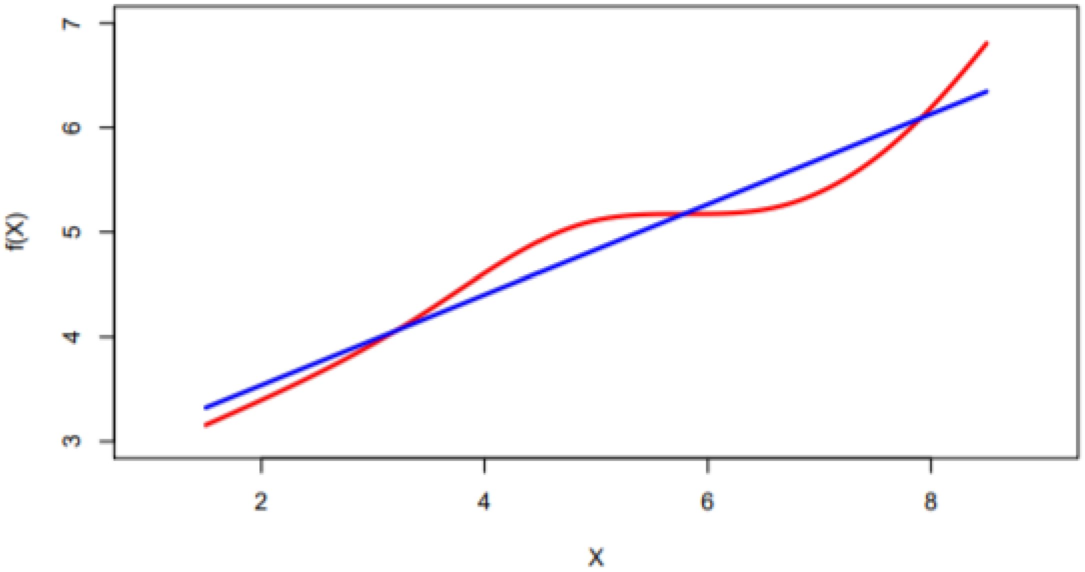
Linear Regression

Linear regression

- Linear regression is a simple approach to supervised learning. It assumes that the dependence of Y on $X_1, X_2, \ldots X_p$ is linear.
- True regression functions are never linear!

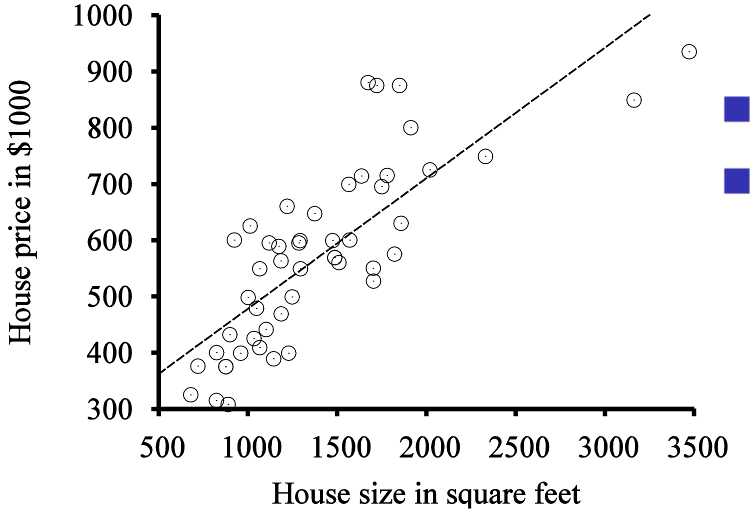


- although it may seem overly simplistic, linear regression is extremely useful both conceptually and practically.

Optimization

- Learning task: minimizing or maximizing an evaluation function $J(w_1, \ldots, w_n)$ given data $\mathcal{D}$
- $w_1, \ldots, w_n$ are the parameters that you need to tune.
- Simple example: Try to fit a line to the following data such that the error is minimized.
- Input: "x", desired output "y"

Linear Regression!

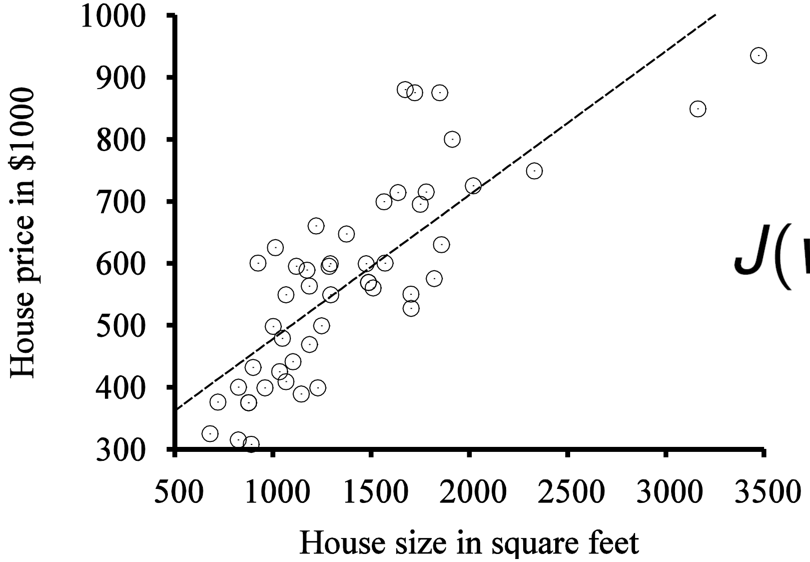


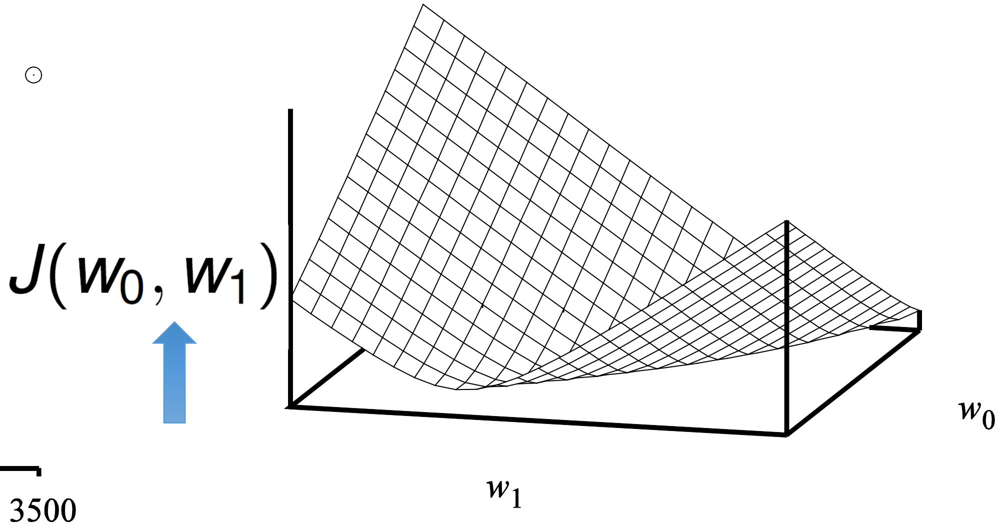
- Equation of line: $y = h(x) = w_0 + w_1 x$
- Error: $J(w_0, w_1) = \sum_{i=1}^{m} (y_i - (w_0 + w_1 x_i))^2$

