



# Machine Learning

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**Session 1  
Date – 19<sup>th</sup> October 2019  
Time – 9 am to 11 am**

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Sugato Ghosal, Prof. S.K. H. Islam from BITS Pilani and many others who made their course materials freely available online.

# Session Content

---

- Objective of course
- Evaluation Plan
- What is Machine Learning?
- Application areas of Machine Learning
- Why Machine Learning is important?
- Design a Learning System
- Issues in Machine Learning

# Objective of course

---

- Introduction to the basic concepts and techniques of Machine Learning
- Gain experience of doing independent study and research in the field of Machine Learning
- Develop skills of using recent machine learning software tools to evaluate learning algorithms and model selection for solving practical problems

# What We'll Cover in this Course

---

- **Supervised learning**
  - Regression
  - Logistic regression
  - Bayesian learning
  - Decision Tree (Random Forest)
  - Neural networks
  - Support vector machines
- **Unsupervised learning**
  - Clustering
  - Dimensionality reduction
- **Applications**
- **Ensemble Learning**

# Books

---

## Text books and Reference book(s)

T1	Tom M. Mitchell: <b>Machine Learning</b> , The McGraw-Hill Companies, Inc. International Edition 1997
T2	Christopher M. Bishop: <b>Pattern Recognition &amp; Machine Learning</b> , Springer, 2006

R1	<b>C.J.C. BURGES: A Tutorial on Support Vector Machines for Pattern Recognition</b> , Kluwer Academic Publishers, Boston.
----	---

# Evaluation Plan

Name	Type	Duration	Weight
Quiz-I (roughly after 6 sessions)	Online	2-Dec-2019 (Before midsem)	5%
Assignment-I (roughly after 3 sessions)	Take Home	15-Nov-2019 (Before midsem)	13%
Assignment-II	Take Home	31-Jan-2019 (after midsem)	12%
Mid-Semester Test (after 7 sessions)	Closed Book	1.5 Hrs	30%
Comprehensive Exam	Open Book	2.5 Hrs	40%

Please note there will be no change in submission dates for quiz and assignment 7

# Lab Plan

Lab No.	Lab Objective
1	Linear Regression and Gradient Descent
2	Logistic Regression classifier
3	Single layer Back propagation NN
4	K-nearest Neighbour
5	SVM
6	K-means clustering

- **Labs not graded**
- **Webinars will be conducted for lab sessions**
- **Labs will be conducted in Python**

# Machine Learning

---

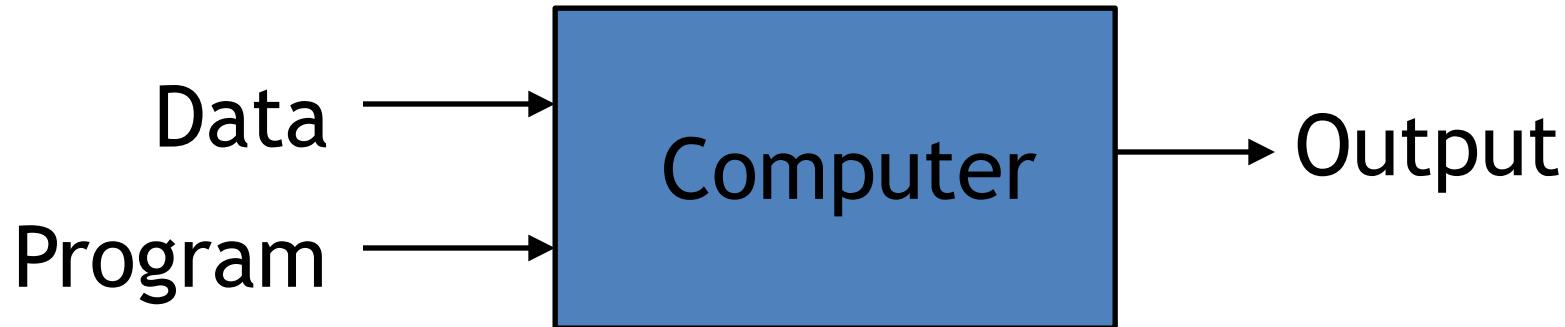
- **Machine learning** is a scientific discipline that explores the construction and study of algorithms that can learn from data.
- Such algorithms operate by building a model based on inputs and using that to make predictions or decisions, rather than following only explicitly programmed instructions.

# A Few Quotes

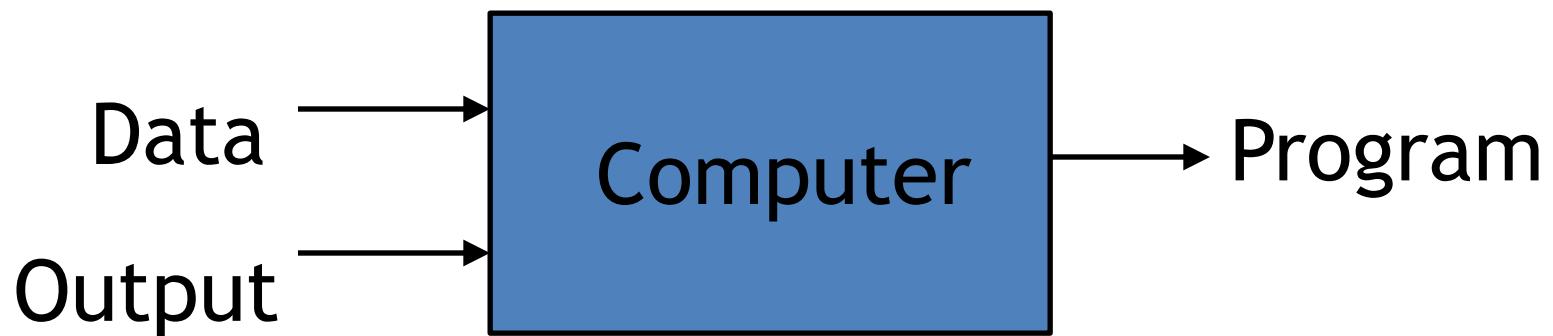
---

- “A breakthrough in machine learning would be worth ten Microsofts” (Bill Gates, Chairman, Microsoft)
- “Machine learning is the next Internet”  
(Tony Tether, Director, DARPA)
- “Web rankings today are mostly a matter of machine learning” (Prabhakar Raghavan, Dir. Research, Yahoo)
- “Machine learning is going to result in a real revolution” (Greg Papadopoulos, CTO, Sun)
- “Machine learning is today’s discontinuity”  
(Jerry Yang, CEO, Yahoo)

# Traditional Programming



# Machine Learning



# What is Machine Learning?

---

Definition by Tom Mitchell (1998):

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." Example: playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

P = the probability that the program will win the next game.

# What is Machine Learning?

---

- To have a learning problem, we must identify
  - The class of tasks
  - The measure of performance to be improved
  - Source of experience

---

# Example of Learning Problems

# A Checker Learning Problem

---

- **Task T:** Playing Checkers
- **Performance Measure P:** Percent of games won against opponents
- **Training Experience E:** To be selected ==> Games Played 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 travelled before an error (as judged by human)
- **Training experience E:** a sequence of images and steering commands recorded while observing a human driver

# Why is Machine Learning Important?

---

- Some tasks cannot be defined well, except by examples.
- Relationships and correlations can be hidden within large amounts of data. Machine Learning may be able to find these relationships.
- Human designers often produce machines that do not work as well as desired in the environments in which they are used.

# Why is Machine Learning Important ?

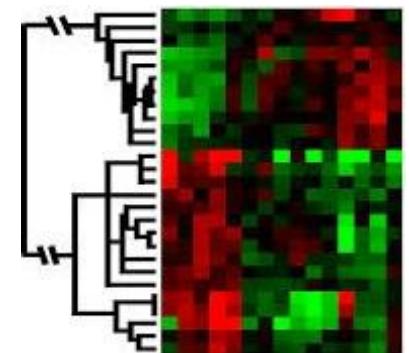
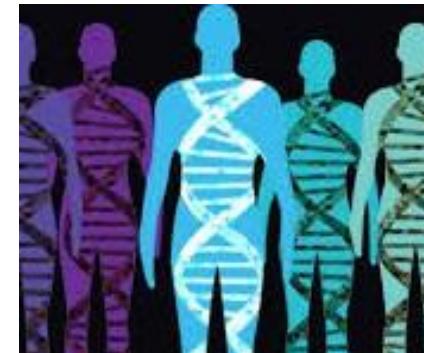
---

- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans (e.g., medical diagnostic).
- New knowledge about tasks is constantly being discovered by humans. It may be difficult to continuously re-design systems “by hand”.

# When Do We Use Machine Learning?

ML is used when:

- Human expertise does not exist (navigating on Mars)
- Humans can't explain their expertise (speech recognition)
- Models must be customized (personalized medicine)
- Models are based on huge amounts of data (genomics)



Learning isn't always useful:

- There is no need to “learn” to calculate payroll

# Applications of Machine Learning

---

- Learning to recognize spoken words (Lee, 1989; Waibel, 1989).
- Detect fraudulent use of credit cards or Learning to drive an autonomous vehicle (Pomerleau, 1989).
- Learning to classify new astronomical structures (Fayyad et al., 1995).

# Applications of Machine Learning

---

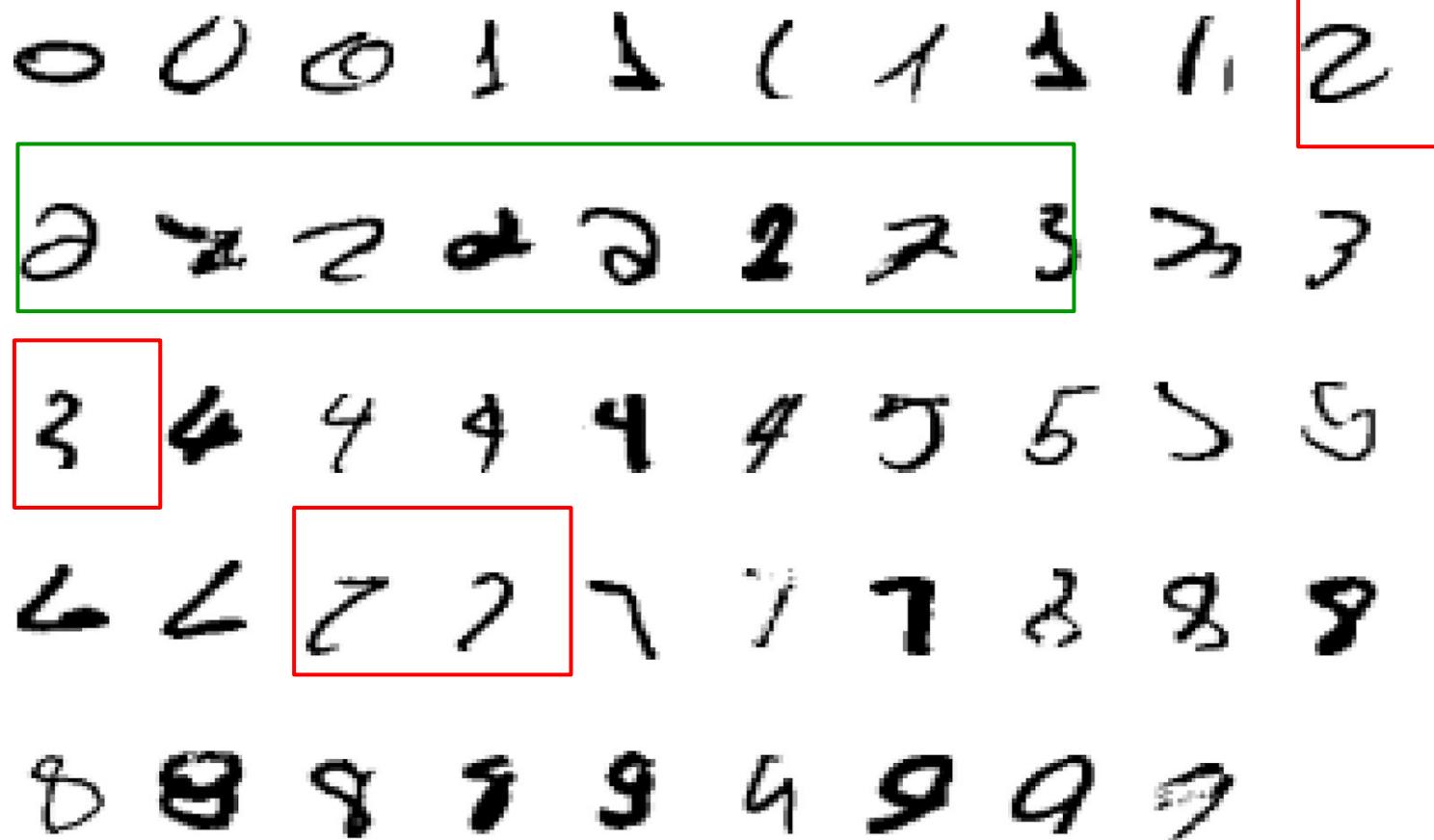
- Learning to play world-class backgammon (Tesauro 1992, 1995).
- Predict recovery rates of pneumonia patients (Copper et al. 1997)

# Application Types: Classification

- Medical diagnosis
- Credit card applications or transactions
- Fraud detection in e-commerce
- Worm detection in network packets
- Spam filtering in email
- Recommended articles in a newspaper
- Recommended books, movies, music, or jokes
- Financial investments
- DNA sequences
- Spoken words
- Handwritten letters
- Astronomical images

# Pattern recognition

It is very hard to say what makes a 2



# Application Domains

---

- Web search
- Computational biology
- Finance
- E-commerce
- Space exploration
- Robotics
- Information extraction
- Social networks
- Language Processing

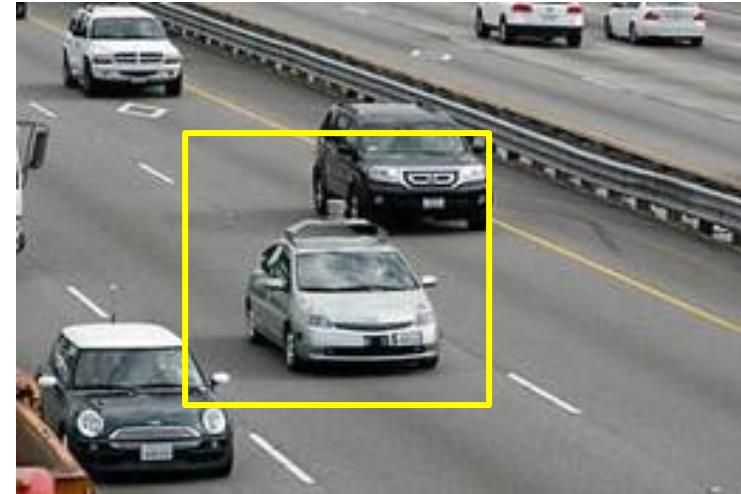
Many more emerging...

---

# State of the Art Applications of Machine Learning

In this course, you will learn principles that will help you understand and build some of these applications.

# Autonomous Cars

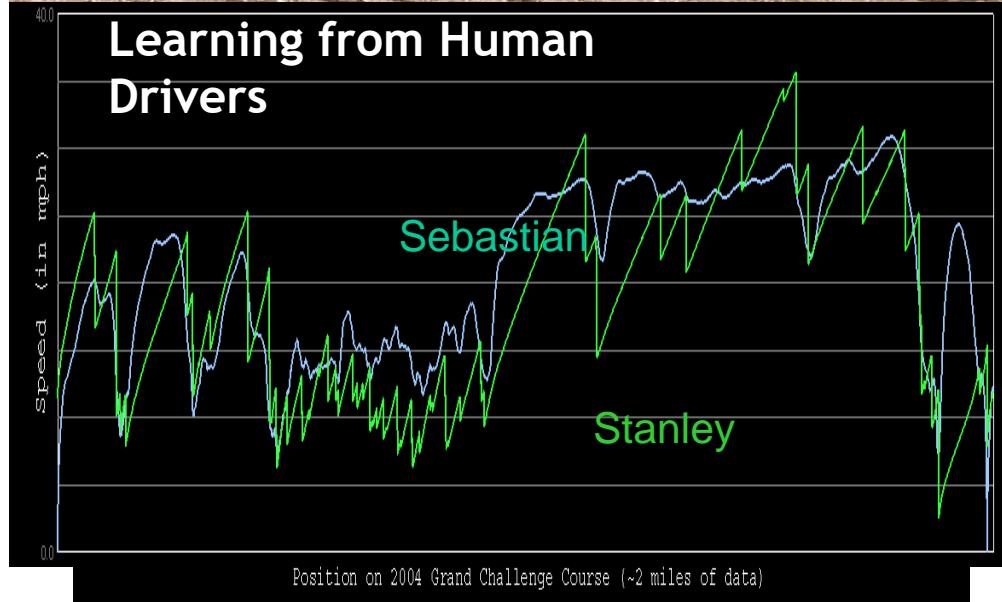
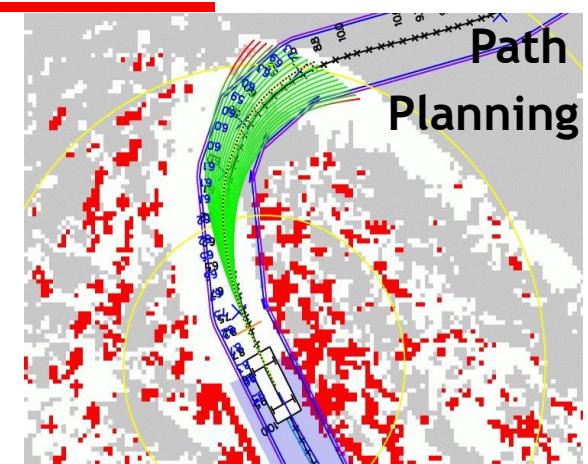


- Nevada made it legal for autonomous cars to drive on roads in June 2011
- As of 2013, four states (Nevada, Florida, California, and Michigan) have legalized autonomous cars

**UPenn's Autonomous Car →**



# Autonomous Car Technology



# Deep Learning in the Headlines

BUSINESS NEWS

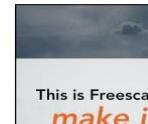
## Is Google Cornering the Market on Deep Learning?

A cutting-edge corner of science is being wooed by Silicon Valley, to the dismay of some academics.

By Antonio Regalado on January 29, 2014



How much are a dozen deep-learning researchers worth? Apparently, more than \$400 million.



This week, Google reportedly paid that much to acquire DeepMind Technologies, a startup based in

**MIT  
Technology  
Review**
**BloombergBusinessweek  
Technology**

Acquisitions

## The Race to Buy the Human Brains Behind Deep Learning Machines

By Ashlee Vance | January 27, 2014

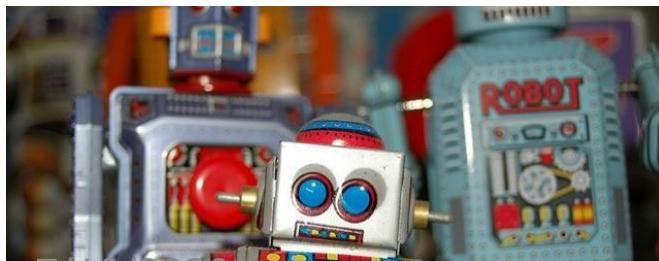
intelligence projects. "DeepMind is bona fide in terms of its research capabilities and depth," says Peter Lee, who heads Microsoft Research.

According to Lee, Microsoft, Facebook (FB), and Google find themselves in a battle for deep learning talent. Microsoft has gone from four full-time deep learning experts to 70 in the past three years. "We would have more if the talent was there to

**WIRED** GEAR SCIENCE ENTERTAINMENT BUSINESS SECURITY DESIGN  
INNOVATION INSIGHTS | [community content](#) | ▾ — featured

## Deep Learning's Role in the Age of Robots

BY JULIAN GREEN, JETPAC 05.02.14 2:56 PM

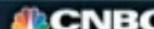


7/10/2020

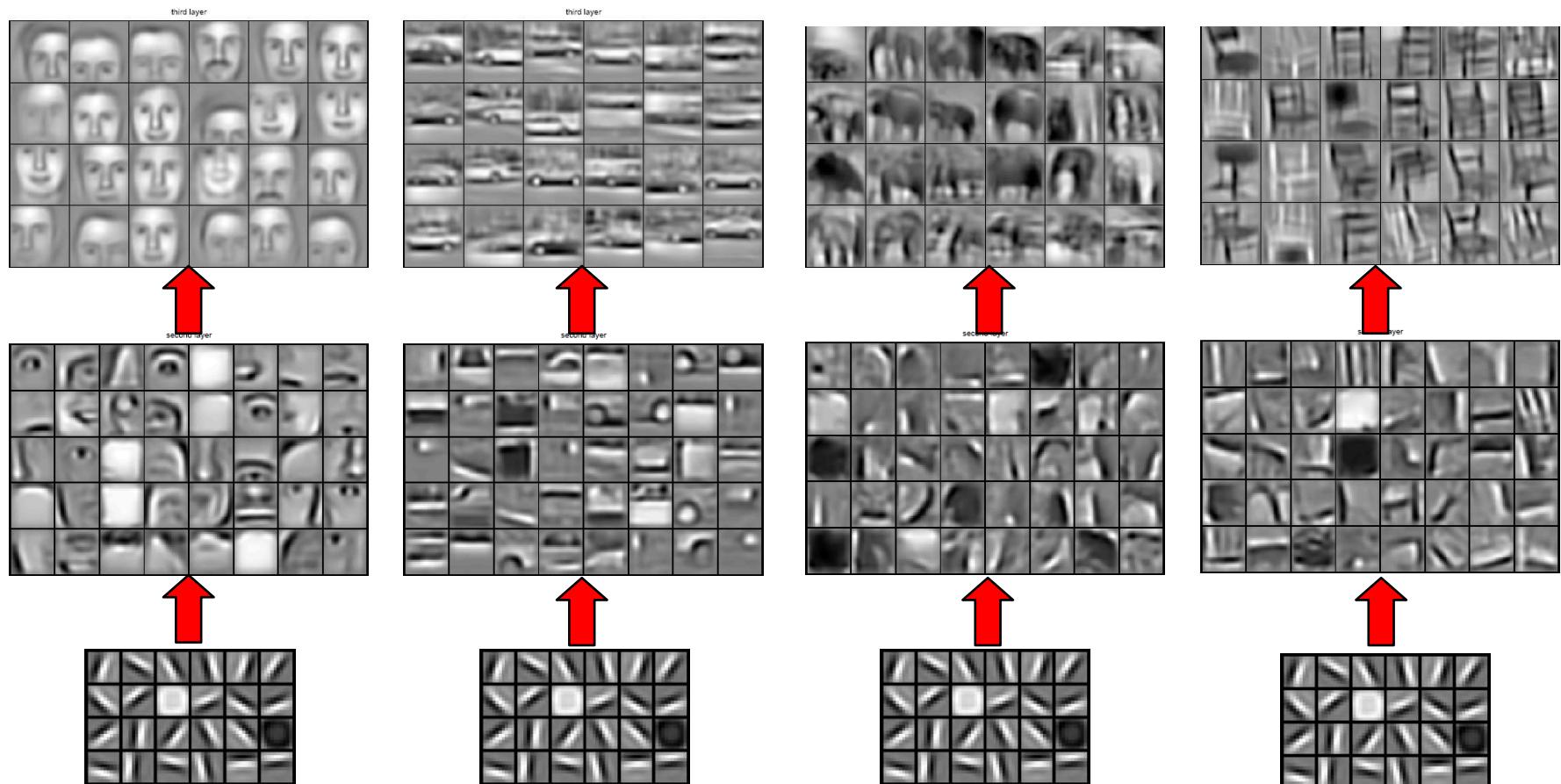
**DEEP LEARNING**

- » Computers learning and growing on their own
- » Able to understand complex, massive amounts of data

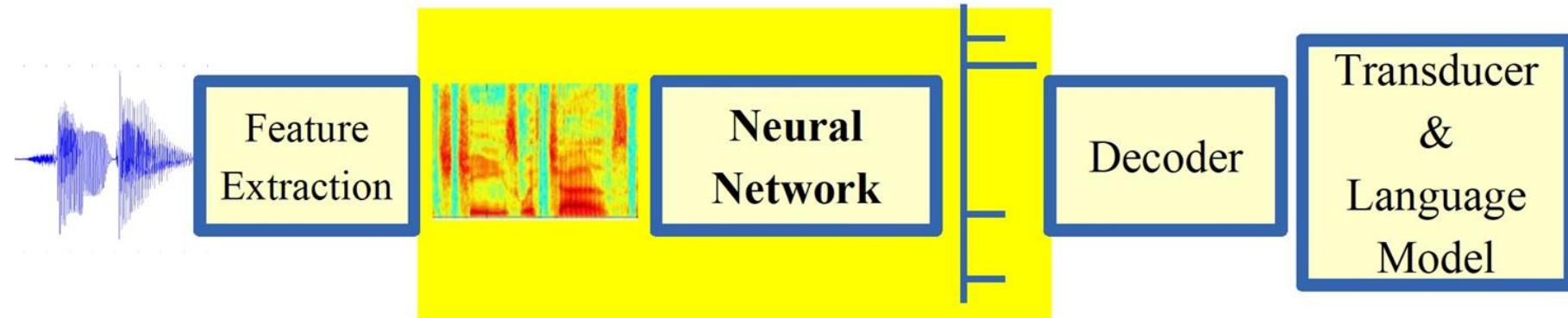
**DATA ECONOMY**  
**DEEP LEARNING**

BROUGHT TO YOU BY:  

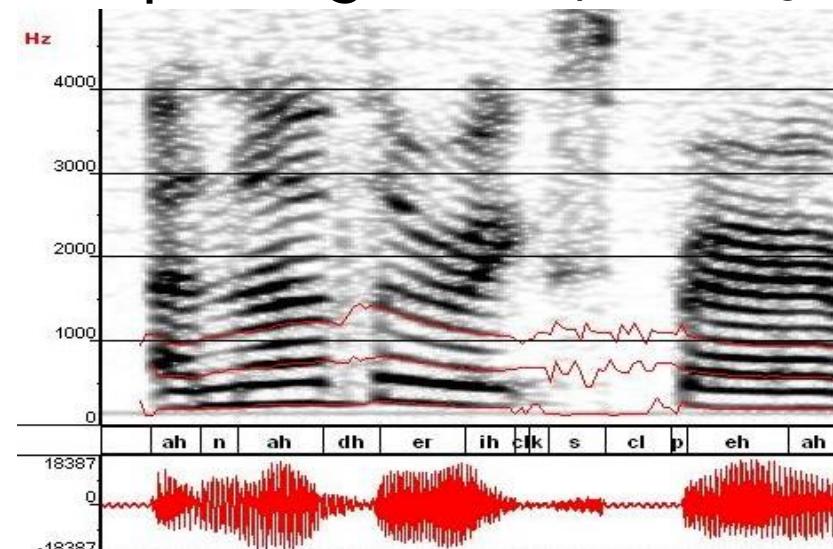
# Learning of Object Parts



# Automatic Speech Recognition



ML used to predict phoneme states from sound spectrogram Deep Learning Based Results



# Hidden Layers	1	2	4	8	10	12
Word Error Rate %	16.0	12.8	11.4	10.9	11.0	11.1

Baseline Gaussian Mixture Model based word error rate = 15.4%

[Zeiler et al. "On rectified linear units for speech recognition" ICASSP 2013]

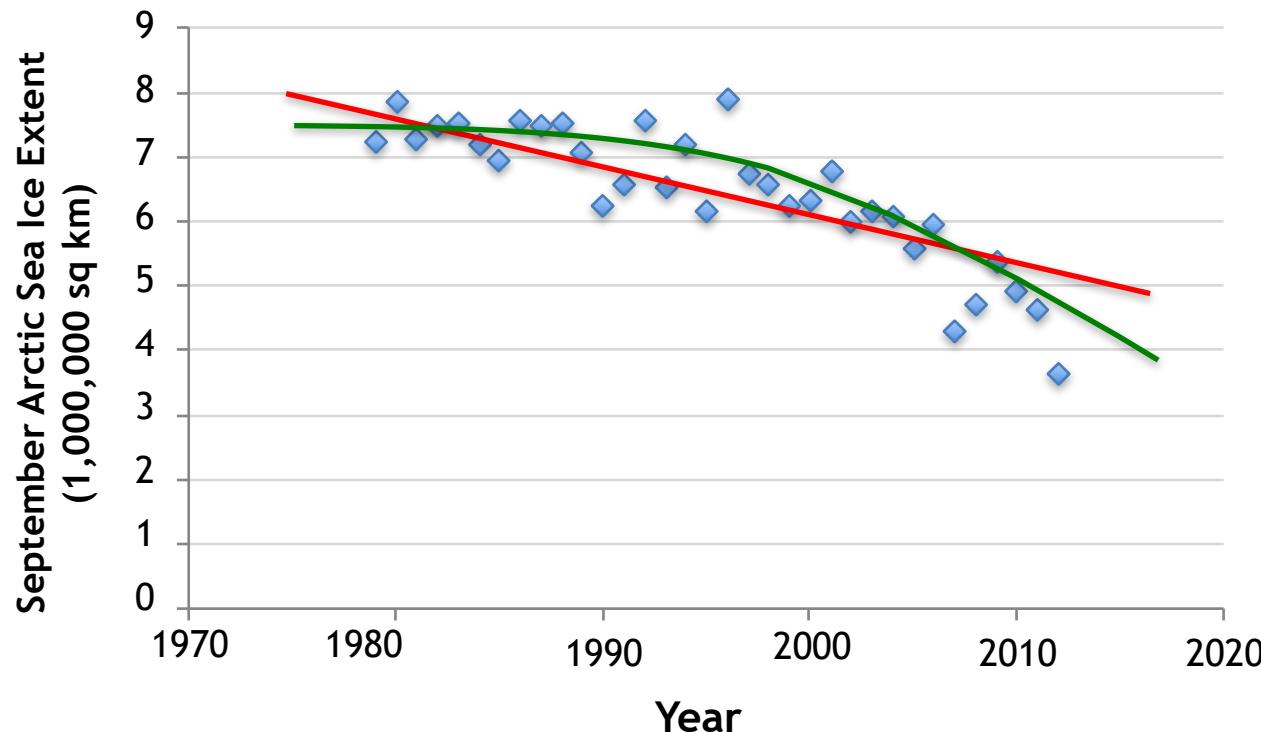
# Types of Learning

---

- **Supervised (inductive) learning**
  - Given: training data, desired outputs (labels)
- **Unsupervised learning**
  - Given: training data (without desired outputs)
- **Semi-supervised learning**
  - Given: training data + a few desired outputs
- **Reinforcement learning**
  - Given: rewards from sequence of actions

# Supervised Learning: Regression

- Given  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
- Learn a function  $f(x)$  to predict  $y$  given  $x$

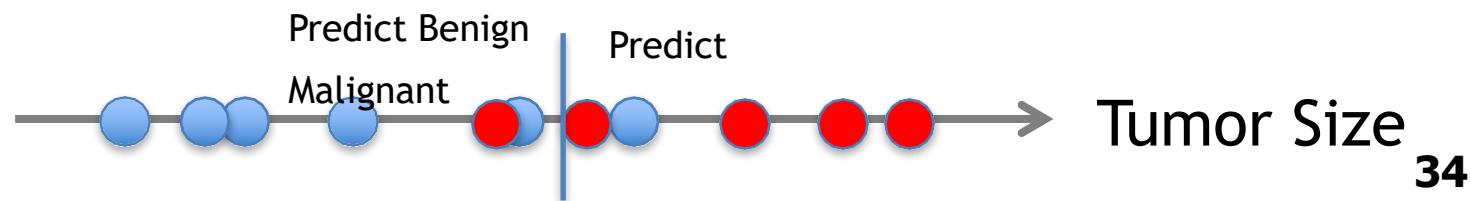
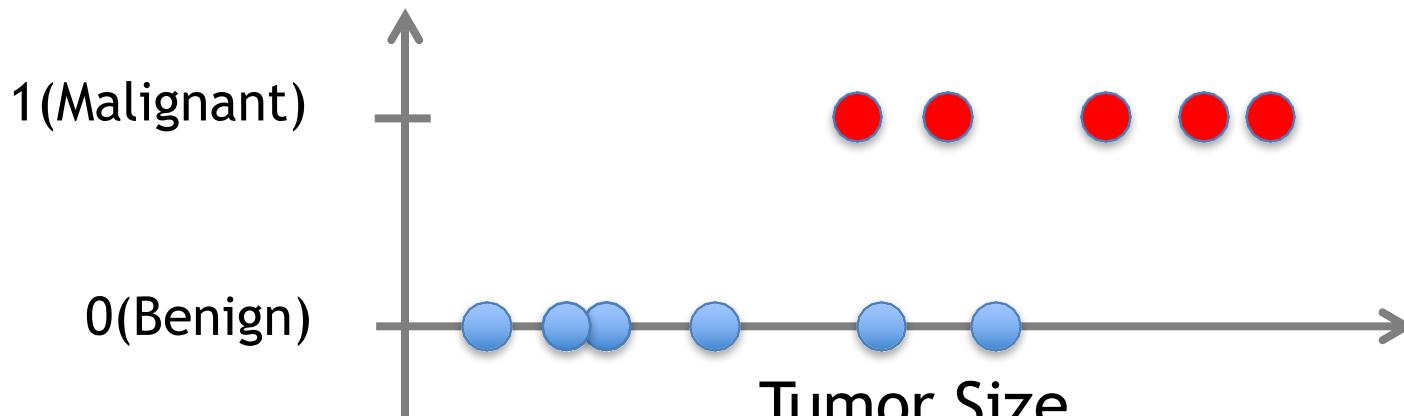


Data from G. Witt. Journal of Statistics Education, Volume 21, Number 1 (2013) Slide Credit: Eric Eaton **33**

# Supervised Learning: Classification

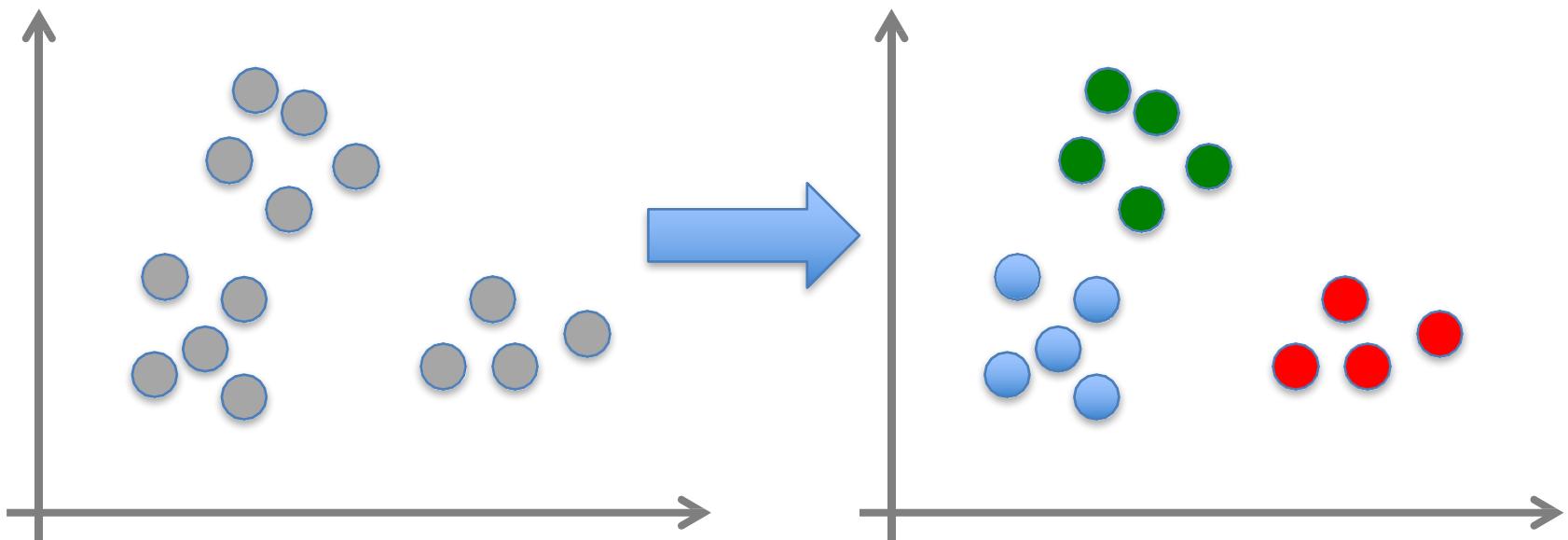
- Given  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
- Learn a function  $f(x)$  to predict  $y$  given  $x$ 
  - $y$  is categorical == classification

Breast Cancer (Malignant / Benign)



# Unsupervised Learning

- Given  $x_1, x_2, \dots, x_n$  (without labels)
- Output hidden structure behind the  $x$ 's
  - E.g., clustering



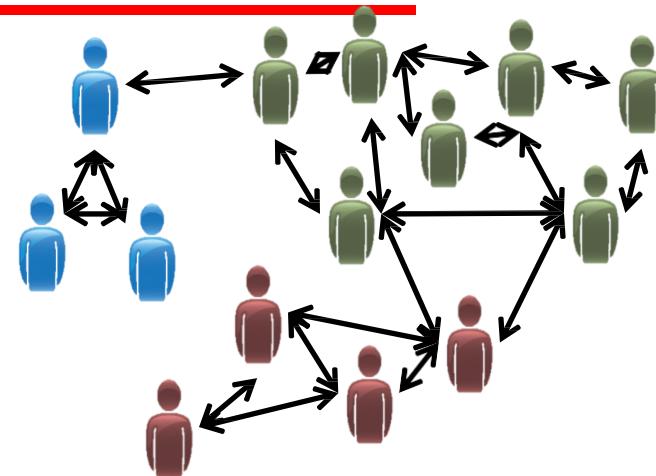
# Unsupervised Learning



Organize computing clusters



Market segmentation



Social network analysis

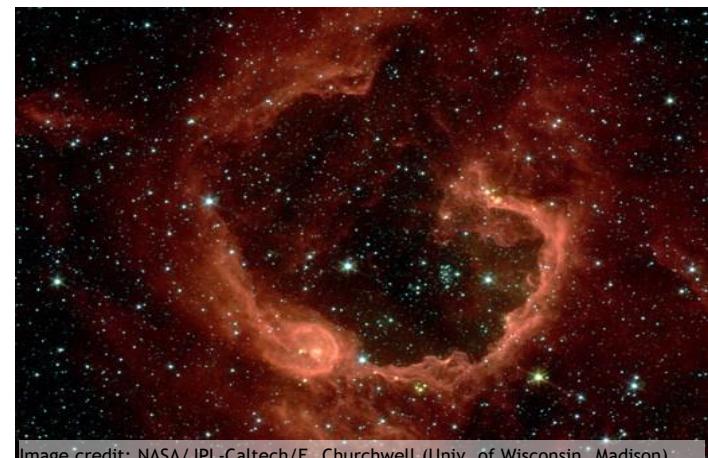


Image credit: NASA/JPL-Caltech/E. Churchwell (Univ. of Wisconsin, Madison)

Astronomical data analysis 36

# Reinforcement Learning

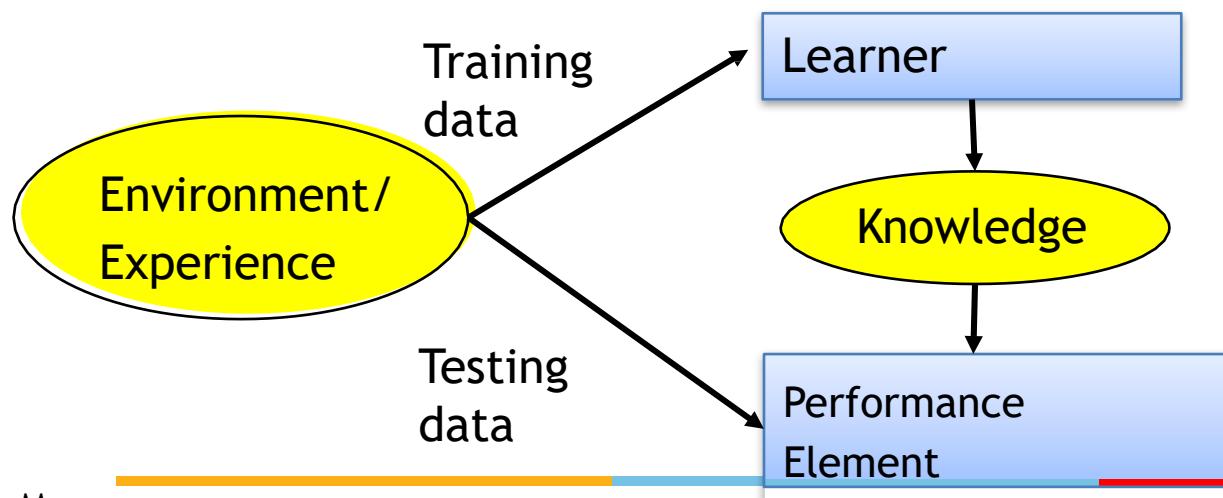
- Given a sequence of states and actions with (delayed) rewards, output a policy
  - Policy is a mapping from states → actions that tells you what to do in a given state
- Examples:
  - Credit assignment problem
  - Game playing
  - Robot in a maze

---

# Design a Learning System

# Designing a Learning System

- Choose the training experience(data)
- Choose exactly what is to be learned
  - i.e. the **target function**
- Choose how to represent the target function
- Choose a learning algorithm to infer the target function from the experience



# Designing a Learning System: An Example

---

1. Problem Description (Ex: Playing checkers)
2. Choosing the Training Experience (data expressed as features)
3. Choosing the Target Function to be learnt (Ex: deciding next board position)
4. Choosing a Representation for the Target Function (design a function as linear etc)
5. Choosing a Function Approximation Algorithm (parameters learning using loss function)
6. Final Design

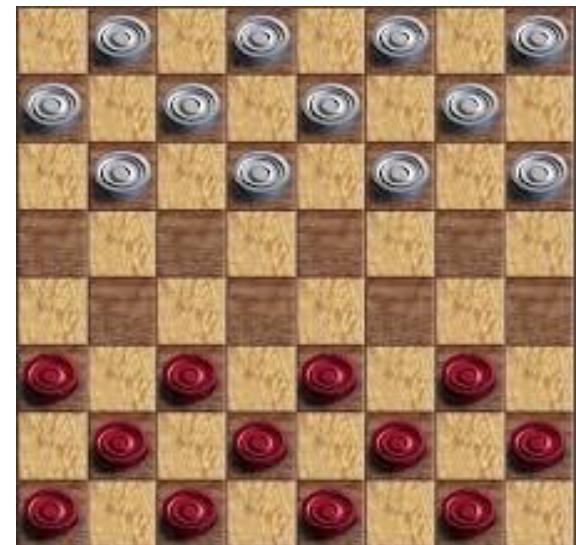
# Choosing the training experience

---

- The first problem to choose the type of training experience from which our system will learn.
  - ✓ The type of training experience available can have a significant impact on success or failure of the learner.
  - ✓ One key attribute is whether the training experience provides ***direct*** or ***indirect*** feedback regarding the choices made by the performance system.

# Choosing the training experience

- In learning to play checkers, the system might learn from **direct training** examples consisting of individual checkers board states and the correct move for each.
- Alternatively, it might have available only **indirect information** consisting of the move sequences and final outcomes of various games played.



# Choosing the training experience

---

- In **indirect training**, information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.
- The learner faces an additional problem of **credit assignment**, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome.
- Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves. Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

# Choosing the training experience

---

- A second important attribute of the training experience is the degree to which the learner controls the sequence of training examples.
  - the learner might rely on the teacher to select informative board states and to provide the correct move for each.
  - the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.

# Choosing the training experience

---

- the learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with no teacher present.
  - the learner may choose between experimenting with novel board states that it has not yet considered, or sharpen its skill by playing minor variations of lines of play it currently finds most promising.

# Choosing the training experience

---

- A third important attribute of the training experience is how well it represents the distribution of examples over which the final system performance  $P$  must be measured.
  - learning is most reliable when the training examples follow a distribution similar to that of future test examples.
  - the performance metric  $P$  is the percent of games the system wins in the world tournament. If its training experience  $E$  consists only of games played against itself, there is an obvious danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.

# Choosing the training experience

---

- For example, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion.
- it is often necessary to learn from a distribution of examples that is somewhat different from those on which the final system will be evaluated
- one distribution of examples will not necessarily lead to strong performance over some other distribution.

# Choosing the Target Function

---

- *ChooseMove*, however, is difficult to learn.
- An easier and related target function to learn is function  $V: B \rightarrow R$ , which assigns a numerical score to each board. The better the board positions B, the higher the score R.
- If the system can successfully learn such a target function  $V$ , then it can easily use it to select the best move from any current board position.
- This can be accomplished by generating the successor board state produced by every legal move, then using  $V$  to choose the best successor state and therefore the best legal move.

# Choosing the Target Function

---

- Let us therefore define the target value  $V(b)$  for an arbitrary board state  $b$  in  $B$ , as follows:
  - If  $b$  is a final board state that is won, then  $V(b) = 100$
  - If  $b$  is a final board state that is lost, then  $V(b) = -100$
  - If  $b$  is a final board state that is draw, then  $V(b) = 0$
  - If  $b$  is a not a final state in the game, then  $V(b) = V(b')$ , where  $b'$  is the best final board state that can be achieved starting from  $b$  and playing optimally until the end of the game.

# Choosing the Target Function

---

- This recursive definition specifies a value of  $V(b)$  for every board state  $b$ .
- It is not usable by player - it is not efficiently computable - it is a *nonoperational definition*.
- The goal of learning in this case is to discover an *operational description of V* i.e., a description that can be used by the checkers - playing program to evaluate states and select moves within realistic time bounds.

# Choosing the Target Function

---

- Reduced the learning task to the problem of discovering an operational description of the ideal target function  $V$  – it is difficult to learn  $V$  perfectly.
- We often expect learning algorithms to acquire only some *approximation* to the target function -- is called *function approximation*. [The actual function can often not be learned and must be approximated]

# Choosing a Representation for the Target Function

- **Expressiveness versus Training set size**
  - More expressive the representation of the target function, the closer to the “truth” we can get.
  - More expressive the representation, the more training examples are necessary to choose among the large number of “representable” possibilities.
- **Example of a representation:**
  - $x_1/x_2 = \# \text{ of black/red pieces on the board}$
  - $x_3/x_4 = \# \text{ of black/red king on the board}$
  - $x_5/x_6 = \# \text{ of black/red pieces threatened by red/black}$

$$\hat{V}(b) = w_0 + w_1.x_1 + w_2.x_2 + w_3.x_3 + w_4.x_4 + w_5.x_5 + w_6.x_6$$

wi's are adjustable  
or “learnable”  
coefficients

# Choosing a Representation for the Target Function

---

- $w_0$  through  $w_6$  are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights  $w_1$  through  $w_6$  will determine the relative importance of the various board features in determining the value of the board, whereas the weight  $w_0$  will provide an additive constant to the board value.

# Choosing a Representation for the Target Function

---

**Partial design of a checkers learning program:**

- Task  $T$ : playing checkers
- Performance measure  $P$ : percent of games won in the world tournament
- Training experience  $E$ : games played against itself
- *Target function*:  $V:Board \rightarrow \mathbb{R}$
- *Target function representation*

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

# Choosing a Function Approximation Algorithm

---

- **Generating Training Examples of the form**

**<math>b, V\_{\text{train}}(b)</math>** [e.g. <math><x\_1=3, x\_2=0, x\_3=1, x\_4=0, x\_5=0, x\_6=0, +100 (=blacks won)></math>]

- $x_1$  = # of black pieces on the board
- $x_2$  = # of red pieces on the board
- $x_3$  = # of black king on the board
- $x_4$  = # of red king on the board
- $x_5$  = # of black pieces threatened by red
- $x_6$  = # of red pieces threatened by black

# Choosing a Function Approximation Algorithm

---

- **Estimating the training Values**
  - only training information available to learner is whether the game was eventually won or lost.
  - we require training examples that assign specific scores to specific board states.
  - easy to assign a value to board states that correspond to the end of the game,
  - less obvious how to assign training values to the more numerous intermediate board states that occur before the game's end.
  - fact that the game was eventually won or lost does not necessarily indicate that every board state along the game path was necessarily good or bad.

# Choosing a Function Approximation Algorithm

---

- Estimating the training Values

- Despite the ambiguity inherent in estimating training values for intermediate board states.
- one simple approach is to assign the training value of  $\hat{V}_{train}(b)$  for any intermediate board state  $b$  to be  $\hat{V}(\text{Successor}(b))$ , where  $\hat{V}$  is the learner's current approximation to  $V$  and where  $\text{Successor}(b)$  denotes the next board state following  $b$
- This rule for estimating training values can be summarized as

$$V_{train}(b) \leftarrow \hat{V}(\text{Successor}(b))$$

# Choosing a Function Approximation Algorithm

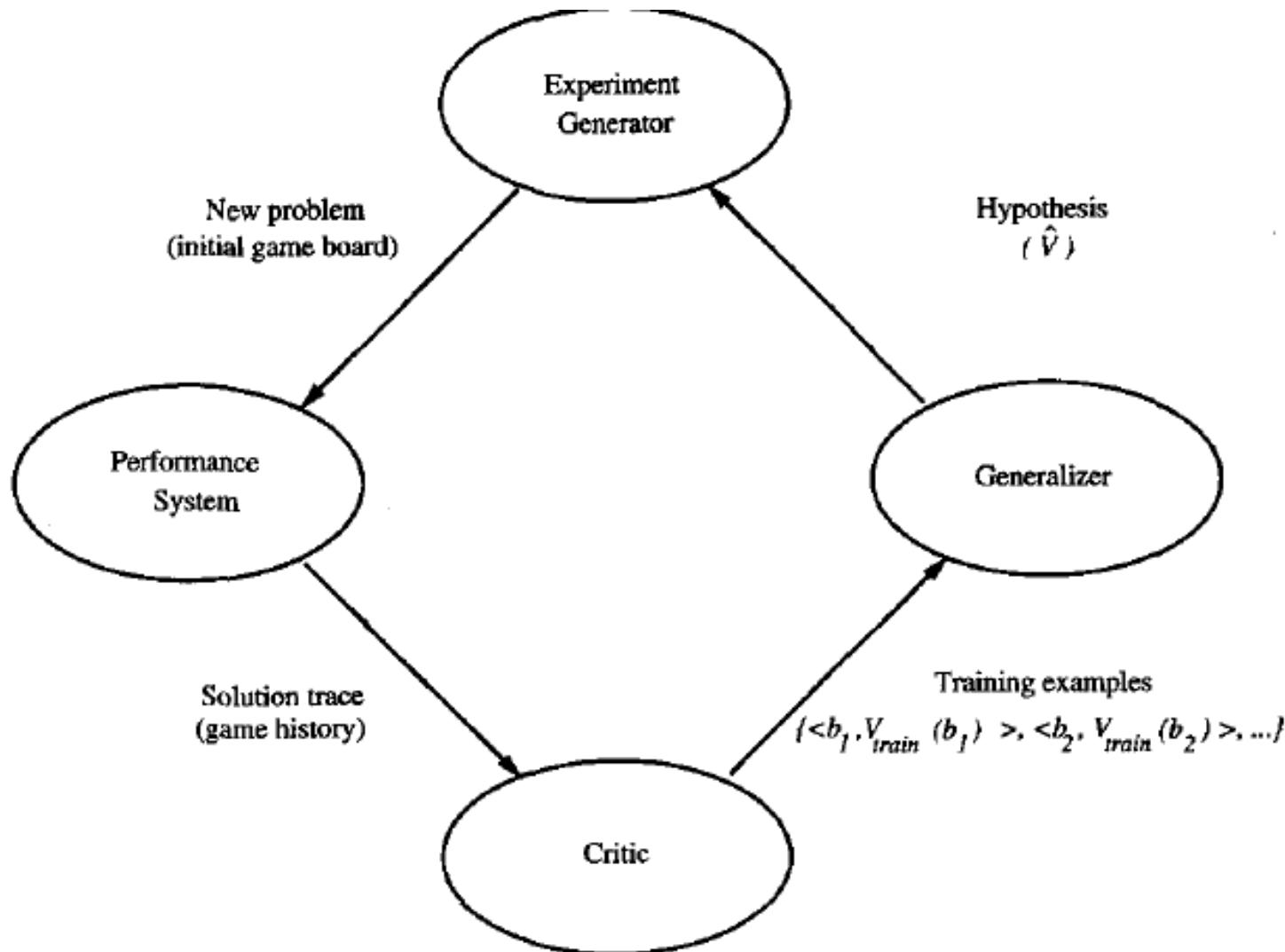
---

- Adjust the weights
  - Specify the learning algorithm for choosing the weights  $w_i$  to best fit the set of training examples  $\{ \langle b, V_{train}(b) \rangle \}$ .
  - One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error  $E$  between the training values and the values predicted by the hypothesis  $\hat{V}$ .

$$E \equiv \sum_{\langle b, V_{train}(b) \rangle \in \text{training examples}} (V_{train}(b) - \hat{V}(b))^2$$

- Thus, we seek the weights, or equivalently the  $\hat{V}$ , that minimize  $E$  for the observed training examples.

# Final Design for Checkers Learning



# Final Design for Checkers Learning

- **The Performance Module (performance)** : Takes as input a new board and outputs a trace of the game it played against itself.
- **The Critic (data generation)** : Takes as input the trace of a game and outputs a set of training examples of the target function
- **The Generalizer (learner)**: Takes as input training examples and outputs a hypothesis which estimates the target function.
- **The Experiment Generator (task)**: Takes as input the current hypothesis (currently learned function) and outputs a new problem (an initial board state) for the performance system to explore

# Issues in Machine Learning

---

- What algorithms are available for learning a concept?  
How well do they perform?
- How much training data is sufficient to learn a concept with high confidence?
- When is it useful to use prior knowledge?
- Are some training examples more useful than others?
- What are the best tasks for a system to learn?
- What is the best way for a system to represent its knowledge?

# ML in a Nutshell

---

- Tens of thousands of machine learning algorithms
  - Hundreds new every year
- Every ML algorithm has three components
  - Representation
  - Optimization
  - Evaluation

# Regression

$$y = f(x)$$

↑              ↑              ↗  
 output      prediction      features  
 function

- **Training:** given a *training set* of labeled examples  $\{(x_1, y_1), \dots, (x_N, y_N)\}$ , estimate the prediction function  $f$  by minimizing the prediction error on the training set
- **Testing:** apply  $f$  to a never before seen *test example*  $x$  and output the predicted value  $y = f(x)$

# Regression Example

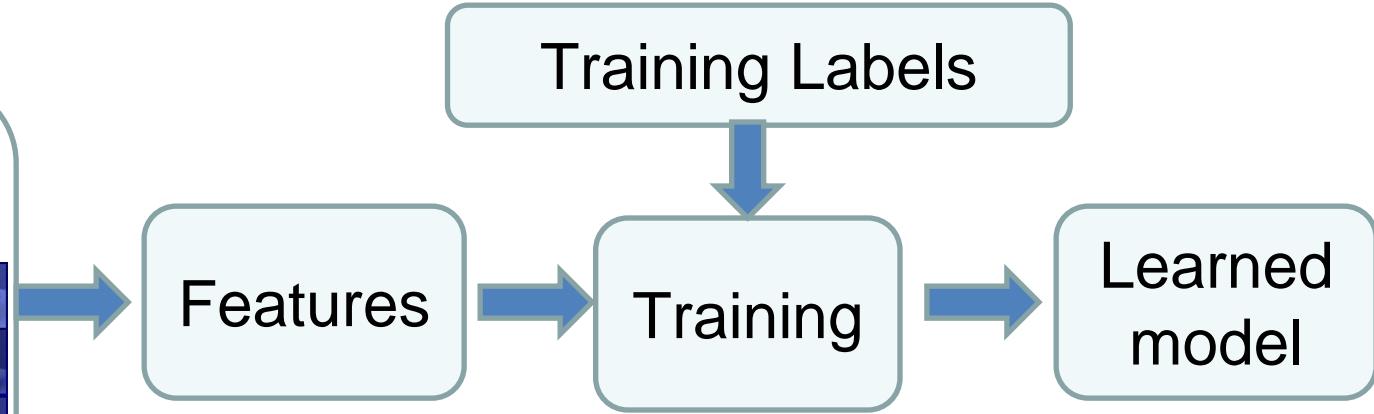
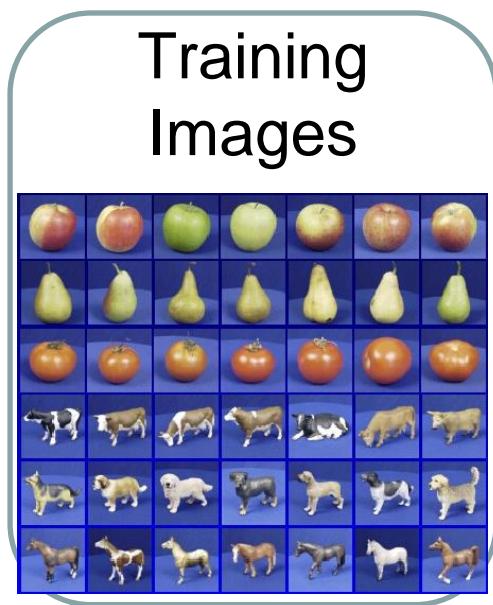
---

- Apply a prediction function to a feature representation of the image to get the desired output:

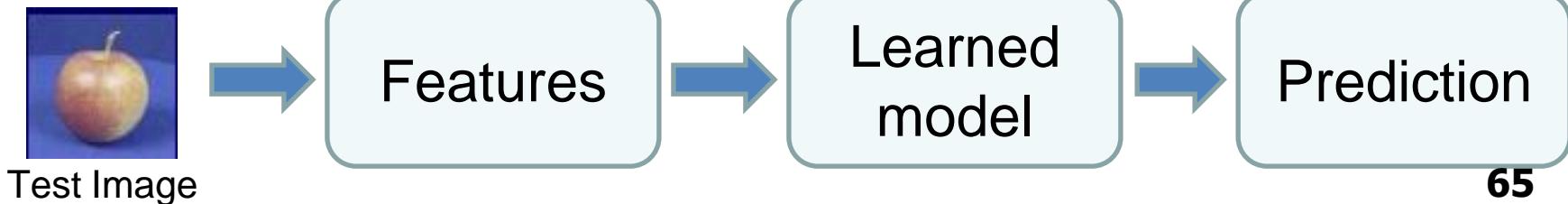
$$f(\text{apple}) = \text{"apple"}$$
$$f(\text{tomato}) = \text{"tomato"}$$
$$f(\text{cow}) = \text{"cow"}$$

# Classification

## Training



## Testing



# Function Representations

- Numerical functions
  - Linear regression
  - Neural networks
  - Support vector machines
- Symbolic functions
  - Decision trees
  - Rules in propositional logic
  - Rules in first-order predicate logic
- Instance-based functions
  - Nearest-neighbor
  - Case-based
- Probabilistic Graphical Models
  - Naïve Bayes
  - Bayesian networks
  - Hidden-Markov Models (HMMs)
  - Probabilistic Context Free Grammars (PCFGs)
  - Markov networks

# Evaluation

---

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- etc.

# Evaluating Performance

---

- If  $y$  is discrete:
  - Accuracy: # correctly classified / # all test examples
  - Precision/recall
    - True Positive, False Positive, True Negative, False Negative
    - Precision =  $TP / (TP + FP) = \# \text{predicted true pos} / \# \text{predicted pos}$
    - Recall =  $TP / (TP + FN) = \# \text{predicted true pos} / \# \text{true pos}$
  - F-measure
    - =  $2PR / (P + R)$
- Want evaluation metric to be in some range, e.g. [0 1]
  - 0 = worst possible classifier, 1 = best possible classifier

# Evaluating Performance

---

- If  $y$  is continuous:
  - Sum-of-Squared-Differences (SSD) error between predicted and true  $y$ :

$$E = \sum_{i=1}^n (f(x_i) - y_i)^2$$

# Training vs Testing

---

- What do we want?
  - High accuracy on training data?
  - No, high accuracy on *unseen/new/test data!*
  - Why is this tricky?
- Training data
  - Features (x) and labels (y) used to learn mapping f
- Test data
  - Features used to make a prediction
  - Labels only used to see how well we've learned f!!!
- Validation data
  - Held-out set of the *training data*
  - Can use both features and labels to tune *parameters* of the model we're learning

# Training vs. Test Distribution

---

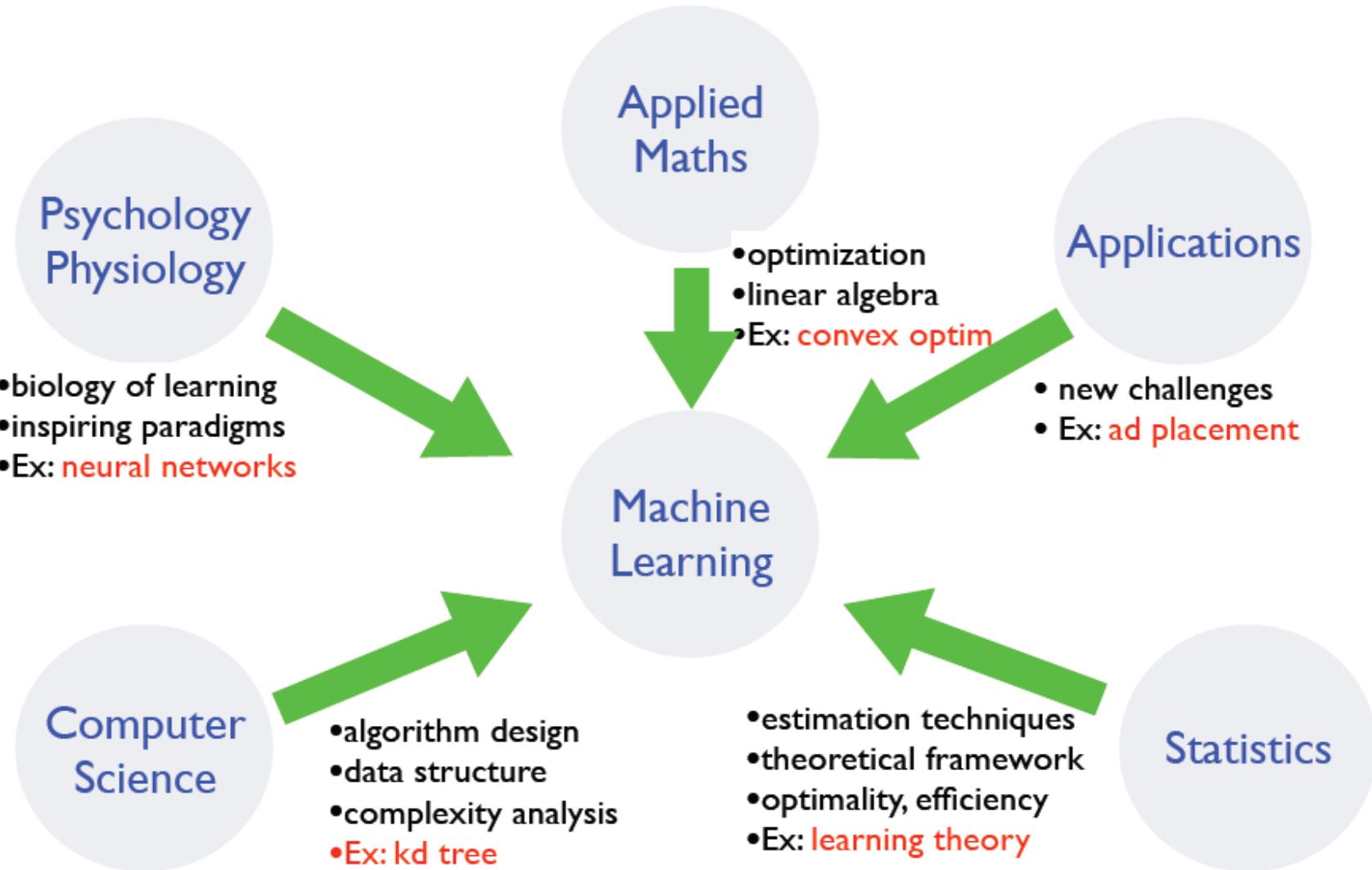
- We generally assume that training and test examples are independently drawn from the same overall distribution of data
  - We call this “i.i.d” which stands for “independent and identically distributed”

Slide credit: Ray Mooney

Loop

- Understand domain, prior knowledge, and goals
- Data integration, selection, cleaning, pre-processing, etc.
- Learn models
- Interpret results
- Consolidate and deploy discovered knowledge

# Where does ML fit in?



# Recommended Readings

---

- Tom M. Mitchell, Machine Learning, The McGraw-Hill Companies, Inc. International Edition 1997. **[Ch. 1]**
- <http://www.cs.princeton.edu/courses/archive/spr08/cos511/> **[Web]**

---

# Thank You



**BITS** Pilani  
Pilani Campus

# Machine Learning DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## **Lecture No. – 2 | Math Preliminaries**

**Date – 02/11/2019**

**Time – 9:00 AM – 11:00 AM**

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Sugato Ghosal, Prof. S.K. H. Islam from BITS Pilani and many others who made their course materials freely available online.

# Session Content

---

- Linear Algebra Review
  - Calculus Review
  - Probability Theory (Ref: 1.2)
  - Decision Theory (Ref: 1.5)
  - Information Theory (Ref: 1.6)
-

# Linear algebra Review

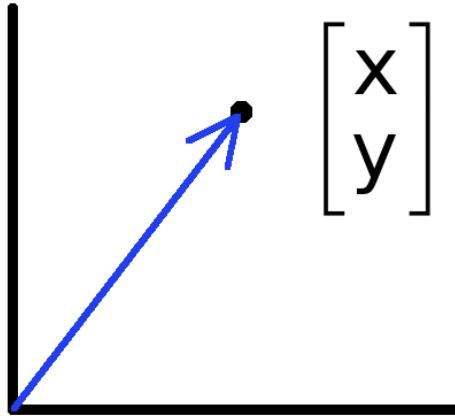
See <http://cs229.stanford.edu/section/cs229-linalg.pdf> for more

# Vectors and Matrices

---

- Collections of ordered numbers that represent movements in space, word counts, movie ratings, pixel brightness, etc.
- Vector is a mathematical quantity that has magnitude and direction

# Vectors



- Vectors can represent an offset in 2D or 3D space
- Points are just vectors from the origin

- Data can also be treated as a vector
- Such vectors don't have a geometric interpretation, but calculations like "distance" still have value

# Vector

---

- A column vector  $\mathbf{v} \in \mathbb{R}^{n \times 1}$  where

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

- A row vector  $\mathbf{v}^T \in \mathbb{R}^{1 \times n}$  where

$$\mathbf{v}^T = [v_1 \quad v_2 \quad \dots \quad v_n]$$

$T$  denotes the transpose operation

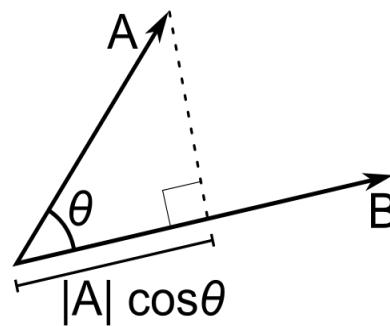
---

# Inner Product

- Multiply corresponding entries of two vectors and add up the result

$$\mathbf{x}^T \mathbf{y} = [x_1 \quad \dots \quad x_n] \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^n x_i y_i \quad (\text{scalar})$$

- If B is a unit vector, then A·B gives the length of A which lies in the direction of B (projection)



(if B is unit-length hence norm is 1)

# Norms

- **Norm** is a function that assigns a strictly positive *length* or *size* to each vector in a vector space—except for the zero vector
- **L<sup>1</sup> norm** - One-dimensional vector spaces

$$\|\mathbf{x}\|_1 := \sum_{i=1}^n |x_i|$$

- **L<sup>2</sup> norm** - *n*-dimensional Euclidean space  $\mathbf{R}^n$ ,
- **L<sup>p</sup> norm** - Let  $p \geq 1$  be a real number. The  $p$  norm of vector  $\mathbf{x}=(x_1, x_2, \dots, x_n)$

$$\|\mathbf{x}\|_p := \left( \sum_{i=1}^n |x_i|^p \right)^{1/p}$$

# Matrix

---

- A matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  is an array of numbers with size  $m \downarrow$  by  $n \rightarrow$ , i.e. m rows and n columns.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & & & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

- If  $m = n$ , we say that  $\mathbf{A}$  is square.

# Matrix Operations

- Addition

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a+1 & b+2 \\ c+3 & d+4 \end{bmatrix}$$

- Can only add a matrix with matching dimensions, or a scalar.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + 7 = \begin{bmatrix} a+7 & b+7 \\ c+7 & d+7 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times 3 = \begin{bmatrix} 3a & 3b \\ 3c & 3d \end{bmatrix}$$

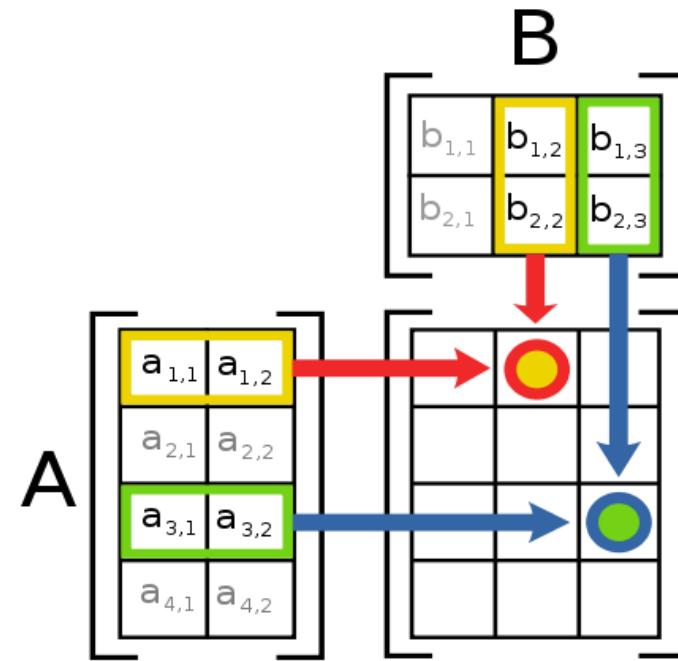
# Matrix Multiplication

---

- Let X be an  $a \times b$  matrix, Y be an  $b \times c$  matrix
- Then  $Z = X^*Y$  is an  $a \times c$  matrix
- Second dimension of first matrix, and first dimension of second matrix have to be the same, for matrix multiplication to be possible

# Matrix Multiplication

- The product  $AB$  is:



- Each entry in the result is (that row of  $A$ ) dot product with (that column of  $B$ )

# Different types of product

---

- $x, y$  = column vectors ( $n \times 1$ )
  - $X, Y$  = matrices ( $m \times n$ )
  - $x, y$  = scalars ( $1 \times 1$ )
- 
- $x^T y = x \cdot y$  = inner product ( $1 \times n \times n \times 1 = \text{scalar}$ )
  - $x \otimes y = xy^T$  = outer product ( $n \times 1 \times 1 \times n = \text{matrix}$ )
- 
- $X * Y$  = matrix product
  - $X .* Y$  = element-wise product

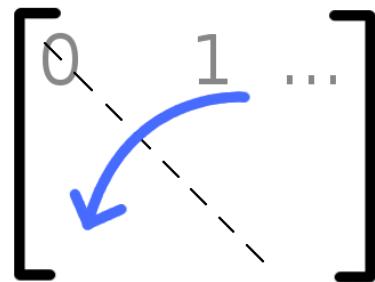
# Inverse

---

- Given a matrix  $A$ , its inverse  $A^{-1}$  is a matrix such that  $AA^{-1} = A^{-1}A = I$
- E.g.  $\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$
- Inverse does not always exist. If  $A^{-1}$  exists,  $A$  is *invertible* or *non-singular*. Otherwise, it's *singular*.

# Matrix Operations

- Transpose – flip matrix, so row 1 becomes column 1

$$\begin{bmatrix} 0 & 1 & \dots \end{bmatrix}$$


$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}^T = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

- A useful identity:

$$(ABC)^T = C^T B^T A^T$$

# Matrix Operation Properties

- Matrix addition is commutative and associative
  - $A + B = B + A$
  - $A + (B + C) = (A + B) + C$
- Matrix multiplication is associative and distributive but *not* commutative
  - $A(B^*C) = (A^*B)C$
  - $A(B + C) = A^*B + A^*C$
  - $A^*B \neq B^*A$

# Linear independence

---

- Suppose we have a set of vectors  $\mathbf{v}_1, \dots, \mathbf{v}_n$
- If we can express  $\mathbf{v}_1$  as a linear combination of the other vectors  $\mathbf{v}_2 \dots \mathbf{v}_n$ , then  $\mathbf{v}_1$  is linearly *dependent* on the other vectors.
  - The direction  $\mathbf{v}_1$  can be expressed as a combination of the directions  $\mathbf{v}_2 \dots \mathbf{v}_n$ . (E.g.  $\mathbf{v}_1 = .7 \mathbf{v}_2 - .7 \mathbf{v}_4$ )
- If no vector is linearly dependent on the rest of the set, the set is linearly *independent*.
  - Common case: a set of vectors  $\mathbf{v}_1, \dots, \mathbf{v}_n$  is always linearly independent if each vector is perpendicular to every other vector (and non-zero)

# Matrix Rank

---

- Column/row rank
  - $\text{col-rank}(A)$ =no. of linearly independent columns
  - $\text{row-rank}(A)$ =no. of linearly independent rows
- Rank of a matrix is the number of linearly independent rows or columns (whichever is smaller).
- Column rank always equals row rank
- If a matrix is not full rank, inverse doesn't exist
  - Inverse also doesn't exist for non-square matrices

# Matrix Rank

---

- Rank of a matrix denotes the **information content** of the matrix.
- The lower the rank, the lower is the "information content".
- For instance, when we say a rank 1 matrix, the matrix can be written as a product of a column vector times a row vector
- i.e. if  $u$  and  $v$  are column vectors, matrix  $uv^T$  is a rank one matrix.
- In general, if we know that a matrix  $A \in \mathbb{R}$  of size  $m \times n$  is of rank  $p$ , then we can write  $A$  as  $UV^T$  where  $U \in \mathbb{R}$  of size  $m \times p$  and is of rank  $p$  and  $V \in \mathbb{R}^{n \times p}$  and is of rank  $p$

# Matrix Rank : Example

---

$$\begin{bmatrix} 1 & 0 & 1 \\ -2 & -3 & 1 \\ 3 & 3 & 0 \end{bmatrix}$$

Matrix has rank 2: the first two columns are linearly independent, so the rank is at least 2,

- the third column is a linear combination of the first two (the second subtracted from the first), the three columns are linearly dependent so the rank must be less than 3

# Orthogonal Matrix

---

- Orthogonal matrix is a square matrix whose columns and rows are orthogonal unit vectors (i.e., orthonormal vectors), i.e.  $Q^T Q = Q Q^T = I$ , where  $I$  is the identity matrix.
- Matrix  $Q$  is orthogonal if its transpose is equal to its inverse:  $Q^T = Q^{-1}$ .

# Singular Value Decomposition (SVD)

---

- There are several computer algorithms that can “factor” a matrix, representing it as the product of some other matrices
- The most useful of these is the Singular Value Decomposition
- **Singular value decomposition (SVD)** is a factorization of a real or complex matrix
- Represents any matrix  $\mathbf{A}$  as a product of three matrices:  $\mathbf{U}\Sigma\mathbf{V}^T$

[https://en.wikipedia.org/wiki/Singular\\_value\\_decomposition](https://en.wikipedia.org/wiki/Singular_value_decomposition)

# Singular Value Decomposition (SVD)

---

- In general, if  $\mathbf{A}$  is  $m \times n$ , then  $\mathbf{U}$  will be  $m \times m$ ,  $\Sigma$  will be  $m \times n$ , and  $\mathbf{V}^T$  will be  $n \times n$ .

$$\begin{matrix}
 U & \Sigma & V^T \\
 \begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix} & \times & \begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} & \times & \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} = & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \\
 A
 \end{matrix}$$

# Singular Value Decomposition (SVD)

For square, positive semi-definite matrix A

$$\mathbf{U}\Sigma\mathbf{V}^T = \mathbf{A}$$

Where **U** and **V** are rotation matrices, and  **$\Sigma$**  is a scaling matrix.

$$U \quad \quad \quad \Sigma \quad \quad \quad V^T \quad \quad \quad A \\ \begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix} \times \begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix} \times \begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}$$

- U and V are unitary matrices, i.e.,  $UU^T=U^TU=I$  (identity matrix)

[https://www.youtube.com/watch?v=NsNNI\\_JPUY](https://www.youtube.com/watch?v=NsNNI_JPUY)

# Positive-semidefinite matrix

---

- Symmetric ( $M = M^T$ )  $n * n$  real matrix  $M$  is said to be **positive-semidefinite** if the scalar  $x^T M x$  is positive or zero for every non-zero column vector  $x$  of  $n$  real numbers  
(domain of  $x = 1^n$ , domain of  $M = n \times n$  and  $x^T = n \times 1$ )
- When interpreting  $M x$  as the output of matrix,  $M$ , that is acting on an input,  $x$ , the property of **positive-semidefinite** implies that the output always is positive or zero ; inner product with the input

$$M \text{ positive semi-definite} \iff x^T M x \geq 0 \text{ for all } x \in \mathbb{R}^n$$

---

---

# Calculus review

# Differentiation

---

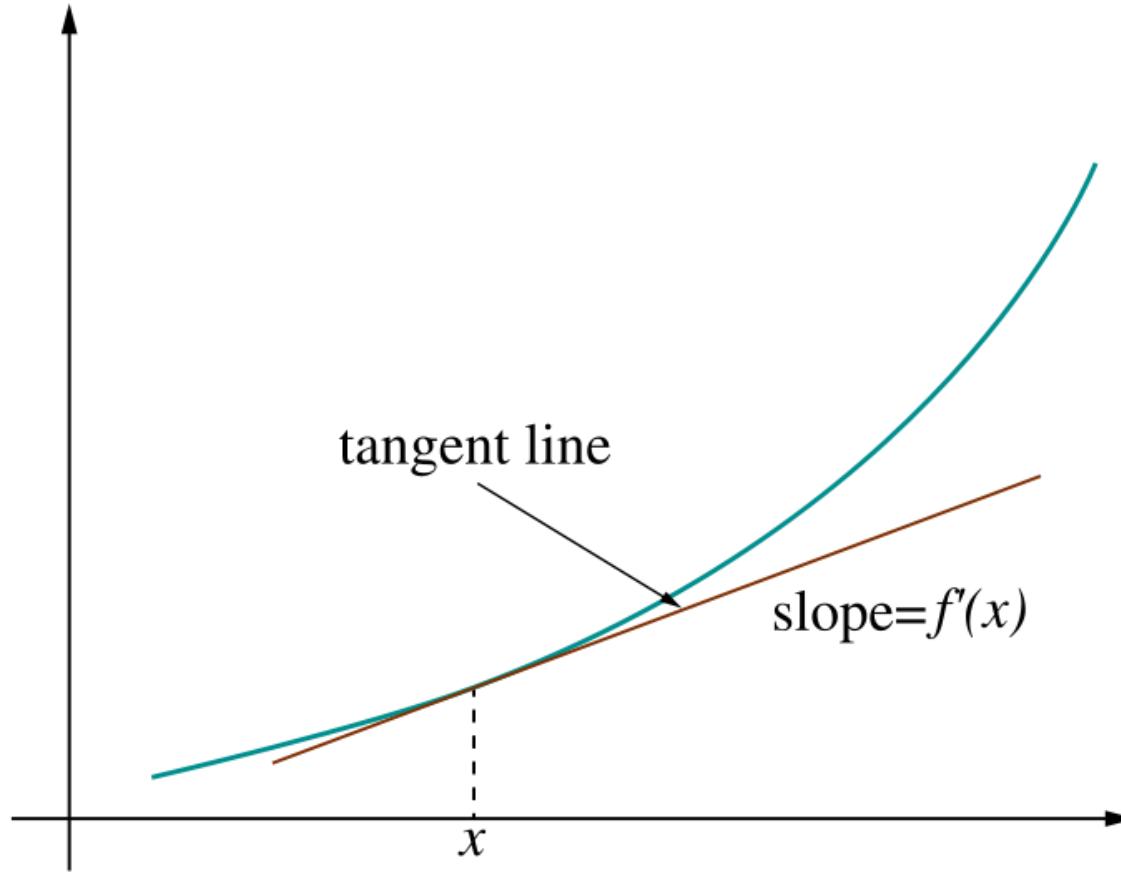
The derivative provides us information about the rate of change of a function.

The derivative of a function is also a function.

Example:

The derivative of the acceleration function is the velocity function.

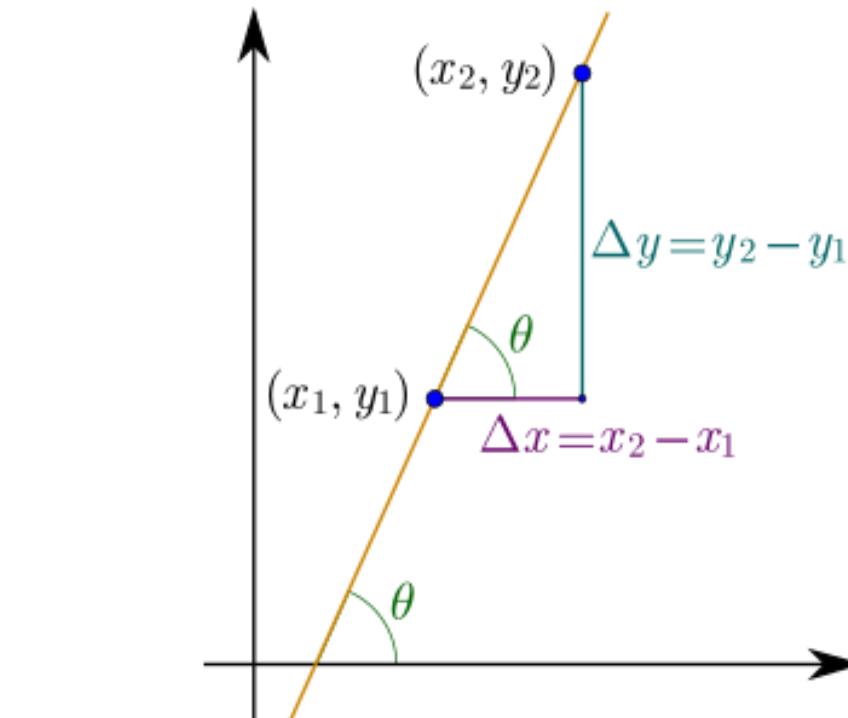
# Derivative = rate of change



# Derivative = rate of change

- Linear function  $y = mx + b$

- Slope  $m = \frac{\text{change in } y}{\text{change in } x} = \frac{\Delta y}{\Delta x},$



# Ways to Write the Derivative

Given the function  $f(x)$ , we can write its derivative in the following ways:

- $f'(x)$
- $\frac{d}{dx}f(x)$

The derivative of  $x$  is commonly written  $dx$ .

# Differentiation Formulas

---

The following are common differentiation formulas:

- The derivative of a constant is 0.

$$\frac{d}{du}c = 0$$

- The derivative of a sum is the sum of the derivatives.

$$\frac{d}{du}(f(u) + g(u)) = f'(u) + g'(u)$$

# More Formulas

---

- The derivative of  $u$  to a constant power:

$$\frac{d}{du} u^n = n * u^{n-1}$$

- The derivative of  $e$ :

$$\frac{d}{du} e^u = e^u$$

- The derivative of  $\log$ :

$$\frac{d}{du} \log(u) = \frac{1}{u}$$

# Product and Quotient

---

The product rule and quotient rules are commonly used in differentiation.

- Product rule:

$$\frac{d}{du}(f(u) * g(u)) = f(u)g'(u) + g(u)f'(u)$$

- Quotient rule:

$$\frac{d}{du}\left(\frac{f(u)}{g(u)}\right) = \frac{g(u)f'(u) - f(u)g'(u)}{g^2(u)}$$

# Chain Rule

---

The chain rule allows you to combine any of the differentiation rules we have already covered.

- First, do the derivative of the outside and then do the derivative of the inside.

$$\frac{d}{du} f(g(u)) = f'(g(u)) * g'(u) * du$$

# Try These

---

$$f(z) = z + 11$$

$$s(y) = 4ye^{2y}$$

$$g(y) = 4y^3 + 2y$$

$$p(x) = \frac{\log(x^2)}{x}$$

$$h(x) = e^{3x}$$

# Solutions

---

$$f'(z) = 1$$

$$s'(y) = 8ye^{2y} + 4e^{2y}$$

$$g'(y) = 12y^2 + 2$$

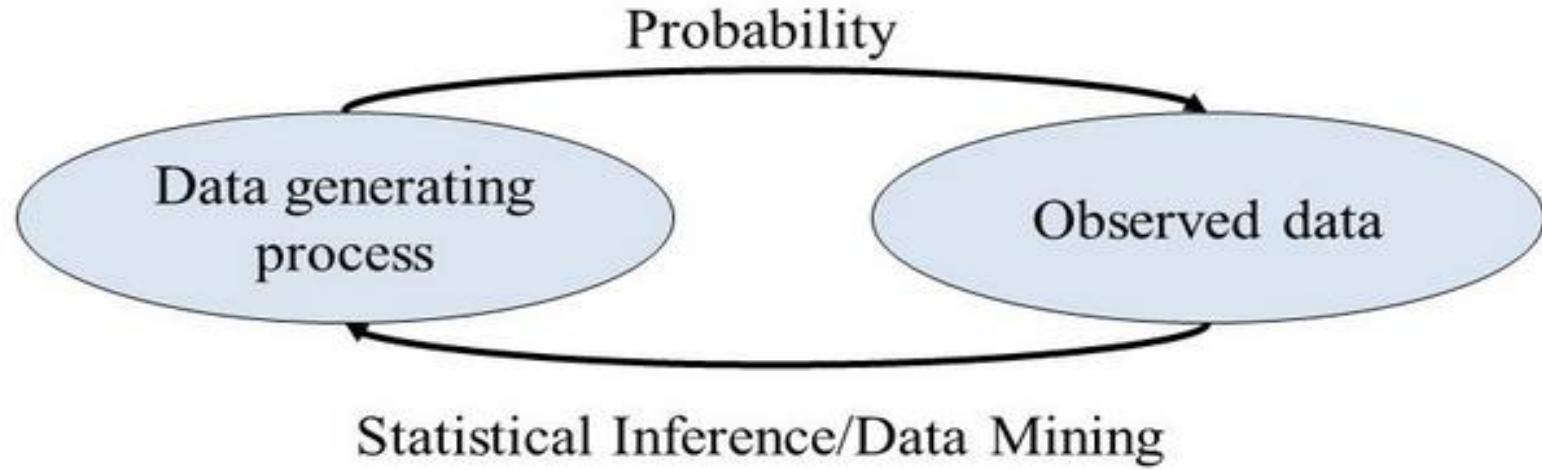
$$p'(x) = \frac{2 - \log(x^2)}{x^2}$$

$$h'(x) = 3e^{3x}$$

---

# Probability Review

# Probability Theory

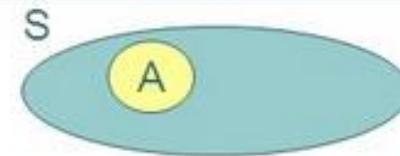


- **Probability Theory**
  - Given a data generating process, what are the properties of the outcome?
- **Statistical Inference**
  - Given the outcome, what can we say about the process that generated the data?
  - How can we generalize these observations and make predictions about future outcomes?

# Probability Theory

Let  $A$  be an event, then we denote

$P(A)$  the probability for  $A$



It always hold that  $0 \leq P(A) \leq 1$      $P(\emptyset) = 0$      $P(S) = 1$

Consider an experiment which has  $N$  **equally likely** outcomes, and let exactly  $n$  of these events correspond to the event  $A$ . Then

$$P(A) = \frac{n}{N} = \frac{\text{\# successful outcomes}}{\text{\# possible outcomes}}$$

**Example:**  
Rolling a dice  
 $P(\text{even number})$

$$= \frac{3}{6} = \frac{1}{2}$$

# Random Variable

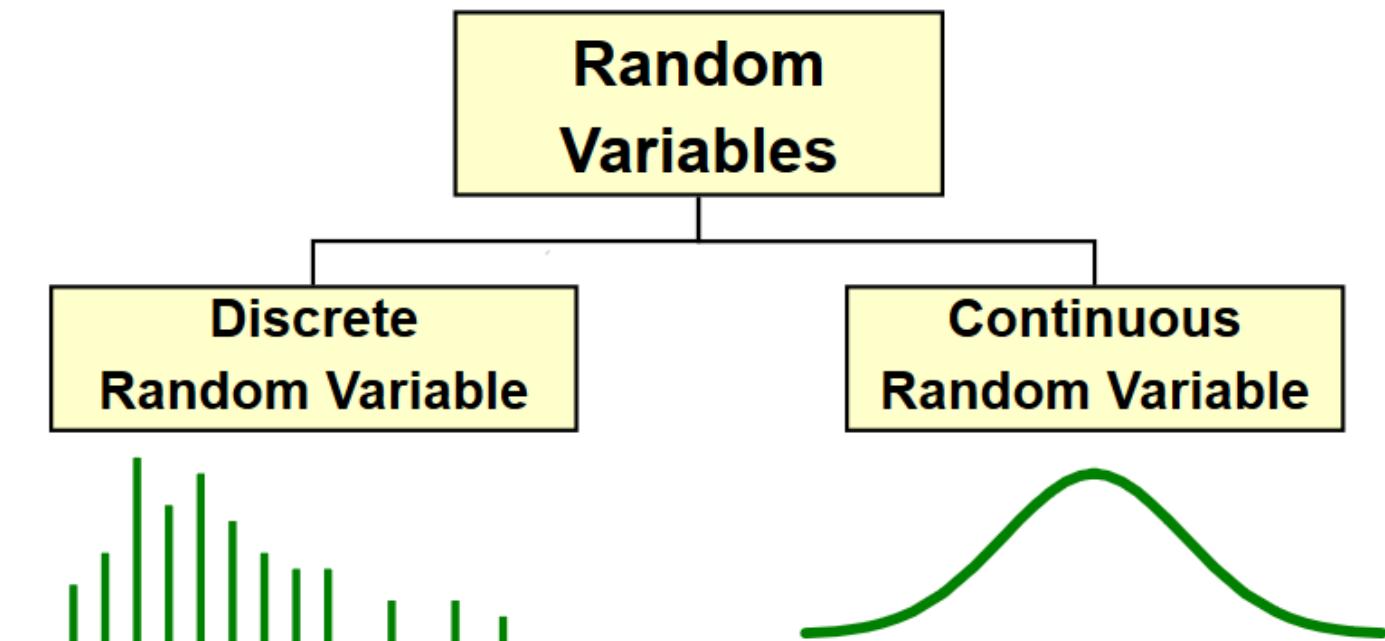
- A **random variable**, usually written  $X$ , is a **variable** whose possible values are numerical outcomes of a **random** phenomenon or experiment.

## Examples

- ✓  $X$  = number of heads when the experiment is flipping a coin 20 times.
- ✓  $C$  = the daily change in a stock price.
- ✓  $R$  = the number of kilometers per litre you get on your car during a family vacation.

# Random Variable

Represents a possible numerical value from a random event



# Random Variable

## Discrete Random Variable

- one that takes on a ***countable*** number of values
- usually count data [Number of]
- list **all** possible outcomes without missing any of them

### Example:

- ✓  $X$  = sum of values on the roll of two dice:  
 $X$  has to be either 2, 3, 4, ..., or 12.
- ✓  $Y$  = number of students in MTech DSE:  
 $Y$  has to be 60,65,70 .....

# Random Variable

## Continuous Random Variable

- Variable that takes on an uncountable number of values
- Usually measurement data [time, weight, distance, etc]
- You can never list all possible outcomes even if you had an infinite amount of time

### Example:

$X$  = time it takes you to drive home from work place:  $X > 0$ , might be 30.1 minutes measured to the nearest tenth but in reality the actual time is 30.1000001..... minutes?)

Exercise: try to list all possible numbers between 0 and 1

# Probability Theory

---

## Notation

- A **random variable  $X$**  represents outcomes or states of the world.
- We will write  $p(x)$  to mean  $\text{Probability}(X = x)$
- **Sample space:** the space of all possible outcomes (may be discrete, continuous, or mixed)
- $p(x)$  is the **probability mass (density) function**
  - Assigns a number to each point in sample space
  - Non-negative, sums (integrates) to 1
  - Intuitively: how often does  $x$  occur, how much do we believe in  $x$ .

# Probability Theory

---

## Joint Probability Distribution

- $\text{Prob}(X=x, Y=y)$ 
  - “Probability of  $X=x$  and  $Y=y$ ”
  - $p(x, y)$

## Conditional Probability Distribution

- $\text{Prob}(X=x | Y=y)$ 
  - “Probability of  $X=x$  given  $Y=y$ ”
  - $p(x|y) = p(x,y)/p(y)$

# Probability Theory

---

## The Rules of Probability

- Sum Rule (marginalization/summing out):

$$p(x) = \sum_y p(x, y)$$

$$p(x_1) = \sum_{x_2} \sum_{x_3} \dots \sum_{x_N} p(x_1, x_2, \dots, x_N)$$

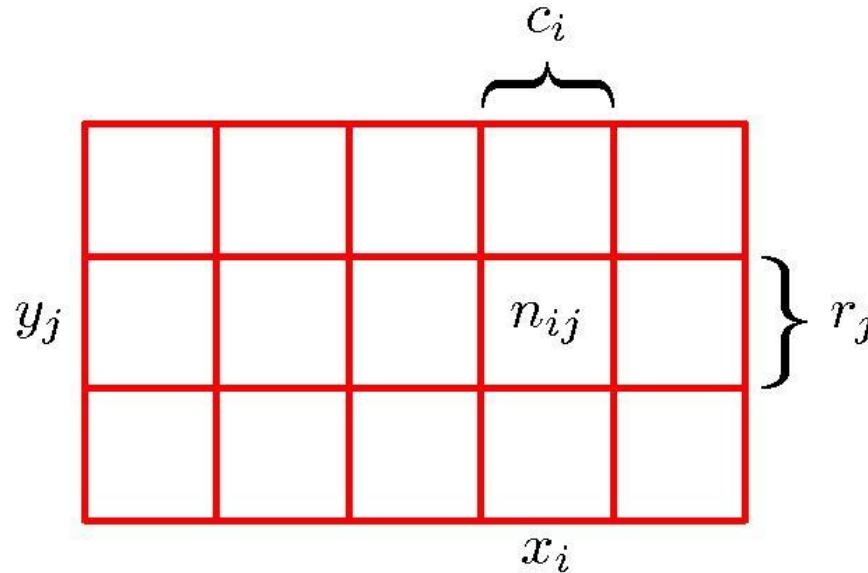
- Product/Chain Rule:

$$p(x, y) = p(y | x)p(x)$$

$$p(x_1, \dots, x_N) = p(x_1)p(x_2 | x_1)\dots p(x_N | x_1, \dots, x_{N-1})$$

# Probability Theory

- X and Y are two discrete random variables.
- X can take any of the values  $x_i$ ,  $i = 1, \dots, M$ , and Y can take the values  $y_j$ ,  $j = 1, \dots, L$ .
- Let a total of N trials in which we sample both of the variables X and Y, and let the number of such trials in which  $X = x_i$  and  $Y = y_j$  be  $n_{ij}$ .
- Let the number of trials in which X takes the value  $x_i$  be denoted by  $c_i$ , and similarly let the number of trials in which Y takes the value  $y_j$  be denoted by  $r_j$ .



# Probability Theory

				$c_i$
$y_j$			$n_{ij}$	
				$x_i$

$\left. \right\} r_j$

Marginal Probability

$$p(X = x_i) = \frac{c_i}{N}.$$

Joint Probability

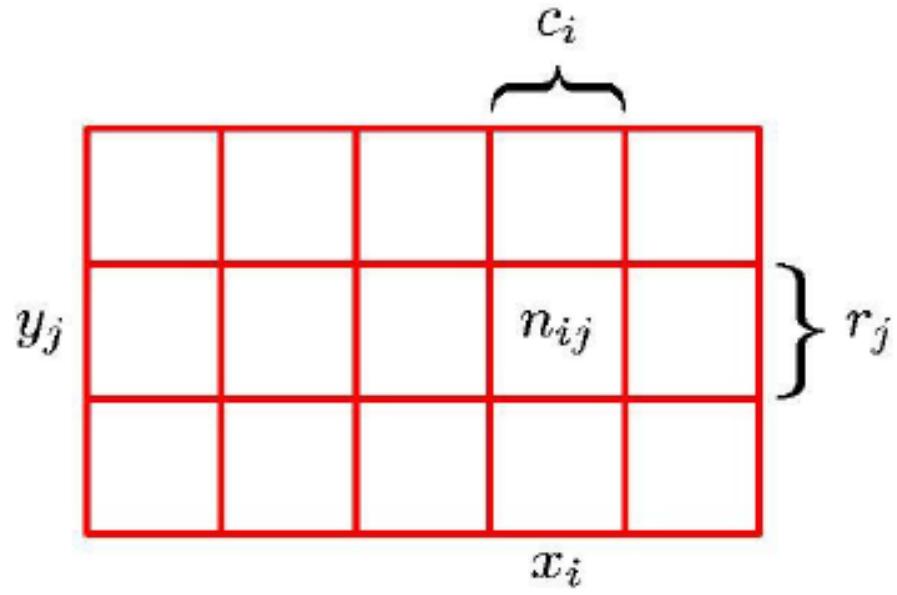
$$p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$$

Conditional Probability

$$p(Y = y_j | X = x_i) = \frac{n_{ij}}{c_i}$$

Here we are implicitly considering the limit  $N \rightarrow \infty$ .

# Probability Theory



**Sum Rule**

$$p(X = x_i) = \frac{c_i}{N} = \frac{1}{N} \sum_{j=1}^L n_{ij}$$

$$= \sum_{j=1}^L p(X = x_i, Y = y_j)$$

**Product Rule**

$$\begin{aligned} p(X = x_i, Y = y_j) &= \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N} \\ &= p(Y = y_j | X = x_i) p(X = x_i) \end{aligned}$$

# Probability Theory

from the definition of conditional distributions:

$$P(A|B)P(B) = P(A, B) = P(B|A)P(A)$$

Hence:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

is known as **Bayes rule**.

example:

$$P(\text{"taking a shower"} | \text{"wet"}) = P(\text{"wet"} | \text{"taking a shower"}) \frac{P(\text{"taking a shower"})}{P(\text{"wet"})}$$

$$P(\text{reason} | \text{observation}) = P(\text{observation} | \text{reason}) \frac{P(\text{reason})}{P(\text{observation})}$$

# Probability Theory



## Bayes rule - Example

if patient has meningitis, then very often a stiff neck is observed

if patient has meningitis, then very often a stiff neck is observed

$$P(S|M) = 0.8 \text{ (can be easily determined by counting)}$$

observation: 'I have a stiff neck! Do I have meningitis?' (is it reasonable to be afraid?)

# Probability Theory

if patient has meningitis, then very often a stiff neck is observed

$$P(S|M) = 0.8 \text{ (can be easily determined by counting)}$$

observation: 'I have a stiff neck! Do I have meningitis?' (is it reasonable to be afraid?)

$$P(M|S) = ?$$

we need to know:  $P(M) = 0.0001$  (one of 10000 people has meningitis)

and  $P(S) = 0.1$  (one out of 10 people has a stiff neck).

then:

$$P(M|S) = \frac{P(S|M)P(M)}{P(S)} = \frac{0.8 \times 0.0001}{0.1} = 0.0008$$

Keep cool. Not very likely

# Probability Densities

---

## Continuous Probability Distribution

Let  $X$  be a continuous rv. Then a *probability distribution or probability density function (pdf)* of  $X$  is a function  $f(x)$  such that for any two numbers  $a$  and  $b$ ,

$$P(a \leq X \leq b) = \int_a^b f(x) dx$$

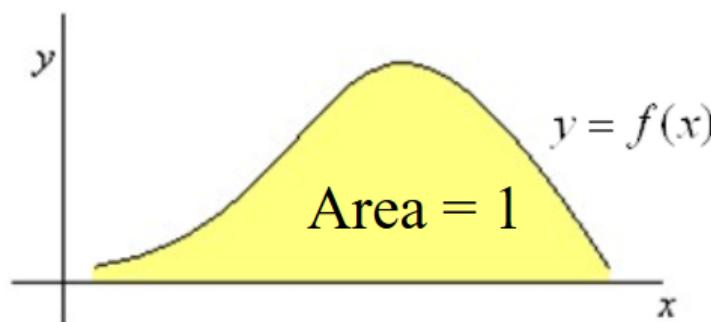
The graph of  $f$  is the *density curve*.

# Probability Densities

## Probability Density Function

For  $f(x)$  to be a pdf

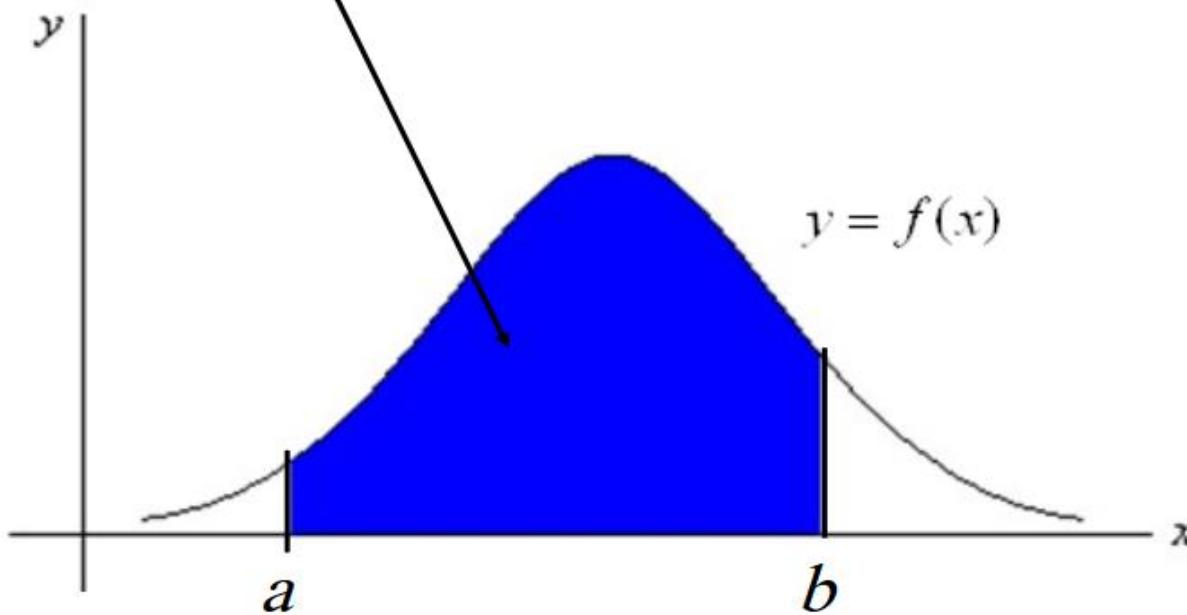
1.  $f(x) > 0$  for all values of  $x$ .
2. The area of the region between the graph of  $f$  and the  $x$  – axis is equal to 1.



# Probability Densities

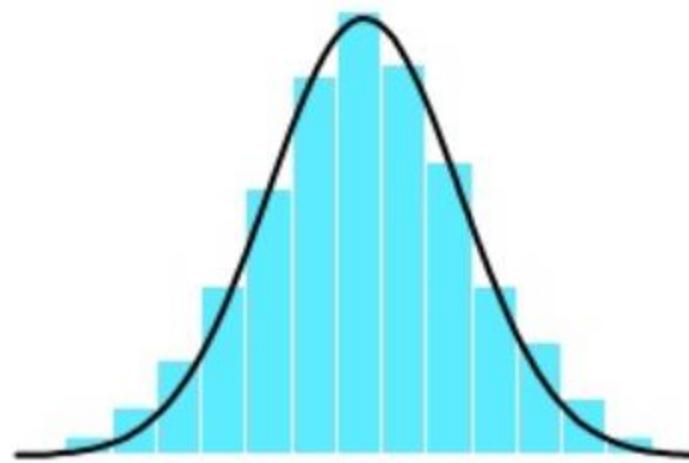
## Probability Density Function

$P(a \leq X \leq b)$  is given by the area of the shaded region.



# Gaussian Distribution

- The commonest and the most useful continuous distribution.
- A symmetrical probability distribution where most results are located in the middle and few are spread on both sides.
- It has the shape of a bell.
- Can entirely be described by its mean and standard deviation.



# Gaussian Distribution

---

## Examples:

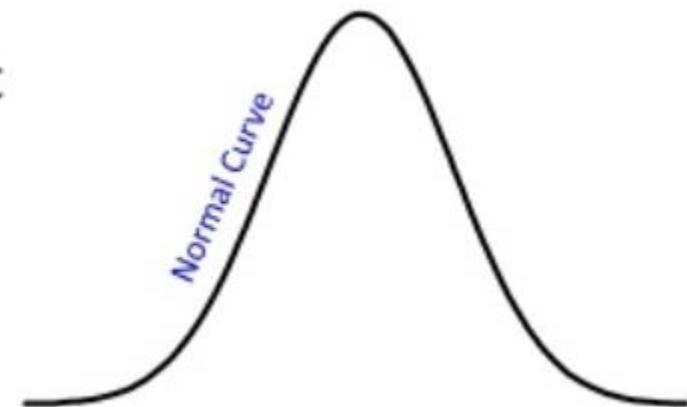
- The body temperature for healthy humans.
- The heights and weights of adults.
- The thickness and dimensions of a product.
- IQ and standardized test scores.
- Quality control test results.
- Errors in measurements.



# Gaussian Distribution

## Normal Curve:

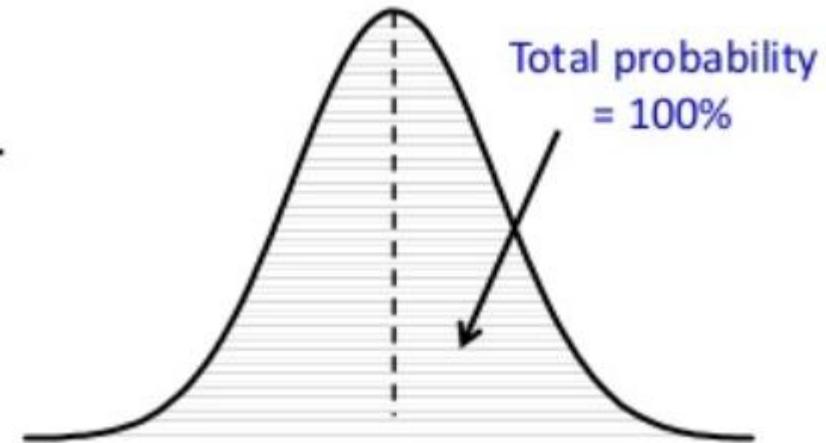
- A graphical representation of the normal distribution.
- It is determined by the mean and the standard deviation.
- It is a symmetric unimodal bell-shaped curve.
- Its tails extend infinitely in both directions.
- The wider the curve, the larger the standard deviation and the more variation exists in the process.
- The spread of the curve is equivalent to six times the standard deviation of the process.



# Gaussian Distribution

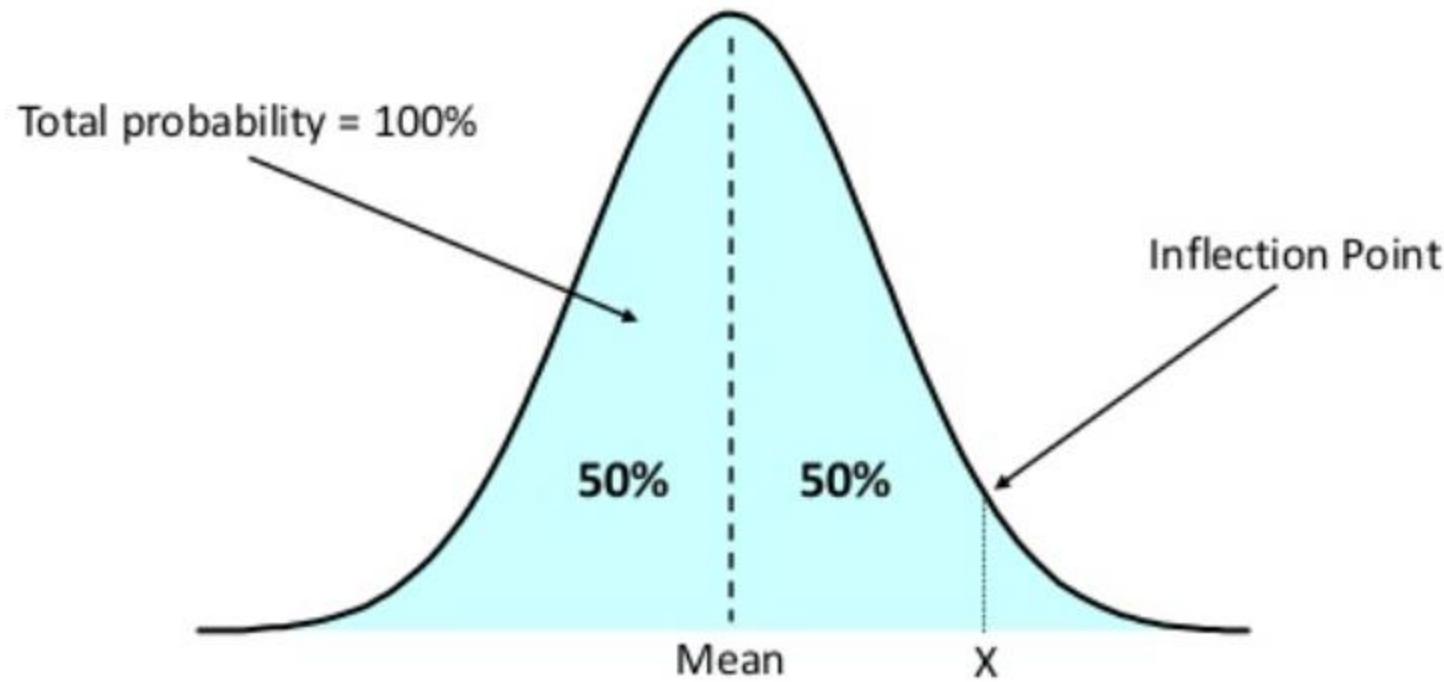
Normal Curve:

- ❑ Helps calculating the probabilities for normally distributed populations.
- ❑ The probabilities are represented by the area under the normal curve.
- ❑ The total area under the curve is equal to **100%** (or **1.00**).
- ❑ This represents the population of the observations.
- ❑ We can get a rough estimate of the probability above a value, below a value, or between any two values.



