Advanced Machine Learning & Deep Learning

Dr. Hazem Shatila

Markov

Key topics to be covered:

- Linear & logistic regression.
- Neural Networks
- Deep Learning, forward & backward Prop.
- Neural network Vectorization
- Neural Networks Hyper-parameters
- Convolutional Neural Network (CNN)
- Classic CNN's
- CNN applications
- Text Mining, TF-IDF
- Recommender Systems
- Sequence Modeling
- Recurrent Neural Network (RNN)
- GRU & LSTM
- Word embedding (NLP)
- Ensemble learners: Bagging & Boosting

Tutorial

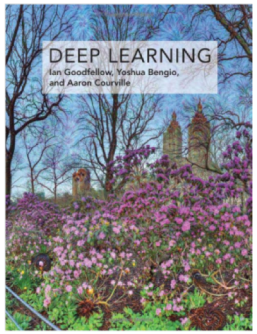
- Use cases in deep learning using python.
- NLP use cases & word embedding.
- Object detection in computer vision.
- Bagging and Boosting use cases.
- Recommenders

Course Evaluation

- Assignments during lectures and tutorials.
- Final class project

References:

- *"Deep Learning",* Ian Goodfellow ,Yoshua Bengio & Aaron Courville.



Markov

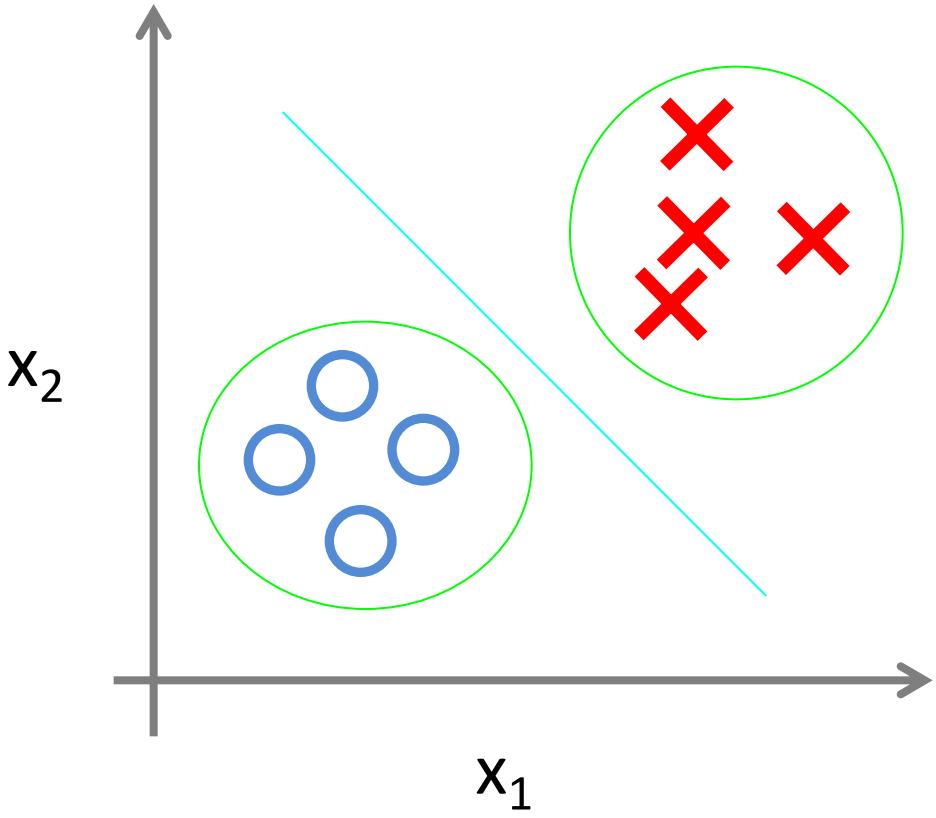
Linear Regression

Machine Learning Algorithms

- Supervised learning
- Unsupervised learning

- Others:
- Reinforcement learning
- Recommender systems.

Supervised Learning & Unsupervised Learning

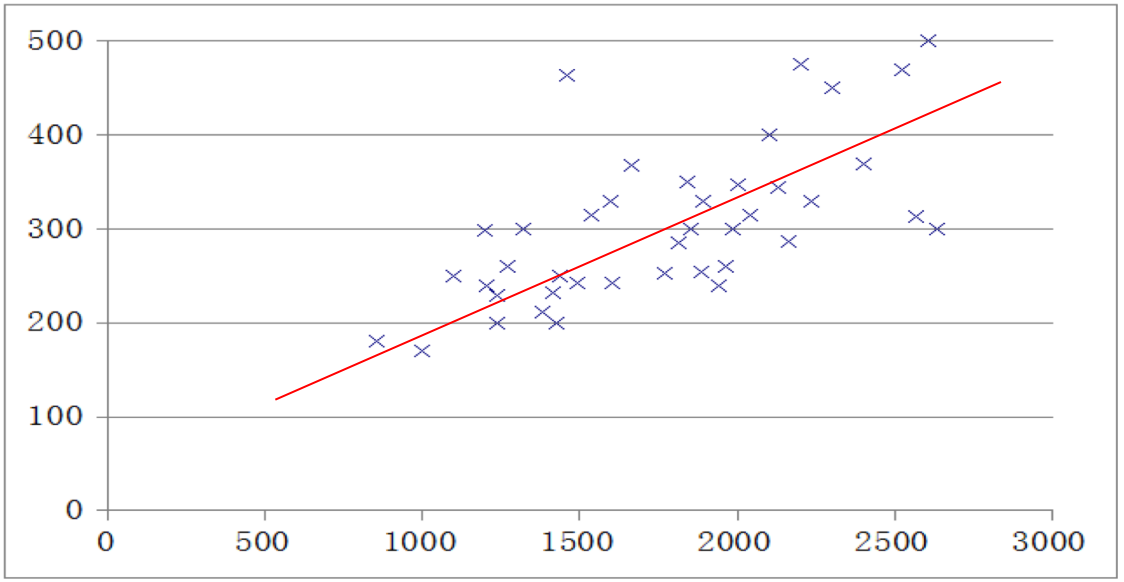


Supervised Learning

Unsupervised Learning

Linear Regression with one Variable

Housing Prices (Portland, OR)



Supervised Learning

Given the "right answer" for each example in the data.

Regression Problem

Predict real-valued output

Markov

Training set of housing prices

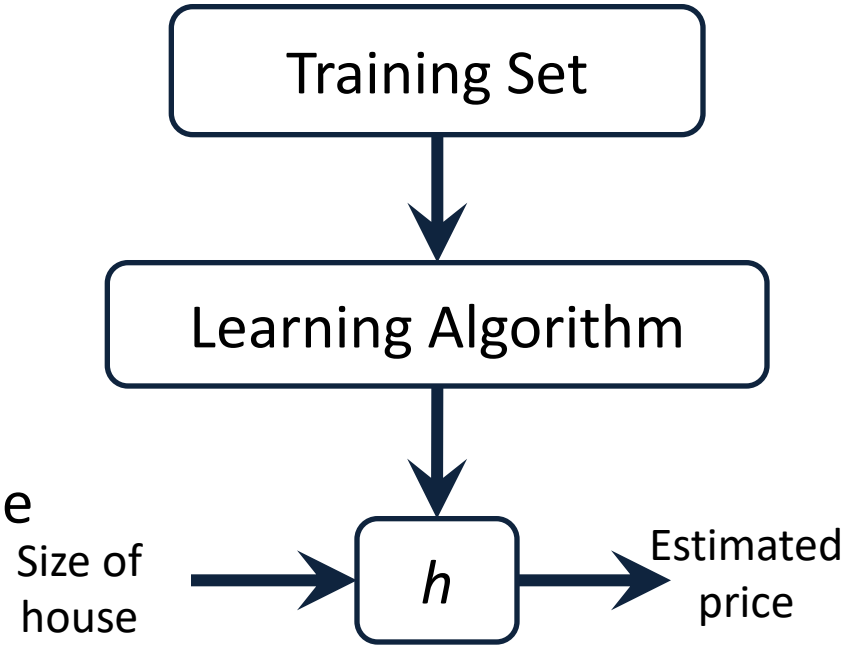
Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

Notation:

m = Number of training examples

x's = "input" variable / features

y's = "output" variable / "target" variable



Question : How to describe ***h***?

Training Set

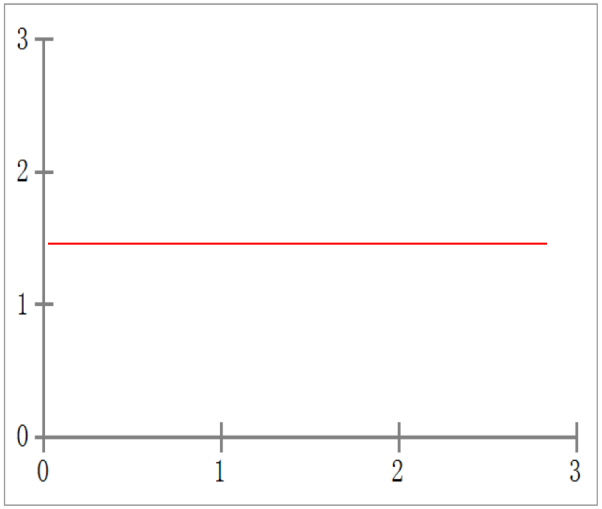
Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

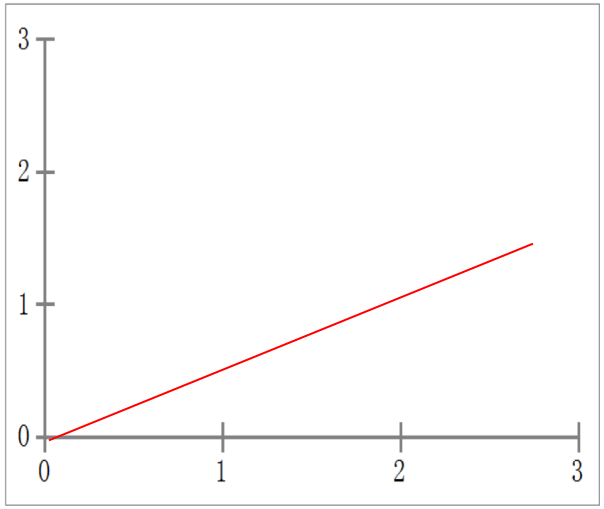
θ_i's: Parameters

How to choose θ_i's ?

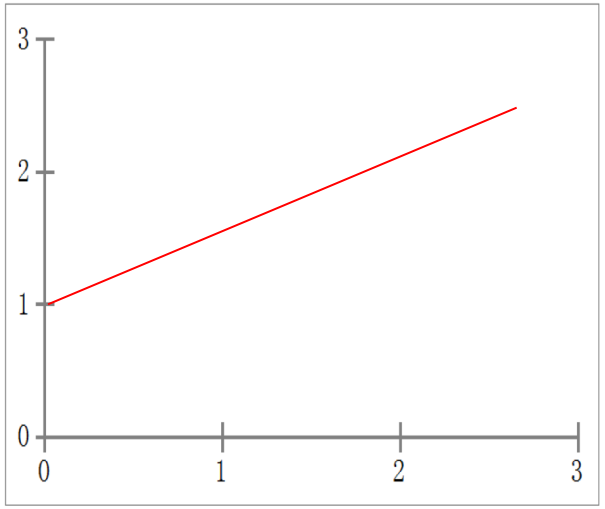
$$h_\theta(x) = \theta_0 + \theta_1 x$$



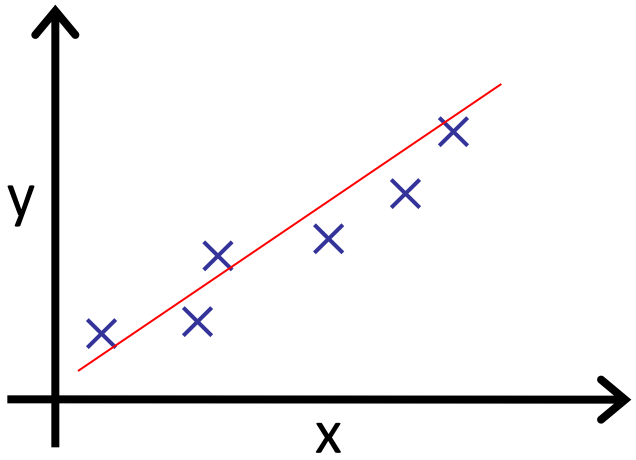
$\theta_0 = 1.5$
$\theta_1 = 0$



$\theta_0 = 0$
$\theta_1 = 0.5$



$\theta_0 = 1$
$\theta_1 = 0.5$



Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

Cost Function

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

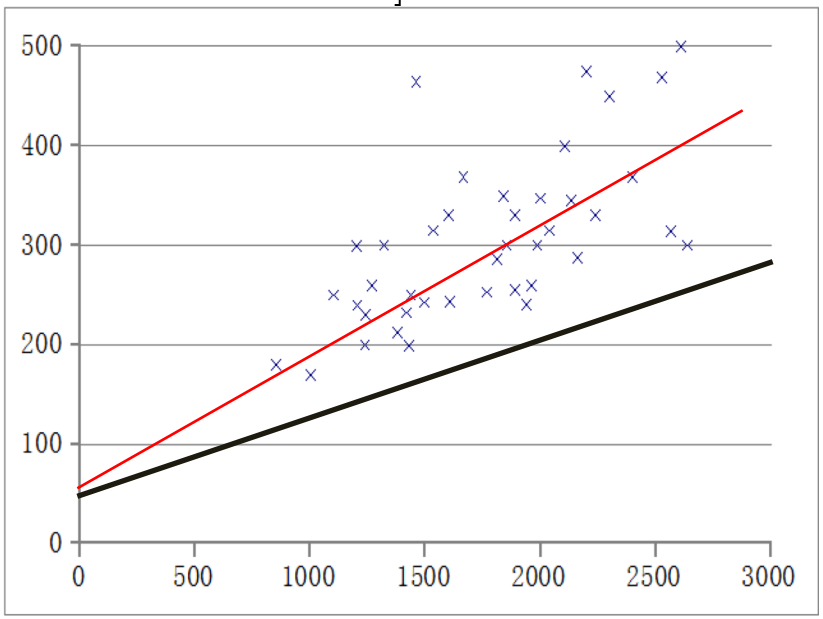
Simplified:

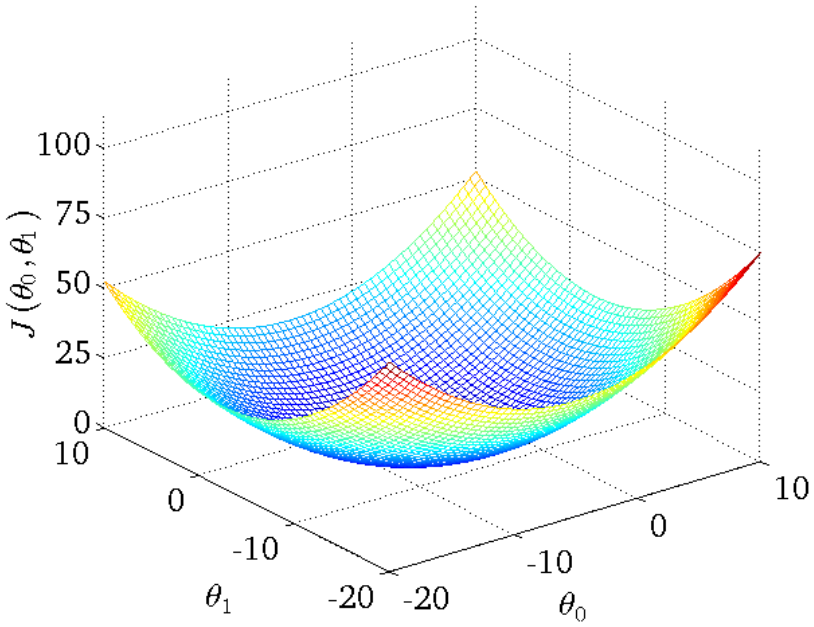
$$h_\theta(x) = \theta_1 x$$

$$\theta_1$$

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$





$$h_\theta(x) = 50 + 0.06x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$$

Question:How to minimize ***J***?

Markov

Gradient Descent

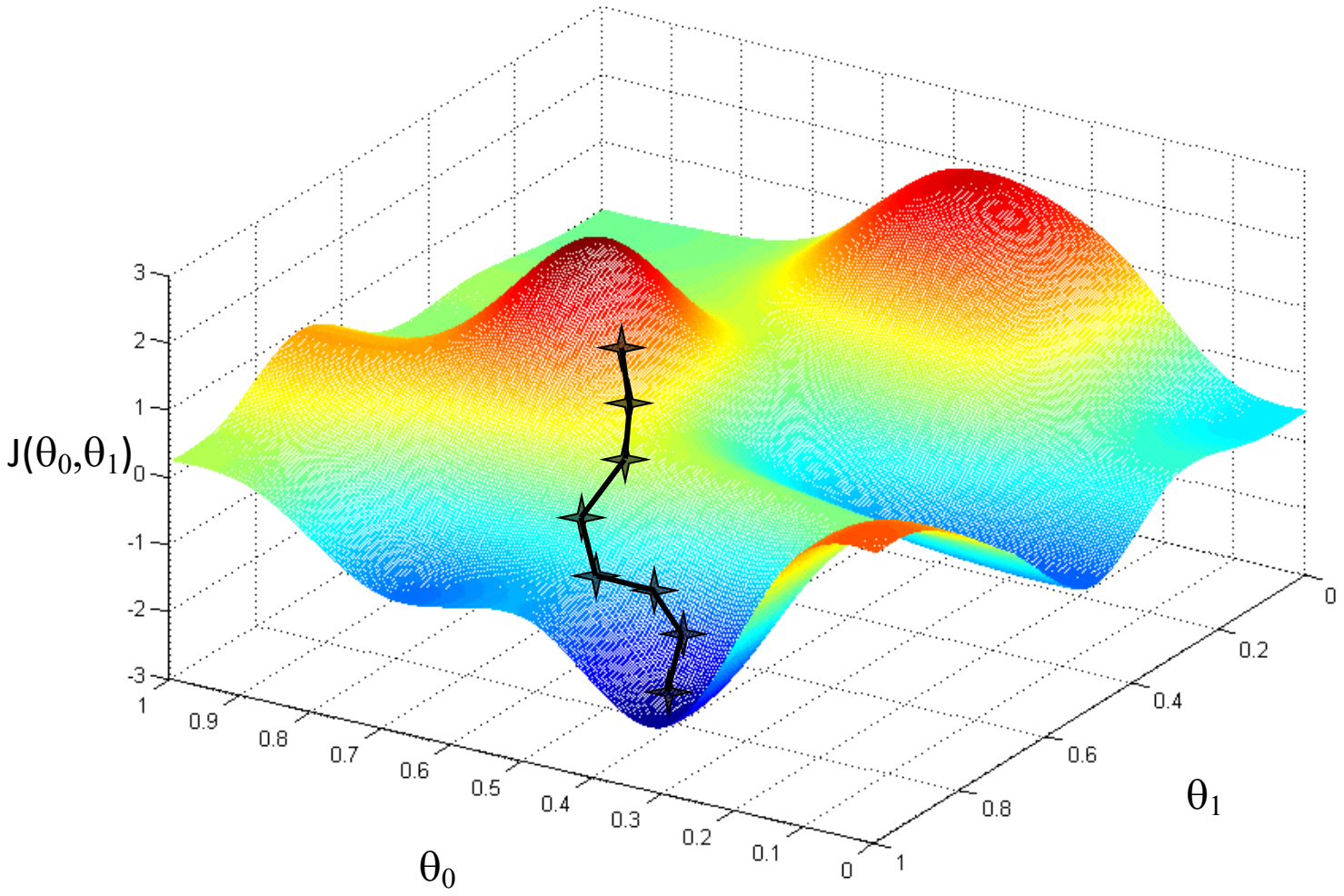
Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$

  until we hopefully end up at a minimum



Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

Gradient descent algorithm

repeat until convergence {

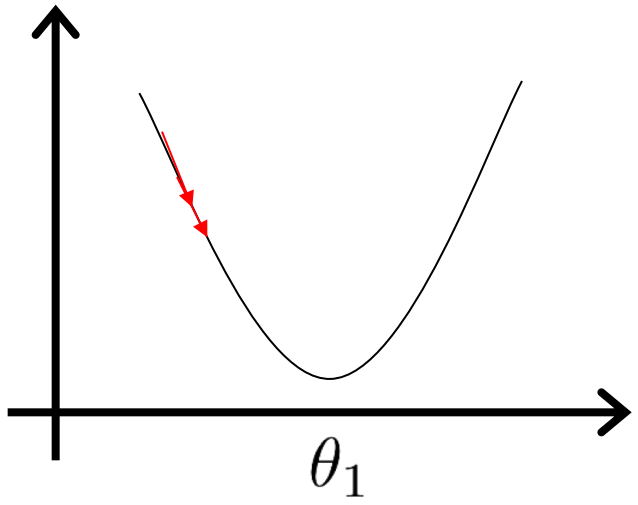
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$ (simultaneously update $j = 0$ and $j = 1$)

}

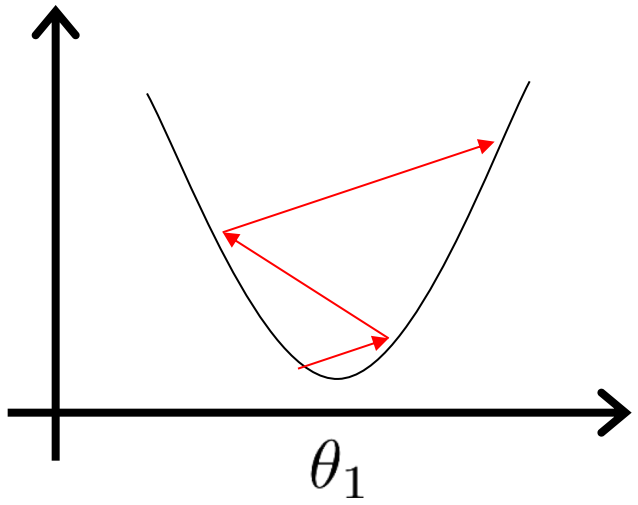
Notice : α is the learning rate.

