CSCE 633: Machine Learning

Lecture 7: Logistic Regression

Texas A&M University

9-9-18

Before we begin

- HW 1 is up!
- Note when it is due! START EARLY
- Some material still to come - read ahead!
- My Office Hours Canceled Thursday

Last Time

- Ordinary Least Squares Optimization
- Linear Regression
- Convexity and Optimization

Goals of this week

- Logistic Regression
- Gradient Descent
- Measures of performance

What is classification?

- A person arrives at the ER with symptoms that could be 1 of 3 conditions. Which condition is it?

What is classification?

- A person arrives at the ER with symptoms that could be 1 of 3 conditions. Which condition is it?
- A bank must determine if a transaction is fraudulent

What is classification?

- A person arrives at the ER with symptoms that could be 1 of 3 conditions. Which condition is it?
- A bank must determine if a transaction is fraudulent
- Which DNA mutations are deleterious and which are not?

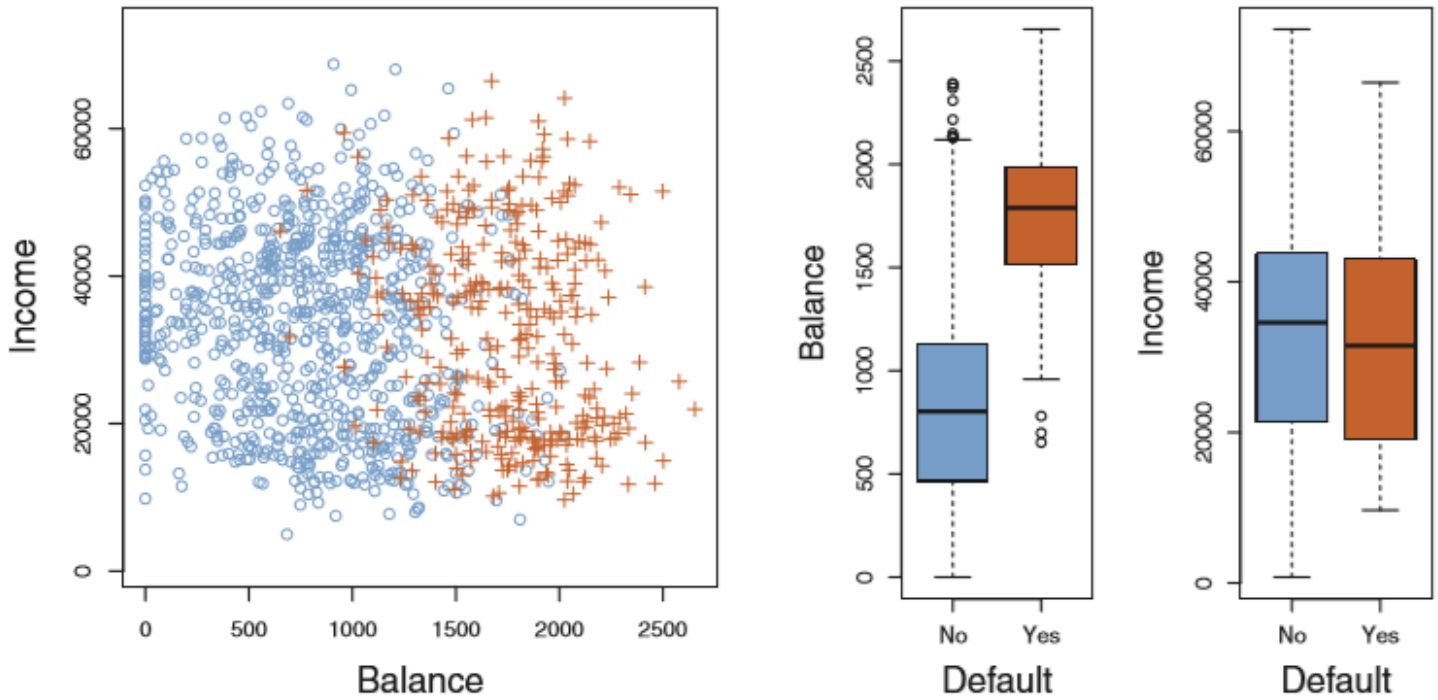
What is classification?

- A person arrives at the ER with symptoms that could be 1 of 3 conditions. Which condition is it?
- A bank must determine if a transaction is fraudulent
- Which DNA mutations are deleterious and which are not?
- Predict likelihood an individual will default on a credit card payment

What is classification?

- A person arrives at the ER with symptoms that could be 1 of 3 conditions. Which condition is it?
- A bank must determine if a transaction is fraudulent
- Which DNA mutations are deleterious and which are not?
- Predict likelihood an individual will default on a credit card payment
- We are still in the data scenario of $D = \{(x_i, y_i)\}_{i=1}^{n}$ but $y_i \in \{0, 1\}$

Credit Default Prediction



- Easily Separable
- Find a function that determines class A vs. class B

Why not linear regression?

- Imagine a response is either stroke, overdose, or epileptic seizure

Why not linear regression?

- Imagine a response is either stroke, overdose, or epileptic seizure
- Encoding instills a natural ordering (e.g., 1, 2, 3)

Why not linear regression?

- Imagine a response is either stroke, overdose, or epileptic seizure
- Encoding instills a natural ordering (e.g., 1, 2, 3)
- But what about 3, 1, 2?

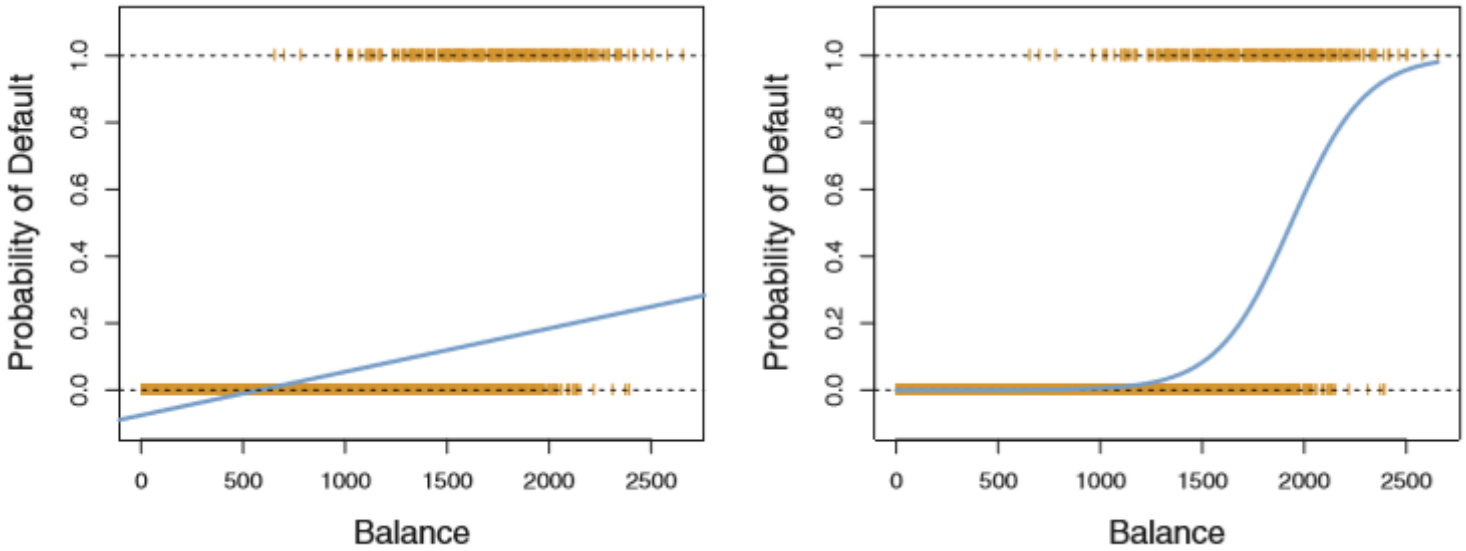
Why not linear regression?

- Imagine a response is either stroke, overdose, or epileptic seizure
- Encoding instills a natural ordering (e.g., 1, 2, 3)
- But what about 3, 1, 2?
- If there is a natural ordering (mild, moderate, severe), linear regression assumes the gap between them is the same.

Why not linear regression?

- Imagine a response is either stroke, overdose, or epileptic seizure
- Encoding instills a natural ordering (e.g., 1, 2, 3)
- But what about 3, 1, 2?
- If there is a natural ordering (mild, moderate, severe), linear regression assumes the gap between them is the same.
- If a single binary decision, maybe linear regression works?

Linear vs. Logistic Regression



- Linear regression states $\hat{y} > 0.5$ as a decision rule - does that work?
- This is hard to interpret as $p(default|x)$
- What if we wanted to model probability of y belonging to a category instead of the actual response?

