



CSE 4621

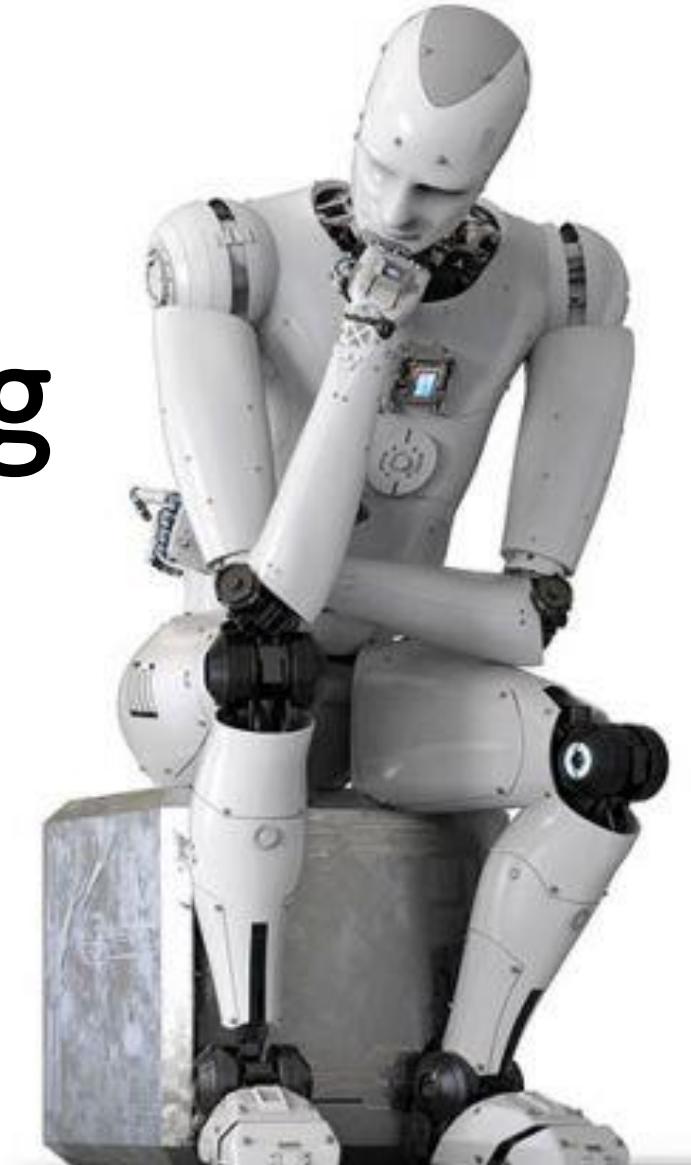
Machine Learning

Lecture 0

Md. Hasanul Kabir, PhD.

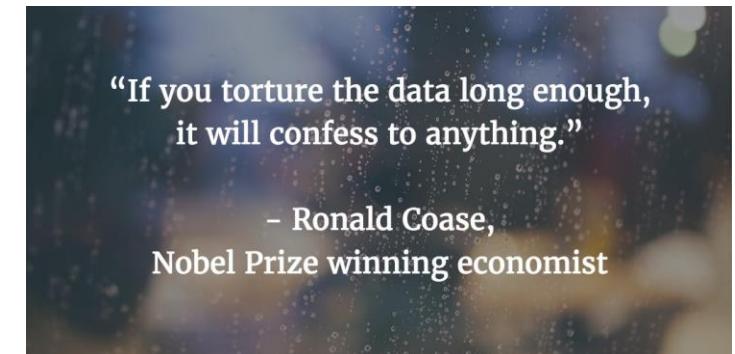
Professor, CSE Department

Email: hasanul@iut-dhaka.edu



Why Machine Learning?

- This is the age of “Big Data”
 - Due to computerization and development of powerful data collection & storage tools.
- We all are producer/generator of data.
 - Purchase, clicks, social media, blogs, and many more
- We also the consumer of data!
 - Products and services specialized to user.
 - Dependent on person, time, location, etc.
- There are patterns in data.
- Who will do it? Computers!



Why Machine Learning?

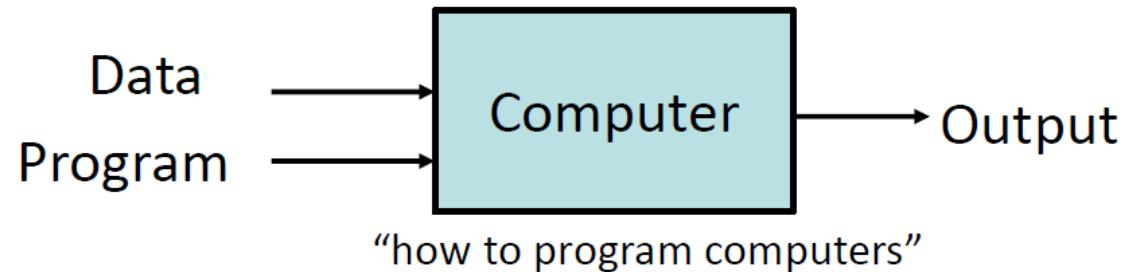
- ❑ Computer/Computing science aims to develop **automated** machinery (i.e., programs) to accomplish non-trivial tasks. This is known as **programming**.

- ❑ A large number of tasks cannot be programmed explicitly by humans
 - E.g., spam detection, hand written digit recognition
 - Such programs **do exist**, but humans cannot program explicitly.
 - We may not be able to identify the process completely.
 - Some machinery (e.g., a machine learning model) is able to yield a program that almost accomplishes the task.

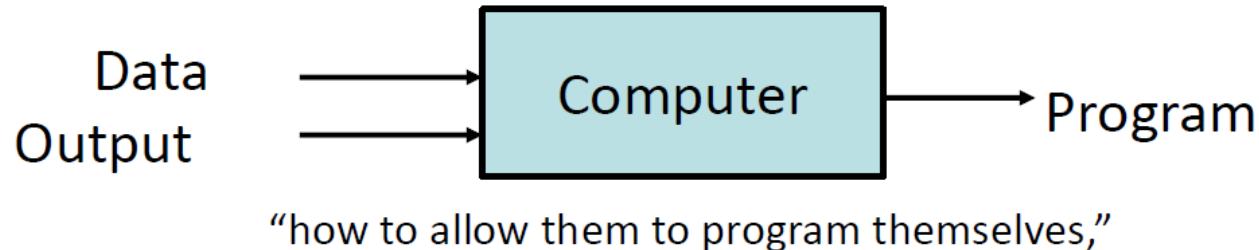


Conventional Programming vs. Machine Learning

– Conventional Programming



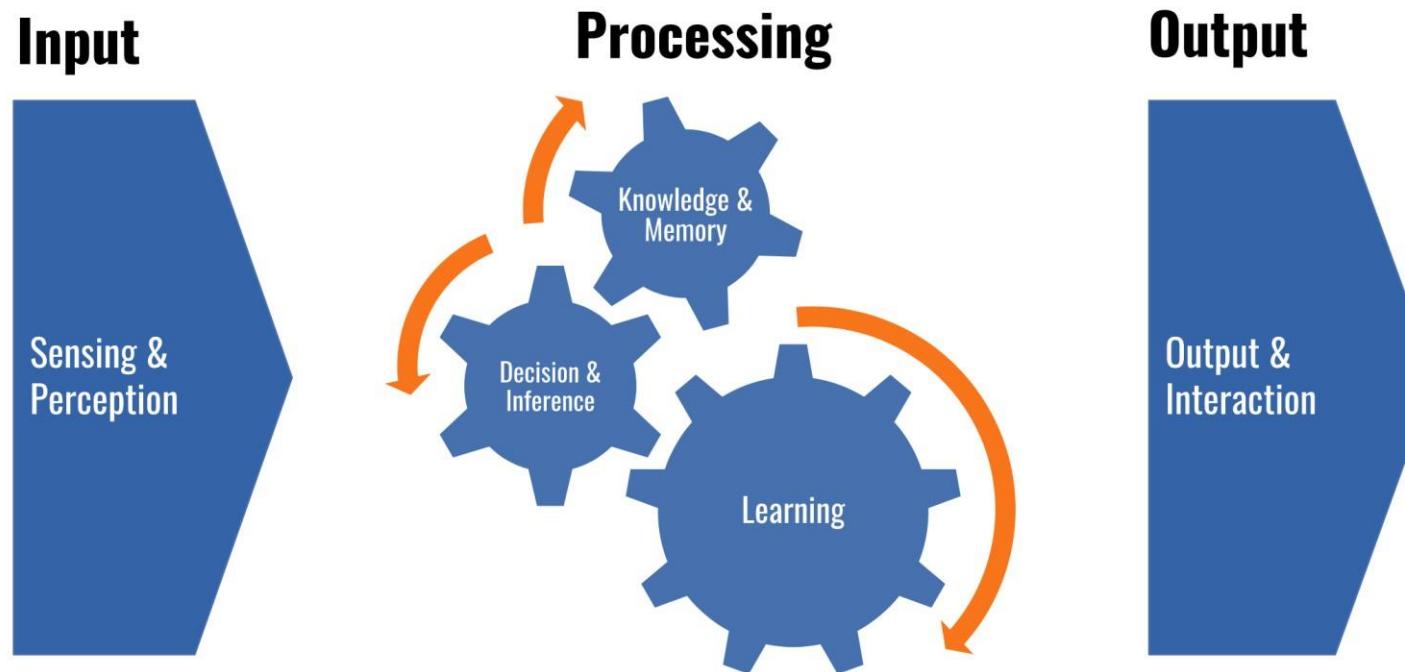
– Machine Learning



– Example: Sentiment Analysis

Human Learning vs. Machine Learning

Human Intelligence



Human learning

Data = history → Model = brain
New Data = Snapshot of Today's world → Prediction = Tomorrow's stock price

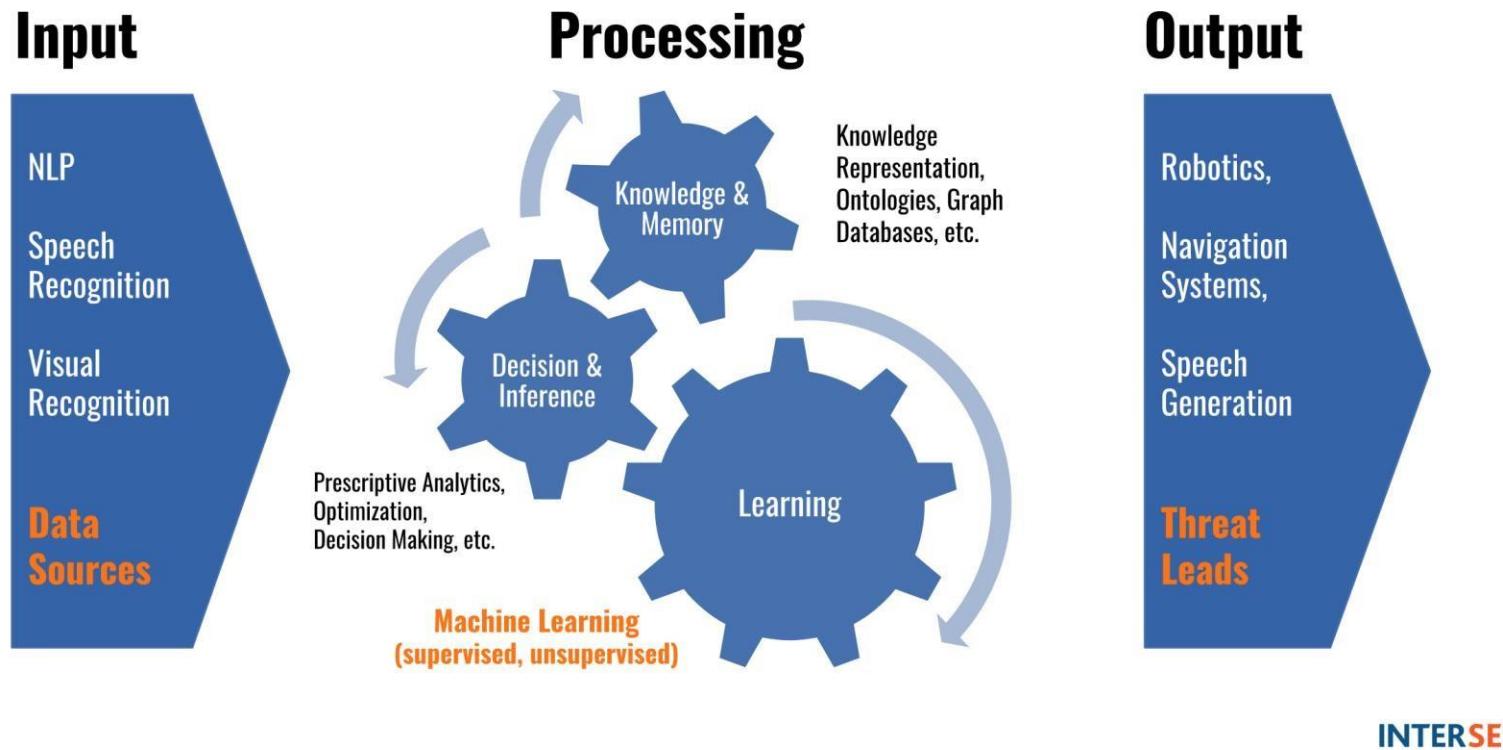
Machine Learning

Machine learning

Data → Algorithm → Model
Possibly new Data → Prediction

Machine Learning is AI?

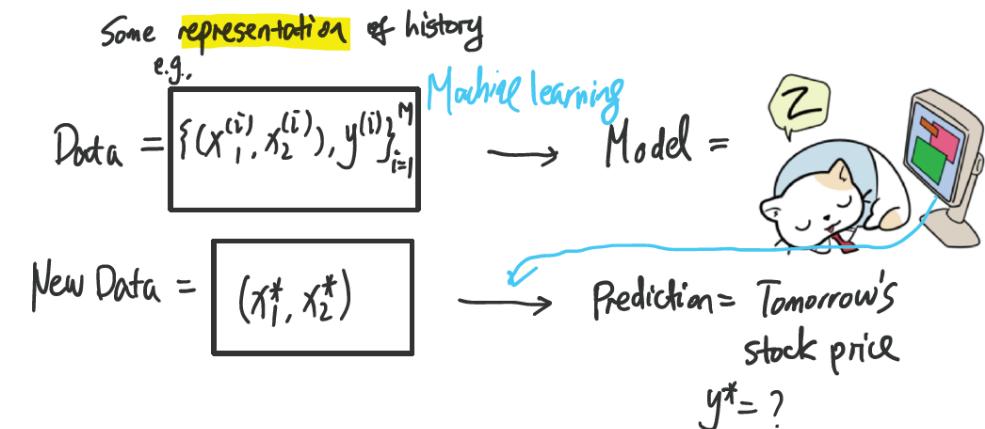
Artificial Intelligence



Source: <https://interset.com/2018/02/20/ai-101-part-1-machine-learning/>

What is Machine Learning (ML)

- Machine learning is programming computers to optimize a performance criterion using example data or past experience. (E. Alpaydin, 2014)
- We have a **model** defined up to some parameters, and learning is the **execution of a computer program to optimize the parameters** of the model using the training data or past experience.
- The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both



More ML Definitions

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Example

“A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.”

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T in this setting?

- Classifying emails as spam or not spam.
- Watching you label emails as spam or not spam.
- The number (or fraction) of emails correctly classified as spam/not spam.
- None of the above—this is not a machine learning problem.

Some Applications of Machine Learning

- **Learning to recognize spoken words.**
 - All of the most successful speech recognition systems employ machine learning in some form that are effective for automatically customizing to individual speakers, vocabularies, microphone characteristics, background noise, etc. Similar techniques have potential applications in many signal-interpretation problems.
- **Learning to drive an autonomous vehicle.**
 - Machine learning methods have been used to train computer-controlled vehicles to steer correctly when driving on a variety of road types.

Some Applications of Machine Learning

- **Learning to classify new astronomical structures.**
 - Machine learning methods have been applied to a variety of large databases to learn general regularities implicit in the data. For example, decision tree learning algorithms have been used by NASA to learn how to classify celestial objects
- **Learning to play world-class backgammon or chess.**
 - The most successful computer programs for playing games such as backgammon are based on machine learning algorithms. Learned its strategy by playing over one million practice games against itself. It now plays at a level competitive with the human world champion.

Course Content (as per Syllabus)

Introduction: Defining machine learning, Scalability, Privacy issues and social impact, Applications in AI, Computer vision, Computer games, Search engines, Marketing, Bioinformatics, Robotics, HCI and Graphics. Graphical models: Introduction to discrete probability, Inference in Bayesian networks, Maximum likelihood and Bayesian learning Model selection.

Supervised learning: Introduction to continuous probability, Linear regression and classification (least squares and ridge), Model assessment and cross-validation, Introduction to optimization, Nonlinear regression (neural nets and Gaussian processes), Boosting and feature selection.

Unsupervised learning: Nearest neighbors and K-means, Spectral kernel methods for clustering and semi-supervised learning. The EM algorithm, Mixture models for discrete and continuous data, Temporal methods: hidden Markov models & Kalman filters, Boltzmann machines and random fields, Examples: web mining, collaborative filtering, music and image clustering, automatic, translation, spam filtering, computer games and object recognition.

Neural Network: Fundamentals of Neural Networks, Back-propagation and related training algorithms, Hebbian learning, Cohen-Grossberg learning, The BAM and the Hopfield Memory, Simulated Annealing, Different type of Neural Networks: Counter-propagation, Probabilistic, Radial Basis Function, Generalized Regression, etc., Adaptive Resonance Theory, Dynamic Systems and Neural Control, The Boltzmann Machine, Self-organizing maps, Spatiotemporal Pattern Classification, The Neocognition, Practical aspects of Neural Networks.

Other forms of learning: Semi-supervised learning, Active learning, Reinforcement learning, Self-taught learning, Evolutionary learning: Genetic algorithm, Genetic programming, CGA.

Course Objectives & Outcomes

- Master the basic techniques on machine learning techniques, including supervised learning, unsupervised learning, and reinforcement learning.
- Apply and implement algorithms to enable machine learning.
- Analyse both strengths and weakness of the machine learning algorithms.
- Design and develop solutions/algorithms for small to medium scale problems.

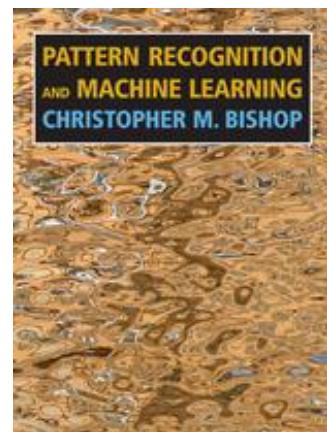
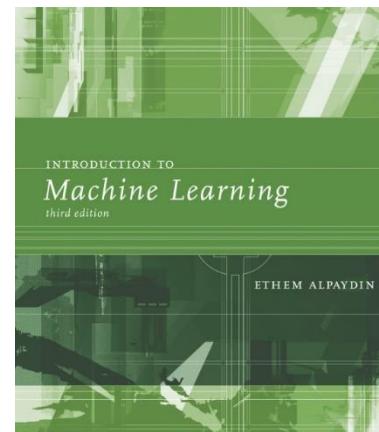
Course Outcomes (BAETE)

- (CO1) Apply algorithms to enable machine learning.
- (CO2) Analyse both strengths and weakness of the machine learning algorithms.
- (CO3) Design solutions/algorithms for small to medium scale problems.

Reading Materials

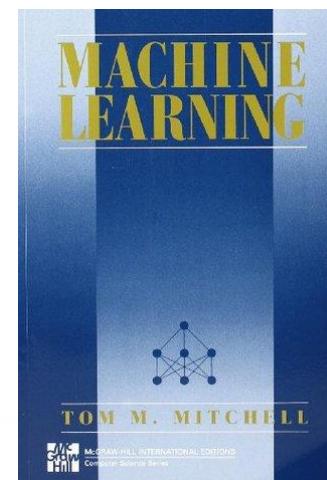
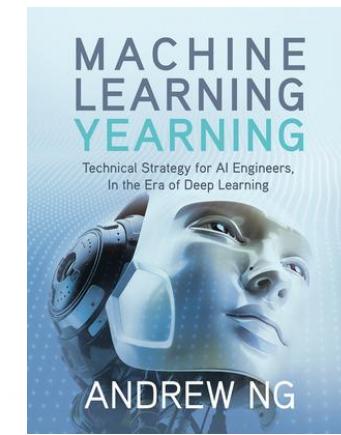
Text Book:

- Introduction to Machine Learning, (3rd/2nd Ed.), MIT Press, 2014
 - E. Alpaydin
- Pattern Recognition and Machine Learning (1st Ed.) Berlin: Springer-Verlag, 2006
 - C. Bishop
- Machine Learning Yearning (Online Version), 2018
 - Andrew Ng



Reference books:

- Machine Learning (1st Ed.), McGraw Hill, 1997
 - T. Mitchell
- Pattern Classification (2nd Edition), Wiley & Sons, 2001.
 - Richard O. Duda, Peter E. Hart & David G. Stork



Online (Coursera) Course:

- Machine Learning - Andrew Ng. at <http://ml-class.org>



Machine Learning

Course Evaluation

As per guidelines provided by CSE Dept., IUT

- Class Tests
- Assignments
- Examinations

Course Material & Announcement:

- Google Classroom Code: **wk3xbql**



CSE 4621

Machine Learning

Lecture 1

Md. Hasanul Kabir, PhD.

Professor, CSE Department

Email: hasanul@iut-dhaka.edu



Introduction

- “Data is abundant and cheap but knowledge is scarce and expensive.”
- **Machine:** Computer / Computer Program
- **Learning:** *ability to learn without being explicitly programmed.*
- Tom Mitchell (1998). Well-posed Learning Problem:
 - *A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E .*

Examples

□ A checkers learning problem:

- **Task T:** playing checkers
- **Performance measure P:** percent of games won against opponents
- **Training experience E:** playing practice games against itself

□ A handwriting recognition learning problem:

- **Task T:** recognizing and classifying handwritten words within images
- **Performance measure P:** percent of words correctly classified
- **Training experience E:** a database of handwritten words with given classifications

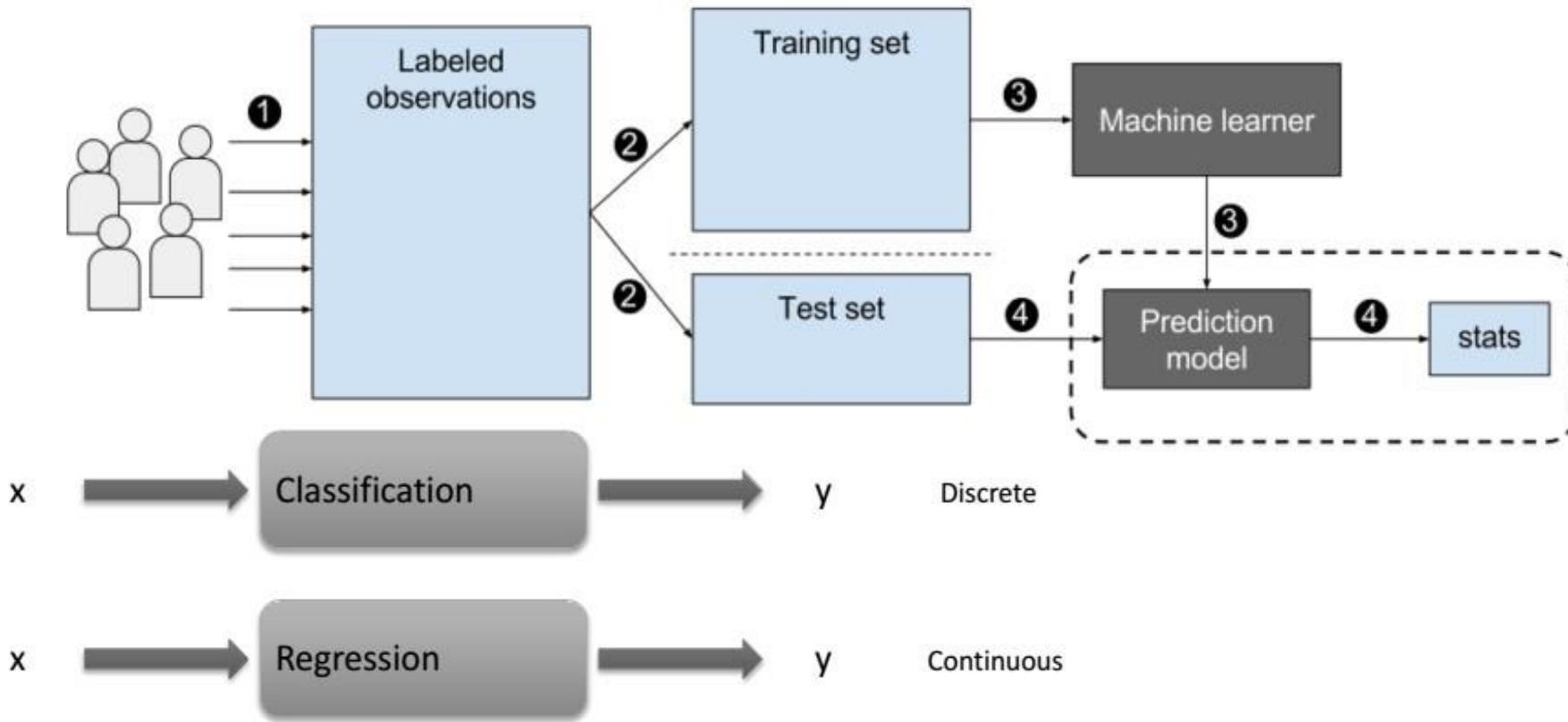
□ A robot driving learning problem:

- **Task T:** driving on public four-lane highways using vision sensors
- **Performance measure P:** average distance traveled before an error (as judged by human overseer)
- **Training experience E:** a sequence of images and steering commands recorded while observing a human driver

Types of Learning

- **Supervised** (inductive) learning
 - Training data includes desired outputs
- **Unsupervised** learning
 - Training data does not include desired outputs
- **Semi-supervised** learning
 - Training data includes a few desired outputs
- **Reinforcement** learning
 - Rewards from sequence of actions
- **Active** learning
 - Let users play an active role in the learning process.

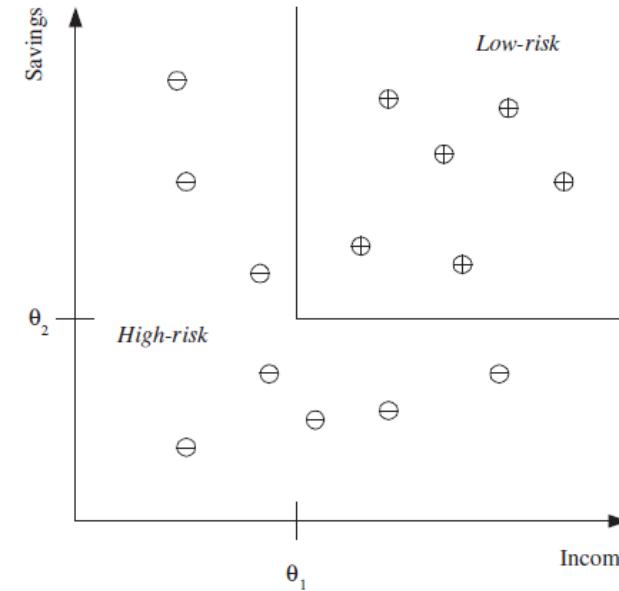
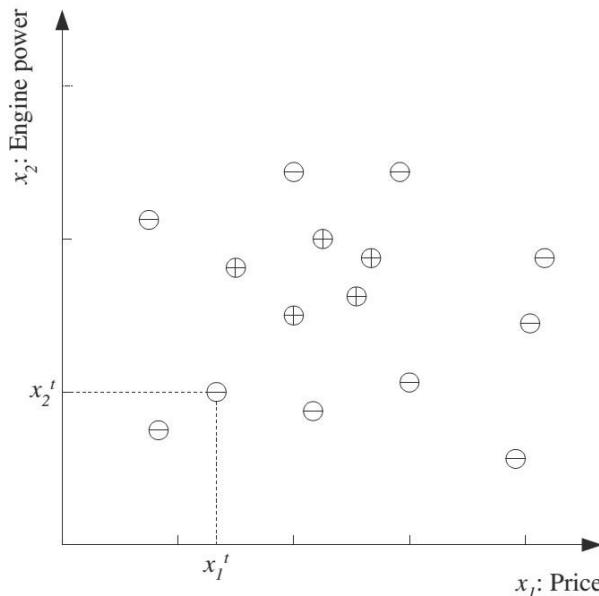
Supervised Learning



- Common Learning Algorithms: Linear Regression, Logistic Regression, Decision Tree, Random Forest, k-Nearest Neighbour, SVM, Neural Network etc.

Supervised Learning

- Given a set of data points $\{x^1, x^{(2)}, \dots, x^{(m)}\}$ associated to a set of outcomes, $\{y^1, y^{(2)}, \dots, y^{(m)}\}$, we want to build a classifier that learns how to predict y from x .



- Estimate the prediction model/function h , by minimizing loss function, e.g., $(y^{(i)} - h(x^i))^2$

Unsupervised Learning

- Looks for previously undetected patterns in a data set with no pre-existing labels.
 - Actual output y is absent!

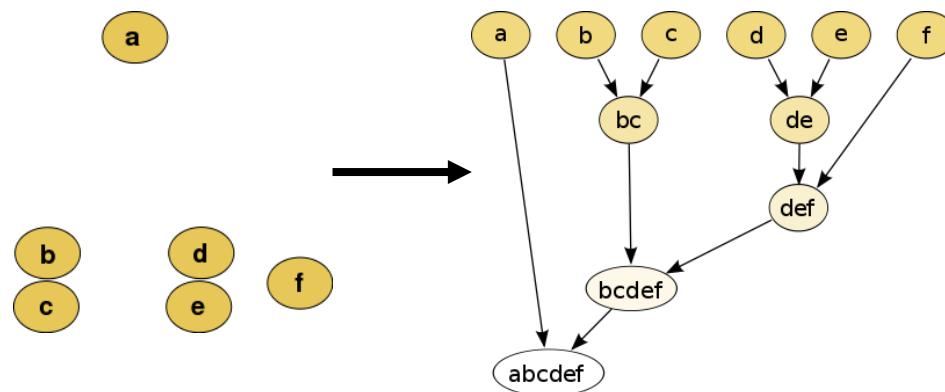


Fig: Agglomerative clustering

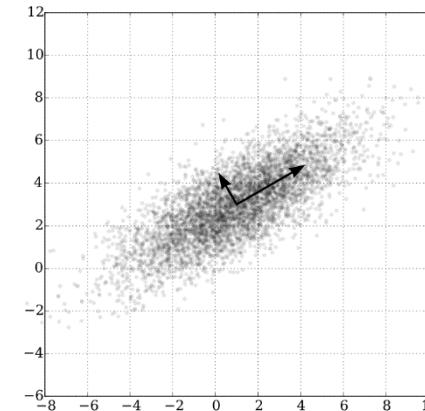


Fig: PCA

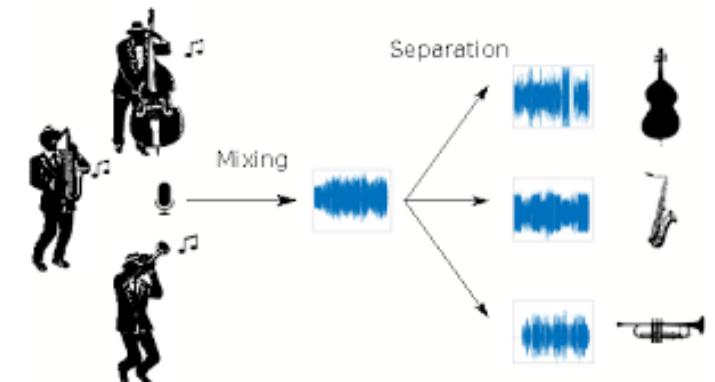
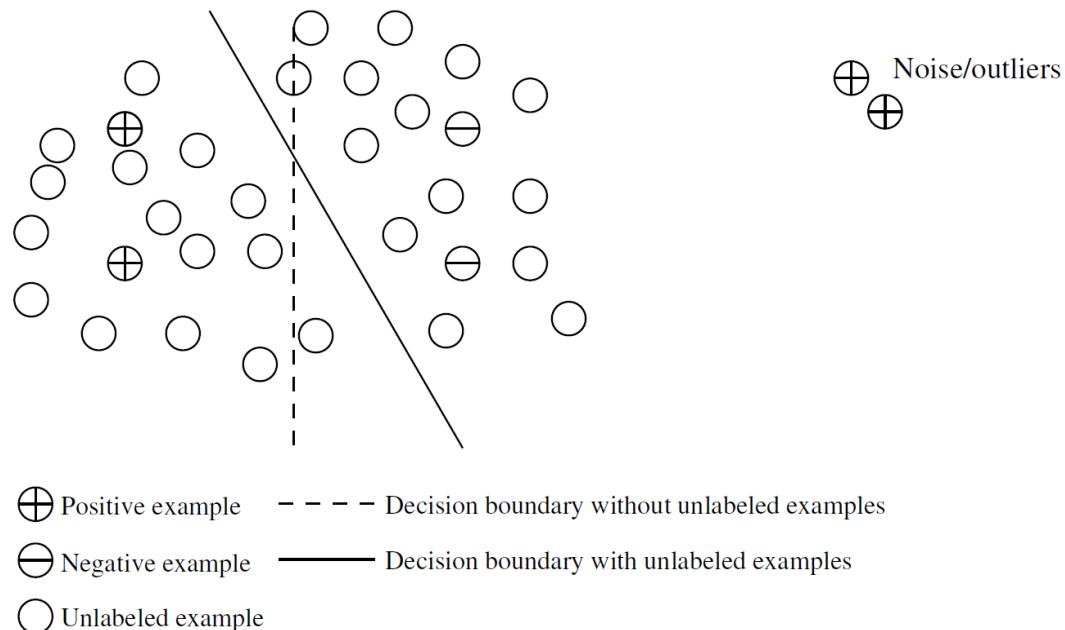


Fig: ICA

- Common Learning Algorithms: K-means, hierarchical clustering, mixture model, Local outlier factor, EM, PCA, ICA, etc.

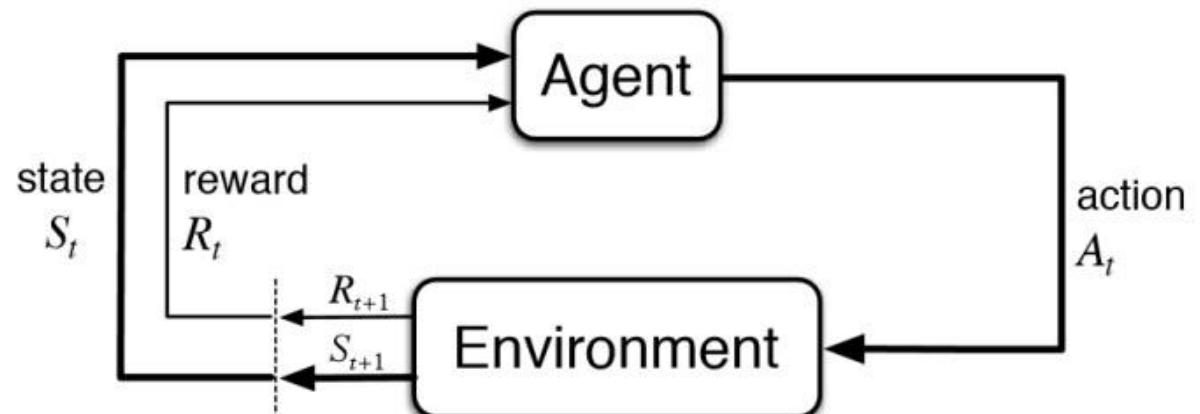
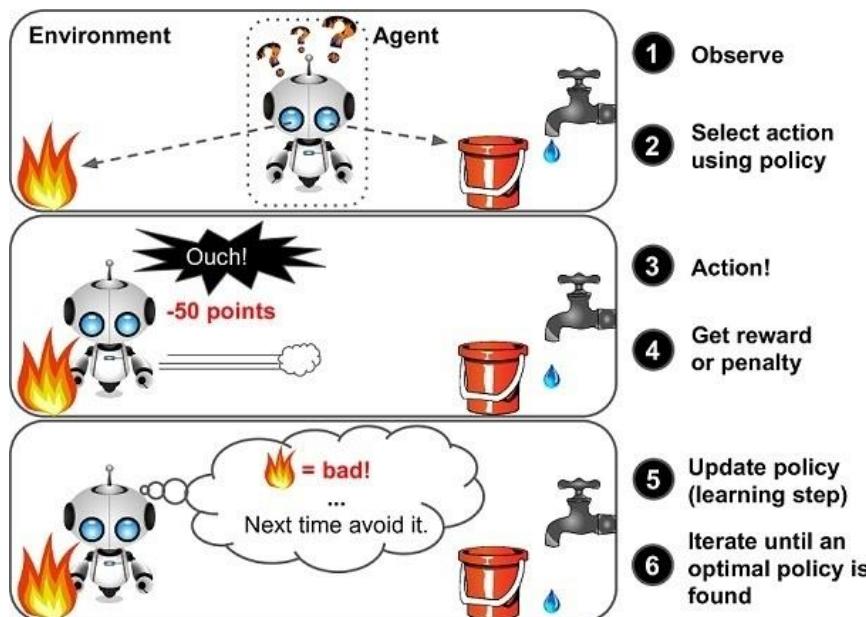
Semi-Supervised Learning

- Makes use of both labeled and unlabeled examples when learning a model.
- Labeled examples are used to learn class models and unlabeled examples are used to refine the boundaries between classes.



Reinforcement Learning

- Should be able to assess the goodness of policies and learn from past good action sequences to be able to generate a policy.
 - what is important is the *policy* that is the sequence of correct actions to reach the goal.



- Common Learning Algorithms: Markov Decision Process, Q-Learning, Deep Q Learning, etc.

Basic Steps to Machine Learning



Our Focus

- Batch learning vs. Online learning
 - Batch: All data available before training
 - Online: Data samples come one after another
- Passive learning vs. Active learning
 - Passive learning: Only observing given data
 - Active learning: Can ask labels for some data as the learning machine wants
- Structured prediction
 - y itself has internal structures, e.g., a sequence, a tree.
- This course mainly focuses on supervised, batch, passive, non-structured ML



CSE 4621

Machine Learning

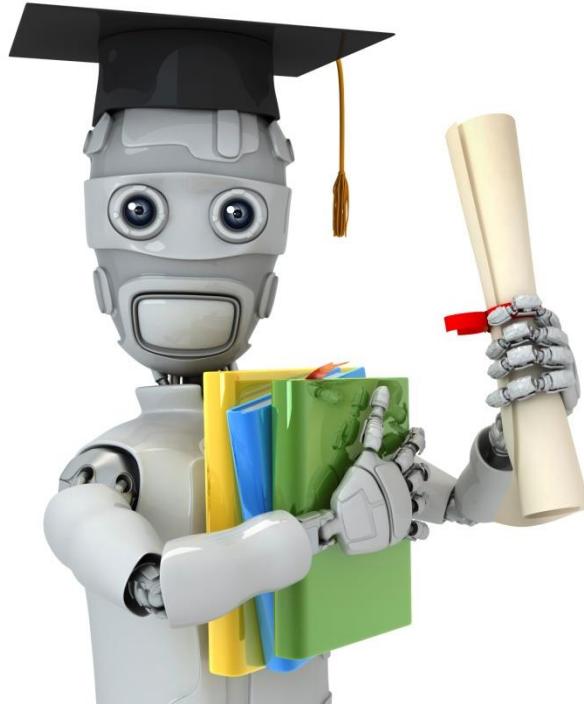
Lecture 2

Md. Hasanul Kabir, PhD.

Professor, CSE Department

Islamic University of Technology (IUT)





Linear regression with one variable

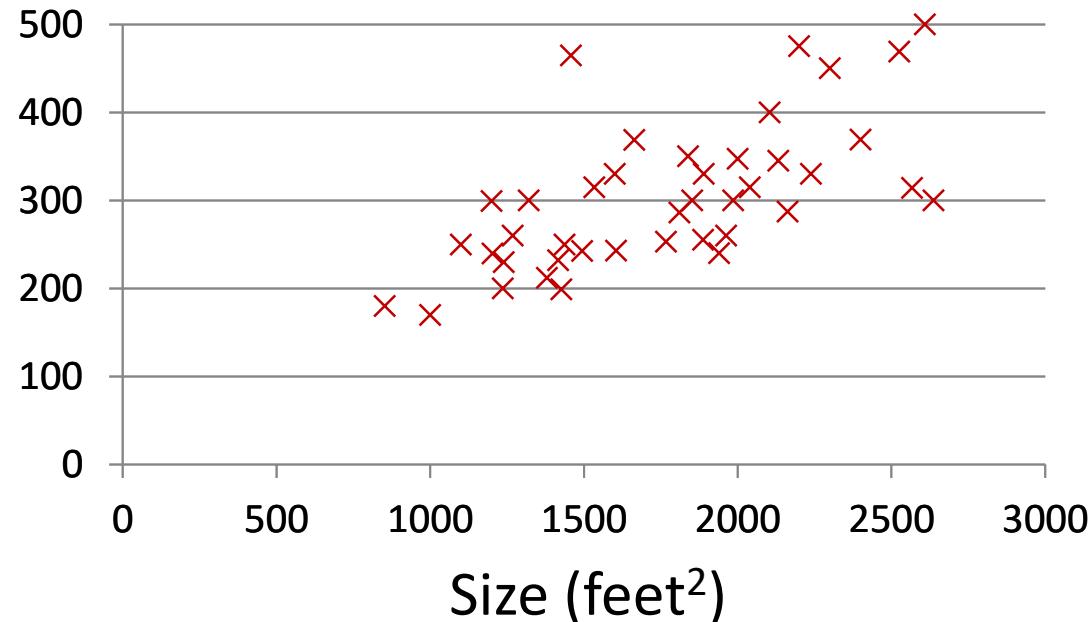
Model representation

Machine Learning

Source & Special Thanks to Andrew Ng (Coursera) Machine Learning Course

Housing Prices (Portland, OR)

Price
(in 1000s
of dollars)



Supervised Learning

Given the “right answer” for each example in the data.

Regression Problem

Predict real-valued output

Training set of housing prices (Portland, OR)

Size in feet ² (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

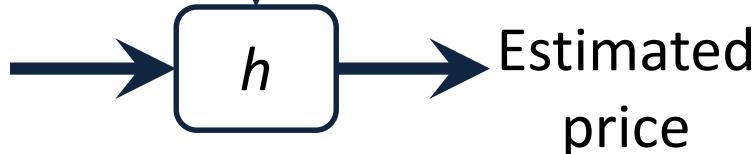
y's = “output” variable / “target” variable

Training Set



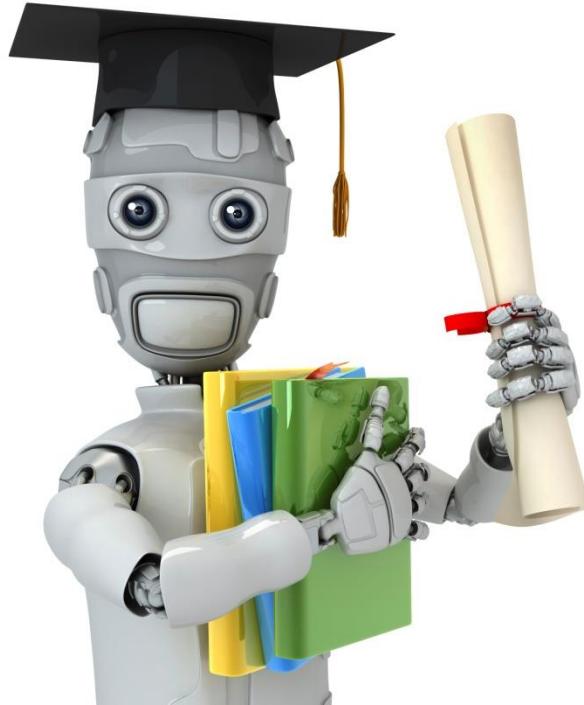
Learning Algorithm

Size of
house



How do we represent h ?

Linear regression with one variable.
Univariate linear regression.



Machine Learning

Linear regression with one variable

Cost function

Training Set

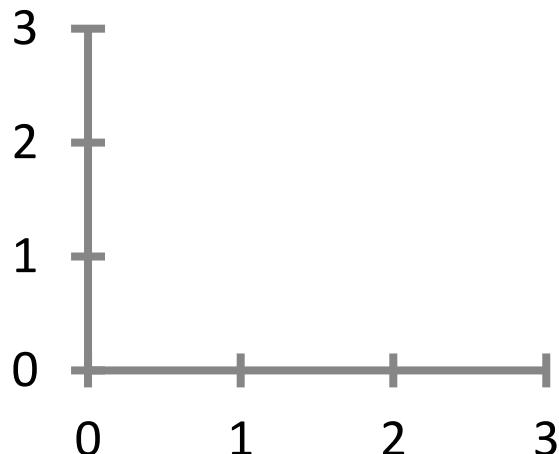
Size in feet ² (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

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

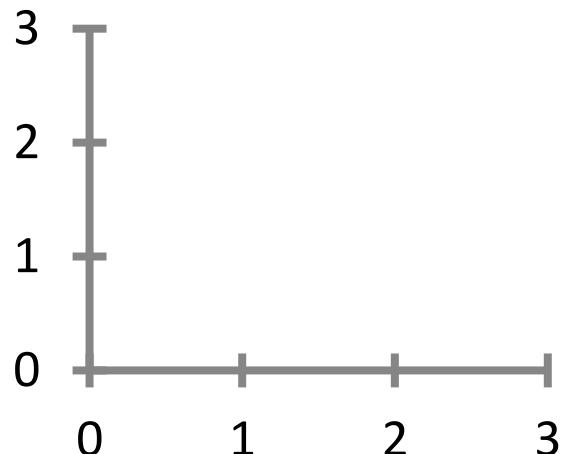
θ_i 's: Parameters

How to choose θ_i 's ?

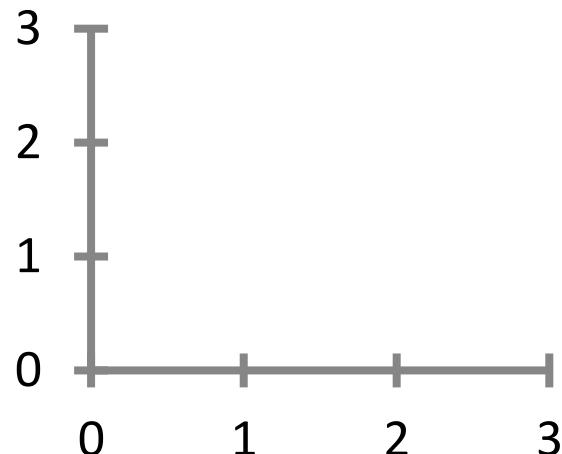
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



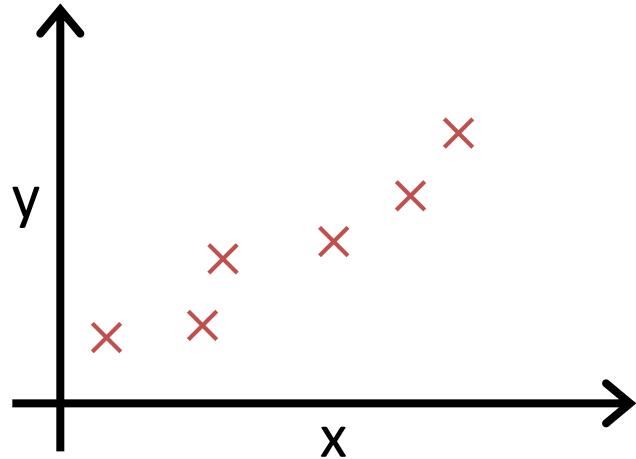
$$\begin{aligned}\theta_0 &= 1.5 \\ \theta_1 &= 0\end{aligned}$$



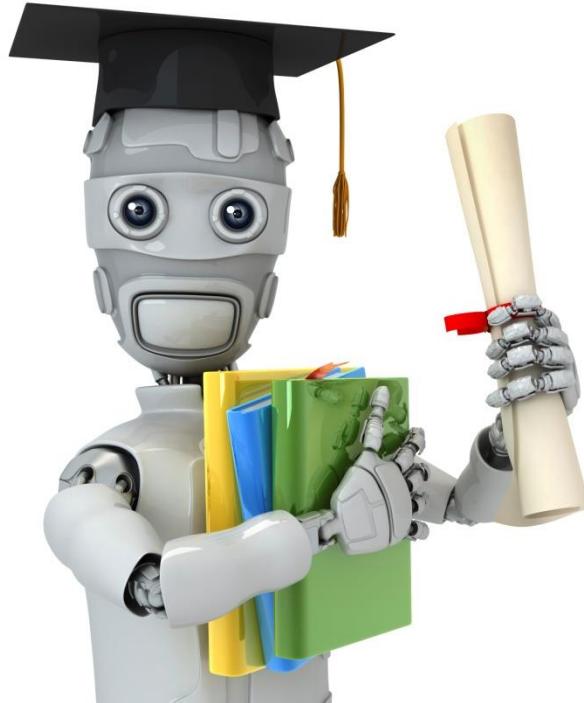
$$\begin{aligned}\theta_0 &= 0 \\ \theta_1 &= 0.5\end{aligned}$$



$$\begin{aligned}\theta_0 &= 1 \\ \theta_1 &= 0.5\end{aligned}$$



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



Machine Learning

Linear regression
with one variable

Cost function
intuition I

Simplified

Hypothesis:

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

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

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

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

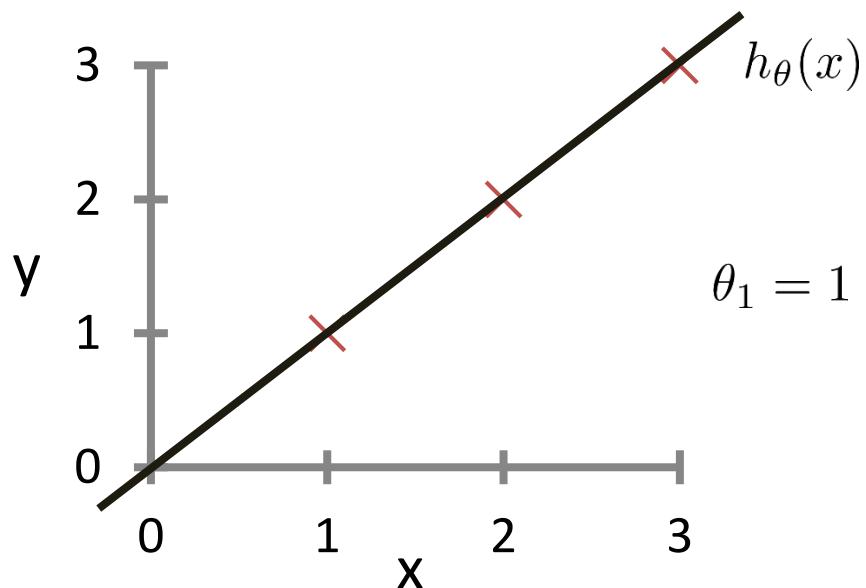
$$h_{\theta}(x) = \theta_1 x$$

$$\theta_1$$

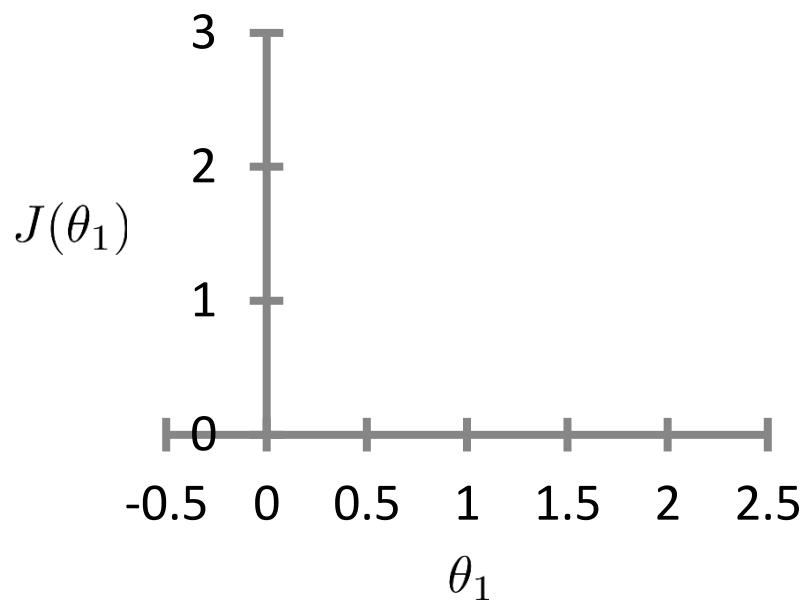
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

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

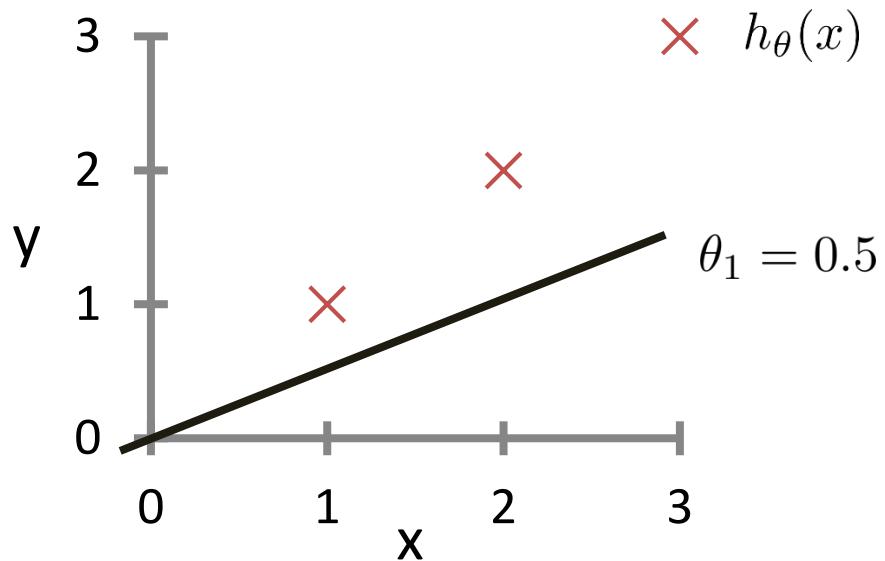
$h_{\theta}(x)$
(for fixed θ_1 , this is a function of x)



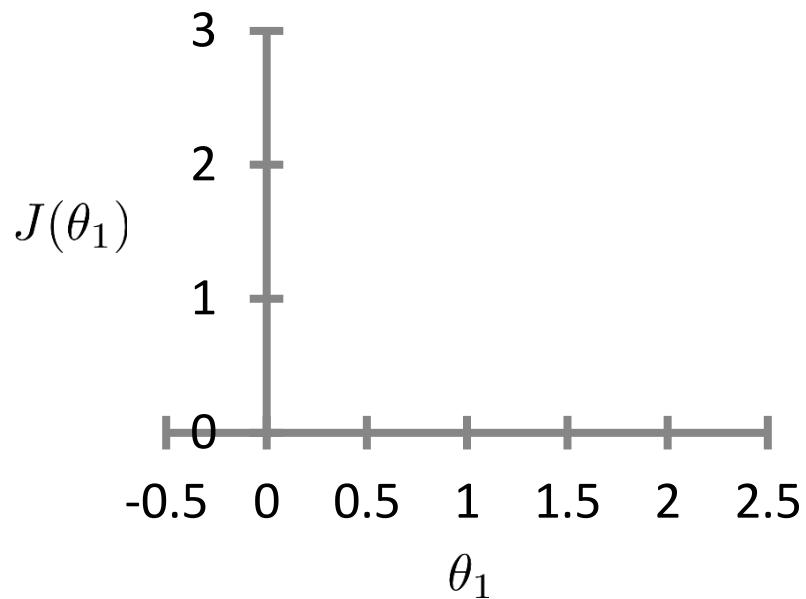
$J(\theta_1)$
(function of the parameter θ_1)



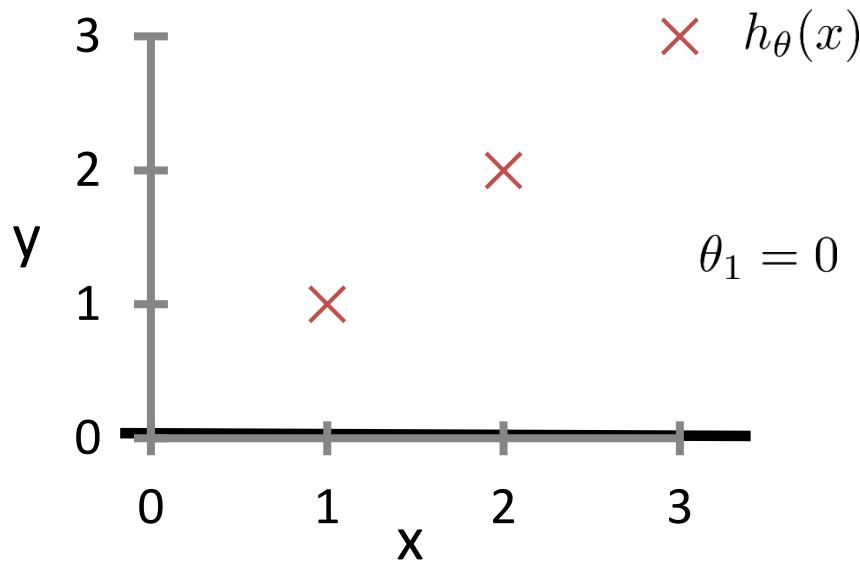
$h_{\theta}(x)$
(for fixed θ_1 , this is a function of x)



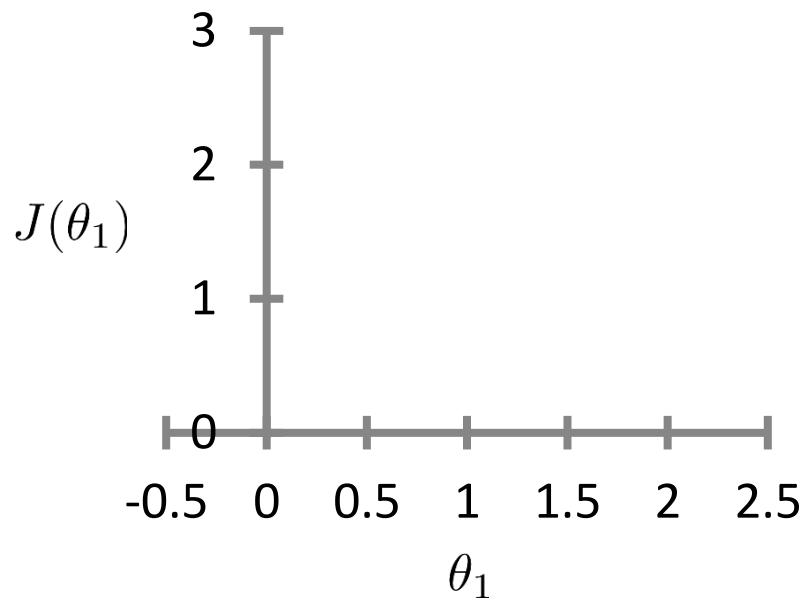
$J(\theta_1)$
(function of the parameter θ_1)

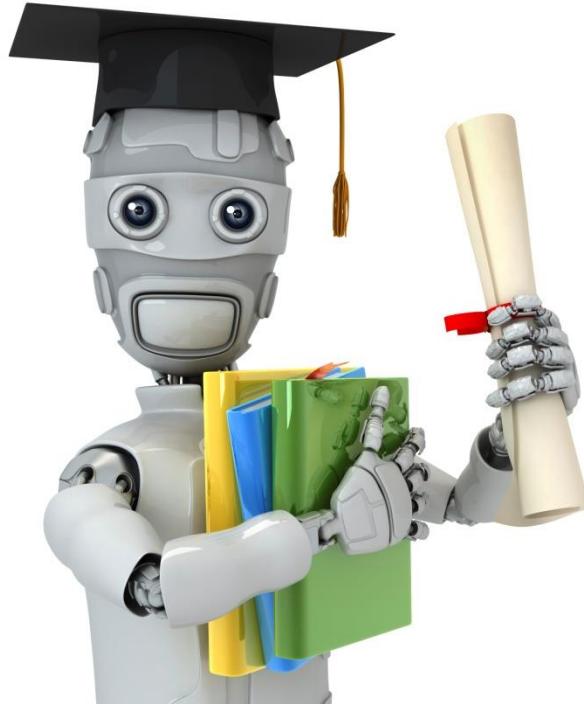


$h_{\theta}(x)$
(for fixed θ_1 , this is a function of x)



$J(\theta_1)$
(function of the parameter θ_1)





Machine Learning

Linear regression
with one variable

Cost function
intuition II

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

Parameters: θ_0, θ_1

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

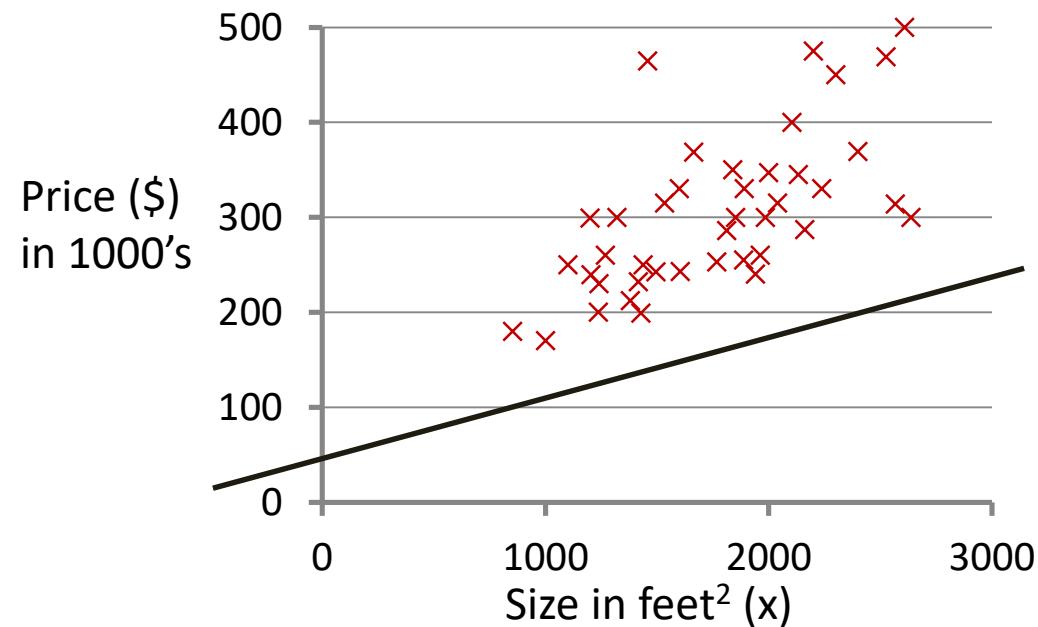
Goal: minimize $J(\theta_0, \theta_1)$
 θ_0, θ_1

$$h_{\theta}(x)$$

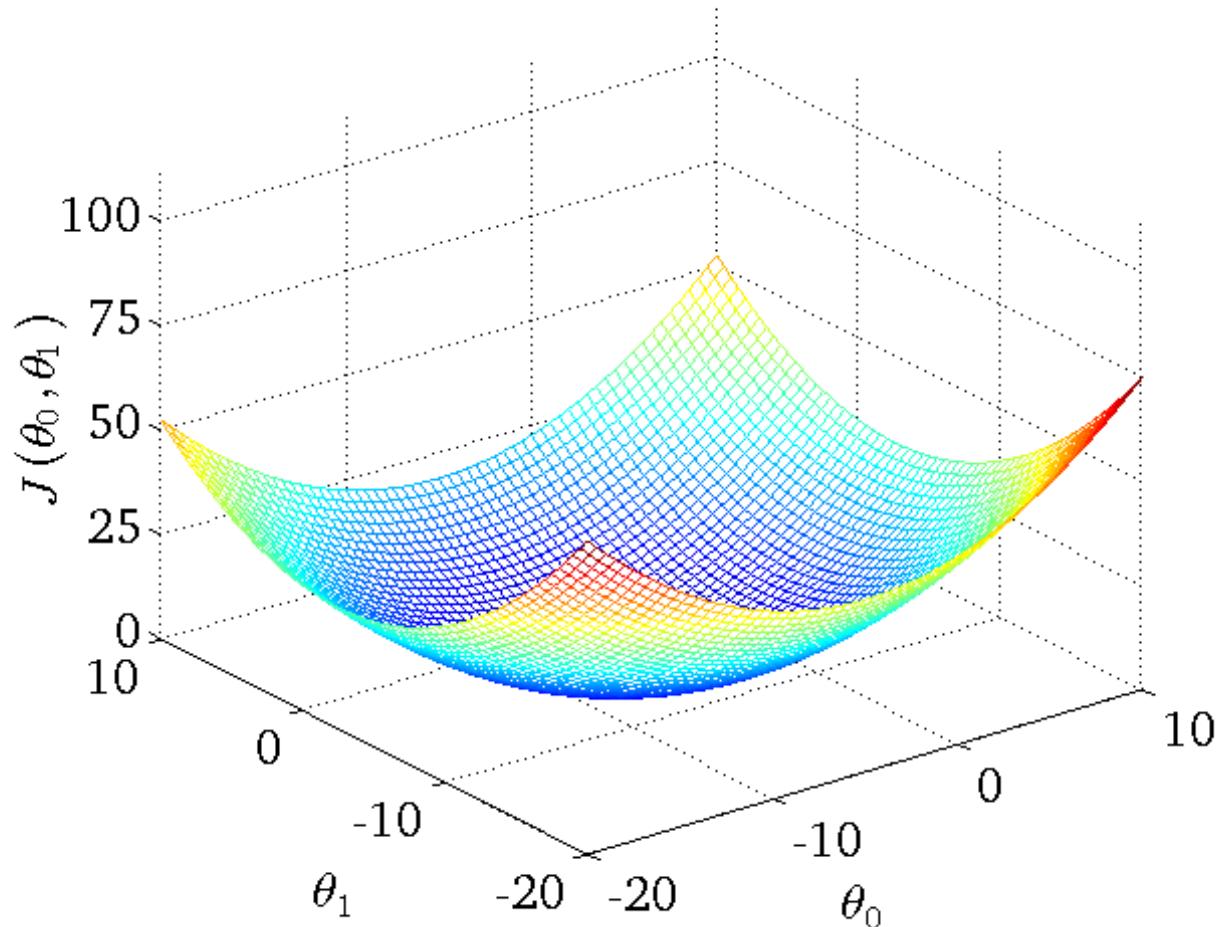
(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)

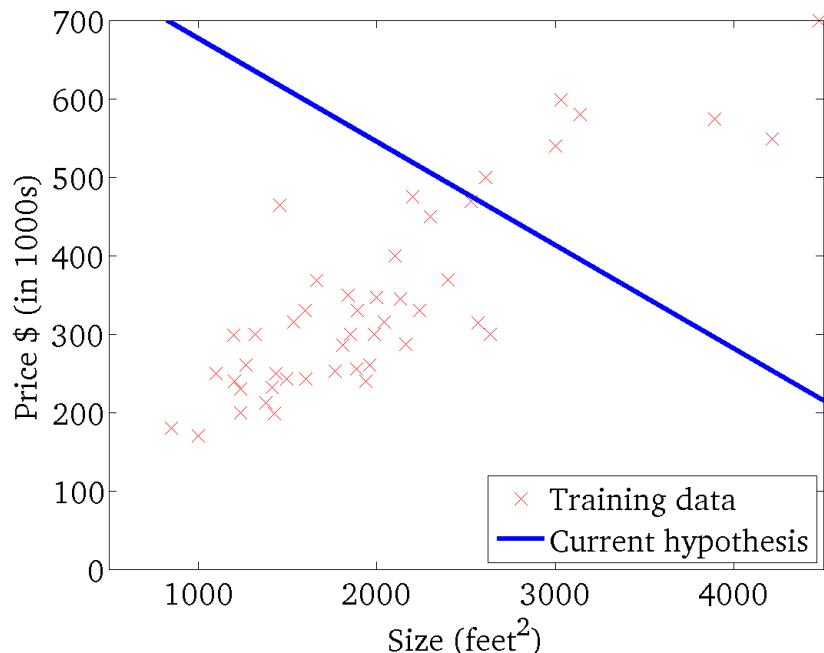


$$h_{\theta}(x) = 50 + 0.06x$$



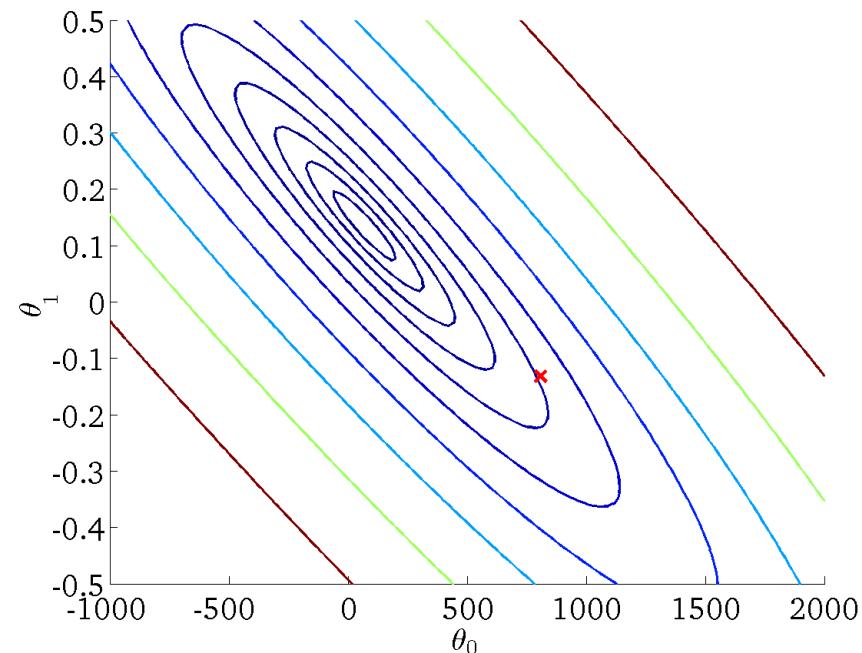
$$h_{\theta}(x)$$

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



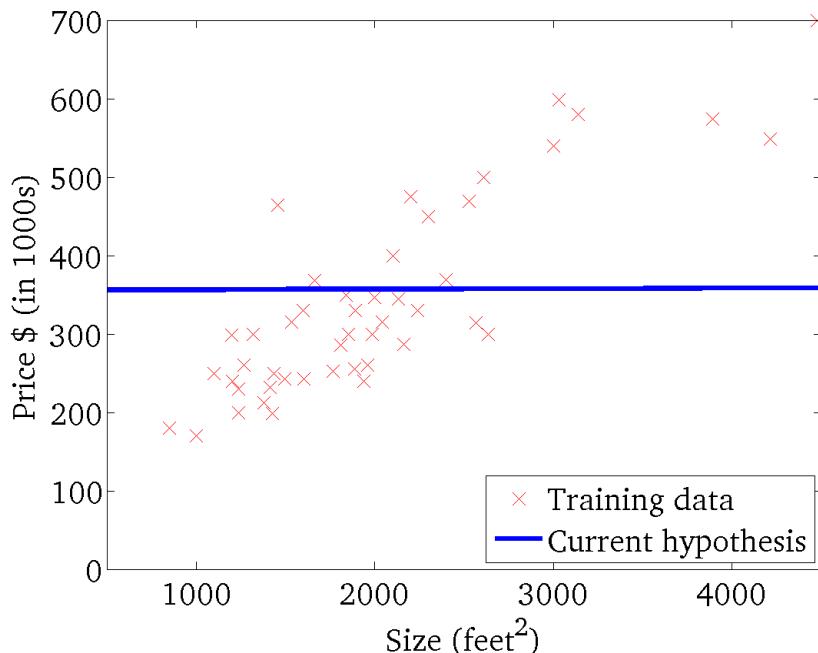
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



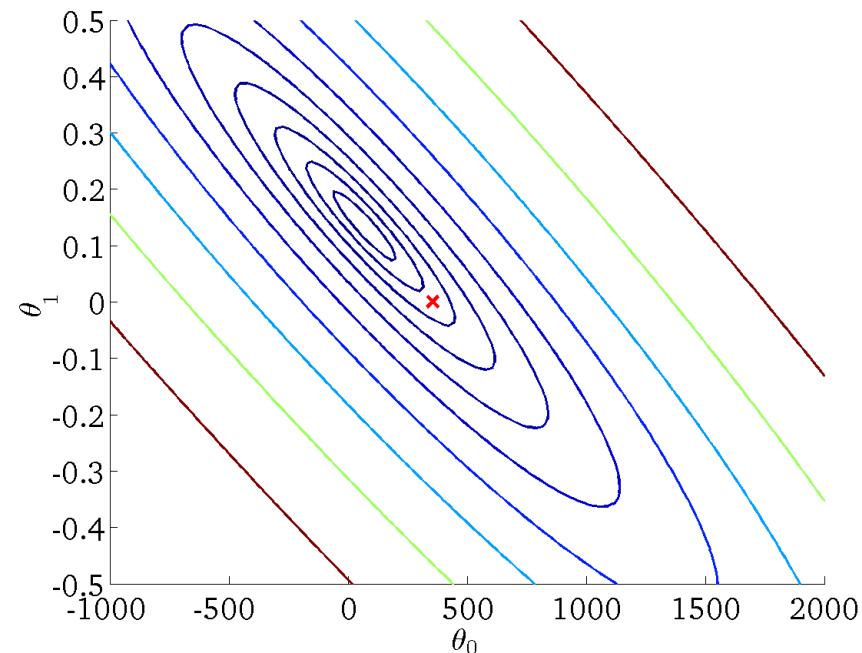
$$h_{\theta}(x)$$

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



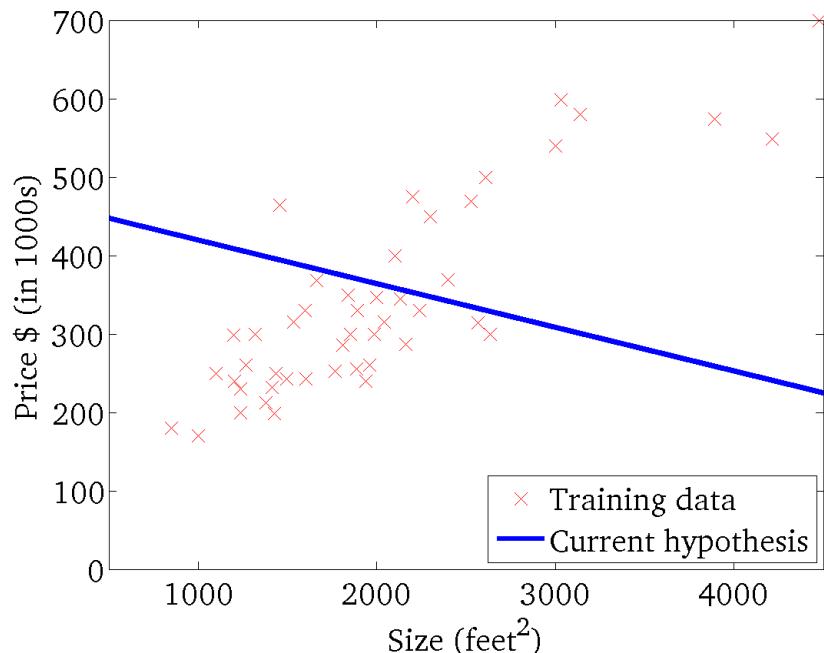
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



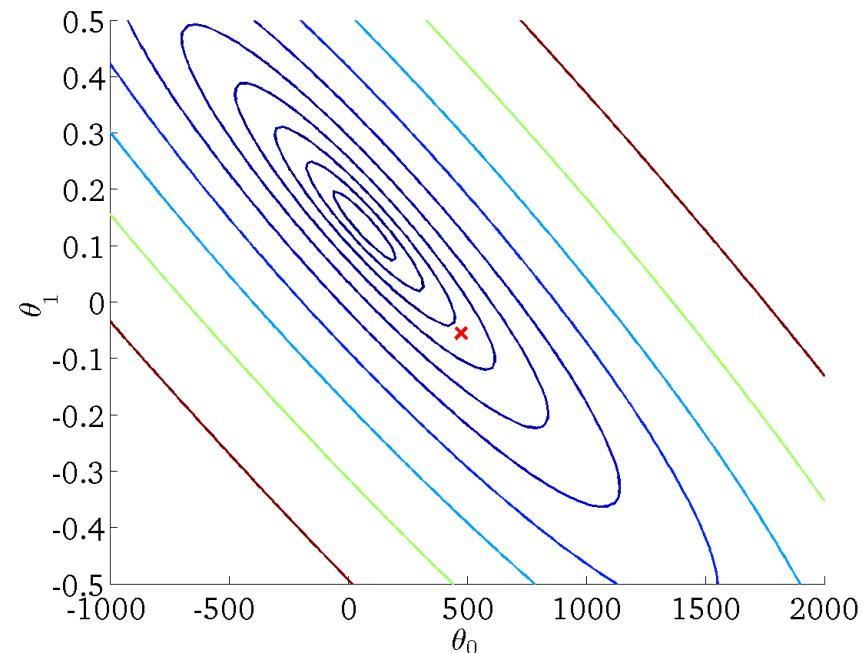
$$h_{\theta}(x)$$

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



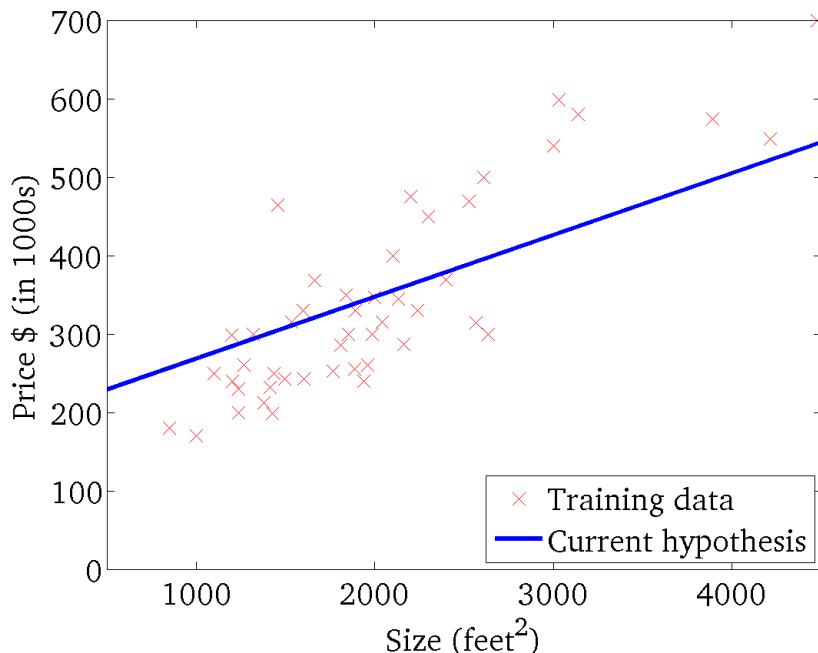
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