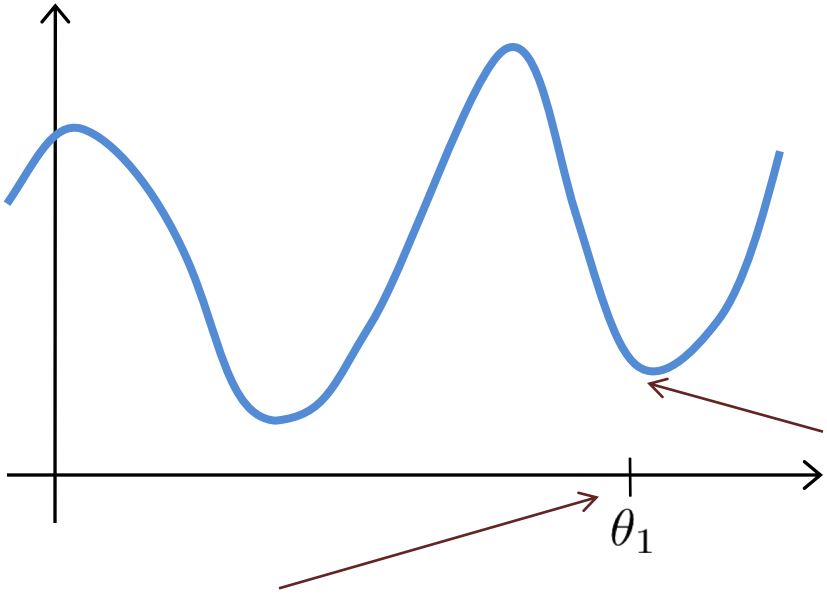
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.





θ_1 at local optima

Current value of θ_1

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Unchange

Gradient descent can converge to a local minimum, even with the learning rate α fixed.

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.

Gradient Descent for Linear Regression

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Gradient descent algorithm

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

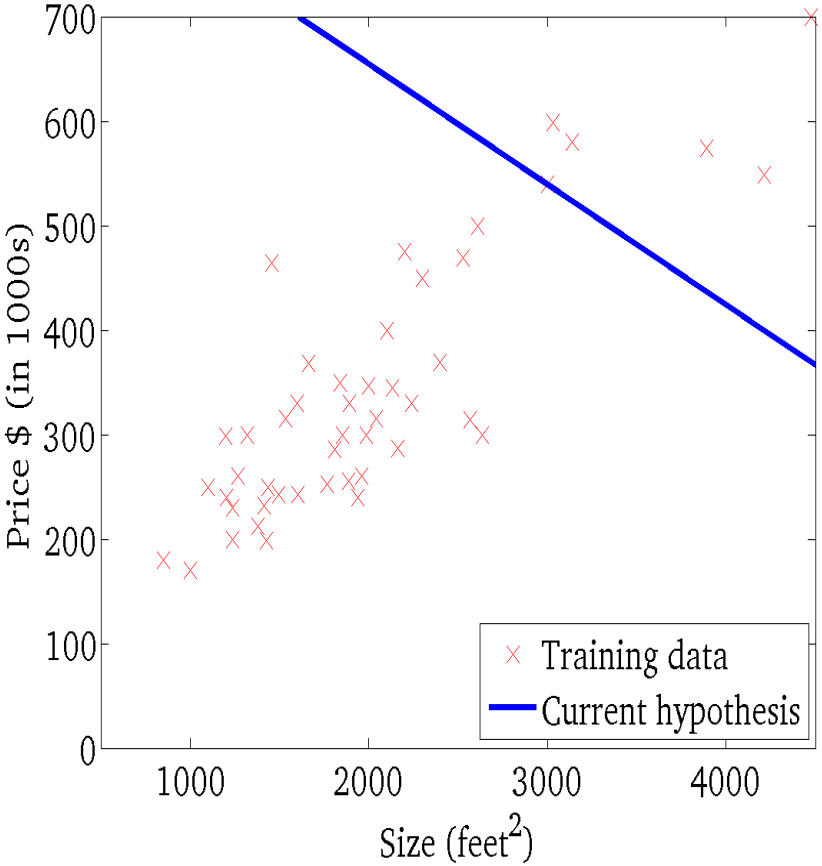
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

update θ_0 and θ_1 simultaneously

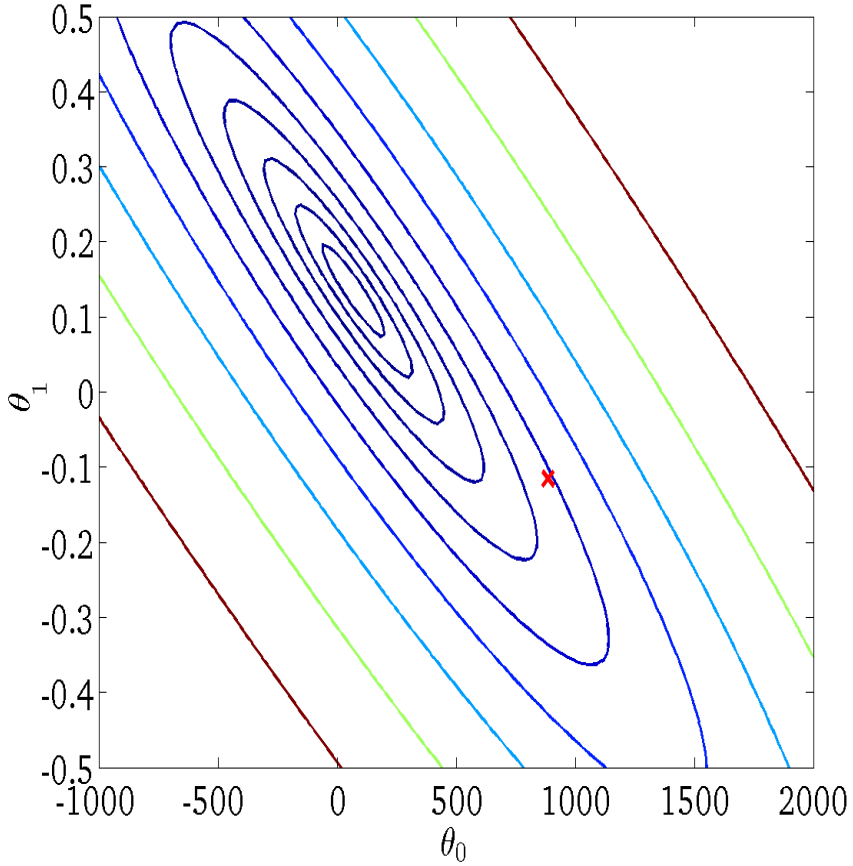
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

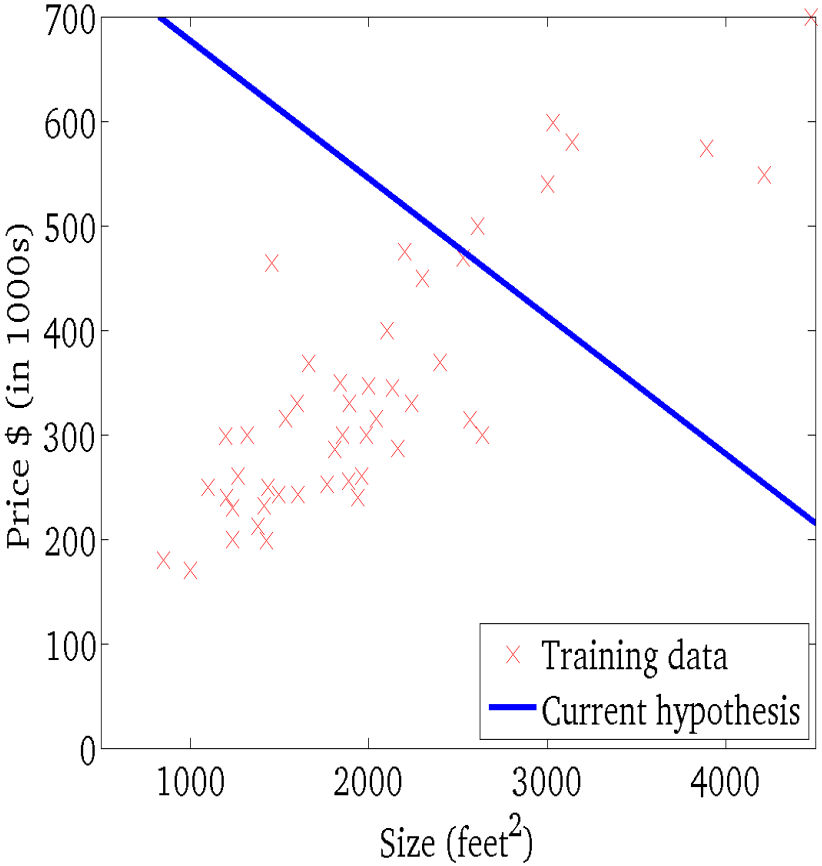
(function of the parameters θ_0, θ_1)



Markov

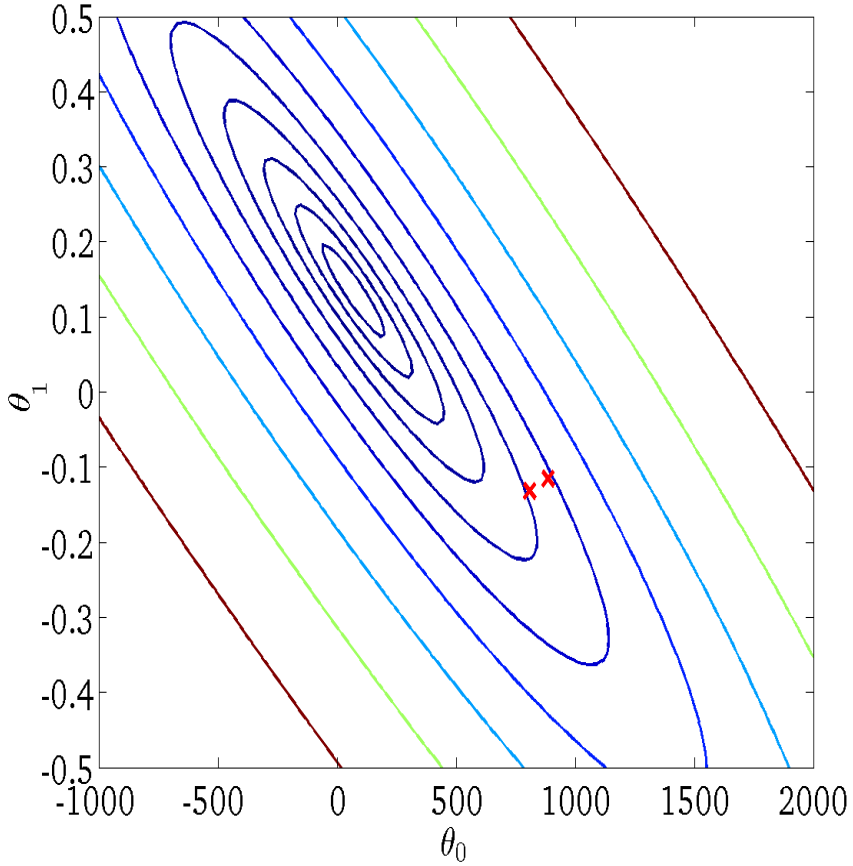
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

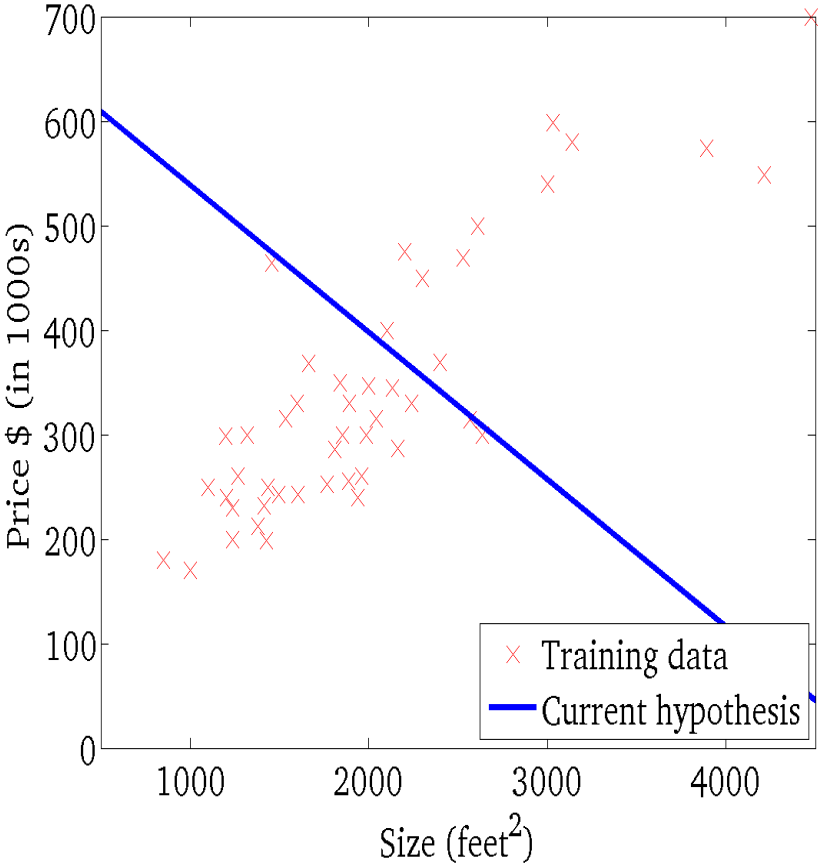
(function of the parameters θ_0, θ_1)



Markov

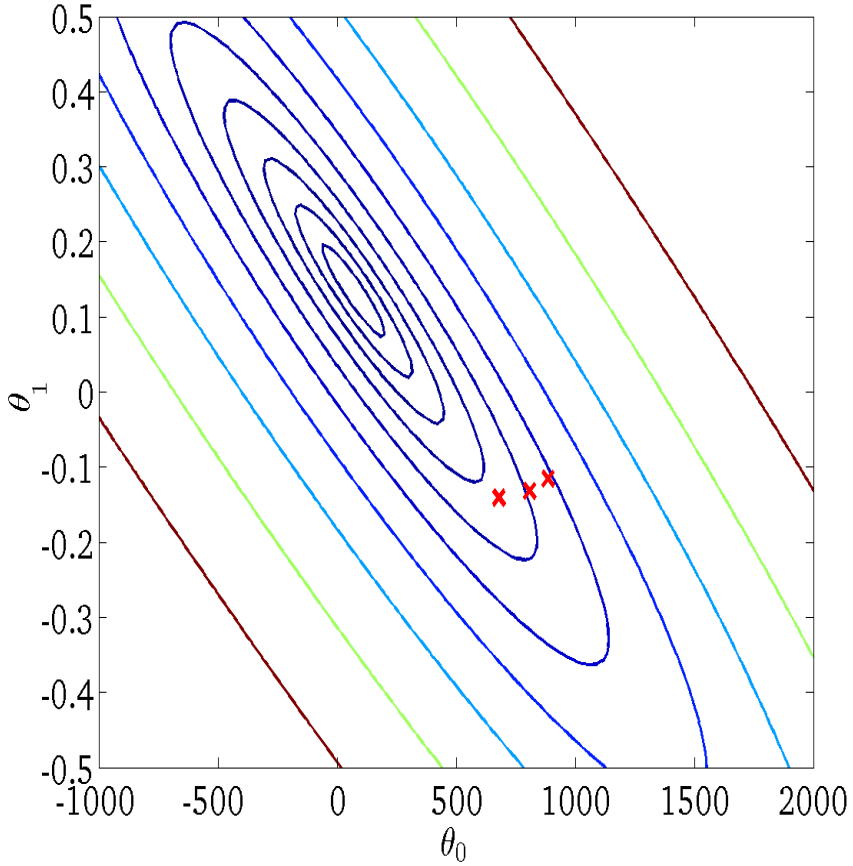
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

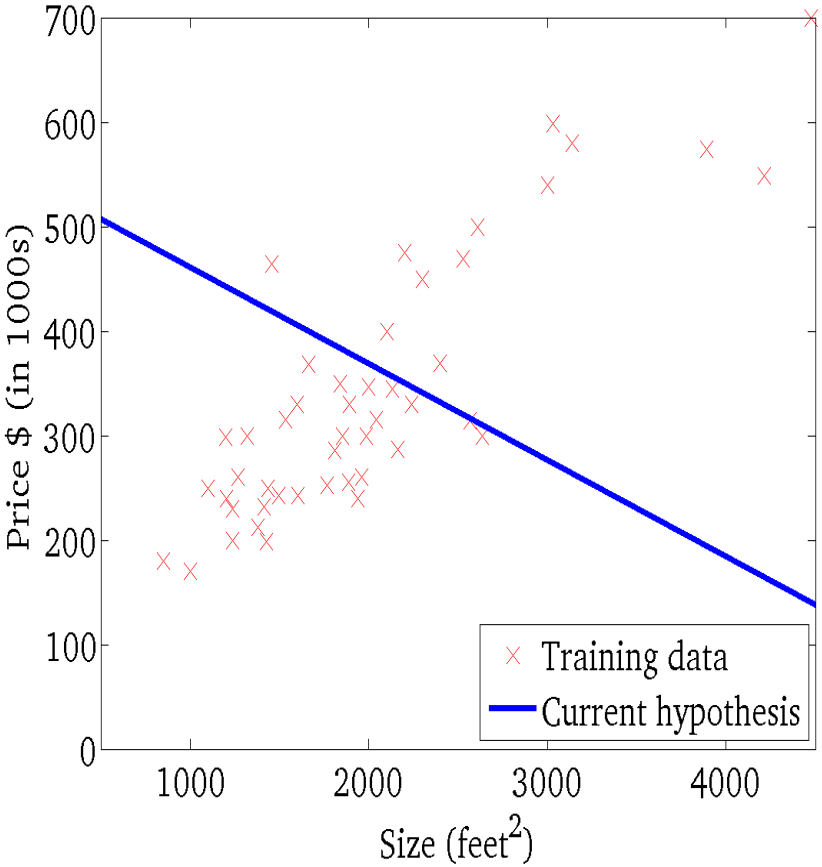
(function of the parameters θ_0, θ_1)