Bernoulli distribution

The Probability Density Function of a single experiment asking a yes/no question

- $Y \sim \text{Bernoulli}(\theta)$, where $Y \in \{0, 1\}$
- $p(y|\theta) = \theta^{\mathbb{I}(y=1)}(1-\theta)^{\mathbb{I}(y=0)} = \begin{cases} \theta & y = 1 \\ 1-\theta & y = 0 \end{cases}$
- e.g. coin toss experiment

Logistic Regression

Parametric classification method (not regression)

Sometimes referred to as "generalization" of linear regression because

- We still compute a linear combination of feature inputs, i.e. $\boldsymbol{\beta}^T \mathbf{x}$
- Instead of predicting a continuous output variable from $\boldsymbol{\beta}^T \mathbf{x}$
  - We pass $\boldsymbol{\beta}^T \mathbf{x}$ through a function $\mu(\boldsymbol{\beta}^T \mathbf{x}) = \sigma(\boldsymbol{\beta}^T \mathbf{x})$

$$\mu(\eta) = \sigma(\eta) = \frac{1}{1 + e^{-\eta}}, \quad 0 \leq \mu(\eta) \leq 1$$

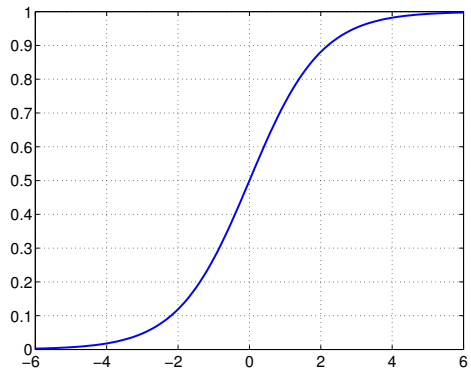
- We replace the Gaussian noise of linear regression with Bernoulli

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \text{Ber}(y|\mu(\boldsymbol{\beta}^T \mathbf{x}))$$

The output belongs to class 1 ($y = 1$) with probability $\mu(\boldsymbol{\beta}^T \mathbf{x})$ and to class 0 ($y = 0$) with probability $1 - \mu(\boldsymbol{\beta}^T \mathbf{x})$.

Why the sigmoid function?

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



Very nice properties

- Bounded between 0 and 1 ← thus interpretable as a probability
- Monotonically increasing ← thus can be used for classification rules
  - $\sigma(\eta) > 0.5$, positive class (y=1)
  - $\sigma(\eta) \leq 0.5$, positive class (y=0)
- Nice computational properties for optimizing criterion function

Logistic Regression: Representation

Setup for two classes

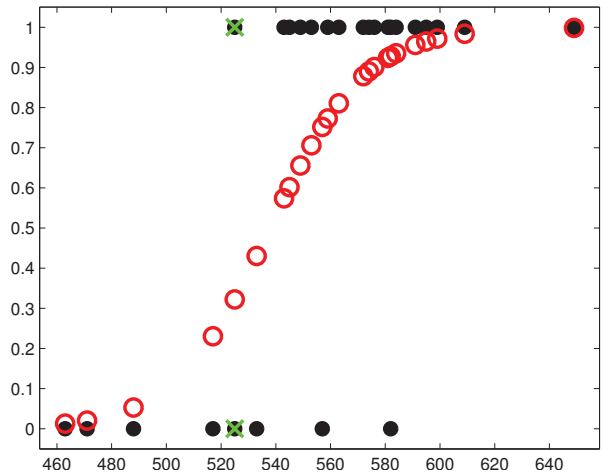
- Input: $\mathbf{x} \in \mathbb{R}^D$
- Output: $y \in \{0, 1\}$
- Training data: $\mathcal{D}^{train} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$
- Model:

$$p(y = 1|\mathbf{x}, \boldsymbol{\beta}) = \sigma(\boldsymbol{\beta}^T \mathbf{x}), \quad \sigma(\eta) = \frac{1}{1 + e^{-\eta}}$$

$$y = \begin{cases} 1, & p(y = 1|\mathbf{x}, \boldsymbol{\beta}) > 0.5 \\ 0, & \text{otherwise} \end{cases}$$

- Model parameters: weights $\boldsymbol{\beta}$

Logistic Regression



Classification task: whether a student passes or not the class

Features: SAT scores

Data: SAT scores v.s. fail/pass (y=0/1) (solid black dots)

Logistic regression:

- Assigns each score to "pass" probability (open red circles)
- If $p(y = 1|x) > 0.5$, then decides $y(x) = 1$. Otherwise, $y(x) = 0$.

Logistic Function

$$p(x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1}}$$

- All values are between 0 and 1
- Fit this model with maximum likelihood
- After some manipulation - we can get $\frac{p(x)}{1-p(x)} = e^{\beta_0+\beta_1}$, called the odds

Logistic Function

$$p(x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1}}$$

- All values are between 0 and 1
- Fit this model with maximum likelihood
- After some manipulation - we can get $\frac{p(x)}{1-p(x)} = e^{\beta_0+\beta_1}$, called the odds
- $log(\frac{p(x)}{1-p(x)}) = \beta_0 + \beta_1 x$ are the log-odds (or logit)

Odds

- Odds can be 0 to inf.

Odds

- Odds can be 0 to inf.
- on average, 1 in 5 people will default with odds 1 out of 4.

Odds

- Odds can be 0 to inf.
- on average, 1 in 5 people will default with odds 1 out of 4.
- $p(x) = 0.2$ and $\frac{0.2}{0.8} = \frac{1}{4}$

Odds

- Odds can be 0 to inf.
- on average, 1 in 5 people will default with odds 1 out of 4.
- $p(x) = 0.2$ and $\frac{0.2}{0.8} = \frac{1}{4}$
- on average, 9 out of 10 will default

Odds

- Odds can be 0 to inf.
- on average, 1 in 5 people will default with odds 1 out of 4.
- $p(x) = 0.2$ and $\frac{0.2}{0.8} = \frac{1}{4}$
- on average, 9 out of 10 will default
- $p(x) = 0.9$ and $\frac{0.9}{0.1} = 9$ - with odds of 9.

Maximum Likelihood Estimation

$$\hat{\theta} = argmax_{\theta} \mathcal{L}(\theta; x)$$

- likelihood is $\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$

Maximum Likelihood Estimation

$$\hat{\theta} = argmax_{\theta} \mathcal{L}(\theta; x)$$

- likelihood is $\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$
- θ are β_0 and β_1

Maximum Likelihood Estimation

$$\hat{\theta} = argmax_{\theta} \mathcal{L}(\theta; x)$$

- likelihood is $\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$
- θ are β_0 and β_1
- Maximizing this means finding β_0 and β_1 that maximizes this product

Multiple Logistic Regression

$$log(\frac{p(x)}{1-p(x)}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

- $p(y|x, \beta) = Ber(y|sigm(\beta^T x))$, with a linear decision boundary of $p(x) > 0.5$

Multiple Logistic Regression

$$log(\frac{p(x)}{1-p(x)}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

- $p(y|x, \beta) = Ber(y|sigm(\beta^T x))$, with a linear decision boundary of $p(x) > 0.5$
- $NLL(\beta) = -\sum_{i=1}^{n}(y_i log \mu_i + (1 - y_i) log(1 - \mu_i))$ where$\mu_i = sigm(\beta^T x)$

Multiple Logistic Regression

$$log(\frac{p(x)}{1-p(x)}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

- $p(y|x, \beta) = Ber(y|sigm(\beta^T x))$, with a linear decision boundary of $p(x) > 0.5$
- $NLL(\beta) = -\sum_{i=1}^{n}(y_i log \mu_i + (1 - y_i) log(1 - \mu_i))$ where$\mu_i = sigm(\beta^T x)$
- Suppose $\tilde{y}_i \in \{-1, 1\}$

Multiple Logistic Regression

$$log(\frac{p(x)}{1-p(x)}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

- $p(y|x, \beta) = Ber(y|sigm(\beta^T x))$, with a linear decision boundary of $p(x) > 0.5$
- $NLL(\beta) = -\sum_{i=1}^{n}(y_i log \mu_i + (1 - y_i) log(1 - \mu_i))$ where$\mu_i = sigm(\beta^T x)$
- Suppose $\tilde{y}_i \in \{-1, 1\}$
- $p(\tilde{y} = 1) = \frac{1}{1+e^{-\beta^T x}}$

Multiple Logistic Regression

$$log(\frac{p(x)}{1-p(x)}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

- $p(y|x, \beta) = Ber(y|sigm(\beta^T x))$, with a linear decision boundary of $p(x) > 0.5$
- $NLL(\beta) = -\sum_{i=1}^{n}(y_i log \mu_i + (1 - y_i) log(1 - \mu_i))$ where$\mu_i = sigm(\beta^T x)$
- Suppose $\tilde{y}_i \in \{-1, 1\}$
- $p(\tilde{y} = 1) = \frac{1}{1+e^{-\beta^T x}}$
- $NLL(\beta) = \sum_{i=1}^{n} log(1 + e^{-\tilde{y}_i \beta^T x_i})$ which has no closed form solution.