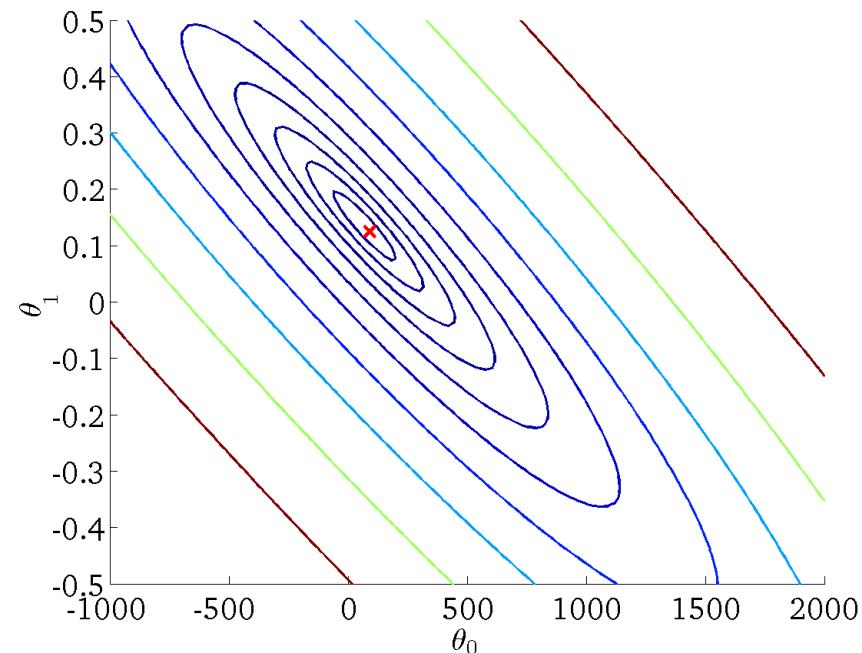
$$h_{\theta}(x)$$

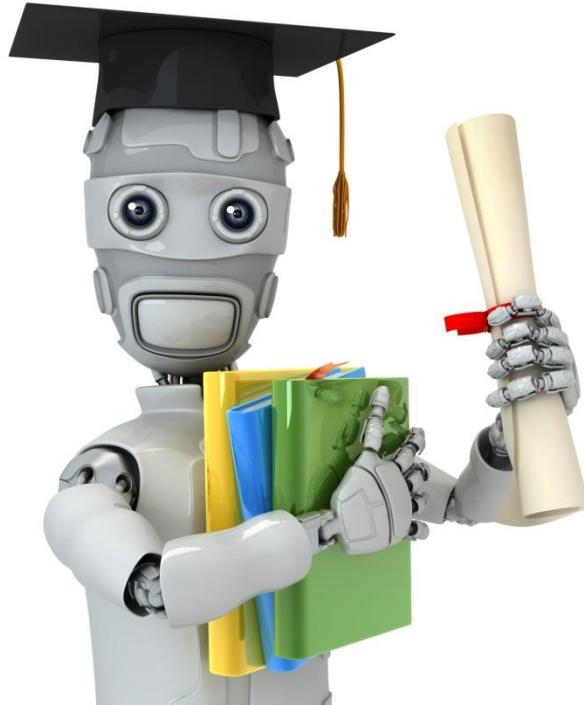
(for fixed θ_0, θ_1 , this is a function of x)



$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)





Machine Learning

Linear regression
with one variable

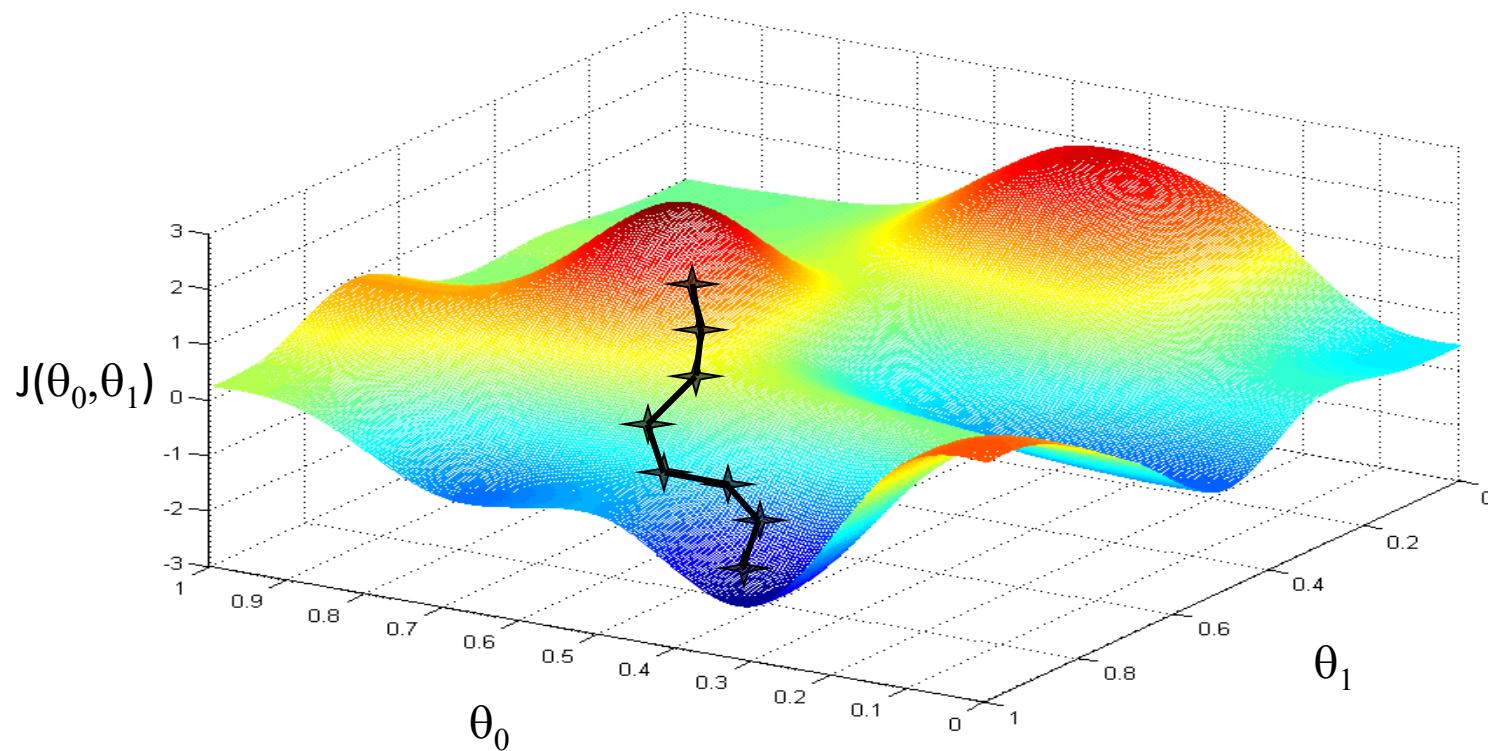
Gradient
descent

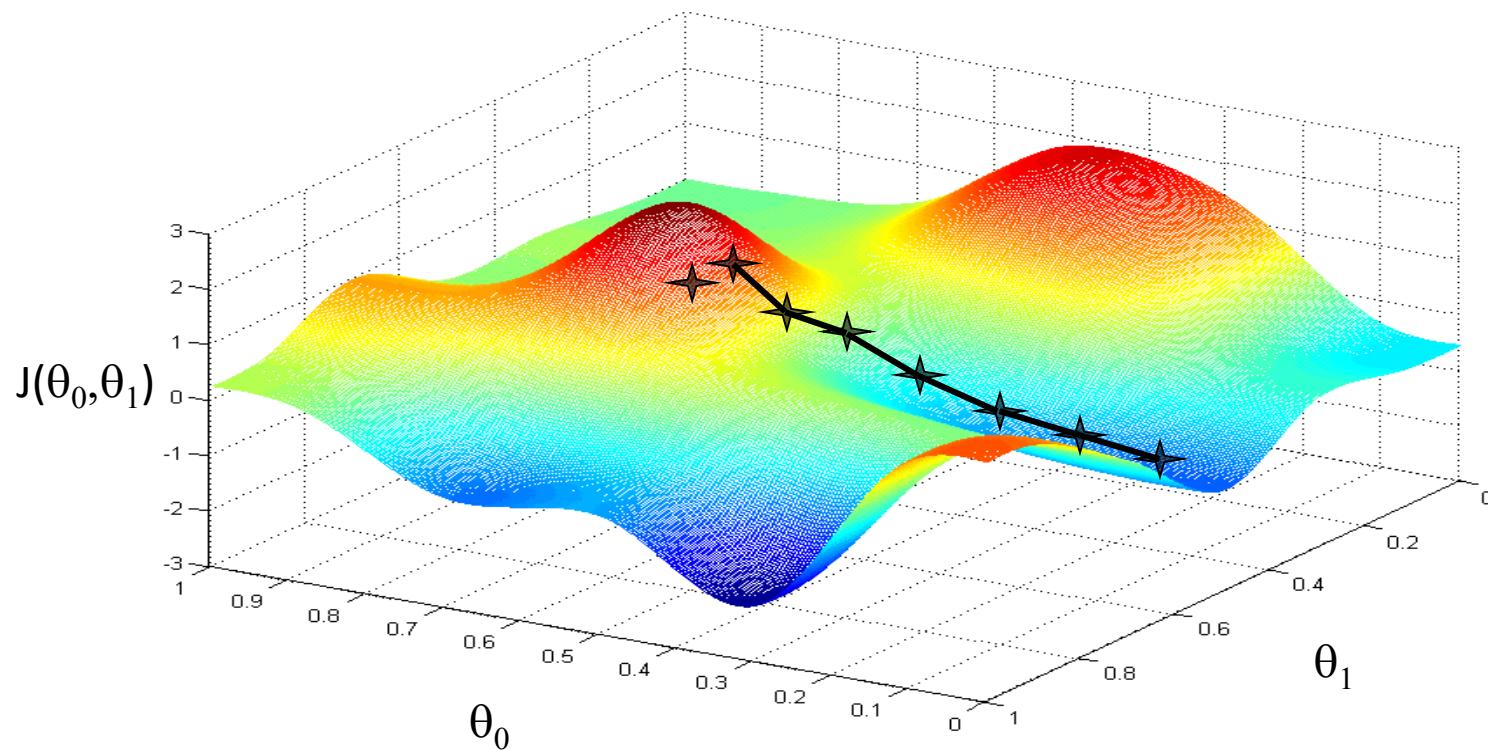
Have some function $J(\theta_0, \theta_1)$

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

Correct: Simultaneous update

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

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

$$\theta_0 := \text{temp0}$$

$$\theta_1 := \text{temp1}$$

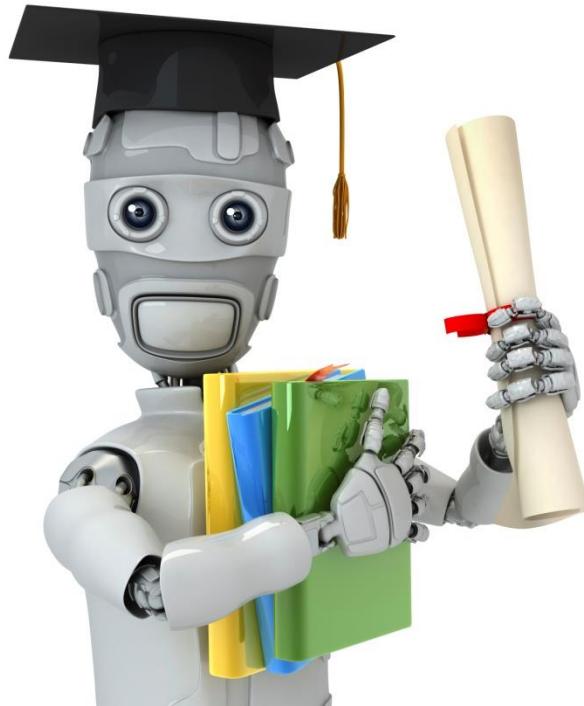
Incorrect:

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

$$\theta_0 := \text{temp0}$$

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

$$\theta_1 := \text{temp1}$$



Machine Learning

Linear regression with one variable

Gradient descent intuition

Gradient descent algorithm

repeat until convergence {

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

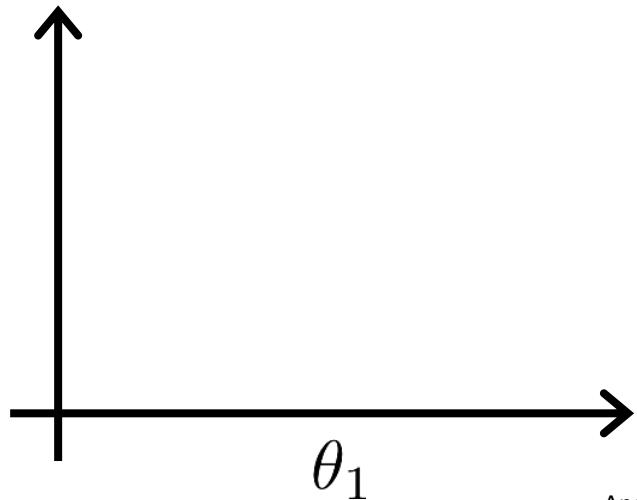
}

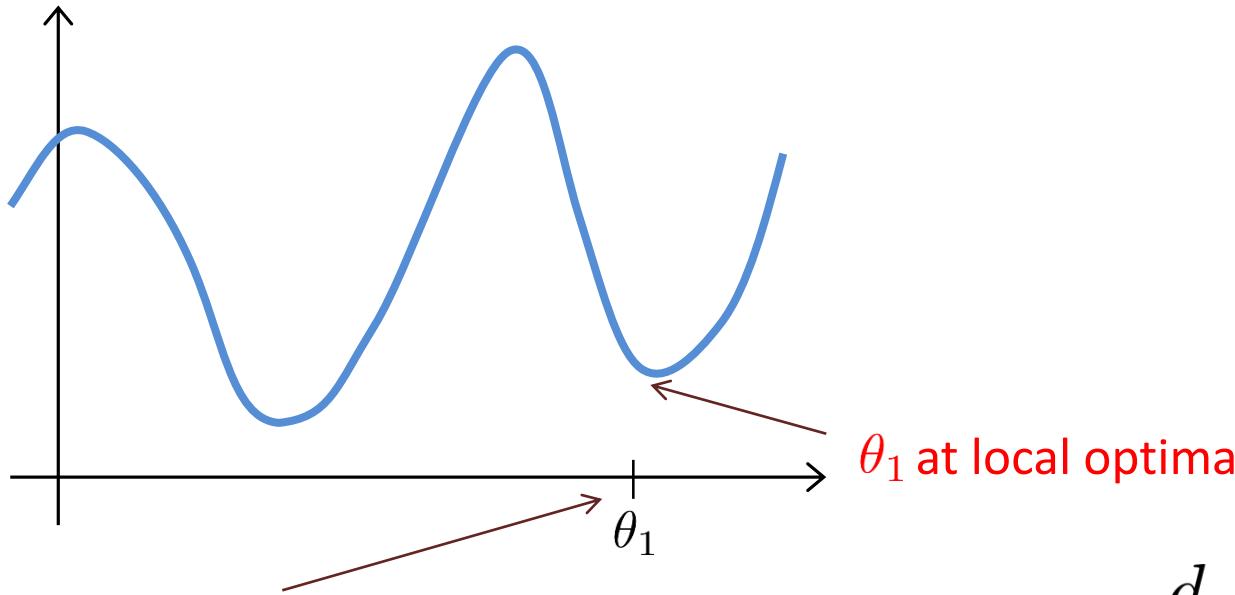


$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

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.





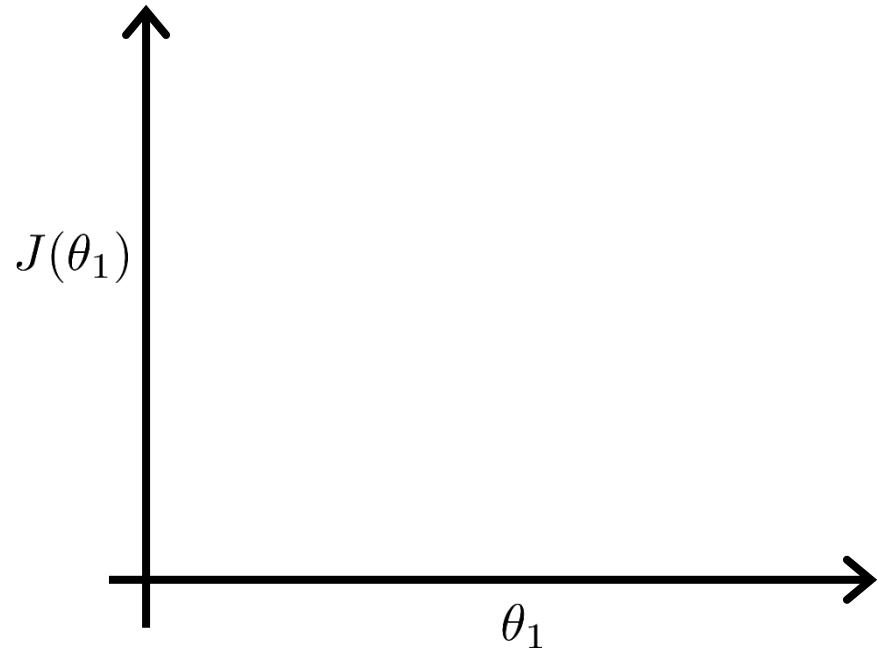
Current value of θ_1

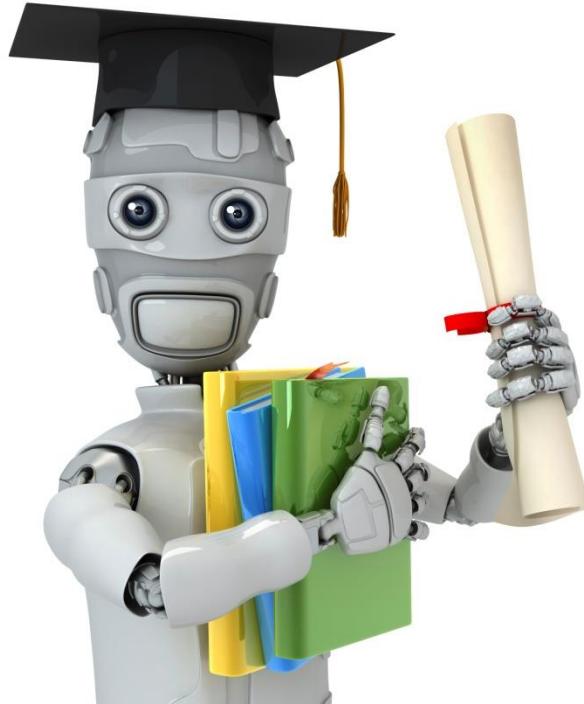
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

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

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

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





Machine Learning

Linear regression with one variable

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

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

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

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

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) =$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) =$$

Gradient descent algorithm

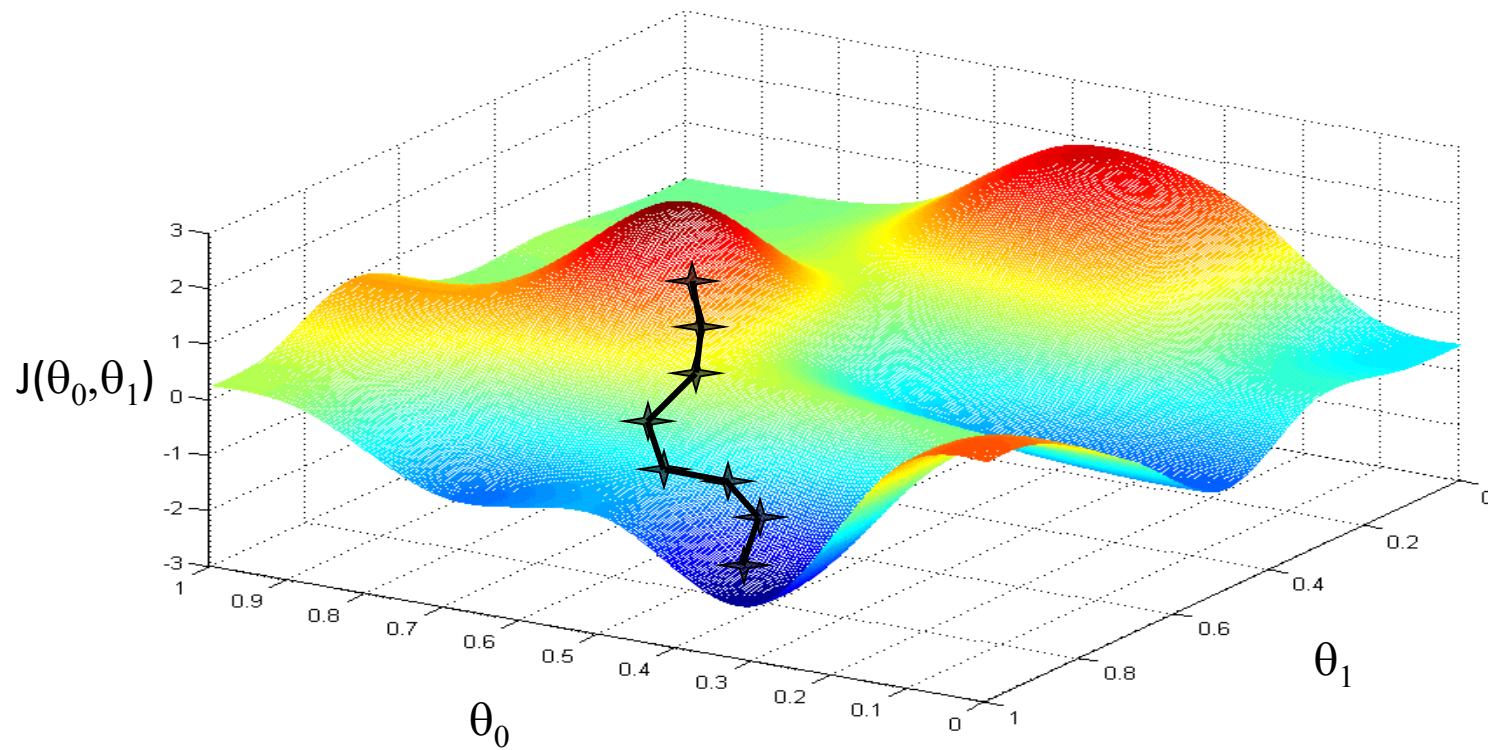
repeat until convergence {

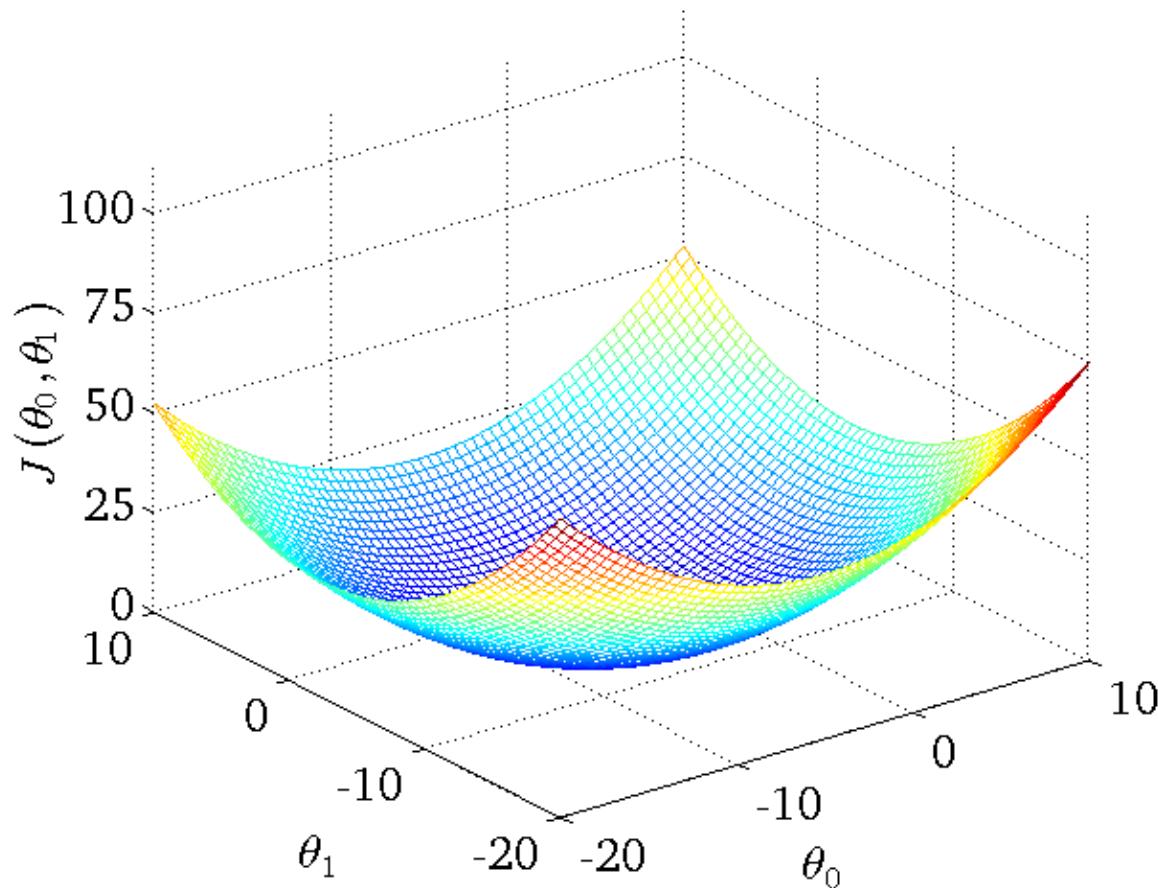
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$$

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

}

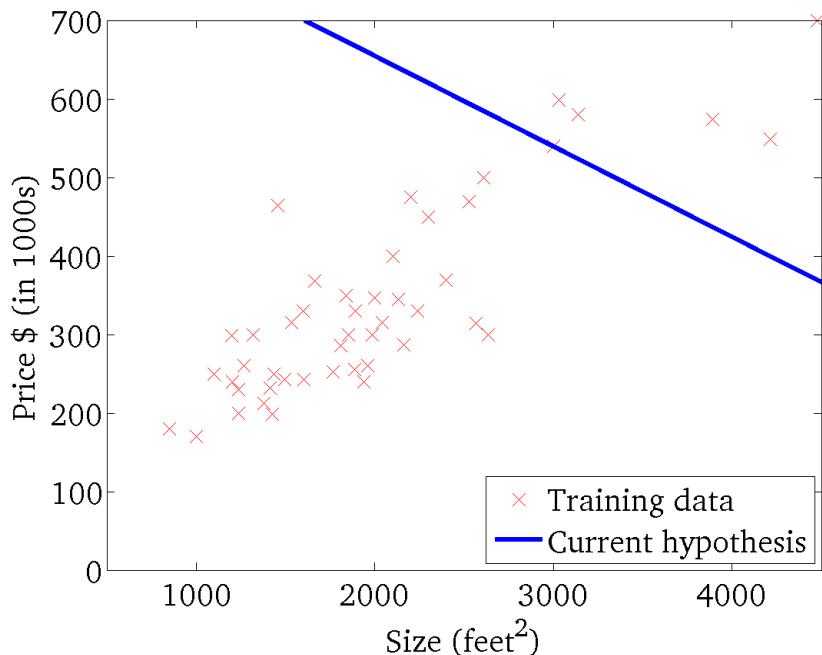
} update
 θ_0 and θ_1
simultaneously





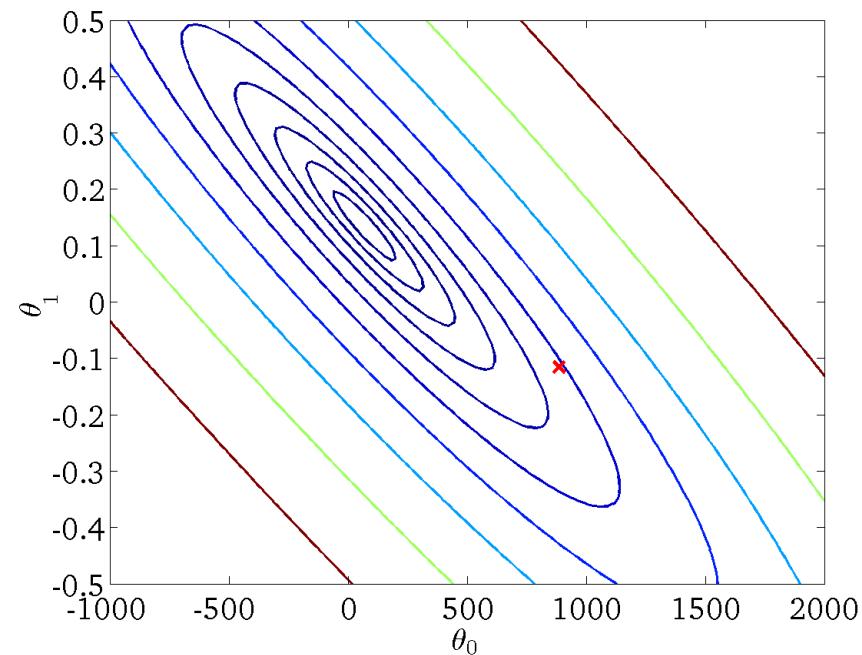
$$h_{\theta}(x)$$

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



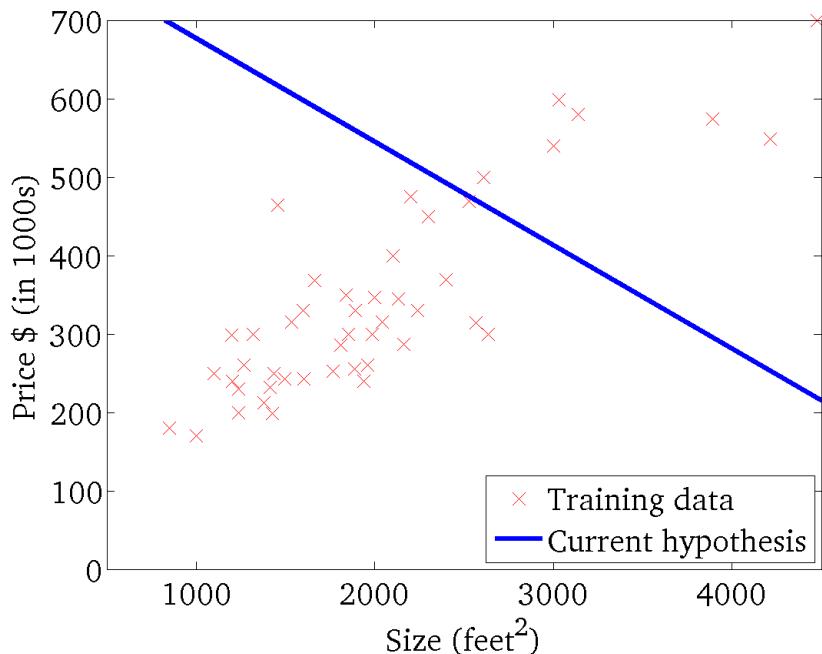
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



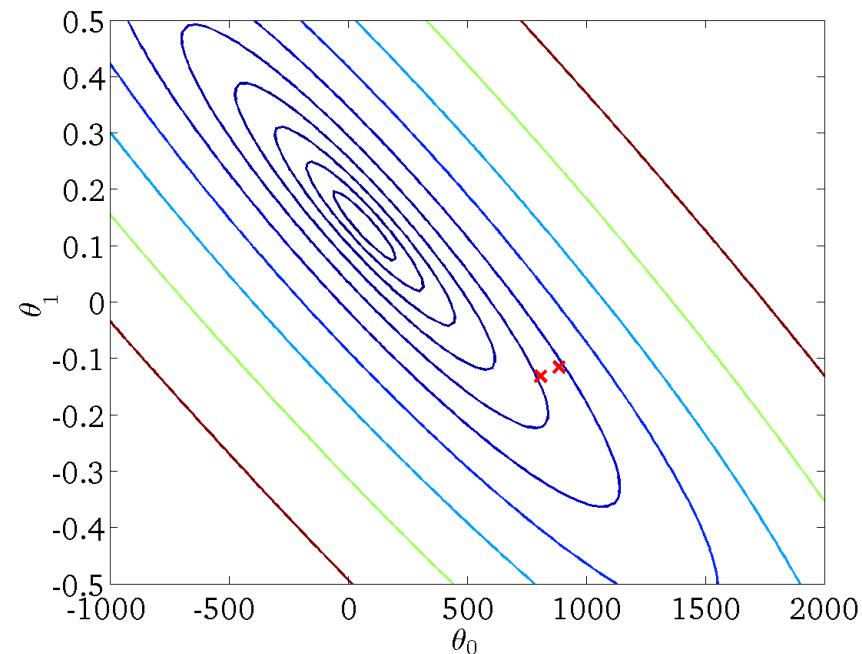
$$h_{\theta}(x)$$

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



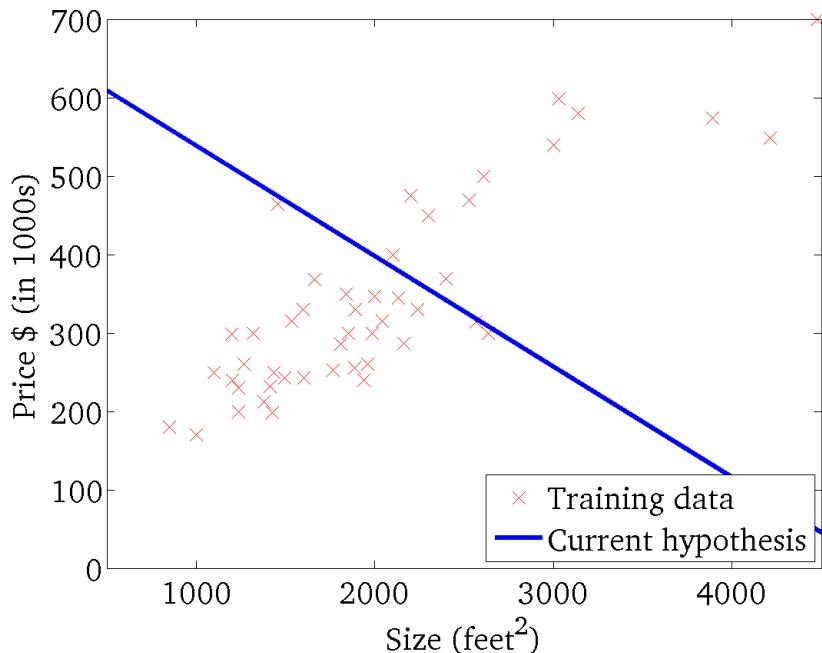
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



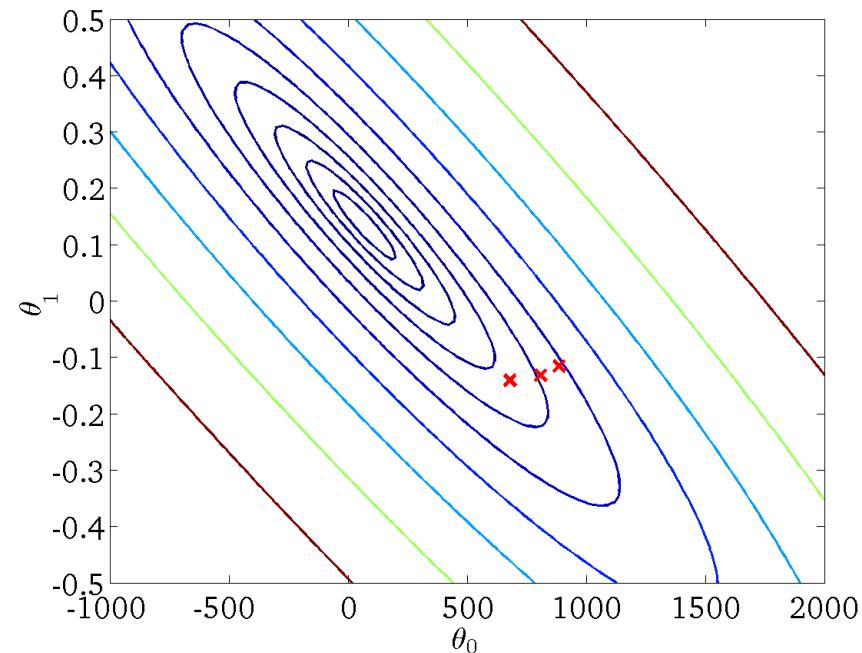
$$h_{\theta}(x)$$

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



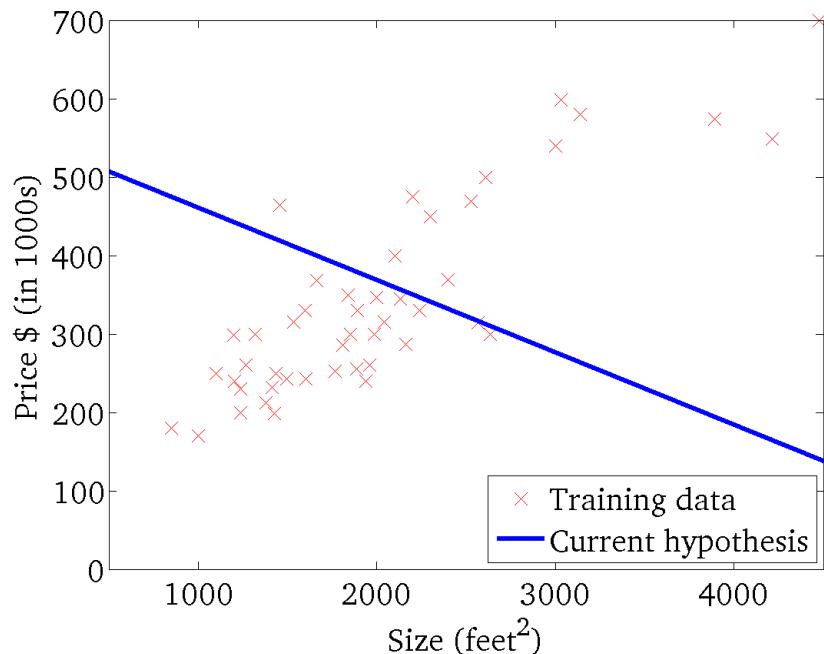
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



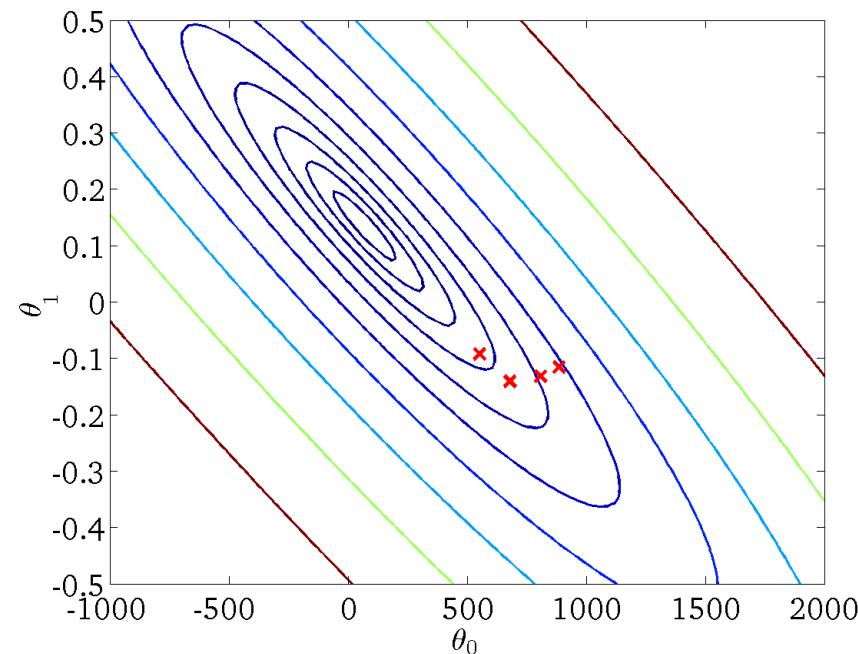
$$h_{\theta}(x)$$

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



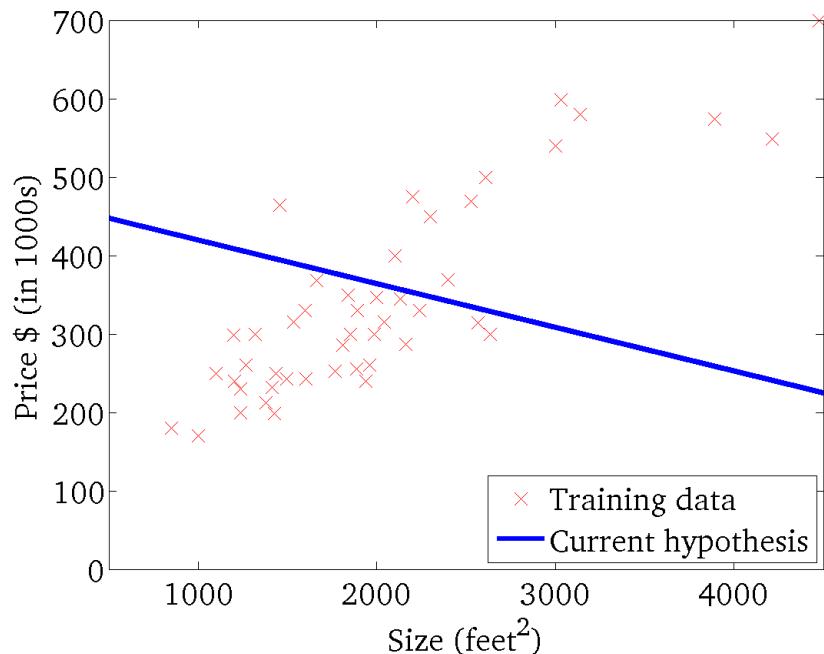
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



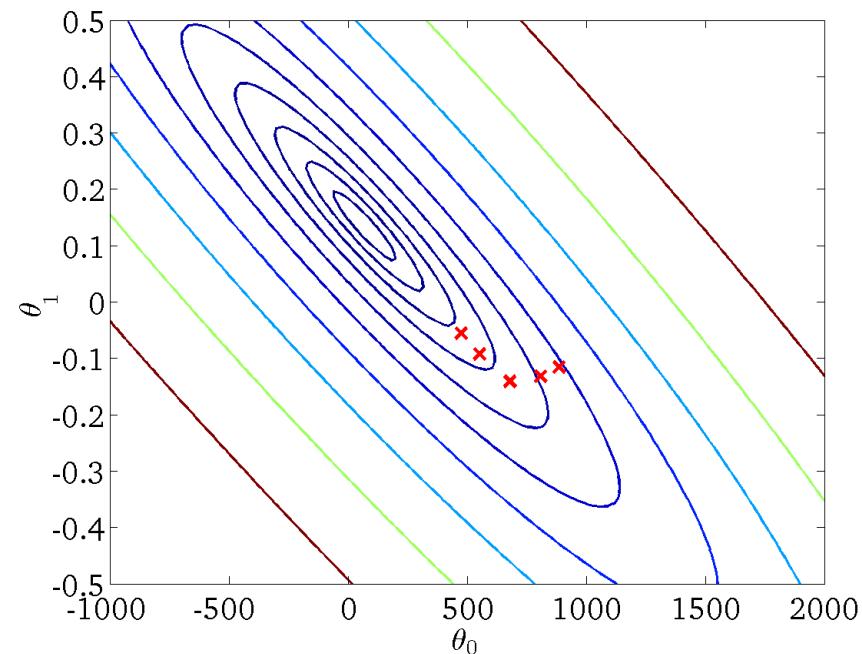
$$h_{\theta}(x)$$

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



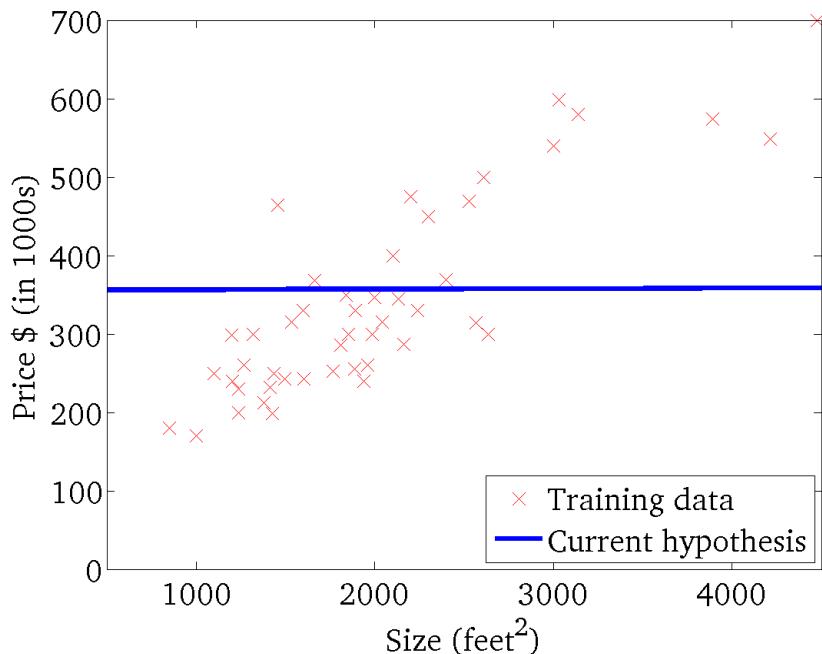
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



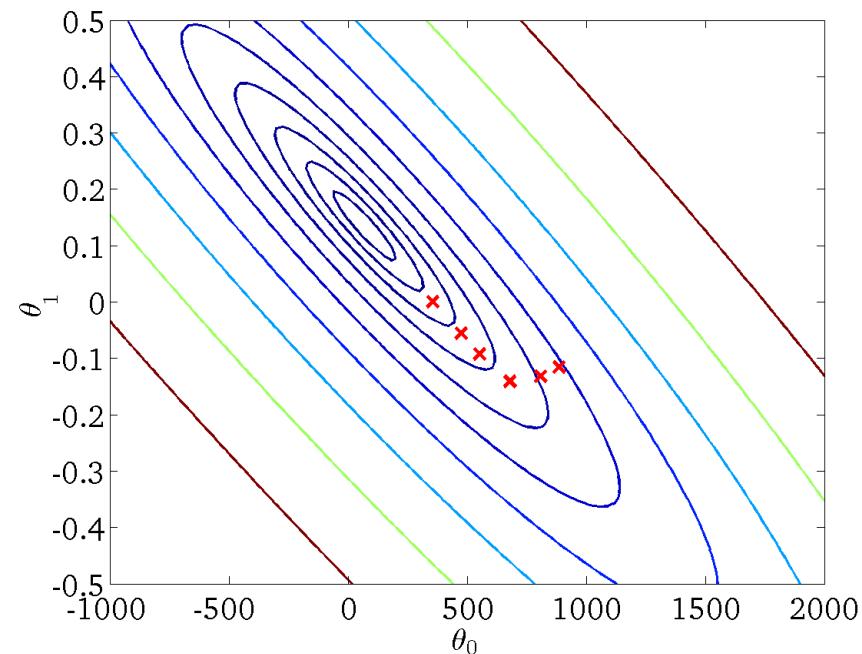
$$h_{\theta}(x)$$

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



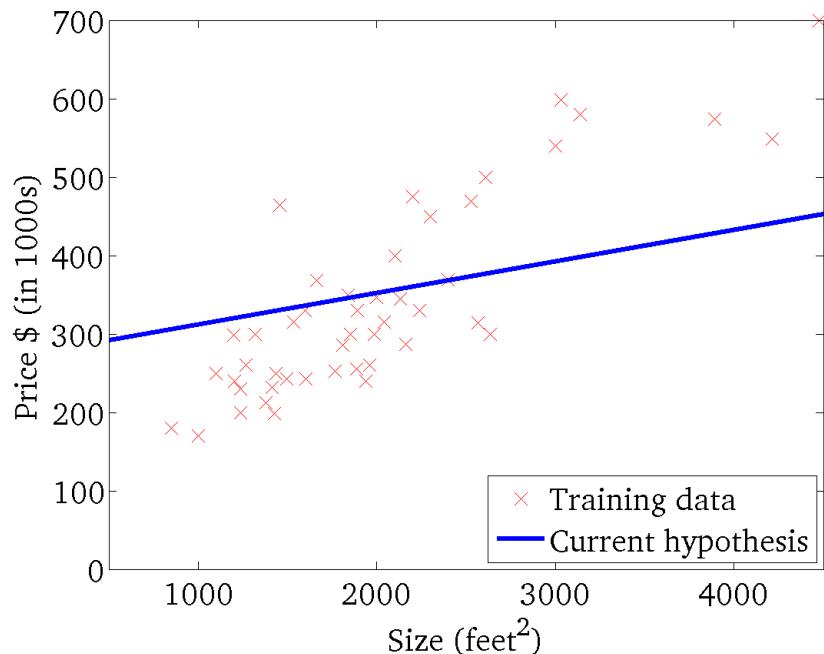
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



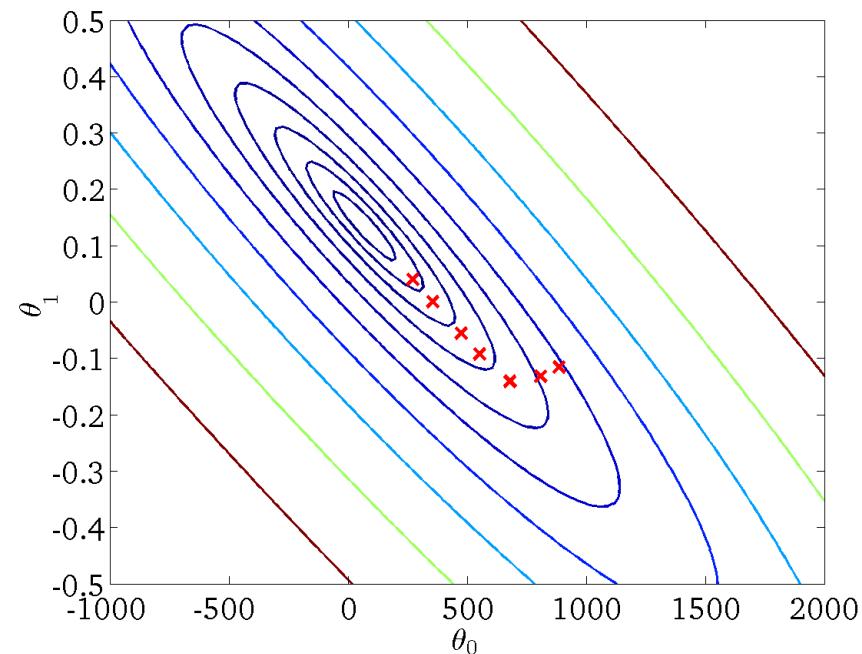
$$h_{\theta}(x)$$

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



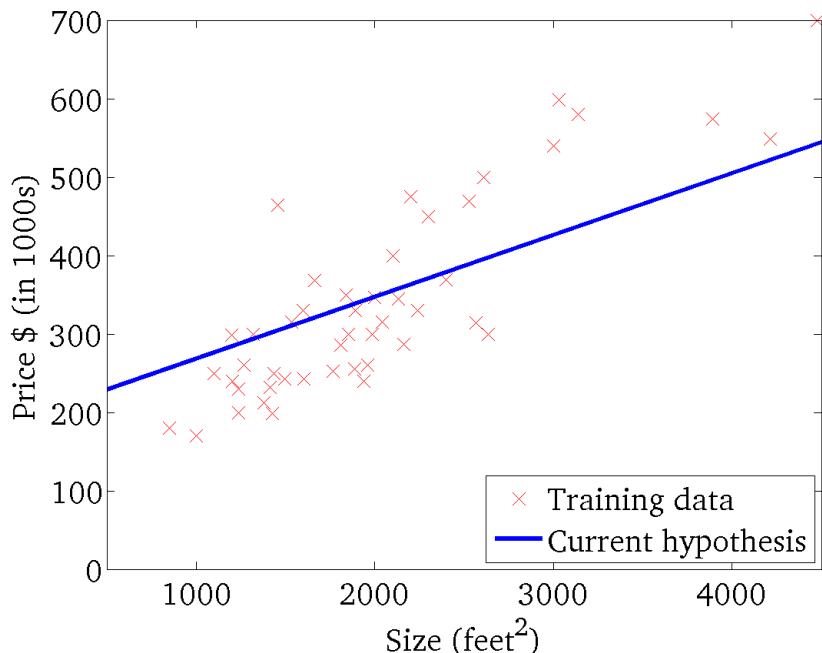
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



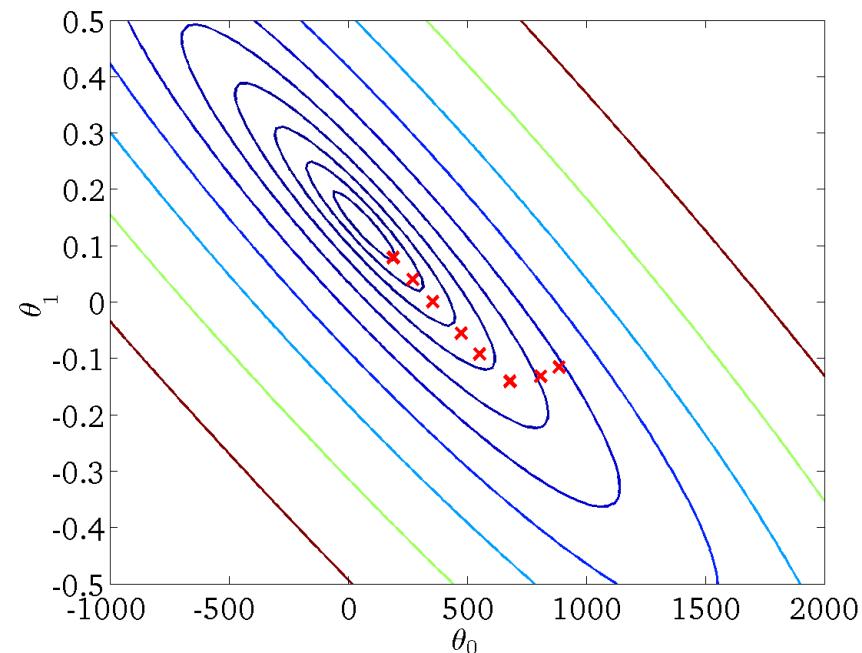
$$h_{\theta}(x)$$

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



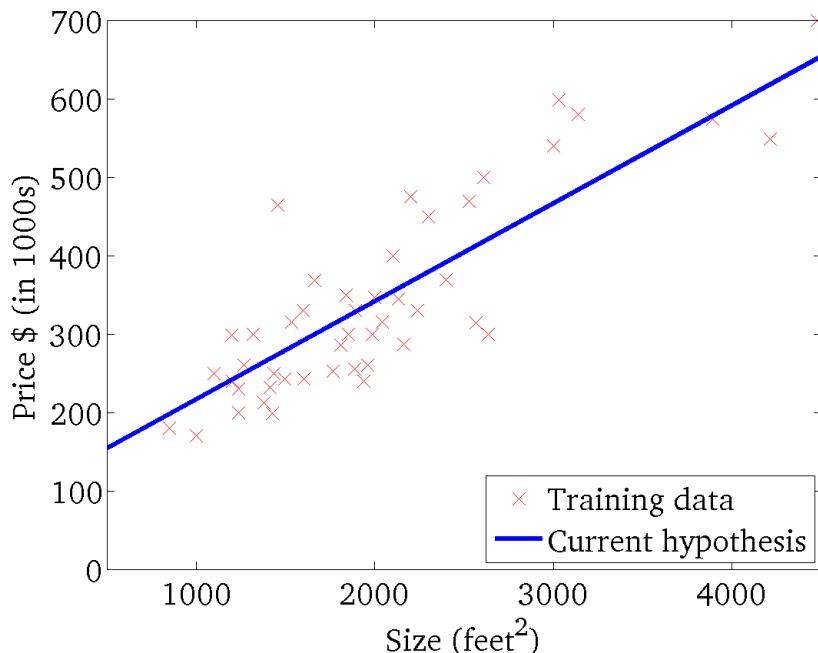
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



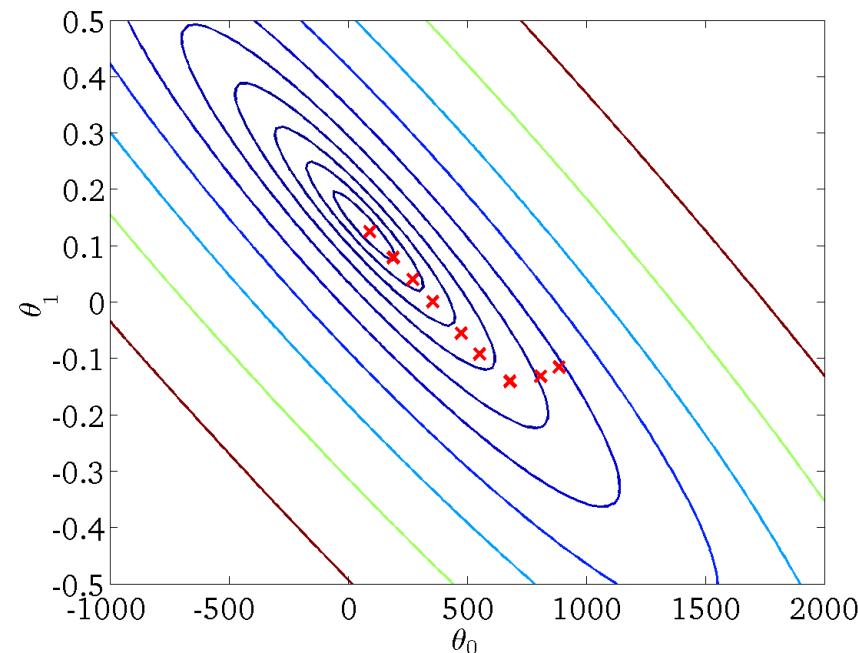
$$h_{\theta}(x)$$

(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)



“Batch” Gradient Descent

“Batch”: Each step of gradient descent uses all the training examples.

- “Batch” Gradient Descent

“Batch”: Each step of gradient descent uses all the training examples.

- “Stochastic” Gradient Descent

“Single”: Each step of gradient descent uses one training example and moves to the next.

03-Linear Regression (Convexity)

Convexity

Definition of convexity

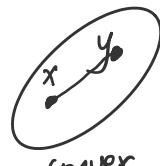
Convex set

Let \mathcal{X} be a vector space, and $S \subseteq \mathcal{X}$

S is a convex set \Leftrightarrow

$$\forall x, y \in S \quad \forall \lambda \in (0, 1)$$

$$\lambda x + (1-\lambda)y \in S$$



convex



Not convex

Convex function

f is a convex function \Leftrightarrow

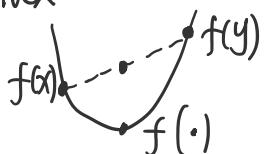
$\text{dom } f$ is a convex set [domain of f]

AND

$$\forall x, y \in \text{dom } f, \quad \forall \lambda \in (0, 1)$$

$$\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$$

convex



not convex



not convex

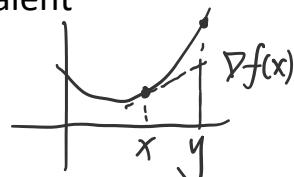


Theorem:

Let f be a twice-differentiable function on a convex open domain $\text{dom } f$.

Then, the following statements are equivalent

a. f is convex



b. (First-order condition)

$$\forall x, y \in \text{dom } f, \quad f(y) \geq f(x) + [\nabla f(x)]^T \cdot (y - x)$$

c. (Second-order condition)

$$\forall \mathbf{x} \in \text{dom } f \quad \nabla^2 f(\mathbf{x}) \succeq 0 \quad \text{positive semi-definite}$$

Calculus review

The **gradient** is the first-order derivatives arranged in the original shape, denoted by $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$ or $\nabla_{\mathbf{x}} f(\mathbf{x})$:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(\mathbf{x}) \\ \frac{\partial}{\partial x_2} f(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_n} f(\mathbf{x}) \end{pmatrix} \in \mathbb{R}^n, \text{ where } \mathbf{x} \in \mathbb{R}^n$$

The **Hessian** is the second-order derivatives, arranged as a matrix denoted by $\mathbf{H}(\mathbf{x})$ or $\nabla^2 f(\mathbf{x})$

$$\mathbf{H}(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \ddots & & \\ \vdots & & \ddots & \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}_{n \times n}$$

Proof: b⇒a; c⇒b in one-dimensional case. Others=homework

(Direction: b⇒a)

$$\forall \mathbf{x}, \mathbf{y} \in \text{dom } f \quad \text{and} \quad \forall \lambda \in (0, 1)$$

$$\text{Def } \mathbf{z} := \lambda \mathbf{x} + (1-\lambda) \mathbf{y}$$

Applying condition b:

$$f(\mathbf{x}) \geq f(\mathbf{z}) + [\nabla f(\mathbf{z})]^T \cdot (\mathbf{x} - \mathbf{z}) \quad (1)$$

$$f(\mathbf{y}) \geq f(\mathbf{z}) + [\nabla f(\mathbf{z})]^T \cdot (\mathbf{y} - \mathbf{z}) \quad (2)$$

$$(1) \cdot \lambda: \quad \lambda f(\mathbf{x}) \geq \lambda f(\mathbf{z}) + \lambda \cdot [\nabla f(\mathbf{z})]^T \cdot (\mathbf{x} - \mathbf{z}) \quad (3)$$

$$(2) \cdot (1-\lambda): \quad (1-\lambda) f(\mathbf{y}) \geq (1-\lambda) f(\mathbf{z}) + (1-\lambda) [\nabla f(\mathbf{z})]^T \cdot (\mathbf{y} - \mathbf{z}) \quad (4)$$

$$(3) + (4):$$

$$\lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}) \geq \lambda f(\mathbf{z}) + (1-\lambda) f(\mathbf{z}) + \lambda [\nabla f(\mathbf{z})]^T \cdot (\mathbf{x} - \mathbf{z}) + (1-\lambda) [\nabla f(\mathbf{z})]^T \cdot (\mathbf{y} - \mathbf{z})$$

$$\begin{aligned} \lambda f(x) + (1-\lambda)f(y) &\geq f(z) + [\nabla f(z)]^T \cdot (\lambda(x-z) + (1-\lambda)(y-z)) \\ &= f(\lambda x + (1-\lambda)y) \quad \begin{matrix} \text{||} \\ \lambda x - \lambda z + y - z - \lambda y + \lambda z \\ \text{||} \\ \lambda x + (1-\lambda)y - z \end{matrix} \end{aligned}$$

(Direction: $\Rightarrow b$ in one-dimensional case)

By Taylor's Theorem:

$$\begin{aligned} \forall x, y \in \text{dom } f, \exists z \in (x, y) \\ f(y) &= f(x) + f'(x)(y-x) + \frac{1}{2} \underbrace{f''(z)}_{\geq 0} \underbrace{(y-x)^2}_{\geq 0} \quad \text{by condition c} \\ \Rightarrow f(y) &\geq f(x) + f'(x)(y-x) \end{aligned} \quad \#$$

Proof sketch for high-dimensional cases:

We consider any 1D line in $\text{dom } f$. If a function cut by any line in the domain is convex, then the function is convex.

Convexity of MSE

First-order derivative

$$\begin{aligned} J(w) &= \frac{1}{2} \sum_{m=1}^M \left(\sum_{k=0}^d w_k \cdot x_k^{(m)} - y^{(m)} \right)^2 \\ \frac{\partial}{\partial w_i} J(w) &= \frac{1}{2} \sum_{m=1}^M \cdot 2 \left(\sum_{k=0}^d w_k \cdot x_k^{(m)} - y^{(m)} \right) \cdot x_i^{(m)} \\ &= \sum_{m=1}^M \left(\sum_{k=0}^d w_k \cdot x_k^{(m)} - y^{(m)} \right) \cdot x_i^{(m)} \end{aligned}$$

From the element-wise derivative, we have

$$\nabla_w J(w) = \underbrace{X^T}_{(d+1) \times M} \underbrace{(Xw - y)}_{M \times 1} \in \mathbb{R}^{d+1}$$

Noticing that $\begin{pmatrix} | & | \\ x_1^{(1)} & \cdots & x_1^{(M)} \\ | & | \\ X^T & & \\ | & | \\ Xw - y & \end{pmatrix}$ has the
implicit summation over M .

The gradient can also be obtained from

$$J(w) = \frac{1}{2} \|Xw - y\|^2$$

by matrix calculus.

Second-order derivative

$$\nabla^2 J(\mathbf{w}) = \mathbf{X}^T \mathbf{X} \succeq 0 \quad (\text{PSD})$$

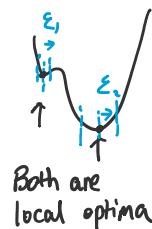
Conclusion: Linear regression is convex in the parameters

Optimality of a convex function

Local optimum

x is a local optimum of f \Leftrightarrow

$\exists \varepsilon > 0$, $\forall z \in \text{dom } f$, if $\|z - x\| < \varepsilon$
then $f(x) \leq f(z)$



Global optimum

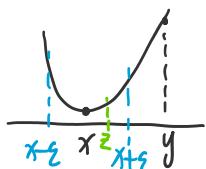
x is a global optimum of f \Leftrightarrow

$\forall z \in \text{dom } f$ $f(x) \leq f(z)$

Theorem: Convex \Rightarrow local optimum is global

Proof. We pick such ε that $\forall z \in N_\varepsilon(x)$, $f(x) \leq f(z)$

ε -neighbor, fancy way of
writing $\|x - z\| < \varepsilon$



If $\|y - x\| < \varepsilon$, we are already happy 😊

If otherwise $\|y - x\| \geq \varepsilon$,

we define $\lambda = \frac{\varepsilon}{2\|y - x\|} \in (0, 1)$

Define $z = (1-\lambda)x + \lambda y$

$$\|z - x\| = \lambda \|x - y\| = \frac{\varepsilon}{2} < \varepsilon \Rightarrow f(x) \leq f(z)$$

By convexity

$$\begin{aligned} (1-\lambda)f(x) + \lambda f(y) &\geq f((1-\lambda)x + \lambda y) \\ &= f(z) \\ &\geq f(x) \end{aligned}$$

Thus $f(y) \geq f(x)$. We are also happy 😊 #

Further reading on convexity analysis in unconstrained scenario:
http://www.princeton.edu/~aaa/Public/Teaching/ORF523/S16/ORF523_S16_Lec7_gh.pdf

Closed-form solution to linear regression

$$\text{Set } \nabla_w J(w) = 0$$

$$X^T X w - X^T y = 0$$

$$w = (X^T X)^{-1} X^T y \text{ is the global optimum}$$

Here, we assume $X^T X$ is invertible. If not use pseudo-inverse.

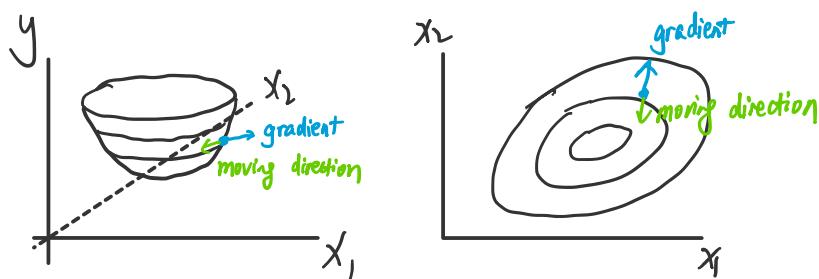
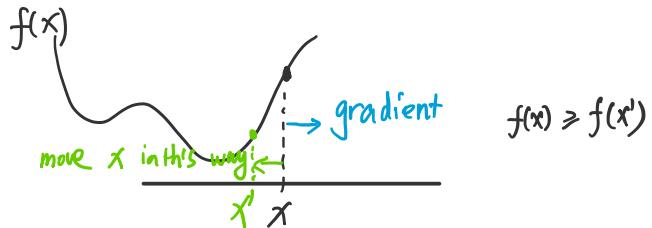
See A.5.4 in https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

Complexity: $\mathcal{O}(n^{2.3}) \sim \mathcal{O}(n^3)$

https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations#endnote_blockinversion

Numerical solution - Gradient descent

If we move a **small** step towards the opposite direction of gradient, Then, the function value decreases.



Iterative update: We compute the gradient and move a small step against the gradient. Repeat the above criteria for many times.

Gradient Descent Algorithm

Initialize $\mathbf{w} = \mathbf{w}^{(0)}$ randomly

Loop over epochs $t = 0, 1, 2, \dots$:

Compute gradient $\nabla J(\mathbf{w}) \Big|_{\mathbf{w}=\mathbf{w}^{(t)}}$

Update parameters

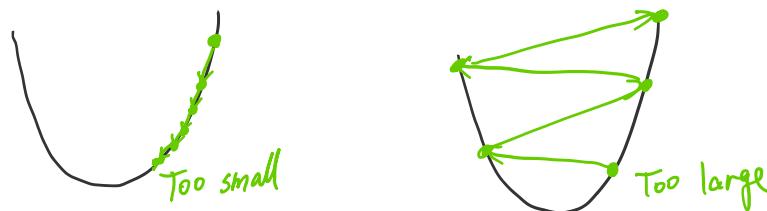
$$\mathbf{w}^{(t+1)} := \mathbf{w} - \alpha \cdot \nabla J(\mathbf{w}) \Big|_{\mathbf{w}=\mathbf{w}^{(t)}}$$

Until stopping criterion satisfies

Comments:

- Initialization of $\mathbf{w}^{(0)}$:
 - Convex: Doesn't matter that much.
 - Non-convex: small random values around the origin $\mathbf{0}$
- Iteration over data
 - One epoch means one iteration over all data samples
 - Stopping criteria
 - Set MaxIter due to budget constraint
 - Until the convergence of J , i.e.,

$$|J(\mathbf{w}^{(t)}) - J(\mathbf{w}^{(t-1)})| < \text{threshold}$$
 - Early stop by validation (TBD in future lectures)
- α is the learning rate
 - Theoretical result: Gradient descent guarantees to converge with some assumptions: <https://www.stat.cmu.edu/~ryantibs/convexopt-F13/scribes/lec6.pdf>
 - In practice:
 - Too small \Rightarrow slow, and performance may not be very good
 - Too large \Rightarrow overshoot



- Could be tuned in a multiplicative way, e.g.,
..., 0.01, 0.03, 0.1, 0.3, 1, 3, ...
- Learning rate decay
 - Start with a relatively large α
 - Decrease α during training
- Recent advances: Adaptive learning rate for different parameters

- Gradient

- Full-batch gradient descent

$$J = \frac{1}{M} \sum_{i=1}^M J^{(i)}$$

- Mini-batch gradient descent

In each iteration in the loop,

$$J = \frac{1}{B} \sum_{b=1}^B J^{(n_b)}$$

where $B < N$, and $n_1, n_2, \dots, n_b \in \{x_1, \dots, x_M\}$,

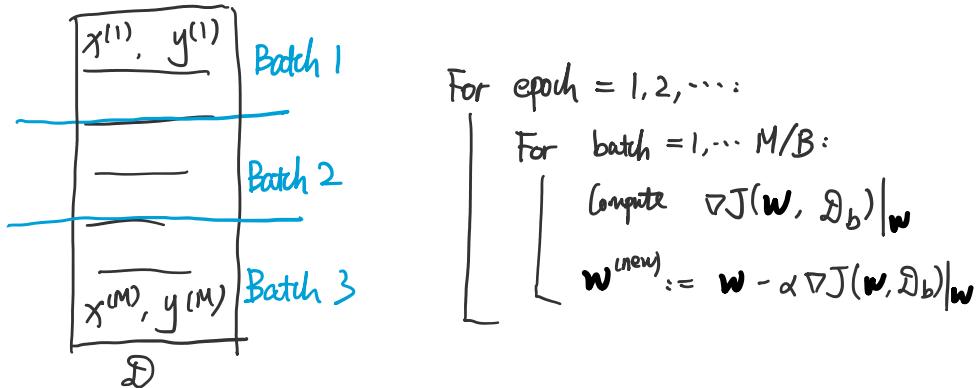
- Stochastic gradient descent (SGD)

Mini-batch with $B = 1$

Convergence: <https://www.cs.ubc.ca/~schmidtm/Courses/540-W19/L11.pdf>

Essentially, a batch is a sampled subset of the entire dataset. Two settings:

- Sample with replacement
- Sample without replacement: In this case, all batches form a partition of the dataset



Pros and Cons

	Full batch	SGD ($B = 1$)
Pros	Exact gradient	Gradient could be noisy
Cons	Slow in each param. update	Efficient for each update

Mini-batch gradient descent is a balance. Besides efficiency concerns, evidence shows that small batch gradient descent is better (yields more generalizable solutions) than full batch. The theoretical reasons are not completely clear.

Comparing closed-form solution and numerical solution

- For linear regression:
 - Closed-form solution converges better (guaranteed global optimum) and is faster if the number of sample is not too large.
 - Numerical solution works regardless of the existence of matrix inverse. It is more efficient if we have too many samples.
- For other optimization/computation:
 - Having a closed-form solution is rare, but should always be considered before trying numerical methods. In gradient descent, for example, we compute the gradient by closed-form formula (although the optimum is not closed-form), instead of numerical gradient computing.
 - Numerical methods are more flexible. Gradient descent works for not only convex problems, but also non-convex ones. In non-convex gradient descent, we may be stuck at a local optimum. Restarting gradient descent by different initial parameters may alleviate this problem.



CSE 4621

Machine Learning

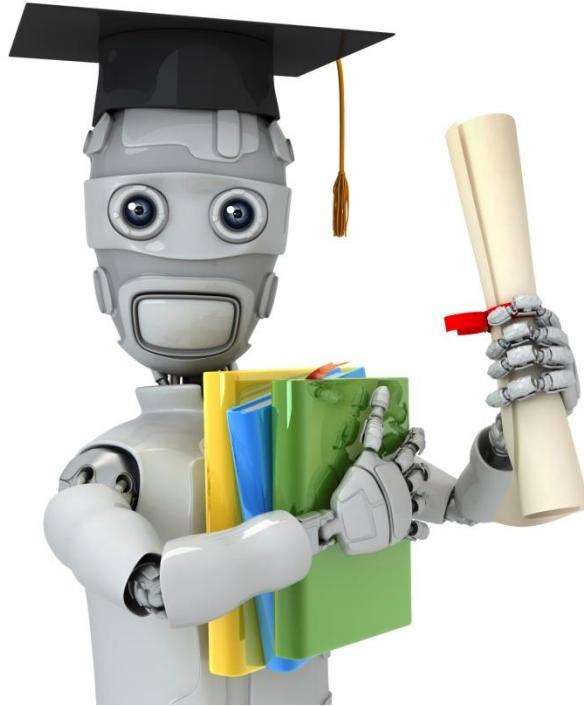
Lecture 4

Md. Hasanul Kabir, PhD.

Professor, CSE Department

Islamic University of Technology (IUT)





Machine Learning

Linear Regression with multiple variables

Multiple features

Source & Special Thanks to Andrew Ng (Coursera) Machine Learning Course

Multiple features (variables).

Size (feet ²)	Price (\$1000)
x	y
2104	460
1416	232
1534	315
852	178
...	...

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

Multiple features (variables).

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

Notation:

n = number of features

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

Hypothesis:

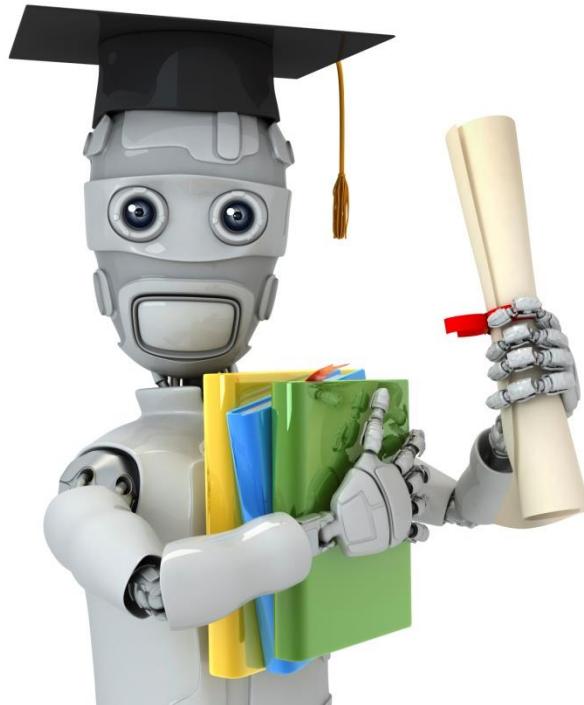
Previously: $h_{\theta}(x) = \theta_0 + \theta_1 x$

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

Multivariate linear regression.

1 * (n+1)



Machine Learning

Linear Regression with multiple variables

Gradient descent for 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: $\theta_0, \theta_1, \dots, \theta_n$

Cost function:

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

Gradient descent:

Repeat {

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

}

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

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
 $j = 0, \dots, n$)

}

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

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

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

...



Machine Learning

Linear Regression with multiple variables

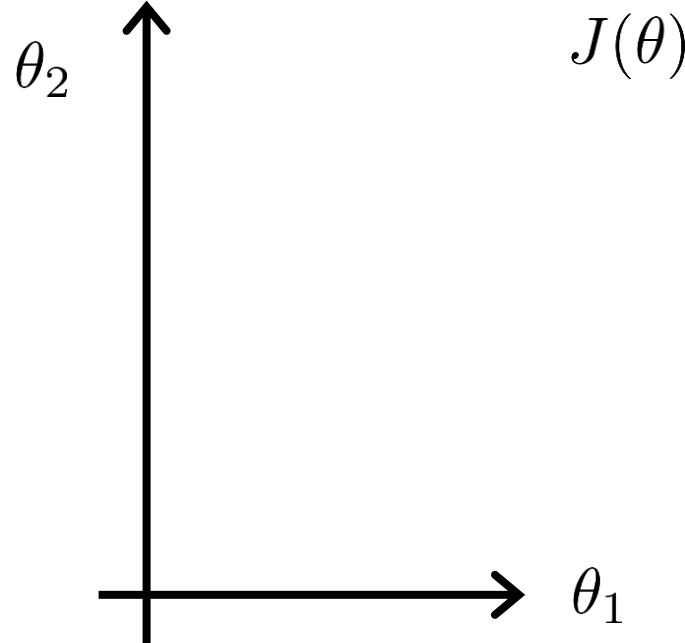
Gradient descent in practice I: Feature Scaling

Feature Scaling

Idea: Make sure features are on a similar scale.

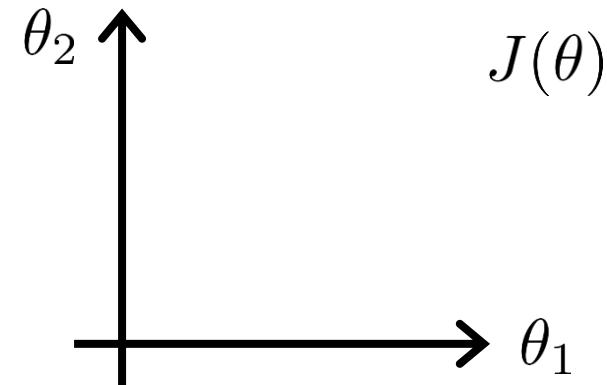
E.g. x_1 = size (0-2000 feet²)

x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2)}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Feature Scaling

Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

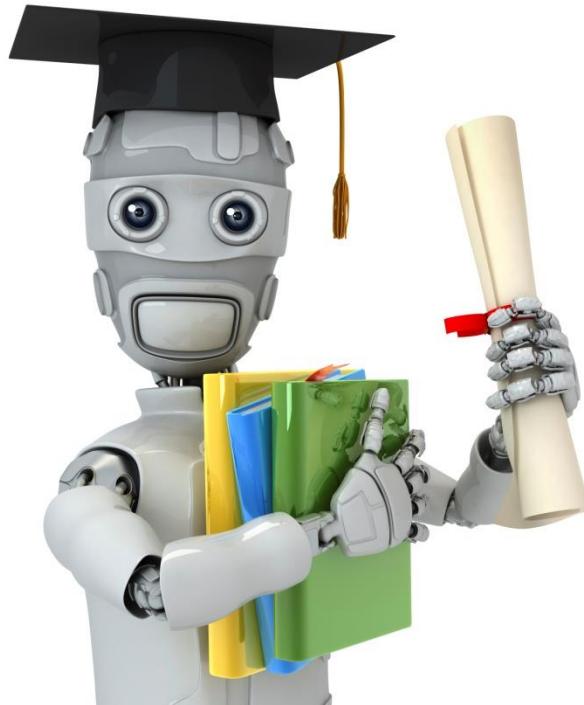
Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean
(Do not apply to $x_0 = 1$).

E.g. $x_1 = \frac{\text{size} - 1000}{2000}$

$$x_2 = \frac{\#\text{bedrooms} - 2}{5}$$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$



Machine Learning

Linear Regression with multiple variables

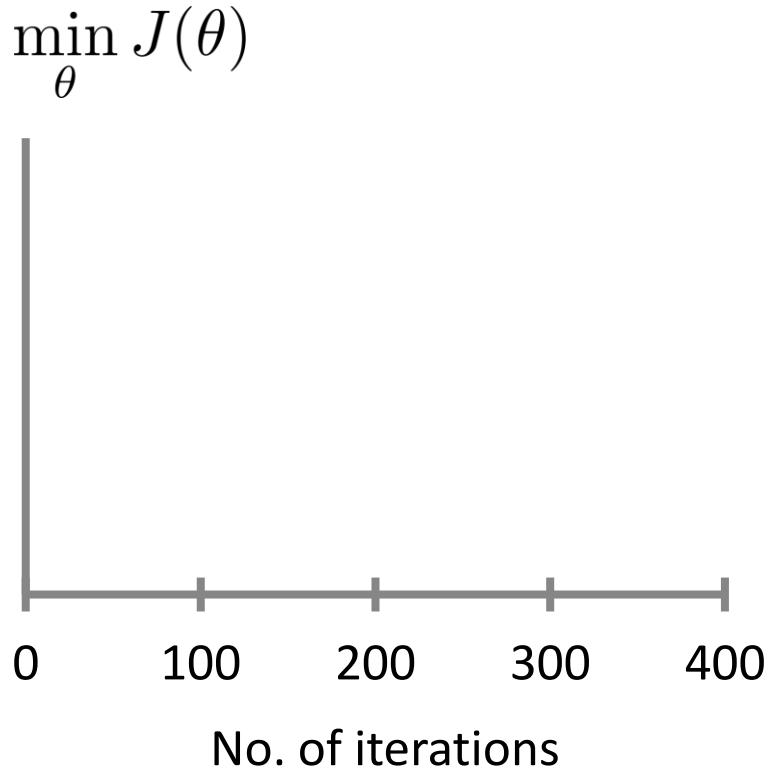
Gradient descent in practice II: Learning rate

Gradient descent

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

- “Debugging”: How to make sure gradient descent is working correctly.
- How to choose learning rate α .