Markov

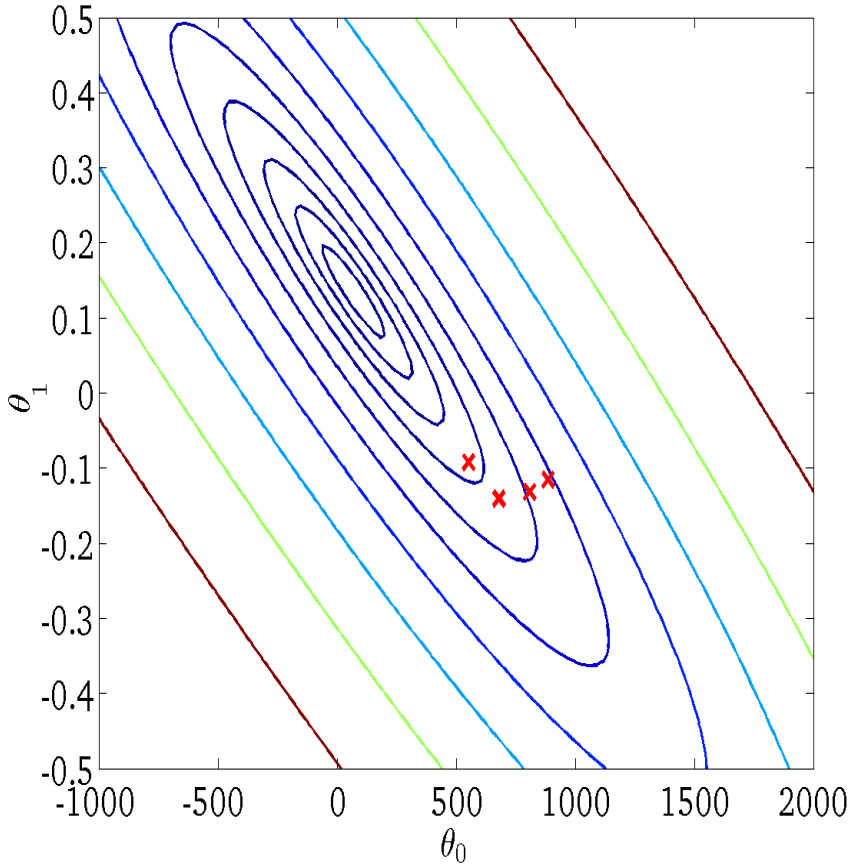
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

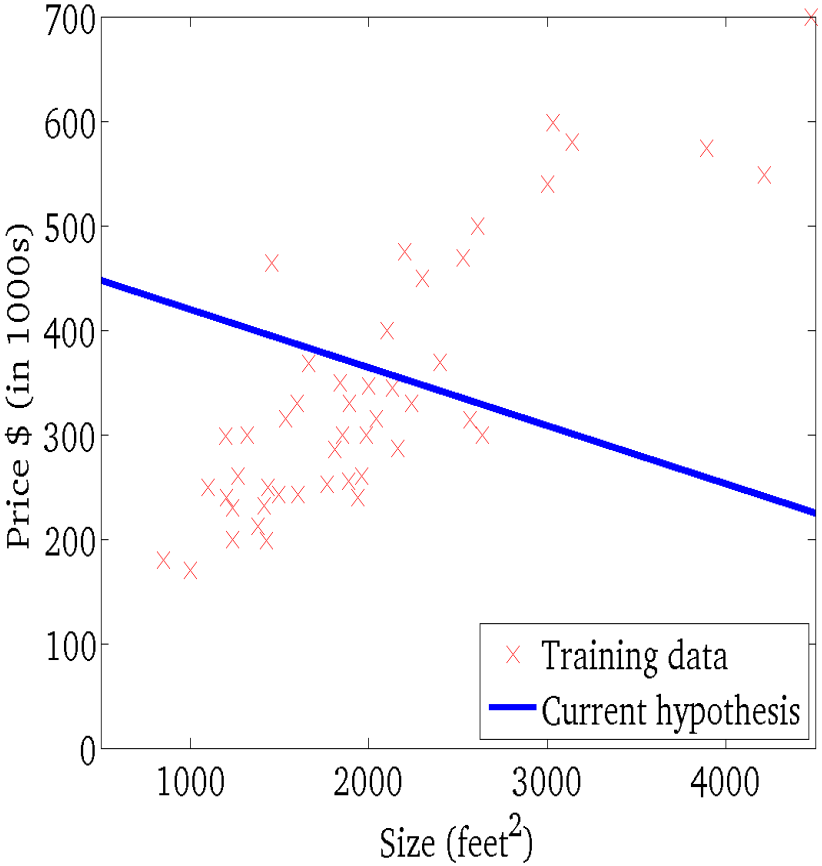
(function of the parameters θ_0, θ_1)



Markov

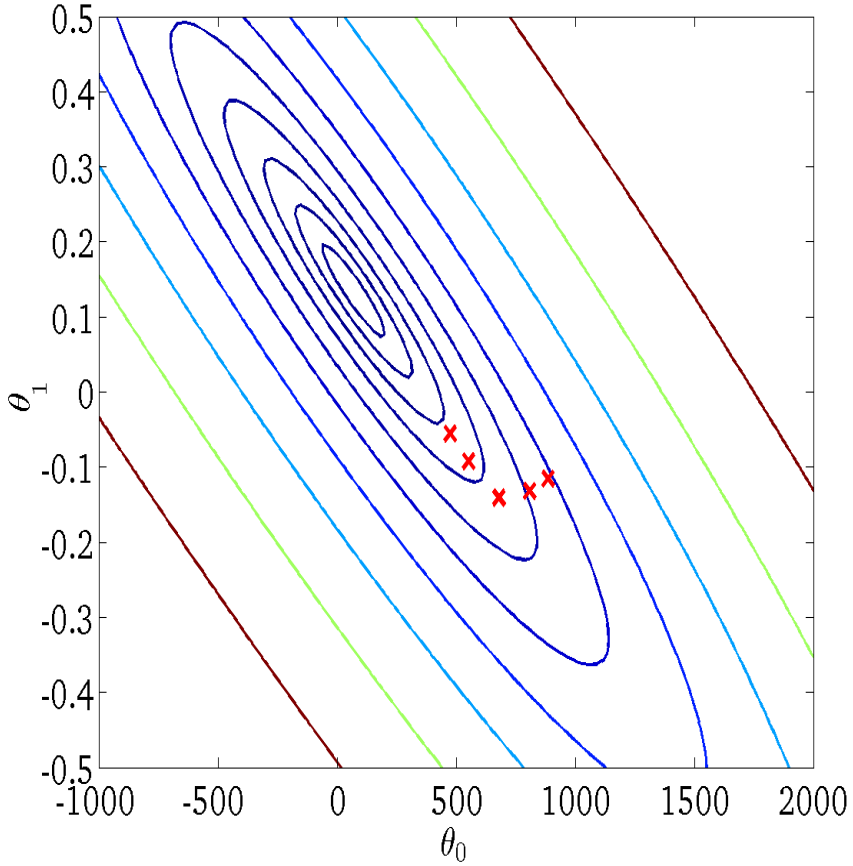
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

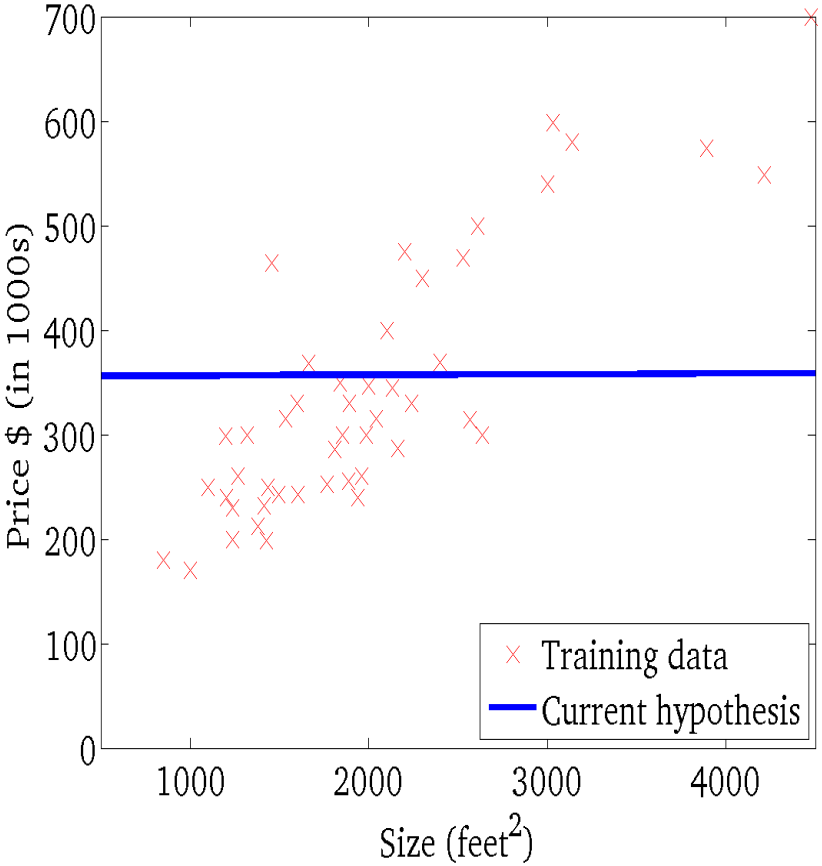
(function of the parameters θ_0, θ_1)



Markov

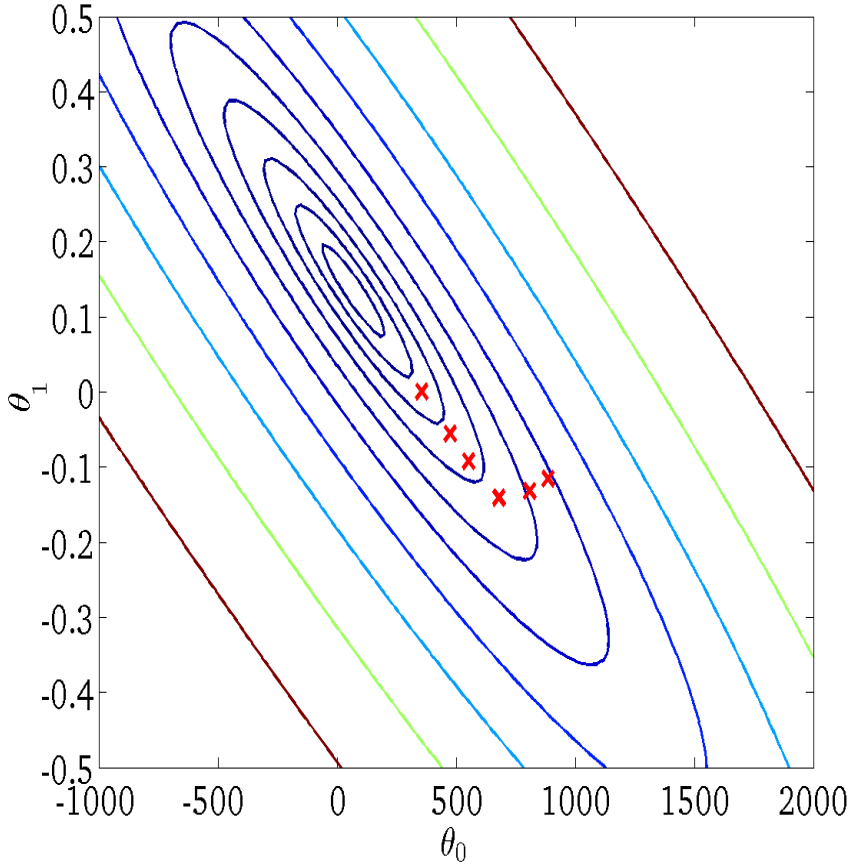
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

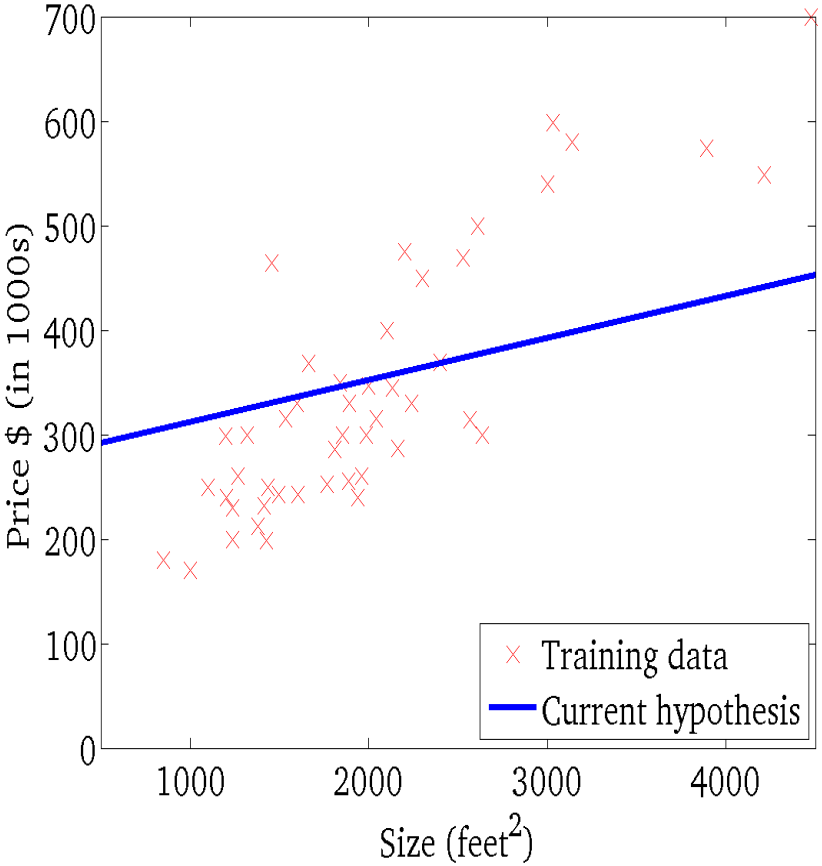
(function of the parameters θ_0, θ_1)



Markov

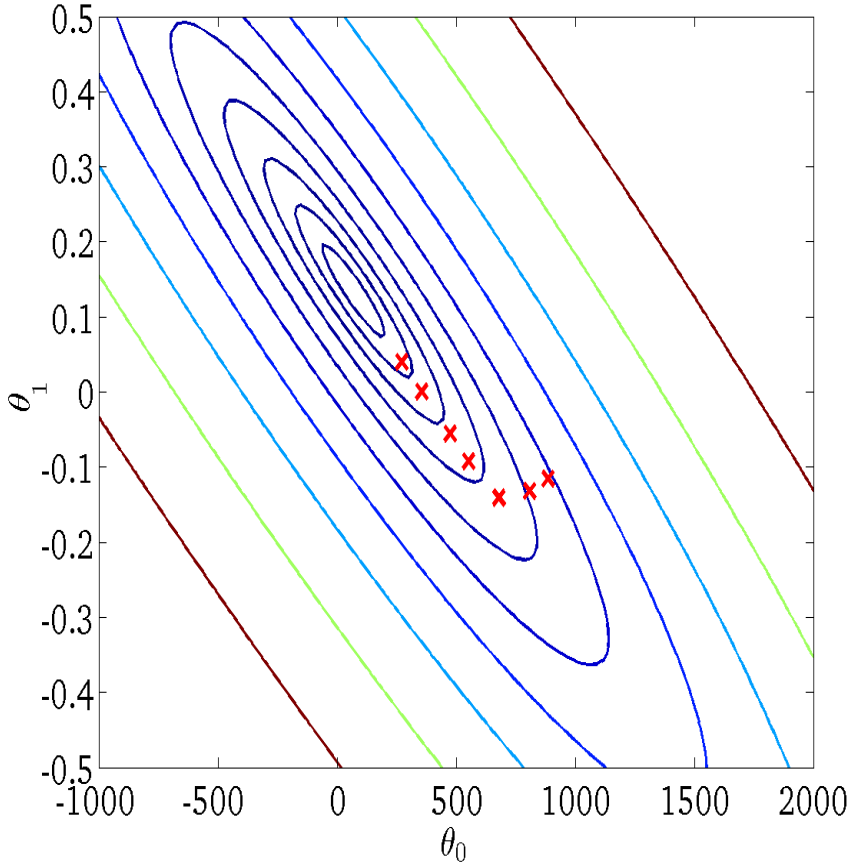
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

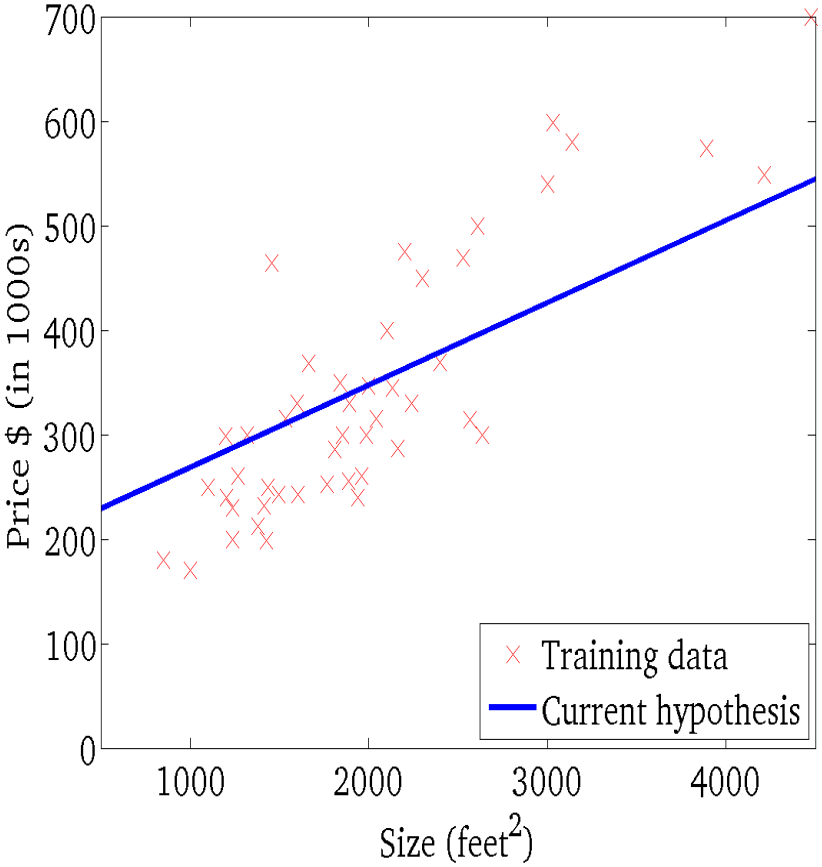
(function of the parameters θ_0, θ_1)



Markov

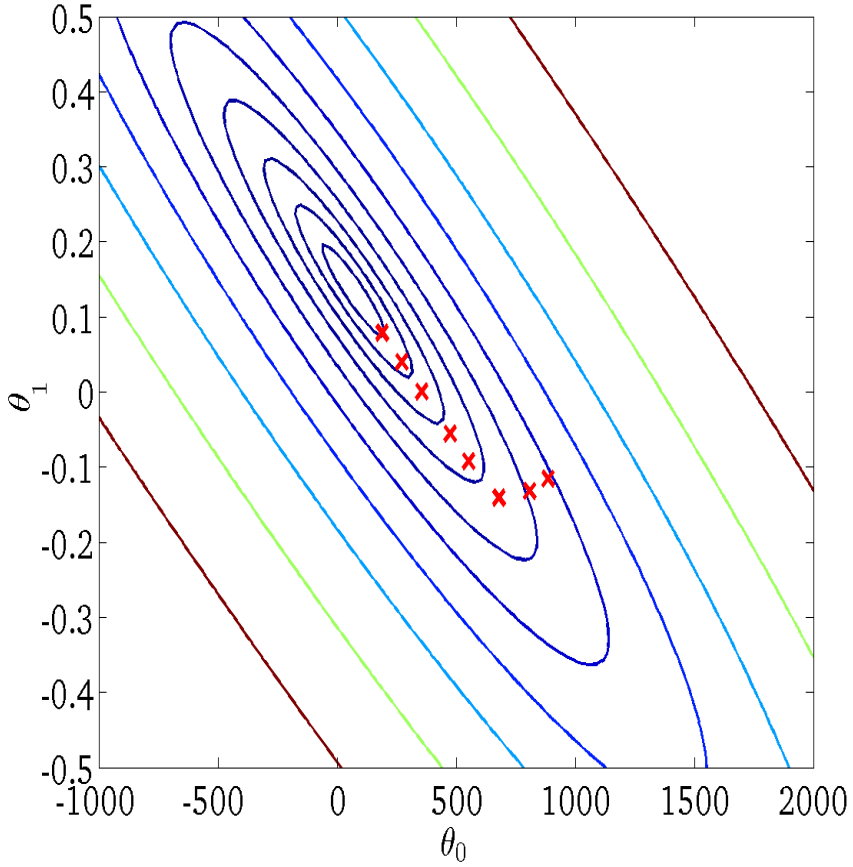
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