(Point-wise) squared error

Problem: Minimize error

Question: How to solve the optimization problem



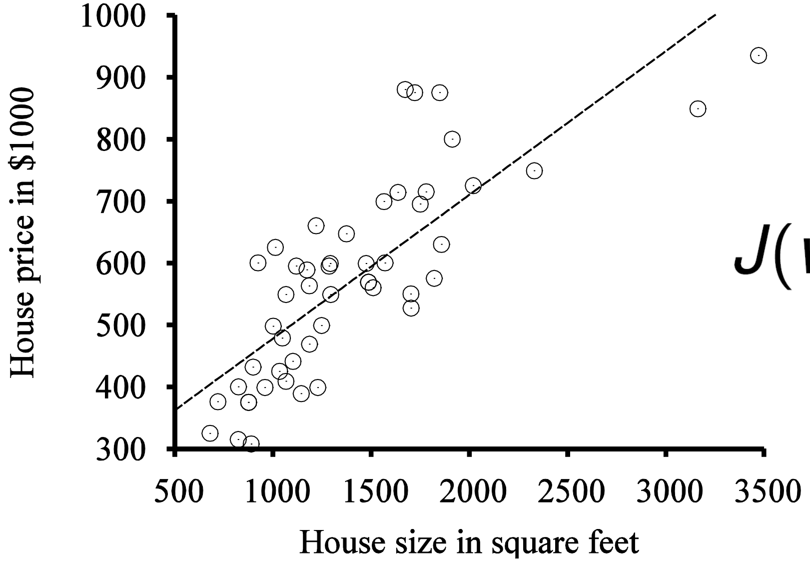


- Set the derivative of "J" to zero and solve

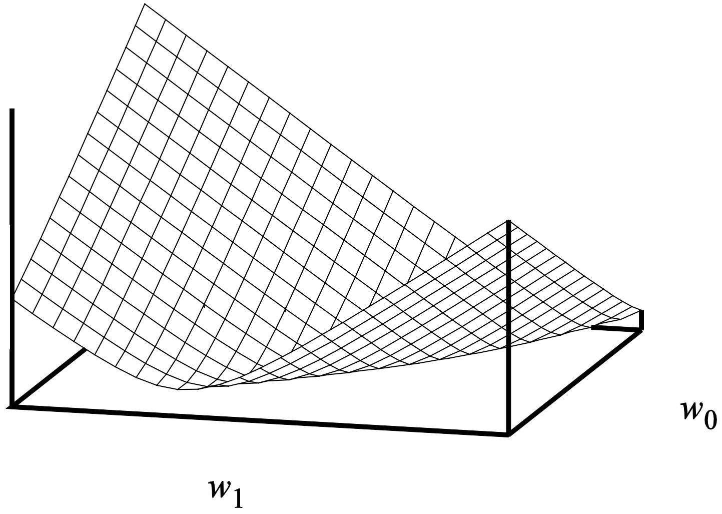
$$J(w_0, w_1) = \sum_{i=1}^{m}(y_i - (w_0 + w_1 x_i))^2$$

$$\frac{\partial}{\partial w_0} J(w_0, w_1) = 0 \qquad \frac{\partial}{\partial w_1} J(w_0, w_1) = 0$$

Question: How to solve the optimization problem



$J(w_0, w_1)$



$$w_1 = \frac{m\sum_{i=1}^{m} x_i y_i - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} y_i}{m\sum_{i=1}^{m} x_i^2 - (\sum_{i=1}^{m} x_i)^2}$$

$$w_0 = \frac{\sum_{i=1}^{m} y_i - w_1 \sum_{i=1}^{m} x_i}{m}$$

Homework:
Prove this!
(Messy; algebraic manipulation)

Multivariate Linear Regression

- Input: **x** is a vector; desired output y.

Assuming a dummy attribute x_0=1 **for all examples**

$$y = h(\mathbf{x}) = w_0 + \sum_{j=1}^{n} w_j x_j = \sum_{j=0}^{n} w_j x_j = \mathbf{w}^T \mathbf{x}$$

Inner product or dot product (yields a number)

$$J(\mathbf{w}) = \sum_{i=1}^{m} (y_i - \mathbf{w}^T x_i)^2$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \cdots \\ \cdots \\ \mathbf{x}_m^T \end{bmatrix}$$