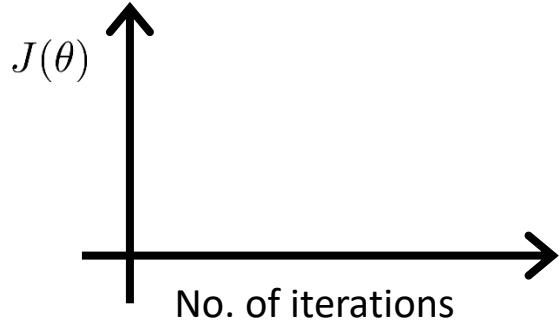
Making sure gradient descent is working correctly.



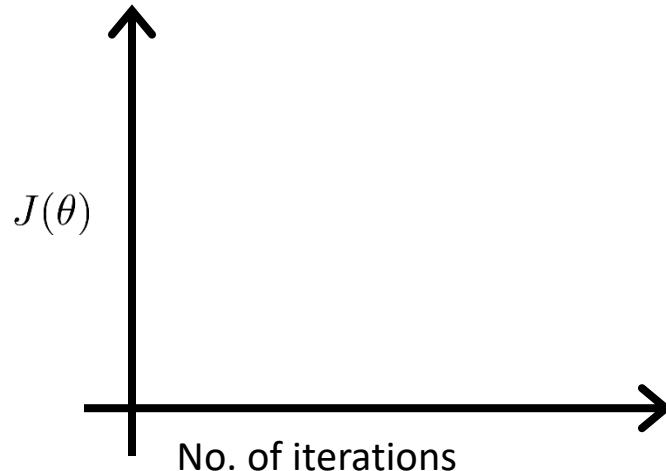
Example automatic convergence test:

Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

Making sure gradient descent is working correctly.



Gradient descent not working.
Use smaller α .



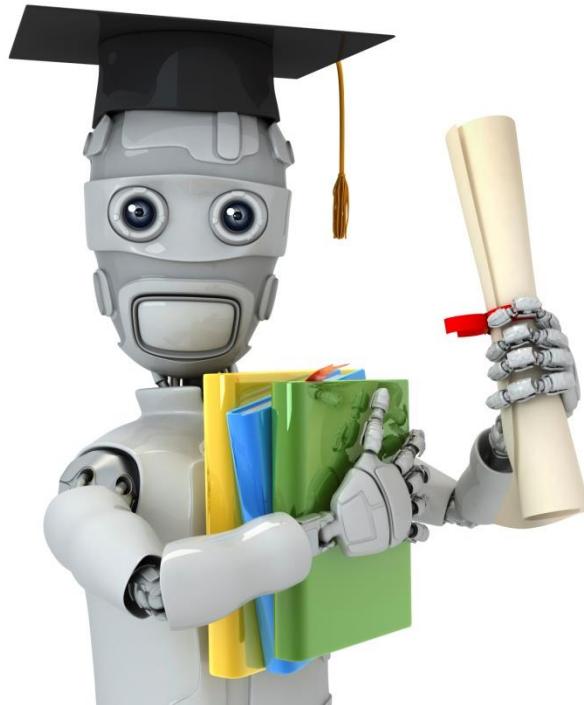
- For sufficiently small α , $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Summary:

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

To choose α , try

$$\dots, 0.001, \quad , 0.01, \quad , 0.1, \quad , 1, \dots$$



Machine Learning

Linear Regression with multiple variables

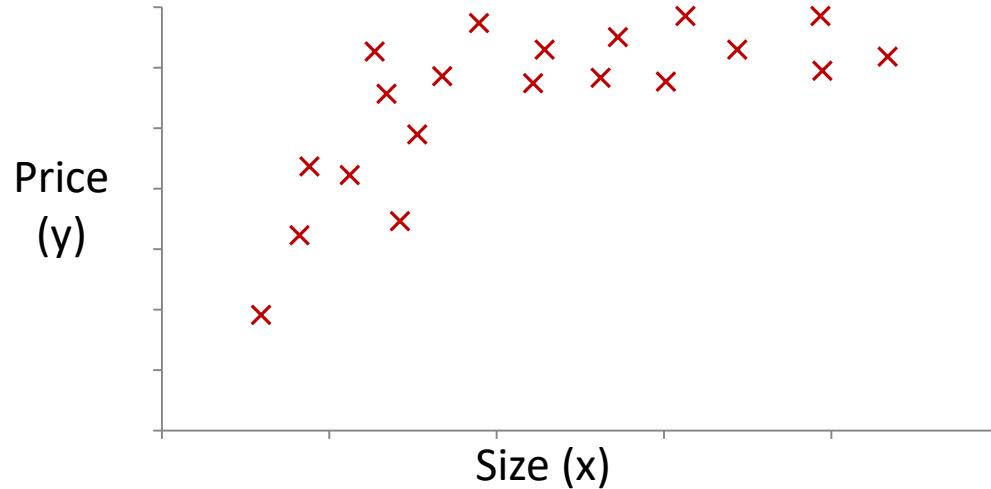
Features and
polynomial regression

Housing prices prediction

$$h_{\theta}(x) = \theta_0 + \theta_1 \times \text{frontage} + \theta_2 \times \text{depth}$$



Polynomial regression



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

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

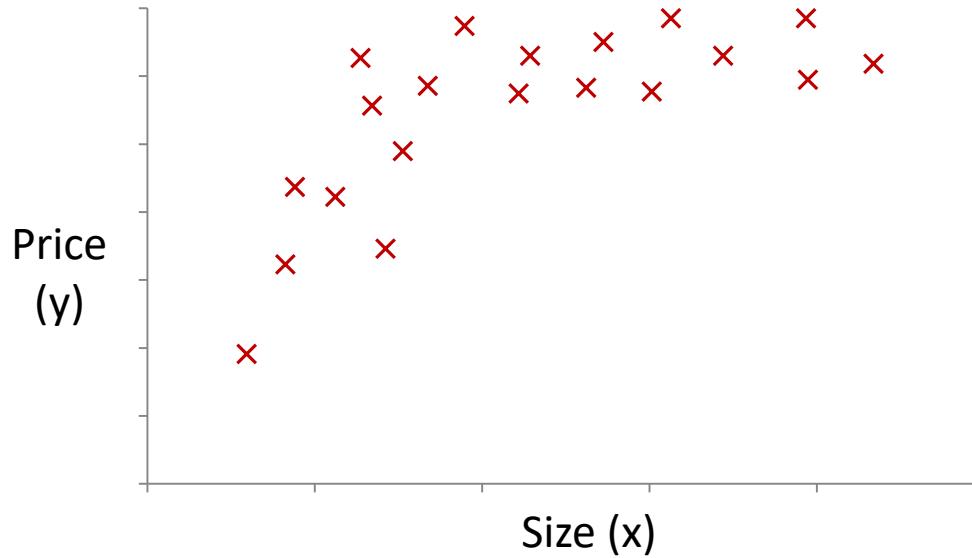
$$\begin{aligned}h_{\theta}(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\&= \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2 + \theta_3(\text{size})^3\end{aligned}$$

$$x_1 = (\text{size})$$

$$x_2 = (\text{size})^2$$

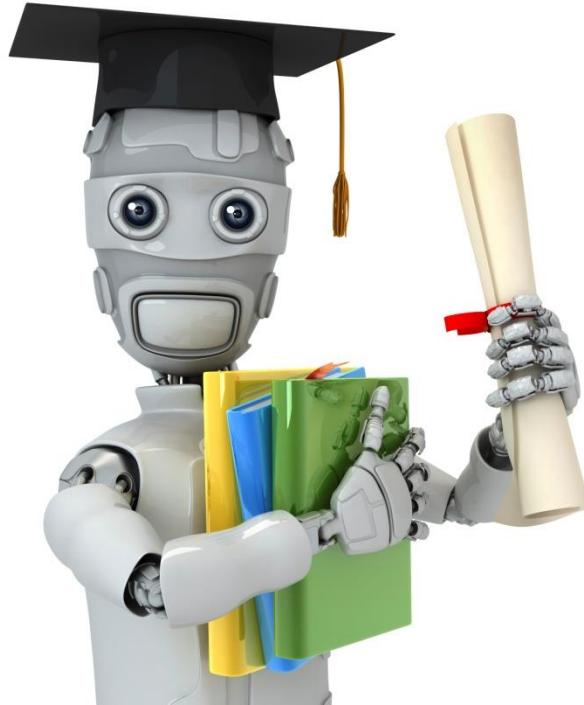
$$x_3 = (\text{size})^3$$

Choice of features



$$h_{\theta}(x) = \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2$$

$$h_{\theta}(x) = \theta_0 + \theta_1(\text{size}) + \theta_2 \sqrt{(\text{size})}$$

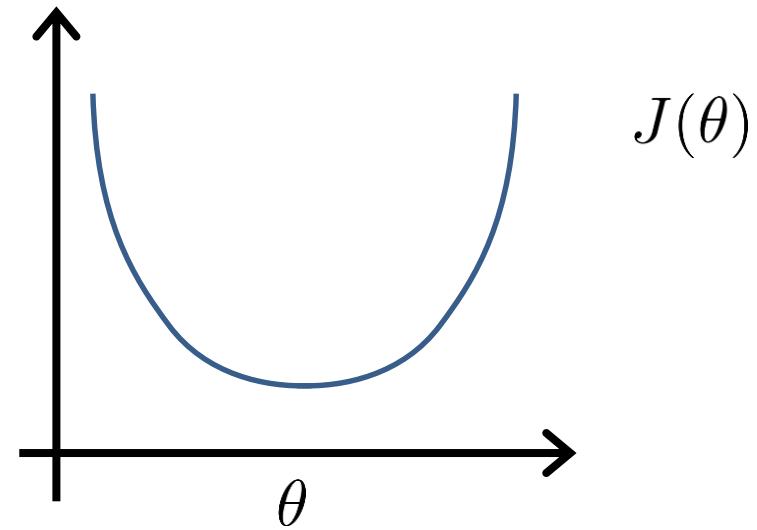


Machine Learning

Linear Regression with multiple variables

Normal equation

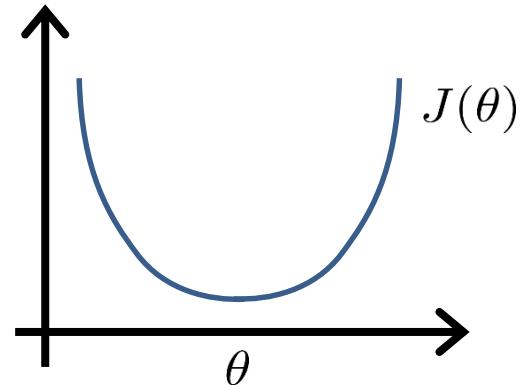
Gradient Descent



Normal equation: Method to solve for θ analytically.

Intuition: If 1D ($\theta \in \mathbb{R}$)

$$J(\theta) = a\theta^2 + b\theta + c$$



$$\theta \in \mathbb{R}^{n+1} \qquad J(\theta_0, \theta_1, \dots, \theta_m) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \dots = 0 \quad (\text{for every } j)$$

Solve for $\theta_0, \theta_1, \dots, \theta_n$

Examples: $m = 4$.

	Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

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

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

m **examples** $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$; *n* **features.**

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

E.g. If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

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

$(X^T X)^{-1}$ is inverse of matrix $X^T X$.

Octave: **pinv (x' *x) *x' *y**

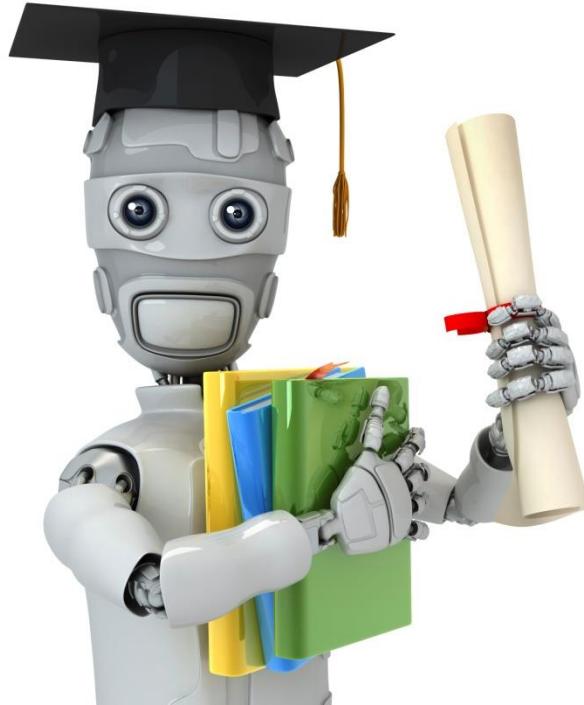
m training examples, n features.

Gradient Descent

- Need to choose α .
- Needs many iterations.
- Works well even when n is large.

Normal Equation

- No need to choose α .
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$
- Slow if n is very large.



Machine Learning

Linear Regression with multiple variables

Normal equation
and non-invertibility
(optional)

Normal equation

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

- What if $X^T X$ is non-invertible? (singular/
degenerate)
- Octave: `pinv(X' *X) *X' *y`

What if $X^T X$ is non-invertible?

- Redundant features (linearly dependent).
E.g. $x_1 = \text{size in feet}^2$
 $x_2 = \text{size in m}^2$
- Too many features (e.g. $m \leq n$).
 - Delete some features, or use regularization.



CSE 4621

Machine Learning

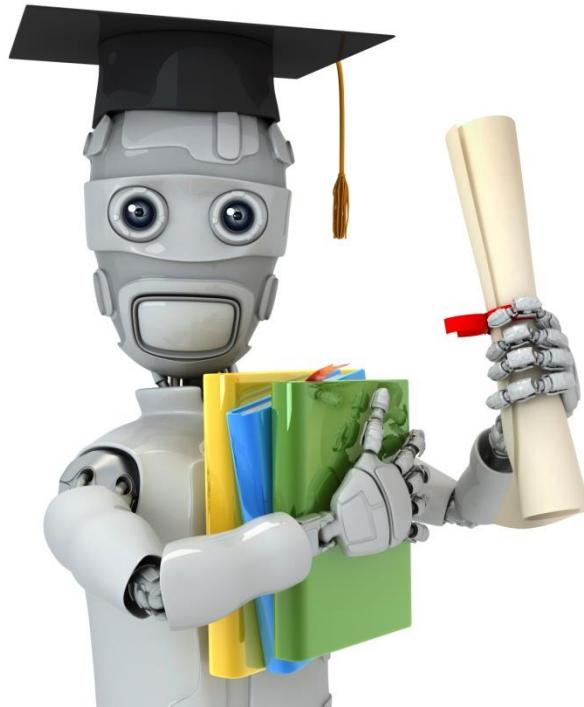
Lecture 5

Md. Hasanul Kabir, PhD.

Professor, CSE Department

Islamic University of Technology (IUT)





Logistic Regression

Classification

Machine Learning

Source & Special Thanks to Andrew Ng (Coursera) Machine Learning Course

Classification

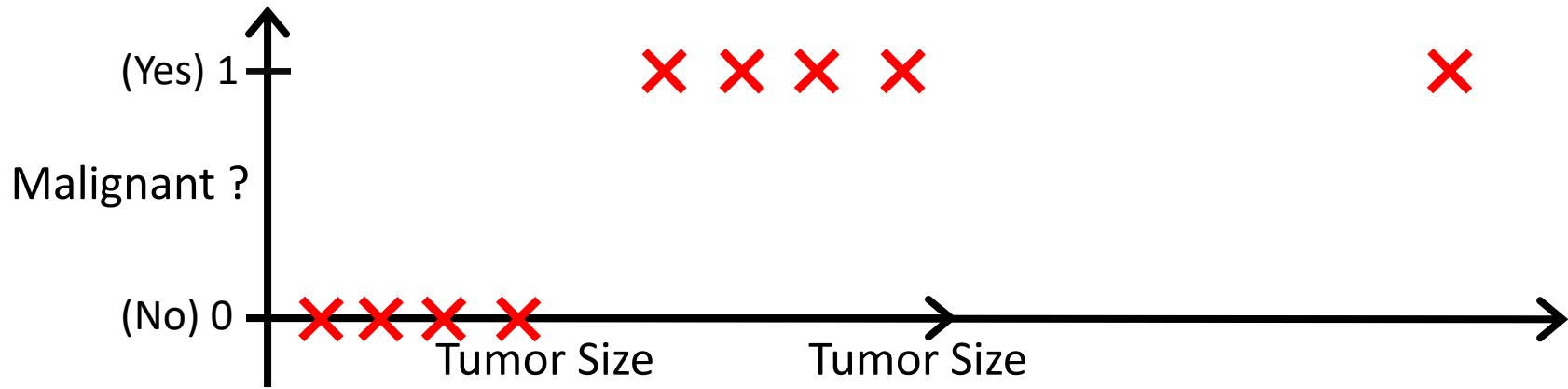
Email: Spam / Not Spam?

Online Transactions: Fraudulent (Yes / No)?

Tumor: Malignant / Benign ?

$y \in \{0, 1\}$

0: “Negative Class” (e.g., benign tumor)
1: “Positive Class” (e.g., malignant tumor)



Threshold classifier output $h_{\theta}(x)$ at 0.5:

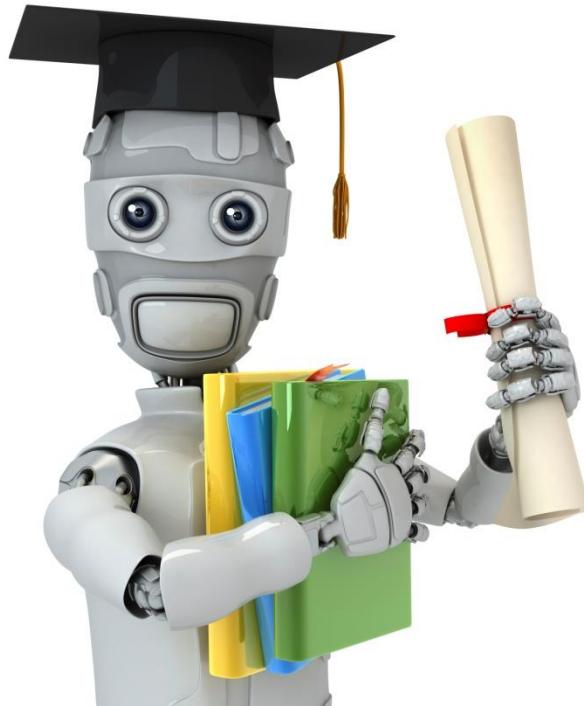
If $h_{\theta}(x) \geq 0.5$, predict “y = 1”

If $h_{\theta}(x) < 0.5$, predict “y = 0”

Classification: $y = 0$ or 1

$h_\theta(x)$ can be > 1 or < 0

Logistic Regression: $0 \leq h_\theta(x) \leq 1$



Machine Learning

Logistic Regression

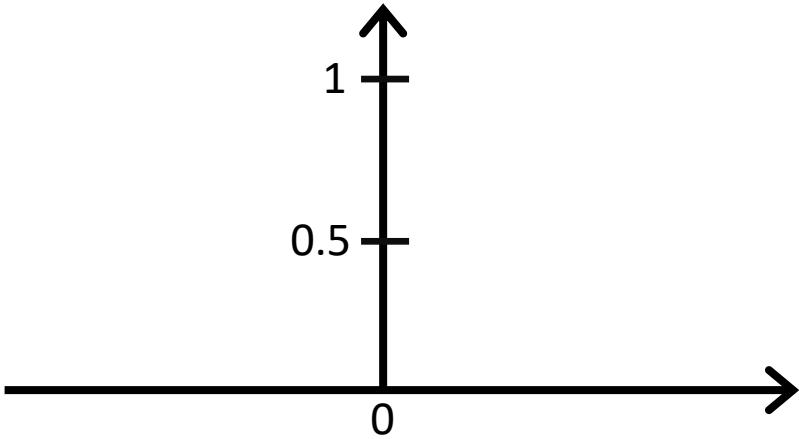
Hypothesis Representation

Logistic Regression Model

Want $0 \leq h_\theta(x) \leq 1$

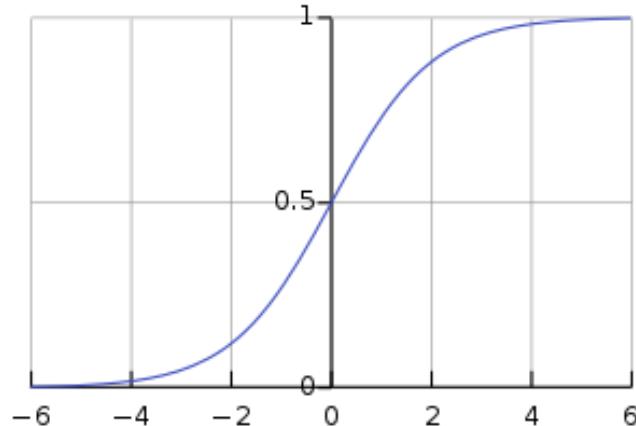
$$h_\theta(x) = \theta^T x$$

Sigmoid function
Logistic function

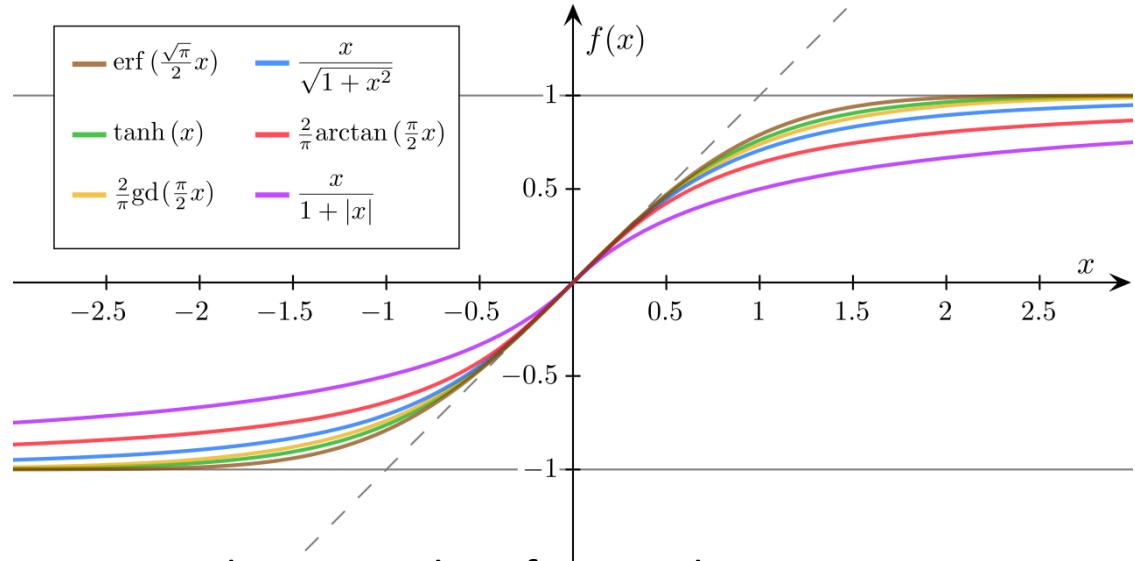


Sigmoid Functions

- Having a characteristic "S"-shaped curve or **sigmoid curve**.



Logistic Function



Other Examples of Sigmoid Functions

Interpretation of Hypothesis Output

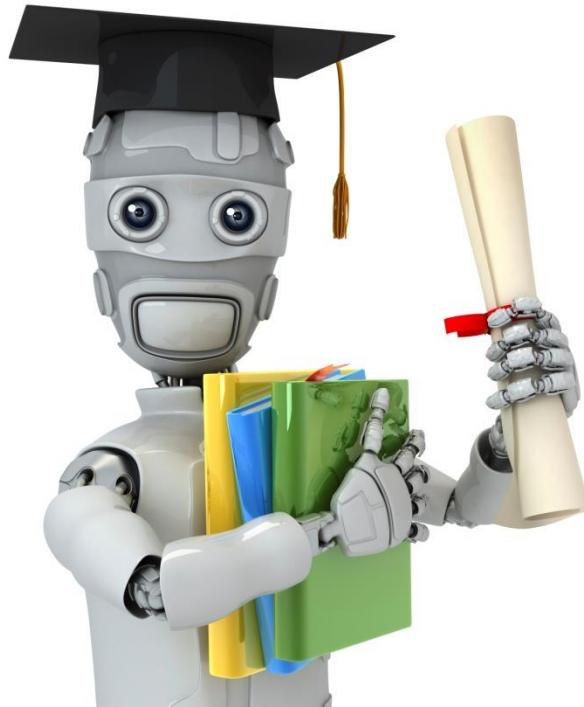
$h_{\theta}(x)$ = estimated probability that $y = 1$ on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$
 $h_{\theta}(x) = 0.7$

Tell patient that 70% chance of tumor being malignant

“probability that $y = 1$, given x ,
parameterized by θ ”

$$P(y = 0|x; \theta) + P(y = 1|x; \theta) = 1$$
$$P(y = 0|x; \theta) = 1 - P(y = 1|x; \theta)$$



Machine Learning

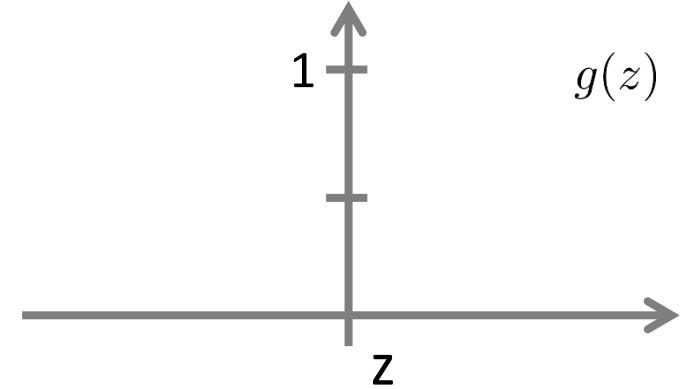
Logistic Regression

Decision boundary

Logistic regression

$$h_{\theta}(x) = g(\theta^T x)$$

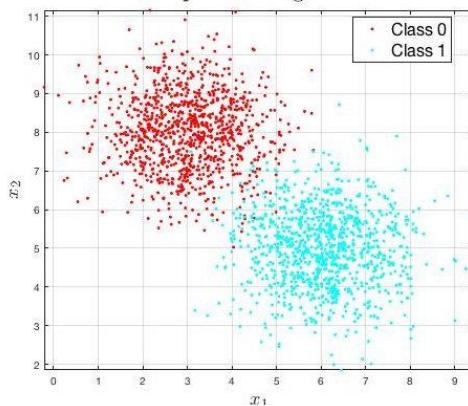
$$g(z) = \frac{1}{1+e^{-z}}$$



Suppose predict “ $y = 1$ “ if $h_{\theta}(x) \geq 0.5$

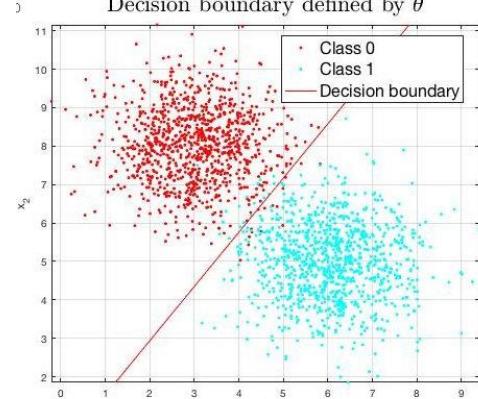
predict “ $y = 0$ “ if $h_{\theta}(x) < 0.5$

Input training data

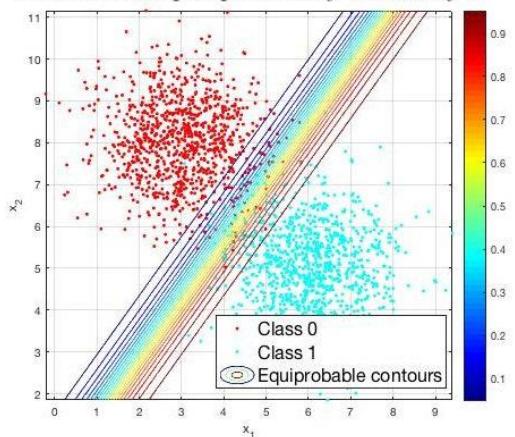


Logistic Function

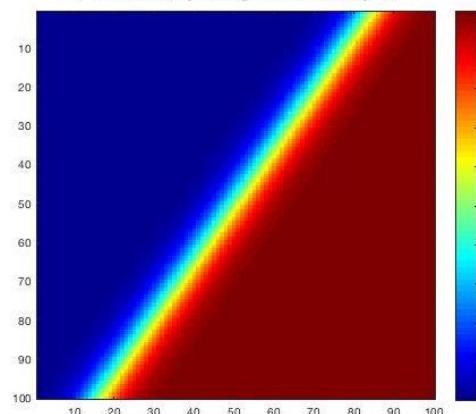
Decision boundary defined by θ



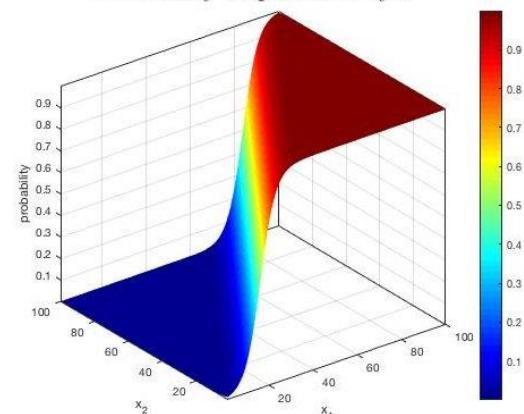
Contours of equal probability defined by θ



Probability map defined by θ



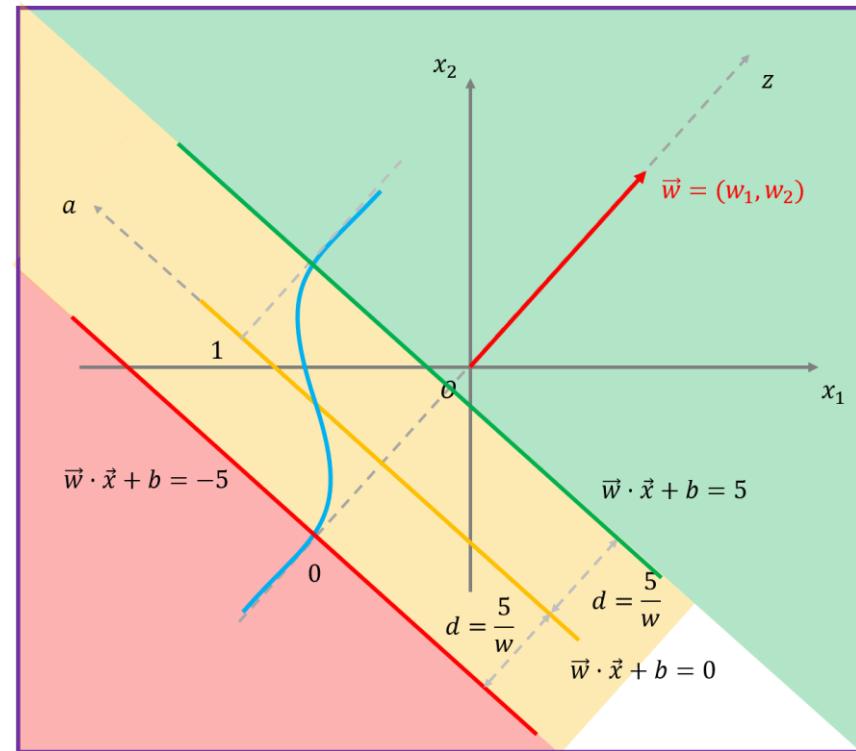
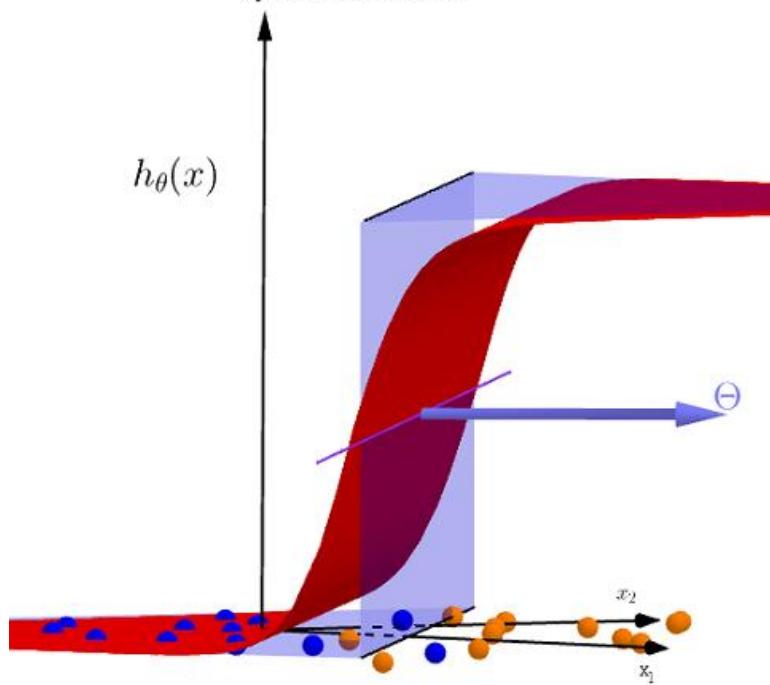
Probability map defined by θ



Logistic Function

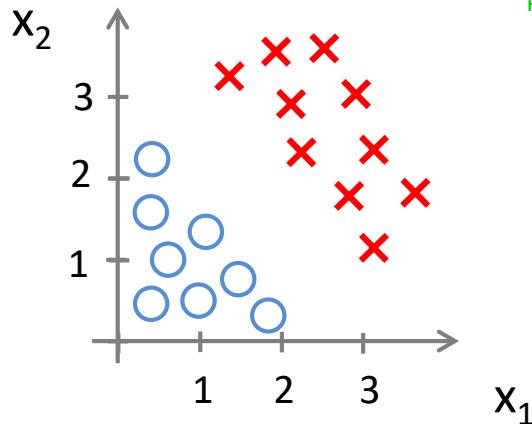
Decision Boundary Hyperplane

Sigmoid activation function



How to fit (find) Parameter θ

Decision Boundary

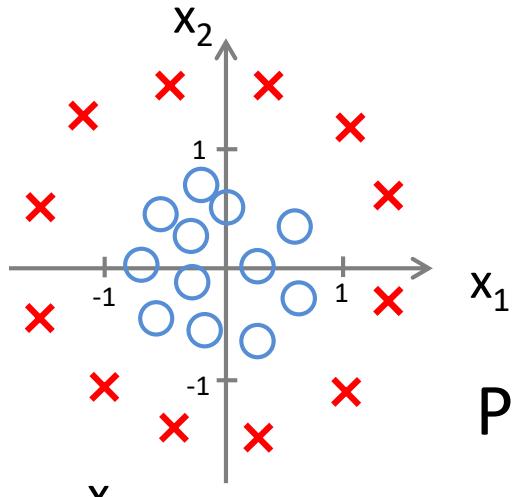


Parameter θ ($\theta_0, \theta_1, \theta_2$) defines the decision boundary
not the training set. Training set may be used to find the
Parameter θ

$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

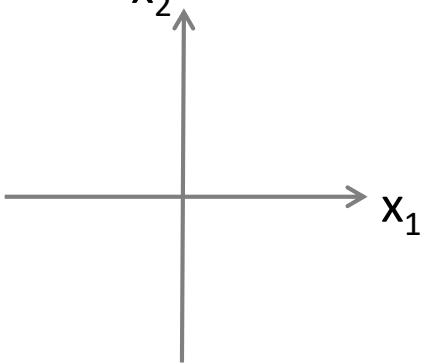
Predict “ $y = 1$ ” if $-3 + x_1 + x_2 \geq 0$

Non-linear decision boundaries

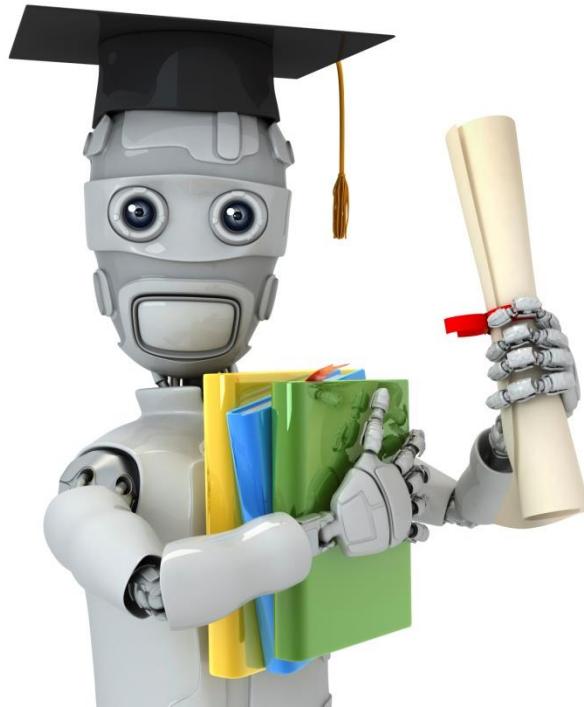


$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Predict " $y = 1$ " if $-1 + x_1^2 + x_2^2 \geq 0$



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$



Machine Learning

Logistic Regression

Cost function

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

m examples

$$x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$$

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

How to choose parameters θ ?

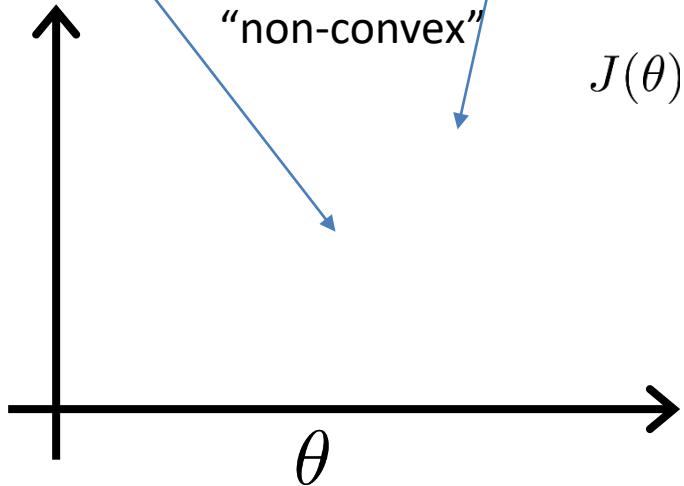
Cost function

$J(\theta)$ is non-linear because of the presence of non-linear sigmoid function

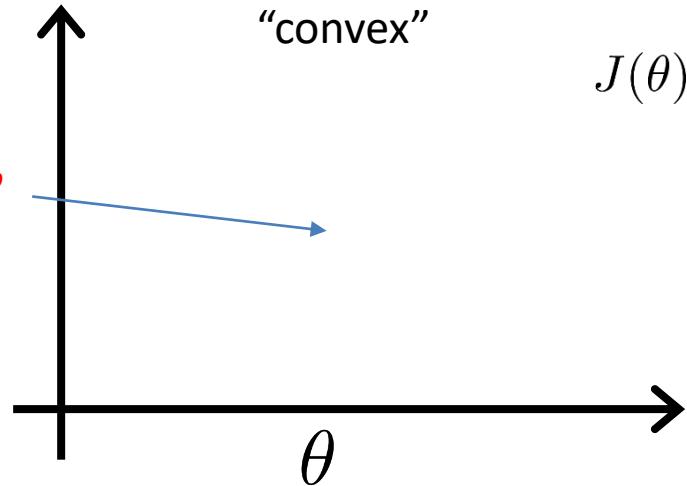
Linear regression:
$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Non-Linear Function

$$\text{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$$



We want $J(\theta)$ to behave like this



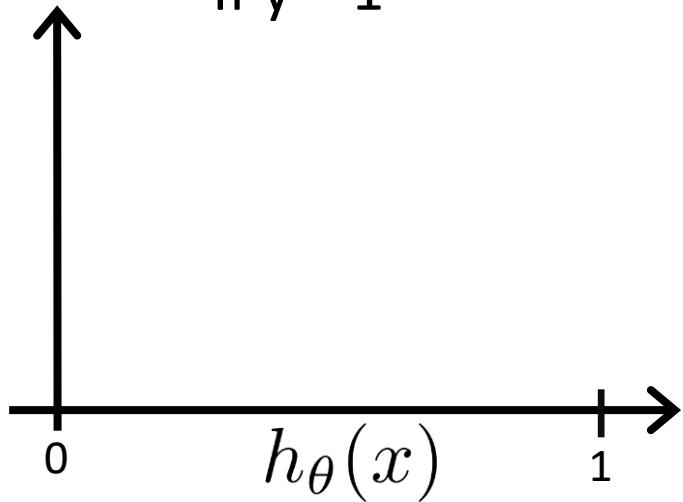
Logistic regression cost function

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

Different
Cost Function

If $y = 1$

Cost = 0 if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \rightarrow 0$
 $Cost \rightarrow \infty$

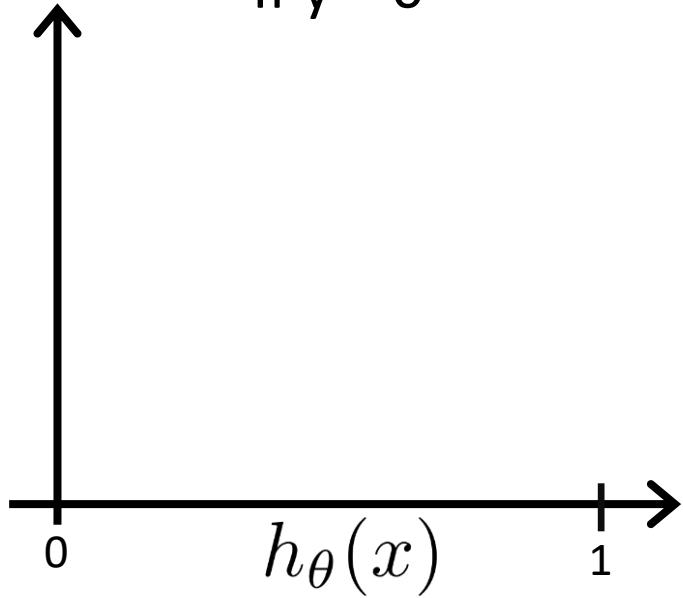


Captures intuition that if $h_\theta(x) = 0$,
(predict $P(y = 1|x; \theta) = 0$), but $y = 1$,
we'll penalize learning algorithm by a very
large cost.

Logistic regression cost function

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

If $y = 0$



In logistic regression, the cost function for our hypothesis outputting (predicting) $h_{\theta}(x)$ on a training example that has label $y \in \{0, 1\}$ is:

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

Which of the following are true? Check all that apply.

If $h_{\theta}(x) = y$, then $\text{cost}(h_{\theta}(x), y) = 0$ (for $y = 0$ and $y = 1$).

Well done!

If $y = 0$, then $\text{cost}(h_{\theta}(x), y) \rightarrow \infty$ as $h_{\theta}(x) \rightarrow 1$.

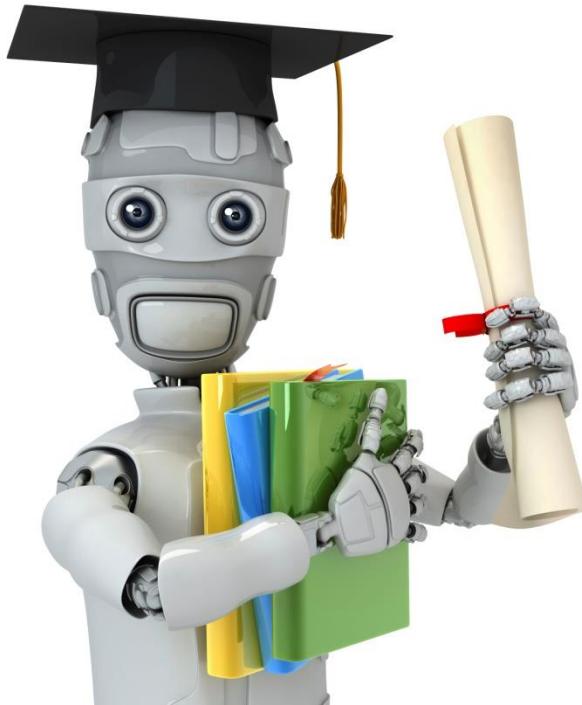
Well done!

If $y = 0$, then $\text{cost}(h_{\theta}(x), y) \rightarrow \infty$ as $h_{\theta}(x) \rightarrow 0$.

Well done!

Regardless of whether $y = 0$ or $y = 1$, if $h_{\theta}(x) = 0.5$, then $\text{cost}(h_{\theta}(x), y) > 0$.

Well done!



Machine Learning

Logistic Regression

Simplified cost function
and gradient descent

Logistic regression cost function

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

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

Note: $y = 0$ or 1 always

Logistic regression cost function

$x^{(i)}$ = input (features) of i^{th} training example.

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

$x_j^{(i)}$ = value of feature j in i^{th} training example

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

To fit parameters θ :

$$\min_{\theta} J(\theta)$$

To make a prediction given new x :

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

Gradient Descent

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

Want $\min_{\theta} J(\theta)$:

Repeat {

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

}

(simultaneously update all θ_j)

Gradient Descent

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

Want $\min_{\theta} J(\theta)$:

Repeat {

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

(simultaneously update all θ_j)

}

Algorithm looks identical to linear regression!

Optimization algorithm

Cost function $J(\theta)$. Want $\min_{\theta} J(\theta)$.

Given θ , we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \dots, n$)

Gradient descent:

Repeat {

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

}

Optimization algorithm

Given θ , we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \dots, n$)

Optimization algorithms:

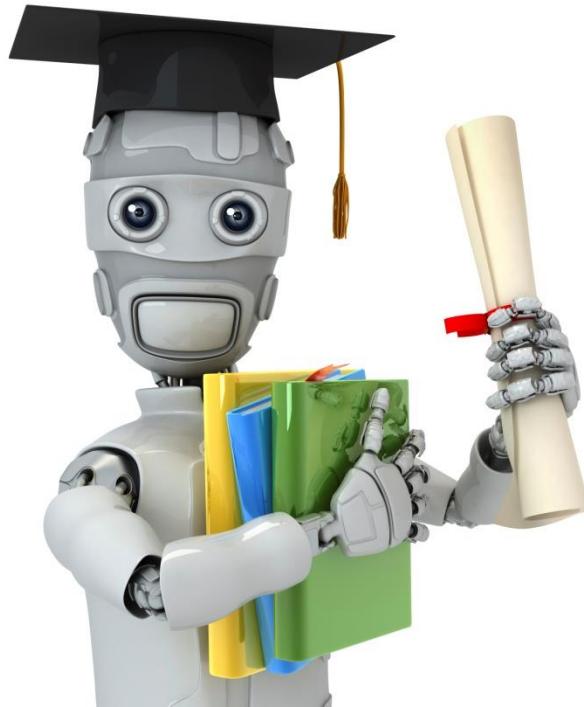
- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

- More complex



Machine Learning

Logistic Regression

Multi-class classification:
One-vs-all

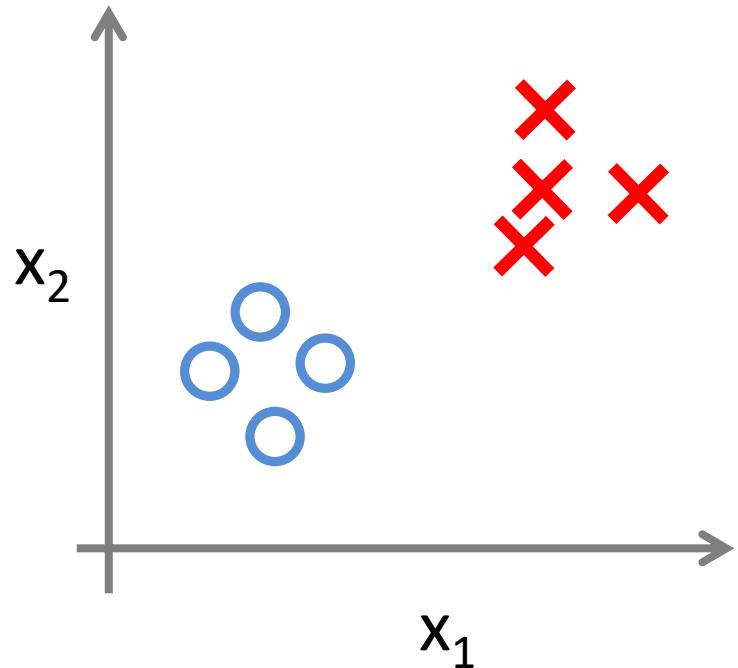
Multiclass classification

Email foldering/tagging: Work, Friends, Family, Hobby

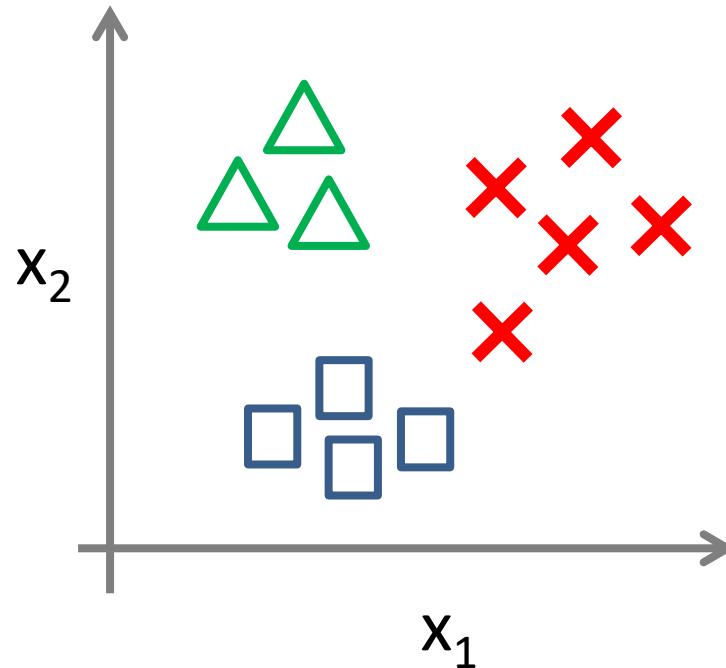
Medical Diagnosis: Not ill, Cold, Flu

Weather State Prediction: Sunny, Cloudy, Rain, Snow

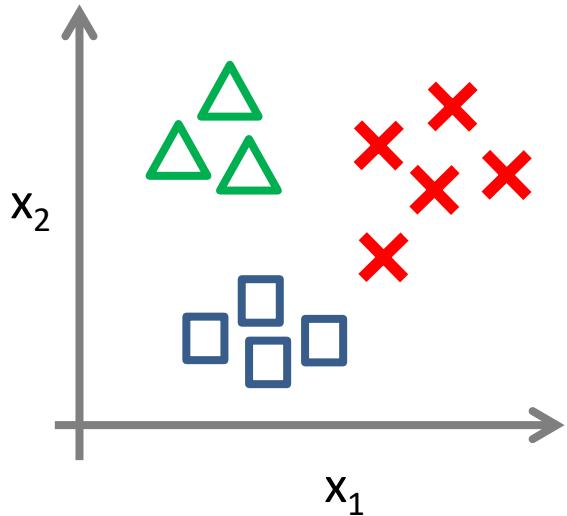
Binary classification:



Multi-class classification:



One-vs-all (one-vs-rest):

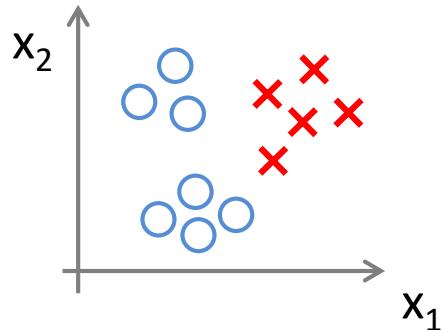
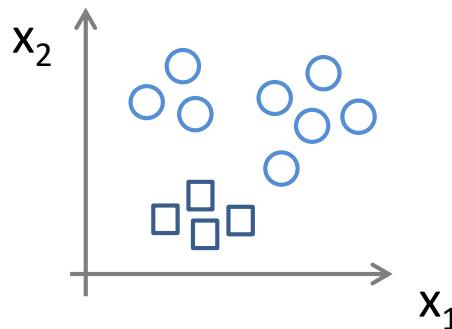
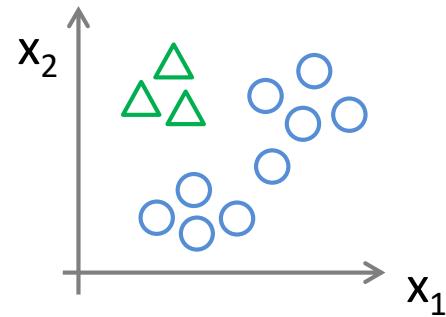
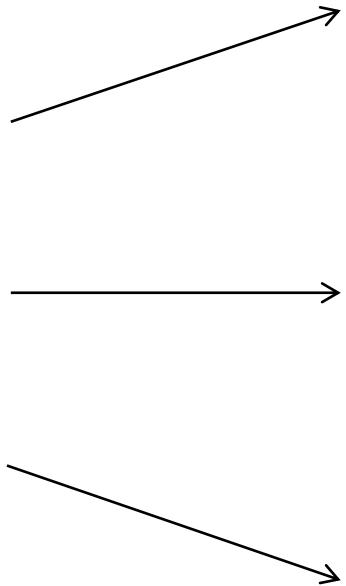


Class 1:

Class 2:

Class 3:

$$h_{\theta}^{(i)}(x) = P(y = i|x; \theta) \quad (i = 1, 2, 3)$$



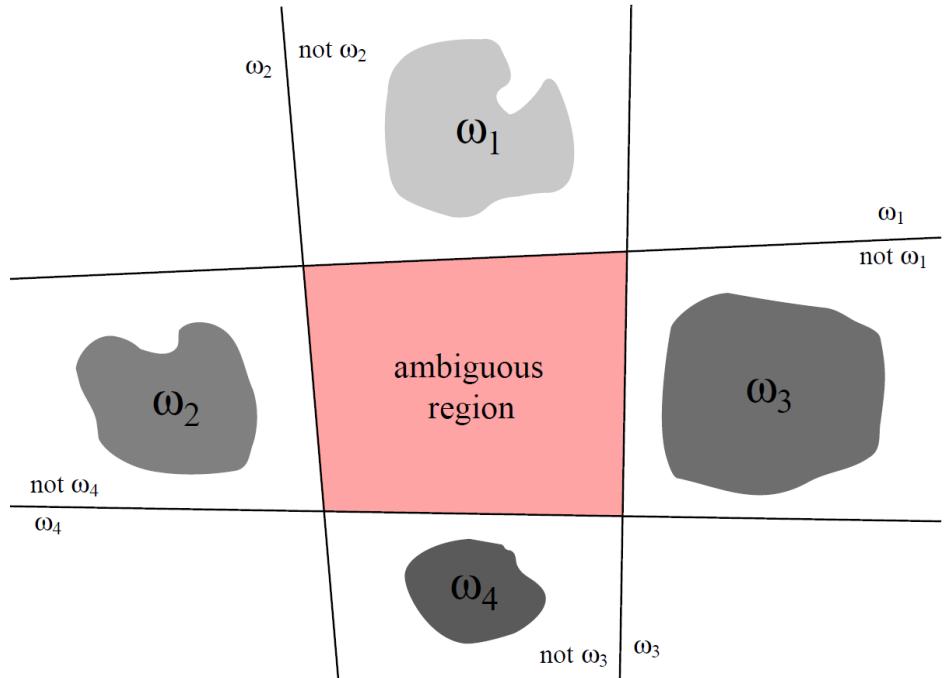
One-vs-all

Train a logistic regression classifier $h_{\theta}^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x , to make a prediction, pick the class i that maximizes

$$\max_i h_{\theta}^{(i)}(x)$$

One-vs-All



One-vs-One

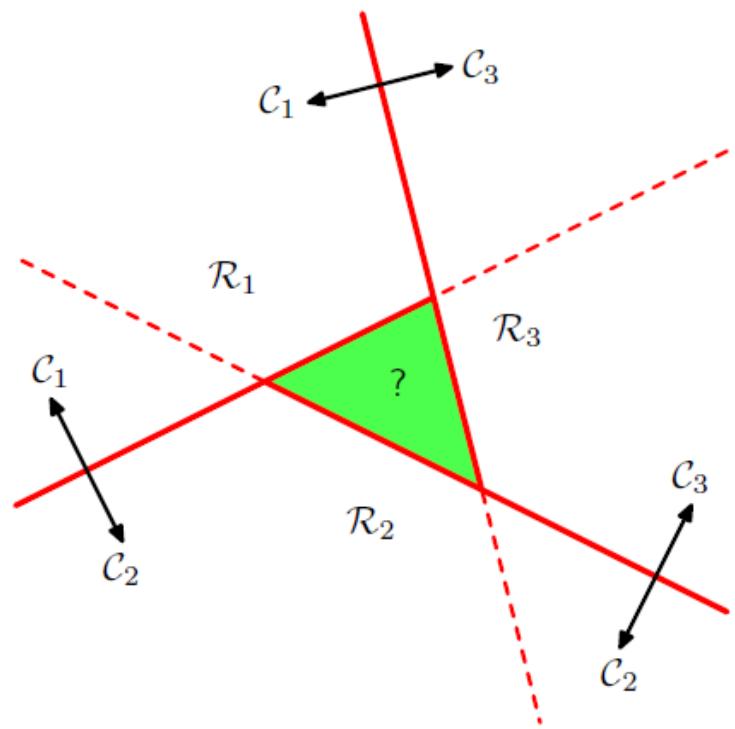


Fig: 3-class problem

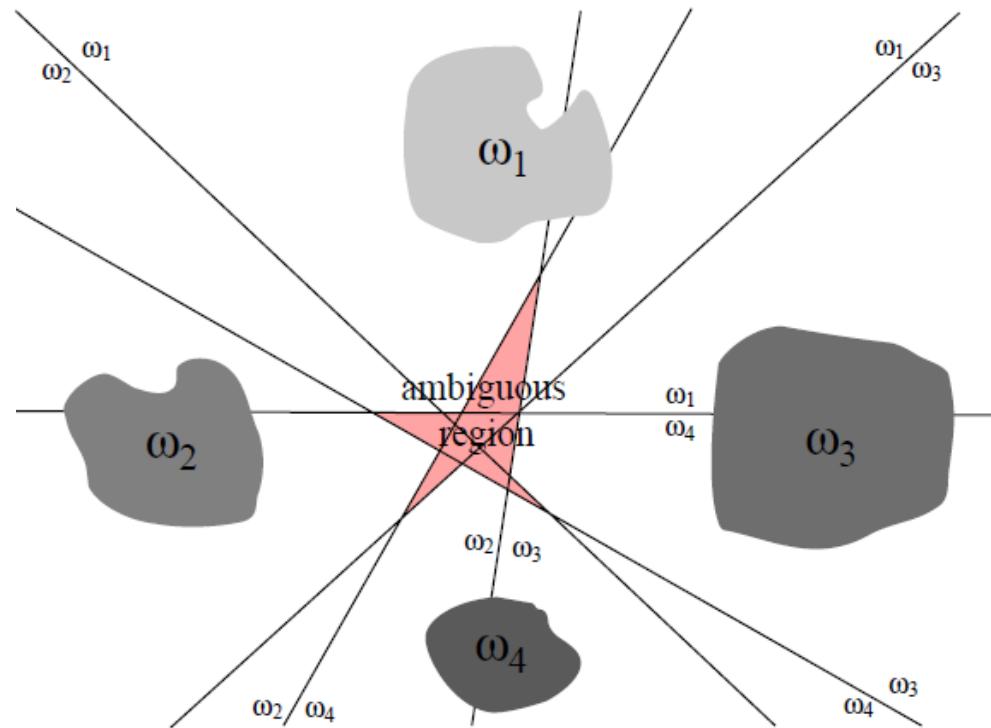


Fig: 4-class problem

Linear Machine

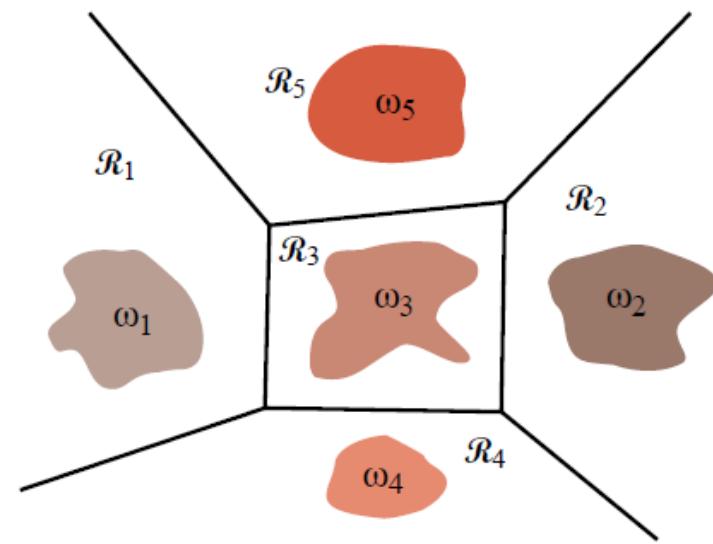
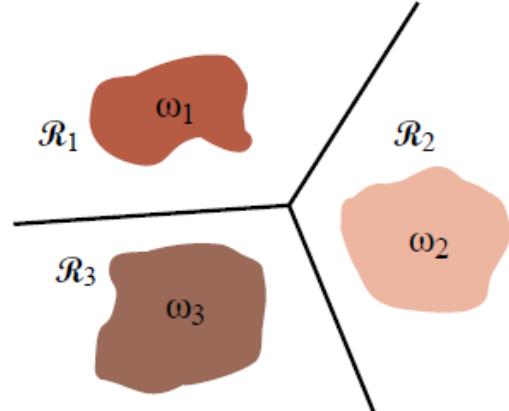


Figure 5.4: Decision boundaries produced by a linear machine for a three-class problem and a five-class problem.



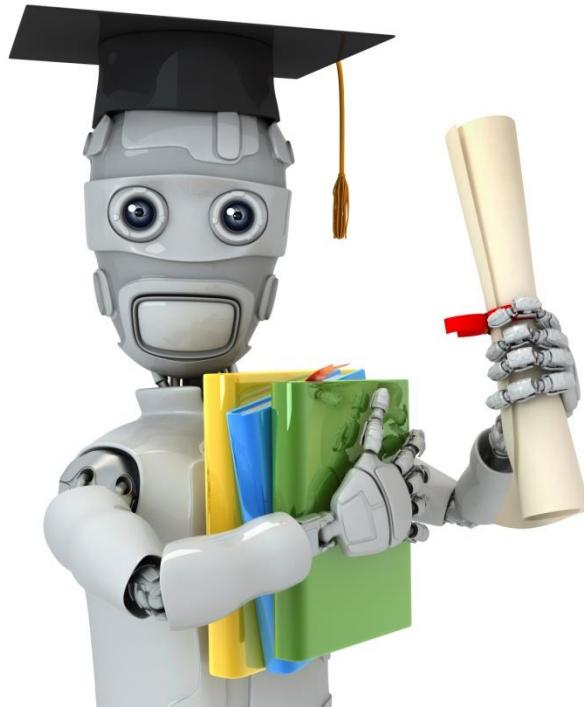
CSE 4621

Machine Learning

Lecture 6

Md. Hasanul Kabir, PhD.
Professor, CSE Department
Islamic University of Technology (IUT)





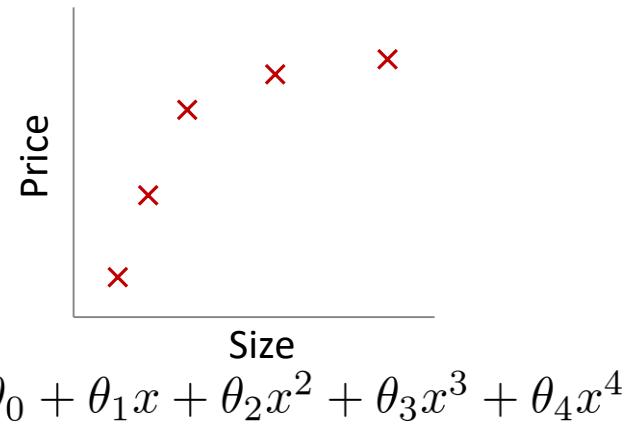
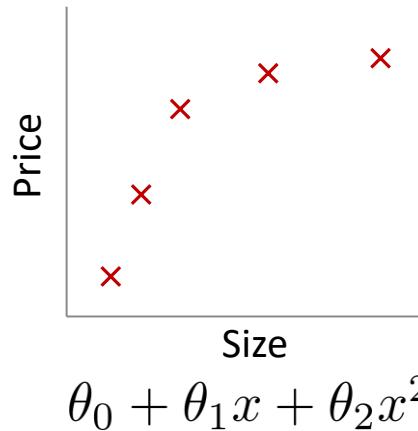
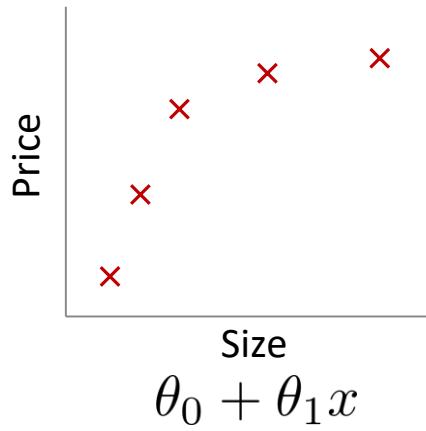
Machine Learning

Regularization

The problem of overfitting

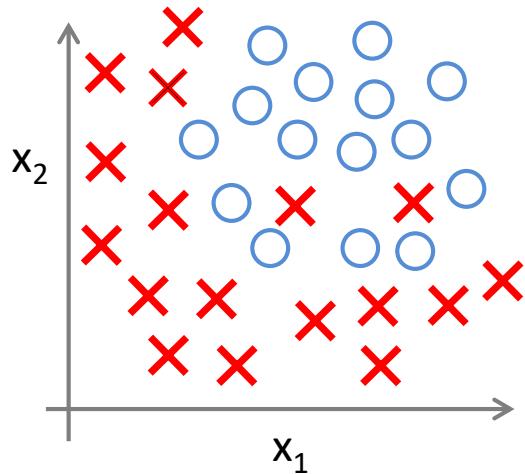
Source & Special Thanks to Andrew Ng (Coursera) Machine Learning Course

Example: Linear regression (housing prices)



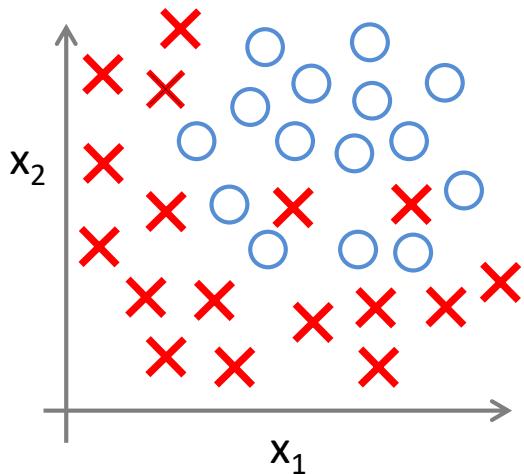
Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Example: Logistic regression

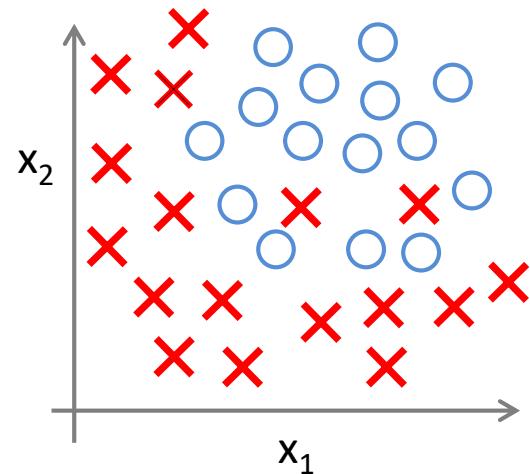


$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

(g = sigmoid function)



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$$

Addressing overfitting:

x_1 = size of house

x_2 = no. of bedrooms

x_3 = no. of floors

x_4 = age of house

x_5 = average income in neighborhood

x_6 = kitchen size

⋮

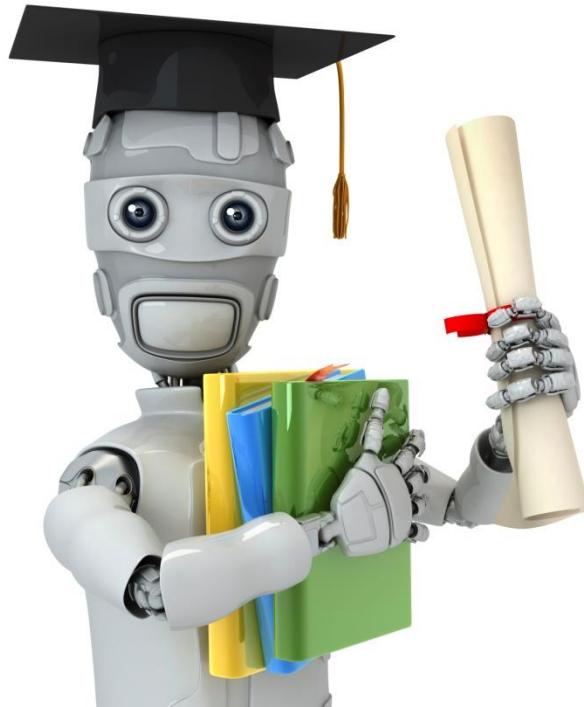
x_{100}



Addressing overfitting:

Options:

1. Reduce number of features.
 - Manually select which features to keep.
 - Model selection algorithm.
2. Regularization.
 - Keep all the features, but reduce magnitude/values of parameters θ_j .
 - Works well when we have a lot of features, each of which contributes a bit to predicting y .

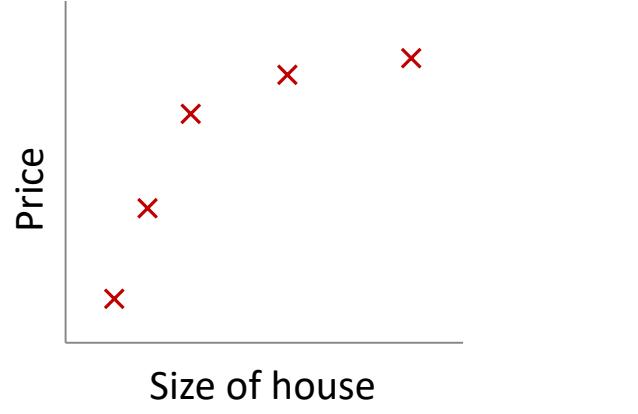
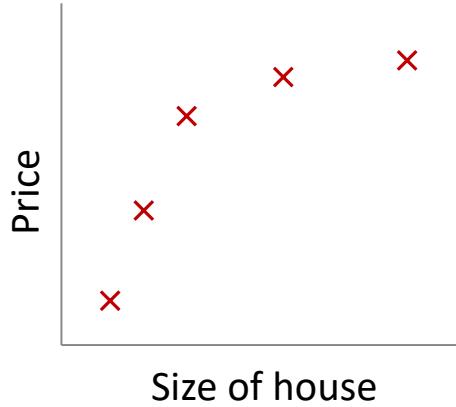


Machine Learning

Regularization

Cost function

Intuition



Suppose we penalize and make θ_3, θ_4 really small.

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Regularization.

Small values for parameters $\theta_0, \theta_1, \dots, \theta_n$

- “Simpler” hypothesis
- Less prone to overfitting

Housing:

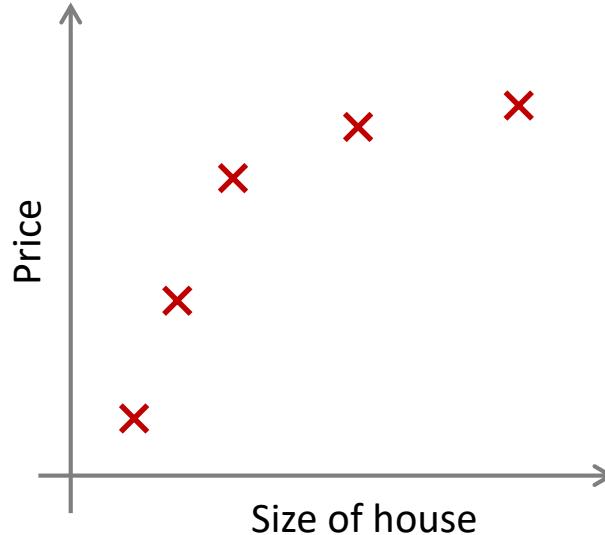
- Features: x_1, x_2, \dots, x_{100}
- Parameters: $\theta_0, \theta_1, \theta_2, \dots, \theta_{100}$

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

Regularization.

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

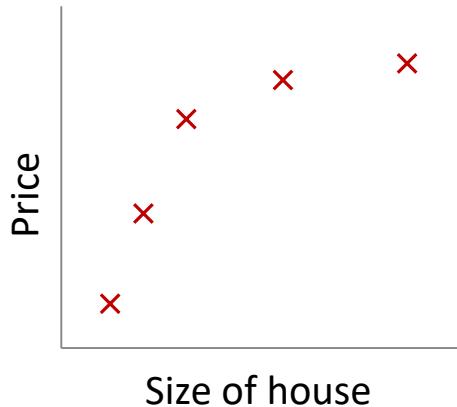
$$\min_{\theta} J(\theta)$$



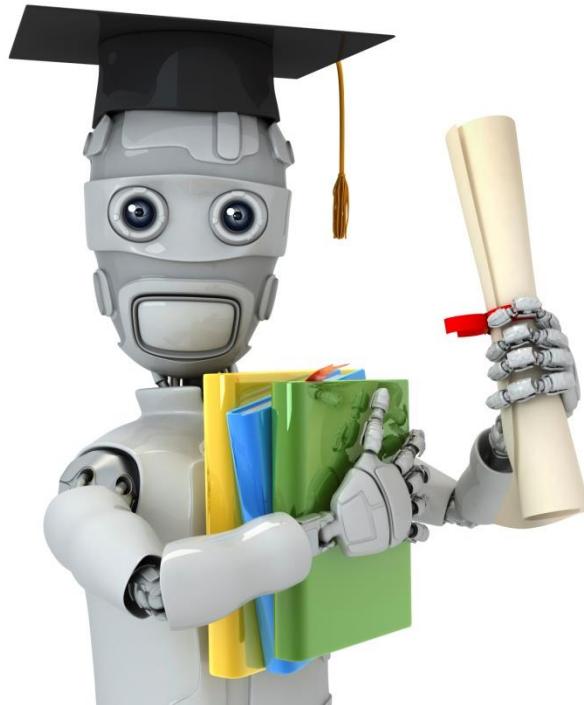
In regularized linear regression, we choose θ to minimize

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

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?



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



Machine Learning

Regularization

Regularized linear regression

Regularized linear regression

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

$$\min_{\theta} J(\theta)$$

Gradient descent

Repeat {

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

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad | \\ (j = \text{X}, 1, 2, 3, \dots, n)$$

}

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

Normal equation

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$\min_{\theta} J(\theta)$$

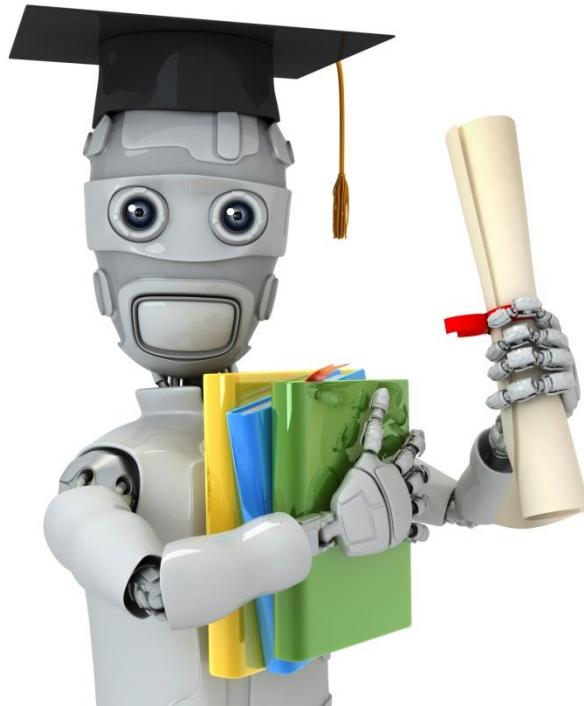
Non-invertibility (optional/advanced).

Suppose $m \leq n$,
(#examples) (#features)

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

If $\lambda > 0$,

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

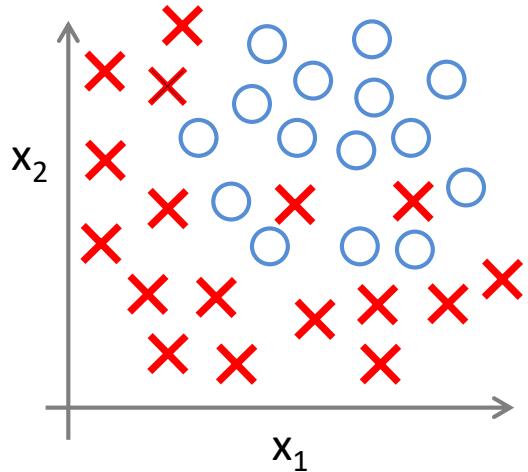


Machine Learning

Regularization

Regularized logistic regression

Regularized logistic regression.



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

Cost function:

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

Gradient descent

Repeat {

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

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

$(j = \textcolor{red}{X}, 1, 2, 3, \dots, n)$

}

L1 (Lasso) Regularization

- In L1 regularization, unimportant feature parameters shrink to zero.
 - Leads to sparse solution
- In Sparse solution majority of the input features have zero weights and very few features have non zero weights.
- L1 regularization does feature selection. It does this by assigning insignificant input features with zero weight and useful features with a non zero weight.

Cost
Function

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^n |\theta_i|$$

Regularization Term

L2 (Ridge) Regularization

- L2 regularization forces the weights to be small but does not make them zero and gives non-sparse solution.
- L2 regularization is less likely than L1 to produce zero weight coefficients.
- Ridge regression performs better when all the input features influence the output.
- *L2 regularization is able to learn complex data patterns*

Cost Function

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^n \theta_i^2$$

Get Less than Perfect Solution!

- Regularization reduce the chances of overfitting because introducing λ makes us shift *away* from the very θ that was going to cause us overfitting problems.
- With Gradient Descent (for 1D feature space x)

$$\theta := \theta - (\theta^T x - y) x = \theta - D$$

Without Regularization

$$\theta := \begin{cases} \theta - (\theta^T x - y) x - \lambda, & \text{if } \theta > 0 \\ \theta - (\theta^T x - y) x + \lambda, & \text{if } \theta < 0 \end{cases}$$

L1 Regularization

$$\theta := \theta - (\theta^T x - y) x - \lambda \theta$$

L2 Regularization

Why?