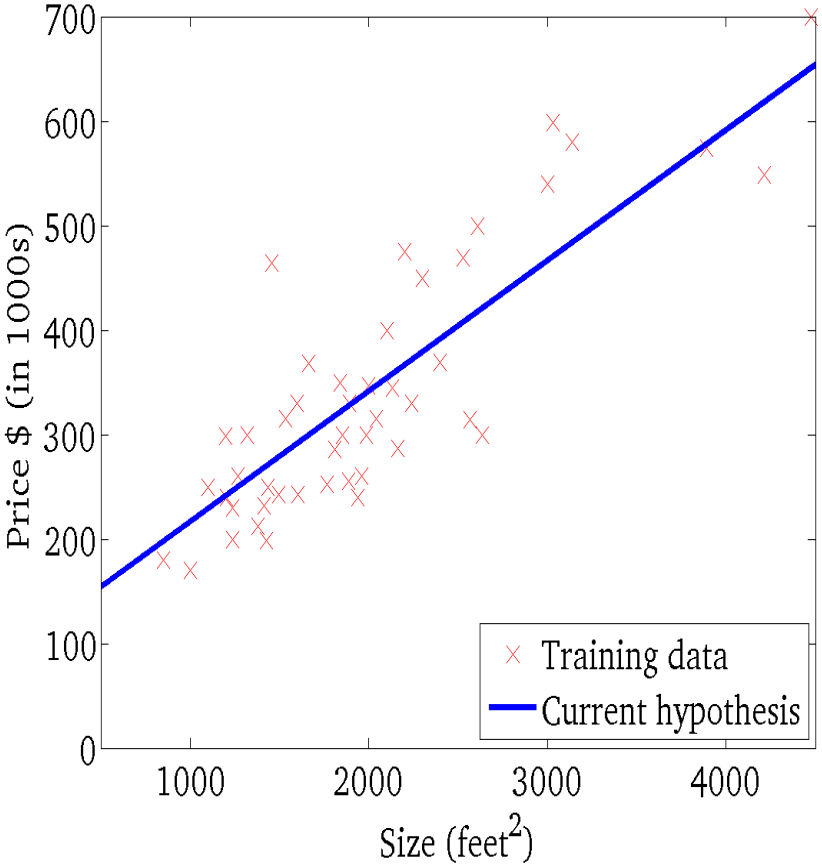
(function of the parameters θ_0, θ_1)



Markov

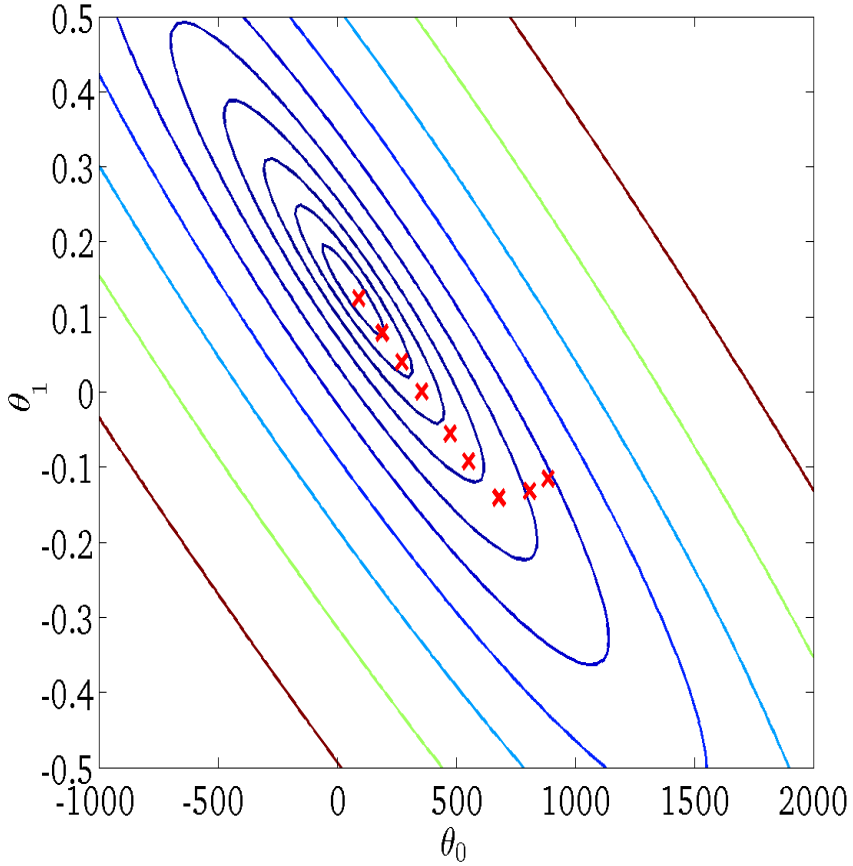
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Markov

Linear Regression with multiple variables

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

Repeat $\{$

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Gradient Descent

Previously (n=1):

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial\theta_0}J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

$\ldots$

Examples: $m = 4$.

x_0	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$$\theta = (X^T X)^{-1} X^T y$$

simultaneously update

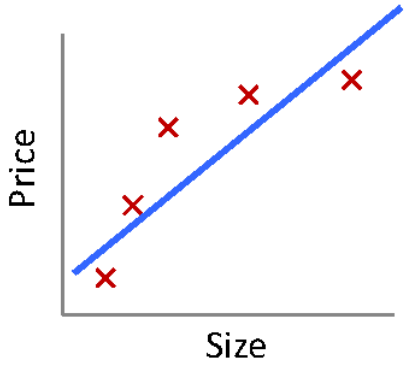
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\mathrm{A}^{-1} = \frac{1}{\det \mathrm{A}} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Gradient Descent vs Closed Form

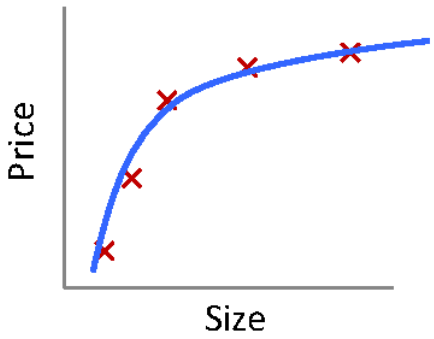
Gradient Descent	Closed Form Solution
• Requires multiple iterations • Need to choose α • Works well when n is large • Can support incremental learning	• Non-iterative • No need for α • Slow if n is large – Computing $(X^T X)^{-1}$ is roughly $O(n^3)$

Quality of Fit



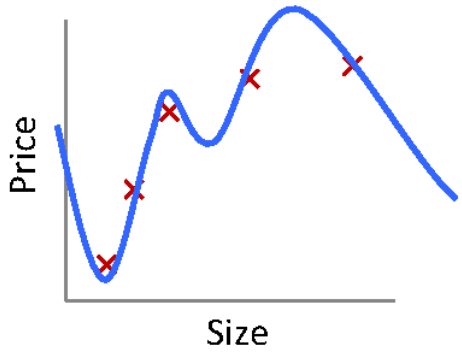
$\theta_0 + \theta_1 x$

Underfitting
(high bias)



$\theta_0 + \theta_1 x + \theta_2 x^2$

Correct fit



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfitting
(high variance)

Overfitting:

- The learned hypothesis may fit the training set very well ($J(\boldsymbol{\theta}) \approx 0$)
- ...but fails to generalize to new examples

Regularization

- A method for automatically controlling the complexity of the learned hypothesis
- **Idea**: penalize for large values of θ_j
  - Can incorporate into the cost function
  - Works well when we have a lot of features, each that contributes a bit to predicting the label
- Can also address overfitting by eliminating features (either manually or via model selection)

Regularization

- Linear regression objective function

$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{regularization}}$$

  - λ is the regularization parameter ($\lambda \geq 0$)
  - No regularization on θ_0!

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Note that $\sum_{j=1}^{d}\theta_j^2 = \|\boldsymbol{\theta}_{1:d}\|_2^2$
  - This is the magnitude of the feature coefficient vector!

- We can also think of this as:

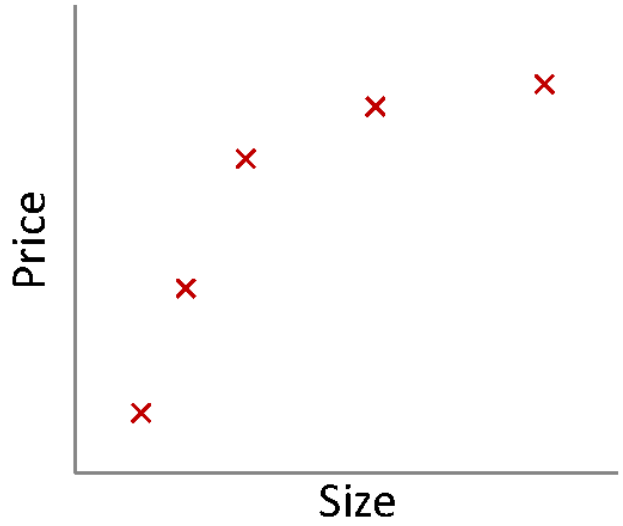
$$\sum_{j=1}^{d}(\theta_j - 0)^2 = \|\boldsymbol{\theta}_{1:d} - \vec{\mathbf{0}}\|_2^2$$

  - L_2 regularization pulls coefficients toward 0

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



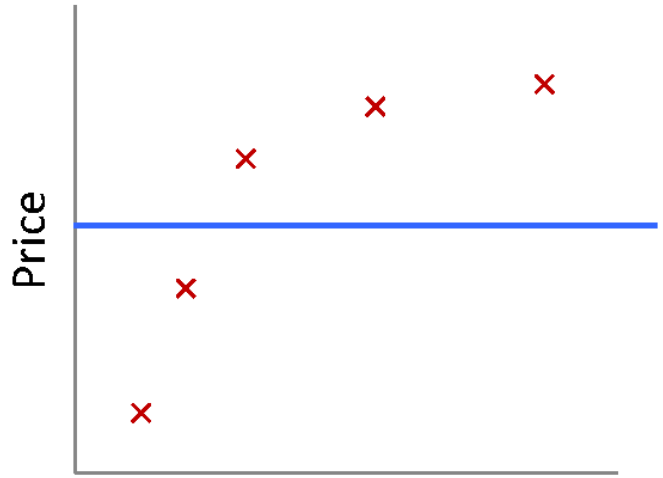
$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Based on example by Andrew Ng

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

0 0 0 0

Based on example by Andrew Ng

Regularized Linear Regression

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Gradient update:

$\frac{\partial}{\partial\theta_0}J(\theta)$

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$\frac{\partial}{\partial\theta_j}J(\theta)$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} - \lambda\theta_j$$

regularization

Markov

Regularized Linear Regression

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} - \lambda\theta_j$$

- We can rewrite the gradient step as:

$$\theta_j \leftarrow \theta_j\left(1 - \alpha\lambda\right) - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Regularized Linear Regression

- To incorporate regularization into the closed form solution:

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^\intercal \boldsymbol{X} \right)^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$$

Regularized Linear Regression

- To incorporate regularization into the closed form solution:

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^\intercal \boldsymbol{X} + \lambda \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \right)^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$$

- Can derive this the same way, by solving $\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$
- Can prove that for $\lambda > 0$, inverse exists in the equation above

Assignment #1

- Use the daily bike data set-day.csv. (https://archive.ics.uci.edu/ml/machine-learning-databases/00275/)
- Build a linear regression model using your inputs (normalized temperature (atemp), humidity and windspeed) and output is the “cnt”.
- Compare your results with that of the closed form.
- Plot your cost function with the number of iterations.

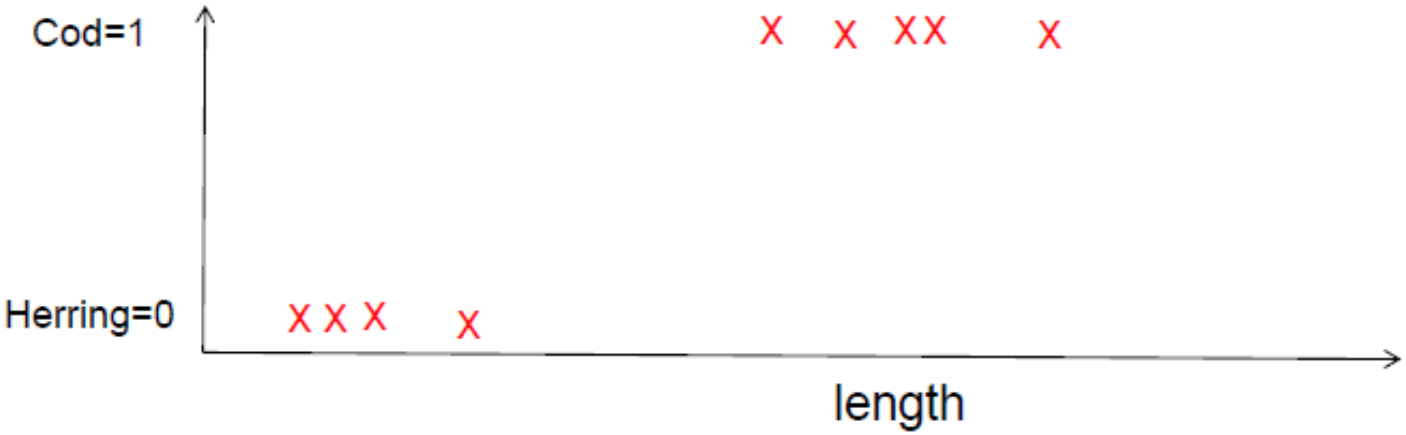
Hint: Normalize your data to speed your convergence

Markov

Logistic Regression

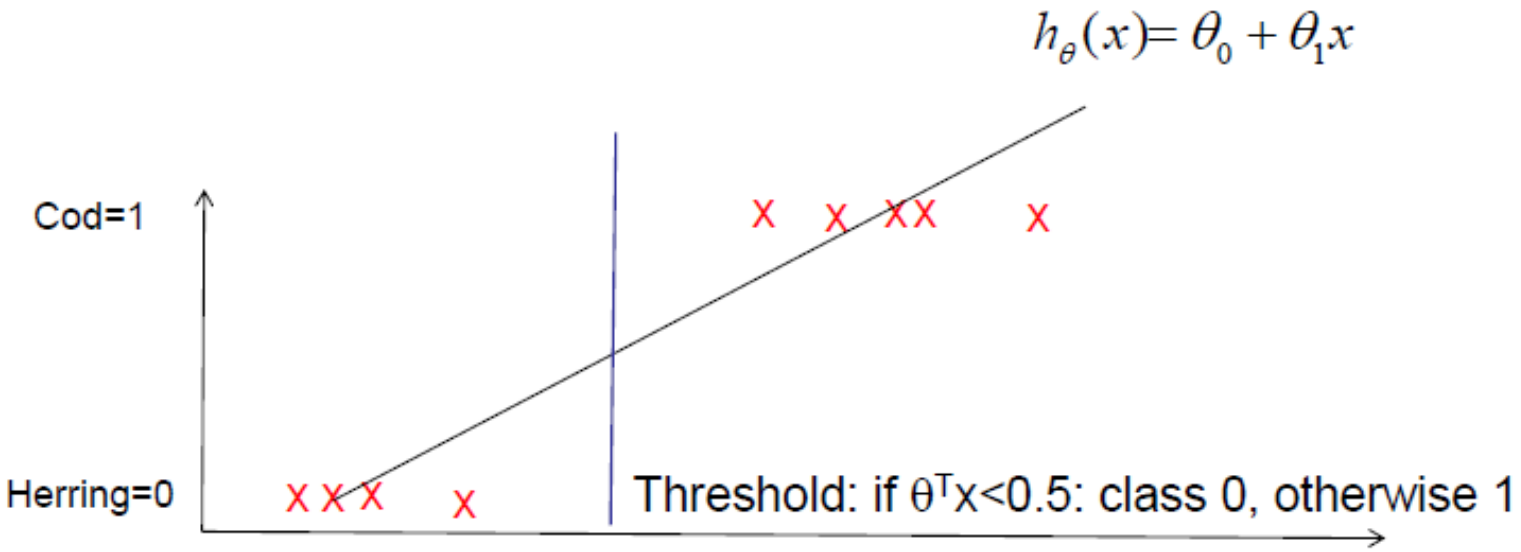
Introduction

- Consider classification into 2 classes. Call the classes 0 and 1 (or negative and positive)
- Example: classify fish species based on length



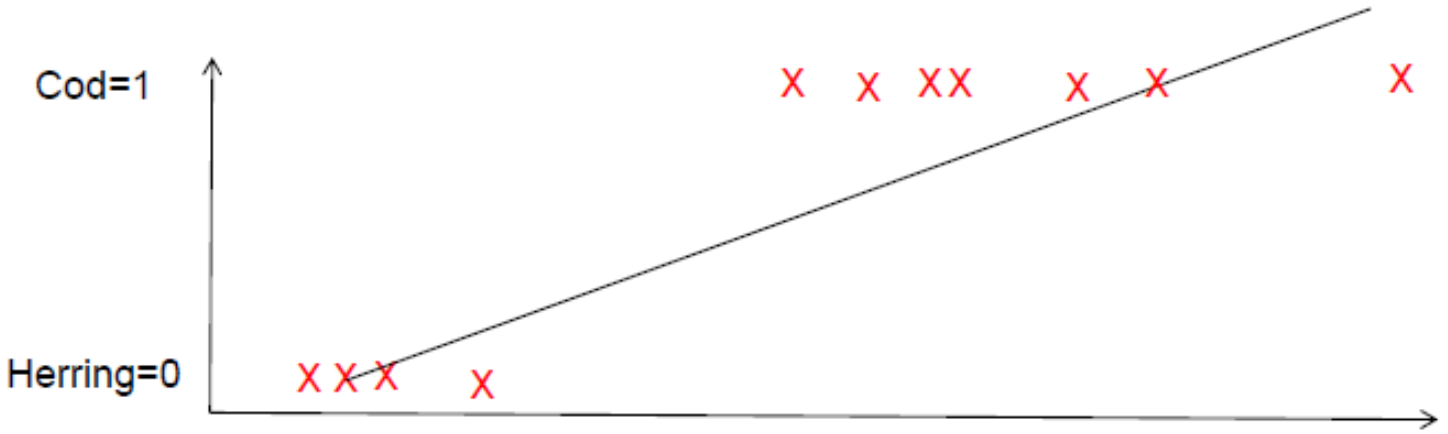
What would linear regression give?

- Maybe we would threshold this?



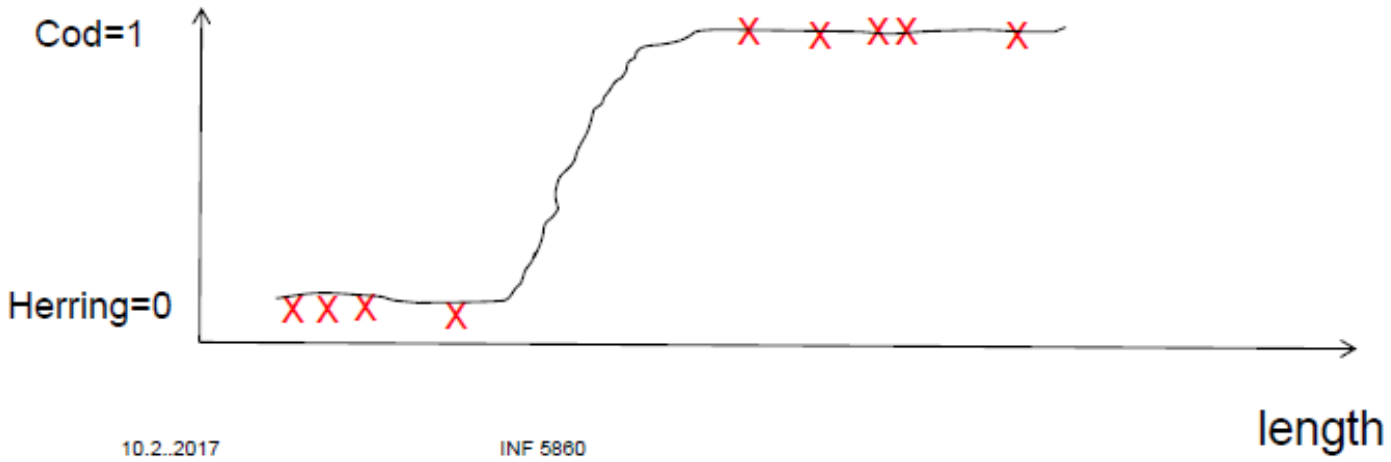
What would linear regression give?

- But what if we got more data?
  The line (and threshold) would change completely!



What if we fitted it to a function that is close to either 0 or 1?

