
Lecture 2

Linear Regression

Outline

■ Linear Regression

- Simple Example
- Model

■ Learning

- Gradient Descent
- SGD
- Closed Form

■ Advanced Topics

- Probabilistic Interpretation of LMS
- L2 Regularization
- L1 Regularization

Linear Regression

- Given a data set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$
where $\mathbf{x}_i = (x_{i1}; x_{i2}; \dots; x_{id})$ $y_i \in \mathbb{R}$
- The aim of linear regression
 - Learn a linear model that can accurately predict the real-valued output labels
- For discrete variables
 - An ordinal relationship exists between values
 - Convert the variables into real-valued variables
 - No ordinal relationship exists
 - Convert the discrete variable with k possible values into a k -dimensional vector

Linear Regression

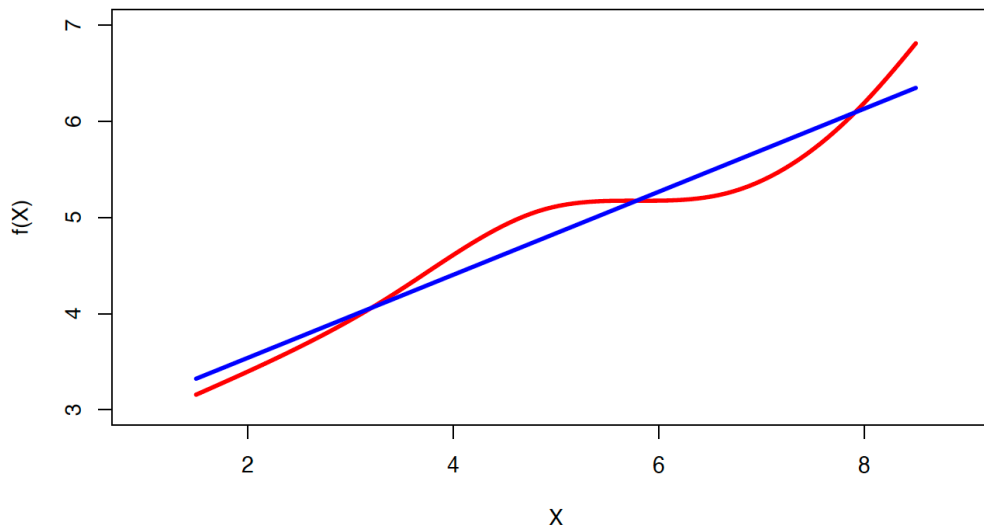
- Linear regression aims to learn the function:

$$f(x) = wx + b, \text{ such that } f(x_i) \simeq y_i, i = 1, \dots, m$$

- True regression functions are never linear!

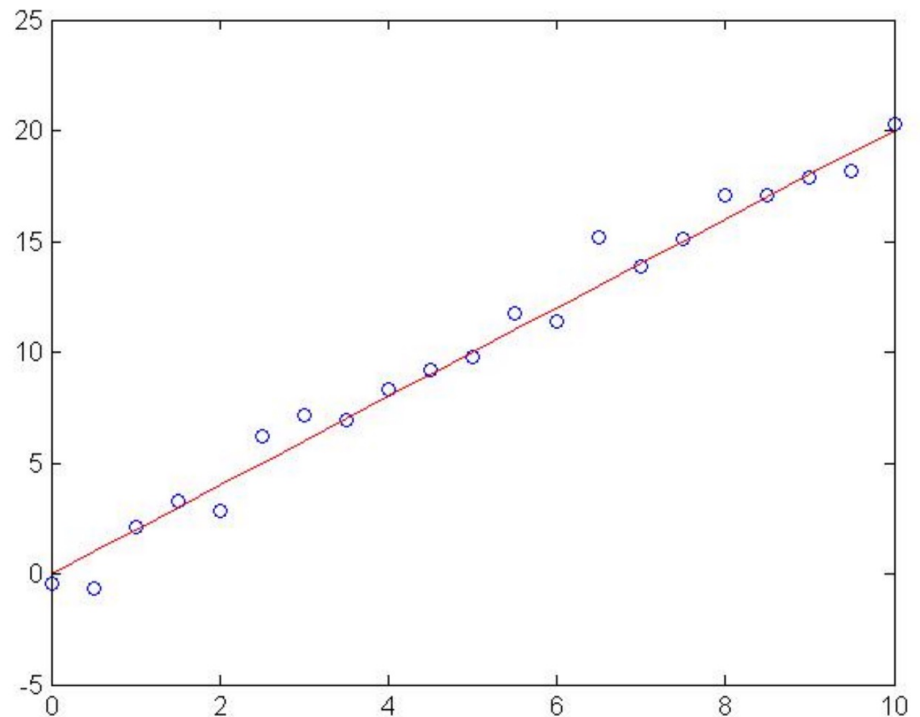
$$y = wx + b + \epsilon$$

where ϵ is an error term of measurement error or other noise



Linear Regression example

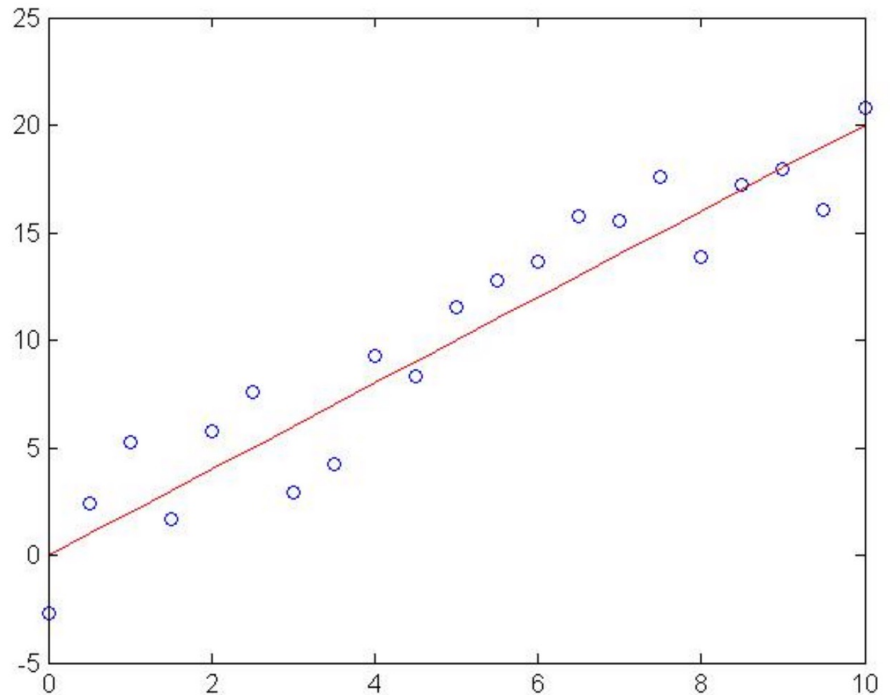
- Generated: $w=2$
- Recovered: $w=2.03$
- Noise: $\text{std}=1$



Slide courtesy of William Cohen

Linear Regression example

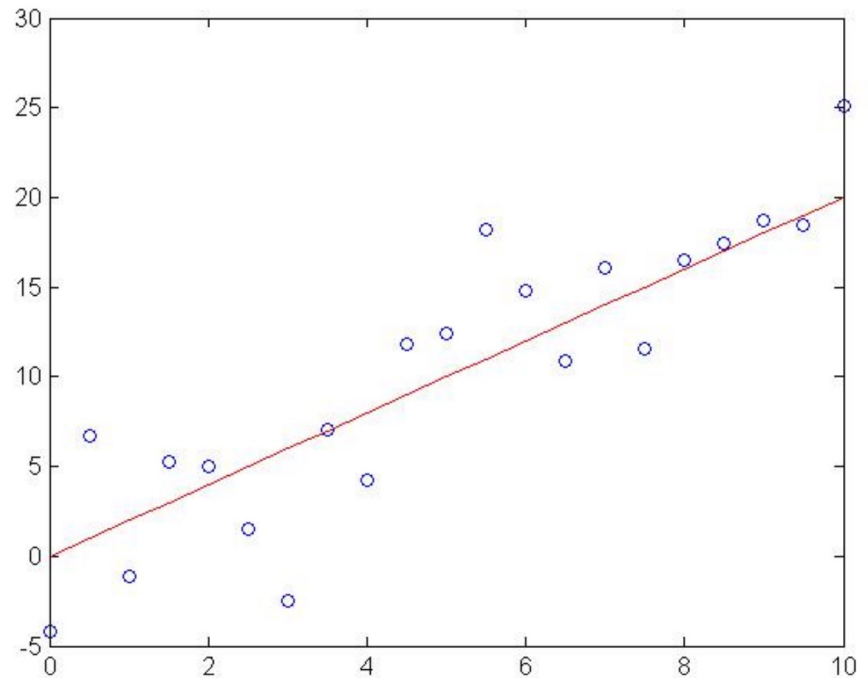
- Generated: $w=2$
- Recovered: $w=2.05$
- Noise: $\text{std}=2$



Slide courtesy of William Cohen

Linear Regression example

- Generated: $w=2$
- Recovered: $w=2.08$
- Noise: $\text{std}=4$



Slide courtesy of William Cohen

Linear Regression

Data: Inputs are continuous vectors of length d . Outputs are continuous scalars.

$$\mathcal{D} = \left\{ \mathbf{x}^{(i)}, y^{(i)} \right\}_{i=1}^m \text{ where } \mathbf{x} \in \mathbb{R}^d \text{ and } y \in \mathbb{R}$$

Prediction: Output is a linear function of the inputs.

$$\hat{y} = f_{\theta}(\mathbf{x}) = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d$$

$$\hat{y} = f_{\theta}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} \quad (\text{We assume } x_1 \text{ is } 1)$$

Learning: finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Least Squares

- Our goal is to learn a linear function:

$$f(x) = wx + b, \text{ such that } f(x_i) \simeq y_i, i = 1, \dots, m$$

- We minimize the sum of the squares:

$$\begin{aligned} J(\theta) &\Rightarrow (w^*, b^*) = \arg \min_{(w,b)} \sum_{i=1}^m (f(x_i) - y_i)^2 \\ \theta = \begin{bmatrix} w \\ b \end{bmatrix} &= \arg \min_{(w,b)} \sum_{i=1}^m (y_i - wx_i - b)^2 \end{aligned}$$