- Introducing a penalty to the sum of the weights means that the model has to “distribute” its weights optimally, so naturally most of this “resource” will go to the simple features that explain most of the variance, with complex features getting small or zero weights.

$$\theta := \theta - (\theta^T x - y) x = \theta - D$$

Without Regularization

$$\theta := \begin{cases} \theta - \lambda / m - D, & \text{if } \theta > 0 \\ \theta + \lambda / m - D, & \text{if } \theta < 0 \end{cases}$$

L1 Regularization

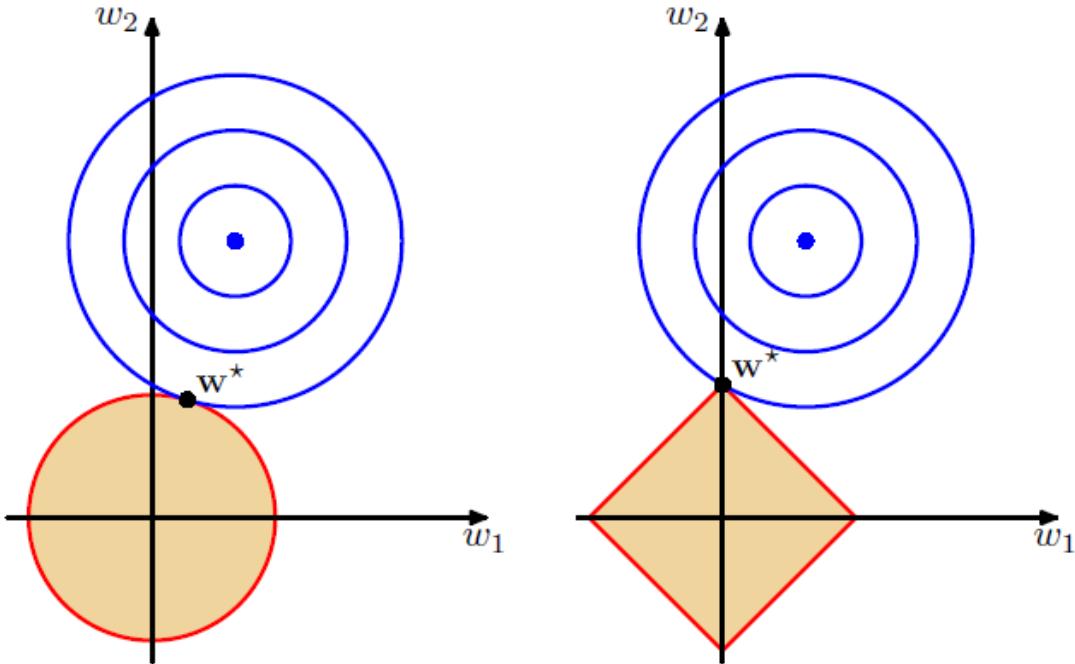
$$\theta := \theta(1 - \lambda / m) - D$$

L2 Regularization

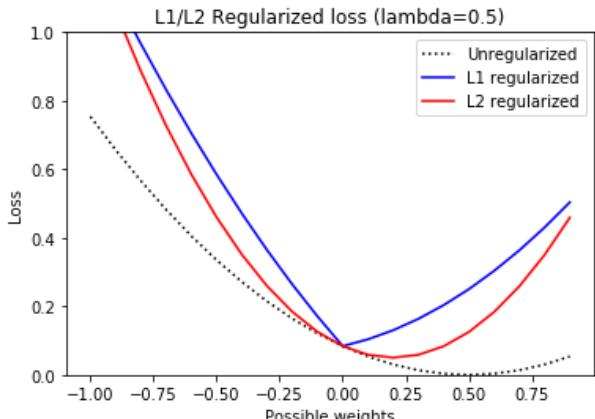
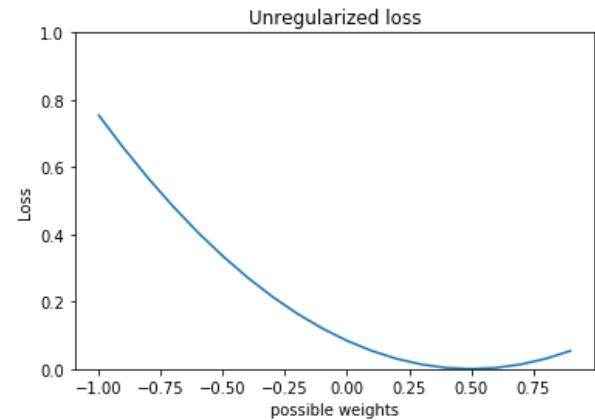
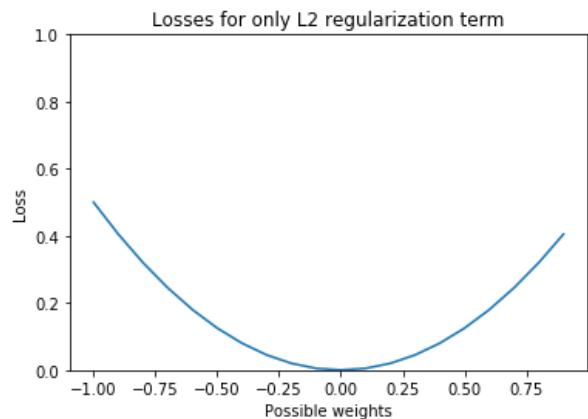
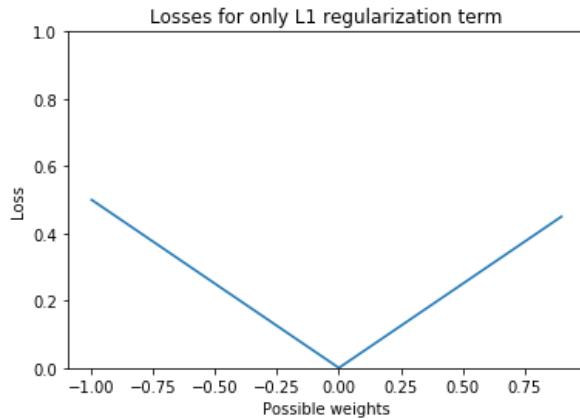
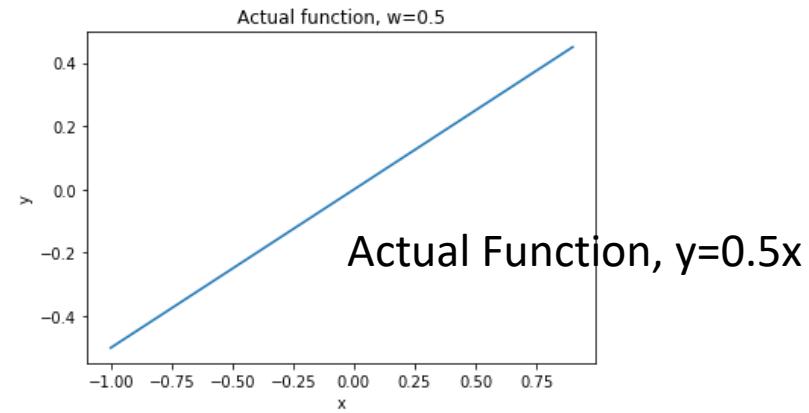
Example weights: $\hat{y} = 0.4561x_1 - 0.0007x_2 + 0.3251x_3 + 0.0009x_4 + 0.0001x_5 - 0.9142x_6 - 0.553$

Visual Interpretations

Figure 3.4 Plot of the contours of the unregularized error function (blue) along with the constraint region (3.30) for the quadratic regularizer $q = 2$ on the left and the lasso regularizer $q = 1$ on the right, in which the optimum value for the parameter vector w is denoted by w^* . The lasso gives a sparse solution in which $w_1^* = 0$.



Visual Interpretations



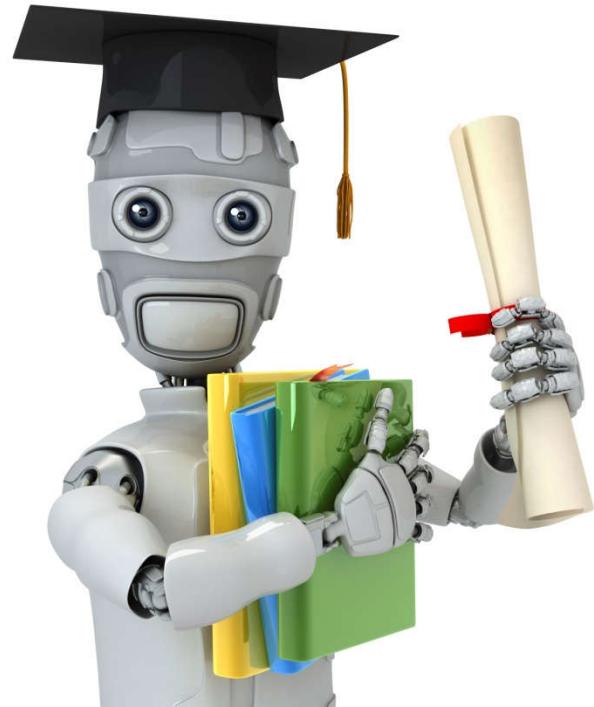


CSE 4621 Machine Learning

Lecture 7

Md. Hasanul Kabir, PhD.
Professor, CSE Department
Islamic University of Technology (IUT)





Machine Learning

Neural Network

Introduction

Source & Special Thanks to (Coursera) Machine Learning / NN&DL Courses

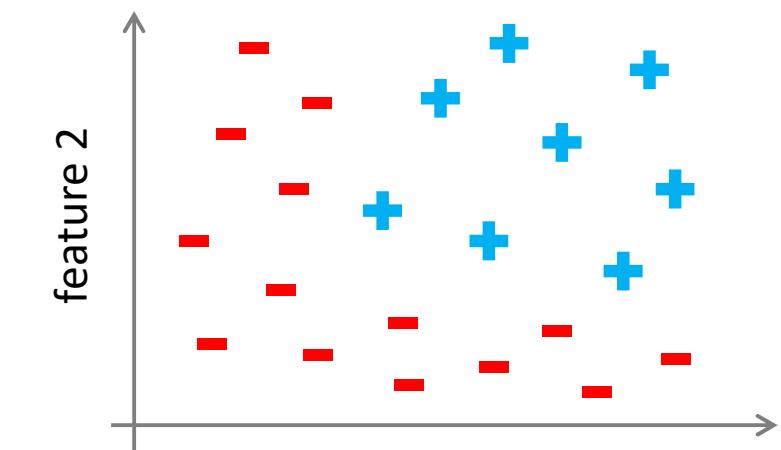
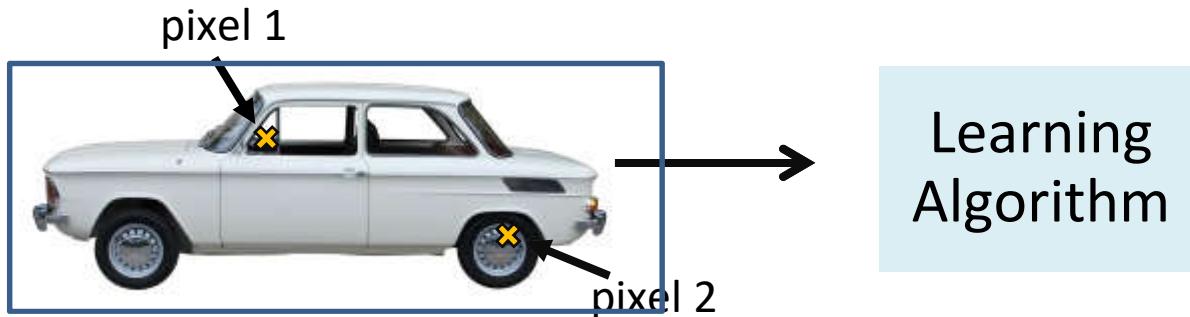
Computer Vision: Car detection



Testing:



What is this?



+ Cars
- "Non"-Cars

50×50 pixel images \rightarrow 2500 pixels
 $n = 2500$ (7500 if RGB)

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

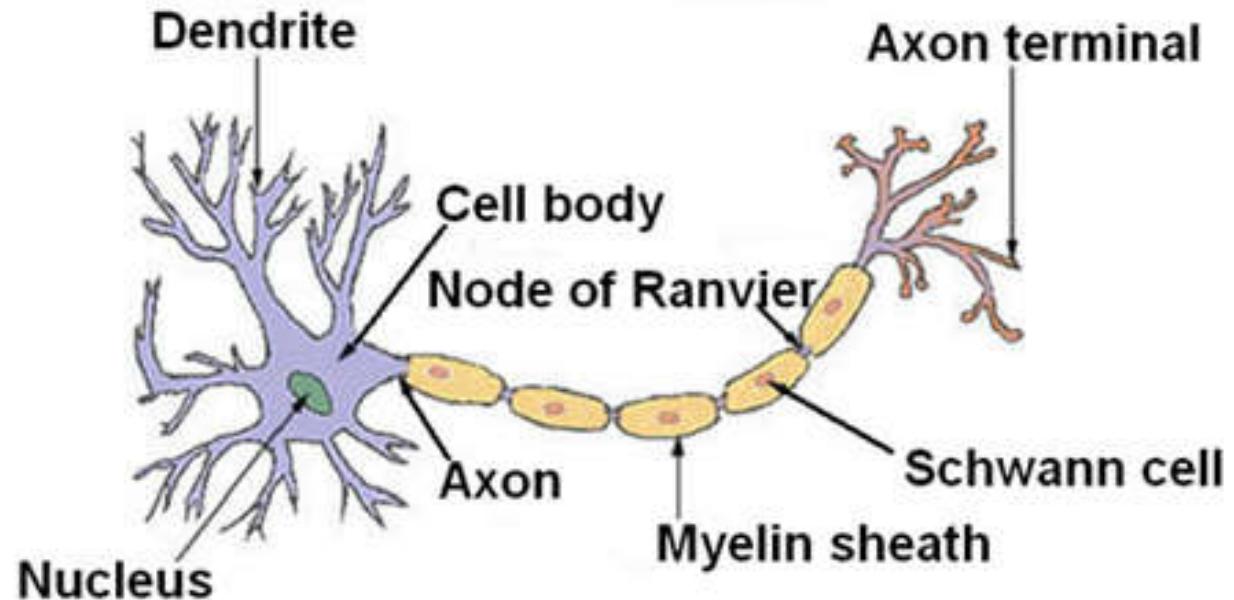
Quadratic features ($x_i \times x_j$): ≈ 3 million features $O(n^2)$

Neural Networks (NN)

- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s;
 - popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications & due to high computing devices.
- NN is another non-linear classifier
 - Decision boundary is non-linear

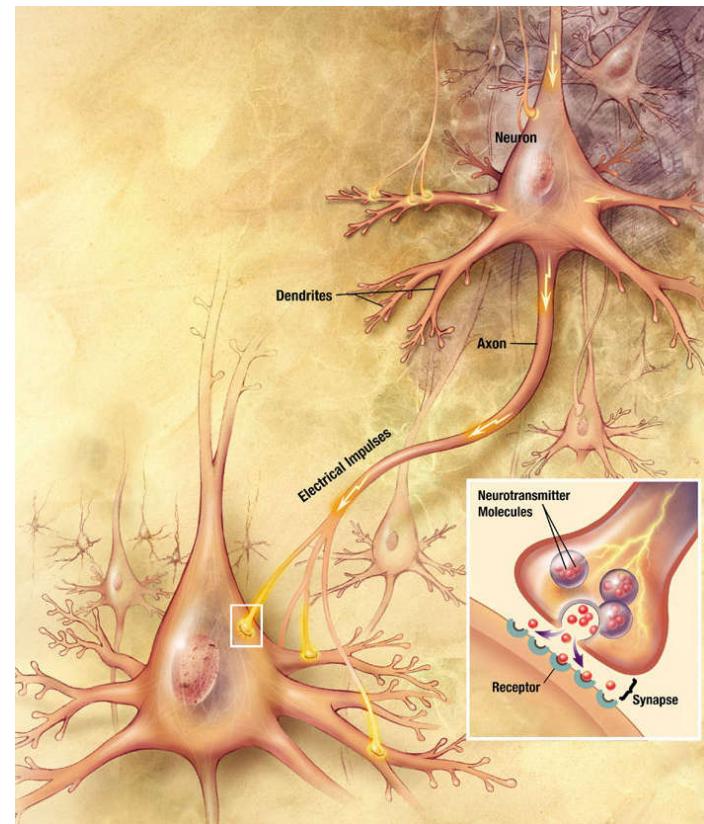
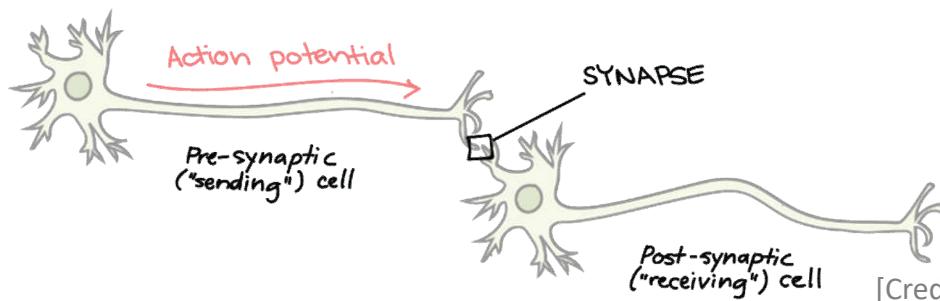
Neuron in the brain

- Biological Neurons are the core components of the human brain.
- A neuron consists of a cell body, dendrites, and an axon.



Neurons in the brain

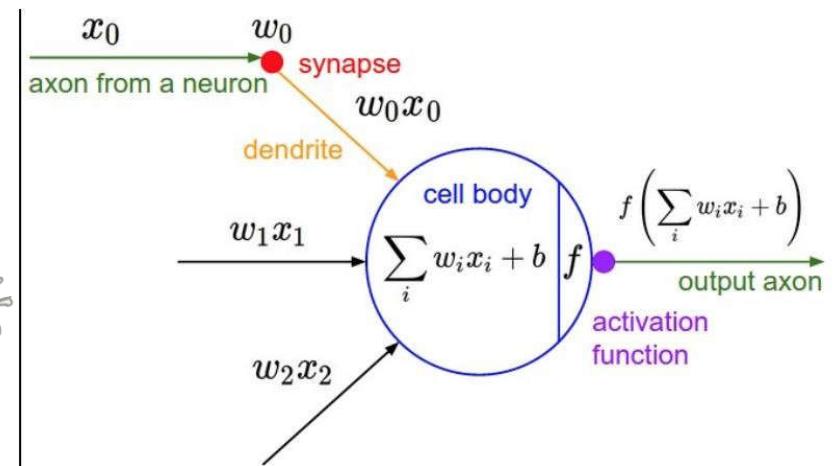
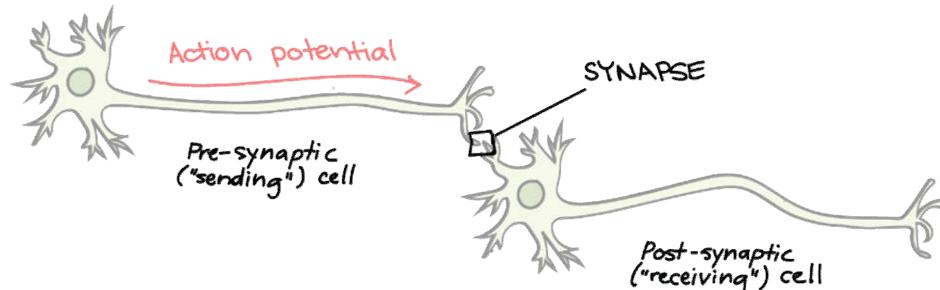
- Neurons process and transmit information to other neurons by emitting electrical signals.
- Each neuron receives input signals from its dendrites and produces output signals along its axon.
- The axon branches out and connects via synapses to dendrites of other neurons.



[Credit: US National Institutes of Health, National Institute on Aging]

Biological Neuron VS. Artificial Neuron

- Artificial neurons are inspired by biological neurons
- An artificial neuron has a finite number of inputs with weights associated to them, and an activation function (also called transfer function).
- The output of the neuron is the result of the activation function applied to the weighted sum of inputs.
- Artificial neurons are connected with each others to form artificial neural networks.

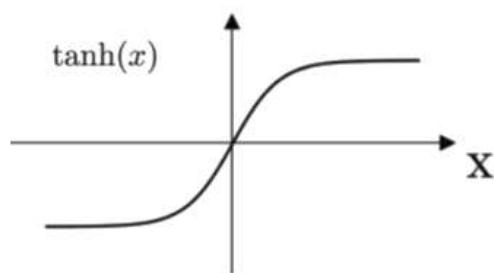


Activation Function

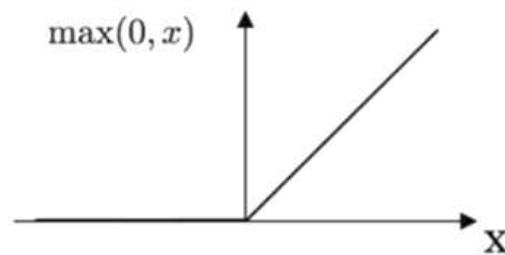
- Activation functions transform the weighted sum of inputs that goes into the artificial neurons.
- These functions should be non-linear to encode complex patterns of the data.
- The most popular activation functions are
 - Logistic
 - Tanh
 - Rectified Linear Unit (ReLU)
 - Leaky RELU
 - Exponential Linear Unit (ELU)

Activation Function

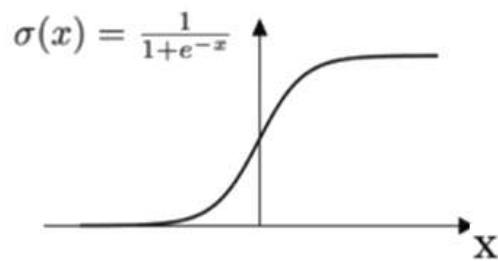
Tanh



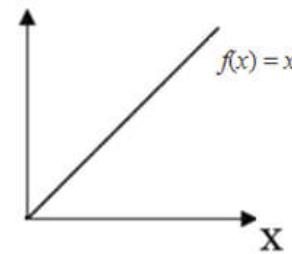
ReLU



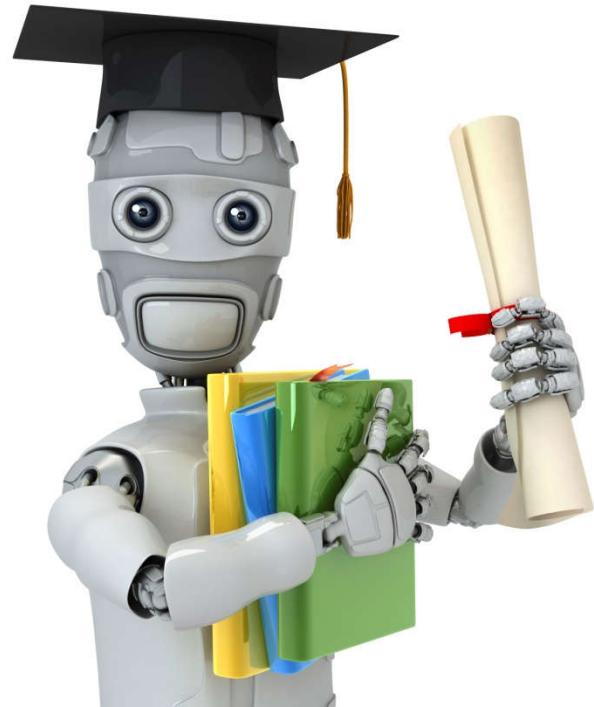
Sigmoid



Linear



Have a look for more: https://en.wikipedia.org/wiki/Activation_function



Machine Learning

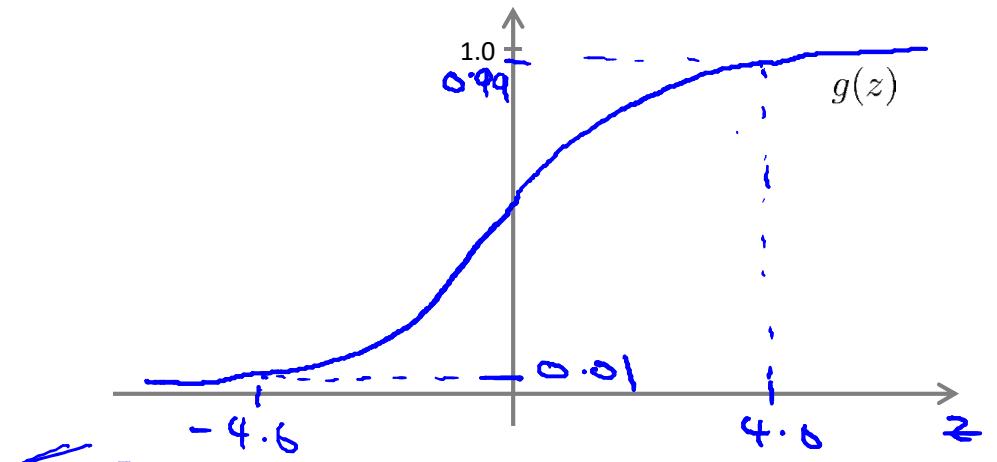
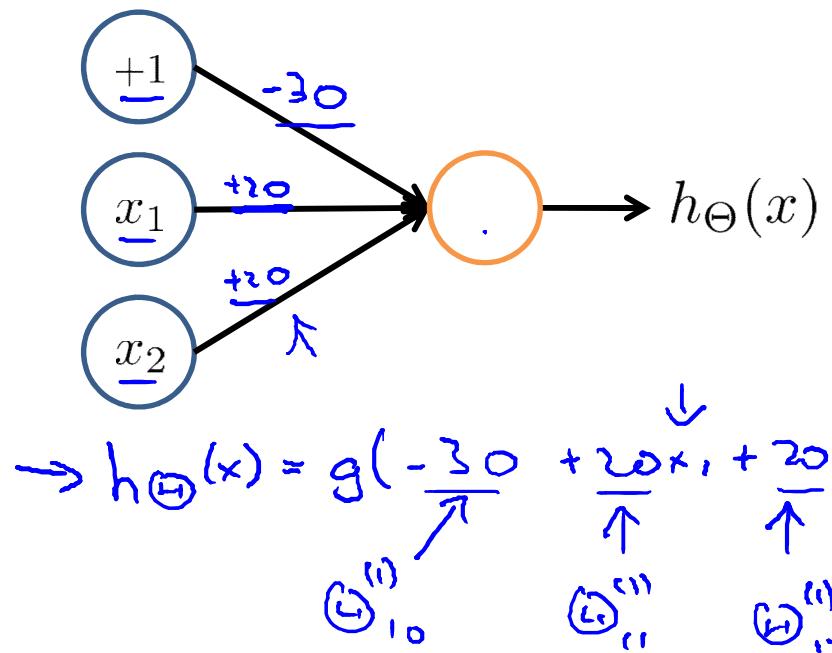
Neural Networks: Representation

Examples and intuitions II

Simple example: AND

$$\rightarrow x_1, x_2 \in \{0, 1\}$$

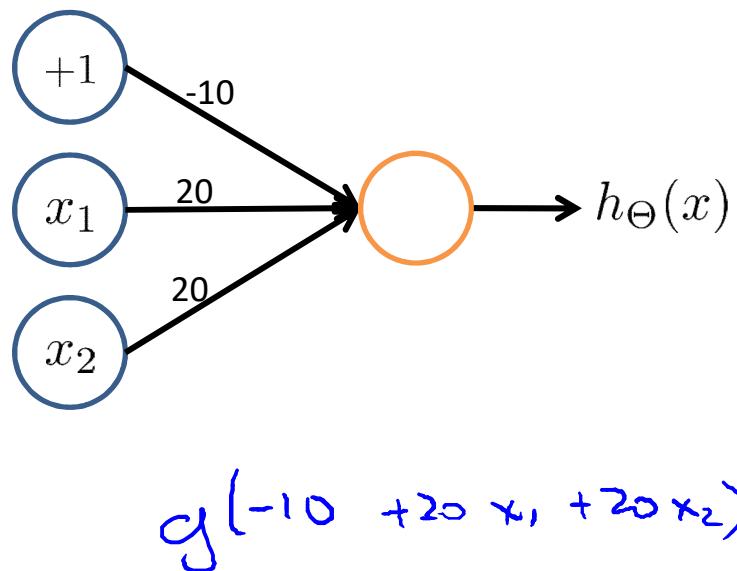
$$\rightarrow y = x_1 \text{ AND } x_2$$



x_1	x_2	$h_{\Theta}(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

$h_{\Theta}(x) \approx x_1 \text{ AND } x_2$

Example: OR function



x_1	x_2	$h_{\Theta}(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	≈ 1
1	1	≈ 1

The table shows the output of the function $h_{\Theta}(x)$ for all combinations of x_1 and x_2 . The values are approximated in blue. A pink hand-drawn bracket highlights the row where $x_1 = 1$ and $x_2 = 1$, showing the value ≈ 1 .

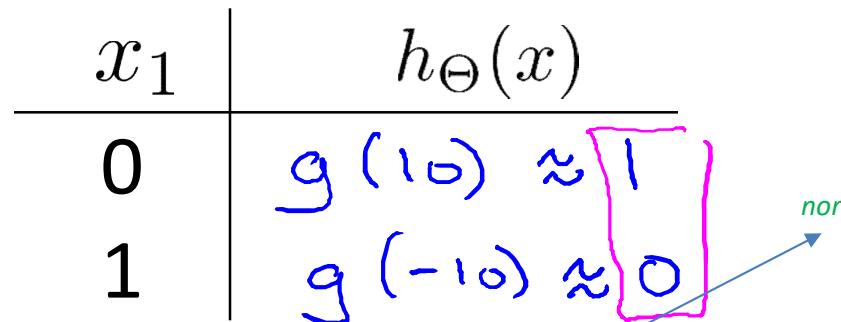
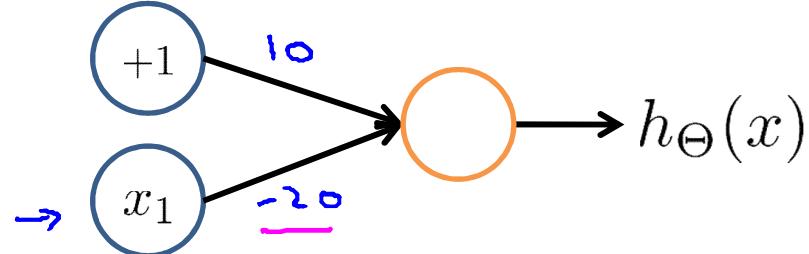
$\rightarrow x_1 \text{ AND } x_2$

$\rightarrow x_1 \text{ OR } x_2$

$\{0,1\}$

Negation:

NOT x_1

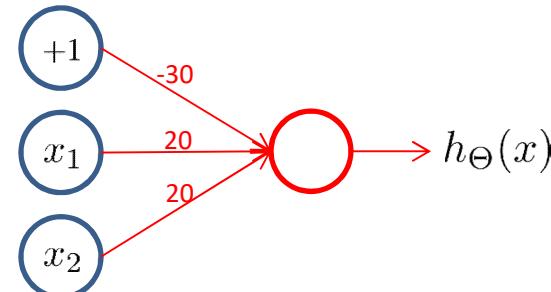


$$h_\Theta(x) = g(10 - 20x_1)$$

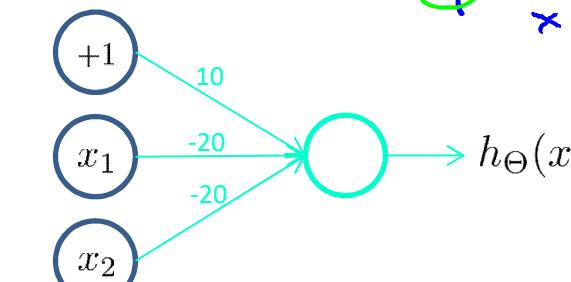
$\rightarrow (\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$

$=1$ if and only if
 $\rightarrow x_1 = x_2 = 0$

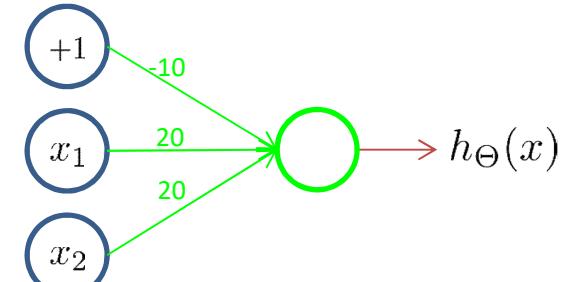
Putting it together: $x_1 \text{ XNOR } x_2$



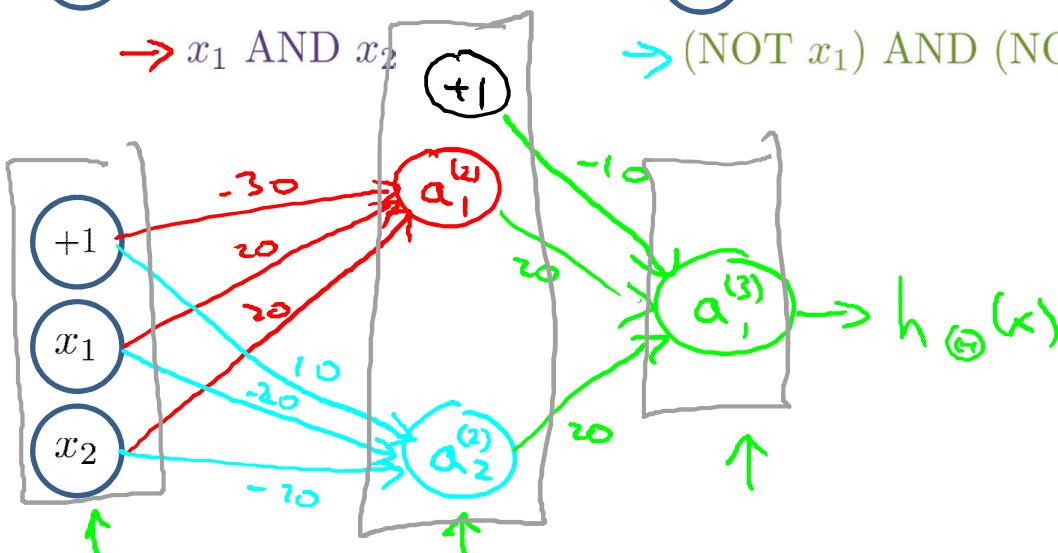
$\rightarrow x_1 \text{ AND } x_2$



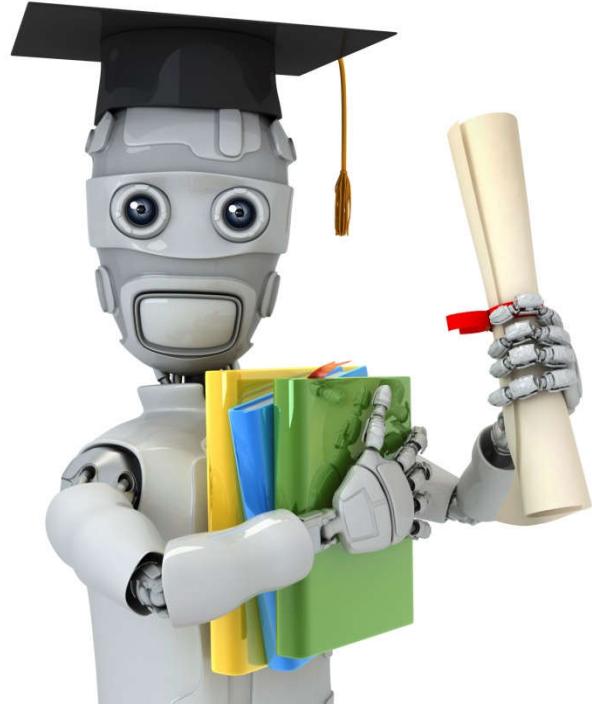
$\rightarrow (\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$



$\rightarrow x_1 \text{ OR } x_2$



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_{\Theta}(x)$
0	0	0	1	1 ←
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1 ←

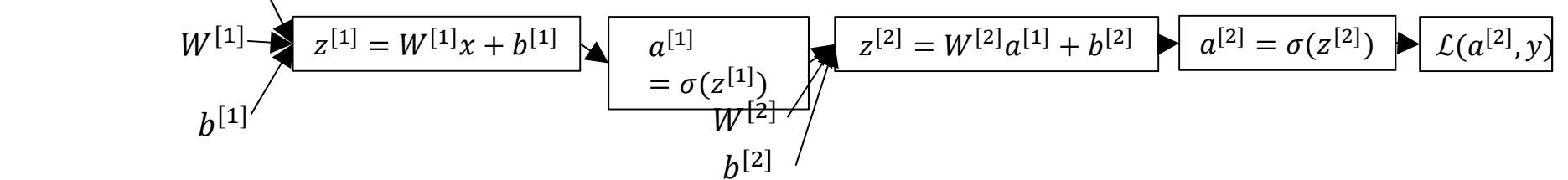
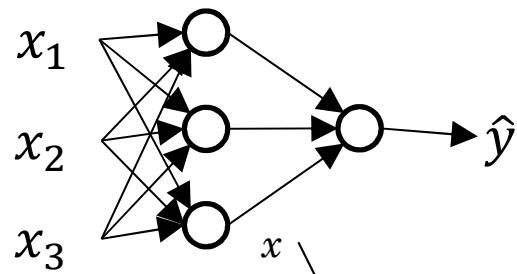
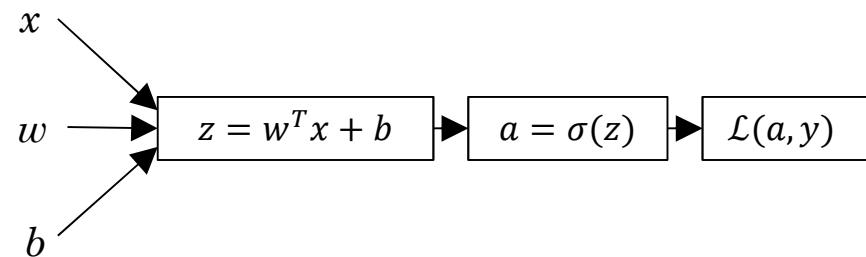
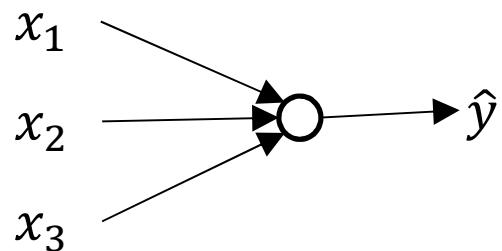


Machine Learning

Neural Networks: Representation

Two-class classification

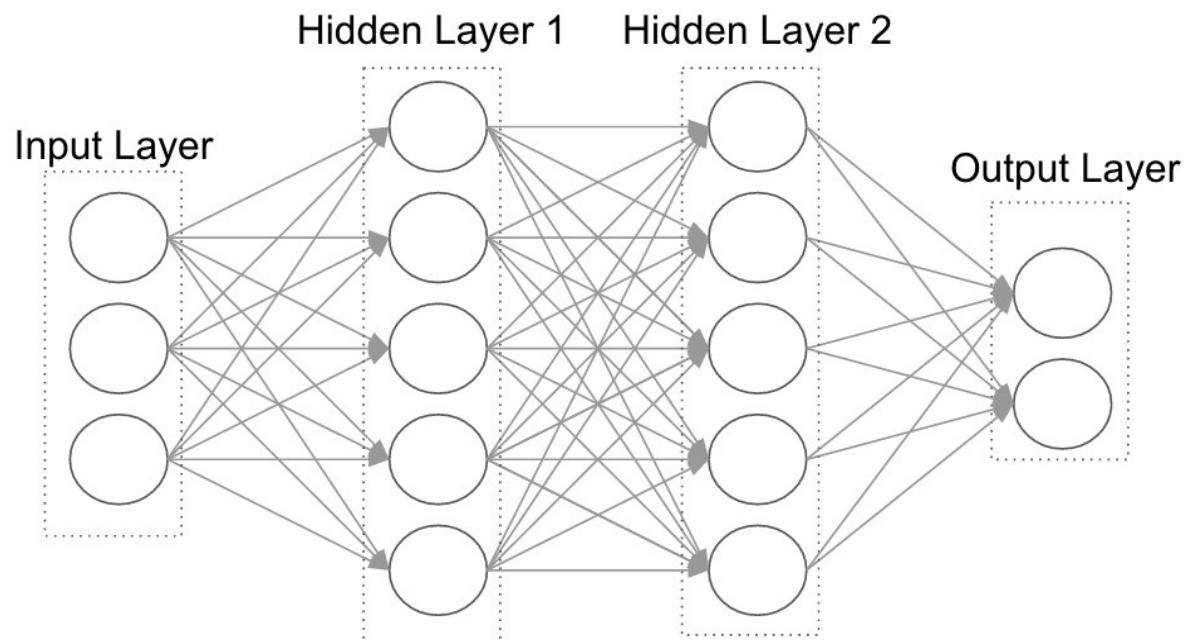
Define a Neural Network



Feed Forward NN

- Feed forward Neural Networks (FFN) are the simplest form of Artificial Neural Networks (ANN).
- These networks have 3 types of layers:
 - Input layer,
 - hidden layer and
 - output layer.
- In FFN, data moves from the input layer through the hidden nodes (if any) and to the output nodes.
- “Fully-connected” means that each node is connected to all the nodes in the next layer.
- The number of hidden layers and their size are the parameters. The larger and deeper the hidden layers, the more complex patterns we can model in theory.

Multi-layer Perceptron (MLP)



Feedforward neural network with 2 hidden layers

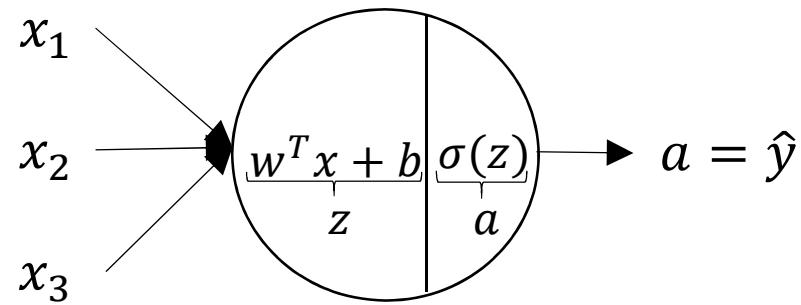


deeplearning.ai

One hidden layer
Neural Network

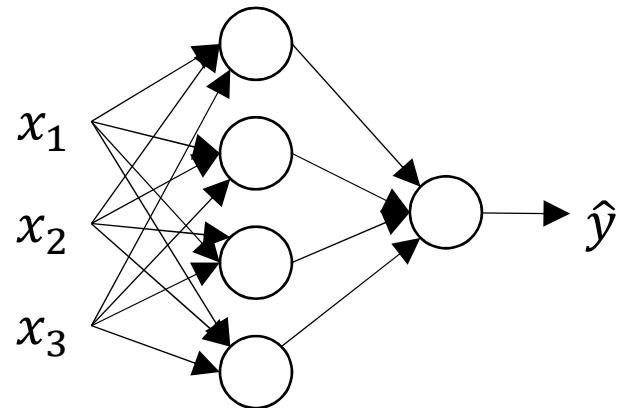
Computing a
Neural Network's
Output

Neural Network Representation



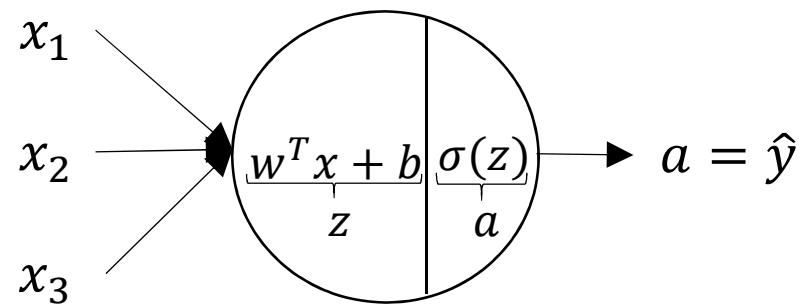
$$z = w^T x + b$$

$$a = \sigma(z)$$



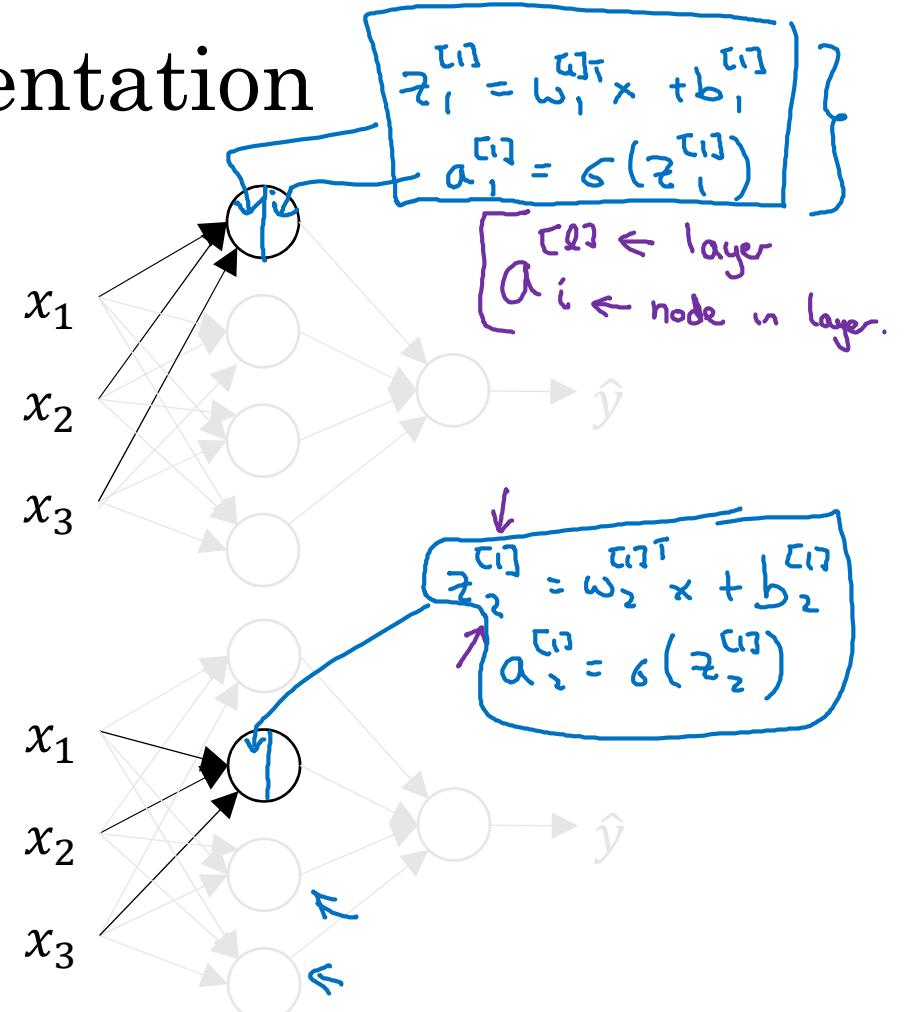
Andrew Ng

Neural Network Representation



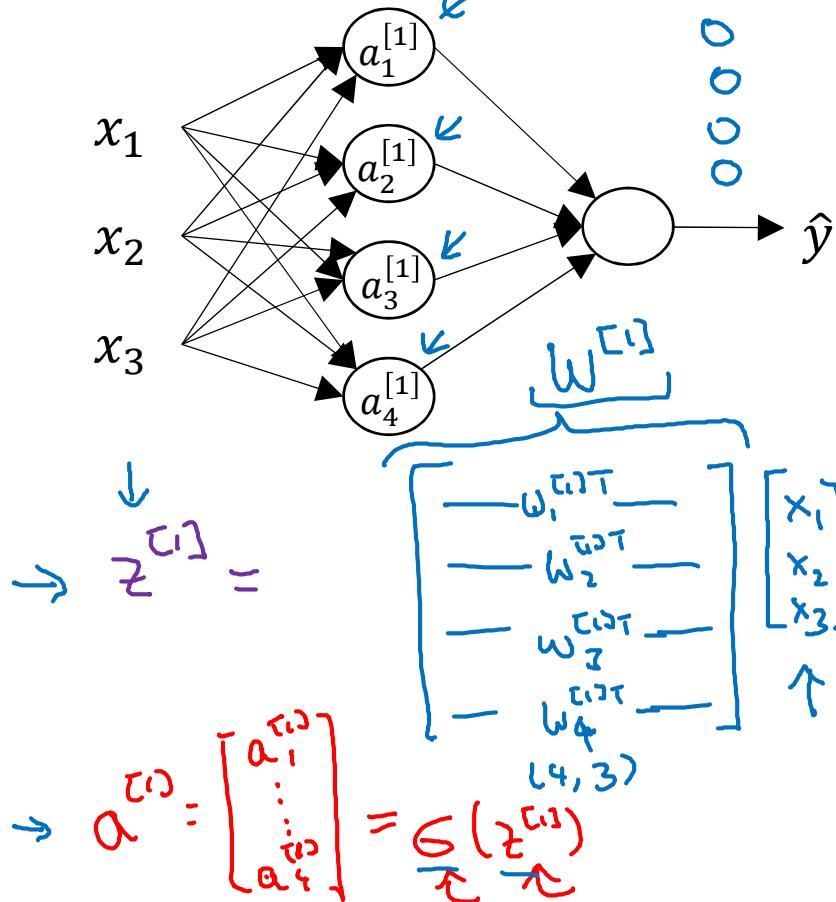
$$z = w^T x + b$$

$$a = \sigma(z)$$



Andrew Ng

Neural Network Representation



$$\begin{aligned}
 z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]} \\
 z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]} \\
 z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]} \\
 z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]}
 \end{aligned}$$

$$\begin{aligned}
 a_1^{[1]} &= \sigma(z_1^{[1]}) \\
 a_2^{[1]} &= \sigma(z_2^{[1]}) \\
 a_3^{[1]} &= \sigma(z_3^{[1]}) \\
 a_4^{[1]} &= \sigma(z_4^{[1]}) \\
 \end{aligned}$$

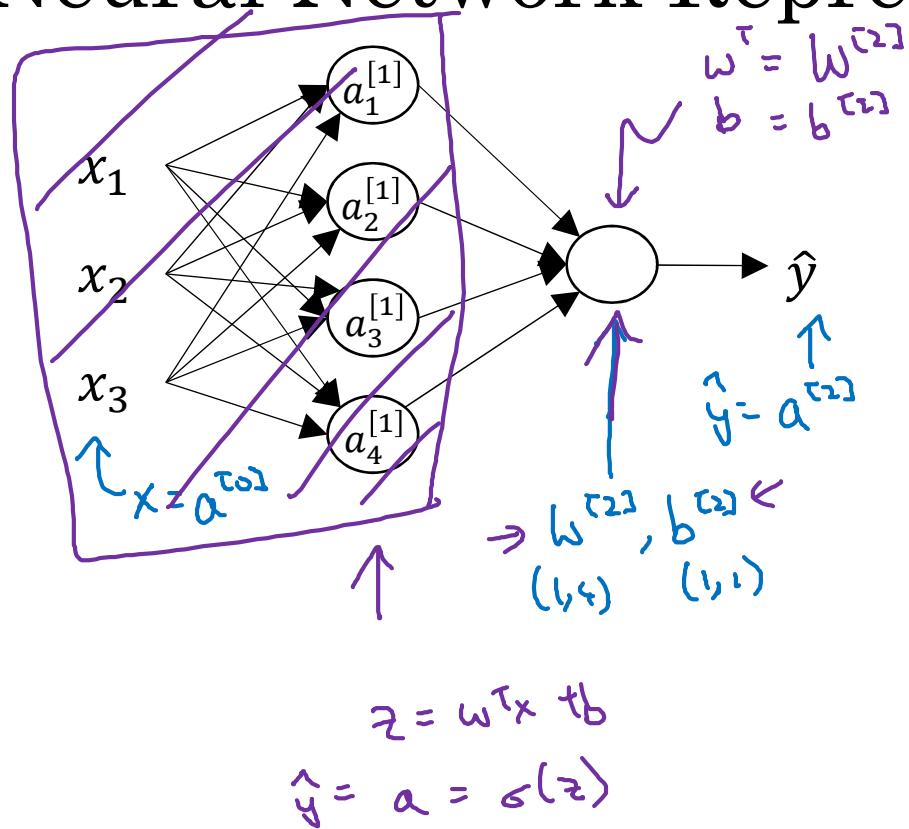
$$\begin{aligned}
 \rightarrow z^{[1]} &= \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} \\
 \rightarrow a^{[1]} &= \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]}) \\
 \end{aligned}$$

$$\begin{aligned}
 \rightarrow w_1^{[1]T} x + b_1^{[1]} &= z_1^{[1]} \\
 \rightarrow w_2^{[1]T} x + b_2^{[1]} &= z_2^{[1]} \\
 \rightarrow w_3^{[1]T} x + b_3^{[1]} &= z_3^{[1]} \\
 \rightarrow w_4^{[1]T} x + b_4^{[1]} &= z_4^{[1]}
 \end{aligned}$$

$$\begin{aligned}
 \Rightarrow & \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}
 \end{aligned}$$

Andrew Ng

Neural Network Representation

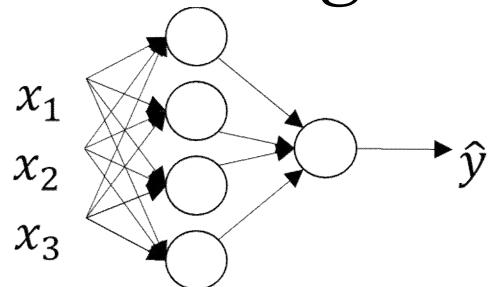


Given input x :

$$\begin{aligned} &\rightarrow z^{[1]} = W^{[1]} x + b^{[1]} \\ &\rightarrow a^{[1]} = \sigma(z^{[1]}) \\ &\rightarrow z^{[2]} = W^{[2]} a^{[1]} + b^{[2]} \\ &\rightarrow a^{[2]} = \sigma(z^{[2]}) \end{aligned}$$

Andrew Ng

Vectorizing across multiple examples



$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & | \\ a^{1} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

for $i = 1$ to m

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

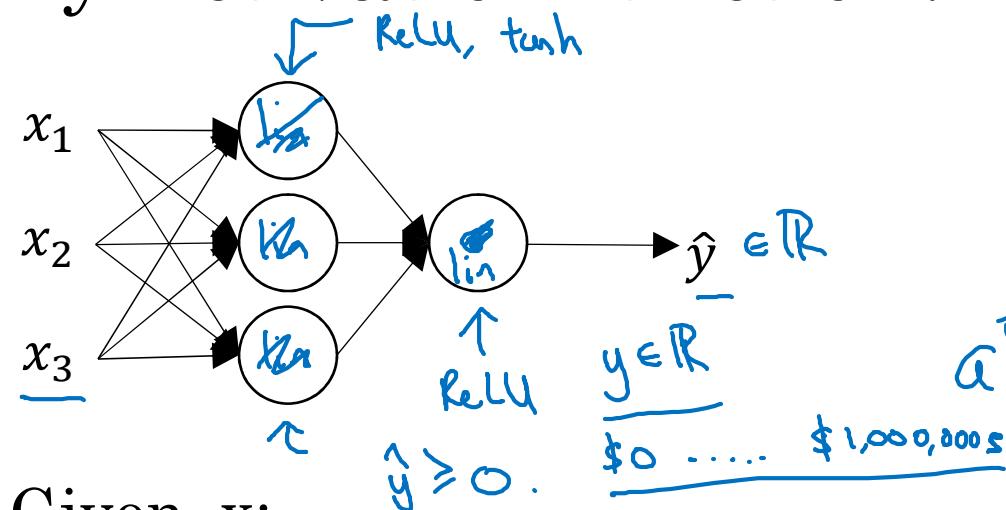
$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

Andrew Ng

Why Activation function?



Given x :

$$\rightarrow z^{[1]} = W^{[1]}x + b^{[1]}$$

$$\rightarrow a^{[1]} = g^{[1]}(z^{[1]}) \geq^{c^{[1]}}$$

$$\rightarrow z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$\rightarrow a^{[2]} = g^{[2]}(z^{[2]}) \geq^{c^{[2]}}$$

$g(z) = z$
"linear activation
function"

$$a^{[1]} = z^{[1]} = \underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}$$

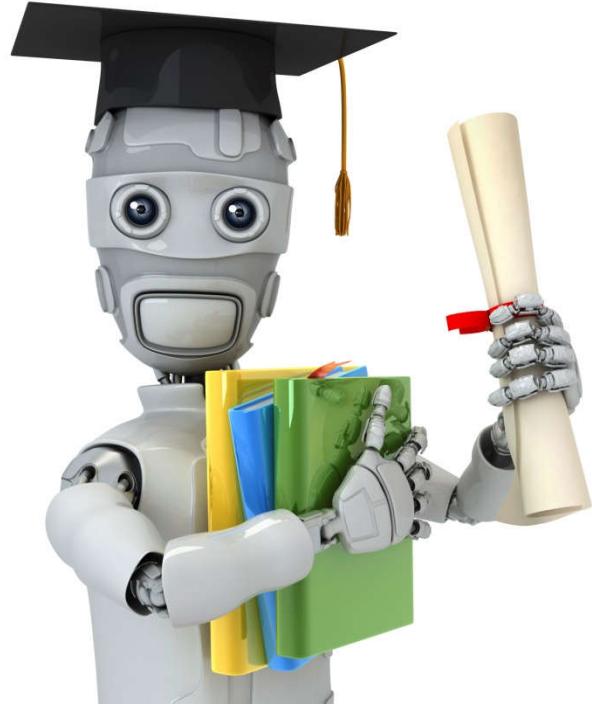
$$a^{[2]} = z^{[2]} = \underbrace{W^{[2]}a^{[1]} + b^{[2]}}_{a^{[2]}}$$

$$a^{[2]} = W^{[2]} \left(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}} \right) + b^{[2]}$$

$$= \underbrace{(W^{[2]} W^{[1]})}_{w'} x + \underbrace{(W^{[2]} b^{[1]} + b^{[2]})}_{b'}$$

$$= \underbrace{w'x + b'}_{g(z) = z}$$

Andrew Ng



Machine Learning

Neural Networks: Representation

Multi-class classification

Multiple output units: One-vs-all.



Pedestrian



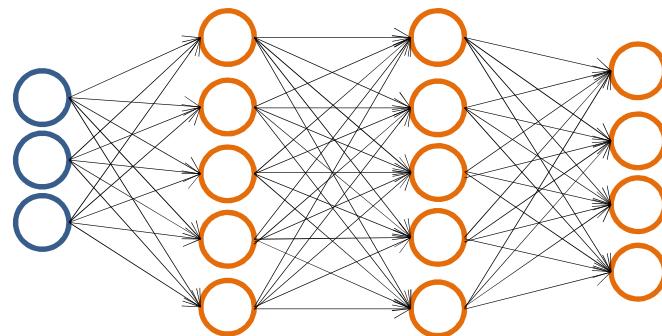
Car



Motorcycle



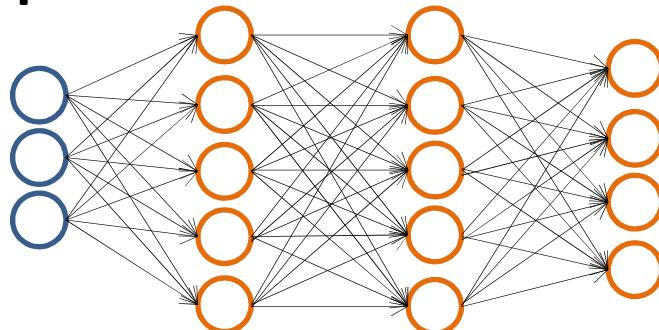
Truck



$$h_{\Theta}(x) \in \mathbb{R}^4$$

Want $h_{\Theta}(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.
when pedestrian when car when motorcycle

Multiple output units: One-vs-all.



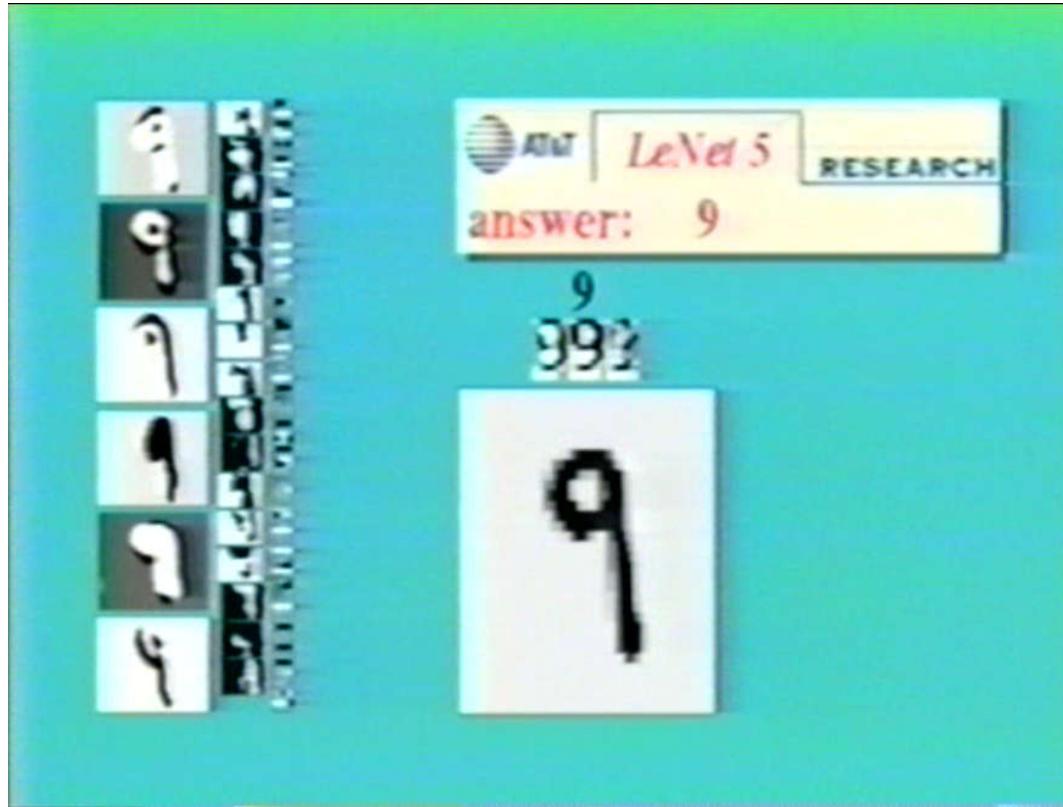
$$h_{\Theta}(x) \in \mathbb{R}^4$$

Want $h_{\Theta}(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.
when pedestrian when car when motorcycle

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

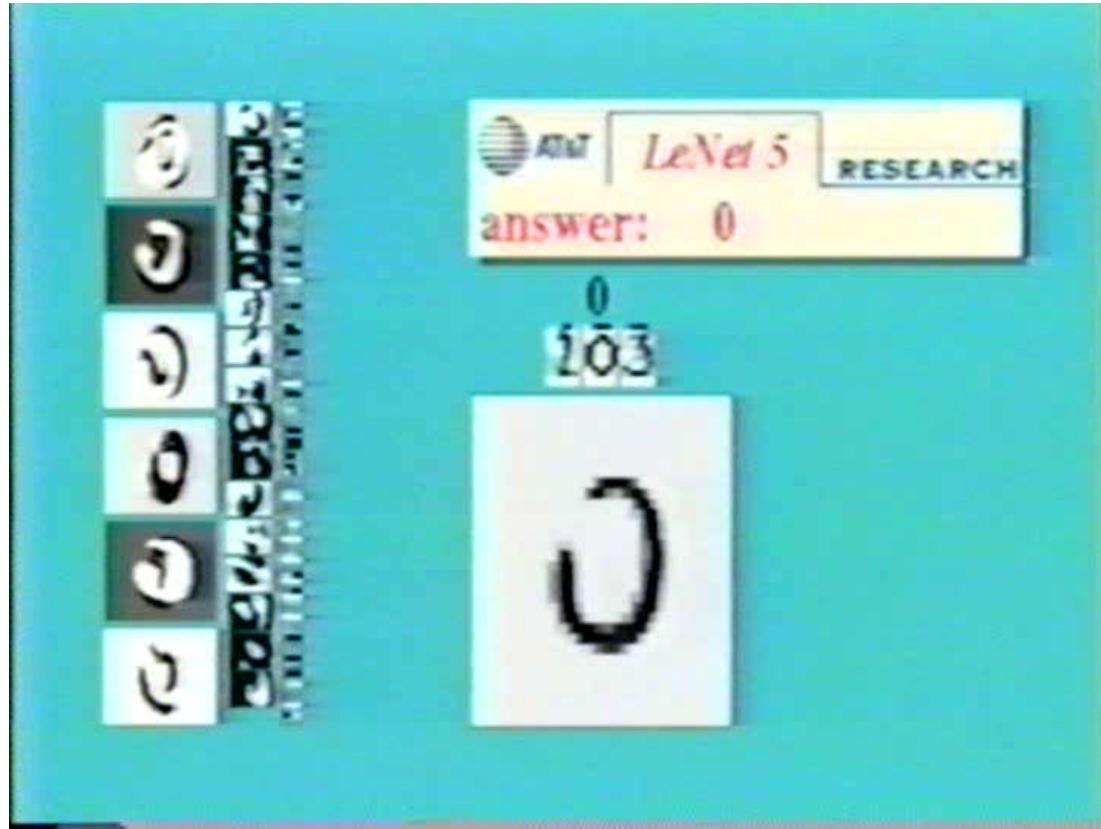
$y^{(i)}$ one of $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$
pedestrian car motorcycle truck

Handwritten digit classification with Convolutional NN (CNN)



[Courtesy of Yann LeCun]

Handwritten digit classification with ConvNet



[Courtesy of Yann LeCun]

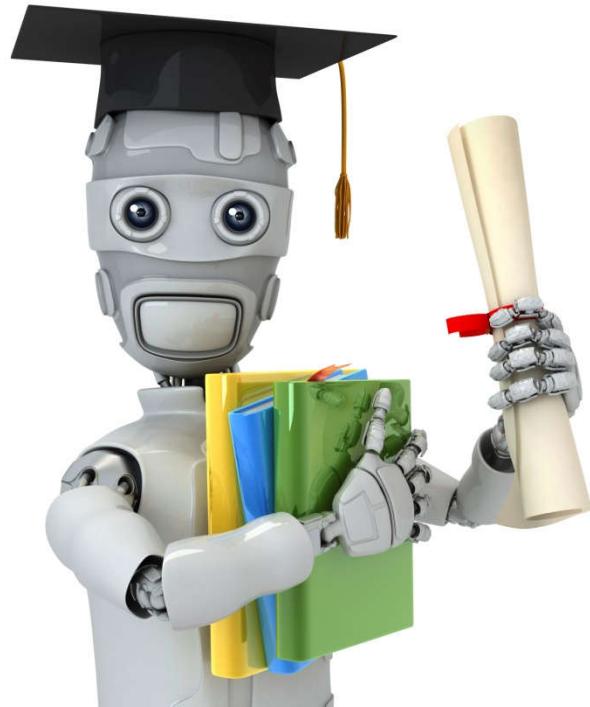


CSE 4621 Machine Learning

Lecture 8

Md. Hasanul Kabir, PhD.
Professor, CSE Department
Islamic University of Technology (IUT)





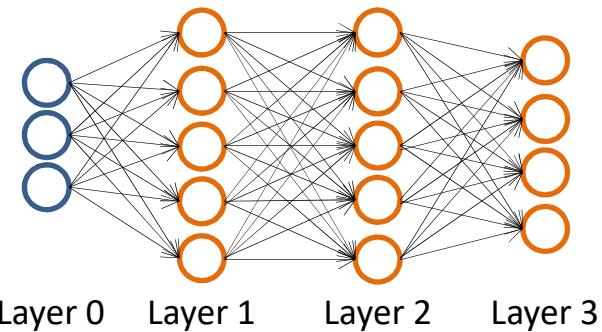
Machine Learning

Neural Networks: Learning

Cost function

Source & Special Thanks to (Coursera) Machine Learning / NN&DL Courses

Neural Network (Classification)



Binary classification

$y = 0 \text{ or } 1$

1 output unit

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

L = total no. of layers in network

$n^{[l]}$ = no. of units (not counting bias unit) in layer l

Multi-class classification (K classes)

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

pedestrian car motorcycle truck

K output units

Cost function

Logistic regression:

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

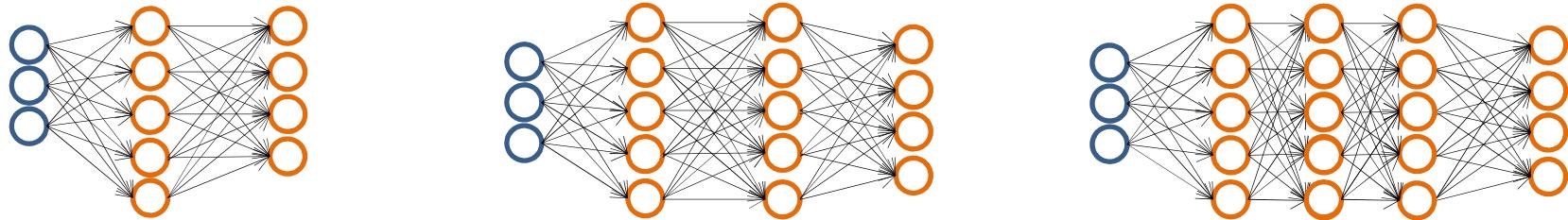
Neural network:

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

$$\begin{aligned} J(\Theta) = & -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] \\ & + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (\Theta_{ij}^{[l]})^2 \end{aligned}$$

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



No. of input units: Dimension of features $x^{(i)}$

No. output units: Number of classes

Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)



deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks

Gradient Computation

Cost Function: $J(W, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \text{Regularization Term}$

Objective: $\min J(W, b)$

Need to Compute:

- $J(W, b)$
- $\frac{\partial J(W, b)}{\partial w_{ij}^{(l)}} \quad \frac{\partial J(W, b)}{\partial b_i^{(l)}}$

Use Update equation for both W & b

Gradient descent for neural networks

Parameters: $(w^{[0]}, b^{[0]})$, $(w^{[1]}, b^{[1]})$, $(w^{[2]}, b^{[2]})$, \dots
 $n_x = n^{[0]}$, $n^{[1]}$, $n^{[2]} = 1$

Cost function: $J(w^{[0]}, b^{[0]}, w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}_i, y_i)$

Gradient descent:

→ Repeat {

 → Compute predicts $(\hat{y}^{(i)}, i=1 \dots m)$

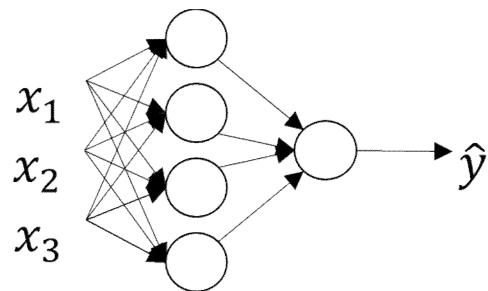
$$\frac{\partial J}{\partial w^{[l]}} = \frac{\partial J}{\partial w^{[l]}} , \quad \frac{\partial J}{\partial b^{[l]}} = \frac{\partial J}{\partial b^{[l]}} , \dots$$

$$w^{[l]} := w^{[l]} - \alpha \frac{\partial J}{\partial w^{[l]}}$$

$$b^{[l]} := b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}}$$

↳

Forward Calculation



$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & | \\ a^{1} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

for $i = 1$ to m

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

Formulas for computing derivatives

Forward propagation:

$$z^{[1]} = w^{[1]T}x + b^{[1]}$$

$$A^{[1]} = g^{[1]}(z^{[1]}) \leftarrow$$

$$z^{[2]} = w^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(z^{[2]}) = \underline{c}(z^{[2]})$$

Back propagation:

$$dz^{[2]} = A^{[2]} - Y \leftarrow$$

$$dW^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} \text{np.sum}(dz^{[2]}, \text{axis}=1, \text{keepdims=True})$$

$$dz^{[1]} = \underbrace{w^{[2]T} dz^{[2]}}_{(n^{[2]}, m)} \times \underbrace{g^{[1]}'(z^{[1]})}_{\text{element-wise product}} \quad (n^{[1]}, m)$$

$$dW^{[1]} = \frac{1}{n} dz^{[1]} X^T$$

$$\overline{db^{[1]}} = \frac{1}{n} \text{np.sum}(dz^{[1]}, \text{axis=1}, \text{keepdims=True})$$

$$\quad \quad \quad \overline{(n^{[1]}, 1)} \quad \quad \quad \overline{(n^{[1]},)} \quad \quad \quad \overline{\text{reshape} \uparrow}$$

$$Y = [y^{(1)}, y^{(2)}, \dots, y^{(n)}]$$

$$(n^{[0]}) \leftarrow$$

$$\downarrow (n^{[1]}, 1) \leftarrow$$



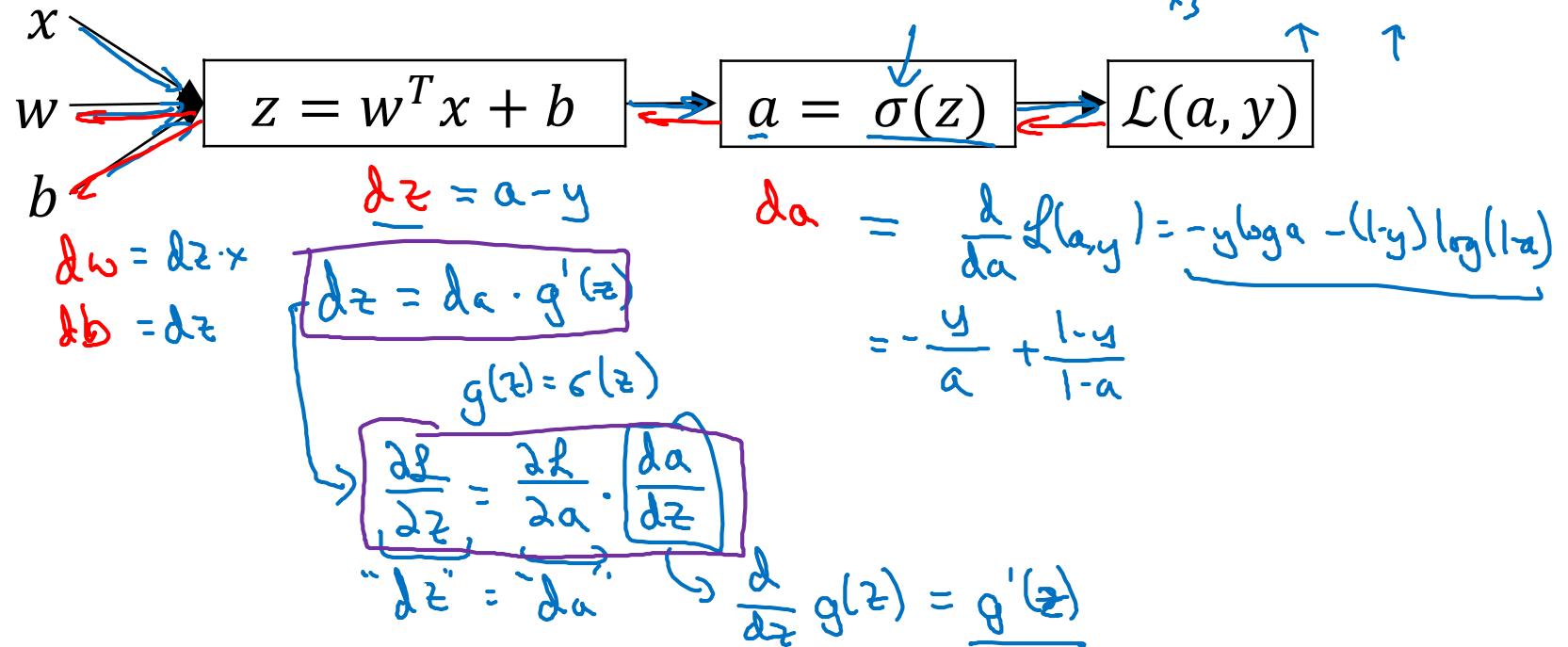
deeplearning.ai

One hidden layer Neural Network

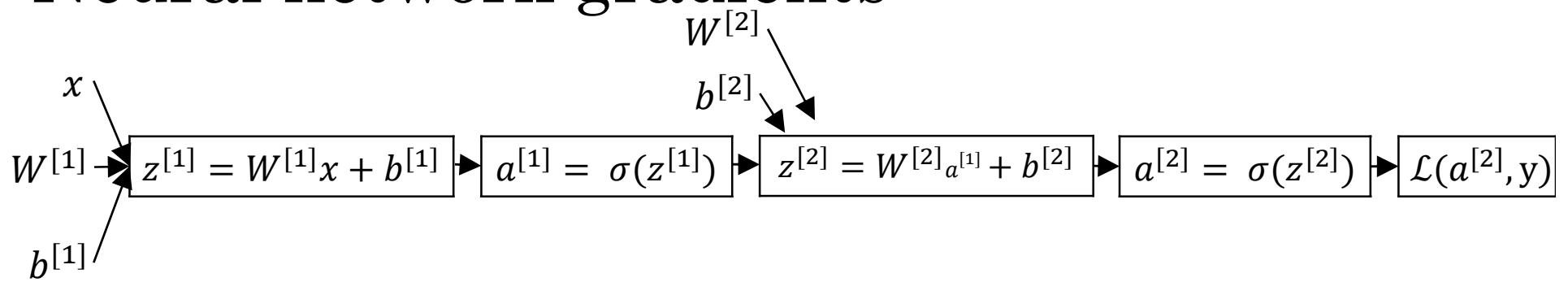
Backpropagation intuition (Calculation)

Computing gradients

Logistic regression

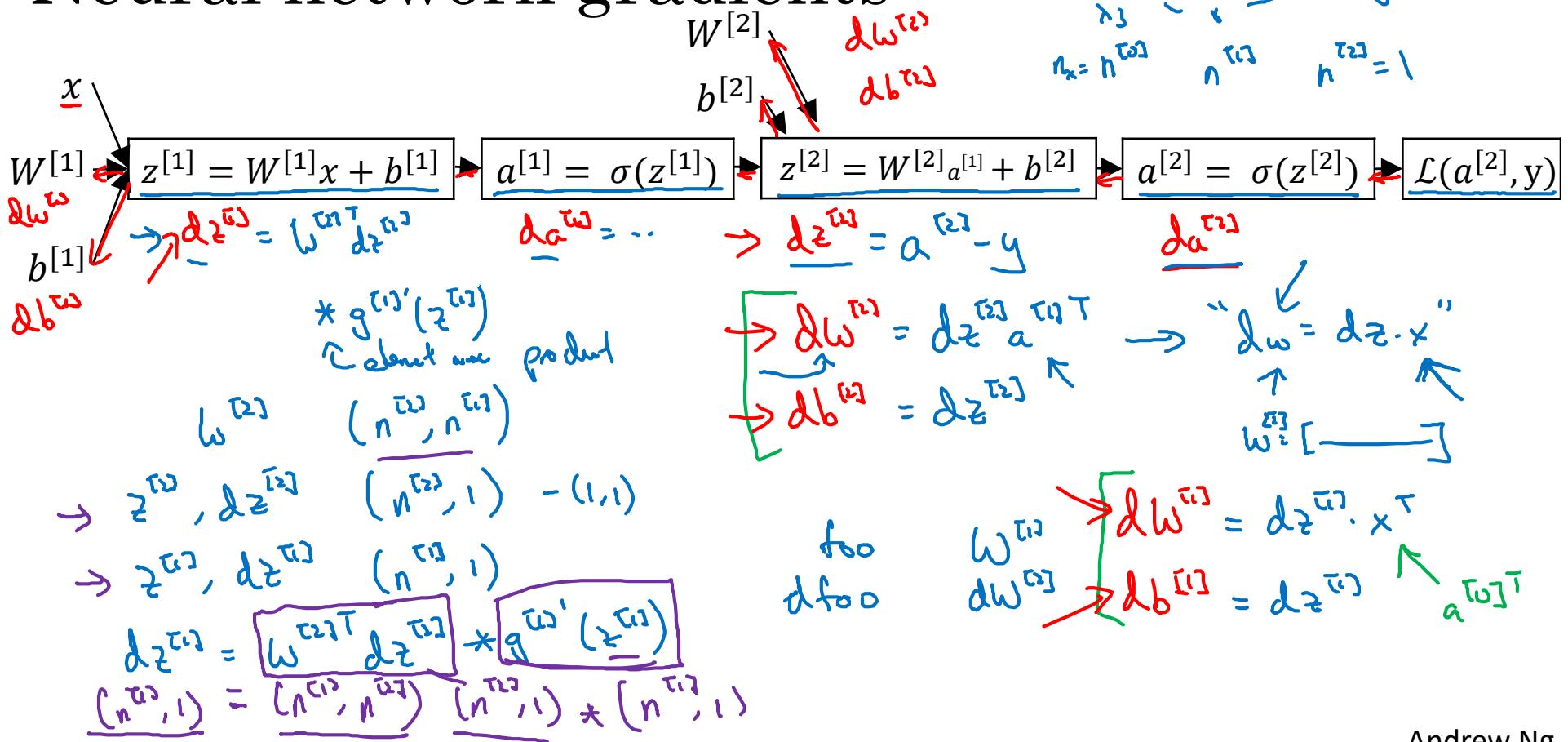


Neural network gradients



Andrew Ng

Neural network gradients



Andrew Ng

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

Vectorized implementation:

$$\begin{array}{l} \downarrow \\ z^{[1]} = w^{[1]} x + b^{[1]} \end{array}$$

$$\nearrow a^{[1]} = g^{[1]}(z^{[1]})$$

$$\begin{array}{c} \downarrow \\ z^{[1]} = \begin{bmatrix} | & | & | & \dots & | \\ z^{1} & z^{[1](2)} & \dots & z^{[1](n)} \end{bmatrix} \end{array}$$

$$\begin{array}{l} \downarrow \\ z^{[1]} = w^{[1]} x + b^{[1]} \end{array}$$

$$\nearrow A^{[1]} = g^{[1]}(z^{[1]})$$

Andrew Ng

Summary of gradient descent

$$\underline{dz^{[2]}} = \underline{a^{[2]}} - \underline{y}$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$

(n^{T₁}, 1)

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

$$\begin{aligned} dZ^{[2]} &= A^{[2]} - Y \\ dW^{[2]} &= \frac{1}{m} dZ^{[2]} A^{[1]T} \end{aligned}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})}_{\text{elementwise product}}$$

(n^{T₂}, m) (n^{T₂}, m) (n^{T₁}, m)

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}_i, y_i)$$

Andrew Ng

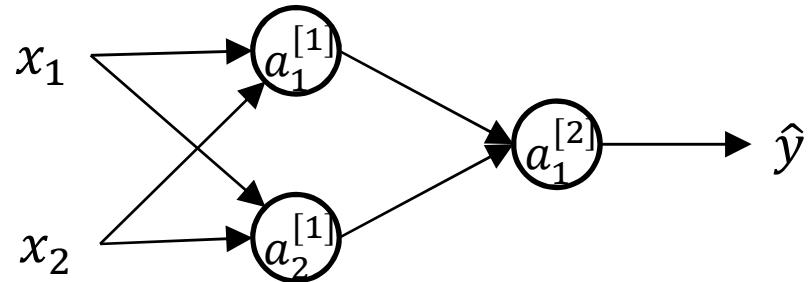


deeplearning.ai

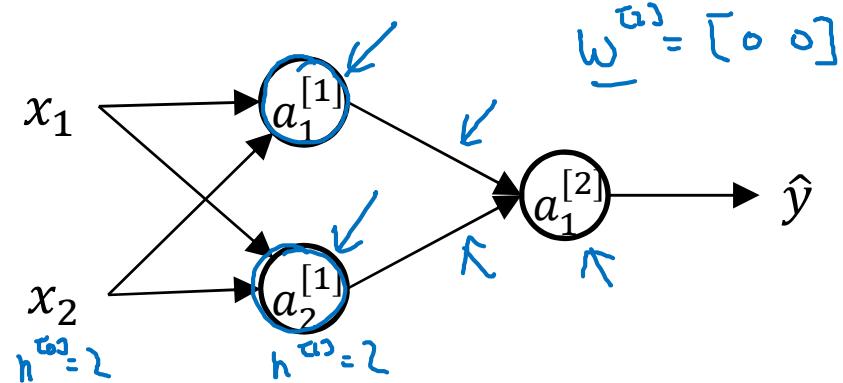
One hidden layer Neural Network

Random Initialization

What happens if you initialize weights to zero?