# Gaussian Distribution

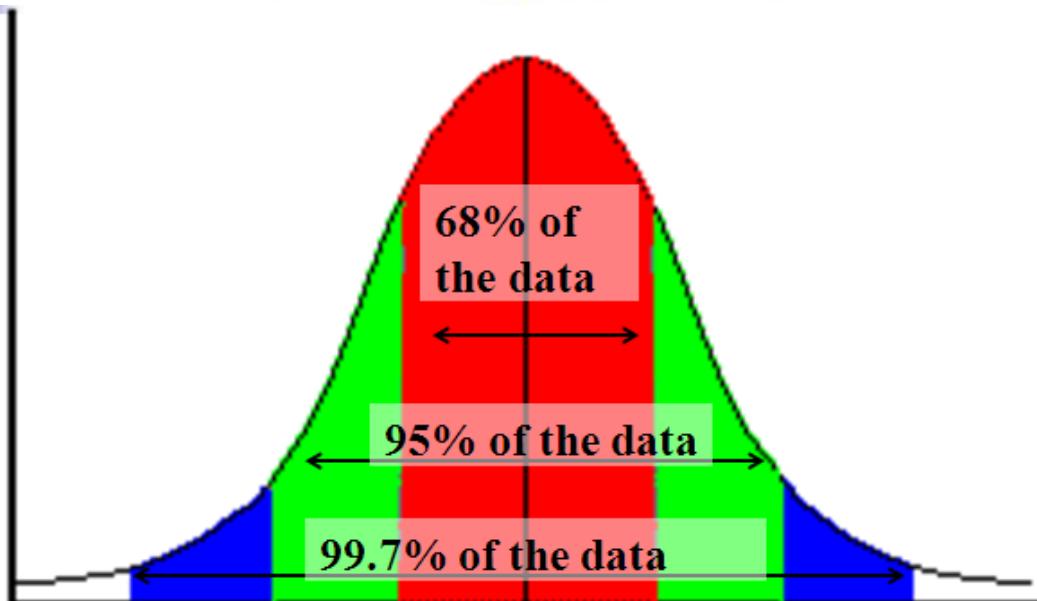
- Since the normal curve is symmetrical, **50 percent** of the data lie on each side of the curve.



# Gaussian Distribution

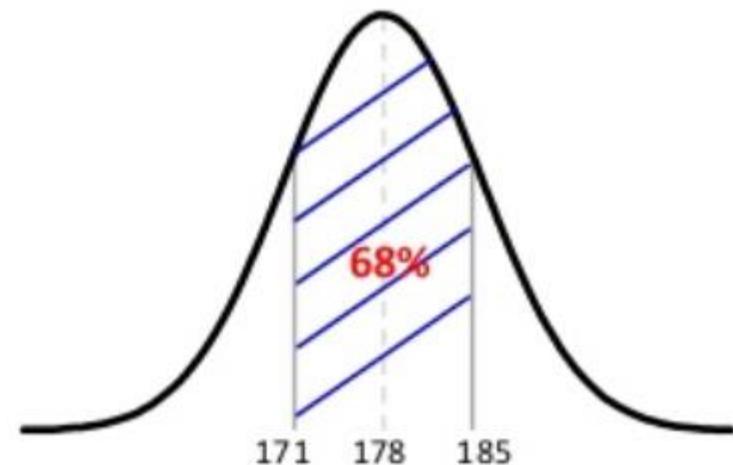
## Empirical Rule:

- For any normally distributed data:
  - **68%** of the data fall within **1** standard deviation of the mean.
  - **95%** of the data fall within **2** standard deviations of the mean.
  - **99.7%** of the data fall within **3** standard deviations of the mean.



# Gaussian Distribution

- Suppose that the heights of a sample men are normally distributed.
- The mean height is **178** cm and a standard deviation is **7** cm.
  
- **We can generalize that:**
  - **68%** of population are between **171** cm and **185** cm.
  - This might be a generalization, but it's true if the data is normally distributed.



# Mean, Variance & Standard Deviation

---

- ✓ The mean of a discrete random variable is the ***weighted average*** of all of its values. The weights are the probabilities.
- ✓ This parameter is also called the expected value of X and is represented by  $E(X)$ .

$$E(X) = \mu = \sum_{all \ x} xP(x)$$

- ✓ The variance is

$$V(X) = \sigma^2 = \sum_{all \ x} (x - \mu)^2 P(x)$$

- ✓ The standard deviation is

$$\sigma = \sqrt{\sigma^2}$$

# Gaussian Distribution

## Mean ( $\mu$ )

- \* Mean : Measure of Central tendency
- \* Center or middle of data set around which observations are lying
- \* Assuming : frequency in each class is uniformly distributed and representable by mid point
- \* Mean for grouped data is given by

$$\bar{X} = \frac{1}{n} [ \sum f_i * x_i ] \text{ where } n = \text{no of observations}$$

$f_i$  = frequency of each (ith) class interval

$x_i$  = mid point of each class interval

# Gaussian Distribution

## Standard Deviation $\sigma$

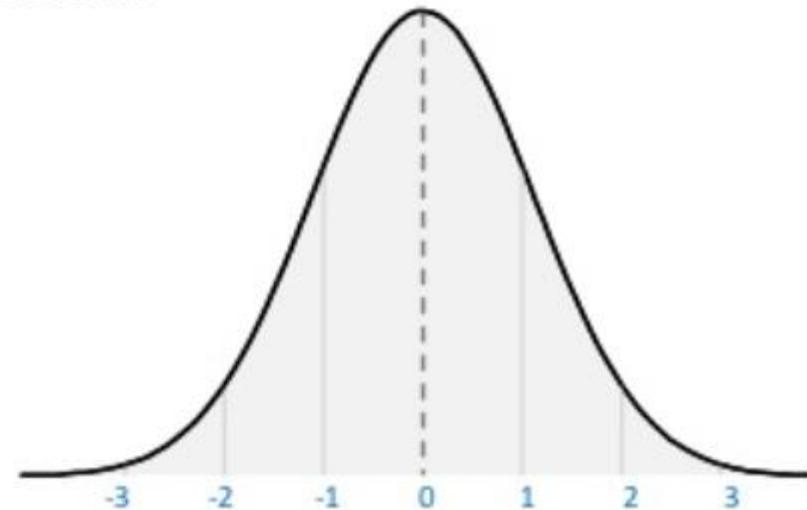
- \* Standard Deviation : Measure of Dispersion
- \* Average deviation of observations around the mean
- \* Compactness or variation of data
- \* SD = root mean square deviation
- \*  $SD = \sqrt{\text{variance}} = \sqrt{\frac{1}{n} \sum (x_i - \bar{X})^2}$
- \* where n = no of observations

$\bar{X}$  = mean of the frequency distribution  
 $x_i$  = mid point of each class interval

# Gaussian Distribution

## Standard Normal Distribution:

- A common practice to convert any normal distribution to the standardized form and then use the standard normal table to find probabilities.
- The **Standard Normal Distribution** (Z distribution) is a way of standardizing the normal distribution.
- It always has a mean of **0** and a standard deviation of **1**.



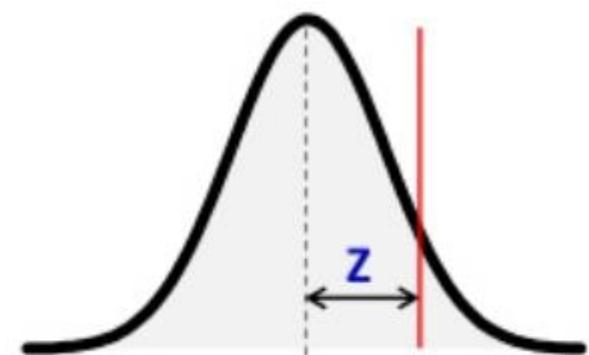
# Gaussian Distribution

Standard Normal Distribution:

- Any normally distributed data can be converted to the standardized form using the formula:

$$Z = (X - \mu) / \sigma$$

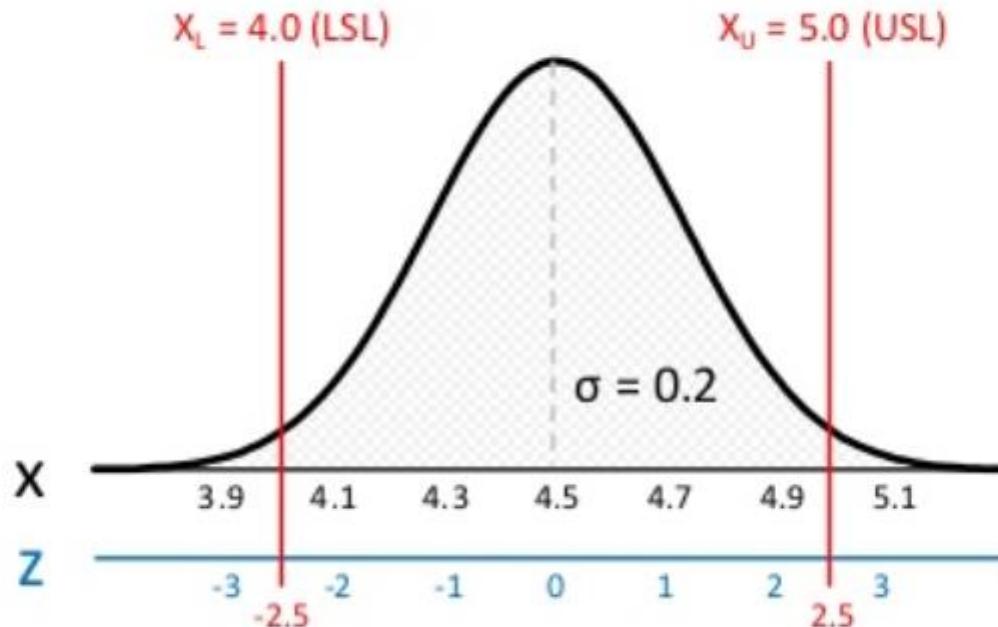
- where:
  - 'X' is the data point in question.
  - 'Z' (or **Z-score**) is a measure of the number of standard deviations of that data point from the mean.



# Gaussian Distribution

Standard Normal Distribution:

- Converting from 'X' to 'Z':



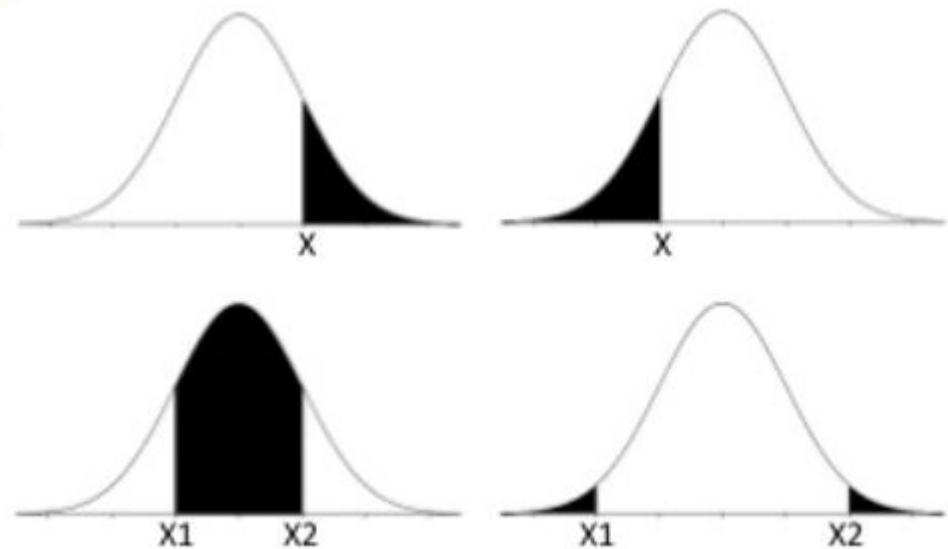
The specification limits at 4.0 and 5.0mm respectively, and the corresponding z values

The X-scale is for the actual values and the Z-scale is for the standardized values

# Gaussian Distribution

Standard Normal Distribution:

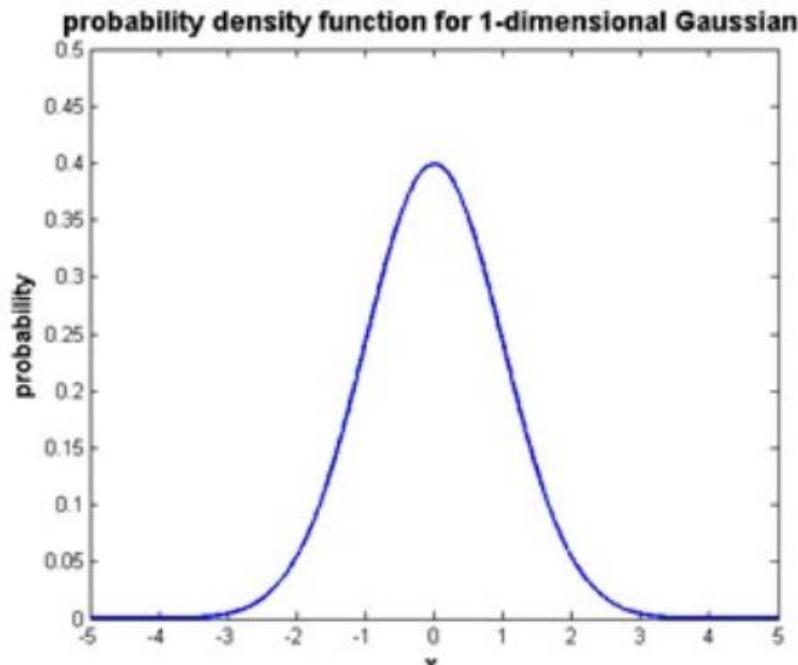
- You can then use this information to determine the area under the normal distribution curve that is:
  - To the right of your data point.
  - To the left of the data point.
  - Between two data points.
  - Outside of two data points.



# Gaussian Distribution

In one dimension

$$N(x | \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



# Gaussian Distribution

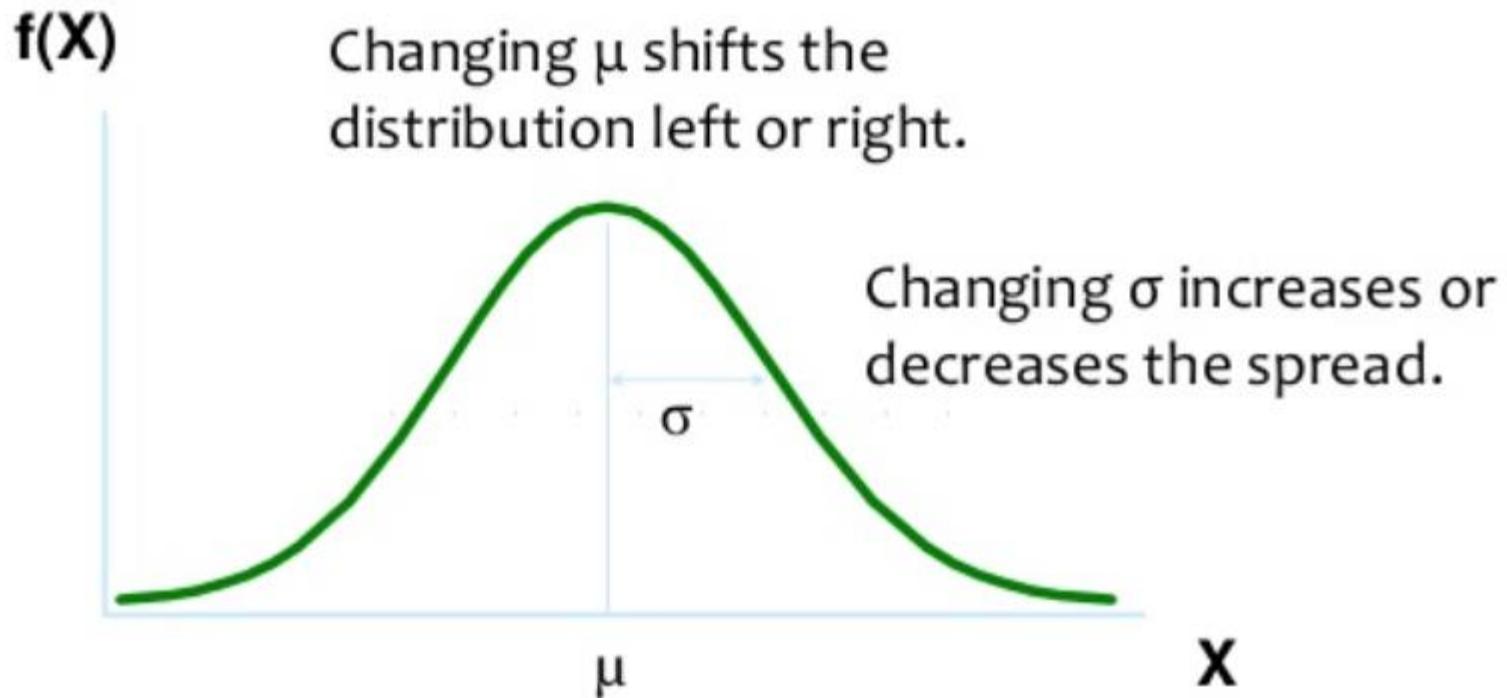
In one dimension

Causes pdf to decrease as distance from center increases

$$N(x | \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

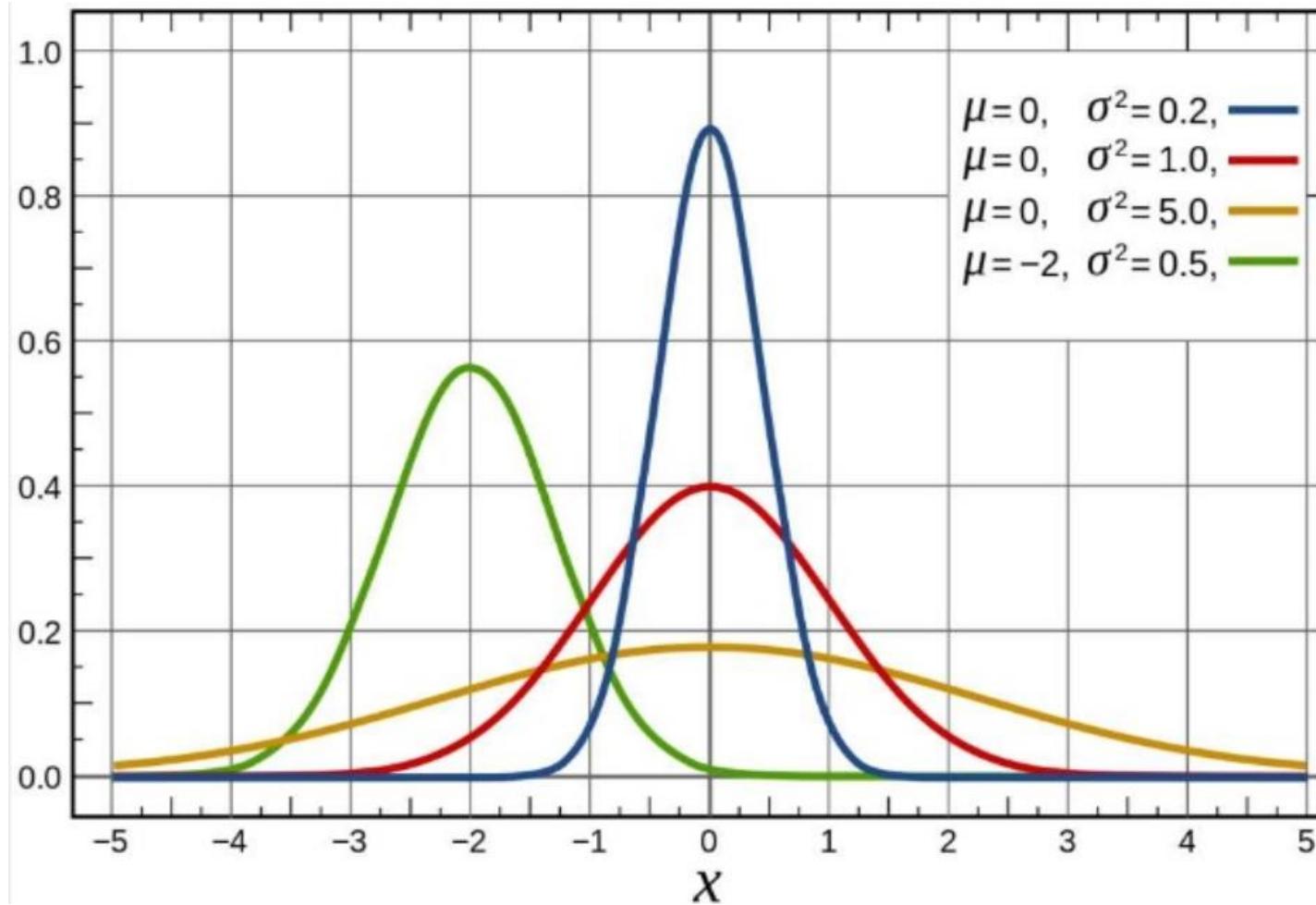
Normalizing constant:  
insures that distribution  
integrates to 1

# Gaussian Distribution



The normal curve is not a single curve but a family of curves, each of which is determined by its mean and standard deviation.

# Gaussian Distribution



# Multivariate Gaussian Distribution

In  $d$  dimensions

$$N(\mathbf{x} | \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}} \frac{1}{|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})}$$

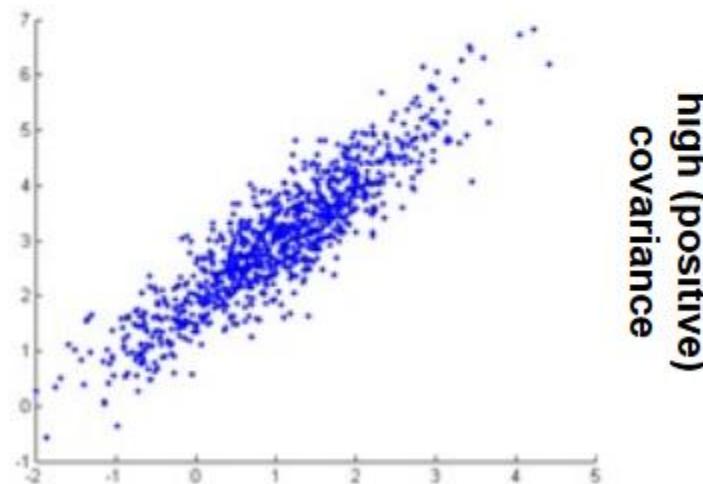
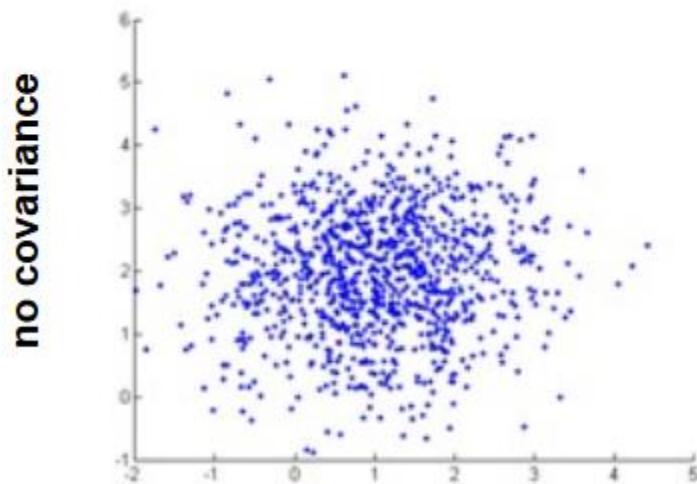
$\mathbf{x}$  and  $\boldsymbol{\mu}$  now  $d$ -dimensional vectors

- $\boldsymbol{\mu}$  gives center of distribution in  $d$ -dimensional space
- $\sigma^2$  replaced by  $\Sigma$ , the  $d \times d$  covariance matrix
- $\Sigma$  contains pairwise covariances of every pair of features
- Diagonal elements of  $\Sigma$  are variances  $\sigma^2$  of individual features
- $\Sigma$  describes distribution's shape and spread

# Multivariate Gaussian Distribution

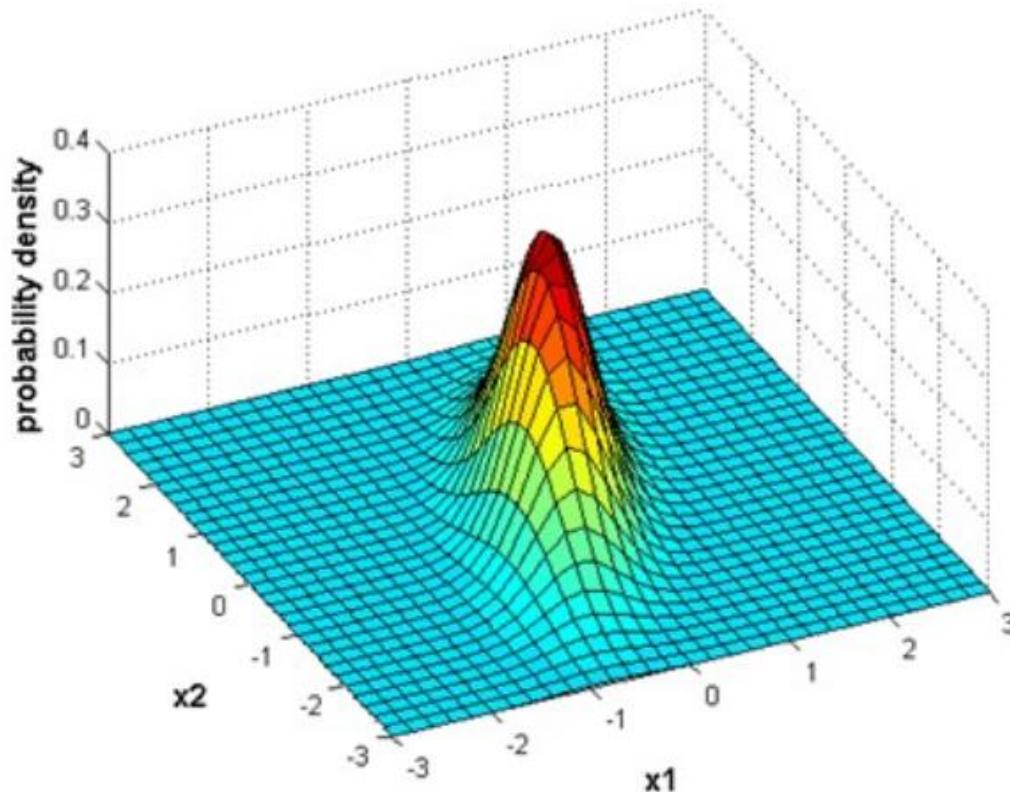
## Covariance

- Measures tendency for two variables to deviate from their means in same (or opposite) directions at same time



# Multivariate Gaussian Distribution

In two dimensions



$$\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 0.25 & 0.3 \\ 0.3 & 1 \end{bmatrix}$$

---

# Decision Theory

# Decision Theory

---

- Suppose  $x$  is an input vector together with a corresponding vector  $t$  of target variables
- Goal: predict  $t$  given a new value for  $x$ .
- The joint probability distribution  $p(x, t)$  provides a complete summary of the uncertainty associated with these variables.
- Determination of  $p(x, t)$  from a set of training data is called ***inference*** and is a difficult problem.

# Decision Theory

---

Inference step

Determine either  $p(t|\mathbf{x})$  or  $p(\mathbf{x}, t)$ .

Decision step

For given  $\mathbf{x}$ , determine optimal  $t$ .

# Example : Medical diagnosis problem

Input: X-ray image of a patient

Input vector  $x$  is the set of pixel intensities in the image

Output: Presence of cancer = Class C1,

Absence of cancer, = Class C2.

Choose  $t$  to be a binary variable such that

$t = 0$  corresponds to C1 and  $t = 1$  corresponds to C2.

We are interested in the probabilities of the two classes given the image, which are given by  $p(C_k|x)$ .

Using Bayes' theorem,

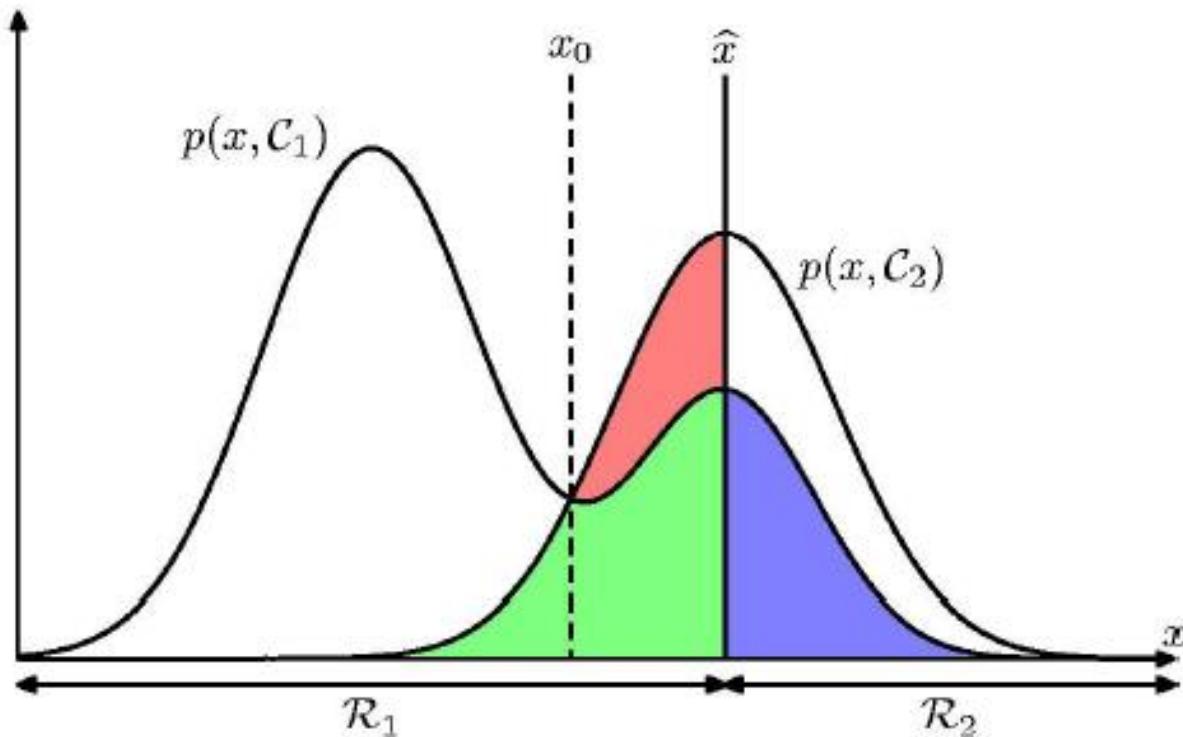
$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}$$

# Minimum Misclassification Rate

---

- Divide the input space into regions  $R_k$  called decision regions, one for each class, such that all points in  $R_k$  are assigned to class  $C_k$
- Boundaries between decision regions are called decision boundaries or decision surfaces
- A mistake occurs when an input vector belonging to class  $C_1$  is assigned to class  $C_2$  or vice versa.

# Minimum Misclassification Rate



$$\begin{aligned}
 p(\text{mistake}) &= p(\mathbf{x} \in \mathcal{R}_1, \mathcal{C}_2) + p(\mathbf{x} \in \mathcal{R}_2, \mathcal{C}_1) \\
 &= \int_{\mathcal{R}_1} p(\mathbf{x}, \mathcal{C}_2) d\mathbf{x} + \int_{\mathcal{R}_2} p(\mathbf{x}, \mathcal{C}_1) d\mathbf{x}.
 \end{aligned}$$

# Minimum Misclassification Rate

---

$$\begin{aligned} p(\text{correct}) &= \sum_{k=1}^K p(\mathbf{x} \in \mathcal{R}_k, \mathcal{C}_k) \\ &= \sum_{k=1}^K \int_{\mathcal{R}_k} p(\mathbf{x}, \mathcal{C}_k) d\mathbf{x} \end{aligned}$$

# Inference and Decision

---

- We have broken the **classification problem** down into two separate stages, 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.
- 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***.
- **Three distinct approaches to solving decision problems**

# Inference and Decision

---

## 1<sup>st</sup> approach

- Determine the class-conditional densities  $p(\mathbf{x} | C_k)$  for each class  $C_k$  individually.
- Separately infer the prior class probabilities  $p(C_k)$ . Then use Bayes' theorem

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

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

# Inference and Decision

---

## 2<sup>nd</sup> approach

- 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***.

# Inference and Decision

---

## 3<sup>rd</sup> approach

- Find a function  $f(x)$ , called a ***discriminant function***, which maps each input  $x$  directly onto a class label.
- Example - For 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.

# Inference and Decision

---

- 1<sup>st</sup> approach is the most demanding because it involves finding  $p(x, C_k)$ .
- For many applications,  $x$  will have high dimensionality, and consequently we may need a large training set in order to be able to determine the class-conditional densities to reasonable accuracy.
- The class priors  $p(C_k)$  can often be estimated simply from the fractions of the training set data points in each of the classes.

# Inference and Decision

---

- One advantage of 1<sup>st</sup> approach, however, is that it also allows  $p(x)$  to be determined. This can be useful for detecting new data points that have low probability under the model and for which the predictions may be of low accuracy, which is known as ***outlier detection*** or ***novelty detection***.

$$p(x) = \sum_k p(x|\mathcal{C}_k)p(\mathcal{C}_k)$$

# Inference and Decision

---

- If we only wish to make classification decisions, then it can be wasteful of computational resources, and excessively demanding of data, to find  $p(x, C_k)$  when in fact we only really need the posterior probabilities  $p(C_k/x)$ , which can be obtained directly through 2<sup>nd</sup> approach.

# Inference and Decision

---

- The 3<sup>rd</sup> approach is simple, where the data is used to find a ***discriminant function***  $f(x)$  that maps each  $x$  directly onto a class label, thereby combining the inference and decision stages into a single learning problem.



# Information Theory

# Measure of Information

---

- It rained heavily in Shillong yesterday
- There was a heavy rainfall in Rajasthan last night.
- The amount of information conveyed by a message is inversely proportional to its probability of occurrence.  
That is

$$I_k \propto \frac{1}{p_k}$$

# Measure of Information

---

- The information conveyed by a message cannot be negative. It has to be at least 0, i.e.,

$$I_k \geq 0$$

- If  $p_k = 1$ , then  $I_k = 0$ .
- The information conveyed composite statement which is independent is simply given by the sum of the individual self-information contents, i.e.,

$$I(m_1, m_2) = I(m_1) + I(m_2)$$

# Measure of Information

---

- The only mathematical operator satisfies above properties is the logarithmic operator. Therefore,

$$I_k = \log_r \frac{1}{p_k} \text{ units}$$

- If  $r = 2$ , unit is bits

# Average Information Content (Entropy)

---

- Consider a source  $x = \{x_1, x_2, \dots, x_N\}$ , with probabilities  $P = \{p_1, p_2, \dots, p_N\}$
- A message of length  $L$  emitted by the source, then it contains
  - $p_1 \times L$  number of symbol  $x_1$
  - $p_2 \times L$  number of symbol  $x_2, \dots,$
  - $p_N \times L$  number of symbol  $x_N$

# Average Information Content (Entropy)

---

- The self-information of  $x_1$  is

$$I_{x_1} = \log_2 \frac{1}{p_1} \text{ bits}$$

- Each symbol  $x_1$  conveys information of  $\log_2(1/p_1)$  bits and such  $(p_1 \times L)$  number of  $x_1$  symbols are present on an average in a length of  $L$  symbols.
- The total information conveyed by symbols of type  $x_i$  is

$$p_i \times L \times \log_2 \frac{1}{p_i} \text{ bits}$$

# Average Information Content (Entropy)

---

- The total information conveyed by the source is

$$I_T = p_1 \times L \times \log_2 \frac{1}{p_1} + p_2 \times L \times \log_2 \frac{1}{p_2} + \cdots + p_N \times L \times \log_2 \frac{1}{p_N}$$

- The average information conveyed by the source by emitting L symbols is denoted by its entropy  $H[x]$

$$H[x] = \frac{I_T}{L} = p_1 \times \log_2 \frac{1}{p_1} + p_2 \times \log_2 \frac{1}{p_2} + \cdots + p_N \times \log_2 \frac{1}{p_N}$$

# Entropy

---

$$H[x] = - \sum_x p(x) \log_2 p(x)$$

Important quantity in

- coding theory
- statistical physics
- machine learning

# Entropy

---

Consider a discrete random variable  $x$  with 8 possible states, each of which is equally likely,

How many bits to transmit the state of  $x$ ?

$$H[x] = -8 \times \frac{1}{8} \log_2 \frac{1}{8} = 3 \text{ bits.}$$

# Entropy

$x$	a	b	c	d	e	f	g	h
$p(x)$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{64}$	$\frac{1}{64}$	$\frac{1}{64}$	$\frac{1}{64}$
code	0	10	110	1110	111100	111101	111110	111111

$$\begin{aligned}
 H[x] &= -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{4} \log_2 \frac{1}{4} - \frac{1}{8} \log_2 \frac{1}{8} - \frac{1}{16} \log_2 \frac{1}{16} - \frac{4}{64} \log_2 \frac{1}{64} \\
 &= 2 \text{ bits}
 \end{aligned}$$

$$\begin{aligned}
 \text{average code length} &= \frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{8} \times 3 + \frac{1}{16} \times 4 + 4 \times \frac{1}{64} \times 6 \\
 &= 2 \text{ bits}
 \end{aligned}$$

# Conditional Entropy

---

- Suppose we have a joint distribution  $p(x, y)$  from which we draw  $x$  and  $y$ .
- If  $x$  is already known, then the additional information needed to specify  $y$  is given by  $-\ln p(y|x)$ .
- Thus the average additional information needed to specify  $y$  can be written as

$$H[y|x] = - \iint p(y, x) \ln p(y|x) dy dx$$

$$H[x, y] = H[y|x] + H[x]$$

---

# Mutual Information

---

The mutual information of two random variables is a measure of the variables' mutual dependence.

It determines how similar the joint distribution  $p(X,Y)$  is to the products of factored marginal distribution  $p(X)p(Y)$ .

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left( \frac{p(x,y)}{p(x)p(y)} \right),$$

$$I(X;Y) = \int_Y \int_X p(x,y) \log \left( \frac{p(x,y)}{p(x)p(y)} \right) dx dy,$$

$$I[\mathbf{x},\mathbf{y}] = H[\mathbf{x}] - H[\mathbf{x}|\mathbf{y}] = H[\mathbf{y}] - H[\mathbf{y}|\mathbf{x}]$$



# Thank You



**BITS** Pilani  
Pilani Campus

# Machine Learning DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



BITS Pilani, Pilani Campus



**BITS Pilani**  
Pilani Campus



**Lecture No. – 3 | Bayesian Learning**

**Date – 09/11/2019**

**Time – 9:00 AM – 11:00 AM**



# Session Content

---

## Bayesian learning

- Basics (T1 book by Tom Mitchell - 6.1)
- Bayes Theorem (T1 book by Tom Mitchell - 6.2)
- MAP Hypothesis (T1 book by Tom Mitchell - 6.3)
- MLE Hypothesis (T1 book by Tom Mitchell - 6.4)
- Minimum Description Length (MDL) principle  
(T1 book by Tom Mitchell - 6.6)

# Probability Theory

- Classical Definition
  - Consider an experiment which has N equally likely outcomes, and let n of these outcomes correspond to a specific event A, then
  - $P(A) = \frac{\# \text{ successful outcomes}}{\# \text{ possible outcomes}} = n / N$

$$\Omega = \{ \text{ } \} \quad \text{Coin toss}$$

$$\Omega = \{ \text{ } \} \quad \text{Die toss}$$

# AXIOMS of Probability Theory

---

- A probability  $p(\omega)$  for each outcome  $\omega$  must satisfy the following axioms

$$p(\omega) \geq 0, \quad \sum_{\omega \in \Omega} p(\omega) = 1$$

E.g.,  $p(\text{heads}) = .6$

$p(\text{tails}) = .4$



# Random Variable

## Notation

- A **random variable  $X$**  represents outcomes or states of the world.
- We will write  $p(x)$  to mean  $\text{Probability}(X = x)$
- **Sample space:** the space of all possible outcomes (may be discrete, continuous, or mixed)
- $p(x)$  is the **probability mass (density) function**
  - Assigns a number to each point in sample space
  - Non-negative, sums (integrates) to 1
  - Intuitively: how often does  $x$  occur, how much do we believe in  $x$ .



# Probability Distributions

---

- The outcomes for random variables and their associated probabilities can be organized in to distributions
- Two types of distributions based on types of Random variables: Discrete and Continuous
- Discrete:
  - Binomial, Poisson, Geometric distributions
- Continuous
  - Gaussian, exponential, t, F, chi-squared distibutions



# Describing distributions

---

- One way is to construct a graph and analyze the graph to make inferences
  - Discrete: Prob Mass Function (pmf), Cumulative density function
  - Continuous: prob density function (pdf)
- Mean, variance and standard deviations to represent the entire distribution



# Bernoulli Distribution

---

- A r.v.  $X$  is said to follow Bernoulli's distribution when there are only two possible outcomes
  - By convention either Success (1) or Failure (0)
  - And there is only one trial
  - Ex: tossing a coin at the start f the match
- Let  $p$  represents the probability of success and  $(1-p)$  represent the probability of failure, then the probability mass function is defined as

$$f(x) = \begin{cases} p & \text{if } x=1 \\ 1-p & x=0 \end{cases} \quad p^x (1-p)^{(1-x)}$$



# Binomial Distribution

- Again binary outcomes, but for n independent trials
  - Probability of success ( $p$ ) remains the same for all the trials
  - Probability of r success is given by
$$P(X=r) = {}^nC_r p^r (1-p)^{n-r}$$

$$\text{Mean } E(X) = np$$

$$\text{Variance } \text{Var}(X) = npq \quad (\text{where } q=1-p)$$



# Estimate Probabilities from Data

---

## I For continuous attributes:

- **Probability density estimation:**
  - ◆ Assume attribute follows a normal distribution
  - ◆ Use data to estimate parameters of distribution (e.g., mean and standard deviation)
  - ◆ Once probability distribution is known, use it to estimate the conditional probability  $P(X_i | Y)$



# Probability Densities

## Continuous Probability Distribution

Let  $X$  be a continuous rv. Then a *probability distribution or probability density function (pdf)* of  $X$  is a function  $f(x)$  such that for any two numbers  $a$  and  $b$ ,

$$P(a \leq X \leq b) = \int_a^b f(x) dx$$

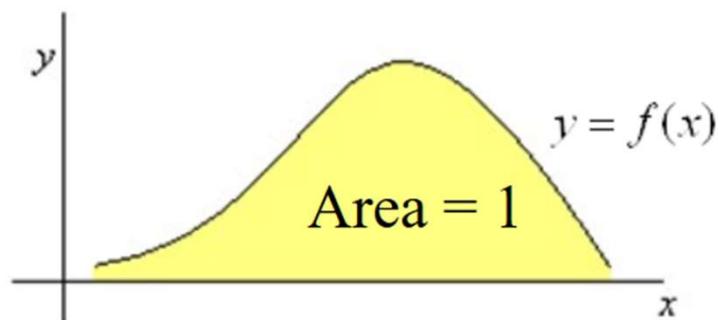
The graph of  $f$  is the *density curve*.

# Probability Densities

## Probability Density Function

For  $f(x)$  to be a pdf

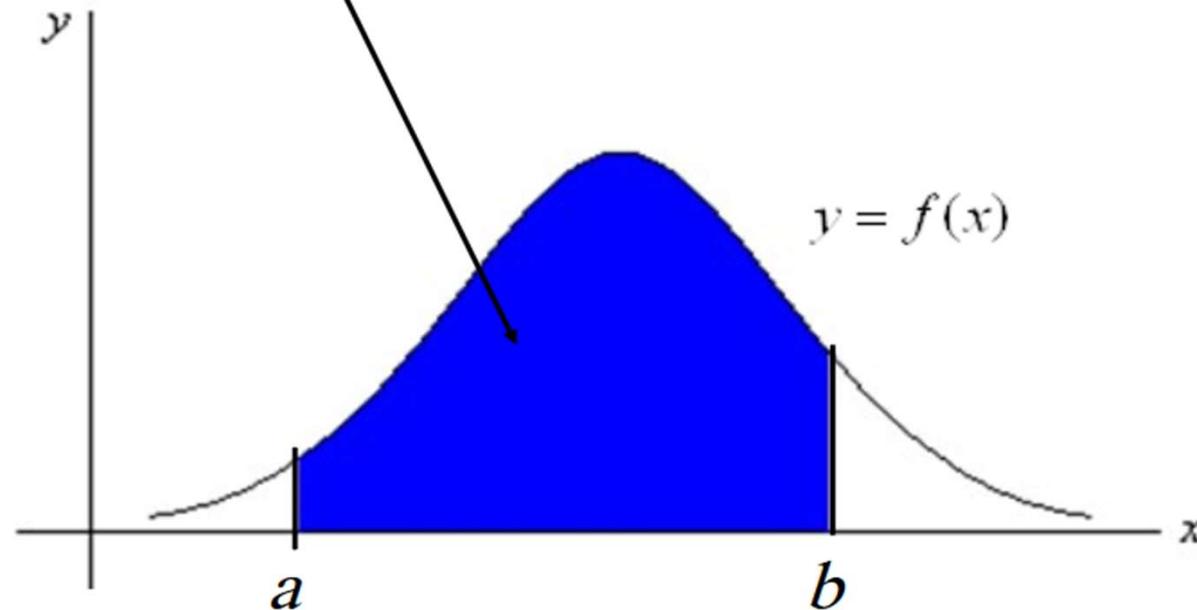
1.  $f(x) > 0$  for all values of  $x$ .
2. The area of the region between the graph of  $f$  and the  $x$ -axis is equal to 1.



# Probability Densities

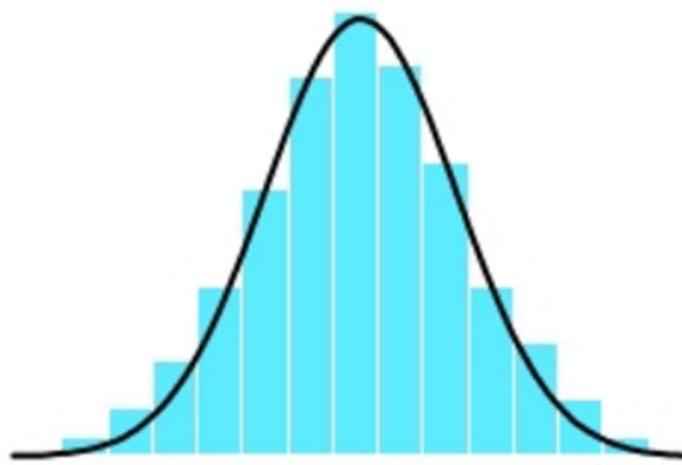
## Probability Density Function

$P(a \leq X \leq b)$  is given by the area of the shaded region.



# Gaussian Distribution

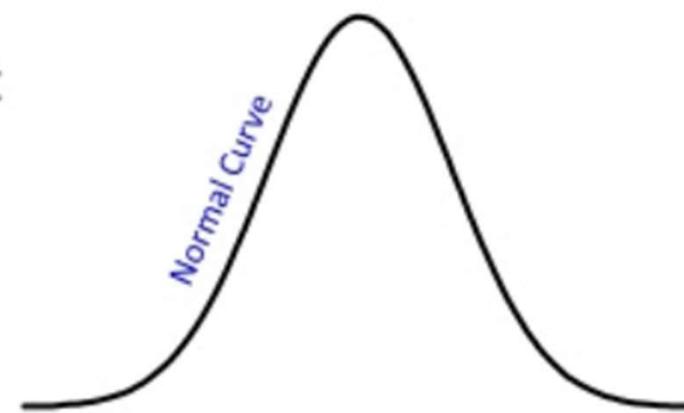
- The commonest and the most useful continuous distribution.
- A symmetrical probability distribution where most results are located in the middle and few are spread on both sides.
- It has the shape of a bell.
- Can entirely be described by its mean and standard deviation.



# Gaussian Distribution

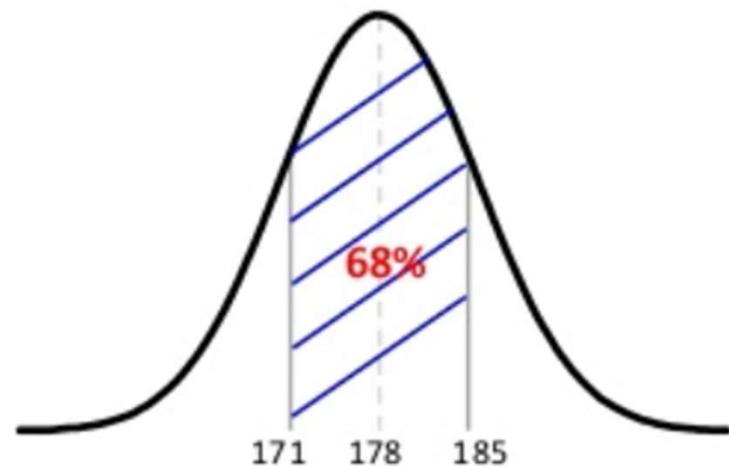
## Normal Curve:

- A graphical representation of the normal distribution.
- It is determined by the mean and the standard deviation.
- It a symmetric unimodal bell-shaped curve.
- Its tails extending infinitely in both directions.
- The wider the curve, the larger the standard deviation and the more variation exists in the process.
- The spread of the curve is equivalent to six times the standard deviation of the process.



# Gaussian Distribution

- ❑ Suppose that the heights of a sample men are normally distributed.
- ❑ The mean height is **178** cm and a standard deviation is **7** cm.
  
- ❑ **We can generalize that:**
  - **68%** of population are between **171** cm and **185** cm.
  - This might be a generalization, but it's true if the data is normally distributed.





# Mean, Variance & Standard Deviation

---

- ✓ The mean of a discrete random variable is the **weighted average** of all of its values. The weights are the probabilities.
- ✓ This parameter is also called the expected value of X and is represented by  $E(X)$ .

$$E(X) = \mu = \sum_{all \ x} xP(x)$$

- ✓ The variance is

$$V(X) = \sigma^2 = \sum_{all \ x} (x - \mu)^2 P(x)$$

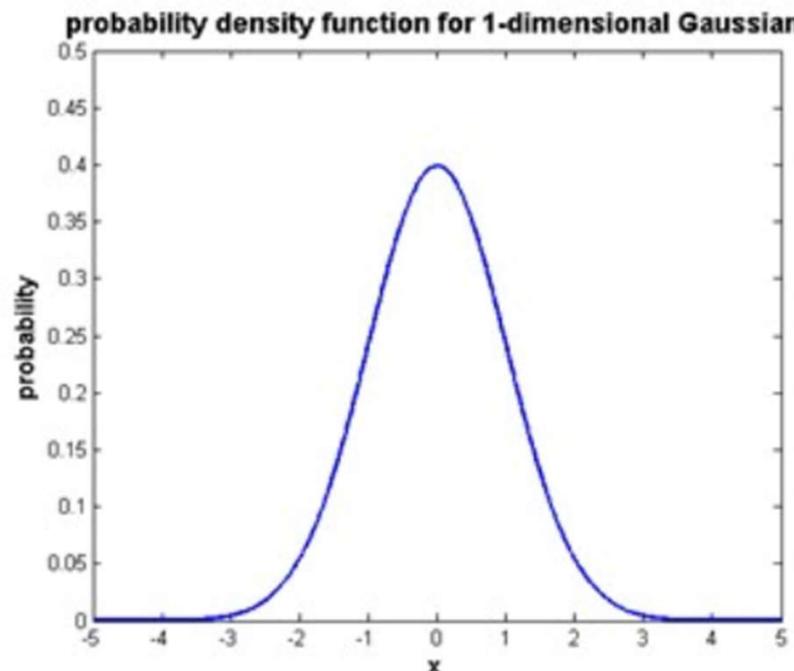
- ✓ The standard deviation is

$$\sigma = \sqrt{\sigma^2}$$

# Gaussian Distribution

## In one dimension

$$N(x | \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



# Gaussian Distribution

## In one dimension

Causes pdf to decrease as distance from center increases

$$N(x | \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Controls width of curve

Normalizing constant:  
insures that distribution  
integrates to 1

# Estimate Probabilities from Data



Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

| Normal distribution:

$$P(X_i | Y_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(X_i - \mu_{ij})^2}{2\sigma_{ij}^2}}$$

– One for each  $(X_i, Y_j)$  pair

| For (Income, Class=No):

– If Class=No

◆ sample mean = 110

◆ sample variance = 2975

$$P(Income = 120 | No) = \frac{1}{\sqrt{2\pi}(54.54)} e^{-\frac{(120-110)^2}{2(2975)}} = 0.0072$$

21



# JOINT Distributions

- Probability distribution of two random variables  $X \{x_1, x_2, \dots, x_n\}$  and  $Y \{y_1, y_2, \dots, y_k\}$ 
  - Occurrence of  $X=x_i$  and  $Y=y_i$  together

- Example:

- $P(X=0, Y \leq 1)$

- $P(X=1)$

$$= \sum_{y=0}^2 P(X = 1, Y)$$

$$= 1/6 + 1/6 + 1/8$$

		Y		
		0	1	2
X	0	1/4	1/6	1/8
	1	1/6	1/6	1/8



# JOINT Distribution

- Marginal Distribution
  - Sum over any one variable is called Marginal Distribution

$$P(X=x) = \sum_{y \in Y} P(X, Y)$$

$$P(Y=y) = \sum_{x \in X} P(X, Y)$$



# Conditional Probability

- Estimating probability for two or more related events
  - Measure influence of one variable over another
- Let A and B be two events,  $p(B) > 0$ 
$$p(A|B) = p(A \cap B) / p(B)$$
- Using random variable notations,
  - $p(a|b)$  denotes the probability of  $A=a$  and  $B=b$ 
    - i.e.  $p(A=a | B=b)$



# Basic Formulas for Probabilities

- *Product Rule*: probability  $P(A \wedge B)$  of a conjunction of two events A and B:

$$P(A \wedge B) = P(A | B) P(B) = P(B | A) P(A)$$

- *Sum Rule*: probability of a disjunction of two events A and B:

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

- *Theorem of total probability*: if events  $A_1, \dots, A_n$  are mutually exclusive with  $\sum_{i=1}^n P(A_i) = 1$ , then

$$P(B) = \sum_{i=1}^n P(B|A_i)P(A_i)$$



# Example 1

---

$$P(A | B) = \frac{P(B | A)P(A)}{P(B | A)P(A) + P(B | \sim A)P(\sim A)}$$

A = you have the flu, B = you just coughed

Assume:

$$P(A) = 0.05$$

$$P(B|A) = 0.80$$

$$P(B| \sim A) = 0.20$$

what is  $P(\text{flu} | \text{cough}) = P(A|B)$ ?



## Example 2

---

- Does a patient have cancer or not?
  - The patient takes a lab test and the test returns a correct positive result in only 98% of the cases in which the disease is actually present
  - And a correct negative result in only 97% of the cases in which the disease is not present
  - Of the total population, only 0.008% has cancer



## Example 2

Given:

$$P(\text{cancer}) =$$

$$P(\neg \text{cancer}) =$$

$$P(+ | \text{cancer}) =$$

$$P(- | \text{cancer}) =$$

$$P(+ | \neg \text{cancer}) =$$

$$P(- | \neg \text{cancer}) =$$

Using Bayes Theorem:

$$P(\text{cancer} | +) = P(+ | \text{cancer}) P(\text{cancer}) / P(+)$$

$$P(\neg \text{cancer} | +) = P(+ | \neg \text{cancer}) P(\neg \text{cancer}) / P(+)$$

Since denominator is common

$$P(\text{cancer} | +) \text{ proportional to } P(+ | \text{cancer}) P(\text{cancer})$$

$$P(\neg \text{cancer} | +) \text{ proportional to } P(+ | \neg \text{cancer}) P(\neg \text{cancer})$$



## Example 2

$$P(\text{cancer}) = 0.008$$

$$P(+|\text{cancer}) = 0.98$$

$$P(+|\neg\text{cancer}) = 1-0.97=0.3 \quad P(-|\neg\text{cancer}) = 0.97$$

$$P(\neg\text{cancer}) = 1-0.008=0.992$$

$$P(-|\text{cancer}) = 0.02$$



## Remember: Some terminology

---

- Likelihood function:  $P(\text{data} | \theta)$
- Prior:  $P(\theta)$
- Posterior:  $P(\theta | \text{data})$
- **Conjugate prior:**  $P(\theta)$  is the conjugate prior for likelihood function  $P(\text{data} | \theta)$  if the forms of  $P(\theta)$  and  $P(\theta | \text{data})$  are the same.



# Bayesian Learning

---

- Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practical approaches to certain types of learning problems
- For example: Problem of learning to classify text documents such as electronic news articles.
- For such learning tasks, the naive Bayes classifier is among the most effective algorithms known



# Features of Bayesian learning

---

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct.
- Flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions (e.g., hypotheses such as "this pneumonia patient has a 93% chance of complete recovery").



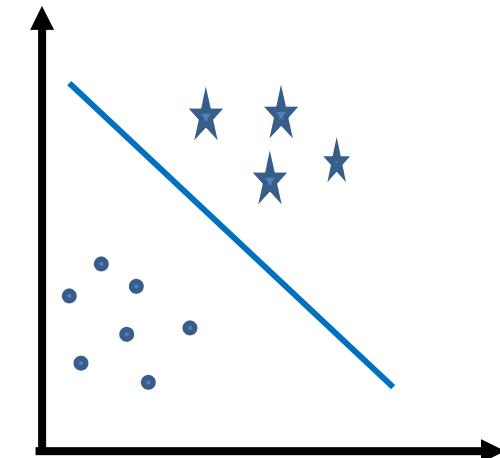
# Features of Bayesian learning

- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis.
- Prior knowledge is provided by asserting
  - prior probability for each candidate hypothesis, and
  - probability distribution over observed data for each possible hypothesis.
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.



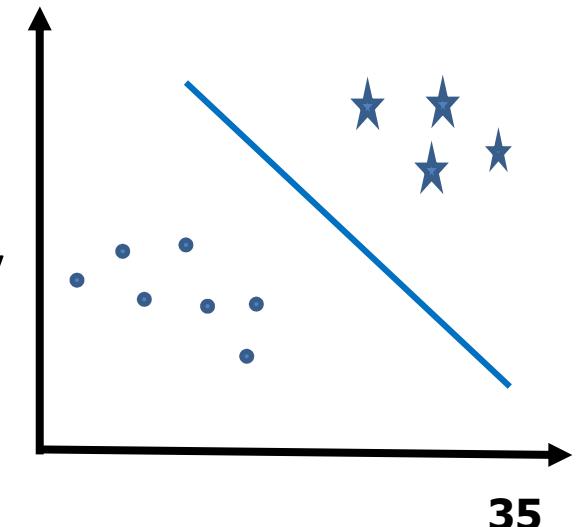
# Machine Learning

- Objective of ML: Given data  $X$  and target variable  $Y$ , determine the joint distribution  $P(X, Y)$
- Classification Problem
  - Decision boundary that separates one class from another class
  - Determine  $P(Y|X)$
  - These models are called Deterministic Or Discriminative models



# Machine Learning

- Alternate approach is to understand the process that generated the data
  - Generative Models  $P(X,Y)$
  - Build a model for all the positive cases or category 1
  - Build another model for all the negative cases or category 2
  - For predicting a new test case
    - Run the test case with both the models and choose the model with maximum probability





# Machine Learning

- Generative models
  - Build model to estimate the posterior probability  $P(Y|X)$  by estimating
  - likelihood of data given target (hypothesis)  $P(X|Y)$
  - Prior probabilities over target  $P(Y)$
  - In general, for a specific class  $Y=c_k$ ,

$$P(Y = c_k | X) = \frac{P(X|Y = c_k) * P(Y=c_k)}{P(X)}$$



# Hypothesis

---

- Relationship between the input and output values.
- Lets say that **target function**  $y=f(x)$   
However,  $f(\cdot)$  is unknown function to us.
- Machine learning algorithms try to guess a hypothesis function  $h(x)$  that approximates the unknown  $f(\cdot)$
- Set of all possible hypotheses is known as the Hypothesis set or space  $H(\cdot)$
- Goal is the learning process is to find the final hypothesis that best approximates the unknown target function.



# Bayes Theorem

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- $P(h)$  = prior probability of hypothesis  $h$
- $P(D)$  = prior probability of training data  $D$
- $P(h|D)$  = probability of  $h$  given  $D$
- $P(D|h)$  = probability of  $D$  given  $h$



# Choosing Hypotheses

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- Generally want the most probable hypothesis given the training data

*Maximum a posteriori* hypothesis  $h_{MAP}$ :

$$\begin{aligned} h_{MAP} &= \arg \max_{h \in H} P(h|D) \\ &= \arg \max_{h \in H} \frac{P(D|h)P(h)}{P(D)} \\ &= \arg \max_{h \in H} P(D|h)P(h) \end{aligned}$$

- If assume  $P(h_i) = P(h_j)$  then can further simplify, and choose the *Maximum likelihood* (ML) hypothesis

$$h_{ML} = \arg \max_{h_i \in H} P(D|h_i)$$



# Brute Force MAP Hypothesis

- 
1. For each hypothesis  $h$  in  $H$ , calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis  $h_{MAP}$  with the highest posterior probability

$$h_{MAP} = \operatorname{argmax}_{h \in H} P(h|D)$$



# MAP Hypothesis

- Using Bayes theorem, we compute the MAP hypothesis for all probable hypothesis (or all unique class labels)
- Identify the best hypothesis describing the data as

$$\begin{aligned} h_{MAP} &= \operatorname{argmax}_{h \in H} P(h|D) \\ &= \operatorname{argmax}_{h \in H} \frac{P(D|h)P(h)}{P(D)} \\ &= \operatorname{argmax}_{h \in H} P(D|h)P(h) \end{aligned}$$

H: set of all hypothesis

P(D) is independent of h and is same for all hypothesis, therefore dropped



# Maximum Likelihood estimation

---

- When no prior information is available, all hypothesis are equally likely i.e.  $p(h_i) = p(h_j)$ 
  - This is also true for a balanced class problem where all the classes are equally likely
  - This is known as Uniform prior
  - MAP hypothesis further simplifies to:

$$H_{ML} = \operatorname{argmax}_{h \in H} P(D | h)$$

This is called Maximum Likelihood Hypothesis



# ML setting

---

- Bayesian Analysis
  - start with some belief about the system, called a prior.
  - Then we obtain some data and use it to update our belief.
  - The outcome is called a posterior.
  - Should we obtain even more data, the old posterior becomes a new prior and the cycle repeats.
  - People often use likelihood for evaluation of models: a model that gives higher likelihood to real data is better



# ML Setting

- $P(h | D)$  a posterior determines the class label
- It's a probability distribution over model parameters obtained from prior beliefs and data.
- When one uses likelihood to get point estimates of model parameters, it's called Maximum Likelihood estimation or MLE.
- If one also takes the prior into account, then it's maximum a posteriori estimation (MAP).
- MLE and MAP are the same if the prior is uniform
- This forms the basis for Naïve Bayes classifier



# Relation to Concept Learning

---

Concept learning can be formulated as

- problem of searching through a predefined space of potential hypotheses space  $H$  for the hypothesis that best fits the training examples  $D$
- consider the FindS learning algorithm (outputs most specific hypothesis from the version space  $VS_{H,D}$ )
- Does *FindS* output a MAP hypothesis??



# Relation to Concept Learning

- Assume fixed set of instances  $\langle x_1, \dots, x_m \rangle$
- Assume  $D$  is the set of classifications:  $D = \langle c(x_1), \dots, c(x_m) \rangle$
- Choose  $P(D|h)$ :
  - $P(D|h) = 1$  if  $h$  consistent with  $D$
  - $P(D|h) = 0$  otherwise
- Choose  $P(h)$  to be *uniform* distribution
  - $P(h) = 1/|H|$  for all  $h$  in  $H$



# Relation to Concept Learning

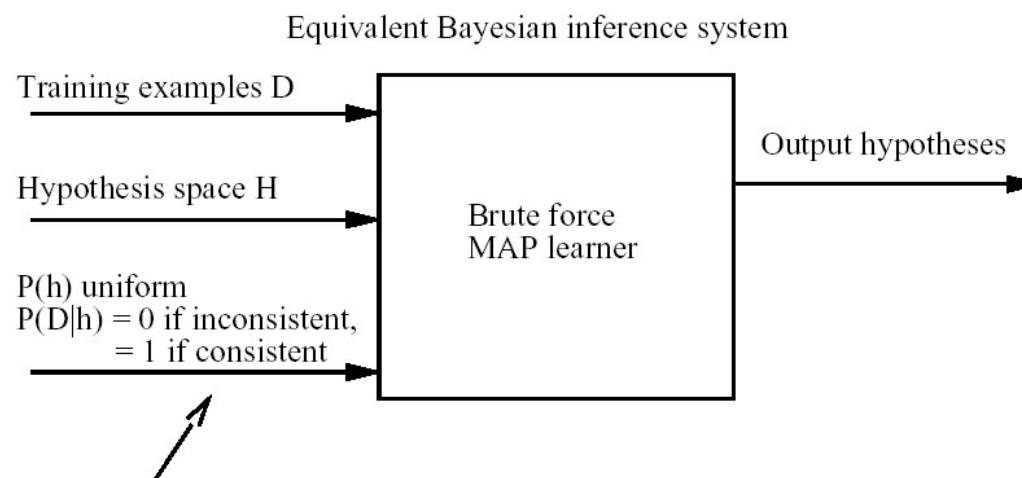
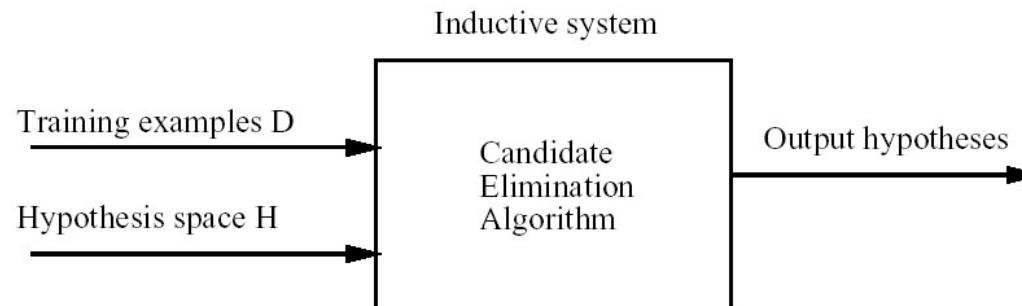
- $P(h|D) = P(D|h)P(h) = 0$  if  $h$  is inconsistent with  $D$  ie.  $P(D|h)=0$

$$\begin{aligned} P(h|D) &= \frac{1 \cdot \frac{1}{|H|}}{P(D)} \\ &= \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} \\ &= \frac{1}{|VS_{H,D}|} \text{ if } h \text{ is consistent with } D \end{aligned}$$

$VS_{H,D}$  is the subset of hypotheses from  $H$  that are consistent with  $D$

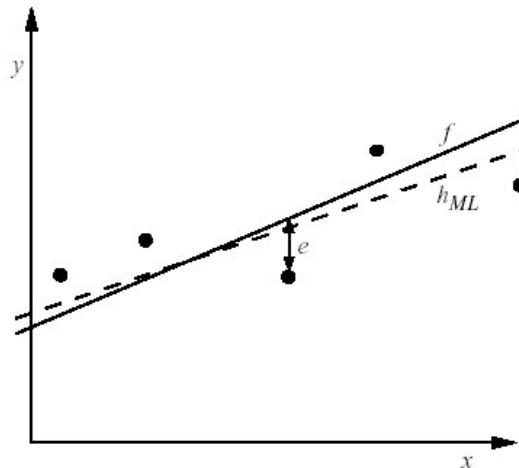
$$P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases}$$

# Characterizing Learning Algorithms by Equivalent MAP Learners



*Prior assumptions  
made explicit*

# Learning A Real Valued Function



Consider any real-valued target function  $f$

Training examples  $\langle x_i, d_i \rangle$ , where  $d_i$  is noisy training value

- $d_i = f(x_i) + e_i$
- $e_i$  is random variable (noise) drawn independently for each  $x_i$  according to some Gaussian distribution with mean=0

Then the maximum likelihood hypothesis  $h_{ML}$  is the one that minimizes

$$\text{the sum of squared errors: } h_{ML} = \arg \min_{h \in H} \sum_{i=1}^m (d_i - h(x_i))^2$$



# Learning A Real Valued Function

$$\begin{aligned} h_{ML} &= \operatorname{argmax}_{h \in H} p(D|h) \\ &= \operatorname{argmax}_{h \in H} \prod_{i=1}^m p(d_i|h) \\ &= \operatorname{argmax}_{h \in H} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{d_i-h(x_i)}{\sigma}\right)^2} \end{aligned}$$

- Maximize natural log of this instead...

$$\begin{aligned} h_{ML} &= \operatorname{argmax}_{h \in H} \sum_{i=1}^m \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2} \left( \frac{d_i - h(x_i)}{\sigma} \right)^2 \\ &= \operatorname{argmax}_{h \in H} \sum_{i=1}^m -\frac{1}{2} \left( \frac{d_i - h(x_i)}{\sigma} \right)^2 \\ &= \operatorname{argmax}_{h \in H} \sum_{i=1}^m -(d_i - h(x_i))^2 \\ &= \operatorname{argmin}_{h \in H} \sum_{i=1}^m (d_i - h(x_i))^2 \end{aligned}$$

# Minimum Description Length Principle

---

$$\begin{aligned}
 h_{MAP} &= \arg \max_{h \in H} P(D|h)P(h) \\
 &= \arg \max_{h \in H} \log_2 P(D|h) + \log_2 P(h) \\
 &= \arg \min_{h \in H} -\log_2 P(D|h) - \log_2 P(h) \quad (1)
 \end{aligned}$$

Interesting fact from information theory:

The optimal (shortest expected coding length) code for an event with probability  $p$  is  $-\log_2 p$  bits.

So interpret (1):

- $L_{C1}(h) = \text{length}(h) = -\log_2 P(h)$
- $L_{C2}(D|h) = \text{length}(\text{misclassifications}) = -\log_2 P(D|h)$

→ prefer the hypothesis that minimizes

$$\text{length}(h) + \text{length}(\text{misclassifications})$$



# Minimum Description Length Principle

---

Occam's razor: prefer the shortest hypothesis

MDL: prefer the hypothesis  $h$  that minimizes

$$h_{MDL} = \operatorname{argmin}_{h \in H} L_{C_1}(h) + L_{C_2}(D|h)$$

where  $L_C(x)$  is the description length of  $x$  under encoding  $C$

---

Example:  $H$  = decision trees hypothesis,  $D$  = training data labels

- $L_{C_1}(h)$  is # bits to describe tree  $h$
- $L_{C_2}(D|h)$  is # bits to describe  $D$  given  $h$ 
  - Note  $L_{C_2}(D|h) = 0$  if examples classified perfectly by  $h$ . Need only describe exceptions
- Hence  $h_{MDL}$  trades off tree size for training errors



# Minimum Description Length Principle

---

- MDL principle provides a way of trading off hypothesis complexity for the number of errors committed by the hypothesis.
- May select a shorter hypothesis that makes a few errors over a longer hypothesis that perfectly classifies the training data.
- Provides one method for dealing with the issue of overfitting the data.

# Most Probable Classification of New Instances

---



- So far we've sought the most probable *hypothesis* given the data  $D$  (i.e.,  $h_{MAP}$ )
- Given new instance  $x$ , what is its most probable *classification*?
  - $h_{MAP}(x)$  is not the most probable classification!
- Consider:
  - Three possible hypotheses:
$$P(h_1|D) = .4, P(h_2|D) = .3, P(h_3|D) = .3$$
  - Given new instance  $x$ ,
$$h_1(x) = +, h_2(x) = -, h_3(x) = -$$
  - What's most probable classification of  $x$ ?



# Bayes Optimal Classifier

- **Bayes optimal classification:**

$$\arg \max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

- Example:

$$P(h_1|D) = .4, P(-|h_1) = 0, P(+|h_1) = 1$$

$$P(h_2|D) = .3, P(-|h_2) = 1, P(+|h_2) = 0$$

$$P(h_3|D) = .3, P(-|h_3) = 1, P(+|h_3) = 0$$

therefore

$$\sum_{h_i \in H} P(+|h_i)P(h_i|D) = .4$$

$$\sum_{h_i \in H} P(-|h_i)P(h_i|D) = .6$$

and

$$\arg \max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = -$$



# Gibbs Classifier

- Bayes optimal classifier provides best result, but can be expensive if many hypotheses.
- Gibbs algorithm:
  1. Choose one hypothesis at random, according to  $P(h | D)$
  2. Use this to classify new instance
- Surprising fact: Assume target concepts are drawn at random from  $H$  according to priors on  $H$ . Then:

$$E[\text{error}_{\text{Gibbs}}] \leq 2E[\text{error}_{\text{BayesOptional}}]$$

- Suppose correct, uniform prior distribution over  $H$ , then
  - Pick any hypothesis from  $VS$ , with uniform probability
  - Its expected error no worse than twice Bayes optimal



# Practical Issues of Bayesian learning

---

- Require initial knowledge of many probabilities
  - Often estimated based on background knowledge, previously available data, and assumptions about the form of the underlying distributions.
- Significant computational cost required to determine the Bayes optimal hypothesis in the general case (linear in the number of candidate hypotheses)



---

# Thank You



# Machine Learning

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**Lecture No. – 3 | Bayesian Learning**

**Date – 09/11/2019**

**Time – 9:00 AM – 11:00 AM**

# Session Content

---

- Probabilistic Generative Classifiers
- Bayes optimal classifier (T1 book by Tom Mitchell - 6.7)
- Gibbs Algorithm (T1 book by Tom Mitchell - 6.8)
- Naïve Bayes Classifier (T1 book by Tom Mitchell - 6.9)
- Text classification model (T1 book by Tom Mitchell - 6.9)

# Bayesian Learning

---

- Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practical approaches to certain types of learning problems
- For example: Problem of learning to classify text documents such as electronic news articles.
- For such learning tasks, the naive Bayes classifier is among the most effective algorithms known

# Features of Bayesian learning

---

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct.
- Flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions (e.g., hypotheses such as "this pneumonia patient has a 93% chance of complete recovery").

# Features of Bayesian learning

---

- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis.
- Prior knowledge is provided by asserting
  - prior probability for each candidate hypothesis, and
  - probability distribution over observed data for each possible hypothesis.
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.

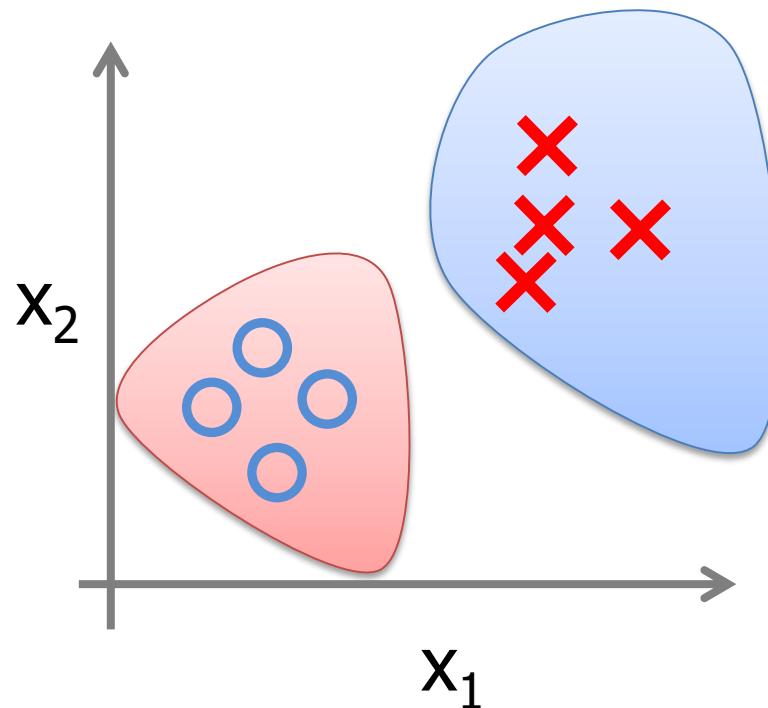
# Probabilistic Generative Classifiers

---

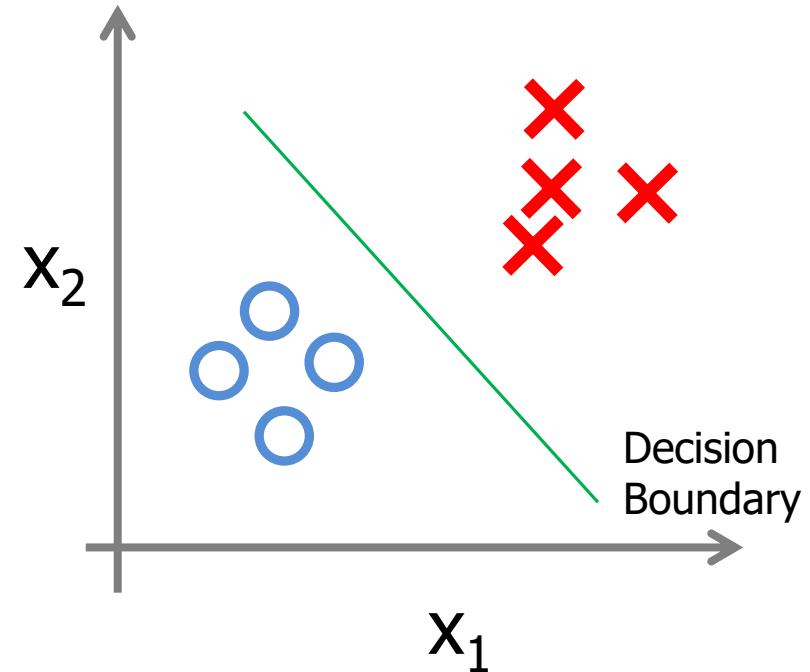
- Approach is to understand the process that generated the data
  - Generative Models  $P(X,Y)$
  - Build a model for all the positive cases or category 1
  - Build another model for all the negative cases or category 2
  - For predicting a new test case
    - Run the test case with both the models and choose the model with maximum probability

# Probabilistic Generative Model versus Probabilistic Discriminative Model

Generative:



Discriminative:



# Probabilistic Models: Generative/Discriminative

- Model  $p(C_k|x)$  in an *inference* stage and use it to make optimal decisions
- Approaches to computing the  $p(C_k|x)$

## 1. Generative

- Model class conditional densities by  $p(x|C_k)$  together with prior probabilities  $p(C_k)$
- Then use Bayes rule to compute posterior

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

## 2. Discriminative

- Directly model conditional probabilities  $p(C_k|x)$

13

9

# Probabilistic Generative Model versus Probabilistic Discriminative Model

Generative	Discriminative
Ex: Naïve Bayes	Ex: Logistic Regression
Estimate $P(Y)$ and $P(X Y)$	Finds class label directly $P(Y X)$
Prediction $\hat{y} = \text{argmax}_y P(Y = y)P(X = x Y = y)$	Prediction $\hat{y} = P(Y = y X = x)$

# Generative Models

---

- Generative models
  - Build model to estimate the posterior probability  $P(Y|X)$  by estimating
  - likelihood of data given target (hypothesis)  $P(X|Y)$
  - Prior probabilities over target  $P(Y)$
  - In general, for a specific class  $Y=c_k$ ,

$$P(Y = c_k | X) = \frac{P(X|Y = c_k) * P(Y=c_k)}{P(X)}$$

# Most Probable Classification of New Instances

---



- So far we've sought the most probable *hypothesis* given the data  $D$  (i.e.,  $h_{MAP}$ )
- Given new instance  $x$ , what is its most probable *classification*?
  - $h_{MAP}(x)$  is not the most probable classification!
- Consider:
  - Three possible hypotheses:
$$P(h_1|D) = .4, P(h_2|D) = .3, P(h_3|D) = .3$$
  - Given new instance  $x$ ,
$$h_1(x) = +, h_2(x) = -, h_3(x) = -$$
  - What's most probable classification of  $x$ ?

# Bayes Optimal Classifier

- **Bayes optimal classification:**

$$\arg \max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

- Example:

$$P(h_1|D) = .4, P(-|h_1) = 0, P(+|h_1) = 1$$

$$P(h_2|D) = .3, P(-|h_2) = 1, P(+|h_2) = 0$$

$$P(h_3|D) = .3, P(-|h_3) = 1, P(+|h_3) = 0$$

therefore

$$\sum_{h_i \in H} P(+|h_i)P(h_i|D) = .4$$

$$\sum_{h_i \in H} P(-|h_i)P(h_i|D) = .6$$

and

$$\arg \max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = -$$

# Gibbs Classifier

---

- Bayes optimal classifier provides best result, but can be expensive if many hypotheses.
- Gibbs algorithm:
  1. Choose one hypothesis at random, according to  $P(h | D)$
  2. Use this to classify new instance
- Surprising fact: Assume target concepts are drawn at random from  $H$  according to priors on  $H$ . Then:

$$E[\text{error}_{\text{Gibbs}}] \leq 2E[\text{error}_{\text{BayesOptional}}]$$

- Suppose correct, uniform prior distribution over  $H$ , then
  - Pick any hypothesis from  $VS$ , with uniform probability
  - Its expected error no worse than twice Bayes optimal

# Conditional independence

---

- **Definition:**  $X$  is conditionally independent of  $Y$  given  $Z$ , if the probability distribution governing  $X$  is independent of the value of  $Y$ , given the value of  $Z$

$$(\forall i, j, k) P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z_k)$$

$$P(X|Y, Z) = P(X|Z)$$

Example:

$$P(\text{Thunder}|\text{Rain, Lightning}) = P(\text{Thunder}|\text{Lightning})$$

Slide credit: Tom  
Mitchell

# Applying conditional independence

---

- Naïve Bayes assumes  $X_i$  are conditionally independent given  $Y$

e.g.,  $P(X_1|X_2, Y) = P(X_1|Y)$

$$\begin{aligned}P(X_1, X_2|Y) &= P(X_1|X_2, Y)P(X_2|Y) \\&= P(X_1|Y)P(X_2|Y)\end{aligned}$$

General form:  $P(X_1, \dots, X_n|Y) = \prod_{j=1}^n P(X_j|Y)$

Slide credit: Tom  
Mitchell

# Naïve Bayes Independence assumption

---

- Assumption:

$$P(X_1, \dots, X_n | Y) = \prod_{j=1}^n P(X_j | Y)$$

- i.e.,  $X_i$  and  $X_j$  are conditionally independent given  $Y$  for  $i \neq j$

Slide credit: Tom  
Mitchell

# Naïve Bayes classifier

- Bayes rule:

$$P(Y = y_k | X_1, \dots, X_n) = \frac{P(Y = y_k)P(X_1, \dots, X_n | Y = y_k)}{\sum_j P(Y = y_j)P(X_1, \dots, X_n | Y = y_j)}$$

- Assume conditional independence among  $X_i$ 's:

$$P(Y = y_k | X_1, \dots, X_n) = \frac{P(Y = y_k)\prod_i P(X_i | Y = y_k)}{\sum_j P(Y = y_j)\prod_i P(X_i | Y = y_j)}$$

- Pick the most probable (MAP)  $Y$

$$\hat{Y} \leftarrow \operatorname{argmax}_{y_k} P(Y = y_k)\prod_i P(X_i | Y = y_k)$$

↑  
 Prior  
 Probability      ↑  
 MLE

Slide credit: Tom Mitchell

# NAÏVE BAYES CLASSIFIER

---

- Assume independence among attributes  $X_i$  when class is given:
  - $P(X_1, X_2, \dots, X_d | Y_j) = P(X_1 | Y_j) P(X_2 | Y_j) \dots P(X_d | Y_j)$
  - Now we can estimate  $P(X_i | Y_j)$  for all  $X_i$  and  $Y_j$  combinations from the training data
  - New point is classified to  $Y_j$  if  $P(Y_j) \prod P(X_i | Y_j)$  is maximal.

# Naive Bayes Classifier

- Assume target function  $f: X \rightarrow V$ , where each instance  $x$  described by attributes  $\langle a_1, a_2 \dots a_n \rangle$ .
- Most probable value of  $f(x)$  is:

$$\begin{aligned}
 v_{MAP} &= \operatorname{argmax}_{v_j \in V} P(v_j | a_1, a_2 \dots a_n) \\
 v_{MAP} &= \operatorname{argmax}_{v_j \in V} \frac{P(a_1, a_2 \dots a_n | v_j) P(v_j)}{P(a_1, a_2 \dots a_n)} \\
 &= \operatorname{argmax}_{v_j \in V} P(a_1, a_2 \dots a_n | v_j) P(v_j)
 \end{aligned}$$

Naive Bayes assumption:

$$P(a_1, a_2 \dots a_n | v_j) = \prod_i P(a_i | v_j)$$

**Naive Bayes classifier:**

$$v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$$

# Naive Bayes Algorithm

---

- Naive Bayes Learn(*examples*)

For each target value  $v_j$

$$\hat{P}(v_j) \leftarrow \text{estimate } P(v_j)$$

For each attribute value  $a_i$  of each attribute  $a$

$$\hat{P}(a_i | v_j) \leftarrow \text{estimate } P(a_i | v_j)$$

- Classify New Instance( $x$ )

$$v_{NB} = \operatorname{argmax}_{v_j \in V} \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i | v_j)$$

# Example 1

Name	Give Birth	Can Fly	Live in Water	Have Legs	Class
human	yes	no	no	yes	mammals
python	no	no	no	no	non-mammals
salmon	no	no	yes	no	non-mammals
whale	yes	no	yes	no	mammals
frog	no	no	sometimes	yes	non-mammals
komodo	no	no	no	yes	non-mammals
bat	yes	yes	no	yes	mammals
pigeon	no	yes	no	yes	non-mammals
cat	yes	no	no	yes	mammals
leopard shark	yes	no	yes	no	non-mammals
turtle	no	no	sometimes	yes	non-mammals
penguin	no	no	sometimes	yes	non-mammals
porcupine	yes	no	no	yes	mammals
eel	no	no	yes	no	non-mammals
salamander	no	no	sometimes	yes	non-mammals
gila monster	no	no	no	yes	non-mammals
platypus	no	no	no	yes	mammals
owl	no	yes	no	yes	non-mammals
dolphin	yes	no	yes	no	mammals
eagle	no	yes	no	yes	non-mammals

A: attributes

M: mammals

N: non-mammals

$$P(A | M) = \frac{6}{7} \times \frac{6}{7} \times \frac{2}{7} \times \frac{2}{7} = 0.06$$

$$P(A | N) = \frac{1}{13} \times \frac{10}{13} \times \frac{3}{13} \times \frac{4}{13} = 0.0042$$

$$P(A | M)P(M) = 0.06 \times \frac{7}{20} = 0.021$$

$$P(A | N)P(N) = 0.004 \times \frac{13}{20} = 0.0027$$

$$P(A|M)P(M) > P(A|N)P(N)$$

=> Mammals

Give Birth	Can Fly	Live in Water	Have Legs	Class
yes	no	yes	no	?

# Issues with Naïve Bayes Classifier

Consider the table with Tid = 7 deleted

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	No	Single	85K	Yes
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

## Naïve Bayes Classifier:

$$P(\text{Refund} = \text{Yes} | \text{No}) = 2/6$$

$$P(\text{Refund} = \text{No} | \text{No}) = 4/6$$

→  $P(\text{Refund} = \text{Yes} | \text{Yes}) = 0$

$$P(\text{Refund} = \text{No} | \text{Yes}) = 1$$

$$P(\text{Marital Status} = \text{Single} | \text{No}) = 2/6$$

→  $P(\text{Marital Status} = \text{Divorced} | \text{No}) = 0$

$$P(\text{Marital Status} = \text{Married} | \text{No}) = 4/6$$

$$P(\text{Marital Status} = \text{Single} | \text{Yes}) = 2/3$$

$$P(\text{Marital Status} = \text{Divorced} | \text{Yes}) = 1/3$$

$$P(\text{Marital Status} = \text{Married} | \text{Yes}) = 0/3$$

For Taxable Income:

If class = No: sample mean = 91

sample variance = 685

If class = Yes: sample mean = 90

sample variance = 25

Given X = (Refund = Yes, Divorced, 120K)

$$P(X | \text{No}) = 2/6 \times 0 \times 0.0083 = 0$$

$$P(X | \text{Yes}) = 0 \times 1/3 \times 1.2 \times 10^{-9} = 0$$

**Naïve Bayes will not be able to classify X as Yes or No!**

# Issues with Naïve Bayes Classifier

- | If one of the conditional probabilities is zero, then the entire expression becomes zero
- | Need to use other estimates of conditional probabilities than simple fractions
- | Probability estimation:

$$\text{Original : } P(A_i | C) = \frac{N_{ic}}{N_c}$$

$$\text{Laplace : } P(A_i | C) = \frac{N_{ic} + 1}{N_c + c}$$

$$\text{m - estimate : } P(A_i | C) = \frac{N_{ic} + mp}{N_c + m}$$

c: number of classes

p: prior probability of the class

m: parameter

$N_c$ : number of instances in the class

$N_{ic}$ : number of instances having attribute value  $A_i$  in class  $c$

# A Simple Example

Text	Tag	Which tag does the sentence <i>A very close game</i> belong to? i.e. $P(\text{sports}   \text{A very close game})$
“A great game”	Sports	Feature Engineering: Bag of words i.e use word frequencies without considering order
“The election was over”	Not sports	
“Very clean match”	Sports	Using Bayes Theorem: $P(\text{sports}   \text{A very close game})$ $= \frac{P(\text{A very close game}   \text{sports}) P(\text{sports})}{P(\text{A very close game})}$ -----
“It was a close election”	Not sports	

We assume that every word in a sentence is **independent** of the other ones

$$P(\text{a very close game}) = P(a) \times P(\text{very}) \times P(\text{close}) \times P(\text{game})$$

$$\begin{aligned} P(\text{a very close game} | \text{Sports}) &= P(a | \text{Sports}) \times P(\text{very} | \text{Sports}) \times \\ &P(\text{close} | \text{Sports}) \times P(\text{game} | \text{Sports}) \end{aligned}$$

“close” doesn’t appear in sentences of sports tag, So  $P(\text{close} | \text{sports}) = 0$ , which makes product 0

# Laplace smoothing

---

- Laplace smoothing: we add 1 or in general constant k to every count so it's never zero.
- To balance this, we add the number of possible words to the divisor, so the division will never be greater than 1
- In our case, the possible words are ['a', 'great', 'very', 'over', 'it', 'but', 'game', 'election', 'clean', 'close', 'the', 'was', 'forgettable', 'match'].

# Apply Laplace Smoothing

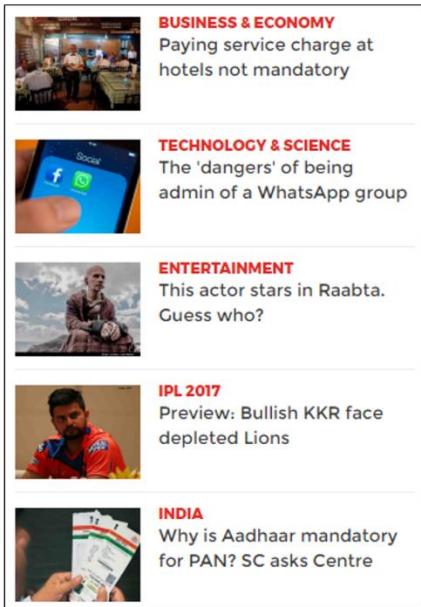
Word	P(word   Sports)	P(word   Not Sports)
a	2+1 / 11+14	1+1 / 9+14
very	1+1 / 11+14	0+1 / 9+14
close	0+1 / 11+14	1+1 / 9+14
game	2+1 / 11+14	0+1 / 9+14

$$\begin{aligned}
 & P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports) \times \\
 & P(Sports) \\
 & = 2.76 \times 10^{-5} \\
 & = 0.0000276
 \end{aligned}$$

$$\begin{aligned}
 & P(a|Not\ Sports) \times P(very|Not\ Sports) \times P(close|Not\ Sports) \times \\
 & P(game|Not\ Sports) \times P(Not\ Sports) \\
 & = 0.572 \times 10^{-5} \\
 & = 0.00000572
 \end{aligned}$$

# Naïve Bayes Classifier Applications

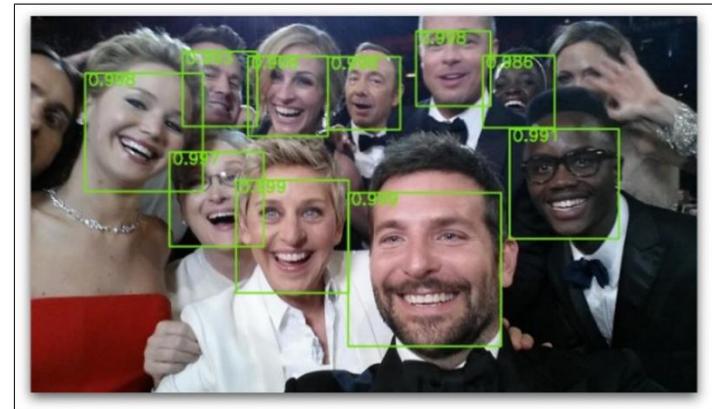
## Categorizing News



## Email Spam Detection



## Face Recognition



## Sentiment Analysis



# Naive Bayes Classifier

---

- Along with decision trees, neural networks, one of the most practical learning methods.
- When to use
  - Moderate or large training set available
  - Attributes that describe instances are conditionally independent given classification
- Successful applications:
  - Diagnosis
  - Classifying text documents

# Learning to Classify Text

---

- Why?
  - Learn which news articles are of interest
  - Learn to classify web pages by topic
- Naive Bayes is among most effective algorithms
- What attributes shall we use to represent text documents??

# Case Study: Text Classification

- Classify e-mails
  - $Y = \{\text{Spam}, \text{NotSpam}\}$
- Classify news articles
  - $Y = \text{what is the topic of the article?}$
- Classify webpages
  - $Y = \{\text{student}, \text{professor}, \text{project}, \dots\}$
- What about the features  $X$ ?
  - The text!

# Features $X$ are entire document - $X_i$ for $i$ th word in article

## Article from rec.sport.hockey

---

Path: cantaloupe.srv.cs.cmu.edu!das-news.harvard.e  
From: xxx@yyy.zzz.edu (John Doe)  
Subject: Re: This year's biggest and worst (opinic  
Date: 5 Apr 93 09:53:39 GMT

I can only comment on the Kings, but the most obvious candidate for pleasant surprise is Alex Zhitnik. He came highly touted as a defensive defenseman, but he's clearly much more than that. Great skater and hard shot (though wish he were more accurate). In fact, he pretty much allowed the Kings to trade away that huge defensive liability Paul Coffey. Kelly Hrudey is only the biggest disappointment if you thought he was any good to begin with. But, at best, he's only a mediocre goaltender. A better choice would be Tomas Sandstrom, though not through any fault of his own, but because some thugs in Toronto decided

# Baseline: Bag of Words Approach

*the world of*

**TOTAL**



***all about the company***

Our energy exploration, production, and distribution operations span the globe, with activities in more than 100 countries.

At TOTAL, we draw our greatest strength from our fast-growing oil and gas reserves. Our strategic emphasis on natural gas provides a strong position in a rapidly expanding market.

Our expanding refining and marketing operations in Asia and the Mediterranean Rim complement already solid positions in Europe, Africa, and the U.S.

Our growing specialty chemicals sector adds balance and profit to the core energy business.

**All About The Company**

- ▶ All About The Company
- Global Activities
- Corporate Structure
- TOTAL's Story
- Upstream Strategy
- Downstream Strategy
- Chemicals Strategy
- TOTAL Foundation
- Homepage

aardvark	0
about	2
all	2
Africa	1
apple	0
anxious	0
...	
gas	1
...	
oil	1
...	
Zaire	0

# Learning to Classify Text

---

LEARN\_NAIVE\_BAYES\_TEXT (*Examples*,  $V$ )

**1.** collect all words and other tokens that occur in *Examples*

- *Vocabulary*  $\leftarrow$  all distinct words and other tokens in *Examples*

**2.** calculate the required  $P(v_j)$  and  $P(w_k \mid v_j)$  probability terms

- For each target value  $v_j$  in  $V$  do
  - $docs_j \leftarrow$  subset of *Examples* for which the target value is  $v_j$
  - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
  - $Text_j \leftarrow$  a single document created by concatenating all members of  $docs_j$

# Learning to Classify Text

- $n \leftarrow$  total number of words in  $Text_j$  (counting duplicate words multiple times)
- for each word  $w_k$  in  $Vocabulary$ 
  - \*  $n_k \leftarrow$  number of times word  $w_k$  occurs in  $Text_j$
  - \*  $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

## CLASSIFY\_NAIVE\_BAYES\_TEXT ( $Doc$ )

- $positions \leftarrow$  all word positions in  $Doc$  that contain tokens found in  $Vocabulary$
- Return  $v_{NB}$  where  $v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_{i \in positions} P(a_i|v_j)$

**LEARN\_NAIVE\_BAYES\_TEXT(*Examples*,  $V$ )**

*Examples* is a set of text documents along with their target values.  $V$  is the set of all possible target values. This function learns the probability terms  $P(w_k|v_j)$ , describing the probability that a randomly drawn word from a document in class  $v_j$  will be the English word  $w_k$ . It also learns the class prior probabilities  $P(v_j)$ .

1. collect all words, punctuation, and other tokens that occur in *Examples*

- *Vocabulary*  $\leftarrow$  the set of all distinct words and other tokens occurring in any text document from *Examples*

2. calculate the required  $P(v_j)$  and  $P(w_k|v_j)$  probability terms

- For each target value  $v_j$  in  $V$  do
  - $docs_j \leftarrow$  the subset of documents from *Examples* for which the target value is  $v_j$
  - $P(v_j) \leftarrow \frac{|docs_j|}{|\text{Examples}|}$
  - $Text_j \leftarrow$  a single document created by concatenating all members of  $docs_j$
  - $n \leftarrow$  total number of distinct word positions in  $Text_j$
  - for each word  $w_k$  in *Vocabulary*
    - $n_k \leftarrow$  number of times word  $w_k$  occurs in  $Text_j$
    - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|\text{Vocabulary}|}$

**CLASSIFY\_NAIVE\_BAYES\_TEXT(*Doc*)**

Return the estimated target value for the document *Doc*.  $a_i$  denotes the word found in the  $i$ th position within *Doc*.

- *positions*  $\leftarrow$  all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return  $v_{NB}$ , where

$$v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_{i \in \text{positions}} P(a_i | v_j)$$

# Naïve Bayes for Text Classification

- **Naïve Bayes assumption helps a lot!**
  - $P(X_i = x_i | Y = y)$  is just the probability of observing word  $x_i$  at the  $i$ th position in a document on topic  $y$ .
  - Assume  $X_i$  is independent of all other words in document given the label  $y$ :  
$$P(X_i = x_i | Y = y, X_{-i}) = P(X_i = x_i | Y = y).$$

$$h_{NB}(x) = \arg \max_y P(y) \prod_{i=1}^{\text{lengthDoc}} P(X_i = x_i | y)$$

- For each label  $y$ , have 1000 distributions of size 10000 to estimate.
- This is  $10000 \times 1000$  items, which is big but much less than  $10000^{1000}$  ...

# Bag of Words model

- Typical additional assumption – **Position in document doesn't matter:**

$$P(X_i = x_i | Y = y) = P(X_k = x_i | Y = y)$$

- “**Bag of Words**” model – order of words on the page ignored

Can simplify further:

$$\prod_{i=1}^{\text{lengthDoc}} P(x_i|y) = \prod_{w=1}^W P(w|y)^{\text{count}(w)}$$



# Bag of Words representation

- Since we are assuming the order of words doesn't matter, an alternative representation of document is as vector of counts:
  - $x^{(j)}$  = number of occurrences of word  $j$  in document  $x$ .
  - Typical document: [0 0 1 0 0 3 0 0 0 1 0 0 0 1 0 0 2 0 0 ...]
  - Called “bag of words” or “term vector” or “vector space model” representation



# Naïve Bayes with Bag of Words for text classification

- Learning phase
  - Class Prior  $P(Y)$
  - $P(X_i|Y)$
- Test phase:
  - For each document
    - Use naïve Bayes decision rule

$$h_{NB}(x) = \arg \max_y P(y) \prod_{i=1}^{1000} P(x_i|y)$$



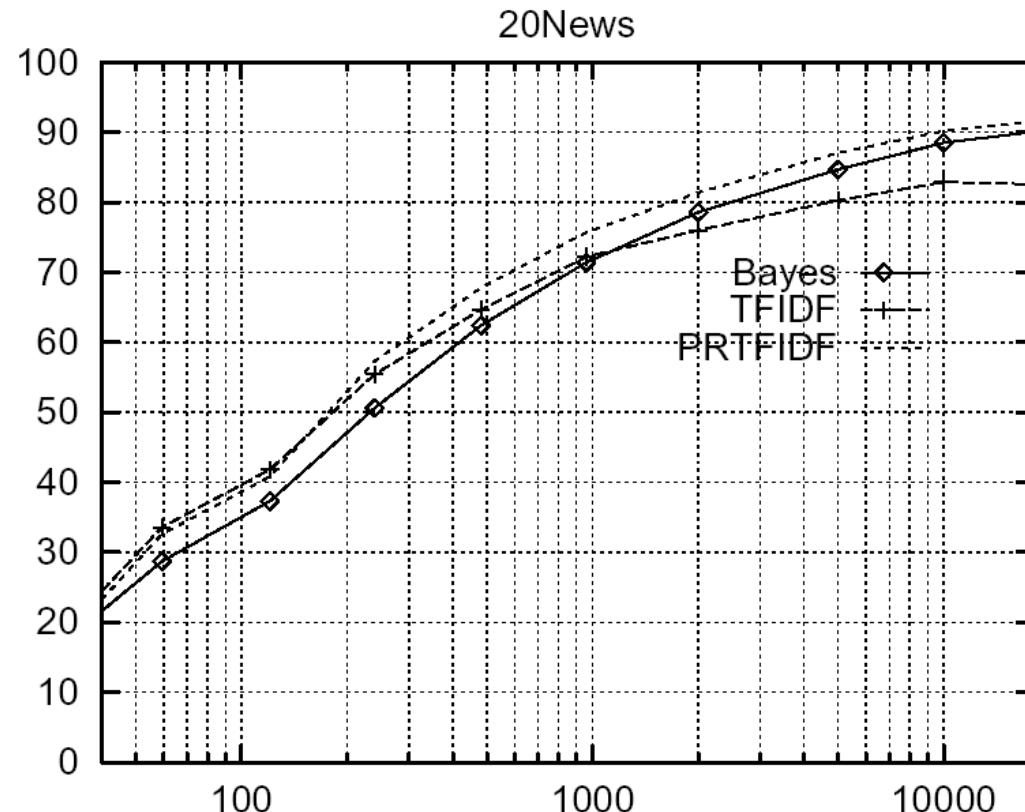
# Twenty NewsGroups

- Given 1000 training documents from each group Learn to classify new documents according to which newsgroup it came from

comp.graphics	misc.forsale	alt.atheism	sci.space
comp.os.ms-windows.misc	rec.autos	soc.religion.christian	sci.crypt
comp.sys.ibm.pc.hardware	rec.motorcycles	talk.religion.misc	sci.electronics
comp.sys.mac.hardware	rec.sport.baseball	talk.politics.mideast	sci.med
comp.windows.x	rec.sport.hockey	talk.politics.misc	
		talk.politics.guns	

- Naive Bayes: 89% classification accuracy

# Learning Curve for 20 Newsgroups



- Accuracy vs. Training set size (1/3 withheld for test)

# Some references for more examples

---

- Movie Review:

[https://www.youtube.com/watch?time\\_continue=16&v=EGKeC2S44Rs](https://www.youtube.com/watch?time_continue=16&v=EGKeC2S44Rs)

- Spam mails by Prof. Andrew Ng:

<https://www.youtube.com/watch?v=z5UQyCESW64>

<https://www.youtube.com/watch?v=NFd0ZQk5bR4>

- NLP by Prof. Dan Jurafsky, Stanford:

<https://www.youtube.com/watch?v=Fmu65a0v6Sw>

# In our next session

---

We will cover:

Bishop

- 4.1 Discriminant Functions,(4.1.1, 4.1.2)
- 4.3 Probabilistic Discriminative Classifiers,
- 4.3.1, 4.3.2 logistic regression
- Difference between Generative and Discriminative classifier



# Machine Learning

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**BITS** Pilani  
Pilani Campus



**Lecture No. – 5 | Probabilistic Discriminative Classifiers**  
**Date – 23/11/2019**  
**Time – 9:00 AM – 11:00 AM**

# Session Content

---

- Review of Naïve Bayes
  - Text classification model, image classification
- Discriminant Functions
- Probabilistic Discriminative Classifiers
- Logistic regression
- Difference between Naïve Bayes Classifier and Logistic Regression

# Naive Bayes Classifier

- Assume target function  $f: X \rightarrow V$ , where each instance  $x$  described by attributes  $\langle a_1, a_2 \dots a_n \rangle$ .
- Most probable value of  $f(x)$  is:

$$\begin{aligned}
 v_{MAP} &= \operatorname{argmax}_{v_j \in V} P(v_j | a_1, a_2 \dots a_n) \\
 v_{MAP} &= \operatorname{argmax}_{v_j \in V} \frac{P(a_1, a_2 \dots a_n | v_j) P(v_j)}{P(a_1, a_2 \dots a_n)} \\
 &= \operatorname{argmax}_{v_j \in V} P(a_1, a_2 \dots a_n | v_j) P(v_j)
 \end{aligned}$$

Naive Bayes assumption:

$$P(a_1, a_2 \dots a_n | v_j) = \prod_i P(a_i | v_j)$$

**Naive Bayes classifier:**

$$v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$$

# Naive Bayes Algorithm

---

- Naive Bayes Learn(*examples*)

For each target value  $v_j$

$$\hat{P}(v_j) \leftarrow \text{estimate } P(v_j)$$

For each attribute value  $a_i$  of each attribute  $a$

$$\hat{P}(a_i | v_j) \leftarrow \text{estimate } P(a_i | v_j)$$

- Classify New Instance( $x$ )

$$v_{NB} = \operatorname{argmax}_{v_j \in V} \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i | v_j)$$

# Example 1

Name	Give Birth	Can Fly	Live in Water	Have Legs	Class
human	yes	no	no	yes	mammals
python	no	no	no	no	non-mammals
salmon	no	no	yes	no	non-mammals
whale	yes	no	yes	no	mammals
frog	no	no	sometimes	yes	non-mammals
komodo	no	no	no	yes	non-mammals
bat	yes	yes	no	yes	mammals
pigeon	no	yes	no	yes	non-mammals
cat	yes	no	no	yes	mammals
leopard shark	yes	no	yes	no	non-mammals
turtle	no	no	sometimes	yes	non-mammals
penguin	no	no	sometimes	yes	non-mammals
porcupine	yes	no	no	yes	mammals
eel	no	no	yes	no	non-mammals
salamander	no	no	sometimes	yes	non-mammals
gila monster	no	no	no	yes	non-mammals
platypus	no	no	no	yes	mammals
owl	no	yes	no	yes	non-mammals
dolphin	yes	no	yes	no	mammals
eagle	no	yes	no	yes	non-mammals

A: attributes

M: mammals

N: non-mammals

$$P(A | M) = \frac{6}{7} \times \frac{6}{7} \times \frac{2}{7} \times \frac{2}{7} = 0.06$$

$$P(A | N) = \frac{1}{13} \times \frac{10}{13} \times \frac{3}{13} \times \frac{4}{13} = 0.0042$$

$$P(A | M)P(M) = 0.06 \times \frac{7}{20} = 0.021$$

$$P(A | N)P(N) = 0.004 \times \frac{13}{20} = 0.0027$$

$$P(A|M)P(M) > P(A|N)P(N)$$

=> Mammals

Give Birth	Can Fly	Live in Water	Have Legs	Class
yes	no	yes	no	?