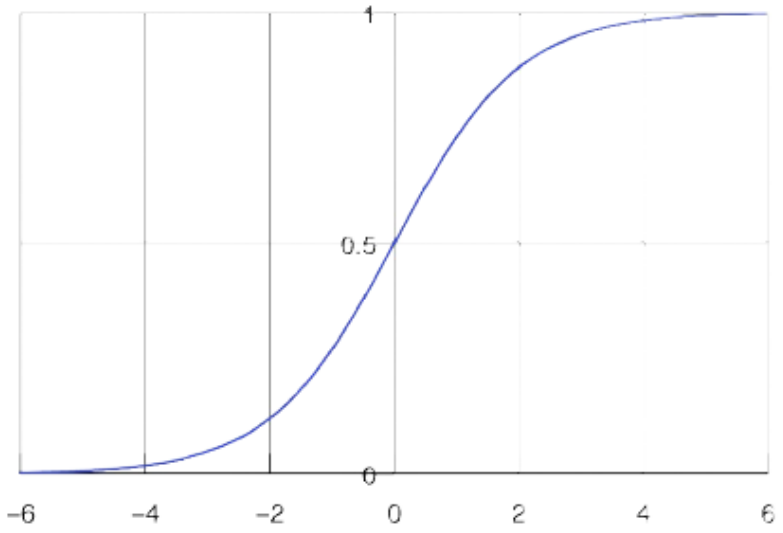
- Hypothesis $h_\theta(x)$ is now a non-linear function of x
  Classification: y=0 or 1
  Threshold $h_\theta(x)$: if $h_\theta(x)>1$: set y=1, otherwise set y=0
- Desirable to have $0 \leq h_\theta(x) \leq 1$



Logistic Function (Sigmoid Function)

$$g(z) = \frac{1}{1+e^{-z}}$$

- **maps $\mathbb{R}$ into interval** [0;1]
- 0 **asymptote for** $x \to -\infty$
- 1 **asymptote for** $x \to \infty$



Sigmoid Function (S-shape)

Logistic Function

Markov

- **Hypothesis** $h_\theta(x) = g(\theta^T x)$

$$= \frac{1}{1 + e^{-\theta^T x}}$$

- **Interpretation**

$$h_\theta(x) = p(y = 1 | x, \theta)$$

- **Because probabilites should sum to 1, define**

$$p(y = 0 | x, \theta) := 1 - p(y = 1 | x, \theta)$$

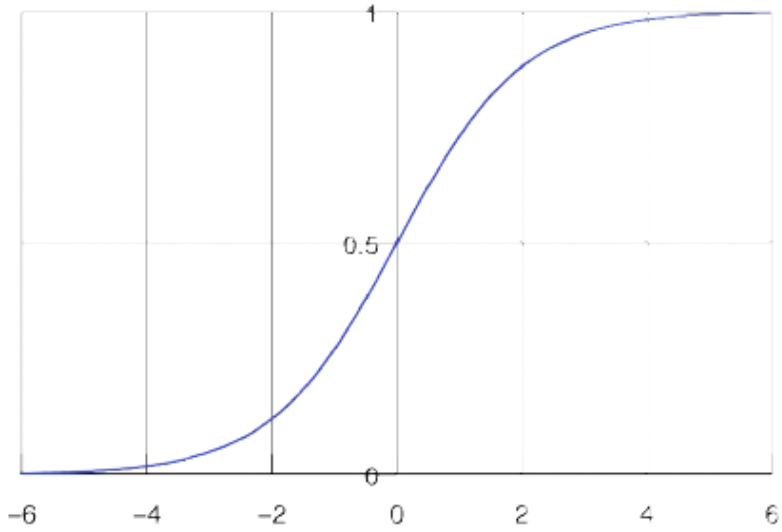
- **If $h_\theta(x) = 0.7$ interpret as 70% chance data point belongs to class**
- **If $h_\theta(x) \geq 0.5$ classify as *positive sentiment, malignant tumor, ...***

Markov

Logistic regression
Decision boundary

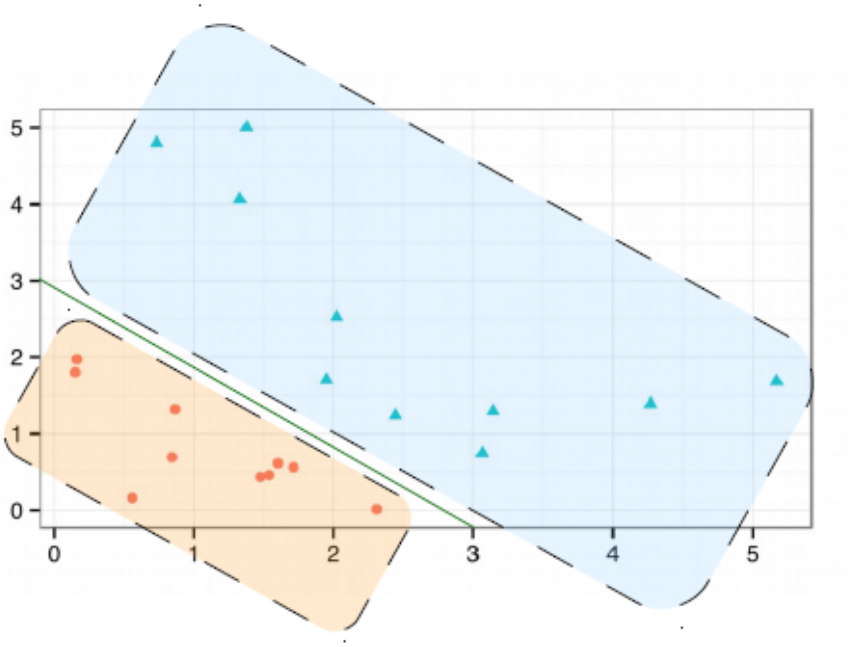
$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

- **If** $h_\theta(x) \geq 0.5$
  or equivalently $\theta^T x \geq 0$
  predict y = 1
- **If** $h_\theta(x) < 0.5$
  or equivalently $\theta^T x < 0$
  predict y = 0



Markov

Example



- If $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

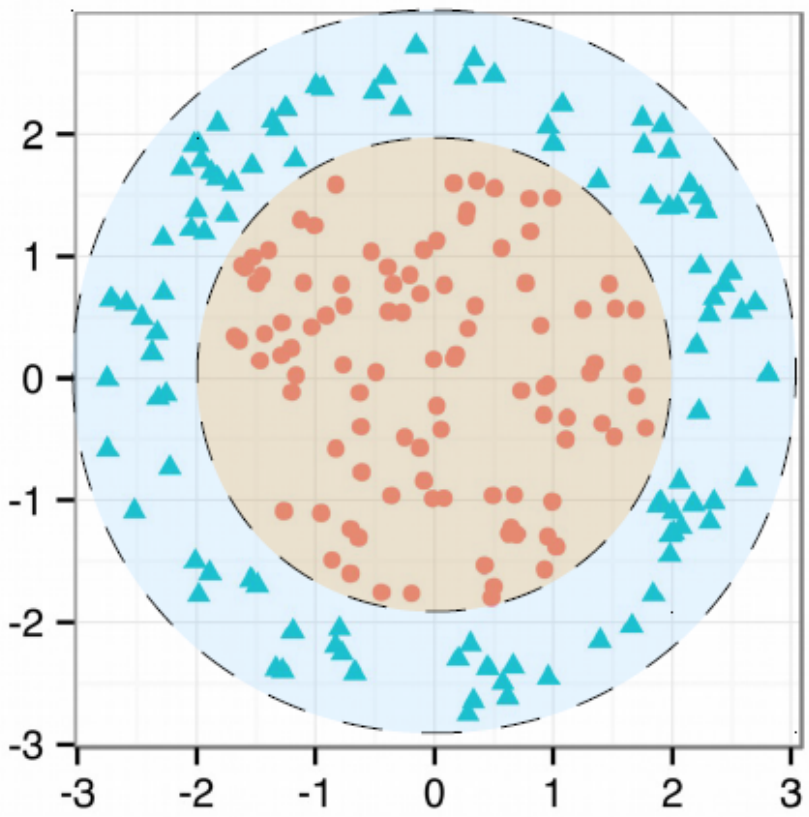
  and $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$

- **Prediction y = 1 whenever**

$$\begin{aligned} & \theta^T x && \geq 0 \\ \Leftrightarrow \quad & -3 + x_1 + x_2 && \geq 0 \\ \Leftrightarrow \quad & x_1 + x_2 && \geq 3 \end{aligned}$$

Markov

Example



- **If**

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

and

$$\theta = \begin{bmatrix} -2 & 0 & 0 & 1 & 1 \end{bmatrix}^T$$

- **Prediction y = 1 whenever**

$$x_1^2 + x_2^2 \geq 2$$

Markov

Logistic regression
Cost Function

Training and cost function

- **Training data wih m datapoints, n features**

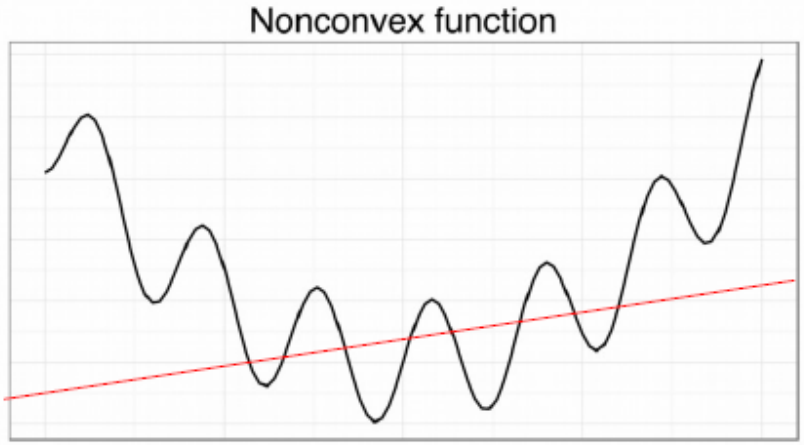
$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

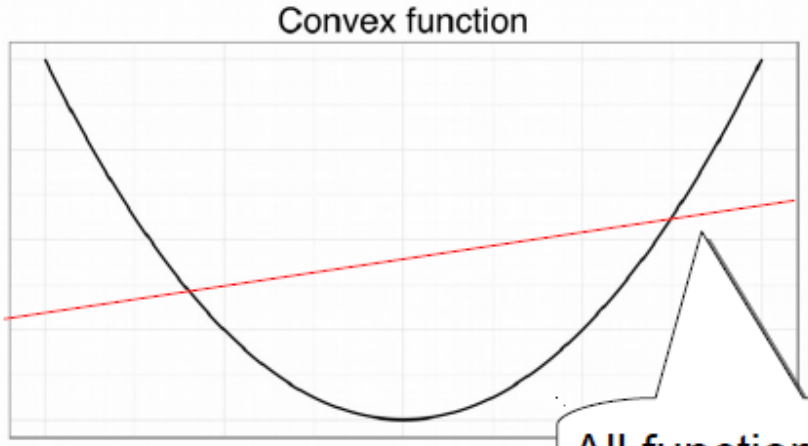
where

$$x \in \mathbb{R}^{n+1} \text{ with } x_0 := 1, y \in \{0, 1\}$$

- **Average cost**

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$





All function values below intersection with *any* line

- **Cost from linear regression**

$$Cost(h_\theta(x), y) := \frac{1}{2}\left(h_\theta(x) - y\right)^2$$

with logistic regression hypothesis

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

leads to non-convex average cost

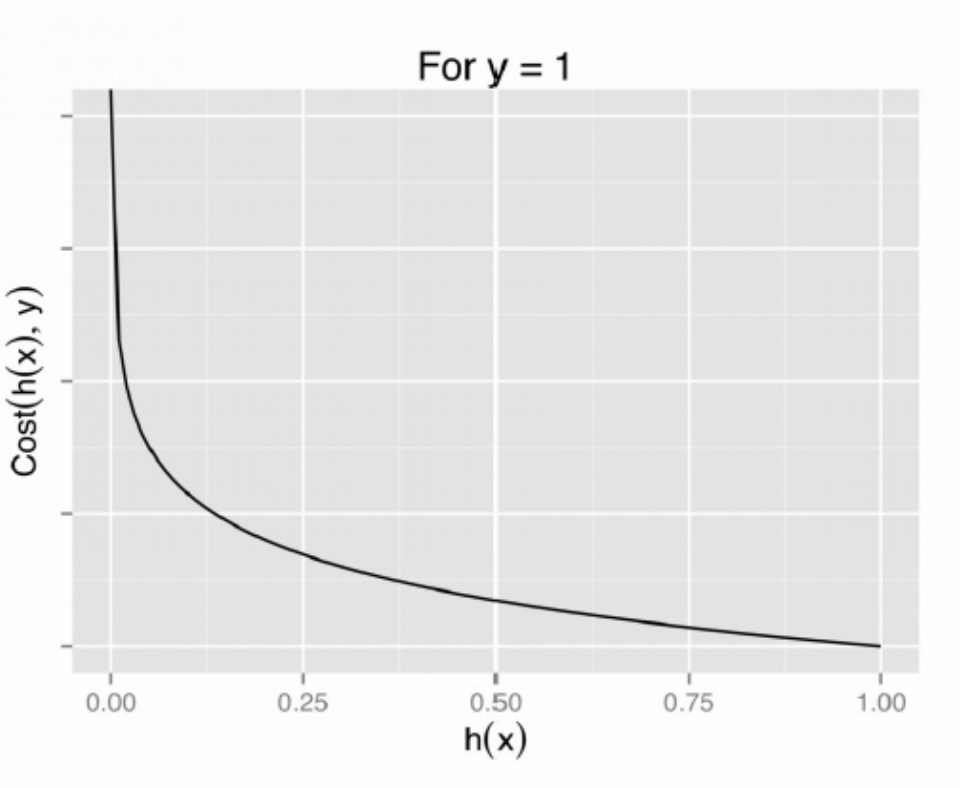
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

- **Convex *J* easier to optimize (no local optima)**

Markov

Logistic Regression Cost function

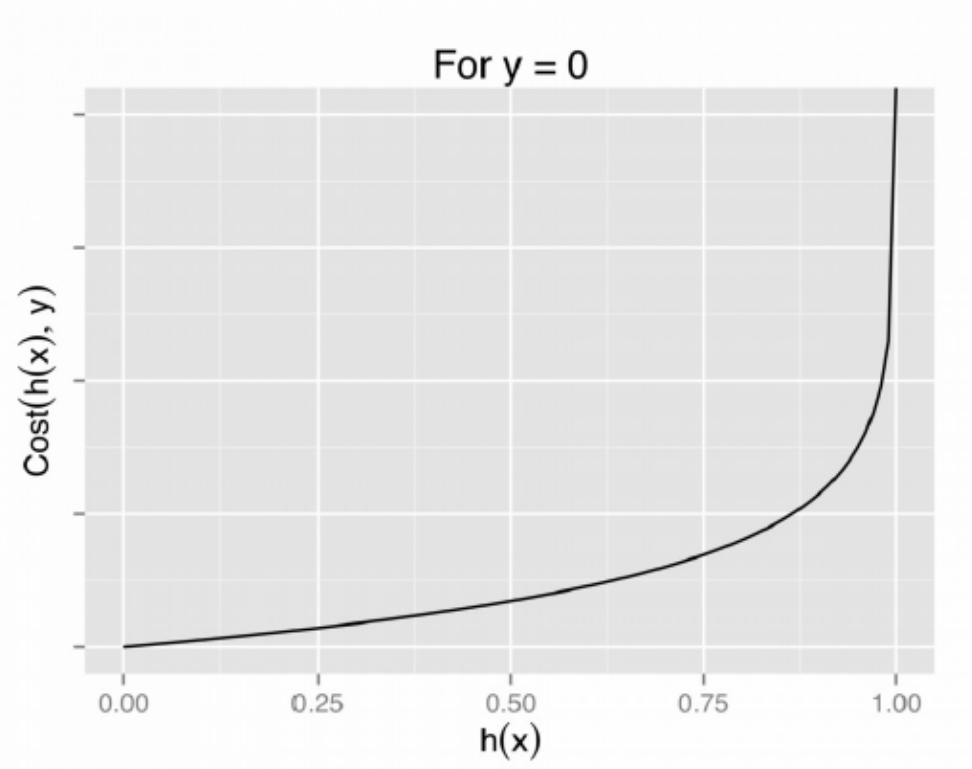
$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



- **If** y = 1 **and** h(x) = 1, Cost = 0
- **But for** $h(x) \to 0$
  $Cost \to \infty$
- **Corresponds to intuition: if prediction is** h(x) = 0 **but actual value was** y = 1, **learning algorithm will be penalized by large cost**

Logistic Regression Cost function

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



- **If** y = 0 **and** h(x) = 0, Cost = 0
- **But for** $h(x) \to 1$

  $Cost \to \infty$

Logistic regression
Simplified Cost Function & Gradient Descent

Simplified Cost Function (1)

- **Original cost of single training example**

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

- **Because we always have *y* = *0* or *y* = *1* we can simplify the cost function definition to**

$$Cost(h_\theta(x), y) \quad = \quad -y \; log(h_\theta(x)) - (1 - y) log(1 - h_\theta(x))$$

- **To convince yourself, use the simplified cost function to calculate**

$$\begin{aligned} Cost(h_\theta(x), 1) &= -log(h_\theta(x)) \\ Cost(h_\theta(x), 0) &= -log(1 - h_\theta(x)) \end{aligned}$$

Markov

Simplified Cost Function (2)

- **Cost function for training set**

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)}\ log(h_\theta(x^{(i)})) + (1-y^{(i)}) log(1-h_\theta(x^{(i)}))\right)$$

- **Find parameter argument θ' that minimizes *J*:** $\underset{\theta}{argmin}\ J(\theta)$
- **To make predictions given new *x* output**

$$h_{\theta'}(x) = \frac{1}{1+e^{-\theta^T x}}$$

$$= p(y=1|x,\theta')$$

Gradient Descent for logistic regression

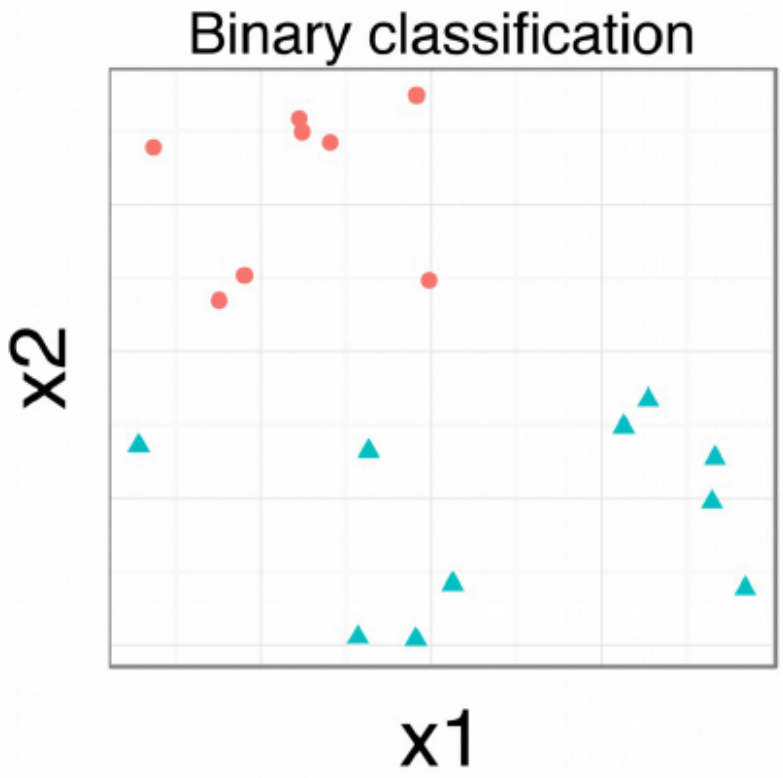
- **Gradient Descent to minimize logistic regression cost function**