Logistic Regression: Evaluation

Data likelihood for 1 training sample

$$p(y_n|\mathbf{x_n}, \boldsymbol{\beta}) = \left\{ \begin{array}{ll} \sigma(\boldsymbol{\beta}^T\mathbf{x_n}), & y_n = 1 \\ 1 - \sigma(\boldsymbol{\beta}^T\mathbf{x_n}), & y_n = 0 \end{array} \right\} = \left[\sigma(\boldsymbol{\beta}^T\mathbf{x_n})\right]^{y_n} \left[1 - \sigma(\boldsymbol{\beta}^T\mathbf{x_n})\right]^{1-y_n}$$

Data likelihood for all training data

$$L(\mathcal{D}|\boldsymbol{\beta}) = \prod_{n=1}^{N} p(y_n|\mathbf{x_n}, \boldsymbol{\beta}) = \prod_{n=1}^{N} \left[\sigma(\boldsymbol{\beta}^T\mathbf{x_n})\right]^{y_n} \left[1 - \sigma(\boldsymbol{\beta}^T\mathbf{x_n})\right]^{1-y_n}$$

Log-likelihood for all training data

$$l(\mathcal{D}|\boldsymbol{\beta}) = \sum_{n=1}^{N} \left\{ y_n \log\left[\sigma(\boldsymbol{\beta}^T\mathbf{x_n})\right] + (1 - y_n) \log\left[1 - \sigma(\boldsymbol{\beta}^T\mathbf{x_n})\right] \right\}$$

Cross-entropy error (negative log-likelihood)

$$\mathcal{E}(\boldsymbol{\beta}) = -\sum_{n=1}^{N} \left\{ y_n \log\left[\sigma(\boldsymbol{\beta}^T\mathbf{x_n})\right] + (1 - y_n) \log\left[1 - \sigma(\boldsymbol{\beta}^T\mathbf{x_n})\right] \right\}$$

Logistic Regression: Optimization

Cross-entropy error (negative log-likelihood)

$$\mathcal{E}(\boldsymbol{\beta}) = -\sum_{n=1}^{N} \left\{ y_n \log \left[\sigma(\boldsymbol{\beta}^T \mathbf{x_n}) \right] + (1 - y_n) \log \left[1 - \sigma(\boldsymbol{\beta}^T \mathbf{x_n}) \right] \right\}$$

How to find the weights $\boldsymbol{\beta}$ of the logistic regression?

We can maximize data likelihood or minimize cross-entropy error

$$\boldsymbol{\beta}^* = \min_{\boldsymbol{\beta}} \mathcal{E}(\boldsymbol{\beta})$$

Unlike linear regression, we cannot find a closed-form solution.

So we use approximate methods, e.g. Gradient Descent.

To do that, we need to compute the gradient $\nabla \mathcal{E}(\boldsymbol{\beta})$

$$\nabla \mathcal{E} = \sum_{n=1}^{N} \underbrace{\left(\sigma(\boldsymbol{\beta}^T \mathbf{x_n}) - y_n \right)}_{\text{error}} \mathbf{x_n}$$

Linear Regression: Computational Complexity

- Bottleneck for computing the solution $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ is to invert the matrix $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{D\times D}$
- Computational complexity is $O((D+1)^3)$ using Gauss-Jordan elimination
  - Impractical for large D
- Alternative
  - Find approximate solution through an iterative optimization algorithm
  - e.g. Gradient Descent

Logistic Regression: Optimization

Is the cross-entropy error a convex function?

Yes, it is convex.

$$\nabla \mathcal{E} = \frac{\vartheta \mathcal{E}(\boldsymbol{\beta})}{\vartheta \boldsymbol{\beta}} = \sum_{n=1}^{N} \underbrace{\Big(\sigma(\boldsymbol{\beta}^T \mathbf{x_n}) - y_n\Big)}_{\text{error}} \mathbf{x_n}$$

$$\mathbf{H} = \frac{\vartheta^2 \mathcal{E}(\boldsymbol{\beta})}{\vartheta \boldsymbol{\beta} \boldsymbol{\beta}^T} = \sum_{n=1}^{N} \underbrace{\sigma(\boldsymbol{\beta}^T \mathbf{x_n})}_{\in [0,1]} \cdot \underbrace{\Big(1 - \sigma(\boldsymbol{\beta}^T \mathbf{x_n})\Big)}_{\in [0,1]} \cdot \underbrace{\big(\mathbf{x_n} \cdot \mathbf{x_n}^T\big)}_{\in \mathcal{R}^{D \times D}}$$

For all $\mathbf{v}$, substituting $\mu = \sigma(\boldsymbol{\beta}^T \mathbf{x_n}) \Big(1 - \sigma(\boldsymbol{\beta}^T \mathbf{x_n})\Big) \geq 0$, we have:

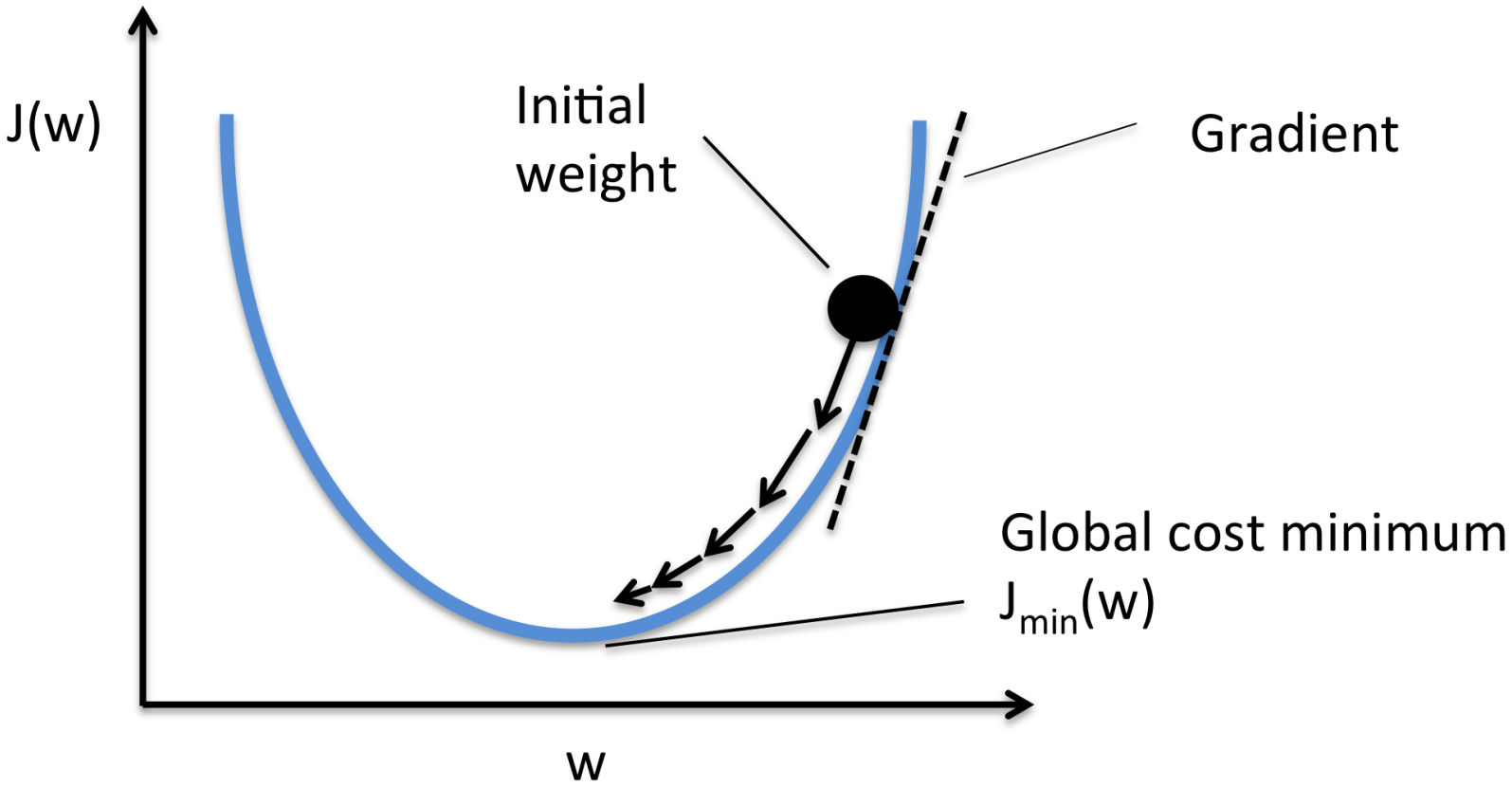
$$\mathbf{v}^T \mathbf{H} \mathbf{v} = \mu \cdot \mathbf{v}^T \left(\sum_{n=1}^{N} \mathbf{x_n} \mathbf{x_n}^T \right) \mathbf{v} = \mu \sum_{n=1}^{N} (\mathbf{x_n}^T \mathbf{v})^T (\mathbf{x_n}^T \mathbf{v}) = \mu \sum_{n=1}^{N} \|\mathbf{x_n}^T \mathbf{v}\|_2^2 \geq 0$$