## Example 2

Text	Tag	Which tag does the sentence <i>A very close game</i> belong to? i.e. $P(\text{sports}   \text{A very close game})$
“A great game”	Sports	Feature Engineering: Bag of words i.e use word frequencies without considering order
“The election was over”	Not sports	
“Very clean match”	Sports	Using Bayes Theorem: $P(\text{sports}   \text{A very close game})$ $= \frac{P(\text{A very close game}   \text{sports}) P(\text{sports})}{P(\text{A very close game})}$ -----
“It was a close election”	Not sports	

We assume that every word in a sentence is **independent** of the other ones

$$P(\text{a very close game}) = P(a) \times P(\text{very}) \times P(\text{close}) \times P(\text{game})$$

$$\begin{aligned} P(\text{a very close game} | \text{Sports}) &= P(a | \text{Sports}) \times P(\text{very} | \text{Sports}) \times \\ &P(\text{close} | \text{Sports}) \times P(\text{game} | \text{Sports}) \end{aligned}$$

“close” doesn’t appear in sentences of sports tag, So  $P(\text{close} | \text{sports}) = 0$ , which makes product 0

# Laplace smoothing

---

- Laplace smoothing: we add 1 or in general constant k to every count so it's never zero.
- To balance this, we add the number of possible words to the divisor, so the division will never be greater than 1
- In our case, the possible words are ['a', 'great', 'very', 'over', 'it', 'but', 'game', 'election', 'clean', 'close', 'the', 'was', 'forgettable', 'match'].

# Apply Laplace Smoothing

Word	P(word   Sports)	P(word   Not Sports)
a	2+1 / 11+14	1+1 / 9+14
very	1+1 / 11+14	0+1 / 9+14
close	0+1 / 11+14	1+1 / 9+14
game	2+1 / 11+14	0+1 / 9+14

$$\begin{aligned}
 & P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports) \times \\
 & P(Sports) \\
 & = 2.76 \times 10^{-5} \\
 & = 0.0000276
 \end{aligned}$$

$$\begin{aligned}
 & P(a|Not\ Sports) \times P(very|Not\ Sports) \times P(close|Not\ Sports) \times \\
 & P(game|Not\ Sports) \times P(Not\ Sports) \\
 & = 0.572 \times 10^{-5} \\
 & = 0.00000572
 \end{aligned}$$

# Learning to Classify Text

---

`LEARN_NAIVE_BAYES_TEXT (Examples,  $V$ )`

**1.** collect all words and other tokens that occur in *Examples*

- $\text{Vocabulary} \leftarrow$  all distinct words and other tokens in *Examples*

**2.** calculate the required  $P(v_j)$  and  $P(w_k \mid v_j)$  probability terms

- For each target value  $v_j$  in  $V$  do
  - $\text{docs}_j \leftarrow$  subset of *Examples* for which the target value is  $v_j$
  - $P(v_j) \leftarrow \frac{|\text{docs}_j|}{|\text{Examples}|}$
  - $\text{Text}_j \leftarrow$  a single document created by concatenating all members of  $\text{docs}_j$

# Learning to Classify Text

- $n \leftarrow$  total number of words in  $Text_j$  (counting duplicate words multiple times)
- for each word  $w_k$  in  $Vocabulary$ 
  - \*  $n_k \leftarrow$  number of times word  $w_k$  occurs in  $Text_j$
  - \*  $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

## CLASSIFY\_NAIVE\_BAYES\_TEXT ( $Doc$ )

- $positions \leftarrow$  all word positions in  $Doc$  that contain tokens found in  $Vocabulary$
- Return  $v_{NB}$  where  $v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_{i \in positions} P(a_i|v_j)$

**LEARN\_NAIVE\_BAYES\_TEXT(*Examples*,  $V$ )**

*Examples* is a set of text documents along with their target values.  $V$  is the set of all possible target values. This function learns the probability terms  $P(w_k|v_j)$ , describing the probability that a randomly drawn word from a document in class  $v_j$  will be the English word  $w_k$ . It also learns the class prior probabilities  $P(v_j)$ .

1. collect all words, punctuation, and other tokens that occur in *Examples*

- *Vocabulary*  $\leftarrow$  the set of all distinct words and other tokens occurring in any text document from *Examples*

2. calculate the required  $P(v_j)$  and  $P(w_k|v_j)$  probability terms

- For each target value  $v_j$  in  $V$  do
  - $docs_j \leftarrow$  the subset of documents from *Examples* for which the target value is  $v_j$
  - $P(v_j) \leftarrow \frac{|docs_j|}{|\text{Examples}|}$
  - $Text_j \leftarrow$  a single document created by concatenating all members of  $docs_j$
  - $n \leftarrow$  total number of distinct word positions in  $Text_j$
  - for each word  $w_k$  in *Vocabulary*
    - $n_k \leftarrow$  number of times word  $w_k$  occurs in  $Text_j$
    - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|\text{Vocabulary}|}$

**CLASSIFY\_NAIVE\_BAYES\_TEXT(*Doc*)**

Return the estimated target value for the document *Doc*.  $a_i$  denotes the word found in the  $i$ th position within *Doc*.

- *positions*  $\leftarrow$  all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return  $v_{NB}$ , where

$$v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_{i \in \text{positions}} P(a_i | v_j)$$

## What if features are continuous?

- E.g., character recognition:  $X_i$  is intensity at  $i$ th pixel
- Gaussian Naïve Bayes (GNB):

$$P(X_i = x | Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{-\frac{(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$



distribution of feature  $X_i$  is Gaussian with a mean and variance that can depend on the label  $y_k$  and which feature  $X_i$  it is



# What if features are continuous?

- E.g., character recognition:  $X_i$  is intensity at  $i$ th pixel
- Gaussian Naïve Bayes (GNB):

$$P(X_i = x|Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{-\frac{(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

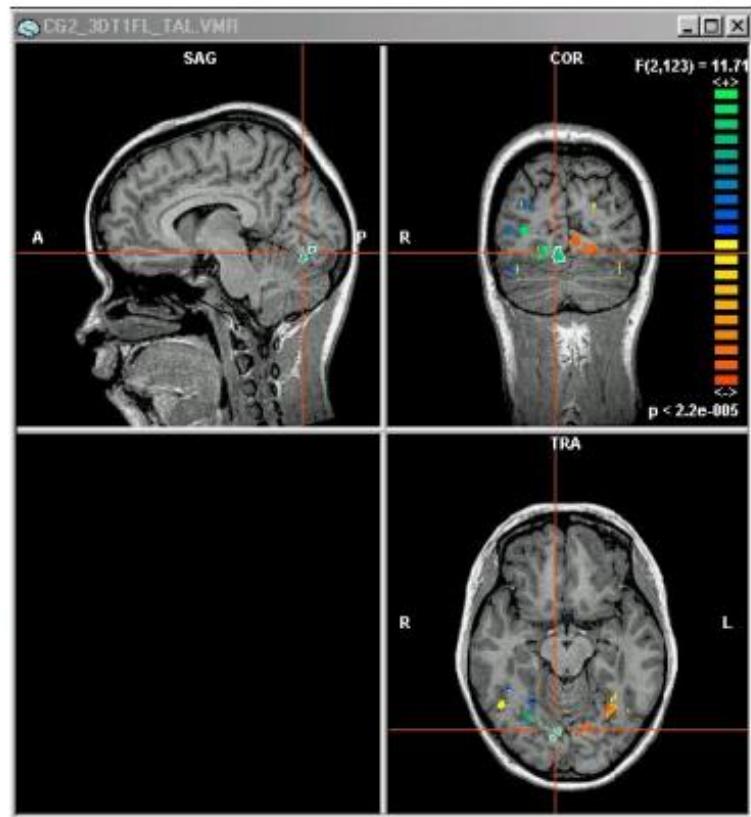
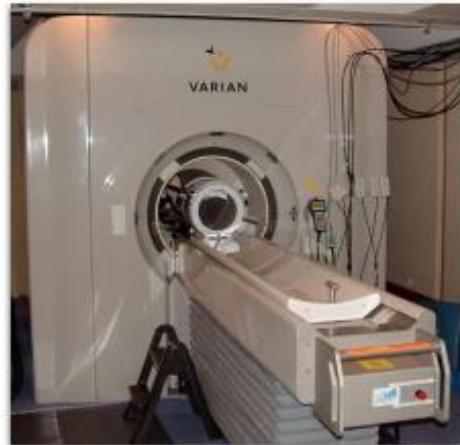


- Different mean and variance for each class  $k$  and each pixel  $i$ .
- Sometimes assume variance:
  - Is independent of  $Y$  (i.e., just have  $\sigma_i$ )
  - Or independent of  $X$  (i.e., just have  $\sigma_k$ )
  - Or both (i.e., just have  $\sigma$ )



# Example: GNB for classifying mental states

[Mitchell et al.]



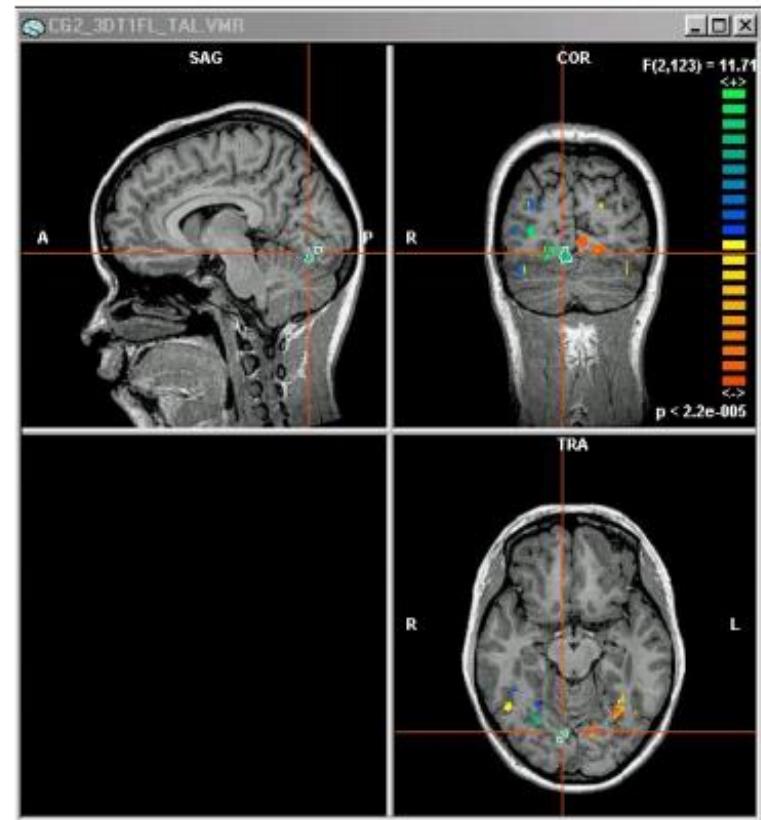
- Classify a person's cognitive state, based on brain image
  - reading a sentence or viewing a picture?
  - reading the word describing a "Tool" or "Building"?
  - reading the word describing a "Person" or an "Animal"?
- Training: Patients were shown words of different categories and then a measurement was done to see what parts of the brain responded.

# Example: GNB for classifying mental states

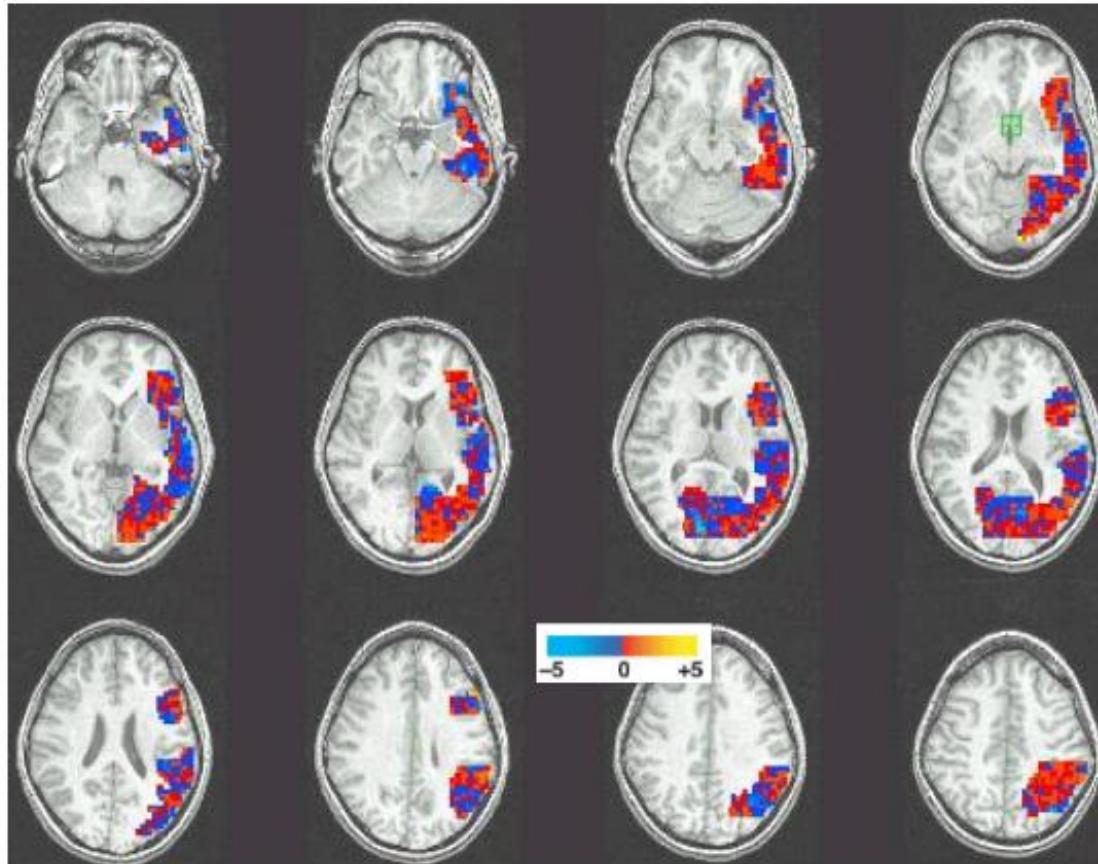
[Mitchell et al.]



- ~1mm resolution
- ~2 images per sec.
- 15,000 voxels/image
- Non-invasive, save
- Measures Blood Oxygen Level Dependent response (BOLD)



# Gaussian Naïve Bayes: Learned $\mu_{\text{voxel}, \text{word}}$



[Mitchell et al.]

15,000 voxels  
or features

10 training  
examples or  
subjects per  
class

---

# Logistic Regression

# Logistic Regression

Idea:

- Naïve Bayes allows computing  $P(Y|X)$  by learning  $P(Y)$  and  $P(X|Y)$
- Why not learn  $P(Y|X)$  directly?

# Linear Regression versus logistic regression

---



- **Linear Regression** could help us predict the student's test score on a scale of 0 - 100. Linear regression predictions are continuous (numbers in a range).
- **Logistic Regression** could help use predict whether the student passed or failed. Logistic regression predictions are discrete (only specific values or categories are allowed). We can also view probability scores underlying the model's classifications.

# Linear Regression versus Logistic Regression

---

Classification requires discrete values:

$$y = 0 \text{ or } 1$$

For linear Regression output values:

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

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

# Sigmoid/Logistic Function

---

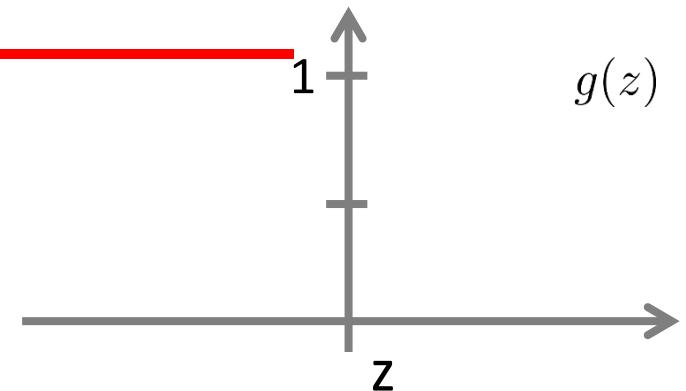
- Sigmoid/logistic function takes a real value as input and outputs another value between 0 and 1
- That framework is called logistic regression
  - Logistic: A special mathematical sigmoid function it uses
  - Regression: Combines a weight vector with observations to create an answer

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

# 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  $h_{\theta}(x) < 0.5$

# Learning model parameters

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

m examples

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

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

How to choose parameters  
(feature weights)  $\theta$  ?

# Error (Cost) Function

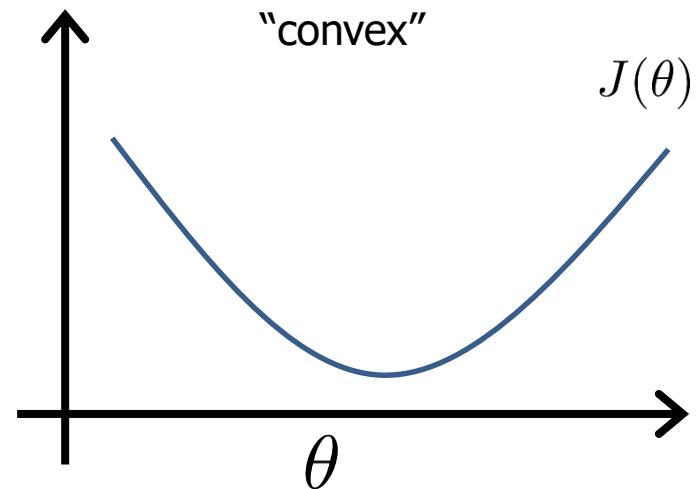
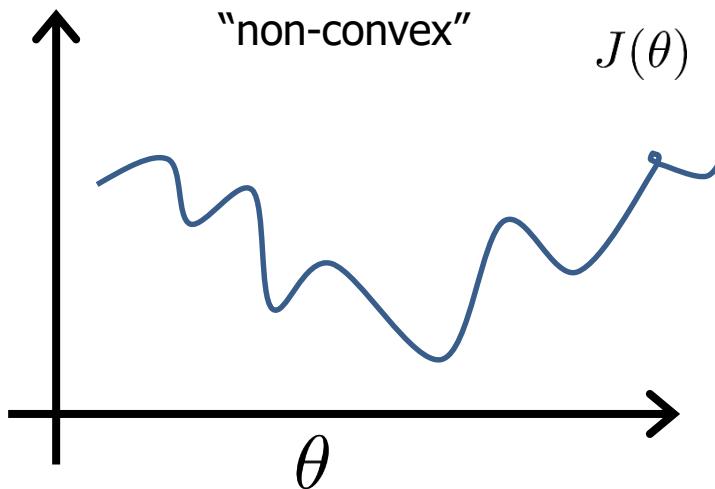
---

- Our prediction function is non-linear (due to sigmoid transform)
- Squaring this prediction as we do in MSE results in a non-convex function with many local minima.
- If our cost function has many local minimums, gradient descent may not find the optimal global minimum.
- So instead of Mean Squared Error, we use a error/cost function called Cross-Entropy, also known as Log Loss.

# MSE Cost Function

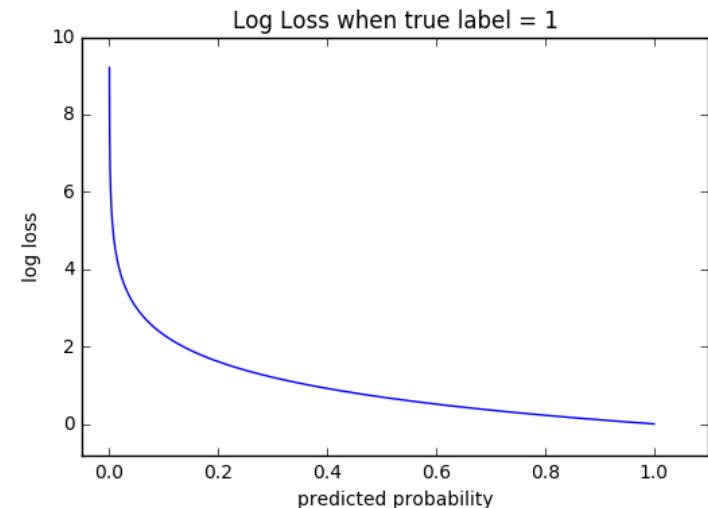
Linear regression:  $J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$

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



# Cross Entropy

- Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.
- Cross-entropy loss increases as the predicted probability diverges from the actual label.
- So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value.
- A perfect model would have a log loss of 0.
- Cross-entropy loss can be divided into two separate cost functions:  
one for  $y=1$  and  
one for  $y=0$ .



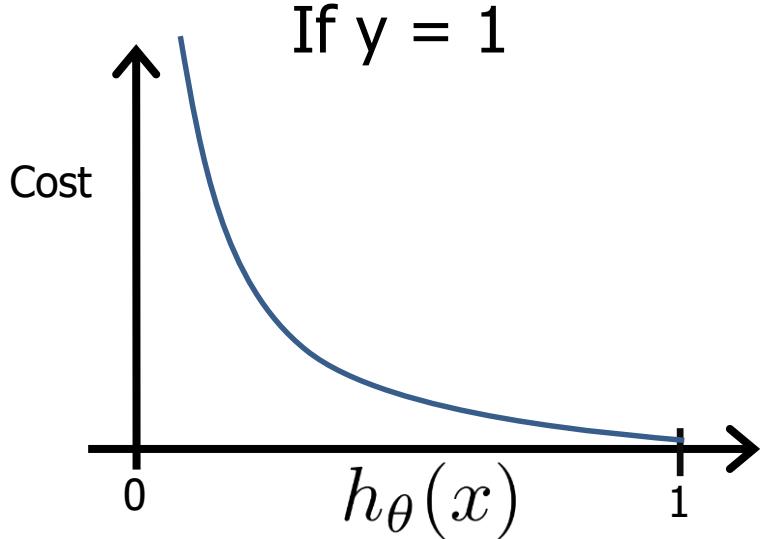
# Logistic regression cost function (cross entropy)

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

If  $y = 1$

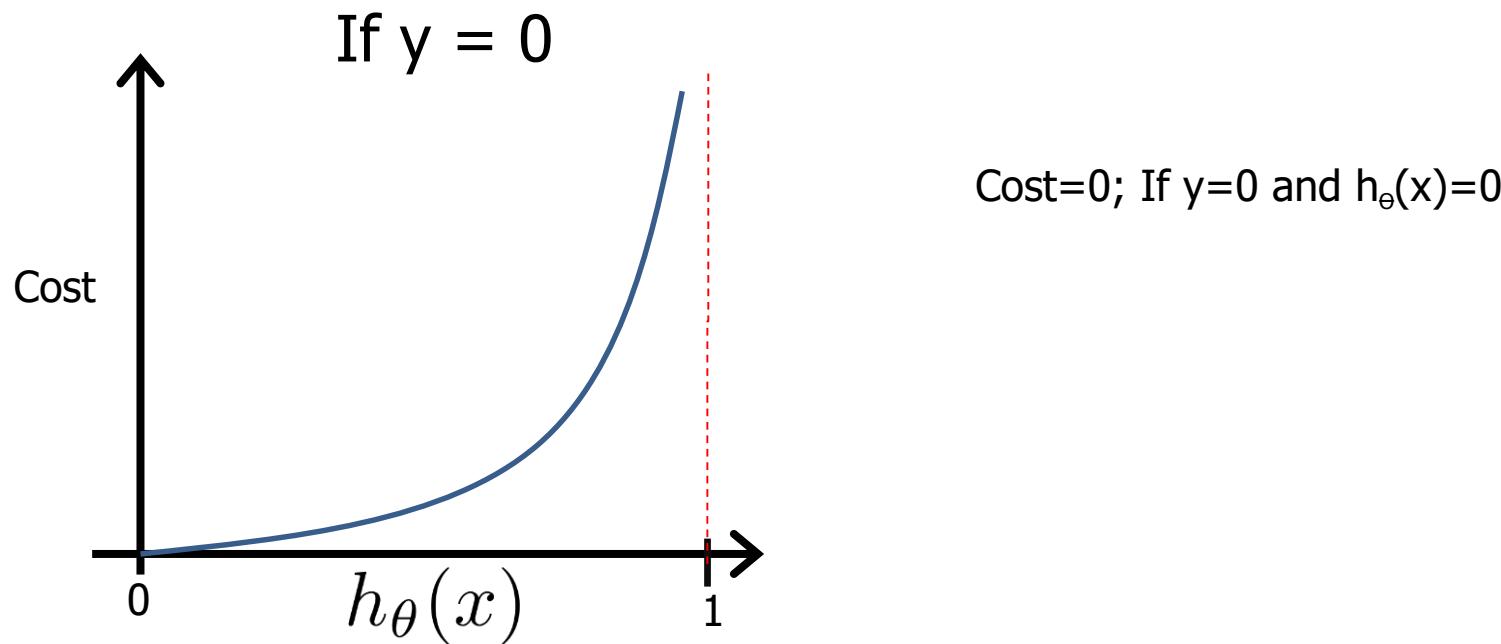
$\text{Cost} = 0$  if  $y = 1, h_\theta(x) = 1$   
 But as  $h_\theta(x) \rightarrow 0$   
 $\text{Cost} \rightarrow \infty$

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

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



# Cost function

$$\begin{aligned} J(\theta) &= \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\ &= -\frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right] \end{aligned}$$

To fit parameters  $\theta$  : [Apply Gradient Descent Algorithm](#)

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

To make a prediction given new  $x$ :

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

# Derivative of sigmoid function

---

- Maximum likelihood to determine the parameters of the logistic regression model.
- To do this, we shall make use of the derivative of the logistic sigmoid function
- Use any algorithm like the gradient descent algorithm to minimize cost function by using derivative

<https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e>

# Logistic Regression

---

- Consider learning  $f: X \rightarrow Y$ , where
  - $X$  is a vector of real-valued features,  $\langle X_1 \dots X_n \rangle$
  - $Y$  is boolean
  - assume all  $X_i$  are conditionally independent given  $Y$
  - model  $P(X_i | Y = y_k)$  as Gaussian  $N(\mu_{ik}, \sigma_i)$
  - model  $P(Y)$  as Bernoulli (two-point) distribution( $\pi$ )
- What does that imply about the form of  $P(Y|X)$ ?

$$P(Y = 1 | X = \langle X_1, \dots, X_n \rangle) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

# Very convenient!

---



---



---


$$P(Y = 1|X = \langle X_1, \dots, X_n \rangle) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

implies

$$P(Y = 0|X = \langle X_1, \dots, X_n \rangle) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

implies

$$\frac{P(Y = 0|X)}{P(Y = 1|X)} = \exp(w_0 + \sum_i w_i X_i)$$

implies

$$\ln \frac{P(Y = 0|X)}{P(Y = 1|X)} = w_0 + \sum_i w_i X_i$$

# Very convenient!

$$P(Y = 1|X = \langle X_1, \dots, X_n \rangle) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

implies

$$P(Y = 0|X = \langle X_1, \dots, X_n \rangle) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

implies

$$\frac{P(Y = 0|X)}{P(Y = 1|X)} = \exp(w_0 + \sum_i w_i X_i)$$

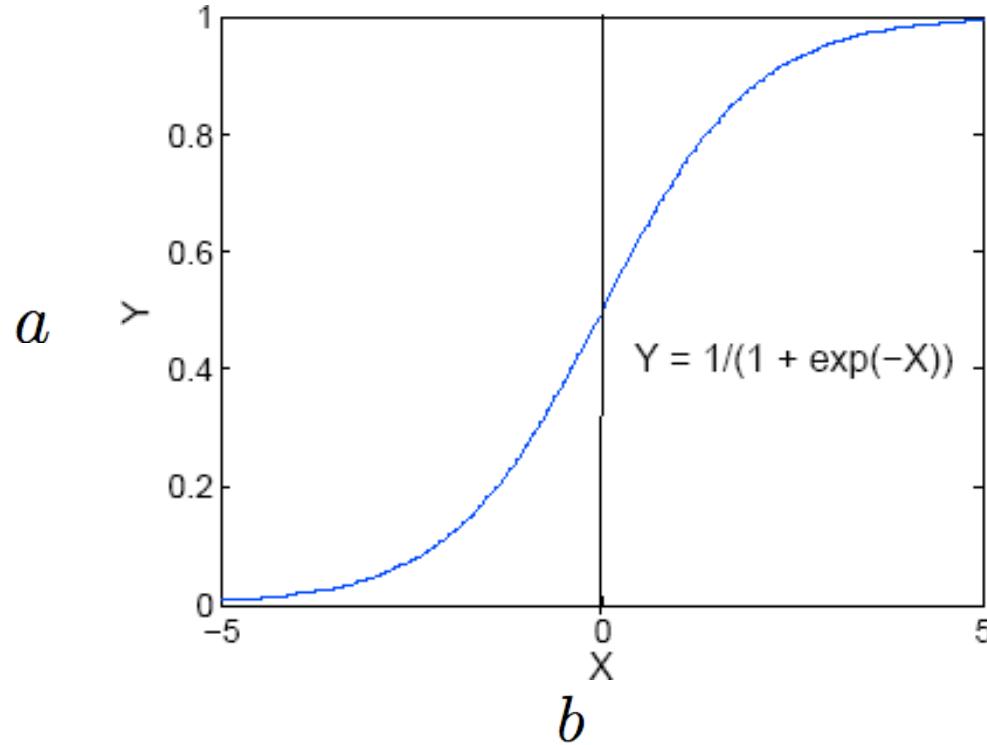
linear  
 classification  
 rule!

implies

$$\ln \frac{P(Y = 0|X)}{P(Y = 1|X)} = w_0 + \sum_i w_i X_i$$

# Logistic function

$$a = \frac{1}{1 + \exp(-b)}$$



$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

# Logistic Regression and Gaussian Naïve Bayes Classifier

---

- Interestingly, the parametric form of  $P(Y|X)$  used by Logistic Regression is precisely the form implied by the assumptions of a Gaussian Naive Bayes classifier.
- Therefore, we can view Logistic Regression as a closely related alternative to GNB, though the two can produce different results in many cases

## Derive form for $P(Y|X)$ for Gaussian $P(X_i|Y=y_k)$ assuming $\sigma_{ik} = \sigma_i$

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)}$$

$$= \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}$$

$$= \frac{1}{1 + \exp(\ln \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

$$P(x | y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

$$= \frac{1}{1 + \exp( (\ln \frac{1-\pi}{\pi}) + \sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})} ; \text{As } P(Y=1) = \pi$$

$$\sum_i \left( \frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right)$$

$$P(Y=1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

# Gaussian $P(X_i | Y = y_k)$

$$\begin{aligned}
 \sum_i \ln \frac{P(X_i | Y = 0)}{P(X_i | Y = 1)} &= \sum_i \ln \frac{\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(X_i - \mu_{i0})^2}{2\sigma_i^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(X_i - \mu_{i1})^2}{2\sigma_i^2}\right)} \\
 &= \sum_i \ln \exp\left(\frac{(X_i - \mu_{i1})^2 - (X_i - \mu_{i0})^2}{2\sigma_i^2}\right) \\
 &= \sum_i \left( \frac{(X_i - \mu_{i1})^2 - (X_i - \mu_{i0})^2}{2\sigma_i^2} \right) \\
 &= \sum_i \left( \frac{(X_i^2 - 2X_i\mu_{i1} + \mu_{i1}^2) - (X_i^2 - 2X_i\mu_{i0} + \mu_{i0}^2)}{2\sigma_i^2} \right) \\
 &= \sum_i \left( \frac{2X_i(\mu_{i0} - \mu_{i1}) + \mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right) \\
 &= \sum_i \left( \frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right)
 \end{aligned}$$

# Parameter estimation of generic logistic regression

---

- Logistic Regression holds in many problem settings beyond the GNB problem
- General method required for estimating it in a more broad range of cases.
- In many cases we may suspect the GNB assumptions are not perfectly satisfied.
- We may wish to estimate the  $w_i$  parameters directly from the data

# Estimating parameters

---

- we have L training examples:  $\{\langle X^1, Y^1 \rangle, \dots \langle X^L, Y^L \rangle\}$
- maximum likelihood estimate for parameters W

$$= \arg \max_W \prod_l P(\langle X^l, Y^l \rangle | W)$$

- maximum conditional likelihood estimate

# Training Logistic Regression: MCLE

- Choose parameters  $W = \langle w_0, \dots, w_n \rangle$  to maximize conditional likelihood of training data

where

$$P(Y = 0|X, W) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1|X, W) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

- Training data  $D = \{\langle X^1, Y^1 \rangle, \dots, \langle X^L, Y^L \rangle\}$
- Data likelihood =  $\prod_l P(X^l, Y^l | W)$
- Data conditional likelihood =  $\prod_l P(Y^l | X^l, W)$

$$W_{MCLE} = \arg \max_W \prod_l P(Y^l | W, X^l)$$

# Expressing Conditional Log Likelihood

---

$$l(W) \equiv \ln \prod_l P(Y^l | X^l, W) = \sum_l \ln P(Y^l | X^l, W)$$

$$P(Y = 0 | X, W) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1 | X, W) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l = 1 | X^l, W) + (1 - Y^l) \ln P(Y^l = 0 | X^l, W) \\ &= \sum_l Y^l \ln \frac{P(Y^l = 1 | X^l, W)}{P(Y^l = 0 | X^l, W)} + \ln P(Y^l = 0 | X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$


---

# Maximizing Conditional Log Likelihood

---

$$P(Y = 0|X, W) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1|X, W) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$\begin{aligned} l(W) &\equiv \ln \prod_l P(Y^l | X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

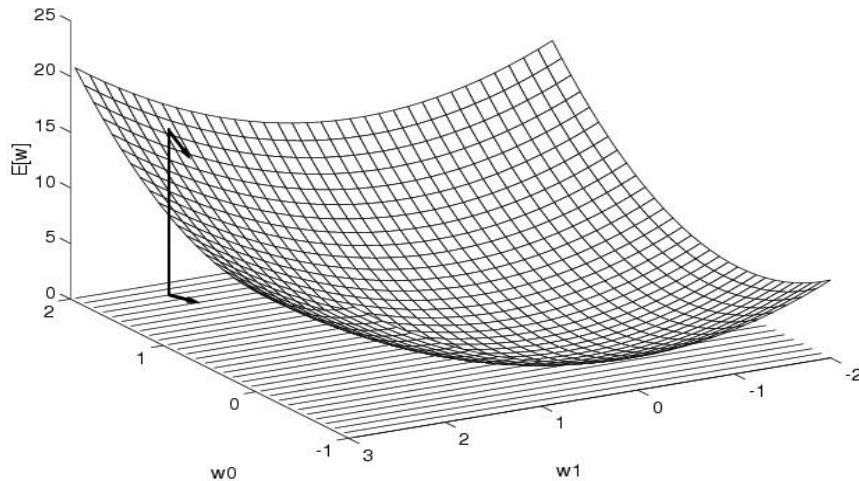
Bad news: no closed-form solution(that can be evaluated in a finite number of operations) to maximize  $l(W)$

# Maximize Conditional Log Likelihood: Gradient Ascent

$$\begin{aligned} l(W) &\equiv \ln \prod_l P(Y^l | X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

# Gradient Descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

# Maximize Conditional Log Likelihood: Gradient Ascent

$$\begin{aligned}
 l(W) &\equiv \ln \prod_l P(Y^l | X^l, W) \\
 &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l))
 \end{aligned}$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

Gradient ascent algorithm: iterate until change  $< \varepsilon$

For all  $i$ , repeat

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

# MAP

---

- Maximum conditional likelihood estimate

$$W \leftarrow \arg \max_W \ln \prod_l P(Y^l | X^l, W)$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

# MAP estimates and Regularization

- Maximum a posteriori estimate

$$W \leftarrow \arg \max_W \ln \prod_l P(Y^l | X^l, W)$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

$$w_i \leftarrow w_i - \eta \lambda w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

$\lambda$  is called a “regularization” term

- helps reduce overfitting
- keep weights nearer to zero
- used very frequently in Logistic Regression

# The Bottom Line

---

- Consider learning  $f: X \rightarrow Y$ , where
  - $X$  is a vector of real-valued features,  $\langle X_1 \dots X_n \rangle$
  - $Y$  is boolean
  - assume all  $X_i$  are conditionally independent given  $Y$
  - model  $P(X_i | Y = y_k)$  as Gaussian  $N(\mu_{ik}, \sigma_i)$
  - model  $P(Y)$  as Bernoulli ( $\pi$ )
- Then  $P(Y|X)$  is of this form, and we can directly estimate  $w$

$$P(Y = 1 | X = \langle X_1, \dots, X_n \rangle) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

# Logistic regression more generally

- Logistic regression when  $Y$  not boolean (but still discrete-valued).
- Now  $y \in \{y_1 \dots y_R\}$  : learn  $R-1$  sets of weights

for  $k < R$     
$$P(Y = y_k | X) = \frac{\exp(w_{k0} + \sum_{i=1}^n w_{ki}X_i)}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji}X_i)}$$

for  $k = R$     
$$P(Y = y_R | X) = \frac{1}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji}X_i)}$$

# How does logistic regression handle missing values?

---



- Replace missing values with column averages (i.e. replace missing values in feature 1 with the average for feature 1).
- Replace missing values with column medians.
- Impute missing values using the other features.
- Remove records that are missing features.
- Use a machine learning technique that uses classification trees, e.g. Decision tree

# Logistic Regression Applications

---

- **Credit Card Fraud** : Predicting if a given credit card transaction is fraud or not
- **Health** : Predicting if a given mass of tissue is benign or malignant
- **Marketing** : Predicting if a given user will buy an insurance product or not
- **Banking** : Predicting if a customer will default on a loan.

# Generative vs. Discriminative Classifiers

---

Training classifiers involves estimating  $f: X \rightarrow Y$ , or  $P(Y|X)$

Generative classifiers (e.g., Naïve Bayes)

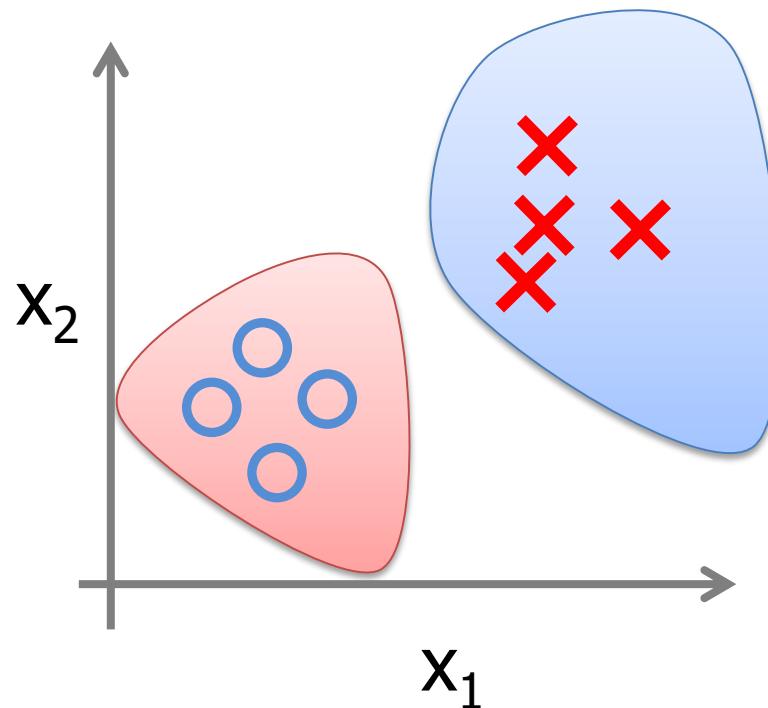
- Assume some functional form for  $P(X|Y)$ ,  $P(X)$
- Estimate parameters of  $P(X|Y)$ ,  $P(X)$  directly from training data
- Use Bayes rule to calculate  $P(Y|X=x_i)$

Discriminative classifiers (e.g., Logistic regression)

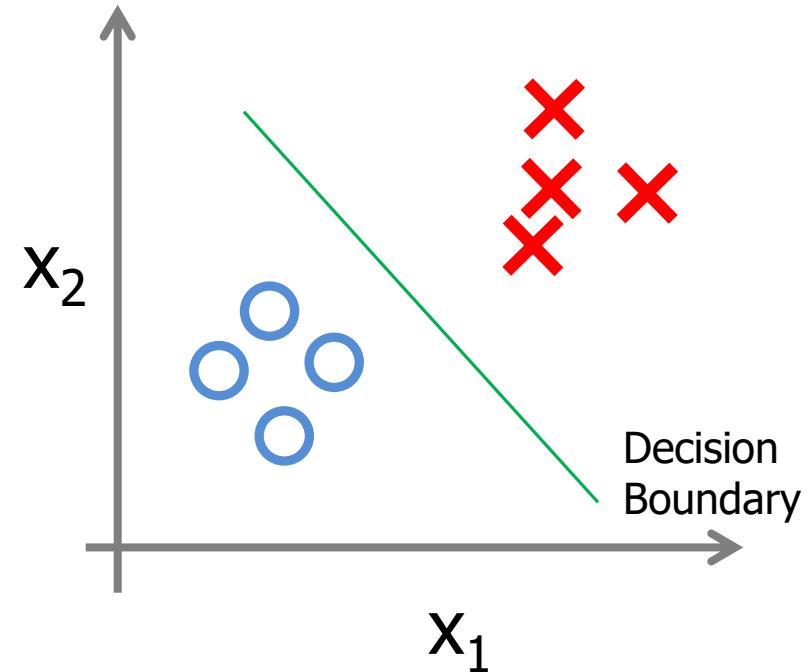
- Assume some functional form for  $P(Y|X)$
- Estimate parameters of  $P(Y|X)$  directly from training data

# Probabilistic Generative Model versus Probabilistic Discriminative Model

Generative:



Discriminative:



# Probabilistic Generative Model versus Probabilistic Discriminative Model

Generative	Discriminative
Ex: Naïve Bayes	Ex: Logistic Regression
Estimate $P(Y)$ and $P(X Y)$	Finds class label directly $P(Y X)$
Prediction $\hat{y} = \text{argmax}_y P(Y = y)P(X = x Y = y)$	Prediction $\hat{y} = P(Y = y X = x)$

# Naïve Bayes versus Logistic Regression

---

- Naïve Bayes are Generative Models
- Logistic Regression are Discriminative Models
- Naïve Bayes easy to construct
- Naive Bayes also assumes that the features are conditionally independent. Real data sets are never perfectly independent
- When the training size reaches infinity, logistic regression performs better than the generative model Naive Bayes.
  - Optional reading by Ng and Jordan has proofs and experiments

# Good references

---

<http://www.cs.cmu.edu/~tom/NewChapters.html>

- <http://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf>
- [https://medium.com/@sangha\\_deb/naive-bayes-vs-logistic-regression-a319b07a5d4c](https://medium.com/@sangha_deb/naive-bayes-vs-logistic-regression-a319b07a5d4c)

# In our next session

---

We will cover:

Linear basis function models

Bias-variance decomposition

Bayesian linear regression

---

## CHAPTER 3

---

# GENERATIVE AND DISCRIMINATIVE CLASSIFIERS: NAIVE BAYES AND LOGISTIC REGRESSION

*Machine Learning*

Copyright ©2015. Tom M. Mitchell. All rights reserved.

\*DRAFT OF September 23, 2017\*

\*PLEASE DO NOT DISTRIBUTE WITHOUT AUTHOR'S PERMISSION\*

This is a rough draft chapter intended for inclusion in the upcoming second edition of the textbook *Machine Learning*, T.M. Mitchell, McGraw Hill. You are welcome to use this for educational purposes, but do not duplicate or repost it on the internet. For online copies of this and other materials related to this book, visit the web site [www.cs.cmu.edu/~tom/mlbook.html](http://www.cs.cmu.edu/~tom/mlbook.html).

Please send suggestions for improvements, or suggested exercises, to [Tom.Mitchell@cmu.edu](mailto:Tom.Mitchell@cmu.edu).

## 1 Learning Classifiers based on Bayes Rule

Here we consider the relationship between supervised learning, or function approximation problems, and Bayesian reasoning. We begin by considering how to design learning algorithms based on Bayes rule.

Consider a supervised learning problem in which we wish to approximate an unknown target function  $f : X \rightarrow Y$ , or equivalently  $P(Y|X)$ . To begin, we will assume  $Y$  is a boolean-valued random variable, and  $X$  is a vector containing  $n$  boolean attributes. In other words,  $X = \langle X_1, X_2 \dots, X_n \rangle$ , where  $X_i$  is the boolean random variable denoting the  $i$ th attribute of  $X$ .

Applying Bayes rule, we see that  $P(Y = y_i|X)$  can be represented as

$$P(Y = y_i|X = x_k) = \frac{P(X = x_k|Y = y_i)P(Y = y_i)}{\sum_j P(X = x_k|Y = y_j)P(Y = y_j)}$$

where  $y_m$  denotes the  $m$ th possible value for  $Y$ ,  $x_k$  denotes the  $k$ th possible vector value for  $X$ , and where the summation in the denominator is over all legal values of the random variable  $Y$ .

One way to learn  $P(Y|X)$  is to use the training data to estimate  $P(X|Y)$  and  $P(Y)$ . We can then use these estimates, together with Bayes rule above, to determine  $P(Y|X = x_k)$  for any new instance  $x_k$ .

**A NOTE ON NOTATION:** We will consistently use upper case symbols (e.g.,  $X$ ) to refer to random variables, including both vector and non-vector variables. If  $X$  is a vector, then we use subscripts (e.g.,  $X_i$  to refer to each random variable, or feature, in  $X$ ). We use lower case symbols to refer to *values* of random variables (e.g.,  $X_i = x_{ij}$  may refer to random variable  $X_i$  taking on its  $j$ th possible value). We will sometimes abbreviate by omitting variable names, for example abbreviating  $P(X_i = x_{ij}|Y = y_k)$  to  $P(x_{ij}|y_k)$ . We will write  $E[X]$  to refer to the expected value of  $X$ . We use superscripts to index training examples (e.g.,  $X_i^j$  refers to the value of the random variable  $X_i$  in the  $j$ th training example.). We use  $\delta(x)$  to denote an “indicator” function whose value is 1 if its logical argument  $x$  is true, and whose value is 0 otherwise. We use the  $\#D\{x\}$  operator to denote the number of elements in the set  $D$  that satisfy property  $x$ . We use a “hat” to indicate estimates; for example,  $\hat{\theta}$  indicates an estimated value of  $\theta$ .

## 1.1 Unbiased Learning of Bayes Classifiers is Impractical

If we are going to train a Bayes classifier by estimating  $P(X|Y)$  and  $P(Y)$ , then it is reasonable to ask how much training data will be required to obtain reliable estimates of these distributions. Let us assume training examples are generated by drawing instances at random from an unknown underlying distribution  $P(X)$ , then allowing a teacher to label this example with its  $Y$  value.

A hundred independently drawn training examples will usually suffice to obtain a maximum likelihood estimate of  $P(Y)$  that is within a few percent of its correct value<sup>1</sup> when  $Y$  is a boolean variable. However, accurately estimating  $P(X|Y)$  typically requires many more examples. To see why, consider the number of parameters we must estimate when  $Y$  is boolean and  $X$  is a vector of  $n$  boolean attributes. In this case, we need to estimate a set of parameters

$$\theta_{ij} \equiv P(X = x_i|Y = y_j)$$

where the index  $i$  takes on  $2^n$  possible values (one for each of the possible vector values of  $X$ ), and  $j$  takes on 2 possible values. Therefore, we will need to estimate approximately  $2^{n+1}$  parameters. To calculate the exact number of required parameters, note for any fixed  $j$ , the sum over  $i$  of  $\theta_{ij}$  must be one. Therefore, for any particular value  $y_j$ , and the  $2^n$  possible values of  $x_i$ , we need compute only  $2^n - 1$  independent parameters. Given the two possible values for  $Y$ , we must estimate a total of  $2(2^n - 1)$  such  $\theta_{ij}$  parameters. Unfortunately, this corresponds to two

---

<sup>1</sup>Why? See Chapter 5 of edition 1 of *Machine Learning*.

distinct parameters for *each* of the distinct instances in the instance space for  $X$ . Worse yet, to obtain reliable estimates of each of these parameters, we will need to observe each of these distinct instances multiple times! This is clearly unrealistic in most practical learning domains. For example, if  $X$  is a vector containing 30 boolean features, then we will need to estimate more than 3 billion parameters.

## 2 Naive Bayes Algorithm

Given the intractable sample complexity for learning Bayesian classifiers, we must look for ways to reduce this complexity. The Naive Bayes classifier does this by making a conditional independence assumption that dramatically reduces the number of parameters to be estimated when modeling  $P(X|Y)$ , from our original  $2(2^n - 1)$  to just  $2n$ .

### 2.1 Conditional Independence

*Definition:* Given three sets of random variables  $X, Y$  and  $Z$ , we say  $X$  is **conditionally independent** of  $Y$  given  $Z$ , if and only if the probability distribution governing  $X$  is independent of the value of  $Y$  given  $Z$ ; that is

$$(\forall i, j, k)P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

As an example, consider three boolean random variables to describe the current weather: *Rain*, *Thunder* and *Lightning*. We might reasonably assert that *Thunder* is independent of *Rain* given *Lightning*. Because we know *Lightning* causes *Thunder*, once we know whether or not there is *Lightning*, no additional information about *Thunder* is provided by the value of *Rain*. Of course there is a clear dependence of *Thunder* on *Rain* in general, but there is no *conditional* dependence once we know the value of *Lightning*. Although  $X, Y$  and  $Z$  are each single random variables in this example, more generally the definition applies to sets of random variables. For example, we might assert that variables  $\{A, B\}$  are conditionally independent of  $\{C, D\}$  given variables  $\{E, F\}$ .

### 2.2 Derivation of Naive Bayes Algorithm

The Naive Bayes algorithm is a classification algorithm based on Bayes rule and a set of conditional independence assumptions. Given the goal of learning  $P(Y|X)$  where  $X = \langle X_1, \dots, X_n \rangle$ , the Naive Bayes algorithm makes the assumption that each  $X_i$  is conditionally independent of each of the other  $X_k$ s given  $Y$ , and also independent of each subset of the other  $X_k$ 's given  $Y$ .

The value of this assumption is that it dramatically simplifies the representation of  $P(X|Y)$ , and the problem of estimating it from the training data. Consider, for example, the case where  $X = \langle X_1, X_2 \rangle$ . In this case

$$\begin{aligned}
P(X|Y) &= P(X_1, X_2|Y) \\
&= P(X_1|X_2, Y)P(X_2|Y) \\
&= P(X_1|Y)P(X_2|Y)
\end{aligned}$$

Where the second line follows from a general property of probabilities, and the third line follows directly from our above definition of conditional independence. More generally, when  $X$  contains  $n$  attributes which satisfy the conditional independence assumption, we have

$$P(X_1 \dots X_n|Y) = \prod_{i=1}^n P(X_i|Y) \quad (1)$$

Notice that when  $Y$  and the  $X_i$  are boolean variables, we need only  $2n$  parameters to define  $P(X_i = x_{ik}|Y = y_j)$  for the necessary  $i, j, k$ . This is a dramatic reduction compared to the  $2(2^n - 1)$  parameters needed to characterize  $P(X|Y)$  if we make no conditional independence assumption.

Let us now derive the Naive Bayes algorithm, assuming in general that  $Y$  is any discrete-valued variable, and the attributes  $X_1 \dots X_n$  are any discrete or real-valued attributes. Our goal is to train a classifier that will output the probability distribution over possible values of  $Y$ , for each new instance  $X$  that we ask it to classify. The expression for the probability that  $Y$  will take on its  $k$ th possible value, according to Bayes rule, is

$$P(Y = y_k|X_1 \dots X_n) = \frac{P(Y = y_k)P(X_1 \dots X_n|Y = y_k)}{\sum_j P(Y = y_j)P(X_1 \dots X_n|Y = y_j)}$$

where the sum is taken over all possible values  $y_j$  of  $Y$ . Now, assuming the  $X_i$  are conditionally independent given  $Y$ , we can use equation (1) to rewrite this as

$$P(Y = y_k|X_1 \dots X_n) = \frac{P(Y = y_k)\prod_i P(X_i|Y = y_k)}{\sum_j P(Y = y_j)\prod_i P(X_i|Y = y_j)} \quad (2)$$

Equation (2) is the fundamental equation for the Naive Bayes classifier. Given a new instance  $X^{new} = \langle X_1 \dots X_n \rangle$ , this equation shows how to calculate the probability that  $Y$  will take on any given value, given the observed attribute values of  $X^{new}$  and given the distributions  $P(Y)$  and  $P(X_i|Y)$  estimated from the training data. If we are interested only in the most probable value of  $Y$ , then we have the Naive Bayes classification rule:

$$Y \leftarrow \arg \max_{y_k} \frac{P(Y = y_k)\prod_i P(X_i|Y = y_k)}{\sum_j P(Y = y_j)\prod_i P(X_i|Y = y_j)}$$

which simplifies to the following (because the denominator does not depend on  $y_k$ ).

$$Y \leftarrow \arg \max_{y_k} P(Y = y_k) \prod_i P(X_i|Y = y_k) \quad (3)$$

## 2.3 Naive Bayes for Discrete-Valued Inputs

To summarize, let us precisely define the Naive Bayes learning algorithm by describing the parameters that must be estimated, and how we may estimate them.

When the  $n$  input attributes  $X_i$  each take on  $J$  possible discrete values, and  $Y$  is a discrete variable taking on  $K$  possible values, then our learning task is to estimate two sets of parameters. The first is

$$\theta_{ijk} \equiv P(X_i = x_{ij} | Y = y_k) \quad (4)$$

for each input attribute  $X_i$ , each of its possible values  $x_{ij}$ , and each of the possible values  $y_k$  of  $Y$ . Note there will be  $nJK$  such parameters, and note also that only  $n(J - 1)K$  of these are independent, given that they must satisfy  $1 = \sum_j \theta_{ijk}$  for each pair of  $i, k$  values.

In addition, we must estimate parameters that define the prior probability over  $Y$ :

$$\pi_k \equiv P(Y = y_k) \quad (5)$$

Note there are  $K$  of these parameters,  $(K - 1)$  of which are independent.

We can estimate these parameters using either maximum likelihood estimates (based on calculating the relative frequencies of the different events in the data), or using Bayesian MAP estimates (augmenting this observed data with prior distributions over the values of these parameters).

Maximum likelihood estimates for  $\theta_{ijk}$  given a set of training examples  $D$  are given by

$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} | Y = y_k) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\}}{\#D\{Y = y_k\}} \quad (6)$$

where the  $\#D\{x\}$  operator returns the number of elements in the set  $D$  that satisfy property  $x$ .

One danger of this maximum likelihood estimate is that it can sometimes result in  $\theta$  estimates of zero, if the data does not happen to contain any training examples satisfying the condition in the numerator. To avoid this, it is common to use a “smoothed” estimate which effectively adds in a number of additional “hallucinated” examples, and which assumes these hallucinated examples are spread evenly over the possible values of  $X_i$ . This smoothed estimate is given by

$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} | Y = y_k) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\} + l}{\#D\{Y = y_k\} + lJ} \quad (7)$$

where  $J$  is the number of distinct values  $X_i$  can take on, and  $l$  determines the strength of this smoothing (i.e., the number of hallucinated examples is  $lJ$ ). This expression corresponds to a MAP estimate for  $\theta_{ijk}$  if we assume a Dirichlet prior distribution over the  $\theta_{ijk}$  parameters, with equal-valued parameters. If  $l$  is set to 1, this approach is called Laplace smoothing.

Maximum likelihood estimates for  $\pi_k$  are

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\}}{|D|} \quad (8)$$

where  $|D|$  denotes the number of elements in the training set  $D$ .

Alternatively, we can obtain a smoothed estimate, or equivalently a MAP estimate based on a Dirichlet prior over the  $\pi_k$  parameters assuming equal priors on each  $\pi_k$ , by using the following expression

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\} + l}{|D| + lK} \quad (9)$$

where  $K$  is the number of distinct values  $Y$  can take on, and  $l$  again determines the strength of the prior assumptions relative to the observed data  $D$ .

## 2.4 Naive Bayes for Continuous Inputs

In the case of continuous inputs  $X_i$ , we can of course continue to use equations (2) and (3) as the basis for designing a Naive Bayes classifier. However, when the  $X_i$  are continuous we must choose some other way to represent the distributions  $P(X_i|Y)$ . One common approach is to assume that for each possible discrete value  $y_k$  of  $Y$ , the distribution of each continuous  $X_i$  is Gaussian, and is defined by a mean and standard deviation specific to  $X_i$  and  $y_k$ . In order to train such a Naive Bayes classifier we must therefore estimate the mean and standard deviation of each of these Gaussians:

$$\mu_{ik} = E[X_i|Y = y_k] \quad (10)$$

$$\sigma_{ik}^2 = E[(X_i - \mu_{ik})^2|Y = y_k] \quad (11)$$

for each attribute  $X_i$  and each possible value  $y_k$  of  $Y$ . Note there are  $2nK$  of these parameters, all of which must be estimated independently.

Of course we must also estimate the priors on  $Y$  as well

$$\pi_k = P(Y = y_k) \quad (12)$$

The above model summarizes a Gaussian Naive Bayes classifier, which assumes that the data  $X$  is generated by a mixture of class-conditional (i.e., dependent on the value of the class variable  $Y$ ) Gaussians. Furthermore, the Naive Bayes assumption introduces the additional constraint that the attribute values  $X_i$  are independent of one another within each of these mixture components. In particular problem settings where we have additional information, we might introduce additional assumptions to further restrict the number of parameters or the complexity of estimating them. For example, if we have reason to believe that noise in the observed  $X_i$  comes from a common source, then we might further assume that all of the  $\sigma_{ik}$  are identical, regardless of the attribute  $i$  or class  $k$  (see the homework exercise on this issue).

Again, we can use either maximum likelihood estimates (MLE) or maximum a posteriori (MAP) estimates for these parameters. The maximum likelihood estimator for  $\mu_{ik}$  is

$$\hat{\mu}_{ik} = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j X_i^j \delta(Y^j = y_k) \quad (13)$$

where the superscript  $j$  refers to the  $j$ th training example, and where  $\delta(Y = y_k)$  is 1 if  $Y = y_k$  and 0 otherwise. Note the role of  $\delta$  here is to select only those training examples for which  $Y = y_k$ .

The maximum likelihood estimator for  $\sigma_{ik}^2$  is

$$\hat{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j (X_i^j - \hat{\mu}_{ik})^2 \delta(Y^j = y_k) \quad (14)$$

This maximum likelihood estimator is biased, so the minimum variance unbiased estimator (MVUE) is sometimes used instead. It is

$$\hat{\sigma}_{ik}^2 = \frac{1}{(\sum_j \delta(Y^j = y_k)) - 1} \sum_j (X_i^j - \hat{\mu}_{ik})^2 \delta(Y^j = y_k) \quad (15)$$

### 3 Logistic Regression

Logistic Regression is an approach to learning functions of the form  $f : X \rightarrow Y$ , or  $P(Y|X)$  in the case where  $Y$  is discrete-valued, and  $X = \langle X_1 \dots X_n \rangle$  is any vector containing discrete or continuous variables. In this section we will primarily consider the case where  $Y$  is a boolean variable, in order to simplify notation. In the final subsection we extend our treatment to the case where  $Y$  takes on any finite number of discrete values.

Logistic Regression assumes a parametric form for the distribution  $P(Y|X)$ , then directly estimates its parameters from the training data. The parametric model assumed by Logistic Regression in the case where  $Y$  is boolean is:

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (16)$$

and

$$P(Y = 0|X) = \frac{\exp(w_0 + \sum_{i=1}^n w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (17)$$

Notice that equation (17) follows directly from equation (16), because the sum of these two probabilities must equal 1.

One highly convenient property of this form for  $P(Y|X)$  is that it leads to a simple linear expression for classification. To classify any given  $X$  we generally want to assign the value  $y_k$  that maximizes  $P(Y = y_k|X)$ . Put another way, we assign the label  $Y = 0$  if the following condition holds:

$$1 < \frac{P(Y = 0|X)}{P(Y = 1|X)}$$

substituting from equations (16) and (17), this becomes

$$1 < \exp(w_0 + \sum_{i=1}^n w_i X_i)$$

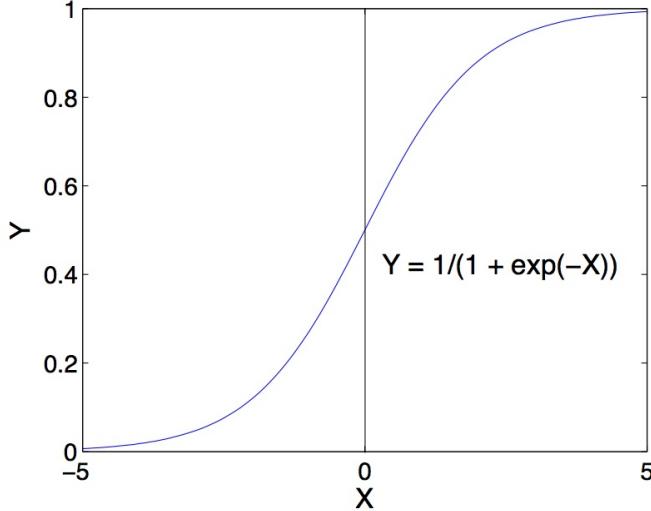


Figure 1: Form of the logistic function. In Logistic Regression,  $P(Y|X)$  is assumed to follow this form.

and taking the natural log of both sides we have a linear classification rule that assigns label  $Y = 0$  if  $X$  satisfies

$$0 < w_0 + \sum_{i=1}^n w_i X_i \quad (18)$$

and assigns  $Y = 1$  otherwise.

Interestingly, the parametric form of  $P(Y|X)$  used by Logistic Regression is precisely the form implied by the assumptions of a Gaussian Naive Bayes classifier. Therefore, we can view Logistic Regression as a closely related alternative to GNB, though the two can produce different results in many cases.

### 3.1 Form of $P(Y|X)$ for Gaussian Naive Bayes Classifier

Here we derive the form of  $P(Y|X)$  entailed by the assumptions of a Gaussian Naive Bayes (GNB) classifier, showing that it is precisely the form used by Logistic Regression and summarized in equations (16) and (17). In particular, consider a GNB based on the following modeling assumptions:

- $Y$  is boolean, governed by a Bernoulli distribution, with parameter  $\pi = P(Y = 1)$
- $X = \langle X_1 \dots X_n \rangle$ , where each  $X_i$  is a continuous random variable
- For each  $X_i$ ,  $P(X_i|Y = y_k)$  is a Gaussian distribution of the form  $N(\mu_{ik}, \sigma_i)$
- For all  $i$  and  $j \neq i$ ,  $X_i$  and  $X_j$  are conditionally independent given  $Y$

Note here we are assuming the standard deviations  $\sigma_i$  vary from attribute to attribute, but do not depend on  $Y$ .

We now derive the parametric form of  $P(Y|X)$  that follows from this set of GNB assumptions. In general, Bayes rule allows us to write

$$P(Y = 1|X) = \frac{P(Y = 1)P(X|Y = 1)}{P(Y = 1)P(X|Y = 1) + P(Y = 0)P(X|Y = 0)}$$

Dividing both the numerator and denominator by the numerator yields:

$$P(Y = 1|X) = \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}$$

or equivalently

$$P(Y = 1|X) = \frac{1}{1 + \exp(\ln \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

Because of our conditional independence assumption we can write this

$$\begin{aligned} P(Y = 1|X) &= \frac{1}{1 + \exp(\ln \frac{P(Y=0)}{P(Y=1)} + \sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})} \\ &= \frac{1}{1 + \exp(\ln \frac{1-\pi}{\pi} + \sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})} \end{aligned} \quad (19)$$

Note the final step expresses  $P(Y = 0)$  and  $P(Y = 1)$  in terms of the binomial parameter  $\pi$ .

Now consider just the summation in the denominator of equation (19). Given our assumption that  $P(X_i|Y = y_k)$  is Gaussian, we can expand this term as follows:

$$\begin{aligned} \sum_i \ln \frac{P(X_i|Y = 0)}{P(X_i|Y = 1)} &= \sum_i \ln \frac{\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(X_i - \mu_{i0})^2}{2\sigma_i^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(X_i - \mu_{i1})^2}{2\sigma_i^2}\right)} \\ &= \sum_i \ln \exp\left(\frac{(X_i - \mu_{i1})^2 - (X_i - \mu_{i0})^2}{2\sigma_i^2}\right) \\ &= \sum_i \left( \frac{(X_i - \mu_{i1})^2 - (X_i - \mu_{i0})^2}{2\sigma_i^2} \right) \\ &= \sum_i \left( \frac{(X_i^2 - 2X_i\mu_{i1} + \mu_{i1}^2) - (X_i^2 - 2X_i\mu_{i0} + \mu_{i0}^2)}{2\sigma_i^2} \right) \\ &= \sum_i \left( \frac{2X_i(\mu_{i0} - \mu_{i1}) + \mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right) \\ &= \sum_i \left( \frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right) \end{aligned} \quad (20)$$

Note this expression is a linear weighted sum of the  $X_i$ 's. Substituting expression (20) back into equation (19), we have

$$P(Y = 1|X) = \frac{1}{1 + \exp(\ln \frac{1-\pi}{\pi} + \sum_i \left( \frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right))} \quad (21)$$

Or equivalently,

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (22)$$

where the weights  $w_1 \dots w_n$  are given by

$$w_i = \frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2}$$

and where

$$w_0 = \ln \frac{1-\pi}{\pi} + \sum_i \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}$$

Also we have

$$P(Y = 0|X) = 1 - P(Y = 1|X) = \frac{\exp(w_0 + \sum_{i=1}^n w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (23)$$

### 3.2 Estimating Parameters for Logistic Regression

The above subsection proves that  $P(Y|X)$  can be expressed in the parametric form given by equations (16) and (17), under the Gaussian Naive Bayes assumptions detailed there. It also provides the value of the weights  $w_i$  in terms of the parameters estimated by the GNB classifier. Here we describe an alternative method for estimating these weights. We are interested in this alternative for two reasons. First, the form of  $P(Y|X)$  assumed by Logistic Regression holds in many problem settings beyond the GNB problem detailed in the above section, and we wish to have a general method for estimating it in a more broad range of cases. Second, in many cases we may suspect the GNB assumptions are not perfectly satisfied. In this case we may wish to estimate the  $w_i$  parameters directly from the data, rather than going through the intermediate step of estimating the GNB parameters which forces us to adopt its more stringent modeling assumptions.

One reasonable approach to training Logistic Regression is to choose parameter values that maximize the conditional data likelihood. The conditional data likelihood is the probability of the observed  $Y$  values in the training data, conditioned on their corresponding  $X$  values. We choose parameters  $W$  that satisfy

$$W \leftarrow \arg \max_W \prod_l P(Y^l | X^l, W)$$

where  $W = \langle w_0, w_1 \dots w_n \rangle$  is the vector of parameters to be estimated,  $Y^l$  denotes the observed value of  $Y$  in the  $l$ th training example, and  $X^l$  denotes the observed

value of  $X$  in the  $l$ th training example. The expression to the right of the arg max is the conditional data likelihood. Here we include  $W$  in the conditional, to emphasize that the expression is a function of the  $W$  we are attempting to maximize.

Equivalently, we can work with the log of the conditional likelihood:

$$W \leftarrow \arg \max_W \sum_l \ln P(Y^l | X^l, W)$$

This conditional data log likelihood, which we will denote  $l(W)$  can be written as

$$l(W) = \sum_l Y^l \ln P(Y^l = 1 | X^l, W) + (1 - Y^l) \ln P(Y^l = 0 | X^l, W)$$

Note here we are utilizing the fact that  $Y$  can take only values 0 or 1, so only one of the two terms in the expression will be non-zero for any given  $Y^l$ .

To keep our derivation consistent with common usage, we will in this section flip the assignment of the boolean variable  $Y$  so that we assign

$$P(Y = 0 | X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (24)$$

and

$$P(Y = 1 | X) = \frac{\exp(w_0 + \sum_{i=1}^n w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (25)$$

In this case, we can reexpress the log of the conditional likelihood as:

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l = 1 | X^l, W) + (1 - Y^l) \ln P(Y^l = 0 | X^l, W) \\ &= \sum_l Y^l \ln \frac{P(Y^l = 1 | X^l, W)}{P(Y^l = 0 | X^l, W)} + \ln P(Y^l = 0 | X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

where  $X_i^l$  denotes the value of  $X_i$  for the  $l$ th training example. Note the superscript  $l$  is not related to the log likelihood function  $l(W)$ .

Unfortunately, there is no closed form solution to maximizing  $l(W)$  with respect to  $W$ . Therefore, one common approach is to use gradient ascent, in which we work with the gradient, which is the vector of partial derivatives. The  $i$ th component of the vector gradient has the form

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

where  $\hat{P}(Y^l | X^l, W)$  is the Logistic Regression prediction using equations (24) and (25) and the weights  $W$ . To accommodate weight  $w_0$ , we assume an imaginary  $X_0 = 1$  for all  $l$ . This expression for the derivative has an intuitive interpretation: the term inside the parentheses is simply the prediction error; that is, the difference

between the observed  $Y^l$  and its predicted probability! Note if  $Y^l = 1$  then we wish for  $\hat{P}(Y^l = 1|X^l, W)$  to be 1, whereas if  $Y^l = 0$  then we prefer that  $\hat{P}(Y^l = 1|X^l, W)$  be 0 (which makes  $\hat{P}(Y^l = 0|X^l, W)$  equal to 1). This error term is multiplied by the value of  $X_i^l$ , which accounts for the magnitude of the  $w_i X_i^l$  term in making this prediction.

Given this formula for the derivative of each  $w_i$ , we can use standard gradient ascent to optimize the weights  $W$ . Beginning with initial weights of zero, we repeatedly update the weights in the direction of the gradient, on each iteration changing every weight  $w_i$  according to

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

where  $\eta$  is a small constant (e.g., 0.01) which determines the step size. Because the conditional log likelihood  $l(W)$  is a concave function in  $W$ , this gradient ascent procedure will converge to a global maximum. Gradient ascent is described in greater detail, for example, in Chapter 4 of Mitchell (1997). In many cases where computational efficiency is important it is common to use a variant of gradient ascent called conjugate gradient ascent, which often converges more quickly.

### 3.3 Regularization in Logistic Regression

Overfitting the training data is a problem that can arise in Logistic Regression, especially when data is very high dimensional and training data is sparse. One approach to reducing overfitting is *regularization*, in which we create a modified “penalized log likelihood function,” which penalizes large values of  $W$ . One approach is to use the penalized log likelihood function

$$W \leftarrow \arg \max_W \sum_l \ln P(Y^l|X^l, W) - \frac{\lambda}{2} \|W\|^2$$

which adds a penalty proportional to the squared magnitude of  $W$ . Here  $\lambda$  is a constant that determines the strength of this penalty term.

Modifying our objective by adding in this penalty term gives us a new objective to maximize. It is easy to show that maximizing it corresponds to calculating the MAP estimate for  $W$  under the assumption that the prior distribution  $P(W)$  is a Normal distribution with mean zero, and a variance related to  $1/\lambda$ . Notice that in general, the MAP estimate for  $W$  involves optimizing the objective

$$\sum_l \ln P(Y^l|X^l, W) + \ln P(W)$$

and if  $P(W)$  is a zero mean Gaussian distribution, then  $\ln P(W)$  yields a term proportional to  $\|W\|^2$ .

Given this penalized log likelihood function, it is easy to rederive the gradient descent rule. The derivative of this penalized log likelihood function is similar to

our earlier derivative, with one additional penalty term

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W)) - \lambda w_i$$

which gives us the modified gradient descent rule

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W)) - \eta \lambda w_i \quad (26)$$

In cases where we have prior knowledge about likely values for specific  $w_i$ , it is possible to derive a similar penalty term by using a Normal prior on  $W$  with a non-zero mean.

### 3.4 Logistic Regression for Functions with Many Discrete Values

Above we considered using Logistic Regression to learn  $P(Y|X)$  only for the case where  $Y$  is a boolean variable. More generally, if  $Y$  can take on any of the discrete values  $\{y_1, \dots, y_K\}$ , then the form of  $P(Y = y_k|X)$  for  $Y = y_1, Y = y_2, \dots, Y = y_{K-1}$  is:

$$P(Y = y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^n w_{ki} X_i)}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji} X_i)} \quad (27)$$

When  $Y = y_K$ , it is

$$P(Y = y_K|X) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji} X_i)} \quad (28)$$

Here  $w_{ji}$  denotes the weight associated with the  $j$ th class  $Y = y_j$  and with input  $X_i$ . It is easy to see that our earlier expressions for the case where  $Y$  is boolean (equations (16) and (17)) are a special case of the above expressions. Note also that the form of the expression for  $P(Y = y_K|X)$  assures that  $[\sum_{k=1}^K P(Y = y_k|X)] = 1$ .

The primary difference between these expressions and those for boolean  $Y$  is that when  $Y$  takes on  $K$  possible values, we construct  $K - 1$  different linear expressions to capture the distributions for the different values of  $Y$ . The distribution for the final,  $K$ th, value of  $Y$  is simply one minus the probabilities of the first  $K - 1$  values.

In this case, the gradient descent rule with regularization becomes:

$$w_{ji} \leftarrow w_{ji} + \eta \sum_l X_i^l (\delta(Y^l = y_j) - \hat{P}(Y^l = y_j | X^l, W)) - \eta \lambda w_{ji} \quad (29)$$

where  $\delta(Y^l = y_j) = 1$  if the  $l$ th training value,  $Y^l$ , is equal to  $y_j$ , and  $\delta(Y^l = y_j) = 0$  otherwise. Note our earlier learning rule, equation (26), is a special case of this new learning rule, when  $K = 2$ . As in the case for  $K = 2$ , the quantity inside the parentheses can be viewed as an error term which goes to zero if the estimated conditional probability  $\hat{P}(Y^l = y_j | X^l, W)$  perfectly matches the observed value of  $Y^l$ .

## 4 Relationship Between Naive Bayes Classifiers and Logistic Regression

To summarize, Logistic Regression directly estimates the parameters of  $P(Y|X)$ , whereas Naive Bayes directly estimates parameters for  $P(Y)$  and  $P(X|Y)$ . We often call the former a discriminative classifier, and the latter a generative classifier.

We showed above that the assumptions of one variant of a Gaussian Naive Bayes classifier imply the parametric form of  $P(Y|X)$  used in Logistic Regression. Furthermore, we showed that the parameters  $w_i$  in Logistic Regression can be expressed in terms of the Gaussian Naive Bayes parameters. In fact, if the GNB assumptions hold, then asymptotically (as the number of training examples grows toward infinity) the GNB and Logistic Regression converge toward identical classifiers.

The two algorithms also differ in interesting ways:

- When the GNB modeling assumptions do not hold, Logistic Regression and GNB typically learn different classifier functions. In this case, the asymptotic (as the number of training examples approach infinity) classification accuracy for Logistic Regression is often better than the asymptotic accuracy of GNB. Although Logistic Regression is consistent with the Naive Bayes assumption that the input features  $X_i$  are conditionally independent given  $Y$ , it is not rigidly tied to this assumption as is Naive Bayes. Given data that disobeys this assumption, the conditional likelihood maximization algorithm for Logistic Regression will adjust its parameters to maximize the fit to (the conditional likelihood of) the data, even if the resulting parameters are inconsistent with the Naive Bayes parameter estimates.
- GNB and Logistic Regression converge toward their asymptotic accuracies at different rates. As Ng & Jordan (2002) show, GNB parameter estimates converge toward their asymptotic values in order  $\log n$  examples, where  $n$  is the dimension of  $X$ . In contrast, Logistic Regression parameter estimates converge more slowly, requiring order  $n$  examples. The authors also show that in several data sets Logistic Regression outperforms GNB when many training examples are available, but GNB outperforms Logistic Regression when training data is scarce.

## 5 What You Should Know

The main points of this chapter include:

- We can use Bayes rule as the basis for designing learning algorithms (function approximators), as follows: Given that we wish to learn some target function  $f : X \rightarrow Y$ , or equivalently,  $P(Y|X)$ , we use the training data to learn estimates of  $P(X|Y)$  and  $P(Y)$ . New  $X$  examples can then be classified using these estimated probability distributions, plus Bayes rule. This

type of classifier is called a *generative* classifier, because we can view the distribution  $P(X|Y)$  as describing how to generate random instances  $X$  conditioned on the target attribute  $Y$ .

- Learning Bayes classifiers typically requires an unrealistic number of training examples (i.e., more than  $|X|$  training examples where  $X$  is the instance space) unless some form of prior assumption is made about the form of  $P(X|Y)$ . The *Naive Bayes* classifier assumes all attributes describing  $X$  are conditionally independent given  $Y$ . This assumption dramatically reduces the number of parameters that must be estimated to learn the classifier. Naive Bayes is a widely used learning algorithm, for both discrete and continuous  $X$ .
- When  $X$  is a vector of discrete-valued attributes, Naive Bayes learning algorithms can be viewed as linear classifiers; that is, every such Naive Bayes classifier corresponds to a hyperplane decision surface in  $X$ . The same statement holds for Gaussian Naive Bayes classifiers if the variance of each feature is assumed to be independent of the class (i.e., if  $\sigma_{ik} = \sigma_i$ ).
- Logistic Regression is a function approximation algorithm that uses training data to directly estimate  $P(Y|X)$ , in contrast to Naive Bayes. In this sense, Logistic Regression is often referred to as a *discriminative* classifier because we can view the distribution  $P(Y|X)$  as directly discriminating the value of the target value  $Y$  for any given instance  $X$ .
- Logistic Regression is a linear classifier over  $X$ . The linear classifiers produced by Logistic Regression and Gaussian Naive Bayes are identical in the limit as the number of training examples approaches infinity, *provided* the Naive Bayes assumptions hold. However, if these assumptions do not hold, the Naive Bayes bias will cause it to perform less accurately than Logistic Regression, in the limit. Put another way, Naive Bayes is a learning algorithm with greater bias, but lower variance, than Logistic Regression. If this bias is appropriate given the actual data, Naive Bayes will be preferred. Otherwise, Logistic Regression will be preferred.
- We can view function approximation learning algorithms as statistical estimators of functions, or of conditional distributions  $P(Y|X)$ . They estimate  $P(Y|X)$  from a sample of training data. As with other statistical estimators, it can be useful to characterize learning algorithms by their bias and expected variance, taken over different samples of training data.

## 6 Further Reading

Wasserman (2004) describes a Reweighted Least Squares method for Logistic Regression. Ng and Jordan (2002) provide a theoretical and experimental comparison of the Naive Bayes classifier and Logistic Regression.

## EXERCISES

1. At the beginning of the chapter we remarked that “A hundred training examples will usually suffice to obtain an estimate of  $P(Y)$  that is within a few percent of the correct value.” Describe conditions under which the 95% confidence interval for our estimate of  $P(Y)$  will be  $\pm 0.02$ .
2. Consider learning a function  $X \rightarrow Y$  where  $Y$  is boolean, where  $X = \langle X_1, X_2 \rangle$ , and where  $X_1$  is a boolean variable and  $X_2$  a continuous variable. State the parameters that must be estimated to define a Naive Bayes classifier in this case. Give the formula for computing  $P(Y|X)$ , in terms of these parameters and the feature values  $X_1$  and  $X_2$ .
3. In section 3 we showed that when  $Y$  is Boolean and  $X = \langle X_1 \dots X_n \rangle$  is a vector of continuous variables, then the assumptions of the Gaussian Naive Bayes classifier imply that  $P(Y|X)$  is given by the logistic function with appropriate parameters  $W$ . In particular:

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

and

$$P(Y = 0|X) = \frac{\exp(w_0 + \sum_{i=1}^n w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

Consider instead the case where  $Y$  is Boolean and  $X = \langle X_1 \dots X_n \rangle$  is a vector of *Boolean* variables. Prove for this case also that  $P(Y|X)$  follows this same form (and hence that Logistic Regression is also the discriminative counterpart to a Naive Bayes generative classifier over Boolean features).

*Hints:*

- Simple notation will help. Since the  $X_i$  are Boolean variables, you need only one parameter to define  $P(X_i|Y = y_k)$ . Define  $\theta_{i1} \equiv P(X_i = 1|Y = 1)$ , in which case  $P(X_i = 0|Y = 1) = (1 - \theta_{i1})$ . Similarly, use  $\theta_{i0}$  to denote  $P(X_i = 1|Y = 0)$ .
- Notice with the above notation you can represent  $P(X_i|Y = 1)$  as follows

$$P(X_i|Y = 1) = \theta_{i1}^{X_i} (1 - \theta_{i1})^{(1-X_i)}$$

Note when  $X_i = 1$  the second term is equal to 1 because its exponent is zero. Similarly, when  $X_i = 0$  the first term is equal to 1 because its exponent is zero.

4. (based on a suggestion from Sandra Zilles). This question asks you to consider the relationship between the MAP hypothesis and the Bayes optimal hypothesis. Consider a hypothesis space  $H$  defined over the set of instances  $X$ , and containing just two hypotheses,  $h1$  and  $h2$  with equal prior probabilities  $P(h1) = P(h2) = 0.5$ . Suppose we are given an arbitrary set of training

data  $D$  which we use to calculate the posterior probabilities  $P(h1|D)$  and  $P(h2|D)$ . Based on this we choose the MAP hypothesis, and calculate the Bayes optimal hypothesis. Suppose we find that the Bayes optimal classifier is not equal to either  $h1$  or to  $h2$ , which is generally the case because the Bayes optimal hypothesis corresponds to “averaging over” all hypotheses in  $H$ . Now we create a new hypothesis  $h3$  which is equal to the Bayes optimal classifier with respect to  $H$ ,  $X$  and  $D$ ; that is,  $h3$  classifies each instance in  $X$  exactly the same as the Bayes optimal classifier for  $H$  and  $D$ . We now create a new hypothesis space  $H' = \{h1, h2, h3\}$ . If we train using the same training data,  $D$ , will the MAP hypothesis from  $H'$  be  $h3$ ? Will the Bayes optimal classifier with respect to  $H'$  be equivalent to  $h3$ ? (Hint: the answer depends on the priors we assign to the hypotheses in  $H'$ . Can you give constraints on these priors that assure the answers will be yes or no?)

## 7 Acknowledgements

I very much appreciate receiving helpful comments on earlier drafts of this chapter from the following: Nathaniel Fairfield, Rainer Gemulla, Vineet Kumar, Andrew McCallum, Anand Prahlad, Wei Wang, Geoff Webb, and Sandra Zilles.

## REFERENCES

- Mitchell, T (1997). *Machine Learning*, McGraw Hill.
- Ng, A.Y. & Jordan, M. I. (2002). On Discriminative vs. Generative Classifiers: A comparison of Logistic Regression and Naive Bayes, *Neural Information Processing Systems*, Ng, A.Y., and Jordan, M. (2002).
- Wasserman, L. (2004). *All of Statistics*, Springer-Verlag.



# Machine Learning

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**BITS** Pilani  
Pilani Campus



**Lecture No. – 6 | Linear Regression**

**Date – 30/11/2019**

**Time – 11:00 AM – 1:00 PM**

# Session Content

---

- Regression (Andre Ng Notes)
- Bayesian linear regression (6.4 Tom Mitchell)
- Linear basis function models (3.1 Bishop)
- Bias-variance decomposition (3.2 Bishop)

# Regression

---

So far, we've been interested in learning  $P(Y|X)$  where  $Y$  has discrete values (called 'classification')

What if  $Y$  is continuous? (called 'regression')

- predict weight from gender, height, age, ...
- predict Google stock price today from Google, Yahoo, MSFT prices yesterday
- predict each pixel intensity in robot's current camera image, from previous image and previous action

# Regression

---

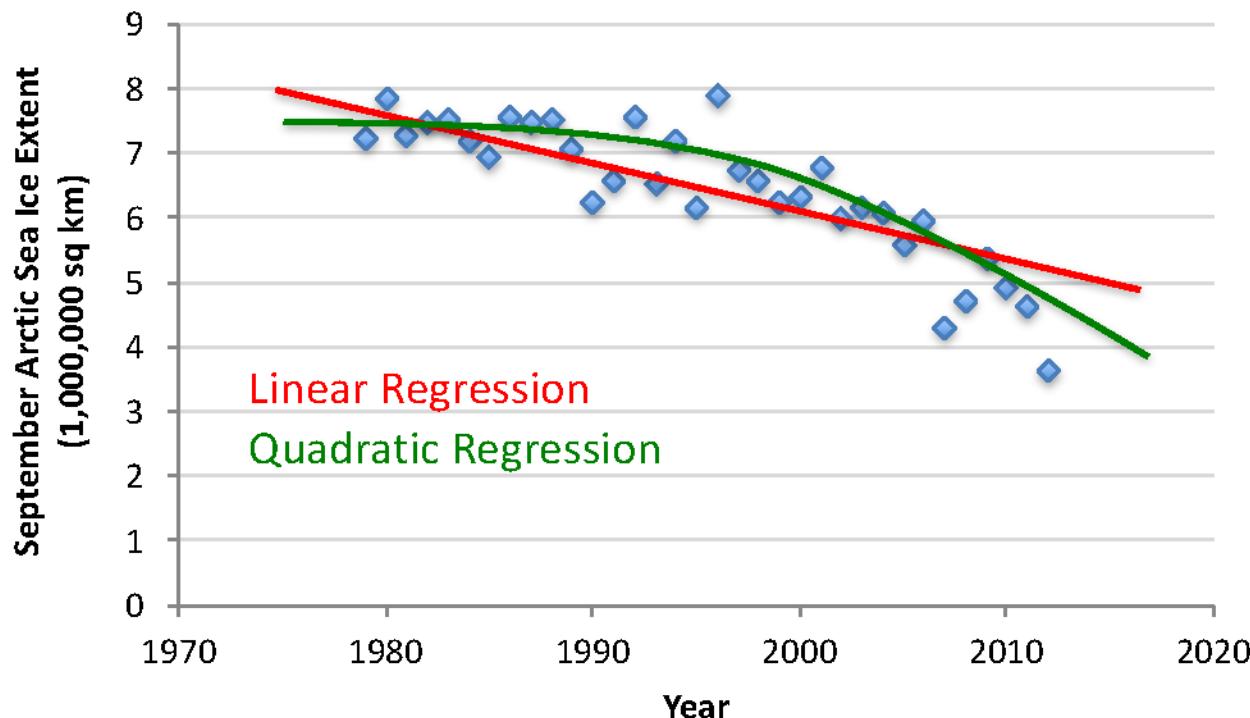
Wish to learn  $f: X \rightarrow Y$ , where  $Y$  is real, given  $\{<x^1, y^1> \dots <x^n, y^n>\}$

- Geometric Approach
  - Least squares function fitting given  $\{<x^1, y^1> \dots <x^n, y^n>\}$
- Bayesian Approach
  - Choose some parameterized form for  $P(Y|X; \theta)$ 
    - ( $\theta$  is the vector of parameters)
  - Derive learning algorithm as MCLE estimate for  $\theta$

# Geometric Approach

Given:

- Data  $\mathbf{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$  where  $\mathbf{x}^{(i)} \in \mathbb{R}^d$
- Corresponding labels  $\mathbf{y} = \{y^{(1)}, \dots, y^{(n)}\}$  where  $y^{(i)} \in \mathbb{R}$



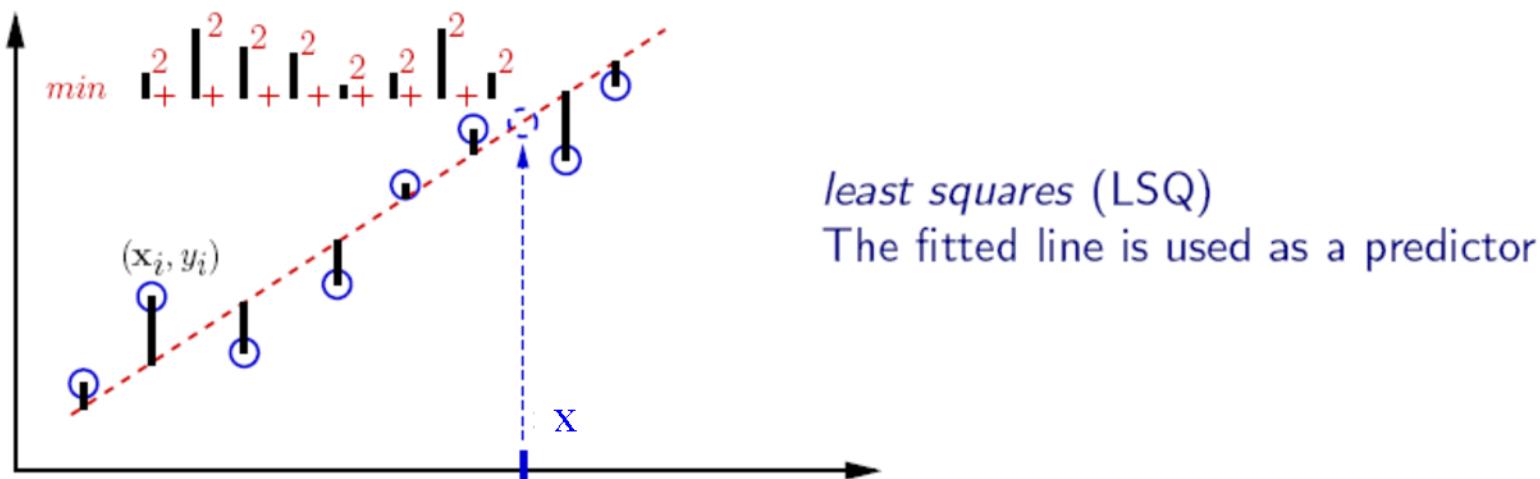
# Linear Regression

- Hypothesis:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_d x_d = \sum_{j=0}^d \theta_j x_j$$

Assume  $x_0 = 1$

- Fit model by minimizing sum of squared errors

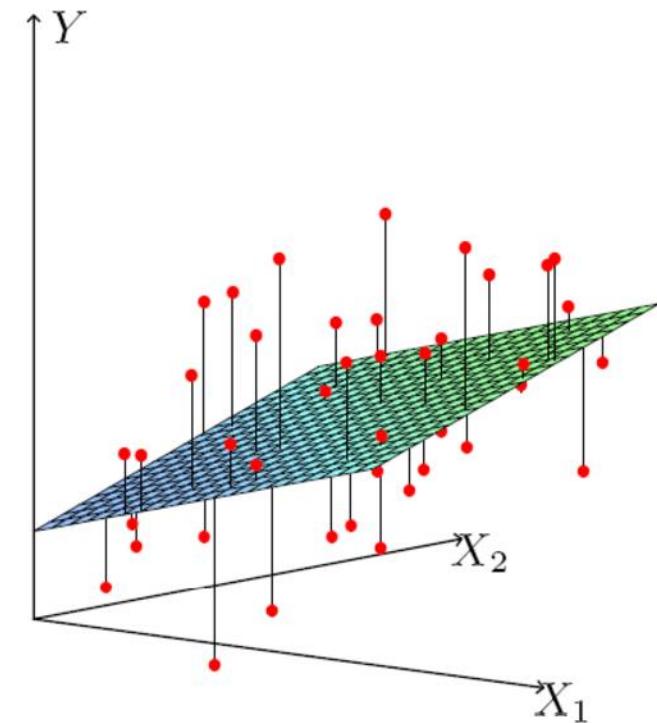
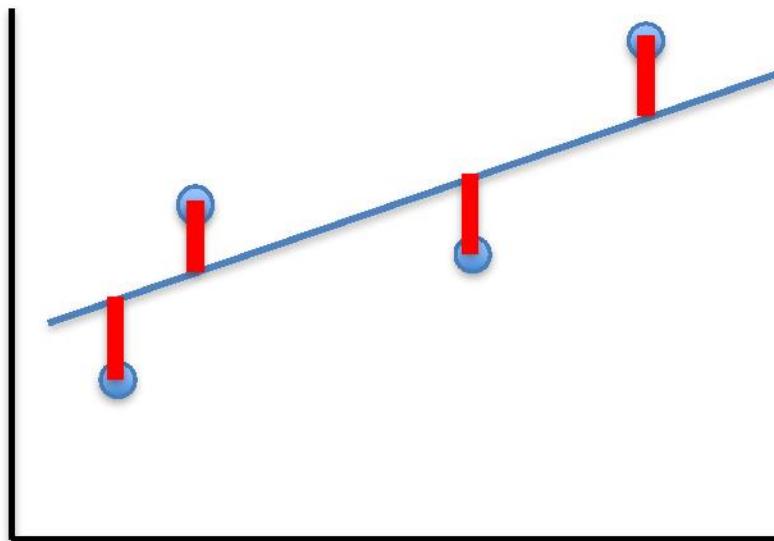


# Least Squares Linear Regression

- Cost Function

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

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



# Intuition Behind Cost Function

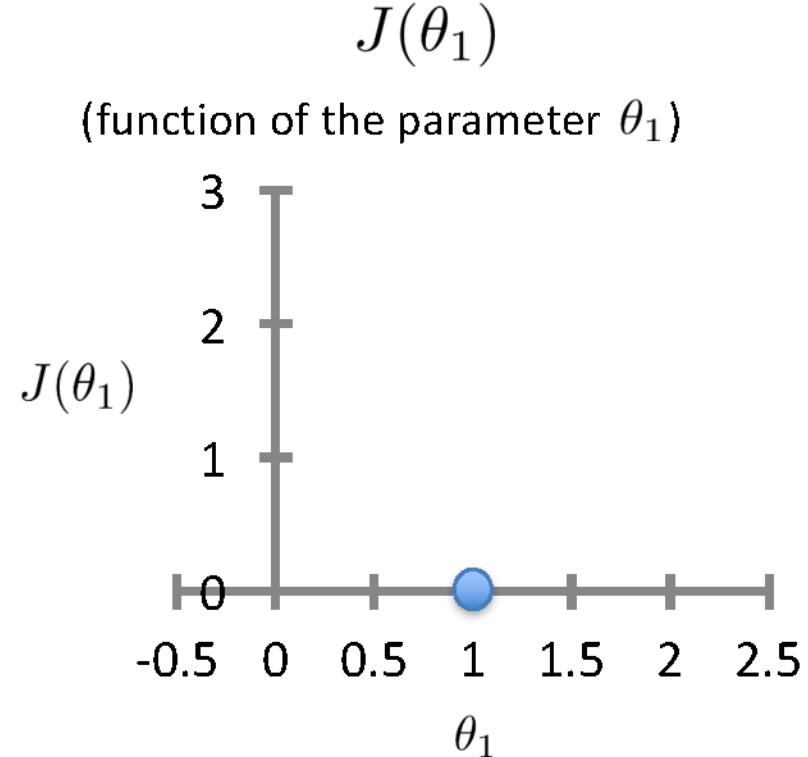
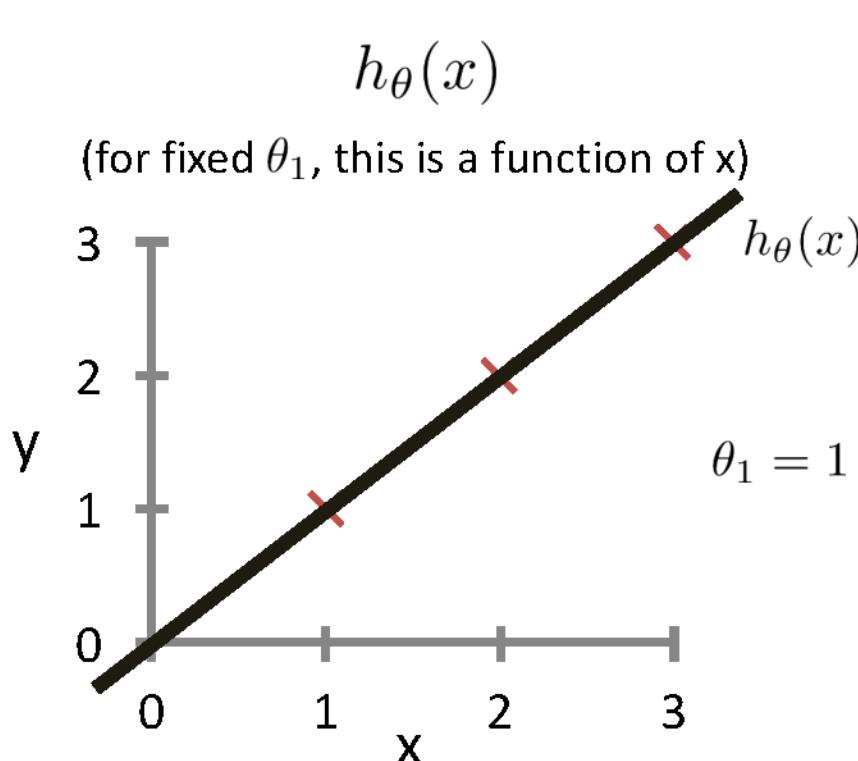
$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^n \left( h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$$

For insight on  $J()$ , let's assume  $x \in \mathbb{R}$  so  $\boldsymbol{\theta} = [\theta_0, \theta_1]$

# Intuition Behind Cost Function

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

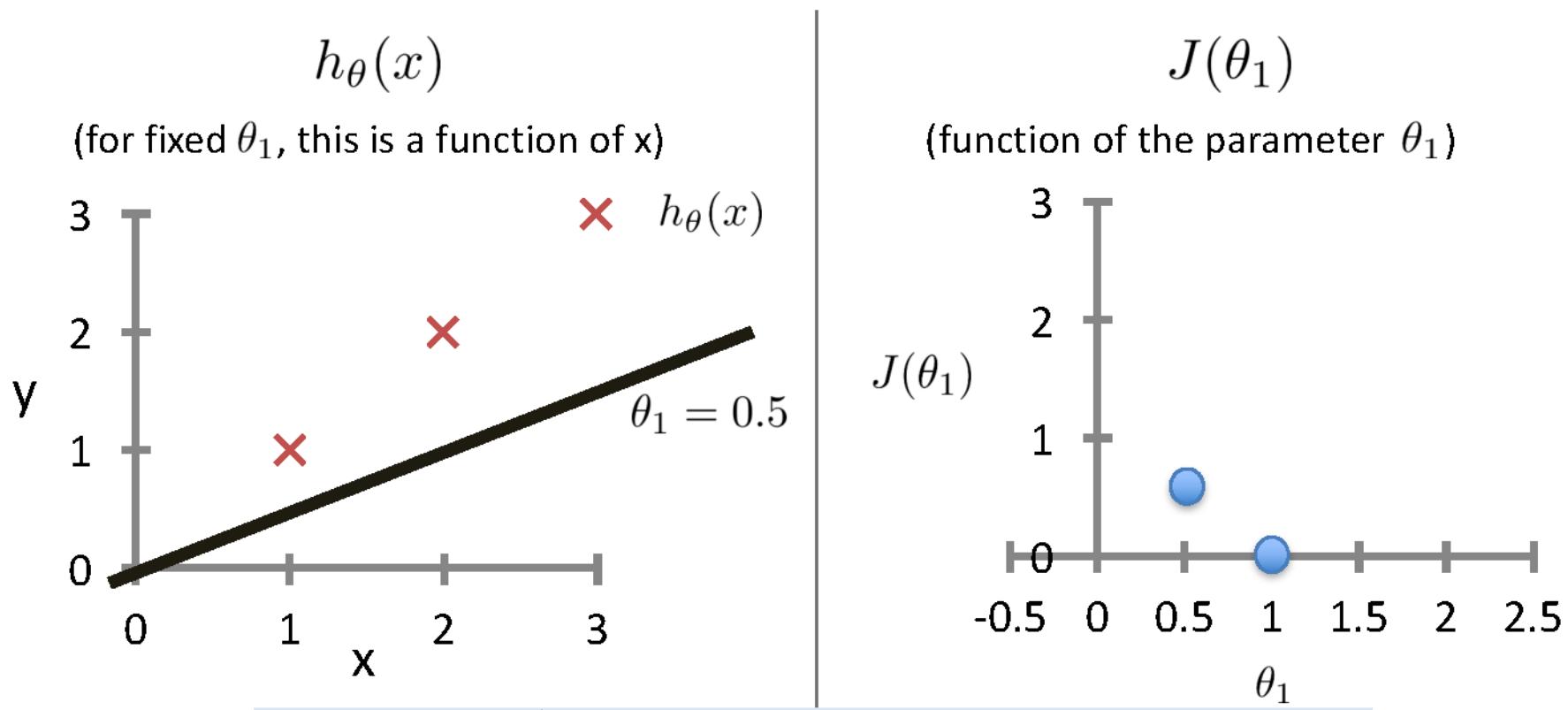
For insight on  $J()$ , let's assume  $x \in \mathbb{R}$  so  $\theta = [\theta_0, \theta_1]$



# Intuition Behind Cost Function

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

For insight on  $J()$ , let's assume  $x \in \mathbb{R}$  so  $\theta = [\theta_0, \theta_1]$



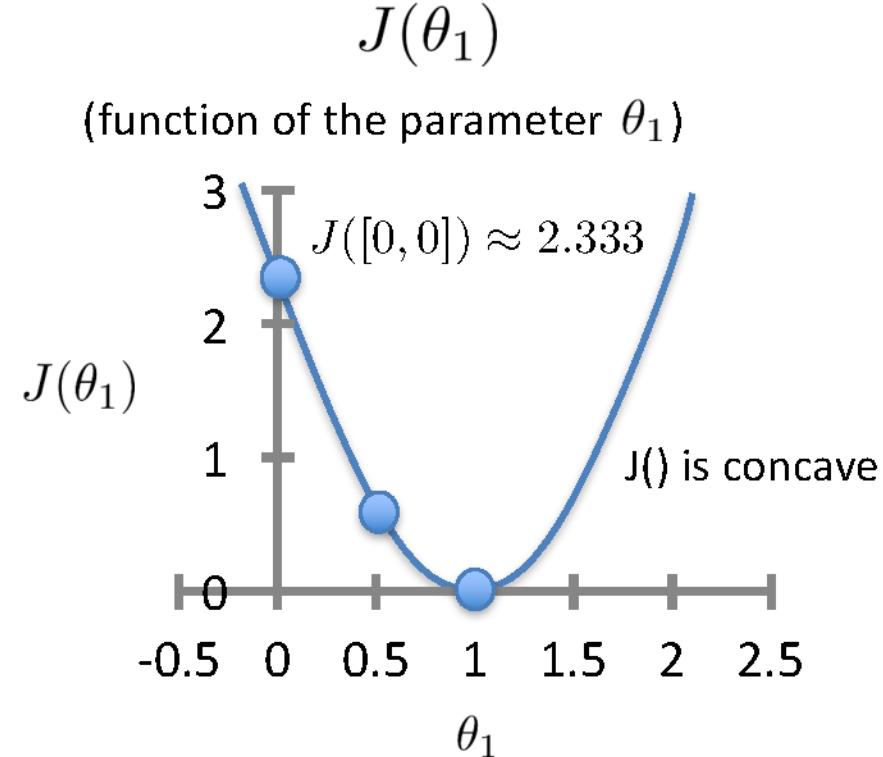
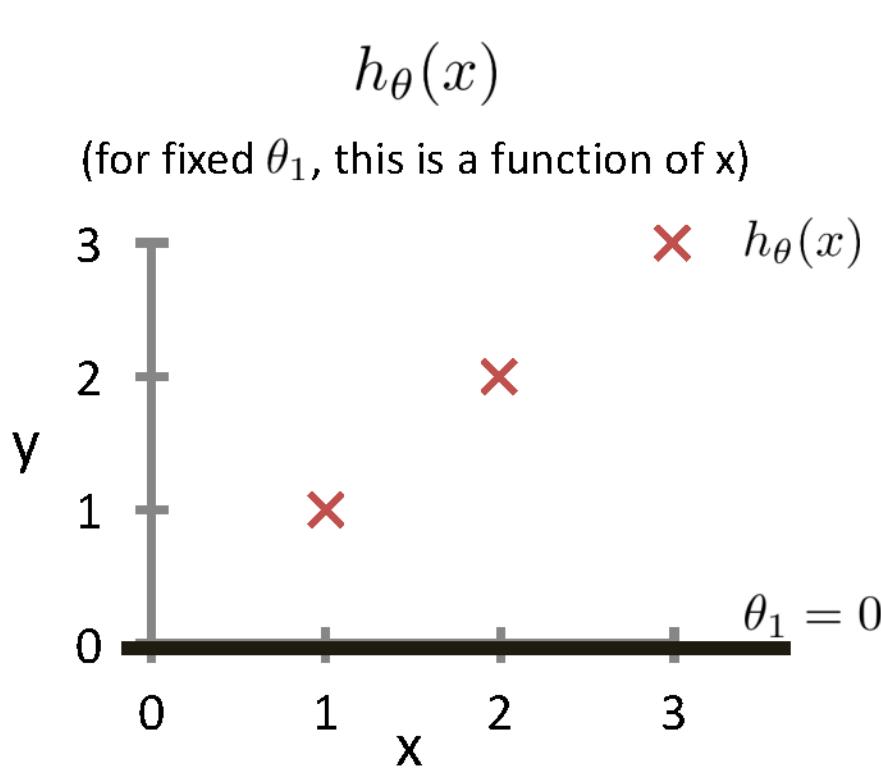
Based on example  
by Andrew Ng

$$J([0, 0.5]) = \frac{1}{2 \times 3} [(0.5 - 1)^2 + (1 - 2)^2 + (1.5 - 3)^2] \approx 0.58$$

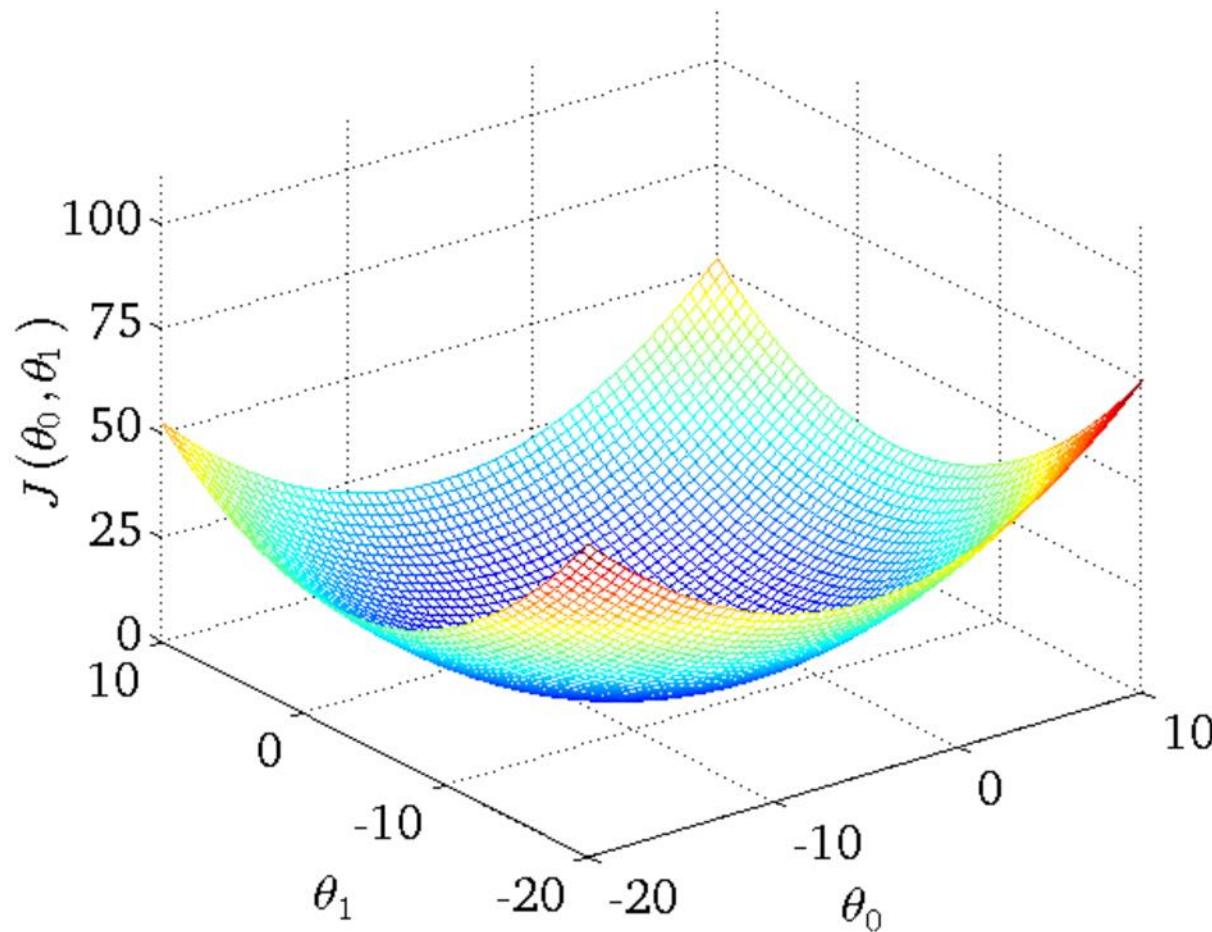
# Intuition Behind Cost Function

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

For insight on  $J()$ , let's assume  $x \in \mathbb{R}$  so  $\theta = [\theta_0, \theta_1]$