Reduces distance between true measurements and predicted hyperplane (line in 1D)

Least Squares

Learning: Three approaches to solving $\theta^* = \arg \min_{\theta} J(\theta)$

- ❑ **Approach 1: Gradient Descent**
(take larger – more certain – steps opposite the gradient)
- ❑ **Approach 2: Stochastic Gradient Descent (SGD)**
(take many small steps opposite the gradient)
- ❑ **Approach 3: Closed Form**
(set derivatives equal to zero and solve for parameters)

Linear Regression – loss function

- Minimize mean-squared error (MSE):

Loss function: **How much \hat{y} differs from the true y**

$$E_{(w,b)} = \sum_{i=1}^m (y_i - wx_i - b)^2$$

- Calculate the derivatives of $E_{(w,b)}$ with respect to w and b :

$$\frac{\partial E_{(w,b)}}{\partial w} = 2 \left(w \sum_{i=1}^m x_i^2 - \sum_{i=1}^m (y_i - b)x_i \right)$$

$$\frac{\partial E_{(w,b)}}{\partial b} = 2 \left(mb - \sum_{i=1}^m (y_i - wx_i) \right)$$

Linear Regression - Least Square Method

- We have the closed-form solutions

$$w = \frac{\sum_{i=1}^m y_i (x_i - \bar{x})}{\sum_{i=1}^m x_i^2 - \frac{1}{m} \left(\sum_{i=1}^m x_i \right)^2}$$

$$b = \frac{1}{m} \sum_{i=1}^m (y_i - wx_i)$$

where $\bar{x} = \frac{1}{m} \sum_{i=1}^m x_i$

Multivariate Linear Regression

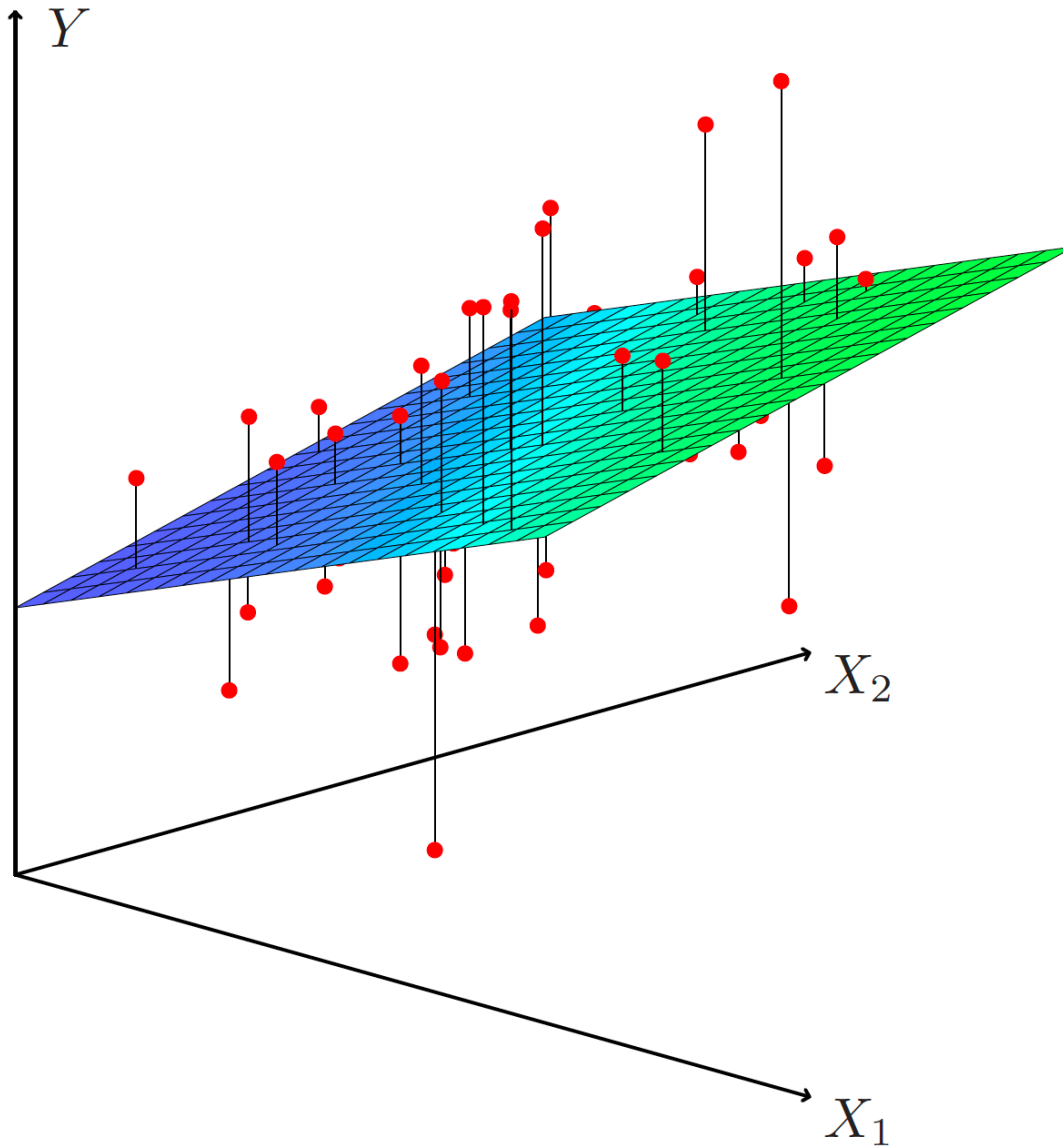
- Given a data set

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$$

$$\mathbf{x}_i = (x_{i1}; x_{i2}; \dots; x_{id}) \quad y_i \in \mathbb{R}$$

- The objective of multivariate linear regression

Find $f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i + b$ such that $f(\mathbf{x}_i) \simeq y_i$



Multivariate Linear Regression

- Rewrite w and b as $\hat{w} = (w; b)$, the data set is represented as

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} & 1 \\ x_{21} & x_{22} & \cdots & x_{2d} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{md} & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^T & 1 \\ \mathbf{x}_2^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^T & 1 \end{pmatrix}$$

$$\mathbf{y} = (y_1; y_2; \cdots; y_m)$$

Multivariate Linear Regression - Least Square Method

□ Least square method

$$\hat{\mathbf{w}}^* = \arg \min_{\hat{\mathbf{w}}} (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})$$

Let $E_{\hat{\mathbf{w}}} = (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})$ and find the derivative with respect to $\hat{\mathbf{w}}$

$$\frac{\partial E_{\hat{\mathbf{w}}}}{\partial \hat{\mathbf{w}}} = 2\mathbf{X}^T (\mathbf{X}\hat{\mathbf{w}} - \mathbf{y})$$

The closed-form solution of $\hat{\mathbf{w}}$ can be obtained by making the equation equal to 0.

Multivariate Linear Regression - Least Square Method

- If $\mathbf{X}^T \mathbf{X}$ is a full-rank matrix or a positive definite matrix, then

$$\hat{\mathbf{w}}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

where $(\mathbf{X}^T \mathbf{X})^{-1}$ is the inverse of $\mathbf{X}^T \mathbf{X}$, the learned multivariate linear regression model is

$$f(\hat{\mathbf{x}}_i) = \hat{\mathbf{x}}_i^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

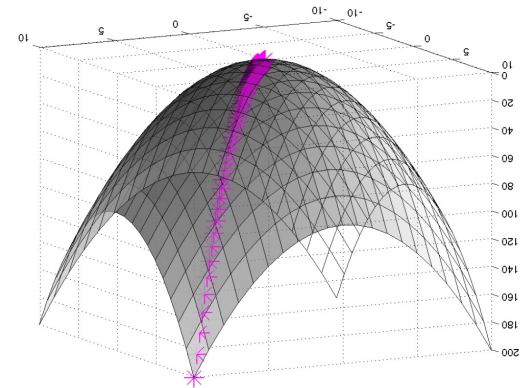
- $\mathbf{X}^T \mathbf{X}$ is often not full-rank
 - gradient descent (which is more broadly applicable)
 - pseudo-inverse

Learning as optimization

- The main idea
 - Make two assumptions:
 - the classifier is linear
 - we want to find the classifier θ that “fits the data best”
 - Formalize as an optimization problem
 - Pick a loss function $J(\theta, \mathcal{D})$, and find $\operatorname{argmin}_{\theta} J(\theta)$
 - OR: Pick a “goodness” function, often $\Pr(\mathcal{D}|\theta)$, and find the $\operatorname{argmax}_{\theta} f_{\mathcal{D}}(\theta)$

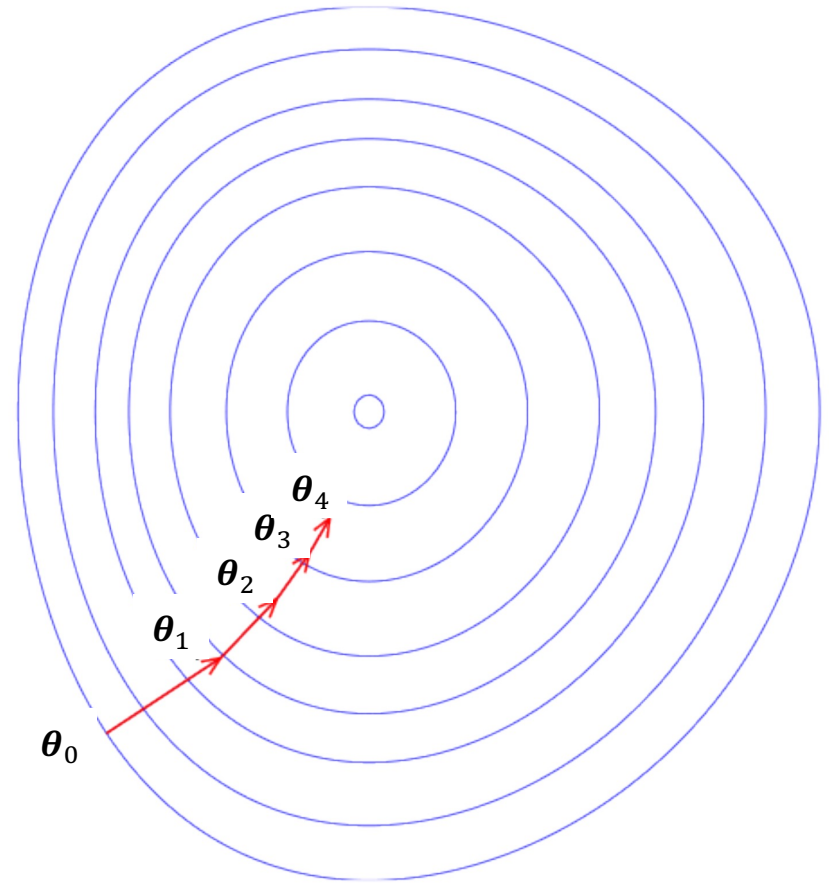
Learning as optimization with gradient ascent

- Goal: Learn the parameter θ of ...
- Dataset: $\mathcal{D} = \{(\mathbf{x}_1, y_1) \dots, (\mathbf{x}_m, y_m)\}$
- Use your model to define
 - $\Pr(\mathcal{D}|\theta) = \dots$
- Set θ to maximize Likelihood
 - Usually we use numeric methods to find the optimum
 - i.e., **gradient ascent**: repeatedly take a small step in the direction of the gradient (direction of **fastest** increase in the function).