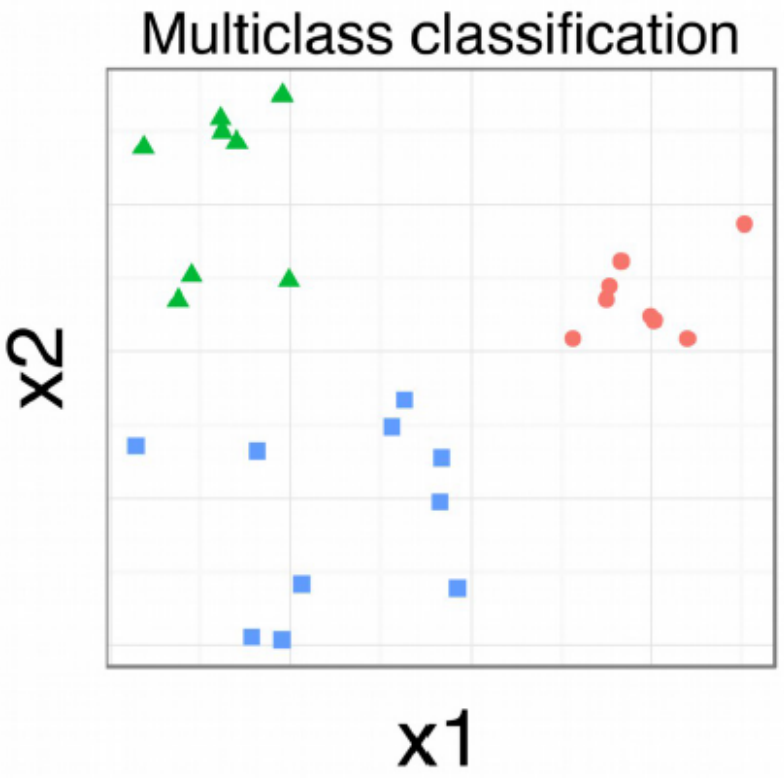
$$J(\theta) = -\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)}\ log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))\right)$$

with identical algorithm as for linear regression

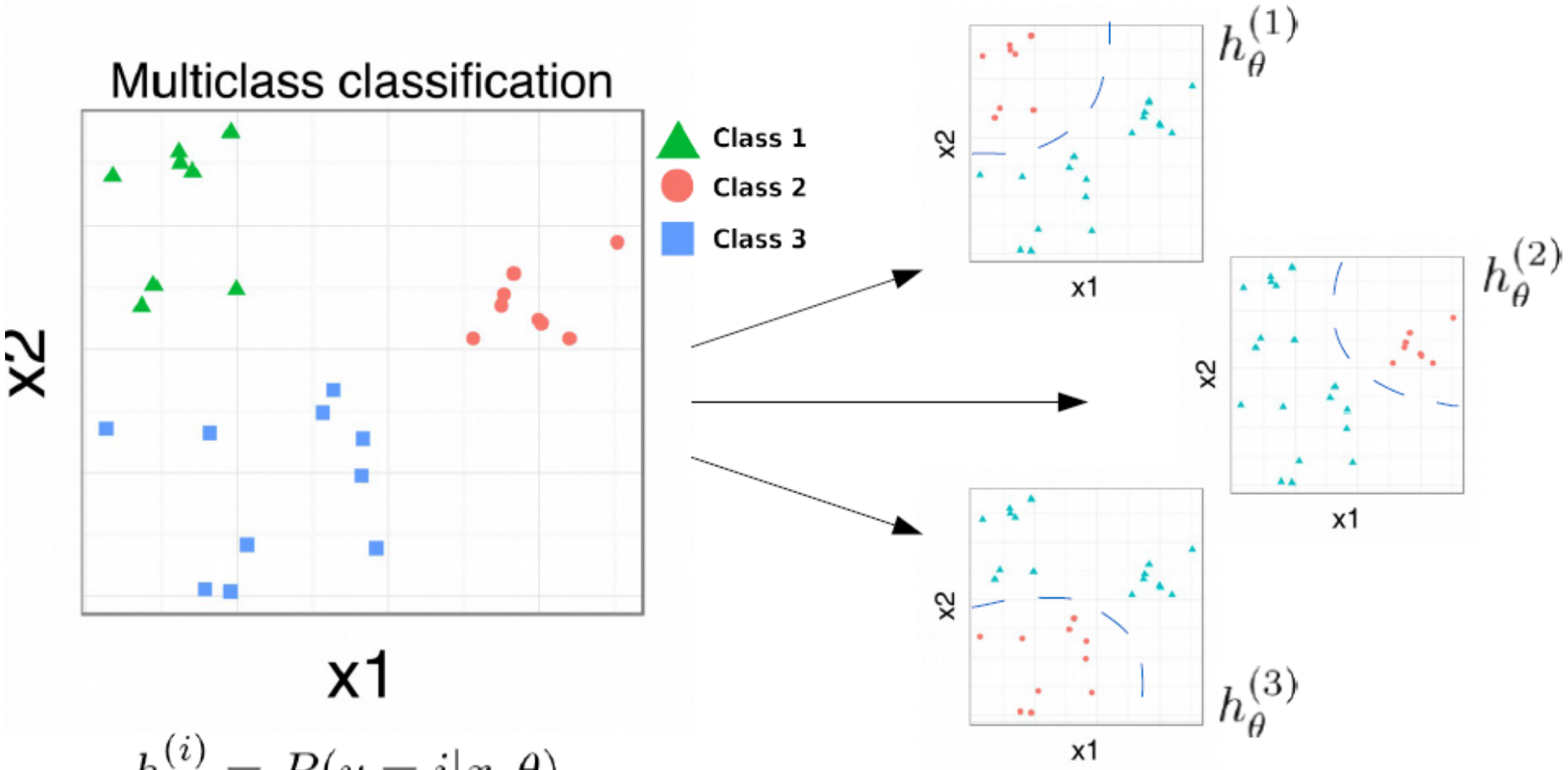
```
while not converged:
    for all j:
        tmp_j := θ_j − α (1/m) Σ_{i=1}^{m} (h_θ(x^(i)) − y^(i)) x_j^(i)
    θ := [tmp_0
          ⋮
          tmp_n]
```

Multiclass Classification


Binary classification
x2
x1


Multiclass classification
x2
x1

Multiclass classification



$$h_\theta^{(i)} = P(y = i|x, \theta)$$

where $i \in \{1, 2, 3\}$

- Train logistic regression classifier $h_\theta^{(i)}(x)$ for each class *i* to predict probability of *y* = *i*
- On new *x* predict class *i* which satisfies

$$\underset{i}{argmax}\ h_\theta^{(i)}(x)$$

In short…Logistic Regression is

Feature Vector → $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$ $w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$ ← Weights

Prediction ←
$$\hat{y} = \frac{1}{1 + e^{-(w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_d \cdot x_d)}}$$

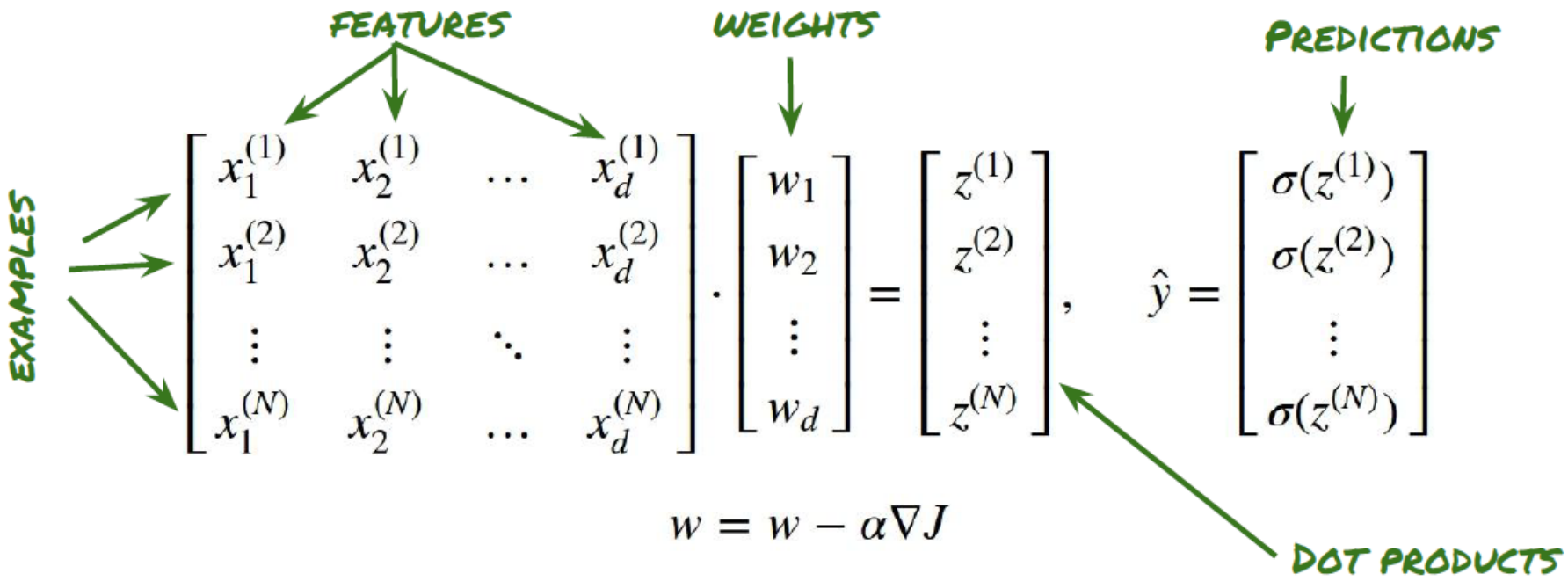
Loss →
$$J = -\frac{1}{N} \sum_{i=1}^{N} y^{(i)} log\hat{y}^{(i)} + (1 - y^{(i)}) log(1 - \hat{y}^{(i)})$$

Weight update →
$$w_k = w_k - \alpha \frac{\partial J}{\partial w_k}$$ ← Derivative of loss

$$w = w - \alpha \nabla J$$ ← Gradient

Markov

Vectorized Logistic Regression



Markov

Vectorized Logistic Regression..

Computing Gradient Descent

$$\nabla J = \frac{1}{N} X^T (\hat{y} - y)$$

$$\nabla J = \frac{1}{N} \cdot \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(N)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ x_d^{(1)} & x_d^{(2)} & \dots & x_d^{(N)} \end{bmatrix} \cdot \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \vdots \\ \hat{y}^{(N)} - y^{(N)} \end{bmatrix}$$

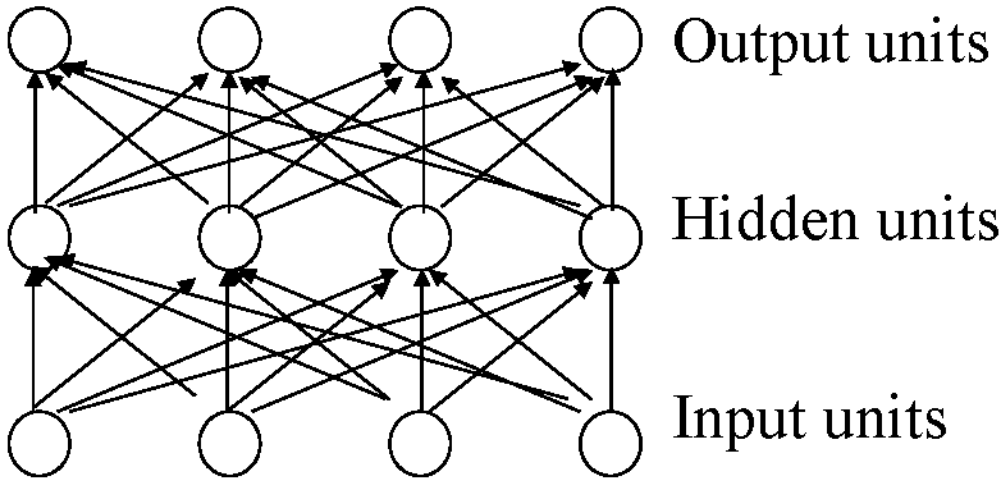
Assignment # 2

- Using Wine data set https://archive.ics.uci.edu/ml/datasets/wine (3 classes of types of wine and the attributes represents the components of each class)
- Use features (alcohol and flavonoids).
- Build a logistic regression model based on above two features and evaluate your model (use L2 regularization).
- Plot the 3 output classes using above features.
- Split data into training and testing data using “sklearn.model_selection import train_test_split”…split it 75%-train and 25%-test.
- Evaluate model using precision ,recall & accuracy. “from sklearn.metrics import accuracy_score, precision_score, recall_score”
- Plot the decision boundaries.

Markov

Neural Network & Deep Learning

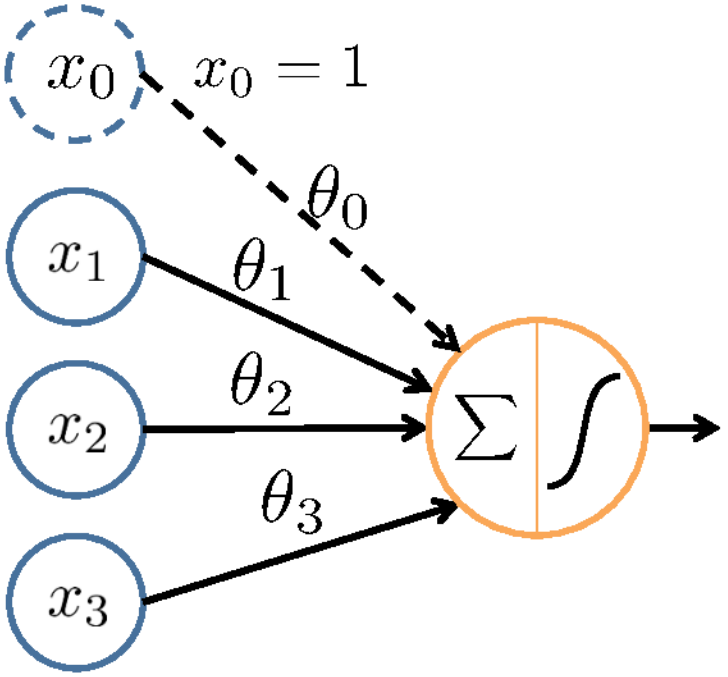
Neural networks



Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Neuron Model: Logistic Unit

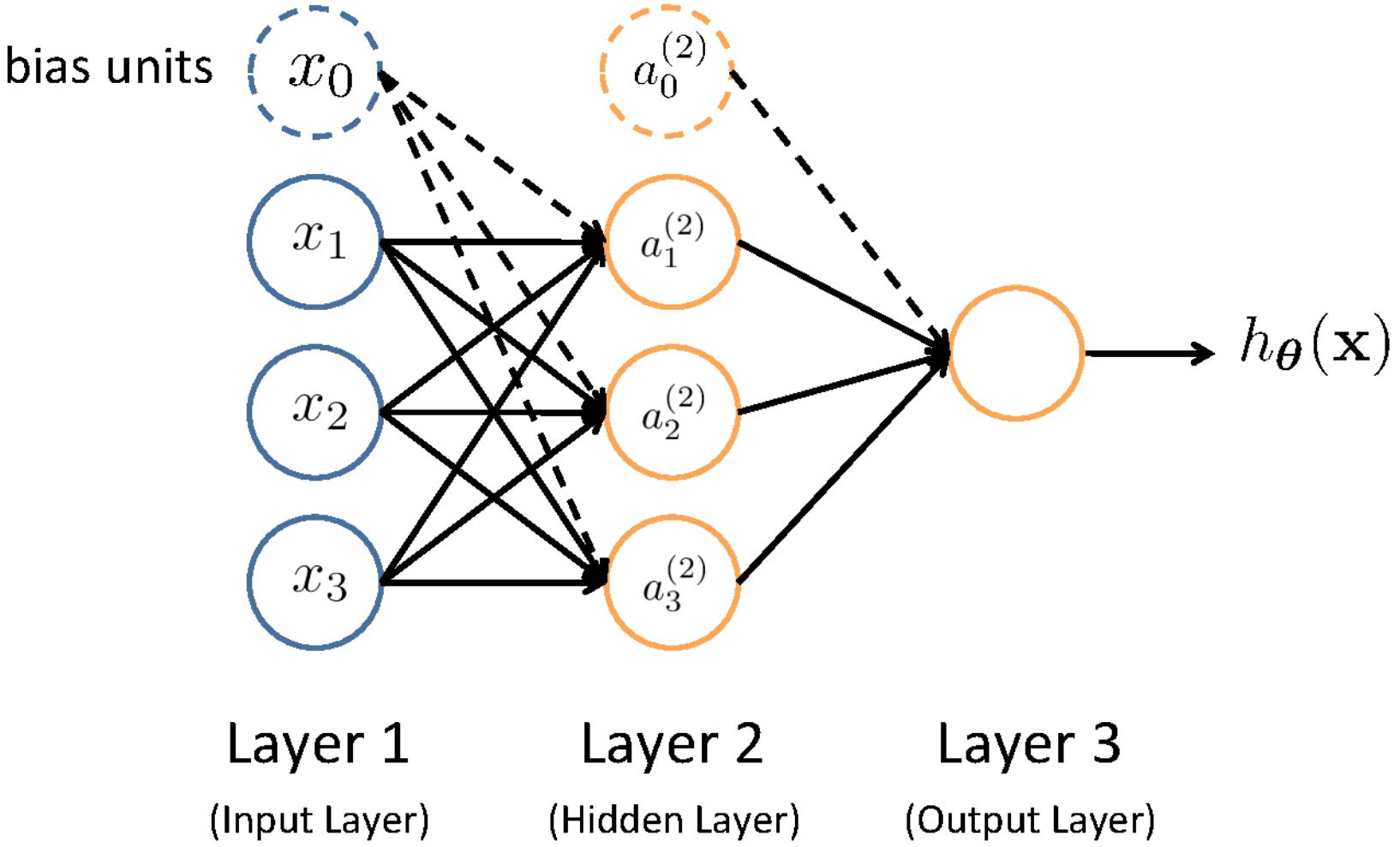


$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

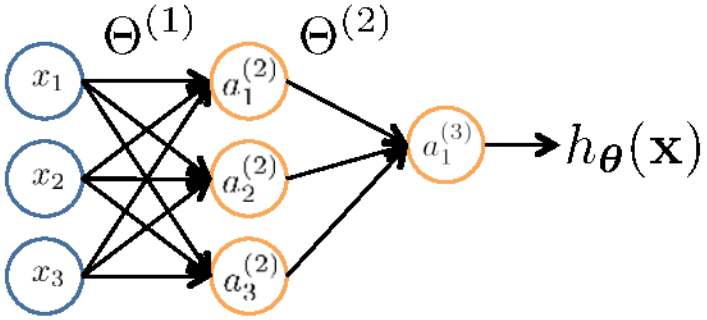
Neural Network



Feed-Forward Process

- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value
  - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the "threshold" of that node

Neural Network



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer j + 1

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

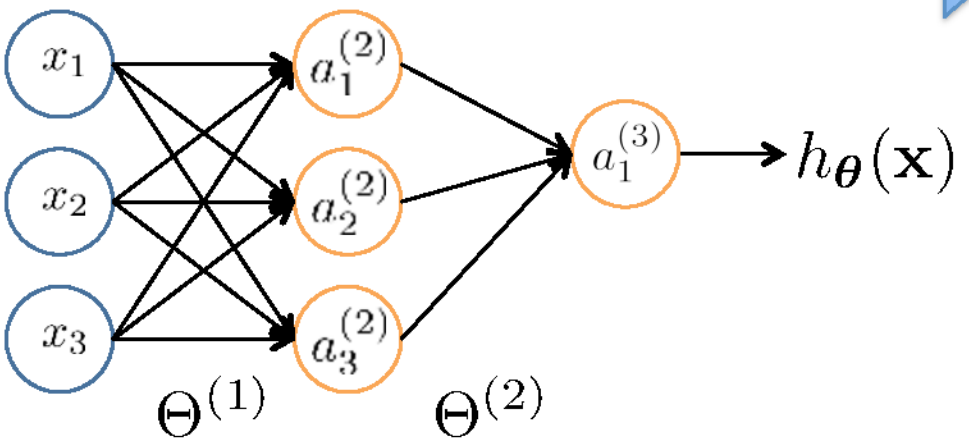
Vectorization

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

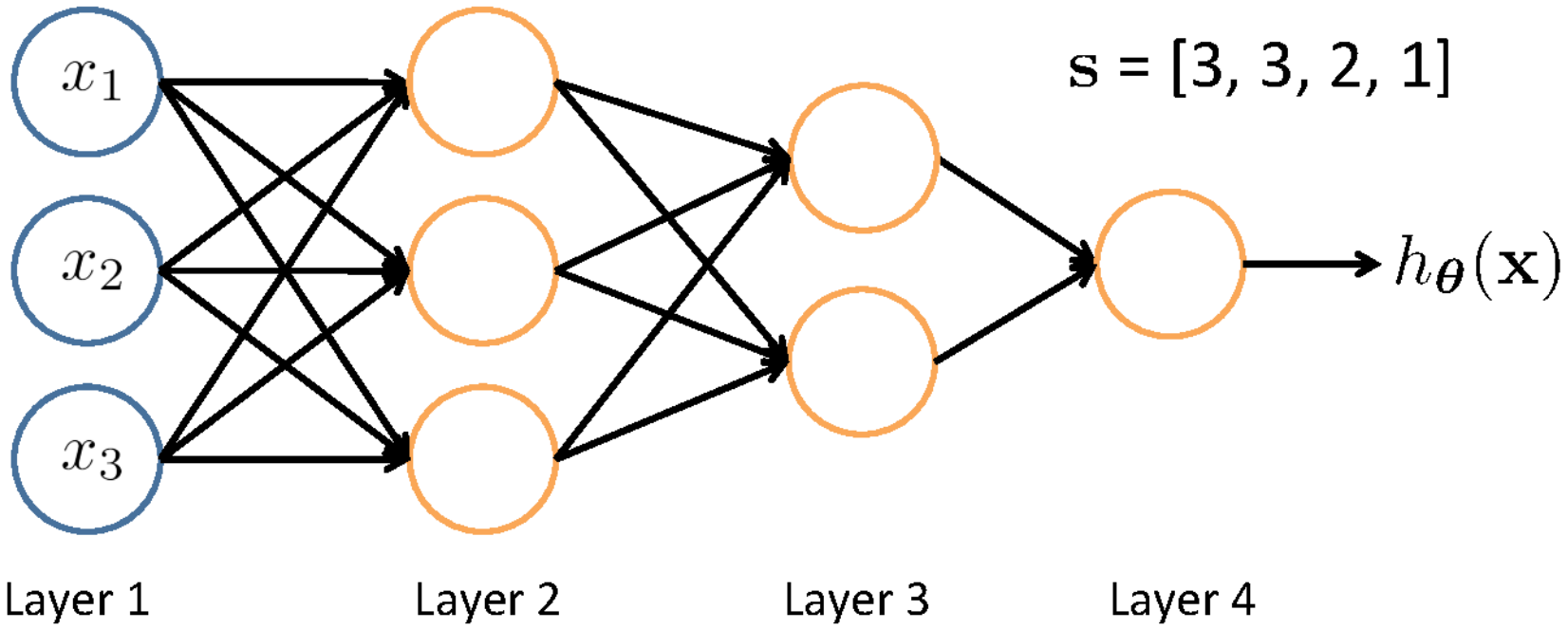
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Other Network Architectures

