b}$

Gradient of Quadratic Functions

* Assumption: A is a symmetric matrix: i.e., $A_{ij} = A_{ji}$

- Quadratic function: $f(\mathbf{x}) = \sum_{i,j=1}^{n} x_i A_{ij} x_j = \mathbf{x}^\top \mathbf{A} \mathbf{x}$
- Gradient: $\frac{\partial f(\mathbf{x})}{\partial x_k} = 2 \sum_{j=1}^{n} A_{kj} x_j = 2(\mathbf{A}\mathbf{x})_k$
- Compact form: $\nabla_{\mathbf{x}} f(\mathbf{x}) = 2\mathbf{A}\mathbf{x}$

Putting together: Solution via matrix calculus

- Compute gradient and set to zero

$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}) &= \nabla_{\mathbf{w}} \left(\frac{1}{2} \mathbf{w}^\top \Phi^\top \Phi \mathbf{w} - \mathbf{w}^\top \Phi^\top \mathbf{y} + \frac{1}{2} \mathbf{y}^\top \mathbf{y} \right) \\ &= \Phi^\top \Phi \mathbf{w} - \Phi^\top \mathbf{y} \\ &= \mathbf{0}\end{aligned}$$

- Solve the resulting equation (normal equation)

$$\Phi^\top \Phi \mathbf{w} = \Phi^\top \mathbf{y}$$
$$\mathbf{w}_{\mathrm{ML}} = (\Phi^\top \Phi)^{-1} \Phi^\top \mathbf{y}$$

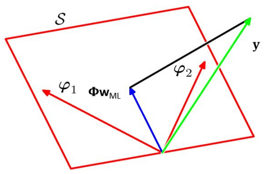
This is the *Moore-Penrose pseudo-inverse*: $\Phi^\dagger = (\Phi^\top \Phi)^{-1} \Phi^\top$

applied to: $\Phi \mathbf{w} \approx \mathbf{y}$

Geometric Interpretation

- Assuming many more observations (N) than the M basis functions $\phi_j(x)$ (j=0,…,M-1)
- View the observed target values $\mathbf{y} = \{y^{(1)}, \ldots, y^{(N)}\}$ as a vector in an N-dim. space.
- The M basis functions $\phi_j(x)$ span the N-dimensional subspace.
  - Where the N-dim vector for ϕ_j is $\{\phi_j(\mathbf{x}^{(1)}), \ldots, \phi_j(\mathbf{x}^{(N)})\}$
- $\mathbf{\Phi w}_{\mathrm{ML}}$ is the point in the subspace with minimal squared error from $\mathbf{y}$.
- It's the projection of $\mathbf{y}$ onto that subspace.

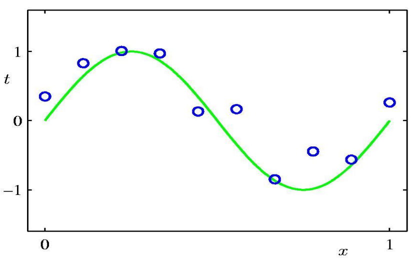
$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}^{(1)}) & \phi_1(\mathbf{x}^{(1)}) & \ldots & \phi_{M-1}(\mathbf{x}^{(1)}) \\ \phi_0(\mathbf{x}^{(2)}) & \phi_1(\mathbf{x}^{(2)}) & \ldots & \phi_{M-1}(\mathbf{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}^{(N)}) & \phi_1(\mathbf{x}^{(N)}) & \ldots & \phi_{M-1}(\mathbf{x}^{(N)}) \end{pmatrix}$$



Slide credit: Ben Kuipers

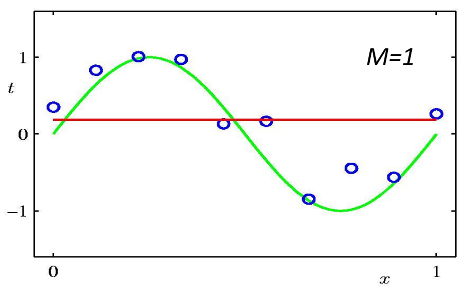
Back to curve-fitting examples

Polynomial Curve Fitting

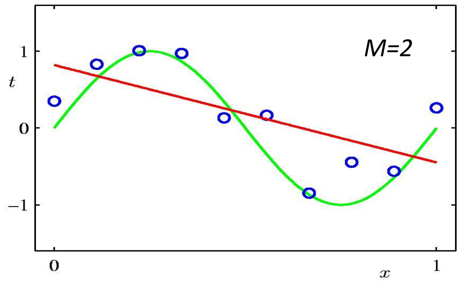


$$h(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_{M-1} x^{M-1} = \sum_{j=0}^{M-1} w_j x^j$$

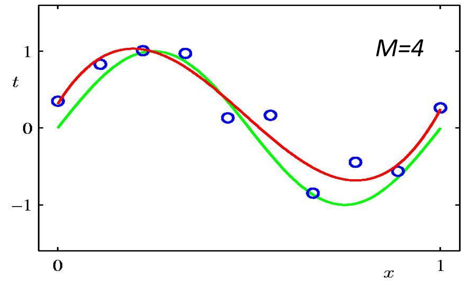
0th Order Polynomial



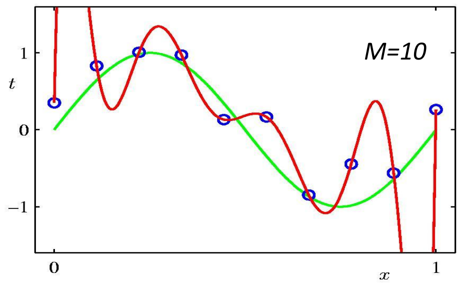
1st Order Polynomial



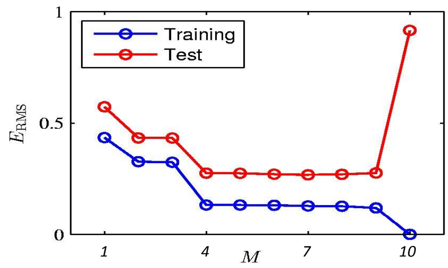
3rd Order Polynomial



9th Order Polynomial



Over-fitting



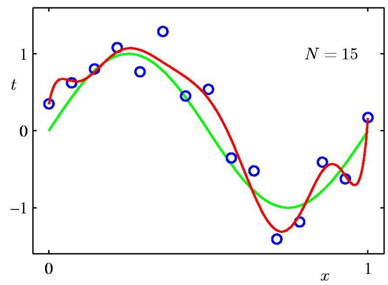
Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$

Polynomial Coefficients

	M=1	*M=2*	*M=4*	*M=10*
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

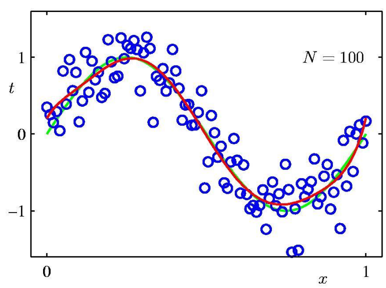
Data Set Size: $N = 15$

9th Order Polynomial



Data Set Size: $N = 100$

9th Order Polynomial



Q. How do we choose the degree of polynomial?

Rule of thumb

- If you have a small number of data points, then you should use low order polynomial (small number of features).
  - Otherwise, your model will overfit
- As you obtain more data points, you can gradually increase the order of the polynomial (more features).
  - However, your model is still limited by the finite amount of the data available (i.e., the optimal model for finite data cannot be infinite dimensional polynomial).
- Controlling model complexity: **regularization**