Gradient ascent

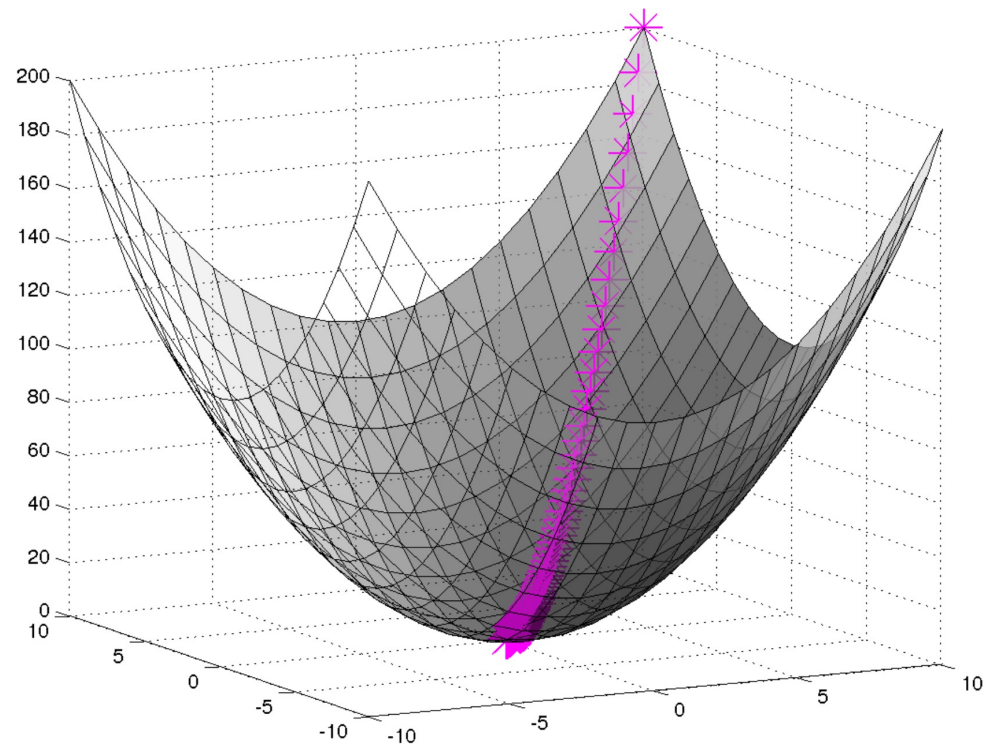
- To find $\operatorname{argmax}_{\theta} f(\theta)$:
 - Start with θ_0
 - For $t = 1 \dots$
 - $\theta_{t+1} = \theta_t + \alpha f'(\theta_t)$
 - where α is a **learning rate**.
 - The larger it is, the faster θ changes.
 - The values of α are typically small, e.g. 0.01 or 0.0001



Slide courtesy of William Cohen

Gradient descent

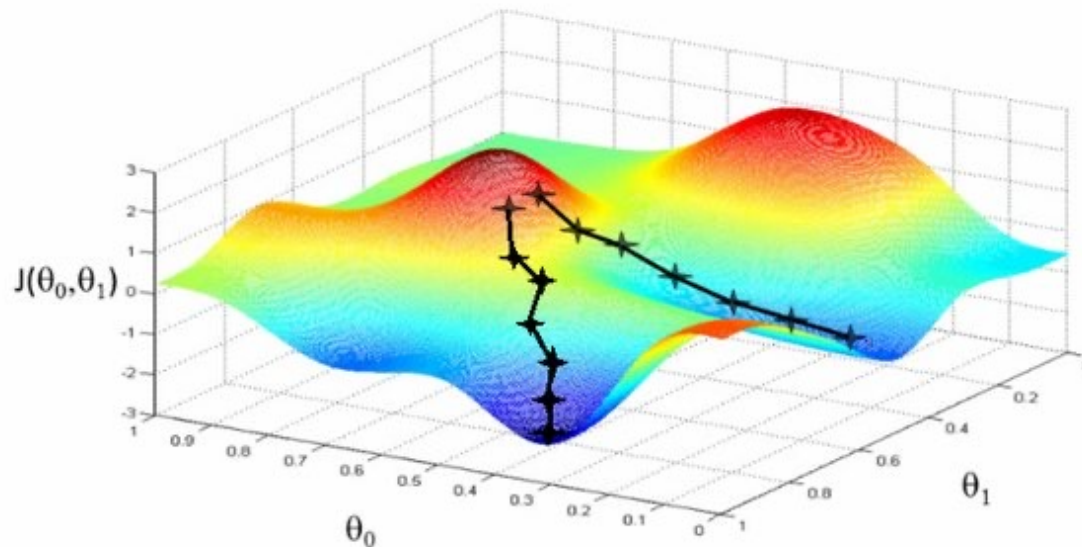
- Likelihood: ascent
- Loss: descent



Slide courtesy of William Cohen

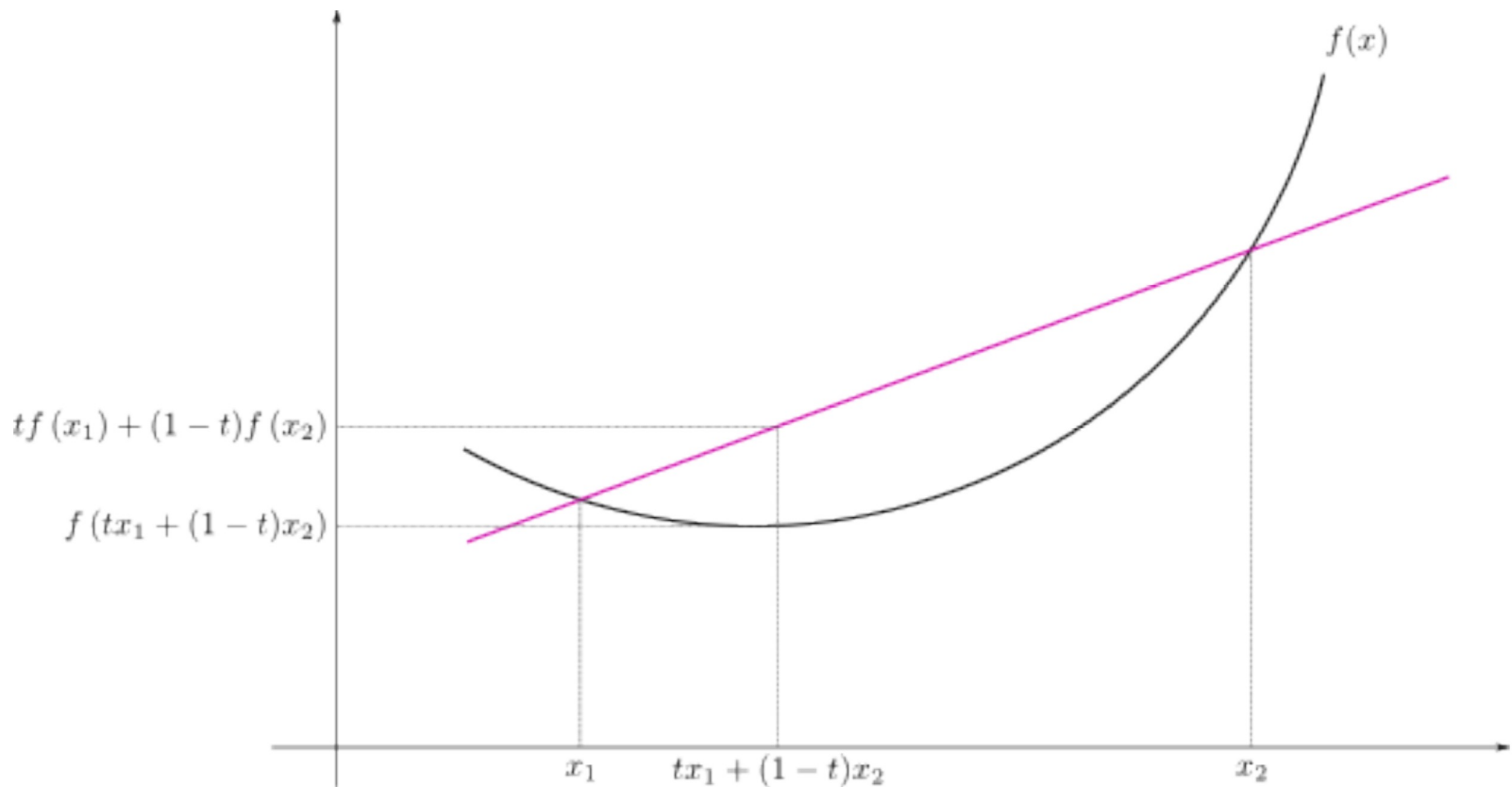
Pros and cons of gradient descent

- Simple and often quite effective
- Often very scalable
- Only applies to smooth functions (**differentiable**)
- Might find a **local minimum**, rather than a global one