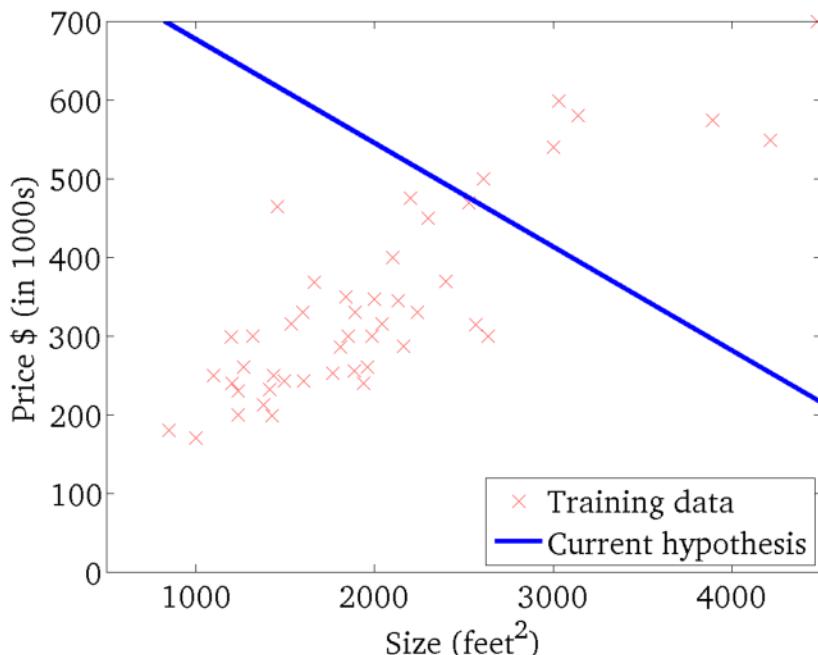
# Intuition Behind Cost Function



# Intuition Behind Cost Function

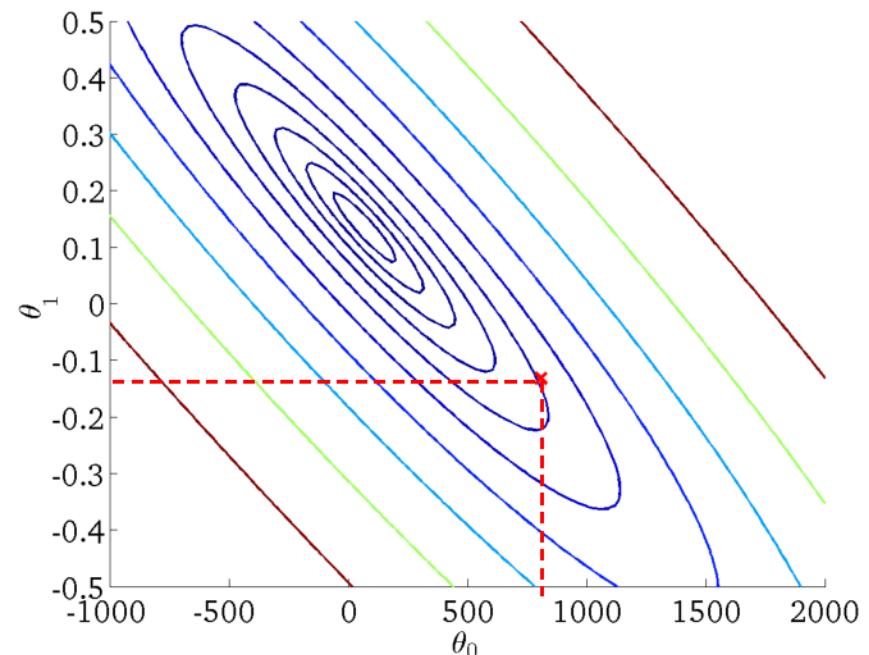
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

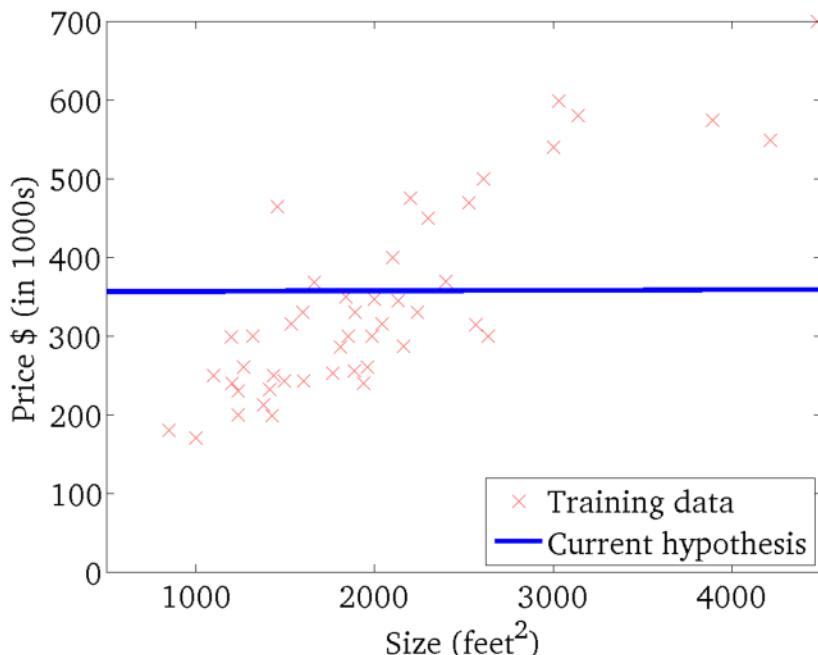
(function of the parameters  $\theta_0, \theta_1$ )



# Intuition Behind Cost Function

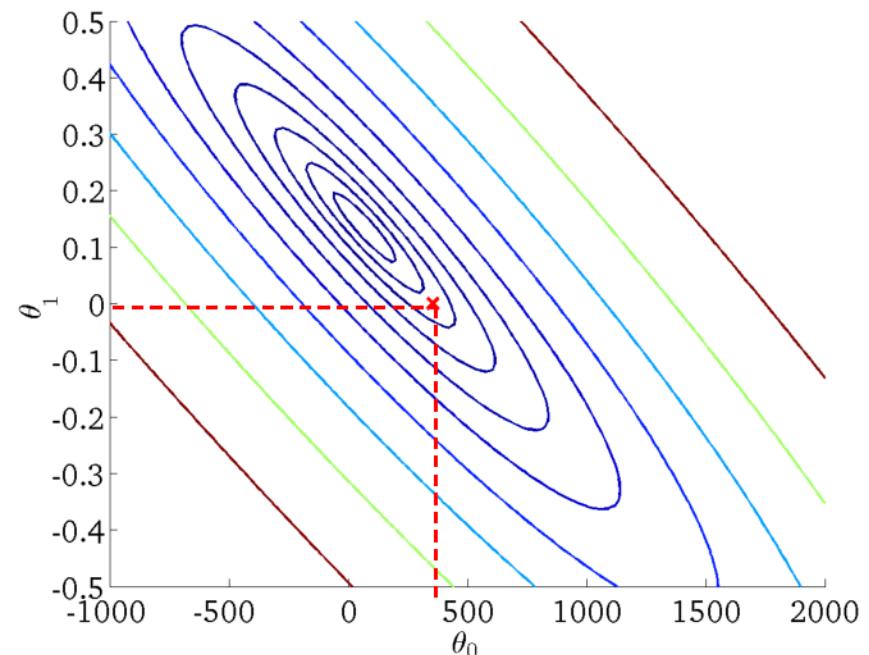
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

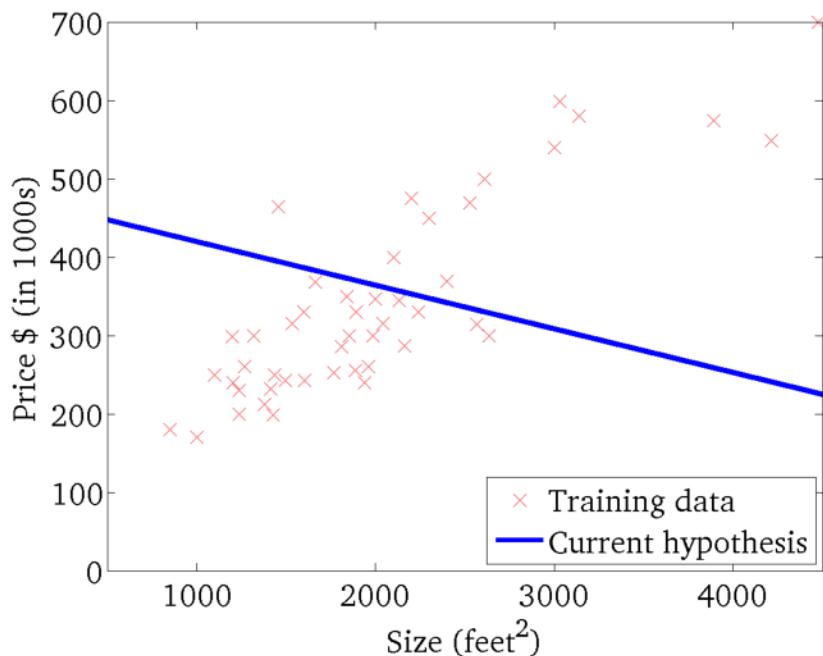
(function of the parameters  $\theta_0, \theta_1$ )



# Intuition Behind Cost Function

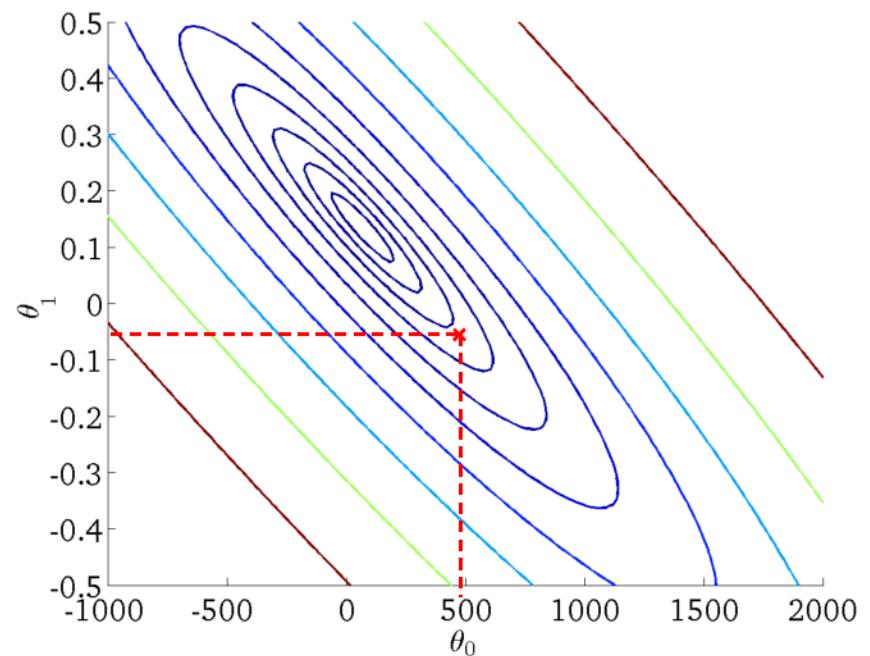
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

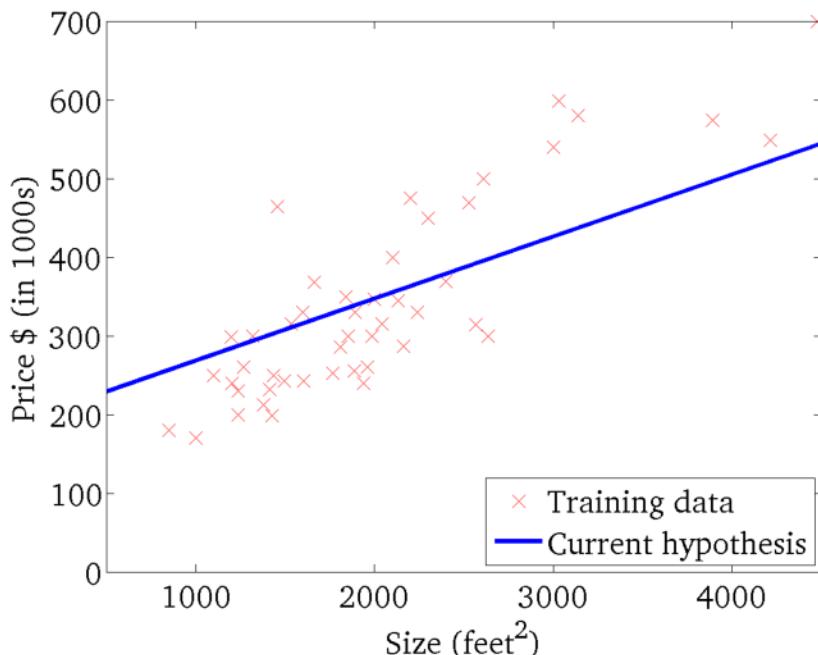
(function of the parameters  $\theta_0, \theta_1$ )



# Intuition Behind Cost Function

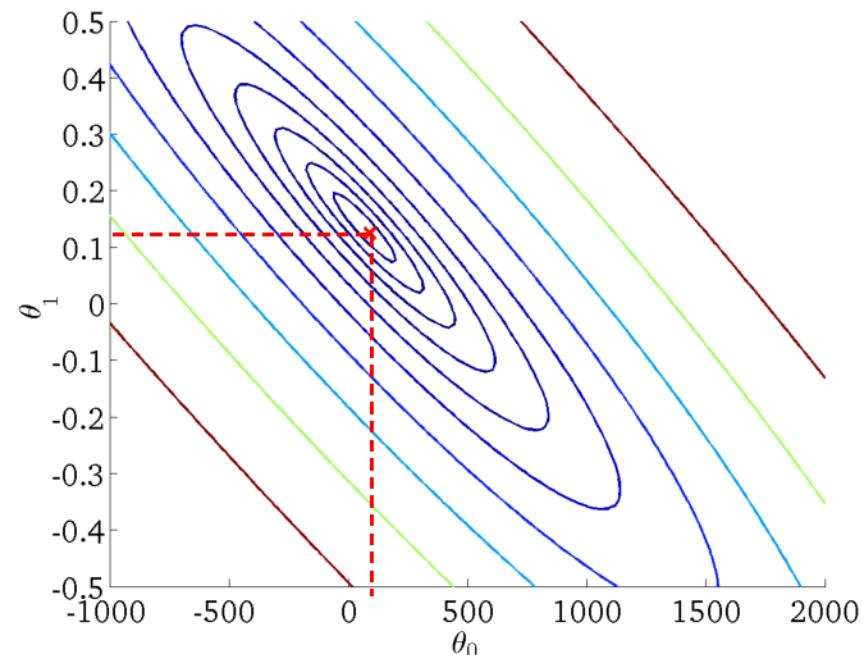
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



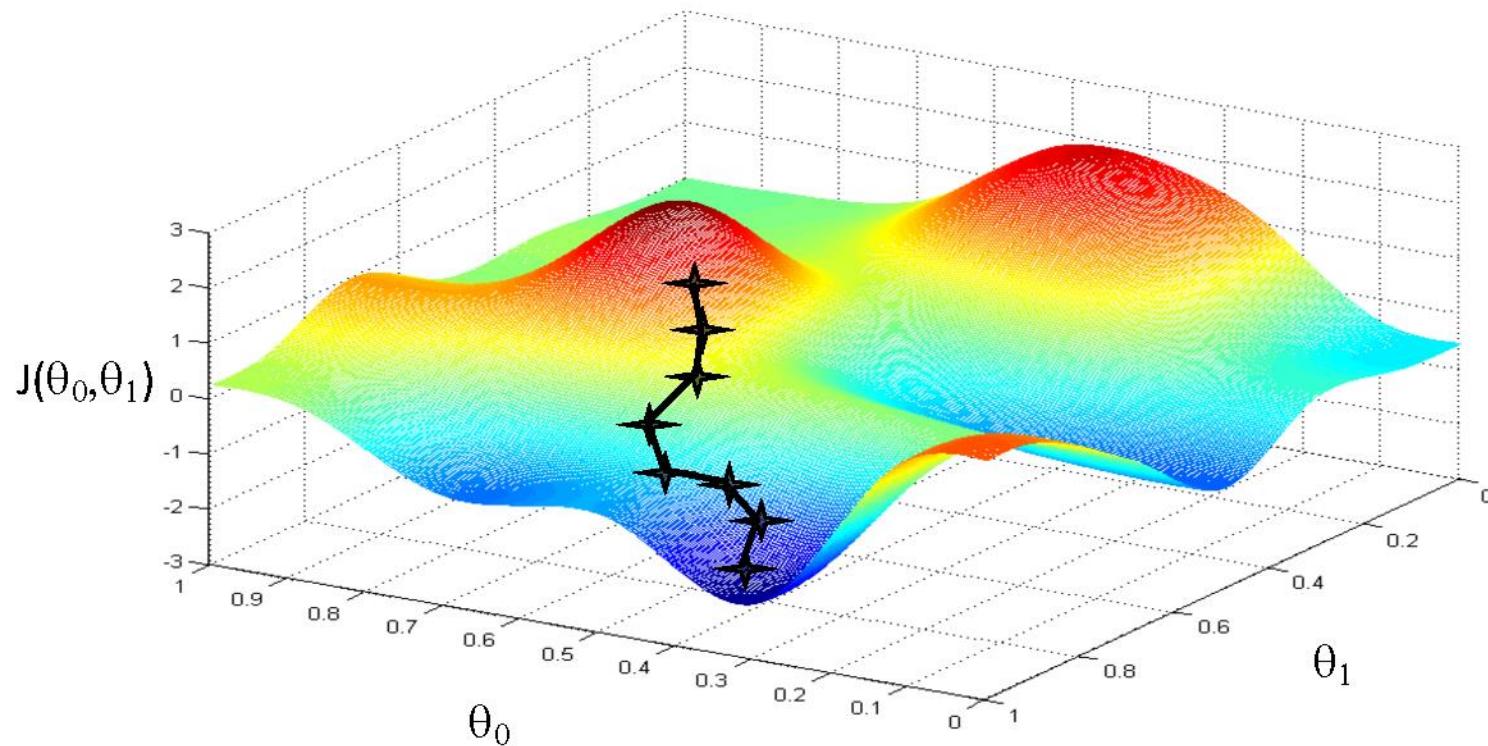
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



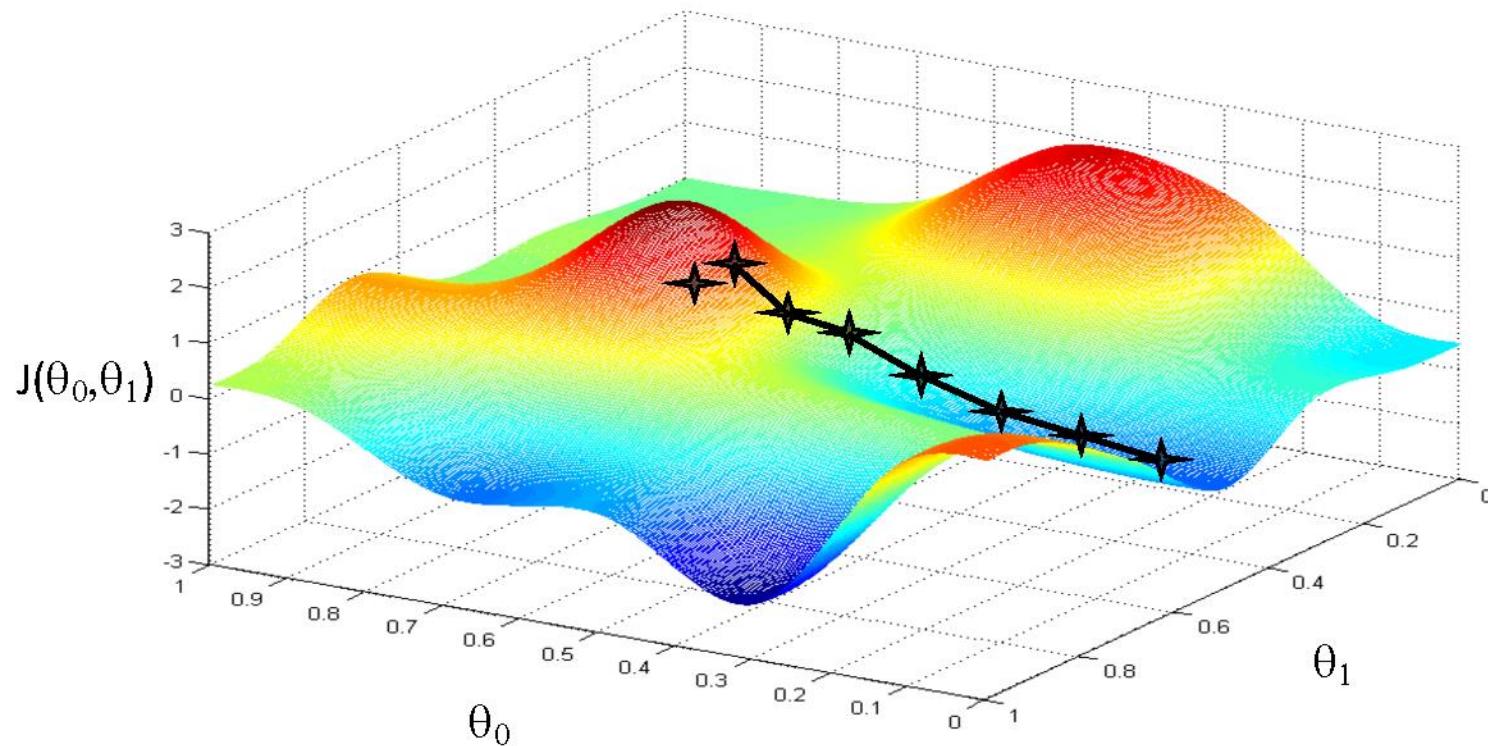
# Basic Search Procedure

- Choose initial value for  $\theta$
- Until we reach a minimum:
  - Choose a new value for  $\theta$  to reduce  $J(\theta)$



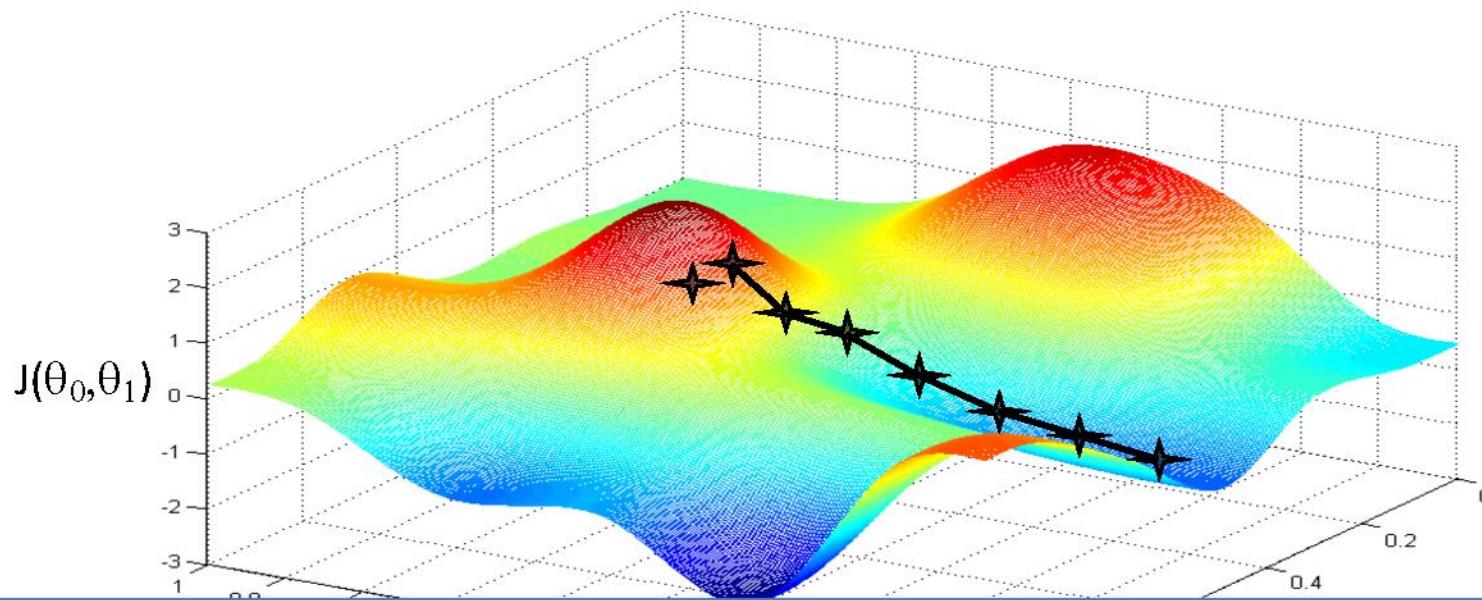
# Basic Search Procedure

- Choose initial value for  $\theta$
- Until we reach a minimum:
  - Choose a new value for  $\theta$  to reduce  $J(\theta)$



# Basic Search Procedure

- Choose initial value for  $\theta$
- Until we reach a minimum:
  - Choose a new value for  $\theta$  to reduce  $J(\theta)$



Since the least squares objective function is convex (concave),  
we don't need to worry about local minima

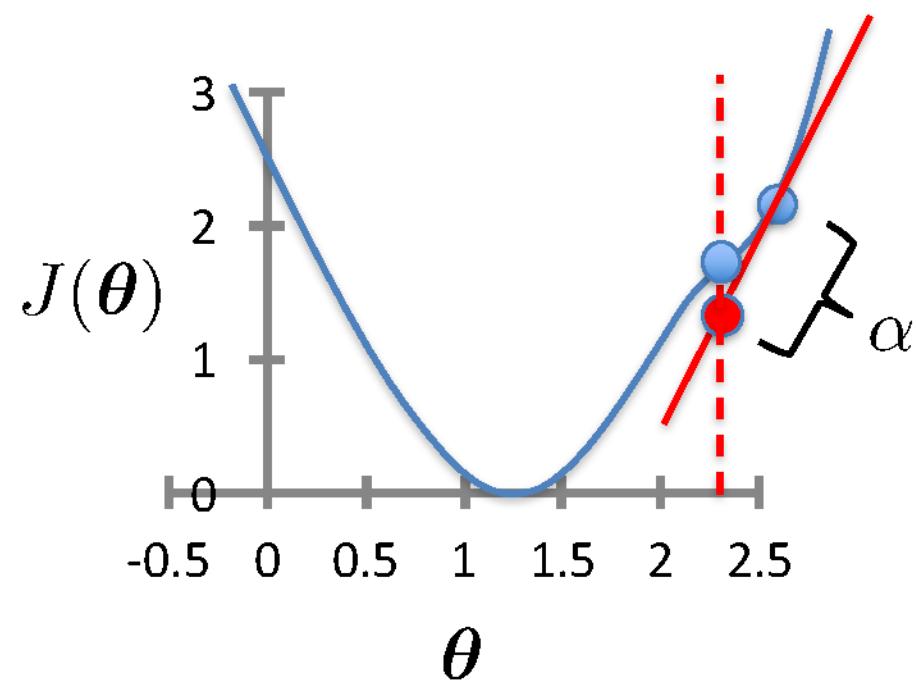
# Gradient Descent

- Initialize  $\theta$
- Repeat until convergence

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

simultaneous update  
for  $j = 0 \dots d$

learning rate (small)  
e.g.,  $\alpha = 0.05$



# Gradient Descent

- Initialize  $\theta$
- Repeat until convergence

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

simultaneous update  
for  $j = 0 \dots d$

For Linear Regression:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^n \left( h_{\theta} (\mathbf{x}^{(i)}) - y^{(i)} \right)^2$$

# Gradient Descent

- Initialize  $\theta$
- Repeat until convergence

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

simultaneous update  
for  $j = 0 \dots d$

For Linear Regression:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^n \left( h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 \\ &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^n \left( \sum_{k=0}^d \theta_k x_k^{(i)} - y^{(i)} \right)^2\end{aligned}$$

# Gradient Descent

- Initialize  $\theta$
- Repeat until convergence

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

simultaneous update  
for  $j = 0 \dots d$

For Linear Regression:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^n \left( h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 \\ &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^n \left( \sum_{k=0}^d \theta_k x_k^{(i)} - y^{(i)} \right)^2 \\ &= \frac{1}{n} \sum_{i=1}^n \left( \sum_{k=0}^d \theta_k x_k^{(i)} - y^{(i)} \right) \times \frac{\partial}{\partial \theta_j} \left( \sum_{k=0}^d \theta_k x_k^{(i)} - y^{(i)} \right)\end{aligned}$$

# Gradient Descent

- Initialize  $\theta$
- Repeat until convergence

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

simultaneous update  
for  $j = 0 \dots d$

For Linear Regression:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^n \left( h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 \\ &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^n \left( \sum_{k=0}^d \theta_k x_k^{(i)} - y^{(i)} \right)^2 \\ &= \frac{1}{n} \sum_{i=1}^n \left( \sum_{k=0}^d \theta_k x_k^{(i)} - y^{(i)} \right) \times \frac{\partial}{\partial \theta_j} \left( \sum_{k=0}^d \theta_k x_k^{(i)} - y^{(i)} \right) \\ &= \frac{1}{n} \sum_{i=1}^n \left( \sum_{k=0}^d \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)}\end{aligned}$$

# Gradient Descent for Linear Regression

- Initialize  $\theta$
- Repeat until convergence

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

simultaneous  
update  
for  $j = 0 \dots d$

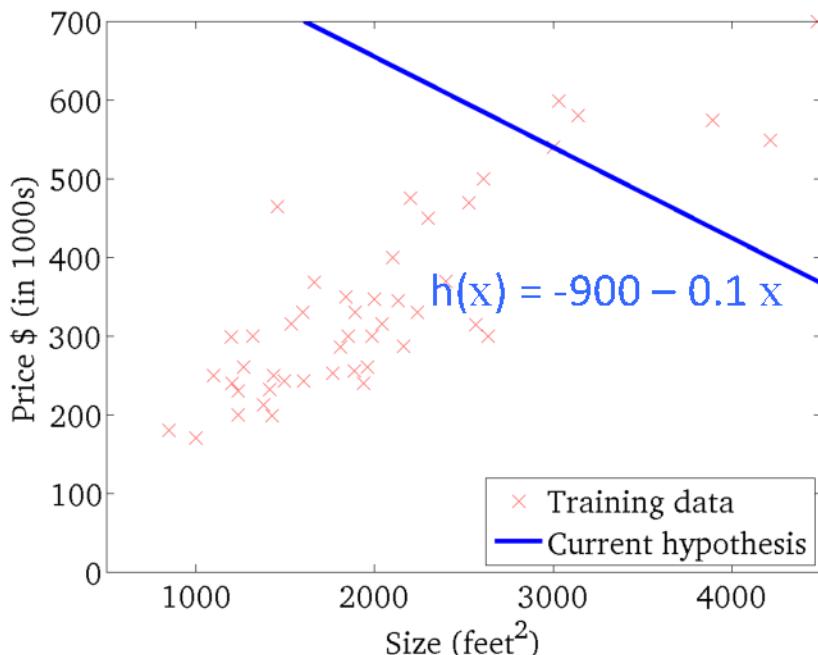
- To achieve simultaneous update
  - At the start of each GD iteration, compute  $h_{\theta}(\mathbf{x}^{(i)})$
  - Use this stored value in the update step loop
- Assume convergence when  $\|\theta_{new} - \theta_{old}\|_2 < \epsilon$

L<sub>2</sub> norm:  $\|\mathbf{v}\|_2 = \sqrt{\sum_i v_i^2} = \sqrt{v_1^2 + v_2^2 + \dots + v_{|v|}^2}$

# Gradient Descent

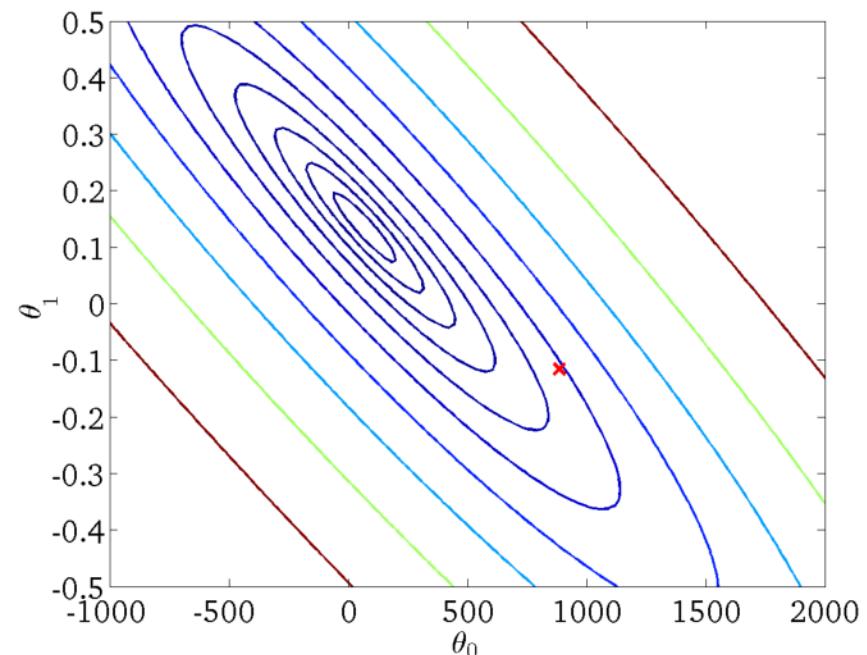
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

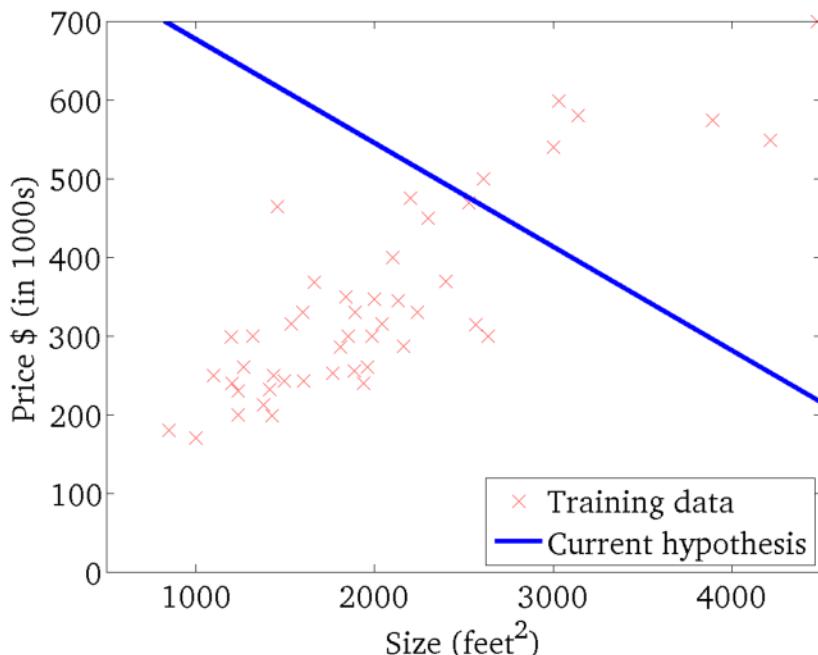
(function of the parameters  $\theta_0, \theta_1$ )



# Gradient Descent

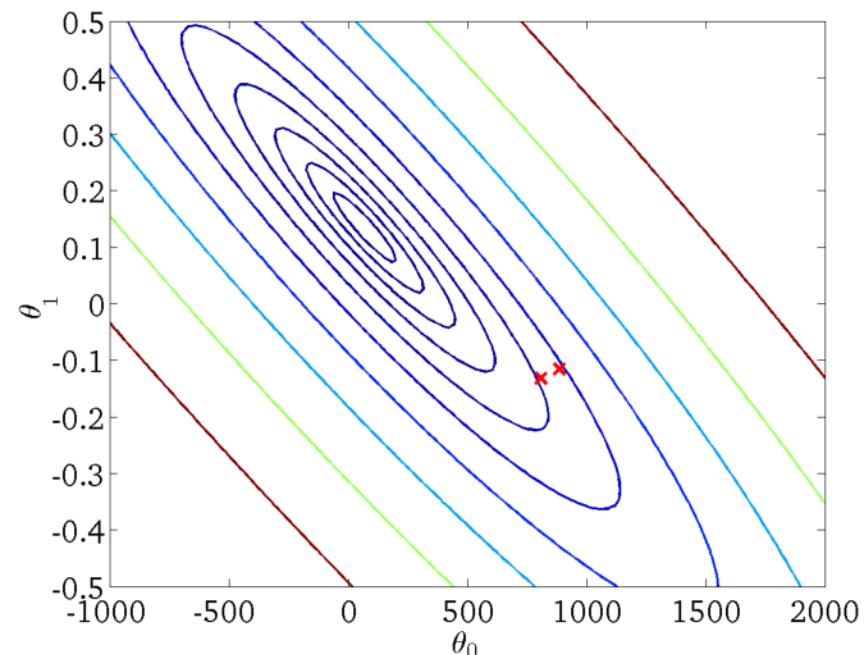
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

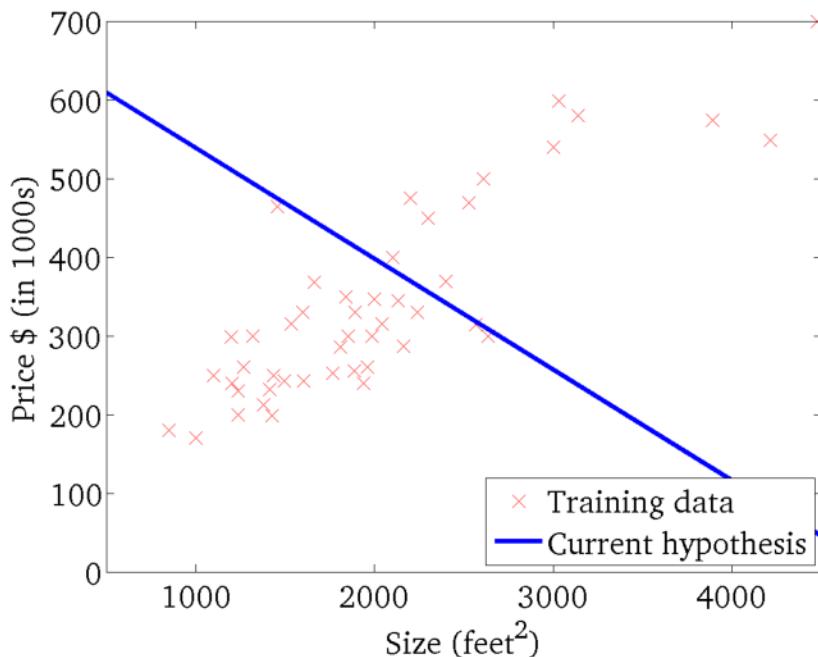
(function of the parameters  $\theta_0, \theta_1$ )



# Gradient Descent

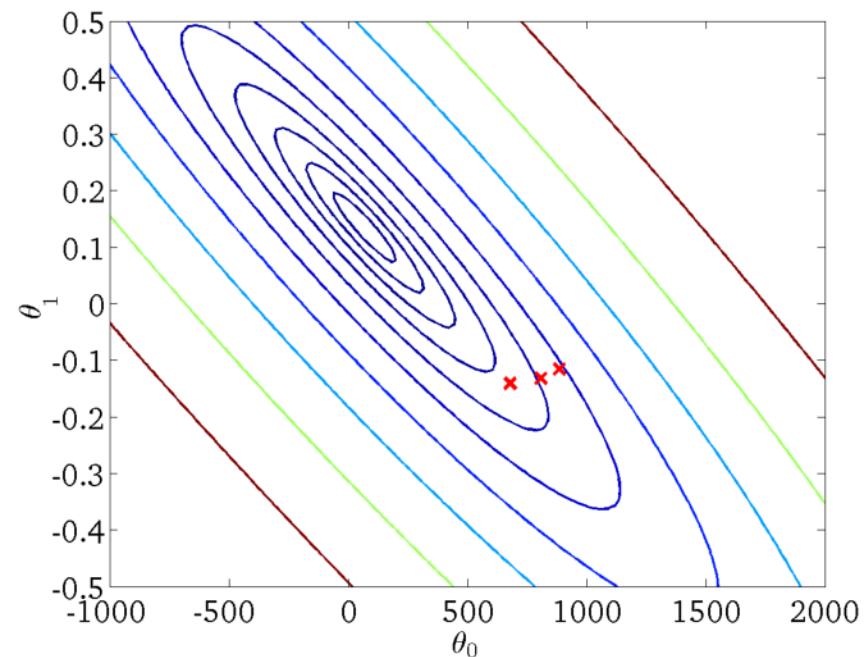
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

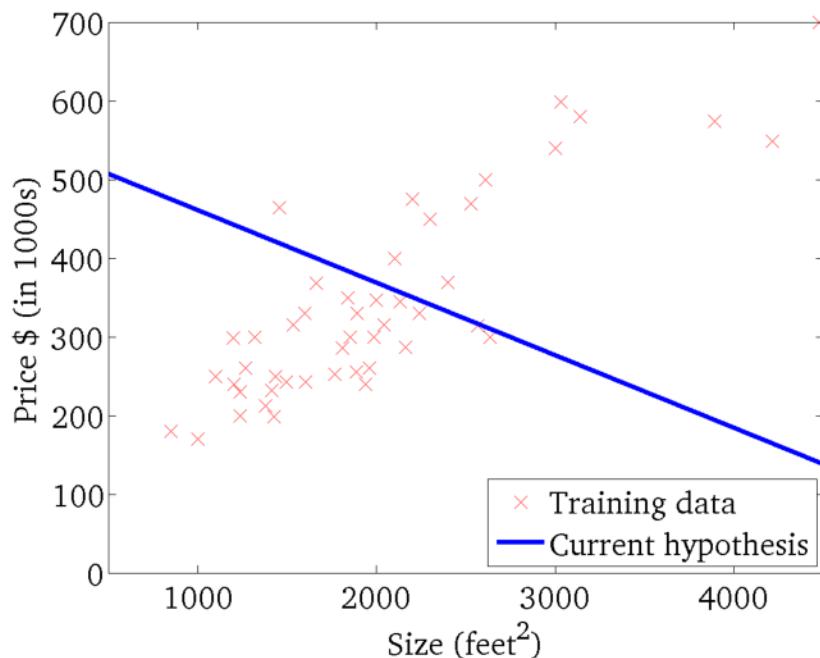
(function of the parameters  $\theta_0, \theta_1$ )



# Gradient Descent

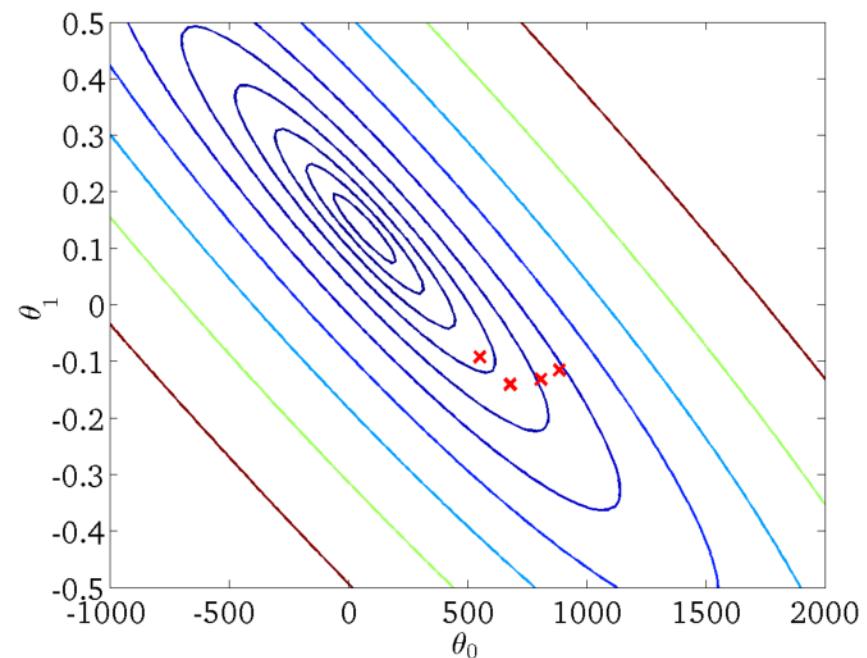
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

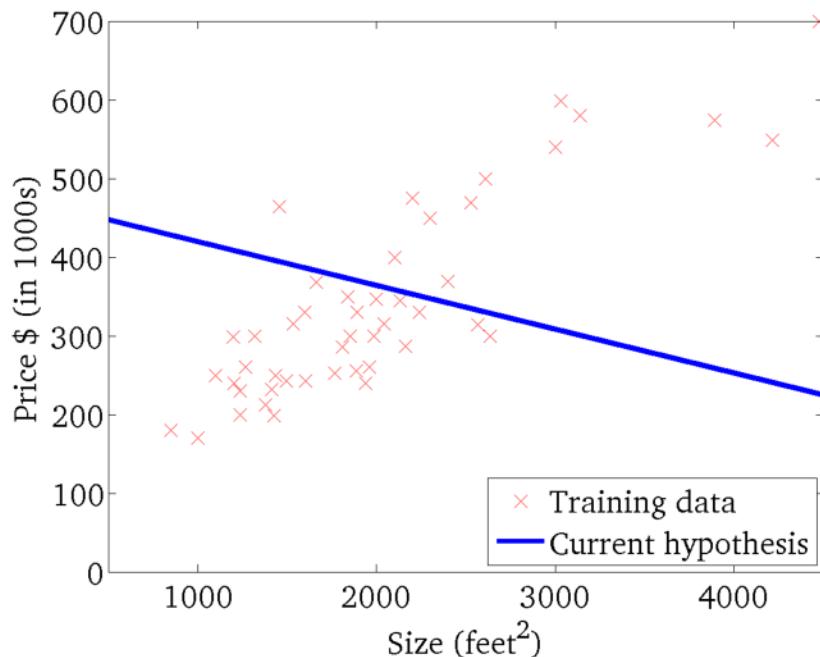
(function of the parameters  $\theta_0, \theta_1$ )



# Gradient Descent

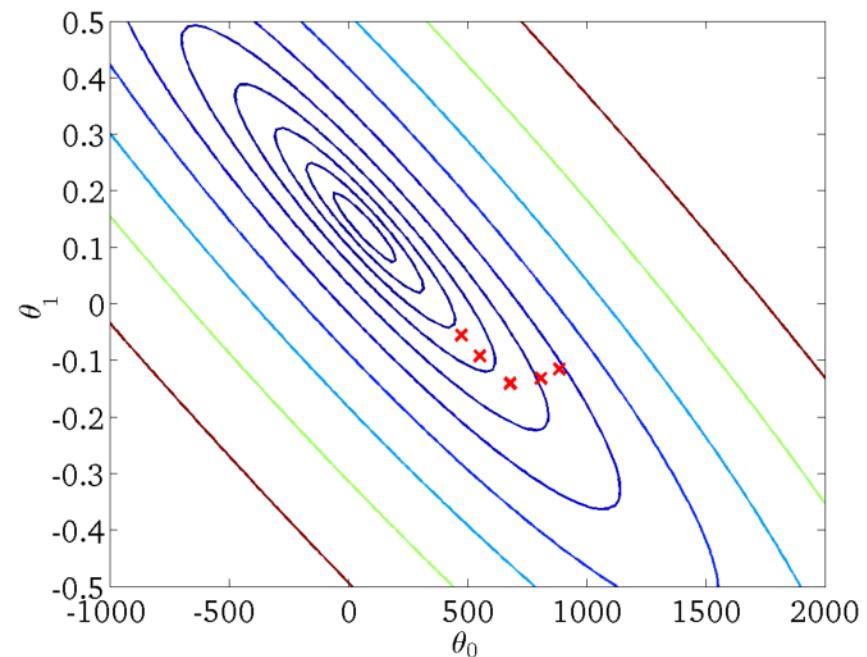
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

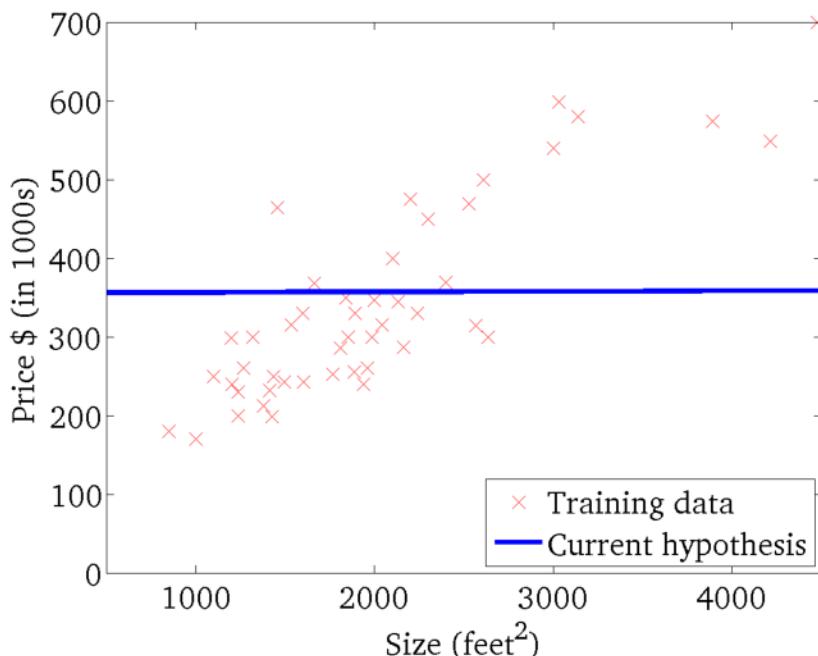
(function of the parameters  $\theta_0, \theta_1$ )



# Gradient Descent

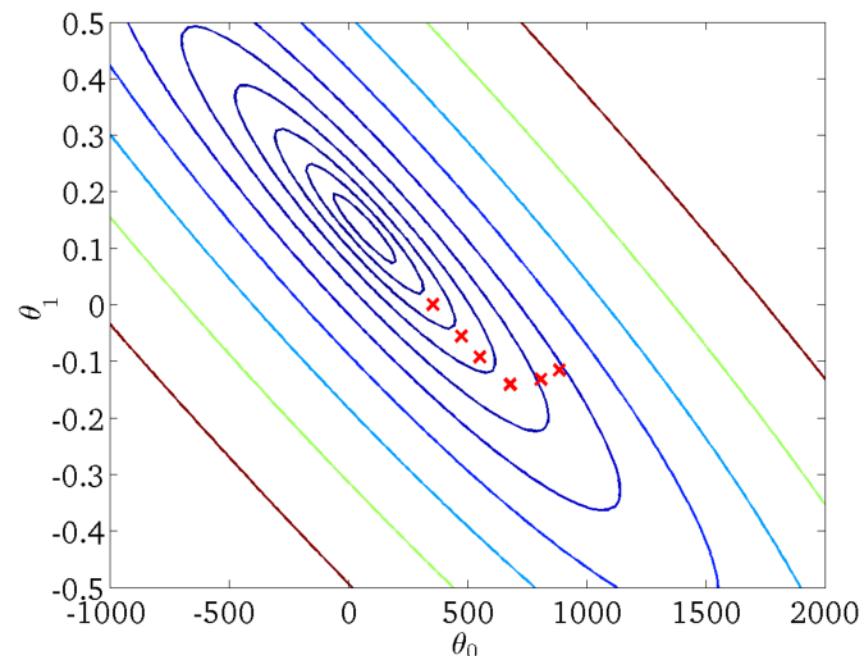
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

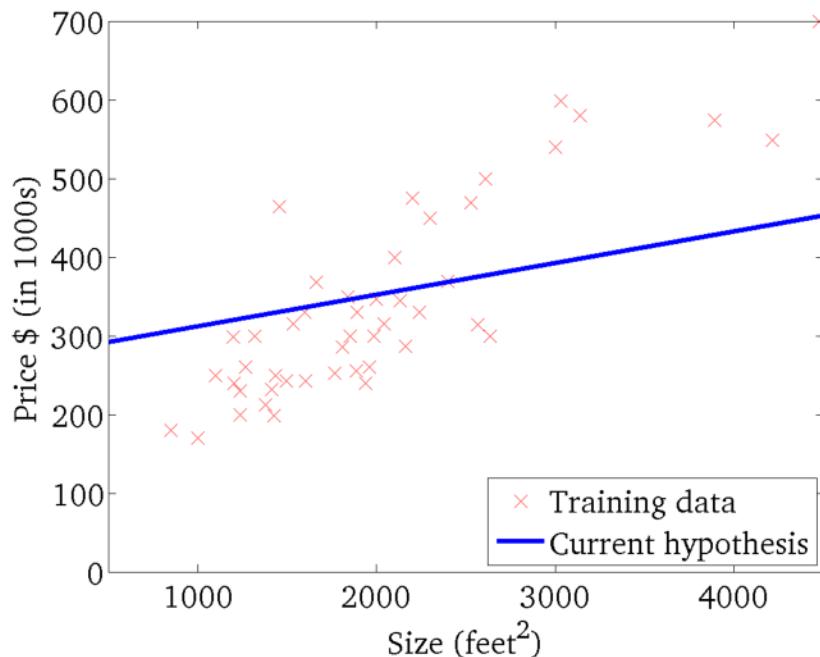
(function of the parameters  $\theta_0, \theta_1$ )



# Gradient Descent

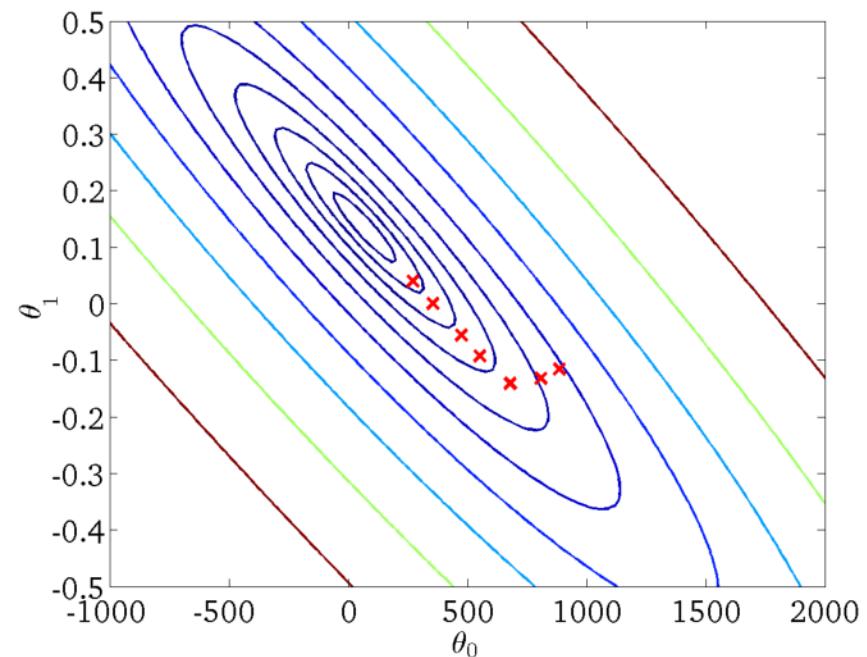
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

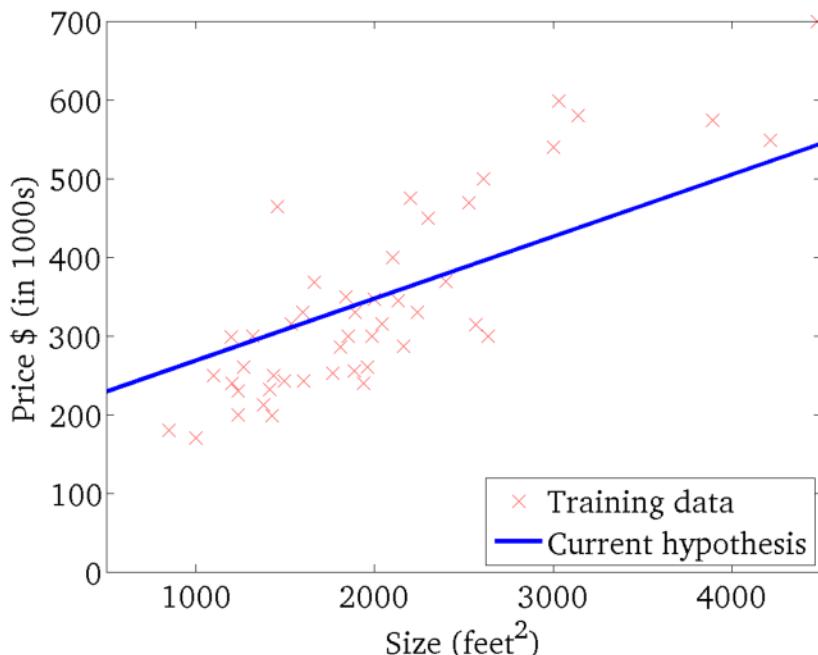
(function of the parameters  $\theta_0, \theta_1$ )



# Gradient Descent

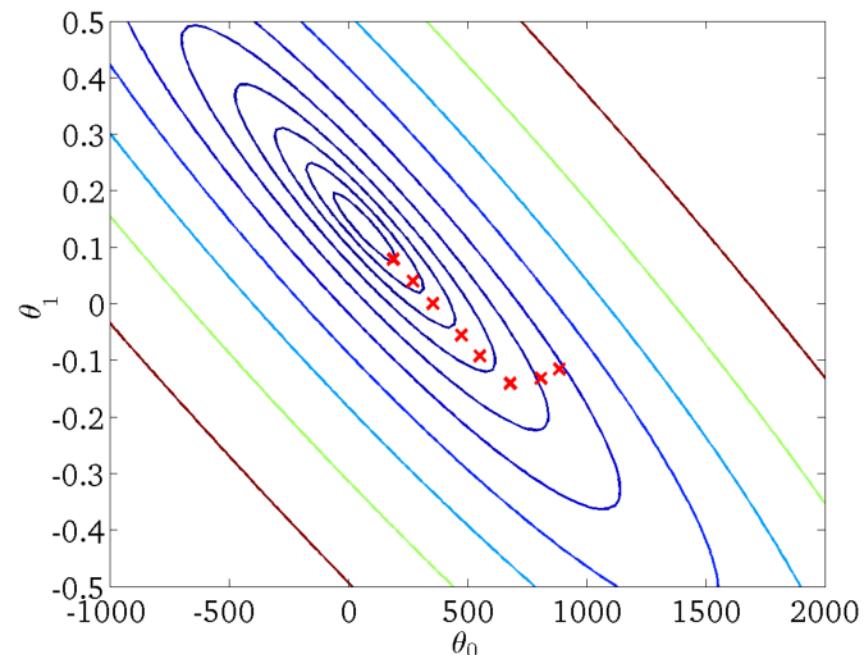
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

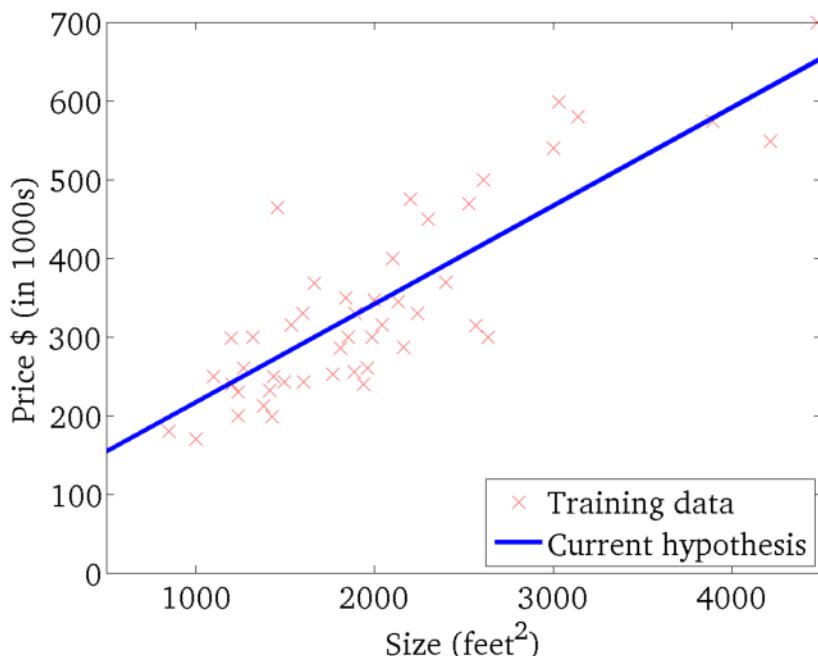
(function of the parameters  $\theta_0, \theta_1$ )



# Gradient Descent

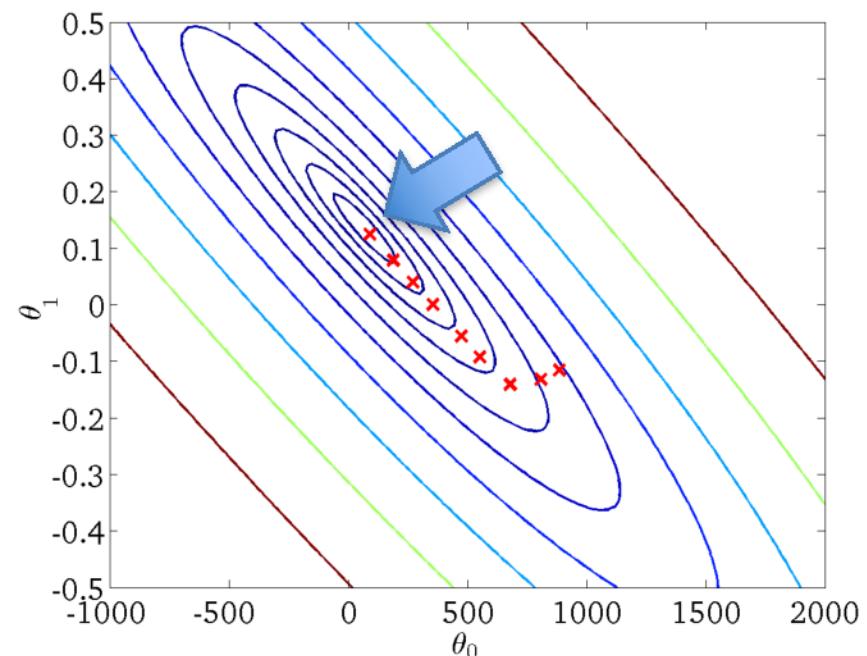
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



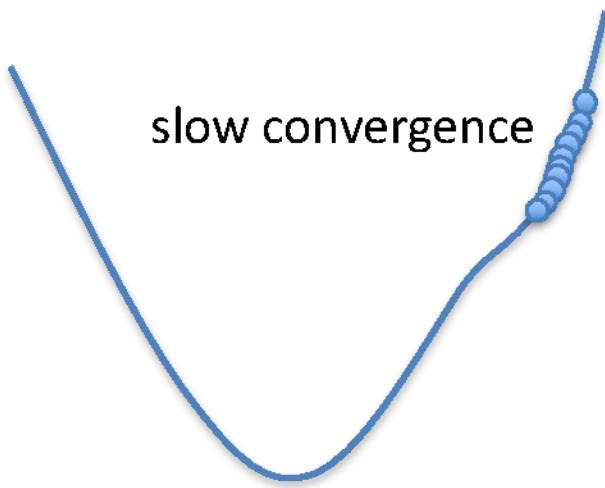
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )

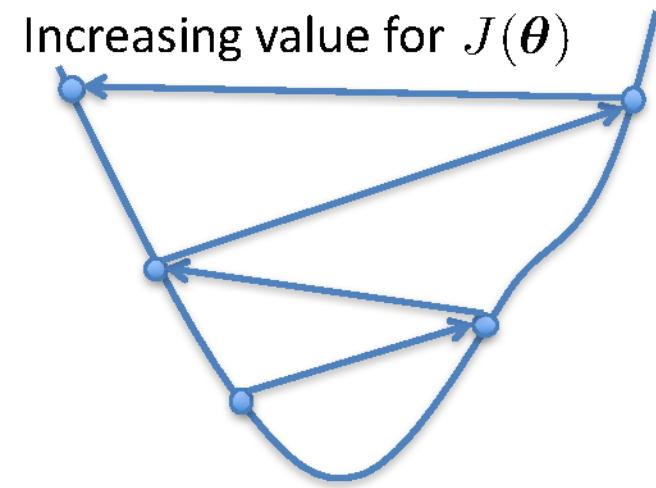


# Choosing $\alpha$

$\alpha$  too small



$\alpha$  too large



- May overshoot the minimum
- May fail to converge
- May even diverge

To see if gradient descent is working, print out  $J(\theta)$  each iteration

- The value should decrease at each iteration
- If it doesn't, adjust  $\alpha$

---

# Learning Model Parameters – Closed Form Solution (using vectorization)

# Vectorization

- Benefits of vectorization
  - More compact equations
  - Faster code (using optimized matrix libraries)

- Consider our model:

$$h(\mathbf{x}) = \sum_{j=0}^d \theta_j x_j$$

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \mathbf{x}^\top = [ 1 \quad x_1 \quad \dots \quad x_d ]$$

- Can write the model in vectorized form as  $h(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x}$

# Vectorization

- Consider our model for n instances:

$$h(\mathbf{x}^{(i)}) = \sum_{j=0}^d \theta_j x_j^{(i)}$$

- Let

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix}$$
$$\mathbb{R}^{(d+1) \times 1} \qquad \qquad \qquad \mathbb{R}^{n \times (d+1)}$$

- Can write the model in vectorized form as  $h_{\theta}(\mathbf{x}) = \mathbf{X}\theta$

# Vectorization

- For the linear regression cost function:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{2n} \sum_{i=1}^n \left( h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 \\ &= \frac{1}{2n} \sum_{i=1}^n \left( \boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)} \right)^2 \\ &= \frac{1}{2n} \underbrace{(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^\top}_{\mathbb{R}^{1 \times n}} \underbrace{(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})}_{\mathbb{R}^{n \times 1}} \end{aligned}$$

$\mathbb{R}^{n \times (d+1)}$   
 $\mathbb{R}^{(d+1) \times 1}$

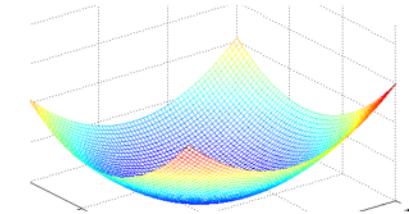
Let:

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

# Closed Form Solution

- Instead of using GD, solve for optimal  $\theta$  analytically
  - Notice that the solution is when  $\frac{\partial}{\partial \theta} J(\theta) = 0$
- Derivation:

$$\begin{aligned}\mathcal{J}(\theta) &= \frac{1}{2n} (\mathbf{X}\theta - \mathbf{y})^\top (\mathbf{X}\theta - \mathbf{y}) \\ &\propto \theta^\top \mathbf{X}^\top \mathbf{X} \theta - \boxed{\mathbf{y}^\top \mathbf{X} \theta} - \boxed{\theta^\top \mathbf{X}^\top \mathbf{y}} + \mathbf{y}^\top \mathbf{y} \\ &\propto \theta^\top \mathbf{X}^\top \mathbf{X} \theta - 2\theta^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y}\end{aligned}$$



Take derivative and set equal to 0, then solve for  $\theta$ :

$$\frac{\partial}{\partial \theta} (\theta^\top \mathbf{X}^\top \mathbf{X} \theta - 2\theta^\top \mathbf{X}^\top \mathbf{y} + \cancel{\mathbf{y}^\top \mathbf{y}}) = 0$$

$$(\mathbf{X}^\top \mathbf{X})\theta - \mathbf{X}^\top \mathbf{y} = 0$$

$$(\mathbf{X}^\top \mathbf{X})\theta = \mathbf{X}^\top \mathbf{y}$$

Closed Form Solution:

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

# Closed Form Solution

- Can obtain  $\theta$  by simply plugging  $X$  and  $y$  into

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

$$X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

- If  $X^T X$  is not invertible (i.e., singular), may need to:
  - Use pseudo-inverse instead of the inverse
    - In python, `numpy.linalg.pinv(a)`
  - Remove redundant (not linearly independent) features
  - Remove extra features to ensure that  $d \leq n$

# Gradient Descent vs Closed Form

## Gradient Descent

- Requires multiple iterations
- Need to choose  $\alpha$
- Works well when  $n$  is large
- Can support incremental learning

## Closed Form Solution

- Non-iterative
- No need for  $\alpha$
- Slow if  $n$  is large
  - Computing  $(X^T X)^{-1}$  is roughly  $O(n^3)$

# Bayesian linear regression

# Bayesian analysis

---

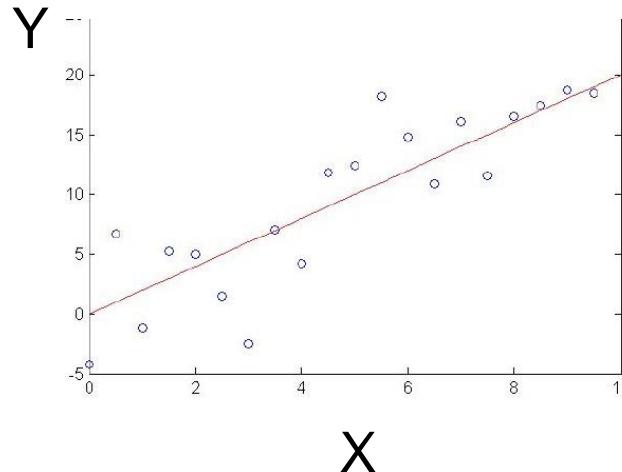
- Bayesian analysis will show that
  - under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood hypothesis

# Maximum likelihood and least-squared error hypotheses

---

- A set of  $m$  training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution.
- Each training example is a pair of the form  $(x_i, d_i)$  where  $d_i = f(x_i) + e_i$ . Here  $f(x_i)$  is the noise-free value of the target function and  $e_i$  is a random variable representing the noise.
  - values of the  $e_i$  are drawn independently and that they are distributed according to a Normal distribution with zero mean

# Choose parameterized form for $P(Y|X; \theta)$



Assume Y is some deterministic  $f(X)$ , plus random noise

$$y = f(x) + \epsilon \quad \text{where } \epsilon \sim N(0, \sigma)$$

Therefore Y is a random variable that follows the distribution

$$p(y|x) = N(f(x), \sigma)$$

and the expected value of y for any given x is  $f(x)$

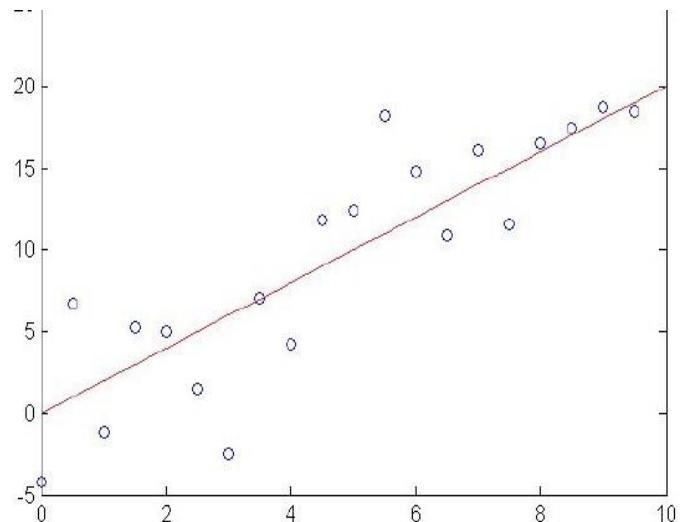
# Consider Linear Regression

$$p(y|x) = N(f(x), \sigma)$$

E.g., assume  $f(x)$  is linear function of  $x$

$$p(y|x) = N(w_0 + w_1x, \sigma)$$

$$E[y|x] = w_0 + w_1x$$



Notation: to make our parameters explicit, let's write

$$W = \langle w_0, w_1 \rangle$$

$$p(y|x; W) = N(w_0 + w_1x, \sigma)$$

# Training Linear Regression : Maximum Conditional Likelihood Estimate (MCLE)

$$p(y|x; W) = N(w_0 + w_1x, \sigma)$$

How can we learn  $W$  from the training data?

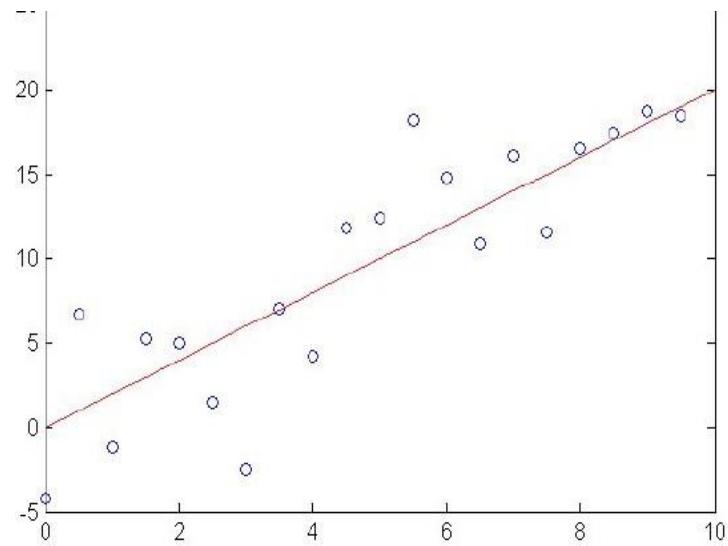
Learn Maximum Conditional Likelihood Estimate!

$$W_{MCLE} = \arg \max_W \prod_l p(y^l|x^l, W)$$

$$W_{MCLE} = \arg \max_W \sum_l \ln p(y^l|x^l, W)$$

where

$$p(y|x; W) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{y-f(x;W)}{\sigma})^2}$$



# Training Linear Regression: MCLE

Learn Maximum Conditional Likelihood Estimate

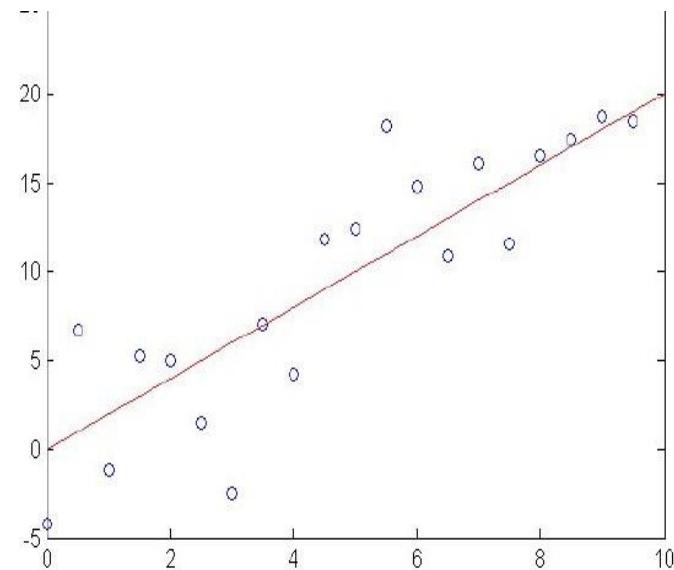
$$W_{MCLE} = \arg \max_W \sum_l \ln p(y^l | x^l, W)$$

where

$$p(y|x; W) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{y-f(x;W)}{\sigma})^2}$$

so:

$$W_{MCLE} = \arg \min_W \sum_l (y - f(x; W))^2$$



# Training Linear Regression: MCLE, MLE



$$h_{ML} = \operatorname{argmax}_{h \in H} \sum_{i=1}^m \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} (d_i - h(x_i))^2$$

The first term in this expression is a constant independent of  $h$ , and can therefore be discarded, yielding

$$h_{ML} = \operatorname{argmax}_{h \in H} \sum_{i=1}^m -\frac{1}{2\sigma^2} (d_i - h(x_i))^2$$

Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity.

$$h_{ML} = \operatorname{argmin}_{h \in H} \sum_{i=1}^m \frac{1}{2\sigma^2} (d_i - h(x_i))^2$$

Finally, we can again discard constants that are independent of  $h$ .

$$h_{ML} = \operatorname{argmin}_{h \in H} \sum_{i=1}^m (d_i - h(x_i))^2 \tag{6.6}$$

# Training Linear Regression

Maximum Conditional Likelihood Estimate is equivalent to minimizing the squared error loss

$$W_{MCLE} = \arg \min_W \sum_l (y - f(x; W))^2$$

Can we derive gradient descent rule for training?

$$\begin{aligned}\frac{\partial \sum_l (y - f(x; W))^2}{\partial w_i} &= \sum_l 2(y - f(x; W)) \frac{\partial (y - f(x; W))}{\partial w_i} \\ &= \sum_l -2(y - f(x; W)) \frac{\partial f(x; W)}{\partial w_i}\end{aligned}$$

# Regression – What you should know

---

Under general assumption

$$p(y|x; W) = N(f(x; W), \sigma)$$

- We can use gradient descent as a general learning algorithm
  - as long as our objective function is differentiable wrt W
  - though we might learn local optima
- Almost nothing we said here required that  $f(x)$  be linear in  $x$

# Extending Linear Regression to More Complex Models

- The inputs  $\mathbf{X}$  for linear regression can be:
  - Original quantitative inputs
  - Transformation of quantitative inputs
    - e.g. log, exp, square root, square, etc.
  - Polynomial transformation
    - example:  $y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \beta_3 \cdot x^3$
  - Basis expansions
  - Dummy coding of categorical inputs
  - Interactions between variables
    - example:  $x_3 = x_1 \cdot x_2$

This allows use of linear regression techniques to fit non-linear datasets.

# Linear Basis Function Models

# Linear Basis Function

---

- Simplest linear model for regression is one that involves a linear combination of the input variables

$$y(x, w) = w_0 + w_1 x_1 + \dots + w_D x_D$$

- Extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables, of the form

$$y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j \varphi_j(x)$$

- where  $\varphi_j(x)$  are known as basis functions.
- By denoting the maximum value of the index j by M - 1, the total number of parameters in this model will be M.

# Linear Basis Function

---

- Convenient to define an additional dummy ‘basis function’  $\phi_0(x)=1$ . So,

$$y(x, w) = \sum_{j=1}^{M-1} w_j \varphi_j(x) = \mathbf{w}^\top \boldsymbol{\Phi}_j(x)$$

where  $w = (w_0, \dots, w_{M-1})^T$  and  $\boldsymbol{\Phi} = (\phi_0, \phi_1, \dots, \phi_n)$

- If the original variables comprise the vector  $x$ , then the features can be expressed in terms of the basis functions  $\{\phi_j(x)\}$

# Linear Basis Function Models

- Generally,

$$h_{\theta}(\mathbf{x}) = \sum_{j=0}^d \theta_j \phi_j(\mathbf{x})$$

basis function

- Typically,  $\phi_0(\mathbf{x}) = 1$  so that  $\theta_0$  acts as a bias
- In the simplest case, we use linear basis functions :

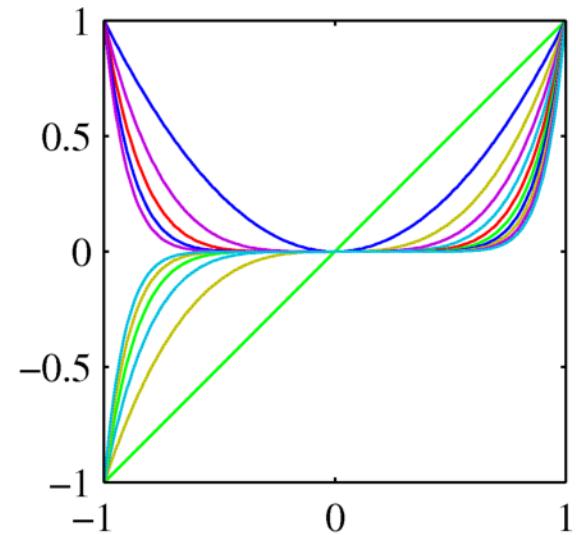
$$\phi_j(\mathbf{x}) = x_j$$

# Linear Basis Function Models

- Polynomial basis functions:

$$\phi_j(x) = x^j$$

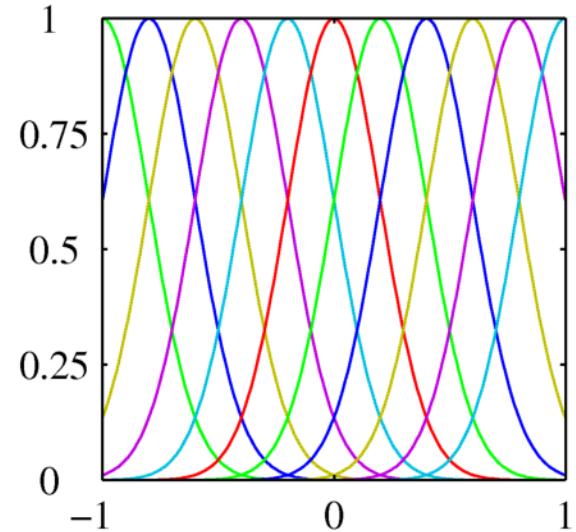
- These are global; a small change in  $x$  affects all basis functions



- Gaussian basis functions:

$$\phi_j(x) = \exp \left\{ -\frac{(x - \mu_j)^2}{2s^2} \right\}$$

- These are local; a small change in  $x$  only affect nearby basis functions.  $\mu_j$  and  $s$  control location and scale (width).



# Linear Basis Function Models

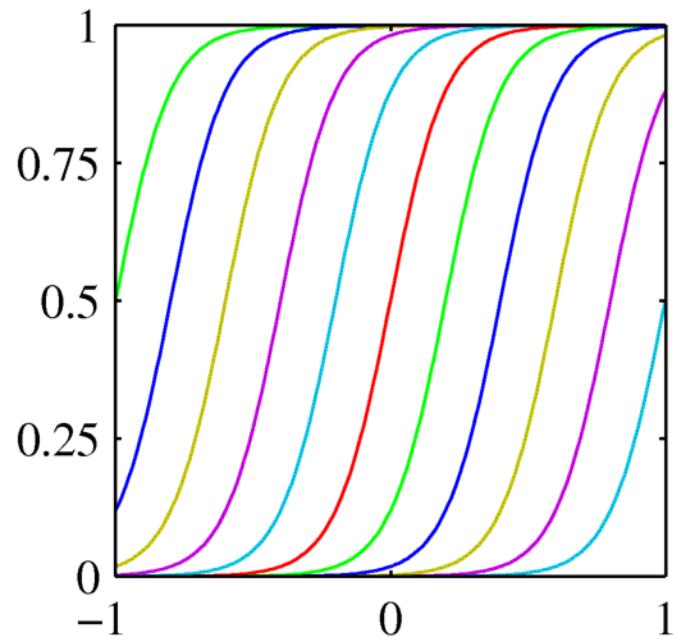
- Sigmoidal basis functions:

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

where

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

- These are also local; a small change in  $x$  only affects nearby basis functions.  $\mu_j$  and  $s$  control location and scale (slope).



# Linear Basis Function Models

- Basic Linear Model:

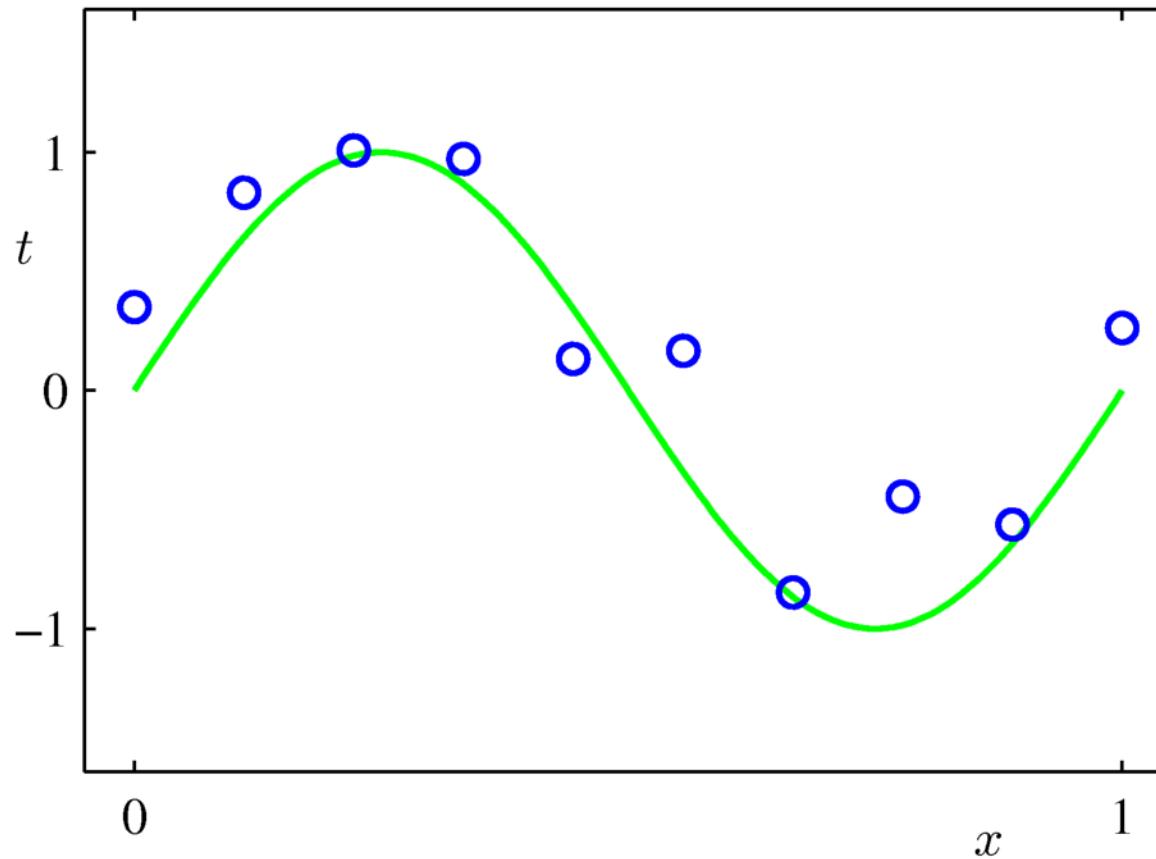
$$h_{\theta}(\mathbf{x}) = \sum_{j=0}^d \theta_j x_j$$

- Generalized Linear Model:

$$h_{\theta}(\mathbf{x}) = \sum_{j=0}^d \theta_j \phi_j(\mathbf{x})$$

- Once we have replaced the data by the outputs of the basis functions, fitting the generalized model is exactly the same problem as fitting the basic model
  - Unless we use the kernel trick – more on that when we cover support vector machines
  - Therefore, there is no point in cluttering the math with basis functions

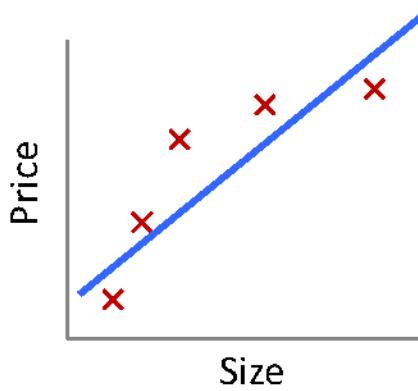
# Example of Fitting a Polynomial Curve with a Linear Model



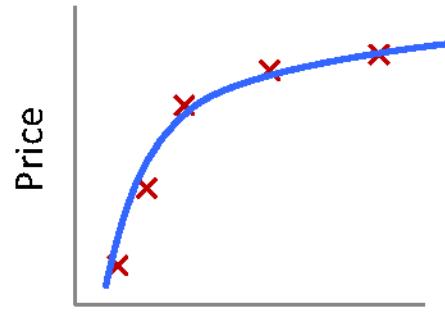
$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_p x^p = \sum_{j=0}^p \theta_j x^j$$

# Bias-Variance Decomposition

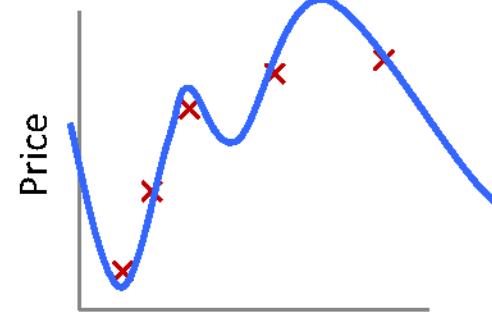
# Quality of Fit



Underfitting  
(high bias)



Correct fit



Overfitting  
(high variance)

## Overfitting:

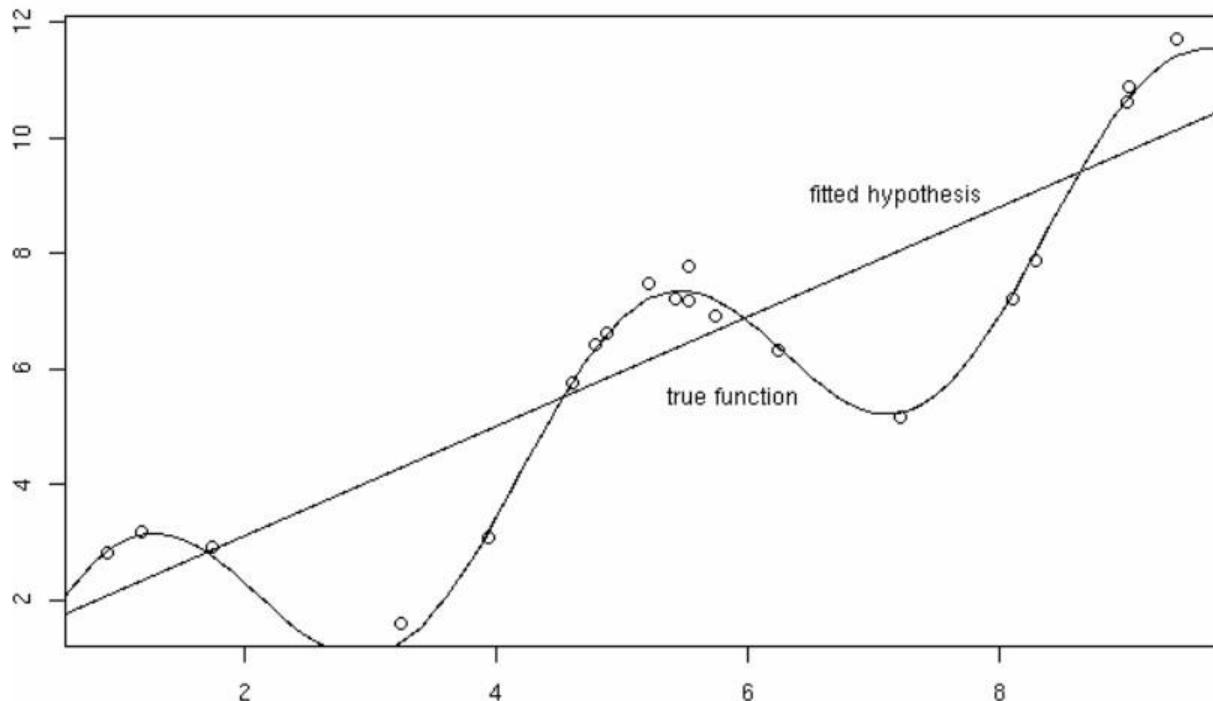
- The learned hypothesis may fit the training set very well ( $J(\theta) \approx 0$ )
- ...but fails to generalize to new examples

# Bias-Variance Tradeoff

- **Bias:** difference between  
**what you expect to learn** and **truth**
  - Measures how well you expect to represent true solution
  - Decreases with more complex model
  
- **Variance:** difference between  
**what you expect to learn** and  
**what you learn from a particular dataset**
  - Measures how sensitive learner is to specific dataset
  - Increases with more complex model

# Example

Tom Dietterich, Oregon St



True Function :  
 $y=f(x)$

$$y = x + 2 \sin(1.5x) + N(0,0.2)$$

# Bias – Variance decomposition of error

$$E_{D,\varepsilon} \left\{ (f(x) + \varepsilon - h_D(x))^2 \right\}$$

dataset and noise      true function      noise      learned from D

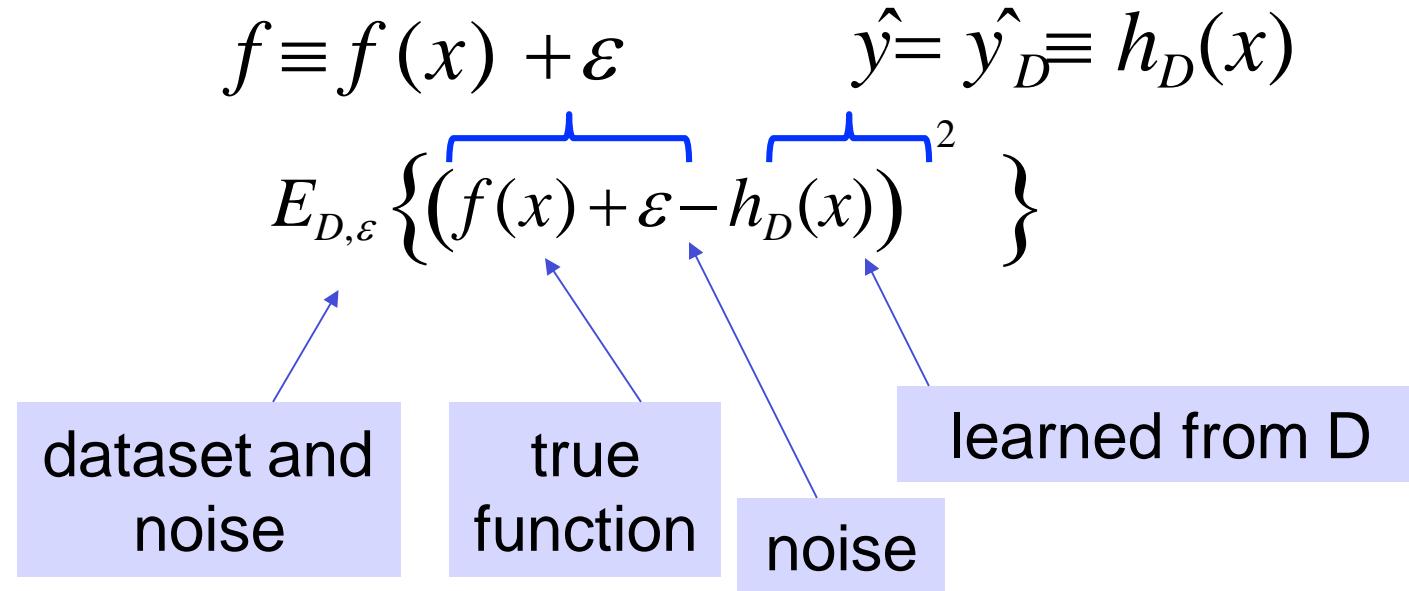
Fix test case  $x$ , then do this experiment:

1. Draw size  $n$  sample  $D = (x_1, y_1), \dots, (x_n, y_n)$
2. Train linear regressor  $h_D$  using  $D$
3. Draw one test example  $(x, f(x) + \varepsilon)$
4. Measure squared error of  $h_D$  on that example  $x$
5. What's the expected error?

# Bias – Variance decomposition of error

Notation - to simplify this

$$f \equiv f(x) + \varepsilon \quad \hat{y} = \hat{y}_D \equiv h_D(x)$$

$$E_{D,\varepsilon} \left\{ (f(x) + \varepsilon - h_D(x))^2 \right\}$$


dataset and noise      true function      noise      learned from D

$$h \equiv E_D \{ h_D(x) \}$$

long-term expectation of learner's prediction  
on this  $x$  averaged over many data sets  $D$

# Bias – Variance decomposition of error

$$\begin{aligned}
 & E_{D,\varepsilon} \left\{ (\hat{y}_D - y)^2 \right\} & h = E_D \left\{ \hat{y}_D \right\} \\
 & = E \left\{ ([f-h] + [h-\hat{y}_D])^2 \right\} & \hat{y}_D = \hat{y}_D(x) \\
 & = E \left\{ [f-h]^2 + [h-\hat{y}_D]^2 + 2[f-h][h-\hat{y}_D] \right\} & f \equiv f(x) + \varepsilon \\
 & = E \left\{ [f-h]^2 + [h-\hat{y}_D]^2 + 2[fh - f\hat{y}_D - h^2 + h\hat{y}_D] \right\} & \\
 & = E[(f-h)^2] + E[(h-\hat{y}_D)^2] + 2 \left( \cancel{E[fh]} - \cancel{E[f\hat{y}_D]} - \cancel{E[h^2]} + \cancel{E[h\hat{y}_D]} \right)
 \end{aligned}$$

# Bias – Variance decomposition of error

$$\begin{aligned}
 & E_{D,\varepsilon} \left\{ (\hat{y}_D - y)^2 \right\} & h = E_D \left\{ \hat{y}_D \right\} \\
 & = E \left\{ ([f-h] + [h-\hat{y}_D])^2 \right\} & \hat{y}_D = \hat{y}_D(x) \\
 & = E \left\{ [f-h]^2 + [h-\hat{y}_D]^2 + 2[f-h][h-\hat{y}_D] \right\} & f \equiv f(x) + \varepsilon \\
 & = E \left\{ [f-h]^2 + [h-\hat{y}_D]^2 + 2[fh - f\hat{y}_D - h^2 + h\hat{y}_D] \right\} & \\
 & = E[(f-h)^2] + E[(h-\hat{y}_D)^2] + 2 \left( \cancel{E[fh]} - \cancel{E[f\hat{y}_D]} - \cancel{E[h^2]} + \cancel{E[h\hat{y}_D]} \right)
 \end{aligned}$$

# Bias – Variance decomposition of error

$$h = E_D \{ \hat{y}_D \} \quad \hat{y} = \hat{y}_D$$

$$f \equiv f(x) + \varepsilon$$

$$\begin{aligned}
 & 2(E[fh] - E[fy] - E[h^2] + E[hy]) \\
 &= E[f]E[h] - E[f]E[y] - E[h]E[h] + E[h]E[y] \\
 &= E[f]E[E[y]] - E[f]E[y] - E[h]E[E[y]] + E[h]E[y] \\
 &= E[f]E[y] - E[f]E[y] - E[h]E[y] + E[h]E[y] \\
 &= 0
 \end{aligned}$$

# Bias – Variance decomposition of error

$$\begin{aligned}
 & E_{D,\varepsilon} \{(f - \hat{y})^2\} \\
 &= E \left\{ ([f - h] + [h - \hat{y}])^2 \right\} \\
 &= E \left\{ [f - h]^2 + [h - \hat{y}]^2 + 2[f - h][h - \hat{y}] \right\} \\
 &= E[(f - h)^2] + E[(h - \hat{y})^2]
 \end{aligned}$$

$$\begin{aligned}
 h &\equiv E_D\{h_D(x)\} \\
 \hat{y} &\equiv \hat{y}_D \equiv h_D(x) \\
 f &\equiv f(x) + \varepsilon
 \end{aligned}$$

BIAS<sup>2</sup>

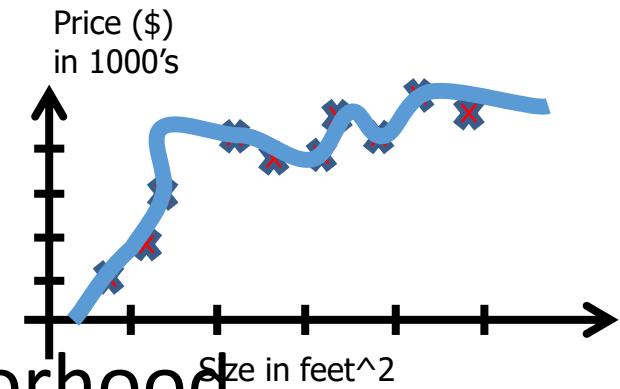
Squared difference between best possible prediction for  $x$ ,  $f(x)$ , and our “long-term” expectation for what the learner will do if we averaged over many datasets  $D$ ,  $E_D[h_D(x)]$

VARIANCE

Squared difference btwn our long-term expectation for the learners performance,  $E_D[h_D(x)]$ , and what we expect in a representative run on a dataset D ( $\hat{y}$ )

# 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
- :
- $x_{100}$



Slide credit: Andrew Ng

# Addressing overfitting

---

- **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  $\theta_j$ .
  - Works well when we have a lot of features, each of which contributes a bit to predicting  $y$ .

Slide credit: Andrew Ng

# Regularization

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

# Regularization

- Linear regression objective function

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

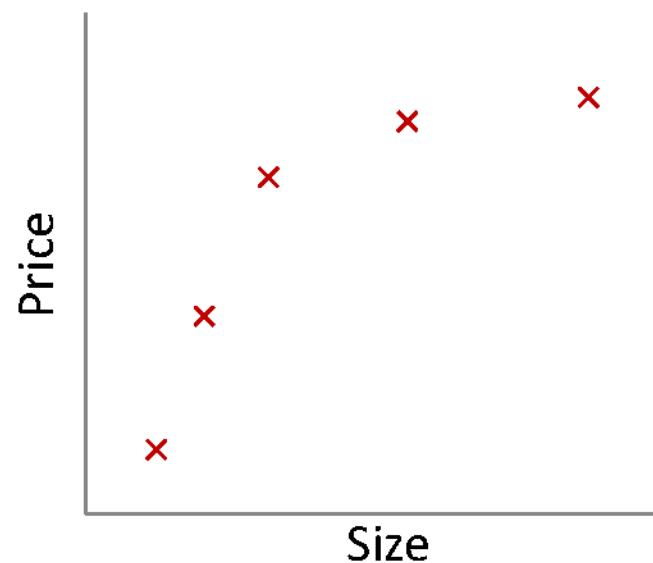

The equation represents the cost function for linear regression. It consists of two parts: a sum of squared differences between the predicted values  $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$  and the actual values  $y^{(i)}$  for all data points  $i$ , scaled by  $\frac{1}{2n}$ ; and a regularization term, which is a sum of the squares of all model parameters  $\theta_j$  for  $j$  from 1 to  $d$ , scaled by  $\frac{\lambda}{2}$ . The regularization parameter  $\lambda$  controls the trade-off between fitting the data well and keeping the model parameters small.

- $\lambda$  is the regularization parameter ( $\lambda \geq 0$ )
- No regularization on  $\theta_0$ !

# Understanding Regularization

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

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

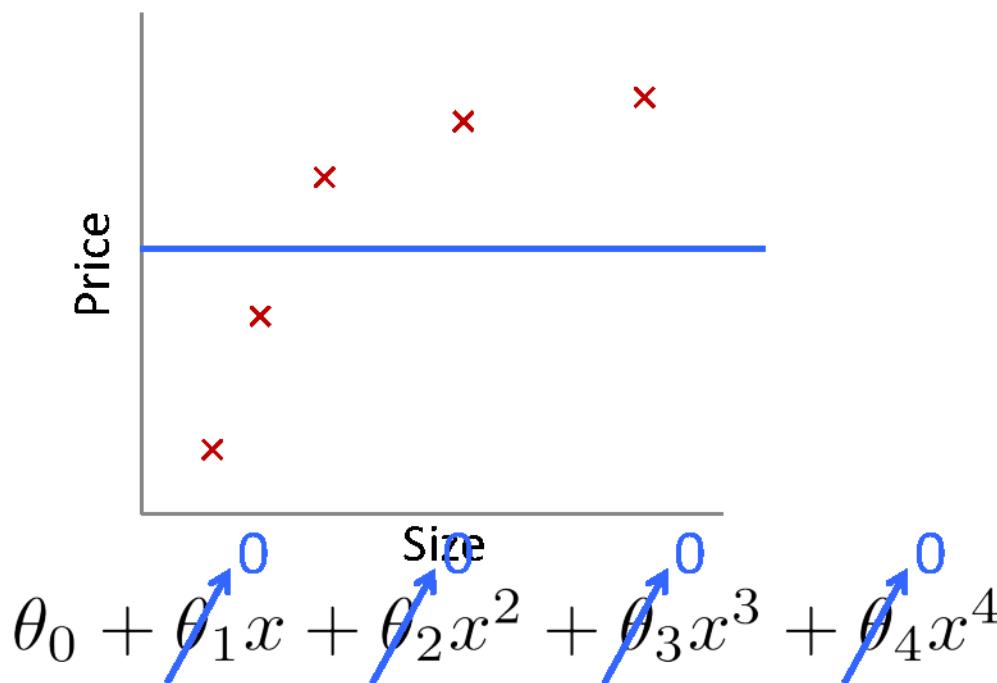


$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

# Understanding Regularization

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

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



# Regularized Linear Regression

- Cost Function

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

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

- Gradient update:

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

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

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

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

# Regularized Linear Regression

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

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

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

- We can rewrite the gradient step as:

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

---

## Funny but simple explanation about bias and variance

- <https://www.youtube.com/watch?v=EuBBz3bl-aA>

# In our next session

We will cover:

Tom Mitchell - Chapter 3

- Decision Tree
- Handling overfitting
- Continuous values
- Missing Values
- Random Forest

# CS229 Lecture notes

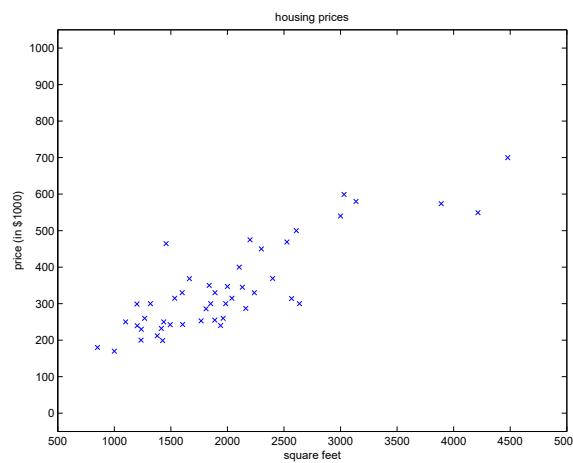
Andrew Ng

## Supervised learning

Let's start by talking about a few examples of supervised learning problems. Suppose we have a dataset giving the living areas and prices of 47 houses from Portland, Oregon:

Living area (feet <sup>2</sup> )	Price (1000\$)
2104	400
1600	330
2400	369
1416	232
3000	540
:	:

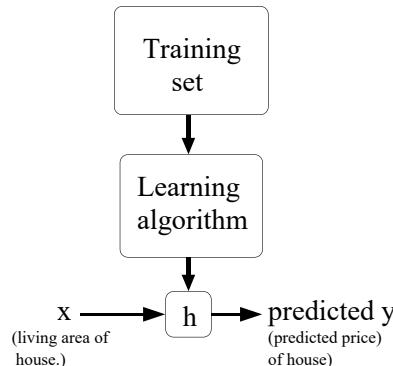
We can plot this data:



Given data like this, how can we learn to predict the prices of other houses in Portland, as a function of the size of their living areas?

To establish notation for future use, we'll use  $x^{(i)}$  to denote the “input” variables (living area in this example), also called input **features**, and  $y^{(i)}$  to denote the “output” or **target** variable that we are trying to predict (price). A pair  $(x^{(i)}, y^{(i)})$  is called a **training example**, and the dataset that we'll be using to learn—a list of  $m$  training examples  $\{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$ —is called a **training set**. Note that the superscript “ $(i)$ ” in the notation is simply an index into the training set, and has nothing to do with exponentiation. We will also use  $\mathcal{X}$  denote the space of input values, and  $\mathcal{Y}$  the space of output values. In this example,  $\mathcal{X} = \mathcal{Y} = \mathbb{R}$ .

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function  $h : \mathcal{X} \mapsto \mathcal{Y}$  so that  $h(x)$  is a “good” predictor for the corresponding value of  $y$ . For historical reasons, this function  $h$  is called a **hypothesis**. Seen pictorially, the process is therefore like this:



When the target variable that we're trying to predict is continuous, such as in our housing example, we call the learning problem a **regression** problem. When  $y$  can take on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment, say), we call it a **classification** problem.

## Part I

# Linear Regression

To make our housing example more interesting, let's consider a slightly richer dataset in which we also know the number of bedrooms in each house:

Living area (feet <sup>2</sup> )	#bedrooms	Price (1000\$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
:	:	:

Here, the  $x$ 's are two-dimensional vectors in  $\mathbb{R}^2$ . For instance,  $x_1^{(i)}$  is the living area of the  $i$ -th house in the training set, and  $x_2^{(i)}$  is its number of bedrooms. (In general, when designing a learning problem, it will be up to you to decide what features to choose, so if you are out in Portland gathering housing data, you might also decide to include other features such as whether each house has a fireplace, the number of bathrooms, and so on. We'll say more about feature selection later, but for now let's take the features as given.)

To perform supervised learning, we must decide how we're going to represent functions/hypotheses  $h$  in a computer. As an initial choice, let's say we decide to approximate  $y$  as a linear function of  $x$ :

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

Here, the  $\theta_i$ 's are the **parameters** (also called **weights**) parameterizing the space of linear functions mapping from  $\mathcal{X}$  to  $\mathcal{Y}$ . When there is no risk of confusion, we will drop the  $\theta$  subscript in  $h_\theta(x)$ , and write it more simply as  $h(x)$ . To simplify our notation, we also introduce the convention of letting  $x_0 = 1$  (this is the **intercept term**), so that

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$

where on the right-hand side above we are viewing  $\theta$  and  $x$  both as vectors, and here  $n$  is the number of input variables (not counting  $x_0$ ).