What happens if you initialize weights to zero?



$$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

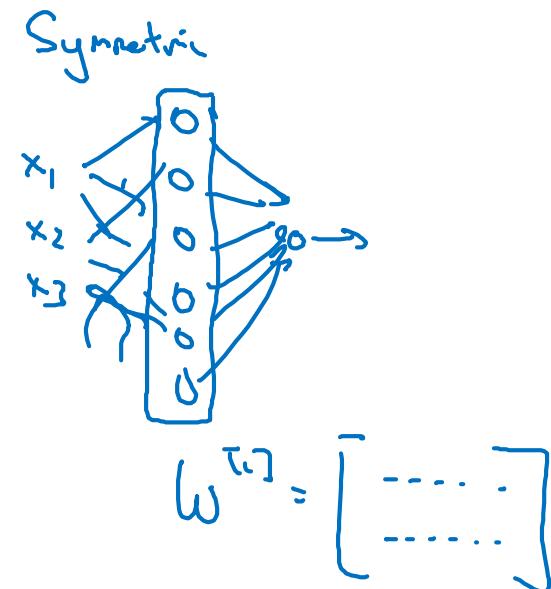
$$a_1^{[1]} = a_2^{[1]}$$

$$\Delta w = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

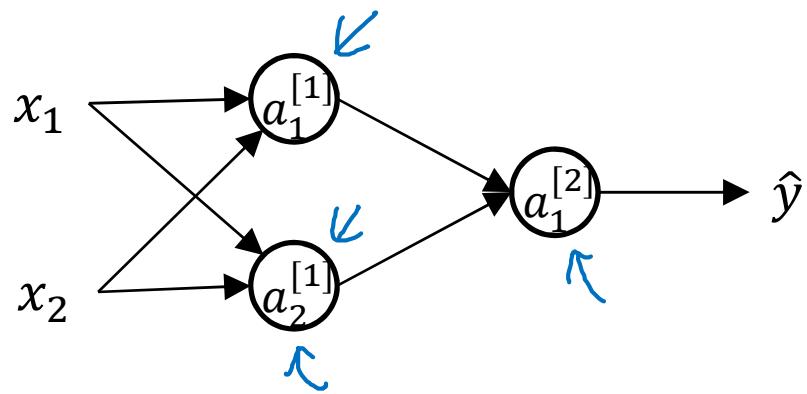
$$b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Delta z_1^{[1]} = \Delta z_2^{[1]}$$

$$w^{[1]} = w^{[1]} - \lambda \Delta w$$



Random initialization



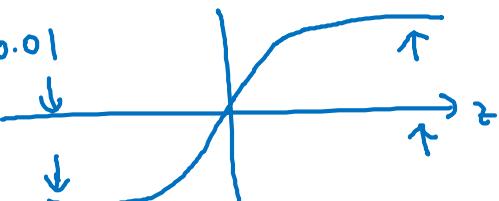
$$\rightarrow \omega^{[1]} = \text{np.random.randn}(2, 2) * \frac{0.01}{100?}$$

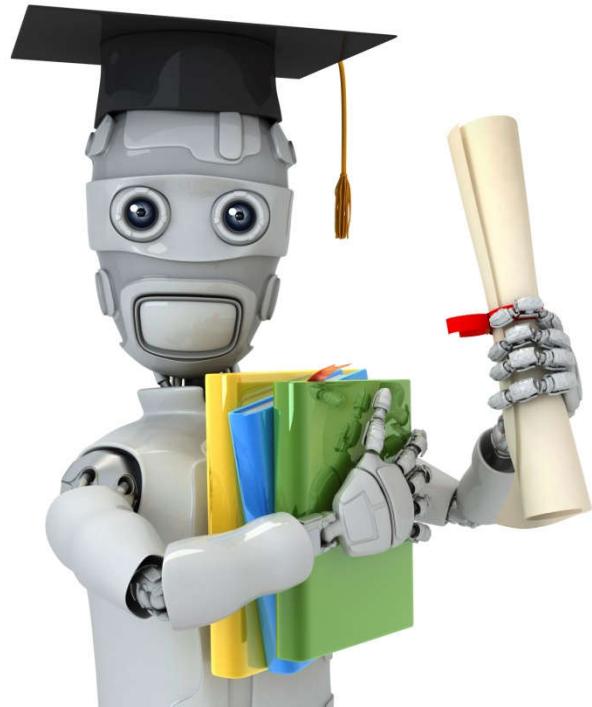
$$b^{[1]} = \text{np.zeros}(2, 1)$$

$$\omega^{[2]} = \text{np.random.randn}(1, 2) * 0.01$$

$$b^{[2]} = 0$$

$$\begin{aligned} z^{[1]} &= \omega^{[1]} x + b^{[1]} \\ a^{[1]} &= g^{[1]}(z^{[1]}) \end{aligned}$$





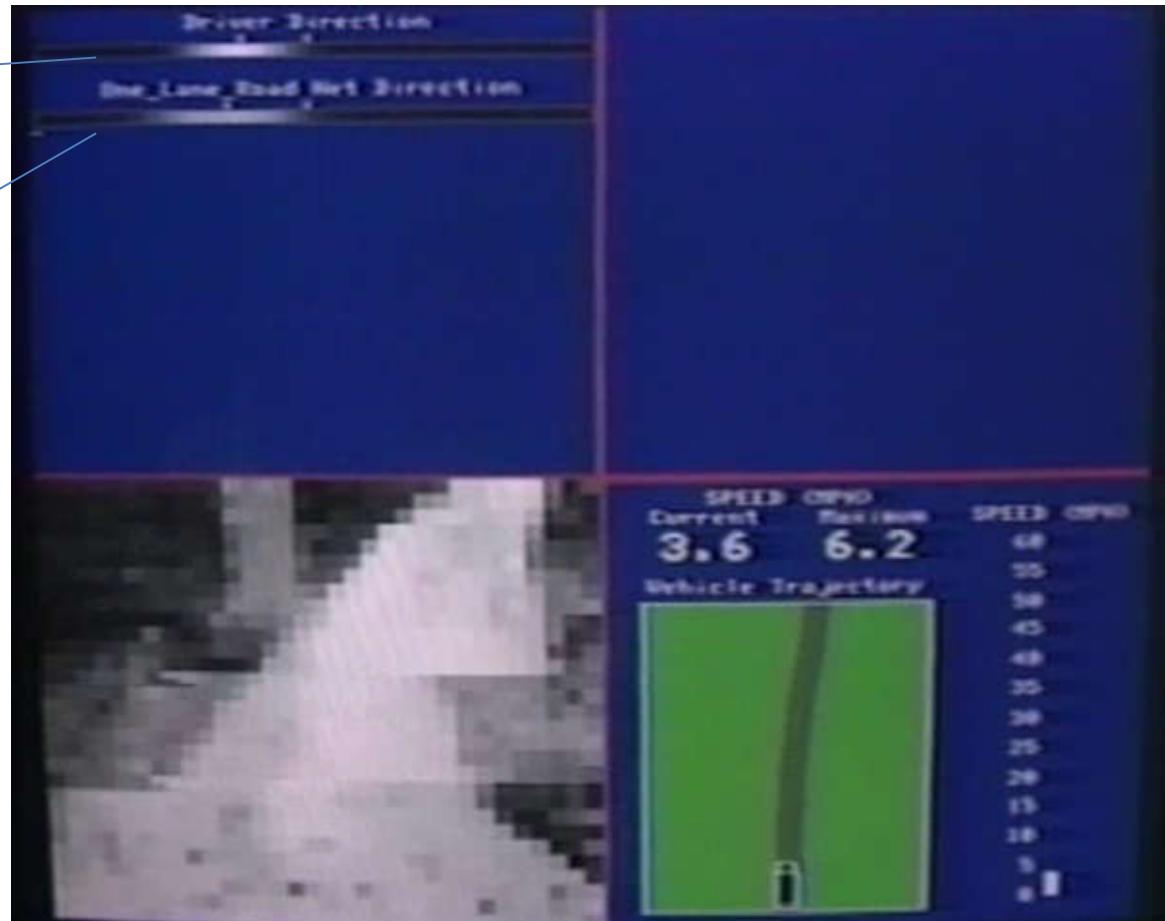
Machine Learning

Neural Networks: Learning

Backpropagation
example: Autonomous
driving

Direction chosen
by human driver

Direction selected
by learning
algorithm



[Courtesy of Dean Pomerleau]

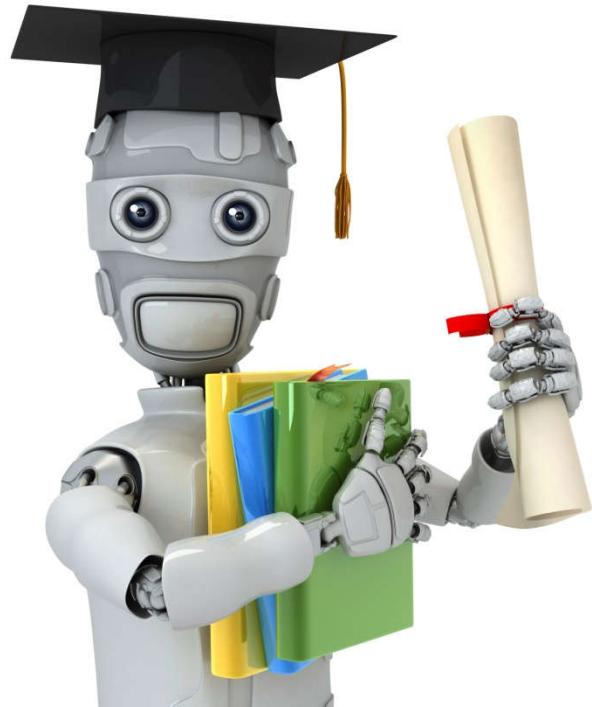


CSE 4621 Machine Learning

Lecture 9

Md. Hasanul Kabir, PhD.
Professor, CSE Department
Islamic University of Technology (IUT)





Machine Learning

Advice for applying machine learning

Evaluating a Learning Algorithm

Source & Special Thanks to (Coursera) Machine Learning / NN&DL Courses

Debugging a learning algorithm:

Suppose you have implemented regularized linear regression to predict housing prices.

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

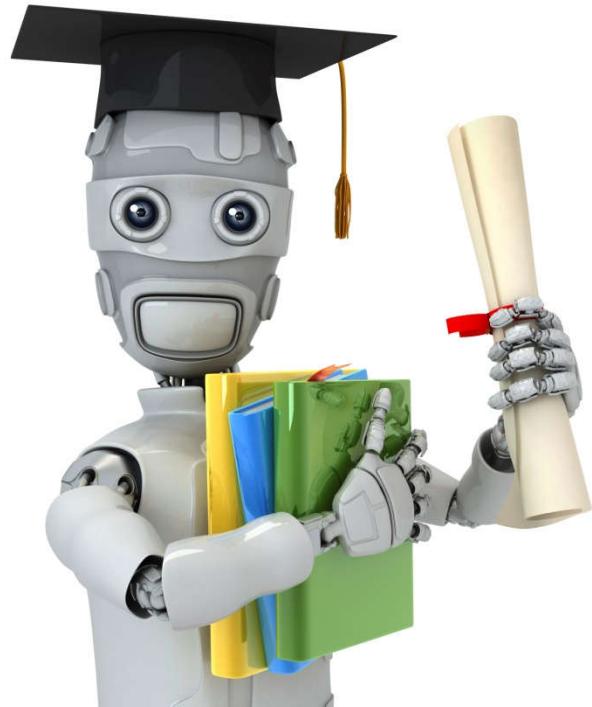
However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features (x_1^2, x_2^2, x_1x_2 ,etc.)
- Try decreasing λ
- Try increasing λ

Machine learning diagnostic:

Diagnostic: A test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

Diagnostics can take time to implement, but doing so can be a very good use of your time.



Machine Learning

Advice for applying machine learning

Evaluating a hypothesis

Evaluating your hypothesis



$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Fails to generalize to new examples not in training set.

x_1 = size of house

x_2 = no. of bedrooms

x_3 = no. of floors

x_4 = age of house

x_5 = average income in neighborhood

x_6 = kitchen size

⋮

x_{100}

Evaluating your hypothesis

Dataset:

Size	Price	
2104	400	$(x^{(1)}, y^{(1)})$
1600	330	$(x^{(2)}, y^{(2)})$
2400	369	\vdots
1416	232	
3000	540	$(x^{(m)}, y^{(m)})$
1985	300	
1534	315	
1427	199	\rightarrow
1380	212	$(x_{test}^{(1)}, y_{test}^{(1)})$
1494	243	$(x_{test}^{(2)}, y_{test}^{(2)})$
		\vdots
		$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

Training/testing procedure for linear regression

- - Learn parameter $\underline{\theta}$ from training data (minimizing training error $J(\theta)$) 70%
- Compute test set error:

$$J_{\text{test}}(\theta) = \frac{1}{2m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} \left(h_{\theta}(x^{(i)}_{\text{test}}) - y^{(i)}_{\text{test}} \right)^2$$

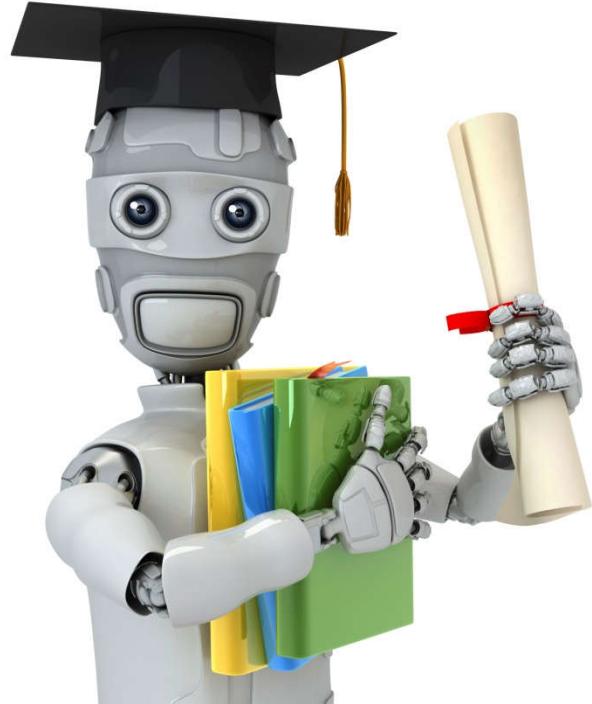
Training/testing procedure for logistic regression

- Learn parameter θ from training data

- Compute test set error:

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log (1 - h_\theta(x_{test}^{(i)}))$$

- Misclassification error (0/1 misclassification error):

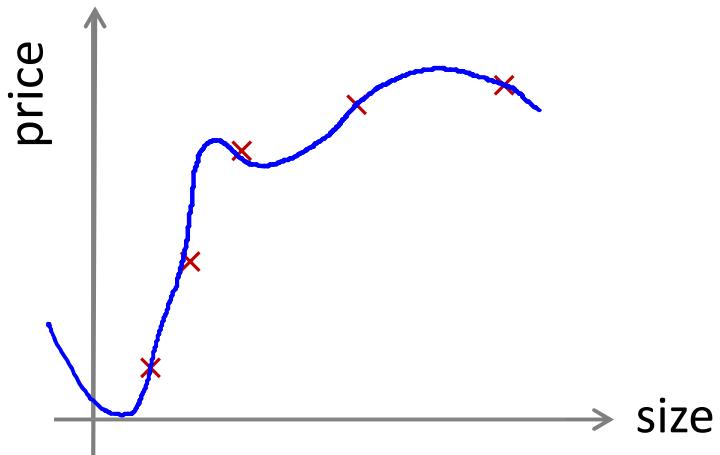


Machine Learning

Advice for applying machine learning

Model selection and
training/validation/test
sets

Overfitting example



$$h_{\theta}(x) = \underline{\theta_0} + \underline{\theta_1}x + \underline{\theta_2}x^2 + \underline{\theta_3}x^3 + \underline{\theta_4}x^4$$

Once parameters $\theta_0, \theta_1, \dots, \theta_4$ were fit to some set of data (training set), the error of the parameters as measured on that data (the training error $J(\theta)$) is likely to be lower than the actual generalization error.

Model selection

$d = \text{degree of polynomial}$

$$d=1 \quad 1. \quad h_{\theta}(x) = \theta_0 + \theta_1 x \rightarrow \Theta^{(1)} \rightarrow J_{test}(\Theta^{(1)})$$

$$d=2 \quad 2. \quad h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \Theta^{(2)} \rightarrow J_{test}(\Theta^{(2)})$$

$$d=3 \quad 3. \quad h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_3 x^3 \rightarrow \Theta^{(3)} \rightarrow J_{test}(\Theta^{(3)})$$

:

:

$$d=10 \quad 10. \quad h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10} \rightarrow \Theta^{(10)} \rightarrow J_{test}(\Theta^{(10)})$$

Choose $\theta_0 + \dots + \theta_5 x^5$ ←

How well does the model generalize? Report test set

error $J_{test}(\theta^{(5)})$. ←

$(\theta_0, \theta_1, \dots)$

↑

Problem: $J_{test}(\theta^{(5)})$ is likely to be an overly optimistic estimate of generalization error. I.e. our extra parameter ($d = \text{degree of polynomial}$) is fit to test set.

Evaluating your hypothesis

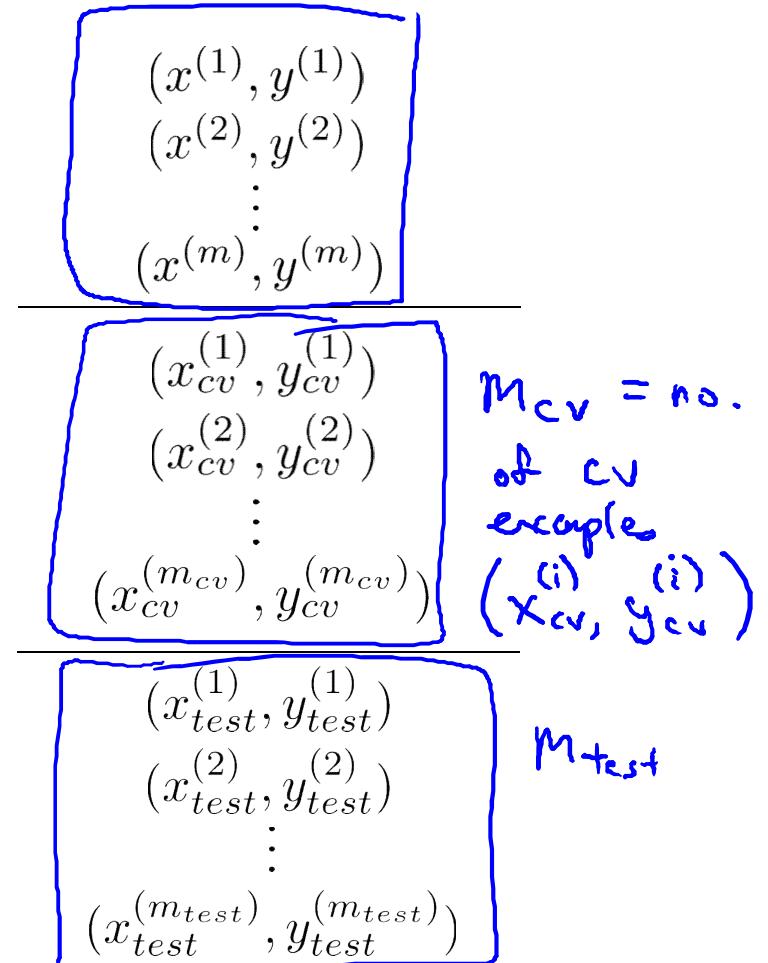
Dataset:

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
<hr/>	
1534	315
1427	199
<hr/>	
1380	212
1494	243

Training set

Cross validation
set (CV)

test set



Train/validation/test error

Training error:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

Test error:

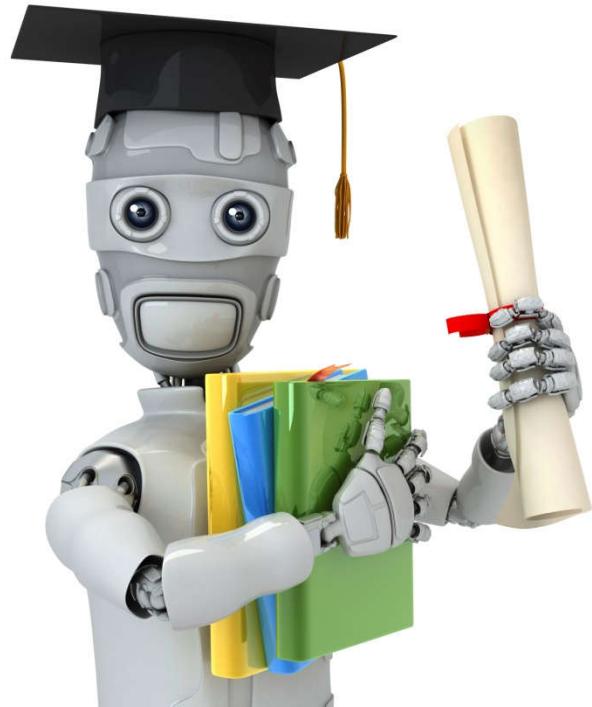
$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

Model selection

1. $h_{\theta}(x) = \theta_0 + \theta_1 x \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
2. $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
3. $h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_3 x^3 \rightarrow \theta^{(3)} \vdots \vdots J_{cv}(\theta^{(3)})$
- ⋮
10. $h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10} \rightarrow \theta^{(10)} \rightarrow J_{cv}(\theta^{(10)})$
- $d=4$ ↑

Pick $\theta_0 + \theta_1 x_1 + \dots + \theta_4 x^4$ ←

Estimate generalization error for test set $J_{test}(\theta^{(4)})$ ←

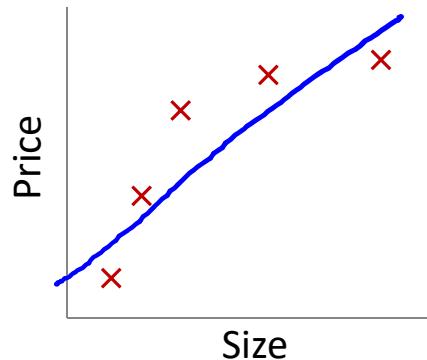


Machine Learning

Advice for applying machine learning

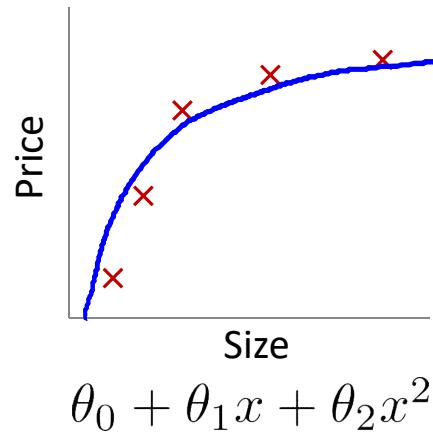
Diagnosing bias vs. variance

Bias/variance



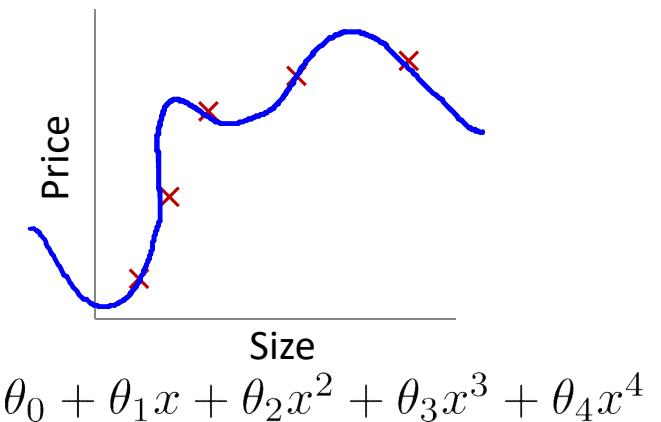
High bias
(underfit)

$d=1$



“Just right”

$d=2$



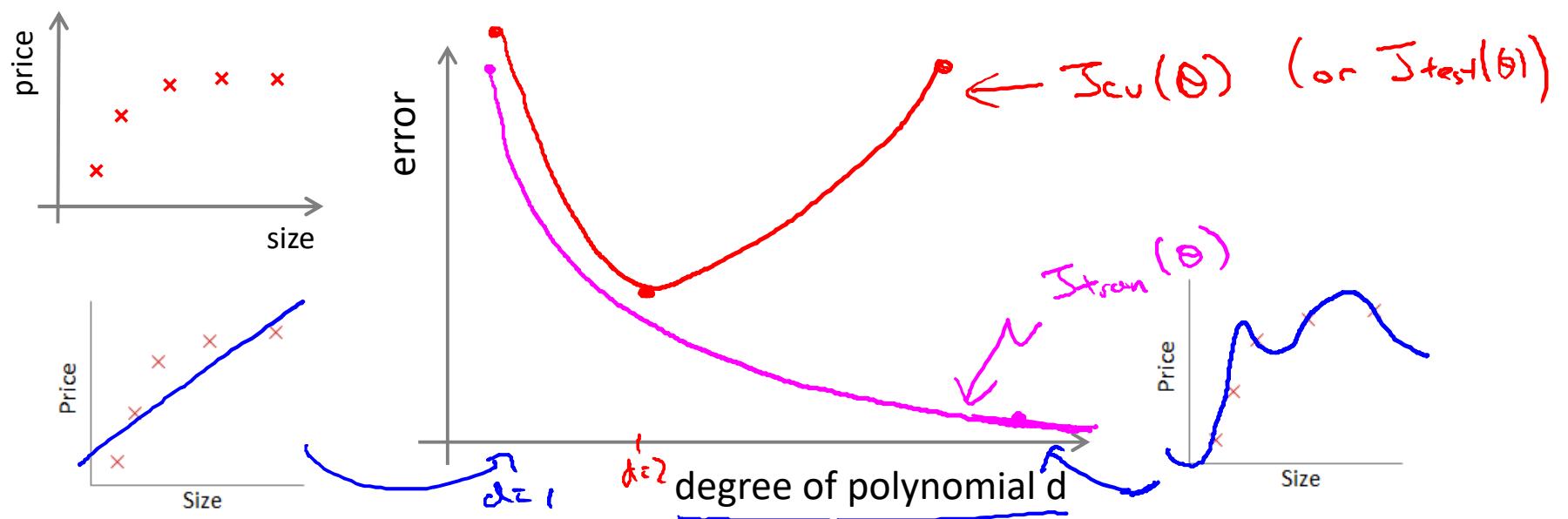
High variance
(overfit)

$d=4$

Bias/variance

Training error: $\underline{J_{train}(\theta)} = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$

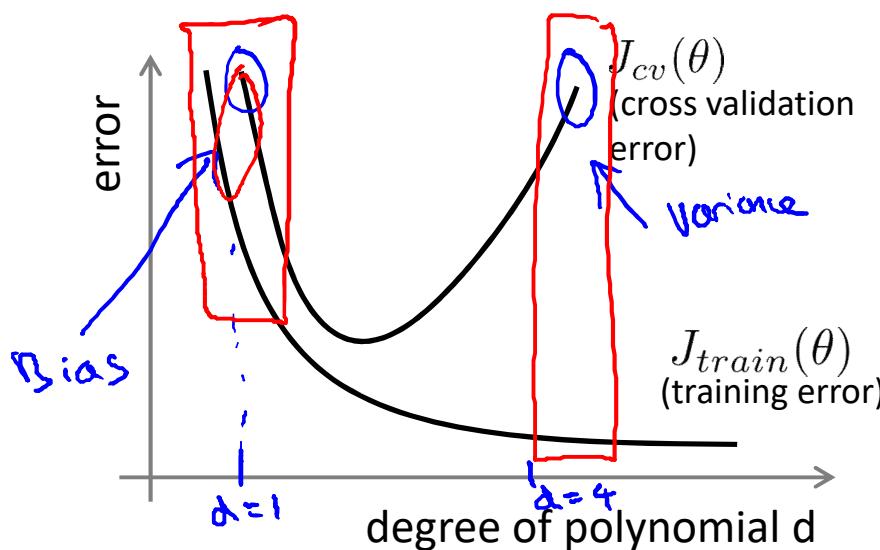
Cross validation error: $\underline{J_{cv}(\theta)} = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$ (or $J_{test}(\theta)$)



Andrew Ng

Diagnosing bias vs. variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Bias (underfit):

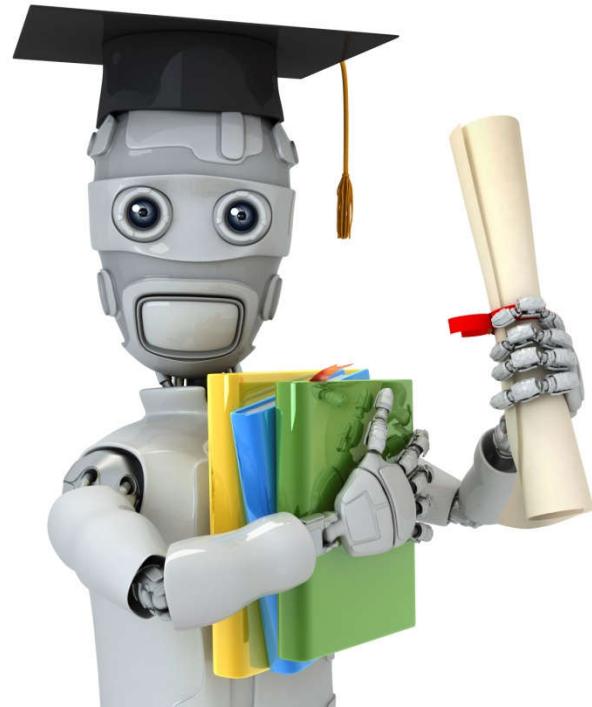
$\rightarrow J_{train}(\theta)$ will be high }
 $J_{cv}(\theta) \approx J_{train}(\theta)$ }

Variance (overfit):

$\rightarrow J_{train}(\theta)$ will be low }
 $J_{cv}(\theta) \gg J_{train}(\theta)$ }

>>

Andrew Ng



Machine Learning

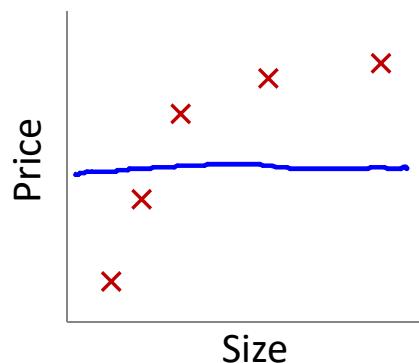
Advice for applying machine learning

Regularization and bias/variance

Linear regression with regularization

Model:
$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

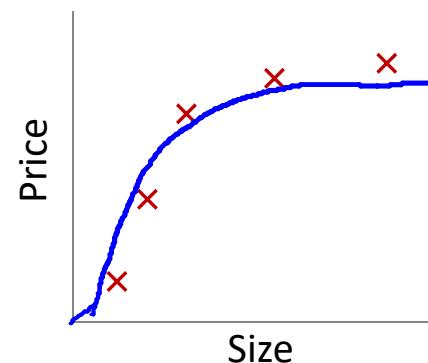
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$



Large λ ←

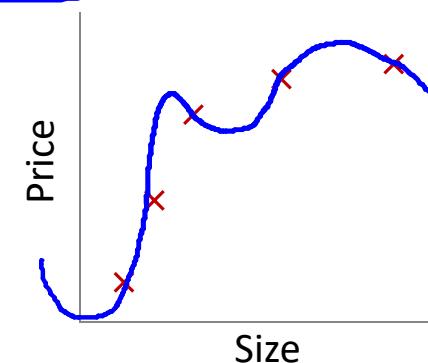
→ High bias (underfit)

→ $\lambda = 10000$. $\underline{\theta_1 \approx 0, \theta_2 \approx 0, \dots}$
 $h_{\theta}(x) \approx \theta_0$



Intermediate λ ←

“Just right”



→ Small λ

High variance (overfit)

→ $\lambda = 0$

Choosing the regularization parameter λ

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

Choosing the regularization parameter λ

Model: $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$

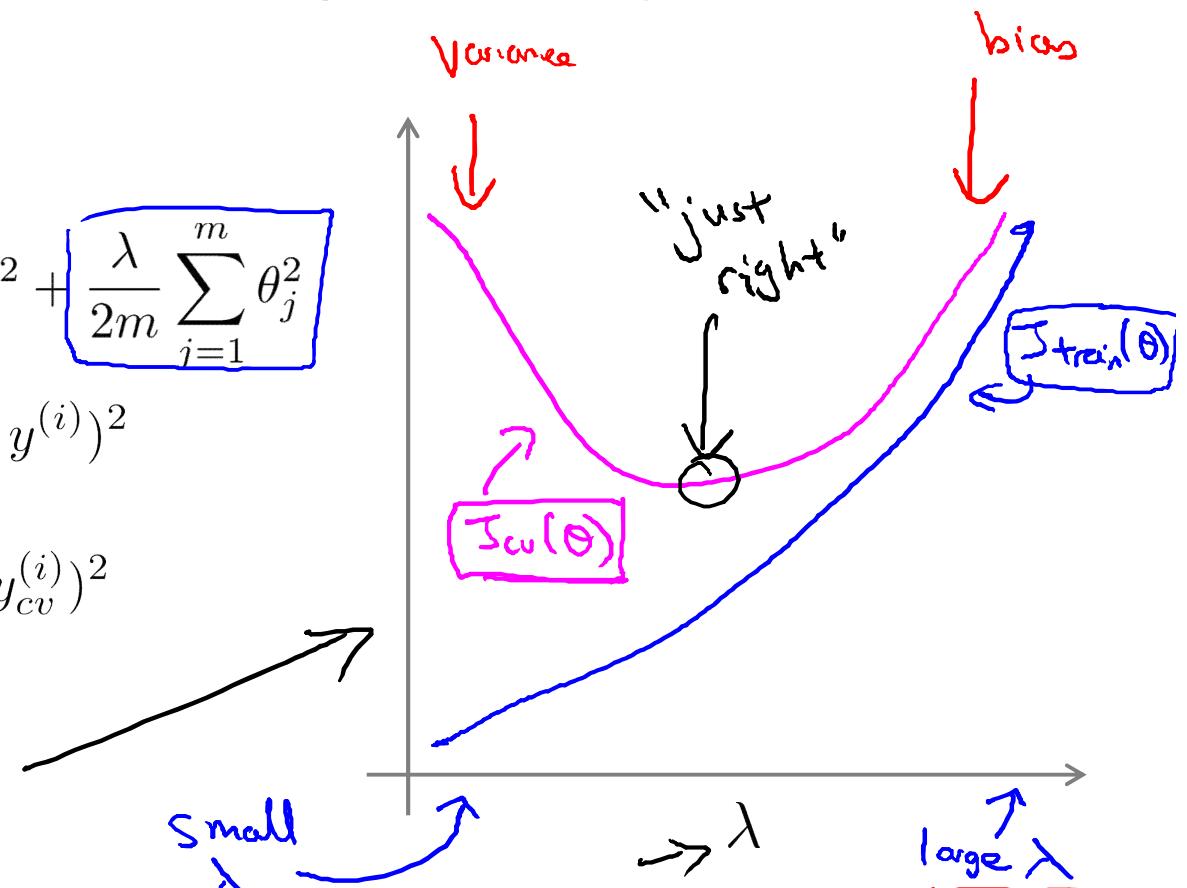
1. Try $\lambda = 0$ $\xrightarrow{\uparrow}$ $\rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(0)} \rightarrow J_{cv}(\theta^{(0)})$
2. Try $\lambda = 0.01$ $\rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
3. Try $\lambda = 0.02$ $\rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
4. Try $\lambda = 0.04$ \vdots
5. Try $\lambda = 0.08$ $\rightarrow \vdots \theta^{(5)} \rightarrow J_{cv}(\theta^{(5)})$
- ⋮
12. Try $\lambda = 10$ $\xrightarrow{\uparrow 10 \cdot 2^4} \theta^{(12)} \rightarrow J_{cv}(\theta^{(12)})$
Pick (say) $\theta^{(5)}$. Test error: $J_{test}(\theta^{(5)})$

Bias/variance as a function of the regularization parameter λ

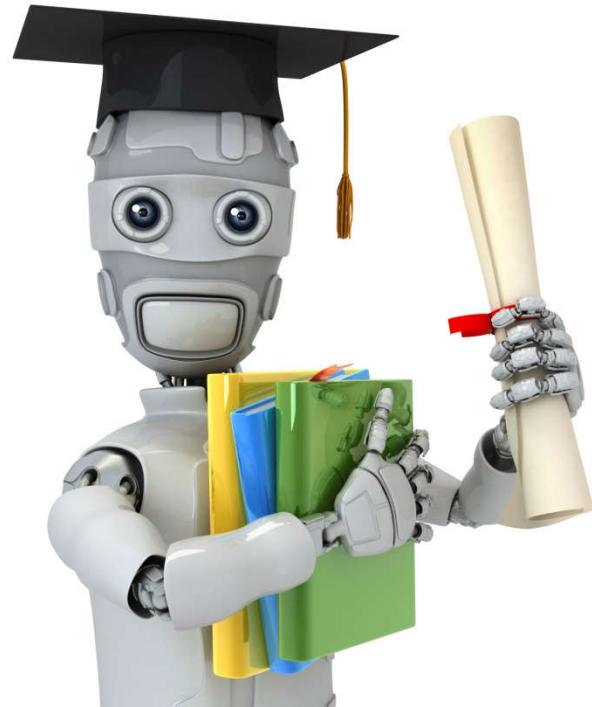
$$\rightarrow J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \boxed{\frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2}$$

$$\rightarrow \underline{J_{train}(\theta)} = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\rightarrow \boxed{J_{cv}(\theta)} = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



Andrew Ng



Machine Learning

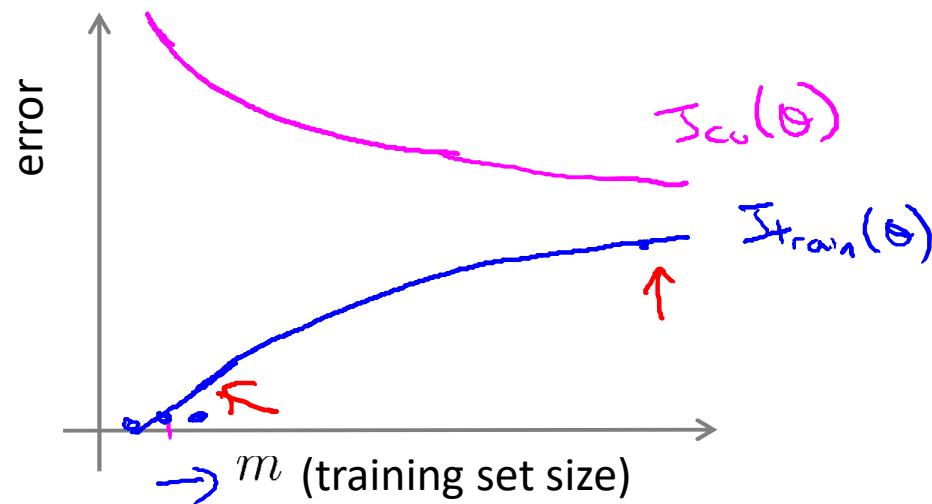
Advice for applying machine learning

Learning curves

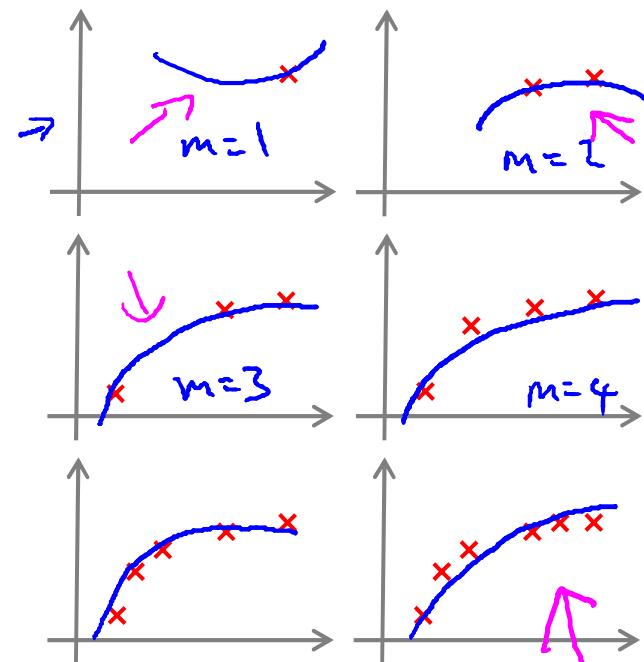
Learning curves

$$\rightarrow \underline{J_{train}(\theta)} = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

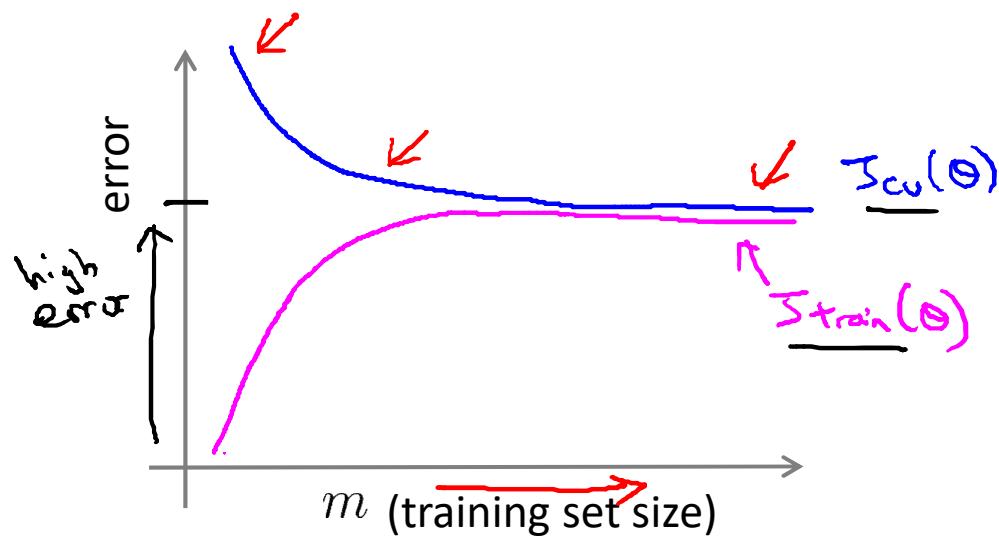
$$\cancel{\rightarrow} J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



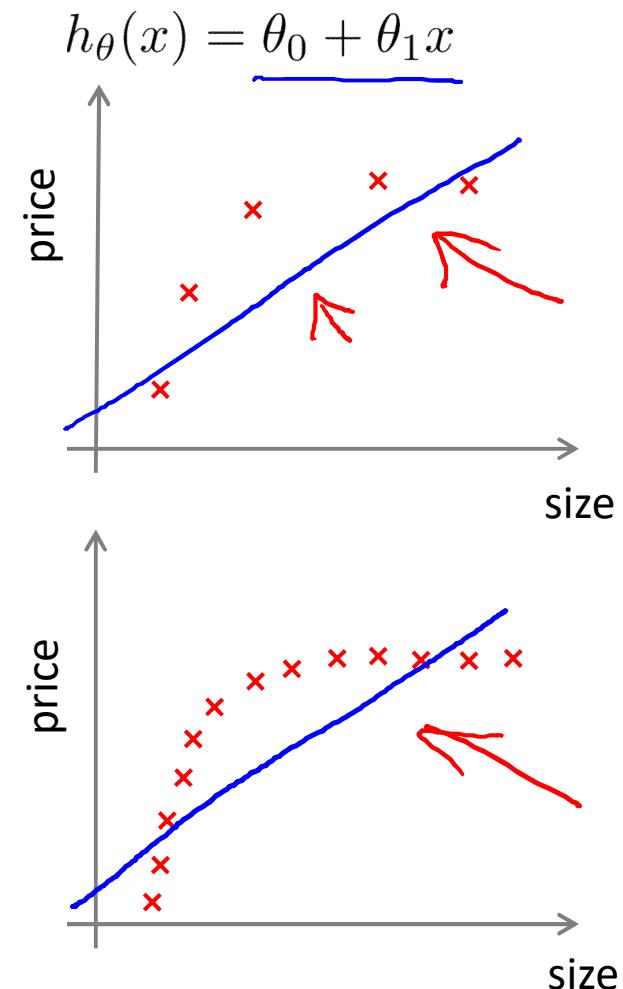
$$h_\theta(x) = \underline{\theta_0 + \theta_1 x + \theta_2 x^2}$$



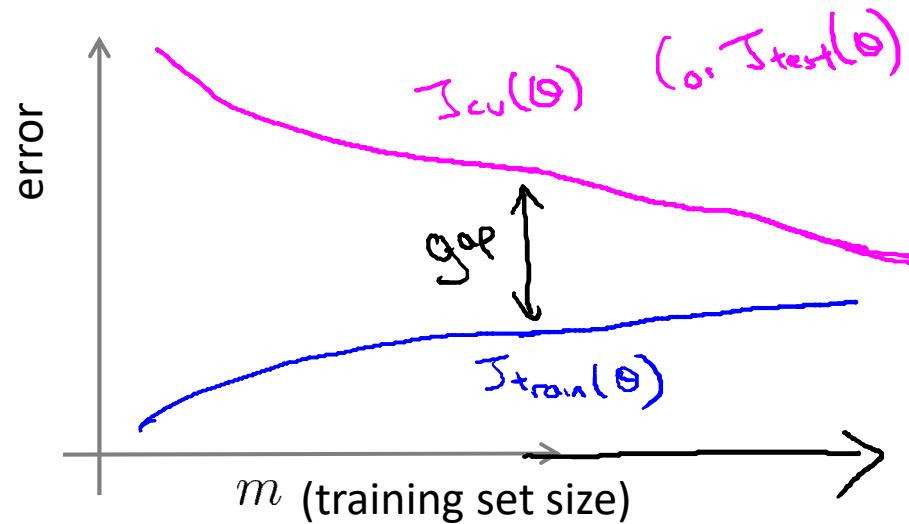
High bias



If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.



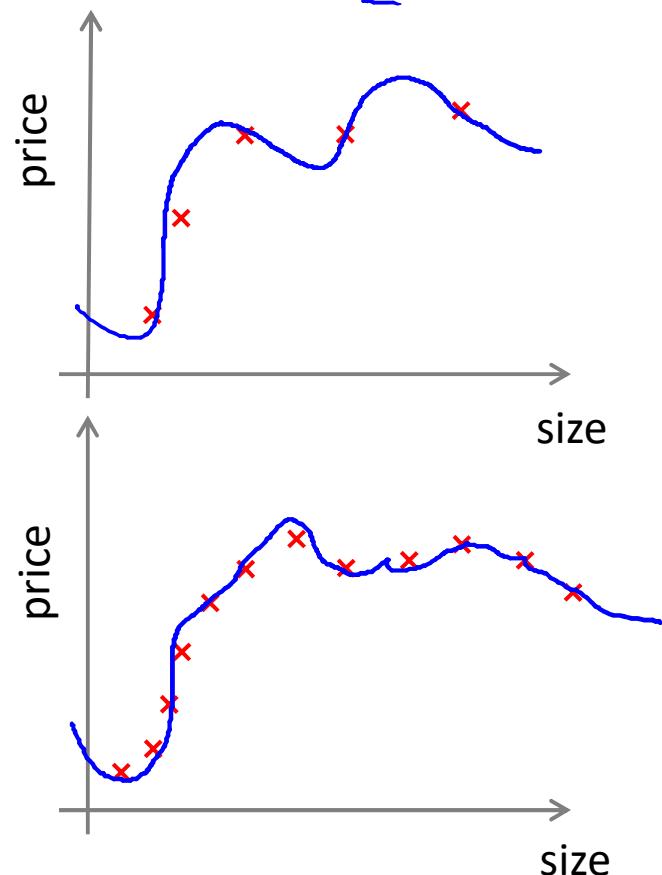
High variance



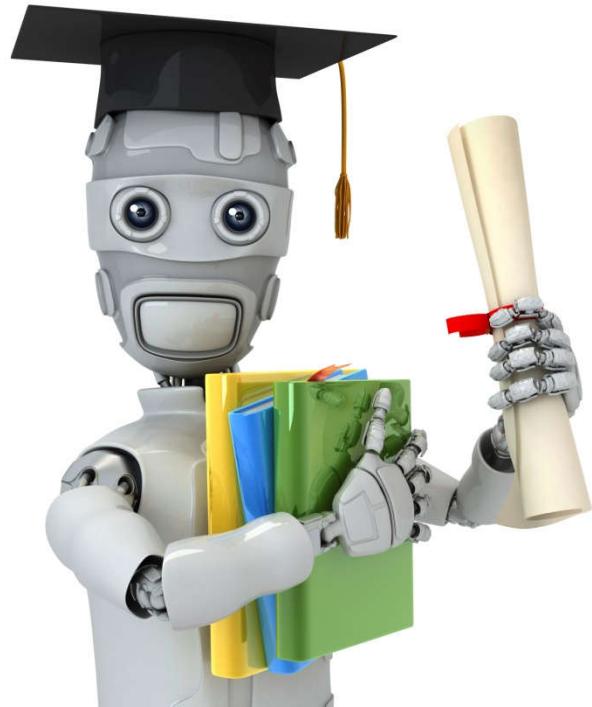
If a learning algorithm is suffering from high variance, getting more training data is likely to help. ↗

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \cdots + \theta_{100} x^{100}$$

(and small λ) ↗



Andrew Ng



Machine Learning

Advice for applying machine learning

Deciding what to try next (revisited)

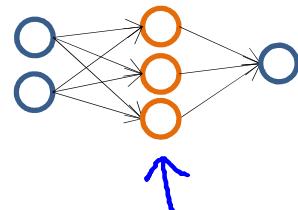
Debugging a learning algorithm:

Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features (x_1^2, x_2^2, x_1x_2 , etc) → fixes high bias.
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

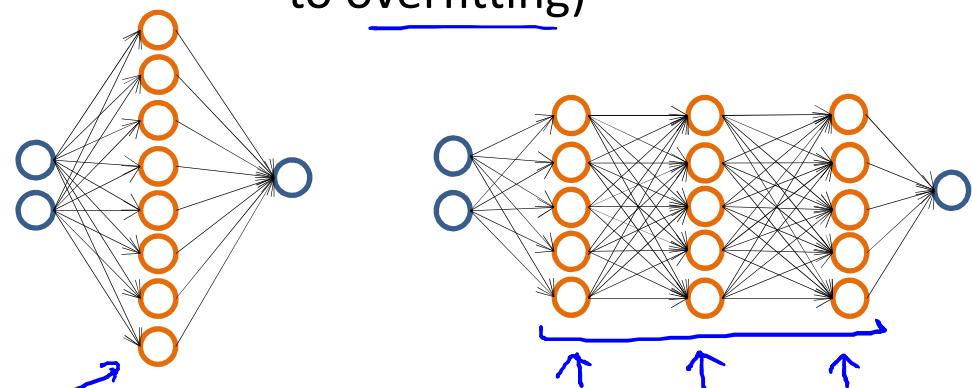
Neural networks and overfitting

→ “Small” neural network
(fewer parameters; more prone to underfitting)



Computationally cheaper

→ “Large” neural network
(more parameters; more prone to overfitting)

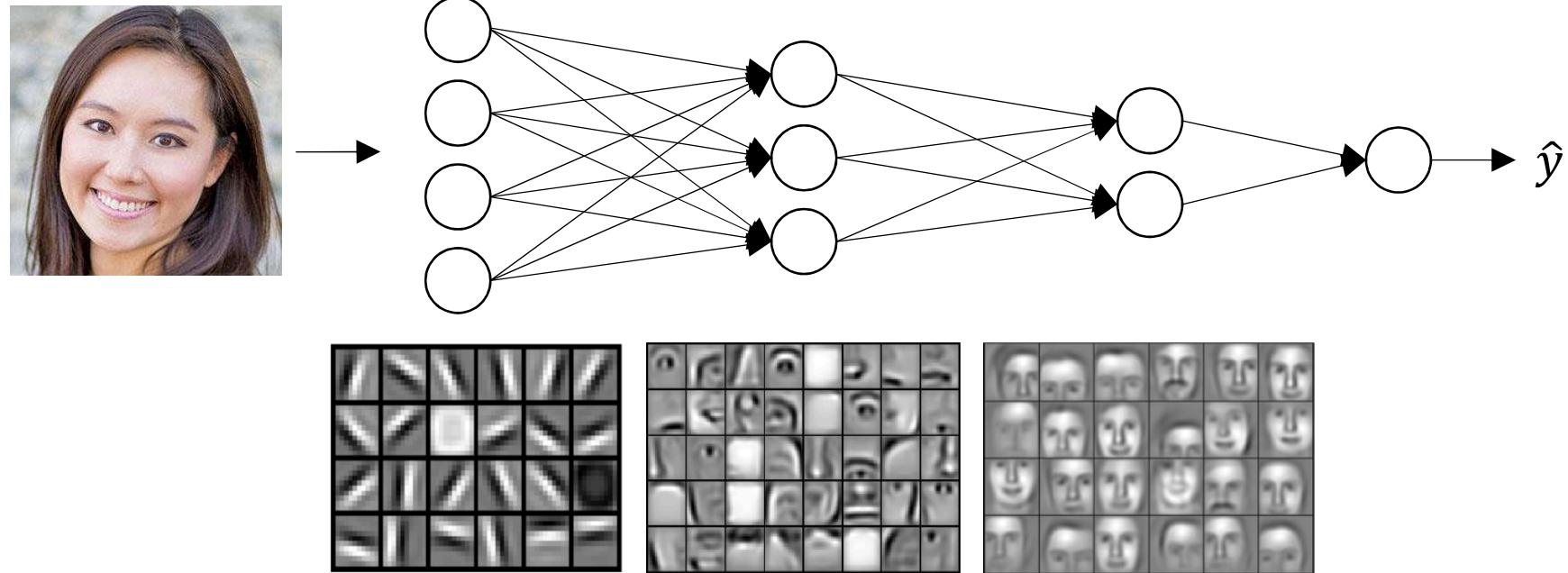


Computationally more expensive.

Use regularization (λ) to address overfitting.

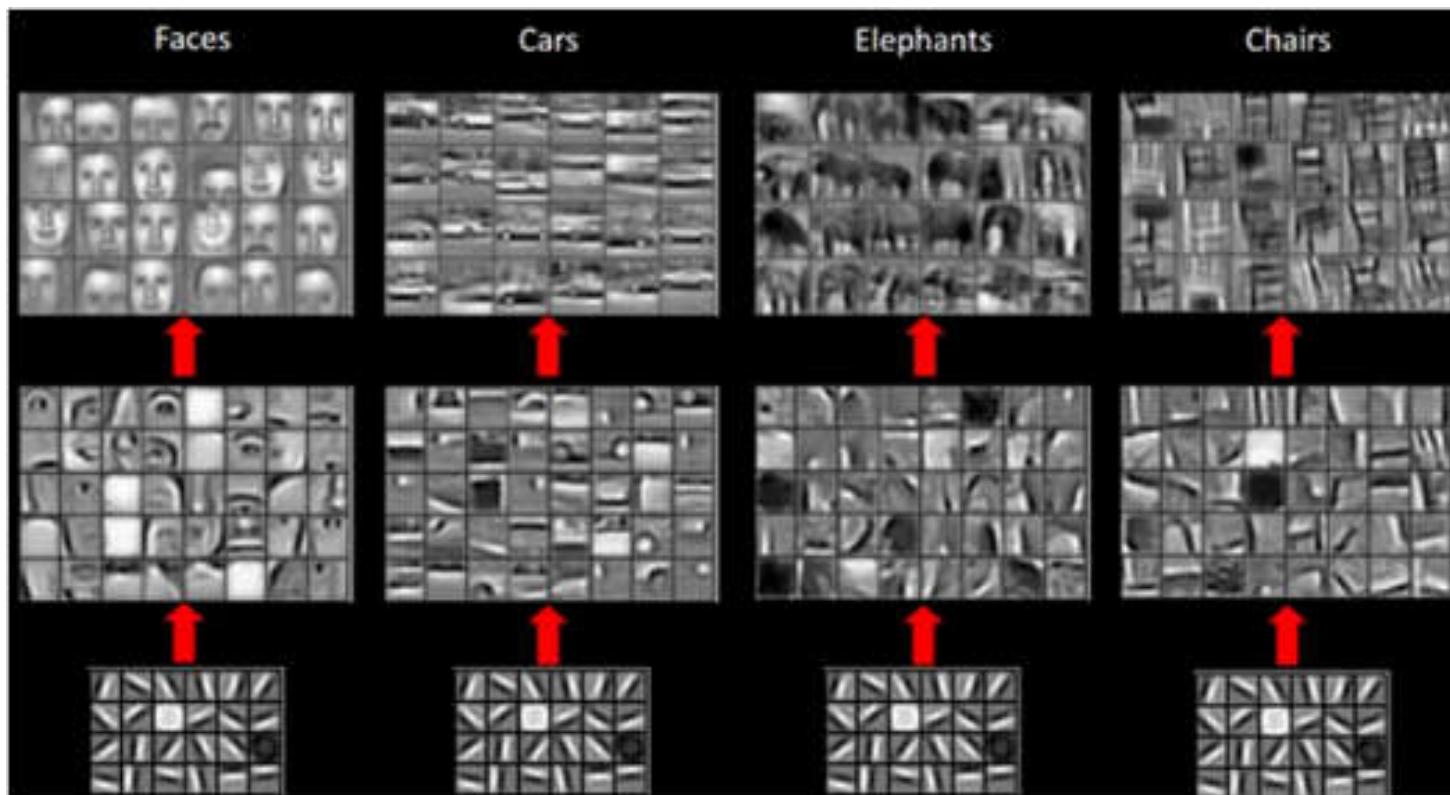
$$J_{\text{reg}}(\Theta)$$

Intuition about deep representation



Andrew Ng

Classification with Deep Neural Network



Andrew Ng



CSE 4621

Machine Learning

Lecture 10

Md. Hasanul Kabir, PhD.
Professor, CSE Department
Islamic University of Technology (IUT)





deeplearning.ai

Convolutional Neural Networks

Introduction

Source & Special Thanks to (Coursera) CNN Course (Deep Learning Specialization)

Computer Vision Problems

Image Classification



64x64

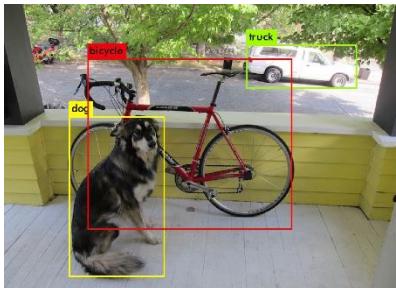


Cat? (0/1)

Neural Style Transfer



Object Detection



Deep Learning on large images



→ Cat? (0/1)

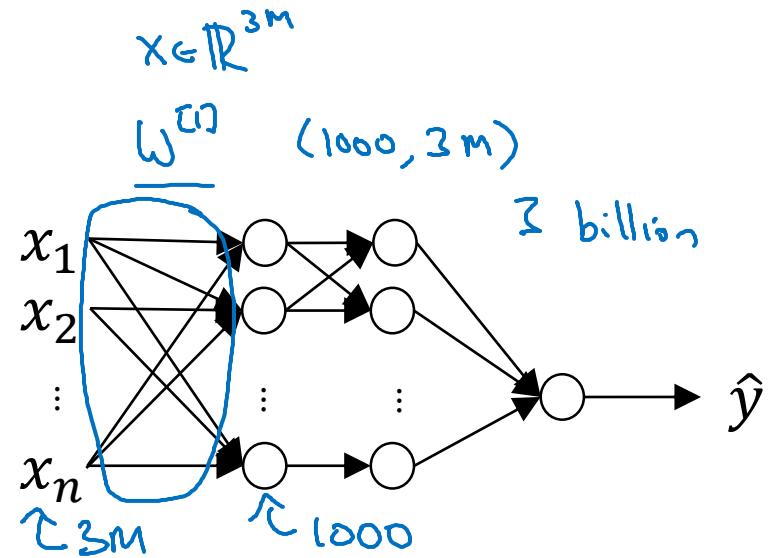
12288

$64 \times 64 \times 3$



$1000 \times 1000 \times 3$
= 3 million

- Learning 3 billion parameters for just one layer is too computationally expensive.
- **Convolution layers** provide solution to this problem.



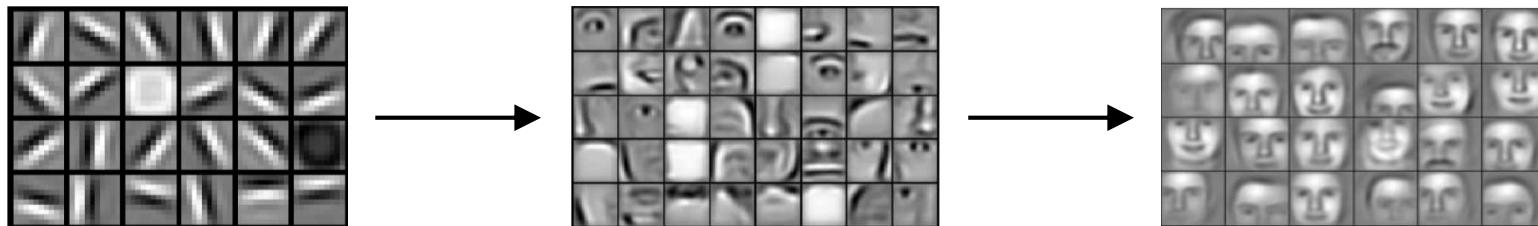


deeplearning.ai

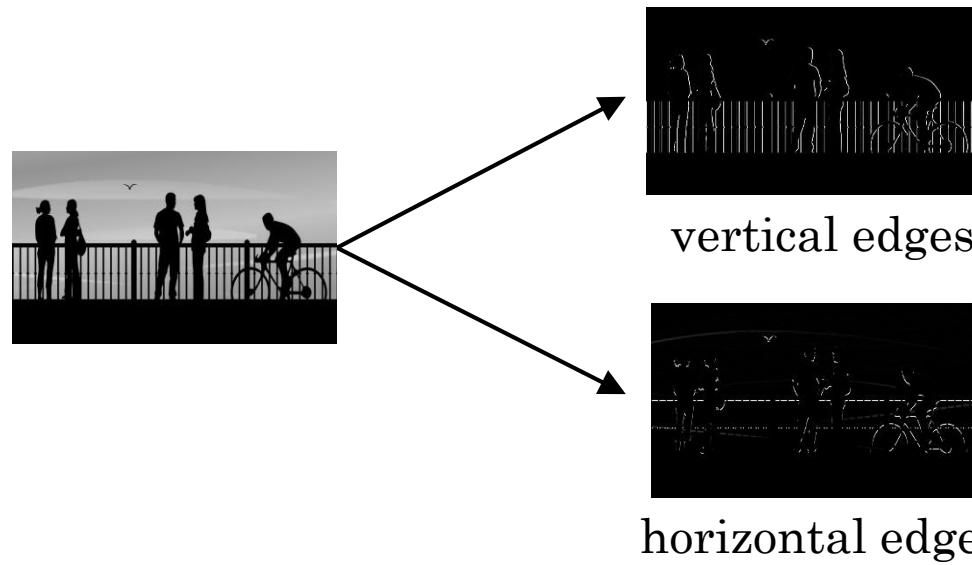
Convolutional Neural Networks

Edge detection with
Convolution

Feature Extraction in Computer Vision



Edge detection is a basic example of **convolution** operation that is a fundamental element in the **convolution layers**.



Convolution Operation (Step1)

<i>Input</i>					
4	9	2	5	8	3
5	6	2	4	0	3
2	4	5	4	5	2
5	6	5	4	7	8
5	7	7	9	2	1
5	8	5	3	8	4

$$n_H \times n_W = 6 \times 6$$

<i>Filter</i>					
1	0	-1			
1	0	-1			
1	0	-1			

*

<i>Result</i>					
2					

Parameters:

Size: $f = 3$

Stride: $s = 1$

Padding: $p = 0$

$$\boxed{2} = 4*1 + 9*0 + 2*(-1) + \\ 5*1 + 6*0 + 2*(-1) + \\ 2*1 + 4*0 + 5*(-1)$$

Convolution Operation (Step 2)

<i>Input</i>						
4	9	2	5	8	3	
6	2	4	0	3		
2	4	5	4	5	2	
5	6	5	4	7	8	
5	7	7	9	2	1	
5	8	5	3	8	4	

$$n_H \times n_W = 6 \times 6$$

<i>Filter</i>						
1	0	-1				
1	0	-1				
1	0	-1				

*

<i>Result</i>						
2	6					

Parameters:

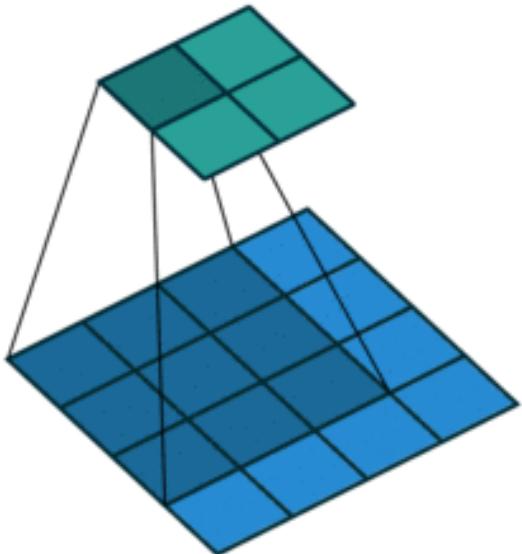
Size: $f = 3$

Stride: $s = 1$

Padding: $p = 0$

$$\boxed{6} = 9*1 + 2*0 + 5*(-1) + \\ 6*1 + 2*0 + 4*(-1) + \\ 4*1 + 5*0 + 4*(-1)$$

Why Convolution Operation?

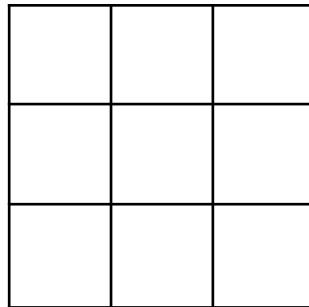


- **Parameter sharing:** A kernel is shared among every section of the input. For example, an edge detector is useful in detecting edges at any part of the image, with just few numbers.
- **Sparsity of connections:** each element of the output depends only on the small section of the input.

Vertical edge detection

3 ¹	0 ⁰	1 ⁻¹	2 ¹	7 ⁻⁰	4 ⁻¹
1 ¹	5 ⁰	8 ⁻¹	9 ⁻¹	3 ⁻⁰	1 ⁻¹
2 ⁻¹	7 ⁰	2 ⁻¹	5 ⁻¹	1 ⁻⁰	3 ⁻¹
0 ¹	1 ⁰	3 ⁻¹	1 ⁻¹	7 ⁻⁰	8 ⁻¹
4	2	1	6	2	8
2	4	5	2	3	9

*



=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

Vertical edge detection

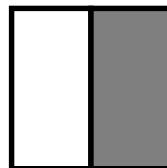
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

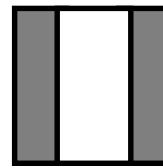
1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



*



Goal: Learning to detect edges

1	0	-1
1	0	-1
1	0	-1



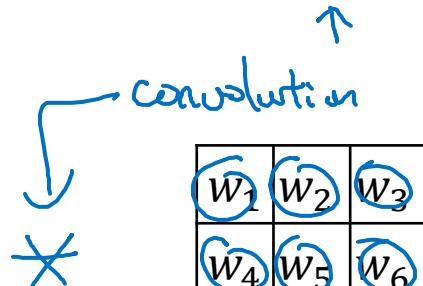
→

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

convolution



3×3

=

45°
 70°
 73°

3	0	-3
10	0	-10
3	0	-3

Scharr filter







deeplearning.ai

Convolutional Neural Networks

Padding

Padding

- Add extra zeros around.
- It allows us to use a CONV layer without necessarily shrinking the height and width of the volumes.

- This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers.

Input

0	0	0	0	0	0	0	0	0
0	4	9	2	5	8	3	0	0
0	5	6	2	4	0	3	0	0
0	2	4	5	4	5	2	0	0
0	5	6	5	4	7	8	0	0
0	5	7	7	9	2	1	0	0
0	5	8	5	3	8	4	0	0
0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

*

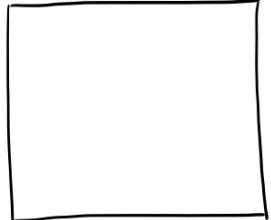
=

Parameters:

Size: $f = 3$
Stride: $s = 2$
Padding: $p = 1$

Dimension: 6×6

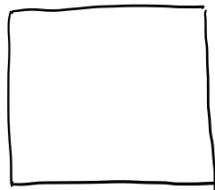
Valid Padding vs. Same Padding



*



=

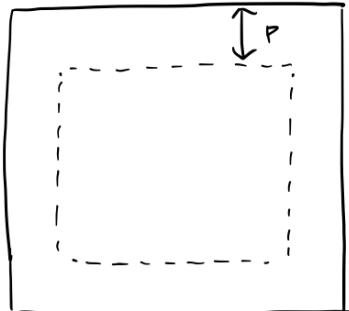


$$n \times n$$

$$f \times f$$

$$(n-f+1) \times (n-f+1)$$

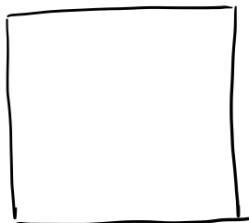
"VALID" CONV : $p = 0$



*



=



"SAME" CONV : $p = \frac{f-1}{2}$

$$(n+2p) \times (n+2p)$$

$$f \times f$$

$$(n+2p-f+1) \times (n+2p-f+1)$$

Valid and Same Convolutions

→ no padding

“Valid”: $n \times n$ \ast $f \times f$ $\rightarrow \frac{n-f+1}{4} \times \frac{n-f+1}{4}$

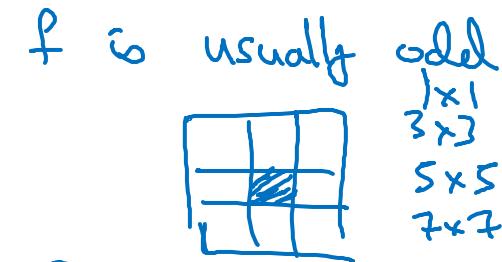
6×6 \ast 3×3 $\rightarrow 4 \times 4$

“Same”: Pad so that output size is the same as the input size.

$$n + 2p - f + 1 \times n + 2p - f + 1$$

$$n + 2p - f + 1 = n \Rightarrow p = \frac{f-1}{2}$$

$$3 \times 3 \quad p = \frac{3-1}{2} = 1 \quad \left| \begin{array}{c} 5 \times 5 \\ f=5 \end{array} \right. \quad p=2$$



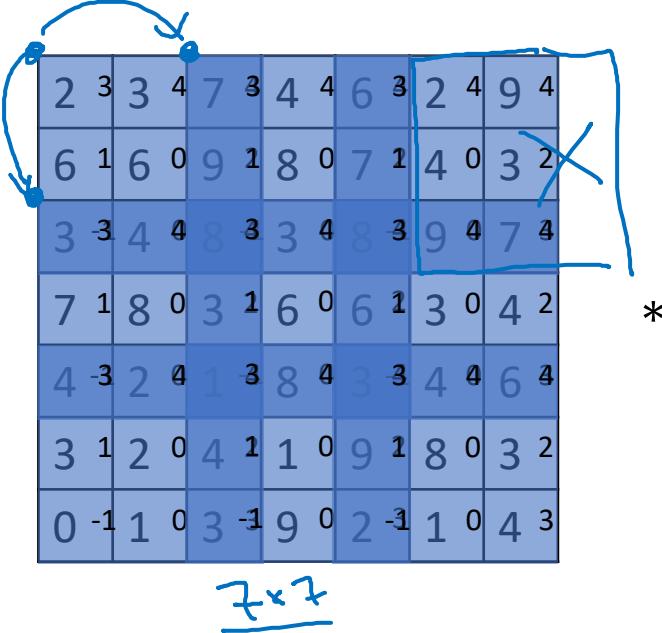


deeplearning.ai

Convolutional Neural Networks

Strided Convolutions

Strided convolution



*

$$\begin{array}{|c|c|c|} \hline 3 & 4 & 4 \\ \hline 1 & 0 & 2 \\ \hline -1 & 0 & 3 \\ \hline \end{array}$$

3x3

Stride = 2

=

$$\begin{array}{|c|c|c|} \hline 91 & 100 & 83 \\ \hline 69 & 91 & 127 \\ \hline 44 & 72 & 74 \\ \hline \end{array}$$

3x3

$\lfloor \frac{z}{s} \rfloor = \text{floor}(\frac{z}{s})$

$n \times n$ * $f \times f$
padding p stride s
 $s=2$

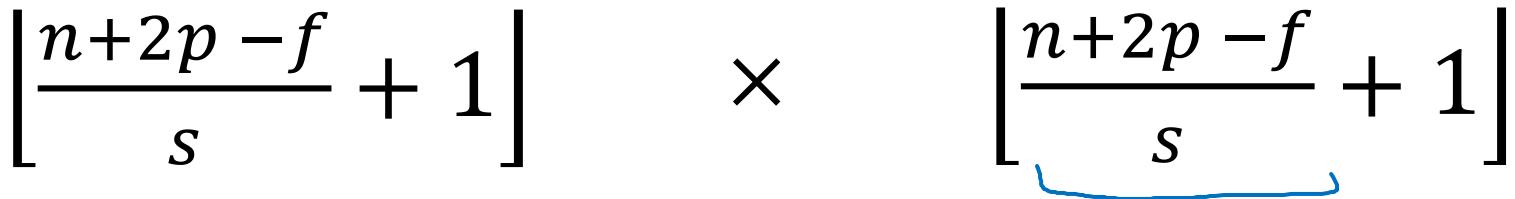
$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$
$$\frac{7+0-3}{2} + 1 = \frac{4}{2} + 1 = 3$$

Summary of convolutions

$n \times n$ image $f \times f$ filter

padding p stride s

Output size:

$$\left[\frac{n+2p-f}{s} + 1 \right] \quad \times \quad \left[\frac{n+2p-f}{s} + 1 \right]$$


Technical note on cross-correlation vs. convolution (Optional)

Convolution in math textbook:

2	3	7	4	6	2
6	6	9	8	7	4
3	4	8	3	8	9
7	8	3	6	6	3
4	2	1	8	3	4
3	2	4	1	9	8

$$\begin{matrix} & \begin{matrix} 3 & 4 & 5 \\ 1 & 0 & 2 \\ -1 & 9 & 7 \end{matrix} \end{matrix}$$



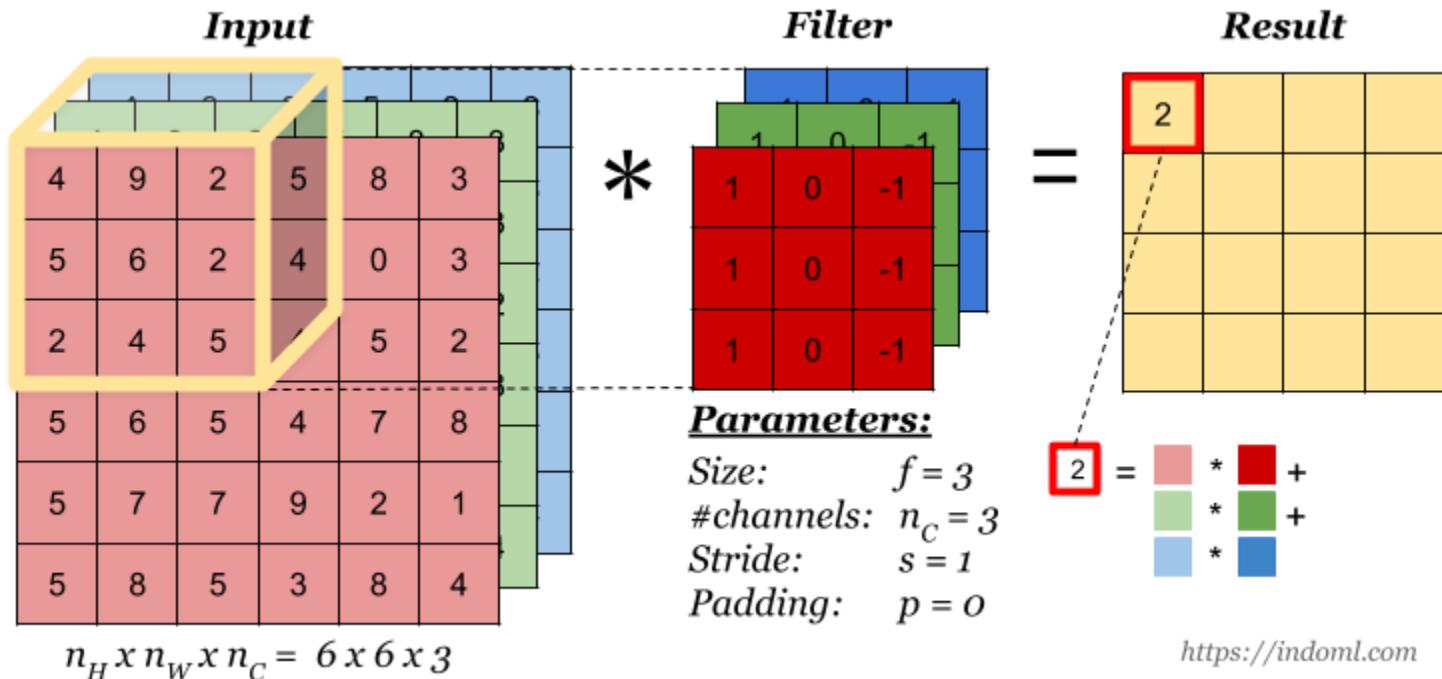
deeplearning.ai

Convolutional Neural Networks

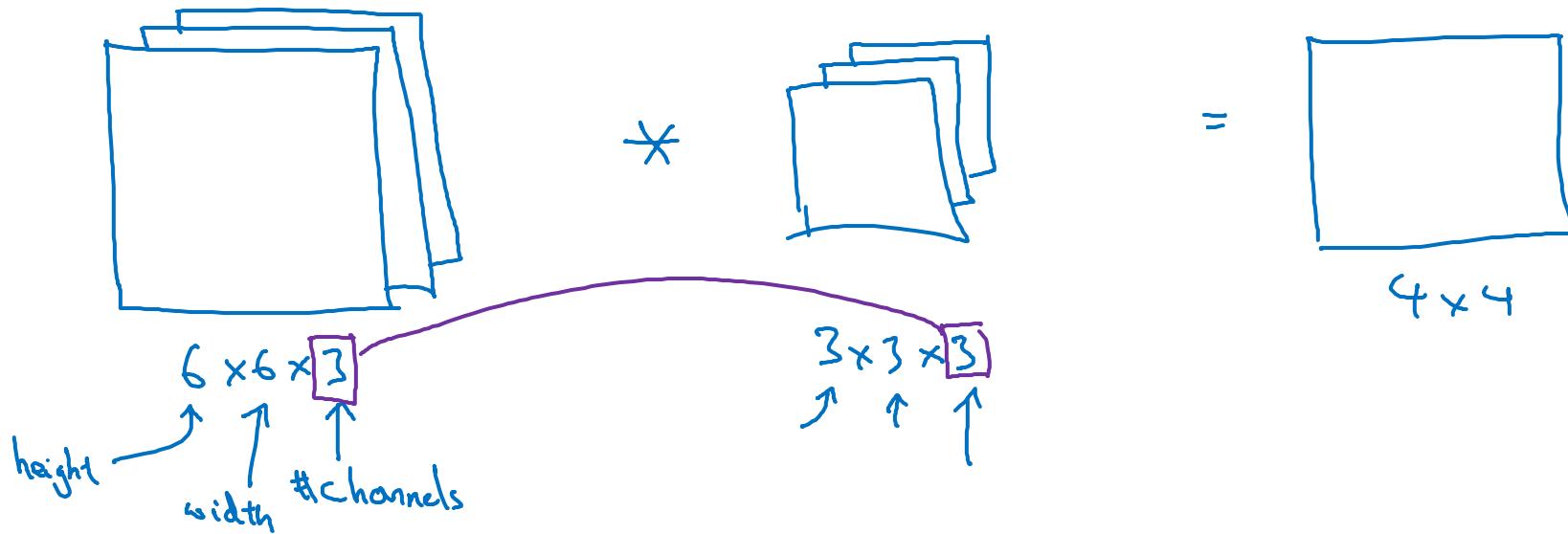
Convolutions over Volumes

Convolution Operation on Volume

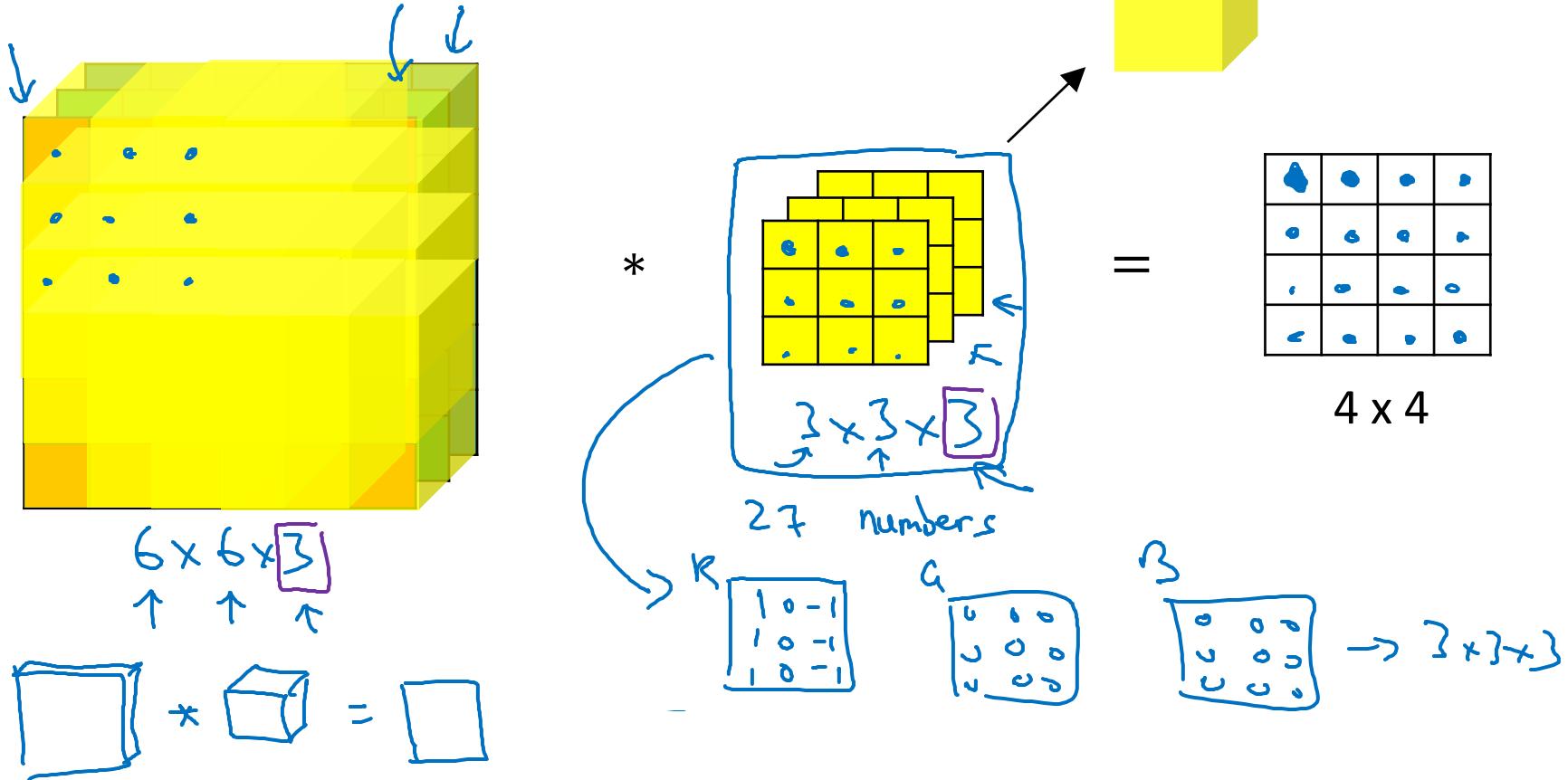
- When input has more than one channels (e.g. an RGB image), the filter should have matching number of channels.