Gradient Descent

- Iterative algorithm finding a function's minimum via local search
- Standard optimization algorithm in many ML applications
  - e.g. linear regression, logistic regression, neural networks
  - scales well to large datasets (e.g. no matrix multiplication)
  - proof that it solves many convex problems
  - good heuristic to non-convex problems as well
- **Input:** Function $J(\boldsymbol{\beta}) \in \mathbb{R}$
- **Output:** Local minimum $\boldsymbol{\beta}^*$
- **Goal:** Minimize $J(\boldsymbol{\beta})$ via greedy local search
- Remember, $\mathbf{w} = \boldsymbol{\beta}$

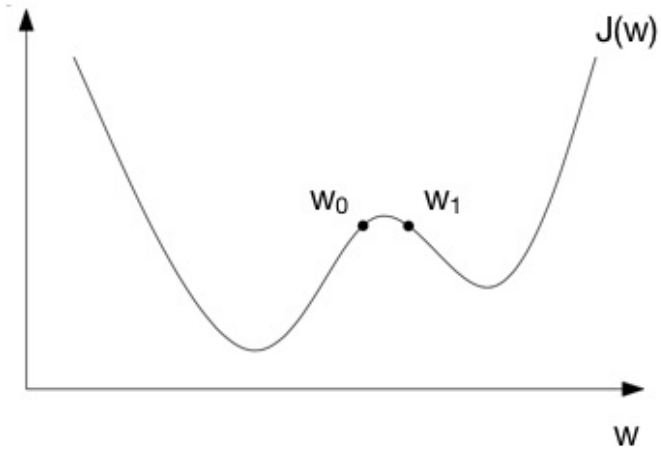
Gradient Descent

1-dimensional example: $J(\beta) = \beta^2$



Gradient Descent: 1-dimensional case

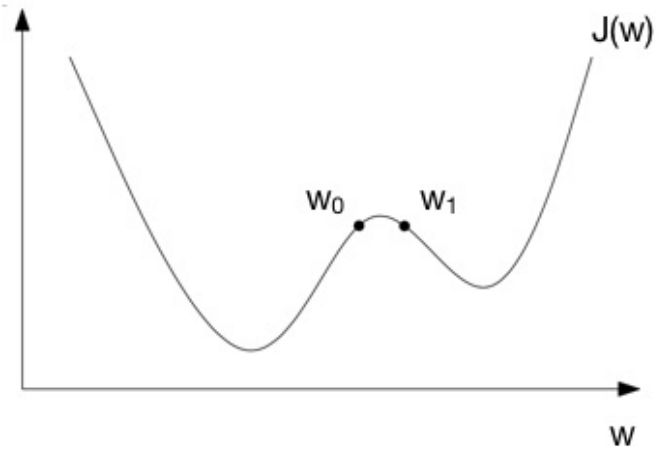
What will happen if we try to minimize $J(\beta)$ via a local search?



- Starting from w_0
  - We look to the right ($J(\beta) \uparrow$) and to the left ($J(\beta) \downarrow$)
  - We take a small step to the left
  - We repeat the above until we reach the left basin
- Starting from w_1
  - We similarly reach the right basin
- It is clear that the outcome depends on the starting point

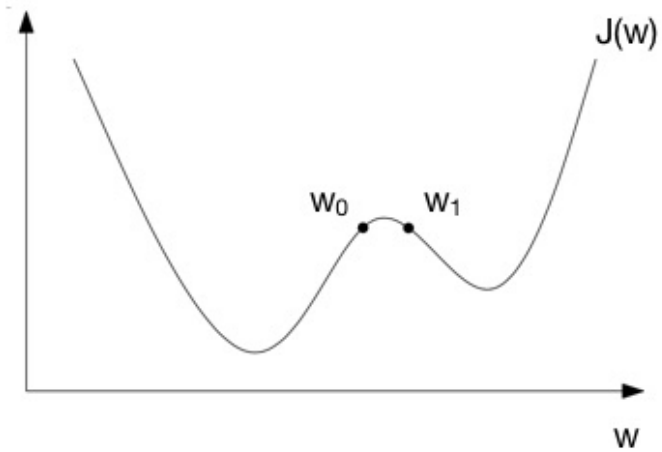
Gradient Descent: 1-dimensional case

More formally, we do the following

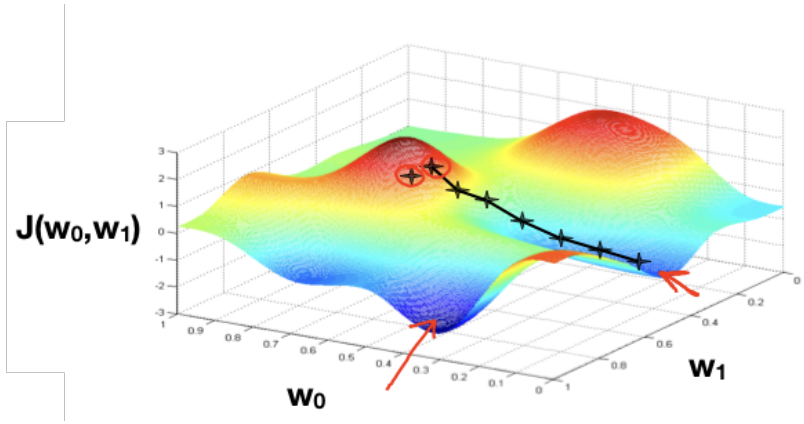


- While $J'(\beta) \neq 0$
  - If $J'(\beta) > 0$ (i.e. $J(\beta) \uparrow$), move β a little bit to the left
  - If $J'(\beta) < 0$ (i.e. $J(\beta) \downarrow$), move β a little bit to the right
- The derivative $J'(w)$ is used to decide which direction to move
- In other words, move β towards the direction of $-J'(\beta)$

Gradient Descent: Algorithm Outline



1-dimensional



[Source: Machine Learning, Coursera, Andrew Ng]

d-dimensional

1 Initialize β, ϵ, $\alpha(\cdot)$, $k := 0$

2 While $|\frac{dJ(\beta)}{d\beta}| > \epsilon$

2a $k := k + 1$

2b $\beta := \beta - \alpha(k) \cdot \frac{dJ(\beta)}{d\beta}$

1 Initialize $\boldsymbol{\beta}$, ϵ, $\alpha(\cdot)$, $k := 0$

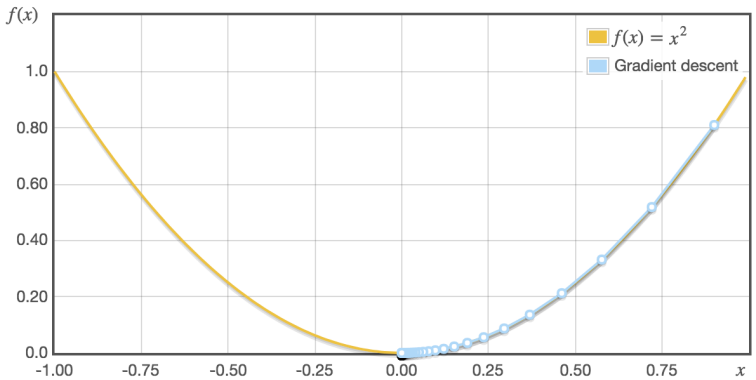
2 While $\|\nabla J(\boldsymbol{\beta})\|_2 > \epsilon$