Now, given a training set, how do we pick, or learn, the parameters  $\theta$ ? One reasonable method seems to be to make  $h(x)$  close to  $y$ , at least for the training examples we have. To formalize this, we will define a function that measures, for each value of the  $\theta$ 's, how close the  $h(x^{(i)})$ 's are to the corresponding  $y^{(i)}$ 's. We define the **cost function**:

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

If you've seen linear regression before, you may recognize this as the familiar least-squares cost function that gives rise to the **ordinary least squares** regression model. Whether or not you have seen it previously, let's keep going, and we'll eventually show this to be a special case of a much broader family of algorithms.

## 1 LMS algorithm

We want to choose  $\theta$  so as to minimize  $J(\theta)$ . To do so, let's use a search algorithm that starts with some "initial guess" for  $\theta$ , and that repeatedly changes  $\theta$  to make  $J(\theta)$  smaller, until hopefully we converge to a value of  $\theta$  that minimizes  $J(\theta)$ . Specifically, let's consider the **gradient descent** algorithm, which starts with some initial  $\theta$ , and repeatedly performs the update:

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

(This update is simultaneously performed for all values of  $j = 0, \dots, n$ .) Here,  $\alpha$  is called the **learning rate**. This is a very natural algorithm that repeatedly takes a step in the direction of steepest decrease of  $J$ .

In order to implement this algorithm, we have to work out what is the partial derivative term on the right hand side. Let's first work it out for the case of if we have only one training example  $(x, y)$ , so that we can neglect the sum in the definition of  $J$ . We have:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\ &= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\ &= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left( \sum_{i=0}^n \theta_i x_i - y \right) \\ &= (h_\theta(x) - y) x_j \end{aligned}$$

For a single training example, this gives the update rule:<sup>1</sup>

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}.$$

The rule is called the **LMS** update rule (LMS stands for “least mean squares”), and is also known as the **Widrow-Hoff** learning rule. This rule has several properties that seem natural and intuitive. For instance, the magnitude of the update is proportional to the **error** term ( $y^{(i)} - h_\theta(x^{(i)})$ ); thus, for instance, if we are encountering a training example on which our prediction nearly matches the actual value of  $y^{(i)}$ , then we find that there is little need to change the parameters; in contrast, a larger change to the parameters will be made if our prediction  $h_\theta(x^{(i)})$  has a large error (i.e., if it is very far from  $y^{(i)}$ ).

We’d derived the LMS rule for when there was only a single training example. There are two ways to modify this method for a training set of more than one example. The first is replace it with the following algorithm:

Repeat until convergence {

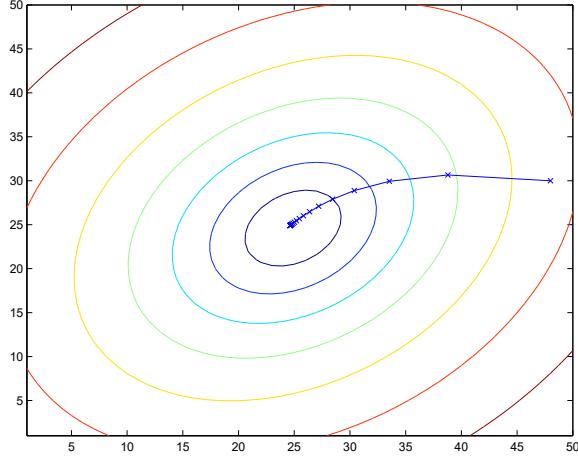
$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \quad (\text{for every } j).$$

}

The reader can easily verify that the quantity in the summation in the update rule above is just  $\partial J(\theta)/\partial\theta_j$  (for the original definition of  $J$ ). So, this is simply gradient descent on the original cost function  $J$ . This method looks at every example in the entire training set on every step, and is called **batch gradient descent**. Note that, while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate  $\alpha$  is not too large) to the global minimum. Indeed,  $J$  is a convex quadratic function. Here is an example of gradient descent as it is run to minimize a quadratic function.

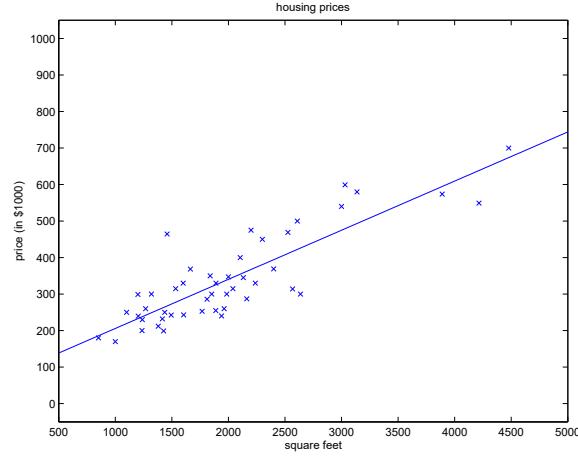
---

<sup>1</sup>We use the notation “ $a := b$ ” to denote an operation (in a computer program) in which we *set* the value of a variable  $a$  to be equal to the value of  $b$ . In other words, this operation overwrites  $a$  with the value of  $b$ . In contrast, we will write “ $a = b$ ” when we are asserting a statement of fact, that the value of  $a$  is equal to the value of  $b$ .



The ellipses shown above are the contours of a quadratic function. Also shown is the trajectory taken by gradient descent, which was initialized at (48,30). The  $x$ 's in the figure (joined by straight lines) mark the successive values of  $\theta$  that gradient descent went through.

When we run batch gradient descent to fit  $\theta$  on our previous dataset, to learn to predict housing price as a function of living area, we obtain  $\theta_0 = 71.27$ ,  $\theta_1 = 0.1345$ . If we plot  $h_\theta(x)$  as a function of  $x$  (area), along with the training data, we obtain the following figure:



If the number of bedrooms were included as one of the input features as well, we get  $\theta_0 = 89.60$ ,  $\theta_1 = 0.1392$ ,  $\theta_2 = -8.738$ .

The above results were obtained with batch gradient descent. There is an alternative to batch gradient descent that also works very well. Consider the following algorithm:

```

Loop {
    for i=1 to m, {
         $\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$  (for every  $j$ ).
    }
}

```

In this algorithm, we repeatedly run through the training set, and each time we encounter a training example, we update the parameters according to the gradient of the error with respect to that single training example only. This algorithm is called **stochastic gradient descent** (also **incremental gradient descent**). Whereas batch gradient descent has to scan through the entire training set before taking a single step—a costly operation if  $m$  is large—stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at. Often, stochastic gradient descent gets  $\theta$  “close” to the minimum much faster than batch gradient descent. (Note however that it may never “converge” to the minimum, and the parameters  $\theta$  will keep oscillating around the minimum of  $J(\theta)$ ; but in practice most of the values near the minimum will be reasonably good approximations to the true minimum.<sup>2)</sup> For these reasons, particularly when the training set is large, stochastic gradient descent is often preferred over batch gradient descent.

## 2 The normal equations

Gradient descent gives one way of minimizing  $J$ . Let’s discuss a second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm. In this method, we will minimize  $J$  by explicitly taking its derivatives with respect to the  $\theta_j$ ’s, and setting them to zero. To enable us to do this without having to write reams of algebra and pages full of matrices of derivatives, let’s introduce some notation for doing calculus with matrices.

---

<sup>2)</sup>While it is more common to run stochastic gradient descent as we have described it and with a fixed learning rate  $\alpha$ , by slowly letting the learning rate  $\alpha$  decrease to zero as the algorithm runs, it is also possible to ensure that the parameters will converge to the global minimum rather than merely oscillate around the minimum.

## 2.1 Matrix derivatives

For a function  $f : \mathbb{R}^{m \times n} \mapsto \mathbb{R}$  mapping from  $m$ -by- $n$  matrices to the real numbers, we define the derivative of  $f$  with respect to  $A$  to be:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}$$

Thus, the gradient  $\nabla_A f(A)$  is itself an  $m$ -by- $n$  matrix, whose  $(i, j)$ -element is  $\partial f / \partial A_{ij}$ . For example, suppose  $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$  is a 2-by-2 matrix, and the function  $f : \mathbb{R}^{2 \times 2} \mapsto \mathbb{R}$  is given by

$$f(A) = \frac{3}{2}A_{11} + 5A_{12}^2 + A_{21}A_{22}.$$

Here,  $A_{ij}$  denotes the  $(i, j)$  entry of the matrix  $A$ . We then have

$$\nabla_A f(A) = \begin{bmatrix} \frac{3}{2} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}.$$

We also introduce the **trace** operator, written “tr.” For an  $n$ -by- $n$  (square) matrix  $A$ , the trace of  $A$  is defined to be the sum of its diagonal entries:

$$\text{tr}A = \sum_{i=1}^n A_{ii}$$

If  $a$  is a real number (i.e., a 1-by-1 matrix), then  $\text{tr } a = a$ . (If you haven’t seen this “operator notation” before, you should think of the trace of  $A$  as  $\text{tr}(A)$ , or as application of the “trace” function to the matrix  $A$ . It’s more commonly written without the parentheses, however.)

The trace operator has the property that for two matrices  $A$  and  $B$  such that  $AB$  is square, we have that  $\text{tr}AB = \text{tr}BA$ . (Check this yourself!) As corollaries of this, we also have, e.g.,

$$\text{tr}ABC = \text{tr}CAB = \text{tr}BCA,$$

$$\text{tr}ABCD = \text{tr}DABC = \text{tr}CDAB = \text{tr}BCDA.$$

The following properties of the trace operator are also easily verified. Here,  $A$  and  $B$  are square matrices, and  $a$  is a real number:

$$\text{tr}A = \text{tr}A^T$$

$$\text{tr}(A + B) = \text{tr}A + \text{tr}B$$

$$\text{tr } aA = a\text{tr}A$$

We now state without proof some facts of matrix derivatives (we won't need some of these until later this quarter). Equation (4) applies only to non-singular square matrices  $A$ , where  $|A|$  denotes the determinant of  $A$ . We have:

$$\nabla_A \text{tr}AB = B^T \quad (1)$$

$$\nabla_{AT} f(A) = (\nabla_A f(A))^T \quad (2)$$

$$\nabla_A \text{tr}ABA^TC = CAB + C^T AB^T \quad (3)$$

$$\nabla_A |A| = |A|(A^{-1})^T. \quad (4)$$

To make our matrix notation more concrete, let us now explain in detail the meaning of the first of these equations. Suppose we have some fixed matrix  $B \in \mathbb{R}^{n \times m}$ . We can then define a function  $f : \mathbb{R}^{m \times n} \mapsto \mathbb{R}$  according to  $f(A) = \text{tr}AB$ . Note that this definition makes sense, because if  $A \in \mathbb{R}^{m \times n}$ , then  $AB$  is a square matrix, and we can apply the trace operator to it; thus,  $f$  does indeed map from  $\mathbb{R}^{m \times n}$  to  $\mathbb{R}$ . We can then apply our definition of matrix derivatives to find  $\nabla_A f(A)$ , which will itself by an  $m$ -by- $n$  matrix. Equation (1) above states that the  $(i, j)$  entry of this matrix will be given by the  $(i, j)$ -entry of  $B^T$ , or equivalently, by  $B_{ji}$ .

The proofs of Equations (1-3) are reasonably simple, and are left as an exercise to the reader. Equations (4) can be derived using the adjoint representation of the inverse of a matrix.<sup>3</sup>

## 2.2 Least squares revisited

Armed with the tools of matrix derivatives, let us now proceed to find in closed-form the value of  $\theta$  that minimizes  $J(\theta)$ . We begin by re-writing  $J$  in matrix-vectorial notation.

Given a training set, define the **design matrix**  $X$  to be the  $m$ -by- $n$  matrix (actually  $m$ -by- $n + 1$ , if we include the intercept term) that contains

---

<sup>3</sup>If we define  $A'$  to be the matrix whose  $(i, j)$  element is  $(-1)^{i+j}$  times the determinant of the square matrix resulting from deleting row  $i$  and column  $j$  from  $A$ , then it can be proved that  $A^{-1} = (A')^T/|A|$ . (You can check that this is consistent with the standard way of finding  $A^{-1}$  when  $A$  is a 2-by-2 matrix. If you want to see a proof of this more general result, see an intermediate or advanced linear algebra text, such as Charles Curtis, 1991, *Linear Algebra*, Springer.) This shows that  $A' = |A|(A^{-1})^T$ . Also, the determinant of a matrix can be written  $|A| = \sum_j A_{ij} A'_{ij}$ . Since  $(A')_{ij}$  does not depend on  $A_{ij}$  (as can be seen from its definition), this implies that  $(\partial/\partial A_{ij})|A| = A'_{ij}$ . Putting all this together shows the result.

the training examples' input values in its rows:

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}.$$

Also, let  $\vec{y}$  be the  $m$ -dimensional vector containing all the target values from the training set:

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

Now, since  $h_\theta(x^{(i)}) = (x^{(i)})^T\theta$ , we can easily verify that

$$\begin{aligned} X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(m)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}. \end{aligned}$$

Thus, using the fact that for a vector  $z$ , we have that  $z^T z = \sum_i z_i^2$ :

$$\begin{aligned} \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) &= \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= J(\theta) \end{aligned}$$

Finally, to minimize  $J$ , let's find its derivatives with respect to  $\theta$ . Combining Equations (2) and (3), we find that

$$\nabla_{A^T} \text{tr}ABA^TC = B^TA^TC^T + BA^TC \quad (5)$$

Hence,

$$\begin{aligned}
\nabla_{\theta} J(\theta) &= \nabla_{\theta} \frac{1}{2} (\vec{X}\theta - \vec{y})^T (\vec{X}\theta - \vec{y}) \\
&= \frac{1}{2} \nabla_{\theta} (\theta^T X^T X \theta - \theta^T X^T \vec{y} - \vec{y}^T X \theta + \vec{y}^T \vec{y}) \\
&= \frac{1}{2} \nabla_{\theta} \text{tr} (\theta^T X^T X \theta - \theta^T X^T \vec{y} - \vec{y}^T X \theta + \vec{y}^T \vec{y}) \\
&= \frac{1}{2} \nabla_{\theta} (\text{tr} \theta^T X^T X \theta - 2 \text{tr} \vec{y}^T X \theta) \\
&= \frac{1}{2} (X^T X \theta + X^T X \theta - 2 X^T \vec{y}) \\
&= X^T X \theta - X^T \vec{y}
\end{aligned}$$

In the third step, we used the fact that the trace of a real number is just the real number; the fourth step used the fact that  $\text{tr}A = \text{tr}A^T$ , and the fifth step used Equation (5) with  $A^T = \theta$ ,  $B = B^T = X^T X$ , and  $C = I$ , and Equation (1). To minimize  $J$ , we set its derivatives to zero, and obtain the **normal equations**:

$$X^T X \theta = X^T \vec{y}$$

Thus, the value of  $\theta$  that minimizes  $J(\theta)$  is given in closed form by the equation

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

### 3 Probabilistic interpretation

When faced with a regression problem, why might linear regression, and specifically why might the least-squares cost function  $J$ , be a reasonable choice? In this section, we will give a set of probabilistic assumptions, under which least-squares regression is derived as a very natural algorithm.

Let us assume that the target variables and the inputs are related via the equation

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)},$$

where  $\epsilon^{(i)}$  is an error term that captures either unmodeled effects (such as if there are some features very pertinent to predicting housing price, but that we'd left out of the regression), or random noise. Let us further assume that the  $\epsilon^{(i)}$  are distributed IID (independently and identically distributed) according to a Gaussian distribution (also called a Normal distribution) with

mean zero and some variance  $\sigma^2$ . We can write this assumption as “ $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ .” I.e., the density of  $\epsilon^{(i)}$  is given by

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right).$$

This implies that

$$p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right).$$

The notation “ $p(y^{(i)}|x^{(i)}; \theta)$ ” indicates that this is the distribution of  $y^{(i)}$  given  $x^{(i)}$  and parameterized by  $\theta$ . Note that we should not condition on  $\theta$  (“ $p(y^{(i)}|x^{(i)}, \theta)$ ”), since  $\theta$  is not a random variable. We can also write the distribution of  $y^{(i)}$  as  $y^{(i)} | x^{(i)}; \theta \sim \mathcal{N}(\theta^T x^{(i)}, \sigma^2)$ .

Given  $X$  (the design matrix, which contains all the  $x^{(i)}$ 's) and  $\theta$ , what is the distribution of the  $y^{(i)}$ 's? The probability of the data is given by  $p(\vec{y}|X; \theta)$ . This quantity is typically viewed a function of  $\vec{y}$  (and perhaps  $X$ ), for a fixed value of  $\theta$ . When we wish to explicitly view this as a function of  $\theta$ , we will instead call it the **likelihood** function:

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X; \theta).$$

Note that by the independence assumption on the  $\epsilon^{(i)}$ 's (and hence also the  $y^{(i)}$ 's given the  $x^{(i)}$ 's), this can also be written

$$\begin{aligned} L(\theta) &= \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta) \\ &= \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right). \end{aligned}$$

Now, given this probabilistic model relating the  $y^{(i)}$ 's and the  $x^{(i)}$ 's, what is a reasonable way of choosing our best guess of the parameters  $\theta$ ? The principle of **maximum likelihood** says that we should choose  $\theta$  so as to make the data as high probability as possible. I.e., we should choose  $\theta$  to maximize  $L(\theta)$ .

Instead of maximizing  $L(\theta)$ , we can also maximize any strictly increasing function of  $L(\theta)$ . In particular, the derivations will be a bit simpler if we

instead maximize the **log likelihood**  $\ell(\theta)$ :

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T x^{(i)})^2.\end{aligned}$$

Hence, maximizing  $\ell(\theta)$  gives the same answer as minimizing

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

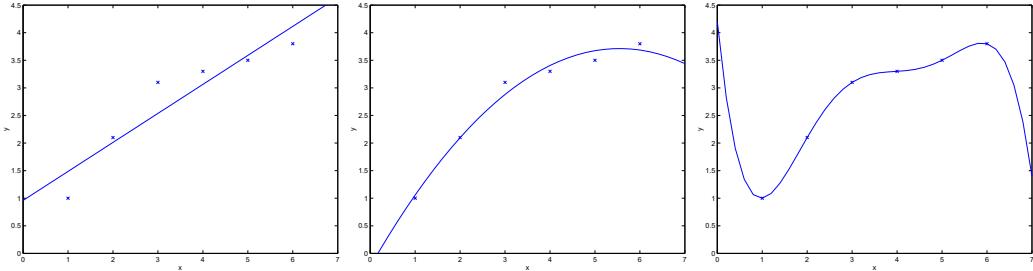
which we recognize to be  $J(\theta)$ , our original least-squares cost function.

To summarize: Under the previous probabilistic assumptions on the data, least-squares regression corresponds to finding the maximum likelihood estimate of  $\theta$ . This is thus one set of assumptions under which least-squares regression can be justified as a very natural method that's just doing maximum likelihood estimation. (Note however that the probabilistic assumptions are by no means *necessary* for least-squares to be a perfectly good and rational procedure, and there may—and indeed there are—other natural assumptions that can also be used to justify it.)

Note also that, in our previous discussion, our final choice of  $\theta$  did not depend on what was  $\sigma^2$ , and indeed we'd have arrived at the same result even if  $\sigma^2$  were unknown. We will use this fact again later, when we talk about the exponential family and generalized linear models.

## 4 Locally weighted linear regression

Consider the problem of predicting  $y$  from  $x \in \mathbb{R}$ . The leftmost figure below shows the result of fitting a  $y = \theta_0 + \theta_1 x$  to a dataset. We see that the data doesn't really lie on straight line, and so the fit is not very good.



Instead, if we had added an extra feature  $x^2$ , and fit  $y = \theta_0 + \theta_1 x + \theta_2 x^2$ , then we obtain a slightly better fit to the data. (See middle figure) Naively, it might seem that the more features we add, the better. However, there is also a danger in adding too many features: The rightmost figure is the result of fitting a 5-th order polynomial  $y = \sum_{j=0}^5 \theta_j x^j$ . We see that even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor of, say, housing prices ( $y$ ) for different living areas ( $x$ ). Without formally defining what these terms mean, we'll say the figure on the left shows an instance of **underfitting**—in which the data clearly shows structure not captured by the model—and the figure on the right is an example of **overfitting**. (Later in this class, when we talk about learning theory we'll formalize some of these notions, and also define more carefully just what it means for a hypothesis to be good or bad.)

As discussed previously, and as shown in the example above, the choice of features is important to ensuring good performance of a learning algorithm. (When we talk about model selection, we'll also see algorithms for automatically choosing a good set of features.) In this section, let us talk briefly talk about the locally weighted linear regression (LWR) algorithm which, assuming there is sufficient training data, makes the choice of features less critical. This treatment will be brief, since you'll get a chance to explore some of the properties of the LWR algorithm yourself in the homework.

In the original linear regression algorithm, to make a prediction at a query point  $x$  (i.e., to evaluate  $h(x)$ ), we would:

1. Fit  $\theta$  to minimize  $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$ .
2. Output  $\theta^T x$ .

In contrast, the locally weighted linear regression algorithm does the following:

1. Fit  $\theta$  to minimize  $\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$ .
2. Output  $\theta^T x$ .

Here, the  $w^{(i)}$ 's are non-negative valued **weights**. Intuitively, if  $w^{(i)}$  is large for a particular value of  $i$ , then in picking  $\theta$ , we'll try hard to make  $(y^{(i)} - \theta^T x^{(i)})^2$  small. If  $w^{(i)}$  is small, then the  $(y^{(i)} - \theta^T x^{(i)})^2$  error term will be pretty much ignored in the fit.

A fairly standard choice for the weights is<sup>4</sup>

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

Note that the weights depend on the particular point  $x$  at which we're trying to evaluate  $x$ . Moreover, if  $|x^{(i)} - x|$  is small, then  $w^{(i)}$  is close to 1; and if  $|x^{(i)} - x|$  is large, then  $w^{(i)}$  is small. Hence,  $\theta$  is chosen giving a much higher “weight” to the (errors on) training examples close to the query point  $x$ . (Note also that while the formula for the weights takes a form that is cosmetically similar to the density of a Gaussian distribution, the  $w^{(i)}$ 's do not directly have anything to do with Gaussians, and in particular the  $w^{(i)}$  are not random variables, normally distributed or otherwise.) The parameter  $\tau$  controls how quickly the weight of a training example falls off with distance of its  $x^{(i)}$  from the query point  $x$ ;  $\tau$  is called the **bandwidth** parameter, and is also something that you'll get to experiment with in your homework.

Locally weighted linear regression is the first example we're seeing of a **non-parametric** algorithm. The (unweighted) linear regression algorithm that we saw earlier is known as a **parametric** learning algorithm, because it has a fixed, finite number of parameters (the  $\theta_i$ 's), which are fit to the data. Once we've fit the  $\theta_i$ 's and stored them away, we no longer need to keep the training data around to make future predictions. In contrast, to make predictions using locally weighted linear regression, we need to keep the entire training set around. The term “non-parametric” (roughly) refers to the fact that the amount of stuff we need to keep in order to represent the hypothesis  $h$  grows linearly with the size of the training set.

---

<sup>4</sup>If  $x$  is vector-valued, this is generalized to be  $w^{(i)} = \exp(-(x^{(i)} - x)^T(x^{(i)} - x)/(2\tau^2))$ , or  $w^{(i)} = \exp(-(x^{(i)} - x)^T\Sigma^{-1}(x^{(i)} - x)/2)$ , for an appropriate choice of  $\tau$  or  $\Sigma$ .

## Part II

# Classification and logistic regression

Let's now talk about the classification problem. This is just like the regression problem, except that the values  $y$  we now want to predict take on only a small number of discrete values. For now, we will focus on the **binary classification** problem in which  $y$  can take on only two values, 0 and 1. (Most of what we say here will also generalize to the multiple-class case.) For instance, if we are trying to build a spam classifier for email, then  $x^{(i)}$  may be some features of a piece of email, and  $y$  may be 1 if it is a piece of spam mail, and 0 otherwise. 0 is also called the **negative class**, and 1 the **positive class**, and they are sometimes also denoted by the symbols “-” and “+.” Given  $x^{(i)}$ , the corresponding  $y^{(i)}$  is also called the **label** for the training example.

## 5 Logistic regression

We could approach the classification problem ignoring the fact that  $y$  is discrete-valued, and use our old linear regression algorithm to try to predict  $y$  given  $x$ . However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn't make sense for  $h_\theta(x)$  to take values larger than 1 or smaller than 0 when we know that  $y \in \{0, 1\}$ .

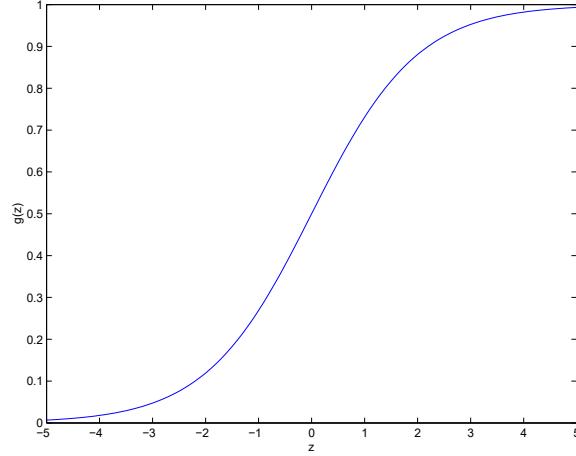
To fix this, let's change the form for our hypotheses  $h_\theta(x)$ . We will choose

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

where

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

is called the **logistic function** or the **sigmoid function**. Here is a plot showing  $g(z)$ :



Notice that  $g(z)$  tends towards 1 as  $z \rightarrow \infty$ , and  $g(z)$  tends towards 0 as  $z \rightarrow -\infty$ . Moreover,  $g(z)$ , and hence also  $h(x)$ , is always bounded between 0 and 1. As before, we are keeping the convention of letting  $x_0 = 1$ , so that  $\theta^T x = \theta_0 + \sum_{j=1}^n \theta_j x_j$ .

For now, let's take the choice of  $g$  as given. Other functions that smoothly increase from 0 to 1 can also be used, but for a couple of reasons that we'll see later (when we talk about GLMs, and when we talk about generative learning algorithms), the choice of the logistic function is a fairly natural one. Before moving on, here's a useful property of the derivative of the sigmoid function, which we write as  $g'$ :

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2} (e^{-z}) \\ &= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

So, given the logistic regression model, how do we fit  $\theta$  for it? Following how we saw least squares regression could be derived as the maximum likelihood estimator under a set of assumptions, let's endow our classification model with a set of probabilistic assumptions, and then fit the parameters via maximum likelihood.

Let us assume that

$$\begin{aligned} P(y = 1 \mid x; \theta) &= h_\theta(x) \\ P(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

Note that this can be written more compactly as

$$p(y \mid x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

Assuming that the  $m$  training examples were generated independently, we can then write down the likelihood of the parameters as

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X; \theta) \\ &= \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^m (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

As before, it will be easier to maximize the log likelihood:

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \end{aligned}$$

How do we maximize the likelihood? Similar to our derivation in the case of linear regression, we can use gradient ascent. Written in vectorial notation, our updates will therefore be given by  $\theta := \theta + \alpha \nabla_\theta \ell(\theta)$ . (Note the positive rather than negative sign in the update formula, since we're maximizing, rather than minimizing, a function now.) Let's start by working with just one training example  $(x, y)$ , and take derivatives to derive the stochastic gradient ascent rule:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} \ell(\theta) &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x) (1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x)) x_j \\ &= (y - h_\theta(x)) x_j \end{aligned}$$

Above, we used the fact that  $g'(z) = g(z)(1 - g(z))$ . This therefore gives us the stochastic gradient ascent rule

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

If we compare this to the LMS update rule, we see that it looks identical; but this is *not* the same algorithm, because  $h_\theta(x^{(i)})$  is now defined as a non-linear function of  $\theta^T x^{(i)}$ . Nonetheless, it's a little surprising that we end up with the same update rule for a rather different algorithm and learning problem. Is this coincidence, or is there a deeper reason behind this? We'll answer this when we get to GLM models. (See also the extra credit problem on Q3 of problem set 1.)

## 6 Digression: The perceptron learning algorithm

We now digress to talk briefly about an algorithm that's of some historical interest, and that we will also return to later when we talk about learning theory. Consider modifying the logistic regression method to "force" it to output values that are either 0 or 1 or exactly. To do so, it seems natural to change the definition of  $g$  to be the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

If we then let  $h_\theta(x) = g(\theta^T x)$  as before but using this modified definition of  $g$ , and if we use the update rule

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}.$$

then we have the **perceptron learning algorithm**.

In the 1960s, this "perceptron" was argued to be a rough model for how individual neurons in the brain work. Given how simple the algorithm is, it will also provide a starting point for our analysis when we talk about learning theory later in this class. Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression and least squares linear regression; in particular, it is difficult to endow the perceptron's predictions with meaningful probabilistic interpretations, or derive the perceptron as a maximum likelihood estimation algorithm.

## 7 Another algorithm for maximizing $\ell(\theta)$

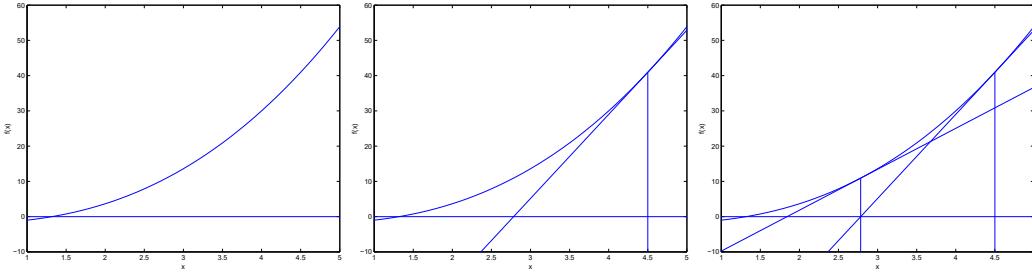
Returning to logistic regression with  $g(z)$  being the sigmoid function, let's now talk about a different algorithm for maximizing  $\ell(\theta)$ .

To get us started, let's consider Newton's method for finding a zero of a function. Specifically, suppose we have some function  $f : \mathbb{R} \mapsto \mathbb{R}$ , and we wish to find a value of  $\theta$  so that  $f(\theta) = 0$ . Here,  $\theta \in \mathbb{R}$  is a real number. Newton's method performs the following update:

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}.$$

This method has a natural interpretation in which we can think of it as approximating the function  $f$  via a linear function that is tangent to  $f$  at the current guess  $\theta$ , solving for where that linear function equals to zero, and letting the next guess for  $\theta$  be where that linear function is zero.

Here's a picture of the Newton's method in action:



In the leftmost figure, we see the function  $f$  plotted along with the line  $y = 0$ . We're trying to find  $\theta$  so that  $f(\theta) = 0$ ; the value of  $\theta$  that achieves this is about 1.3. Suppose we initialized the algorithm with  $\theta = 4.5$ . Newton's method then fits a straight line tangent to  $f$  at  $\theta = 4.5$ , and solves for the where that line evaluates to 0. (Middle figure.) This give us the next guess for  $\theta$ , which is about 2.8. The rightmost figure shows the result of running one more iteration, which the updates  $\theta$  to about 1.8. After a few more iterations, we rapidly approach  $\theta = 1.3$ .

Newton's method gives a way of getting to  $f(\theta) = 0$ . What if we want to use it to maximize some function  $\ell$ ? The maxima of  $\ell$  correspond to points where its first derivative  $\ell'(\theta)$  is zero. So, by letting  $f(\theta) = \ell'(\theta)$ , we can use the same algorithm to maximize  $\ell$ , and we obtain update rule:

$$\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}.$$

(Something to think about: How would this change if we wanted to use Newton's method to minimize rather than maximize a function?)

Lastly, in our logistic regression setting,  $\theta$  is vector-valued, so we need to generalize Newton's method to this setting. The generalization of Newton's method to this multidimensional setting (also called the Newton-Raphson method) is given by

$$\theta := \theta - H^{-1} \nabla_{\theta} \ell(\theta).$$

Here,  $\nabla_{\theta} \ell(\theta)$  is, as usual, the vector of partial derivatives of  $\ell(\theta)$  with respect to the  $\theta_i$ 's; and  $H$  is an  $n$ -by- $n$  matrix (actually,  $n + 1$ -by- $n + 1$ , assuming that we include the intercept term) called the **Hessian**, whose entries are given by

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}.$$

Newton's method typically enjoys faster convergence than (batch) gradient descent, and requires many fewer iterations to get very close to the minimum. One iteration of Newton's can, however, be more expensive than one iteration of gradient descent, since it requires finding and inverting an  $n$ -by- $n$  Hessian; but so long as  $n$  is not too large, it is usually much faster overall. When Newton's method is applied to maximize the logistic regression log likelihood function  $\ell(\theta)$ , the resulting method is also called **Fisher scoring**.

## Part III

# Generalized Linear Models<sup>5</sup>

So far, we've seen a regression example, and a classification example. In the regression example, we had  $y|x; \theta \sim \mathcal{N}(\mu, \sigma^2)$ , and in the classification one,  $y|x; \theta \sim \text{Bernoulli}(\phi)$ , for some appropriate definitions of  $\mu$  and  $\phi$  as functions of  $x$  and  $\theta$ . In this section, we will show that both of these methods are special cases of a broader family of models, called Generalized Linear Models (GLMs). We will also show how other models in the GLM family can be derived and applied to other classification and regression problems.

## 8 The exponential family

To work our way up to GLMs, we will begin by defining exponential family distributions. We say that a class of distributions is in the exponential family if it can be written in the form

$$p(y; \eta) = b(y) \exp(\eta^T T(y) - a(\eta)) \quad (6)$$

Here,  $\eta$  is called the **natural parameter** (also called the **canonical parameter**) of the distribution;  $T(y)$  is the **sufficient statistic** (for the distributions we consider, it will often be the case that  $T(y) = y$ ); and  $a(\eta)$  is the **log partition function**. The quantity  $e^{-a(\eta)}$  essentially plays the role of a normalization constant, that makes sure the distribution  $p(y; \eta)$  sums/integrates over  $y$  to 1.

A fixed choice of  $T$ ,  $a$  and  $b$  defines a *family* (or set) of distributions that is parameterized by  $\eta$ ; as we vary  $\eta$ , we then get different distributions within this family.

We now show that the Bernoulli and the Gaussian distributions are examples of exponential family distributions. The Bernoulli distribution with mean  $\phi$ , written  $\text{Bernoulli}(\phi)$ , specifies a distribution over  $y \in \{0, 1\}$ , so that  $p(y = 1; \phi) = \phi$ ;  $p(y = 0; \phi) = 1 - \phi$ . As we vary  $\phi$ , we obtain Bernoulli distributions with different means. We now show that this class of Bernoulli distributions, ones obtained by varying  $\phi$ , is in the exponential family; i.e., that there is a choice of  $T$ ,  $a$  and  $b$  so that Equation (6) becomes exactly the class of Bernoulli distributions.

---

<sup>5</sup>The presentation of the material in this section takes inspiration from Michael I. Jordan, *Learning in graphical models* (unpublished book draft), and also McCullagh and Nelder, *Generalized Linear Models* (2nd ed.).

We write the Bernoulli distribution as:

$$\begin{aligned} p(y; \phi) &= \phi^y (1 - \phi)^{1-y} \\ &= \exp(y \log \phi + (1 - y) \log(1 - \phi)) \\ &= \exp\left(\left(\log\left(\frac{\phi}{1 - \phi}\right)\right)y + \log(1 - \phi)\right). \end{aligned}$$

Thus, the natural parameter is given by  $\eta = \log(\phi/(1 - \phi))$ . Interestingly, if we invert this definition for  $\eta$  by solving for  $\phi$  in terms of  $\eta$ , we obtain  $\phi = 1/(1 + e^{-\eta})$ . This is the familiar sigmoid function! This will come up again when we derive logistic regression as a GLM. To complete the formulation of the Bernoulli distribution as an exponential family distribution, we also have

$$\begin{aligned} T(y) &= y \\ a(\eta) &= -\log(1 - \phi) \\ &= \log(1 + e^\eta) \\ b(y) &= 1 \end{aligned}$$

This shows that the Bernoulli distribution can be written in the form of Equation (6), using an appropriate choice of  $T$ ,  $a$  and  $b$ .

Let's now move on to consider the Gaussian distribution. Recall that, when deriving linear regression, the value of  $\sigma^2$  had no effect on our final choice of  $\theta$  and  $h_\theta(x)$ . Thus, we can choose an arbitrary value for  $\sigma^2$  without changing anything. To simplify the derivation below, let's set  $\sigma^2 = 1$ .<sup>6</sup> We then have:

$$\begin{aligned} p(y; \mu) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y - \mu)^2\right) \\ &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \cdot \exp\left(\mu y - \frac{1}{2}\mu^2\right) \end{aligned}$$

---

<sup>6</sup>If we leave  $\sigma^2$  as a variable, the Gaussian distribution can also be shown to be in the exponential family, where  $\eta \in \mathbb{R}^2$  is now a 2-dimension vector that depends on both  $\mu$  and  $\sigma$ . For the purposes of GLMs, however, the  $\sigma^2$  parameter can also be treated by considering a more general definition of the exponential family:  $p(y; \eta, \tau) = b(a, \tau) \exp((\eta^T T(y) - a(\eta))/c(\tau))$ . Here,  $\tau$  is called the **dispersion parameter**, and for the Gaussian,  $c(\tau) = \sigma^2$ ; but given our simplification above, we won't need the more general definition for the examples we will consider here.

Thus, we see that the Gaussian is in the exponential family, with

$$\begin{aligned}\eta &= \mu \\ T(y) &= y \\ a(\eta) &= \mu^2/2 \\ &= \eta^2/2 \\ b(y) &= (1/\sqrt{2\pi}) \exp(-y^2/2).\end{aligned}$$

There're many other distributions that are members of the exponential family: The multinomial (which we'll see later), the Poisson (for modelling count-data; also see the problem set); the gamma and the exponential (for modelling continuous, non-negative random variables, such as time-intervals); the beta and the Dirichlet (for distributions over probabilities); and many more. In the next section, we will describe a general “recipe” for constructing models in which  $y$  (given  $x$  and  $\theta$ ) comes from any of these distributions.

## 9 Constructing GLMs

Suppose you would like to build a model to estimate the number  $y$  of customers arriving in your store (or number of page-views on your website) in any given hour, based on certain features  $x$  such as store promotions, recent advertising, weather, day-of-week, etc. We know that the Poisson distribution usually gives a good model for numbers of visitors. Knowing this, how can we come up with a model for our problem? Fortunately, the Poisson is an exponential family distribution, so we can apply a Generalized Linear Model (GLM). In this section, we will we will describe a method for constructing GLM models for problems such as these.

More generally, consider a classification or regression problem where we would like to predict the value of some random variable  $y$  as a function of  $x$ . To derive a GLM for this problem, we will make the following three assumptions about the conditional distribution of  $y$  given  $x$  and about our model:

1.  $y | x; \theta \sim \text{ExponentialFamily}(\eta)$ . I.e., given  $x$  and  $\theta$ , the distribution of  $y$  follows some exponential family distribution, with parameter  $\eta$ .
2. Given  $x$ , our goal is to predict the expected value of  $T(y)$  given  $x$ . In most of our examples, we will have  $T(y) = y$ , so this means we would like the prediction  $h(x)$  output by our learned hypothesis  $h$  to

satisfy  $h(x) = E[y|x]$ . (Note that this assumption is satisfied in the choices for  $h_\theta(x)$  for both logistic regression and linear regression. For instance, in logistic regression, we had  $h_\theta(x) = p(y=1|x; \theta) = 0 \cdot p(y=0|x; \theta) + 1 \cdot p(y=1|x; \theta) = E[y|x; \theta].$ )

3. The natural parameter  $\eta$  and the inputs  $x$  are related linearly:  $\eta = \theta^T x$ .  
(Or, if  $\eta$  is vector-valued, then  $\eta_i = \theta_i^T x$ .)

The third of these assumptions might seem the least well justified of the above, and it might be better thought of as a “design choice” in our recipe for designing GLMs, rather than as an assumption per se. These three assumptions/design choices will allow us to derive a very elegant class of learning algorithms, namely GLMs, that have many desirable properties such as ease of learning. Furthermore, the resulting models are often very effective for modelling different types of distributions over  $y$ ; for example, we will shortly show that both logistic regression and ordinary least squares can both be derived as GLMs.

## 9.1 Ordinary Least Squares

To show that ordinary least squares is a special case of the GLM family of models, consider the setting where the target variable  $y$  (also called the **response variable** in GLM terminology) is continuous, and we model the conditional distribution of  $y$  given  $x$  as a Gaussian  $\mathcal{N}(\mu, \sigma^2)$ . (Here,  $\mu$  may depend  $x$ .) So, we let the *ExponentialFamily* $(\eta)$  distribution above be the Gaussian distribution. As we saw previously, in the formulation of the Gaussian as an exponential family distribution, we had  $\mu = \eta$ . So, we have

$$\begin{aligned} h_\theta(x) &= E[y|x; \theta] \\ &= \mu \\ &= \eta \\ &= \theta^T x. \end{aligned}$$

The first equality follows from Assumption 2, above; the second equality follows from the fact that  $y|x; \theta \sim \mathcal{N}(\mu, \sigma^2)$ , and so its expected value is given by  $\mu$ ; the third equality follows from Assumption 1 (and our earlier derivation showing that  $\mu = \eta$  in the formulation of the Gaussian as an exponential family distribution); and the last equality follows from Assumption 3.

## 9.2 Logistic Regression

We now consider logistic regression. Here we are interested in binary classification, so  $y \in \{0, 1\}$ . Given that  $y$  is binary-valued, it therefore seems natural to choose the Bernoulli family of distributions to model the conditional distribution of  $y$  given  $x$ . In our formulation of the Bernoulli distribution as an exponential family distribution, we had  $\phi = 1/(1 + e^{-\eta})$ . Furthermore, note that if  $y|x; \theta \sim \text{Bernoulli}(\phi)$ , then  $E[y|x; \theta] = \phi$ . So, following a similar derivation as the one for ordinary least squares, we get:

$$\begin{aligned} h_\theta(x) &= E[y|x; \theta] \\ &= \phi \\ &= 1/(1 + e^{-\eta}) \\ &= 1/(1 + e^{-\theta^T x}) \end{aligned}$$

So, this gives us hypothesis functions of the form  $h_\theta(x) = 1/(1 + e^{-\theta^T x})$ . If you are previously wondering how we came up with the form of the logistic function  $1/(1 + e^{-z})$ , this gives one answer: Once we assume that  $y$  conditioned on  $x$  is Bernoulli, it arises as a consequence of the definition of GLMs and exponential family distributions.

To introduce a little more terminology, the function  $g$  giving the distribution's mean as a function of the natural parameter ( $g(\eta) = E[T(y); \eta]$ ) is called the **canonical response function**. Its inverse,  $g^{-1}$ , is called the **canonical link function**. Thus, the canonical response function for the Gaussian family is just the identity function; and the canonical response function for the Bernoulli is the logistic function.<sup>7</sup>

## 9.3 Softmax Regression

Let's look at one more example of a GLM. Consider a classification problem in which the response variable  $y$  can take on any one of  $k$  values, so  $y \in \{1, 2, \dots, k\}$ . For example, rather than classifying email into the two classes spam or not-spam—which would have been a binary classification problem—we might want to classify it into three classes, such as spam, personal mail, and work-related mail. The response variable is still discrete, but can now take on more than two values. We will thus model it as distributed according to a multinomial distribution.

---

<sup>7</sup>Many texts use  $g$  to denote the link function, and  $g^{-1}$  to denote the response function; but the notation we're using here, inherited from the early machine learning literature, will be more consistent with the notation used in the rest of the class.

Let's derive a GLM for modelling this type of multinomial data. To do so, we will begin by expressing the multinomial as an exponential family distribution.

To parameterize a multinomial over  $k$  possible outcomes, one could use  $k$  parameters  $\phi_1, \dots, \phi_k$  specifying the probability of each of the outcomes. However, these parameters would be redundant, or more formally, they would not be independent (since knowing any  $k - 1$  of the  $\phi_i$ 's uniquely determines the last one, as they must satisfy  $\sum_{i=1}^k \phi_i = 1$ ). So, we will instead parameterize the multinomial with only  $k - 1$  parameters,  $\phi_1, \dots, \phi_{k-1}$ , where  $\phi_i = p(y = i; \phi)$ , and  $p(y = k; \phi) = 1 - \sum_{i=1}^{k-1} \phi_i$ . For notational convenience, we will also let  $\phi_k = 1 - \sum_{i=1}^{k-1} \phi_i$ , but we should keep in mind that this is not a parameter, and that it is fully specified by  $\phi_1, \dots, \phi_{k-1}$ .

To express the multinomial as an exponential family distribution, we will define  $T(y) \in \mathbb{R}^{k-1}$  as follows:

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, T(k-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, T(k) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

Unlike our previous examples, here we do *not* have  $T(y) = y$ ; also,  $T(y)$  is now a  $k - 1$  dimensional vector, rather than a real number. We will write  $(T(y))_i$  to denote the  $i$ -th element of the vector  $T(y)$ .

We introduce one more very useful piece of notation. An indicator function  $1\{\cdot\}$  takes on a value of 1 if its argument is true, and 0 otherwise ( $1\{\text{True}\} = 1$ ,  $1\{\text{False}\} = 0$ ). For example,  $1\{2 = 3\} = 0$ , and  $1\{3 = 5 - 2\} = 1$ . So, we can also write the relationship between  $T(y)$  and  $y$  as  $(T(y))_i = 1\{y = i\}$ . (Before you continue reading, please make sure you understand why this is true!) Further, we have that  $E[(T(y))_i] = P(y = i) = \phi_i$ .

We are now ready to show that the multinomial is a member of the

exponential family. We have:

$$\begin{aligned}
p(y; \phi) &= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \cdots \phi_k^{1\{y=k\}} \\
&= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \cdots \phi_k^{1-\sum_{i=1}^{k-1} 1\{y=i\}} \\
&= \phi_1^{(T(y))_1} \phi_2^{(T(y))_2} \cdots \phi_k^{1-\sum_{i=1}^{k-1} (T(y))_i} \\
&= \exp((T(y))_1 \log(\phi_1) + (T(y))_2 \log(\phi_2) + \\
&\quad \cdots + (T(y))_{k-1} \log(\phi_{k-1}) + \log(\phi_k)) \\
&= \exp((T(y))_1 \log(\phi_1/\phi_k) + (T(y))_2 \log(\phi_2/\phi_k) + \\
&\quad \cdots + (T(y))_{k-1} \log(\phi_{k-1}/\phi_k) + \log(\phi_k)) \\
&= b(y) \exp(\eta^T T(y) - a(\eta))
\end{aligned}$$

where

$$\begin{aligned}
\eta &= \begin{bmatrix} \log(\phi_1/\phi_k) \\ \log(\phi_2/\phi_k) \\ \vdots \\ \log(\phi_{k-1}/\phi_k) \end{bmatrix}, \\
a(\eta) &= -\log(\phi_k) \\
b(y) &= 1.
\end{aligned}$$

This completes our formulation of the multinomial as an exponential family distribution.

The link function is given (for  $i = 1, \dots, k$ ) by

$$\eta_i = \log \frac{\phi_i}{\phi_k}.$$

For convenience, we have also defined  $\eta_k = \log(\phi_k/\phi_k) = 0$ . To invert the link function and derive the response function, we therefore have that

$$\begin{aligned}
e^{\eta_i} &= \frac{\phi_i}{\phi_k} \\
\phi_k e^{\eta_i} &= \phi_i \\
\phi_k \sum_{i=1}^k e^{\eta_i} &= \sum_{i=1}^k \phi_i = 1
\end{aligned} \tag{7}$$

This implies that  $\phi_k = 1 / \sum_{i=1}^k e^{\eta_i}$ , which can be substituted back into Equation (7) to give the response function

$$\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}}$$

This function mapping from the  $\eta$ 's to the  $\phi$ 's is called the **softmax** function.

To complete our model, we use Assumption 3, given earlier, that the  $\eta_i$ 's are linearly related to the  $x$ 's. So, have  $\eta_i = \theta_i^T x$  (for  $i = 1, \dots, k-1$ ), where  $\theta_1, \dots, \theta_{k-1} \in \mathbb{R}^{n+1}$  are the parameters of our model. For notational convenience, we can also define  $\theta_k = 0$ , so that  $\eta_k = \theta_k^T x = 0$ , as given previously. Hence, our model assumes that the conditional distribution of  $y$  given  $x$  is given by

$$\begin{aligned} p(y = i|x; \theta) &= \phi_i \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}} \\ &= \frac{e^{\theta_i^T x}}{\sum_{j=1}^k e^{\theta_j^T x}} \end{aligned} \tag{8}$$

This model, which applies to classification problems where  $y \in \{1, \dots, k\}$ , is called **softmax regression**. It is a generalization of logistic regression.

Our hypothesis will output

$$\begin{aligned} h_\theta(x) &= E[T(y)|x; \theta] \\ &= E \left[ \begin{array}{c|c} 1\{y=1\} & \\ 1\{y=2\} & \\ \vdots & \\ 1\{y=k-1\} & \end{array} \middle| x; \theta \right] \\ &= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \frac{\exp(\theta_2^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_{k-1}^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \end{bmatrix}. \end{aligned}$$

In other words, our hypothesis will output the estimated probability that  $p(y = i|x; \theta)$ , for every value of  $i = 1, \dots, k$ . (Even though  $h_\theta(x)$  as defined above is only  $k-1$  dimensional, clearly  $p(y = k|x; \theta)$  can be obtained as  $1 - \sum_{i=1}^{k-1} \phi_i$ .)

Lastly, let's discuss parameter fitting. Similar to our original derivation of ordinary least squares and logistic regression, if we have a training set of  $m$  examples  $\{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$  and would like to learn the parameters  $\theta_i$  of this model, we would begin by writing down the log-likelihood

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}; \theta) \\ &= \sum_{i=1}^m \log \prod_{l=1}^k \left( \frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \right)^{1\{y^{(i)}=l\}}\end{aligned}$$

To obtain the second line above, we used the definition for  $p(y|x; \theta)$  given in Equation (8). We can now obtain the maximum likelihood estimate of the parameters by maximizing  $\ell(\theta)$  in terms of  $\theta$ , using a method such as gradient ascent or Newton's method.



# Machine Learning

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**BITS** Pilani  
Pilani Campus



**Lecture No. – 7 | Decision Tree and Random Forest  
Date – 01/12/2019**

**Time – 2:00 PM – 4:00 PM**

# Session Content

---

(Tom Mitchell Chapter 3 page 67)

- Decision Tree
- Handling overfitting
- Continuous values
- Missing Values
- Random Forest

# Decision trees

---

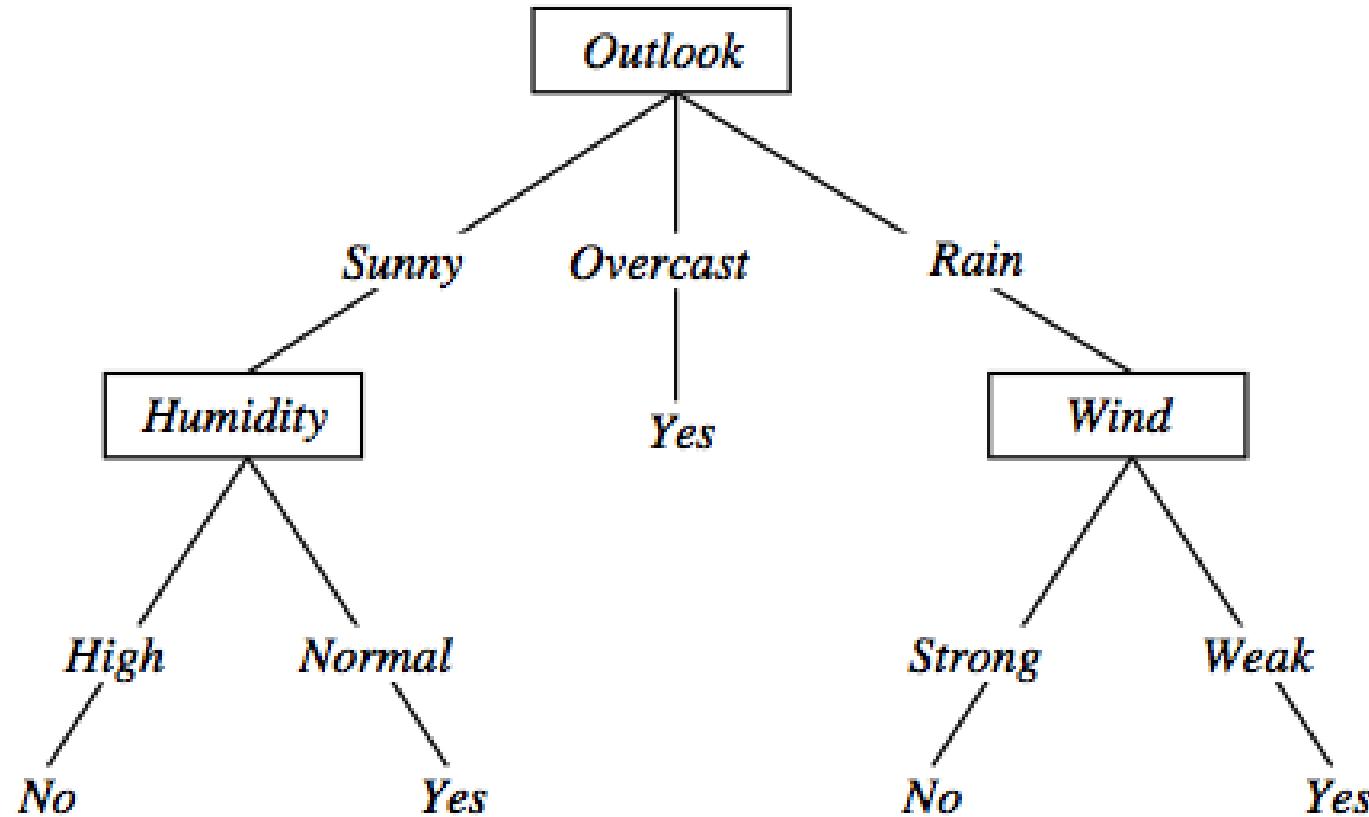
- Decision Trees is one of the most widely used and practical methods of classification
  - Method for approximating discrete-valued functions
  - Learned functions are represented as decision trees (or if-then-else rules)
  - Expressive hypotheses space
-

# Decision Tree

---

- Advantages:
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Can easily handle redundant or irrelevant attributes (unless the attributes are interacting)
- Disadvantages:
  - Space of possible decision trees is exponentially large.  
Greedy approaches are often unable to find the best tree.
  - Does not take into account interactions between attributes
  - Each decision boundary involves only a single attribute

# Decision tree representation (PlayTennis)



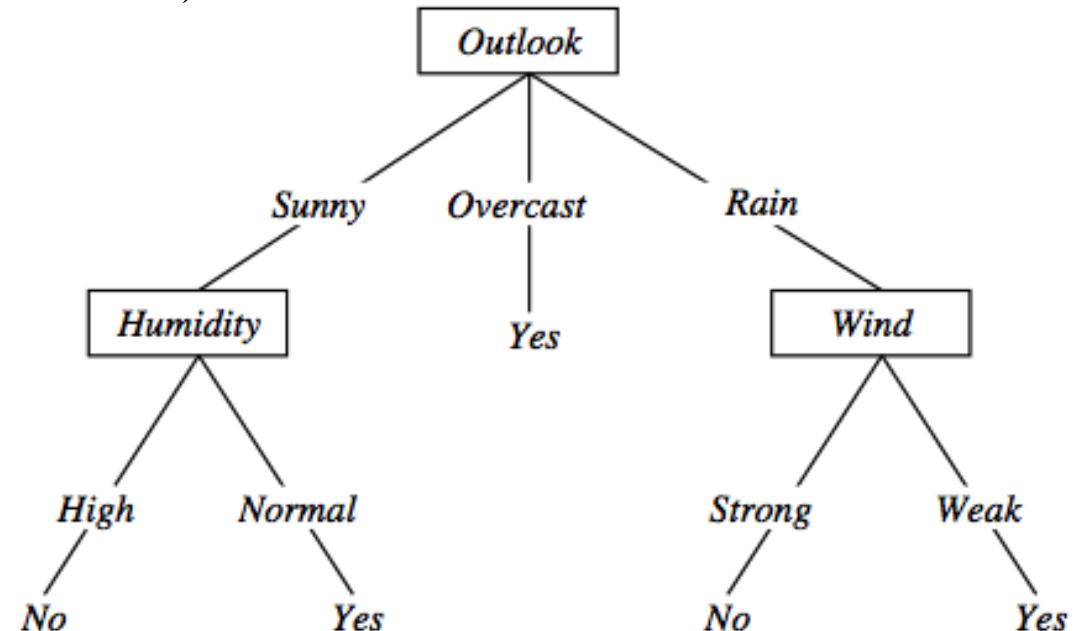
$\langle \text{Outlook}=\text{Sunny}, \text{Temp}=\text{Hot}, \text{Humidity}=\text{High}, \text{Wind}=\text{Strong} \rangle \quad \text{No}$

# Decision trees expressivity

- Decision trees represent a disjunction of conjunctions on constraints on the value of attributes:

$$(Outlook = \text{Sunny} \wedge \text{Humidity} = \text{Normal}) \vee$$

$$(Outlook = \text{Overcast}) \vee$$

$$(Outlook = \text{Rain} \wedge \text{Wind} = \text{Weak})$$


# Measure of Information

---

- The amount of information (surprise element) conveyed by a message is inversely proportional to its probability of occurrence. That is

$$I_k \propto \frac{1}{p_k}$$

- The mathematical operator satisfies above properties is the logarithmic operator.

$$I_k = \log_r \frac{1}{p_k} \text{ units}$$

# Entropy

---

- Entropy of discrete random variable  $X=\{x_1, x_2 \dots x_n\}$   
$$H(X) = E[I(X)] = E[-\log(P(X))].$$
  
; since:  $\log_2(1/P(\text{event})) = -\log_2 P(\text{event})$
- As uncertainty increases, entropy increases
- Entropy across all values

$$H(X) = - \sum_{i=1}^n P(x_i) \log_b P(x_i)$$

# Entropy in general

- Entropy measures the amount of information in a random variable

$$H(X) = -p_+ \log_2 p_+ - p_- \log_2 p_- \quad X = \{+, -\}$$

for binary classification [two-valued random variable]

$$H(X) = - \sum_{i=1}^c p_i \log_2 p_i = \sum_{i=1}^c p_i \log_2 1/p_i \quad X = \{i, \dots, c\}$$

for classification in  $c$  classes

# Entropy in binary classification

- Entropy measures the *impurity* of a collection of examples. It depends from the distribution of the random variable  $p$ .
  - $S$  is a collection of training examples
  - $p_+$  the proportion of positive examples in  $S$
  - $p_-$  the proportion of negative examples in  $S$

$$\text{Entropy}(S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_- \quad [0 \log_2 0 = 0]$$

$$\text{Entropy}([14+, 0-]) = -14/14 \log_2 (14/14) - 0 \log_2 (0) = 0$$

$$\text{Entropy}([9+, 5-]) = -9/14 \log_2 (9/14) - 5/14 \log_2 (5/14) = 0.94$$

$$\begin{aligned} \text{Entropy}([7+, 7-]) &= -7/14 \log_2 (7/14) - 7/14 \log_2 (7/14) \\ &= 1/2 + 1/2 = 1 \end{aligned} \quad [\log_2 1/2 = -1]$$

Note: the log of a number  $< 1$  is negative,  $0 \leq p \leq 1$ ,  $0 \leq \text{entropy} \leq 1$

- <https://www.easycalculation.com/log-base2-calculator.php>

# Information gain as entropy reduction

---

- *Information gain* is the *expected* reduction in entropy caused by partitioning the examples on an attribute.
- The higher the information gain the more effective the attribute in classifying training data.
- Expected reduction in entropy knowing  $A$

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$Values(A)$  possible values for  $A$

$S_v$  subset of  $S$  for which  $A$  has value  $v$

# Example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

# Example: Information gain

---

- Let
  - $Values(Wind) = \{ Weak, Strong \}$
  - $S = [9+, 5-]$
  - $S_{Weak} = [6+, 2-]$
  - $S_{Strong} = [3+, 3-]$
- Information gain due to knowing  $Wind$ :

$$\begin{aligned}Gain(S, Wind) &= Entropy(S) - 8/14 \text{ } Entropy(S_{Weak}) - 6/14 \text{ } Entropy(S_{Strong}) \\&= 0.94 - 8/14 \times 0.811 - 6/14 \times 1.00 \\&= 0.048\end{aligned}$$

---

# Example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

# Which attribute is the best classifier?

Which attribute is the best classifier?

$S: [9+, 5-]$

$E = 0.940$

*Humidity*

*High*

[3+, 4-]

$E = 0.985$

*Normal*

[6+, 1-]

$E = 0.592$

$S: [9+, 5-]$

$E = 0.940$

*Wind*

*Weak*

[6+, 2-]

$E = 0.811$

*Strong*

[3+, 3-]

$E = 1.00$

*Gain (S, Humidity )*

$$= .940 - (7/14).985 - (7/14).592$$

$$= .151$$

*Gain (S, Wind)*

$$= .940 - (8/14).811 - (6/14)1.0$$

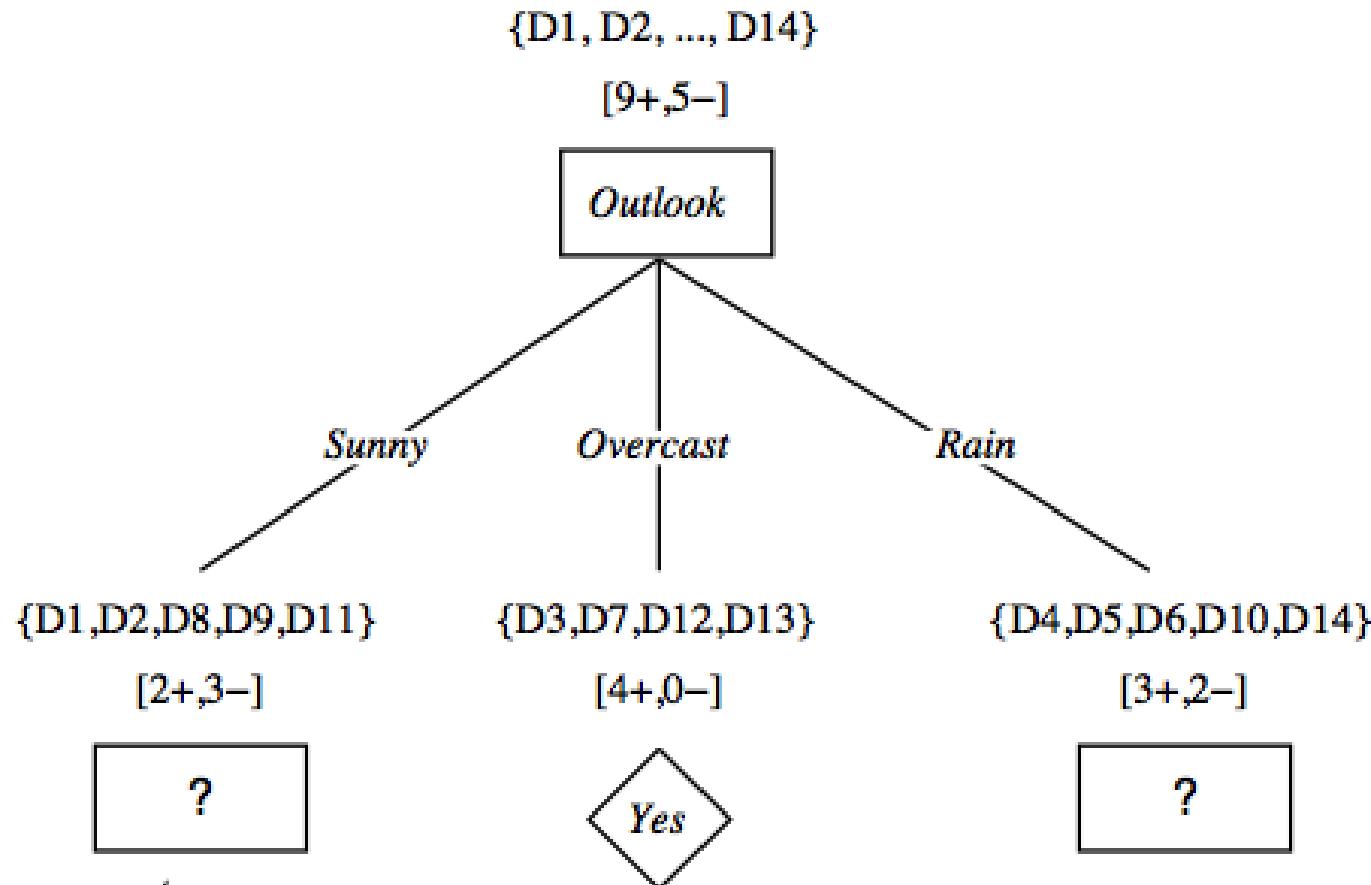
$$= .048$$

# First step: which attribute to test at the root?

---

- Which attribute should be tested at the root?
  - $Gain(S, Outlook) = 0.246$
  - $Gain(S, Humidity) = 0.151$
  - $Gain(S, Wind) = 0.084$
  - $Gain(S, Temperature) = 0.029$
- *Outlook* provides the best prediction for the target
- Lets grow the tree:
  - add to the tree a successor for each possible value of *Outlook*
  - partition the training samples according to the value of *Outlook*

# After first step



# Second step

---

- Working on *Outlook=Sunny* node:

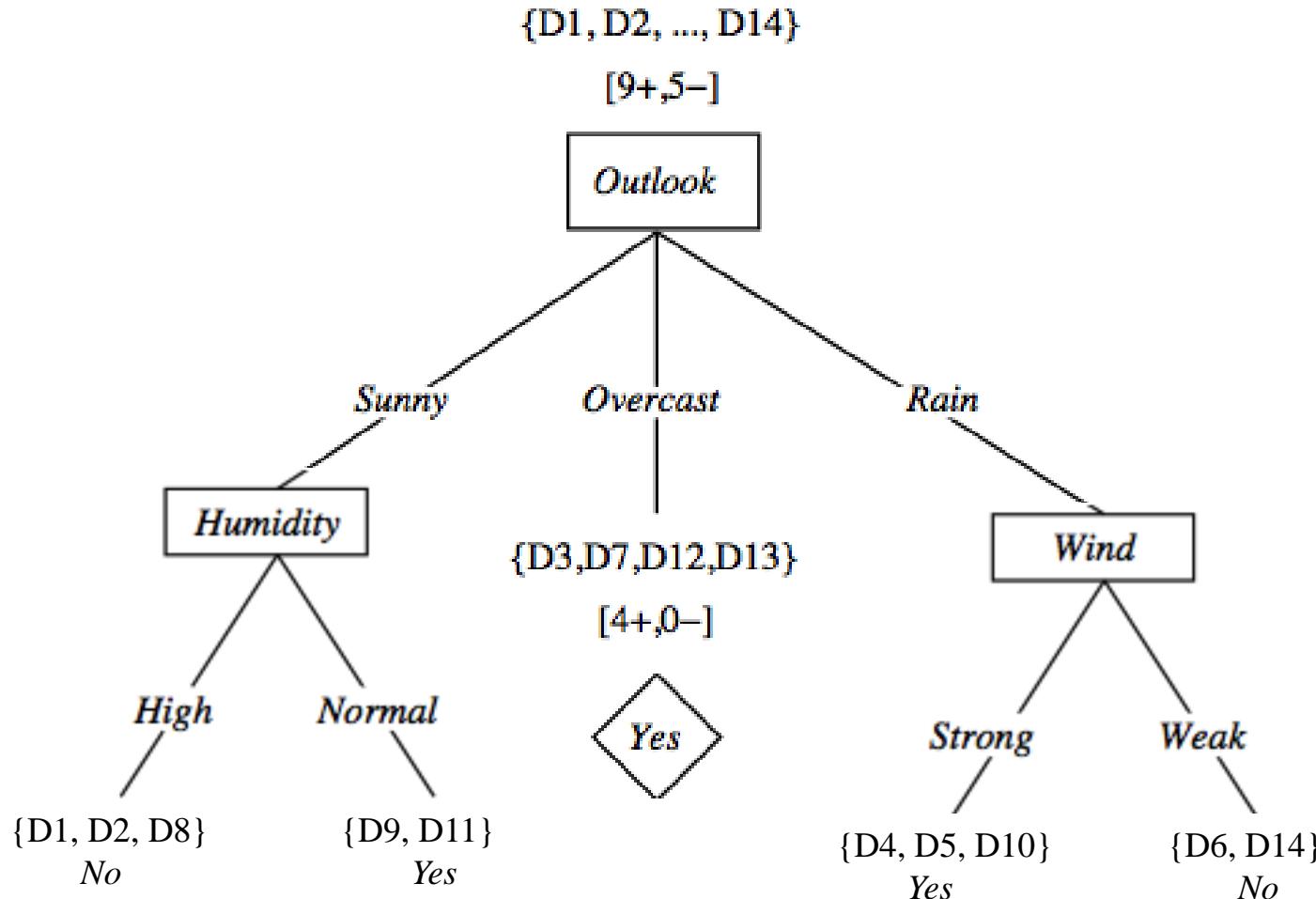
$$Gain(S_{Sunny}, \text{Humidity}) = 0.970 - 3/5 \times 0.0 - 2/5 \times 0.0 = 0.970$$

$$Gain(S_{Sunny}, \text{Wind}) = 0.970 - 2/5 \times 1.0 - 3.5 \times 0.918 = 0.019$$

$$Gain(S_{Sunny}, \text{Temp.}) = 0.970 - 2/5 \times 0.0 - 2/5 \times 1.0 - 1/5 \times 0.0 = 0.570$$

- *Humidity* provides the best prediction for the target
- Lets grow the tree:
  - add to the tree a successor for each possible value of *Humidity*
  - partition the training samples according to the value of *Humidity*

# Second and third steps



# ID3: algorithm

$\text{ID3}(X, T, \text{Attrs})$

$X$ : training examples:

$T$ : target attribute (e.g. *PlayTennis*),

$\text{Attrs}$ : other attributes, initially all attributes

Create *Root* node

*If* all  $X$ 's are +, *return* *Root* with class +

*If* all  $X$ 's are -, *return* *Root* with class -

*If*  $\text{Attrs}$  is empty *return* *Root* with class most common value of  $T$  in  $X$

*else*

$A \leftarrow$  best attribute; decision attribute for *Root*  $\leftarrow A$

*For each* possible value  $v_i$  of  $A$ :

- add a new branch below *Root*, for test  $A = v_i$

- $X_i \leftarrow$  subset of  $X$  with  $A = v_i$

- *If*  $X_i$  is empty *then* add a new leaf with class the most common value of  $T$  in  $X$

- *else* add the subtree generated by  $\text{ID3}(X_i, T, \text{Attrs} - \{A\})$

*return* *Root*

# Prefer shorter hypotheses: Occam's razor

---

- Why prefer shorter hypotheses?
- Arguments in favor:
  - There are fewer short hypotheses than long ones
  - If a short hypothesis fits data unlikely to be a coincidence
  - Elegance and aesthetics
- Arguments against:
  - Not every short hypothesis is a reasonable one.
- Occam's razor says that when presented with competing hypotheses that make the same predictions, one should select the solution which is simple"

# Issues in decision trees learning

---

- Overfitting
  - Reduced error pruning
  - Rule post-pruning
- Extensions
  - Continuous valued attributes
  - Handling training examples with missing attribute values

# Overfitting: definition

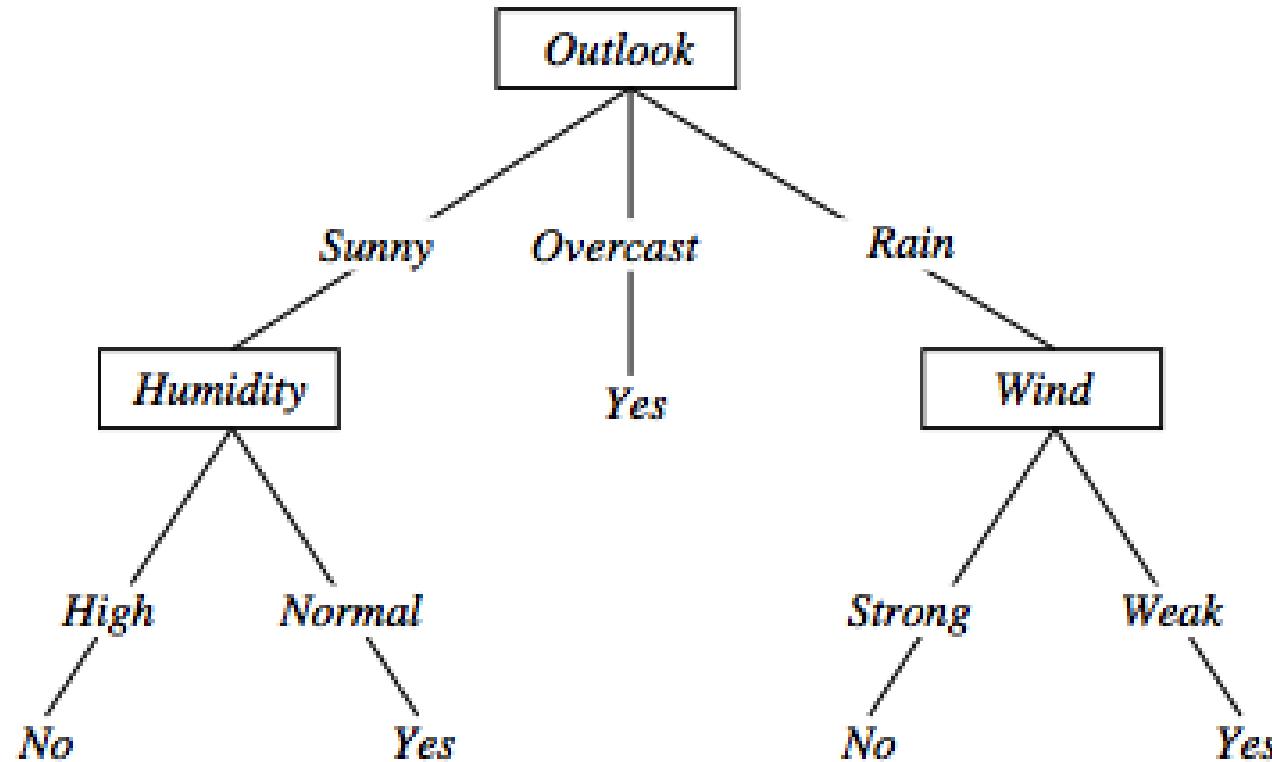
---

- **overfitting** is "the production of an analysis that corresponds too closely or exactly to a particular set of data"
- Building trees that “adapt too much” to the training examples may lead to “overfitting”.
- May therefore fail to fit additional data or predict future observations reliably
- **overfitted model** is a statistical model that contains more parameters than can be justified by the data

# Example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No
D15	Sunny	Hot	Normal	Strong	No

# Overfitting in decision trees



$\langle Outlook=Sunny, Temp=Hot, Humidity=Normal, Wind=Strong, PlayTennis=No \rangle$

New noisy example causes splitting of second leaf node.

# Avoid overfitting in Decision Trees

---

- Two strategies:
  1. Stop growing the tree earlier than perfect classification
  2. Allow the tree to *overfit* the data, and then *post-prune* the tree
- Training and validation set
  - split the training in two parts (training and validation) and use validation to assess the utility of *post-pruning*
    - *Reduced error pruning*
    - *Rule post pruning*

# Reduced-error pruning

---

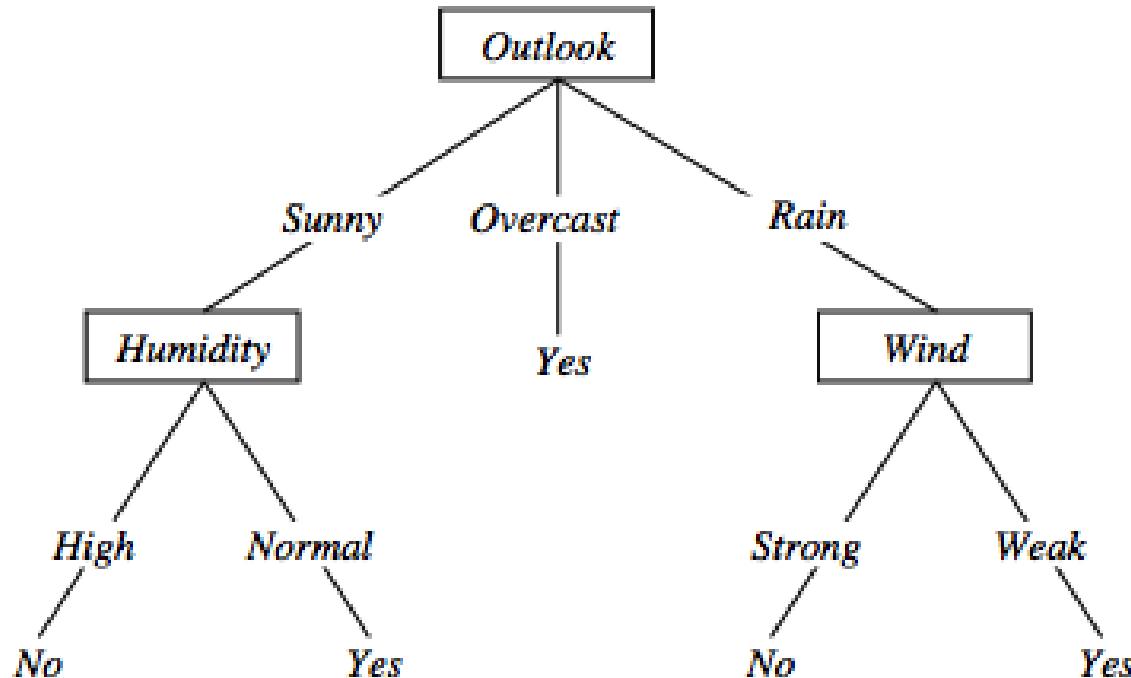
- Each node is a candidate for pruning
- *Pruning* consists in removing a subtree rooted in a node: the node becomes a leaf and is assigned the most common classification
- Nodes are removed only if the resulting tree performs no worse **on the validation set**.
- Nodes are pruned iteratively: at each iteration the node whose removal most increases accuracy on the validation set is pruned.
- Pruning stops when no pruning increases accuracy

# Rule post-pruning

---

1. Create the decision tree from the training set
  2. Convert the tree into an equivalent set of rules
    - Each path corresponds to a rule
    - Each node along a path corresponds to a pre-condition
    - Each leaf classification to the post-condition
  3. Prune (generalize) each rule by removing those preconditions whose removal improves accuracy ...
    - ... over validation set
  4. Sort the rules in estimated order of accuracy, and consider them in sequence when classifying new instances
-

# Converting to rules



$$(Outlook=Sunny) \wedge (Humidity=High) \Rightarrow (PlayTennis=No)$$

# Rule Post-Pruning

---

- Convert tree to rules (one for each path from root to a leaf)
- For each antecedent in a rule, remove it if error rate on validation set does not decrease
- Sort final rule set by accuracy

Outlook=sunny ^ humidity=high -> No  
Outlook=sunny ^ humidity=normal -> Yes  
Outlook=overcast -> Yes  
Outlook=rain ^ wind=strong -> No  
Outlook=rain ^ wind=weak -> Yes

Compare first rule to:

Outlook=sunny->No  
Humidity=high->No

Calculate accuracy of 3 rules based on validation set and pick best version.

# Why converting to rules?

---

- Each distinct path produces a different rule: a condition removal may be based on a local (contextual) criterion. Node pruning is global and affects all the rules
  - Provides flexibility of not removing entire node
  - In rule form, tests are not ordered and there is no book-keeping involved when conditions (nodes) are removed
  - Converting to rules improves readability for humans
-

# Dealing with continuous-valued attributes

- Given a continuous-valued attribute  $A$ , dynamically create a new attribute  $A_c$   
$$A_c = \text{True if } A < c, \text{ False otherwise}$$
- How to determine threshold value  $c$  ?
- Example. *Temperature* in the *PlayTennis* example
  - Sort the examples according to *Temperature*

<i>Temperature</i>	40	48		60	72	80		90
<i>PlayTennis</i>	No	No	54	Yes	Yes	Yes	85	No
  - Determine candidate thresholds by averaging consecutive values where there is a change in classification:  $(48+60)/2=54$  and  $(80+90)/2=85$

# Problems with information gain

---

- Natural bias of information gain: it favors attributes with many possible values.
- Consider the attribute *Date* in the *PlayTennis* example.
  - *Date* would have the highest information gain since it perfectly separates the training data.
  - It would be selected at the root resulting in a very broad tree
  - Very good on the training, this tree would perform poorly in predicting unknown instances. Overfitting.
- The problem is that the partition is too specific, too many small classes are generated.
- We need to look at alternative measures ...

# An alternative measure: gain ratio

$$SplitInformation(S, A) \equiv - \sum_{i=1}^c \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- $S_i$  are the sets obtained by partitioning on value  $i$  of  $A$
- $SplitInformation$  measures the entropy of  $S$  with respect to the values of  $A$ . The more uniformly dispersed the data the higher it is.

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

- $GainRatio$  penalizes attributes that split examples in many small classes such as *Date*. Let  $|S|=n$ , *Date* splits examples in  $n$  classes
  - $SplitInformation(S, Date) = -[(1/n \log_2 1/n) + \dots + (1/n \log_2 1/n)] = -\log_2 1/n = \log_2 n$
- Compare with  $A$ , which splits data in two even classes:
  - $SplitInformation(S, A) = -[(1/2 \log_2 1/2) + (1/2 \log_2 1/2)] = -[-1/2 - 1/2] = 1$

# Handling missing values training data

---



- How to cope with the problem that the value of some attribute may be missing?
  - The strategy: use other examples to guess attribute
    1. Assign the value that is most common among the training examples at the node
    2. Assign a probability to each value, based on frequencies, and assign values to missing attribute, according to this probability distribution
-

# Applications

---

Suited for following classification problems:

- Applications whose Instances are represented by attribute-value pairs.
- The target function has discrete output values
- Disjunctive descriptions may be required
- The training data may contain missing attribute values

Real world applications

- Biomedical applications
- Manufacturing
- Banking sector
- Make-Buy decisions

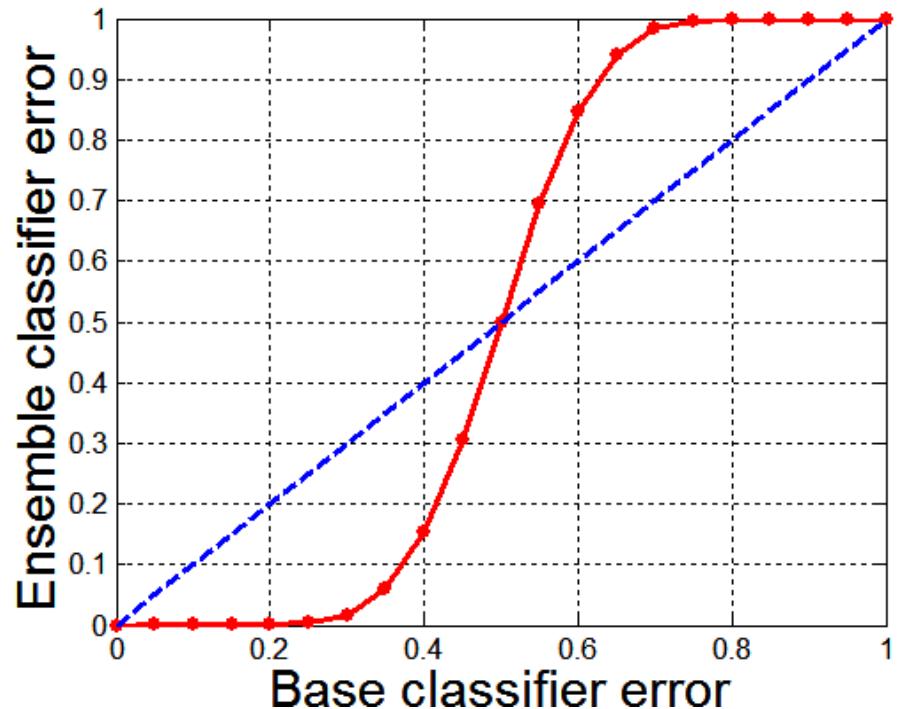
# Ensemble Methods

---

- **Ensemble methods** use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone
- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers
- Tend to reduce problems related to over-fitting of the training data.

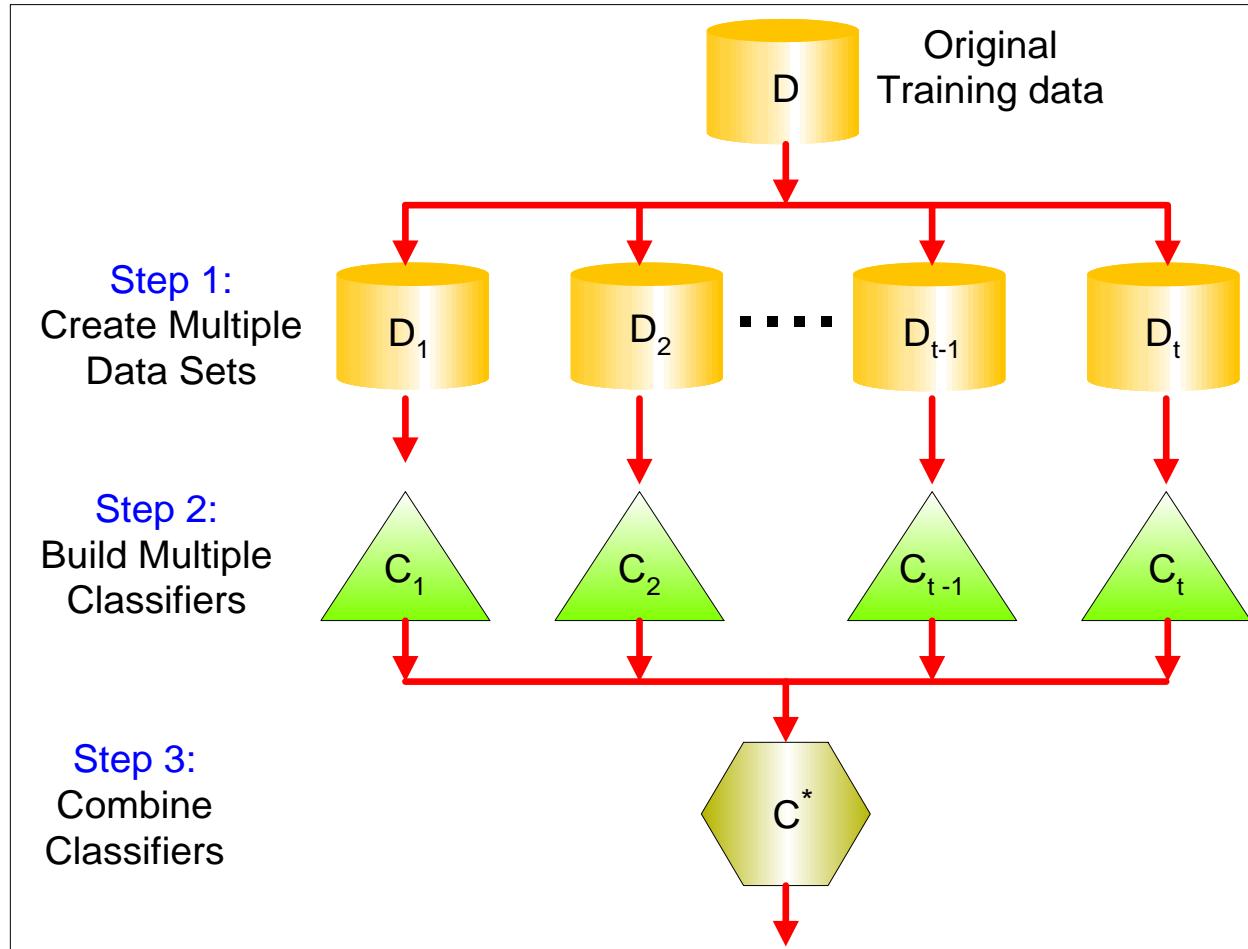
# Why Ensemble Methods work?

- 25 base classifiers
- Each classifier has error rate,  $\varepsilon = 0.35$
- If base classifiers are identical, then the ensemble will misclassify the same examples predicted incorrectly by the base classifiers depicted by dotted line
- Assume errors made by classifiers are uncorrelated
- ensemble makes a wrong prediction only if more than half of the base classifiers predict incorrectly
- Probability that the ensemble classifier makes a wrong prediction:



$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

# General Approach



# Simple Ensemble Techniques

---

## Max Voting

Ex: Movie rating

The result of max voting would be something like this:

- Rating by 5 friends: 5 4 5 4 4

**Averaging-**  $(5+4+5+4+4)/5 = 4.4$  Final rating

## Weighted Average

Weight-0.23 0.23 0.18 0.18 0.18

The result is calculated as  $[(5*0.23) + (4*0.23) + (5*0.18) + (4*0.18) + (4*0.18)] = 4.41.$

# Types of Ensemble Methods

## Manipulate data distribution

- Example: bagging

## Manipulate input features

- Example: random forests

# When does Ensemble work?

---

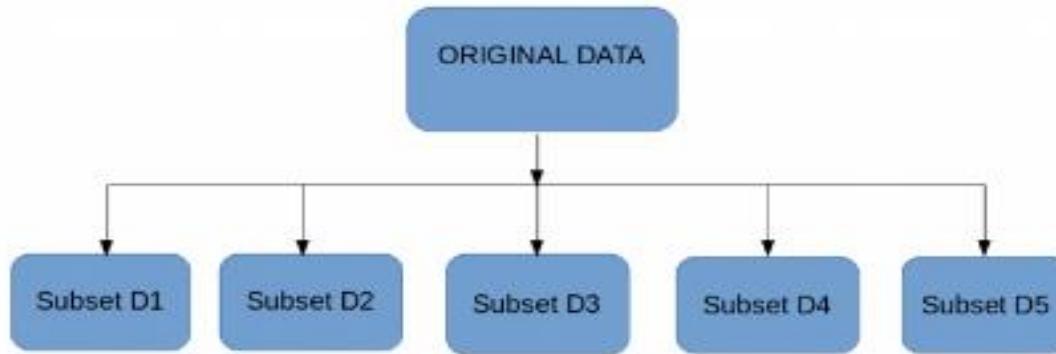
- Ensemble classifier performs better than the base classifiers when error is smaller than 0.5
- Necessary conditions for an ensemble classifier to perform better than a single classifier:
  - Base classifiers should be independent of each other
  - Base classifiers should do better than a classifier that performs random guessing

# Bagging (Bootstrap Aggregating)

---

- Technique uses these subsets (bags) to get a fair idea of the distribution (complete set).
- The size of subsets created for bagging may be less than the original set.
- Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, **with replacement**.
- When you sample with replacement, items are independent. One item does not affect the outcome of the other. You have  $1/7$  chance of choosing the first item and a  $1/7$  chance of choosing the second item.
- If the two items are **dependent**, or linked to each other. When you choose the first item, you have a  $1/7$  probability of picking a item. Assuming you don't replace the item, you only have six items to pick from. That gives you a  $1/6$  chance of choosing a second item.

# Bagging



- Multiple subsets are created from the original dataset, selecting observations with replacement.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.

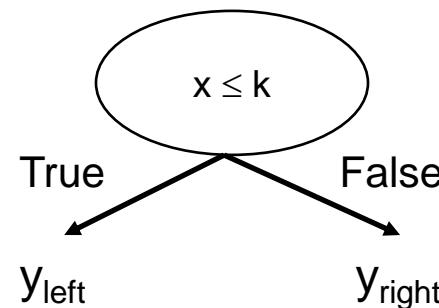
# Bagging Example

- Consider 1-dimensional data set:

**Original Data:**

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule:  $x \leq k$  versus  $x > k$
  - Split point  $k$  is chosen based on entropy



# Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

$x \leq 0.35 \rightarrow y = 1$   
 $x > 0.35 \rightarrow y = -1$

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

$x \leq 0.7 \rightarrow y = 1$   
 $x > 0.7 \rightarrow y = 1$

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.35 \rightarrow y = 1$   
 $x > 0.35 \rightarrow y = -1$

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.3 \rightarrow y = 1$   
 $x > 0.3 \rightarrow y = -1$

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

$x \leq 0.35 \rightarrow y = 1$   
 $x > 0.35 \rightarrow y = -1$

# Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \rightarrow y = -1$   
 $x > 0.75 \rightarrow y = 1$

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

$x \leq 0.75 \rightarrow y = -1$   
 $x > 0.75 \rightarrow y = 1$

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \rightarrow y = -1$   
 $x > 0.75 \rightarrow y = 1$

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \rightarrow y = -1$   
 $x > 0.75 \rightarrow y = 1$

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

$x \leq 0.05 \rightarrow y = 1$   
 $x > 0.05 \rightarrow y = 1$

# Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

# Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

# Bagging Algorithm

---

## Algorithm 5.6 Bagging Algorithm

---

- 1: Let  $k$  be the number of bootstrap samples.
- 2: for  $i = 1$  to  $k$  do
- 3:   Create a bootstrap sample of size  $n$ ,  $D_i$ .
- 4:   Train a base classifier  $C_i$  on the bootstrap sample  $D_i$ .
- 5: end for
- 6:  $C^*(x) = \arg \max_y \sum_i \delta(C_i(x) = y)$ ,  $\{\delta(\cdot) = 1 \text{ if its argument is true, and } 0 \text{ otherwise.}\}$

---

# Random Forest

---

- Random Forest is another ensemble machine learning algorithm that follows the bagging technique.
  - The base estimators in random forest are decision trees.
  - Random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.
-

# Random Forest

---

- As in bagging, we build a number of decision trees on bootstrapped training samples each time a split in a tree is considered, a random sample of  $m$  predictors is chosen as split candidates from the full set of  $p$  predictors.
- Note that if  $m = p$ , then this is bagging.

# Random Forest

---

- Random subsets are created from the original dataset (bootstrapping).
- At each node in the decision tree, only a random set of features are considered to decide the best split.
- A decision tree model is fitted on each of the subsets.
- The final prediction is calculated by averaging the predictions from all decision trees.

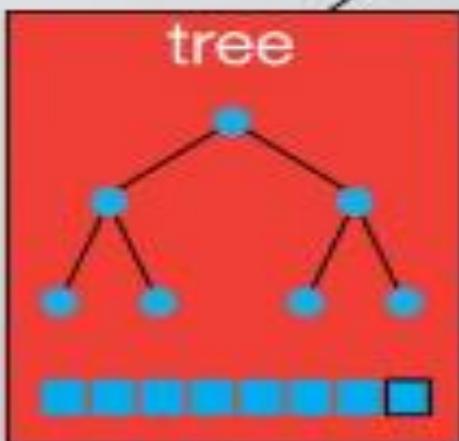
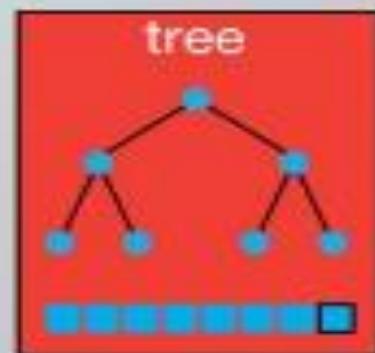
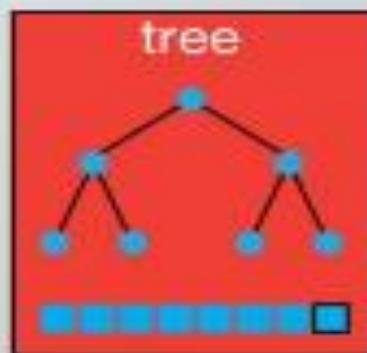
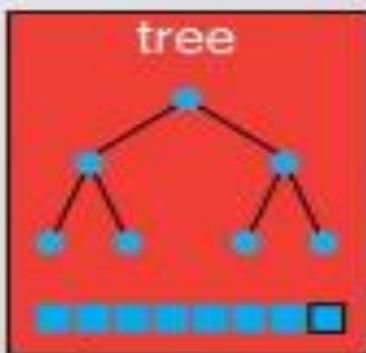
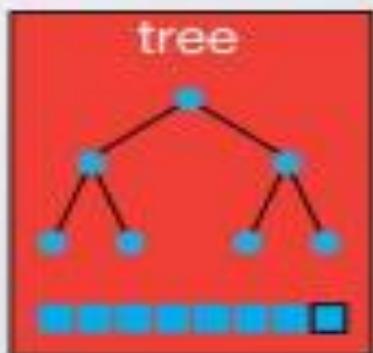
# All Data

random subset

random subset

random subset

random subset



At each node:  
choose some ballsubset of variables at random  
find a variable ( and a value for that variable) which optimizes the split

# Random Forests Algorithm

---

- For  $b = 1$  to  $B$ :
  - (a) Draw a bootstrap sample  $Z^*$  of size  $N$  from the training data.
  - (b) Grow a random-forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size  $n_{\min}$  is reached.
    - i. Select  $m$  variables at random from the  $p$  variables.
    - ii. Pick the best variable/split-point among the  $m$ .
    - iii. Split the node into two daughter nodes. Output the ensemble of trees.
- To make a prediction at a new point  $x$  we do:
  - For regression: average the results
  - For classification: majority vote

# Random Forests Tuning

---

- The inventors make the following recommendations:
  - For classification, the default value for  $m$  is  $\sqrt{p}$  and the minimum node size is one.
  - For regression, the default value for  $m$  is  $p/3$  and the minimum node size is five.
- In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

# Example

---

- 4,718 genes measured on tissue samples from 349 patients.
- Each gene has different expression
- Each of the patient samples has a qualitative label with 15 different levels: either normal or 1 of 14 different types of cancer.
- Use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.

# Random Forests Issues

---

- When the number of variables is large, but the fraction of relevant variables is small, random forests are likely to perform poorly when  $m$  is small

Why?

- Because: At each split the chance can be small that the relevant variables will be selected
- For example, with 3 relevant and 100 not so relevant variables the probability of any of the relevant variables being selected at any split is  $\sim 0.25$

# Advantages of Random Forest

---

- Algorithm can solve both type of problems i.e. classification and regression
- Power of handle large data set with higher dimensionality.
- It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.
- Model outputs **Importance of variable**, which can be a very handy feature (on some random data set).

# Disadvantages of Random Forest

---

- May over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modelers – you have very little control on what the model does. You can at best – try different parameters and random seeds!

# Good References

## Decision Tree

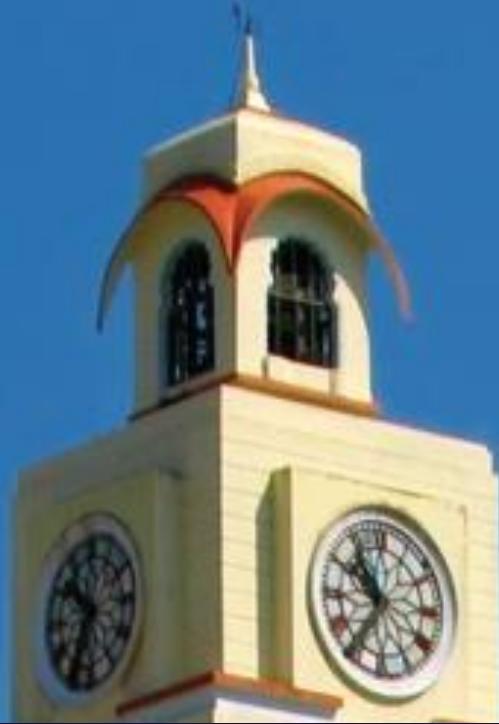
- [https://www.youtube.com/watch?v=eKD5gxPPeY0&list=PLBv09BD7ez\\_4temBw7vLA19p3tdQH6FYO&index=1](https://www.youtube.com/watch?v=eKD5gxPPeY0&list=PLBv09BD7ez_4temBw7vLA19p3tdQH6FYO&index=1)

## Overfitting

- [https://www.youtube.com/watch?time\\_continue=1&v=t56Nid85Thg](https://www.youtube.com/watch?time_continue=1&v=t56Nid85Thg)
- <https://www.youtube.com/watch?v=y6SpA2Wuyt8>

## Random Forest

- <https://www.stat.berkeley.edu/~breiman/RandomForests/>



# Machine Learning

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**BITS** Pilani

Pilani Campus



## **Lecture No. – 9 | Neural Network**

**Date – 11/01/2020**

**Time – 9:00 AM – 11:00 AM**

These slides are prepared by the instructor, with grateful acknowledgement of Tom Mitchell, Andrew Ng and many others who made their course materials freely available online.

# Session Content

- Perceptron (Chapter 4 Tom Mitchell)
- Neural Network Architecture (Andrew Ng Notes and Chapter 4 Tom Mitchell)
- Back propagation Algorithm (Andrew Ng Notes)

## When to Consider Neural Networks

---

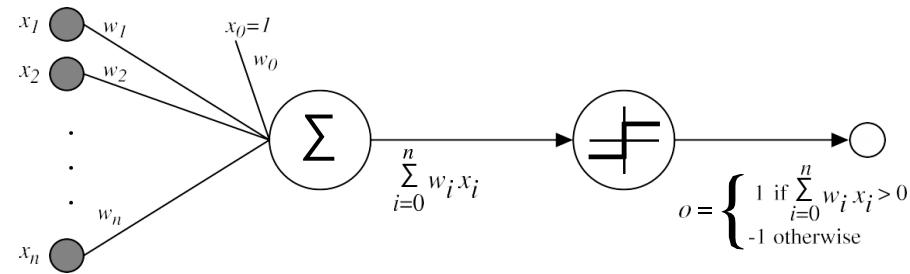
- Input is high-dimensional discrete or real-valued  
(e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

Examples:

- Speech phoneme recognition [Waibel]
- Image classification [Kanade, Baluja, Rowley]
- Financial prediction

# Perceptron

---



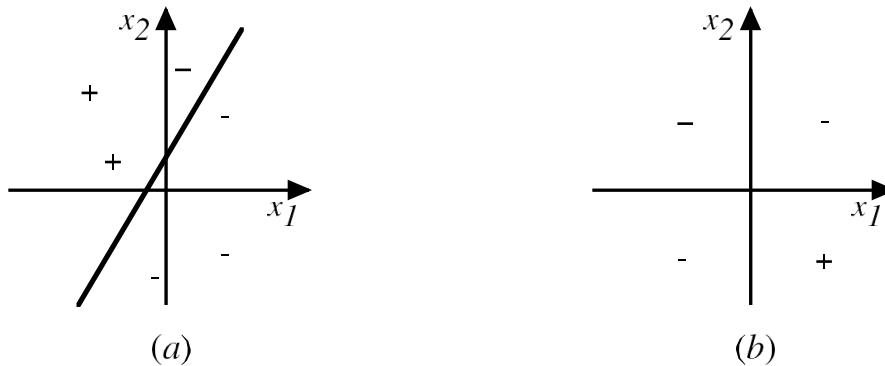
$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

# Decision Surface of Perceptron

---



Represents some useful functions

- What weights represent  
 $g(x_1, x_2) = AND(x_1, x_2)$ ?

But some functions not representable

- e.g., not linearly separable
- Therefore, we'll want networks of these...