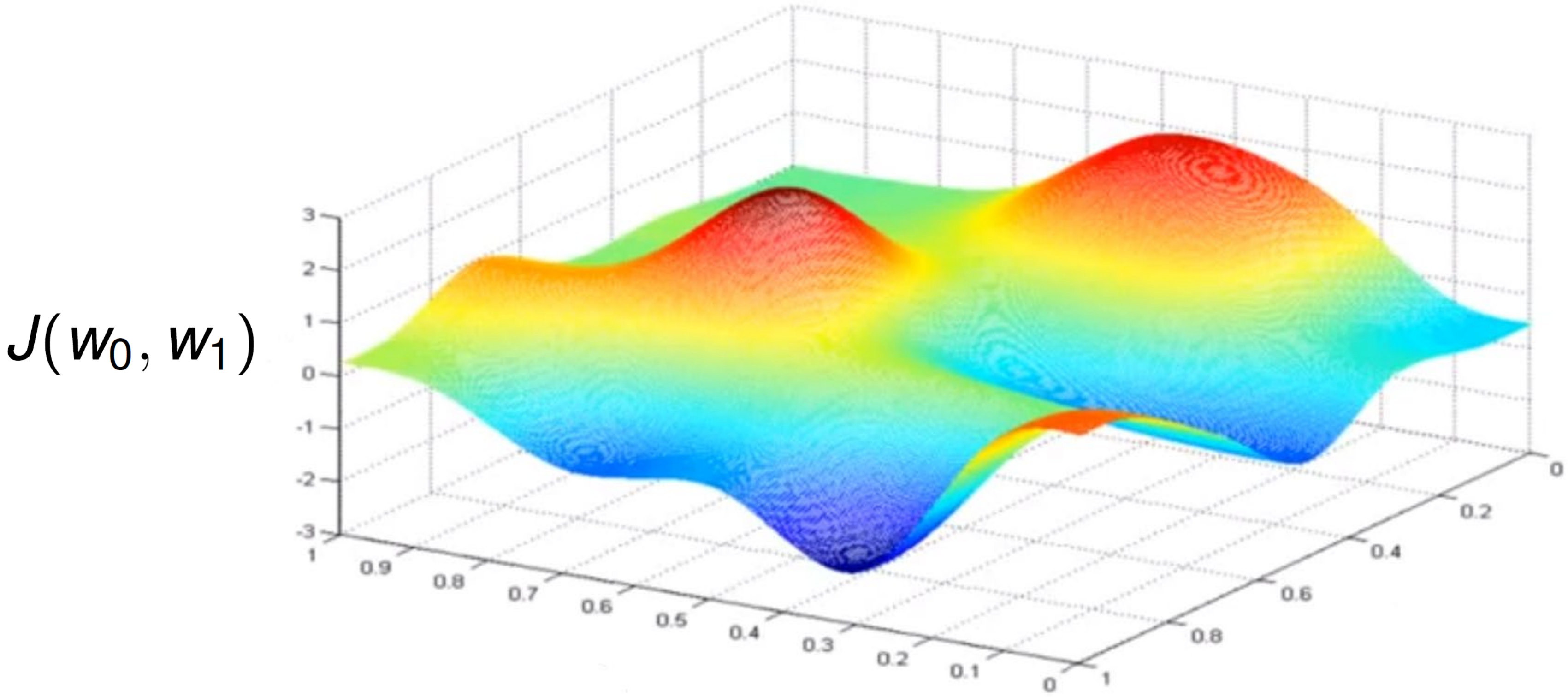
X is a m-by-n matrix

Gradient Descent

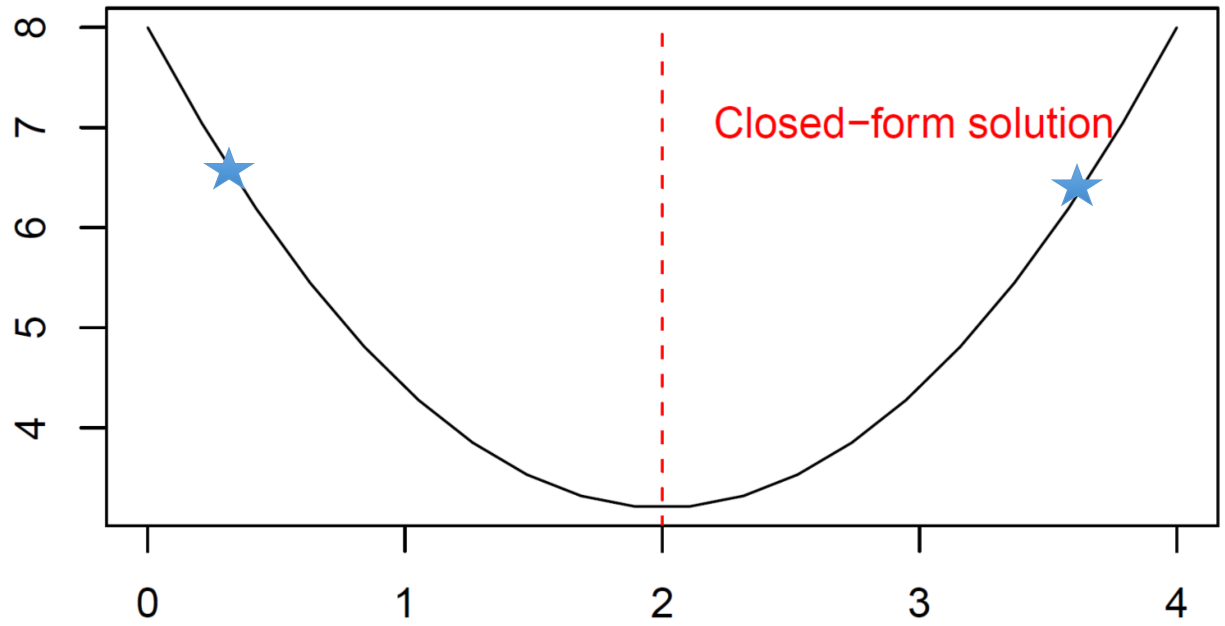
- Closed form solution is not always possible.
- In that case, we can use the following iterative approach.
- **Algorithm Gradient Descent**

  - $\mathbf{w}$ = Any point in the weight space
  - Loop Until Convergence
    - Simultaneously update each w_j in $\mathbf{w}$ as follows:
      - $w_j = w_j - \underbrace{\alpha}_{\textbf{Learning rate}} \frac{\partial}{\partial w_j} J(\mathbf{w})$

Gradient Descent: Example



Gradient Descent: 1-D

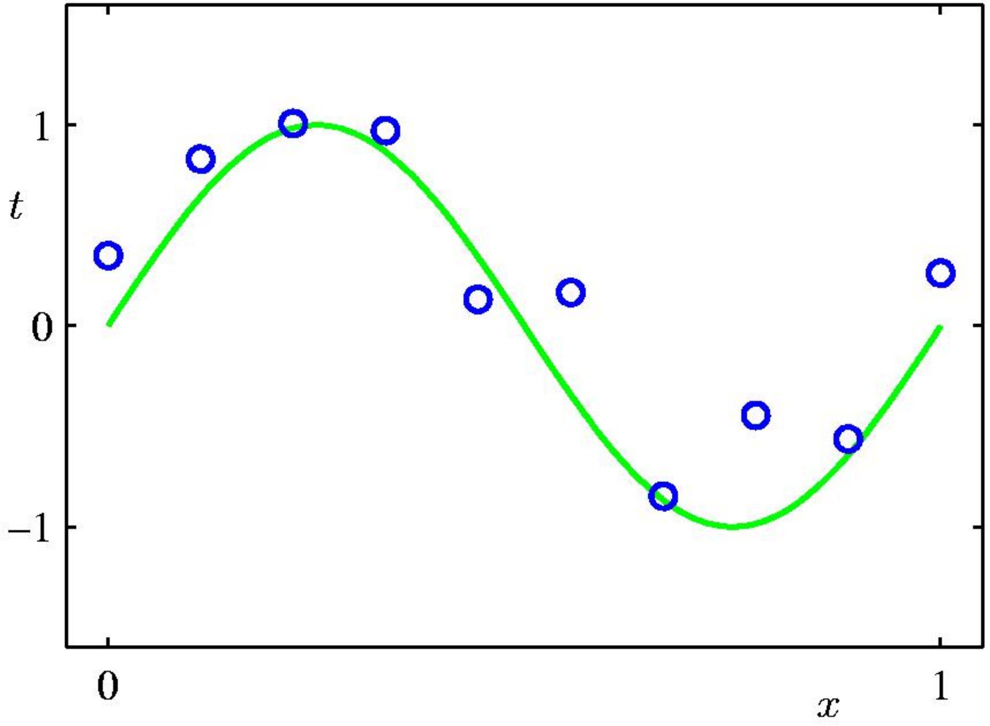


- **Remember:** Derivative is the slope of the line that is tangent to the function
- **Question:** What if the learning rate is small? (Slow convergence)
- **Question:** What if the learning rate is large? (Fail to converge; even diverge)

Rule:

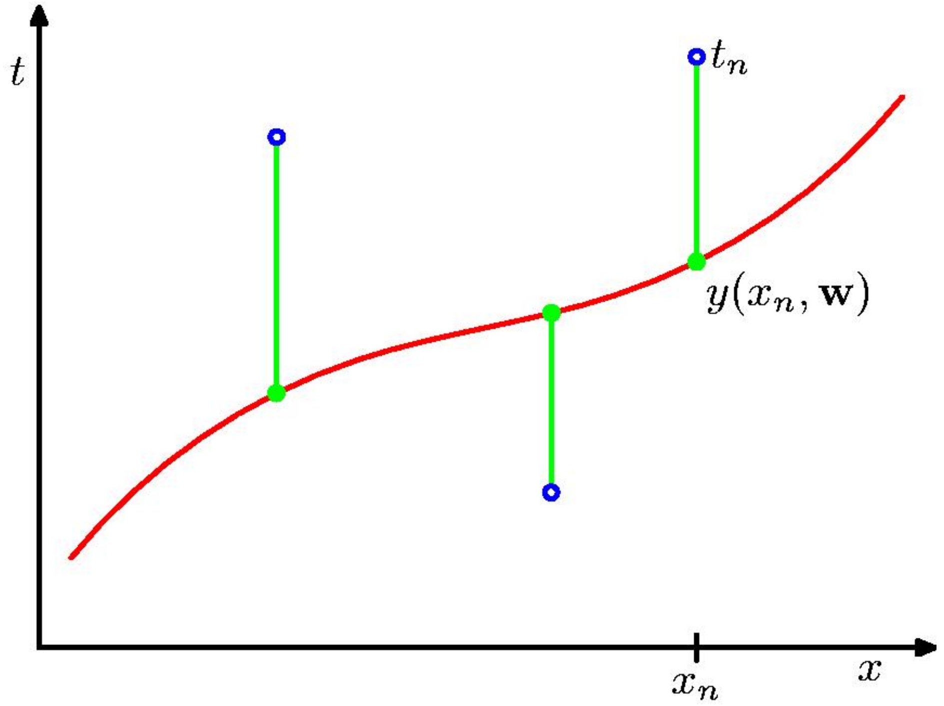
$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w})$$

Polynomial Curve Fitting



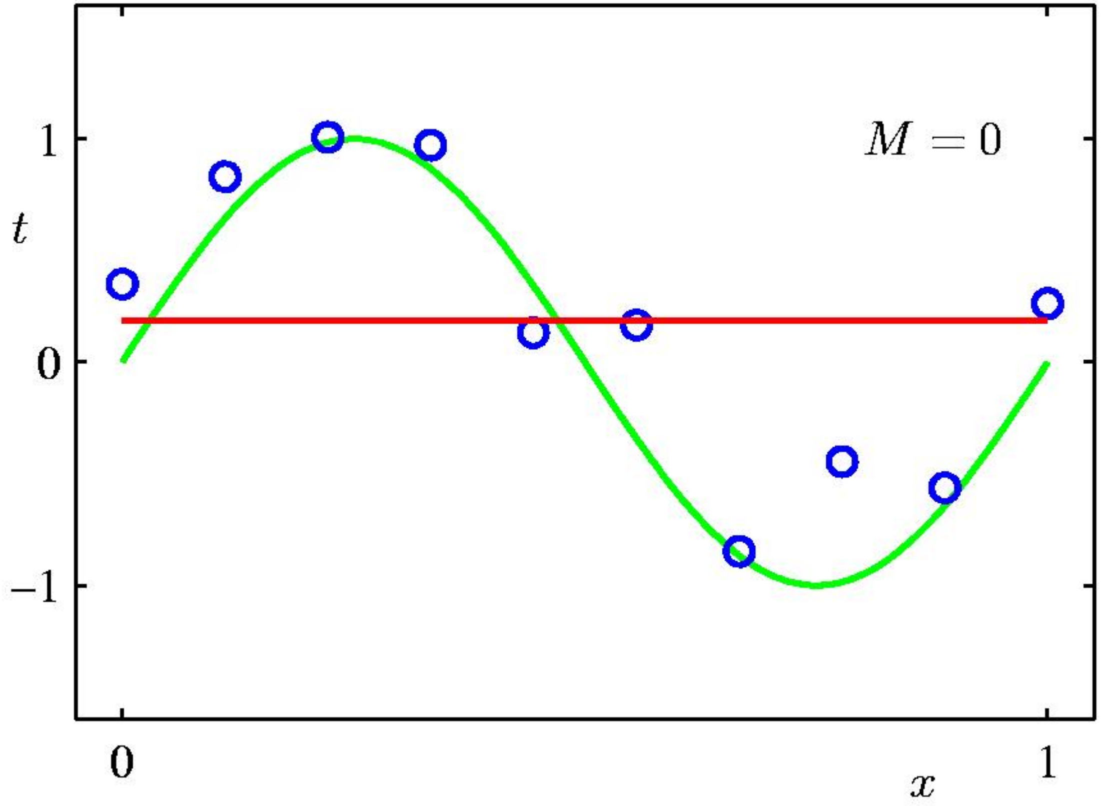
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Sum-of-Squares Error Function