Pros and cons of gradient descent

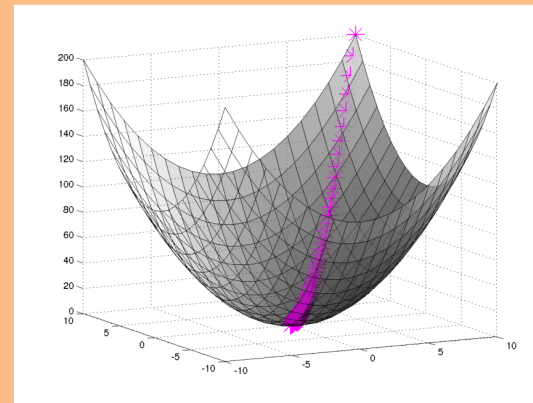
There is only one local optimum if the function is convex



Gradient Descent

Algorithm 1 Gradient Descent

```
1: procedure GD( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:      $\theta \leftarrow \theta - \alpha \nabla_{\theta} J(\theta)$ 
5:   return  $\theta$ 
```



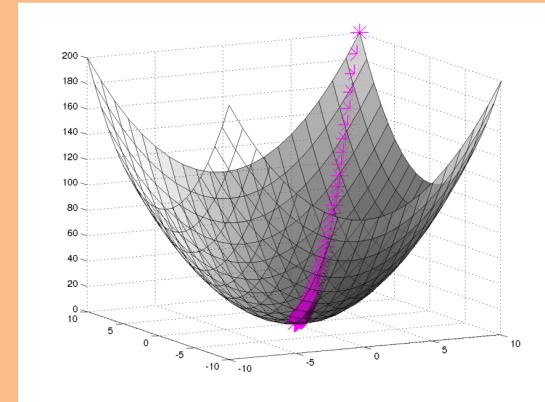
In order to apply GD to Linear Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \frac{d}{d\theta_1} J(\theta) \\ \frac{d}{d\theta_2} J(\theta) \\ \vdots \\ \frac{d}{d\theta_N} J(\theta) \end{bmatrix}$$

Gradient Descent

Algorithm 1 Gradient Descent

```
1: procedure GD( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:      $\theta \leftarrow \theta - \alpha \nabla_{\theta} J(\theta)$ 
5:   return  $\theta$ 
```



There are many possible ways to detect **convergence**. For example, we could check whether the L2 norm of the gradient is below some small tolerance.

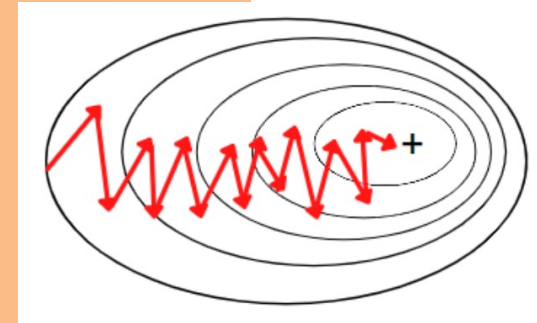
$$\|\nabla_{\theta} J(\theta)\|_2 \leq \epsilon$$

Alternatively we could check that the reduction in the objective function from one iteration to the next is small.

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent (SGD)

```
1: procedure SGD( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:     for  $i \in \text{shuffle}(\{1, 2, \dots, N\})$  do
5:        $\theta \leftarrow \theta - \alpha \nabla_{\theta} J^{(i)}(\theta)$ 
6:   return  $\theta$ 
```



Applied to Linear Regression, SGD is called the **Least Mean Squares (LMS)** algorithm

We need a per-example objective:

$$\text{Let } J(\theta) = \sum_{i=1}^m J^{(i)}(\theta) \quad \text{where } J^{(i)}(\theta) = \frac{1}{2} (\theta^T \mathbf{x}^{(i)} - y^{(i)})^2.$$

Partial Derivatives for Linear Reg.

$$\text{Let } J(\boldsymbol{\theta}) = \sum_{i=1}^m J^{(i)}(\boldsymbol{\theta}) \quad \text{where } J^{(i)}(\boldsymbol{\theta}) = \frac{1}{2} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2.$$

$$\begin{aligned} \frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta}) &= \frac{d}{d\theta_k} \frac{1}{2} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 \\ &= \frac{1}{2} \frac{d}{d\theta_k} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 \\ &= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \frac{d}{d\theta_k} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \\ &= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \frac{d}{d\theta_k} \left(\sum_{k=1}^K \theta_k x_k^{(i)} - y^{(i)} \right) \\ &= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)} \end{aligned}$$

Partial Derivatives for Linear Reg.

$$\text{Let } J(\boldsymbol{\theta}) = \sum_{i=1}^m J^{(i)}(\boldsymbol{\theta}) \quad \text{where } J^{(i)}(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2.$$

$$\frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta}) = (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}$$

Used by SGD
(aka. LMS)

$$\begin{aligned} \frac{d}{d\theta_k} J(\boldsymbol{\theta}) &= \frac{d}{d\theta_k} \sum_{i=1}^N J^{(i)}(\boldsymbol{\theta}) \\ &= \sum_{i=1}^N (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)} \end{aligned}$$

Used by
Gradient
Descent

Least Squares

Learning: Three approaches to solving $\theta^* = \arg \min_{\theta} J(\theta)$

- ❑ **Approach 1: Gradient Descent**
(take larger – more certain – steps opposite the gradient)
 - pros: conceptually simple, guaranteed convergence
 - cons: batch, often slow to converge
- ❑ **Approach 2: Stochastic Gradient Descent (SGD)**
(take many small steps opposite the gradient)
 - pros: memory efficient, fast convergence, less prone to local optima
 - cons: convergence in practice requires tuning and fancier variants
- ❑ **Approach 3: Closed Form**
(set derivatives equal to zero and solve for parameters)
 - pros: one shot algorithm!
 - cons: does not scale to large datasets (matrix inverse is bottleneck)

Maximum-Likelihood Estimation

Find: optimal (MLE) parameter θ of a binomial

Dataset: $\mathcal{D} = \{x_1, \dots, x_n\}$, x_i is 0 or 1, k of them are 1

$$P(\mathcal{D} | \theta) = \text{const} \prod_i \theta^{x_i} (1 - \theta)^{1 - x_i} = \theta^k (1 - \theta)^{n - k}$$

$$\frac{d}{d\theta} P(\mathcal{D} | \theta) = \frac{d}{d\theta} \left(\theta^k (1 - \theta)^{n - k} \right)$$

$$= \left(\frac{d}{d\theta} \theta^k \right) (1 - \theta)^{n - k} + \theta^k \frac{d}{d\theta} (1 - \theta)^{n - k}$$

$$= k\theta^{k-1} (1 - \theta)^{n - k} + \theta^k (n - k)(1 - \theta)^{n - k - 1} (-1)$$

$$= \theta^{k-1} (1 - \theta)^{n - k - 1} \left(\underbrace{k(1 - \theta)}_{\text{blue}} - \underbrace{\theta(n - k)}_{\text{blue}} \right)$$


Slide courtesy of William Cohen


Maximum-Likelihood Estimation

Find: optimal (MLE) parameter θ of a binomial

Dataset: $\mathcal{D} = \{x_1, \dots, x_n\}$, x_i is 0 or 1, k of them are 1

$$\frac{\partial}{\partial \theta} P(D) = \theta^{k-1} (1 - \theta)^{n-k-1} (k(1 - \theta) - \theta(n - k)) = 0$$


$$\theta = 0$$


$$\theta = 1$$


$$k - k\theta - n\theta + k\theta = 0$$

$$\rightarrow n\theta = k$$

$$\rightarrow \theta = k/n$$

Slide courtesy of William Cohen

Maximum-Likelihood Estimation

$$\mathbf{x}_k \sim p(\mathbf{x}|\boldsymbol{\theta})$$

$$(k = 1, \dots, n)$$

$\boldsymbol{\theta}$: Parameters to be estimated

\mathcal{D} : A set of *i.i.d.* examples $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$

The objective function

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{k=1}^n p(\mathbf{x}_k|\boldsymbol{\theta})$$



The likelihood of $\boldsymbol{\theta}$ w.r.t. the set of observed examples

The maximum-likelihood estimation

$$\hat{\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta})$$

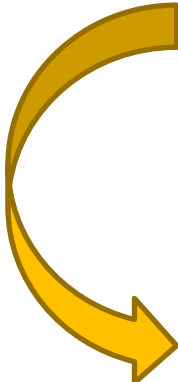



Intuitively, $\hat{\boldsymbol{\theta}}$ best agrees with the actually observed examples


Maximum-Likelihood Estimation (Cont.)

Gradient Operator (梯度算子)

- ✓ Let $\boldsymbol{\theta} = [\theta_1, \dots, \theta_p]^T \in \mathbb{R}^p$ be a p -dimensional vector
- ✓ Let $f : \mathbb{R}^p \rightarrow \mathbb{R}$ be p -variate real-valued function over $\boldsymbol{\theta}$

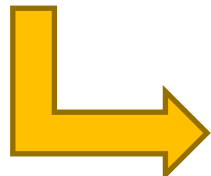

$$\nabla_{\boldsymbol{\theta}} \equiv \begin{bmatrix} \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_p} \end{bmatrix}$$
$$f(\boldsymbol{\theta}) = \theta_1^2 + 3\theta_1\theta_2$$

$$\nabla_{\boldsymbol{\theta}} f = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} 2\theta_1 + 3\theta_2 \\ 3\theta_1 \end{bmatrix}$$

$l(\boldsymbol{\theta}) = \ln p(\mathcal{D}|\boldsymbol{\theta})$ is named as the **log-likelihood function**


$$\hat{\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta}) \quad \hat{\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\theta}} l(\boldsymbol{\theta})$$

Maximum-Likelihood Estimation (Cont.)

$$l(\boldsymbol{\theta}) = \ln p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{k=1}^n \ln p(\mathbf{x}_k|\boldsymbol{\theta})$$


$$\underline{\underline{\nabla_{\boldsymbol{\theta}} l}} = \nabla_{\boldsymbol{\theta}} \left(\sum_{k=1}^n \ln p(\mathbf{x}_k|\boldsymbol{\theta}) \right) = \sum_{k=1}^n \nabla_{\boldsymbol{\theta}} \underline{\underline{\ln p(\mathbf{x}_k|\boldsymbol{\theta})}}$$

p -dimensional vector with each component being a function over $\boldsymbol{\theta}$

p -variate real-valued function over $\boldsymbol{\theta}$ (not over \mathbf{x}_k)