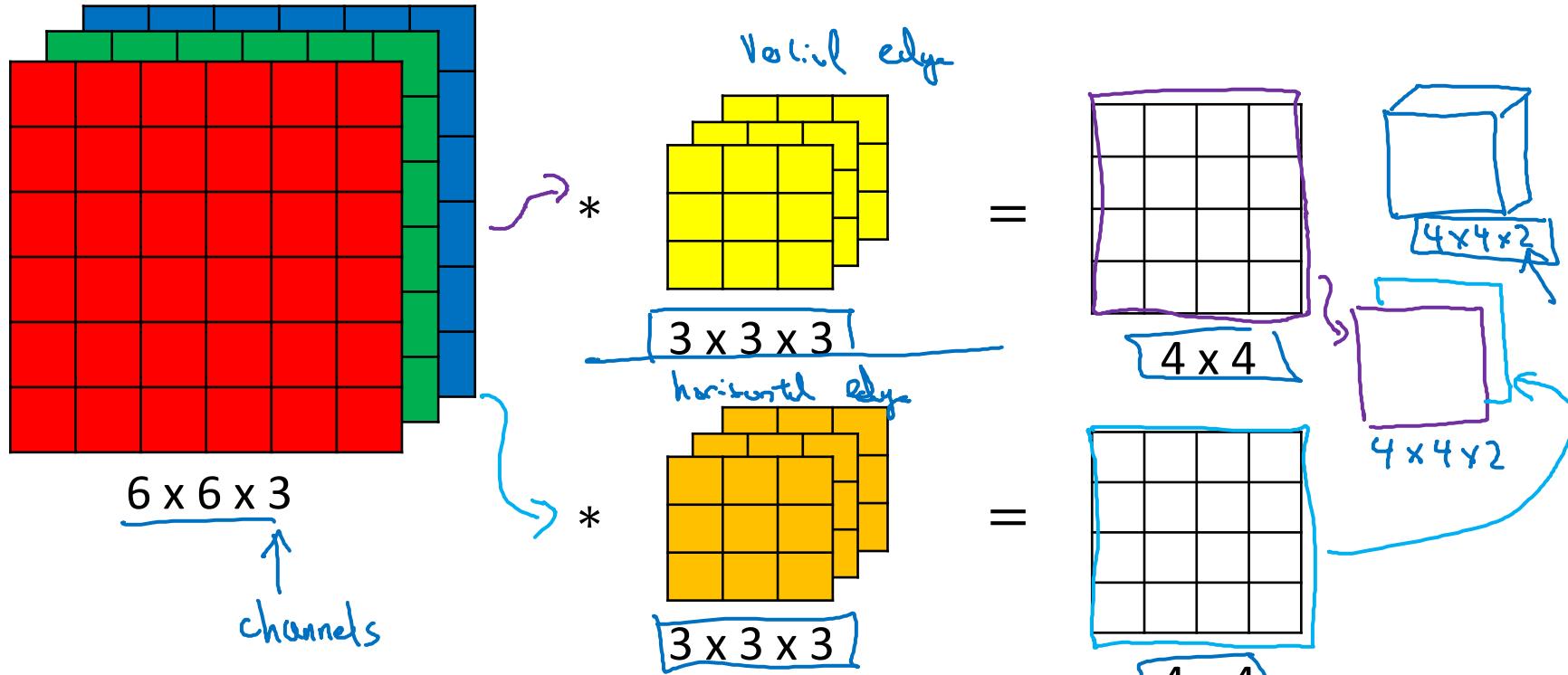
Convolutions on RGB images



Convolutions on RGB image

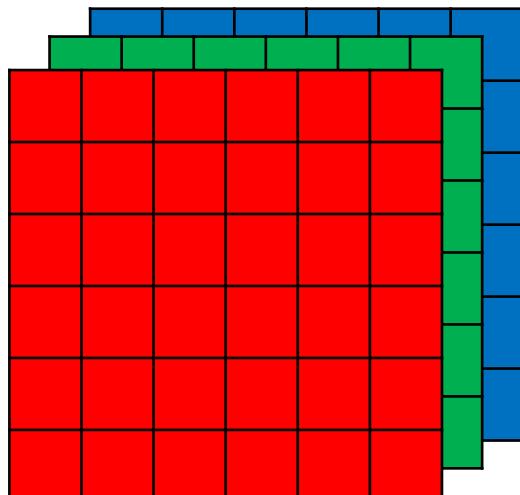


Multiple filters



Summary: $n \times n \times n_c$ $\neq f \times f \times n_c$ $\rightarrow \frac{n-f+1}{4} \times \frac{n-f+1}{4} \times \frac{n_c}{2} \# \text{filters}$

Example of a layer (with bias & activation function)

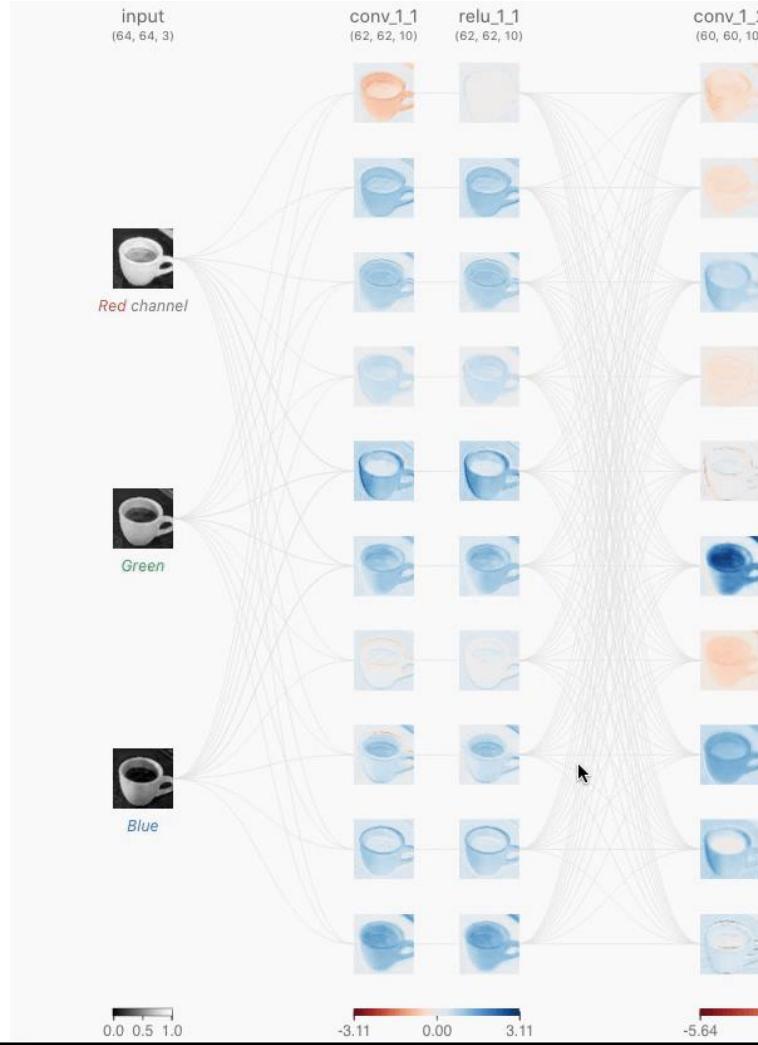


$6 \times 6 \times 3$

$$\begin{array}{c} * \\ \text{3} \times 3 \times 3 \end{array} = \begin{array}{c} \text{4} \times 4 \\ \text{4} \times 4 \end{array}$$

$$\begin{array}{c} * \\ \text{3} \times 3 \times 3 \end{array} = \begin{array}{c} \text{4} \times 4 \\ \text{4} \times 4 \end{array}$$

Example:





deeplearning.ai

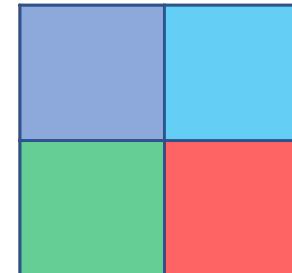
Convolutional Neural Networks

Pooling layers

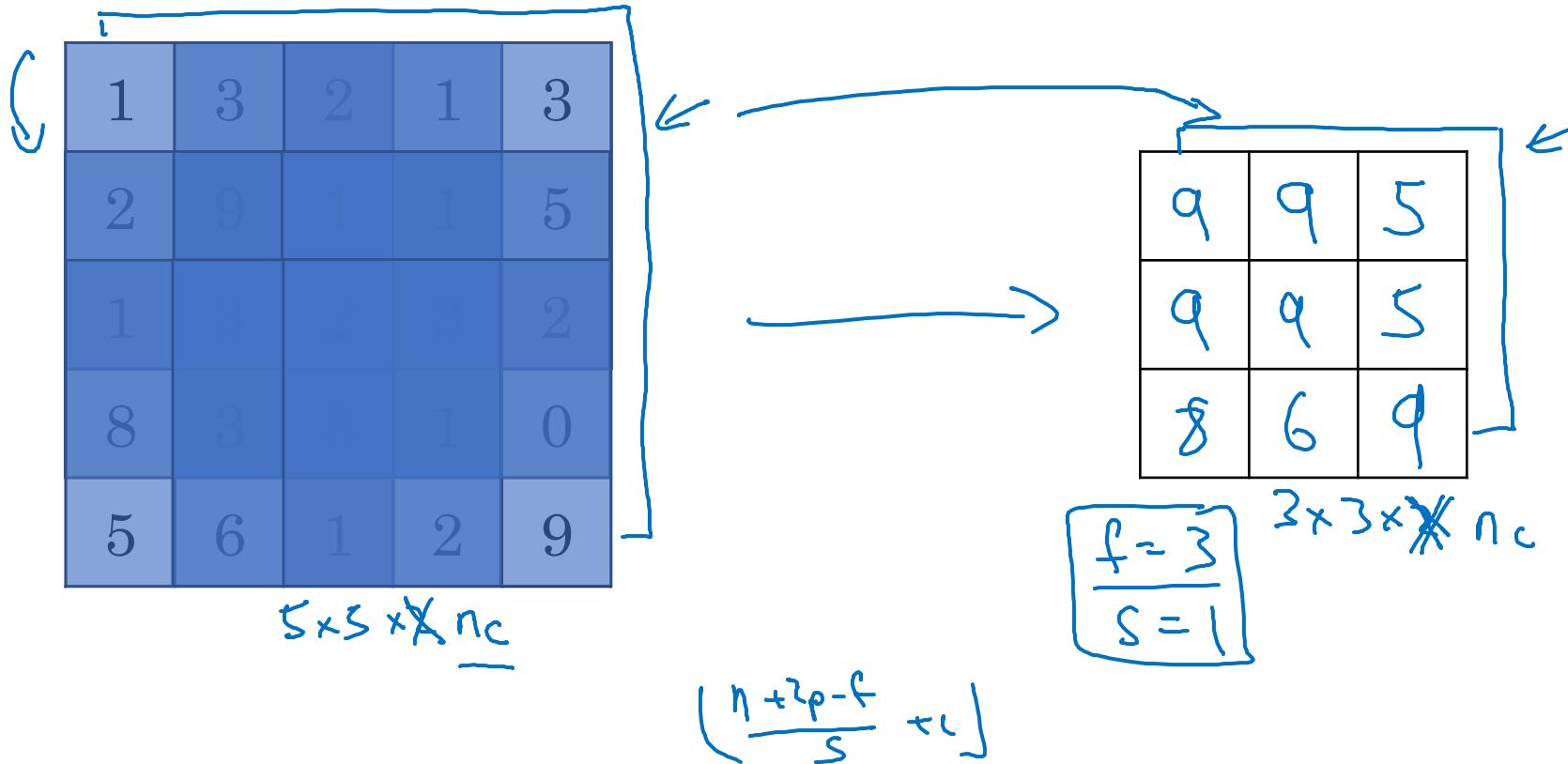
Pooling layer: Max pooling

Pool layer reduces the size of the inputs to speed up computation and make features more robust.

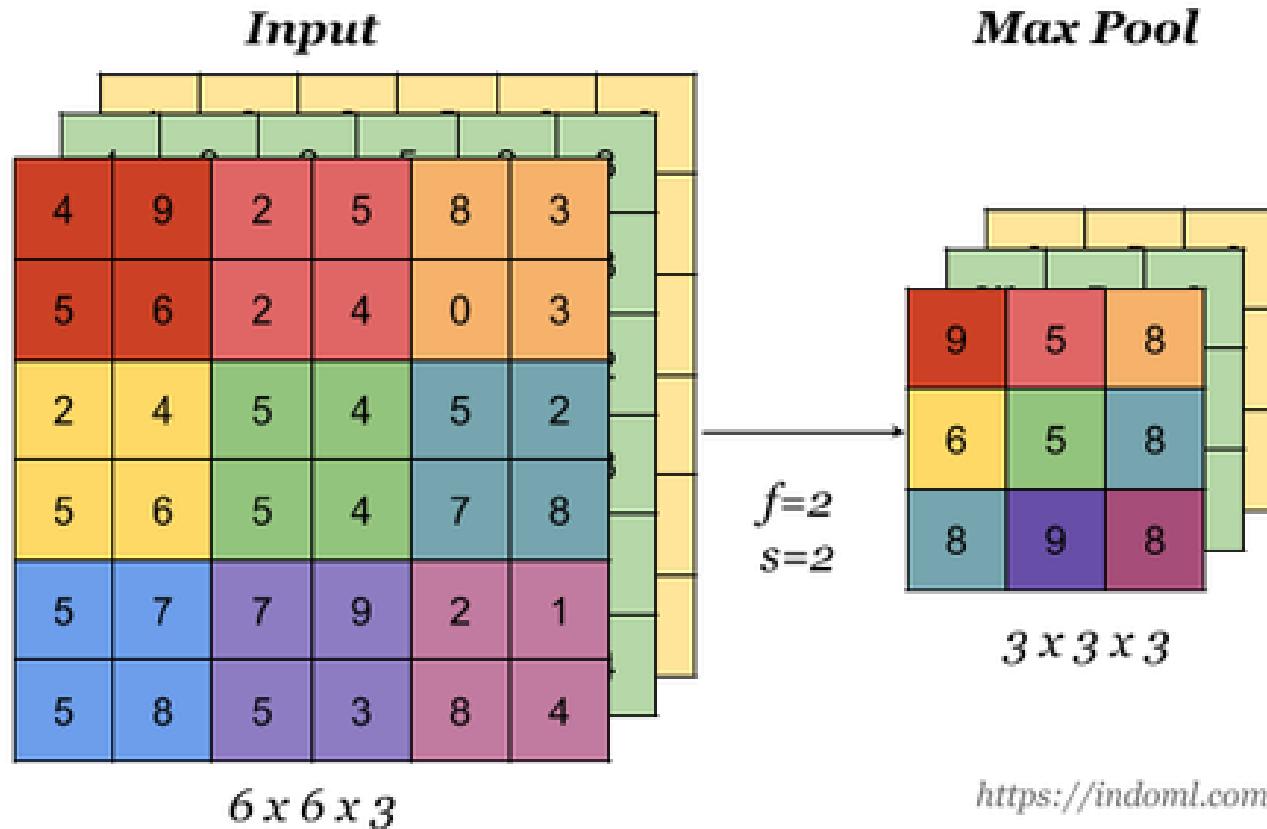
1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2



Pooling layer: Max pooling



Max Pooling with multiple Channels



Pooling layer: Average pooling

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2



3.75	1.25
4	2

$$f=2$$

$$s=2$$

$$\underline{7 \times 7 \times 1000} \rightarrow 1 \times 1 \times 1000$$

Summary of pooling

Hyperparameters:

f: filter size
s: stride

$$\begin{aligned} f &= 2, s = 2 \\ f &= 3, s = 2 \end{aligned}$$

type: Max or Average pooling

~~p: padding~~

No parameters to learn!

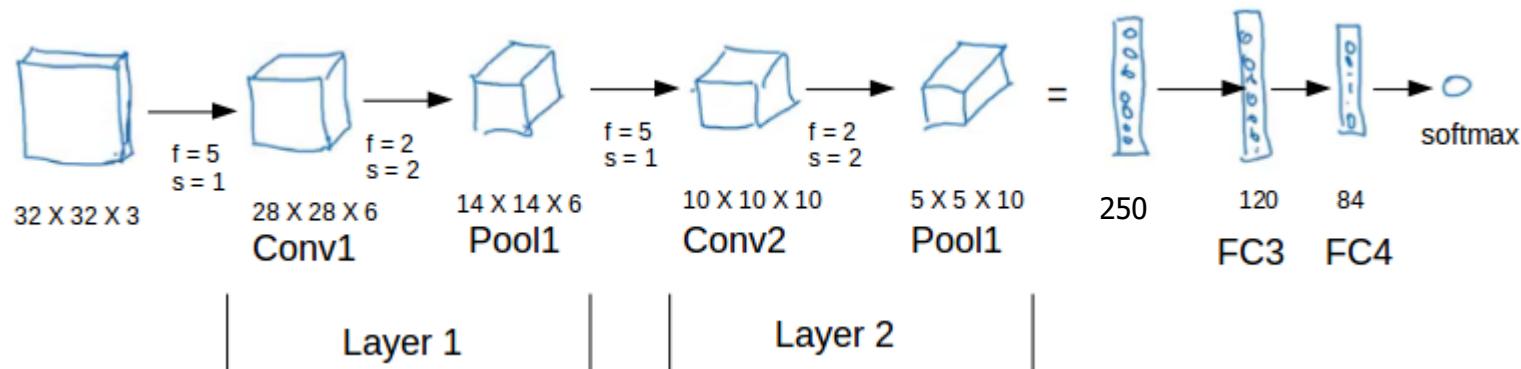
there's nothing for gradient descent to learn!

$$n_H \times n_w \times n_c$$

$$\left\lfloor \frac{n_H - f + p}{s} \right\rfloor \times \left\lfloor \frac{n_w - f}{s} + 1 \right\rfloor \times n_c$$

Simple CNN Example

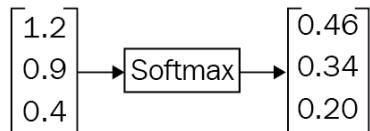
- In most Conv networks, as we propagate forward, the filter sizes get bigger and the outputs get smaller.
- Towards the end, for classification purposes, we unfold (flattening) all the features to use Fully Connected (FC) layers.
 - Fully connected layer involves weights, biases, and neurons. It connects neurons in one layer to neurons in another layer.
- Finally a Softmax Layer to classify the input into various categories.



Softmax

- The **softmax function**, also known as **softargmax** or **normalized exponential function**, is a generalization of the logistic function to multiple dimensions.
- Takes as input a vector \mathbf{z} of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.
- Prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval $(0,1)$, and the components will add up to 1.

$$e(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$



$$\mathbf{z} = (z_1, z_2, \dots, z_K) \in R^K$$

Softmax assumes that each example is a member of exactly one class. Some examples, however, can simultaneously be a member of multiple classes. For such examples:

- You may not use Softmax.
- You must rely on multiple logistic regressions.

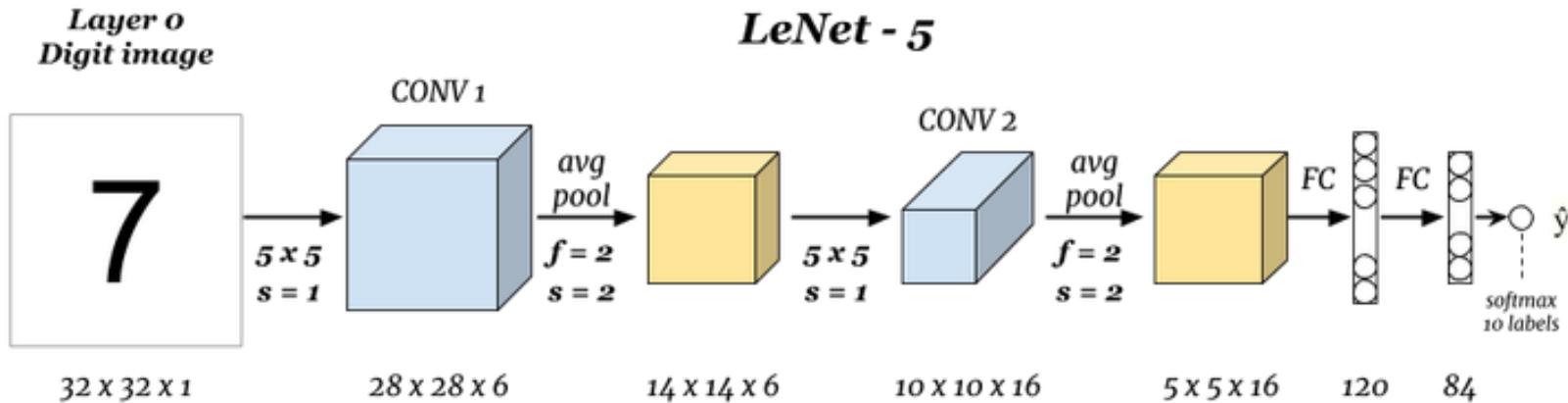


deeplearning.ai

Convolutional Neural Networks

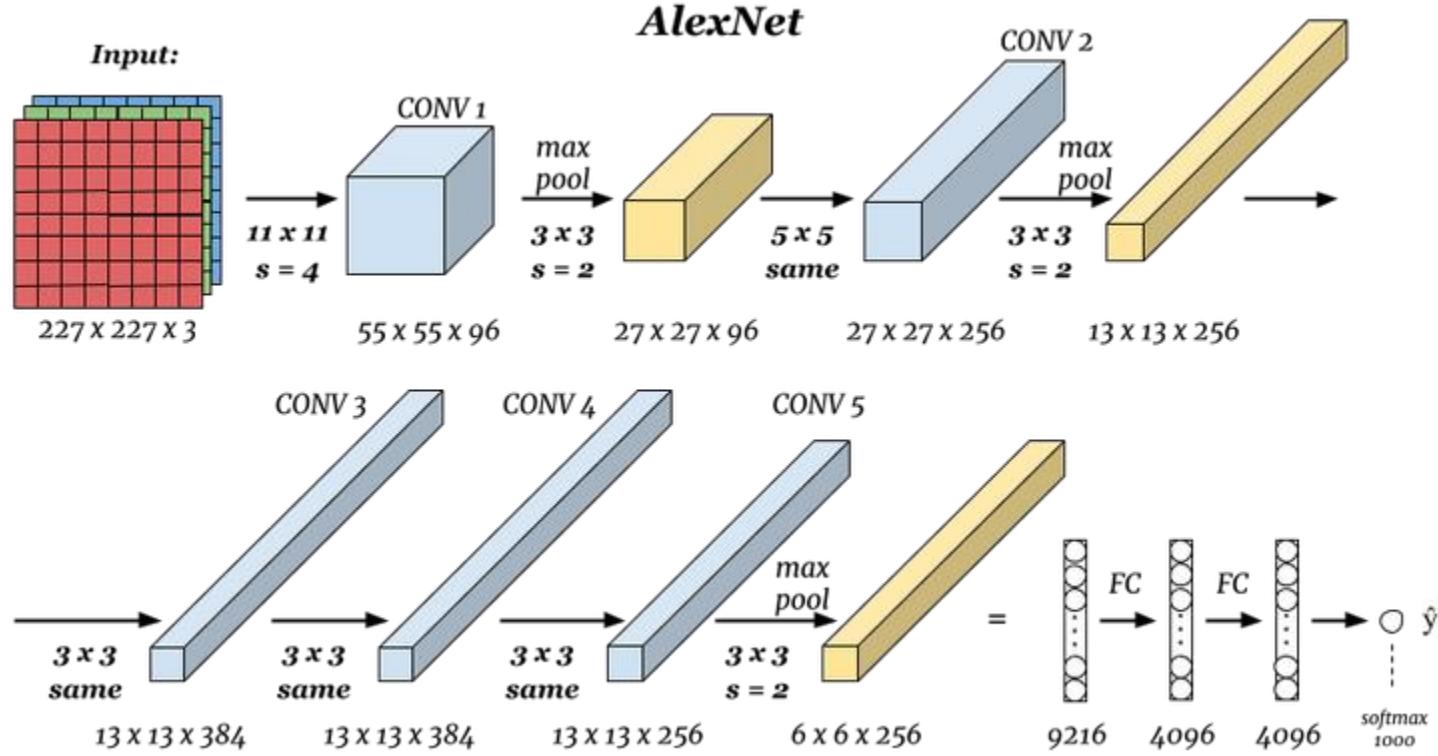
Well Known Architectures

LeNet - 5



- Number of parameters: ~ 60 thousands.

AlexNet

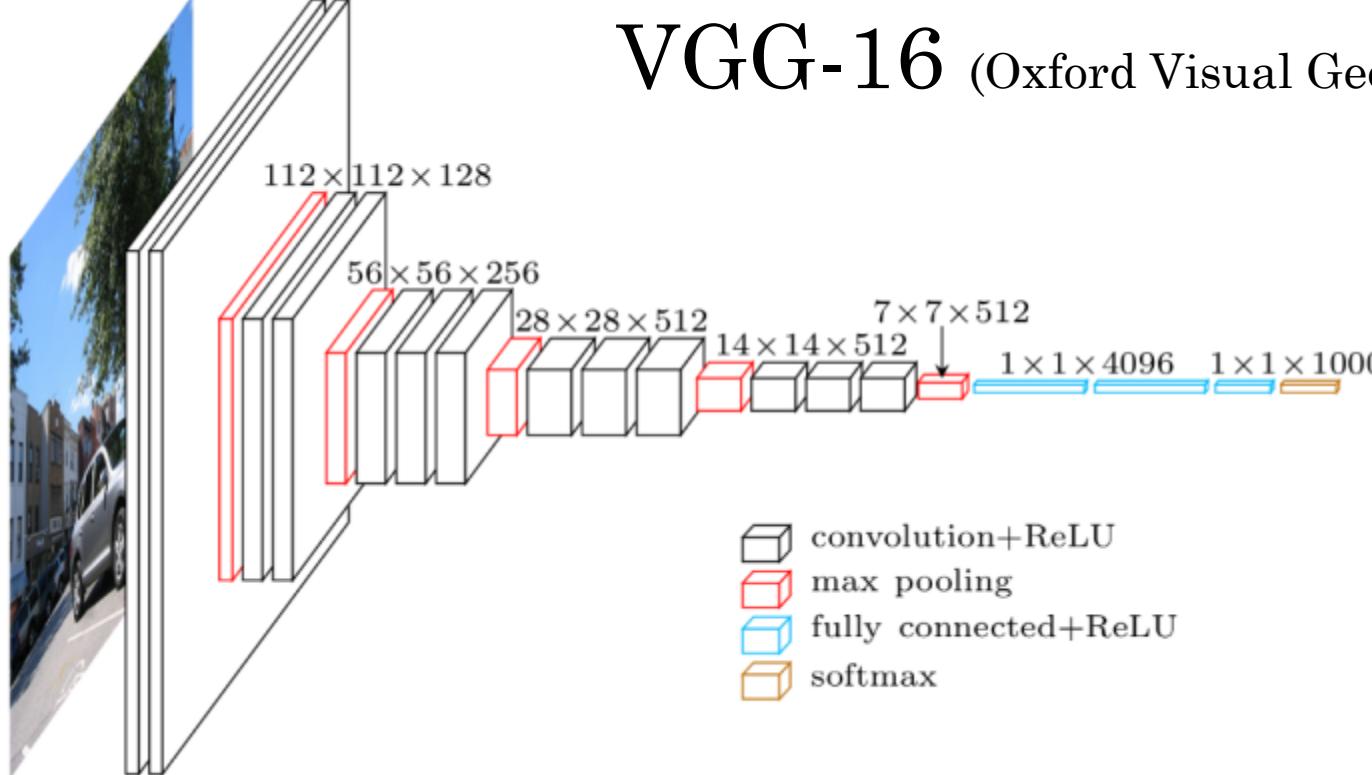


<https://indoml.com>

- Similar to LeNet-5 with just more convolution and pooling layers:
- Number of parameters: ~ 60 million.

[ImageNet Classification with Deep Convolutional Neural Networks](https://www.cs.toronto.edu/~kriz/lenet-5.pdf) paper by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever (2012).

$224 \times 224 \times 3$ $224 \times 224 \times 64$



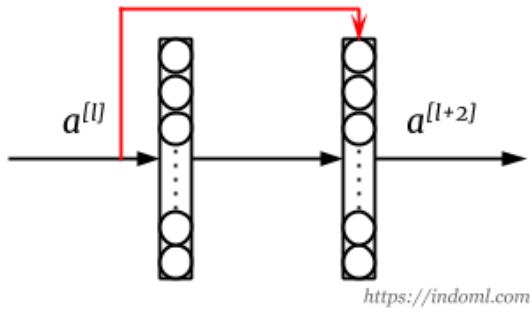
VGG-16 (Oxford Visual Geometry Group)

- Number of parameters: ~ 138 millions.
- The strength is in the simplicity: the dimension is halved and the depth is increased on every step (or stack of layers)

[Very Deep Convolutional Networks for Large-Scale Image Recognition](#) paper by Karen Simonyan and Andrew Zisserman (2014).

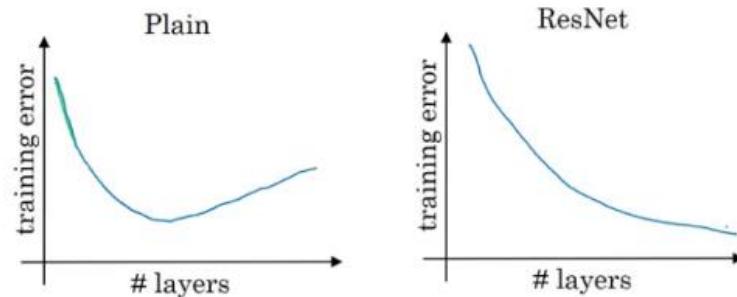
ResNet

- The problem with deeper neural networks are they are harder to train and once the number of layers reach certain number, the training error starts to raise again.
- Deep networks are also harder to train due to **exploding** and **vanishing** gradients problem.
- Residual Network solves these problems by implementing skip connection where output from one layer is fed to layer deeper in the network



$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

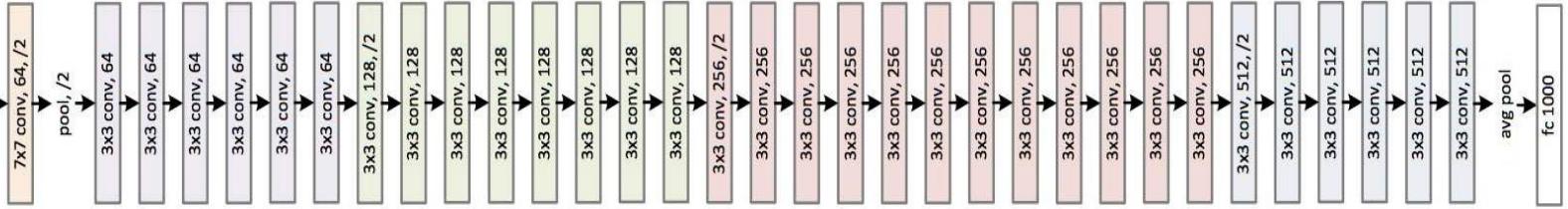
$$a^{[l+2]} = g^{[l+2]}(z^{[l+2]} + a^{[l]})$$



The benefit of training a residual network is that even if we train deeper networks, the training error does not increase.

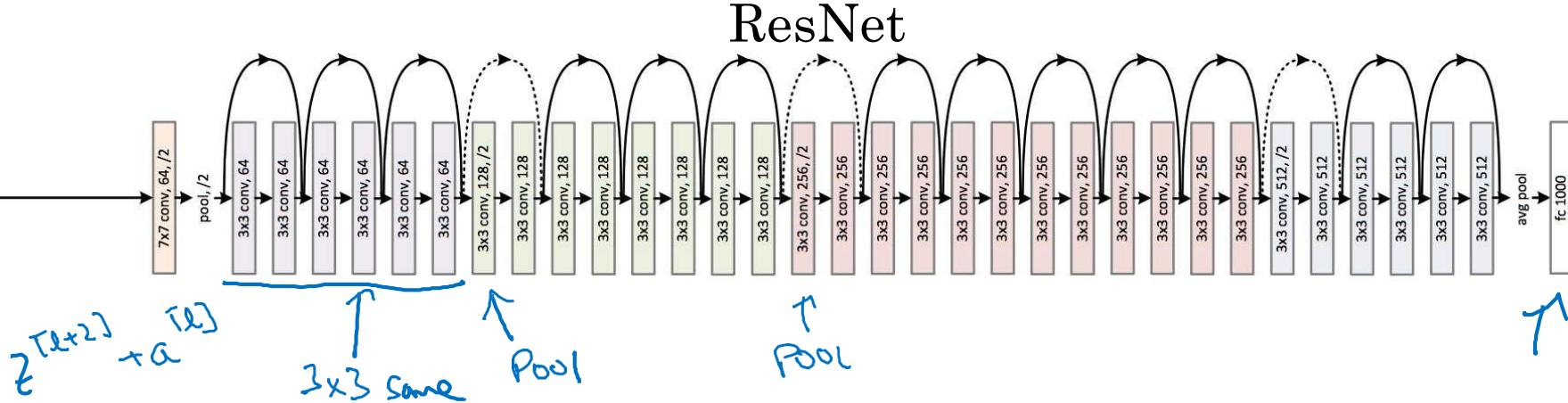
ResNet

34-layer plain



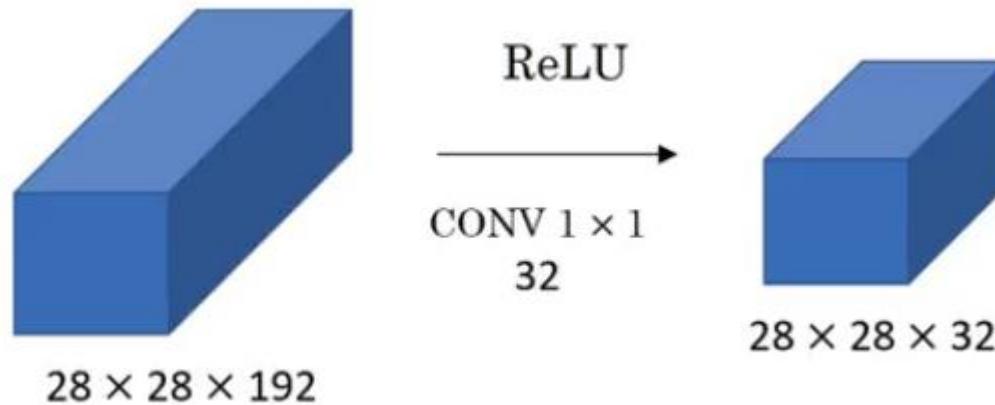
Plain

34-layer residual



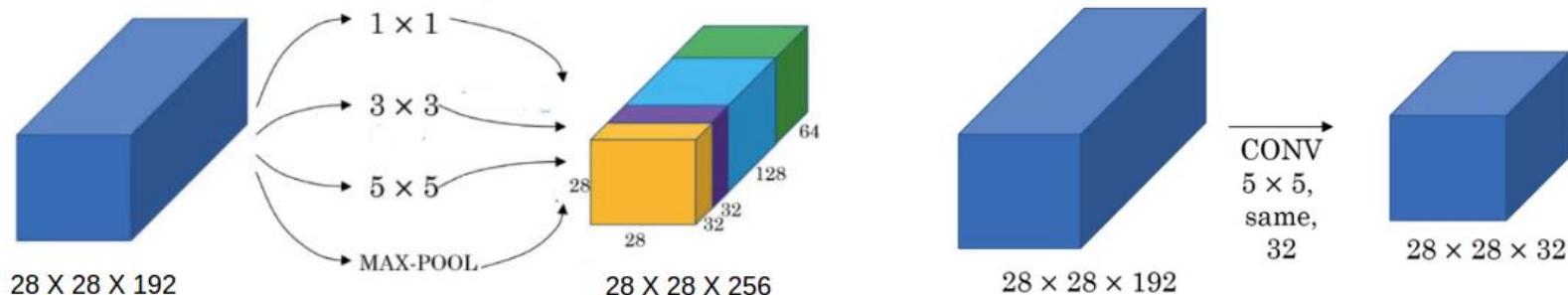
1×1 Convolutions

- The basic idea of using 1×1 convolution is to reduce the number of channels from the image.
 - We generally use a pooling layer to shrink the height and width of the image
 - To reduce the number of channels from an image, we convolve it using a 1×1 filter (hence reducing the computation cost as well)



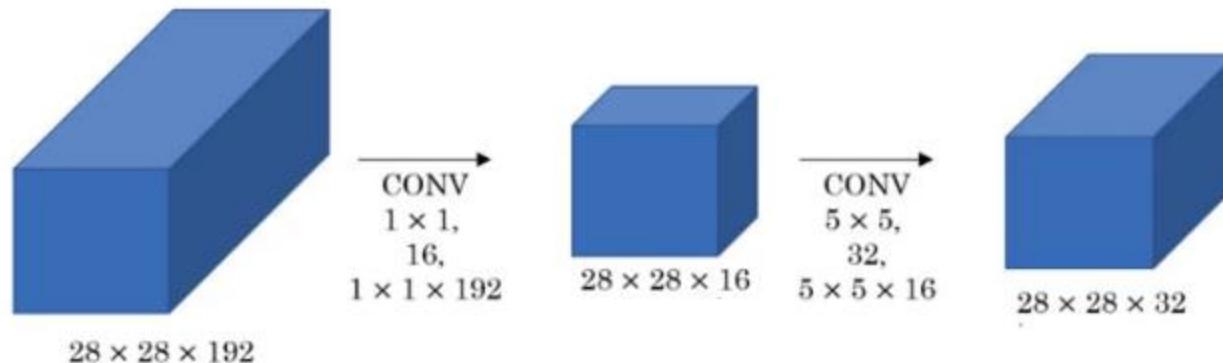
Inception Network - Motivation

- The motivation of the inception network is, rather than requiring us to pick the filter size manually, let the network decide what is best to put in a layer.
- We give it choices and hopefully it will pick up what is best to use in that layer:

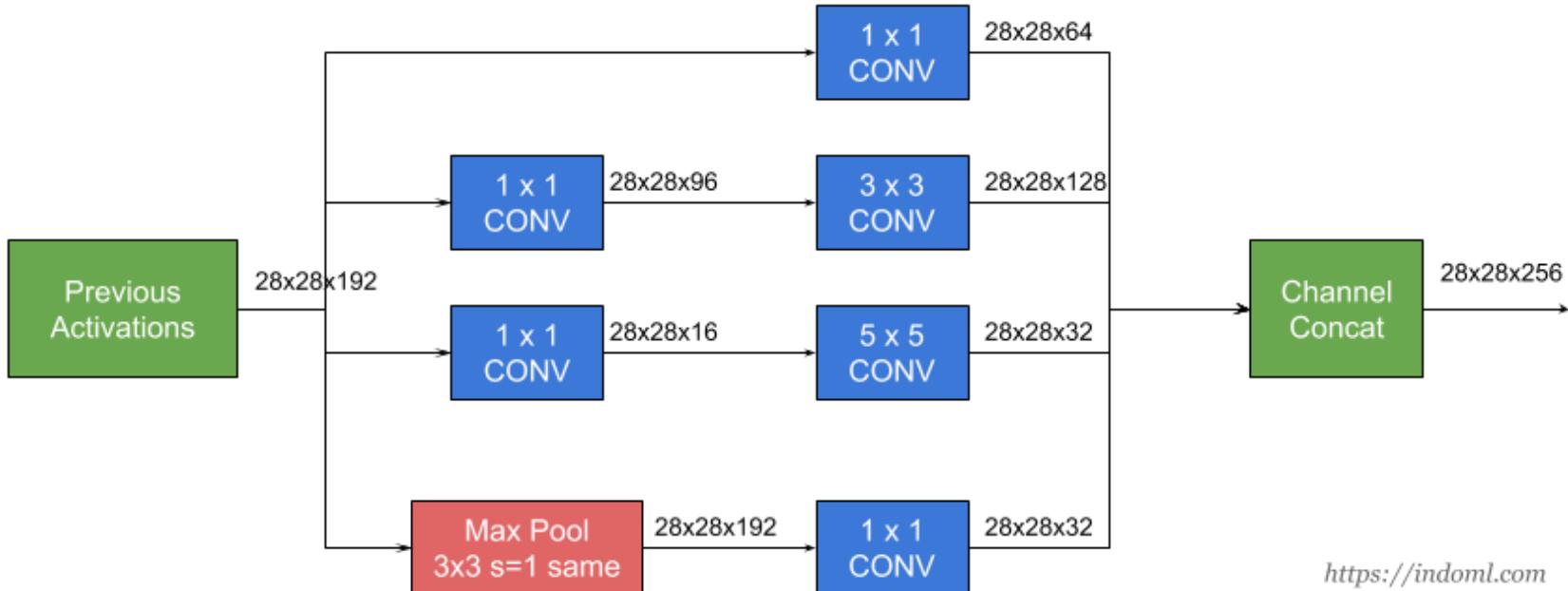


Inception Network - Motivation

- Let's look at the computations a 1×1 convolution and then a 5×5 convolution will give us:

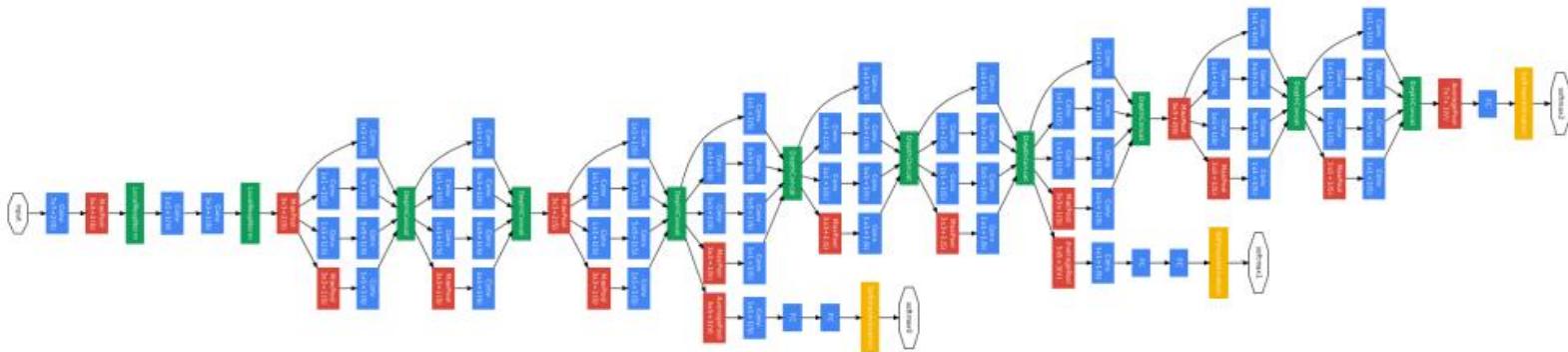


Inception Module



Inception Network (V1)

- Inception network called **GoogLeNet**, described in [Going Deeper with Convolutions paper](#) by Szegedy et al. (2014), (Winner ILSVRC 2014)
 - has 9 inception modules





CSE 4621

Machine Learning

Lecture 11

Md. Hasanul Kabir, PhD.
Professor, CSE Department
Islamic University of Technology (IUT)



Probabilistic Classification

- Probabilistic classification means that the model used for classification is a probabilistic model.
- Probabilistic model can give probability of an instance belonging to positive or negative class. Then it is up to us to decide whether the instance is positive or negative based on the probabilities given by the model.

Probabilistic Classification

Input: $S_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ training examples

$$y_i \in \{c_1, c_2, \dots, c_J\}$$

Goal: $h : X \rightarrow Y$

For each class c_j , estimate

$$P(y = c_j | \mathbf{x}, S_{\text{train}})$$

Assign to \mathbf{x} the class with the highest probability

$$\hat{y} = h(\mathbf{x}) = \arg \max_c P(y = c | \mathbf{x}, S_{\text{train}})$$

- Probabilistic classifier is a classifier that is able to predict, given an observation of an input, a probability distribution over a set of classes, rather than only outputting the most likely class that the observation should belong to.

Decision Theory

- Probability theory provides us with a consistent mathematical framework for quantifying and manipulating uncertainty.
- **Decision theory:** When combined with probability theory, that allows us to make optimal decisions in situations involving uncertainty.
- Suppose we have an input vector \mathbf{x} together with a corresponding vector \mathbf{t} of target variables, and our goal is to predict \mathbf{t} given a new value for \mathbf{x} .
 - For regression problems, \mathbf{t} will comprise continuous variables
 - For classification problems \mathbf{t} or C_k will represent class labels.
- The joint probability distribution $p(\mathbf{x}, \mathbf{t})$ or $p(\mathbf{x}, C_k)$, provides a complete summary of the uncertainty associated with these variables.
- Determination of $p(\mathbf{x}, \mathbf{t})$ from a set of training data is an example of inference
 - is typically a very difficult problem
- Finally, the decision step, it is the subject of decision theory to tell us how to make optimal decisions given the appropriate probabilities.

Example Scenario

- When we obtain the X-ray image \mathbf{x} for a new patient, our goal is to decide which of the two classes to assign to the image. We are interested in the probabilities of the two classes given the image, which are given by $p(C_k|\mathbf{x})$.
- Using Bayes' theorem, these probabilities can be expressed in the form

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})}.$$

- Interpret $p(C_k)$ as the prior probability for the class C_k , and $p(C_k|\mathbf{x})$ as the corresponding posterior probability. Thus $p(C_1)$ represents the probability that a person has cancer, before we take the X-ray measurement.
- If our aim is to minimize the chance of assigning \mathbf{x} to the wrong class, then intuitively we would choose the class having the higher posterior probability.

Approaches to Solving Decision Problems

- Two separate stages in Decision Theory,
 - the *inference stage* in which we use training data to learn a model for $p(C_k/x)$, and
 - the subsequent *decision* stage in which we use these posterior probabilities to make optimal class assignments.
 - Generative Model
 - Discriminative Model
- An alternative possibility would be to solve both problems together and simply learn a function that maps inputs x directly into decisions. Such a function is called a *discriminant function*.
 - Discriminant Function

(Compare between them)

Generative Model

- First solve the inference problem of determining the class-conditional densities $p(\mathbf{x}/C_k)$ for each class C_k individually. Also separately infer the prior class probabilities $p(C_k)$. Then use Bayes' theorem in the form

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})}.$$

- Equivalently, we can model the joint distribution $p(\mathbf{x}, C_k)$ directly and then normalize to obtain the posterior probabilities.
- Approaches that explicitly or implicitly model the distribution of inputs as well as outputs are known as *generative models*,
 - by sampling from them it is possible to generate synthetic data points in the input space.

Examples of Generative Models

- Gaussian mixture model (and other types of mixture model)
- Hidden Markov model
- Probabilistic context-free grammar
- Bayesian network (e.g. **Naive Bayes**, Autoregressive model)
- Averaged one-dependence estimators
- Latent Dirichlet allocation
- Boltzmann machine (e.g. Restricted Boltzmann machine, Deep belief network)
- Variational autoencoder
- Generative adversarial network
- Flow-based generative model
- Energy based model

Discriminative Model

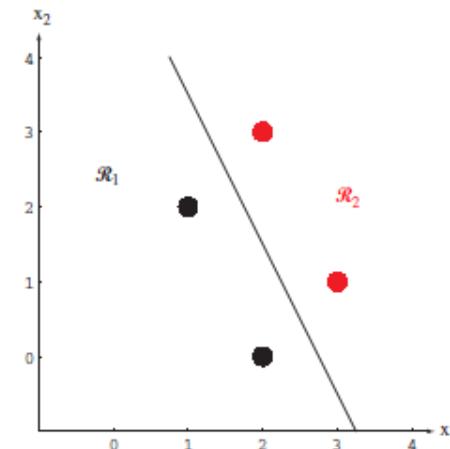
- First solve the inference problem of determining the posterior class probabilities $p(C_k/x)$, and then subsequently use decision theory to assign each new x to one of the classes.
- Approaches that model the posterior probabilities directly are called *discriminative models*.
- Examples:
 k -nearest neighbors algorithm, Logistic regression, Support Vector Machines, Decision Trees, Random Forest, Maximum-entropy Markov models, Conditional random fields, **Neural networks**

Discriminant Function

- Find a function $f(\mathbf{x})$, called a discriminant function, which maps each input \mathbf{x} directly onto a class label.
- For instance, in the case of two-class problems, $f(\cdot)$ might be binary valued and such that $f = 0$ represents class C_1 and $f = 1$ represents class C_2 . In this case, probabilities play no role.

The linear discriminant function $g(\mathbf{x})$ can be written as

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^d w_i x_i,$$



Graphical Model

- *Graphical models*, also called *Bayesian networks*, *belief networks*, or *probabilistic networks*, are composed of nodes and arcs between the nodes.
- Each node corresponds to a random variable, X , and has a value corresponding to the probability of the random variable, $P(X)$. If there is a directed arc from node X to node Y , this indicates that X has a *direct influence* on Y . This influence is specified by the conditional probability $P(Y|X)$. The network is a *directed acyclic graph* (DAG)
- The nodes and the arcs between the nodes define the *structure* of the network, and the conditional probabilities are the *parameters* given the structure.
- Graphical models are frequently used to visualize *generative models* for representing the process that we believe has created the data.

Example

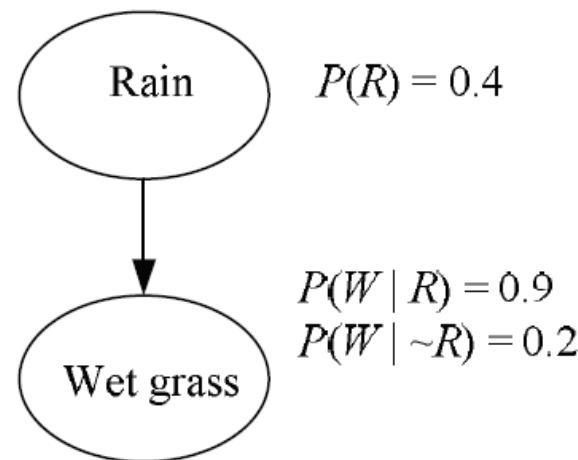
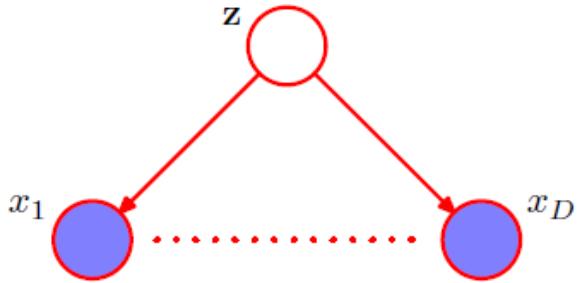


Figure 14.1 Bayesian network modeling that rain is the cause of wet grass.

Naïve Bayes Classifier

- **Naive Bayes classifier** is a probabilistic classifier based on applying Bayes' theorem with strong (naïve) independence assumptions between the features.
- In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
- Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to sometimes outperform even highly sophisticated classification methods.



A graphical representation of the ‘naive Bayes’ model for classification. Conditioned on the class label z , the components of the observed vector $\mathbf{x} = (x_1, \dots, x_D)^T$ are assumed to be independent.

Applications of Naïve Bayes Classifier

- **Disease Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- **Recommendation System:** Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not.

How it works

- Naïve Bayes is a conditional probability model: given a unknown sample to be classified, represented by a vector $x=(x_1, x_2, \dots, x_n)$ representing some n features (independent variables), it assigns to this posterior probabilities

$$P(C_K | x_1, x_2, \dots, x_n) = P(C_K | x)$$

for each of K possible outcomes or classes C_k .

- Bayes theorem provides a way of calculating posterior probability $P(C_k | x)$ from $P(C_k)$, $P(x)$ and $P(x | C_k)$

$$p(C_k | x) = \frac{p(x | C_k)p(C_k)}{p(x)}.$$

- In practice, there is interest only in the numerator of that fraction. The numerator is equivalent to the joint probability model

$$P(C_K, x_1, x_2, \dots, x_n)$$

How it works

$$p(C_k, x_1, \dots, x_n)$$

which can be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$$\begin{aligned} p(C_k, x_1, \dots, x_n) &= p(x_1, \dots, x_n, C_k) \\ &= p(x_1 | x_2, \dots, x_n, C_k) p(x_2, \dots, x_n, C_k) \\ &= p(x_1 | x_2, \dots, x_n, C_k) p(x_2 | x_3, \dots, x_n, C_k) p(x_3, \dots, x_n, C_k) \\ &= \dots \\ &= p(x_1 | x_2, \dots, x_n, C_k) p(x_2 | x_3, \dots, x_n, C_k) \cdots p(x_{n-1} | x_n, C_k) p(x_n | C_k) p(C_k) \end{aligned}$$

Now the "naïve" conditional independence assumptions come into play: assume that all features in \mathbf{x} are mutually independent, conditional on the category C . Under this assumption,

$$p(x_i | x_{i+1}, \dots, x_n, C_k) = p(x_i | C_k)$$

Thus, the joint model can be expressed as

$$\begin{aligned} p(C_k | x_1, \dots, x_n) &\propto p(C_k, x_1, \dots, x_n) \\ &\propto p(C_k) p(x_1 | C_k) p(x_2 | C_k) p(x_3 | C_k) \cdots \\ &\propto p(C_k) \prod_{i=1}^n p(x_i | C_k), \end{aligned}$$

where \propto denotes proportionality.

The naïve Bayes classifier combines this model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the maximum a posteriori or MAP decision rule.

$$\hat{y} = \arg \max_{k \in \{1, 2, \dots, K\}} p(C_k) \prod_{i=1}^n p(x_i | C_k)$$

Outlook

	Yes	No	P(yes)	P(no)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rainy	3	2	3/9	2/5
Total	9	5	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Example

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind

	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play	P(Yes)/P(No)
------	--------------

Yes	9	9/14
-----	---	------

No	5	5/14
----	---	------

Total	14	100%
-------	----	------

Example

- Today = (Outlook = **Sunny**, Temperature = **Hot**, Humidity = **Normal**, Wind = **False**)
- So, probability of playing golf (YES) is given by:
- probability to not play golf (NO) is given by:

Pros:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- *If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency".*
- *On the other side naive Bayes is also known as a bad estimator, so the probability outputs are not to be taken too seriously.*
- *Assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.*



CSE 4621

Machine Learning

Lecture 12

Md. Hasanul Kabir, PhD.

Professor, CSE Department

Islamic University of Technology (IUT)



Decision Tree

- A *decision tree* is a hierarchical data structure implementing the divide-and-conquer strategy.
 - It is an efficient nonparametric method,
 - can be used for both classification and regression.
- A decision tree is composed of internal decision nodes and terminal leaves.
- The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
- Advantage of the decision tree is interpretability. As the tree can be converted to a set of *IF-THEN rules*.
- Tree induction is the construction of the tree given a training sample. For a given training set, there exists many trees that code it with no error.

Decision Tree

- Given an input, at each node, a test is applied and one of the branches is taken depending on the outcome.
- Each *decision node* m implements a test function $f_m(\mathbf{x})$ with discrete outcomes labeling the branches.
- This process starts at the root and is repeated recursively until a *leaf node* is hit, at which point the value written in the leaf constitutes the output.

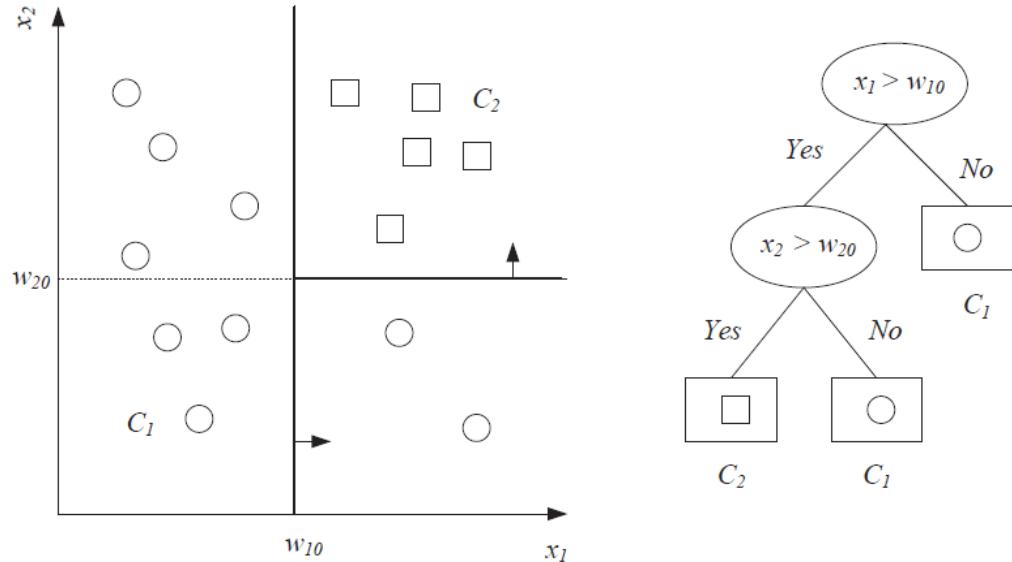


Figure 9.1 Example of a dataset and the corresponding decision tree. Oval nodes are the decision nodes and rectangles are leaf nodes. The univariate decision node splits along one axis, and successive splits are orthogonal to each other. After the first split, $\{\mathbf{x} | x_1 < w_{10}\}$ is pure and is not split further.

Test Function

- Each $f_m(\mathbf{x})$ defines a discriminant in the d -dimensional input space dividing it into smaller regions that are further subdivided down the tree.
- Different decision tree methods assume different models for $f_m(\cdot)$, and the model class defines the shape of the discriminant and the shape of regions.
- The boundaries of the regions are defined by the discriminants that are coded in the internal nodes.
- Hierarchical placement of decisions allows a fast localization of the region covering an input.
 - in best case, the correct region can be found in $\log_2 b$ decisions.

Univariate Tree

- In a *univariate tree*, in each internal node, the test uses only one of the input dimensions/attributes x_j .
 - is discrete, taking one of n possible values, the decision node checks the value of x_j and takes the corresponding branch (n -way split)
 - is numeric, input should be discretized.

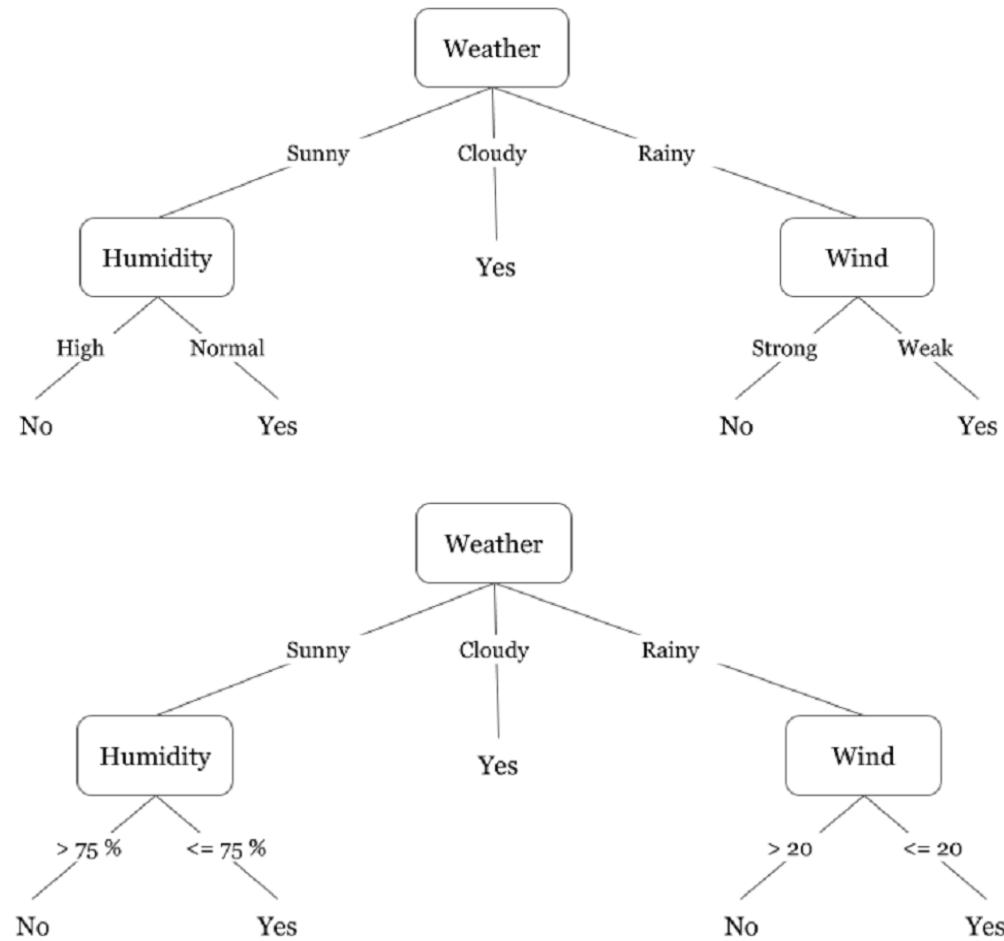
$$f_m(\mathbf{x}) : x_j > w_{m0}$$

where w_{m0} is a suitably chosen threshold value. The decision node divides the input space into two: $L_m = \{\mathbf{x} | x_j > w_{m0}\}$ and $R_m = \{\mathbf{x} | x_j \leq w_{m0}\}$; this is called a *binary split*. Successive decision nodes on a path

- Tree learning algorithms are greedy and, at each step, we look for the best split

Example

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No



Classification Tree

- A decision tree for classification, namely, a *classification tree*, the goodness of a split is quantified by an *impurity measure*.
 - a nonnegative function measuring the impurity of a split if it satisfies the following properties
 - $\phi(1/2, 1/2) \geq \phi(p, 1 - p)$, for any $p \in [0, 1]$.
 - $\phi(0, 1) = \phi(1, 0) = 0$.
 - $\phi(p, 1-p)$ is increasing in p on $[0, 1/2]$ and decreasing in p on $[1/2, 1]$.
- A split is pure if after the split, for all branches, all the instances choosing a branch belong to the same class.

Impurity Test: Entropy

- Entropy in information theory specifies the minimum number of bits needed to encode the class code of an instance.
- Our measure impurity is *entropy*

1. Entropy

$$\phi(p, 1 - p) = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

$$I_m = - \sum_{i=1}^K p_m^i \log_2 p_m^i$$

$$p_m^i = \frac{N_m^i}{N_m}$$

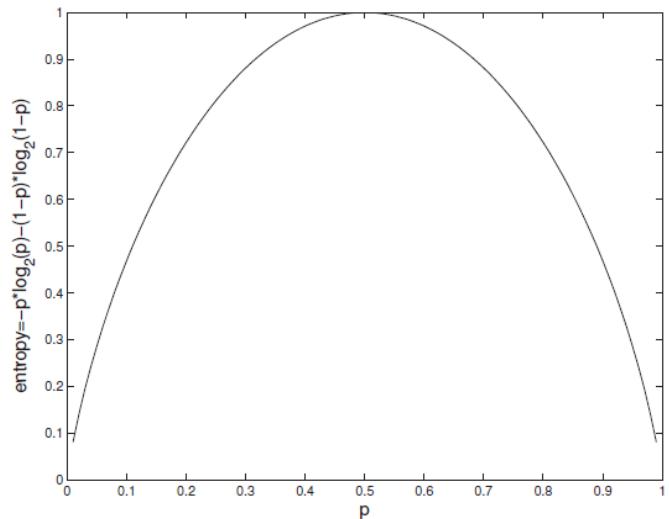


Figure 9.2 Entropy function for a two-class problem.
8

Impurity Test

2. *Gini index* (Breiman et al. 1984)

$$\phi(p, 1 - p) = 2p(1 - p)$$

3. Misclassification error

$$\phi(p, 1 - p) = 1 - \max(p, 1 - p)$$

Split and Sub-tree generation

- If node m is not pure, then the instances should be split to decrease impurity
 - there are multiple possible attributes on which we can split.
 - For a numeric attribute, multiple split positions are possible.
- Among all, take the split that minimizes impurity after the split because we want to generate the smallest tree.
 - this is locally optimal, and we have no guarantee of finding the smallest decision tree.
 - Tree size is measured as the number of nodes in the tree and the complexity of the decision nodes.
 - Finding the smallest tree is NP-complete.

Split and Sub-tree generation

Then given that at node m , the test returns outcome j , the estimate for the probability of class C_i is

$$\hat{P}(C_i | \mathbf{x}, m, j) \equiv p_{mj}^i = \frac{N_{mj}^i}{N_{mj}}$$

and the total impurity after the split is given as

$$I'_m = - \sum_{j=1}^n \frac{N_{mj}}{N_m} \sum_{i=1}^K p_{mj}^i \log_2 p_{mj}^i$$

- In the case of a numeric attribute, here are $N_m - 1$ possible w_{m0} between N_m data points
 - Do not need to test for all. the best split is always between adjacent points belonging to different classes.

Split and Sub-tree generation

- For all attributes, discrete and numeric, and for a numeric attribute for all split positions, we calculate the impurity and choose the one that has the minimum entropy.
- Choose the split that causes the largest decrease in impurity, which is the difference between the impurity of data reaching node m and the total entropy of data reaching its branches after the split.
- Then tree construction continues recursively and in parallel for all the branches that are not pure, until all are pure.
- Algorithms:
 - CART
 - ID3
 - C4.5

Important Factors

- One problem is that such splitting favors attributes with many values
- Nodes with many branches are complex and go against our idea of splitting class discriminants into simple decisions.
- Methods may be proposed to penalize such attributes and to balance the impurity drop and the branching factor.
- Second when there is noise, growing the tree until it is purest, we may grow a very large tree and it overfits.
- To alleviate such overfitting, tree construction ends when nodes become pure enough, namely, a subset of data is not split further if $|l| < \theta$,
 - *do not require that p_{mj}^i be exactly 0 or 1 but close enough.*
- *Generally advised that in a leaf, one stores the posterior probabilities of classes.*

Classification Tree

GenerateTree(X)

If $\text{NodeEntropy}(X) < \theta_I$ /* equation 9.3 */
 Create leaf labelled by majority class in X
 Return
 $i \leftarrow \text{SplitAttribute}(X)$
 For each branch of x_i
 Find X_i falling in branch
 GenerateTree(X_i)

SplitAttribute(X)
 $\text{MinEnt} \leftarrow \text{MAX}$
 For all attributes $i = 1, \dots, d$
 If x_i is discrete with n values
 Split X into X_1, \dots, X_n by x_i
 $e \leftarrow \text{SplitEntropy}(X_1, \dots, X_n)$ /* equation 9.8 */
 If $e < \text{MinEnt}$ $\text{MinEnt} \leftarrow e$; $\text{bestf} \leftarrow i$
 Else /* x_i is numeric */
 For all possible splits
 Split X into X_1, X_2 on x_i
 $e \leftarrow \text{SplitEntropy}(X_1, X_2)$
 If $e < \text{MinEnt}$ $\text{MinEnt} \leftarrow e$; $\text{bestf} \leftarrow i$
 Return bestf

Figure 9.3 Classification tree construction.

Step-by-Step Example

- Sample # 14

Play Golf(14)	
Yes	No
9	5

Fig. Frequency Table

$$\begin{aligned}E(\text{PlayGolf}) &= E(5,9) \\&= -\left(\frac{9}{14} \log_2 \frac{9}{14}\right) - \left(\frac{5}{14} \log_2 \frac{5}{14}\right) \\&= -(0.357 \log_2 0.357) - (0.643 \log_2 0.643) \\&= 0.94\end{aligned}$$

Entropy at Root Node

Attributes				Classes
Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Step-by-Step: Calculate Entropy for Attributes After Split

We need to calculate the entropy after each of the split.

- $E(\text{PlayGolf}, \text{Outlook})$

$$E(\text{PlayGolf}, \text{Outlook}) = P(\text{Sunny})E(\text{Sunny}) + P(\text{Overcast})E(\text{Overcast}) + P(\text{Rainy})E(\text{Rainy})$$

- $E(\text{PlayGolf}, \text{Temperature})$
- $E(\text{PlayGolf}, \text{Humidity})$
- $E(\text{PlayGolf}, \text{Windy})$

$E(PlayGolf, Outlook)$

		PlayGolf(14)		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5

Fig. Frequency Table

$$E(Sunny) = E(3,2)$$

$$= -\left(\frac{3}{5} \log_2 \frac{3}{5}\right) - \left(\frac{2}{5} \log_2 \frac{2}{5}\right)$$

$$= -(0.60 \log_2 0.60) - (0.40 \log_2 0.40)$$

$$= -(0.60 * 0.737) - (0.40 * 0.529)$$

$$= \mathbf{0.971}$$

$$E(Overcast) = E(4,0)$$

$$= -\left(\frac{4}{4} \log_2 \frac{4}{4}\right) - \left(\frac{0}{4} \log_2 \frac{0}{4}\right)$$

$$= -(0) - (0)$$

$$= \mathbf{0}$$

$$E(Rainy) = E(2,3)$$

$$= -\left(\frac{2}{5} \log_2 \frac{2}{5}\right) - \left(\frac{3}{5} \log_2 \frac{3}{5}\right)$$

$$= -(0.40 \log_2 0.40) - (0.6 \log_2 0.60)$$

$$= \mathbf{0.971}$$

$$E(PlayGolf, Outlook) = \frac{5}{14} E(3,2) + \frac{4}{14} E(4,0) + \frac{5}{14} E(2,3)$$

$$= \frac{5}{14} 0.971 + \frac{4}{14} 0.0 + \frac{5}{14} 0.971$$

$$= 0.357 * 0.971 + 0.0 + 0.357 * 0.971$$

$$= \mathbf{0.693}$$

$E(PlayGolf, Temperature)$

$$E(\text{PlayGolf, Temperature}) = \frac{4}{14} * E(\text{Hot}) + \frac{4}{14} * E(\text{Cold}) + \frac{6}{14} * E(\text{Mild})$$

		PlayGolf(14)		
		Yes	No	
Temperature	Hot	2	2	4
	Cold	3	1	4
	Mild	4	2	6

Fig. Frequency Table

$$E(\text{PlayGolf, Temperature}) = \frac{4}{14} * E(2, 2) + \frac{4}{14} * E(3, 1) + \frac{6}{14} * E(4, 2)$$

$$E(\text{PlayGolf, Temperature}) = \frac{4}{14} * -(2/4 \log 2/4) - (2/4 \log 2/4)$$

$$+ \frac{4}{14} * -(3/4 \log 3/4) - (1/4 \log 1/4)$$

$$+ \frac{6}{14} * -(4/6 \log 4/6) - (2/6 \log 2/6)$$

$$E(\text{PlayGolf, Temperature}) = \frac{5}{14} * 1.0$$

$$+ \frac{4}{14} * 1.811$$

$$+ \frac{5}{14} * 0.918$$

$$= \mathbf{0.911}$$

$E(PlayGolf, \text{Humidity})$

$$E(\text{PlayGolf, Humidity}) = 7/14 * E(\text{High}) + 7/14 * E(\text{Normal})$$

		PlayGolf(14)		
		Yes	No	
Humidity	High	3	4	7
	Normal	6	1	7

Fig. Frequency Table

$$\begin{aligned}E(\text{PlayGolf, Humidity}) &= 7/14 * -(3/7 \log 3/7) - (4/7 \log 4/7) \\&\quad + 7/14 * -(6/7 \log 6/7) - (1/7 \log 1/7)\end{aligned}$$

$$\begin{aligned}E(\text{PlayGolf, Humidity}) &= 7/14 * 0.985 \\&\quad + 7/14 * 0.592 \\&= \mathbf{0.788}\end{aligned}$$

$E(PlayGolf, Windy)$

		PlayGolf(14)		
		Yes	No	
Windy	TRUE	3	3	6
	FALSE	6	2	8

$$E(\text{PlayGolf, Windy}) = 6/14 * E(\text{True}) + 8/14 * E(\text{False})$$

Fig. Frequency Table

$$E(\text{PlayGolf, Windy}) = 6/14 * E(3, 3) + 8/14 * E(6, 2)$$

$$\begin{aligned}E(\text{PlayGolf, Windy}) &= 6/14 * -(3/6 \log 3/6) - (3/6 \log 3/6) \\&\quad + 8/14 * -(6/8 \log 6/8) - (2/8 \log 2/8)\end{aligned}$$

$$\begin{aligned}E(\text{PlayGolf, Windy}) &= 6/14 * 1.0 \\&\quad + 8/14 * 0.811 \\&= \mathbf{0.892}\end{aligned}$$

Calculating Information Gain for Each Split

- $E(\text{PlayGolf}, \text{Outlook}) = 0.693$
- $E(\text{PlayGolf}, \text{Temperature}) = 0.911$
- $E(\text{PlayGolf}, \text{Humidity}) = 0.788$
- $E(\text{PlayGolf}, \text{Windy}) = 0.892$
- $Gain(S, T) = Entropy(S) - Entropy(S, T)$
- $Gain(\text{PlayGolf}, \text{Outlook}) = Entropy(\text{PlayGolf}) - Entropy(\text{PlayGolf}, \text{Outlook}) = 0.94 - 0.693 = 0.247$
- $Gain(\text{PlayGolf}, \text{Temperature}) = Entropy(\text{PlayGolf}) - Entropy(\text{PlayGolf}, \text{Temperature}) = 0.94 - 0.911 = 0.029$
- $Gain(\text{PlayGolf}, \text{Humidity}) = Entropy(\text{PlayGolf}) - Entropy(\text{PlayGolf}, \text{Humidity}) = 0.94 - 0.788 = 0.152$
- $Gain(\text{PlayGolf}, \text{Windy}) = Entropy(\text{PlayGolf}) - Entropy(\text{PlayGolf}, \text{Windy}) = 0.94 - 0.892 = 0.048$

Perform the First Split & Repeat

Outlook	Temperature	Humidity	Windy	Play Golf
Sunny	Mild	Normal	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Sunny	Mild	High	TRUE	No

Overcast	Hot	High	FALSE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Overcast	Cool	Normal	TRUE	Yes

Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes



Regression Tree

- Constructed in almost the same manner as a classification tree
- Impurity measure is replaced by a measure appropriate for regression.
 - In regression, the goodness of a split is measured by the mean square error from the estimated value.

$$E_m = \frac{1}{N_m} \sum_t (r^t - g_m)^2 b_m(\mathbf{x}^t) \quad b_m(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{X}_m: \mathbf{x} \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases}$$

- Let us say g_m is the estimated value in node m .
- Use the mean (median if there is too much noise) of the required outputs
- If at a node, the error is acceptable, that is, $E_m < \theta_r$, then a leaf node is created and it stores the g_m value.

$$g_m = \frac{\sum_t b_m(\mathbf{x}^t) r^t}{\sum_t b_m(\mathbf{x}^t)}$$

Split and Sub-tree generation

- If the error is not acceptable, data reaching node m is split further such that the sum of the errors in the branches is minimum.
 - we look for the attribute (and split threshold for a numeric attribute) that minimizes the error.