2a $k := k + 1$

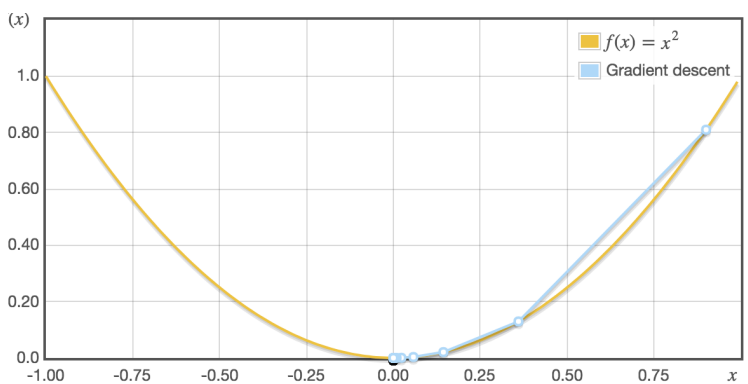
2b $\boldsymbol{\beta} := \boldsymbol{\beta} - \alpha(k) \cdot \nabla J(\boldsymbol{\beta})$

Gradient Descent: Step size (or learning rate) α

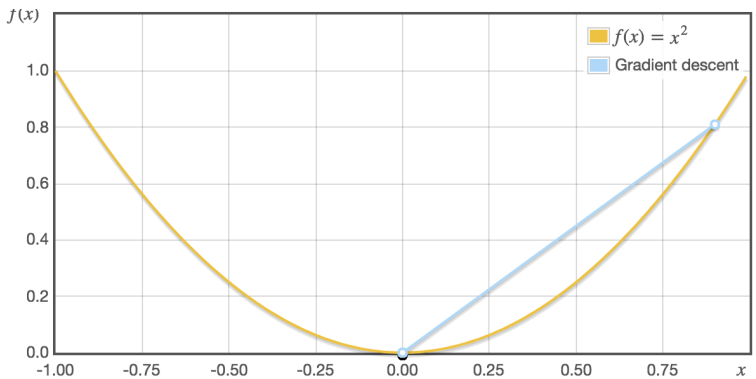
$\alpha = 0.1$



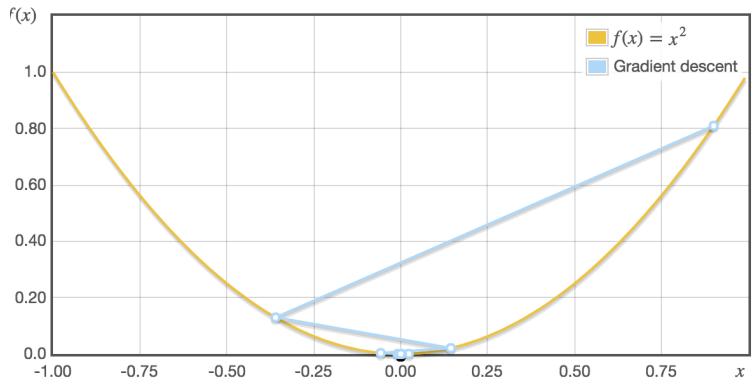
$\alpha = 0.3$



$\alpha = 0.5$



$\alpha = 0.7$



- If $\alpha(k)$ too small, convergence is unnecessarily slow
- If $\alpha(k)$ too large, correction process will overshoot and can diverge

Source: `http://www.onmyphd.com/?p=gradient.descent`

Gradient Descent: Step size (or learning rate) α

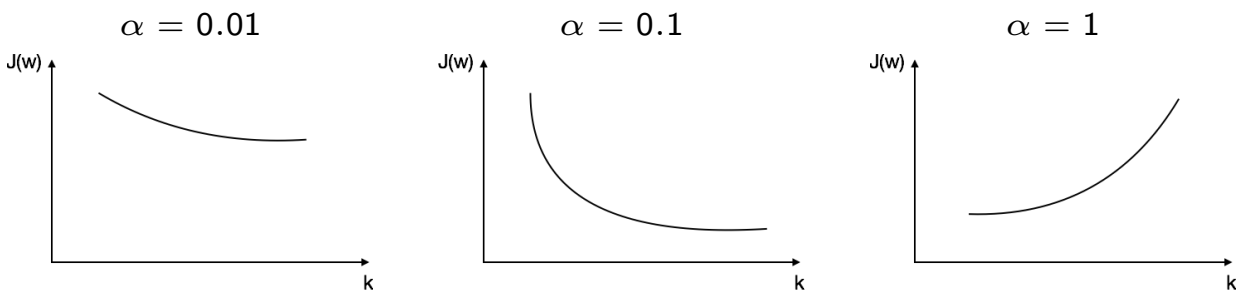
How to chose α?

- There is a principled method for determining α
  - too expensive
- In practice, through experimentation
  - Check how $J(\beta)$ behaves over iterations for multiple α
  - Always use some type of cross-validation framework

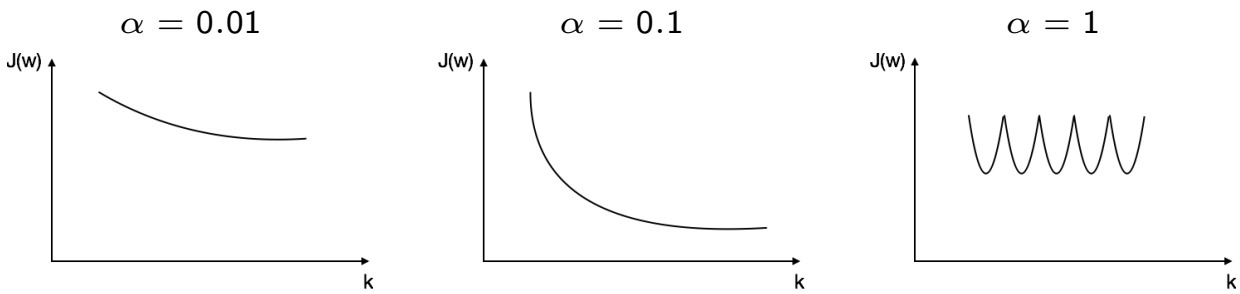
Gradient Descent: Step size (or learning rate) α

Question: A cost function $J(\beta)$ is optimized with Gradient Descent (GD) using different step size values α. We plot $J(\beta)$ w.r.t. the number of GD iterations k. Which of the following graph triplets can hold?

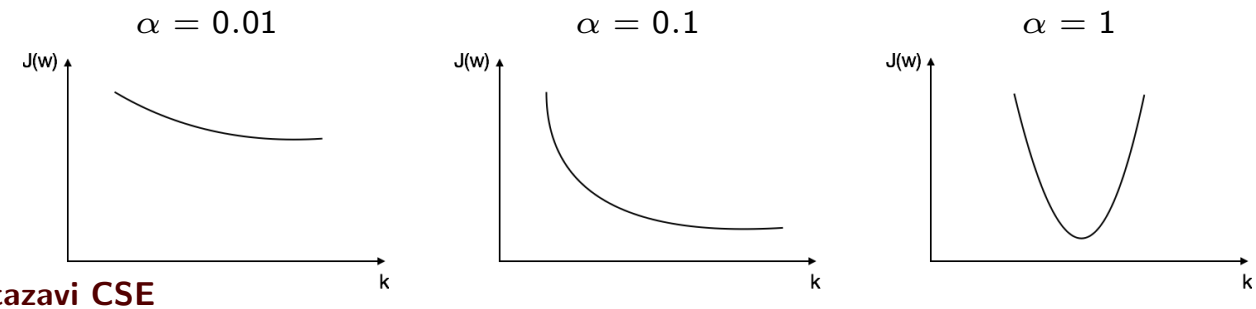
A



B



C



Gradient Descent: Stopping rule

- The parameter ϵ (i.e. $\|\nabla J(\boldsymbol{\beta})\|_2 > \epsilon$) determines when to stop
- Small ϵ: many iterations but higher quality solution
- Large ϵ: less iterations with the cost of more approximate solution
- How to chose ϵ in practice?
  - Try various values to achieve balance between cost and precision
  - Again use some type of cross-validation framework
- Hyperparameters: Parameters set before the beginning of the learning process (e.g. α, ϵ in gradient descent)
- Hyperparameter tuning: The process of choosing a set of optimal hyperparameters for the learning process
- Model parameters: The parameters learned during the learning process (e.g. weights $\boldsymbol{\beta}$ in linear regression)

Gradient Descent in Linear Regression

We can now derive the algorithm outline for minimizing the residual square sum (RSS) error of linear regression with gradient descent and weights β

- The residual sum of squares is the cost function:
$$J(\beta) = RSS(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$
$$= \mathbf{y}^T\mathbf{y} - 2(\mathbf{X}\beta)^T\mathbf{y} + (\mathbf{X}\beta)^T(\mathbf{X}\beta)$$
$$= \mathbf{y}^T\mathbf{y} - 2\beta^T(\mathbf{X}^T\mathbf{y}) + \beta^T(\mathbf{X}^T\mathbf{X})\beta$$
- Gradient Descent optimization expression:
$$\beta := \beta - \alpha(k) \cdot \nabla J(\beta)$$
$$\nabla J(\beta) = \frac{\vartheta RSS(\beta)}{\vartheta \beta} = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\beta = 0$$