Necessary conditions for ML estimate $\hat{\boldsymbol{\theta}}$

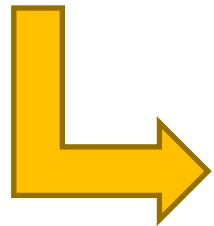
$$\nabla_{\boldsymbol{\theta}} l \big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} = \mathbf{0} \text{ (a set of } p \text{ equations)}$$

The Gaussian Case: Unknown μ

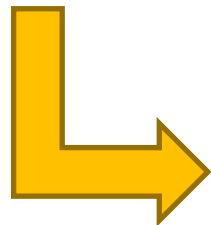
$$\mathbf{x}_k \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$
$$(k = 1, \dots, n)$$

suppose $\boldsymbol{\Sigma}$ is known  $\boldsymbol{\theta} = \{\boldsymbol{\mu}\}$

$$p(\mathbf{x}_k | \boldsymbol{\mu}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x}_k - \boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1} (\mathbf{x}_k - \boldsymbol{\mu}) \right]$$



$$\ln p(\mathbf{x}_k | \boldsymbol{\mu}) = -\frac{1}{2} \ln [(2\pi)^d |\boldsymbol{\Sigma}|] - \frac{1}{2} (\mathbf{x}_k - \boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1} (\mathbf{x}_k - \boldsymbol{\mu})$$
$$= -\frac{1}{2} \ln [(2\pi)^d |\boldsymbol{\Sigma}|] - \frac{1}{2} \mathbf{x}_k^t \boldsymbol{\Sigma}^{-1} \mathbf{x}_k + \boldsymbol{\mu}^t \boldsymbol{\Sigma}^{-1} \mathbf{x}_k - \frac{1}{2} \boldsymbol{\mu}^t \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}$$



$$\nabla_{\boldsymbol{\mu}} \ln p(\mathbf{x}_k | \boldsymbol{\mu}) = \boldsymbol{\Sigma}^{-1} (\mathbf{x}_k - \boldsymbol{\mu})$$

The Gaussian Case: Unknown μ

(Cont.)

$$l(\mu) = \sum_{k=1}^n \ln p(\mathbf{x}_k | \mu)$$

Intuitive result

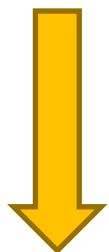
ML estimate for the unknown μ is just the arithmetic average of training samples – *sample mean*



$$\nabla_{\mu} \ln p(\mathbf{x}_k | \mu) = \Sigma^{-1}(\mathbf{x}_k - \mu)$$

$$\nabla_{\mu} l = \sum_{k=1}^n \Sigma^{-1}(\mathbf{x}_k - \mu)$$

$$\hat{\mu} = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_k$$



$\nabla_{\mu} l = \mathbf{0}$ (necessary condition

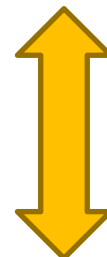
for ML estimate $\hat{\mu}$)

Multiply Σ on both sides

$$\sum_{k=1}^n \Sigma^{-1}(\mathbf{x}_k - \hat{\mu}) = \mathbf{0}$$



$$\sum_{k=1}^n (\mathbf{x}_k - \hat{\mu}) = \mathbf{0}$$



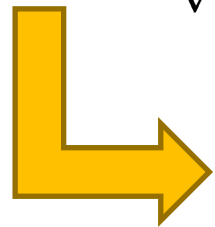
The Gaussian Case: Unknown μ and Σ

$$\mathbf{x}_k \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$
$$(k = 1, \dots, n)$$

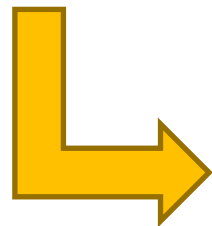
$\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ unknown $\longrightarrow \boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$

Consider *univariate* case

$$p(x_k | \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x - \mu)^2}{2\sigma^2}\right] \quad \left(\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} \mu \\ \sigma^2 \end{bmatrix}\right)$$



$$\ln p(x_k | \boldsymbol{\theta}) = -\frac{1}{2} \ln 2\pi\theta_2 - \frac{1}{2\theta_2} (x_k - \theta_1)^2$$



$$\nabla_{\boldsymbol{\theta}} \ln p(x_k | \boldsymbol{\theta}) = \begin{bmatrix} \frac{1}{\theta_2} (x_k - \theta_1) \\ -\frac{1}{2\theta_2} + \frac{(x_k - \theta_1)^2}{2\theta_2^2} \end{bmatrix}$$

The Gaussian Case: Unknown μ and Σ (Cont.)

$$l(\boldsymbol{\theta}) = \sum_{k=1}^n \ln p(x_k | \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} \ln p(x_k | \boldsymbol{\theta}) =$$

$$\begin{bmatrix} \frac{1}{\theta_2} (x_k - \theta_1) \\ -\frac{1}{2\theta_2} + \frac{(x_k - \theta_1)^2}{2\theta_2^2} \end{bmatrix}$$

$$\nabla_{\boldsymbol{\theta}} l = \begin{bmatrix} \sum_{k=1}^n \frac{1}{\theta_2} (x_k - \theta_1) \\ \sum_{k=1}^n \left(-\frac{1}{2\theta_2} + \frac{(x_k - \theta_1)^2}{2\theta_2^2} \right) \end{bmatrix}$$

$$\sum_{k=1}^n \frac{1}{\hat{\theta}_2} (x_k - \hat{\theta}_1) = 0$$

$$-\sum_{k=1}^n \frac{1}{\hat{\theta}_2} + \sum_{k=1}^n \frac{(x_k - \hat{\theta}_1)^2}{\hat{\theta}_2^2} = 0$$

$\nabla_{\boldsymbol{\theta}} l = 0$ (necessary condition
for ML estimate $\hat{\theta}_1$ and $\hat{\theta}_2$)

The Gaussian Case: Unknown μ and Σ (Cont.)

$$\sum_{k=1}^n \frac{1}{\hat{\theta}_2} (x_k - \hat{\theta}_1) = 0 \quad \longrightarrow \quad \sum_{k=1}^n (x_k - \hat{\theta}_1) = 0 \quad \longrightarrow \quad \hat{\theta}_1 = \frac{1}{n} \sum_{k=1}^n x_k$$

$$-\sum_{k=1}^n \frac{1}{\hat{\theta}_2} + \sum_{k=1}^n \frac{(x_k - \hat{\theta}_1)^2}{\hat{\theta}_2^2} = 0 \quad \longrightarrow \quad \hat{\theta}_2 = \frac{1}{n} \sum_{k=1}^n (x_k - \hat{\theta}_1)^2$$

ML estimate in univariate case

$$\hat{\mu} = \frac{1}{n} \sum_{k=1}^n x_k \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{k=1}^n (x_k - \hat{\mu})^2$$

The Gaussian Case: Unknown μ and Σ

(Cont.)

ML estimate in *multivariate* case

Intuitive
result as well!

$$\hat{\mu} = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_k \quad \longrightarrow \quad \text{Arithmetic average of } n \text{ vectors } \mathbf{x}_k$$

$$\hat{\Sigma} = \frac{1}{n} \sum_{k=1}^n (\mathbf{x}_k - \hat{\mu})(\mathbf{x}_k - \hat{\mu})^t \quad \longrightarrow \quad \begin{array}{l} \text{Arithmetic average} \\ \text{of } n \text{ matrices} \\ (\mathbf{x}_k - \hat{\mu})(\mathbf{x}_k - \hat{\mu})^t \end{array}$$

Probabilistic Interpretation of LMS

- Let us assume that the target variable and the inputs are related by the equation:

$$y_i = \boldsymbol{\theta}^T \mathbf{x}_i + \epsilon_i$$

where ϵ is an error term of unmodeled effects or random noise

- Now assume that ϵ follows a Gaussian $\mathcal{N}(0, \sigma)$, then we have:

$$p(y_i | \mathbf{x}_i; \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

- By independence assumption:

$$L(\boldsymbol{\theta}) = \prod_{i=1}^m p(y_i | \mathbf{x}_i; \boldsymbol{\theta}) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^m \exp\left(-\frac{\sum_{i=1}^m (y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

Probabilistic Interpretation of LMS

- Hence the log-likelihood is:

$$l(\boldsymbol{\theta}) = m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^m (y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2$$

- Do you recognize the last term?

Yes it is:
$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^m (y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2$$

- Thus under independence assumption, LMS is equivalent to MLE of $\boldsymbol{\theta}$!

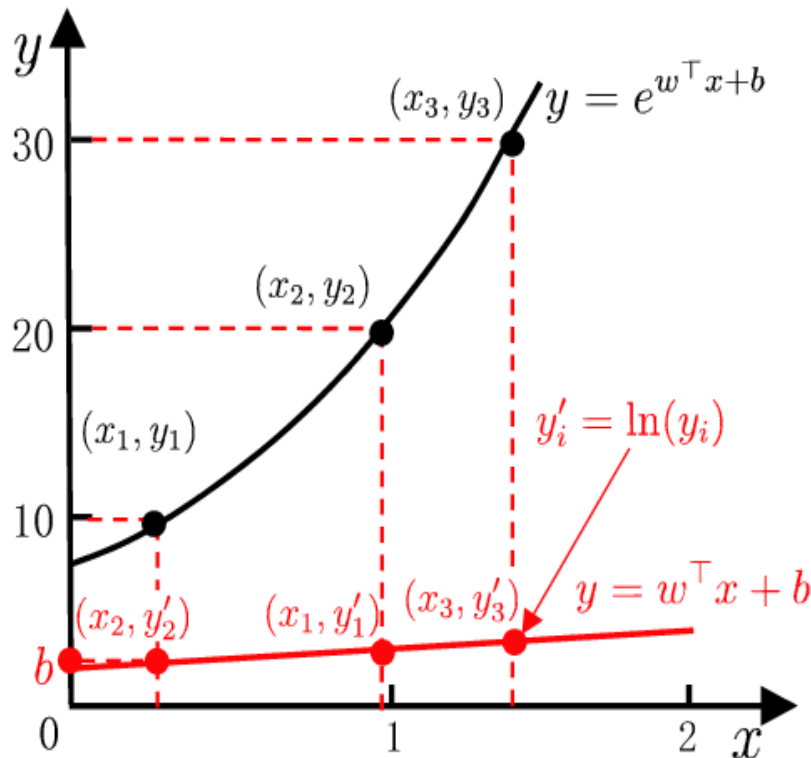
Although the model assumes a Gaussian distribution in the prediction (i.e. Gaussian noise function or error function), there is no such expectation for the inputs to the model (\mathbf{x}).