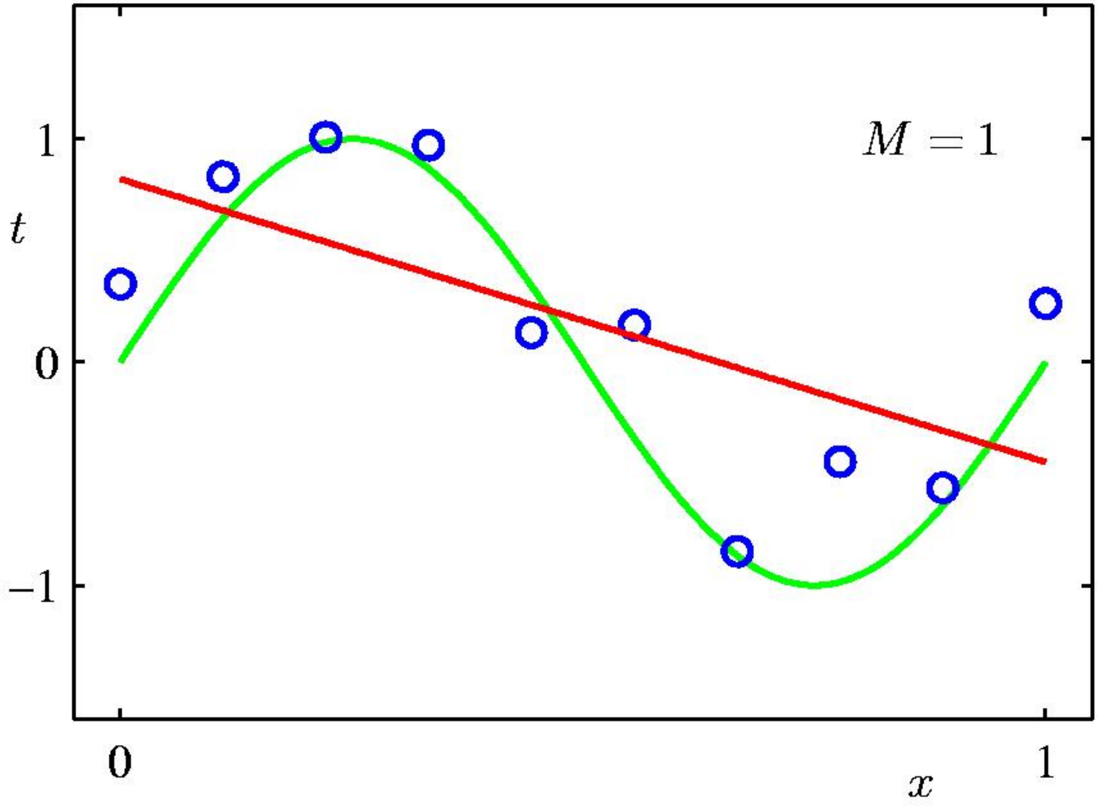


$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

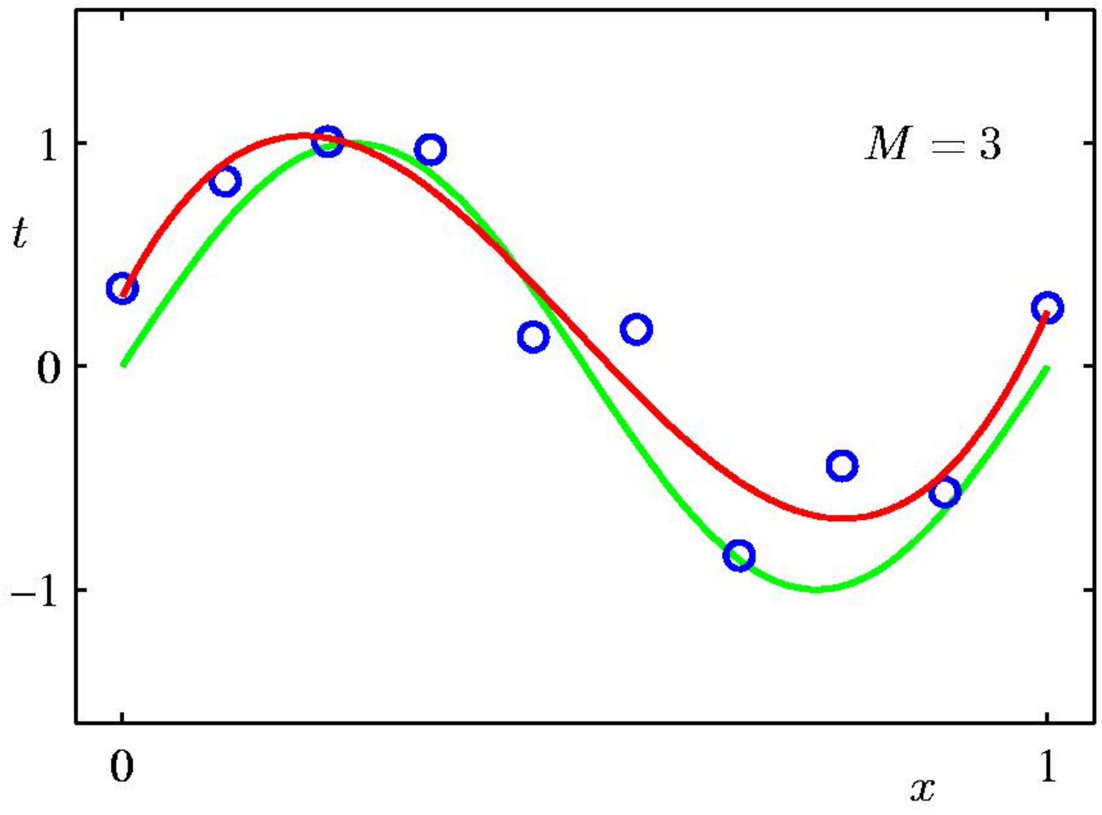
0^{th} Order Polynomial



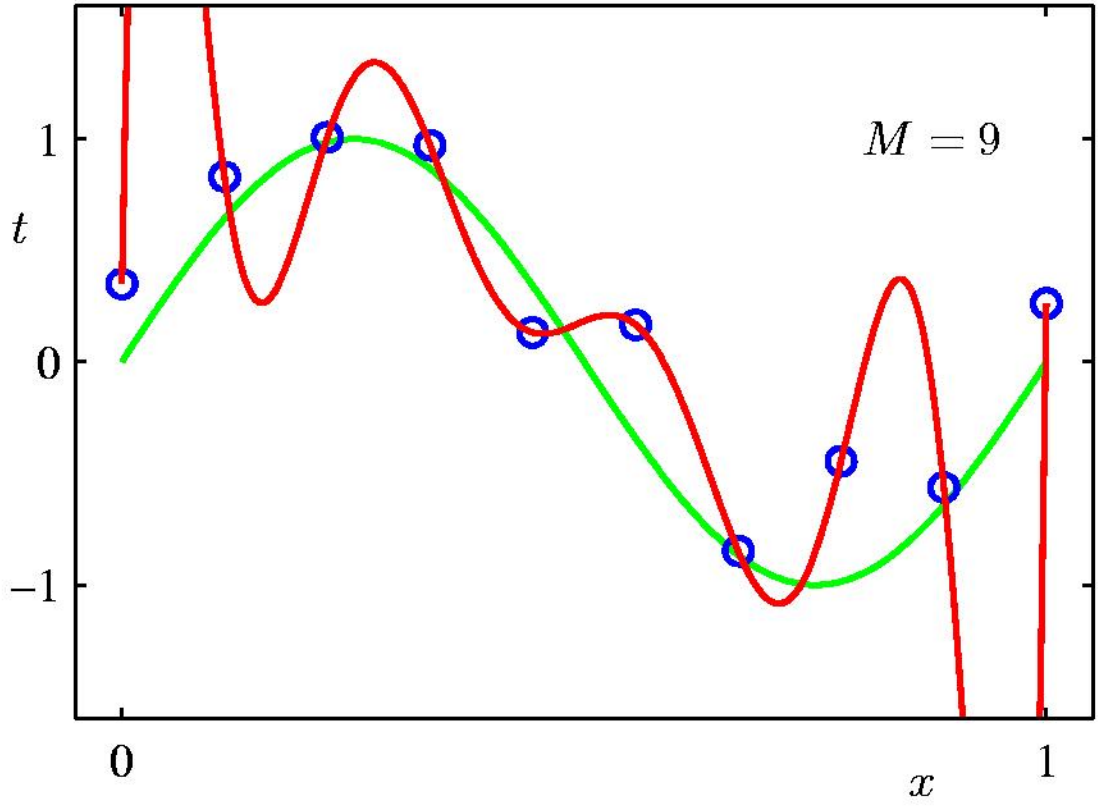
1^{st} Order Polynomial



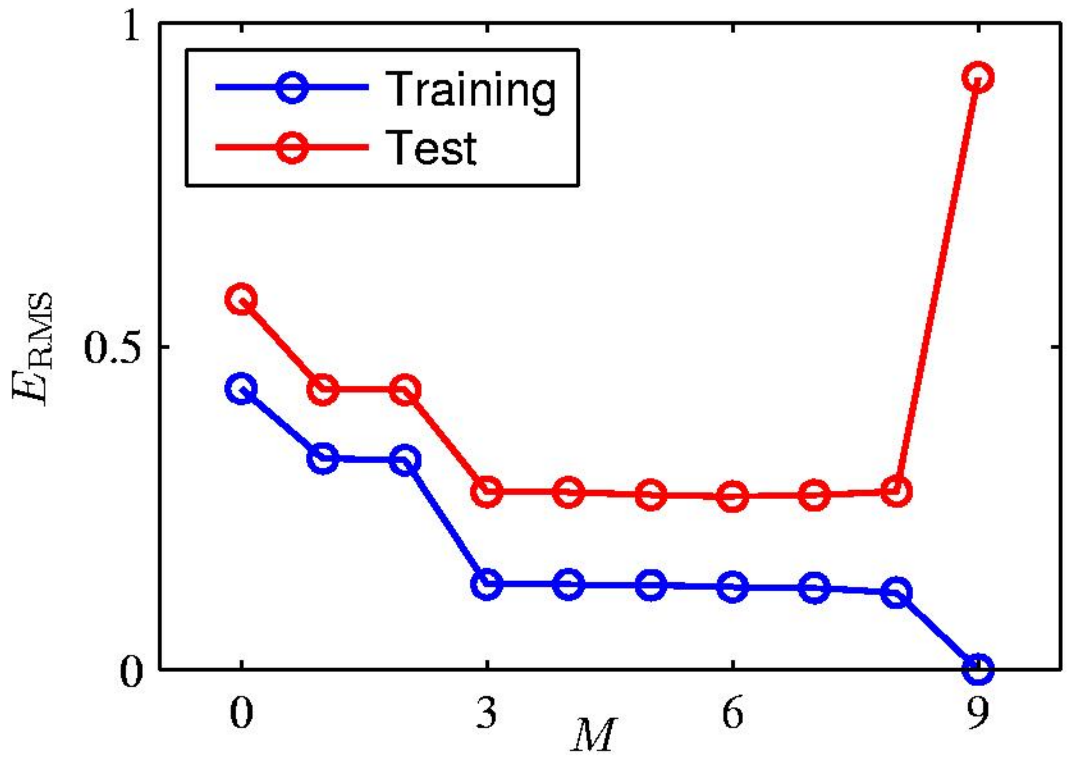