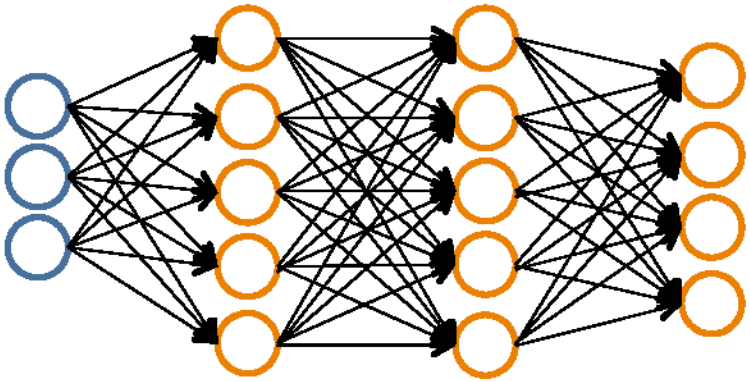


L denotes the number of layers

$\mathbf{s} \in \mathbb{N}^{+L}$ contains the numbers of nodes at each layer

- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L-1} = K$ (# classes)

Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$ contains # nodes at each layer

- $s_0 = d$ (# features)

Binary classification

y = 0 or 1

1 output unit $(s_{L-1} = 1)$

Multi-class classification (K classes)

$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units $(s_{L-1} = K)$

Learning in NN: Backpropagation

- Similar to the perceptron learning algorithm, we cycle through our examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error

- The trick is to assess the blame for the error and divide it among the contributing weights

Cost Function

Logistic Regression:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1-y_i)\log(1-h_{\boldsymbol{\theta}}(\mathbf{x}_i))] + \frac{\lambda}{2n}\sum_{j=1}^{d}\theta_j^2$$

Neural Network:

$$h_\Theta \in \mathbb{R}^K \qquad (h_\Theta(\mathbf{x}))_i = i^{th}\text{output}$$

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\left(1-(h_\Theta(\mathbf{x}_i))_k\right)\right]$$

$$+\frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

k^{th} class: true, predicted
not k^{th} class: true, predicted

Optimizing the Neural Network

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\Big(1-(h_\Theta(\mathbf{x}_i))_k\Big)\right] + \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

Solve via: $\min_\Theta J(\Theta)$

$J(\Theta)$ is not convex, so GD on a neural net yields a local optimum

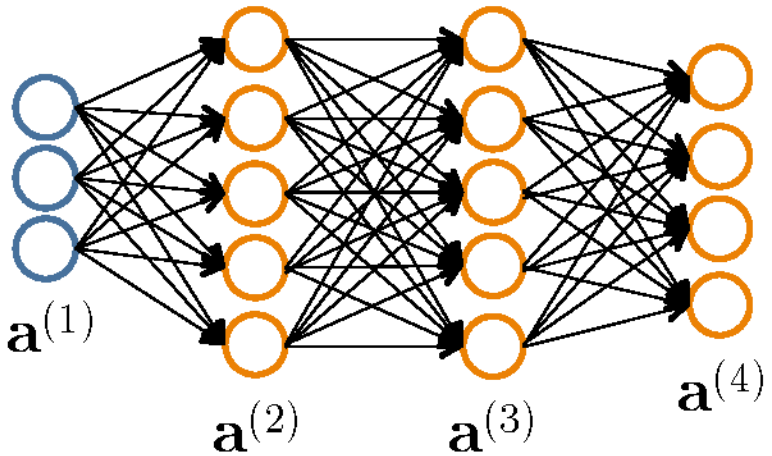
- But, tends to work well in practice

Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Forward Propagation

- Given one labeled training instance $(\mathbf{x}, y)$:



Forward Propagation

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$

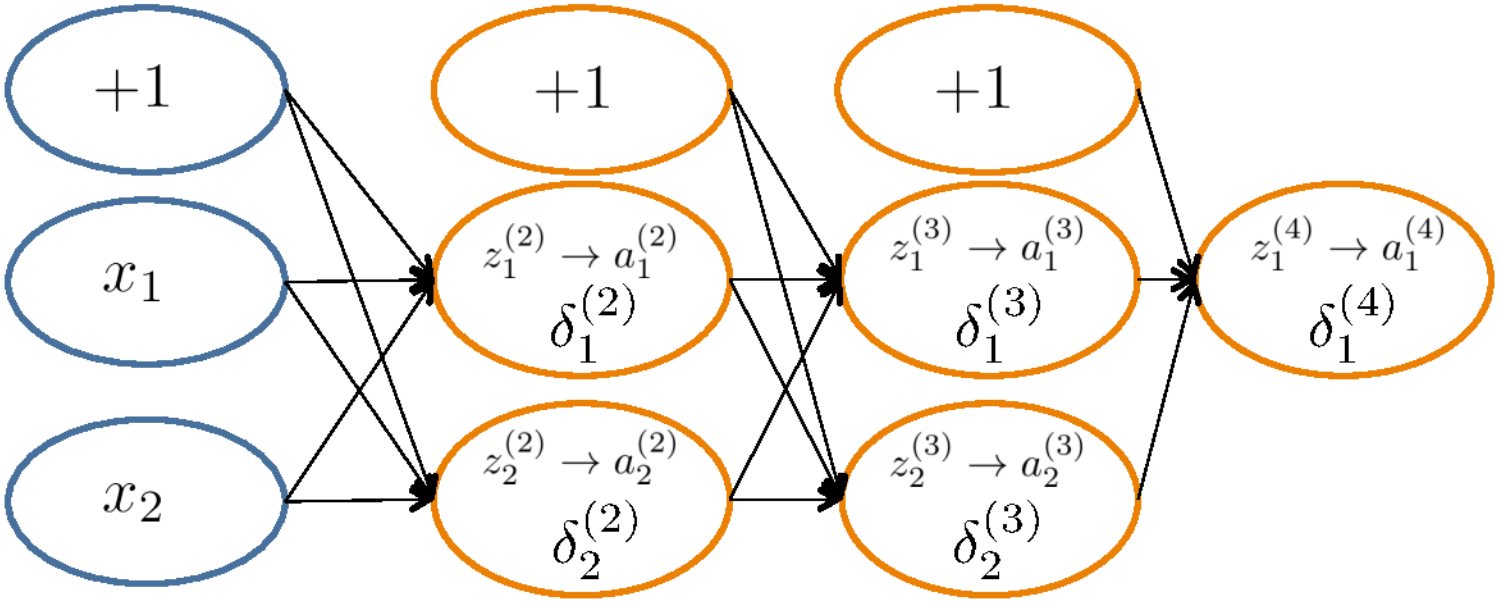
Backpropagation Intuition

- Each hidden node j is "responsible" for some fraction of the error $\delta_j^{(l)}$ in each of the output nodes to which it connects

- $\delta_j^{(l)}$ is divided according to the strength of the connection between hidden node and the output node

- Then, the "blame" is propagated back to provide the error values for the hidden layer

Backpropagation Intuition

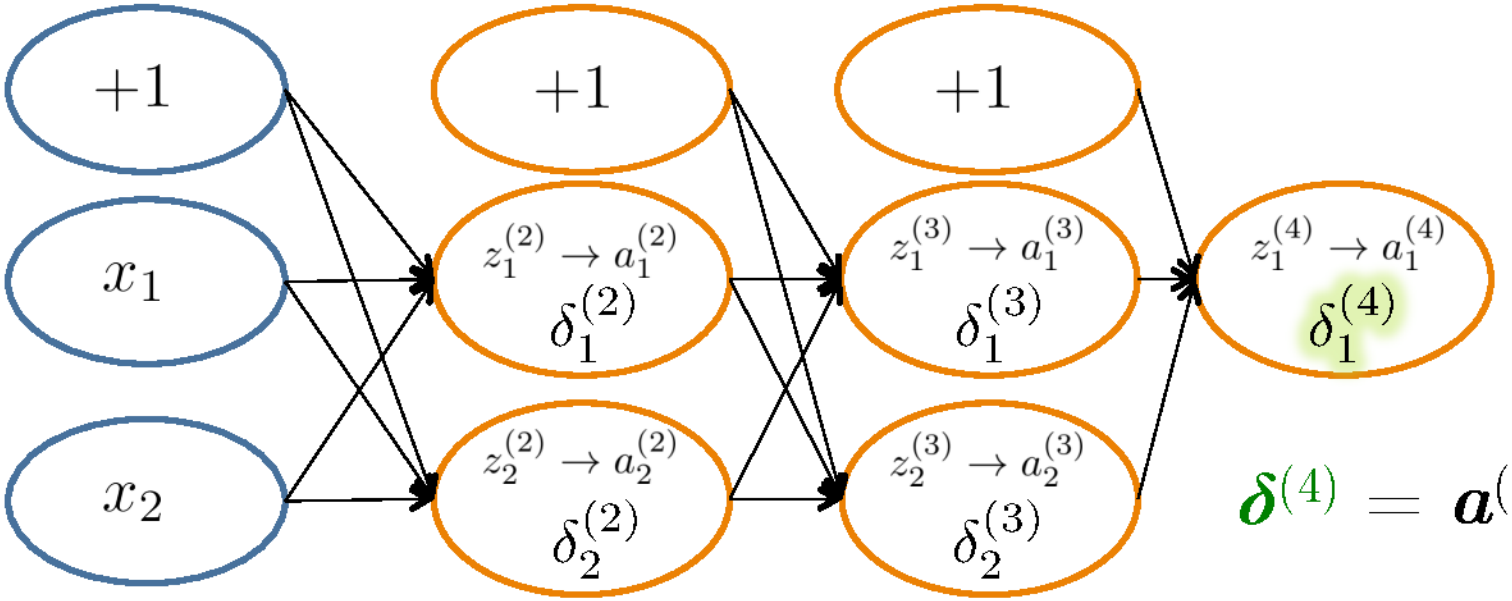


$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -y_i \log h_\Theta(\mathbf{x}_i) - (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



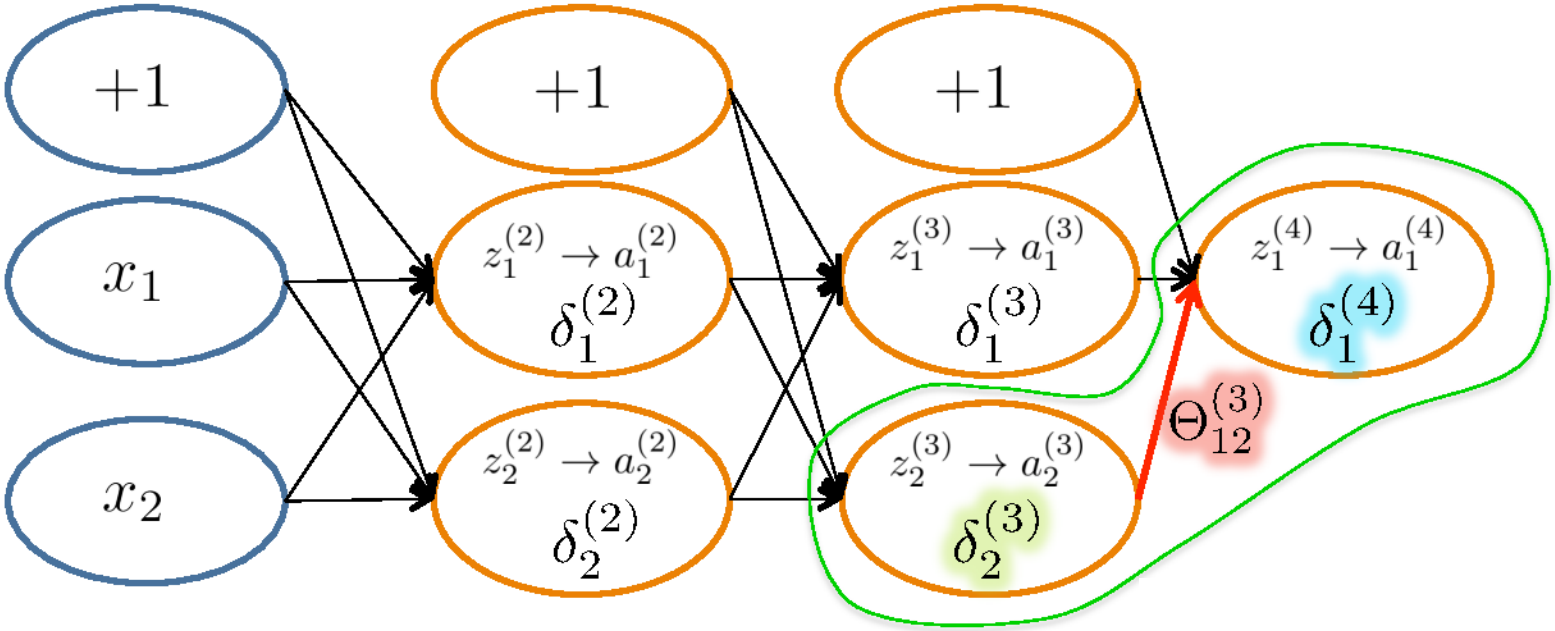
$$\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -y_i \log h_\Theta(\mathbf{x}_i) - (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



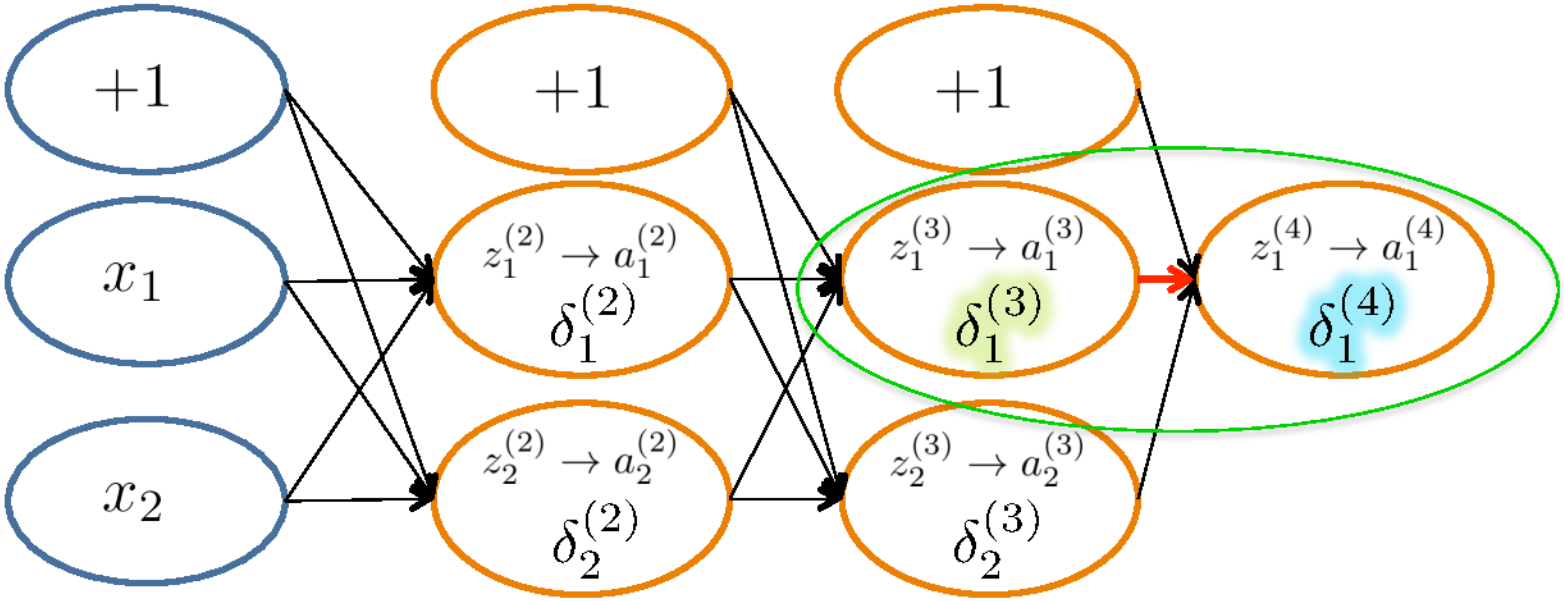
$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)} \times g'(z_2^3)$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -y_i \log h_\Theta(\mathbf{x}_i) - (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)} \times g'(z_1^3)$$

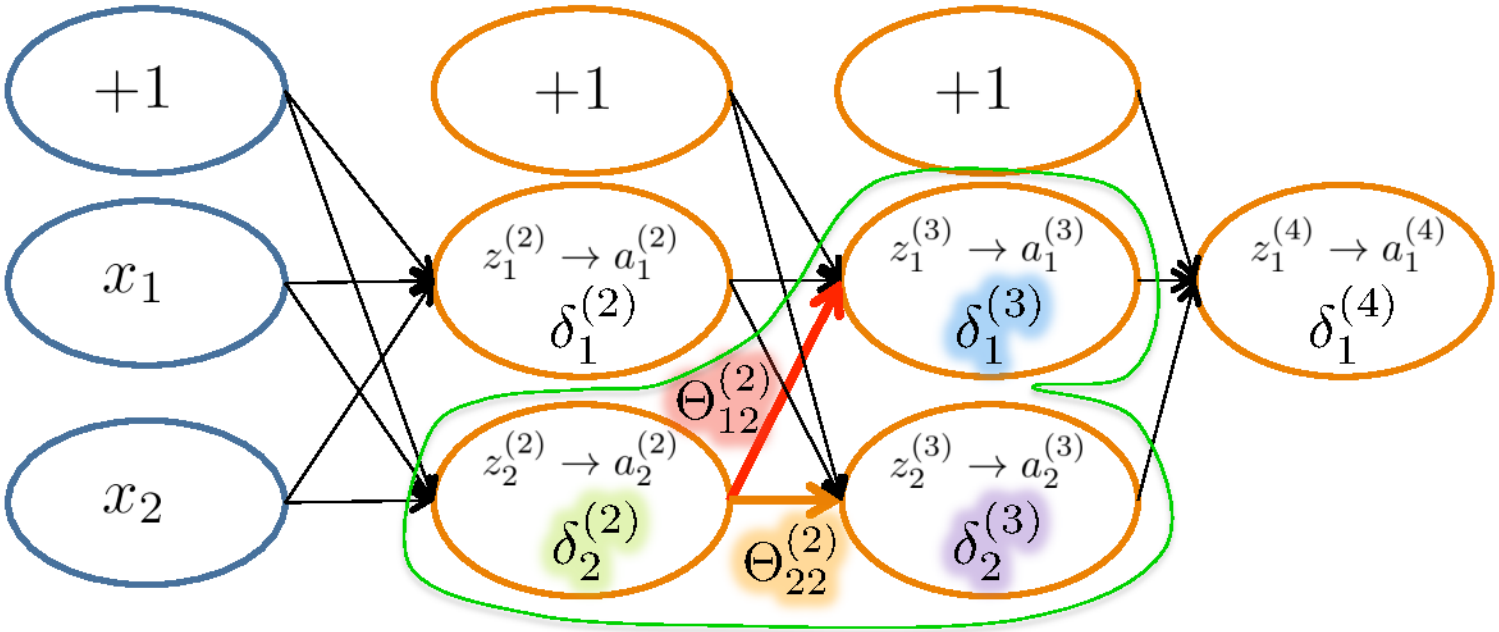
$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)} \times g'(z_2^3)$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -y_i \log h_\Theta(\mathbf{x}_i) - (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$$\delta_2^{(2)} = \Theta_{12}^{(2)} \times \delta_1^{(3)} + \Theta_{22}^{(2)} \times \delta_2^{(3)} \times g'(z_2^2)$$