Gradient Descent in Linear Regression

Question: Derive the algorithm outline for minimizing the residual square sum (RSS) error of linear regression with gradient descent

(Batch) Gradient Descent for Linear Regression

1 Initialize $\boldsymbol{\beta}$, ϵ, $\alpha(\cdot)$, $k := 0$

2 While $\|\nabla RSS(\boldsymbol{\beta})\|_2 > \epsilon$

2a $k := k + 1$

2b $\boldsymbol{\beta} := \boldsymbol{\beta} - \alpha(k) \cdot \left(\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - \mathbf{X}^T\mathbf{y}\right)$

Gradient Descent in Linear Regression

Stochastic Gradient Descent for Linear Regression
Update weights using one sample at a time

1 Initialize $\boldsymbol{\beta}$, ϵ, $\alpha(\cdot)$, $k := 0$

2 Loop until convergence

- 2a $k := k + 1$
- 2b Randomly choose a sample $(\mathbf{x_i}, y_i)$
- 2c Compute its contribution to the gradient $\mathbf{g_i} = (\mathbf{x_i}^T \boldsymbol{\beta} - y_i) \cdot \mathbf{x_i}$
- 2d Update the weights $\boldsymbol{\beta} := \boldsymbol{\beta} - \alpha(k) \cdot \mathbf{g_i}$

Gradient Descent in Linear Regression

Mini-Batch Gradient Descent for Linear Regression
Update weights using subset of samples at a time

1 Initialize $\boldsymbol{\beta}$, ϵ, $\alpha(\cdot)$, $k := 0$

2 Loop until convergence

2a $k := k + 1$

2b Randomly choose a subset of samples
$S = \{(\mathbf{x_i}, y_i), \ldots, (\mathbf{x_{i+M}}, y_{i+M})\}$

2c Form the mini-batch data matrix $\mathbf{X_S} = \begin{bmatrix} \mathbf{x_i}^T \\ \vdots \\ \mathbf{x}_{\mathbf{i+M}}^T \end{bmatrix}$

2d Update the weights $\boldsymbol{\beta} := \boldsymbol{\beta} - \alpha(k) \cdot \left(\mathbf{X_S}^T \mathbf{X_S} \boldsymbol{\beta} - \mathbf{X_S}^T \mathbf{y} \right)$

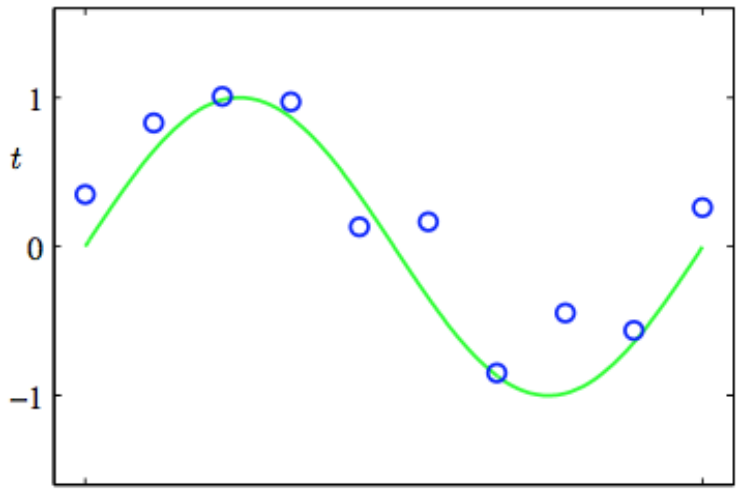
- Good compromise between batch and stochastic gradient descent
- Common mini-batch sizes range between M=50-250 samples
- Typical algorithm for training a neural network with gradient descent

Gradient Descent in Linear Regression

- Batch gradient descent computes exact gradient
- Stochastic gradient descent
  - Computes approximate gradient using one sample per iteration
  - Its expectation equals the true gradient
- Mini-batch gradient descent
  - Computes gradient based on subset of samples
- For large-scale problems stochastic or mini-batch descent often work well

What if data does not fit a line?

Example: Samples from a sine function



We can use a non-linear basis function

$\phi(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^D \rightarrow \mathbf{z} \in \mathbb{R}^M$

We can apply our linear regression model to the new features

$y_i = \beta^T \mathbf{z_i} = \beta^T \phi(\mathbf{x_i})$

$RSS(\beta) = \sum_{n=1}^{N} \left(y_i - \beta^T \phi(\mathbf{x_i})\right)^2, \ \beta \in \mathbb{R}^M$

Non-Linear Basis Function

Residual sum of squares

$$RSS(\beta) = \sum_{n=1}^{N} \left(y_i - \beta^T \phi(\mathbf{x_i}) \right)^2 = (\mathbf{y} - \boldsymbol{\Phi}\beta)^T (\mathbf{y} - \boldsymbol{\Phi}\beta)$$

Non-linear design matrix

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi(\mathbf{x_1})^T \\ \phi(\mathbf{x_2})^T \\ \vdots \\ \phi(\mathbf{x_N})^T \end{bmatrix} \in \mathbb{R}^{N \times M}$$

LMS solution with the non-linear design matrix

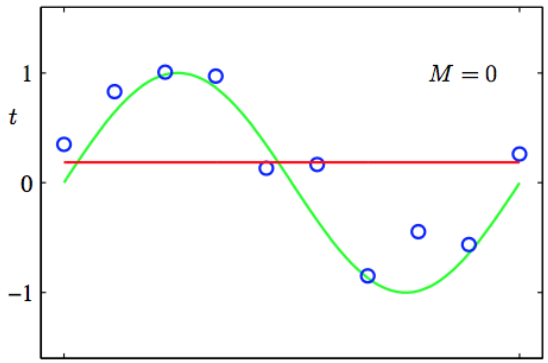
$$\beta^{LMS} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \mathbf{y}$$

Non-Linear Basis Function

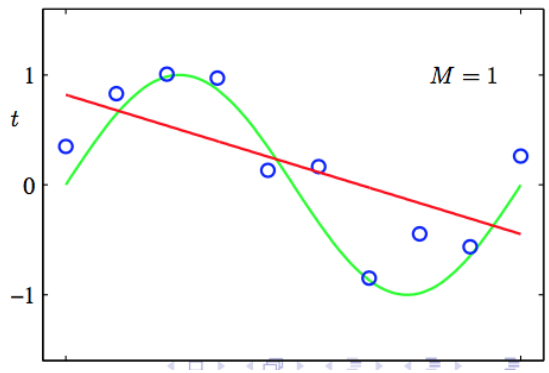
Example: Samples from a sine function

Polynomial basis function $\phi(\mathbf{x}) = [1\ x\ \ldots\ x^M]^T$

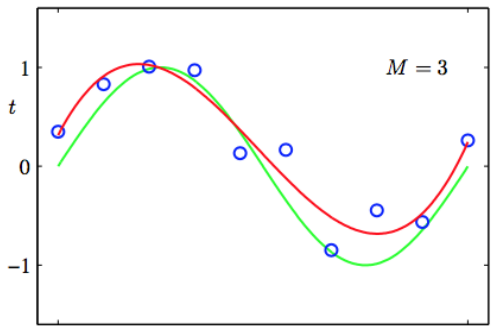
$M = 0$ underfitting



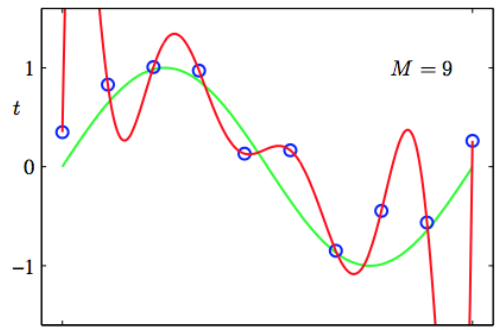
$M = 1$ underfitting



$M = 3$



$M = 9$ overfitting



Overfitting

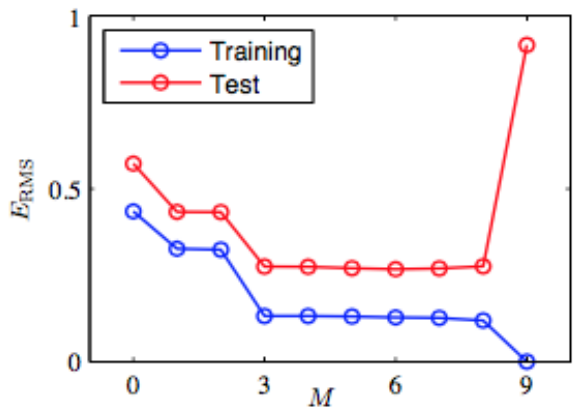
Weights of high order polynomials are very large

$$y_i = \boldsymbol{\beta}^T \mathbf{z_i} = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x_i}), \quad \mathbf{z_i} = \boldsymbol{\phi}(\mathbf{x_i}) \in \mathbb{R}^M$$