3^{rd} Order Polynomial



9th Order Polynomial



Over-fitting



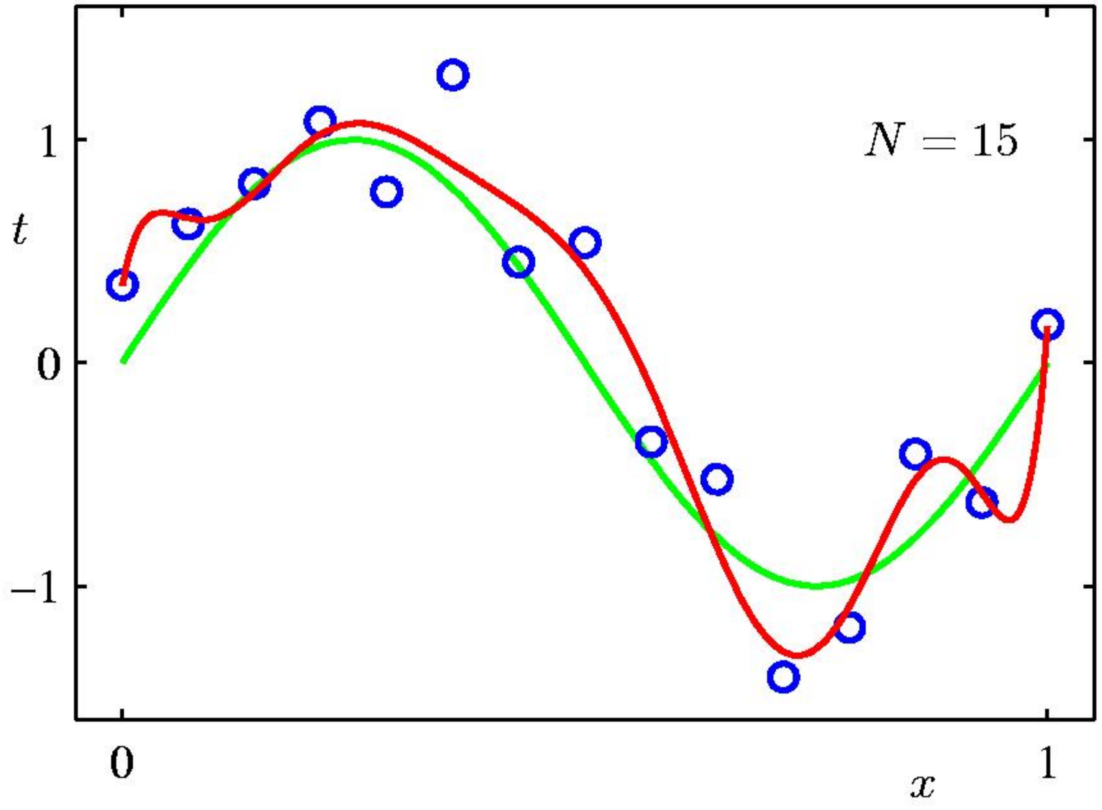
Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$

Polynomial Coefficients

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	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