Gradient Descent

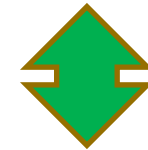
- Why gradient descent, if we can find the optimum directly?
 - GD can be applied to a much broader set of models
 - GD can be easier to implement than direct solutions, especially with automatic differentiation software
 - For regression in high-dimensional spaces, GD is more efficient than
 - direct solution (matrix inversion is an $O(D^3)$ algorithm).

Log-Linear Regression

- The logarithm of the output label can be used for approximation



$$\ln y = w^T x + b$$



$$y = w^T x + b$$

Linear Regression - Generalized Linear Model

- The general form

$$y = g^{-1}(\mathbf{w}^T \mathbf{x} + b)$$

- Where the function $g(\cdot)$ is the link function.
 - a monotonic differentiable function $g(\cdot) = \ln(\cdot)$
- Log-linear regression is a special case of generalized linear models when

Feature mappings

- We get polynomial regression for free!

- Define the feature map
$$\phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \end{pmatrix}$$

- Polynomial regression model:
$$y = \sum_{j=0}^n w_j \phi_j(x)$$

- All of the derivations and algorithms so far in this lecture remain exactly the same!

Non-Linear basis function

- So far we only used the observed values x_1, x_2, \dots
- However, linear regression can be applied in the same way to **functions** of these values
 - E.g.: add a new variable $z = x_1 x_2$ so each example becomes: x_1, x_2, \dots, z
- As long as these functions can be directly computed from the observed values the parameters are still linear in the data and the problem remains a multi-variate linear regression problem

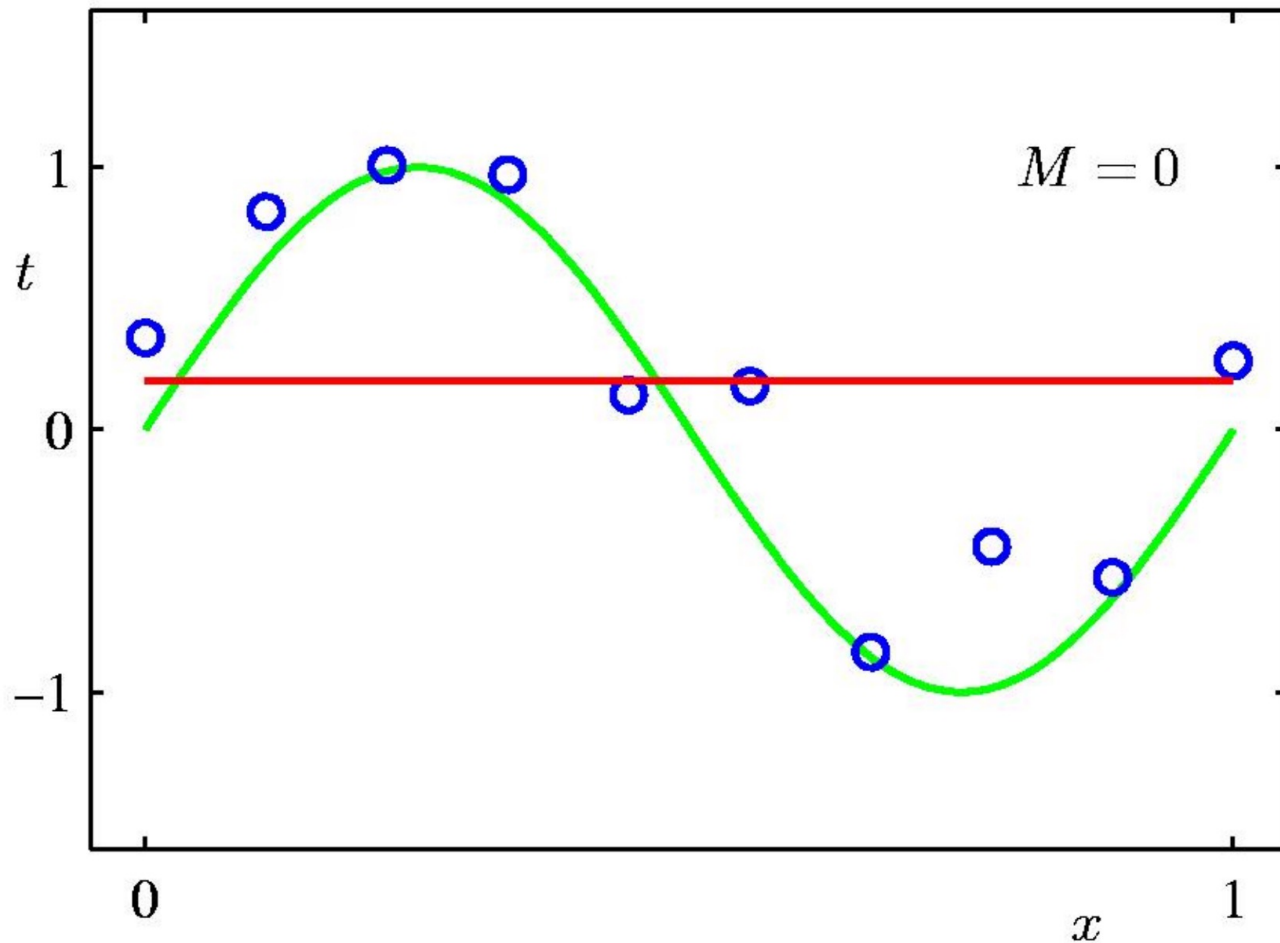
$$y = b + w_1 x_1^2 + \dots + w_d x_d^2 + \epsilon$$

Any function of the input values can be used. The solution for the parameters of the regression remains the same.

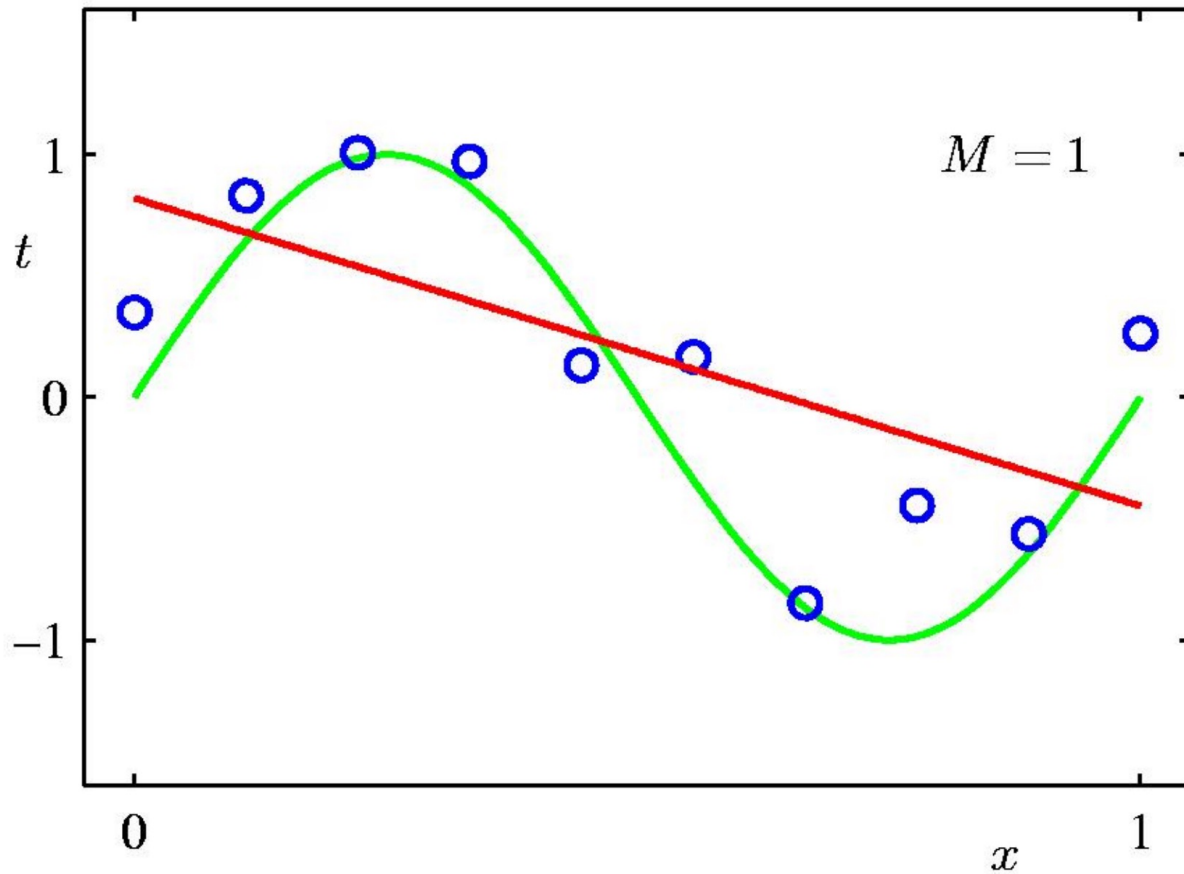
An example: polynomial basis vectors on a small dataset