Let us define \mathcal{X}_{mj} as the subset of \mathcal{X}_m taking branch j : $\cup_{j=1}^n \mathcal{X}_{mj} = \mathcal{X}_m$. We define

$$b_{mj}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{X}_{mj}: \mathbf{x} \text{ reaches node } m \text{ and takes branch } j \\ 0 & \text{otherwise} \end{cases}$$

g_{mj} is the estimated value in branch j of node m .

$$g_{mj} = \frac{\sum_t b_{mj}(\mathbf{x}^t) r^t}{\sum_t b_{mj}(\mathbf{x}^t)}$$

and the error after the split is

$$E'_m = \frac{1}{N_m} \sum_j \sum_t (r^t - g_{mj})^2 b_{mj}(\mathbf{x}^t)$$

Impact of Error Threshold

- Acceptable error threshold θ_r is the complexity parameter
 - when it is small, we generate large trees and risk overfitting;
 - when it is large, we underfit and smooth too much

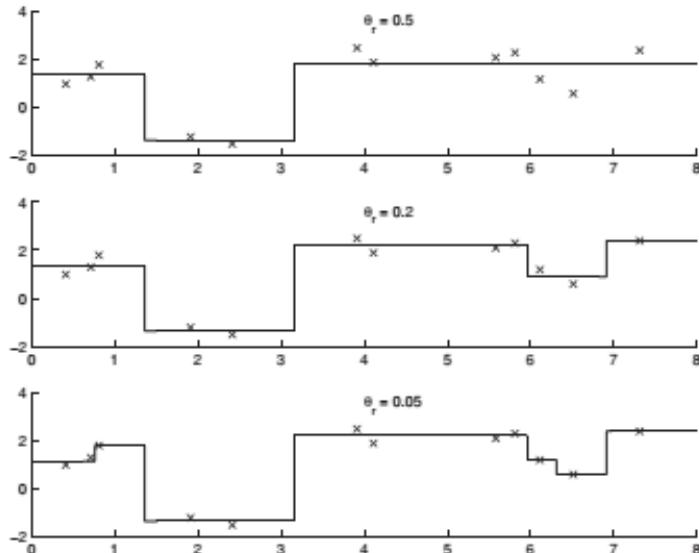
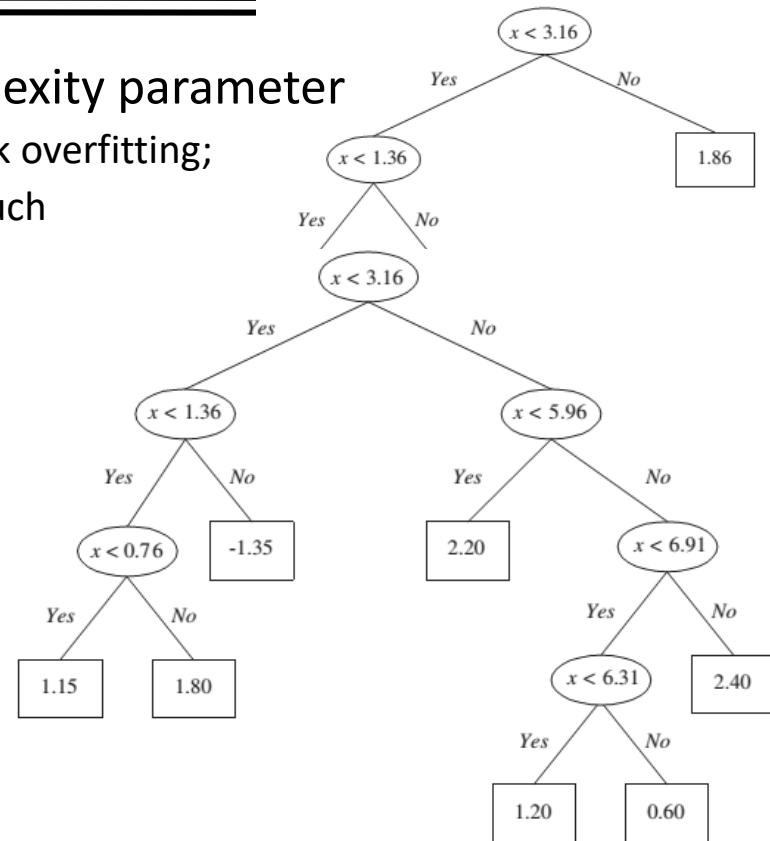


Figure 9.4 Regression tree smooths for various values of θ_r . The corresponding trees are given in figure 9.5.





CSE 4621

Machine Learning

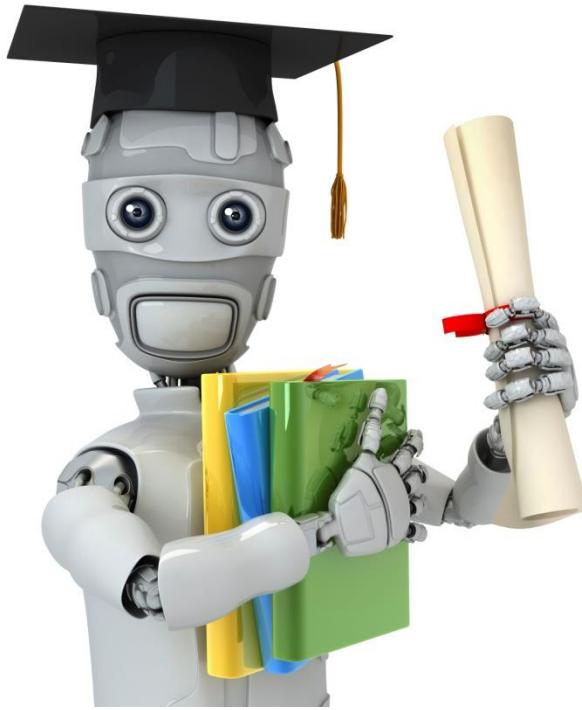
Lecture 13

Md. Hasanul Kabir, PhD.

Professor, CSE Department

Islamic University of Technology (IUT)



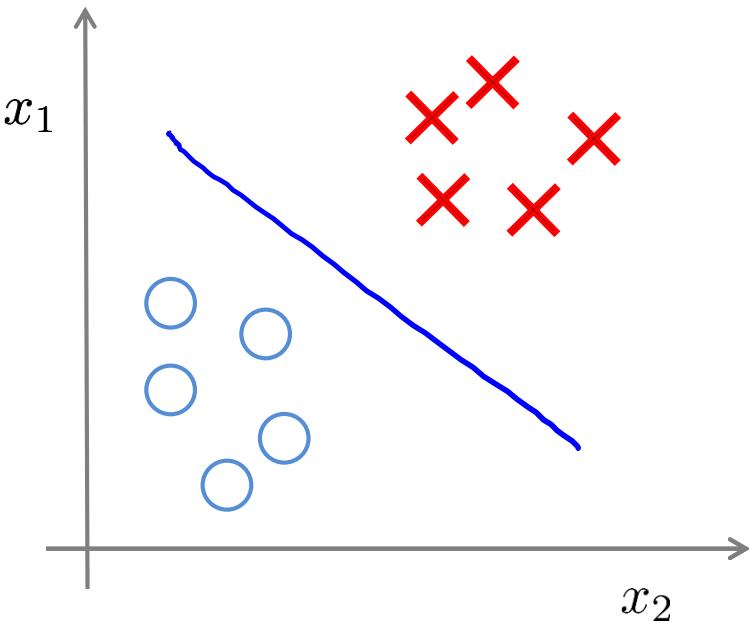


Machine Learning

Unsupervised Learning

Introduction

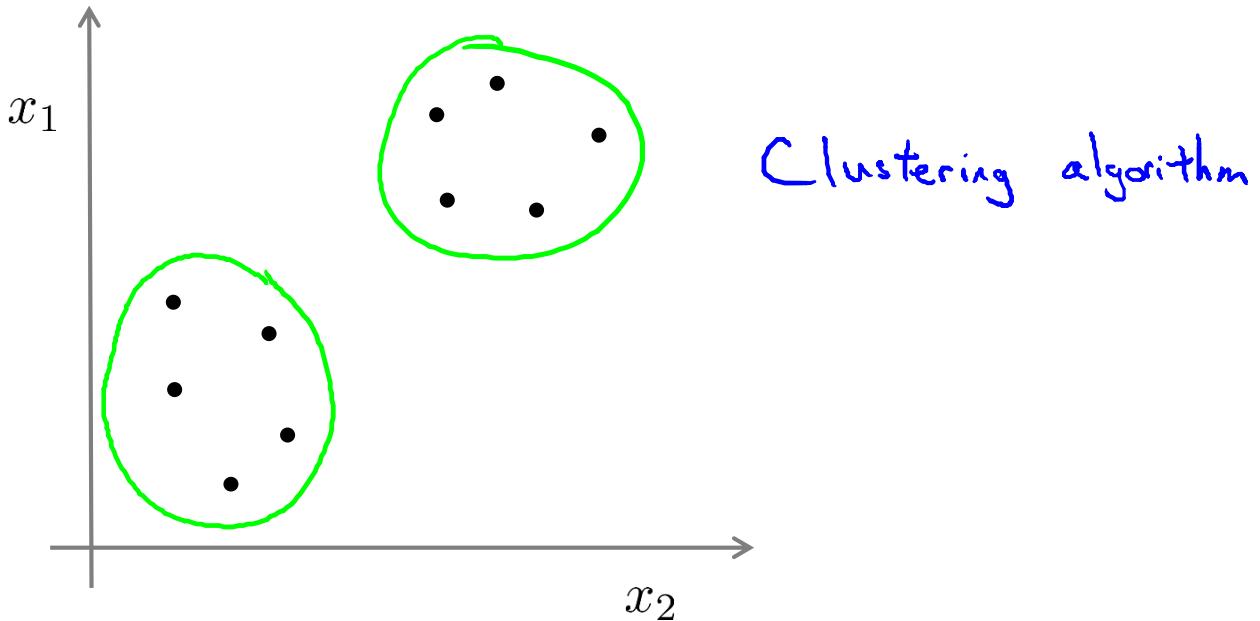
Supervised learning



Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots, (x^{(m)}, y^{(m)})\}$



Unsupervised learning



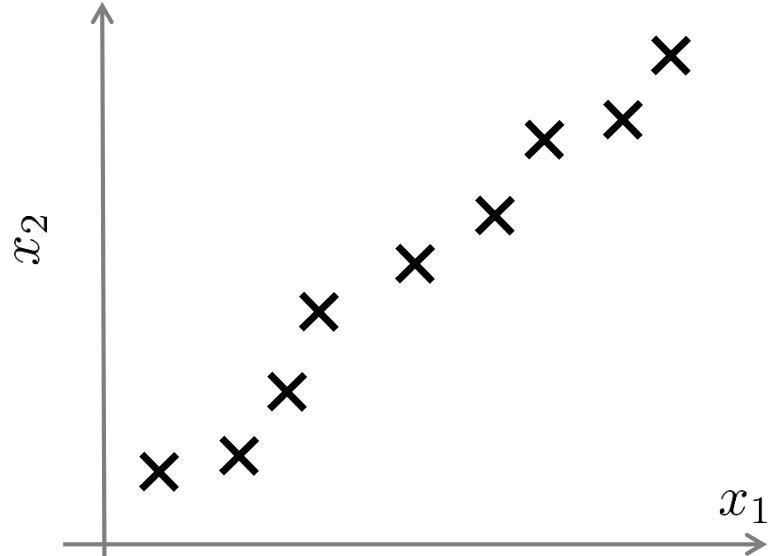
Training set: $\{\underline{x}^{(1)}, \underline{x}^{(2)}, \underline{x}^{(3)}, \dots, \underline{x}^{(m)}\}$ ←

Unsupervised Learning

- Finds undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision.
- Two of the main methods used in unsupervised learning are
 - principal component analysis (PCA), and
 - cluster analysis.
- Cluster Analysis
 - learning to group, or segment, datasets with shared attributes
- Principal Component Analysis
 - learning strategy is to learn a new feature space that captures the characteristics of the original space by maximizing some objective function or minimising some loss function.

Principal Component

- The principal components of a collection of points in a real p-space are a sequence of p direction vectors, where the i -th vector is the direction of a line that best fits the data while being orthogonal to the first $(i-1)$ vectors.
 - Here, a best-fitting line is defined as one that minimizes the average squared distance from the points to the line.



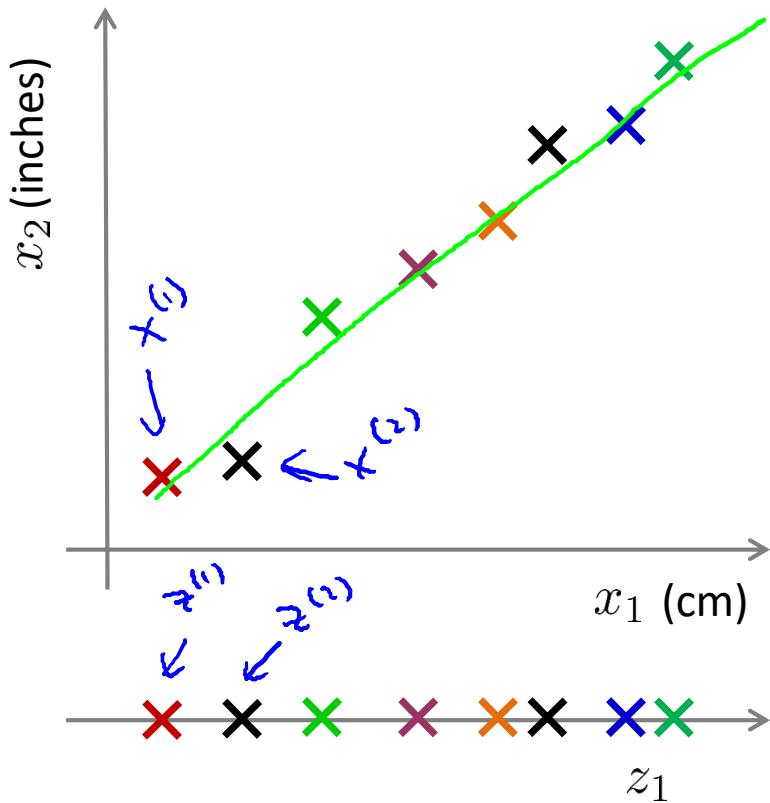
Principal Component Analysis

- **Principal component analysis (PCA)** is the process of computing the principal components and using them to perform a change of basis on the data,
 - Also known as Karhunen–Loève transform (KLT)
- Probably the most widely-used and well-known of the “standard” multivariate methods.
- Invented by Pearson (1901) and Hotelling (1933)
- Principal component analysis (PCA) is a way to reduce data dimensionality
- PCA projects the data in the least square sense— it captures big (principal) variability in the data and ignores small variability

Principal Component Analysis

- Principle
 - **Linear projection method** to reduce the number of variables
 - Transfer a set of correlated variables into a new set of uncorrelated variables
 - Map the data into a space of lower dimensionality
 - Form of unsupervised learning
- Properties
 - It can be viewed as a rotation of the existing axes to new positions in the space defined by original variables
 - New axes are orthogonal and represent the directions with maximum variability

Data Compression



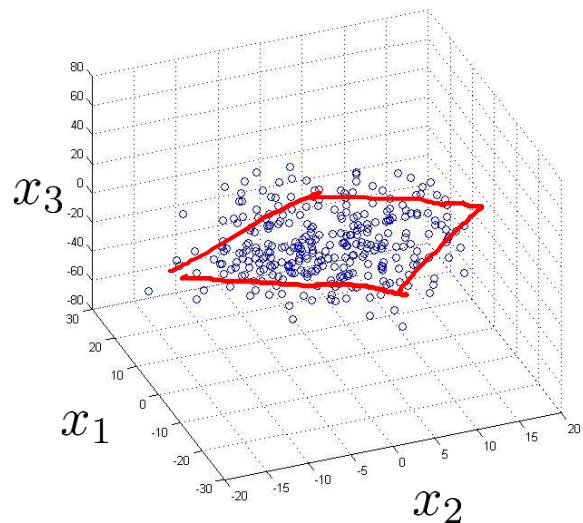
Reduce data from
2D to 1D

$$\begin{aligned} x^{(1)} \in \mathbb{R}^2 &\rightarrow z^{(1)} \in \mathbb{R} \\ x^{(2)} \in \mathbb{R}^2 &\rightarrow z^{(2)} \in \mathbb{R} \\ &\vdots \\ x^{(m)} \in \mathbb{R}^2 &\rightarrow z^{(m)} \in \mathbb{R} \end{aligned}$$

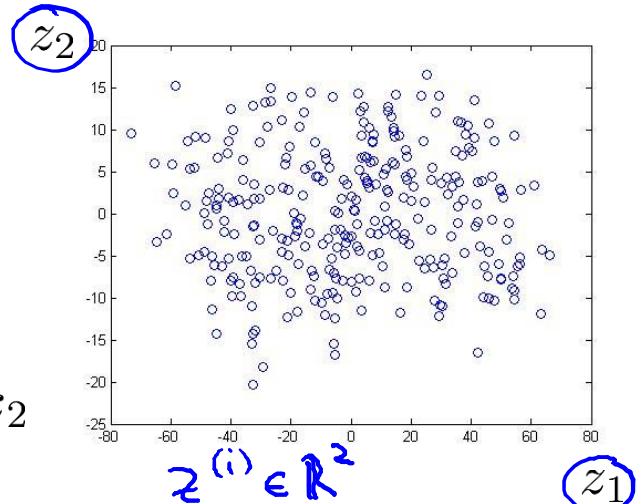
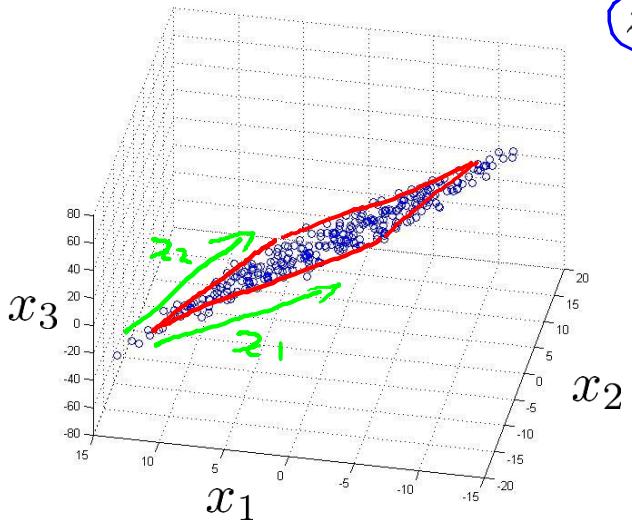
Data Compression

10000 \rightarrow 1000

Reduce data from 3D to 2D



$$x^{(i)} \in \mathbb{R}^3$$

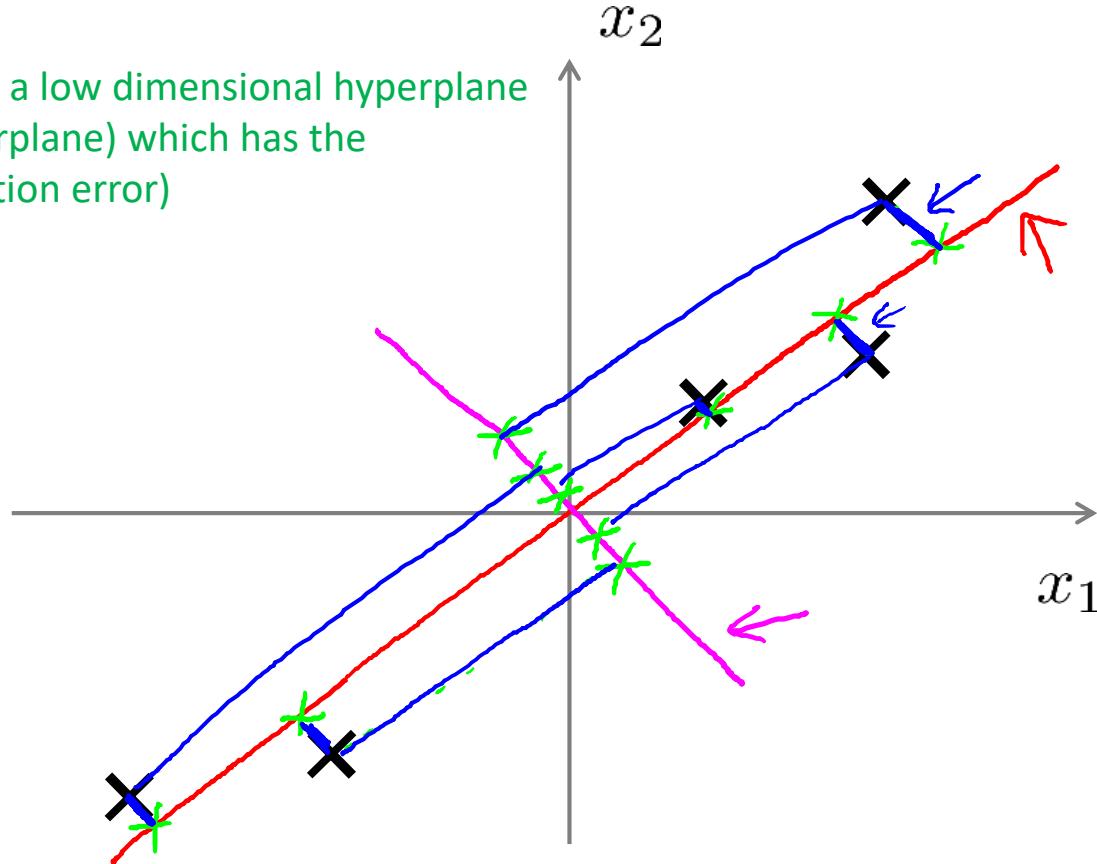


$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

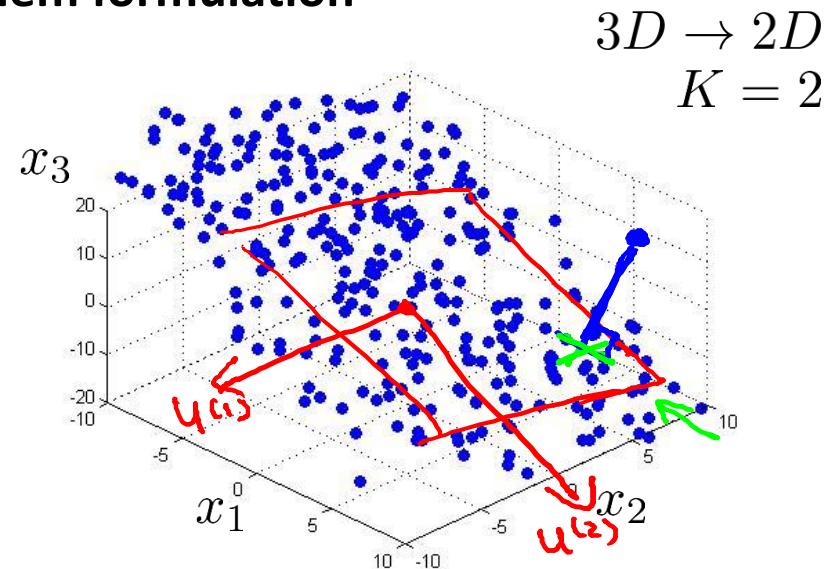
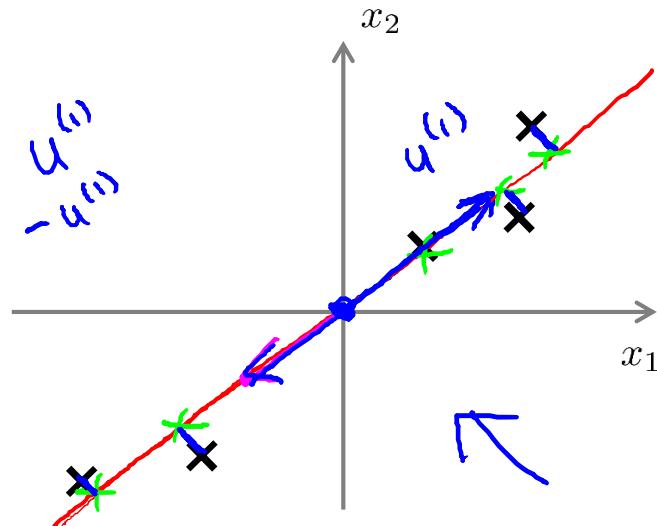
Principal Component Analysis (PCA) problem formulation

(PCA tries to find a low dimensional hyperplane (projection hyperplane) which has the minimum projection error)

$$x \in \mathbb{R}^2$$



Principal Component Analysis (PCA) problem formulation

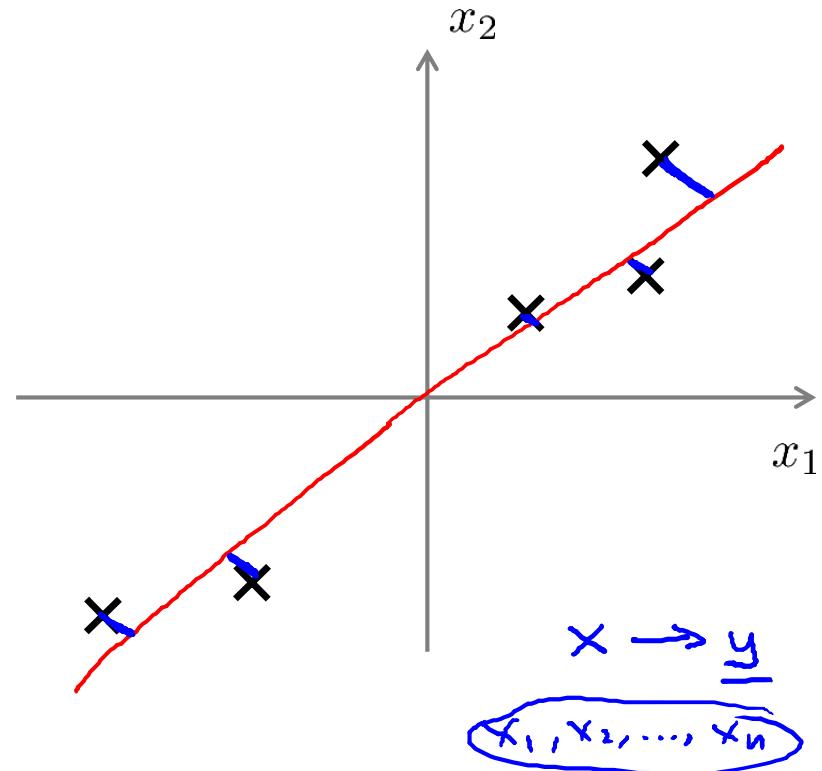
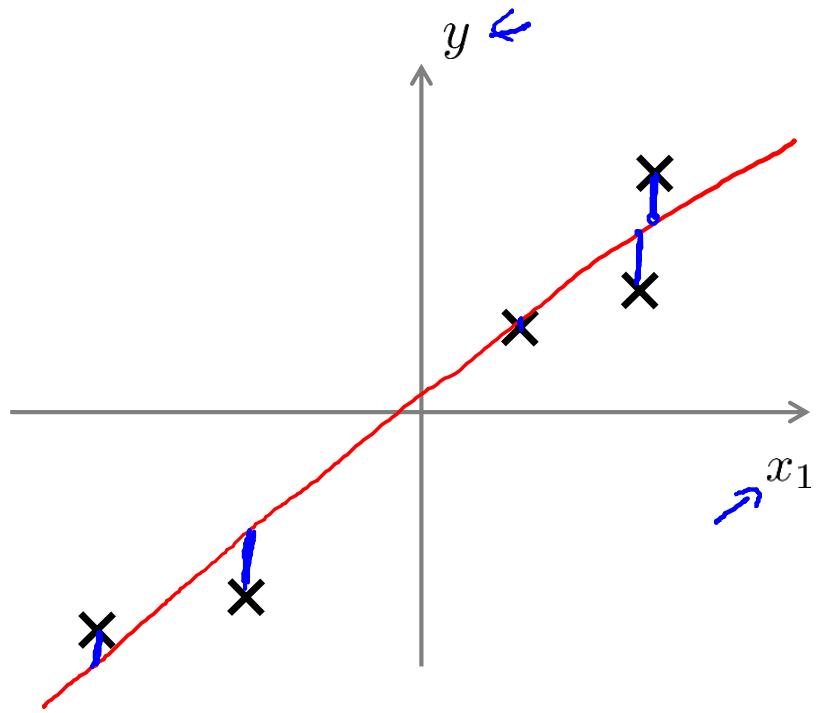


Reduce from 2-dimension to 1-dimension: Find a direction (a vector $\underline{u^{(1)} \in \mathbb{R}^n}$) onto which to project the data so as to minimize the projection error.

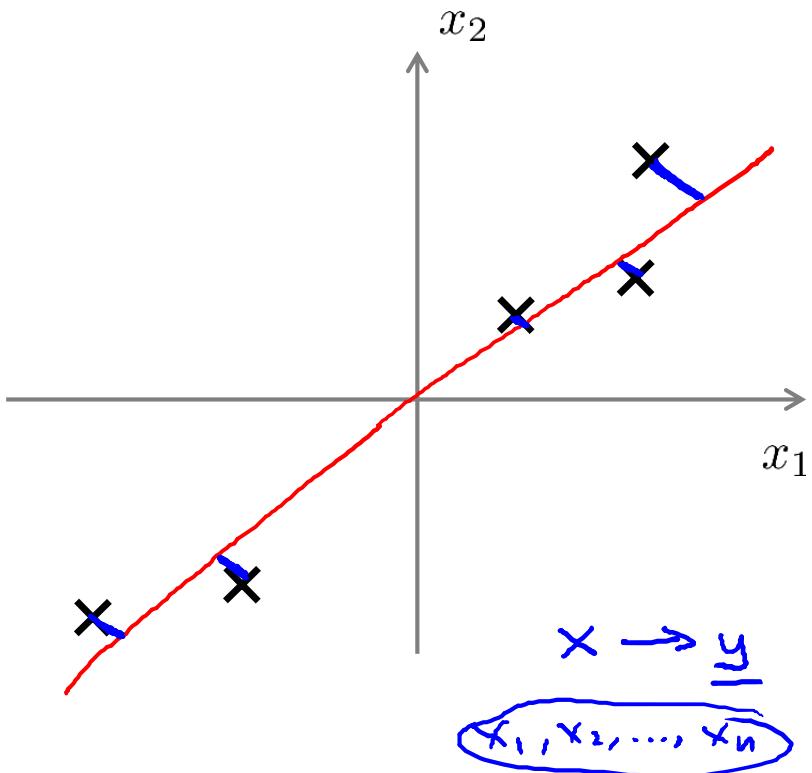
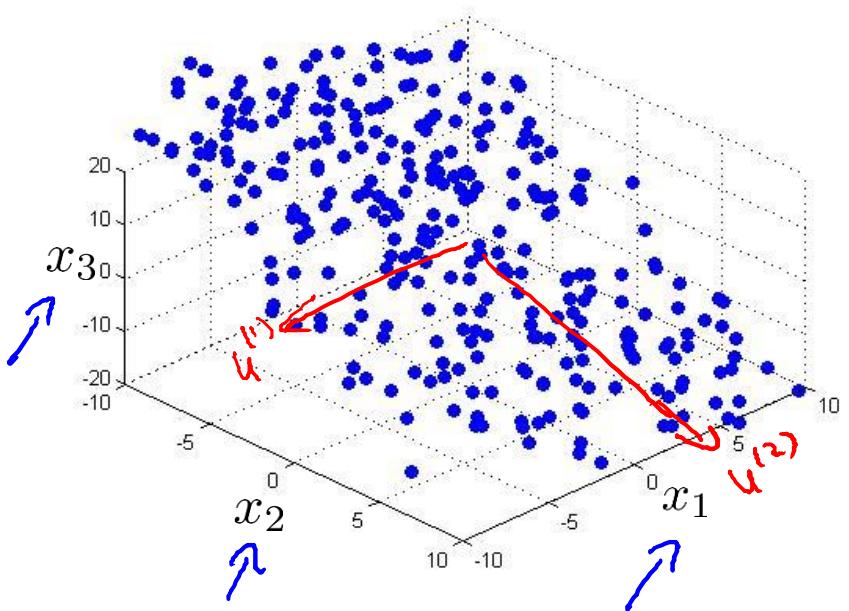
Reduce from n -dimension to k -dimension: Find k vectors $\underline{u^{(1)}, u^{(2)}, \dots, u^{(k)}}$ onto which to project the data, so as to minimize the projection error.

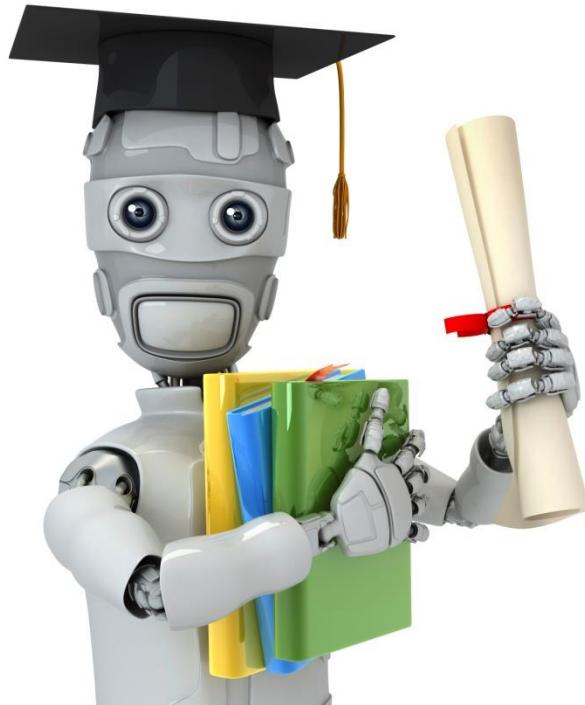
PCA is not linear regression

(PCA tries to minimize (squared) projection error not MSE)



PCA is not linear regression





Machine Learning

Dimensionality Reduction

Principal Component Analysis algorithm

Steps

1. Subtract the mean
2. Calculate the covariance matrix
3. Calculate the eigenvectors and eigenvalues
4. Choosing components and forming a feature vector

1. Subtract Mean

Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

Preprocessing (feature scaling/mean normalization):

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

- Replace each x_j with $(x_j - \mu_j)$.
- If different features on different scales (e.g., x_1 =size of house, x_2 =number of bedrooms), scale features to have comparable range of values.

1. Subtract Mean

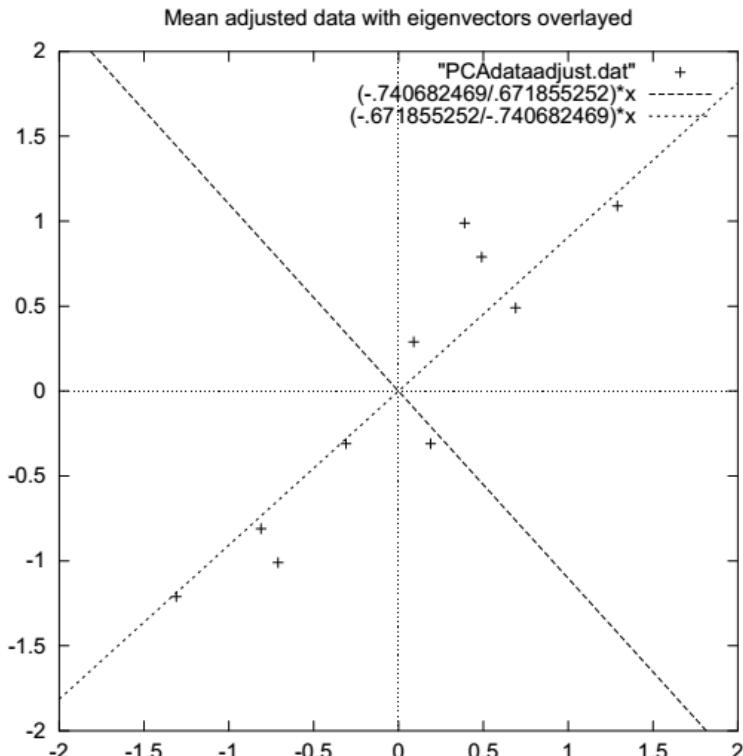
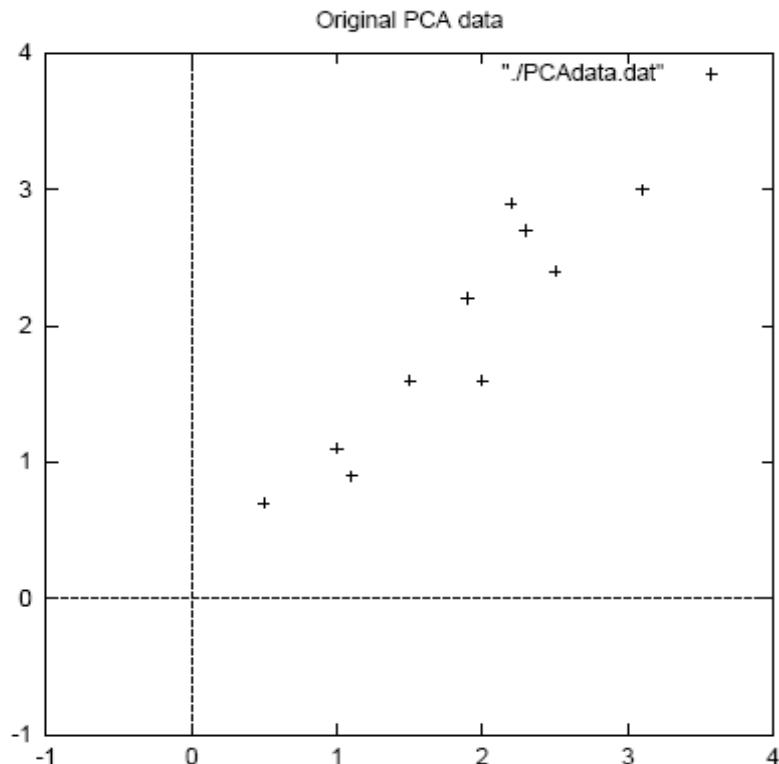


Figure 3.1: PCA example data, original data on the left, data with the means subtracted on the right, and a plot of the data

1. Subtract Mean

	x	y		x	y
Data =	2.5	2.4		.69	.49
	0.5	0.7		-1.31	-1.21
	2.2	2.9		.39	.99
	1.9	2.2		.09	.29
	3.1	3.0	DataAdjust =	1.29	1.09
	2.3	2.7		.49	.79
	2	1.6		.19	-.31
	1	1.1		-.81	-.81
	1.5	1.6		-.31	-.31
	1.1	0.9		-.71	-1.01

Subtracting the mean makes variance and covariance calculation easier by simplifying their equations. The variance and co-variance values are not affected by the mean value.

2. Calculate the covariance matrix

- Compute “covariance matrix”:

$$\Sigma = \frac{1}{m} \sum_{i=1}^n (x^{(i)})(x^{(i)})^T \quad cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

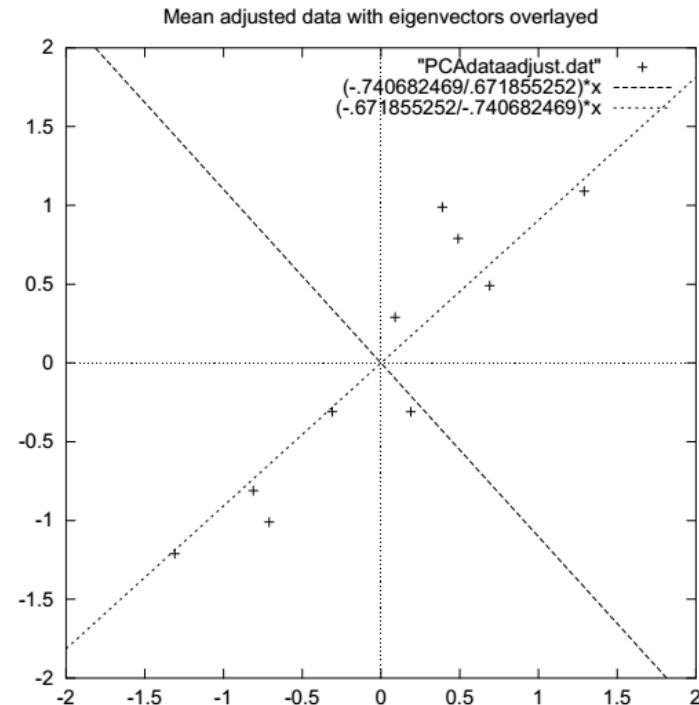
- Since the non-diagonal elements in this covariance matrix are positive, we should expect that both the x and y variable increase together.
- Gives information about shape of data distribution.

3. Calculate the eigenvectors and eigenvalues

- Singular Value Decomposition to get the eigenvectors and eigenvalues:

```
[U,S,V] = svd(Sigma);
```

$$U = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(n)} \end{bmatrix} \in \mathbb{R}^{n \times n}$$



4. Choosing components and forming a feature vector

- The eigenvector with the *highest* eigenvalue is the *principle component* of the data set.
- Once eigenvectors are found from the covariance matrix, the next step is to order them by eigenvalue, highest to lowest.
 - This gives you the components in order of significance.
- Form a new feature vector by projecting onto the selected eigenvectors.

PCA with all Eigenvectors

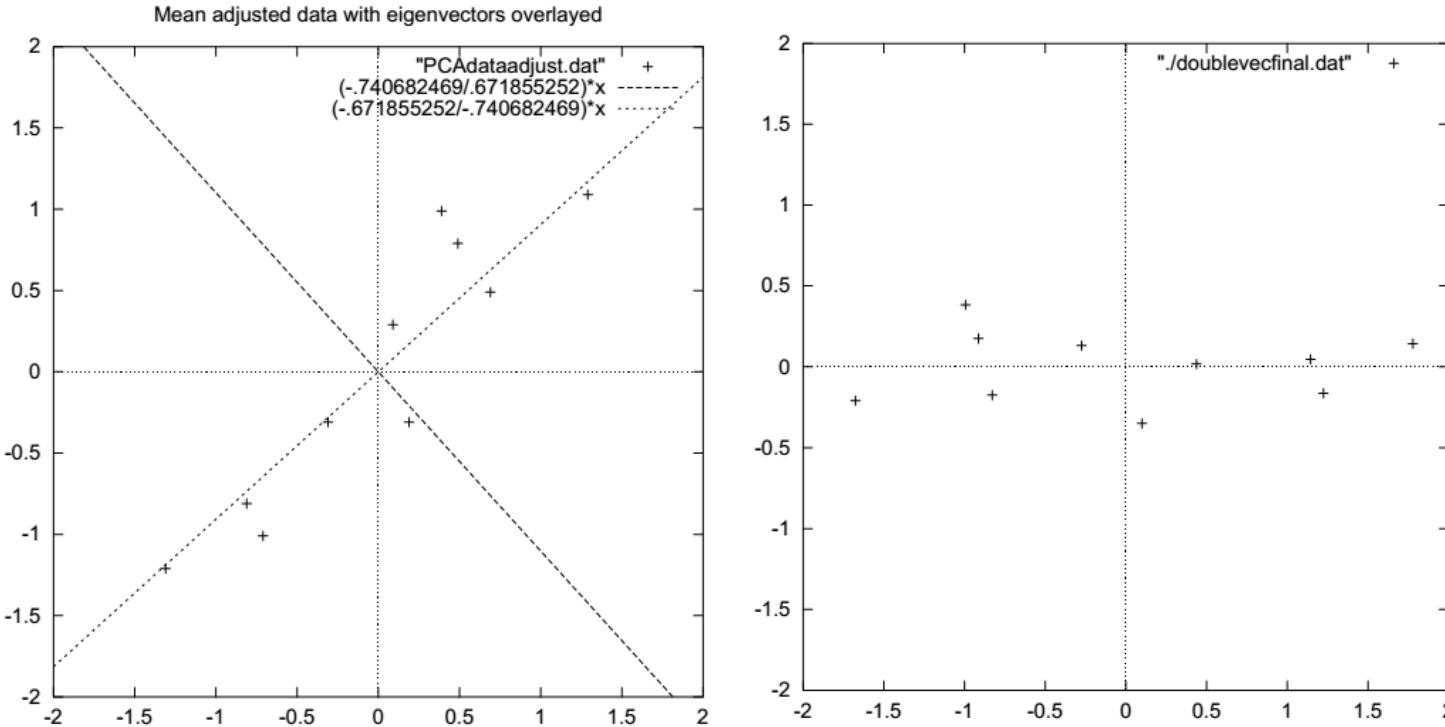
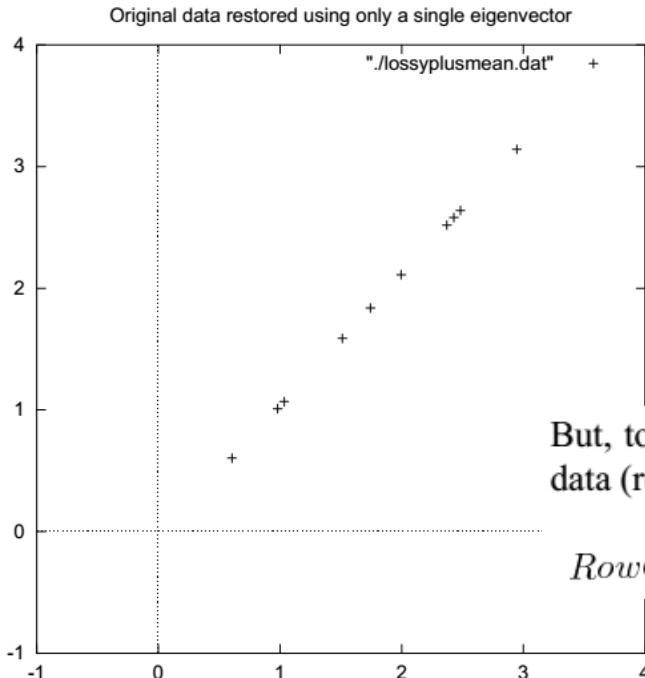


Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

Data Reconstruction



Recall that the final transform is this:

$$\text{FinalData} = \text{RowFeatureVector} \times \text{RowDataAdjust},$$

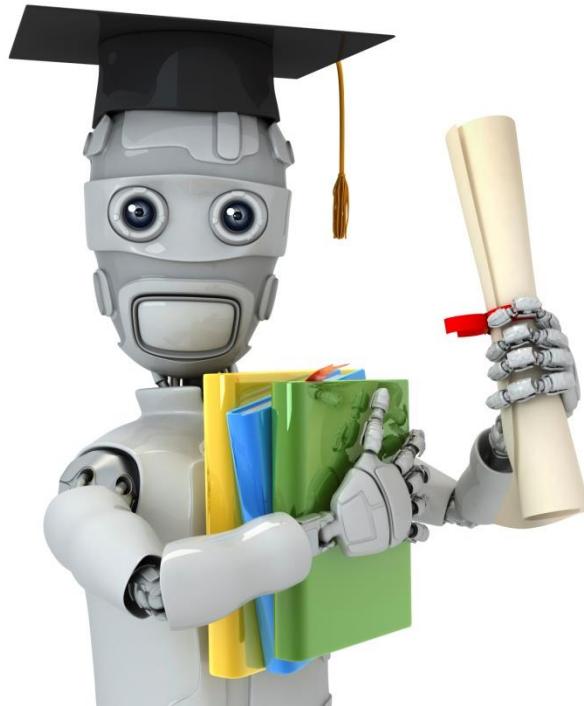
which can be turned around so that, to get the original data back,

$$\text{RowDataAdjust} = \text{RowFeatureVector}^{-1} \times \text{FinalData}$$

But, to get the actual original data back, we need to add on the mean of that original data (remember we subtracted it right at the start). So, for completeness,

$$\text{RowOriginalData} = (\text{RowFeatureVector}^T \times \text{FinalData}) + \text{OriginalMean}$$

Figure 3.5: The reconstruction from the data that was derived using only a single eigenvector



Machine Learning

Dimensionality Reduction

Choosing the number of principal components

Choosing k (number of principal components)

Average squared projection error: $\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data: $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$

Typically, choose k to be smallest value so that

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq \frac{0.01}{0.05} \quad \frac{(1\%)}{5\%}$$

→ “99% of variance is retained”

~~95%~~ 90%

Choosing k (number of principal components)

Algorithm:

Try PCA with $k = 1$ ~~$k=2$~~ ~~$k=3$~~ ~~$k=4$~~

Compute $U_{reduce}, z^{(1)}, z^{(2)}, \dots, z^{(m)}, x_{approx}^{(1)}, \dots, x_{approx}^{(m)}$

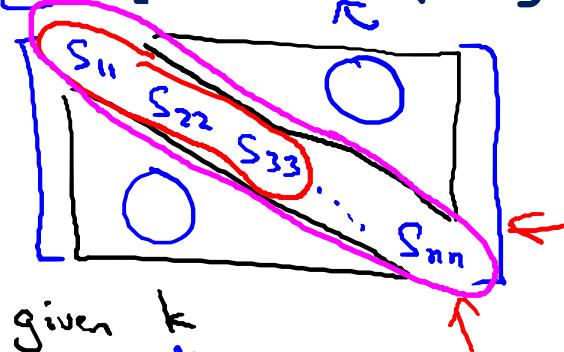
Check if

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01?$$

$k = 17$

$$\rightarrow [U, S, V] = svd(\Sigma)$$

$$\rightarrow S =$$



For given k

$$1 - \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \leq 0.01$$

$$\rightarrow \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \geq 0.99$$

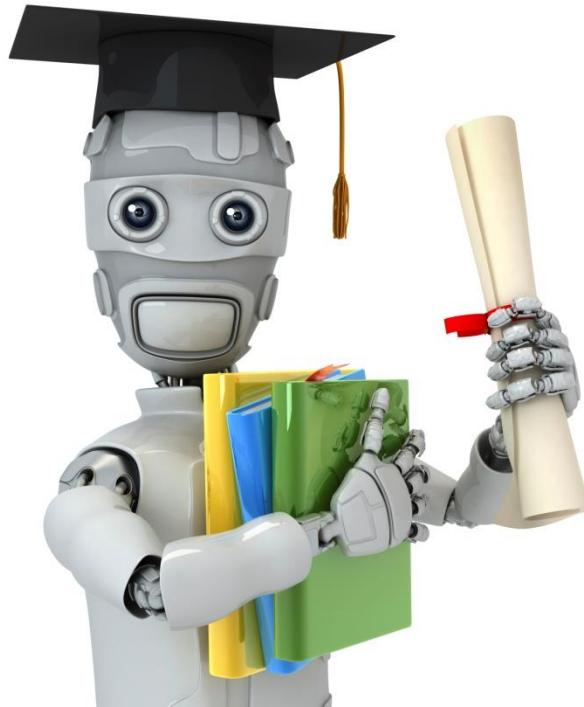
Choosing k (number of principal components)

$[U, S, V] = \text{svd}(\Sigma)$

Pick smallest value of k for which

$$\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^m S_{ii}} \geq 0.99$$

(99% of variance retained)



Machine Learning

Dimensionality Reduction

Advice for applying PCA

Supervised learning speedup

$$X^{(i)} \in \mathbb{R}^{10,000} \quad \begin{matrix} & 100 \\ & \downarrow \\ 100 & \boxed{} \end{matrix}$$

→ $(\underline{x}^{(1)}, y^{(1)}), (\underline{x}^{(2)}, y^{(2)}), \dots, (\underline{x}^{(m)}, y^{(m)})$

Extract inputs:

Unlabeled dataset: $\underline{x}^{(1)}, \underline{x}^{(2)}, \dots, \underline{x}^{(m)} \in \mathbb{R}^{10000}$ $\xleftarrow{\downarrow PCA}$

$$\underline{z}^{(1)}, \underline{z}^{(2)}, \dots, \underline{z}^{(m)} \in \mathbb{R}^{1000} \quad \begin{matrix} & x \\ & \downarrow \\ & z \end{matrix}$$

New training set:

$$(\underline{z}^{(1)}, y^{(1)}), (\underline{z}^{(2)}, y^{(2)}), \dots, (\underline{z}^{(m)}, y^{(m)})$$

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

Note: Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation and test sets.

Application of PCA

- Compression
 - Reduce memory/disk needed to store data
 - Speed up learning algorithm ←

Choose k by % of variance retain

- Visualization

$k=2$ or $k=3$

Bad use of PCA: To prevent overfitting

- Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$.
- Thus, fewer features, less likely to overfit.

Bad!

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

$$\rightarrow \min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \boxed{\frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2}$$


PCA is sometimes used where it shouldn't be

Design of ML system:

- - Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
- - ~~Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$~~
- - Train logistic regression on $\{(z^{(1)}, y^{(1)}), \dots, (z^{(m)}, y^{(m)})\}$
- - Test on test set: Map $x_{test}^{(i)}$ to $z_{test}^{(i)}$. Run $h_\theta(z)$ on $\{(z_{test}^{(1)}, y_{test}^{(1)}), \dots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

- How about doing the whole thing without using PCA?
- Before implementing PCA, first try running whatever you want to do with the original/raw data $x^{(i)}$. Only if that doesn't do what you want, then implement PCA and consider using $\underline{z^{(i)}}$.



CSE 4621

Machine Learning

Lecture 14

Md. Hasanul Kabir, PhD.

Professor, CSE Department

Islamic University of Technology (IUT)



What is Cluster Analysis?

- Cluster: A collection of data objects
 - similar (or related) to one another within the same group
 - dissimilar (or unrelated) to the objects in other groups
- Cluster analysis (or *clustering, data segmentation, ...*)
 - Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters
- **Unsupervised learning**: no predefined classes (i.e., *learning by observations* vs. learning by examples: supervised)
- Typical applications
 - As a **stand-alone tool** to get insight into data distribution
 - As a **preprocessing step** for other algorithms

Clustering for Data Understanding and Applications

- Biology: taxonomy of living things: kingdom, phylum, class, order, family, genus and species
- Information retrieval: document clustering
- Land use: Identification of areas of similar land use in an earth observation database
- Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- City-planning: Identifying groups of houses according to their house type, value, and geographical location
- Earthquake studies: Observed earth quake epicenters should be clustered along continent faults
- Web Search: Clustering can be used to organize the search results into groups and present the results in a concise and easily accessible way.
- Information Retrieval: Cluster documents into topics.

Clustering as a Preprocessing Tool (Utility)

- Summarization:
 - Preprocessing for regression, PCA, classification, and association analysis
- Compression:
 - Image processing: vector quantization
- Finding K-nearest Neighbors
 - Localizing search to one or a small number of clusters
- Outlier detection
 - Outliers are often viewed as those “far away” from any cluster

Quality: What Is Good Clustering?

- A good clustering method will produce high quality clusters
 - high intra-class similarity: **cohesive** within clusters
 - low inter-class similarity: **distinctive** between clusters
- The quality of a clustering method depends on
 - the similarity measure used by the method
 - its implementation, and
 - Its ability to discover some or all of the hidden patterns

Measure the Quality of Clustering

- **Dissimilarity/Similarity metric**
 - Similarity is expressed in terms of a distance function, typically metric: $d(i, j)$
 - The definitions of **distance functions** are usually rather different for interval-scaled, boolean, categorical, ordinal ratio, and vector variables
 - Weights should be associated with different variables based on applications and data semantics
- Quality of clustering:
 - There is usually a separate “quality” function that measures the “goodness” of a cluster.
 - It is hard to define “similar enough” or “good enough”
 - The answer is typically highly subjective

Considerations for Cluster Analysis

- Partitioning criteria
 - Single level vs. hierarchical partitioning (often, multi-level hierarchical partitioning is desirable). E.g. Politics, Sports: Football, Cricket, volleyball, etc.
- Separation of clusters
 - Exclusive (e.g., one customer belongs to only one region) vs. non-exclusive (e.g., one document may belong to more than one class)
- Similarity measure
 - Distance-based (e.g., Euclidian, road network, vector) vs. connectivity-based (e.g., density or contiguity)
 - Distance-based methods can often take advantage of optimization techniques, density- and continuity-based methods can often find clusters of arbitrary shape
- Clustering space
 - Full space (often when low dimensional) vs. subspaces (often in high-dimensional clustering)

Major Clustering Approaches

- Partitioning approach:
 - Construct various partitions and then evaluate them by some criterion, e.g., minimizing the sum of square errors
 - Typical methods: k-means, k-medoids, CLARANS
- Hierarchical approach:
 - Create a hierarchical decomposition of the set of data (or objects) using some criterion
 - Typical methods: Diana, Agnes, BIRCH, CAMELEON
- Density-based approach:
 - Based on connectivity and density functions
 - Typical methods: DBSCAN, OPTICS, DenClue
- Grid-based approach:
 - based on a multiple-level granularity structure
 - quantize object space into a finite number of cells (grid structure)
 - Typical methods: STING, WaveCluster, CLIQUE

Overview of Clustering Methods

Method	General Characteristics
Partitioning methods	<ul style="list-style-type: none">– Find mutually exclusive clusters of spherical shape– Distance-based– May use mean or medoid (etc.) to represent cluster center– Effective for small- to medium-size data sets
Hierarchical methods	<ul style="list-style-type: none">– Clustering is a hierarchical decomposition (i.e., multiple levels)– Cannot correct erroneous merges or splits– May incorporate other techniques like microclustering or consider object “linkages”
Density-based methods	<ul style="list-style-type: none">– Can find arbitrarily shaped clusters– Clusters are dense regions of objects in space that are separated by low-density regions– Cluster density: Each point must have a minimum number of points within its “neighborhood”– May filter out outliers
Grid-based methods	<ul style="list-style-type: none">– Use a multiresolution grid data structure– Fast processing time (typically independent of the number of data objects, yet dependent on grid size)

Partitioning Algorithms: Basic Concept

- Partitioning method: Partitioning a database D of n objects into a set of k clusters, such that the sum of squared distances is minimized (where c_i is the centroid or medoid of cluster C_i)

$$E = \sum_{i=1}^k \sum_{p \in C_i} \| p - c_i \|^2$$

- Given $k \leq n$, find a partition of k clusters that optimizes the chosen partitioning criterion
 - Global optimal: exhaustively enumerate all partitions
 - Heuristic methods: *k-means* and *k-medoids* algorithms
 - *k-means* (MacQueen'67, Lloyd'57/'82): Each cluster is represented by the center of the cluster
 - *k-medoids* or PAM (Partition around medoids) (Kaufman & Rousseeuw'87): Each cluster is represented by one of the objects in the cluster

The *K-Means* Clustering Method

- Given k , the *k-means* algorithm is implemented as

Algorithm: *k-means*. The *k-means* algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

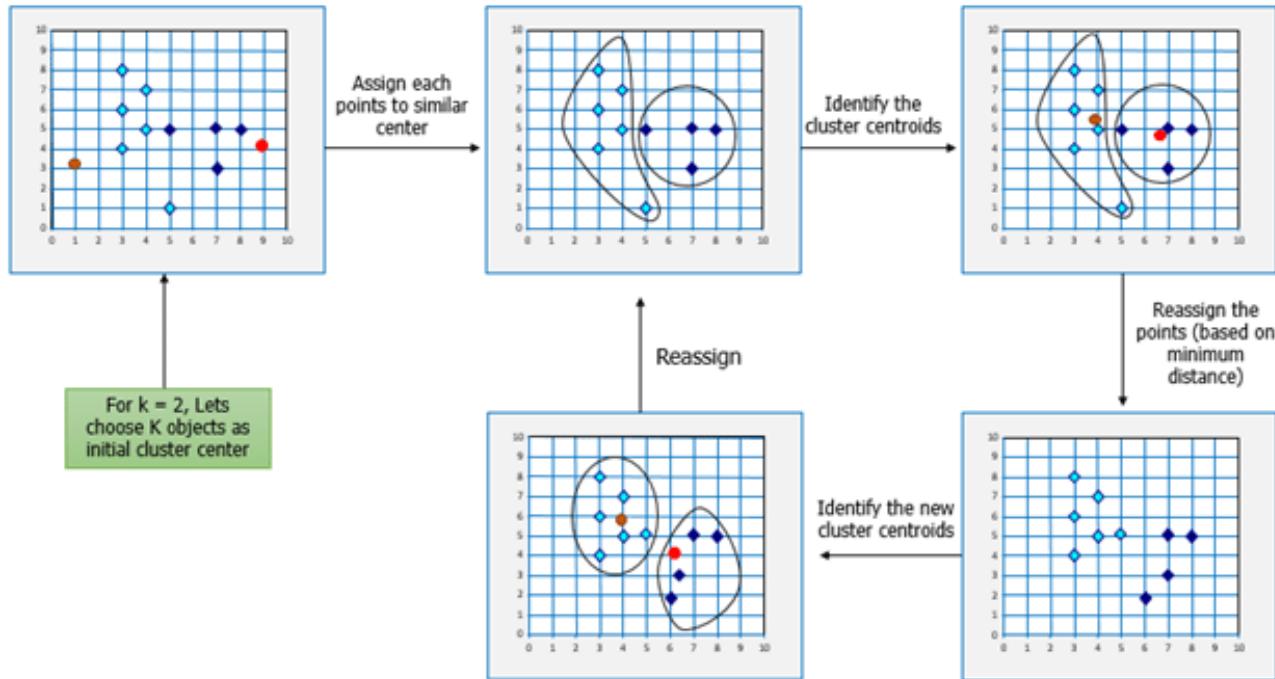
- k : the number of clusters,
- D : a data set containing n objects.

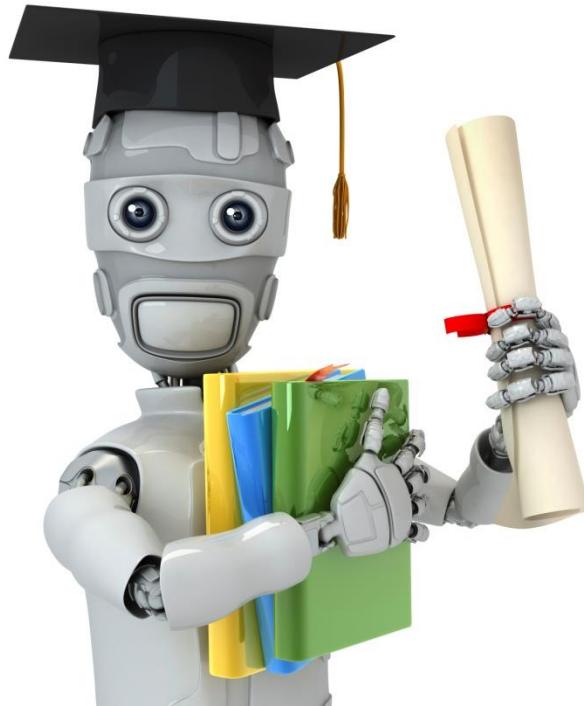
Output: A set of k clusters.

Method:

- (1) arbitrarily choose k objects from D as the initial cluster centers;
- (2) **repeat**
- (3) (re)assign each object to the cluster to which the object is the most similar,
 based on the mean value of the objects in the cluster;
- (4) update the cluster means, that is, calculate the mean value of the objects for
 each cluster;
- (5) **until** no change;