	$M=0$	$M=1$	$M=3$	$M=9$
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

Overfitting

- The risk of using highly flexible (complicated) models without enough data
- Leads to poor generalization
- How to detect overfitting?
  - Plot model complexity (e.g. polynomial order) versus objective function
  - As complexity increases, performance on training improves, while on testing first improves and then deteriorates
- How to avoid overfitting?
  - More data or regularization



Linear Regression: To summarize

- **Representation:** linear and non-linear basis

  $f : \mathbf{x} \rightarrow y, \;\; f(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{x}$

  $f : \mathbf{z} \rightarrow y, \;\; f(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{z} = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}), \;\; \boldsymbol{\phi} : \mathbf{x} \in \mathbb{R}^D \rightarrow \mathbf{z} \in \mathbb{R}^M$

- **Evaluation**: Minimizing residual sum of squares

  $\min_{\boldsymbol{\beta}} RSS(\boldsymbol{\beta}), \;\; RSS(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$

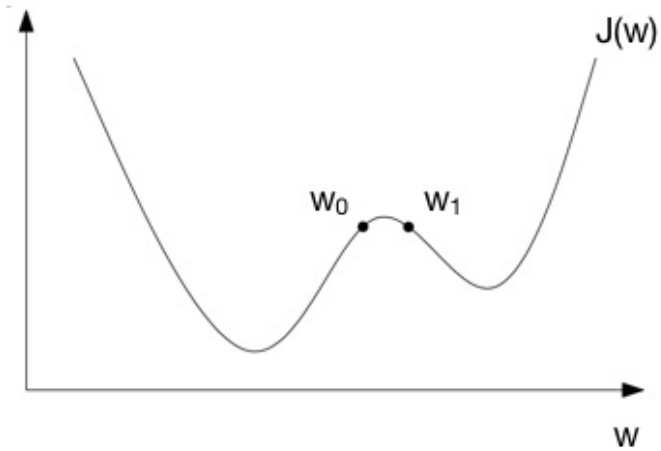
  $\min_{\boldsymbol{\beta}} RSS(\boldsymbol{\beta}), \;\; RSS(\boldsymbol{\beta}) = (\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta})^T (\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta})$

- **Analytic Optimization**: Ordinary least squares (OLS) solution

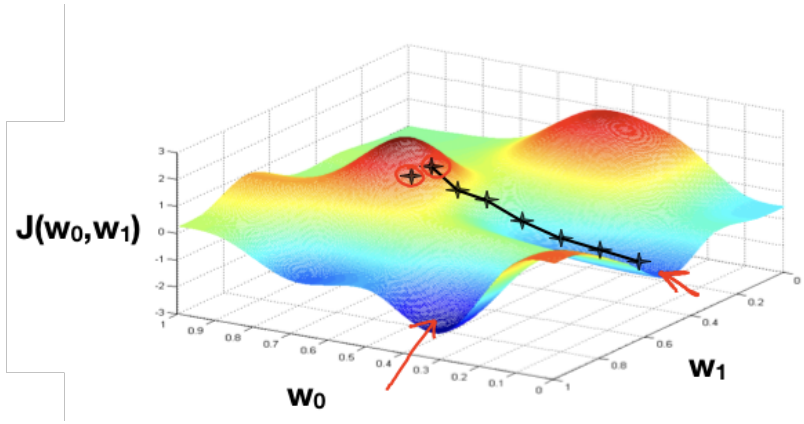
  $\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}, \;\; \boldsymbol{\beta}^* = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \mathbf{y}$

- **Approximate Optimization**: Gradient descent
- Linear regression with **Gaussian** noise
  - MLE is equivalent to OLS solution

Gradient Descent: Algorithm Outline



1-dimensional



[Source: Machine Learning, Coursera, Andrew Ng]

d-dimensional

1 Initialize β, ϵ, $\alpha(\cdot)$, $k := 0$

2 While $|\frac{dJ(\beta)}{d\beta}| > \epsilon$

2a $k := k + 1$

2b $\beta := \beta - \alpha(k) \cdot \frac{dJ(w)}{dw}$

1 Initialize $\boldsymbol{\beta}$, ϵ, $\alpha(\cdot)$, $k := 0$

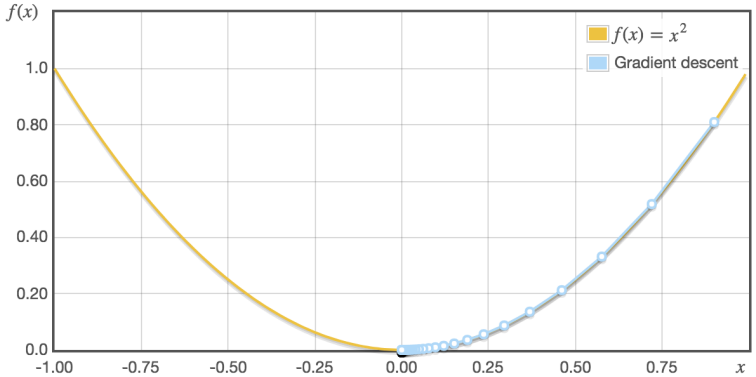
2 While $\|\nabla J(\boldsymbol{\beta})\|_2 > \epsilon$

2a $k := k + 1$

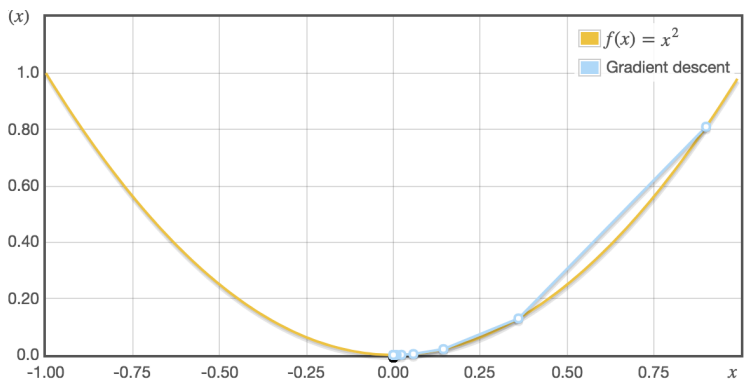
2b $\boldsymbol{\beta} := \boldsymbol{\beta} - \alpha(k) \cdot \nabla J(\boldsymbol{\beta})$