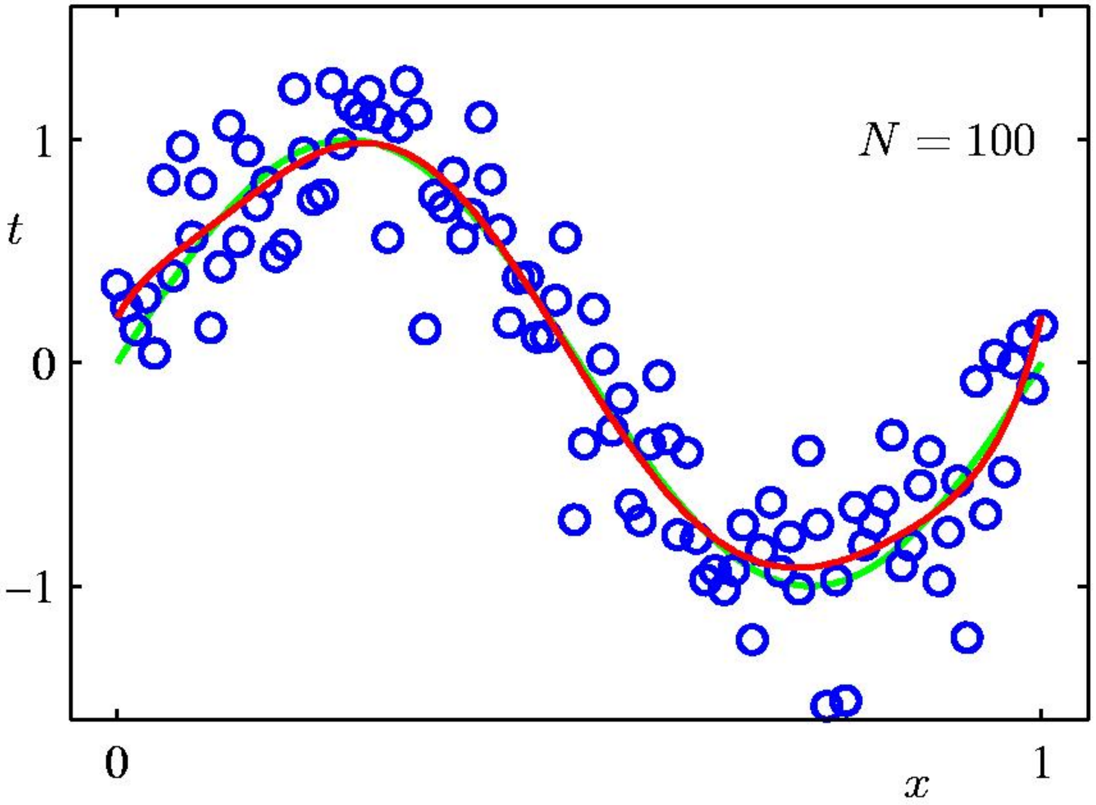
Data Set Size: $N = 15$

9th Order Polynomial



Data Set Size: $N = 100$

9th Order Polynomial

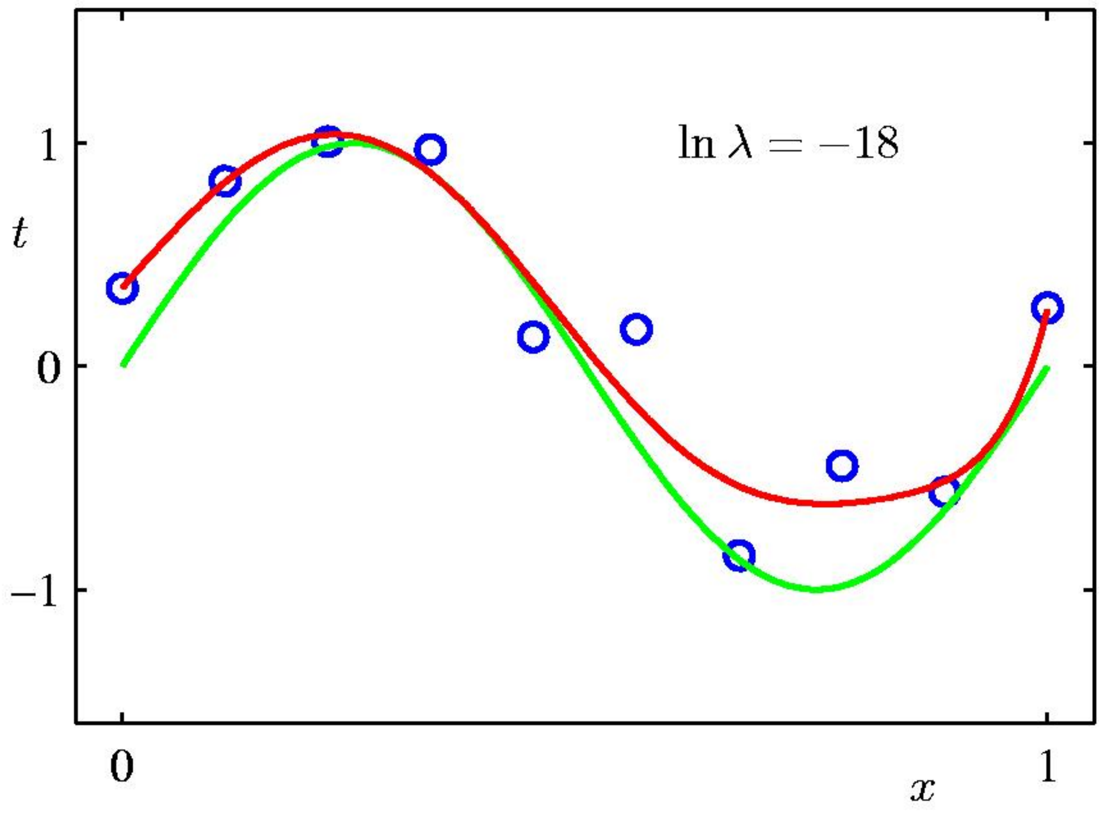


Regularization

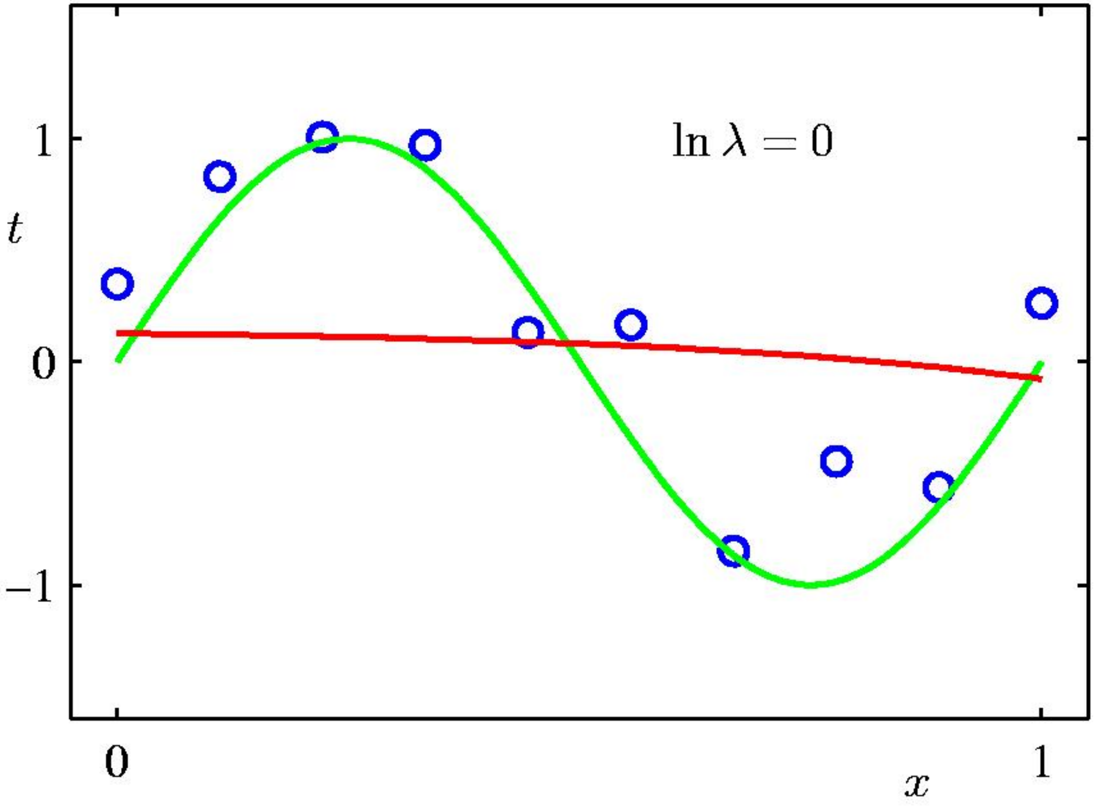
- Penalize large coefficient values

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

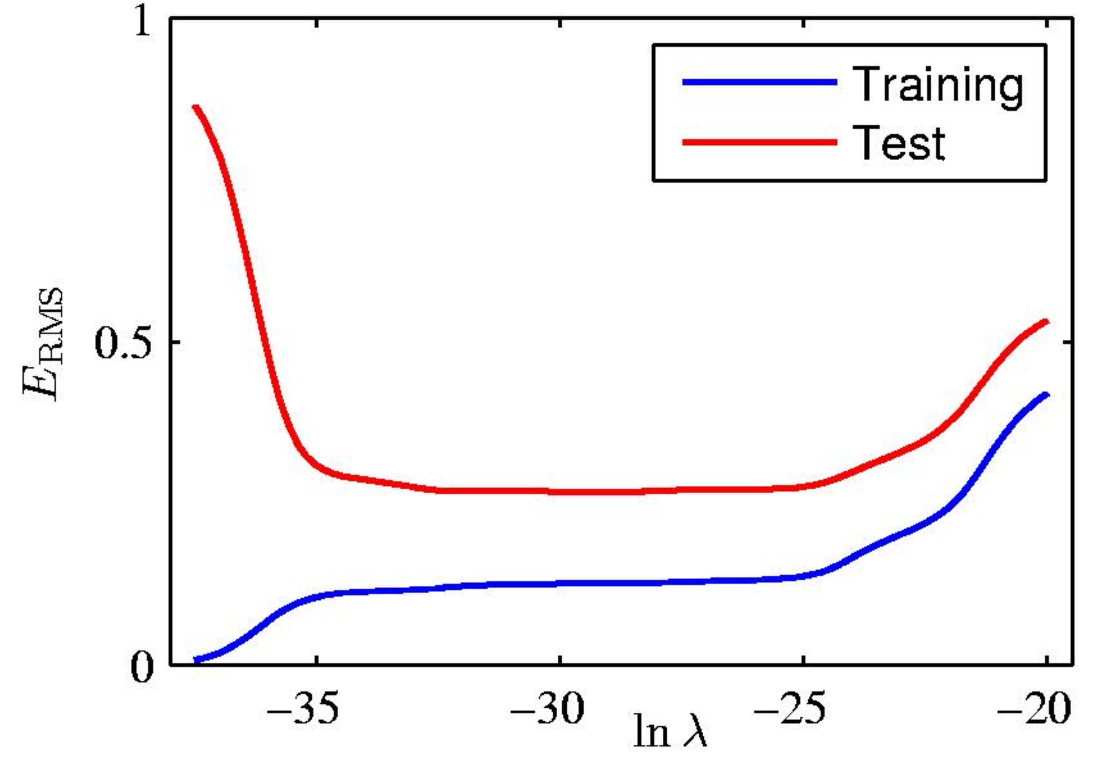
Regularization: $\ln \lambda = -18$



Regularization: $\ln \lambda = 0$



Regularization: E_{RMS} vs. $\ln\lambda$



Polynomial Coefficients

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

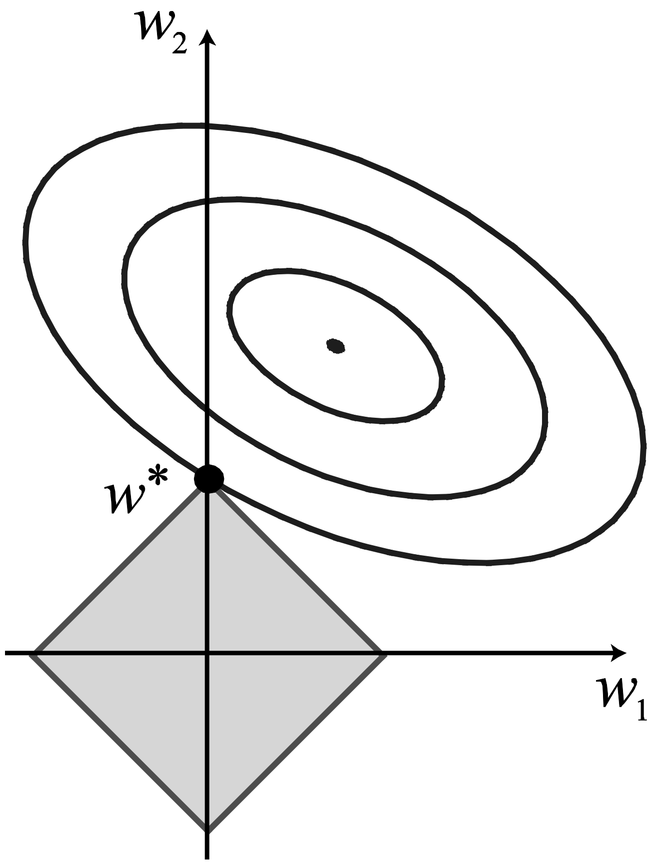
Over-fitting

- MLE estimate: Some weights are large because of chance (coincidental regularities)
- Regularize!!