# Perceptron Training rule

---

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$  is target value
- $o$  is perceptron output
- $\eta$  is small constant (e.g., .1) called *learning rate*

# Gradient Descent

---

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

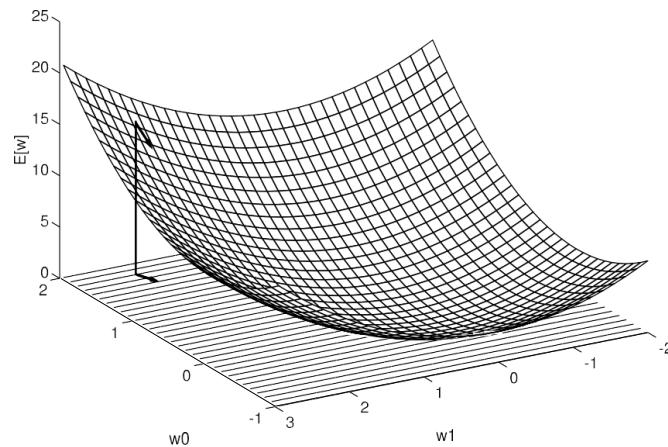
Let's learn  $w_i$ 's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where  $D$  is set of training examples

# Gradient Descent

---



Gradient

$$\nabla E[\vec{w}] \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

# Gradient Descent

---

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ \frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d) (-x_{i,d})\end{aligned}$$

# Gradient Descent

---

GRADIENT-DESCENT(*training\_examples*,  $\eta$ )

*Each training example is a pair of the form  $\langle \vec{x}, t \rangle$ , where  $\vec{x}$  is the vector of input values, and  $t$  is the target output value.  $\eta$  is the learning rate (e.g., .05).*

- Initialize each  $w_i$  to some small random value
- Until the termination condition is met, Do
  - Initialize each  $\Delta w_i$  to zero.
  - For each  $\langle \vec{x}, t \rangle$  in *training\_examples*, Do
    - \* Input the instance  $\vec{x}$  to the unit and compute the output  $o$
    - \* For each linear unit weight  $w_i$ , Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

- For each linear unit weight  $w_i$ , Do

$$w_i \leftarrow w_i + \Delta w_i$$

# Perceptron Training

---

Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate  $\eta$

Linear unit training rule uses gradient descent

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate  $\eta$
- Even when training data contains noise
- Even when training data not separable by  $H$

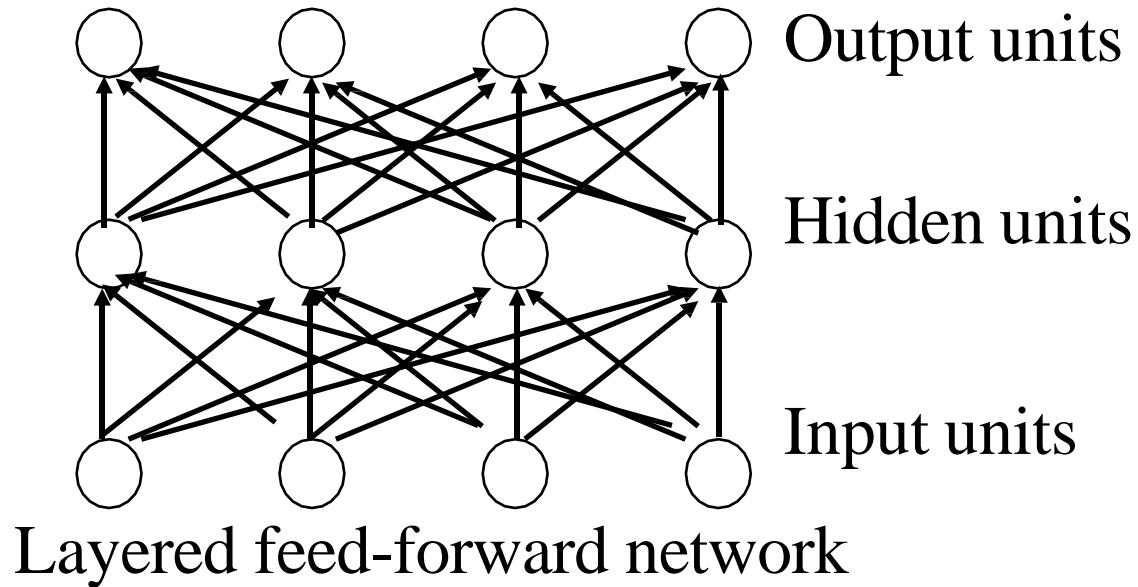
# Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

# Multilayer network

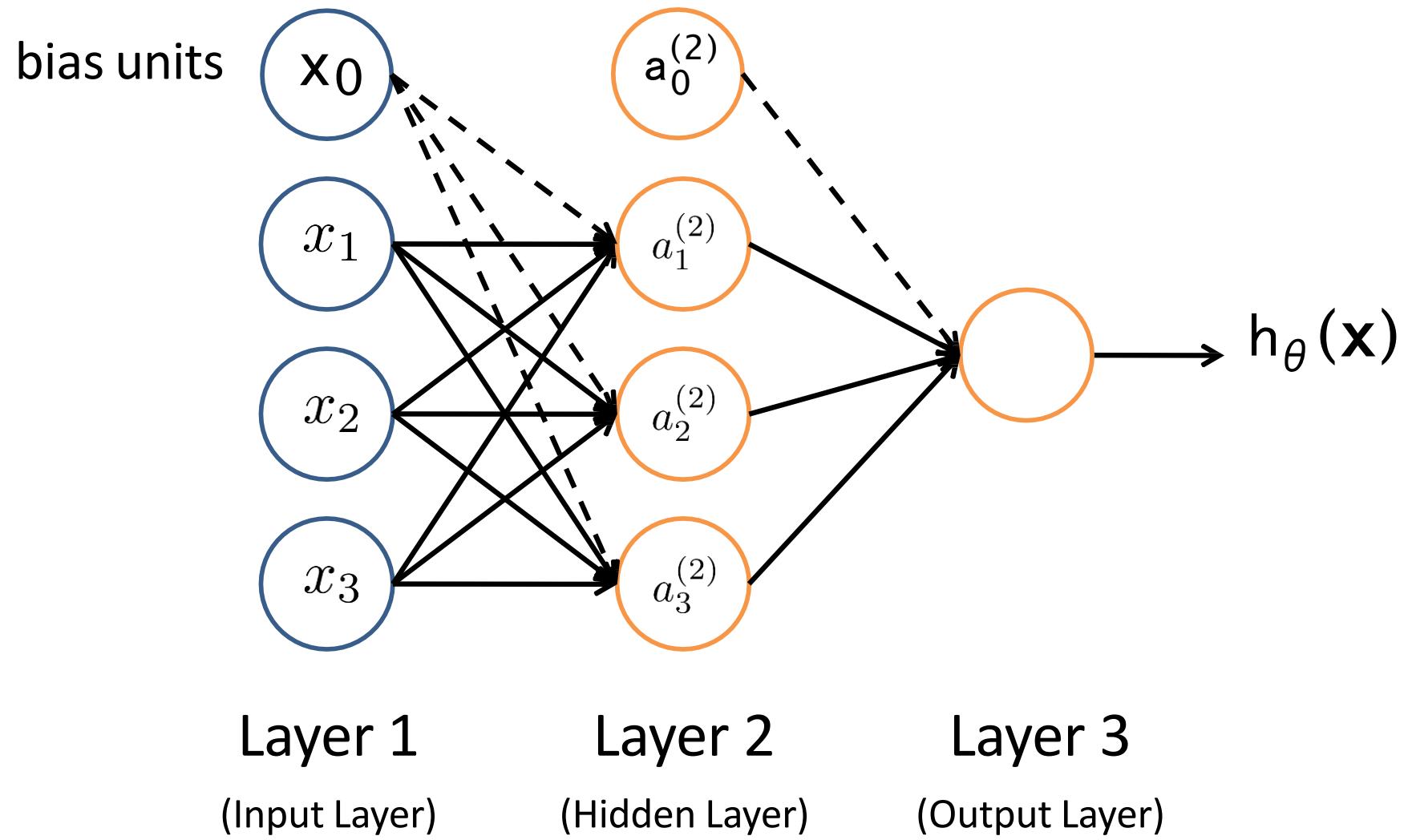
- Single perceptrons can only express linear decision surfaces.
- In contrast, the kind of multilayer networks learned by the BACKPROPAGATION algorithm are capable of expressing a rich variety of nonlinear decision surfaces

# Neural networks



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

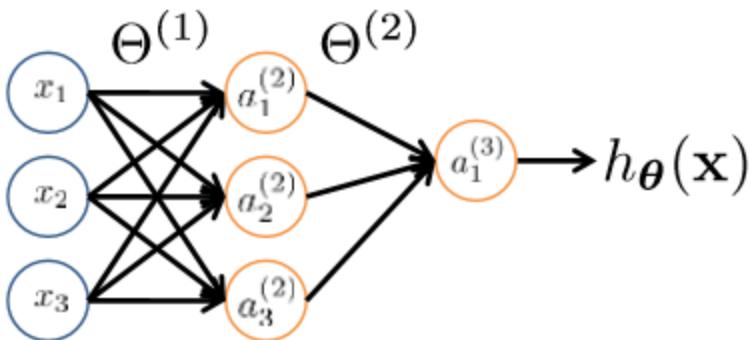
# Neural Network



# Feed-Forward Process

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

# Neural Network



$a_i^{(j)}$  = “activation” of unit  $i$  in layer  $j$   
 $\Theta^{(j)}$  = weight matrix controlling function mapping from layer  $j$  to layer  $j + 1$

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

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

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

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

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

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

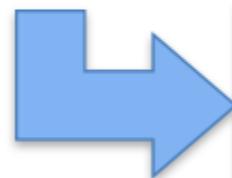
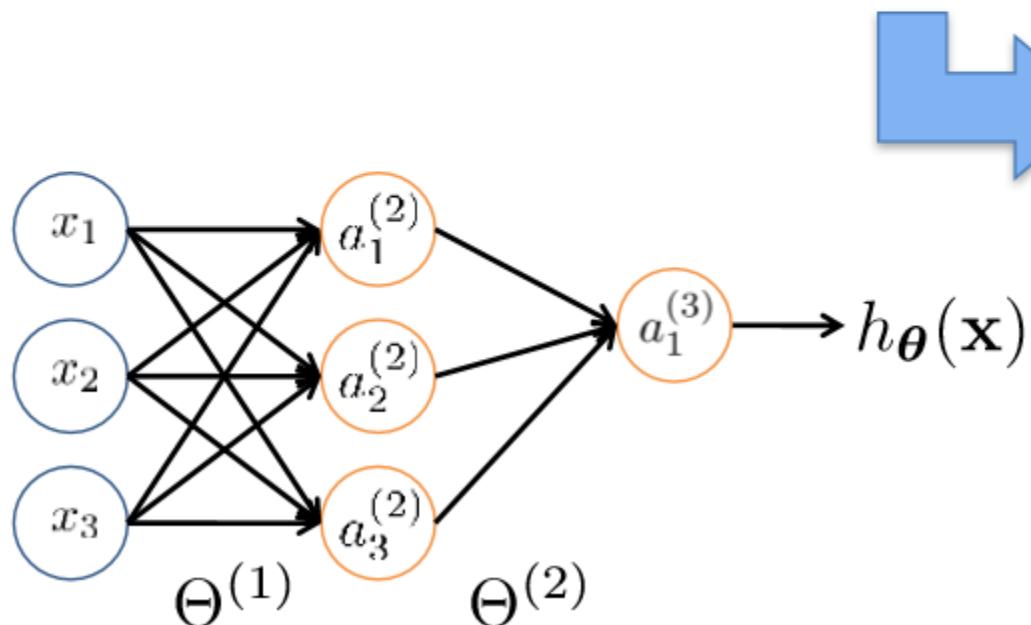
# Vectorization

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

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

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

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



Feed-Forward Steps:

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

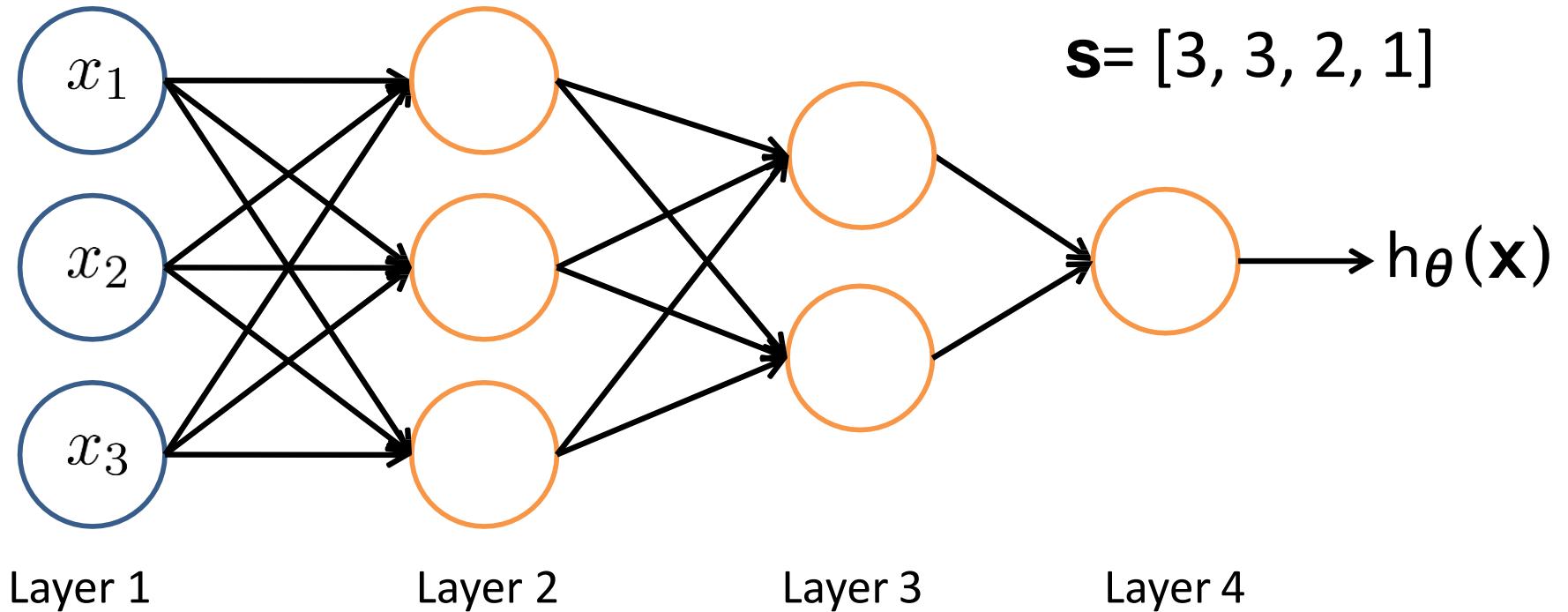
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

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

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

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

# Other Network Architectures



$L$  denotes the number of layers

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

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

# Multiple Output Units: One-vs-Rest



Pedestrian



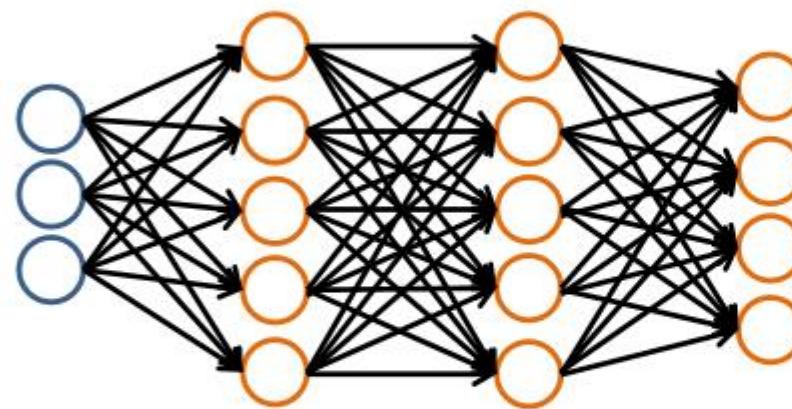
Car



Motorcycle



Truck



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

We want:

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{when pedestrian}$$

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{when car}$$

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{when motorcycle}$$

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{when truck}$$

# Multiple Output Units: One-vs-Rest



We want:

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{when pedestrian}$$

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{when car}$$

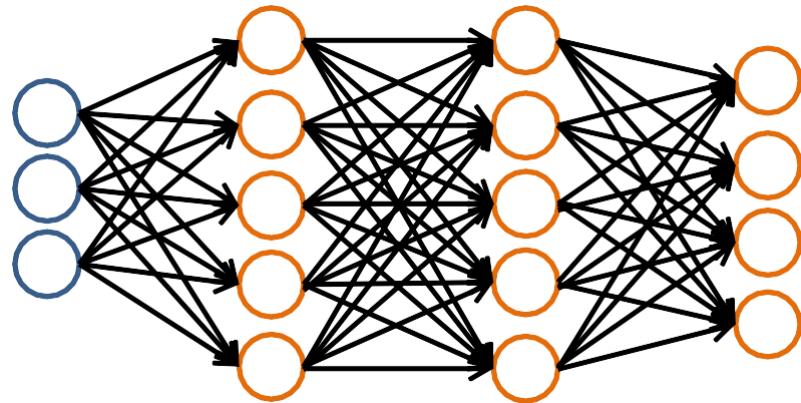
$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{when motorcycle}$$

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{when truck}$$

- Given  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$
- Must convert labels to 1-of- $K$  representation

– e.g.,  $y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$  when motorcycle,  $y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$  when car, etc.

# Neural Network Classification



**Given:**

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

$\mathbf{S} \in \mathbb{N}^{+L}$  contains # nodes at each layer  
—  $s_0 = d$  (#features)

Binary classification

$$y = 0 \text{ or } 1$$

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

Multi-class classification ( $K$  classes)

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

pedestrian    car    motorcycle    truck

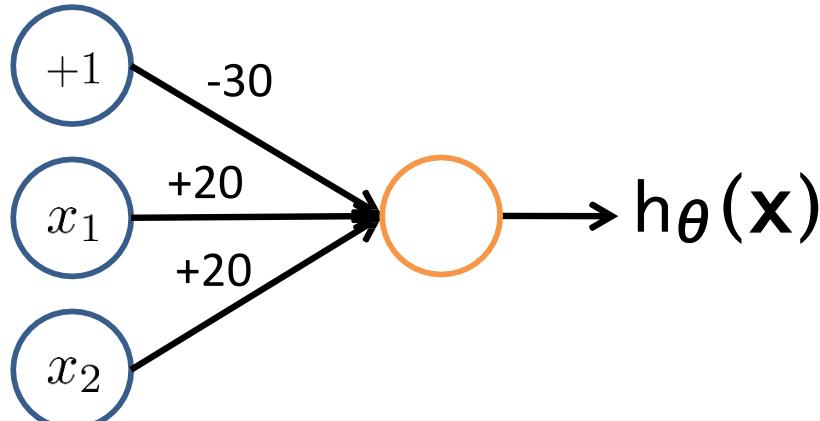
$K$  output units ( $s_{L-1} = K$ )

# Representing Boolean Functions

## Simple example: AND

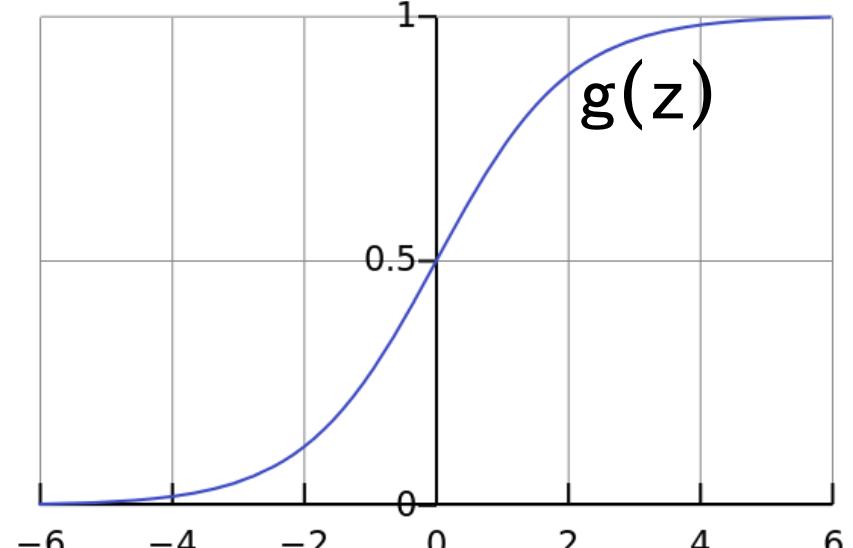
$$x_1, x_2 \in \{0, 1\}$$

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



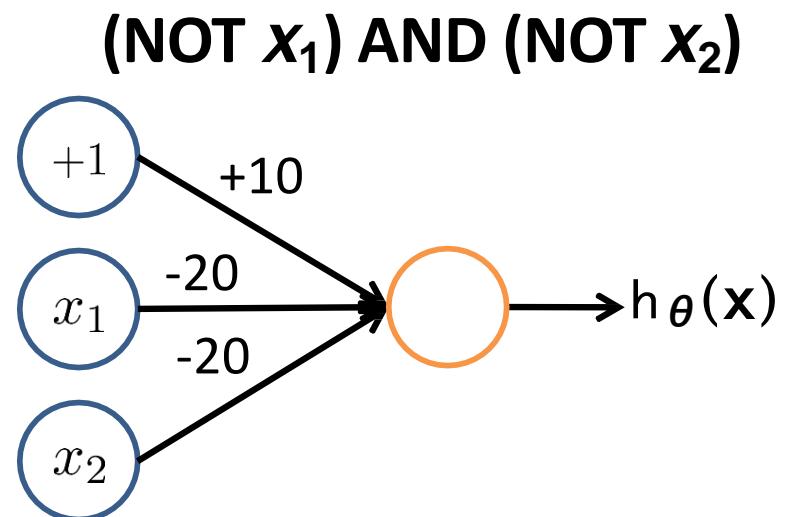
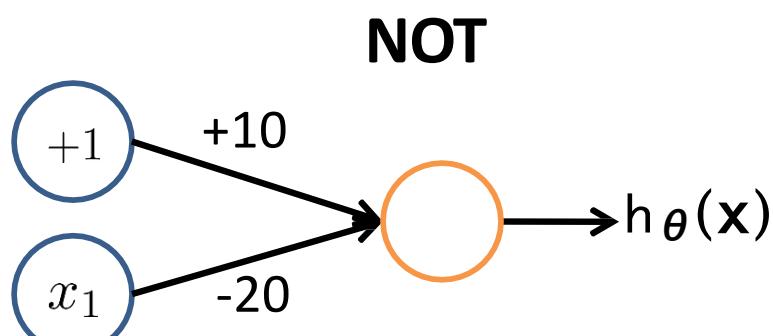
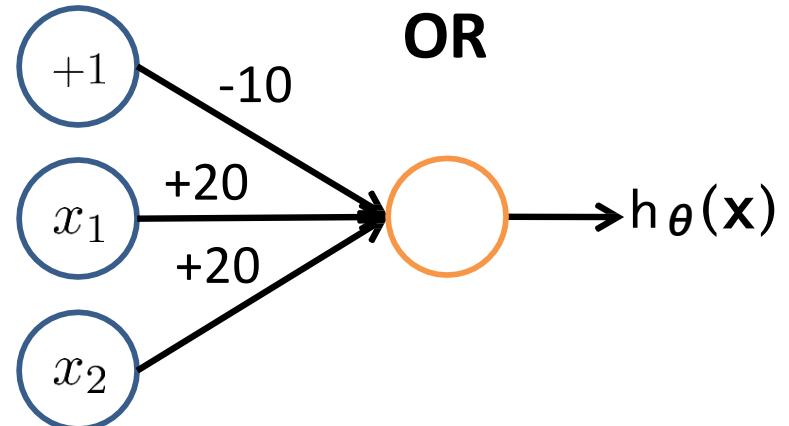
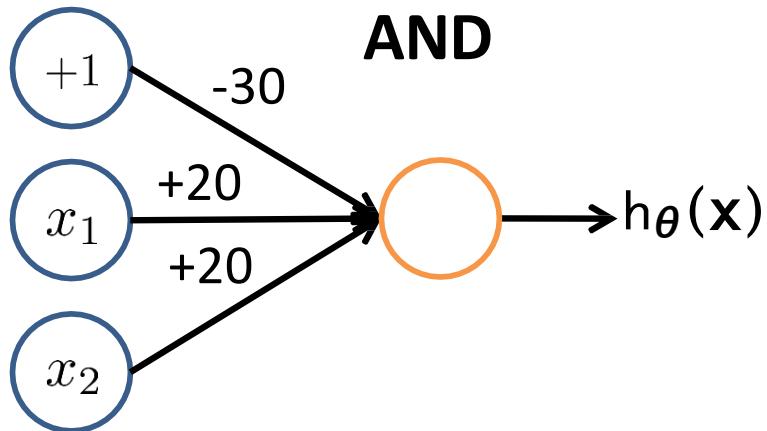
$$h_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

Logistic / Sigmoid Function

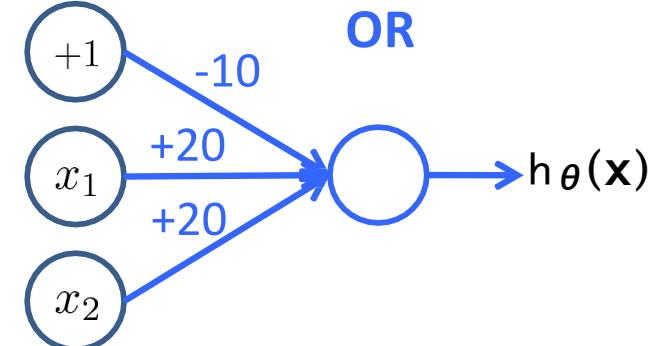
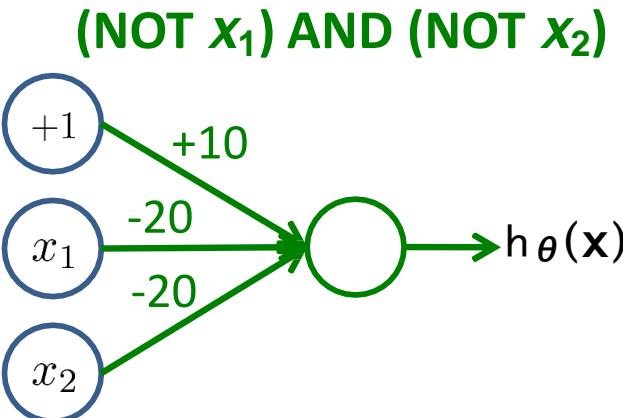
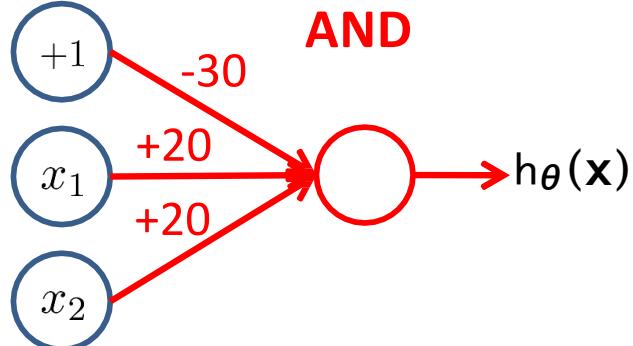


$x_1$	$x_2$	$h_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

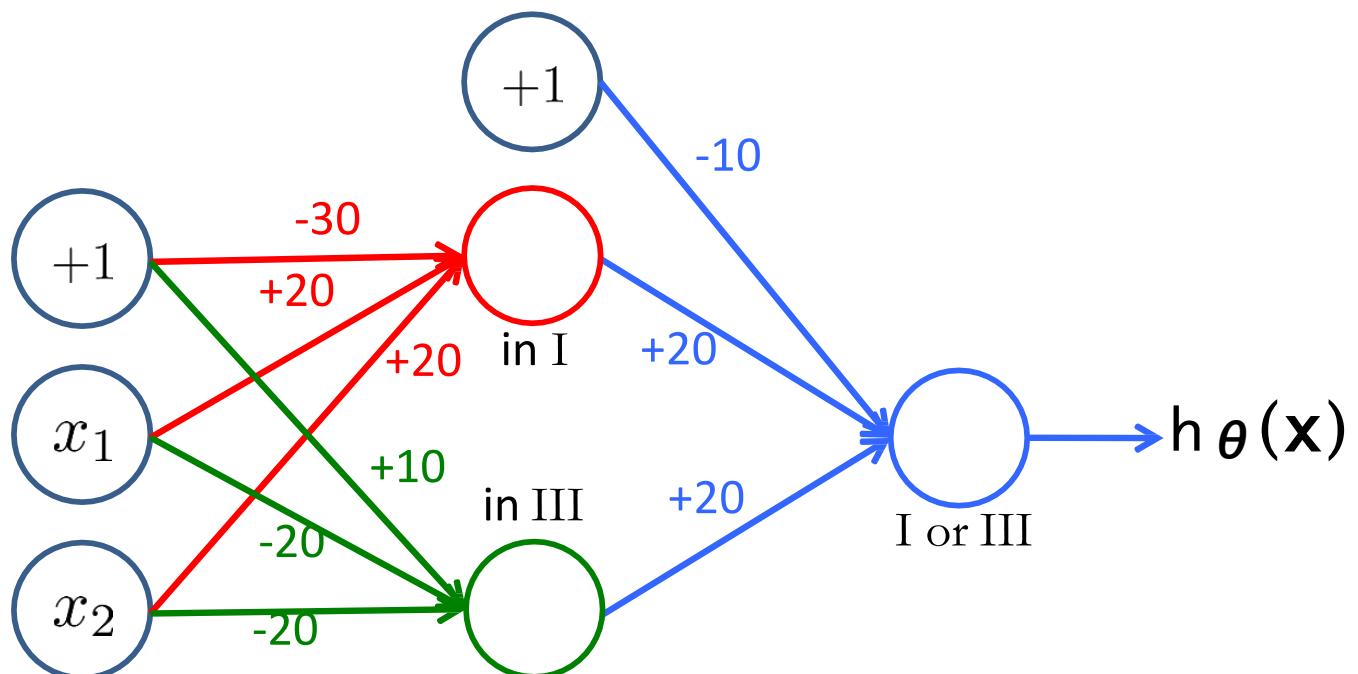
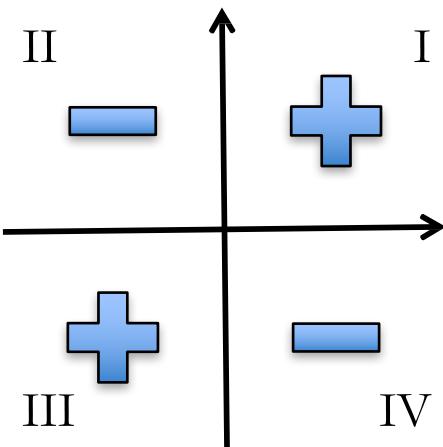
# Representing Boolean Functions



# Combining Representations to Create Non-Linear Functions



not(XOR)

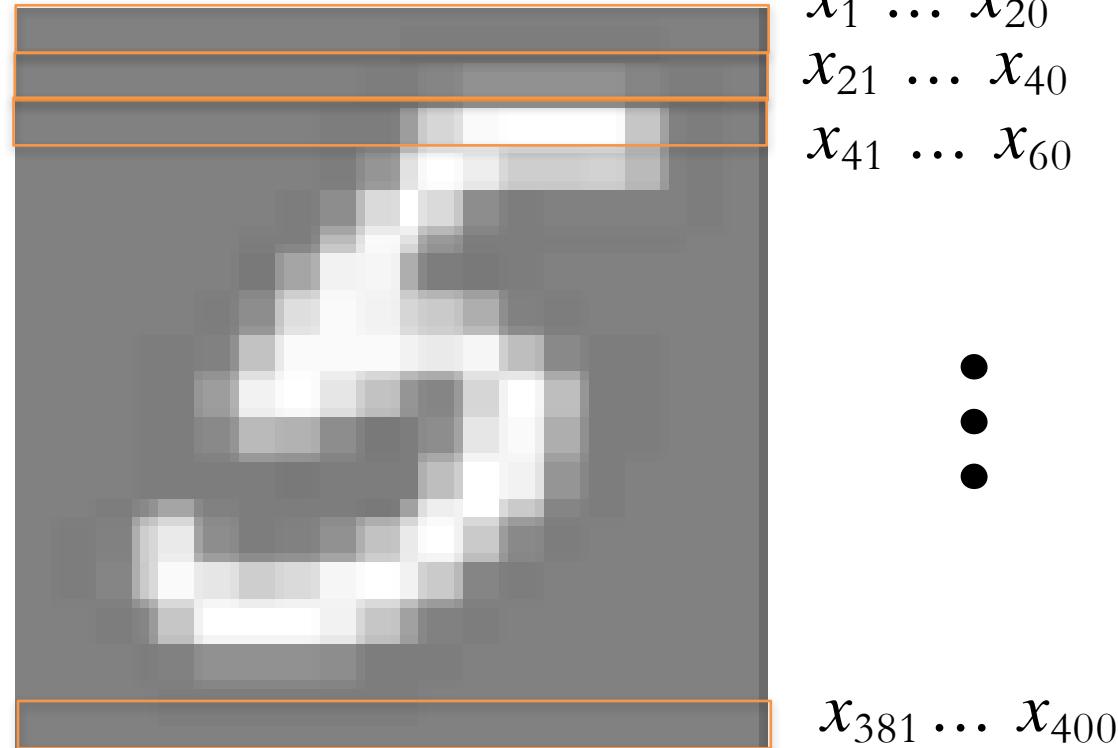


# Layering Representations



20 × 20 pixel images

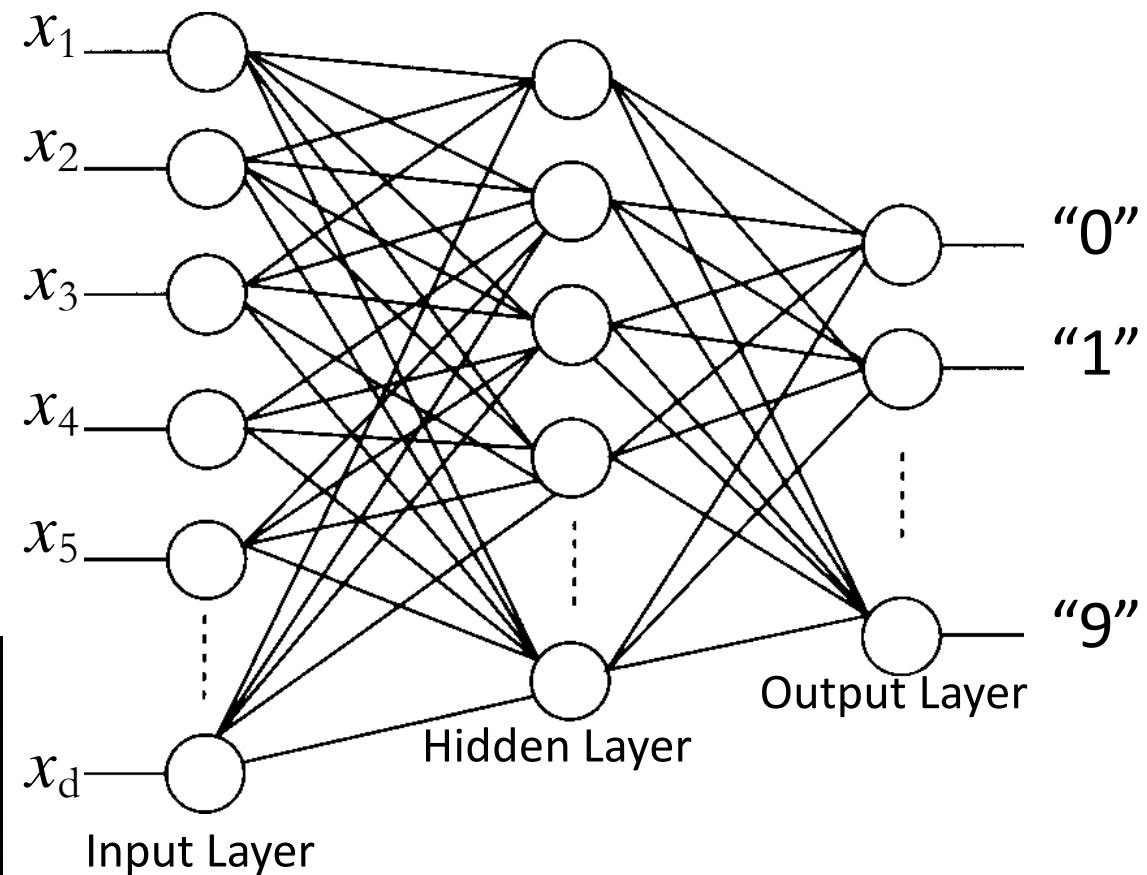
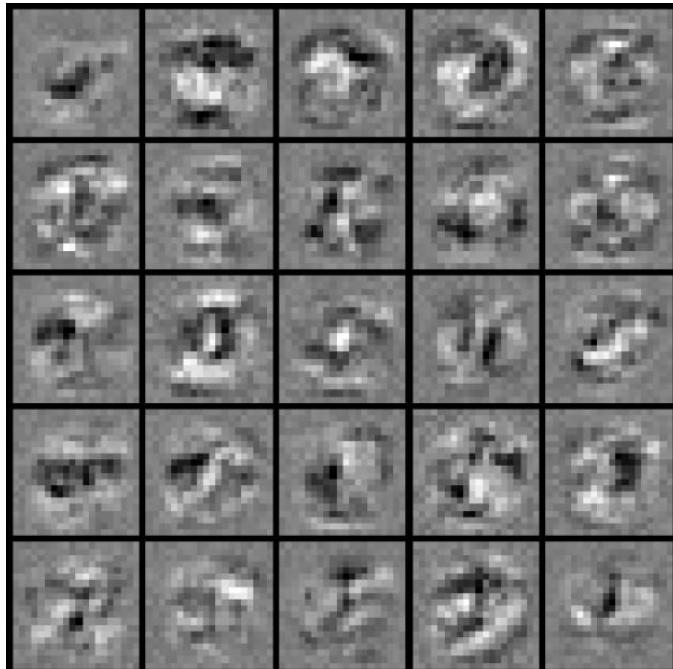
$d = 400$  10 classes



Each image is “unrolled” into a vector  $\mathbf{x}$  of pixel intensities

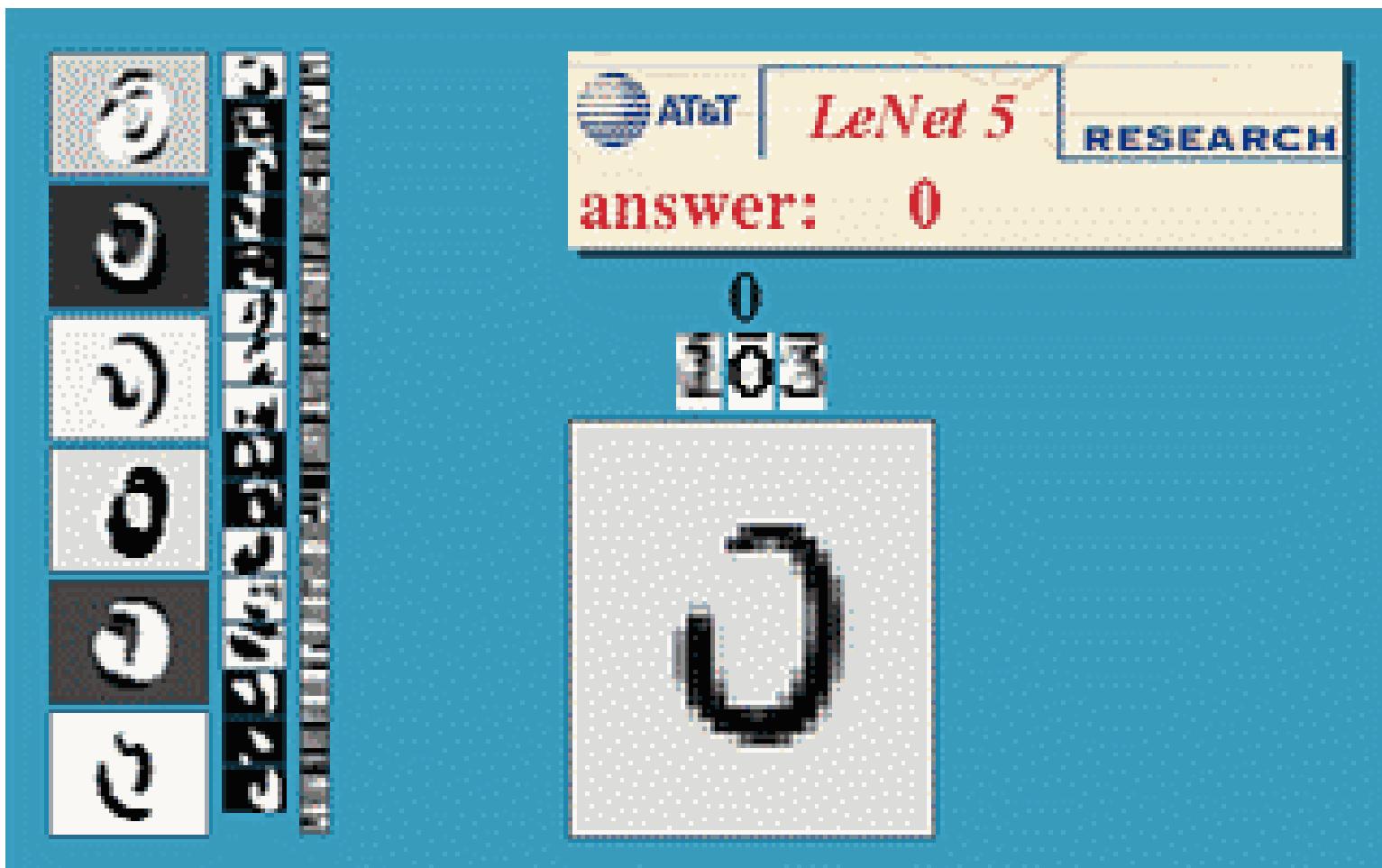
# Layering Representations

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



Visualization of  
Hidden Layer

# Digit Recognition



# Handwriting Recognition

LeNet 5 Demonstration:

<http://yann.lecun.com/exdb/lenet/>

<http://yann.lecun.com/exdb/lenet/weirdos.html>

# Perceptron learning rule

$$\theta \leftarrow \theta + \alpha(y - h(\mathbf{x}))\mathbf{x}$$

Equivalent to the intuitive rules:

- If output is correct, don't change the weights
- If output is low ( $h(\mathbf{x}) = 0, y = 1$ ), increment weights for all the inputs which are 1
- If output is high ( $h(\mathbf{x}) = 1, y = 0$ ), decrement weights for all inputs which are 1

## Perceptron Convergence Theorem:

- If there is a set of weights that is consistent with the training data (i.e., the data is linearly separable), the perceptron learning algorithm will converge [Minsky & Papert, 1969]

# Learning in NN: Backpropagation

- Similar to the perceptron learning algorithm, we cycle through our examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error
- The trick is to assess the blame for the error and divide it among the contributing weights

# Cost Function

(9.1 NN video of Andrew Ng)

Logistic Regression:

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

Neural Network:

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

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

$k^{\text{th}}$  class: true, predicted  
not  $k^{\text{th}}$  class: true, predicted

# Optimizing the Neural Network

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

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

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

- But, tends to work well in practice

Need code to compute:

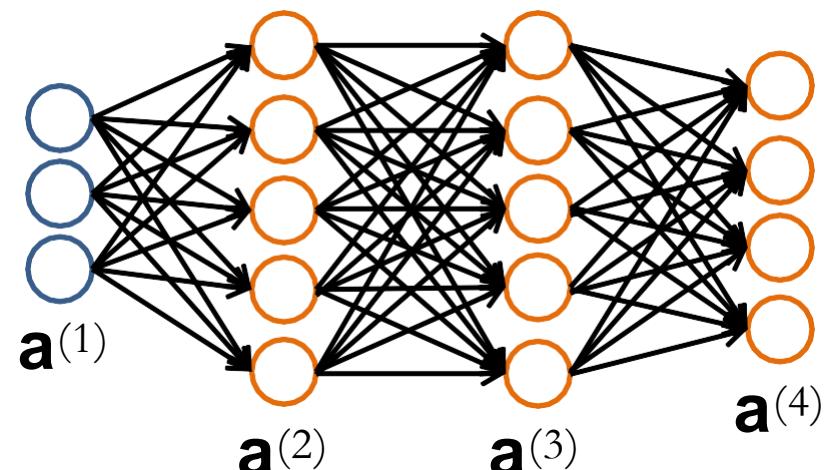
- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

# Forward Propagation

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

## Forward Propagation

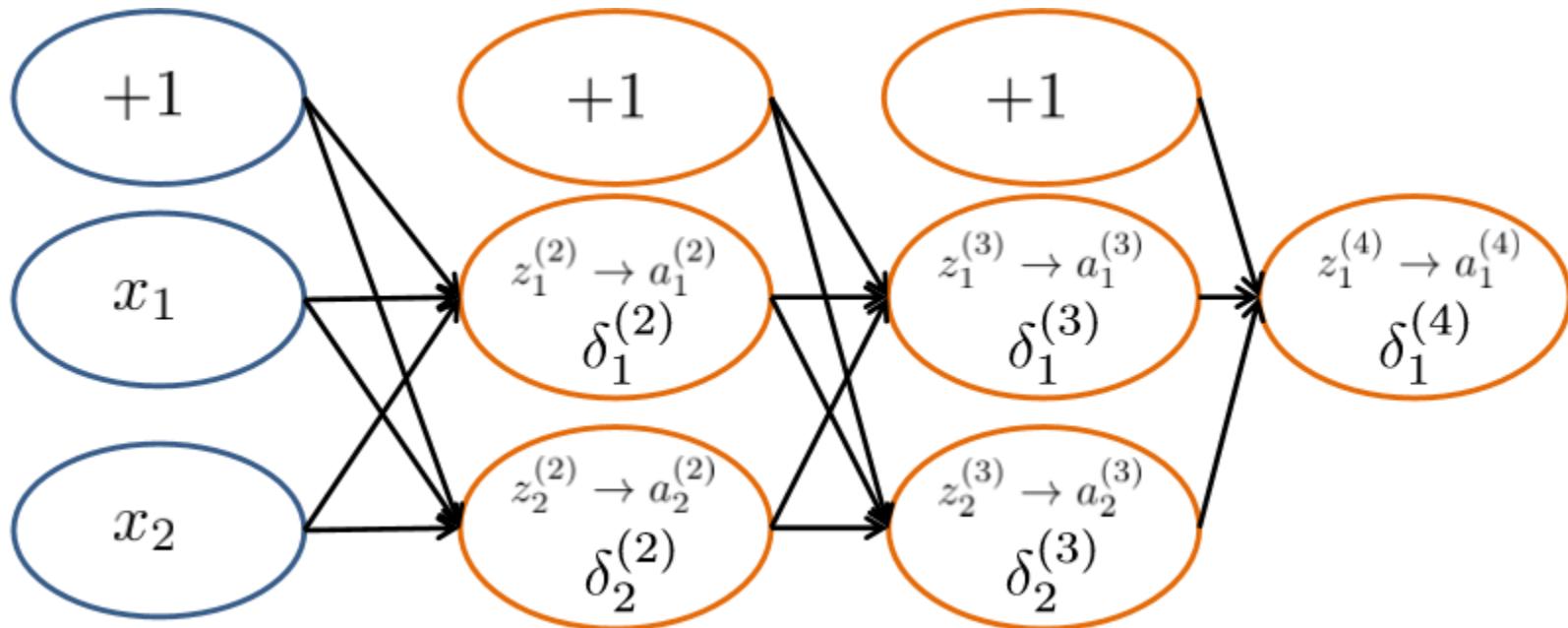
- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$  [add  $a_0^{(2)}$ ]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$  [add  $a_0^{(3)}$ ]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$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



# Backpropagation Intuition

- Each hidden node  $j$  is “responsible” for some fraction of the error  $\delta_j^{(l)}$  in each of the output nodes to which it connects
- $\delta_j^{(l)}$  is divided according to the strength of the connection between hidden node and the output node
- Then, the “blame” is propagated back to provide the error values for the hidden layer

# Backpropagation Intuition

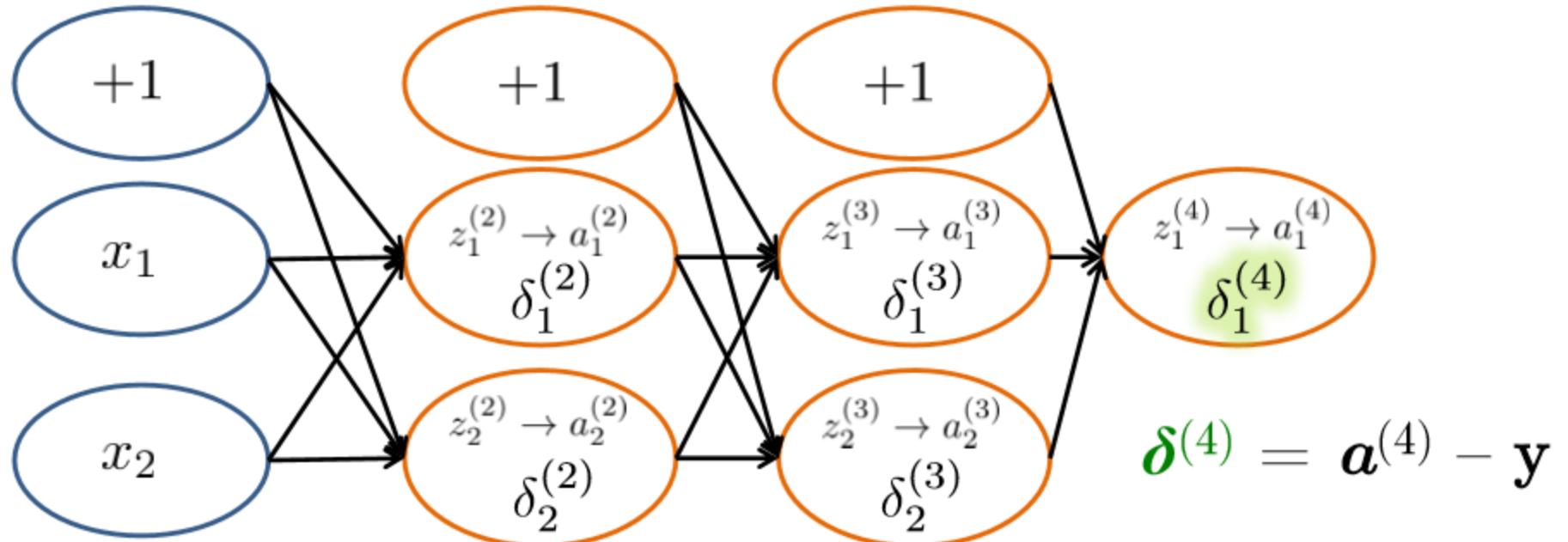


$\delta_j^{(l)}$  = “error” of node  $j$  in layer  $l$

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

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

# Backpropagation Intuition

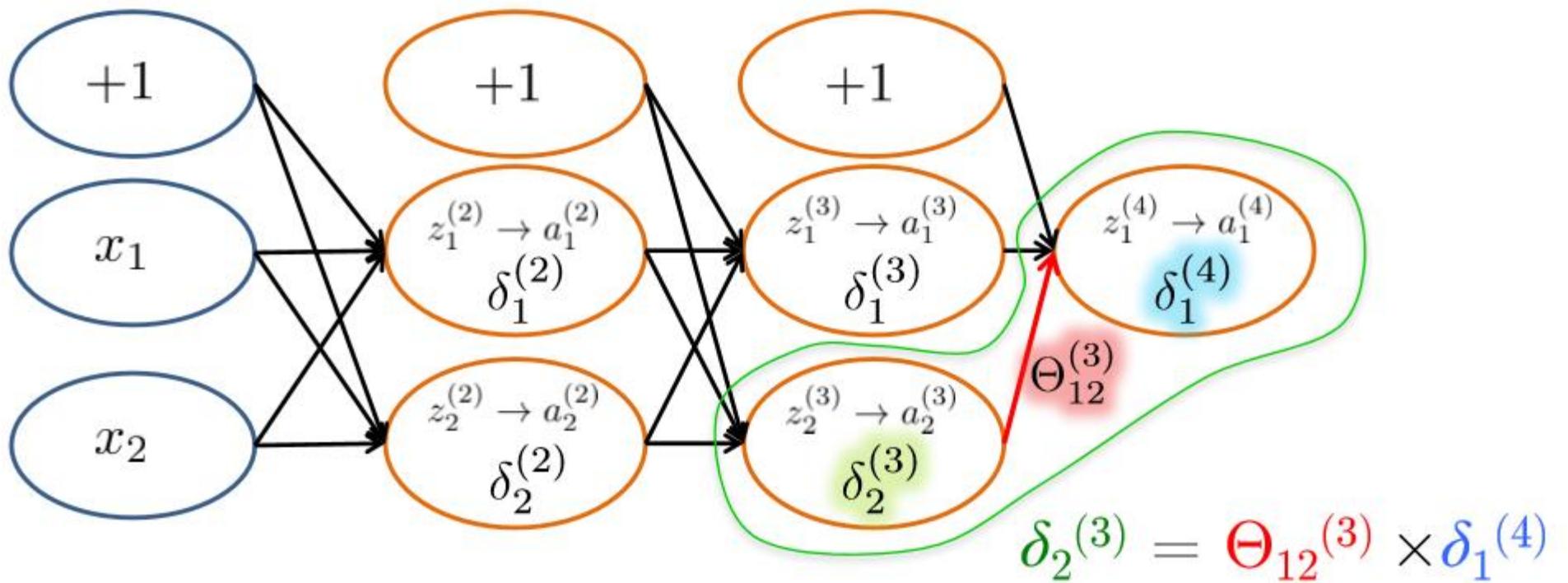


$\delta_j^{(l)}$  = “error” of node  $j$  in layer  $l$

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

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

# Backpropagation Intuition

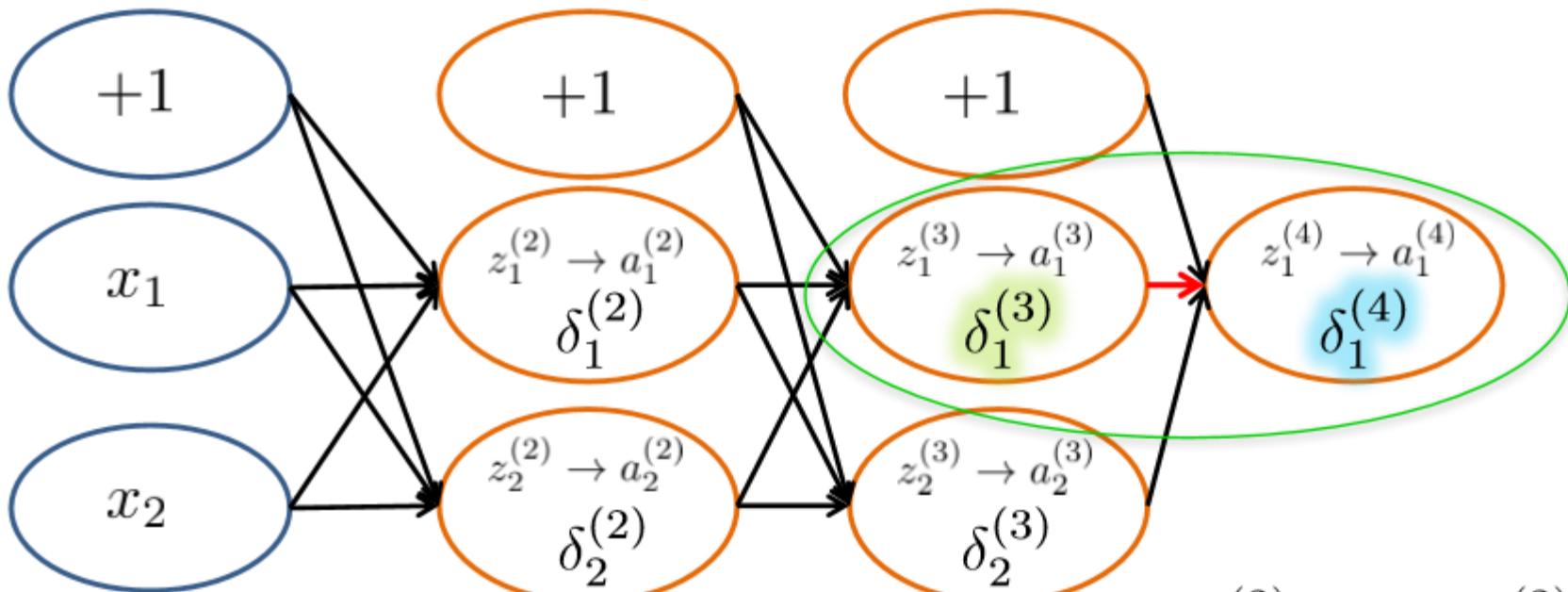


$\delta_j^{(l)}$  = “error” of node  $j$  in layer  $l$

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

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

# Backpropagation Intuition



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

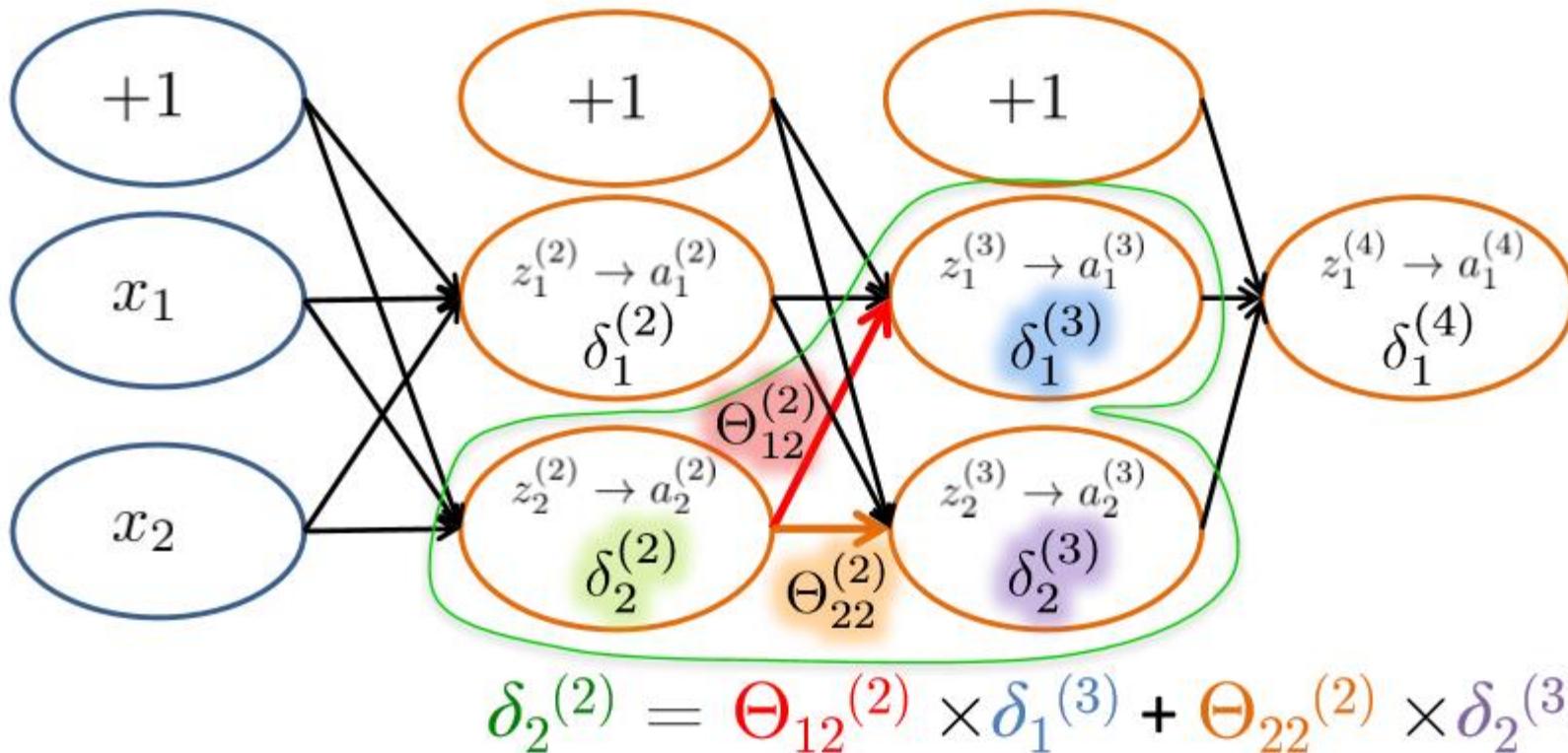
$$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)}$$

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

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

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

# Backpropagation Intuition



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

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

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

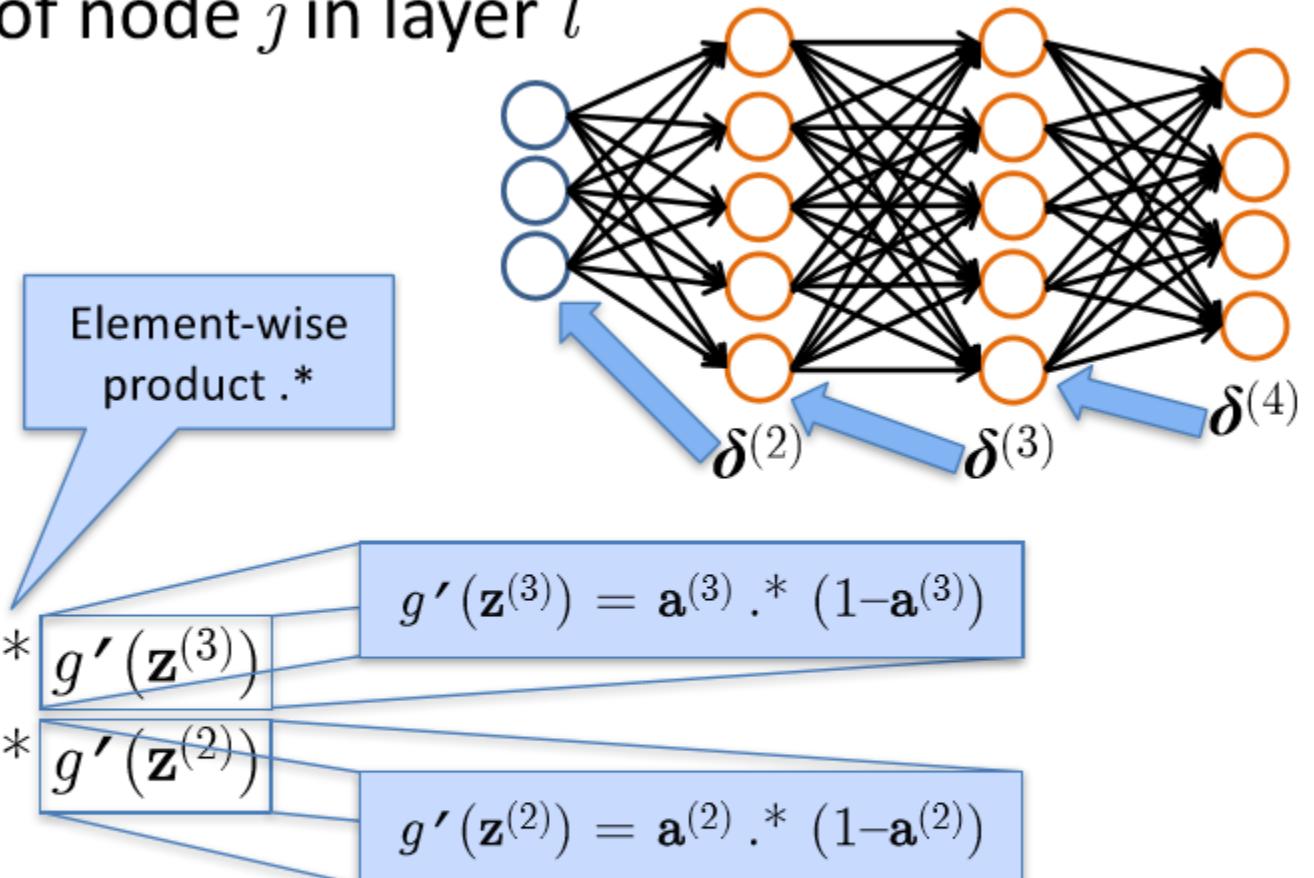
# Backpropagation: Gradient Computation

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

(#layers  $L = 4$ )

## Backpropagation

- $\delta^{(4)} = a^{(4)} - y$
- $\delta^{(3)} = (\Theta^{(3)})^\top \delta^{(4)} .*$
- $\delta^{(2)} = (\Theta^{(2)})^\top \delta^{(3)} .*$
- (No  $\delta^{(1)}$ )



$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\text{ignoring } \lambda; \text{ if } \lambda = 0)$$

# Backpropagation

Set  $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$

(Used to accumulate gradient)

For each training instance  $(\mathbf{x}_i, y_i)$ :

Set  $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute  $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$  via forward propagation

Compute  $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors  $\{\boldsymbol{\delta}^{(L-1)}, \dots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients  $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient  $D_{ij}^{(l)} = \begin{cases} \frac{1}{n} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

$D^{(l)}$  is the matrix of partial derivatives of  $J(\Theta)$

Note: Can vectorize  $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$  as  $\Delta^{(l)} = \Delta^{(l)} + \boldsymbol{\delta}^{(l+1)} \mathbf{a}^{(l)\top}$

# Backpropagation

Given: training set  $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$

Initialize all  $\Theta^{(l)}$  randomly (NOT to 0!)

Loop // each iteration is called an epoch

Set  $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$  (Used to accumulate gradient)

For each training instance  $(\mathbf{x}_i, y_i)$ :

Set  $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute  $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$  via forward propagation

Compute  $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors  $\{\boldsymbol{\delta}^{(L-1)}, \dots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients  $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient  $D_{ij}^{(l)} = \begin{cases} \frac{1}{n} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Update weights via gradient step  $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$

Until weights converge or max #epochs is reached

# Backprop Issues

“Backprop is the cockroach of machine learning. It’s ugly, and annoying, but you just can’t get rid of it.”

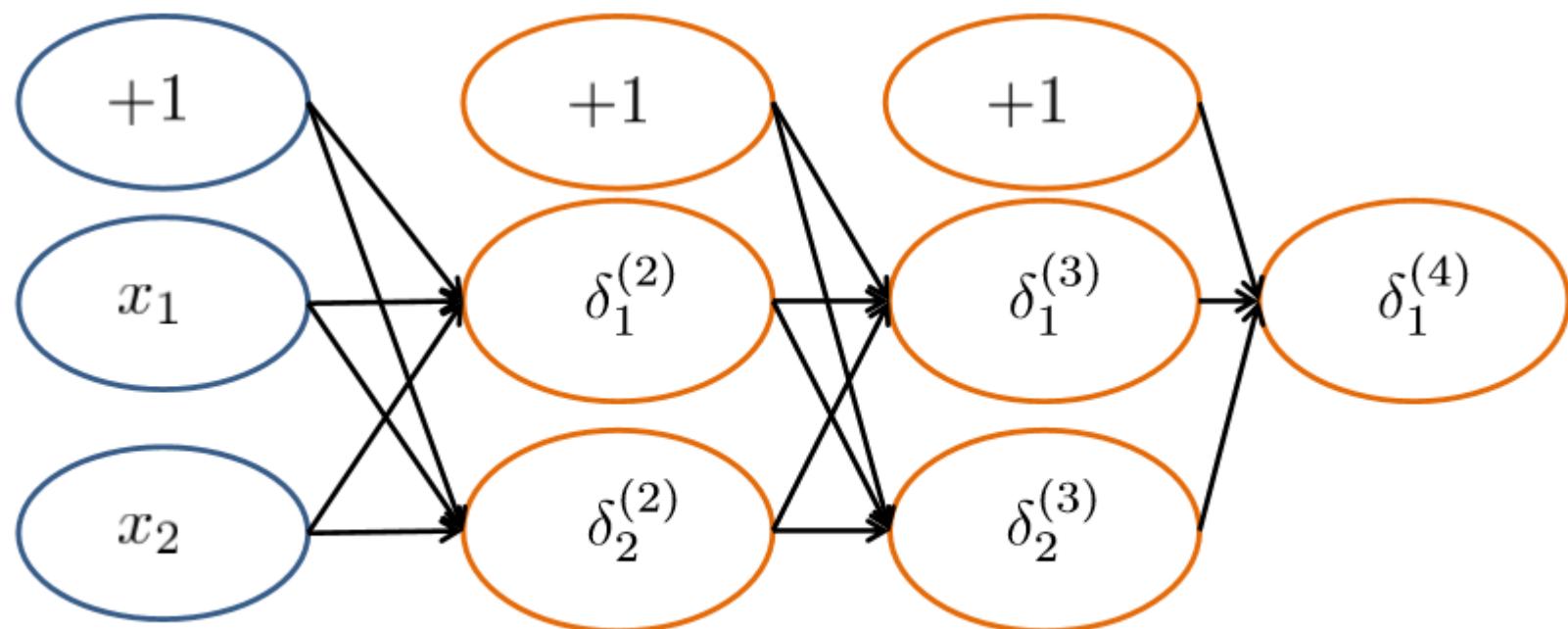
—Geoff Hinton

Problems:

- black box
- local minima

# Random Initialization

- Important to randomize initial weight matrices
- Can't have uniform initial weights, as in logistic regression
  - Otherwise, all updates will be identical & the net won't learn



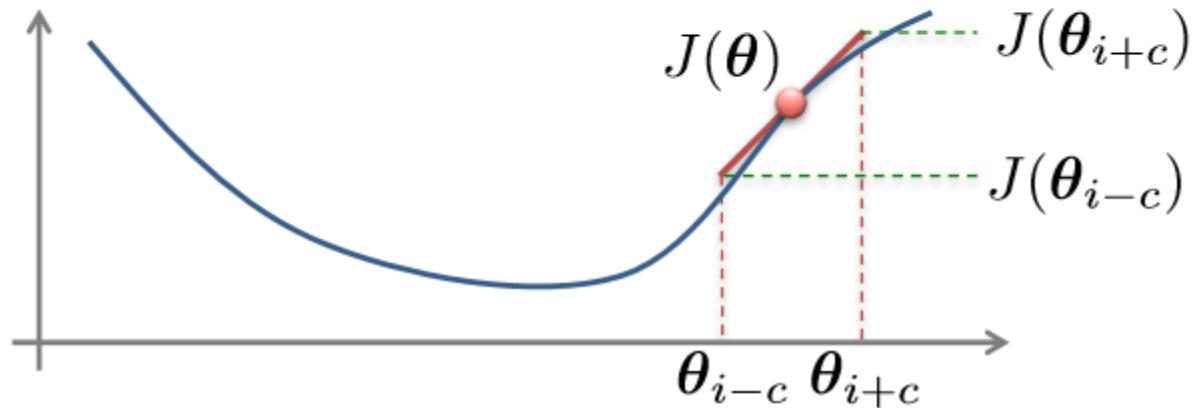
# Implementation Details

- For convenience, compress all parameters into  $\theta$ 
  - “unroll”  $\Theta^{(1)}, \Theta^{(2)}, \dots, \Theta^{(L-1)}$  into one long vector  $\theta$ 
    - E.g., if  $\Theta^{(1)}$  is  $10 \times 10$ , then the first 100 entries of  $\theta$  contain the value in  $\Theta^{(1)}$
  - Use the `reshape` command to recover the original matrices
    - E.g., if  $\Theta^{(1)}$  is  $10 \times 10$ , then

```
theta1 = reshape(theta[0:100], (10, 10))
```
- Each step, check to make sure that  $J(\theta)$  decreases
- Implement a gradient-checking procedure to ensure that the gradient is correct...

# Gradient Checking

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



$$\frac{\partial}{\partial \theta_i} J(\theta) \approx \frac{J(\theta_{i+c}) - J(\theta_{i-c})}{2c} \quad c \approx 1E-4$$

$$\theta_{i+c} = [\theta_1, \theta_2, \dots, \theta_{i-1}, \theta_i + c, \theta_{i+1}, \dots]$$

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

# Gradient Checking

$\theta \in \mathbb{R}^m$      $\theta$  is an “unrolled” version of  $\Theta^{(1)}, \Theta^{(2)}, \dots$

$$\theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_m]$$

Put in vector called `gradApprox`

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

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

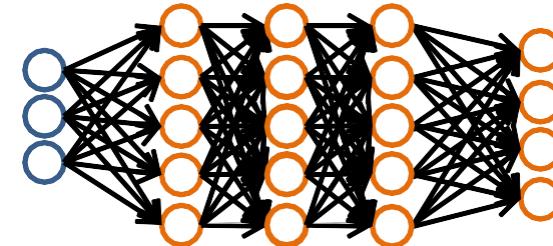
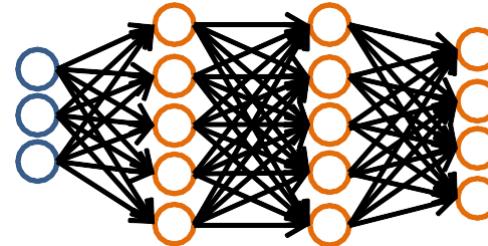
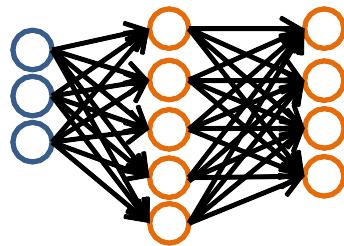
⋮

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

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

# Training a Neural Network

Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

**Reasonable default:** 1 hidden layer

- or if >1 hidden layer, have same # hidden units in every layer (usually the more the better)

# Training a Neural Network

1. Randomly initialize weights
2. Implement forward propagation to get  $h_{\Theta}(\mathbf{x}_i)$  for any instance  $\mathbf{x}_i$
3. Implement code to compute cost function  $J(\Theta)$
4. Implement backprop to compute partial derivatives
$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$$
5. Use gradient checking to compare  $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$  computed using backpropagation vs. the numerical gradient estimate.
  - Then, disable gradient checking code
6. Use gradient descent with backprop to fit the network

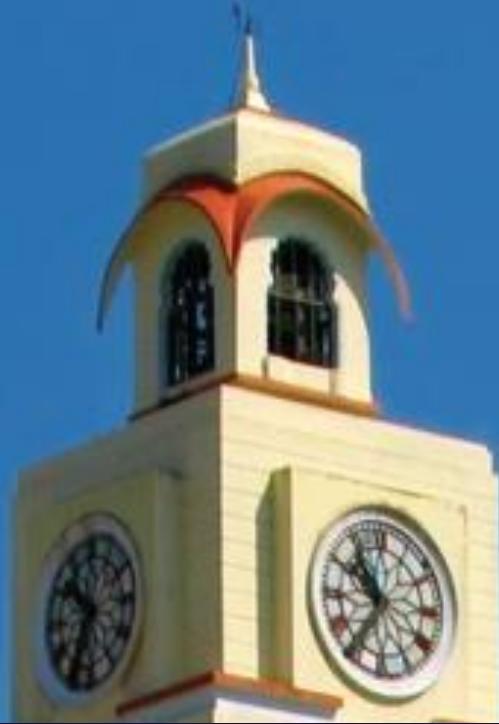
# Good References for understanding Neural Network

Andrew Ng videos on neural network

[https://www.youtube.com/watch?v=EVegrPGfuCY&list=PLLssT5z\\_DsK-h9vYZkQkYNWcItqhlRJLN&index=45](https://www.youtube.com/watch?v=EVegrPGfuCY&list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN&index=45)

Autonomous driving using neural network

[https://www.youtube.com/watch?v=ppFyPUx9RIU&list=PLLssT5z\\_DsK-h9vYZkQkYNWcItqhlRJLN&index=57](https://www.youtube.com/watch?v=ppFyPUx9RIU&list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN&index=57)



# Machine Learning

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**BITS** Pilani

Pilani Campus



## **Lecture No. – 10 | Convolutional Neural Network**

**Date – 18/01/2020**

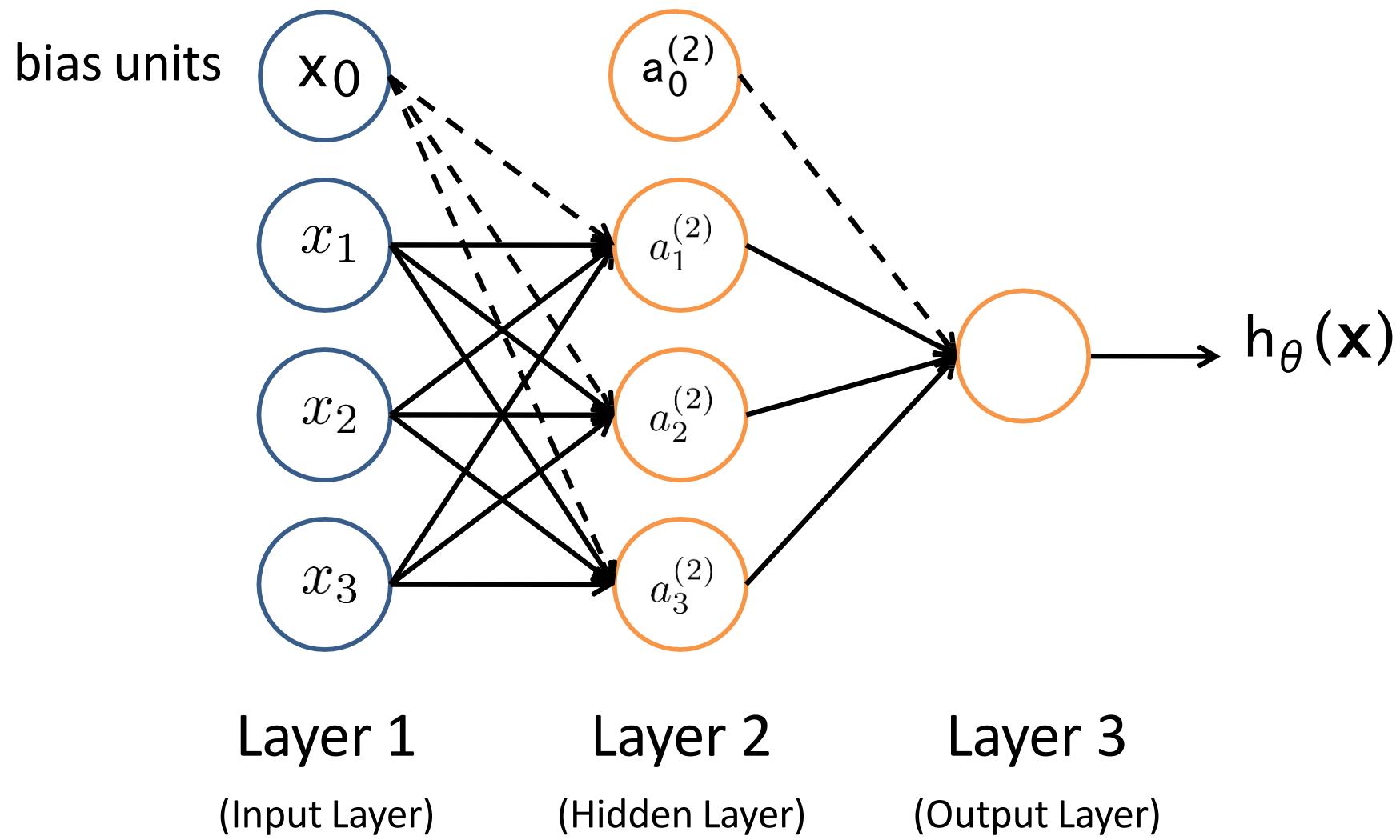
**Time – 9:00 AM – 11:00 AM**

These slides are prepared by the instructor, with grateful acknowledgement of Tom Mitchell, Mitesh Khapra and many others who made their course materials freely available online.

# Session Content

- Back propagation Algorithm (Andrew Ng Notes and Tom Mitchell chapter 4)
  - Training Neural Network
  - Hidden Layer Representations
  - Overfitting in ANN
  - Early Stopping
  - Drop out
  - Parameter sharing
- Convolutional Neural Network

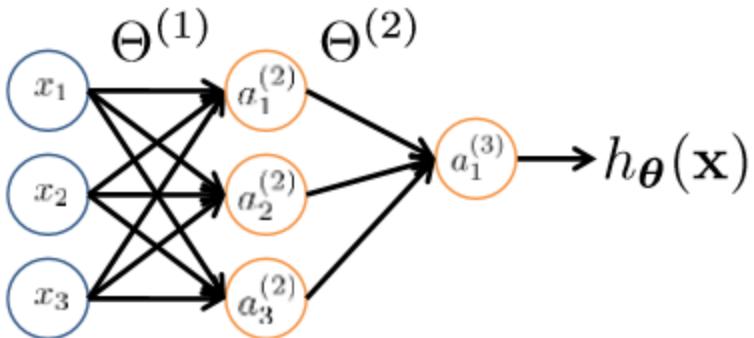
# Neural Network



# Feed-Forward Process

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

# Neural Network



$a_i^{(j)}$  = “activation” of unit  $i$  in layer  $j$   
 $\Theta^{(j)}$  = weight matrix controlling function mapping from layer  $j$  to layer  $j + 1$

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

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

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

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

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

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

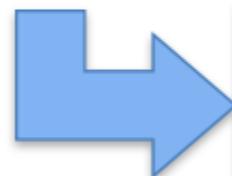
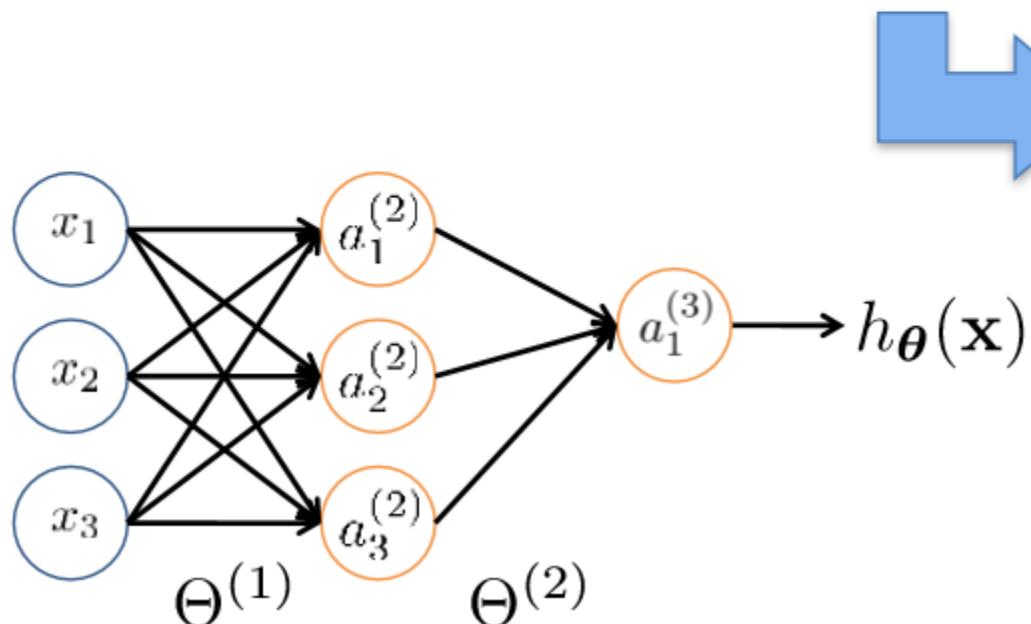
# Vectorization

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

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

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

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



Feed-Forward Steps:

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

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

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

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

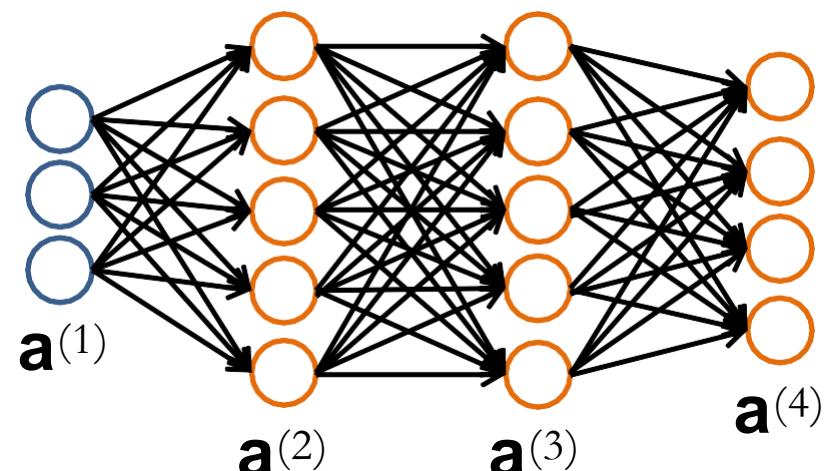
$$h_{\Theta}(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

# Forward Propagation

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

## Forward Propagation

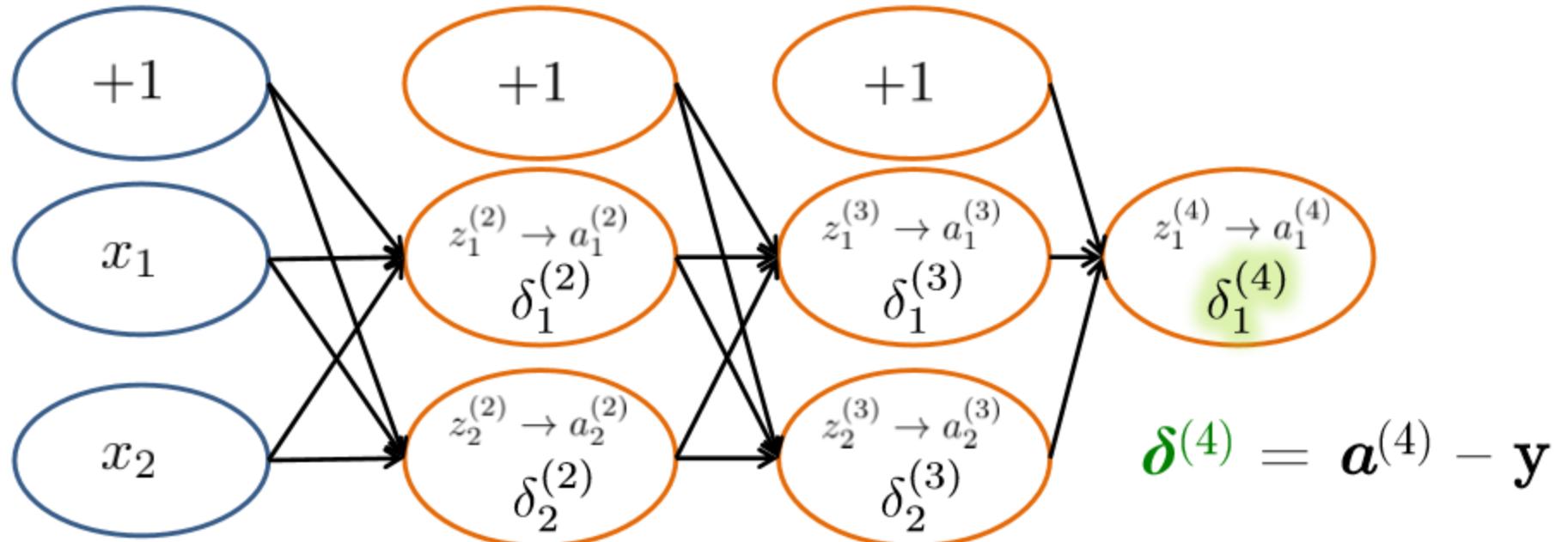
- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$  [add  $a_0^{(2)}$ ]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$  [add  $a_0^{(3)}$ ]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$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



# Backpropagation Intuition

- Each hidden node  $j$  is “responsible” for some fraction of the error  $\delta_j^{(l)}$  in each of the output nodes to which it connects
- $\delta_j^{(l)}$  is divided according to the strength of the connection between hidden node and the output node
- Then, the “blame” is propagated back to provide the error values for the hidden layer

# Backpropagation Intuition

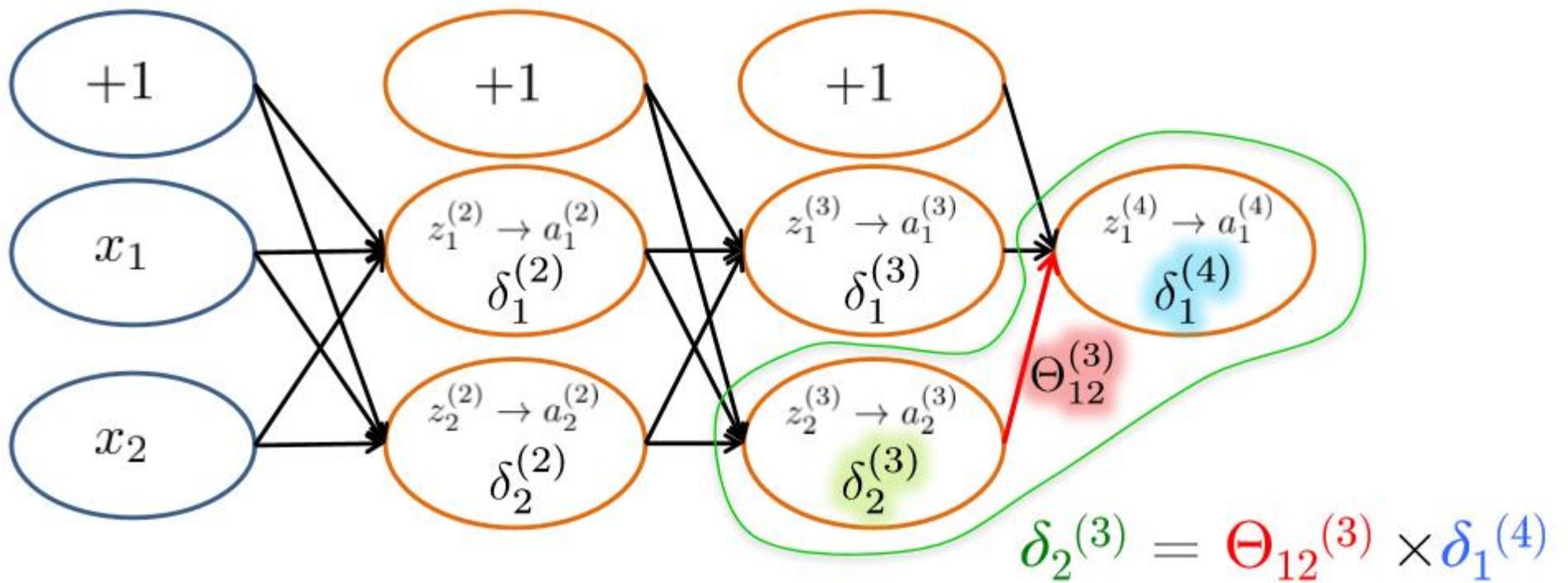


$\delta_j^{(l)}$  = “error” of node  $j$  in layer  $l$

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

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

# Backpropagation Intuition

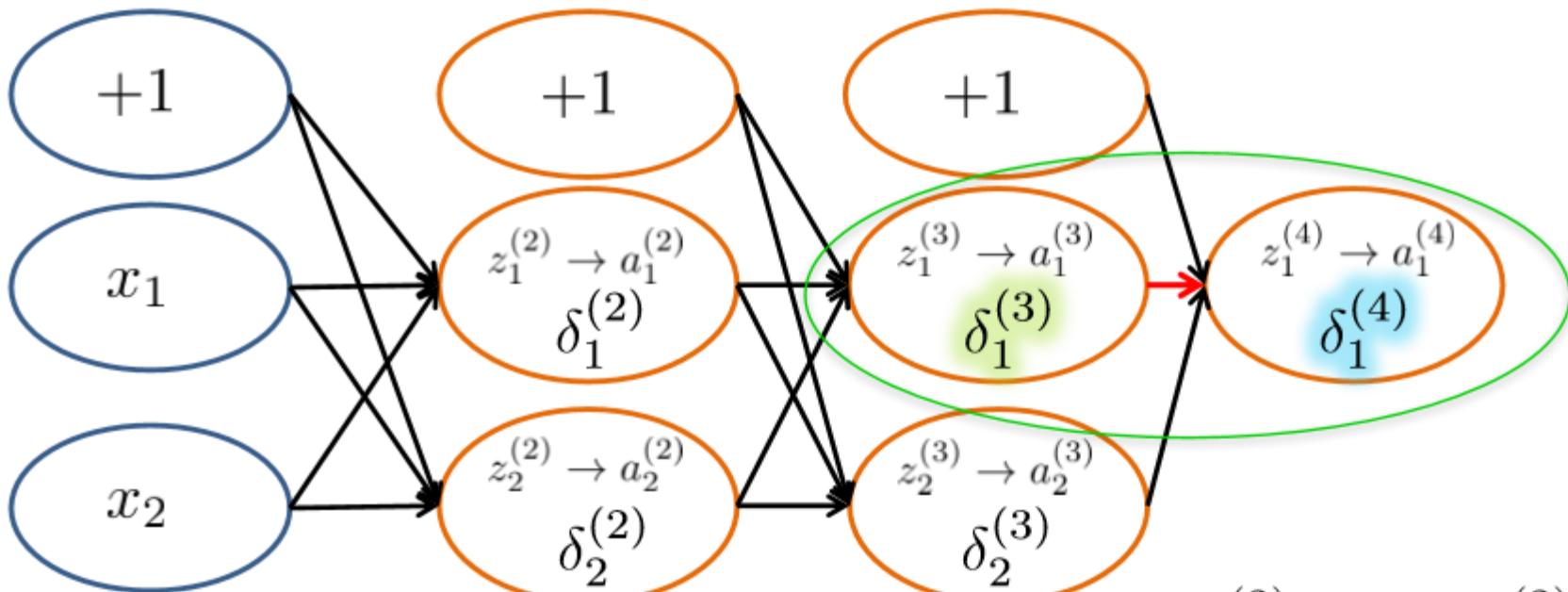


$\delta_j^{(l)}$  = “error” of node  $j$  in layer  $l$

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

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

# Backpropagation Intuition



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

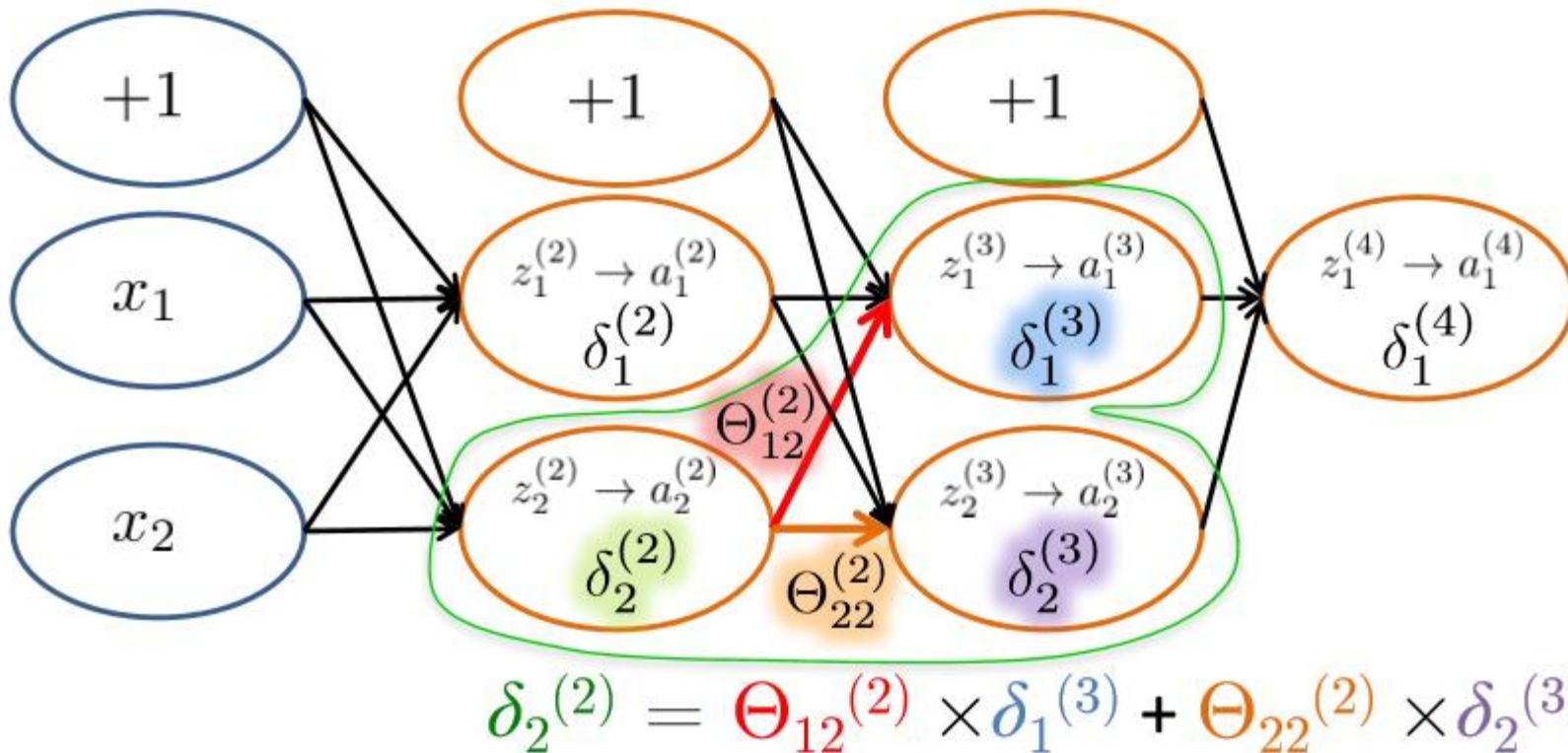
$$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)}$$

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

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

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

# Backpropagation Intuition



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

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

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

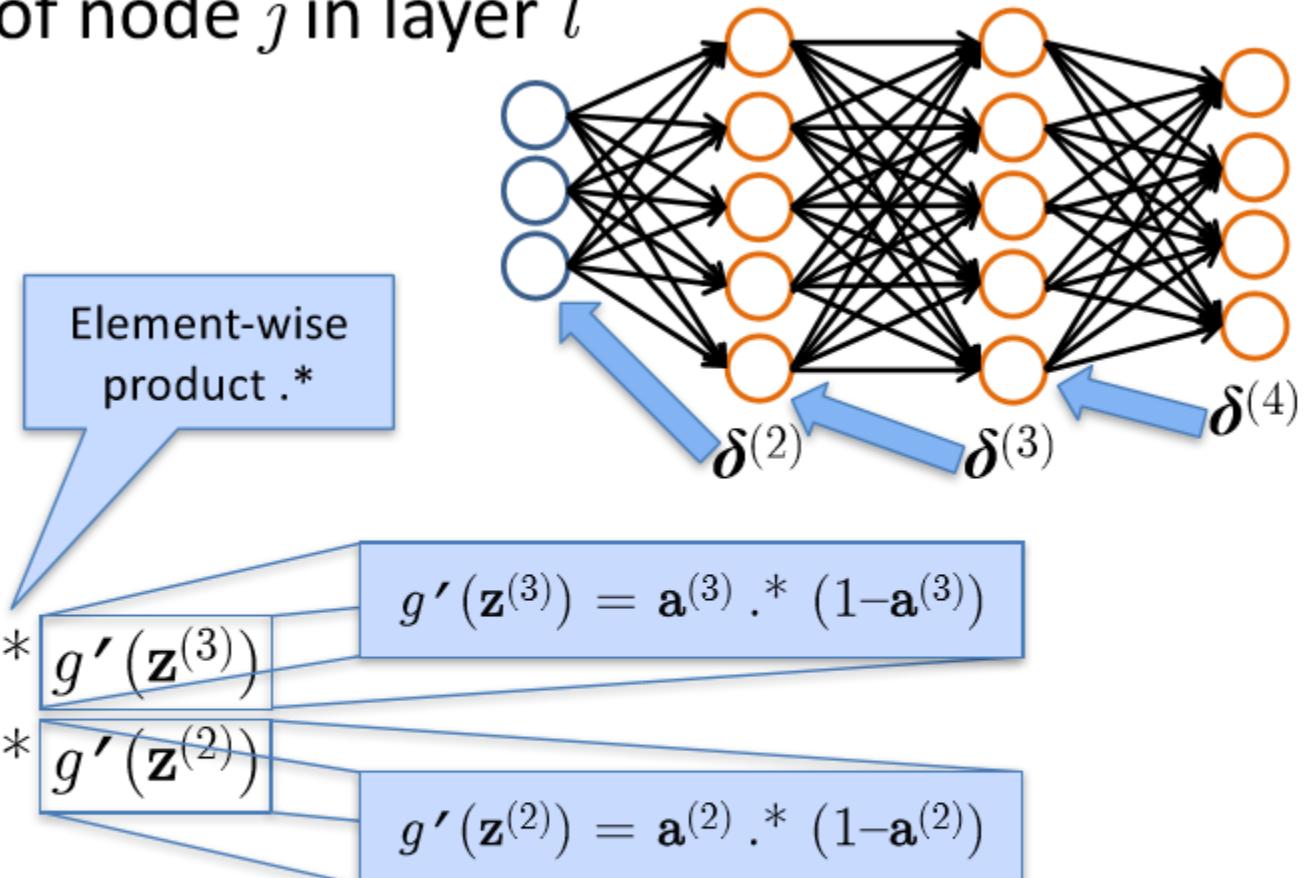
# Backpropagation: Gradient Computation

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

(#layers  $L = 4$ )

## Backpropagation

- $\delta^{(4)} = a^{(4)} - y$
- $\delta^{(3)} = (\Theta^{(3)})^\top \delta^{(4)} .*$
- $\delta^{(2)} = (\Theta^{(2)})^\top \delta^{(3)} .*$
- (No  $\delta^{(1)}$ )



$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\text{ignoring } \lambda; \text{ if } \lambda = 0)$$

# Backpropagation

Given: training set  $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$

Initialize all  $\Theta^{(l)}$  randomly (NOT to 0!)

Loop // each iteration is called an epoch

Set  $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$  (Used to accumulate gradient)

For each training instance  $(\mathbf{x}_i, y_i)$ :

Set  $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute  $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$  via forward propagation

Compute  $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors  $\{\boldsymbol{\delta}^{(L-1)}, \dots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients  $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

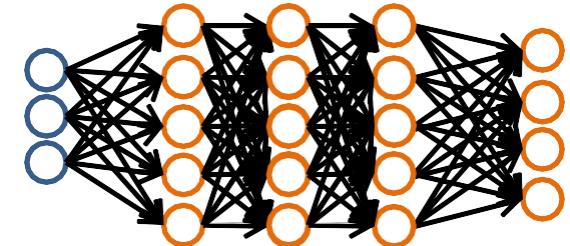
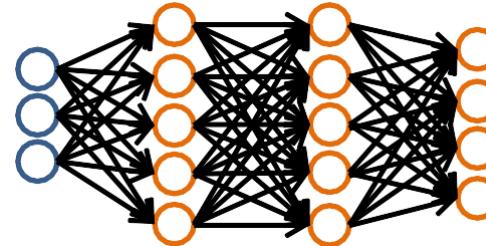
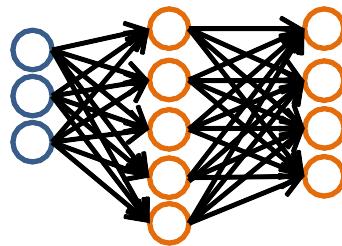
Compute avg regularized gradient  $D_{ij}^{(l)} = \begin{cases} \frac{1}{n} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Update weights via gradient step  $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$

Until weights converge or max #epochs is reached

# Training a Neural Network

Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

**Reasonable default:** 1 hidden layer

- or if >1 hidden layer, have same # hidden units in every layer (usually the more the better)

# Training a Neural Network

1. Randomly initialize weights
2. Implement forward propagation to get  $h_{\Theta}(\mathbf{x}_i)$  for any instance  $\mathbf{x}_i$
3. Implement code to compute cost function  $J(\Theta)$
4. Implement backprop to compute partial derivatives
$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$$
5. Use gradient checking to compare  $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$  computed using backpropagation vs. the numerical gradient estimate.
  - Then, disable gradient checking code
6. Use gradient descent with backprop to fit the network

## More on Backpropagation

---

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum*  $\alpha$ 
$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n - 1)$$
- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations → slow!
- Using network after training is very fast

# Expressive Capabilities of ANNs

---

Boolean functions:

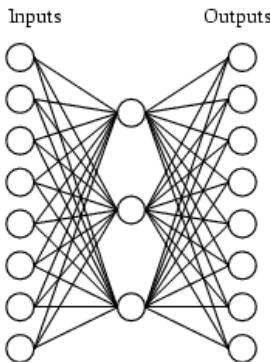
- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

# Learning Hidden Layer Representations

---



A target function:

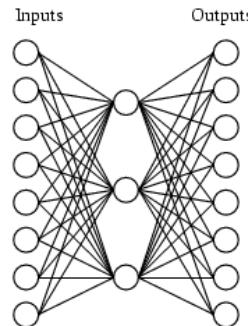
Input	Output
10000000	$\rightarrow$ 10000000
01000000	$\rightarrow$ 01000000
00100000	$\rightarrow$ 00100000
00010000	$\rightarrow$ 00010000
00001000	$\rightarrow$ 00001000
00000100	$\rightarrow$ 00000100
00000010	$\rightarrow$ 00000010
00000001	$\rightarrow$ 00000001

Can this be learned??