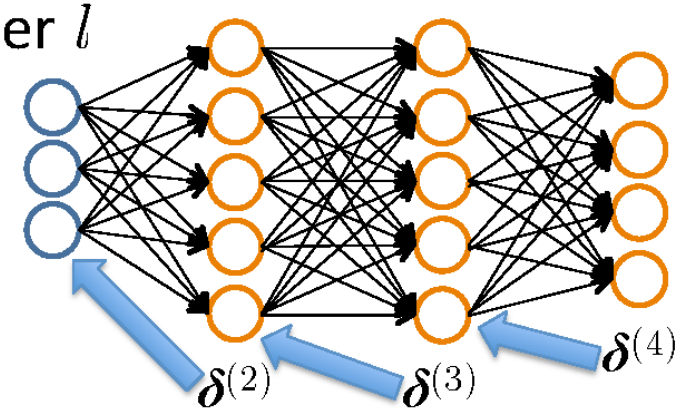
$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -y_i \log h_\Theta(\mathbf{x}_i) - (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation: Gradient Computation

Let $\delta_j^{(l)}$ = "error" of node j in layer l



(#layers $L = 4$)

Element-wise product .*

Backpropagation

- $\delta^{(4)} = a^{(4)} - \mathbf{y}$
- $\delta^{(3)} = (\Theta^{(3)})^{\mathsf{T}}\delta^{(4)}$.* $g'(\mathbf{z}^{(3)})$
- $\delta^{(2)} = (\Theta^{(2)})^{\mathsf{T}}\delta^{(3)}$.* $g'(\mathbf{z}^{(2)})$
- (No $\delta^{(1)}$)

$g'(\mathbf{z}^{(3)}) = \mathbf{a}^{(3)}$.* $(1-\mathbf{a}^{(3)})$

$g'(\mathbf{z}^{(2)}) = \mathbf{a}^{(2)}$.* $(1-\mathbf{a}^{(2)})$

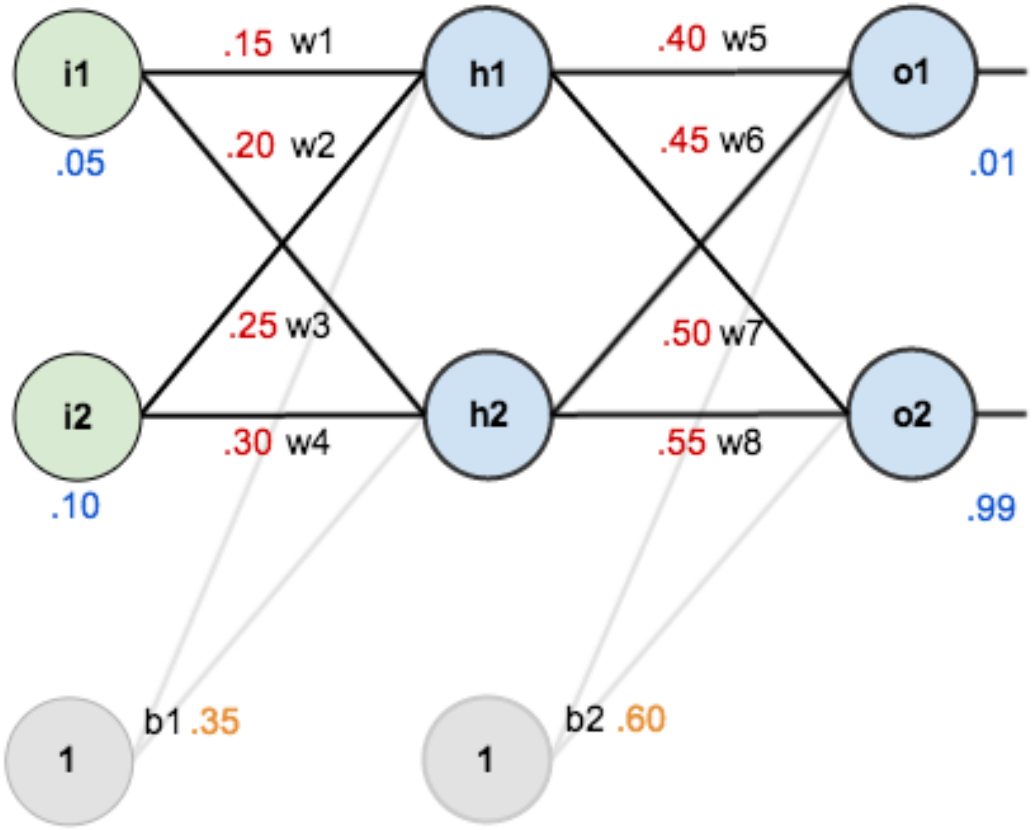
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

(ignoring λ; if $\lambda = 0$)

$$\frac{\partial}{\partial\, \mathrm{b}_{ij}^{(l)}} J(\Theta) = \quad \delta_i^{(l+1)}$$

Markov

Assignment # 3



- Please update the weights for one backward path.
- Weights given are the initial weights, the i/p & o/p in figure the true values.
- Use Sigmoid activation function and learning rate=0.5.
- Try it by hands and on python to check your results.

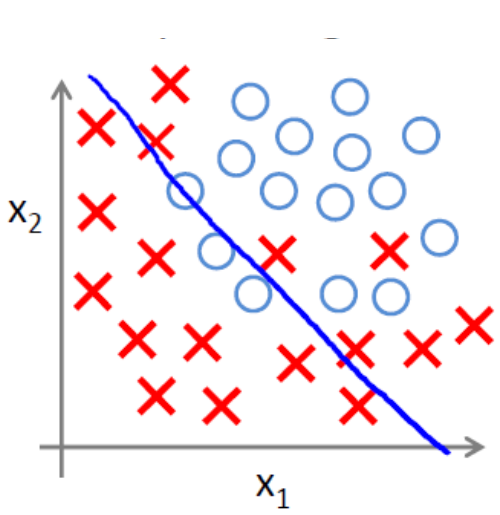
Markov

Train/development/test data sets

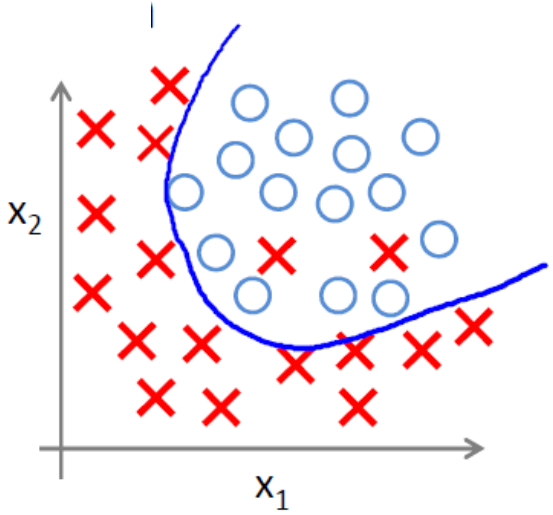
Training set	Development set	Test set

- Development set used to test the model and adjust hyper parameters.
- Checks the best model to be used
- Hyper parameters like: number of layers, nodes, learning rates….etc
- We can have 60/20/20%.
- Sometimes we don’t have a development set, which is no problem.

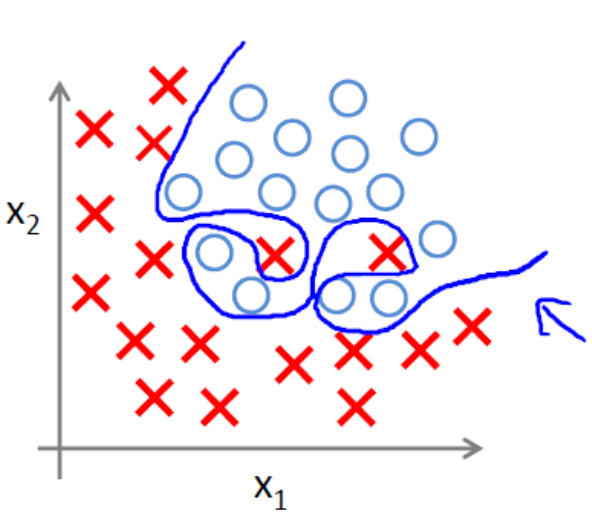
Bias/Variance



High Bias
Underfitting



Good model



High Variance
Overfitting

Error	Results			
Train set error	1%	15%	15%	0.5%
Dev. set error	11%	16%	30%	1%
	HV	HB	HB&HV	LB&LV

HB- Bigger network with more layers and train for longer time.
HV- More data in training set and use regularization (L2)

Regularization

L2 regularization

- Cost function $$J(\Theta) = J_i + \lambda \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_l+1} (\Theta_{ji}^{(l)})^2$$

- Derivative $$\frac{\partial J}{\partial \Theta_{ji}^{(l)}} = \frac{\partial J_i}{\partial \Theta_{ji}^{(l)}} + \lambda \Theta_{ji}^{(l)}$$

$$\text{When } \frac{\partial J}{\partial \Theta_{ji}^{(l)}} = 0, \quad \Theta_{ji}^{(l)} = -\frac{1}{\lambda} \frac{\partial J_i}{\partial \Theta_{ji}^{(l)}}$$

- Keep the weights small unless they have big derivatives
- Tends to prefer many small weights

L1 regularization

- Cost function $J(\Theta) = J_i + \lambda \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} |\Theta_{ji}^{(l)}|$

- L1 regularization has the effect that many weights are set to zero (or close to zero).
- The effect of setting many weights to zero and keeping a few large weights is feature extraction – select only some of the input connections.
- For deep learning, this often does not work as well as L2-regularization.

Markov

Regularization by early stopping

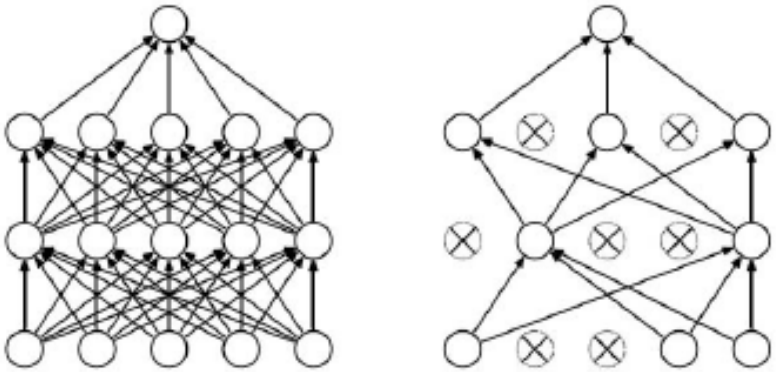
- Another kind of regularization is early stopping: stopping before the model can overfit the training data
- Remember that we initialize the weights to small random numbers.
- As training progresses (without batch normalization), the weights can grow. Too large weights often leads to overfitting.
- We can monitor the training and validation accuracy, and stop when the validation accuracy increases systematically over several steps.

Regularization by data augmentation

- Given a finite data set, we can make the net generalize better by adding noise to the data.
- For image data it is common to simulate larger data sets by affine transforms to
  - Shift
  - Rotate
  - Scale
  - Flip

Dropout Regularization

- Presented in http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf
- Achieves a similar effect as bagging by randomly setting the output of a node to zero (by multipying with a random vector of zeros with probability p).



(a) Standard Neural Net (b) After applying dropout.

Example: cat class with nodes detecting
- Eyes
- Ears
- ~~Tail~~
- Fur
- Legs
- ~~Mouth~~

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Dropout - training

- Choose a dropout probability p
- We can drop both inputs and nodes in hidden layers.
- Create a binary mask for all nodes with probability of zero=p.
- Consider a 3-layer network with dropout in the hidden layers

- Backpropagate as usual, but take into account the drop.

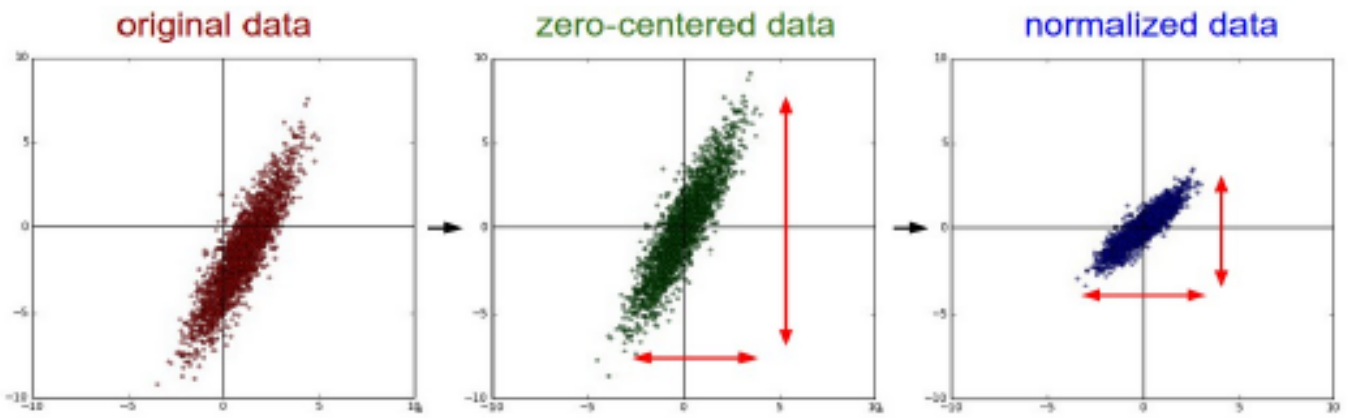
Dropout – predict : naive implementation

- A drop rate of p will scale the outputs during training with a factor $p<1$.
- When we predict new data, without considering this scaling, the outputs will be larger.
- We have to scale the outputs during predict by p:

- Since test-time performance is critical, we normally apply «inverted dropout» and scale at training time.

Normalize Input

- Standardize data to zero mean and unit variance
- For each feature m, compute the mean μ and standdard deviation σ over the training data set and let $x_m = (x_m - \mu)/ \sigma^2$
- Remark: STORE μ and σ^2 because new data/test data must have the same normalization.

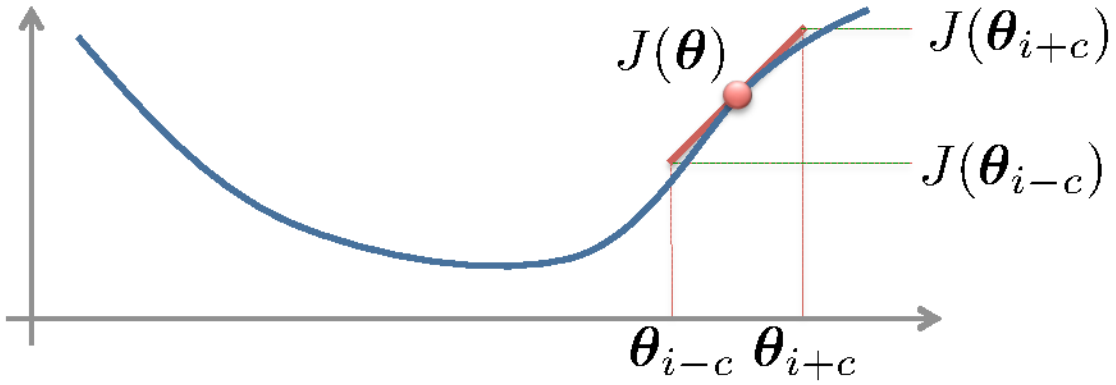


Vanishing & exploding gradient

- Gradient can be very small or large.
- To solve this we use a partial solution that might help, i.e. by proper weight initialization.
- Using variance of 1/n and for Relu activation function we use 2/n.
- Other variance that can be used for tanh activation is $\sqrt{1/n}$

Gradient Checking

Idea: estimate gradient numerically to verify implementation, then turn off gradient checking



$$\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) \approx \frac{J(\boldsymbol{\theta}_{i+c}) - J(\boldsymbol{\theta}_{i-c})}{2c}$$

$$c \approx 1\text{E-}4$$

$$\boldsymbol{\theta}_{i+c} = [\theta_1, \theta_2, \ldots, \theta_{i-1}, \theta_i + c, \theta_{i+1}, \ldots]$$

Change ONLY the i^{th} entry in $\boldsymbol{\theta}$, increasing (or decreasing) it by c

Gradient Checking

$\boldsymbol{\theta} \in \mathbb{R}^m$ $\boldsymbol{\theta}$ is an "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \ldots$

$\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \ldots, \theta_m]$

Put in vector called `gradApprox`

$$\frac{\partial}{\partial \theta_1} J(\boldsymbol{\theta}) \approx \frac{J([\theta_1 + c, \theta_2, \theta_3, \ldots, \theta_m]) - J([\theta_1 - c, \theta_2, \theta_3, \ldots, \theta_m])}{2c}$$

$$\frac{\partial}{\partial \theta_2} J(\boldsymbol{\theta}) \approx \frac{J([\theta_1, \theta_2 + c, \theta_3, \ldots, \theta_m]) - J([\theta_1, \theta_2 - c, \theta_3, \ldots, \theta_m])}{2c}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_m} J(\boldsymbol{\theta}) \approx \frac{J([\theta_1, \theta_2, \theta_3, \ldots, \theta_m + c]) - J([\theta_1, \theta_2, \theta_3, \ldots, \theta_m - c])}{2c}$$

Check that the approximate numerical gradient matches the entries in the D matrices

Batch and Mini Batch

- Batch gradient descent computes the loss summed over ALL training samples before doing gradient descent update.

$$\Theta^{(l)} = \Theta - \eta D^{(l)}$$

- This is slow if the training data set is large.

Repetition: Mini batch gradient descent

- Select randomly a small batch, update, then repeat:
- If batch_size=1, this is called online learning, and sometimes Stochastic Gradient Descent (SGD)
  - *But the term SGD sometimes also means mini batch gradient descent*
- Common parameter value: 32, 64, 128.

Learning rate decay

- Reduce learning rate with time.
- Slowly reduce learning rate as training converges.
- If we have a mini batch for example, we can have:

$$\alpha = \frac{1}{1 + decay\ rate + epoch\ num.} \times \alpha_0$$

α_0=0.2, decay rate=1

There are lots of learning rate decay equations

Gradient Descent with Momentum

- There's an algorithm called momentum, or gradient descent with momentum that almost always works faster than the standard gradient descent algorithm.
- In one sentence, the basic idea is to compute an exponentially weighted average of your gradients, and then use that gradient to update your weights instead.

$v_{dB} = 0 \quad v_{d\theta} = 0$

$v_{d\theta} = \beta v_{d\theta} + \mathrm{d}\theta$

$v_{db} = \beta v_{db} + \mathrm{d}b$

$\theta = \theta - \alpha v_{d\theta}$ b= $b - \alpha v_{db}$

β=0.9 (commonly used value)

Neural Network on Keras

```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import pandas as pd
from keras import optimizers
filename = ‘test1.csv‘ (9 columns file, 9th column is the 0,1 label)
df = pd.read_csv(filename)
X = df.iloc[:,0:8]
Y = df.iloc[:,8]
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, init='uniform', activation='relu'))
model.add(Dense(8, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
# Compile model
sgd = optimizers.SGD(lr=0.01)
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=150, batch_size=10,  verbose=2)
# calculate predictions
predictions = model.predict(X)
# round predictions (training set)
rounded = [round(x[0]) for x in predictions]
print(rounded)
print(model.get_weights())
print(model.summary())
#Test sample Prediction
Xnew = np.array([[1,126,60,0,0,30.1,0.349,47]])
ynew = model.predict_classes(Xnew)
```

Training neural nets - summary

Elements to consider:

- Network architecture/layers/nodes
- Activation functions
- Loss function
- Data preprocessing
- Weight initialization
- Mini-batch size
- Dropout and other types of regularization
- Mini-batch gradient descent update schemes
- Training, validation, and test sets
- Searching for the best parameters
- Monitoring the learning process