An Example of K-Means Clustering



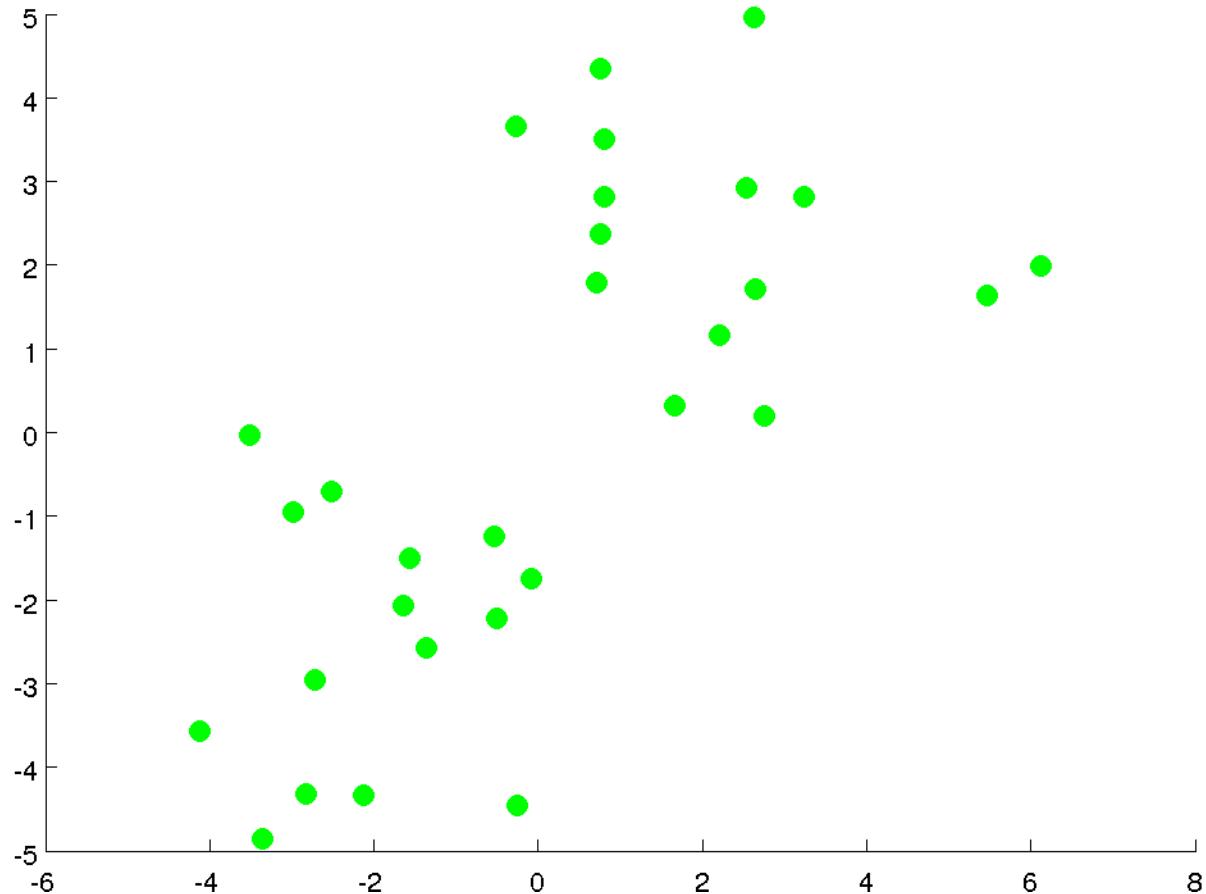


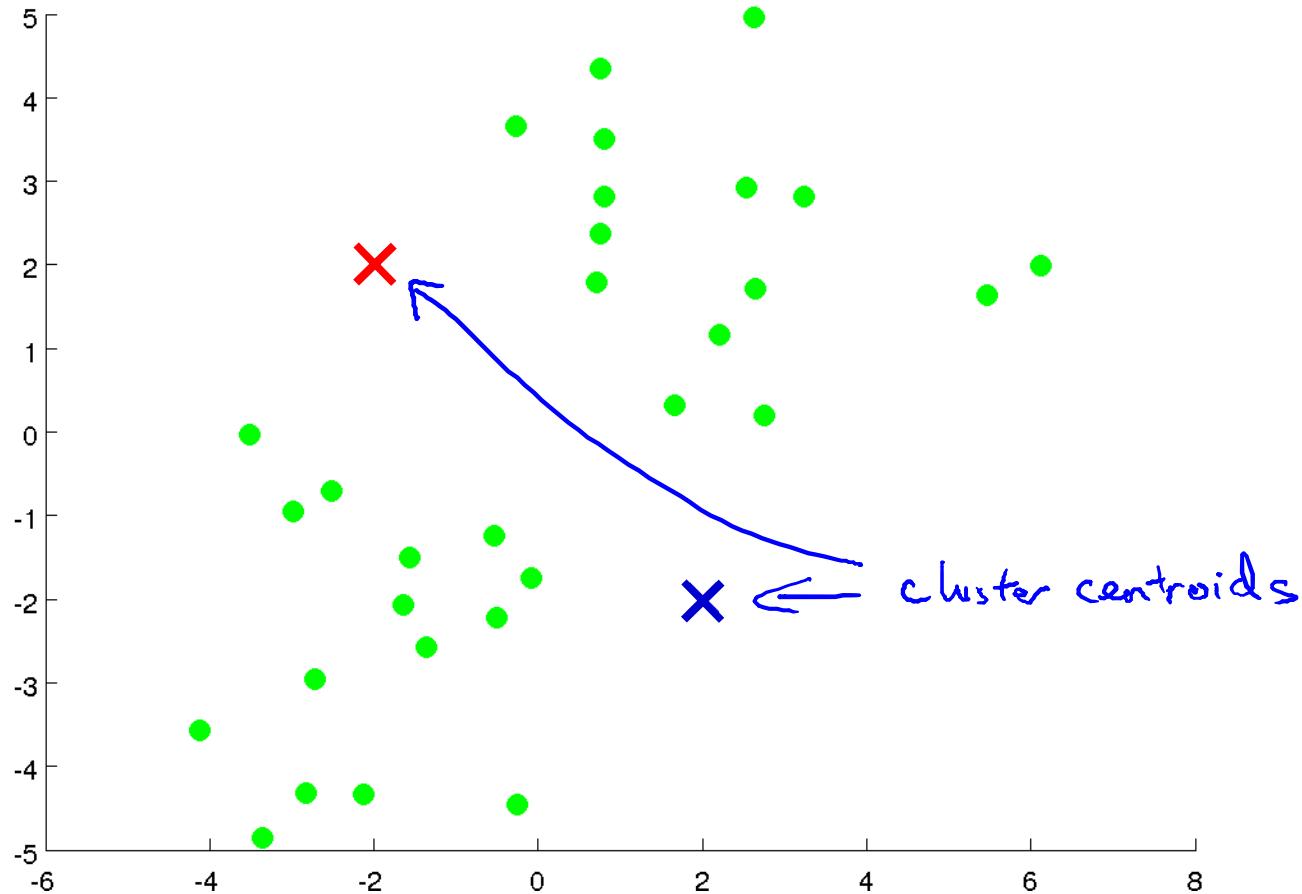
Machine Learning

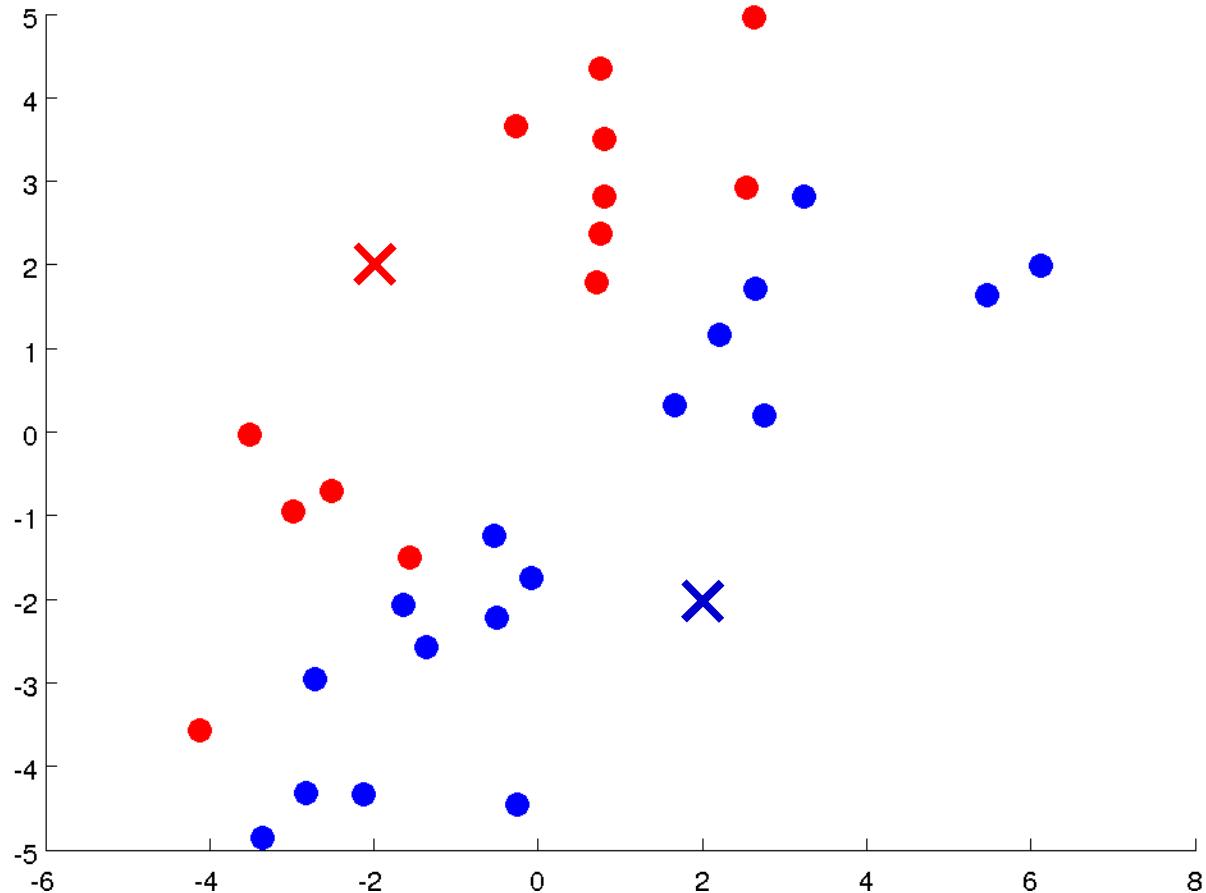
Clustering

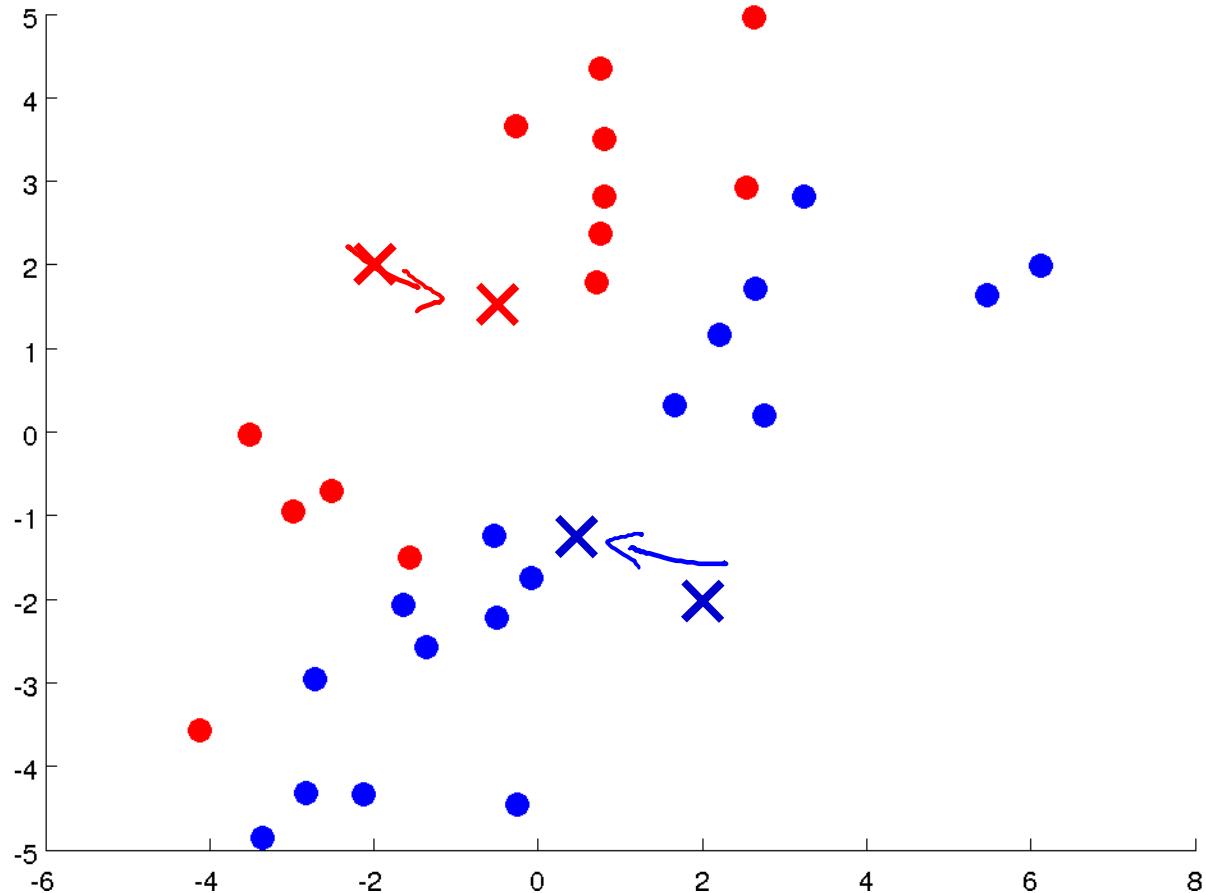
K-means

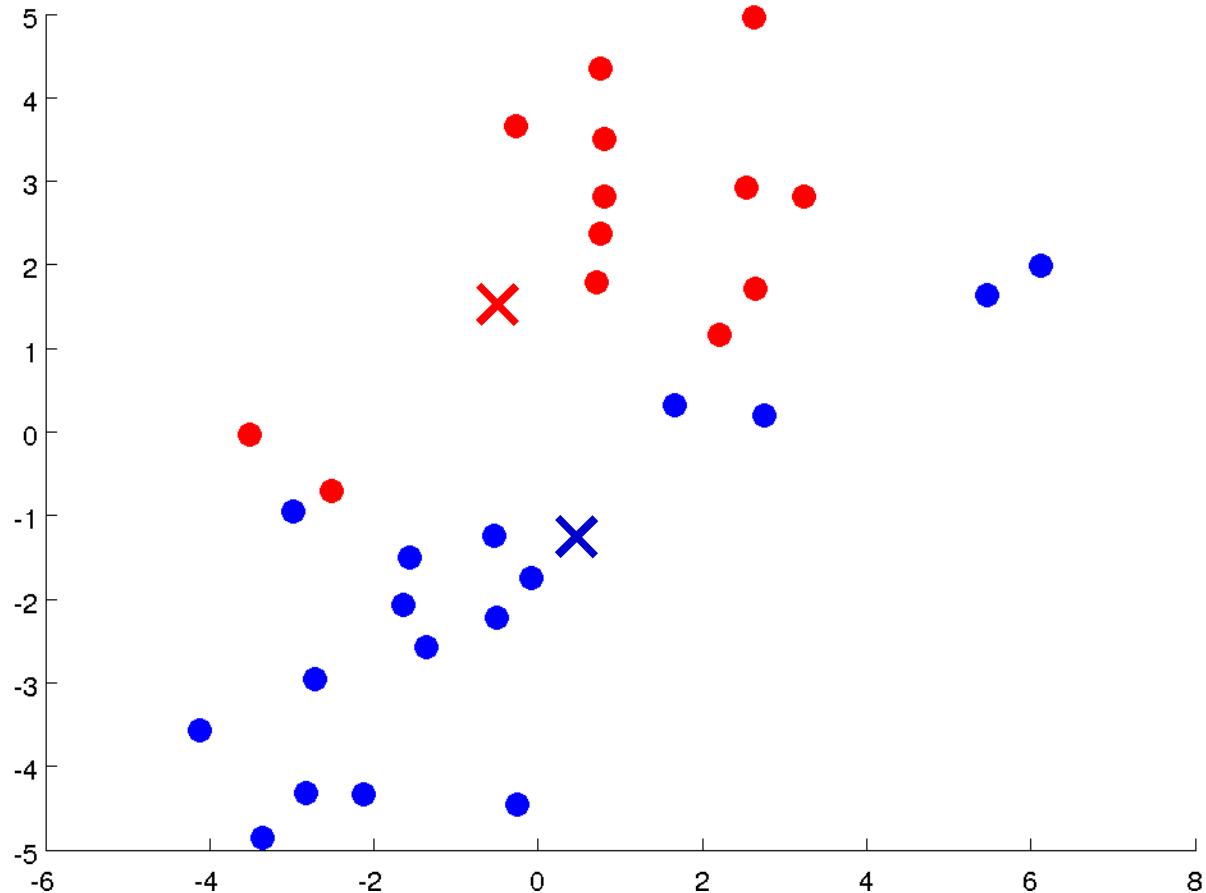
Example

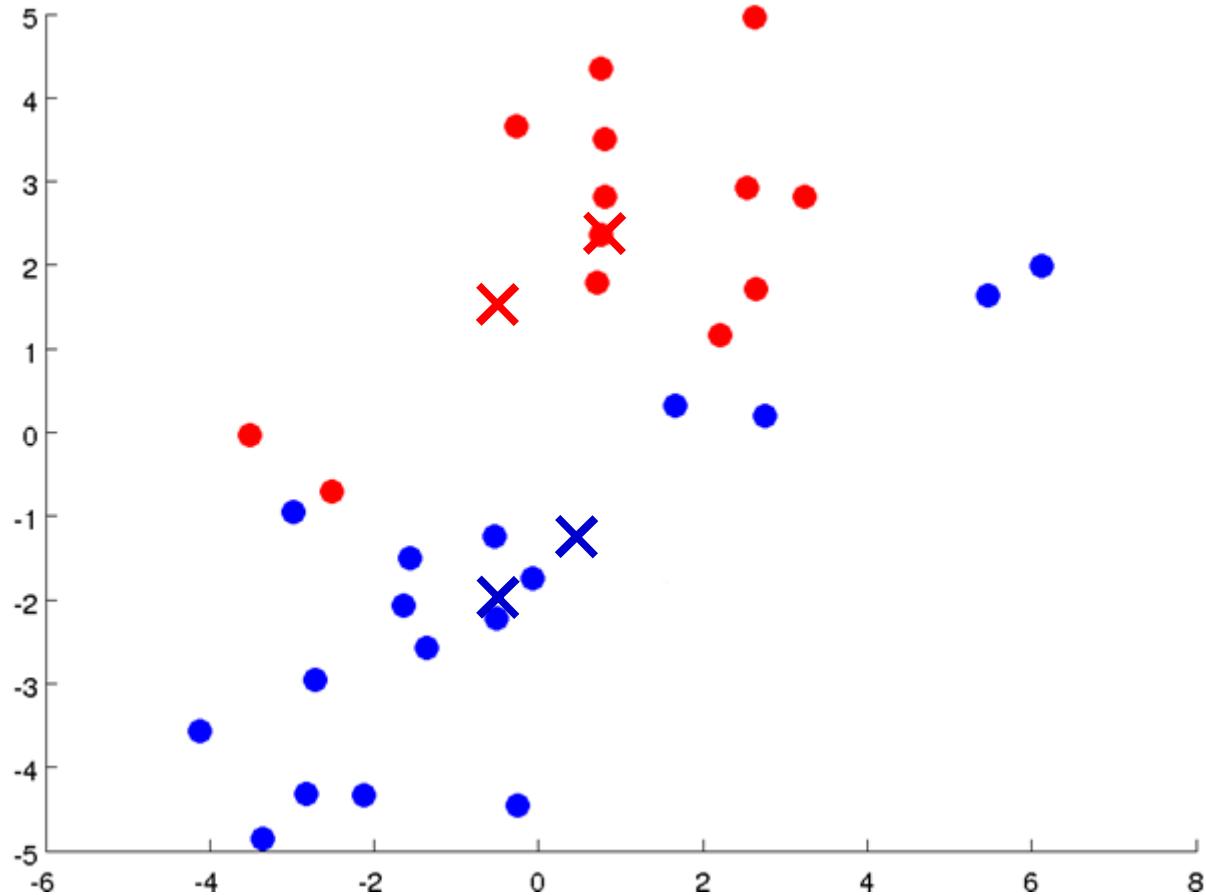


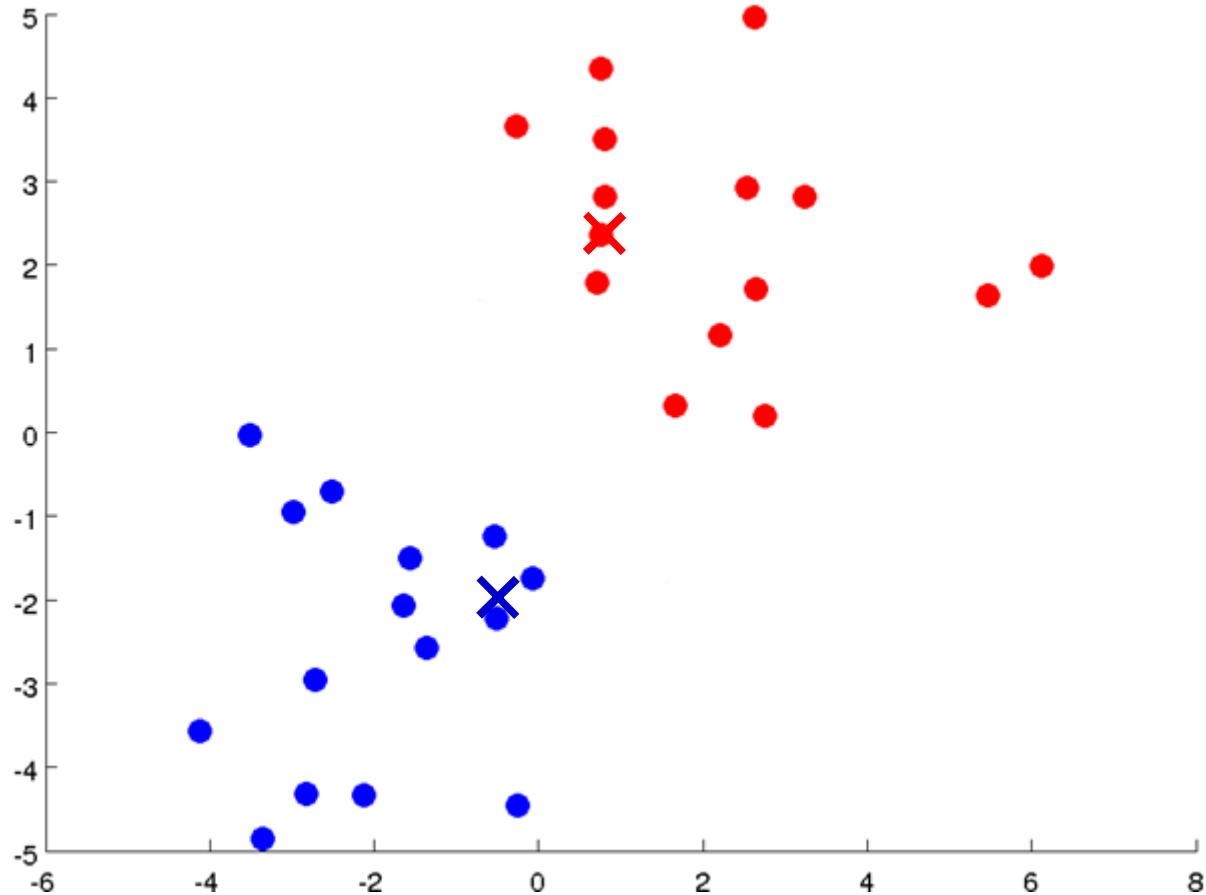


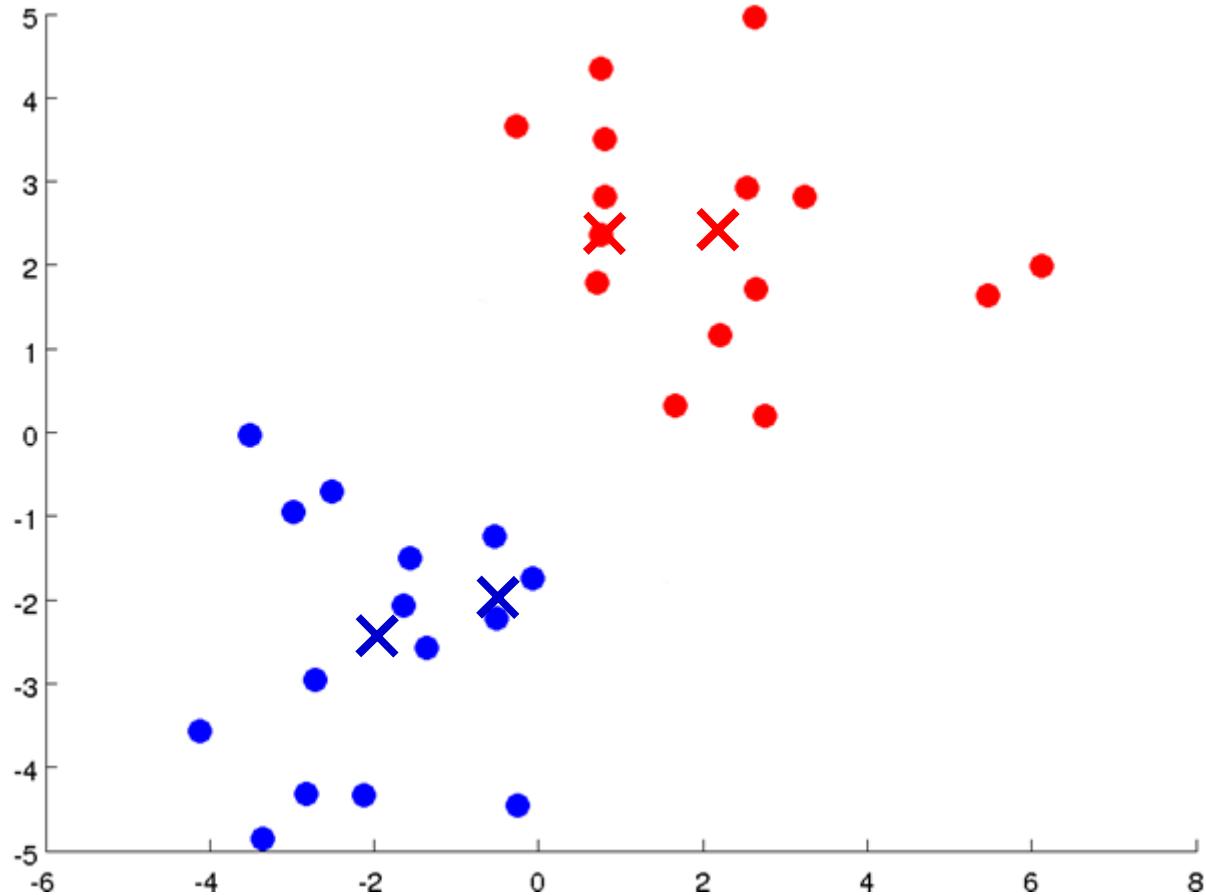


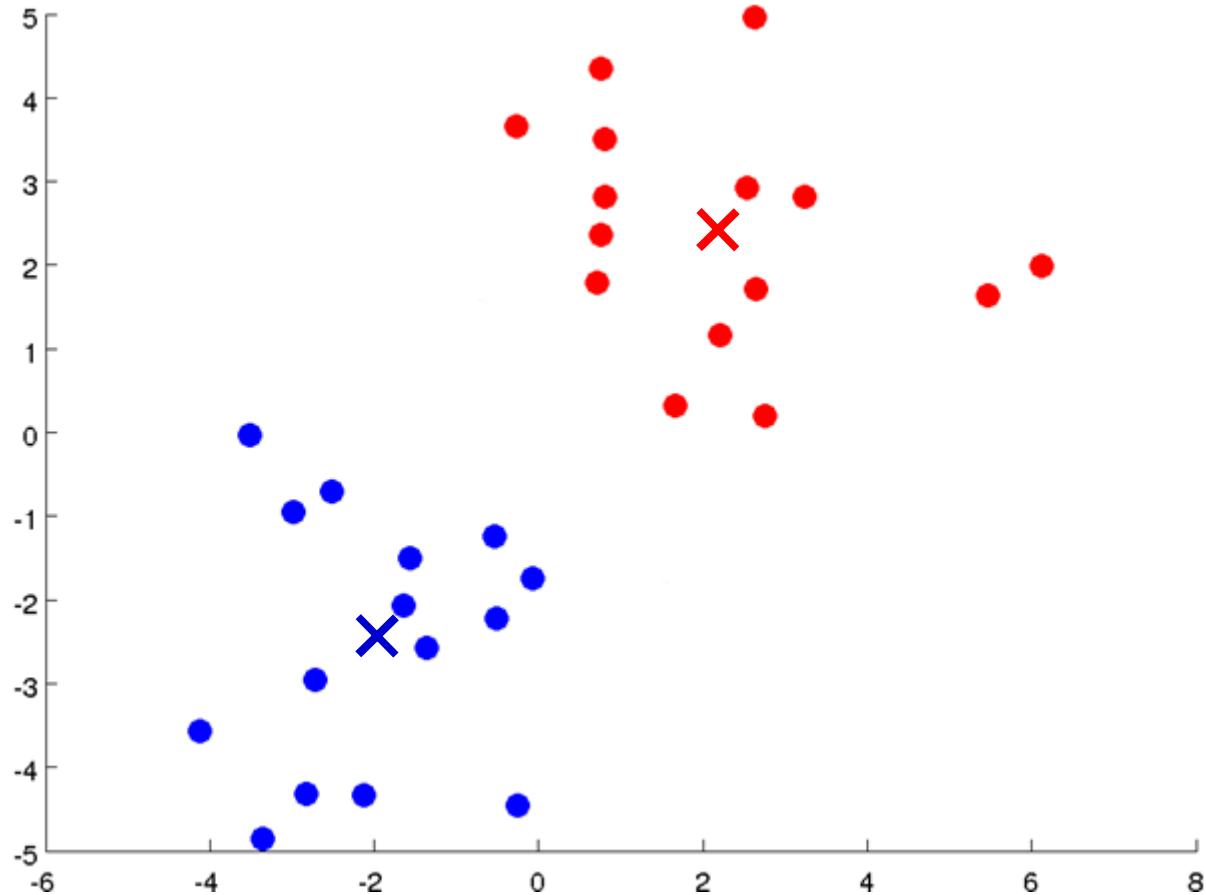




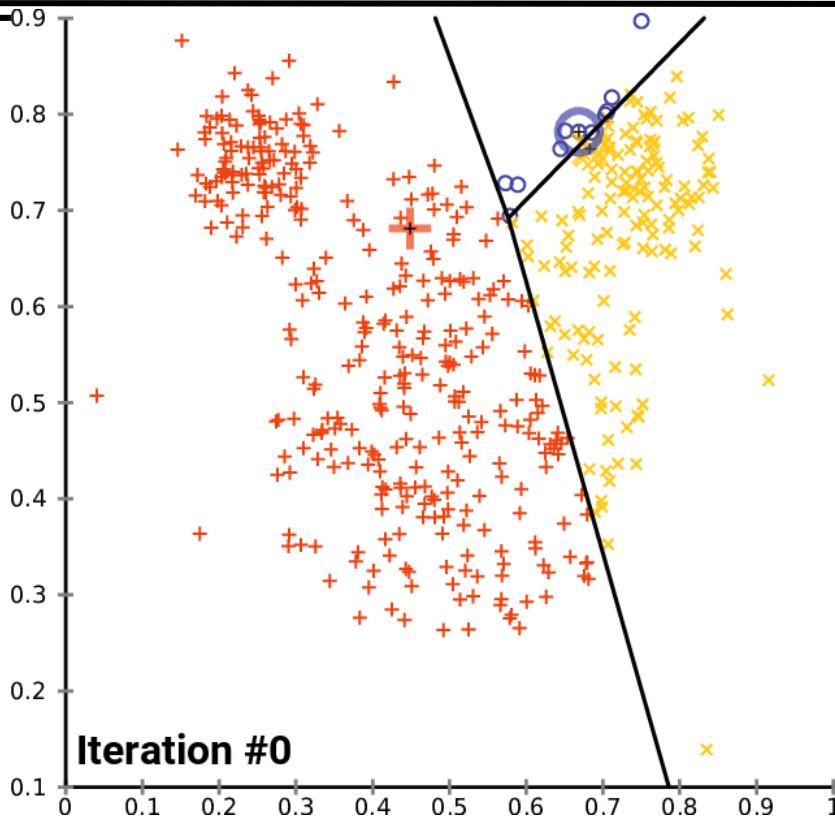




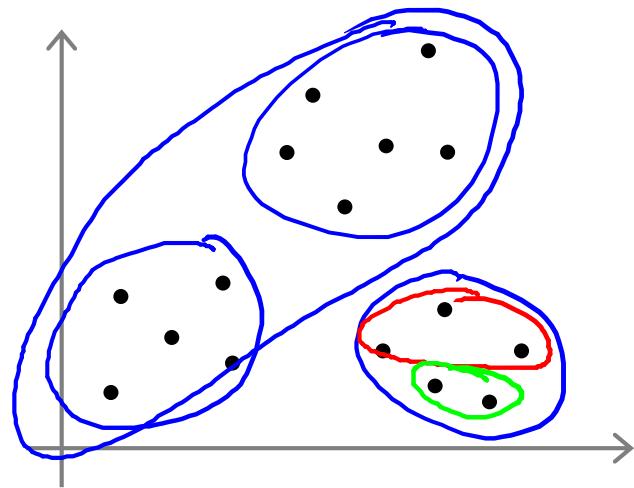




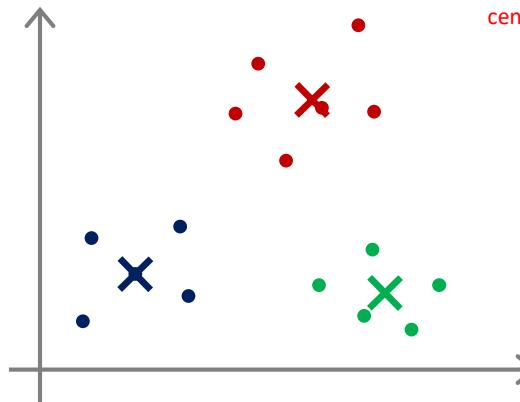
Graphical Example of K-Means Clustering



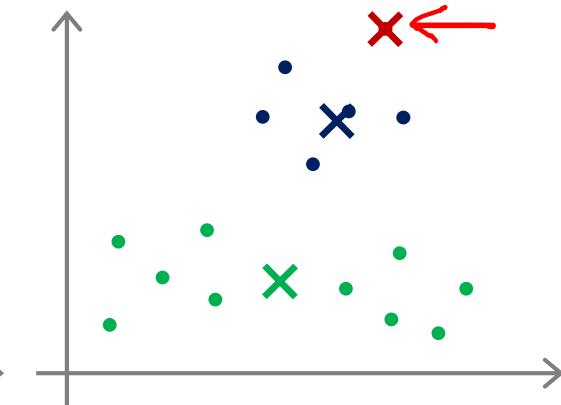
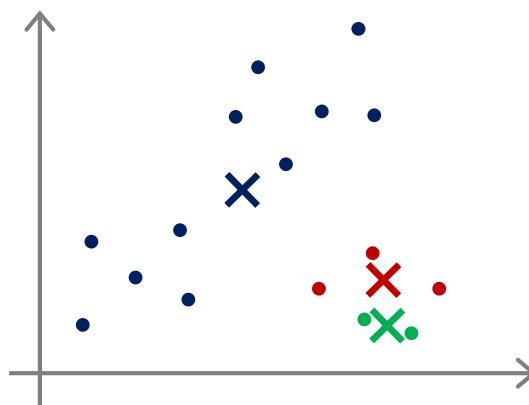
Local optima



$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k)$$



Depending on the initialization of cluster centroids K-means can produce different results



Random initialization

For i = 1 to 100 {

 Randomly initialize K-means.

 Run K-means. Get (c_1, c_2, \dots, c_K) .

 Compute cost function (distortion)

}

$E(c_1, c_2, \dots, c_K)$

Pick clustering that gave lowest cost $E(c_1, c_2, \dots, c_K)$

Comments on the *K-Means* Method

- Strength: Efficient: $O(tkn)$, where n is # objects, k is # clusters, and t is # iterations. Normally, $k, t \ll n$.
 - Comparing: PAM: $O(k(n-k)^2)$, CLARA: $O(ks^2 + k(n-k))$
- Comment: Often terminates at a *local optimal*.
- Weakness
 - Applicable only to objects in a continuous n-dimensional space
 - Using the k-modes method for categorical data
 - In comparison, k-medoids can be applied to a wide range of data
 - Need to specify k , the *number* of clusters, in advance (there are ways to automatically determine the best k (see Hastie et al., 2009))
 - Sensitive to ***noisy data and outliers***
 - Not suitable to discover clusters with *non-convex shapes*

k -means cannot represent density-based clusters

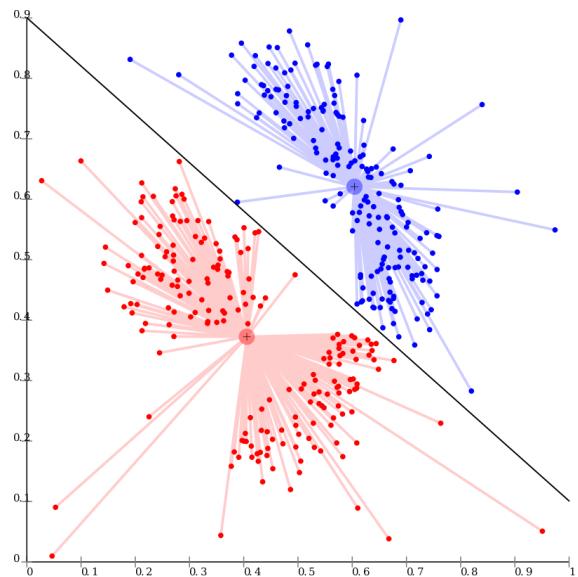


Fig: Convex Shaped

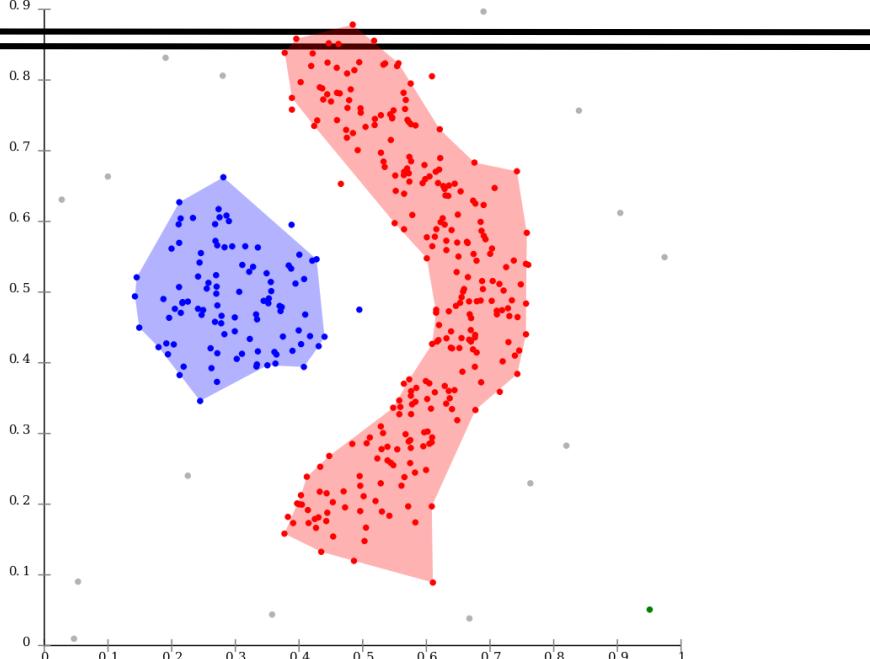
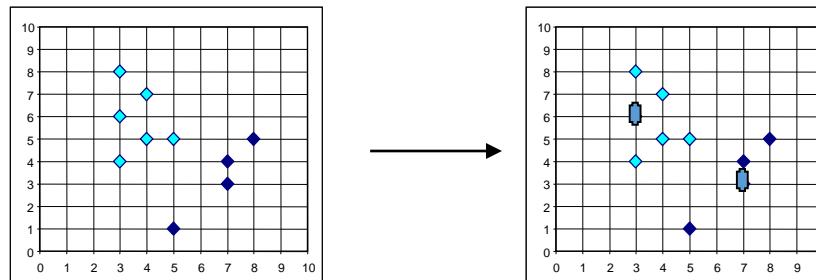


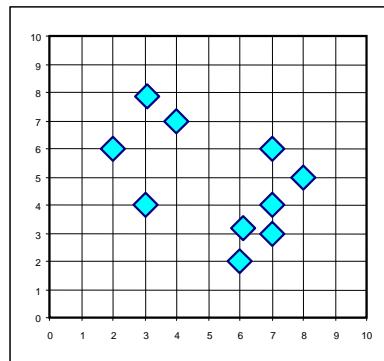
Fig: Non-Convex Shaped

What Is the Problem of the K-Means Method?

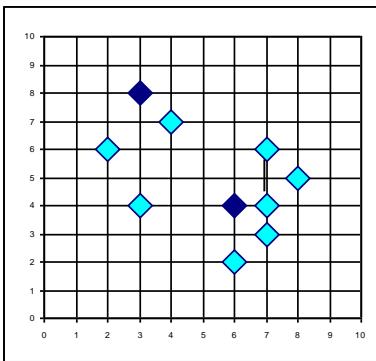
- The k-means algorithm is sensitive to outliers !
 - Since an object with an extremely large value may substantially distort the distribution of the data
- K-Medoids: Instead of taking the **mean** value of the object in a cluster as a reference point, **medoids** can be used, which is the **most centrally located** object in a cluster



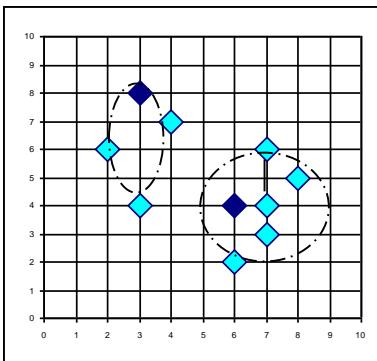
PAM: A Typical K-Medoids Algorithm



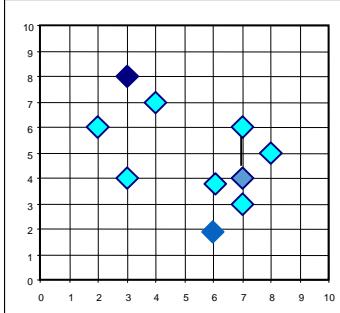
Arbitrarily choose k object as initial medoids



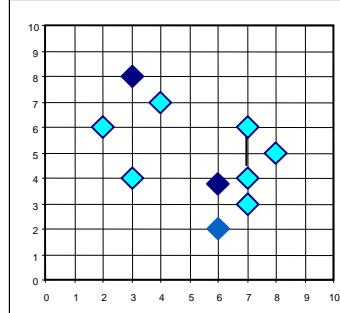
Assign each remaining object to nearest medoids



Randomly select a nonmedoid object



Compute total cost of swapping



Do loop
Until no change

Swapping O and O_{random}
If quality is improved.

The *K-Medoids* Clustering Method

Algorithm: *k-medoids*. PAM, a *k*-medoids algorithm for partitioning based on medoid or central objects.

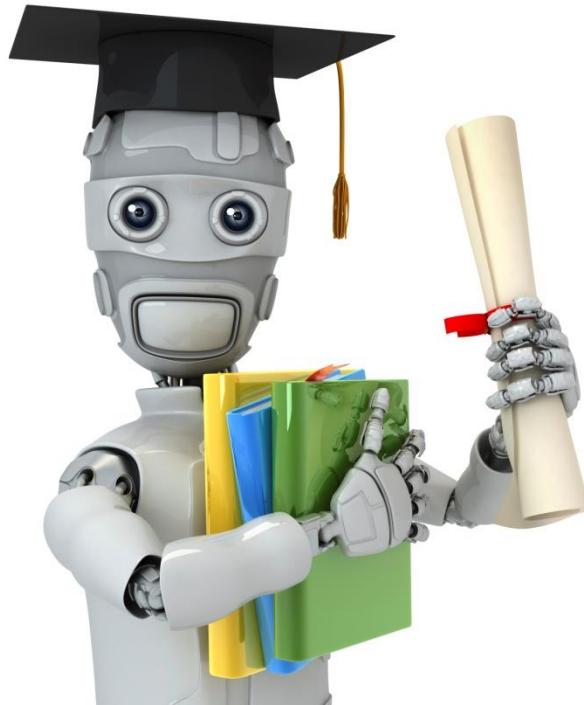
Input:

- k : the number of clusters,
- D : a data set containing n objects.

Output: A set of k clusters.

Method:

- (1) arbitrarily choose k objects in D as the initial representative objects or seeds;
- (2) **repeat**
- (3) assign each remaining object to the cluster with the nearest representative object;
- (4) randomly select a nonrepresentative object, o_{random} ;
- (5) compute the total cost, S , of swapping representative object, o_j , with o_{random} ;
- (6) **if** $S < 0$ **then** swap o_j with o_{random} to form the new set of k representative objects;
- (7) **until** no change;

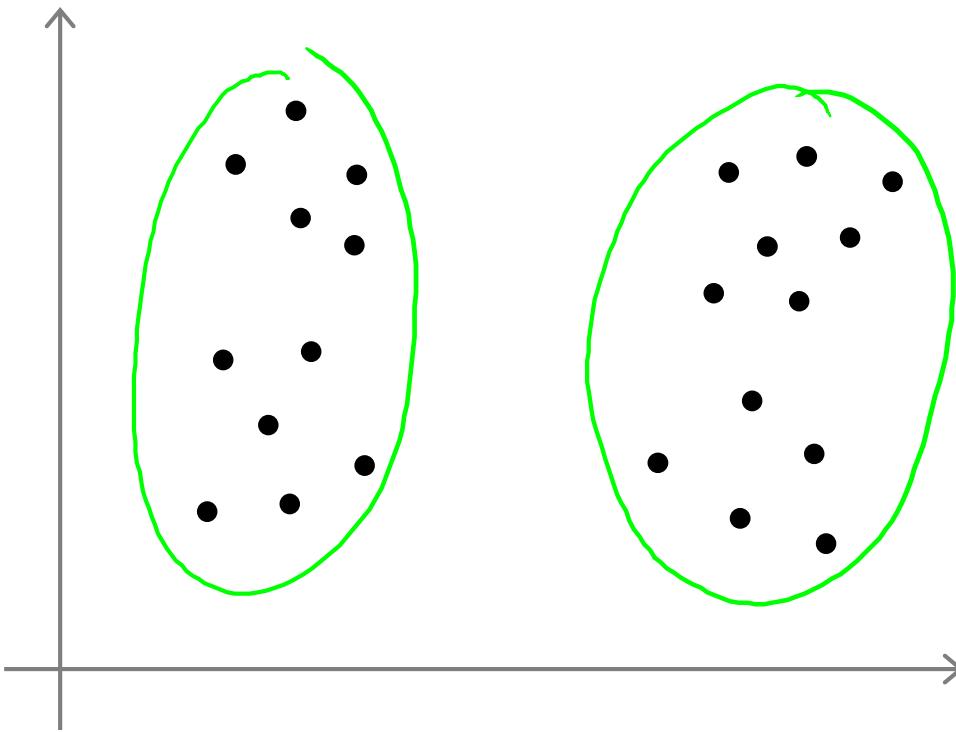


Machine Learning

Clustering

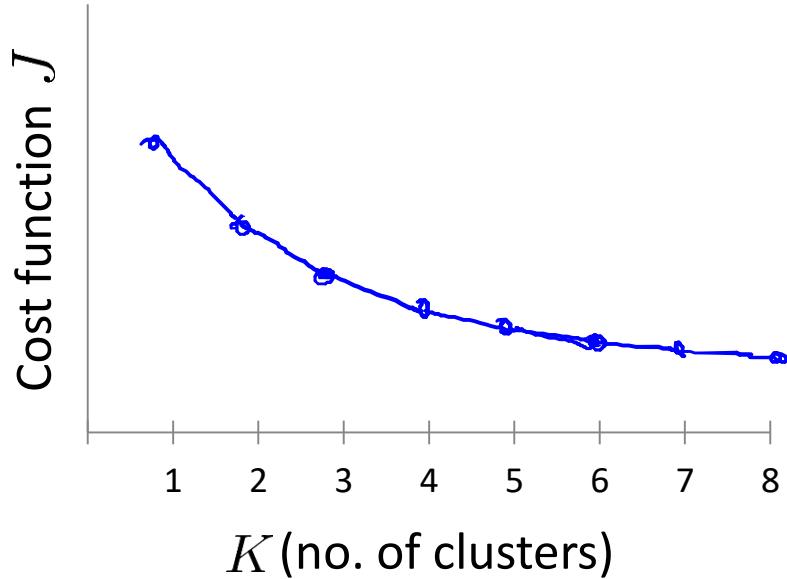
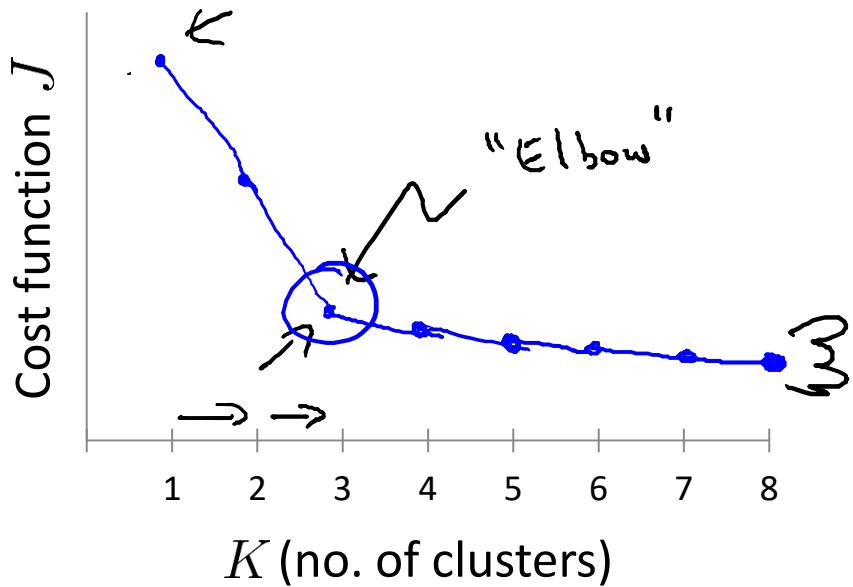
Choosing the number of clusters

What is the right value of K?



Choosing the value of K

Elbow method:

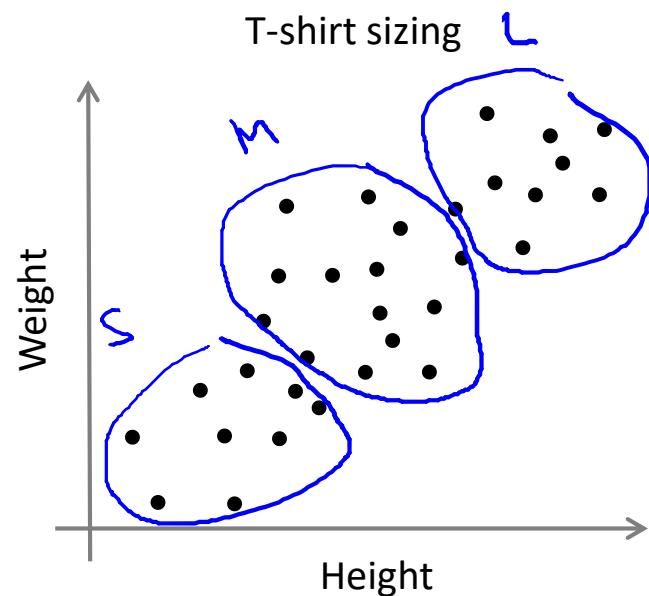


Choosing the value of K

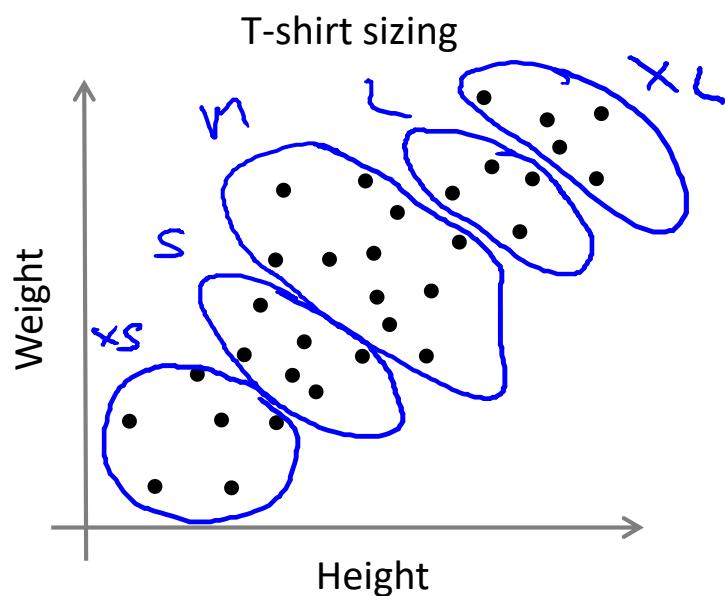
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

$$K=3 \quad S, M, L$$

E.g.



$$K=5 \quad XS, S, M, L, XL$$





CSE 4621

Machine Learning

Lecture 15

Md. Hasanul Kabir, PhD.

Professor, CSE Department

Islamic University of Technology (IUT)



Support Vector Machine & Its Applications

A portion (1/3) of
the slides are taken from
Prof. Andrew Moore's
SVM tutorial at
<http://www.cs.cmu.edu/~awm/tutorials>

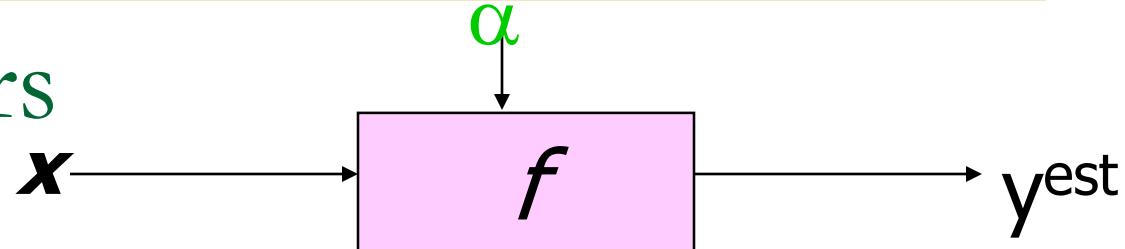
Mingyue Tan

The University of British Columbia
Nov 26, 2004

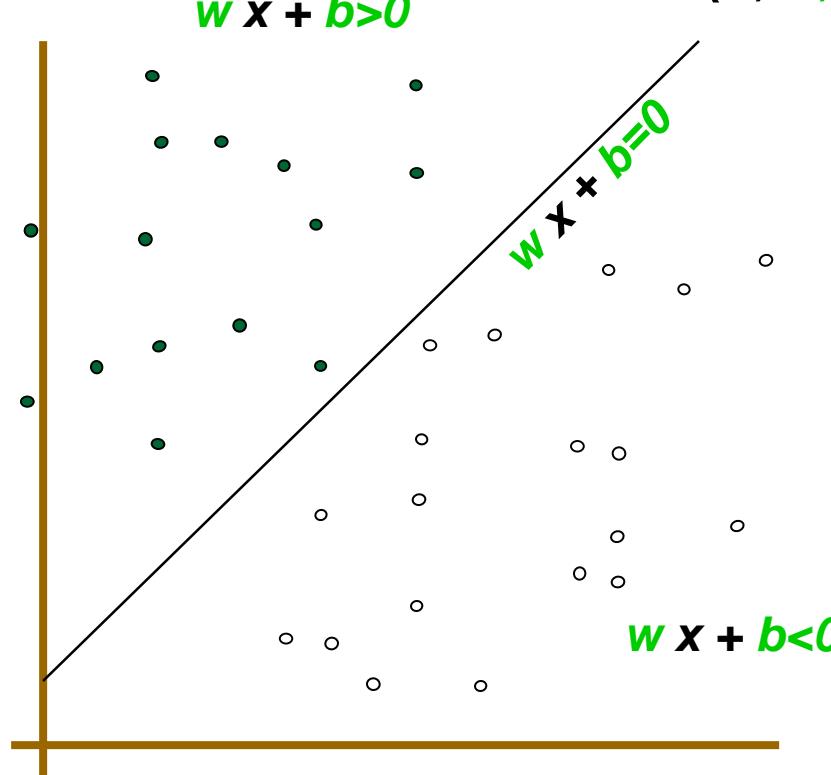
Overview

- Intro. to Support Vector Machines (SVM)
- Properties of SVM
- Applications
 - Gene Expression Data Classification
 - Text Categorization *if time permits*
- Discussion

Linear Classifiers

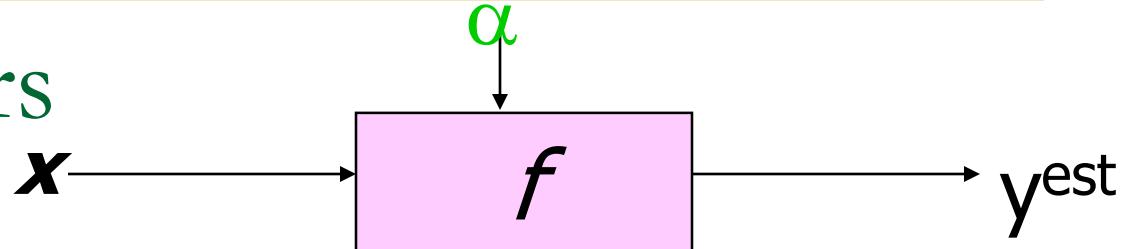


- denotes +1
- denotes -1



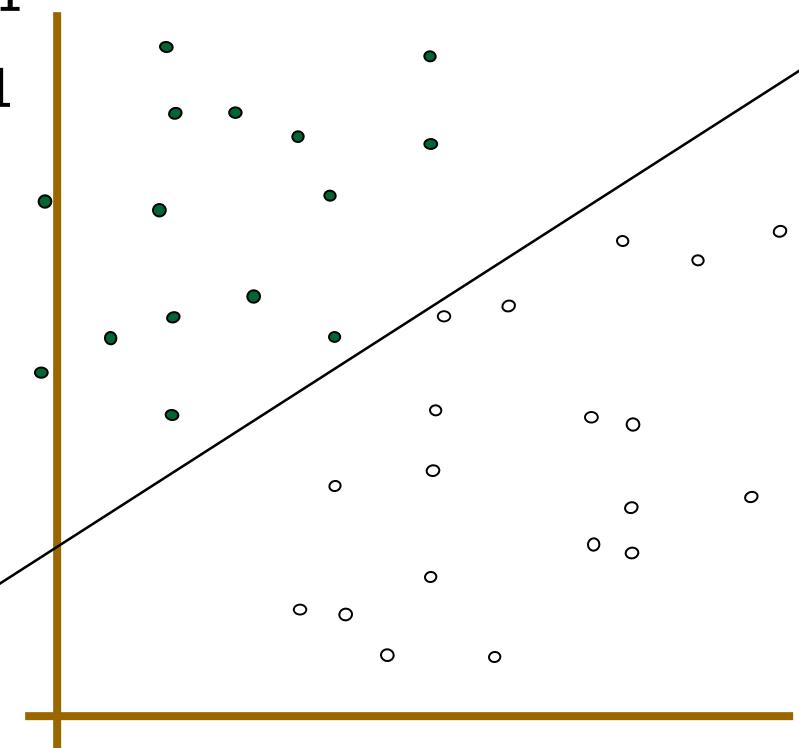
How would you
classify this data?

Linear Classifiers



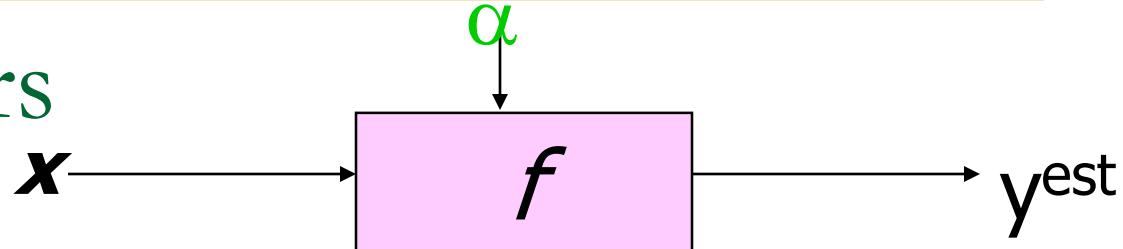
- denotes +1
- denotes -1

$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

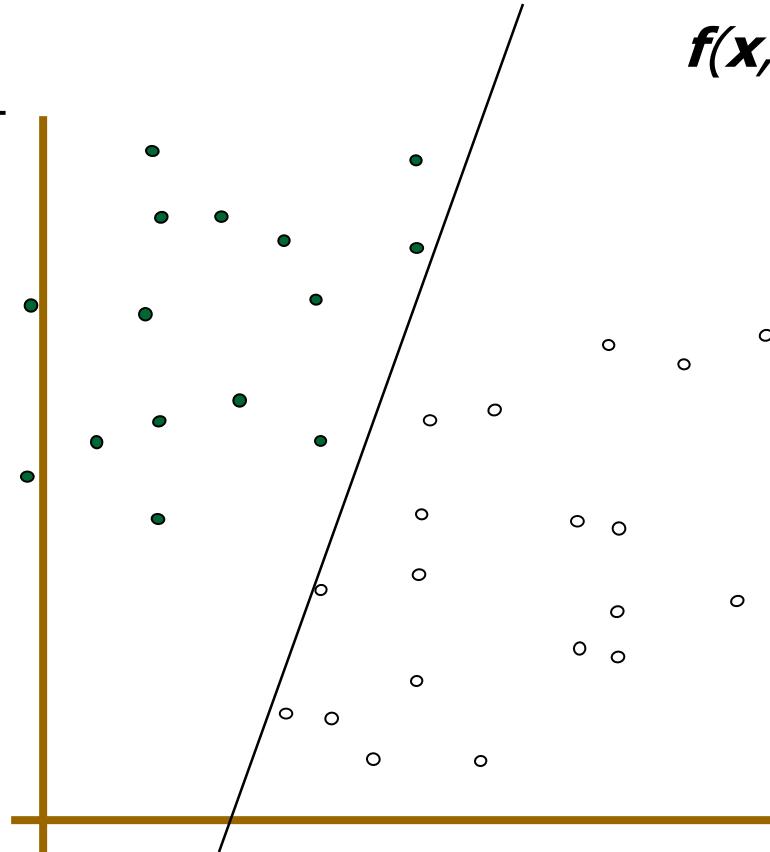


How would you
classify this data?

Linear Classifiers



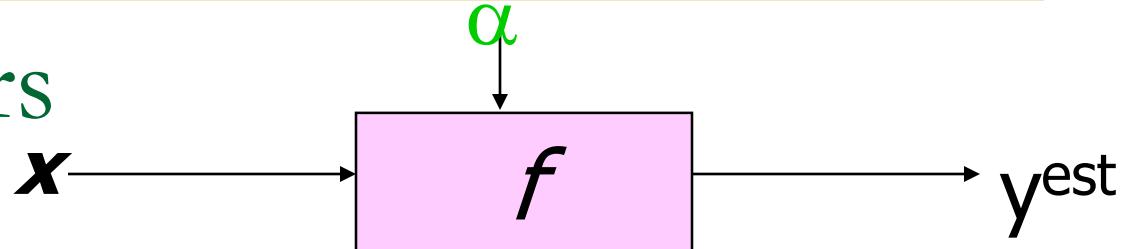
- denotes +1
- denotes -1



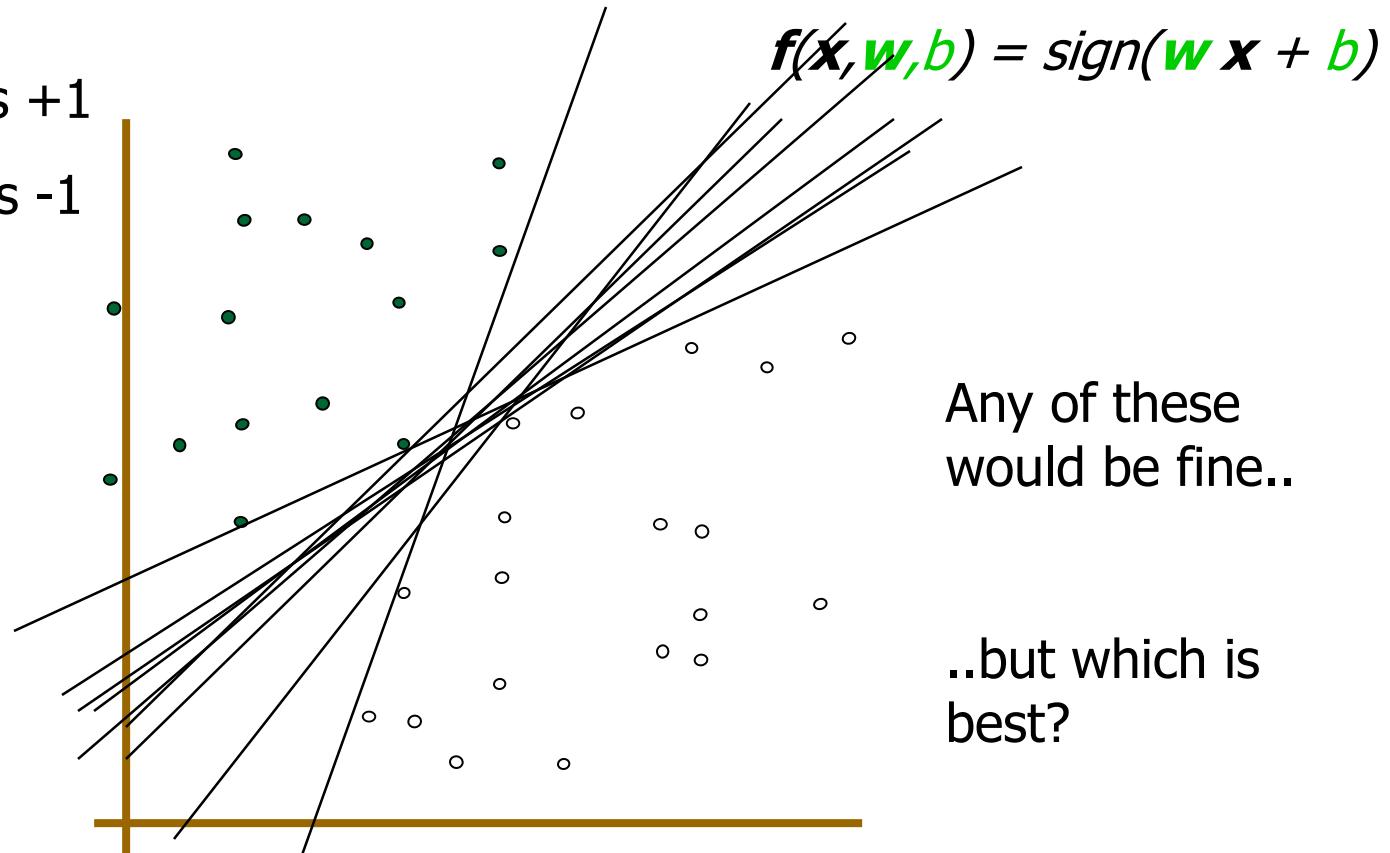
$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

How would you
classify this data?

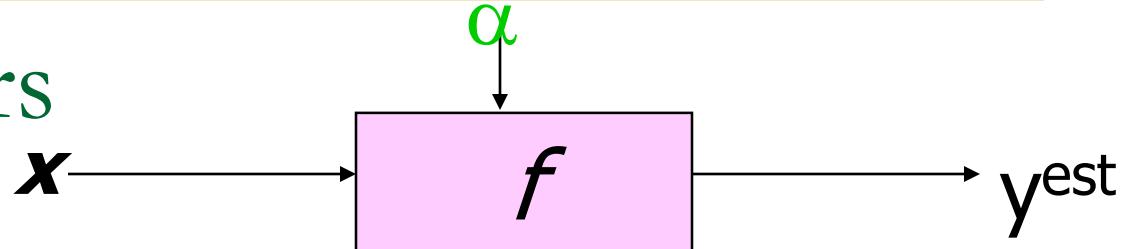
Linear Classifiers



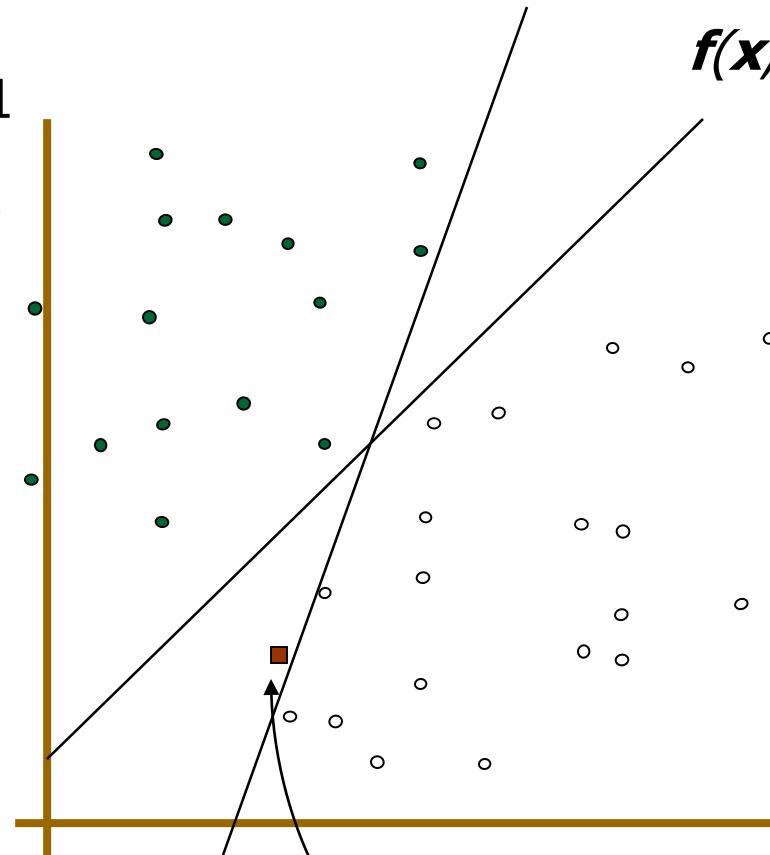
- denotes +1
- denotes -1



Linear Classifiers



- denotes +1
- denotes -1

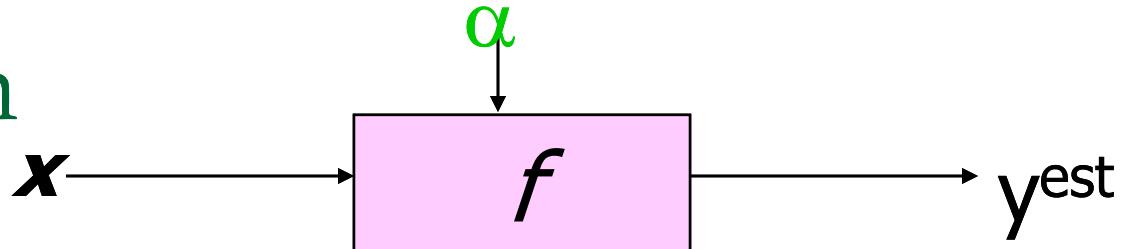


$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

How would you
classify this data?

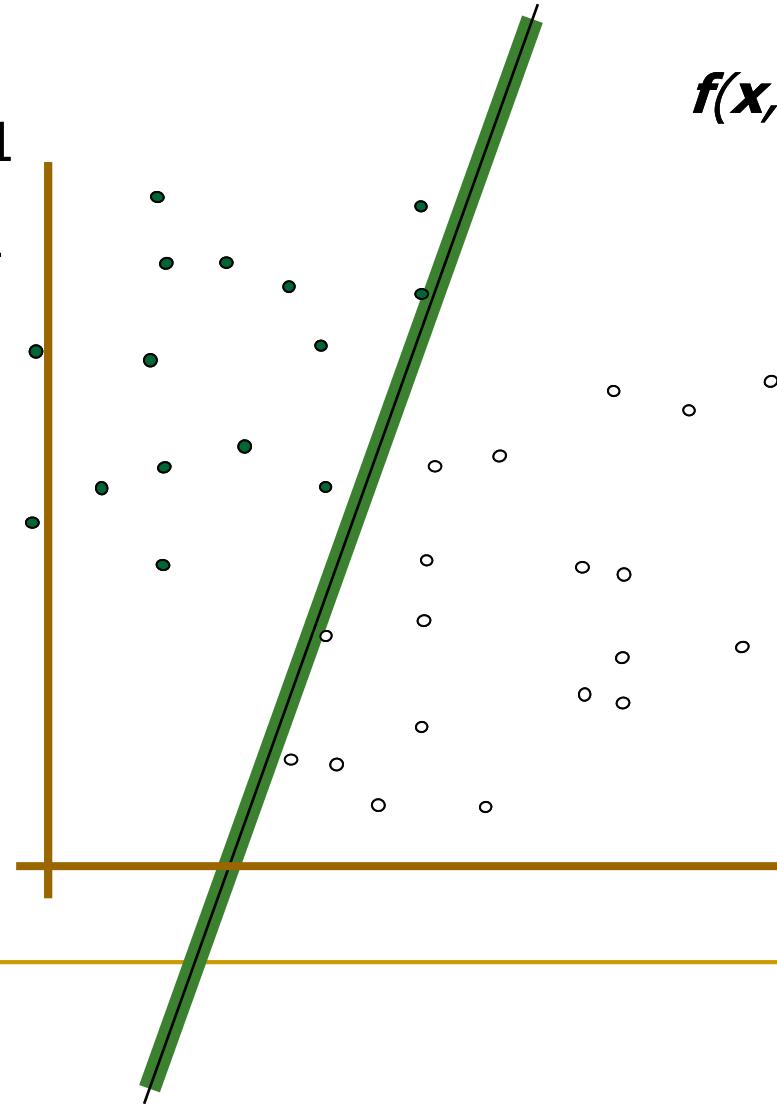
Misclassified
to +1 class

Classifier Margin



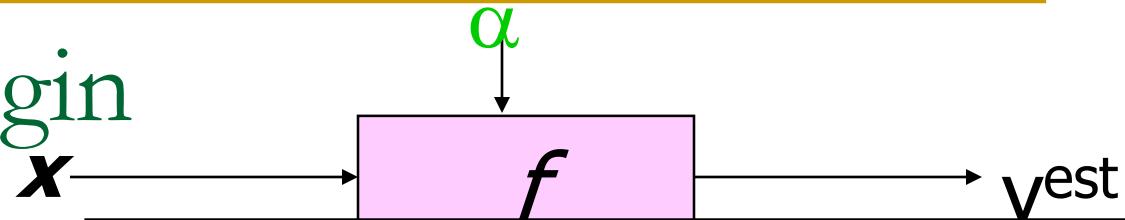
$$f(x, w, b) = \text{sign}(w x + b)$$

- denotes +1
- denotes -1



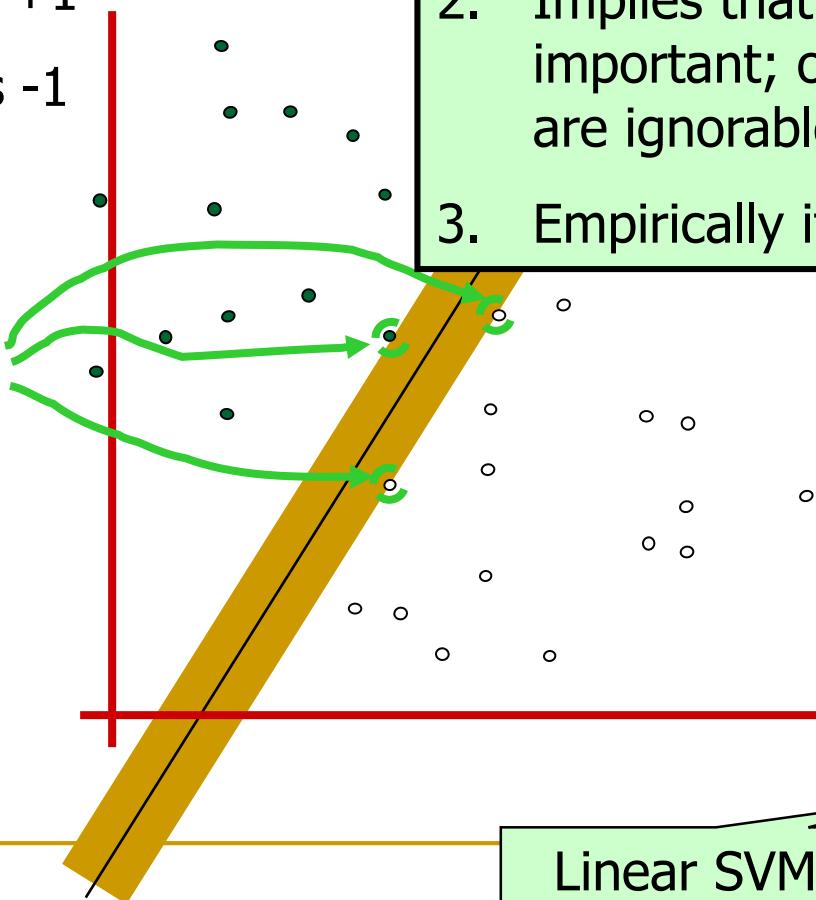
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



- denotes +1
- denotes -1

Support Vectors
are those
datapoints that
the margin
pushes up
against

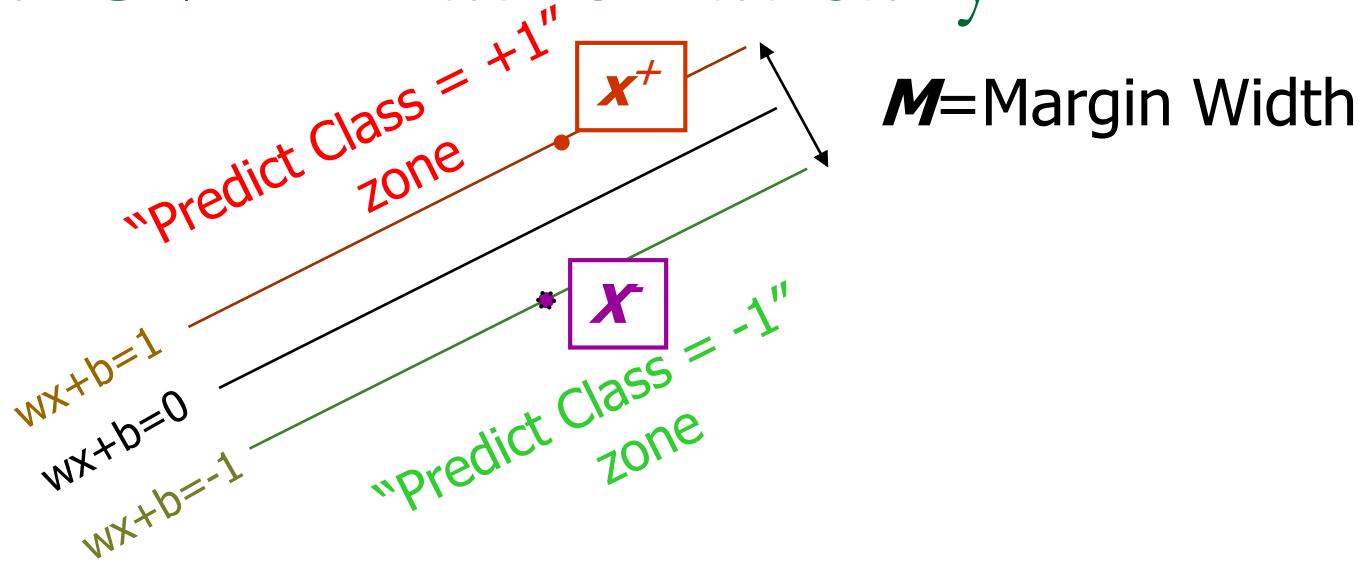


1. Maximizing the margin is good according to intuition and PAC theory
2. Implies that only support vectors are important; other training examples are ignorable.
3. Empirically it works very very well.

linear classifier
with the, um,
maximum margin.

This is the
simplest kind of
SVM (Called an
LSVM)

Linear SVM Mathematically



What we know:

- $w \cdot x^+ + b = +1$
- $w \cdot x^- + b = -1$
- $w \cdot (x^+ - x^-) = 2$

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Linear SVM Mathematically

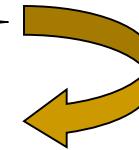
- Goal: 1) Correctly classify all training data

$$\begin{aligned} wx_i + b &\geq 1 & \text{if } y_i = +1 \\ wx_i + b &\leq -1 & \text{if } y_i = -1 \end{aligned}$$

for all i

- 2) Maximize the Margin
same as minimize

$$M = \frac{1}{2} w^t w$$



- We can formulate a Quadratic Optimization Problem and solve for w and b

- Minimize $\Phi(w) = \frac{1}{2} w^t w$

- subject to $y_i (wx_i + b) \geq 1 \quad \forall i$

Solving the Optimization Problem

Find \mathbf{w} and b such that

$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$ is minimized;

and for all $\{(\mathbf{x}_i, y_i)\}$: $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

$$(1) \quad \sum \alpha_i y_i = 0$$

$$(2) \quad \alpha_i \geq 0 \text{ for all } \alpha_i$$

The Optimization Problem Solution

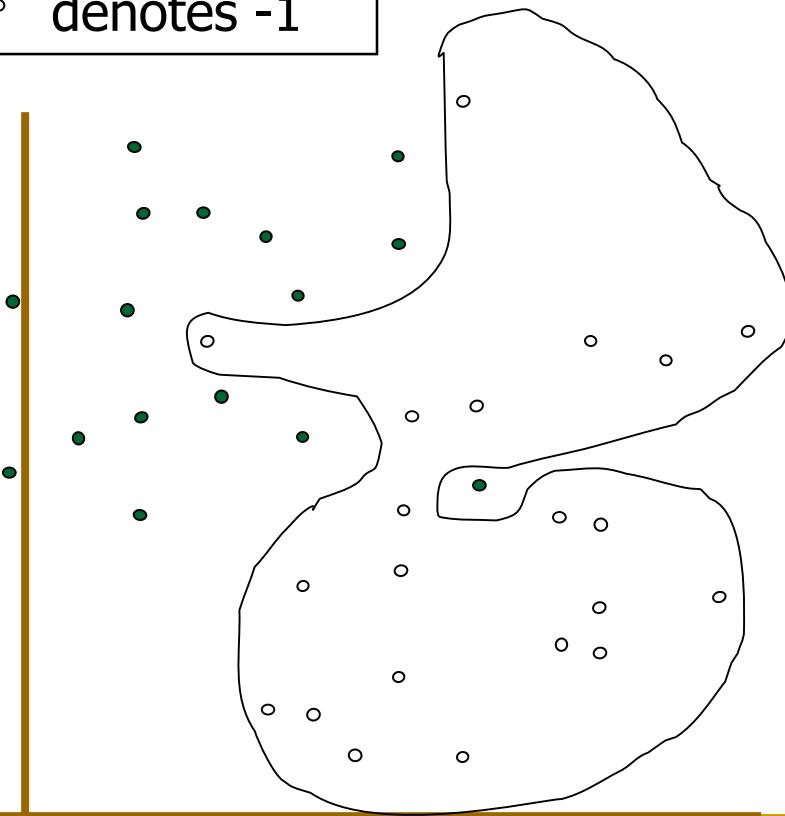
- The solution has the form:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i \quad b = y_k - \mathbf{w}^T \mathbf{x}_k \text{ for any } \mathbf{x}_k \text{ such that } \alpha_k \neq 0$$

- Each non-zero α_i indicates that corresponding \mathbf{x}_i is a support vector.
- Then the classifying function will have the form:
$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$
- Notice that it relies on an *inner product* between the test point \mathbf{x} and the support vectors \mathbf{x}_i – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^T \mathbf{x}_j$ between all pairs of training points.

Dataset with noise

- denotes +1
- denotes -1

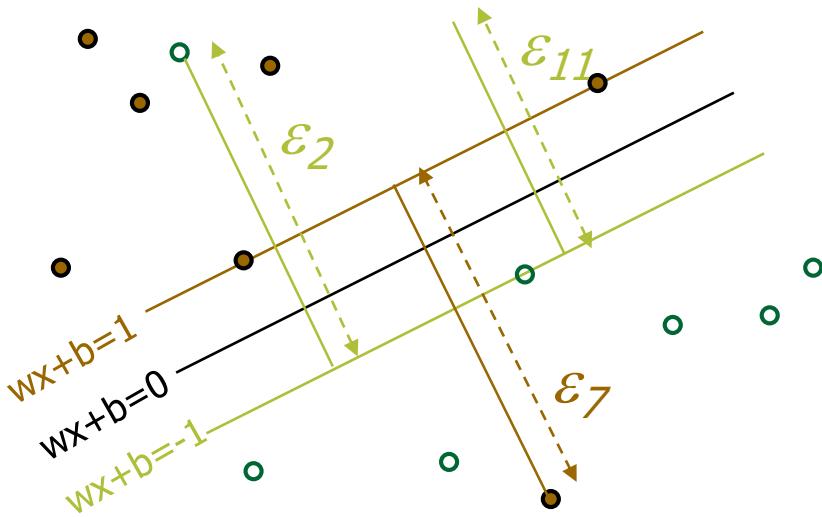


- **Hard Margin:** So far we require all data points be classified correctly
 - No training error
- **What if the training set is noisy?**
 - **Solution 1:** use very powerful kernels

OVERFITTING!

Soft Margin Classification

Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples.



What should our quadratic optimization criterion be?

Minimize

$$\frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_{k=1}^R \varepsilon_k$$

Hard Margin v.s. Soft Margin

- **The old formulation:**

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} \text{ is minimized and for all } \{(\mathbf{x}_i, y_i)\}$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

- **The new formulation incorporating slack variables:**

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum \xi_i \text{ is minimized and for all } \{(\mathbf{x}_i, y_i)\}$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \text{ for all } i$$

- **Parameter C can be viewed as a way to control overfitting.**

Linear SVMs: Overview

- The classifier is a *separating hyperplane*.
- Most “important” training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points x_i are support vectors with non-zero Lagrangian multipliers α_i .
- Both in the dual formulation of the problem and in the solution training points appear only inside dot products:

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

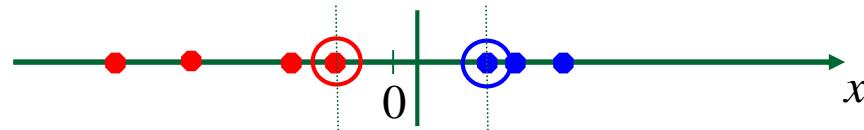
$$(1) \sum \alpha_i y_i = 0$$

$$(2) 0 \leq \alpha_i \leq C \text{ for all } \alpha_i$$

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

Non-linear SVMs

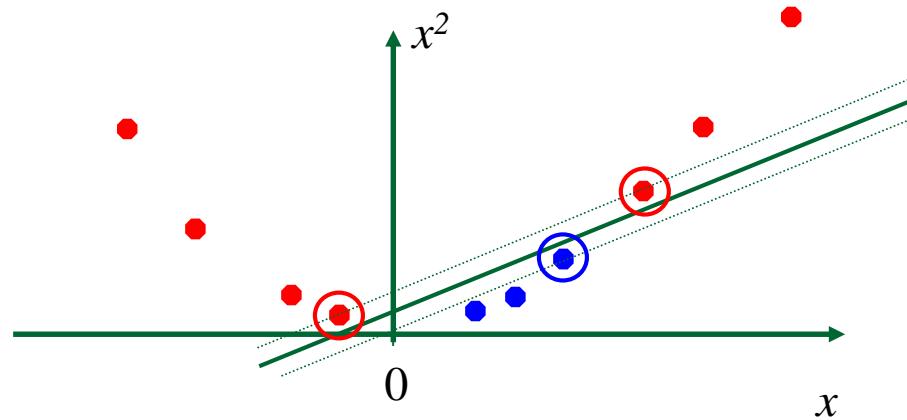
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

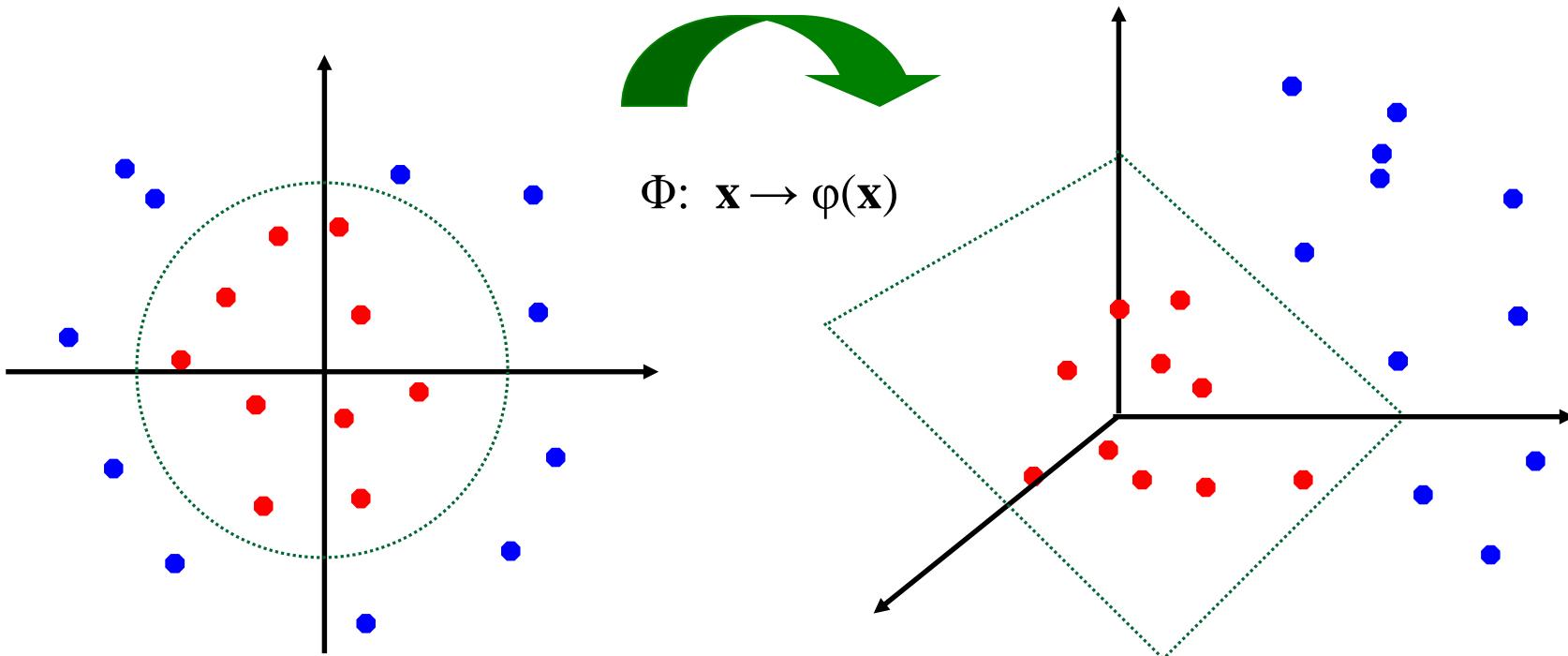


- How about... mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



The “Kernel Trick”

- The linear classifier relies on dot product between vectors $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- If every data point is mapped into high-dimensional space via some transformation $\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})$, the dot product becomes:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.
- Example:

2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$; let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$,

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T \mathbf{x}_j)^2, \\ &= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} \\ &= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T [1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}] \\ &= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \quad \text{where } \phi(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2} x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2] \end{aligned}$$

What Functions are Kernels?

- For some functions $K(\mathbf{x}_i, \mathbf{x}_j)$ checking that

$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ can be cumbersome.

- Mercer's theorem:

Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

$K =$

$K(\mathbf{x}_1, \mathbf{x}_1)$	$K(\mathbf{x}_1, \mathbf{x}_2)$	$K(\mathbf{x}_1, \mathbf{x}_3)$...	$K(\mathbf{x}_1, \mathbf{x}_N)$
$K(\mathbf{x}_2, \mathbf{x}_1)$	$K(\mathbf{x}_2, \mathbf{x}_2)$	$K(\mathbf{x}_2, \mathbf{x}_3)$		$K(\mathbf{x}_2, \mathbf{x}_N)$
...
$K(\mathbf{x}_N, \mathbf{x}_1)$	$K(\mathbf{x}_N, \mathbf{x}_2)$	$K(\mathbf{x}_N, \mathbf{x}_3)$...	$K(\mathbf{x}_N, \mathbf{x}_N)$

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- Polynomial of power p : $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$

- Gaussian (radial-basis function network):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$

Non-linear SVMs Mathematically

- Dual problem formulation:

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j K(x_i, x_j)$ is maximized and

(1) $\sum \alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

- The solution is:

$$f(\mathbf{x}) = \sum \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- Optimization techniques for finding α_i 's remain the same!

Nonlinear SVM - Overview

- SVM locates a separating hyperplane in the feature space and classify points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the dot product in the feature space.

Properties of SVM

- **Flexibility in choosing a similarity function**
- **Sparseness of solution when dealing with large data sets**
 - only support vectors are used to specify the separating hyperplane
- **Ability to handle large feature spaces**
 - complexity does not depend on the dimensionality of the feature space
- **Overfitting can be controlled by soft margin approach**
- **Nice math property:** a simple convex optimization problem which is guaranteed to converge to a single global solution
- **Feature Selection**

SVM Applications

- **SVM has been used successfully in many real-world problems**
 - text (and hypertext) categorization
 - image classification
 - bioinformatics (Protein classification, Cancer classification)
 - hand-written character recognition

Application 1: Cancer Classification

- High Dimensional

- $p > 1000$; $n < 100$

- Imbalanced

- less positive samples

$$K[x, x] = k(x, x) + \lambda \frac{n^+}{N}$$

- Many irrelevant features

- Noisy

SVM is sensitive to noisy (mis-labeled) data ☹

Patients	Genes			
	g-1	g-2	g-p
P-1				
p-2				
.....				
p-n				

FEATURE SELECTION

In the linear case,
 w_i^2 gives the ranking of dim i

Weakness of SVM

- **It is sensitive to noise**

- A relatively small number of mislabeled examples can dramatically decrease the performance

- **It only considers two classes**

- how to do multi-class classification with SVM?

- Answer:

- 1) with output arity m, learn m SVM's

- SVM 1 learns "Output==1" vs "Output != 1"
 - SVM 2 learns "Output==2" vs "Output != 2"
 - :
 - SVM m learns "Output==m" vs "Output != m"

- 2) To predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

Application 2: Text Categorization

- Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content.
 - email filtering, web searching, sorting documents by topic, etc..
- A document can be assigned to more than one category, so this can be viewed as a series of binary classification problems, one for each category

Representation of Text

IR's vector space model (aka bag-of-words representation)

- A doc is represented by a vector indexed by a pre-fixed set or dictionary of terms
- Values of an entry can be binary or weights

$$\phi_i(x) = \frac{\text{tf}_i \log (\text{idf}_i)}{\kappa},$$

- Normalization, stop words, word stems
- Doc $x \Rightarrow \Phi(x)$

Text Categorization using SVM

- The distance between two documents is $\varphi(x) \cdot \varphi(z)$
- $K(x,z) = \langle \varphi(x) \cdot \varphi(z) \rangle$ is a valid kernel, SVM can be used with $K(x,z)$ for discrimination.
- Why SVM?
 - High dimensional input space
 - Few irrelevant features (dense concept)
 - Sparse document vectors (sparse instances)
 - Text categorization problems are linearly separable

Some Issues

- **Choice of kernel**
 - Gaussian or polynomial kernel is default
 - if ineffective, more elaborate kernels are needed
 - domain experts can give assistance in formulating appropriate similarity measures
- **Choice of kernel parameters**
 - e.g. σ in Gaussian kernel
 - σ is the distance between closest points with different classifications
 - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.
- **Optimization criterion** – Hard margin v.s. Soft margin
 - a lengthy series of experiments in which various parameters are tested

Additional Resources

- **An excellent tutorial on VC-dimension and Support Vector Machines:**
C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):955-974, 1998.
- **The VC/SRM/SVM Bible:**
Statistical Learning Theory by Vladimir Vapnik, Wiley-Interscience; 1998

<http://www.kernel-machines.org/>

Reference

- **Support Vector Machine Classification of Microarray Gene Expression Data**, Michael P. S. Brown William Noble Grundy, David Lin, Nello Cristianini, Charles Sugnet, Manuel Ares, Jr., David Haussler
- www.cs.utexas.edu/users/mooney/cs391L/svm.ppt
- **Text categorization with Support Vector Machines:
learning with many relevant features**
T. Joachims, ECML - 98