– From Bishop Ch 1

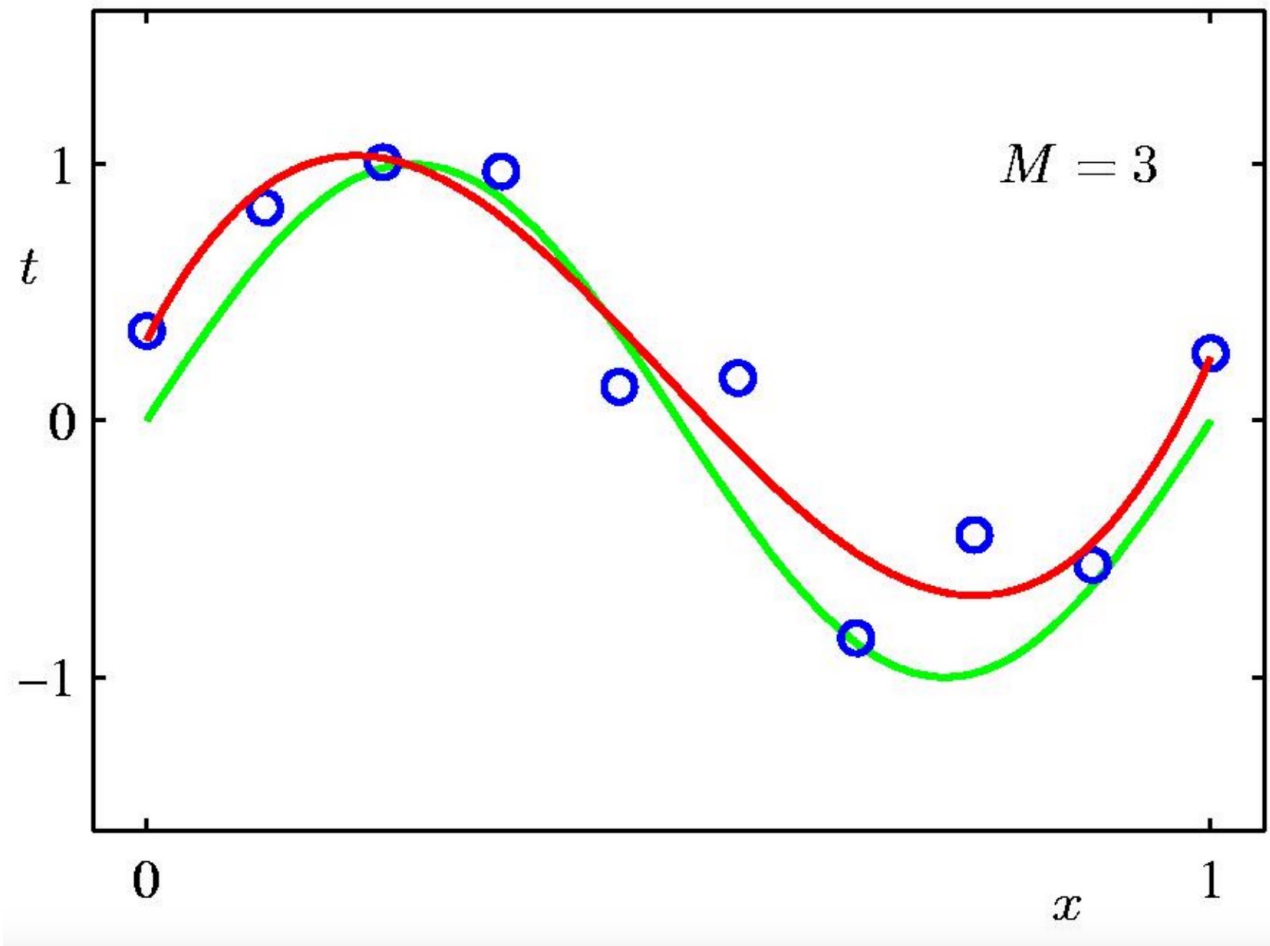
0th Order Polynomial



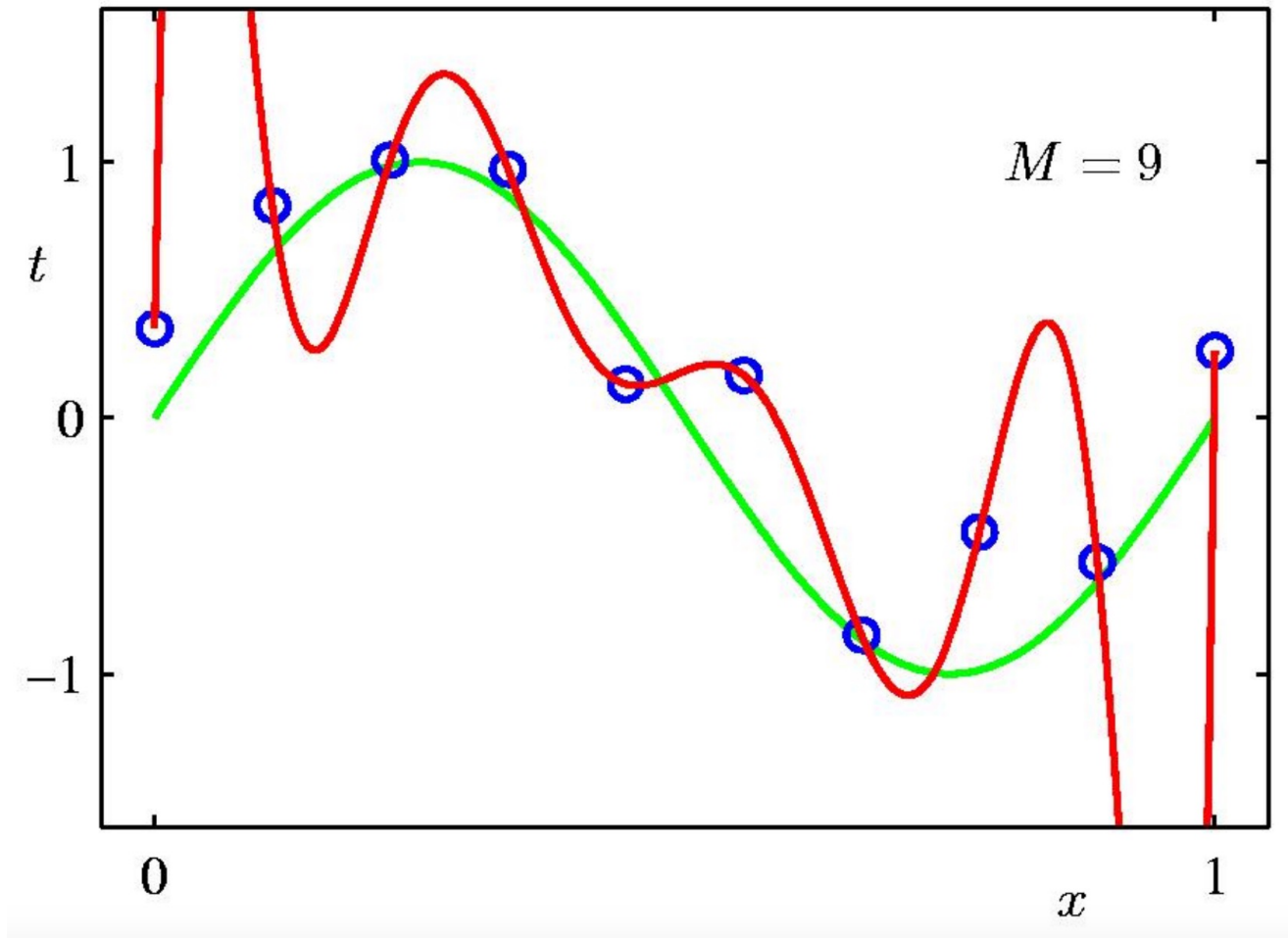
1st Order Polynomial



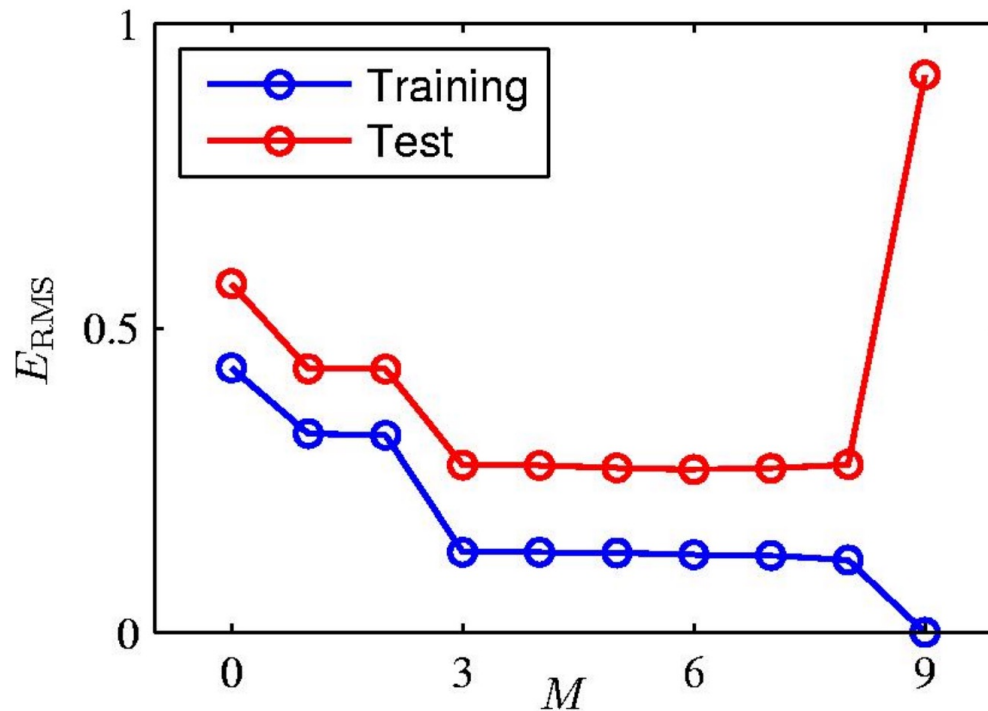
3rd Order Polynomial



9th Order Polynomial



Overfitting



Root-Mean-Square (RMS) Error

Polynomial Coefficients

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
w_0^*	0.19	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

Regularizer: a function that quantifies how much we prefer one hypothesis vs. another

Ridge regression and *Lasso*

- We can fit a model containing all features using a technique that constrains or regularizes the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero.
- It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance.

Ridge Regression


- Recall that the least squares fitting procedure estimates $\beta_0, \beta_1, \dots, \beta_p$ using the values that minimize

$$J(\boldsymbol{\beta}) = \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2$$

- In contrast, the ridge regression coefficient estimates are the values that minimize

$$J_{RR}(\boldsymbol{\beta}) = \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

What's the solution?