  - Penalize high weights (complex hypothesis)
  - Minimize cost: Loss + Complexity

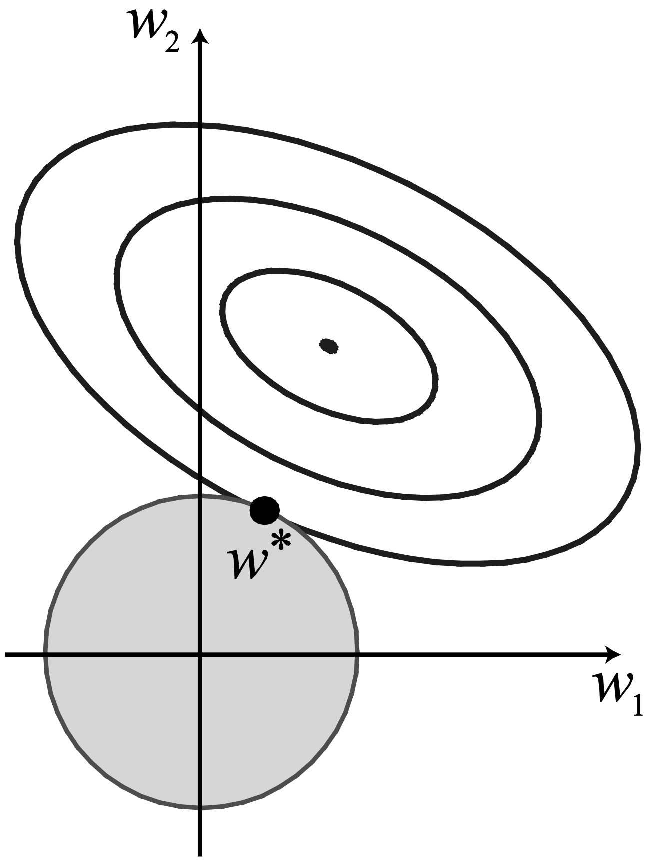
$$JR(\mathbf{w}) = \sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{n} w_j x_{i,j} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{m} |w_j|^q$$

q=1: L1 regularization (Lasso)
q=2: L2 regularization (Ridge)

Regularization



L1



L2

Overfitting

- **Modify J function to include model complexity parameter**

$$J(W) = \sum_{i=1}^{m} \left(y_i - \sum_{j=0}^{n} w_j \; x_{i,j}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{m} \left|w_j\right|^2$$

- ***Solve for W vector by taking derivative of J() w.r.t to W and set to zero.***

$$W = \left(X^T X + \lambda I_n\right)^{-1} X^T Y$$