# Learning Hidden Layer Representations

---

A network:

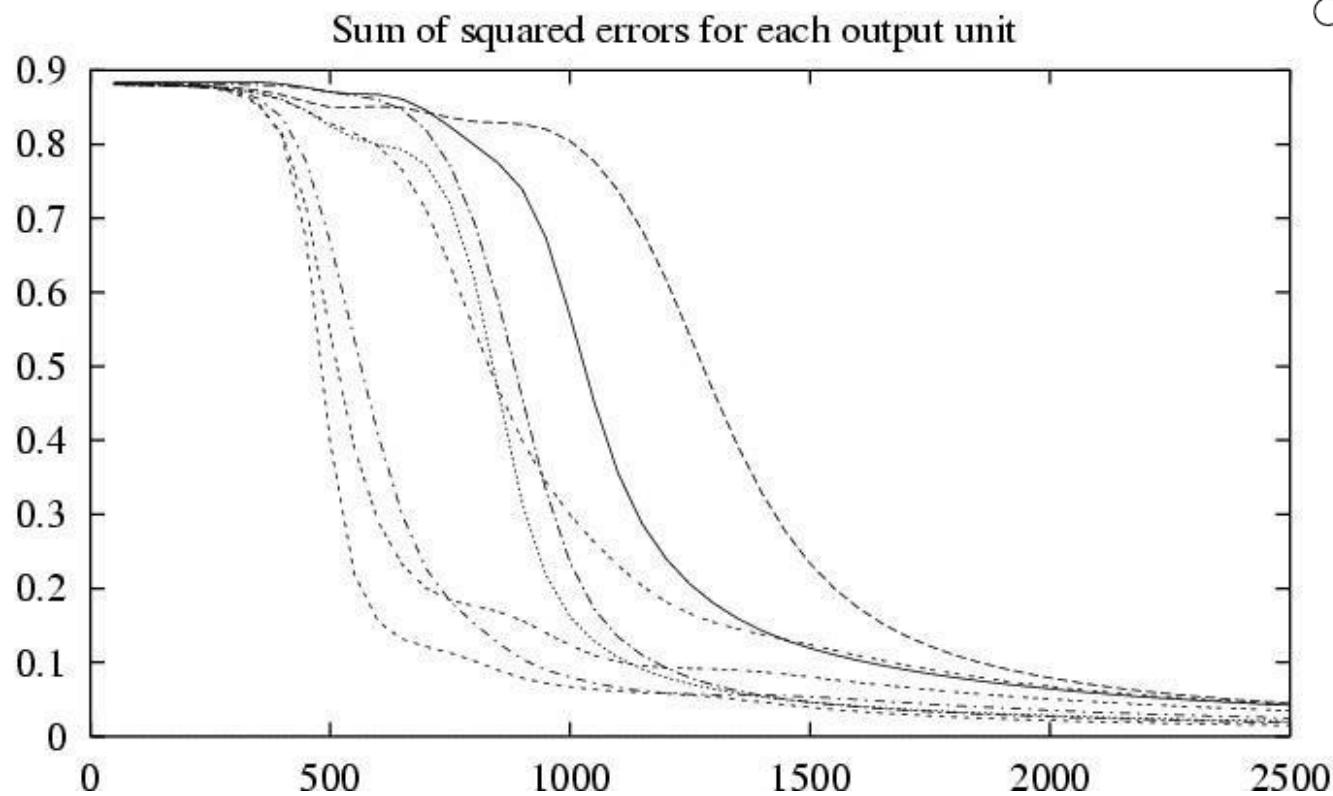
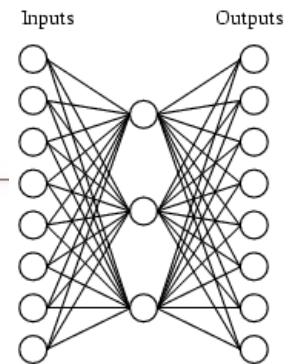


Learned hidden layer representation:

Input	Hidden Values	Output
10000000	→ .89 .04 .08	→ 10000000
01000000	→ .01 .11 .88	→ 01000000
00100000	→ .01 .97 .27	→ 00100000
00010000	→ .99 .97 .71	→ 00010000
00001000	→ .03 .05 .02	→ 00001000
00000100	→ .22 .99 .99	→ 00000100
00000010	→ .80 .01 .98	→ 00000010
00000001	→ .60 .94 .01	→ 00000001

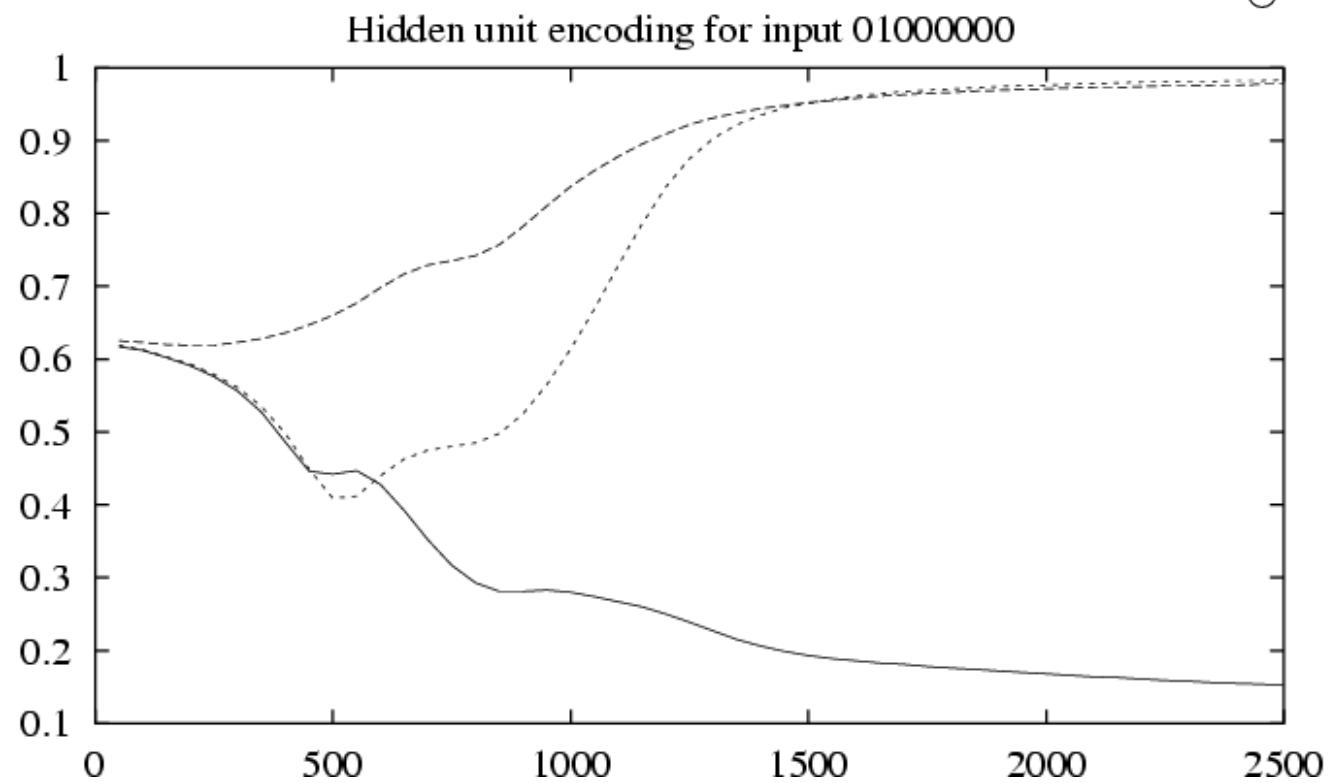
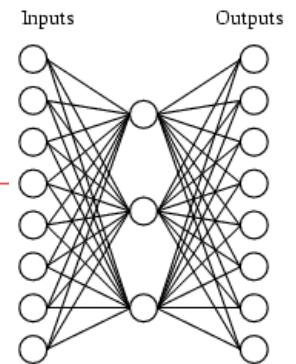
# Training

---



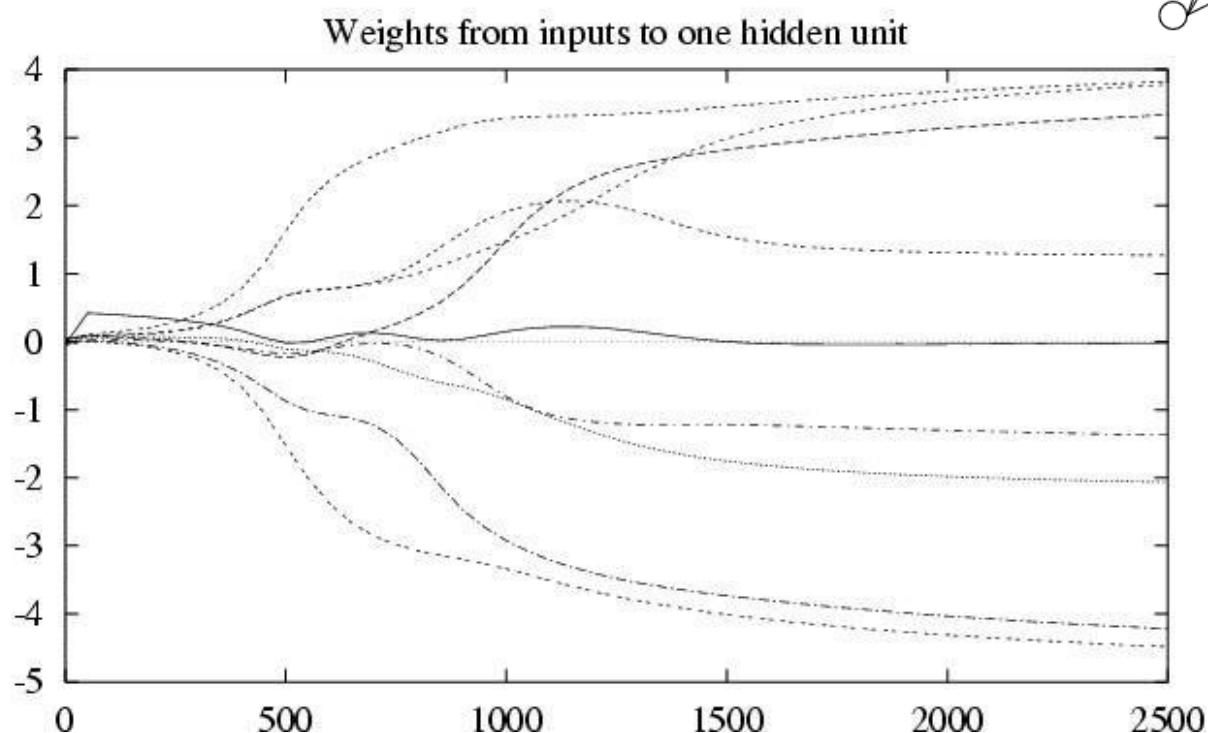
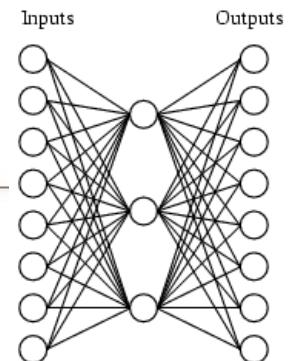
# Training

---



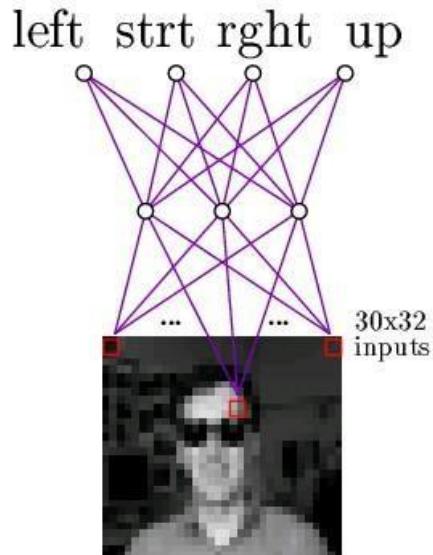
# Training

---



# Neural Nets for Face Recognition

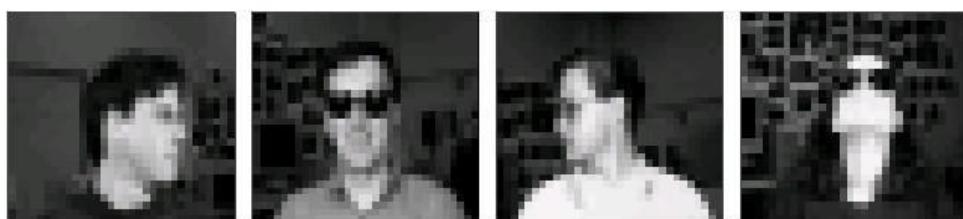
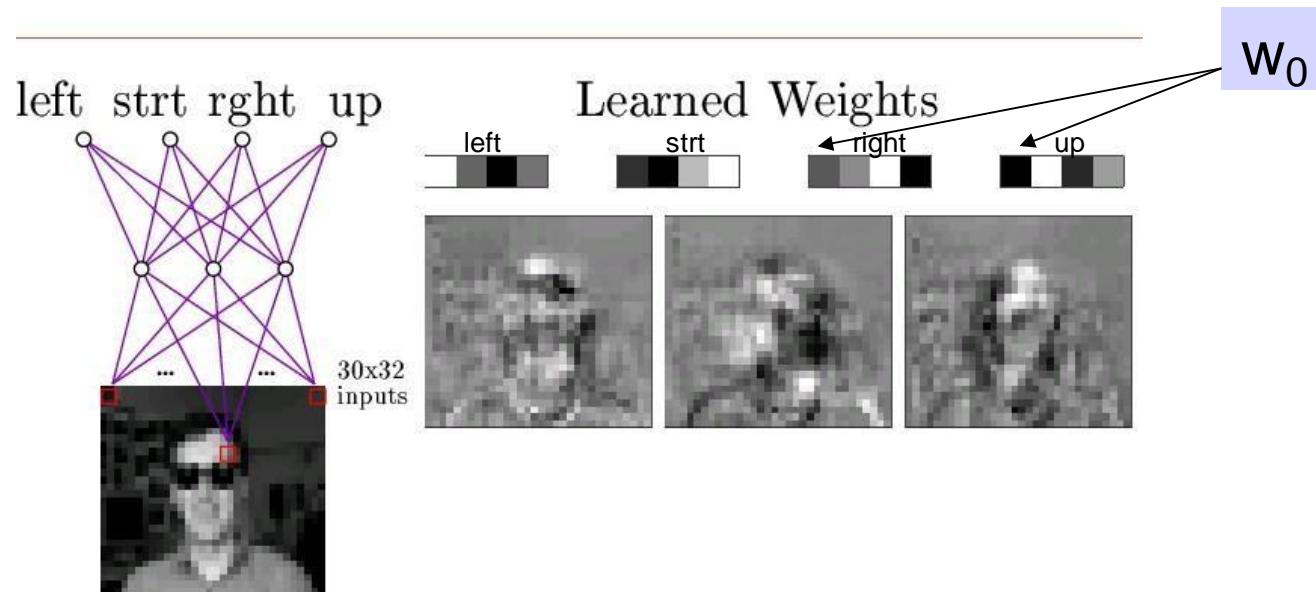
---



Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

# Learned Hidden Unit Weights

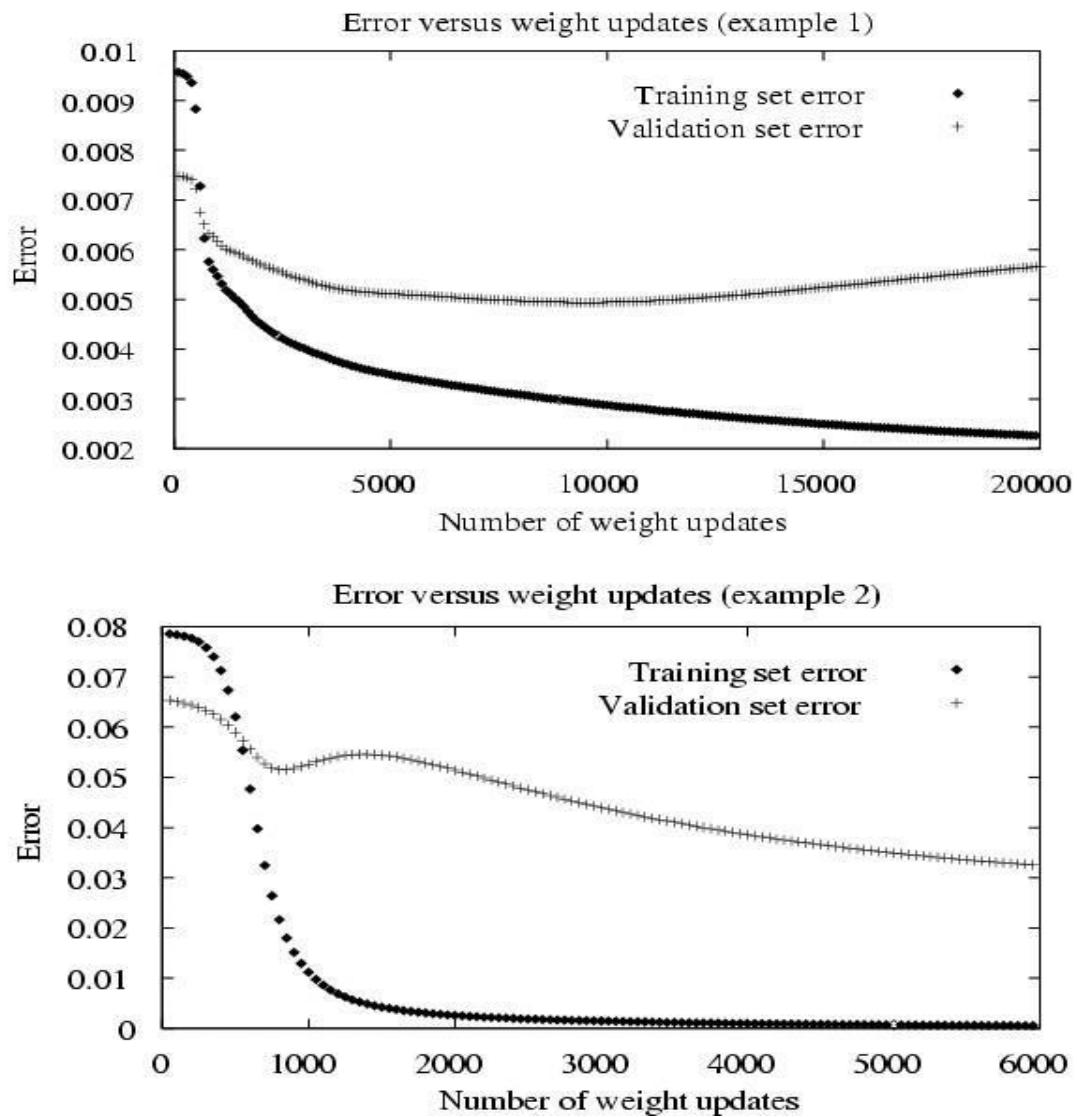


Typical input images

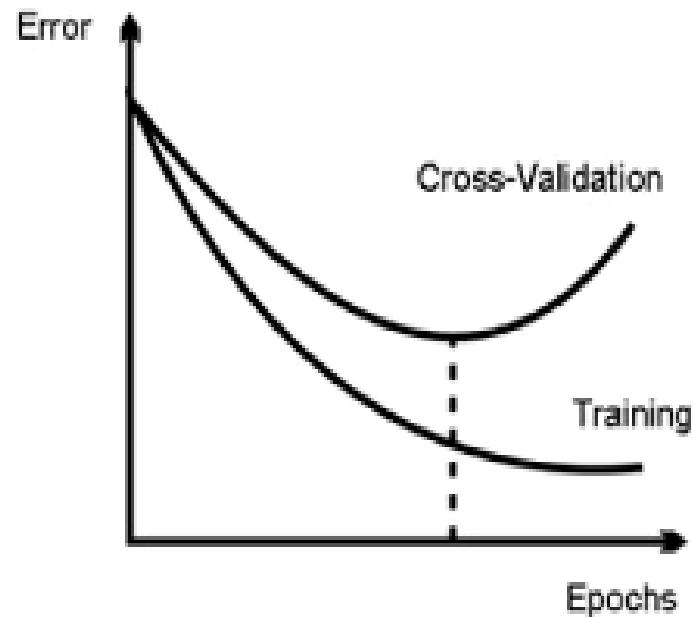
<http://www.cs.cmu.edu/~tom/faces.html>

# Overfitting in ANNs

---



# Early Stopping



**Figure 2:** Profiles for training and cross-validation errors.

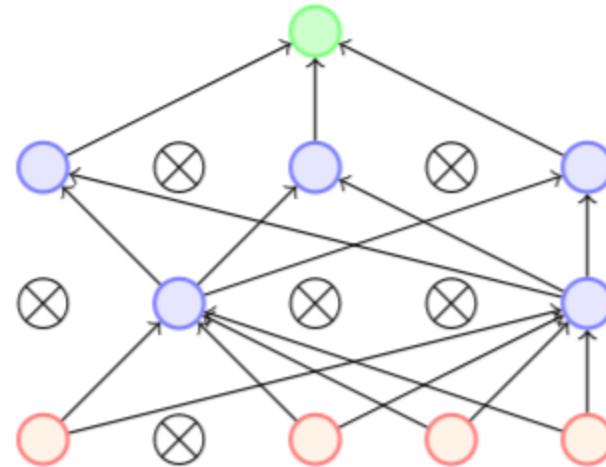
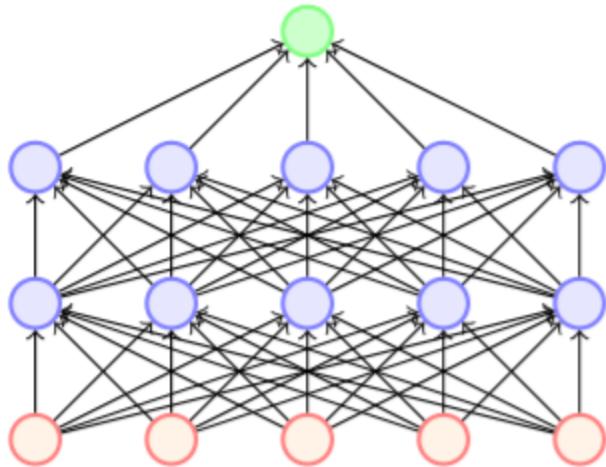
If we let a complex model train long enough on a given data set it can eventually learn the data exactly.

Given data that isn't represented in the training set, the model will perform poorly when analyzing the data (overfitting).

**How is the sweet spot for training located?**

When the error on the training set begins to deviate from the error on the validation set, a threshold can be set to determine the early stopping condition and the ideal number of epochs to train.

# Dropout



- Dropout refers to dropping out units
- Temporarily remove a node and all its incoming/outgoing connections resulting in a thinned network
- Each node is retained with a fixed probability (typically  $p = 0.5$ ) for hidden nodes and  $p = 0.8$  for visible nodes

# Dropouts

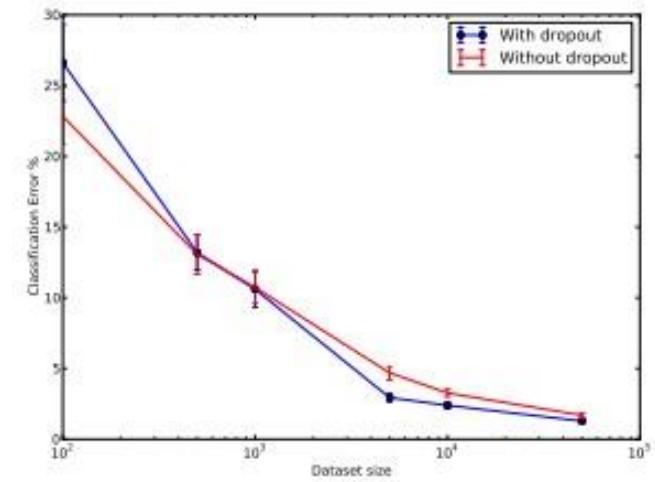
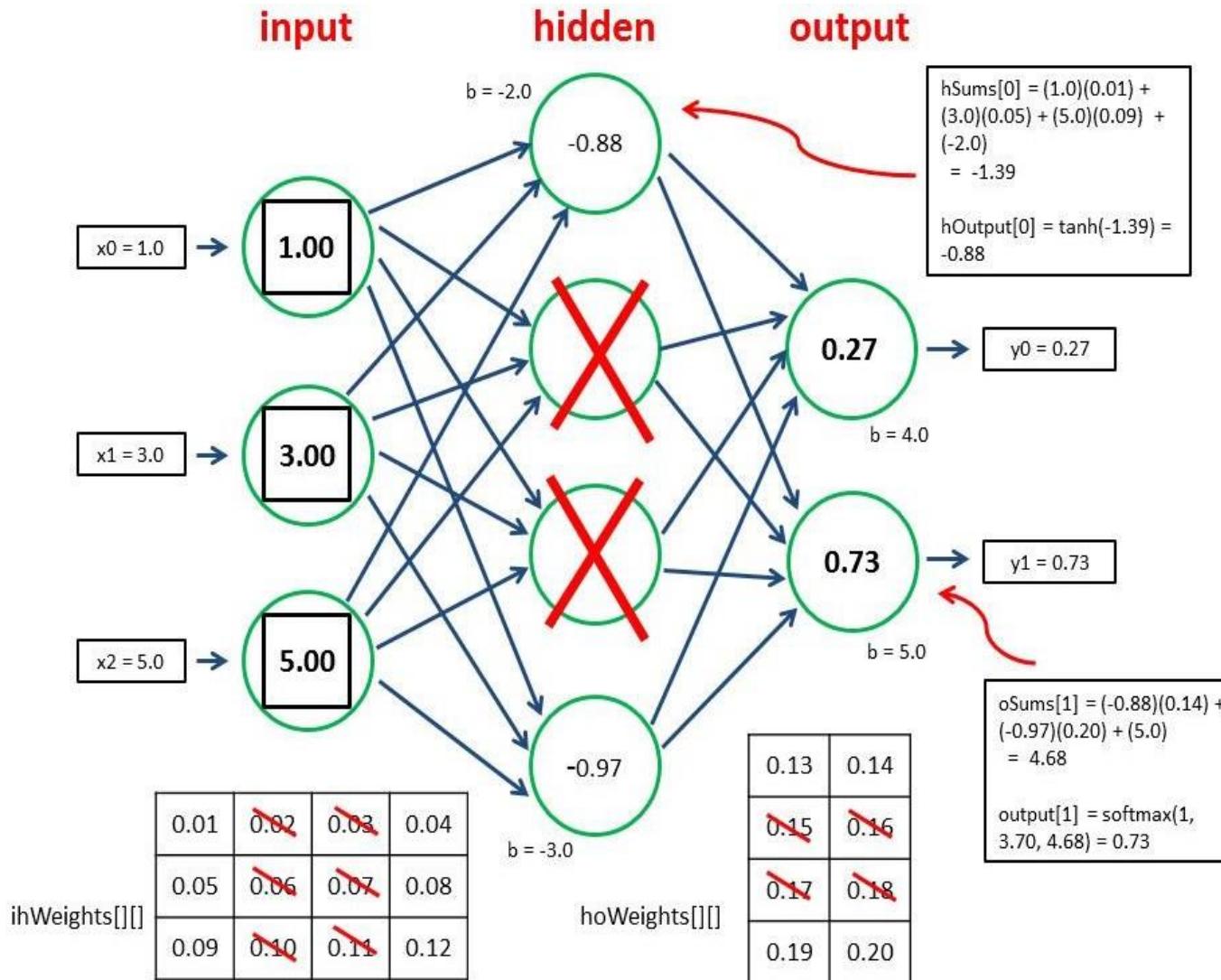


Figure 10: Effect of varying data set size.

# Parameter Sharing

---

Parameter sharing is where we:

- Force sets of parameters to be equal

- Because we interpret various models or model components as sharing a unique set of parameters
- Only a subset of the parameters needs to be stored in memory
  - In a CNN significant reduction in the memory footprint of the model

Sharing parameters gives the network the ability to look for a given feature everywhere in the data(image), rather than in just a certain area. This is extremely useful when the object of interest could be anywhere in the data(image).

It is uncommon to have training data where useful features will usually always be in the same area, so this is not seen often.

# Convolutional Neural Network

# Convolution



$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (x * w)_t$$

input                      filter  
                                convolution

A mathematical equation illustrating convolution. It shows the output  $s_t$  as a weighted sum of past input measurements  $x_{t-a}$  and a filter  $w_{-a}$ . The result is labeled as the convolution of the input and filter at time  $t$ .

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average

# Convolution



- We can think of images as 2D inputs
- We would now like to use a 2D filter ( $m \times n$ )
- First let us see what the 2D formula looks like
- This formula looks at all the preceding neighbours ( $i - a, j - b$ )

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a, j-b} K_{a,b}$$

# Convolution



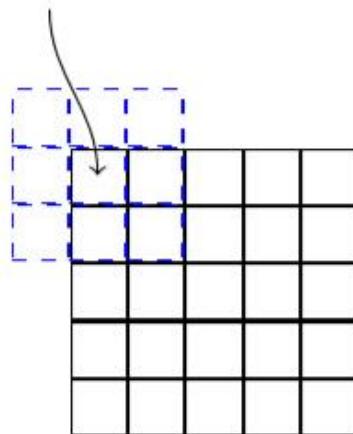
$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a,b}$$

- We can think of images as 2D inputs
- We would now like to use a 2D filter ( $m \times n$ )
- First let us see what the 2D formula looks like
- This formula looks at all the preceding neighbours ( $i - a, j - b$ )
- In practice, we use the following formula which looks at the succeeding neighbours

# Convolution

$$S_{ij} = (I * K)_{ij} = \sum_{a=\left\lfloor -\frac{m}{2} \right\rfloor}^{\left\lfloor \frac{m}{2} \right\rfloor} \sum_{b=\left\lfloor -\frac{n}{2} \right\rfloor}^{\left\lfloor \frac{n}{2} \right\rfloor} I_{i-a, j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$

pixel of interest



- For the rest of the discussion we will use the following formula for convolution
- In other words we will assume that the kernel is centered on the pixel of interest
- So we will be looking at both preceding and succeeding neighbors

# Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

Output

aw+bx+ey+fz		

# Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	

Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

Output

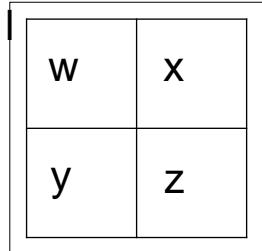
aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz

# Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kerne



Output

$aw+bx+ey+fz$	$bw+cx+fy+gz$	$cw+dx+gy+hz$
$ew+fx+iy+jz$		

# Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

Output

$aw+bx+ey+fz$	$bw+cx+fy+gz$	$cw+dx+gy+hz$
$ew+fx+iy+jz$	$fw+gx+jy+kz$	

# Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

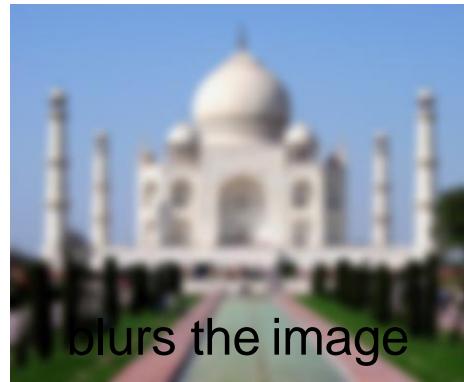
Output

$aw+bx+ey+fz$	$bw+cx+fy+gz$	$cw+dx+gy+hz$
$ew+fx+iy+jz$	$fw+gx+jy+kz$	$gw+hx+ky+Az$

# Example of kernel: Blur



$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$



# Example of kernel: Edge detection

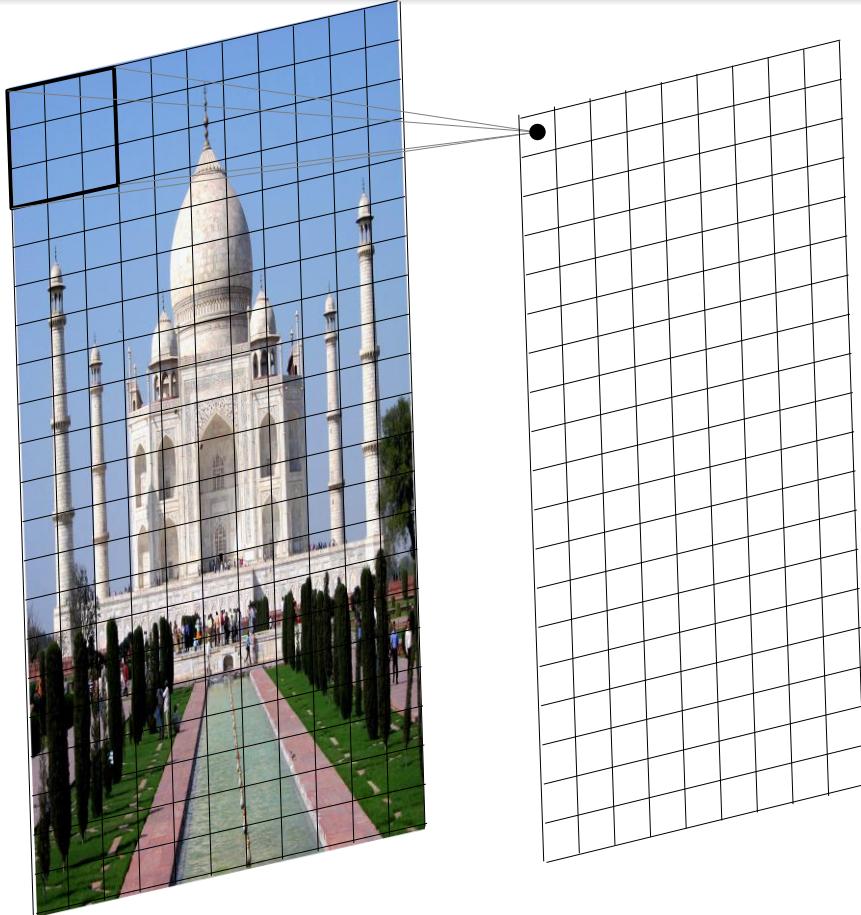


$$\begin{array}{ccc} * & \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} & = \end{array}$$



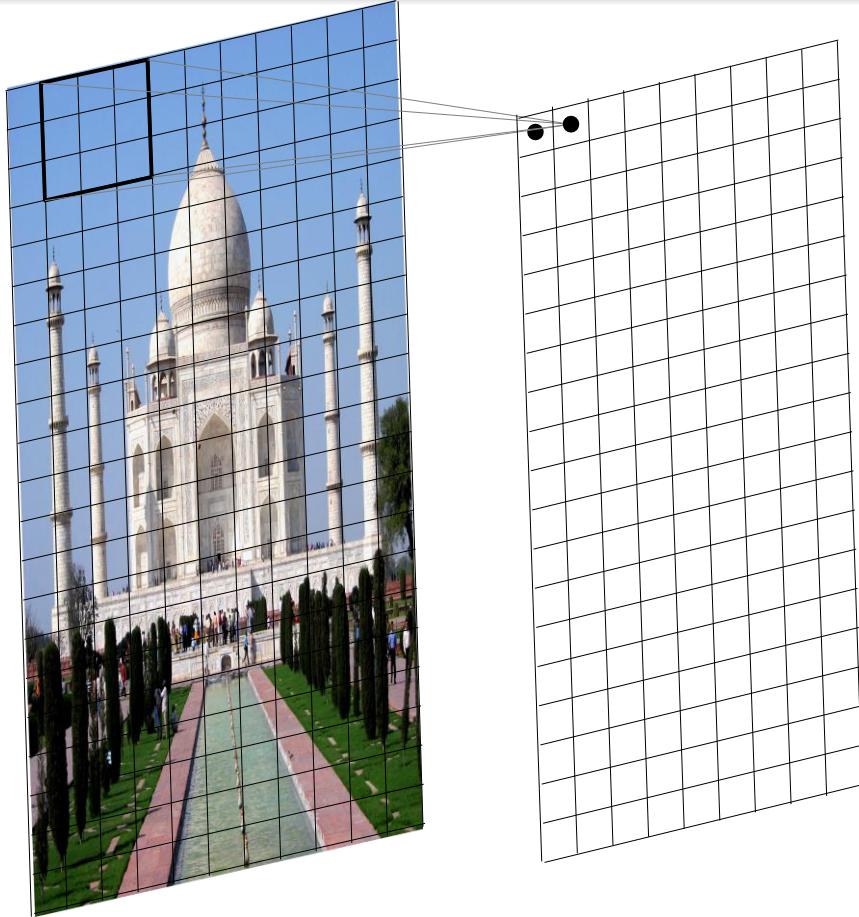
edges

# Convolution



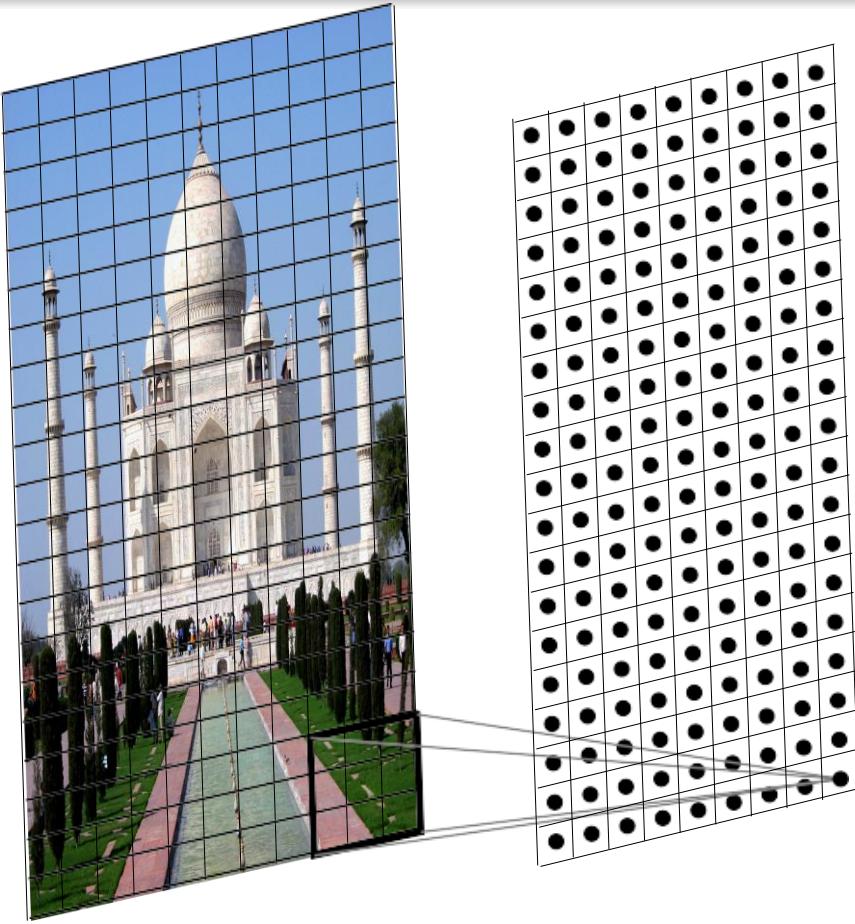
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

# Convolution

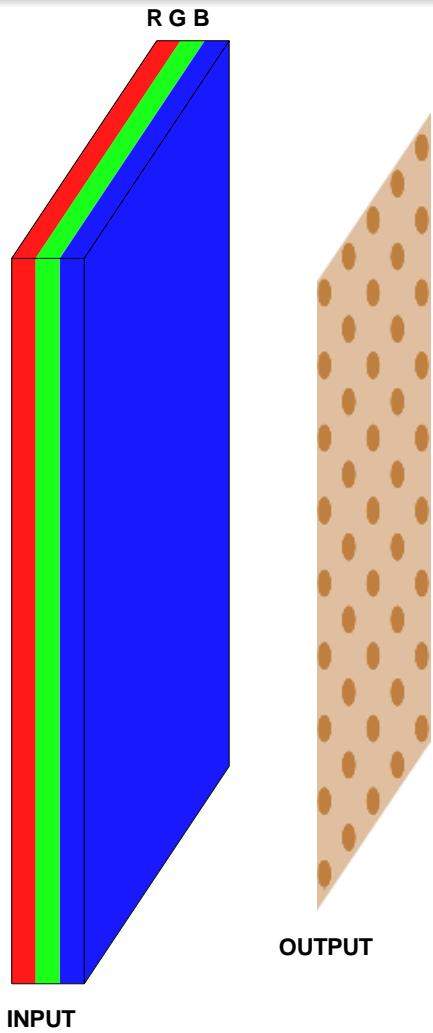


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

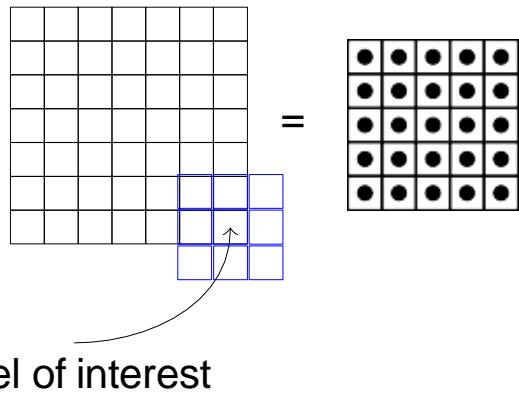
# 2D convolutions applied to images



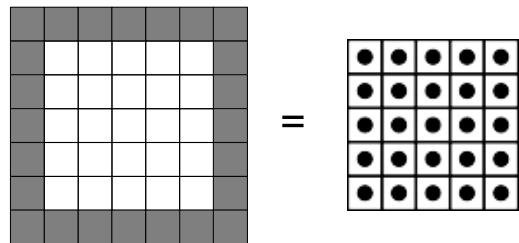
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.



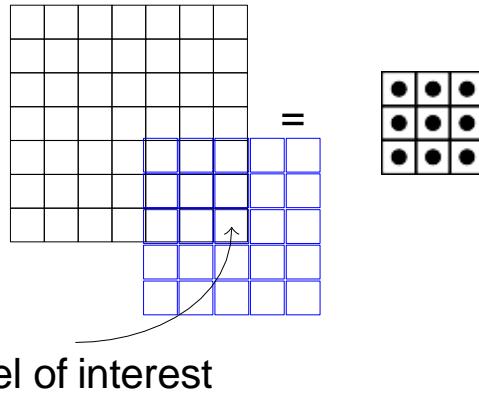
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- We will assume that the filter always extends to the depth of the image
  
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)
- As a result the output will be 2D (only width and height, no depth)
- Once again we can apply multiple filters to get multiple feature maps



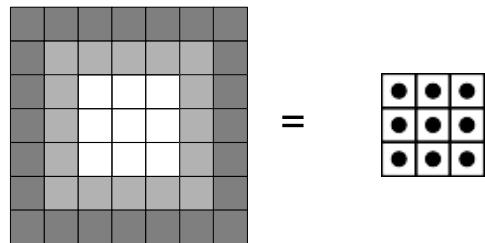
- Let us compute the dimension ( $W_2, H_2$ ) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary



- Let us compute the dimension ( $W_2, H_2$ ) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels



- Let us compute the dimension ( $W_2, H_2$ ) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a  $5 \times 5$  kernel
- We have an even smaller output now



*In general,*  $W_2 = W_1 - F + 1$

$$H_2 = H_1 - F + 1$$

*We will refine this formula further*

- Let us compute the dimension ( $W_2, H_2$ ) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a  $5 \times 5$  kernel
- We have an even smaller output now

$$\begin{array}{ccccccccc}
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
0 & & & & & & & & 0 \\
0 & & & & & & & & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array} = 
 \begin{array}{ccccccccc}
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet
\end{array}$$

- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad  $P = 1$  with a  $3 \times 3$  kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

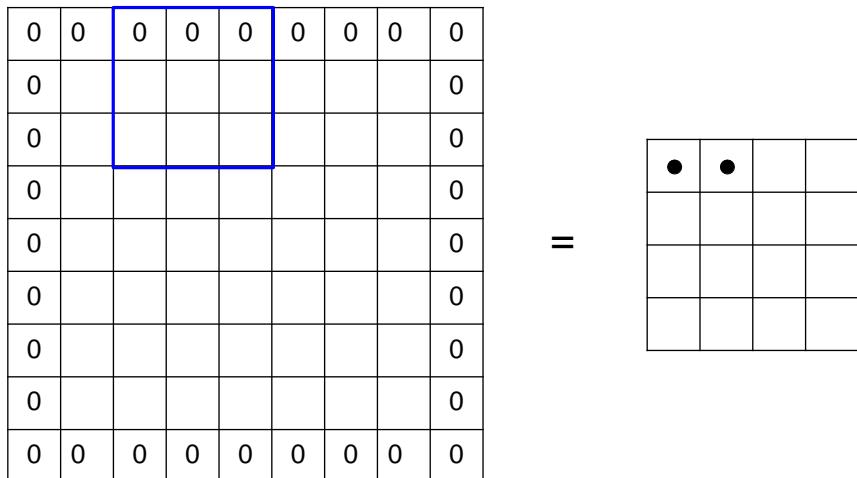
We now have,  $W_2 = W_1 - F + 2P + 1$   $H_2 = H_1 - F + 2P + 1$

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

=

•			

- What does the stride S do?
- It defines the intervals at which the filter is applied (here  $S = 2$ )
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

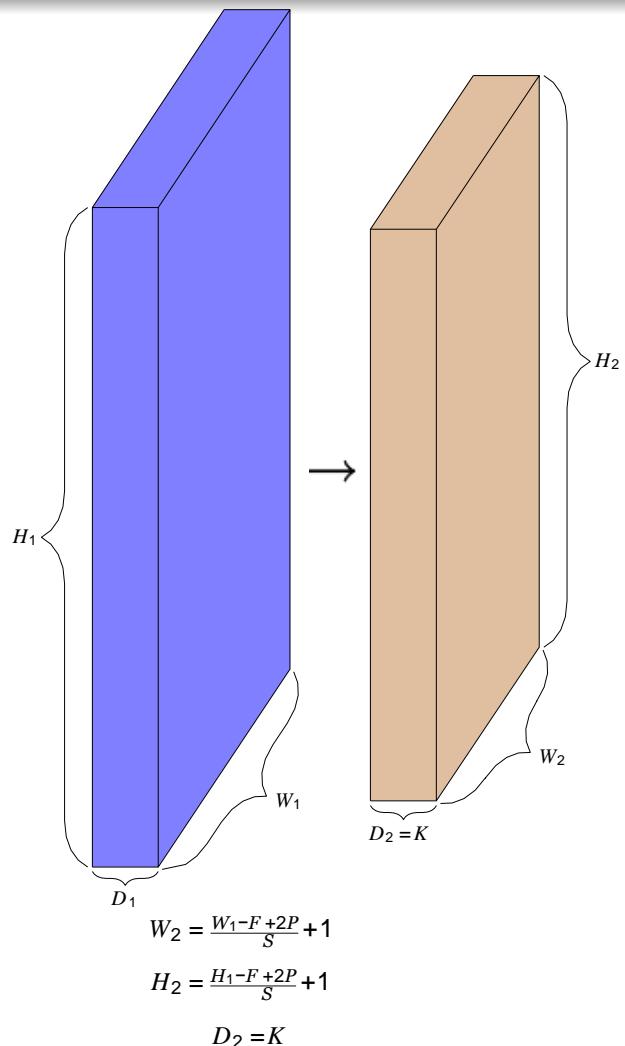


- What does the stride S do?
- It defines the intervals at which the filter is applied (here  $S = 2$ )
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

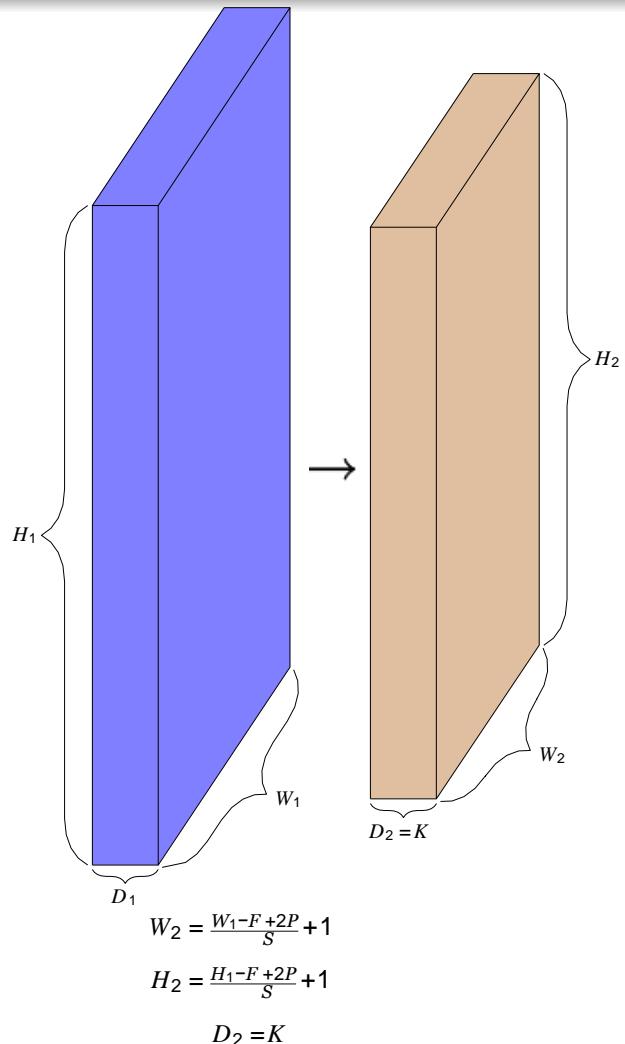
*So what should our final formula look like,*

$$W = \frac{W_1 - F + 2P}{S} + 1$$

$$H = \frac{H_1 - F + 2P}{S} + 1$$

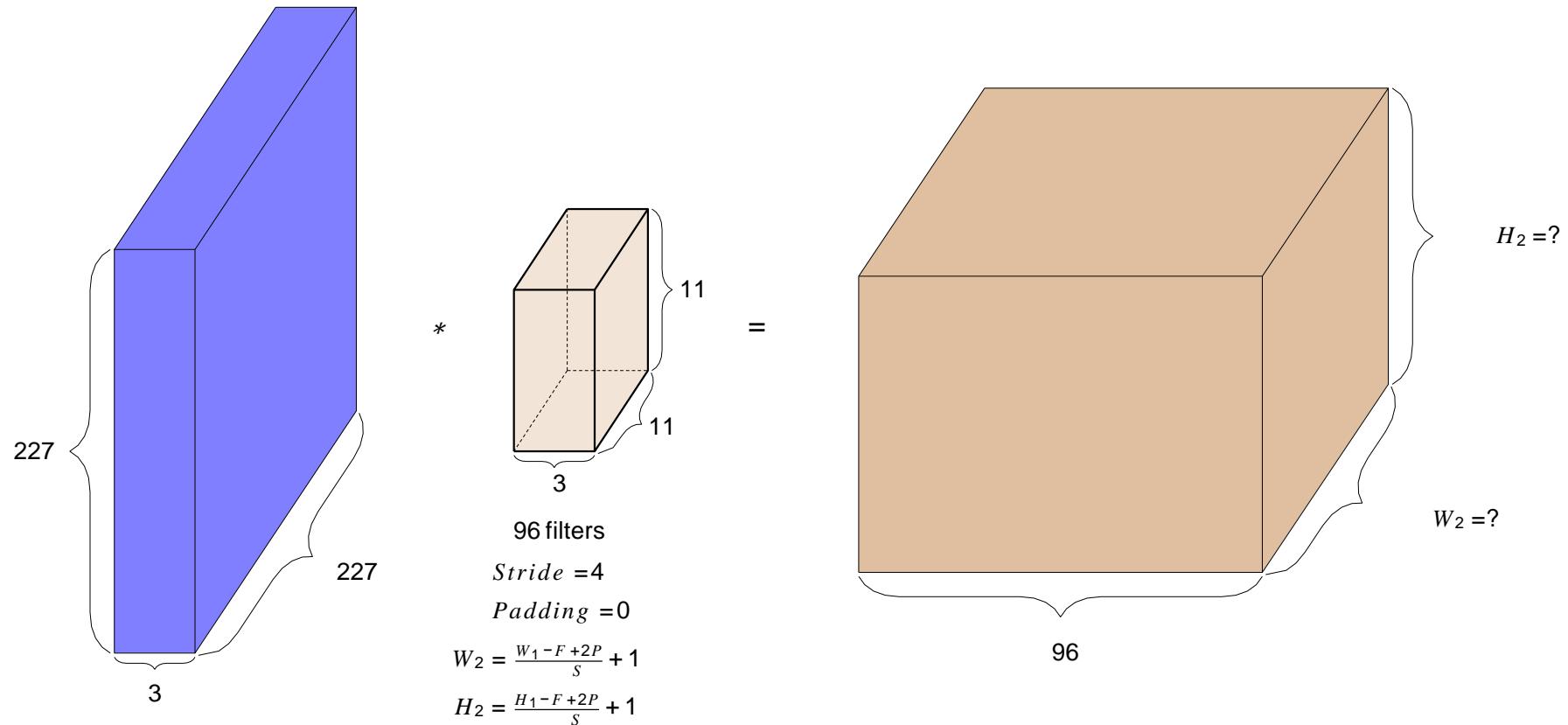


- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.
- $K$  filters will give us  $K$  such 2D outputs

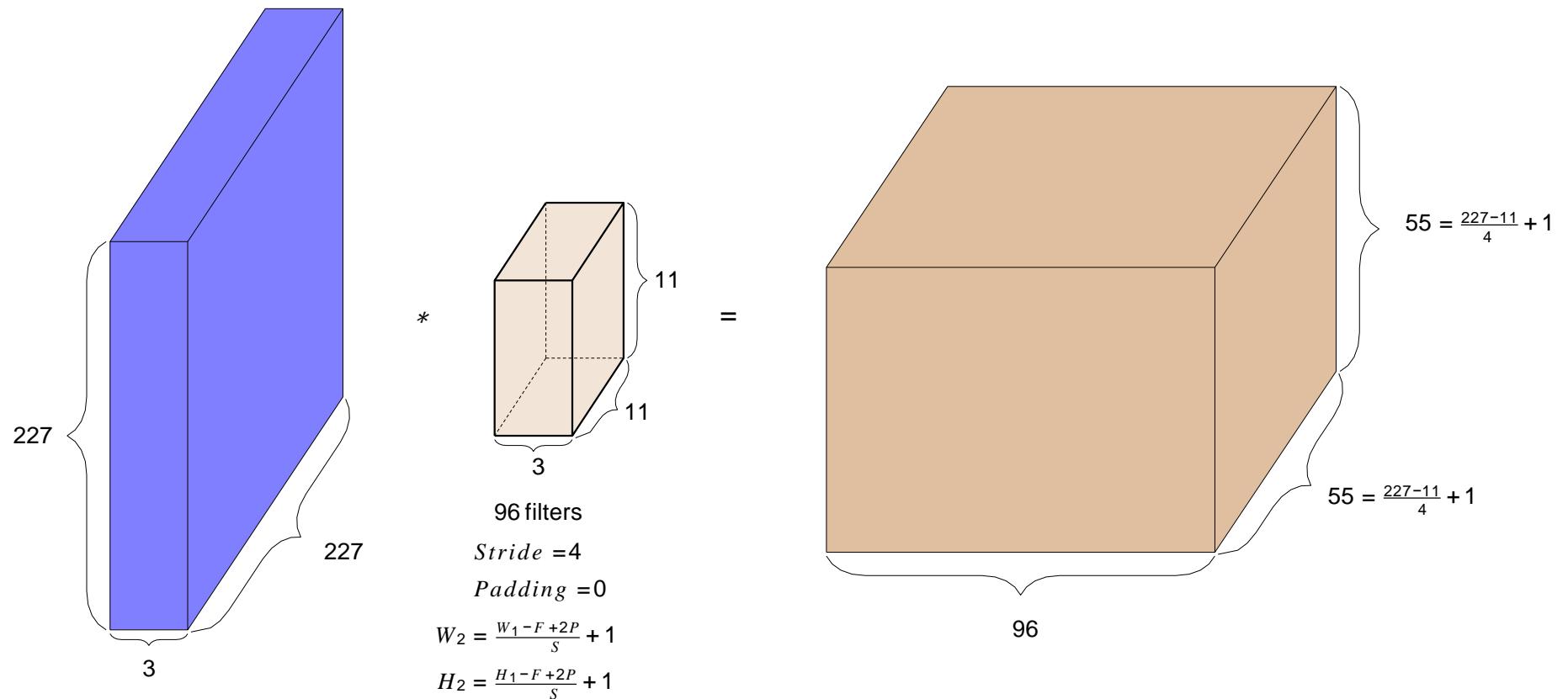


- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.
- $K$  filters will give us  $K$  such 2D outputs
- We can think of the resulting output as  $K \times W_2 \times H_2$  volume
- Thus  $D_2 = K$

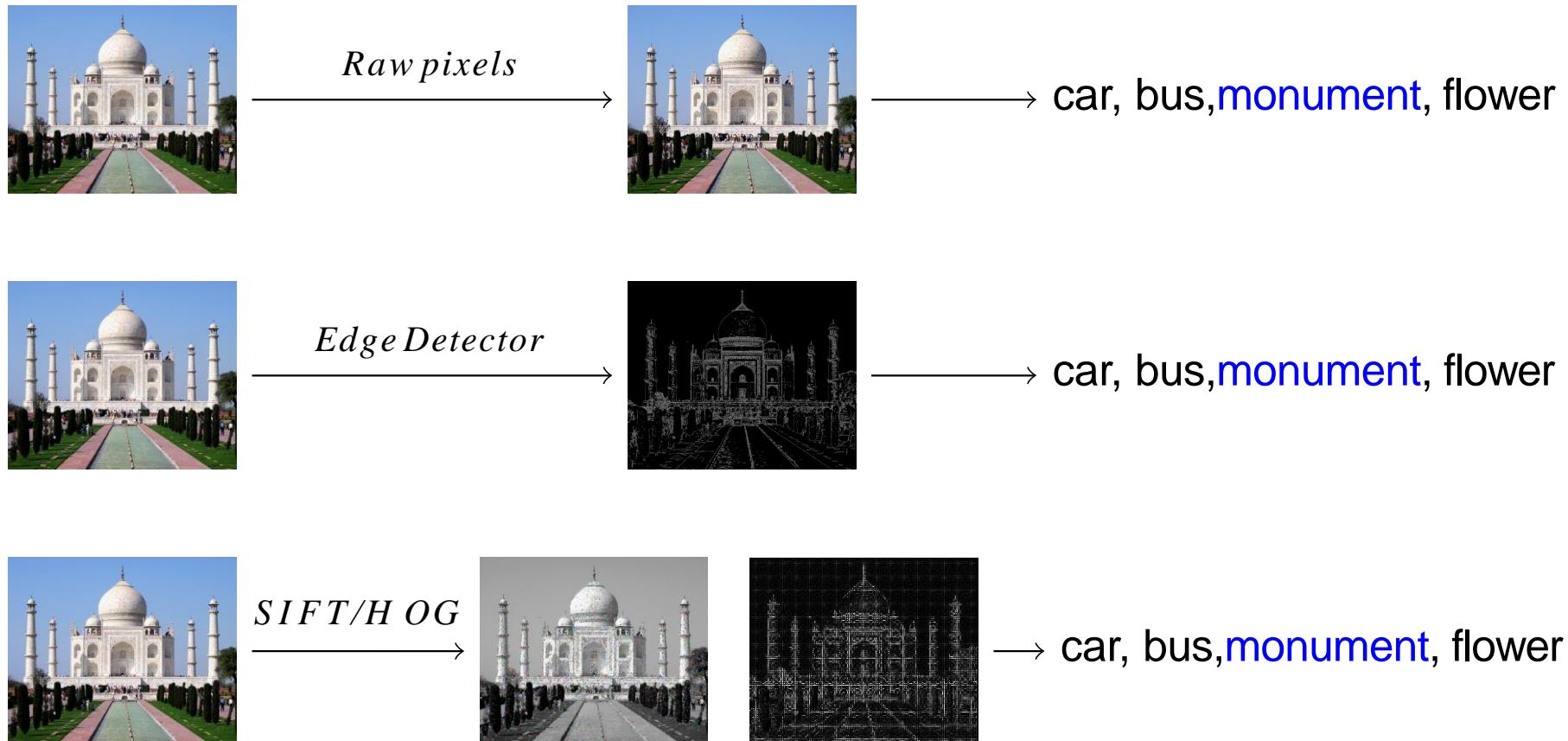
# Example

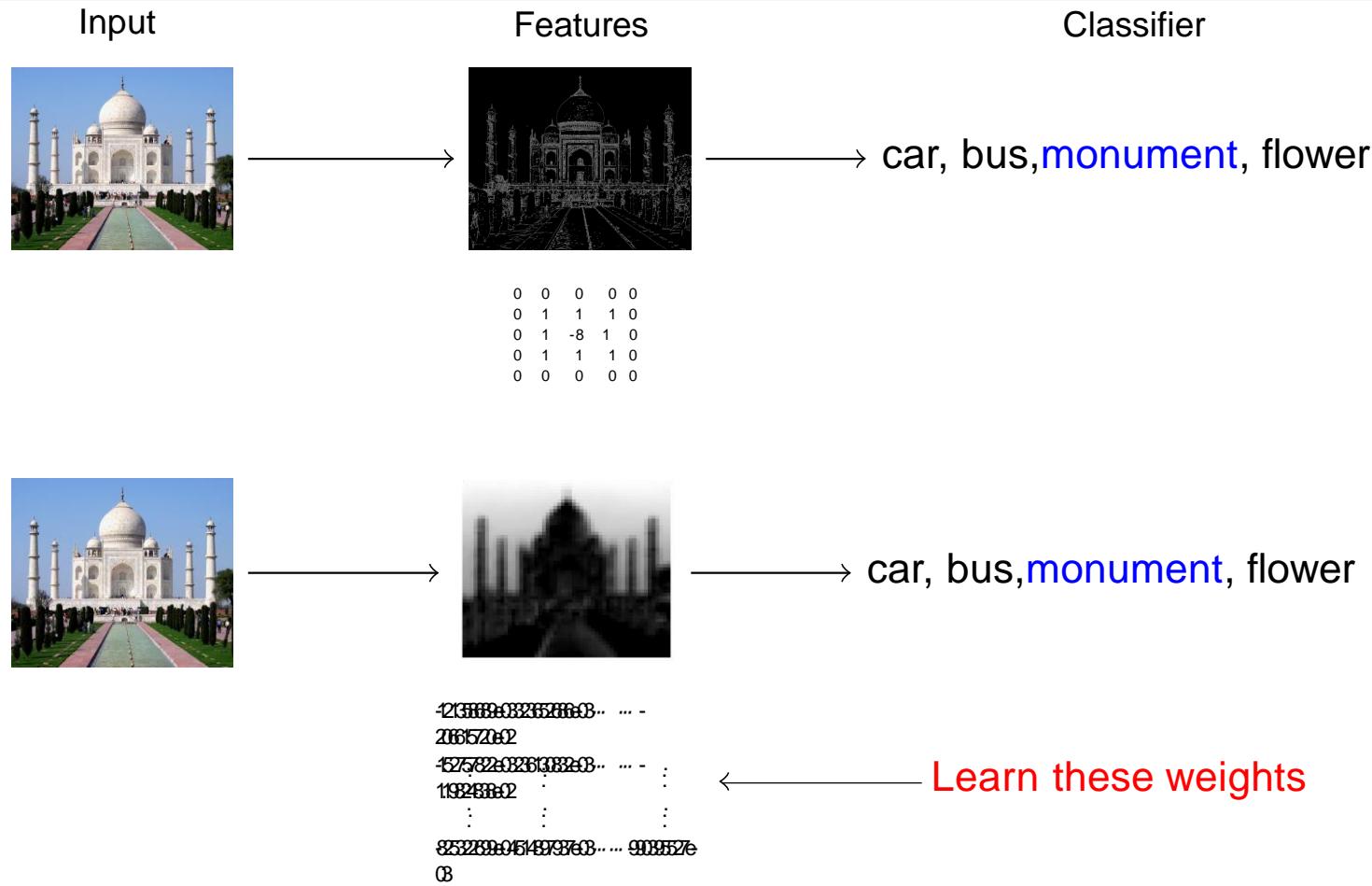


# Example

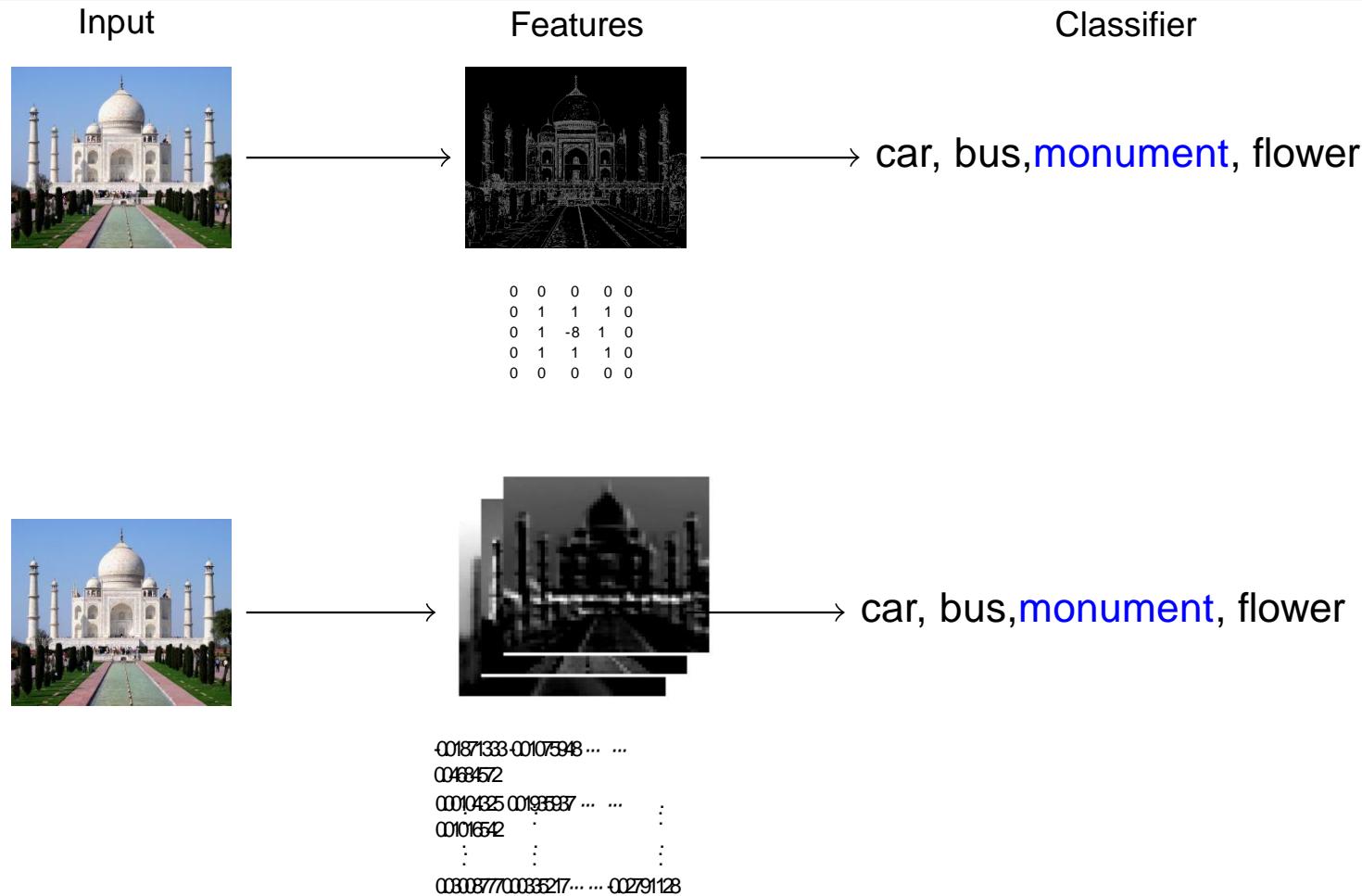


# Features



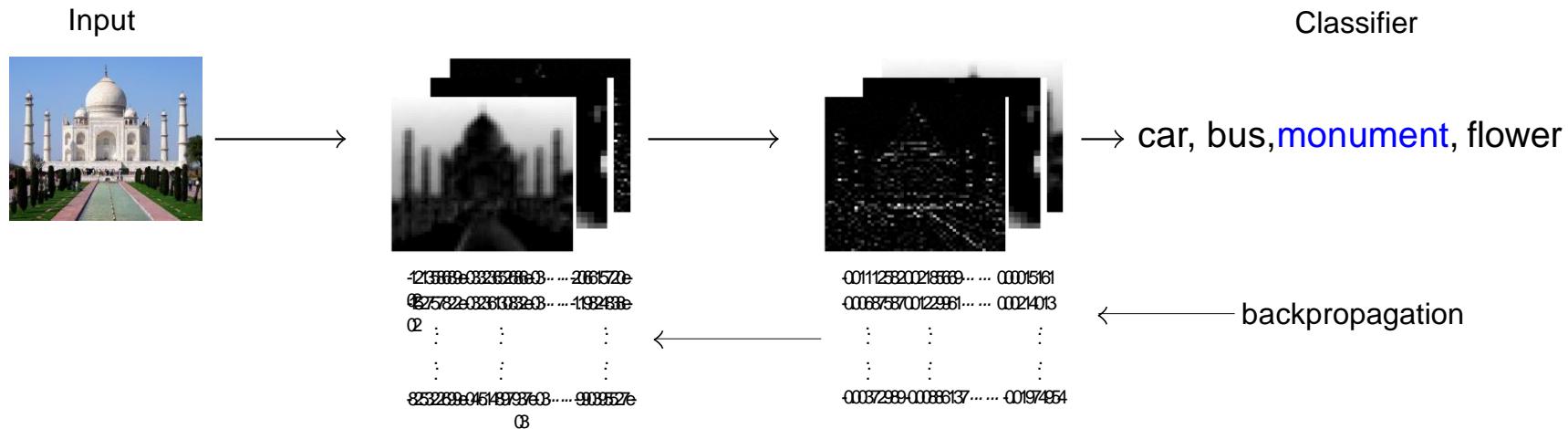


■ Instead of using handcrafted kernels such as edge detectors **can we learn meaningful kernels/filters in addition to learning the weights of the classifier?**

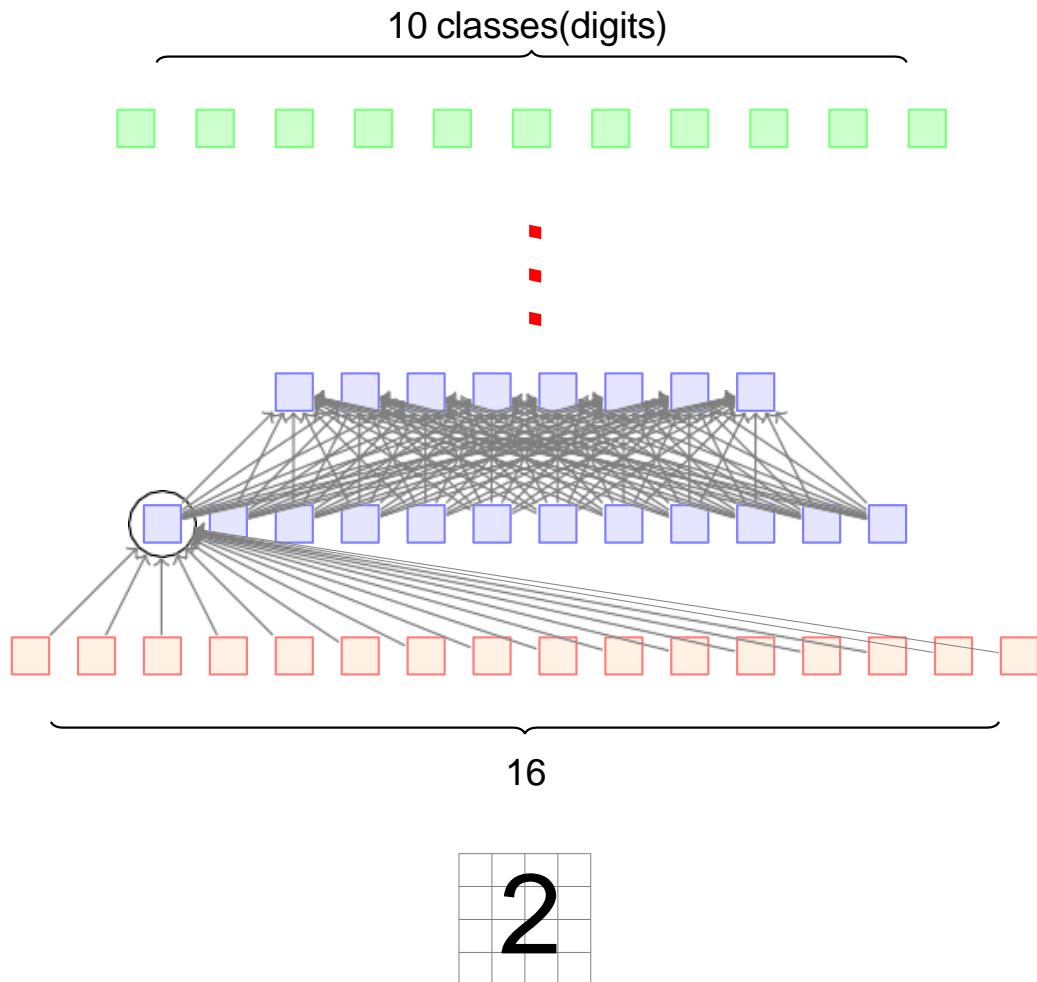


- **Even better:** Instead of using handcrafted kernels (such as edge detectors) **can we learn multiple meaningful kernels/filters in addition to learning the weights of the classifier?**

# Convolutional Neural Network

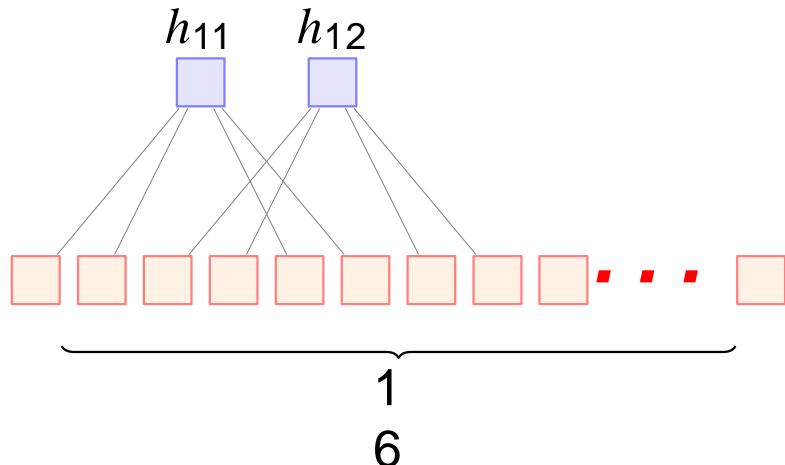


- Can we learn multiple **layers** of meaningful kernels/filters in addition to learning the weights of the classifier?
    - Yes, we can !
    - Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)
    - Such a network is called a Convolutional Neural Network.



- This is what a regular feed-forward neural network will look like
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of  $h_{11}$
- Contrast this to what happens in the case of convolution

# Sparse Connectivity

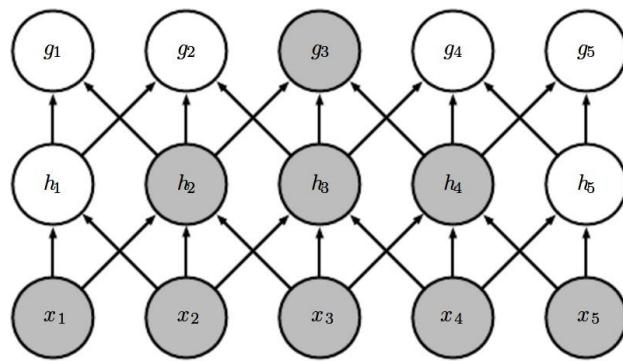


A 4x4 input matrix on the left contains a large black '2' and several red dots. It is multiplied by a 3x3 weight matrix in the middle, which has blue dots in its top-left 2x2 submatrix. The result is a single blue square on the right.

- Only a few local neurons participate in the computation of  $h_{11}$
- For example, only pixels 1, 2, 5, 6 contribute to  $h_{11}$
- The connections are much sparser
- We are taking advantage of the structure of the image (interactions)
- This **sparse connectivity** reduces the number of parameters in the model

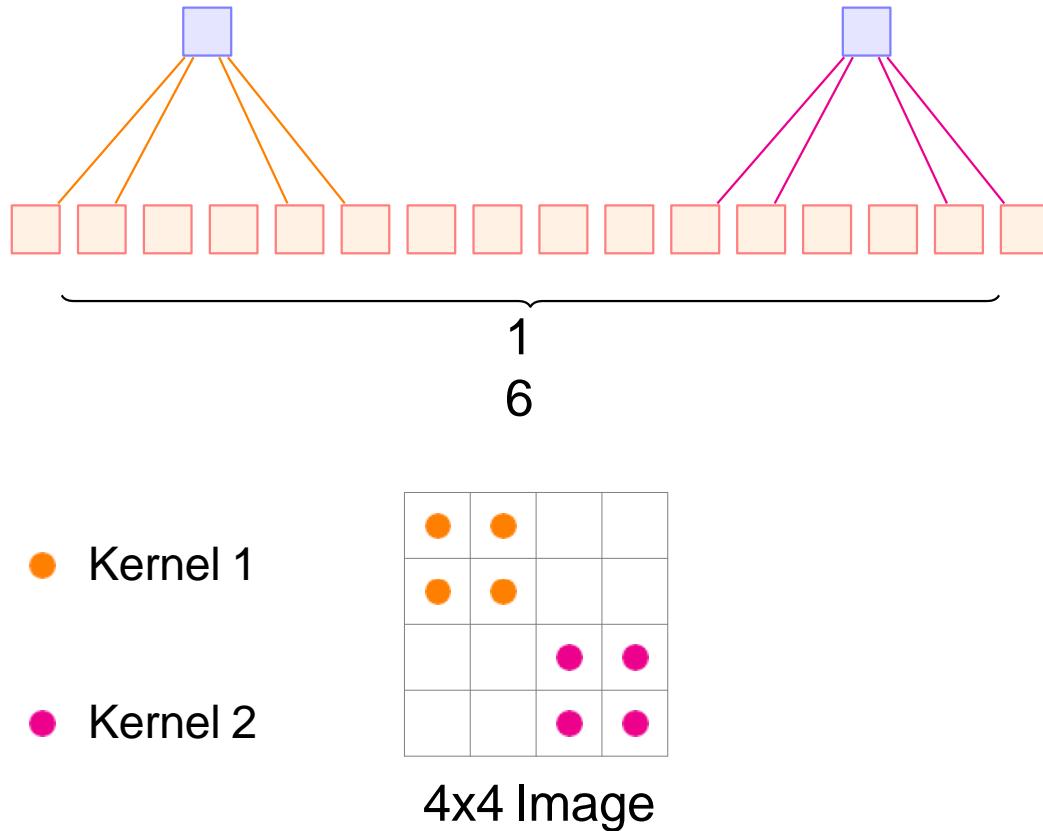
# Sparse Connectivity

- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons ( $x_1$  &  $x_5$ ) do not interact in *layer 1*
- But they indirectly contribute to the computation of  $g_3$  and hence interact indirectly



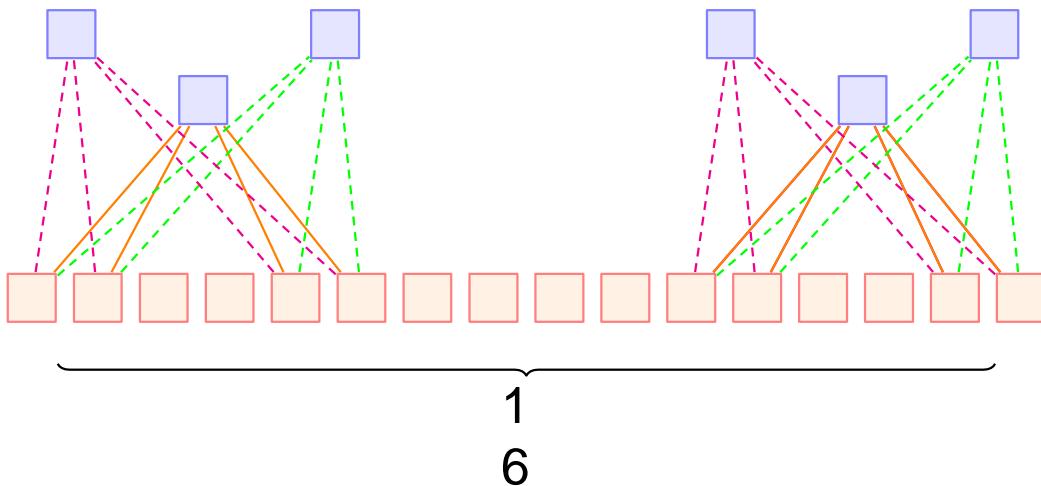
\*Goodfellow-et-al-2016

# Weight-sharing

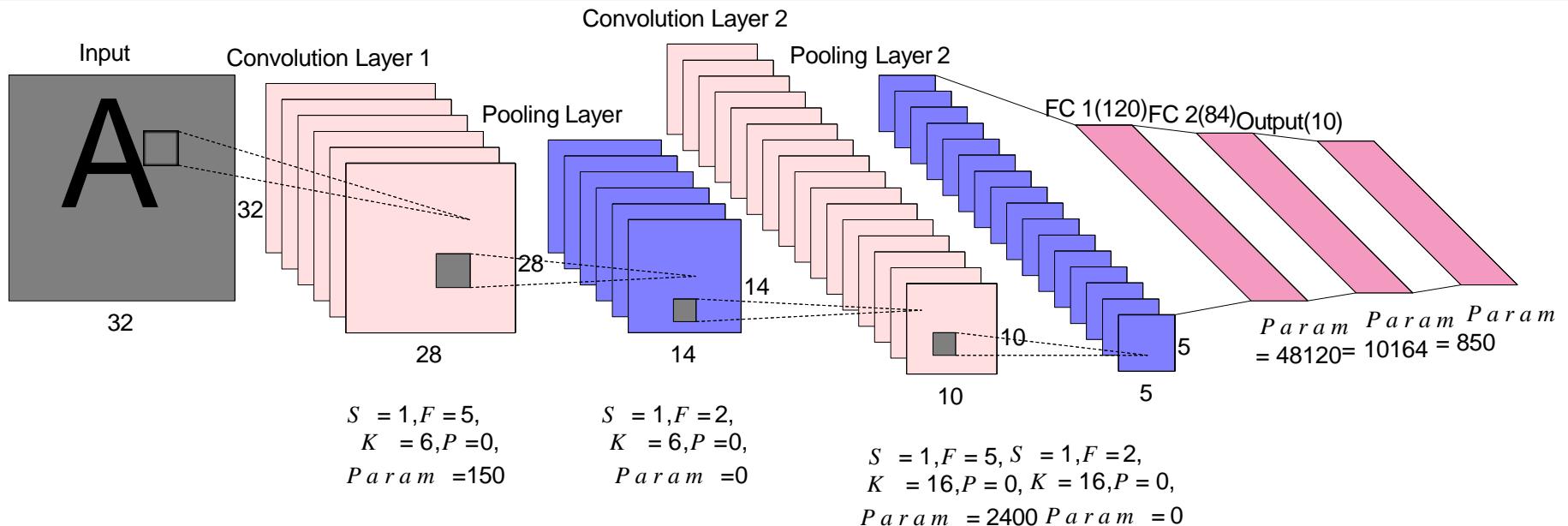


- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?
- Imagine that we are trying to learn a kernel that detects edges
- Shouldn't we be applying the same kernel at all the portions of the image?

# Weight-sharing

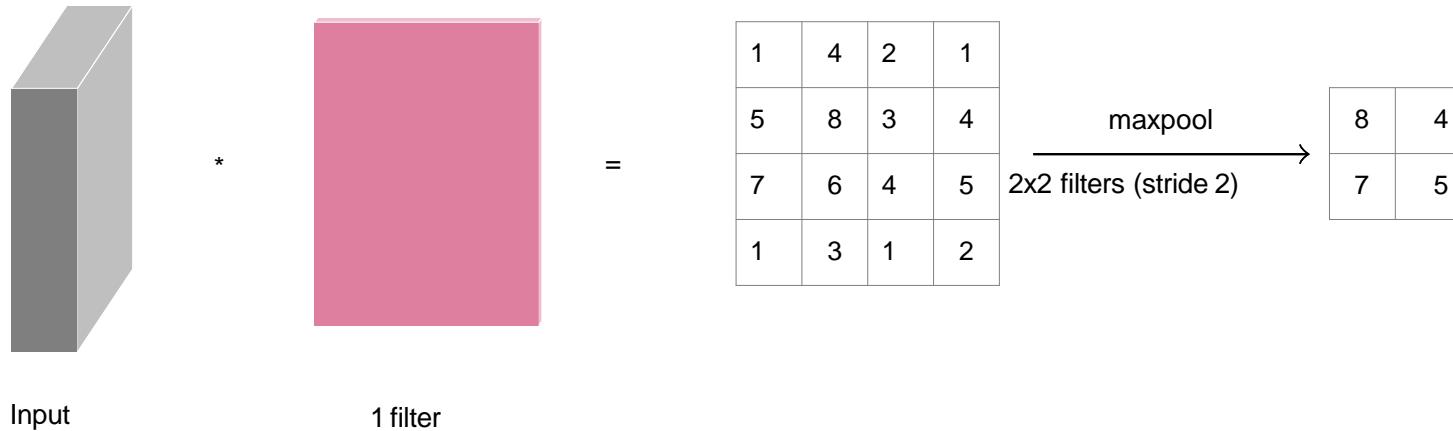


- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image
- This is called “weight sharing”

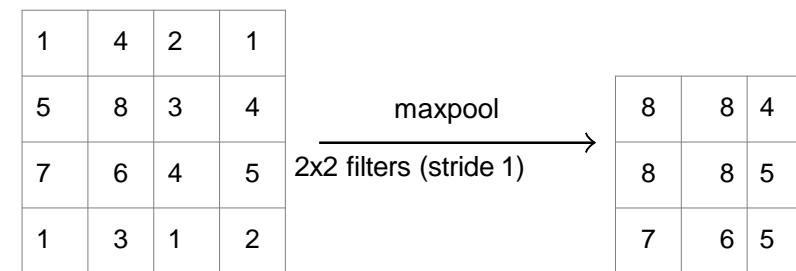
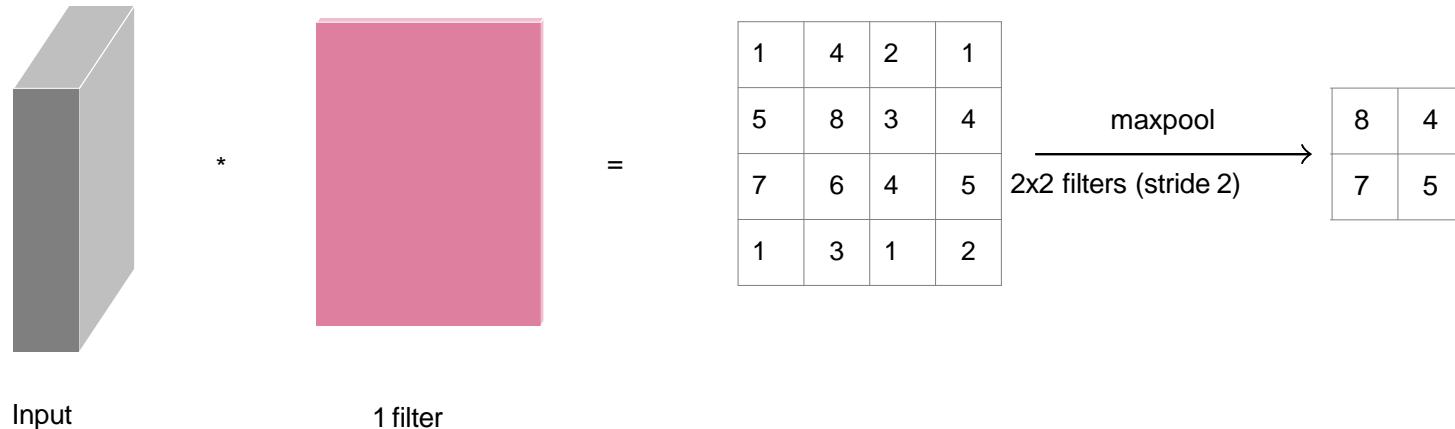


- It has alternate convolution and pooling layers
- What does a pooling layer do?
- Let us see

# Pooling



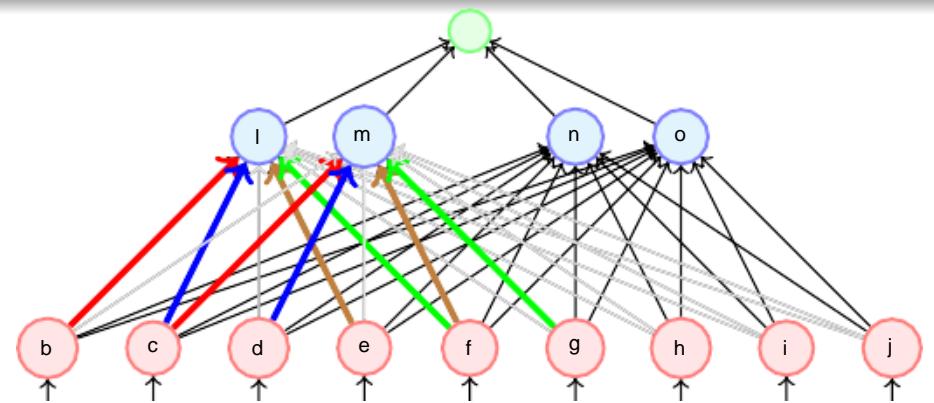
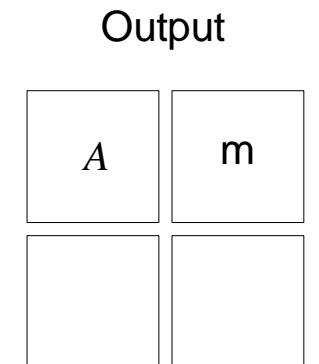
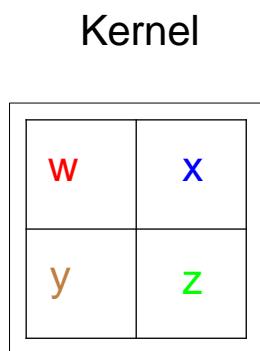
# Pooling



- Instead of max pooling we can also do average pooling

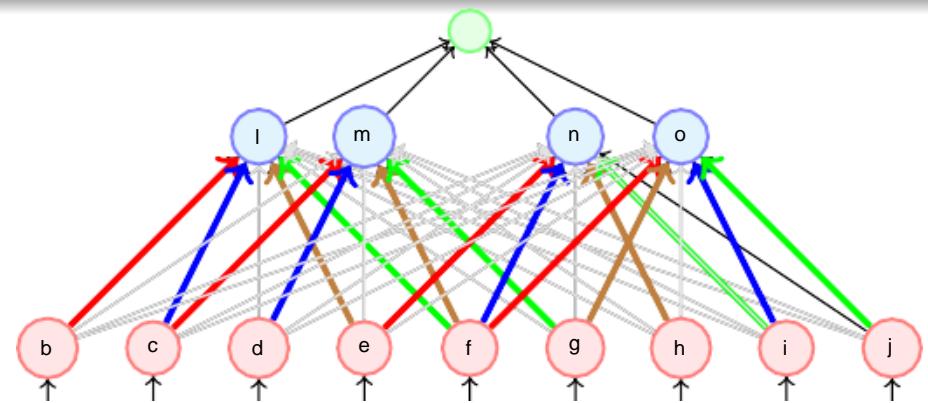
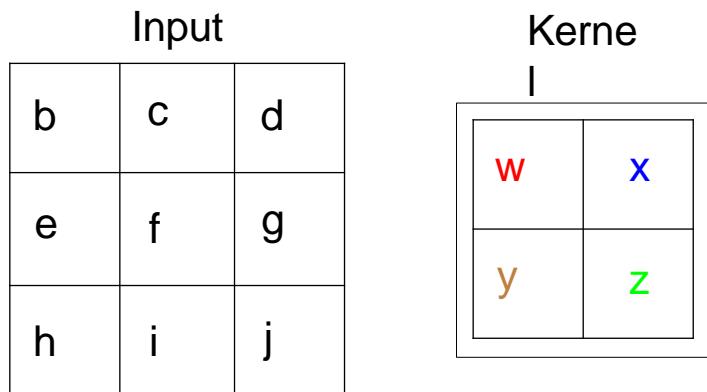
# Training CNN

Input		
b	c	d
e	f	g
h	i	j



- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

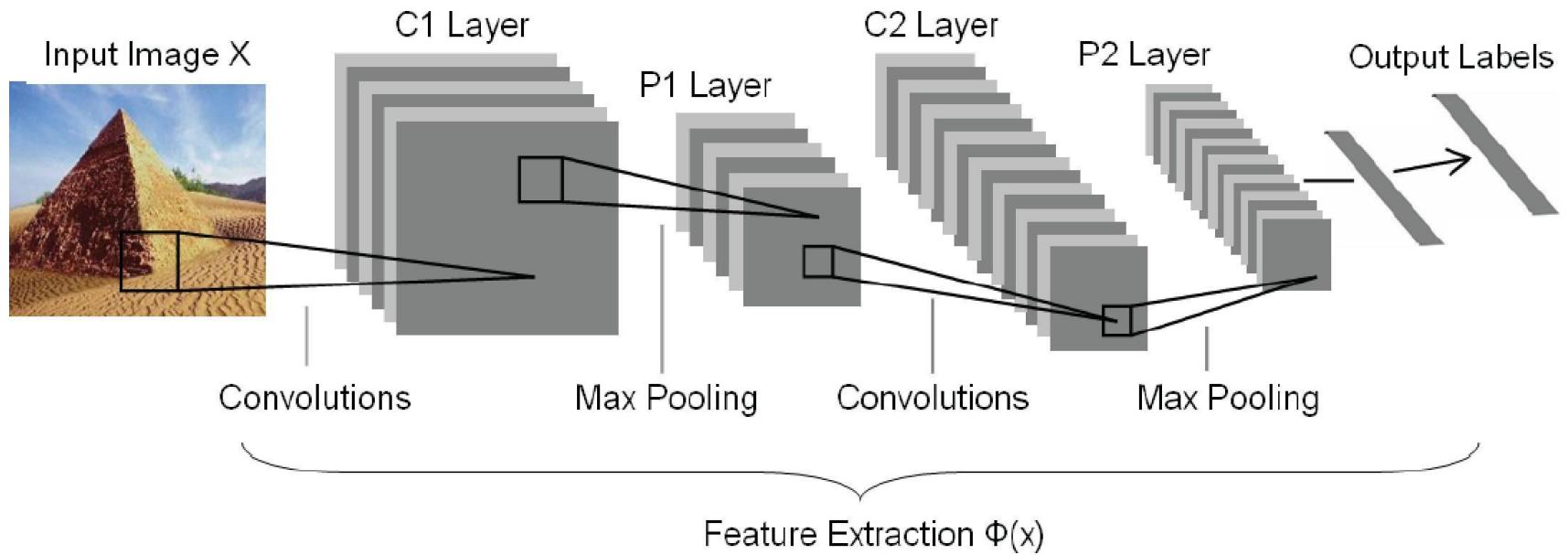
# Training CNN



- We can thus train a convolution neural network using backpropagation by thinking of it as a feedforward neural network with sparse connections
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

# Convolutional Neural Nets for Image Recognition

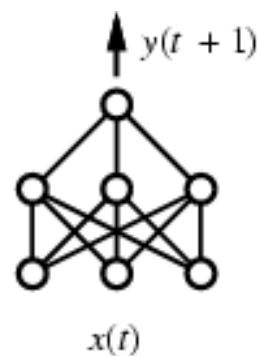
[Le Cun, 1992]



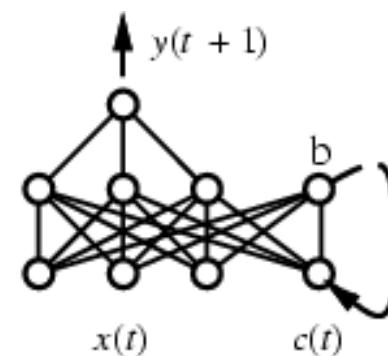
- specialized architecture: mix different types of units, not completely connected, motivated by primate visual cortex
- many shared parameters, stochastic gradient training
- very successful! now many specialized architectures for vision, speech, translation, ...

# Recurrent Networks: Time Series

- Suppose we want to predict next state of world
  - and it depends on history of unknown length
  - e.g., robot with forward-facing sensors trying to predict next sensor reading as it moves and turns
- Idea: use hidden layer in network to capture state history



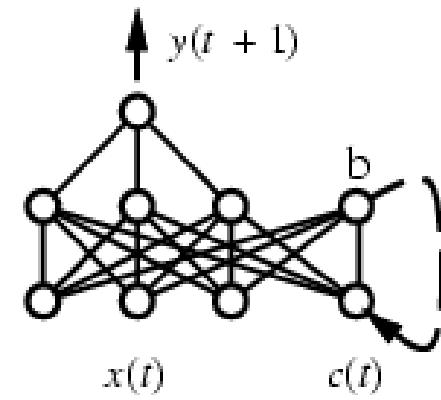
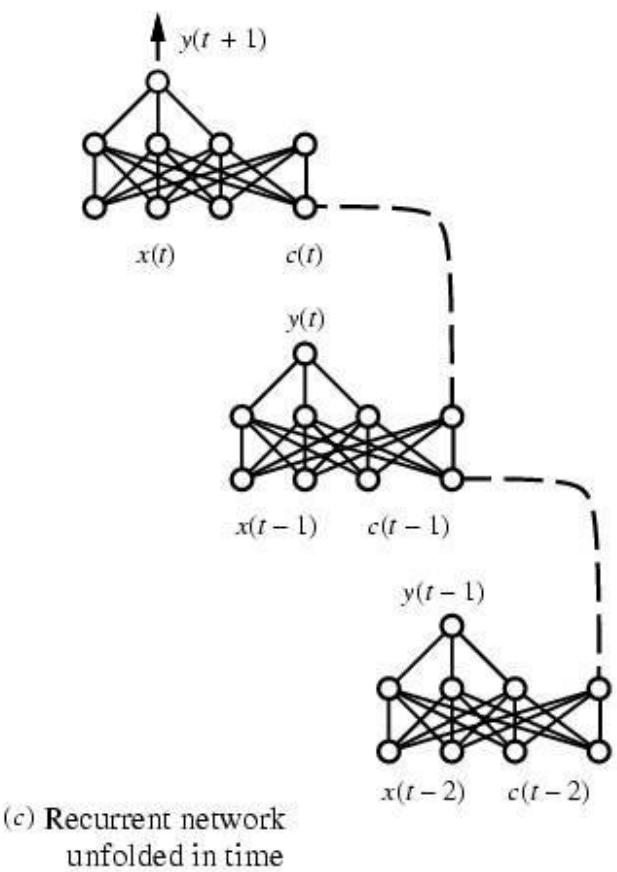
(a) Feedforward network



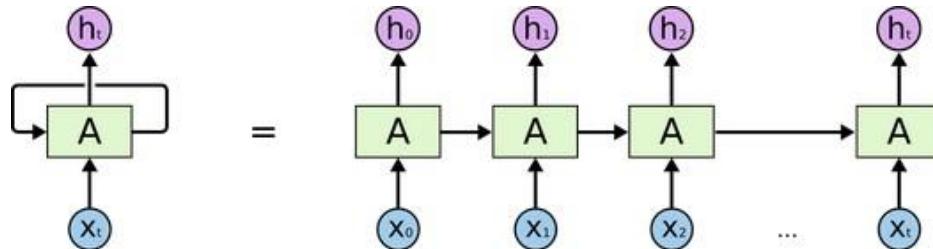
(b) Recurrent network

# Recurrent Networks on Time Series

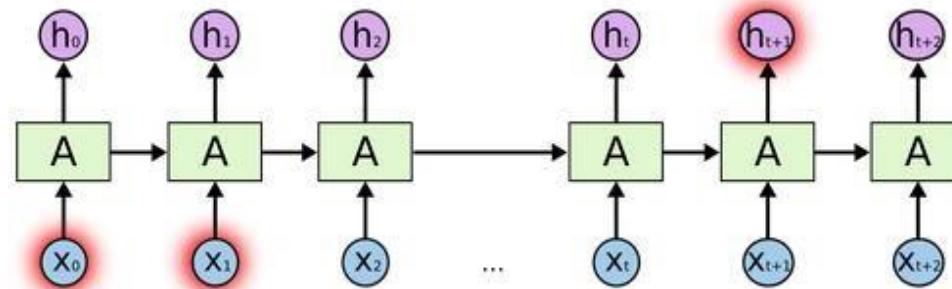
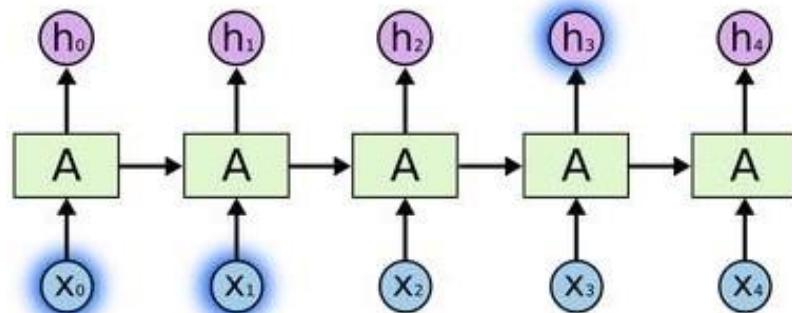
How can we train recurrent net??



# Recurrent Neural Networks



An unrolled recurrent neural network.



## Why RNNs:

Traditional neural networks can't have persistence.

For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

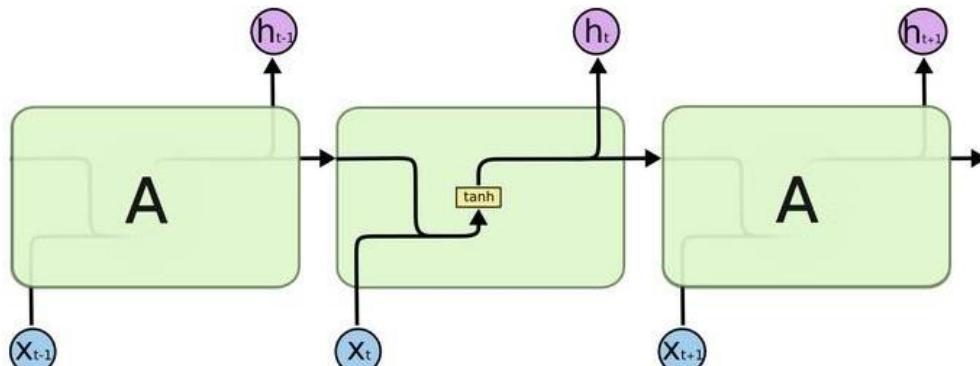
RNNs address this issue. They are networks with loops in them, allowing information to persist.

## Problem of Long Term Dependency:

Consider trying to predict the last word in the text "I grew up in France... I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

# Long Short Term Memory Networks

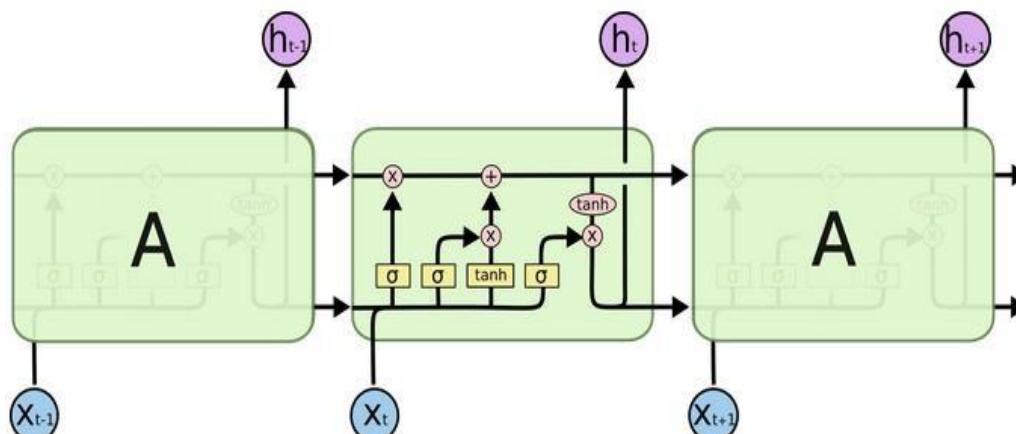


The repeating module in a standard RNN contains a single layer.

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

- Forget gate layer
- Input gate layer
- Control gate layer
- Output gate layer



The repeating module in an LSTM contains four interacting layers.

The LSTM have the ability to remove or add information to the cell state, carefully regulated by structures called **gates**.

The sigmoid layer outputs numbers between 0 and 1, describing how much of each component should be let through. A value of 0 means “let nothing through,” while a value of 1 means “let everything through!”

# Artificial Neural Networks: Summary

- Highly non-linear regression/classification
- Hidden layers learn intermediate representations
- Potentially millions of parameters to estimate
- Stochastic gradient descent, local minima problems
- Deep networks have produced real progress in many fields
  - computer vision
  - speech recognition
  - mapping images to text
  - recommender systems
  - ...
- They learn very useful non-linear representations

# Good References for understanding Neural Network

Mitesh Khapra

<https://www.youtube.com/watch?v=yw8xwS15Pf4>

Visualization of CNN

<https://www.youtube.com/watch?v=cNBBNAxC8I4>



**BITS** Pilani  
Pilani Campus

# Instance-based Learning

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

# Topics to be covered

---

- Instance based learning
  - K-Nearest Neighbour Learning
  - Locally Weighted Regression (LWR)  
Learning
  - Radial Basis Functions
-

# Instance Based Learning

Key idea: just store all training examples  $\langle x_i, f(x_i) \rangle$

Nearest neighbor:

- Given query instance  $x_q$ , first locate nearest training example  $x_n$ , then estimate  $f^*(x_q) = f(x_n)$

K-nearest neighbor:

- Given  $x_q$ , take vote among its k nearest neighbors (if discrete-valued target function)
- Take mean of f values of k nearest neighbors (if real-valued)  $f^*(x_q) = \sum_{i=1}^k f(x_i)/k$

# k-Nearest Neighbor Classifier

---

- Nearest Neighbour classifier is an instance based classifier
- ‘lazy learning’, as learning is postponed until a new instance is encountered
- Constructs a local approximation to the target function, applicable in the neighbourhood of new instance
- Suitable in cases where target function is complex over the entire input space, but easily describable in local approximations
- Real world applications found in recommendation systems (amazon).
- Caveat is the high cost of classification, which happens at the time of processing rather than before hand (there’s no training phase)

# When to Consider Nearest Neighbors

---

- Instances map to points in  $\mathbb{R}^N$
- Less than 20 attributes per instance
- Lots of training data

## Advantages:

- Training is very fast
- Learn complex target functions
- Do not lose information

## Disadvantages:

- Slow at query time
- Easily fooled by irrelevant attributes

# k-Nearest Neighbor Classifier

---

- Considers all instances as members of n-dimensional space
- Nearest neighbours of an instance is determined based on Euclidean distance
- Distance between two n-dimensional instances  $x_i$  and  $x_j$  is given by:

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^n (a_r(x_i) - a_r(x_j))^2}$$

- For NN classifier, target function can be discrete or continuous

# Discrete and Continuous-valued function

- **discrete-valued target function:**

- $f : \mathbb{R}^n \rightarrow V$  where  $V$  is the finite set  $\{v_1, v_2, \dots, v_s\}$
- the target function value is the most common value among the  $k$  nearest training examples

$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^k \delta(v, f(x_i))$$

where  $\delta(a, b) = (a == b)$

- **continuous-valued target function:**

- algorithm has to calculate the mean value instead of the most common value
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k f(x_i)}{k}$$

# k-Nearest Neighbor Classifier

---

Training algorithm:

- For each training example  $(x, f(x))$ , add the example to the list *training\_examples*

Classification algorithm:

- Given a query instance  $x_q$  to be classified,
  - Let  $x_1 \dots x_k$  denote the  $k$  instances from *training\_examples* that are nearest to  $x_q$
  - Return

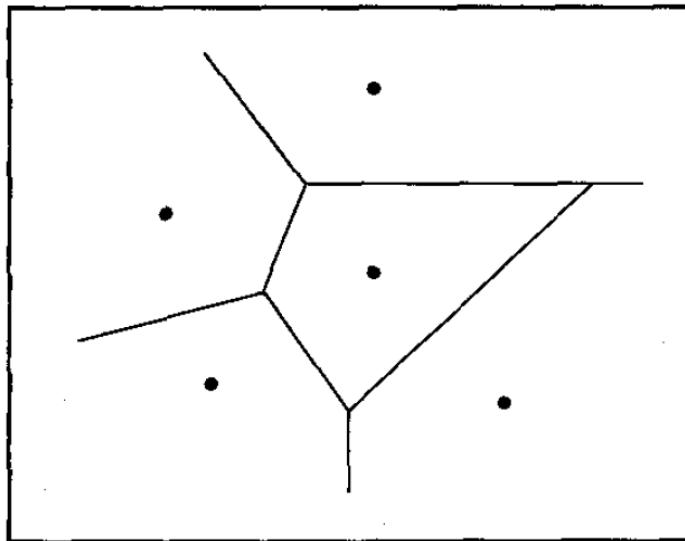
$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^k \delta(v, f(x_i))$$

where  $\delta(a, b) = 1$  if  $a = b$  and where  $\delta(a, b) = 0$  otherwise.

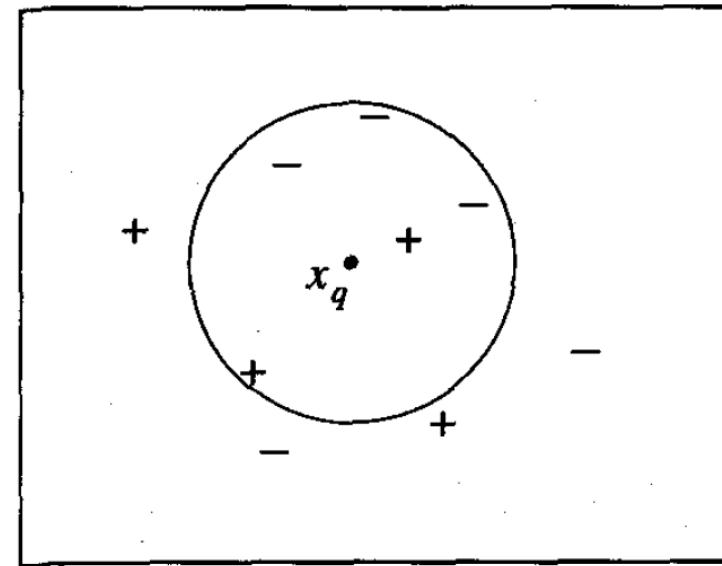
---

\* It can be used for Regression as well.

# k-NN examples



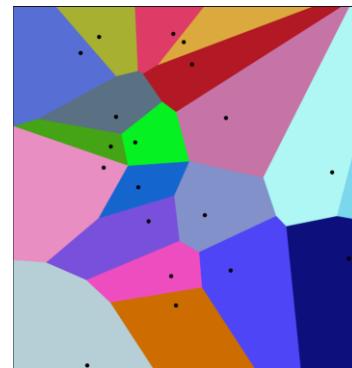
K=1



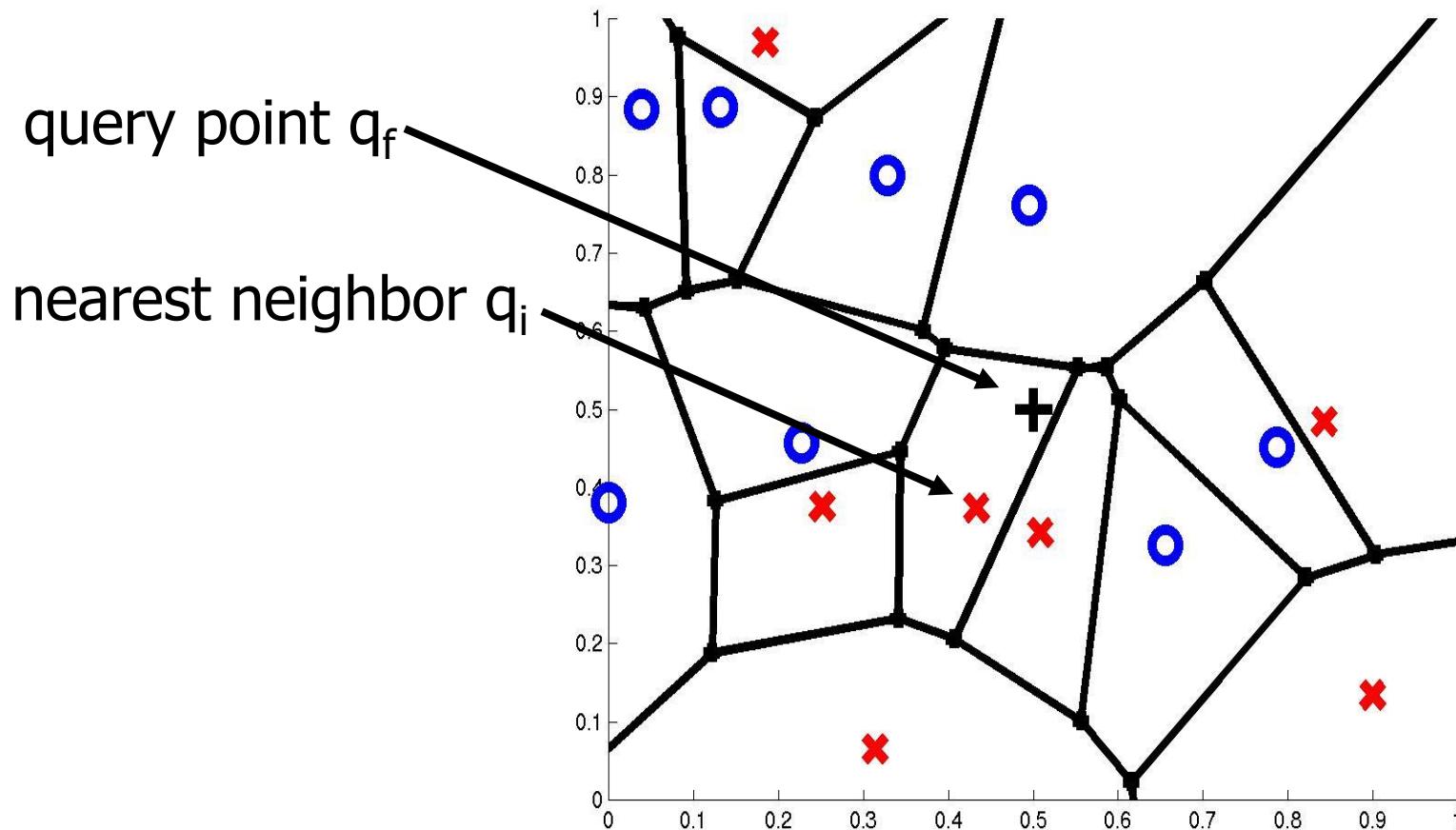
K=5

# Voronoi Diagram

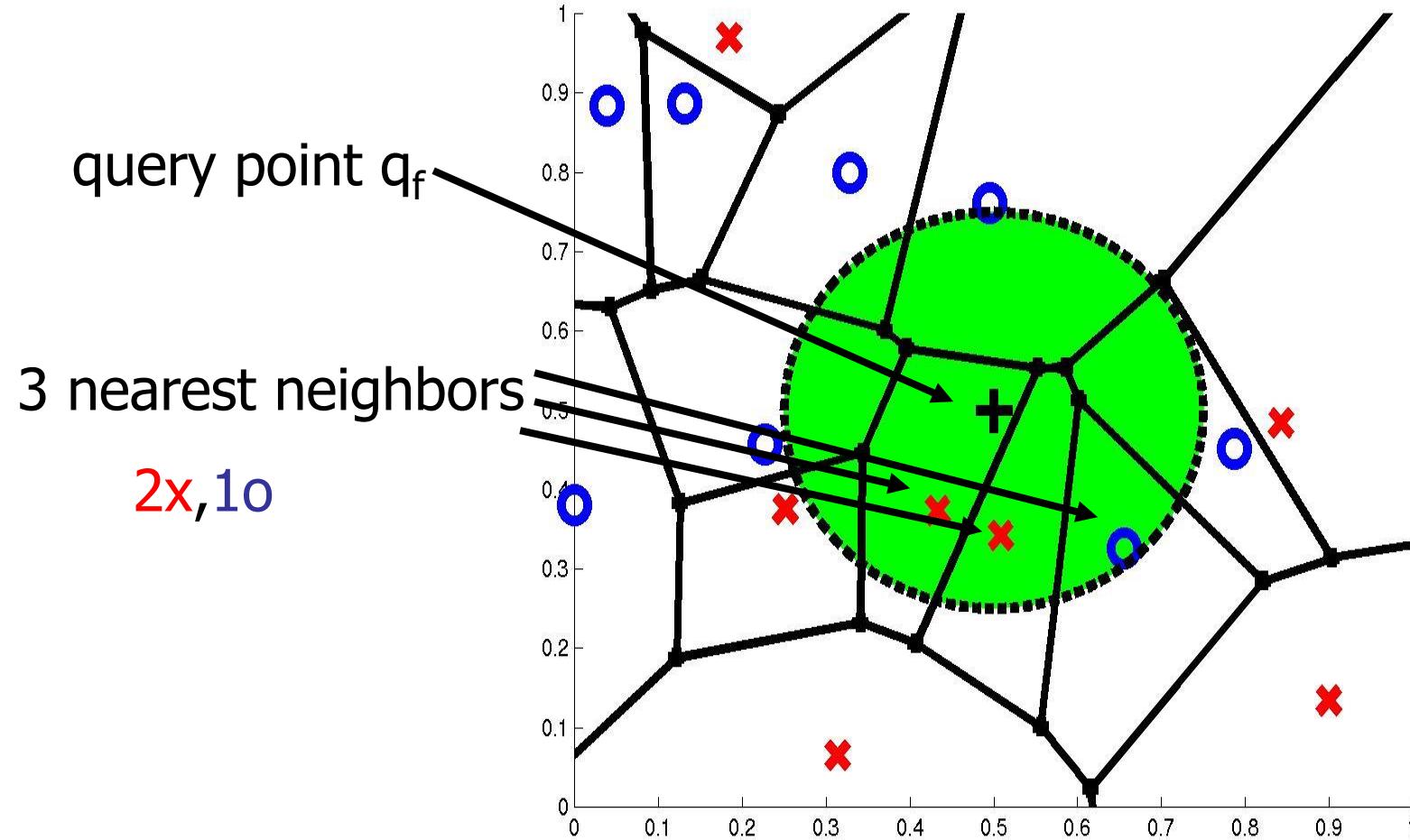
- It is a partition of a plane into regions close to each of a given set of objects.