$\lambda \|\boldsymbol{\beta}\|_2^2$

where $\lambda \geq 0$ is a *tuning parameter*, to be determined separately.

Ridge Regression

- As with least squares, ridge regression seeks coefficient estimates that fit the data well.
- However, the second term, $\lambda \sum_j \beta_j^2$, called a *shrinkage penalty*, is small when $\beta_0, \beta_1, \dots, \beta_p$ are close to zero, and so it has the effect of *shrinking* the estimates of β_j towards zero.
- The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates.
- Selecting a good value for λ is critical; cross-validation is used for this.

Ridge Regression

Bayesian interpretation: MAP estimation with a **Gaussian prior** on the parameters

$$\boldsymbol{\beta}^{MAP} = \operatorname{argmax}_{\boldsymbol{\beta}} \sum_{i=1}^m \log p_{\boldsymbol{\beta}}(y^{(i)} | \mathbf{x}^{(i)}) + \log p(\boldsymbol{\beta})$$

$$= \operatorname{argmin}_{\boldsymbol{\beta}} J_{RR}(\boldsymbol{\beta})$$

where $p(\boldsymbol{\beta}) \sim \mathcal{N}(0, \frac{1}{\lambda})$

Ridge Regression

- Recall the gradient descent update:

$$\boldsymbol{\beta}^{t+1} \leftarrow \boldsymbol{\beta}^t - \alpha \frac{\partial \mathcal{J}}{\partial \boldsymbol{\beta}}$$

- The gradient descent update of the regularized cost has an interesting interpretation as **weight decay**:


$$\begin{aligned}\boldsymbol{\beta}^{t+1} &\leftarrow \boldsymbol{\beta}^t - \alpha \left(\frac{\partial \mathcal{J}}{\partial \boldsymbol{\beta}} + \lambda \frac{\partial \mathcal{R}}{\partial \boldsymbol{\beta}} \right) \\ &= \boldsymbol{\beta}^t - \alpha \left(\frac{\partial \mathcal{J}}{\partial \boldsymbol{\beta}} + \lambda \boldsymbol{\beta}^t \right) \\ &= (1 - \alpha\lambda) \boldsymbol{\beta}^t - \alpha \frac{\partial \mathcal{J}}{\partial \boldsymbol{\beta}}\end{aligned}$$

The Lasso

- Ridge regression does have one obvious disadvantage: ridge regression will include all features in the final model
- The Lasso is a relatively recent alternative to ridge regression that overcomes this disadvantage. The lasso coefficients minimize the quantity

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

What's the solution?



$\lambda \|\beta\|_1$

- In parlance, the lasso uses an ℓ_1 (pronounced “ell 1”) penalty statistical instead of an ℓ_2 penalty.

Optimization for LASSO

- The L_1 penalty is subdifferentiable (i.e. not differentiable at 0)
- An array of optimization algorithms exist to handle this issue:
 - Coordinate Descent
 - Block coordinate Descent (Tseng & Yun, 2009)
 - Sparse Reconstruction by Separable Approximation (SpaRSA) (Wright et al., 2009)
 - Fast Iterative Shrinkage Thresholding Algorithm (FISTA) (Beck & Teboulle, 2009)

The Lasso

- As with ridge regression, the lasso shrinks the coefficient estimates towards zero.
- However, in the case of the lasso, the ℓ_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large.
- Hence, much like best subset selection, the lasso performs *feature selection*.
- We say that the lasso yields *sparse* models — that is, models that involve only a subset of the variables.
- As in ridge regression, selecting a good value of λ for the lasso is critical; cross-validation is again the method of choice.

The Lasso

- Bayesian interpretation: MAP estimation with a **Laplace prior** on the parameters

$$\begin{aligned}\boldsymbol{\beta}^{MAP} &= \operatorname{argmax}_{\boldsymbol{\beta}} \sum_{i=1}^m \log p_{\boldsymbol{\beta}}(y^{(i)} \mid \mathbf{x}^{(i)}) + \log p(\boldsymbol{\beta}) \\ &= \operatorname{argmin}_{\boldsymbol{\beta}} J_{RR}(\boldsymbol{\beta})\end{aligned}$$

where $p(\boldsymbol{\beta}) \sim \text{Laplace}(0, f(\lambda))$

The Feature Selection Property of the Lasso

Why is it that the lasso, unlike ridge regression, results in coefficient estimates that are exactly equal to zero?

One can show that the lasso and ridge regression coefficient estimates solve the problems

$$\text{minimize}_{\beta} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^p |\beta_j| \leq s$$

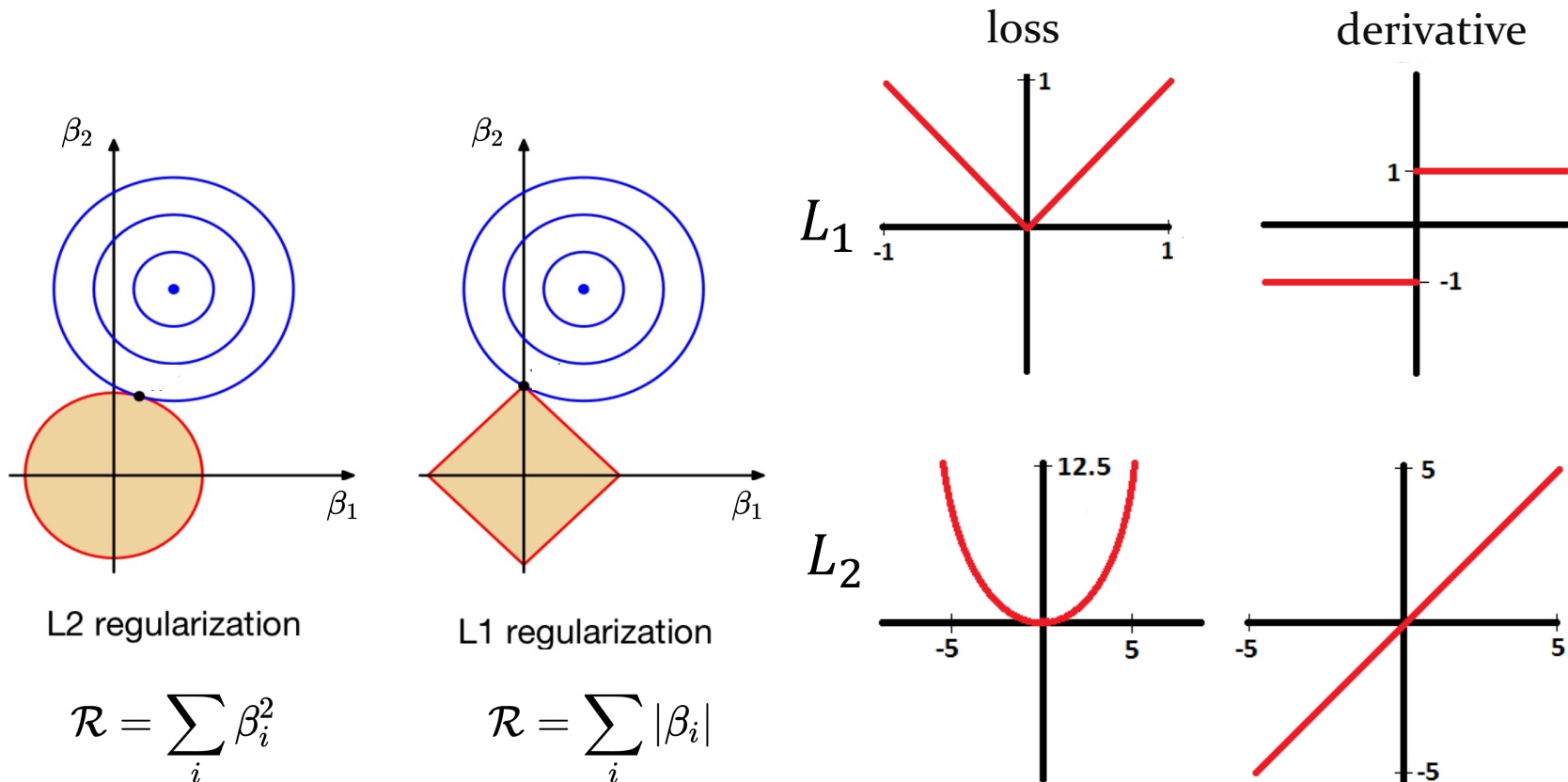
and

$$\text{minimize}_{\beta} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^p \beta_j^2 \leq s,$$

respectively.

Ridge Regression vs. Lasso

Why is it that the lasso, unlike ridge regression, results in coefficient estimates that are exactly equal to zero?



Example: Stock Prices

- Suppose we wish to predict Google's stock price at time $t + 1$
- What features should we use? (putting all computational concerns aside)
 - Stock prices of all other stocks at times $t, t - 1, t - 2, \dots, t - k$
 - Mentions of Google with positive / negative sentiment words in all newspapers and social media outlets
- Do we believe that all of these features are going to be useful?

Advantages of the Linear Model

- Simple form and ease of modeling
- Comprehensibility
- Nonlinear models can be derived from linear models
 - Introducing multi-layer structures or high-dimensional mapping.

■ An example

- Determine the ripeness of a watermelon by considering its **color**, **root** and **sound** information.

$$f_{\text{ripe}}(\mathbf{x}) = 0.2 \cdot x_{\text{color}} + 0.5 \cdot x_{\text{root}} + 0.3 \cdot x_{\text{sound}} + 1.$$

- From the coefficients, we know that root is the most important variable, and sound is more important than color.

Regularization

- The linear model has distinct advantages in terms of its interpretability and often shows good predictive performance.
- ***Model Interpretability:*** By removing irrelevant features —that is, by setting the corresponding coefficient estimates to zero — we can obtain a model that is more easily interpreted. We will present some approaches for automatically performing *feature selection*.

Summary

- Linear regression **predicts** its output as a **linear function** of its inputs
- Learning **optimizes** a function (equivalently **likelihood or mean squared error**) using **standard techniques** (gradient descent, SGD, closed form)
- Regularization enables **shrinkage** and **model selection**