Gradient Descent: Step size (or learning rate) α

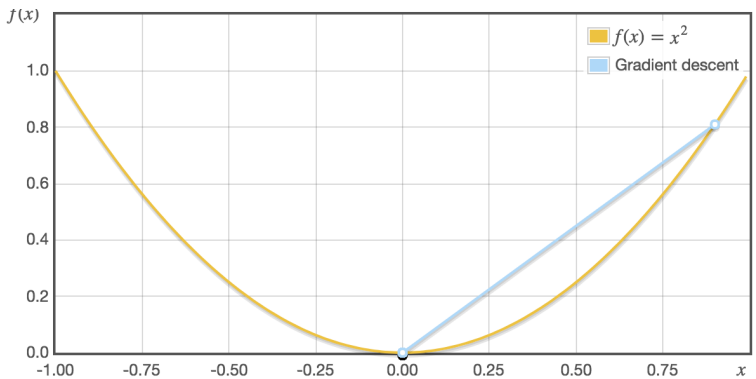
$\alpha = 0.1$



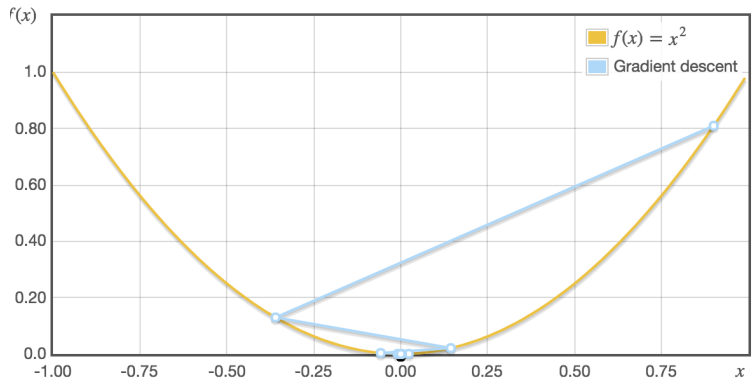
$\alpha = 0.3$



$\alpha = 0.5$



$\alpha = 0.7$



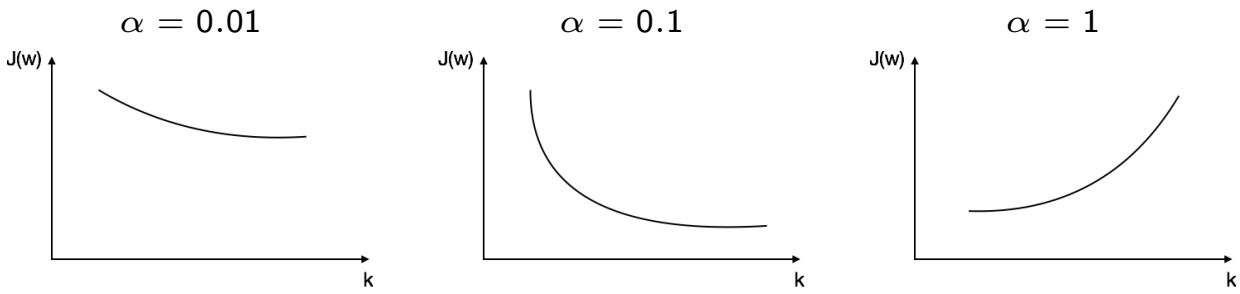
- If $\alpha(k)$ too small, convergence is unnecessarily slow
- If $\alpha(k)$ too large, correction process will overshoot and can diverge

Source: `http://www.onmyphd.com/?p=gradient.descent`

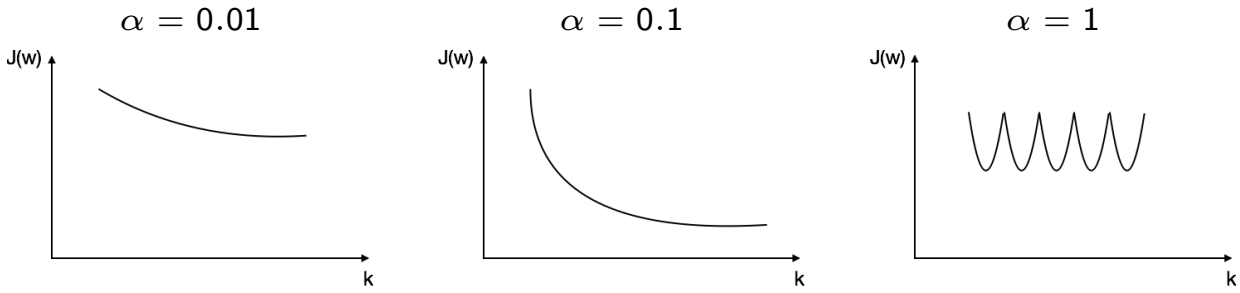
Gradient Descent: Step size (or learning rate) α

Question: A cost function $J(\beta)$ is optimized with Gradient Descent (GD) using different step size values α. We plot $J(\beta)$ w.r.t. the number of GD iterations k. Which of the following graph triplets can hold?

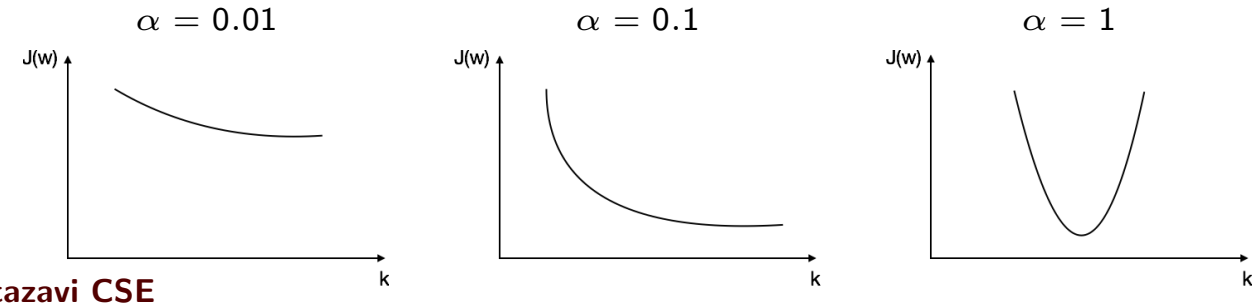
A



B



C



All answers can occur

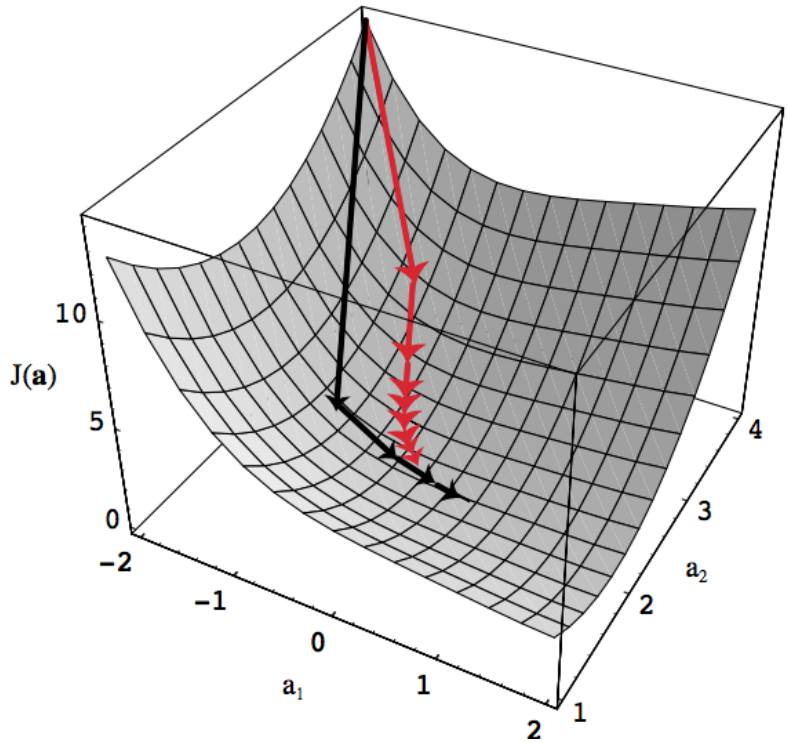
Newton Descent

- Alternative approach for minimizing $J(\boldsymbol{\beta})$
- Aims to minimize the second order expansion of $J(\boldsymbol{\beta})$

Algorithm Outline

1. Initialize $\boldsymbol{\beta}$, θ
2. While $|\mathbf{H}^{-1} \cdot \nabla J(\boldsymbol{\beta})| > \theta$
   - 2a $\boldsymbol{\beta} := \boldsymbol{\beta} - \mathbf{H}^{-1} \cdot \nabla J(\boldsymbol{\beta})$

Newton Descent and Gradient Descent



Sequence of parameter values $\boldsymbol{\beta}$ when minimizing $J(\boldsymbol{\beta})$ with Gradient (red) and Newton Descent (black)

[Source: Duda, Hart & Stork]

Newton Descent and Gradient Descent

Gradient Descent $\boldsymbol{\beta} := \boldsymbol{\beta} - \alpha(k) \cdot \nabla J(\boldsymbol{\beta})$	**Newton Descent** $\boldsymbol{\beta} := \boldsymbol{\beta} - \mathbf{H}^{-1} \cdot \nabla J(\boldsymbol{\beta})$
Cheaper implementation $O(d)$	More expensive $O(d^3)$ (because of $\mathbf{H}^{-1}$)
Need to chose learning rate α	No parameters need to set
Needs more iterations (100-1000)	Needs fewer iterations (5-15)
Use when #samples is large	Use when #samples is small (< 1000)

[Source: Andew Ng]

Benefits of logistic regression

Commonly used classification method

- Computationally easy and fast optimization algorithms
- Interpretable model
- Easy to extend for multi-class classification
- Easy to extend to non-linear decision boundaries using kernel functions

Logistic Regression

- Generalization of linear regression
  - Linear combination of input features $\beta^T \mathbf{x}$
  - Transform through sigmoid function $\sigma(\beta^T \mathbf{x}) \rightarrow$ interpretable as probability
  - Decision rule based on whether $\sigma(\beta^T \mathbf{x}) \lessgtr 0.5$
- Evaluation through data likelihood, or cross-entropy error

$$\mathcal{E}(\beta) = -\sum_{n=1}^{N} \left\{ y_n \log \left[\sigma(\beta^T \mathbf{x_n}) \right] + (1 - y_n) \log \left[1 - \sigma(\beta^T \mathbf{x_n}) \right] \right\}$$

Takeaways and Next Time

- Logistic Regression
- Gradient Descent
- Next Time: Measures of Performance