# Voronoi Diagram



# 3-Nearest Neighbors

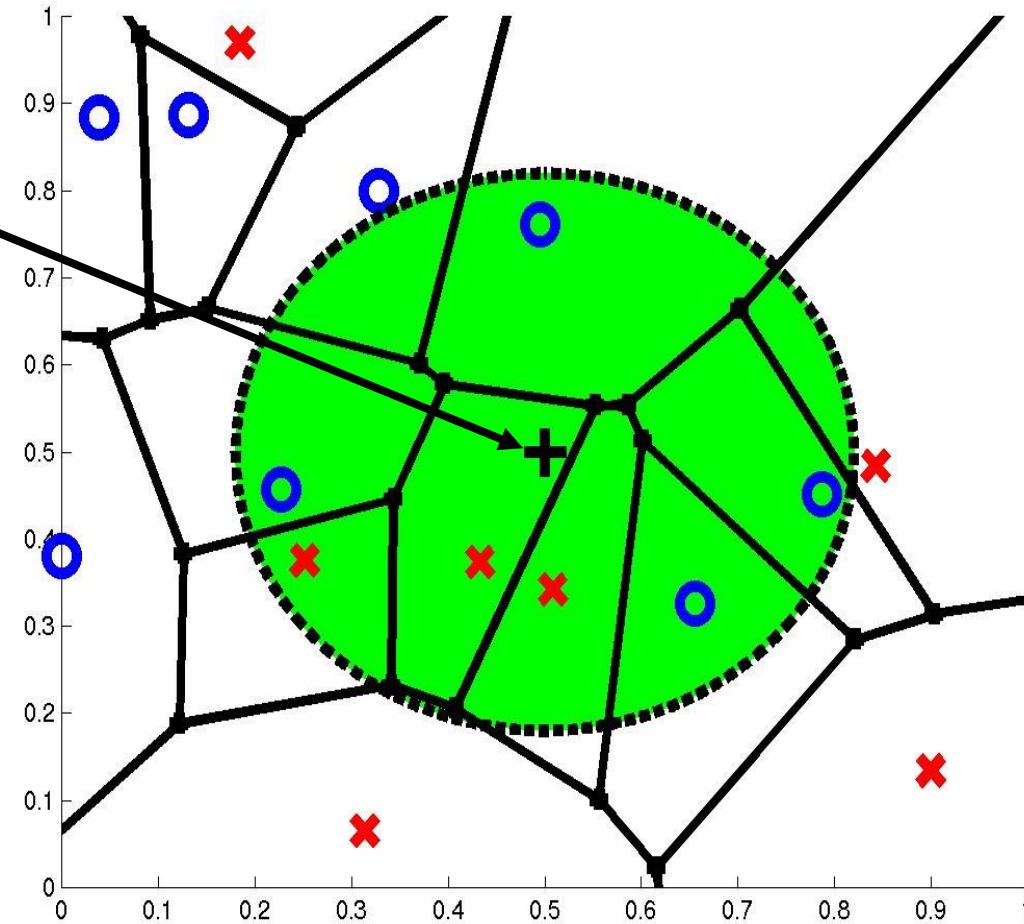


# 7-Nearest Neighbors

query point  $q_f$

7 nearest neighbors

$3x, 4o$



# Distance weighted nearest neighbor

- ➊ contribution of each of the  $k$  nearest neighbors is weighted accorded to their distance to  $x_q$ 
  - discrete-valued target functions

$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^k w_i \delta(v, f(x_i))$$

where  $w_i \equiv \frac{1}{d(x_q, x_i)^2}$  and  $\hat{f}(x_q) = f(x_i)$  if  $x_q = x_i$

- ➋ continuous-valued target function:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i}$$

# Distance Weighted k-NN

---

Give more weight to neighbors closer to the query point

$$- f^*(x_q) = \sum_{i=1}^k w_i f(x_i) / \sum_{i=1}^k w_i \\ ; w_i = K(d(x_q, x_i))$$

and

–  $d(x_q, x_i)$  is the distance between  $x_q$  and  $x_i$

Variation: Instead of only k-nearest neighbors use all training examples (Shepard's method)

# Distance Weighted Average

---

Weighting the data

$$- f^*(x_q) = \sum_i f(x_i) K(d(x_i, x_q)) / \sum_i K(d(x_i, x_q))$$

Relevance of a data point  $(x_i, f(x_i))$  is measured by calculating the distance  $d(x_i, x_q)$  between the query  $x_q$  and the input vector  $x_i$ .

Weighting the error criterion

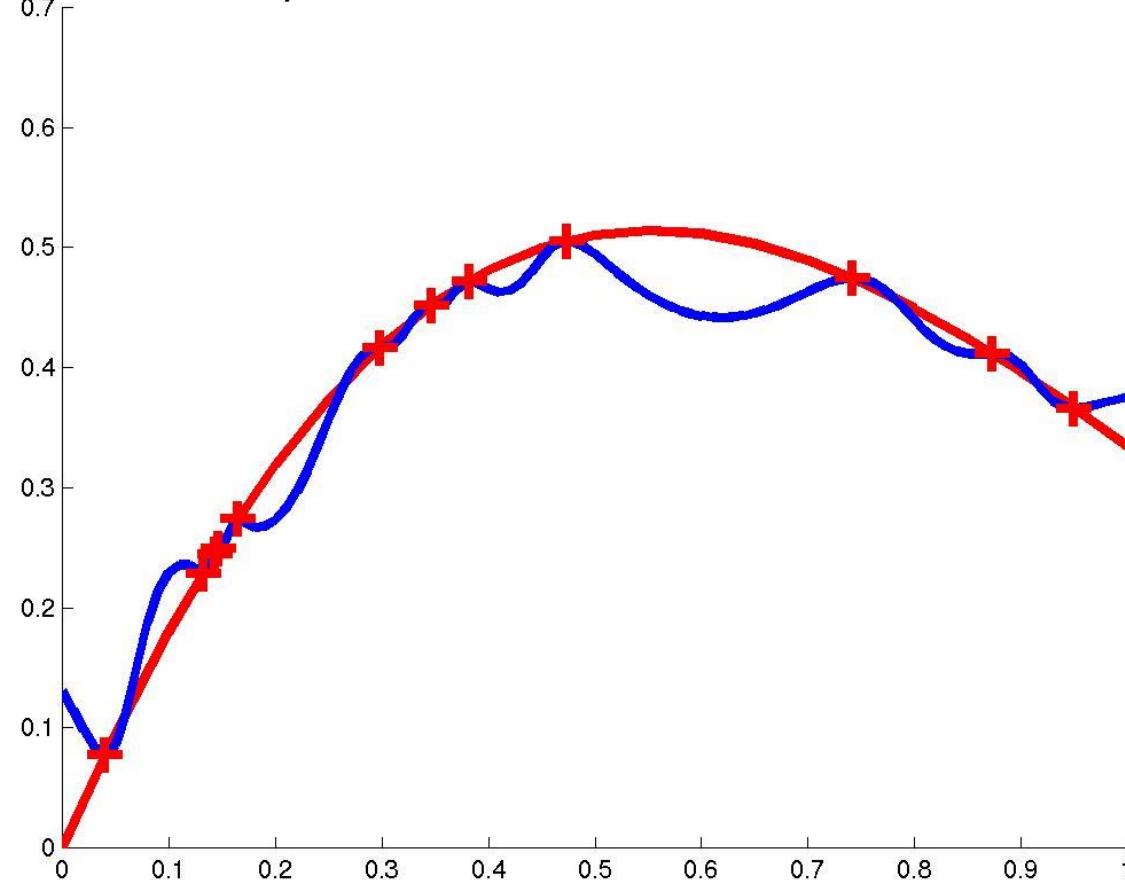
$$- E(x_q) = \sum_i (f^*(x_q) - f(x_i))^2 K(d(x_i, x_q))$$

Best estimate  $f^*(x_q)$  will minimize the cost  $E(x_q)$ , therefore

$$\partial E(x_q) / \partial f^*(x_q) = 0$$

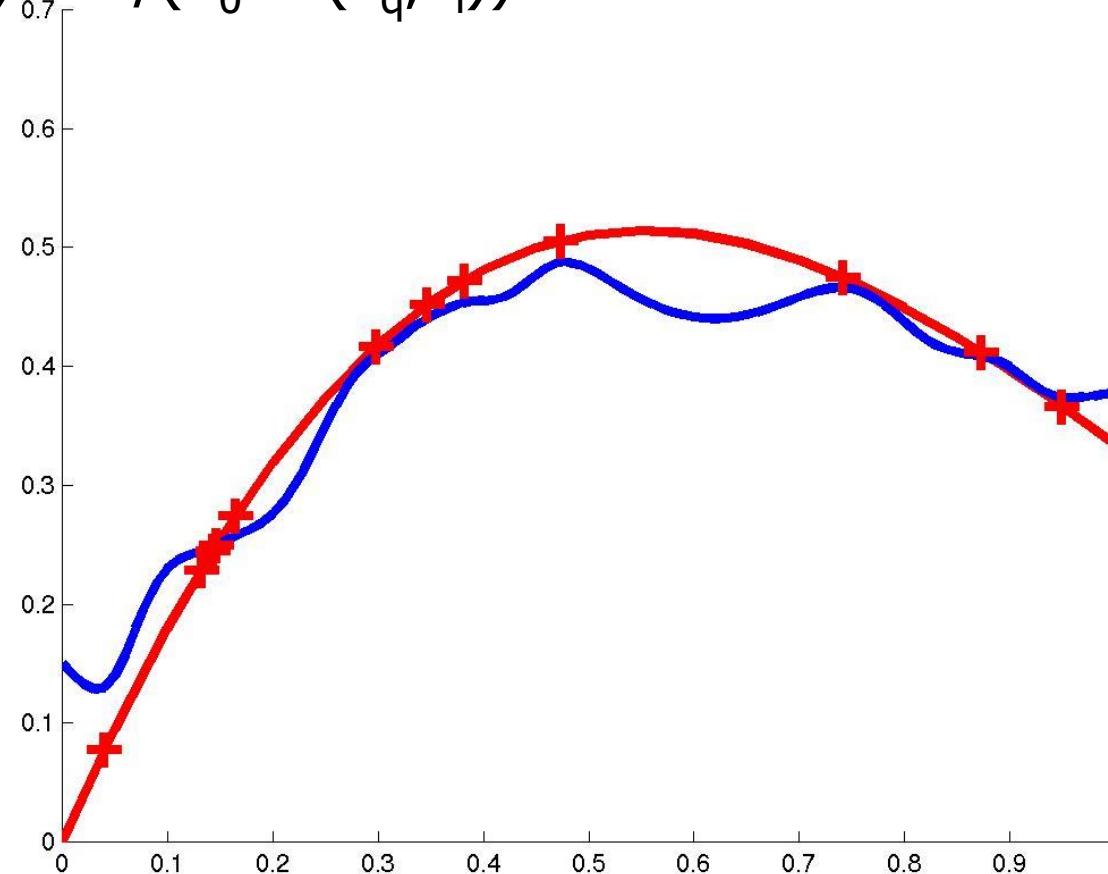
# Distance Weighted NN

$$K(d(x_q, x_i)) = 1/d(x_q, x_i)^2$$



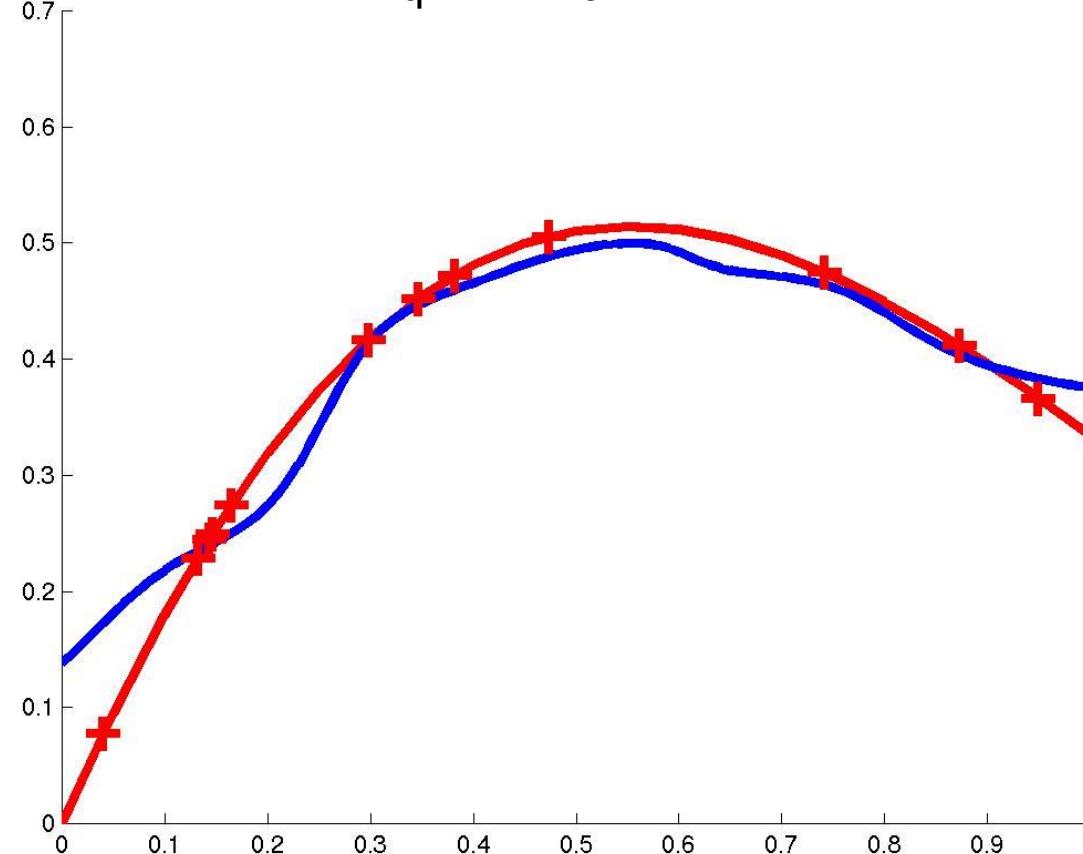
# Distance Weighted NN

$$K(d(x_q, x_i)) = 1/(d_0 + d(x_q, x_i))^2$$



# Distance Weighted NN

$$K(d(x_q, x_i)) = \exp(-(d(x_q, x_i)/\sigma_0)^2)$$



# Curse of Dimensionality

---

Imagine instances described by 20 attributes but only a few are relevant to target function

*Curse of dimensionality:* nearest neighbor is easily misled when instance space is high-dimensional

One approach:

Stretch  $j$ -th axis by weight  $z_j$ , where  $z_1, \dots, z_n$  chosen to minimize prediction error

Use cross-validation to automatically choose weights

$z_1, \dots, z_n$

Note setting  $z_j$  to zero eliminates this dimension altogether (feature subset selection)

---

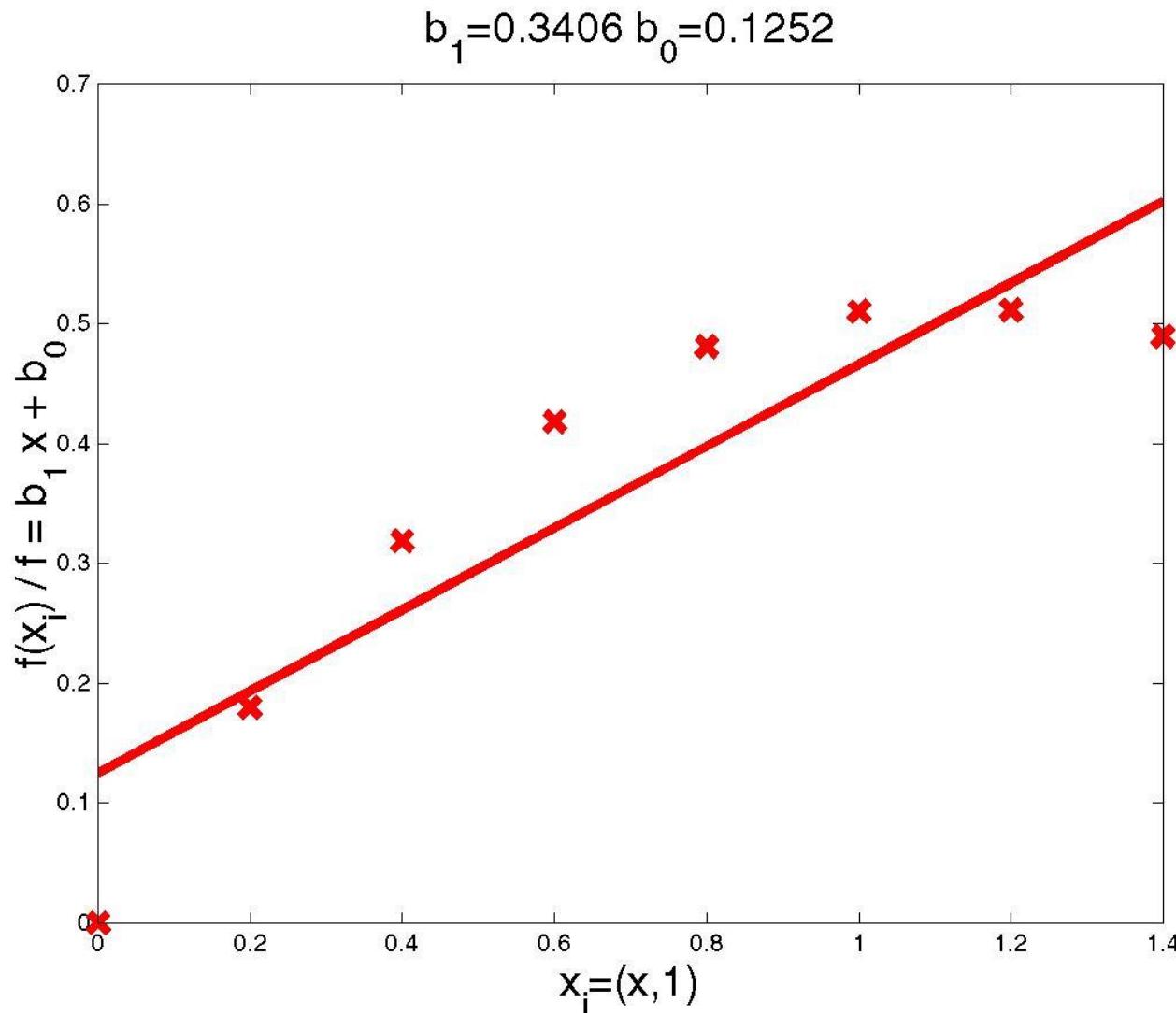
# Locally Weighted Regression

# Locally Weighted Regression

---

- Locally – Function approximated based on data near query point
- Weighted – Contribution by each training example is weighted by its distance from query point
- Regression- Approximates real-valued target function

# Linear Regression Example



# Locally weighted regression

---

- ➊ a note on terminology:
    - ➊ *Regression* means approximating a real-valued target function
    - ➋ *Residual* is the error  $\hat{f}(x) - f(x)$  in approximating the target function
    - ⌂ *Kernel function* is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function  $K$  such that  $w_i = K(d(x_i, x_q))$
  - ➋ nearest neighbor approaches can be thought of as approximating the target function at the single query point  $x_q$
  - ➌ locally weighted regression is a generalization to this approach, because it constructs an explicit approximation of  $f$  over a local region surrounding  $x_q$
-

# Locally weighted regression

---

- target function is approximated using a **linear function**
$$\hat{f}(x) = w_0 + w_1 a_1(x) + \dots + w_n a_n(x)$$
  - methods like **gradient descent** can be used to calculate the coefficients  $w_0, w_1, \dots, w_n$  to minimize the error in fitting such linear functions
  - ANNs require a global approximation to the target function
  - here, just a local approximation is needed
- ⇒ the error function has to be redefined
-

# Locally weighted regression



possibilities to redefine the error criterion  $E$

1. Minimize the squared error over just the  $k$  nearest neighbors

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neighbors}} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set  $D$ , while weighting the error of each training example by some decreasing function  $K$  of its distance from  $x_q$

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

3. Combine 1 and 2

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neighbors}} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

# Locally weighted regression

- ➊ choice of the error criterion
  - $E_2$  is the most esthetically criterion, because it allows every training example to have impact on the classification of  $x_q$
  - however, computational effort grows with the number of training examples
  - $E_3$  is a good approximation to  $E_2$  with constant effort

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest neighbors}} K(d(x_q, x))(f(x) - \hat{f}(x))a_j$$

- ➋ Remarks on locally weighted linear regression:
  - in most cases, constant, linear or quadratic functions are used
  - costs for fitting more complex functions are prohibitively high
  - simple approximations are good enough over a sufficiently small subregion of  $X$

# Linear Local Model Example

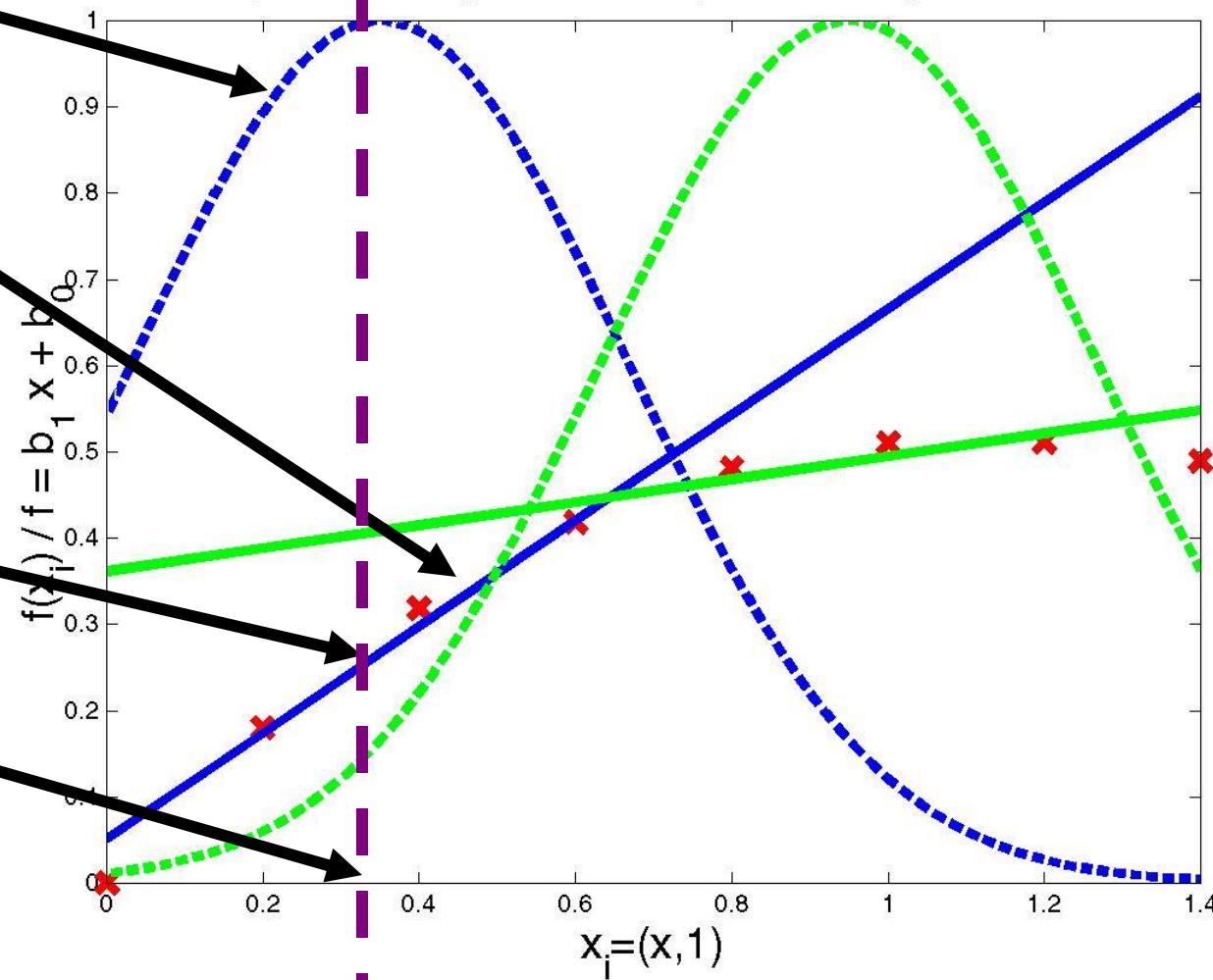
Kernel  $K(x, x_q)$

Local linear  
model:  
 $f^*(x) = b_1 x + b_0$

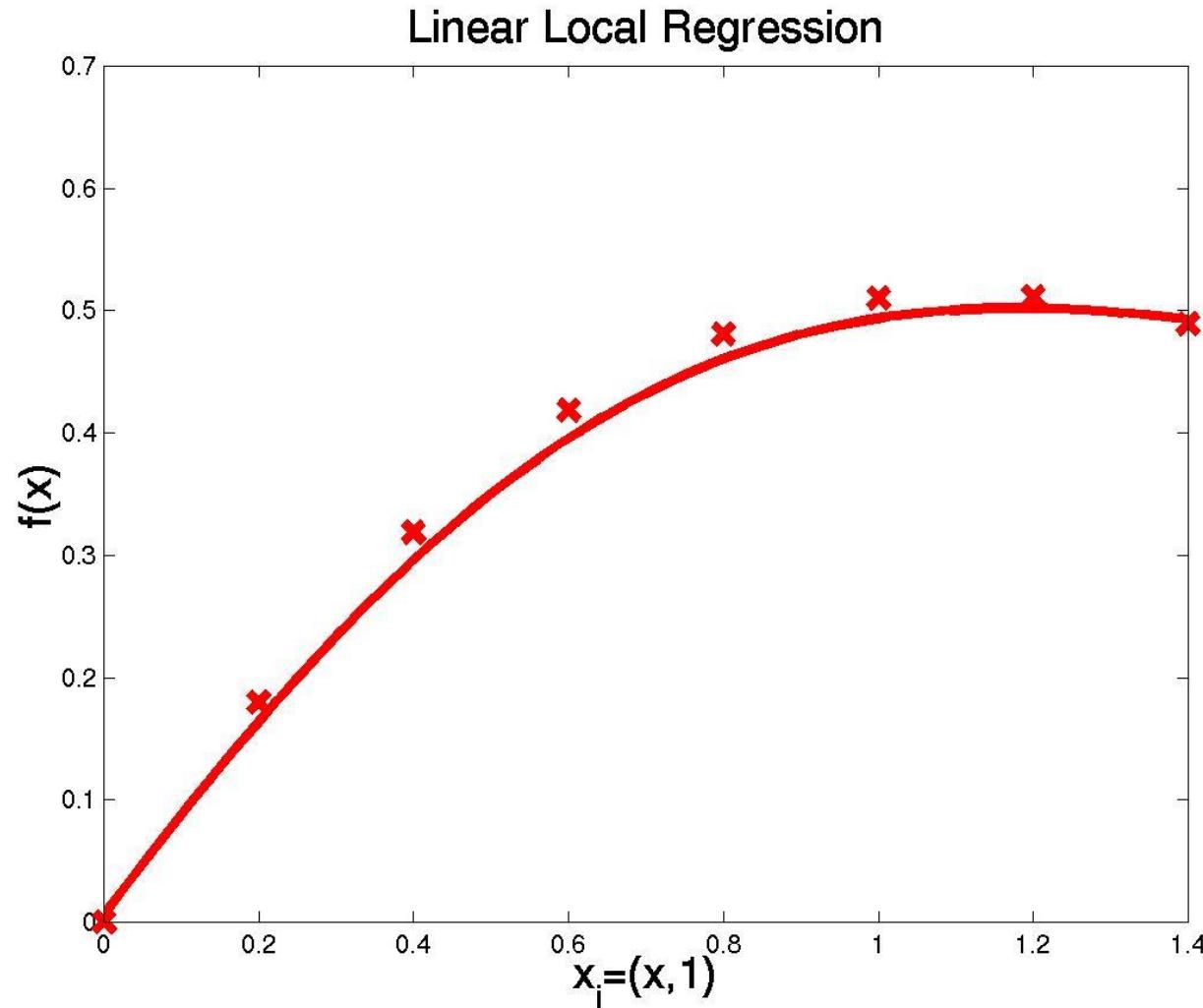
$f^*(x_q) = 0.266$

query point  
 $x_q = 0.35$

$$b_1 = 0.6154 \quad b_0 = 0.0506 \quad b_1 = 0.1331 \quad b_0 = 0.3617$$



# Linear Local Model Example



# Design Issues in Local Regression

---

- Local model order (constant, linear, quadratic)
- Distance function  $d$ 
  - feature scaling:  $d(x,q) = (\sum_{j=1}^d m_j (x_j - q_j)^2)^{1/2}$
  - irrelevant dimensions  $m_j = 0$
- kernel function  $K$

See paper by Atkeson [1996] "Locally Weighted Learning"

---

# Radial Basis Function

# Radial Basis Function

---

- closely related to distance-weighted regression and to ANNs
- learned hypotheses have the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^k w_u \cdot K_u(d(x_u, x))$$

where

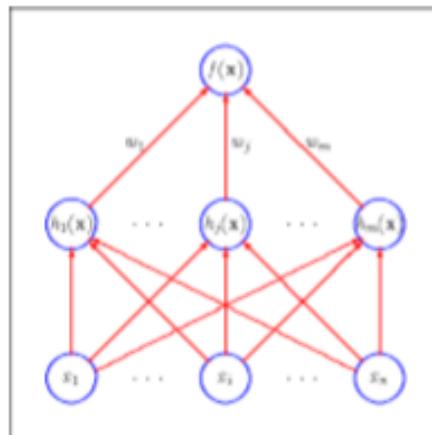
- each  $x_u$  is an instance from  $X$  and
  - $K_u(d(x_u, x))$  decreases as  $d(x_u, x)$  increases and
  - $k$  is a user-provided constant
- 
- though  $\hat{f}(x)$  is a global approximation to  $f(x)$ , the contribution of each of the  $K_u$  terms is localized to a region nearby the point  $x_u$

# Radial Basis Function

- it is common to choose each function  $K_u(d(x_u, x))$  to be a Gaussian function centered at  $x_u$  with some variance  $\sigma^2$

$$K_u(d(x_u, x)) = e^{-\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

- the function of  $\hat{f}(x)$  can be viewed as describing a two-layer network where the first layer of units computes the various  $K_u(d(x_u, x))$  values and the second layer a linear combination of the results



# Training Radial Basis Function Networks

---

**How to choose the center  $x_n$  for each Kernel function  $K_n$ ?**

- scatter uniformly across instance space
- use distribution of training instances (clustering)

**How to train the weights?**

- Choose mean  $x_n$  and variance  $\sigma_n$  for each  $K_n$   
non-linear optimization or EM
- Hold  $K_n$  fixed and use local linear regression to  
compute the optimal weights  $w_n$

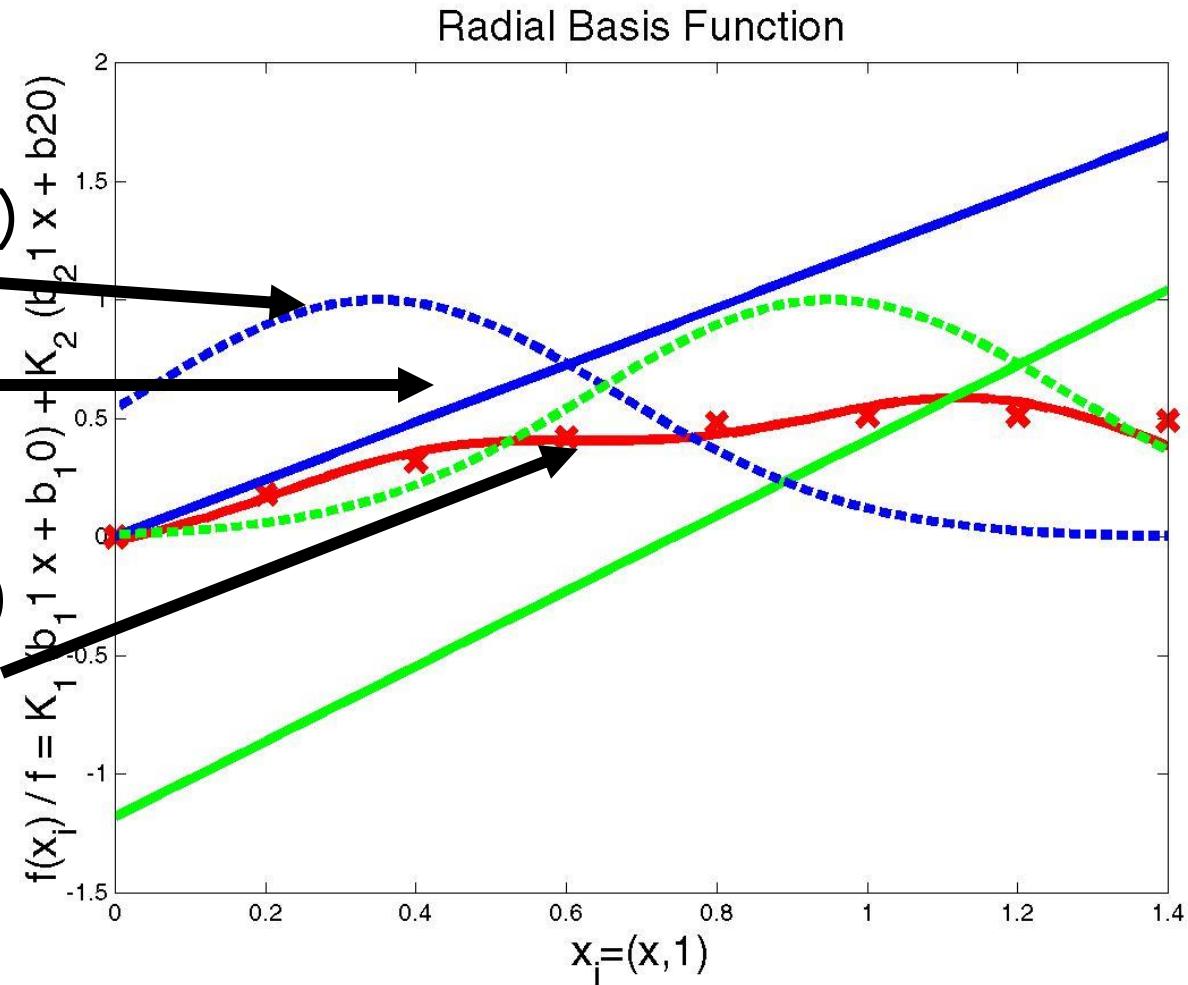
# Radial Basis Network Example

$$K_1(d(x_1, x)) =$$

$$\exp(-1/2 d(x_1, x)^2 / \sigma^2)$$

$$w_1 x + w_0$$

$$\hat{f}(x) = K_1(w_1 x + w_0) + K_2(w_3 x + w_2)$$



# and Eager Learning

---

## **Lazy: wait for query before generalizing**

- k-nearest neighbors, weighted linear regression

## **Eager: generalize before seeing query**

- Radial basis function networks, decision trees, back-propagation
- Eager learner must create global approximation

## **Lazy learner can create local approximations**

**If they use the same hypothesis space, lazy can represent more complex functions ( $H=$ linear functions)**

# Literature & Software

---

- T. Mitchell, “Machine Learning”, chapter 8, “Instance-Based Learning”
- “Locally Weighted Learning”, Christopher Atkeson, Andrew Moore, Stefan Schaal
- R. Duda et al., “Pattern recognition”, chapter 4 “Non-Parametric Techniques”

## Netlab toolbox

- k-nearest neighbor classification
- Radial basis function networks

# Thank You



**BITS** Pilani  
Pilani Campus

# Ensemble Learning

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

# Contents

---

- Combining classifiers
  - Bagging
  - Boosting
  - Random Forest Algorithm
  - AdaBoost Algorithm
  - Gradient Boosting
-

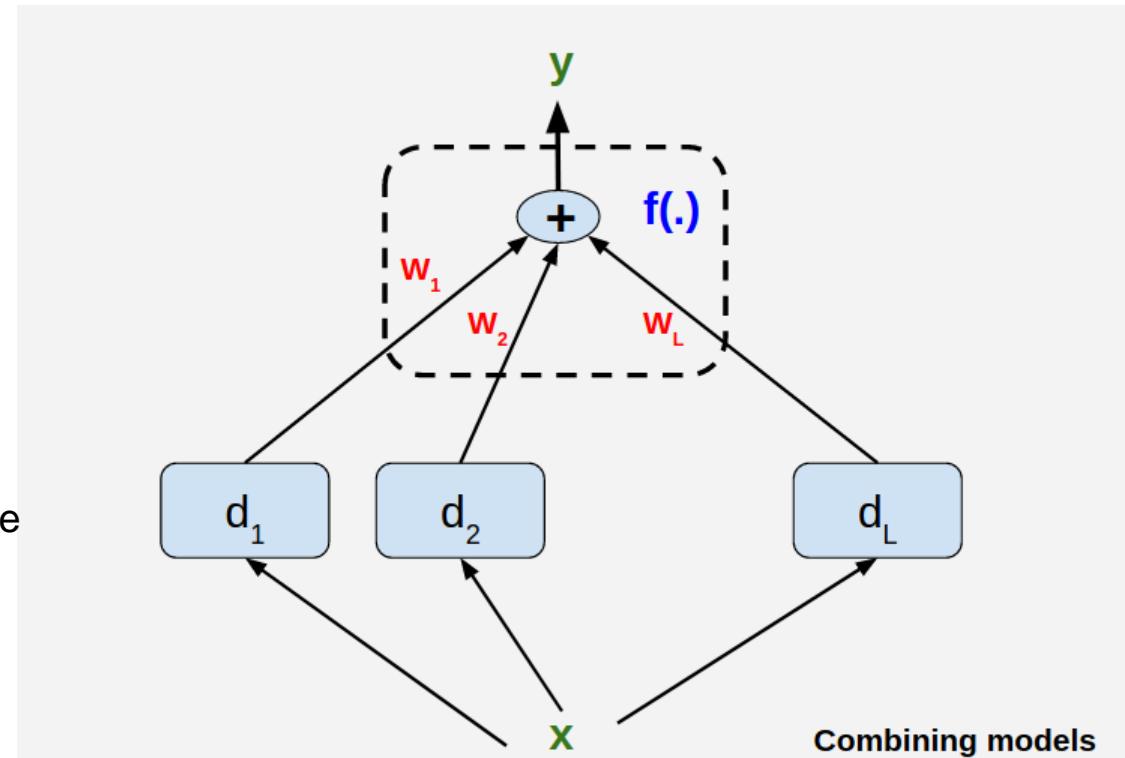
# Getting Started

---

- No Free Lunch Theorem: There is no algorithm that is always the most accurate
- Each learning algorithm dictates a certain model that comes with a set of assumptions
  - Each algorithm converges to a different solution and fails under different circumstances
    - The best tuned learners could miss some or examples and there could be other learners which works better on (may be only) those !
  - In the absence of a single expert (*a superior model*) , a committee (*combinations of models*) can do better !
    - A committee can work in many ways ...

# Committee of Models

- Committee Members are base learners !
- Major challenges dealing with this committee
  - Expertise of each of the members (Does it help / not?)
  - Combining the results from the members for better performance

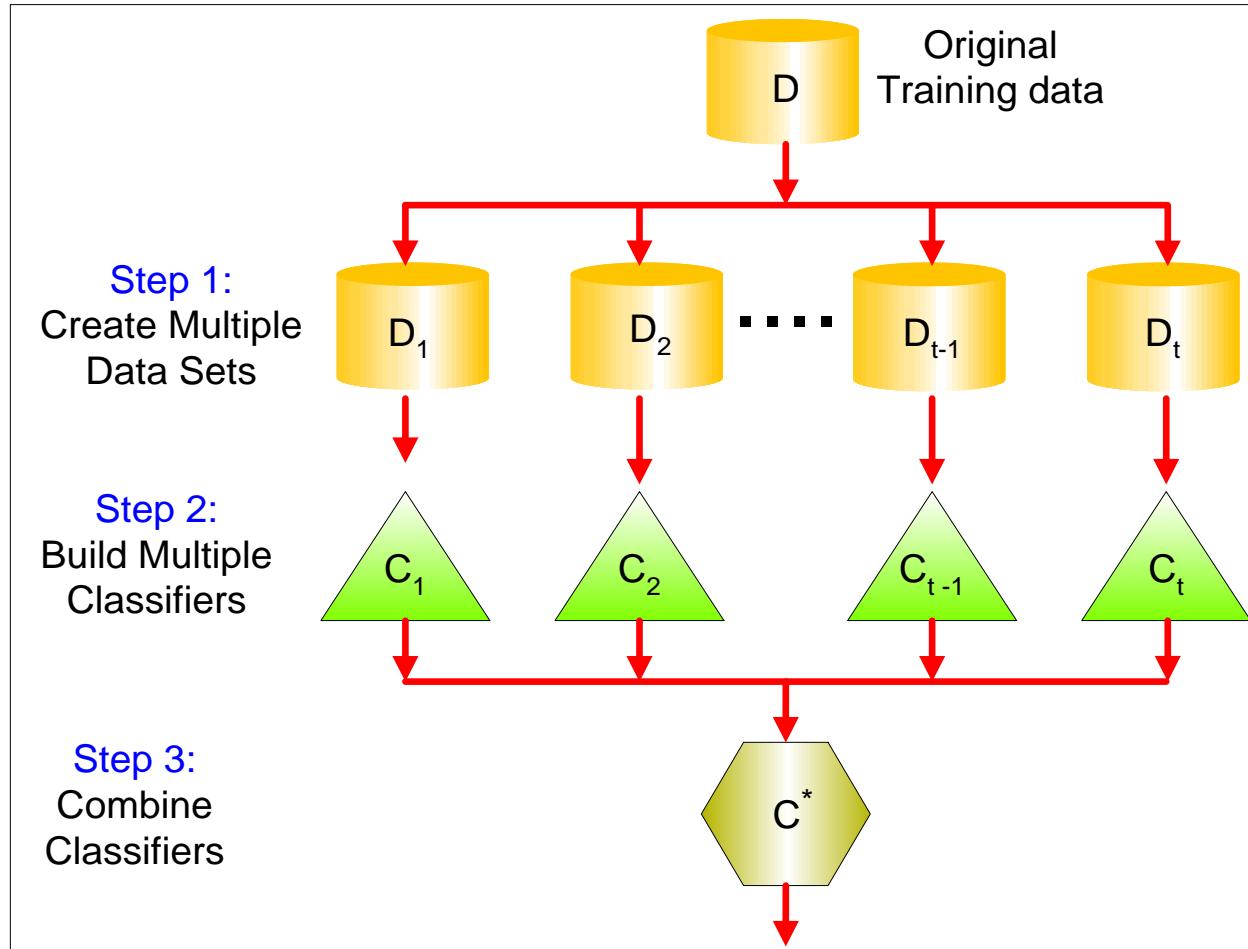


# Ensemble Methods

---

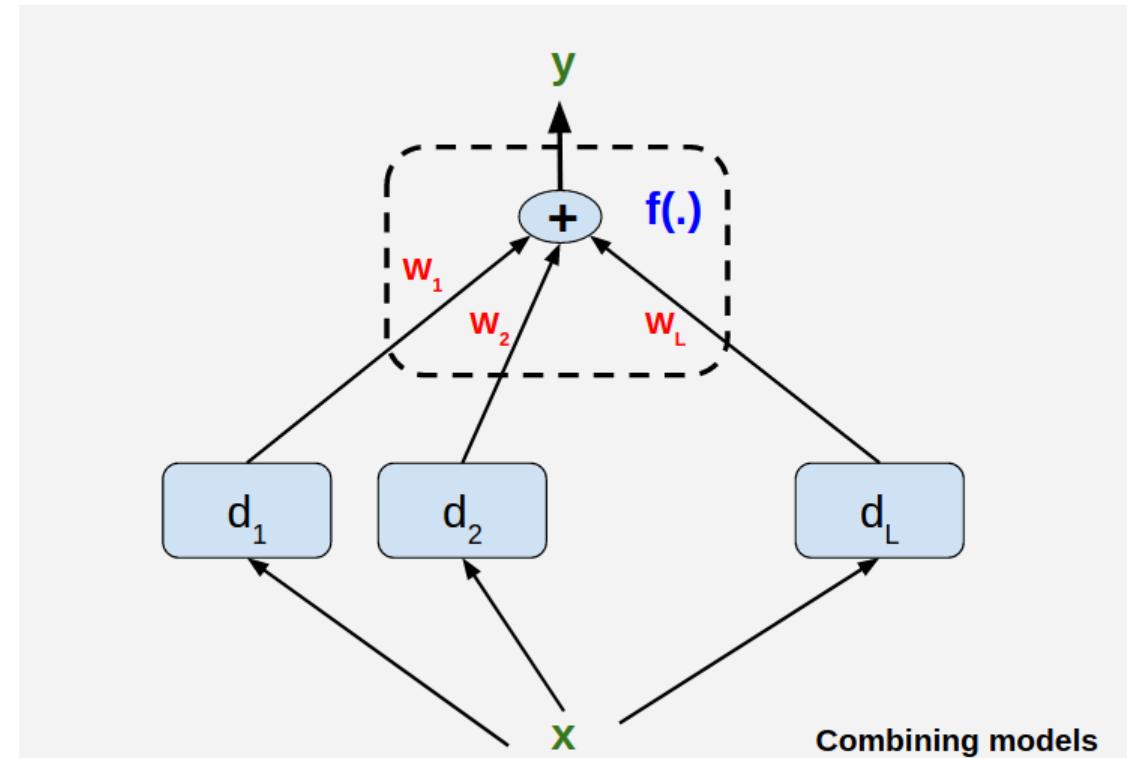
- **Ensemble methods** use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone
- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers
- Tend to reduce problems related to over-fitting of the training data.

# General Approach



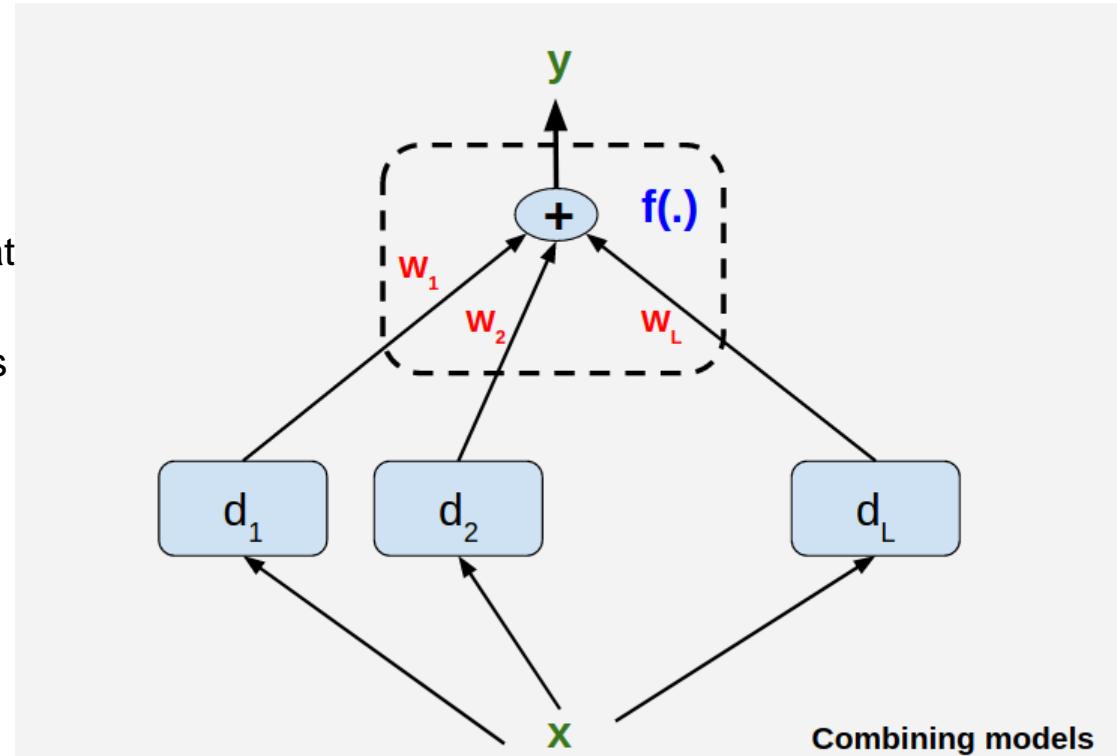
# Issue 1 : On the members ( Base Learners )

- It does not help if all learners are good/bad at roughly same thing
  - Need Diverse Learners



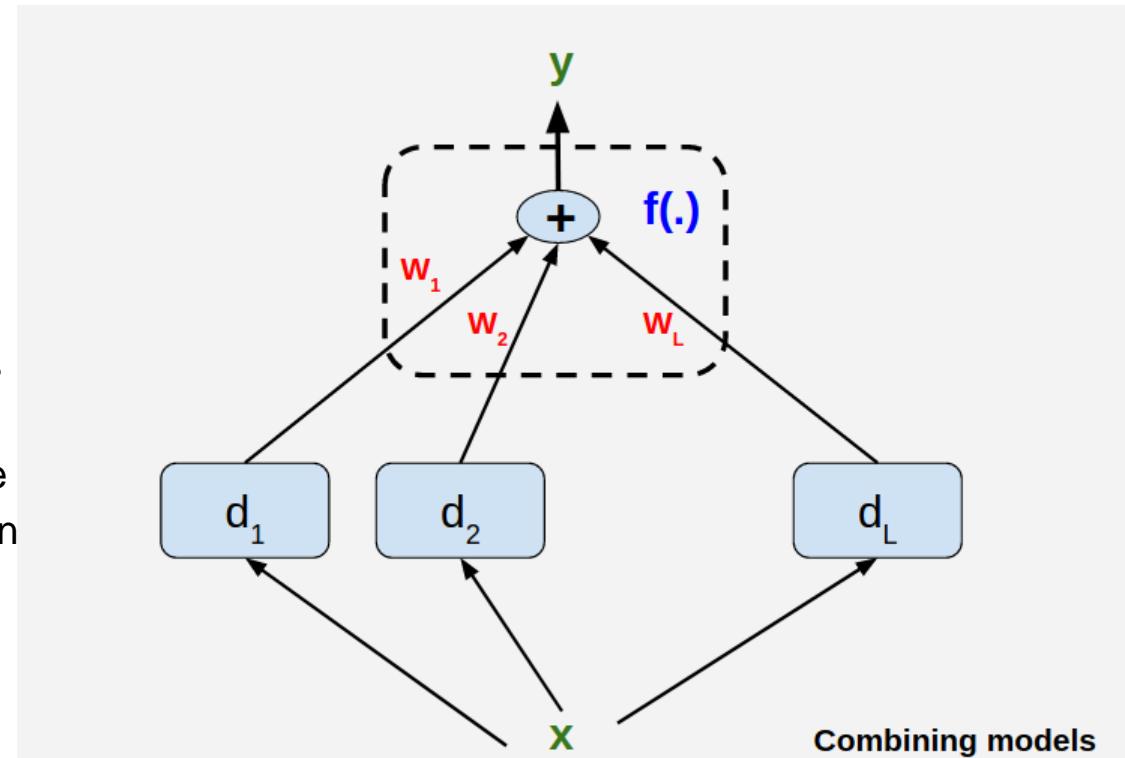
# Issue 1 : On the members ( Base Learners )

- Use Different Algorithms
  - Different algorithms make different assumptions
- Use Different Hyperparameters, that is ,
  - vary the structure of neural nets



# Issue 1 : On the members ( Base Learners )

- Different input representations
  - Uttered words + video information of speakers clips
  - image + text annotations
- Different training sets
  - Draw different random samples of data
  - Partition data in the input space and have learners specialized in those spaces (mixture of experts)



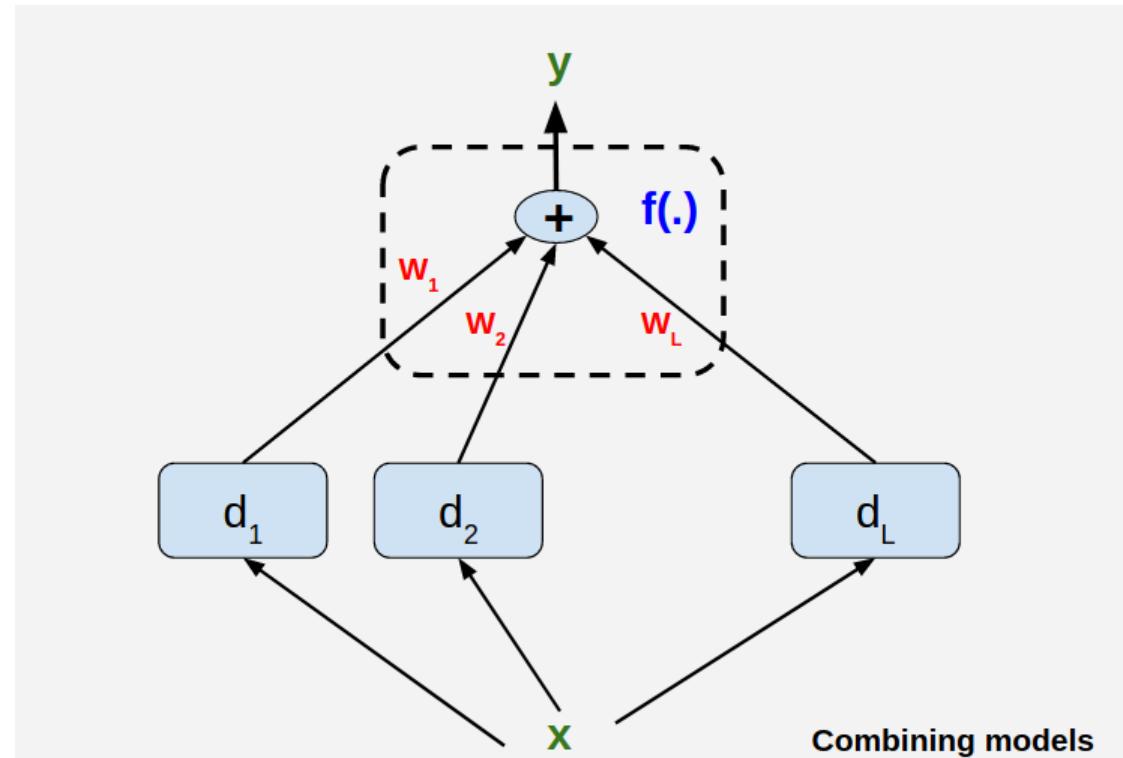
# Issue -2 : Combining Results

$$y = f(d_1, d_2, \dots, d_L | \Phi)$$

A Simple Combination Scheme:

$$y = \sum_{j=1}^L w_j d_j$$

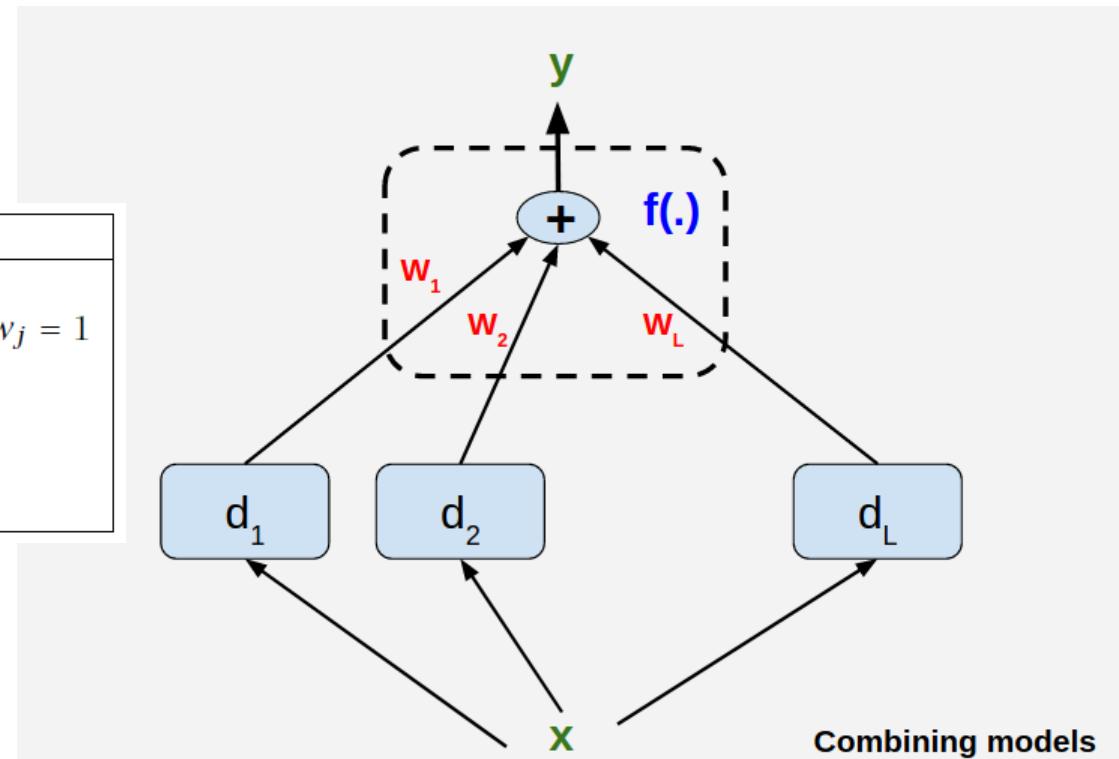
$$w_j \geq 0 \text{ and } \sum_{j=1}^L w_j = 1$$



# Issue -2 : Combining Results

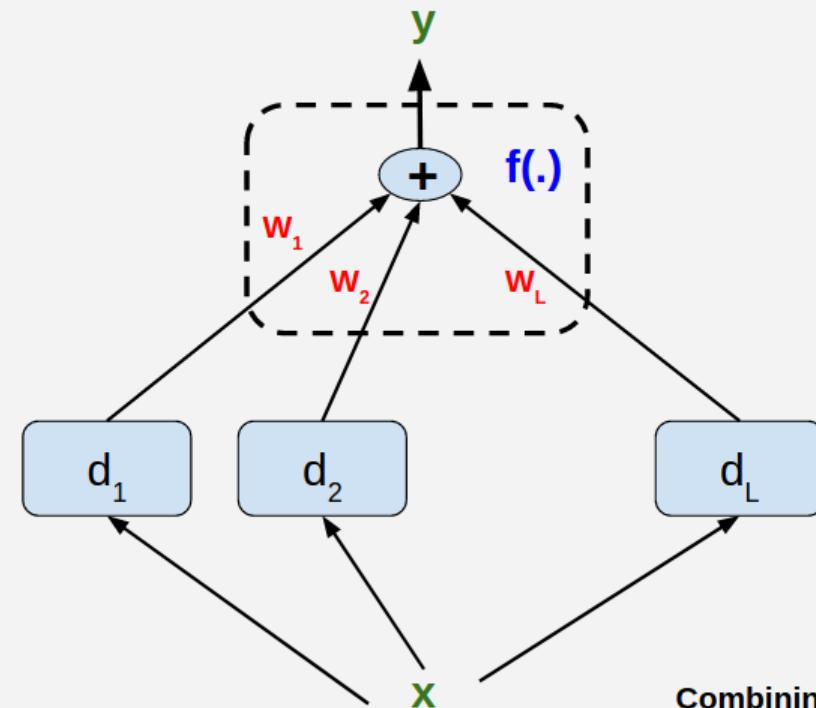
$$y = f(d_1, d_2, \dots, d_L | \Phi)$$

Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L} \sum_{j=1}^L d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$



# Issue -2 : Combining Results

	$C_1$	$C_2$	$C_3$
$d_1$	0.2	0.5	0.3
$d_2$	0.0	0.6	0.4
$d_3$	0.4	0.4	0.2
Sum	0.2	<b>0.5</b>	0.3
Median	0.2	<b>0.5</b>	0.4
Minimum	0.0	<b>0.4</b>	0.2
Maximum	0.4	<b>0.6</b>	0.4
Product	0.0	<b>0.12</b>	0.032



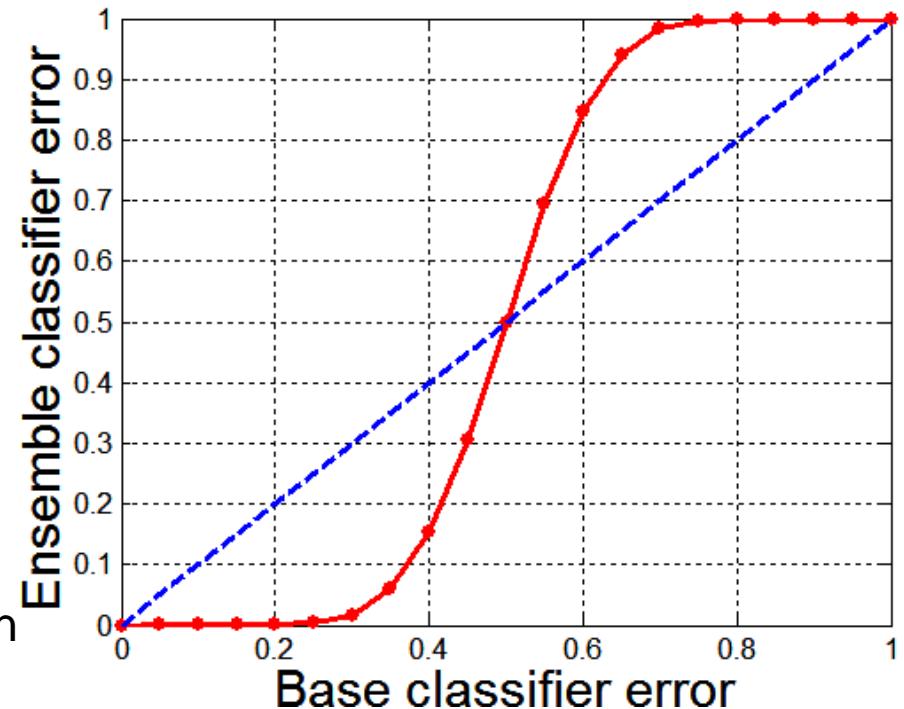
# When does Ensemble work?

---

- Ensemble classifier performs better than the base classifiers when  $e$  is smaller than 0.5
- Necessary conditions for an ensemble classifier to perform better than a single classifier:
  - Base classifiers should be independent of each other
  - Base classifiers should do better than a classifier that performs random guessing

# Why Ensemble Methods work?

- 25 base classifiers
- Each classifier has error rate,  $\varepsilon = 0.35$
- If base classifiers are identical, then the ensemble will misclassify the same examples predicted incorrectly by the base classifiers depicted by dotted line
- Assume errors made by classifiers are uncorrelated
- Ensemble makes a wrong prediction only if base classifiers error is more than 0.5



# Types of Ensemble Methods

---

## Manipulate data distribution

- Example: bagging, boosting

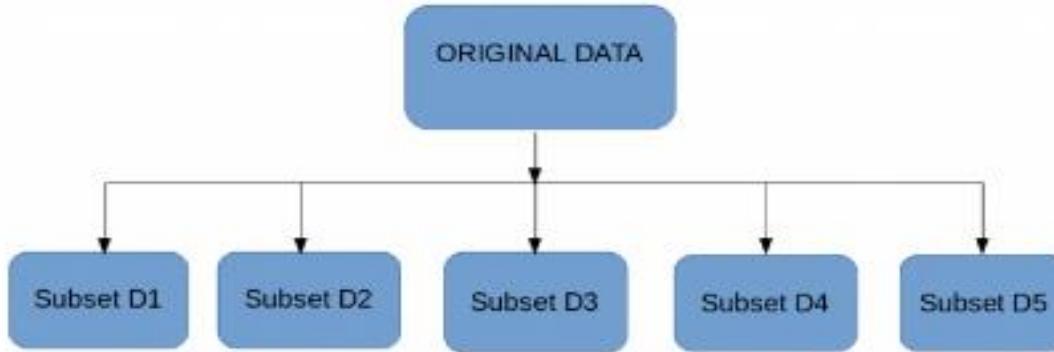
## Manipulate input features

- Example: random forests

# Bagging (Bootstrap Aggregating)

- Technique uses these subsets (bags) to get a fair idea of the distribution (complete set).
- The size of subsets created for bagging may be less than the original set.
- Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, **with replacement**.
- When you sample with replacement, items are independent. One item does not affect the outcome of the other. You have  $1/7$  chance of choosing the first item and a  $1/7$  chance of choosing the second item.
- If the two items are **dependent**, or linked to each other. When you choose the first item, you have a  $1/7$  probability of picking a item. Assuming you don't replace the item, you only have six items to pick from. That gives you a  $1/6$  chance of choosing a second item.

# Bagging



- Multiple subsets are created from the original dataset, selecting observations with replacement.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.

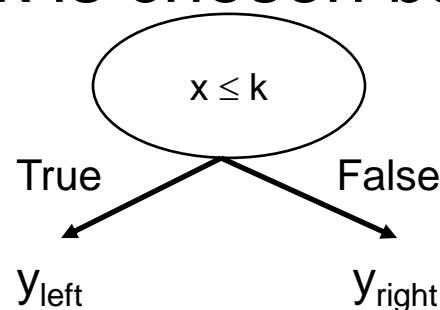
# Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule:  $x \leq k$  versus  $x > k$
  - Split point  $k$  is chosen based on entropy



# Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

$x \leq 0.35 \rightarrow y = 1$   
 $x > 0.35 \rightarrow y = -1$

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

$x \leq 0.7 \rightarrow y = 1$   
 $x > 0.7 \rightarrow y = 1$

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.35 \rightarrow y = 1$   
 $x > 0.35 \rightarrow y = -1$

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.3 \rightarrow y = 1$   
 $x > 0.3 \rightarrow y = -1$

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

$x \leq 0.35 \rightarrow y = 1$   
 $x > 0.35 \rightarrow y = -1$

# Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

$$x \leq 0.75 \rightarrow y = -1$$
$$x > 0.75 \rightarrow y = 1$$

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

$$x \leq 0.75 \rightarrow y = -1$$
$$x > 0.75 \rightarrow y = 1$$

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$$x \leq 0.75 \rightarrow y = -1$$
$$x > 0.75 \rightarrow y = 1$$

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$$x \leq 0.75 \rightarrow y = -1$$
$$x > 0.75 \rightarrow y = 1$$

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

$$x \leq 0.05 \rightarrow y = 1$$
$$x > 0.05 \rightarrow y = 1$$

# Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

# Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted  
Class

# Bagging Algorithm

---

## Algorithm 5.6 Bagging Algorithm

---

- 1: Let  $k$  be the number of bootstrap samples.
  - 2: for  $i = 1$  to  $k$  do
  - 3:   Create a bootstrap sample of size  $n$ ,  $D_i$ .
  - 4:   Train a base classifier  $C_i$  on the bootstrap sample  $D_i$ .
  - 5: end for
  - 6:  $C^*(x) = \arg \max_y \sum_i \delta(C_i(x) = y)$ ,  $\{\delta(\cdot) = 1 \text{ if its argument is true, and } 0 \text{ otherwise.}\}$
-

# Boosting

---

- What if a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.
- Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model.
- The succeeding models are dependent on the previous model.

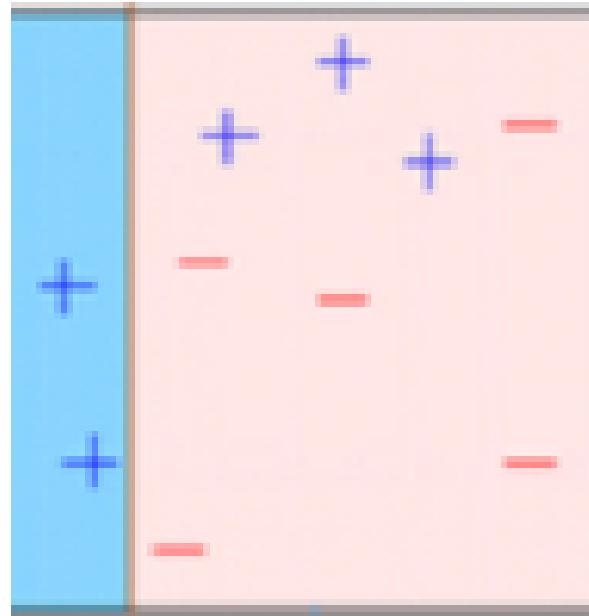
# Boosting

---

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all  $N$  records are assigned equal weights
  - Unlike bagging, weights may change at the end of each boosting round

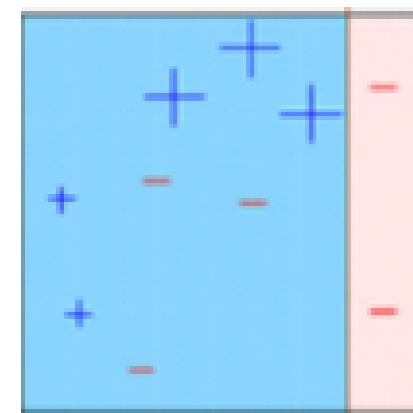
# Boosting

- A subset is created from the original dataset.
- Initially, all data points are given equal weights.
- A base model is created on this subset.
- This model is used to make predictions on the whole dataset.



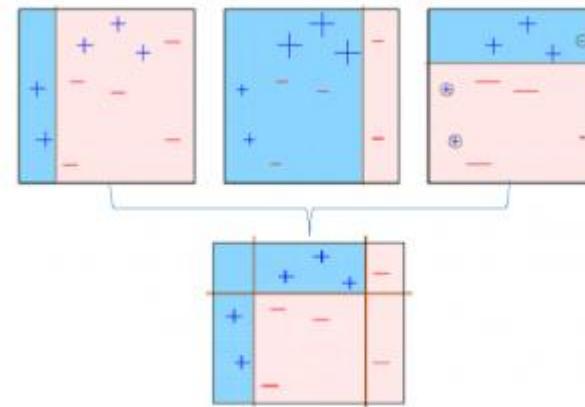
# Boosting

- Errors are calculated using the actual values and predicted values.
- The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)
- Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)



# Boosting

- Similarly, multiple models are created, each correcting the errors of the previous model.
- The final model (strong learner) is the weighted mean of all the models (weak learners).



- Individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.

# Algorithms based on Bagging and Boosting

---

## Bagging algorithms:

- Random forest

## Boosting algorithms:

- AdaBoost

# Random Forest

---

- Random Forest is ensemble machine learning algorithm that follows the bagging technique.
  - The base estimators in random forest are decision trees.
  - Random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.
-

# Random Forest

---

- Random subsets are created from the original dataset (bootstrapping).
- At each node in the decision tree, only a random set of features are considered to decide the best split.
- A decision tree model is fitted on each of the subsets.
- The final prediction is calculated by averaging the predictions from all decision trees.

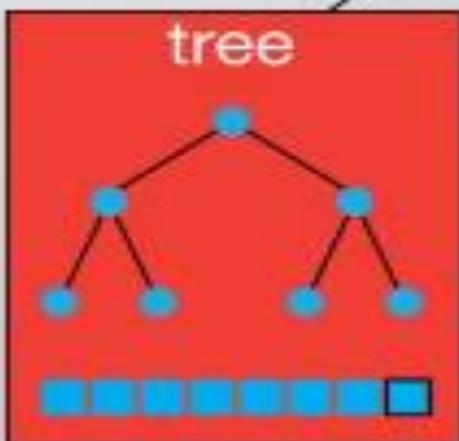
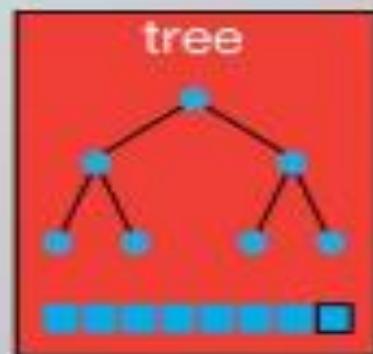
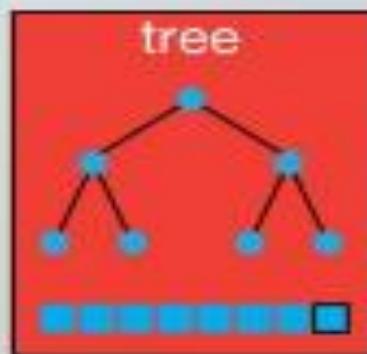
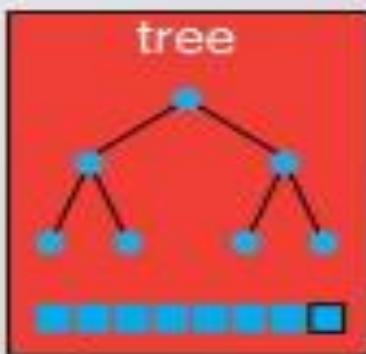
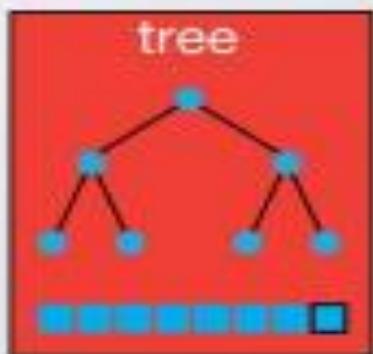
# All Data

random subset

random subset

random subset

random subset



At each node:  
choose some ballsubset of variables at random  
find a variable ( and a value for that variable) which optimizes the split

# Advantages of Random Forest

- Algorithm can solve both type of problems i.e. classification and regression
- Power to handle large data set with higher dimensionality.
- It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.
- Model outputs **Importance of variable**, which can be a very handy feature (on some random data set).

# Disadvantages of Random Forest

- May over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modelers – you have very little control on what the model does. You can at best – try different parameters and random seeds!

# AdaBoost

---

- Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model.
- AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

# AdaBoost Algorithm

---

- Initially, all observations ( $n$ ) in the dataset are given equal weights ( $1/n$ ).
- A model is built on a subset of data.
- Using this model, predictions are made on the whole dataset.
- Errors are calculated by comparing the predictions and actual values.
- While creating the next model, higher weights are given to the data points which were predicted incorrectly.

# Adaboost Algorithm

---

- Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
- This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

- Base classifiers  $C_i$ :  $C_1, C_2, \dots, C_T$
- Error rate:
  - $N$  input samples

$$\varepsilon_i = \frac{1}{N} \sum_{j=1}^N w_j \delta(C_i(x_j) \neq y_j)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

[https://en.wikipedia.org/wiki/AdaBoost#Choosing\\_at](https://en.wikipedia.org/wiki/AdaBoost#Choosing_at)

# AdaBoost Algorithm

---

## Algorithm 5.7 AdaBoost Algorithm

---

```

1:  $w = \{w_j = 1/n \mid j = 1, 2, \dots, n\}$ . {Initialize the weights for all  $n$  instances.}
2: Let  $k$  be the number of boosting rounds.
3: for  $i = 1$  to  $k$  do
4:   Create training set  $D_i$  by sampling (with replacement) from  $D$  according to  $w$ .
5:   Train a base classifier  $C_i$  on  $D_i$ .
6:   Apply  $C_i$  to all instances in the original training set,  $D$ .
7:    $\epsilon_i = \frac{1}{n} [\sum_j w_j \delta(C_i(x_j) \neq y_j)]$  {Calculate the weighted error}
8:   if  $\epsilon_i > 0.5$  then
9:      $w = \{w_j = 1/n \mid j = 1, 2, \dots, n\}$ . {Reset the weights for all  $n$  instances.}
10:    Go back to Step 4.
11:   end if
12:    $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$ .
13:   Update the weight of each instance according to equation (5.88).
14: end for
15:  $C^*(x) = \arg \max_y \sum_{j=1}^T \alpha_j \delta(C_j(x) = y)$ .

```

---

# AdaBoost: Weight Update

Weight Update:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases} \quad \text{Eqn:5.88}$$

where  $Z_j$  is the normalization factor

$$C^*(x) = \arg \max_y \sum_{j=1}^T \alpha_j \delta(C_j(x) = y)$$

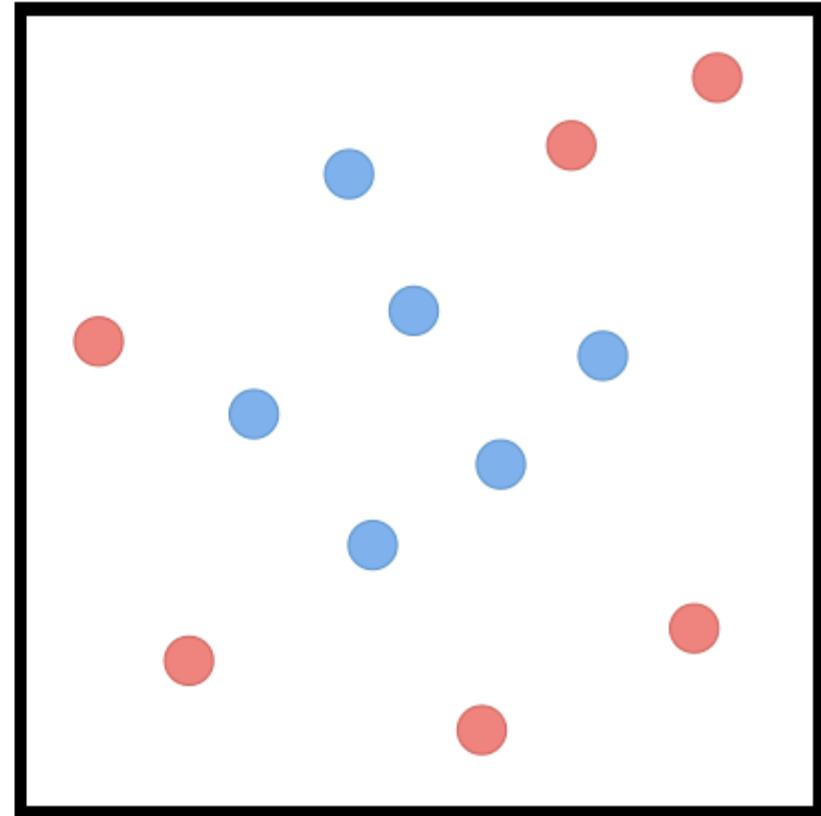
- Reduce weight if correctly classified else increase
- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to  $1/n$  and the resampling procedure is repeated

# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$


```



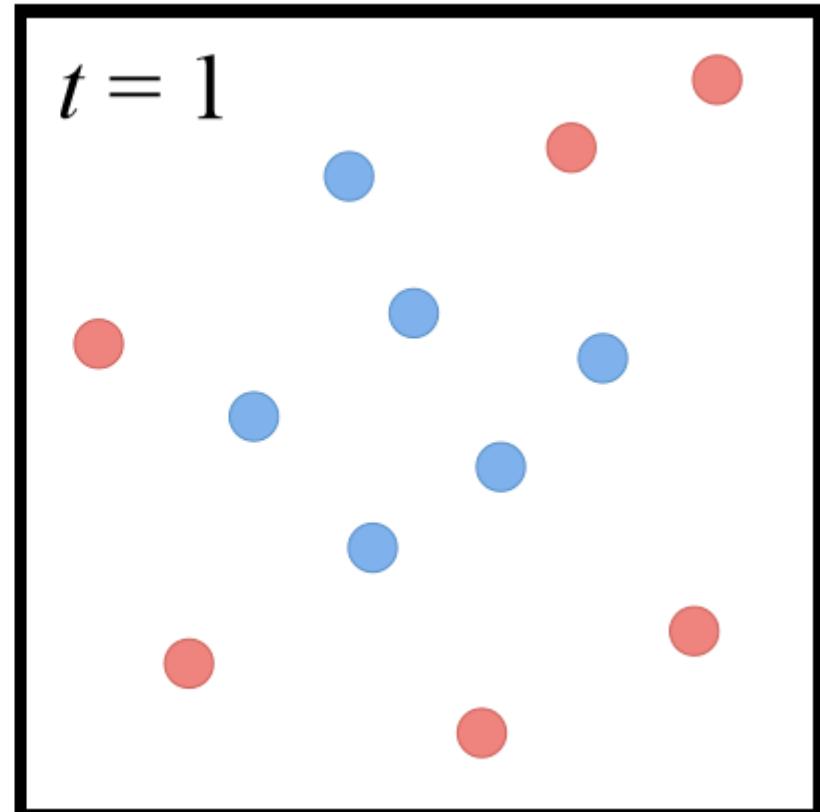
- Size of point represents the instance's weight

# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

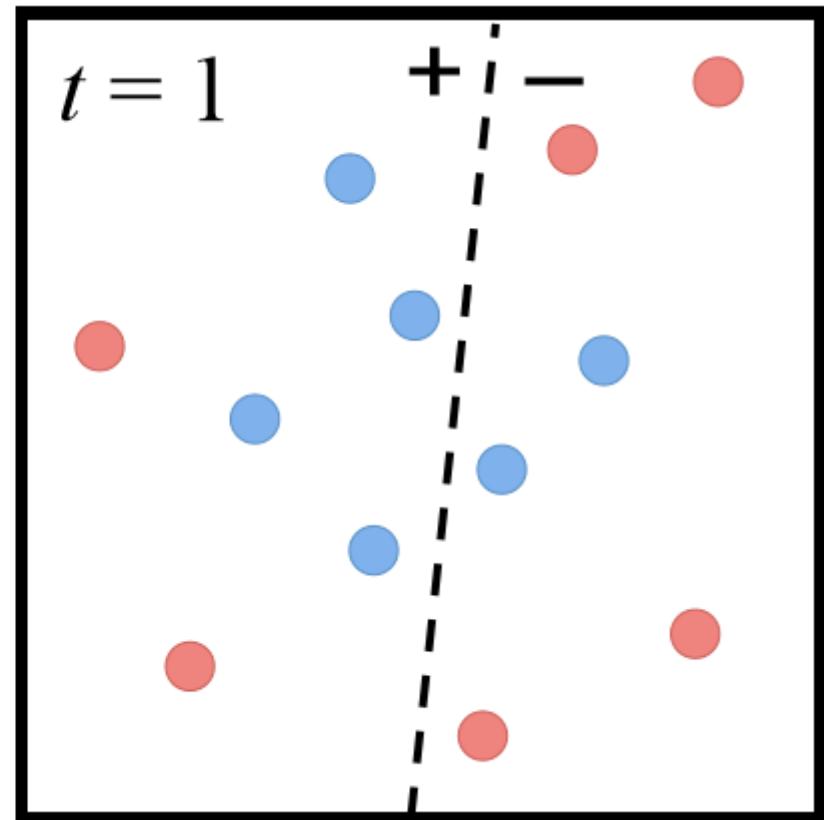


# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

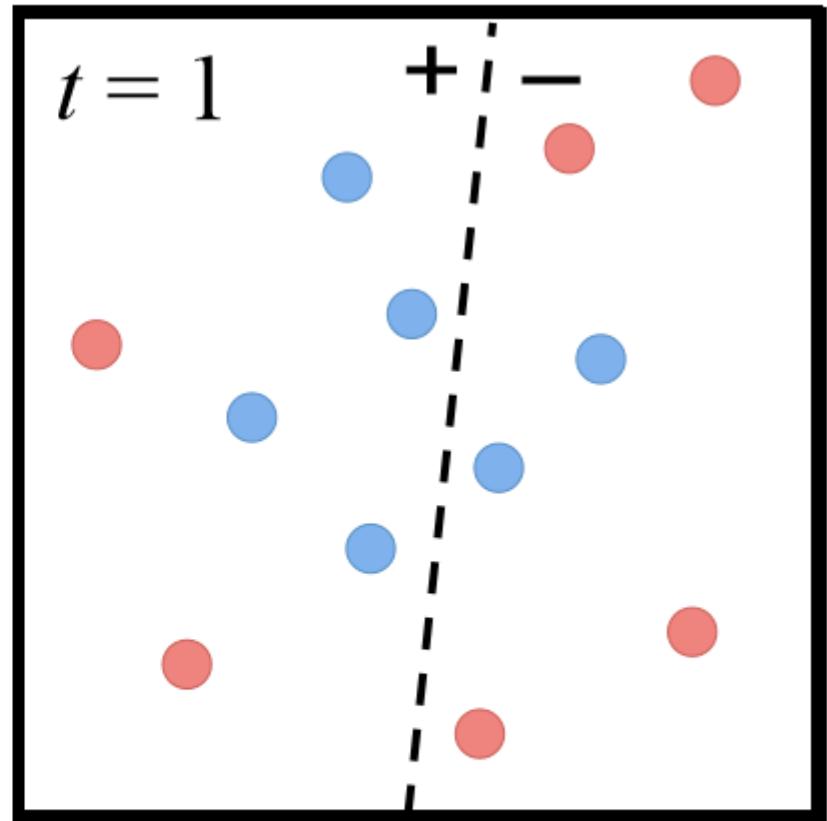


# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



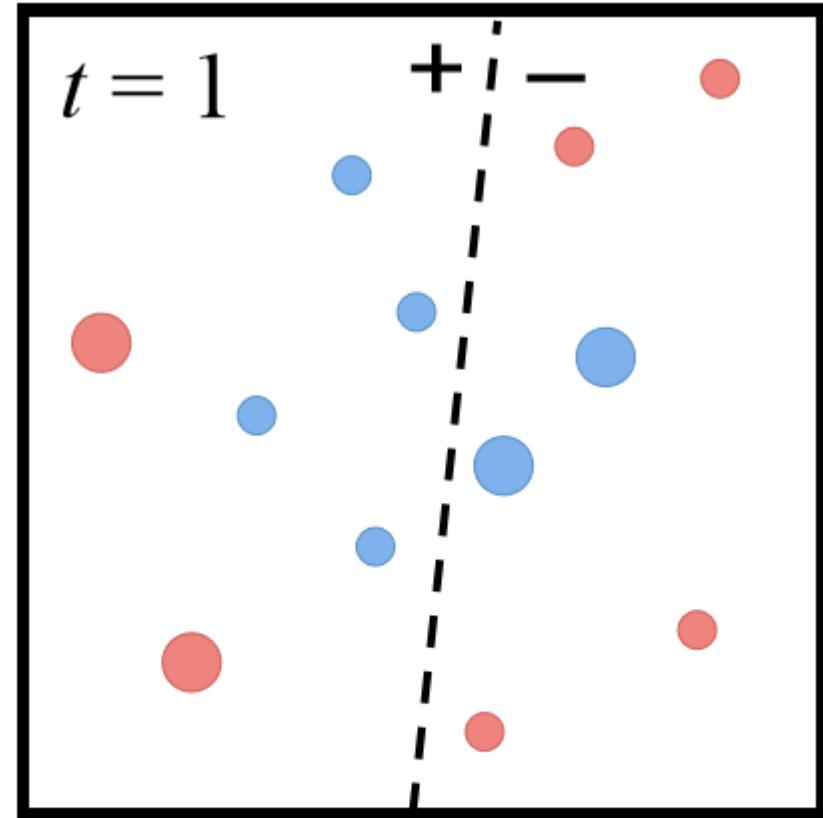
- $\beta_t$  measures the importance of  $h_t$
- If  $\epsilon_t \leq 0.5$ , then  $\beta_t \geq 0$  ( $\beta_t$  grows as  $\epsilon_t$  gets smaller)

# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



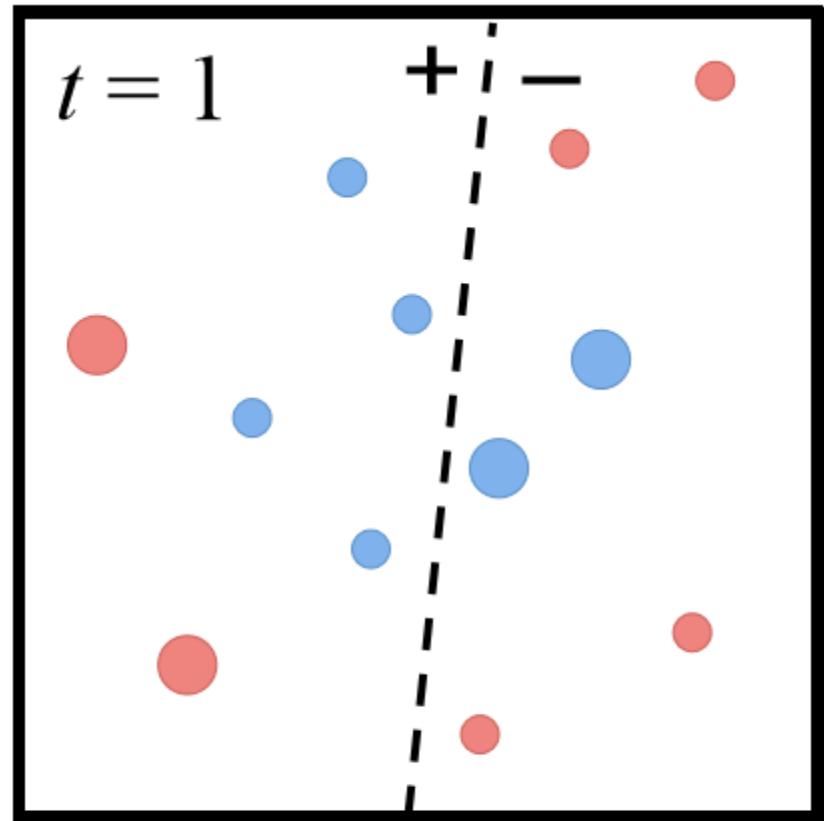
- Weights of correct predictions are multiplied by  $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by  $e^{\beta_t} \geq 1$

# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



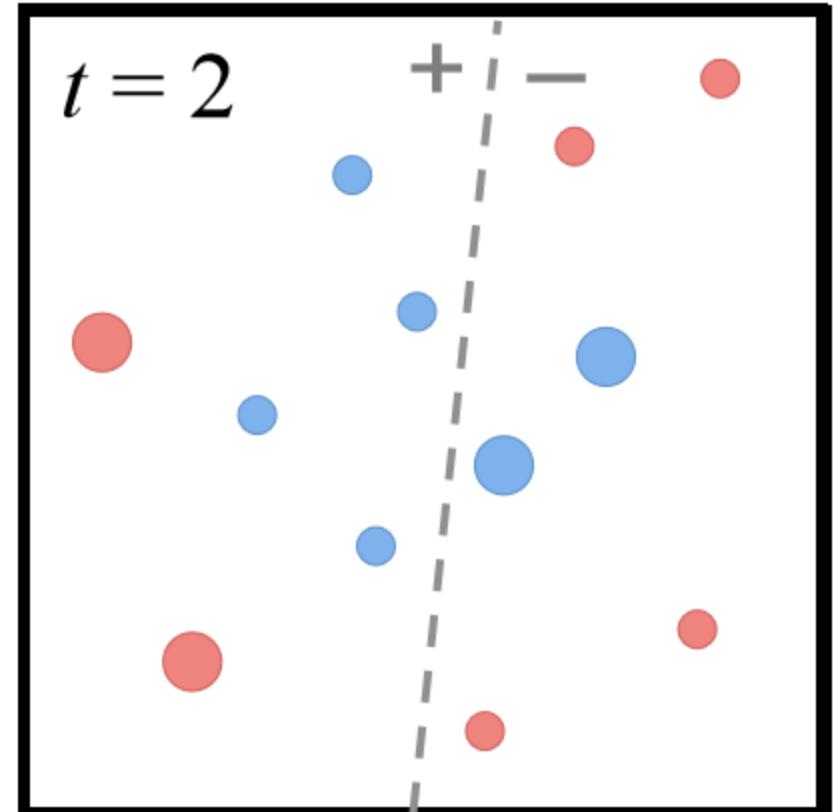
**Disclaimer:** Note that resized points in the illustration above are not necessarily to scale with  $\beta_t$

# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

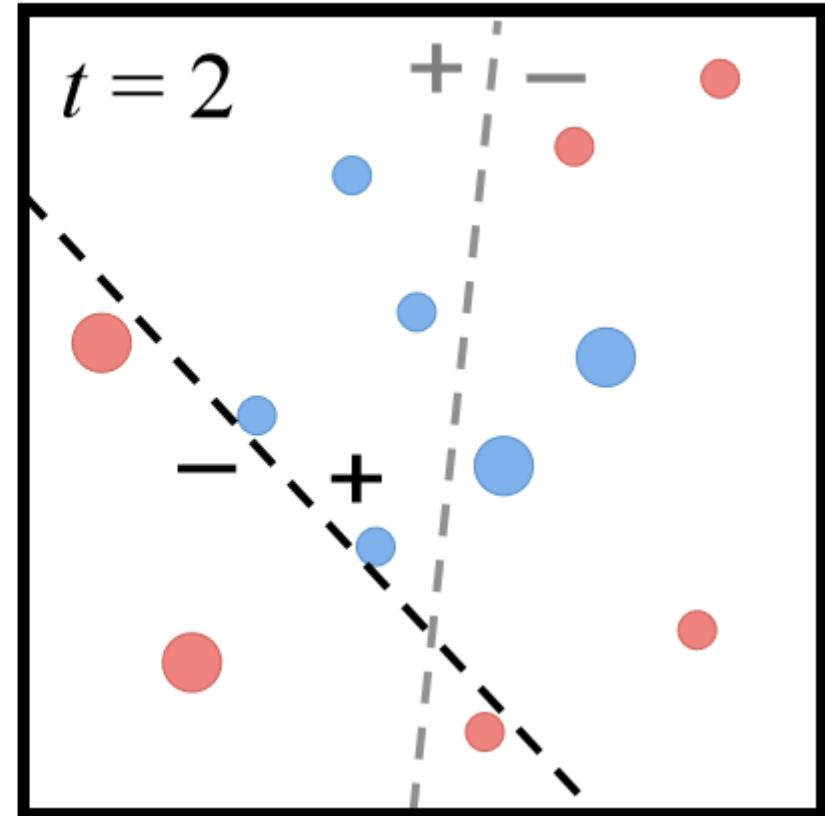


# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

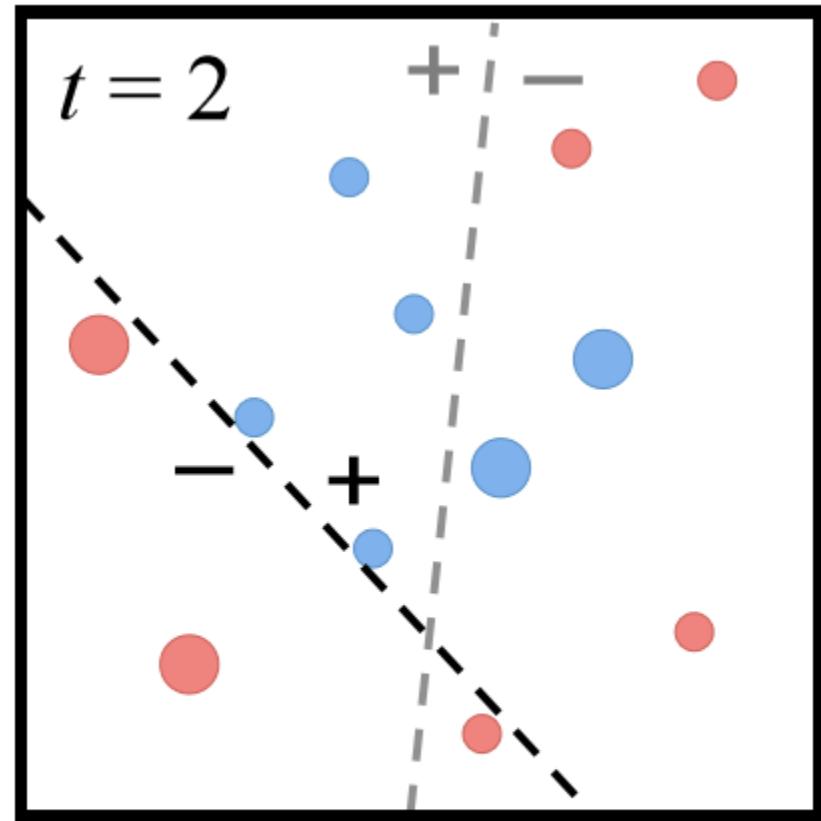


# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



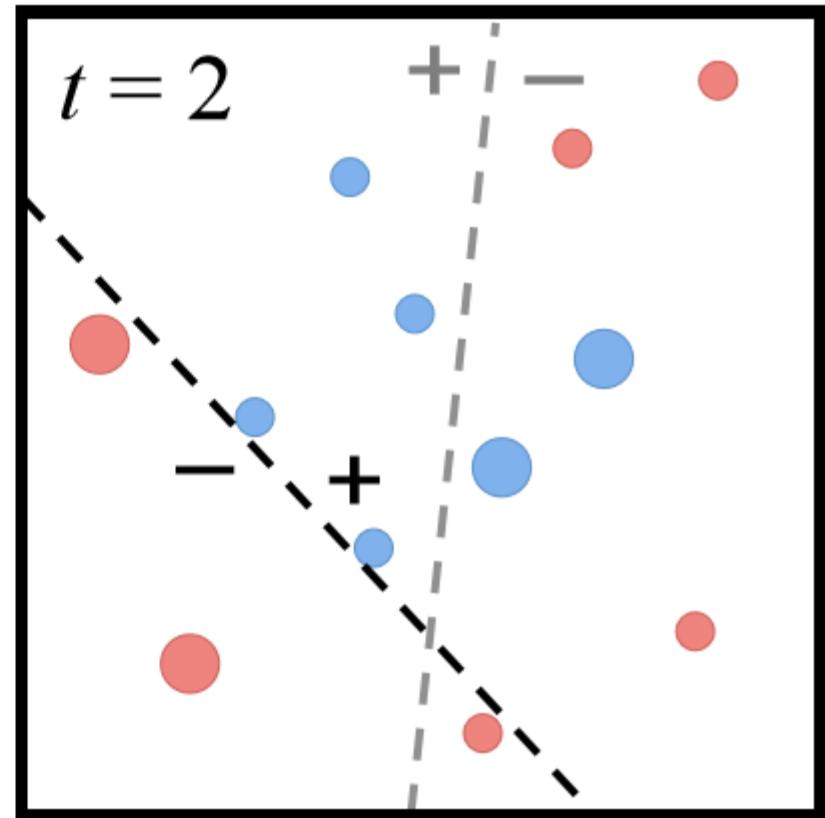
- $\beta_t$  measures the importance of  $h_t$
- If  $\epsilon_t \leq 0.5$ , then  $\beta_t \geq 0$  ( $\beta_t$  grows as  $\epsilon_t$  gets smaller)

# AdaBoost Algorithm

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:     Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:     Compute the weighted training error of  $h_t$
- 5:     Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6:     Update all instance weights:  

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$$
- 7:     Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



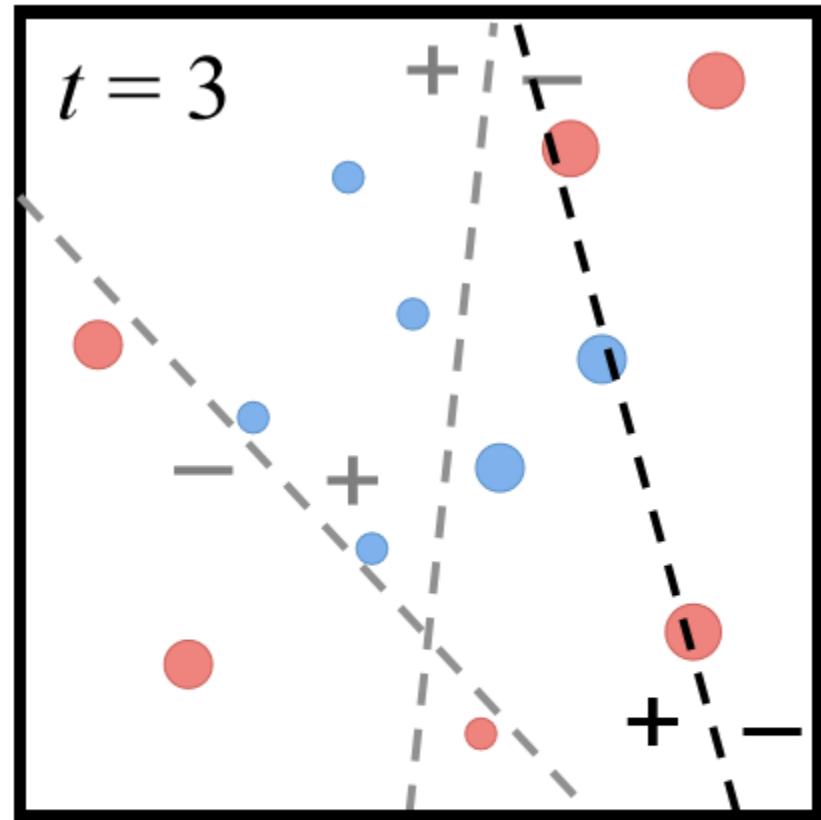
- Weights of correct predictions are multiplied by  $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by  $e^{\beta_t} \geq 1$

# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
   $H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$ 

```



- $\beta_t$  measures the importance of  $h_t$
- If  $\epsilon_t \leq 0.5$ , then  $\beta_t \geq 0$  ( $\beta_t$  grows as  $\epsilon_t$  gets smaller)

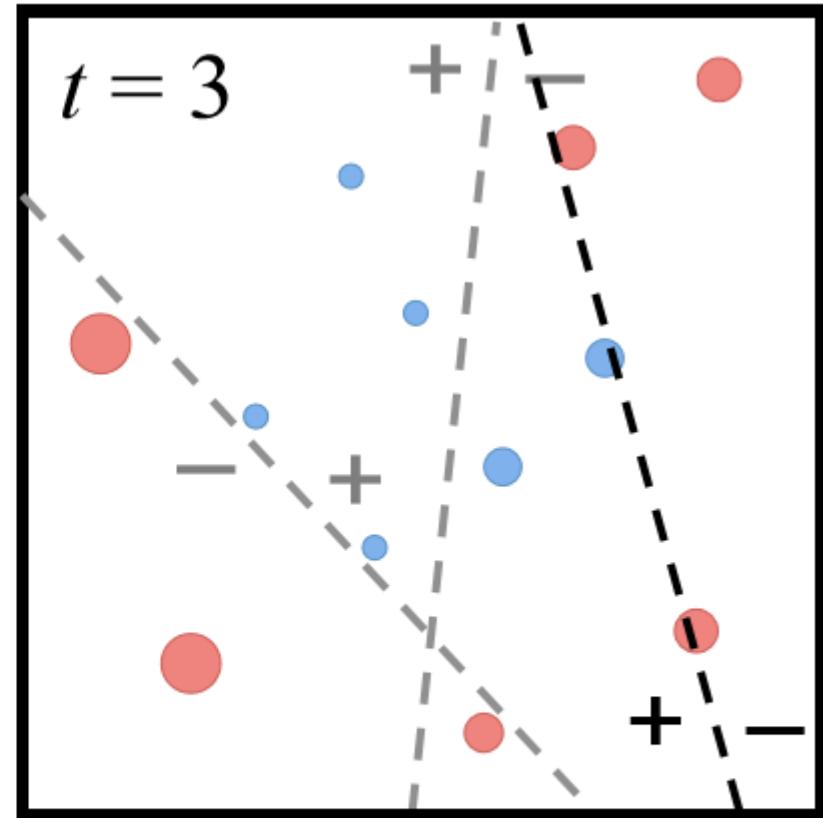
# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis

```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



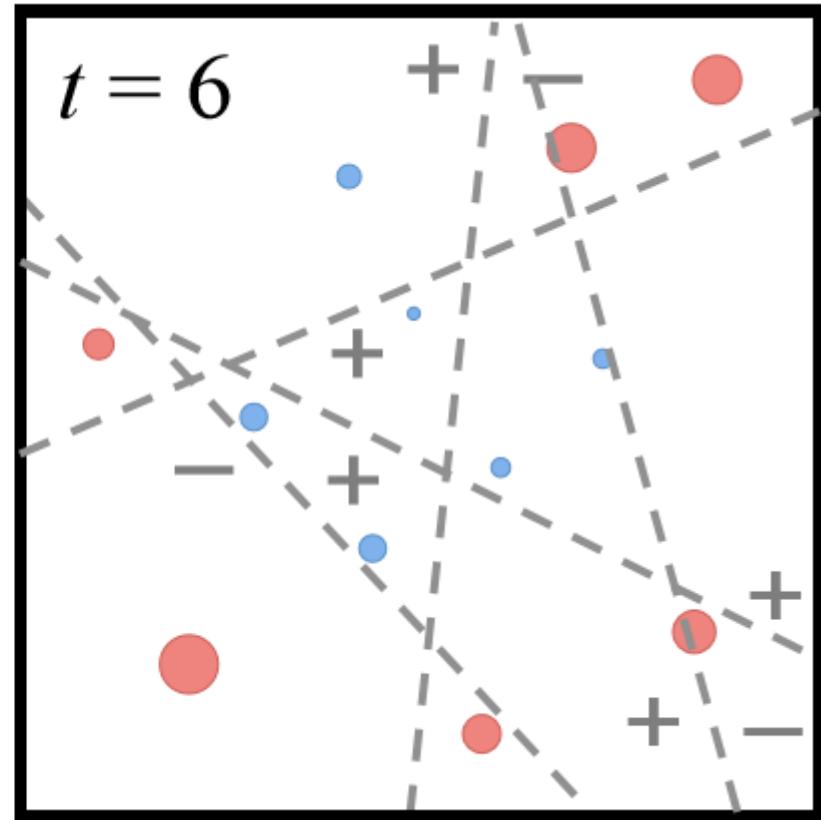
- Weights of correct predictions are multiplied by  $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by  $e^{\beta_t} \geq 1$

# AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



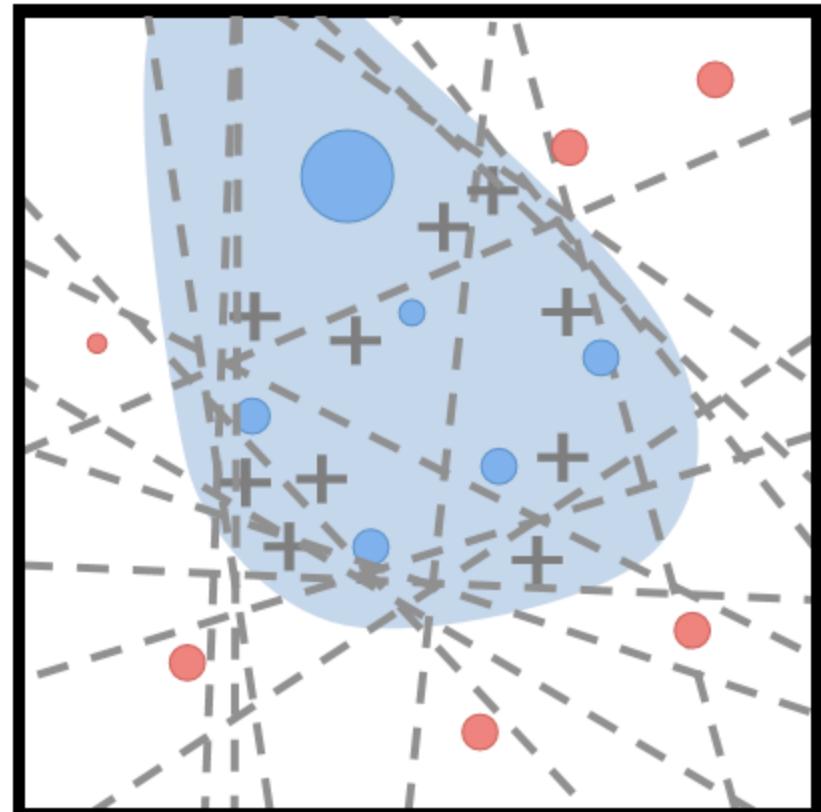
# AdaBoost Algorithm

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:     Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:     Compute the weighted training error of  $h_t$
- 5:     Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6:     Update all instance weights:  

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$$
- 7:     Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

$t = T$



- Final model is a **weighted combination** of members
  - Each member weighted by its importance

# AdaBoost Algorithm

**INPUT:** training data  $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ ,  
the number of iterations  $T$

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$

2: **for**  $t = 1, \dots, T$

3:     Train model  $h_t$  on  $X, y$  with instance weights  $\mathbf{w}_t$

4:     Compute the weighted training error rate of  $h_t$ :

$$\epsilon_t = \sum_{i:y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$$

5:     Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$

6:     Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i)) \quad \forall i = 1, \dots, n$$

7:     Normalize  $\mathbf{w}_{t+1}$  to be a distribution:

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^n w_{t+1,j}} \quad \forall i = 1, \dots, n$$

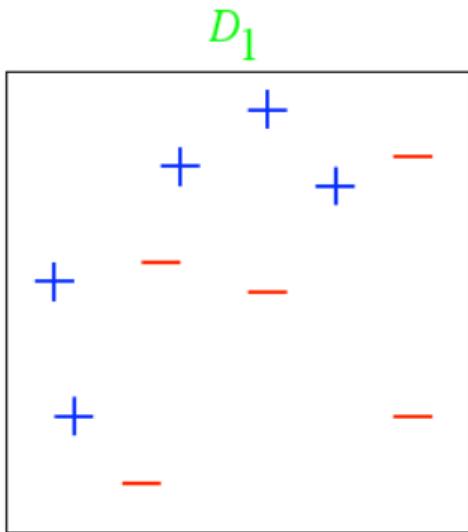
8: **end for**

9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

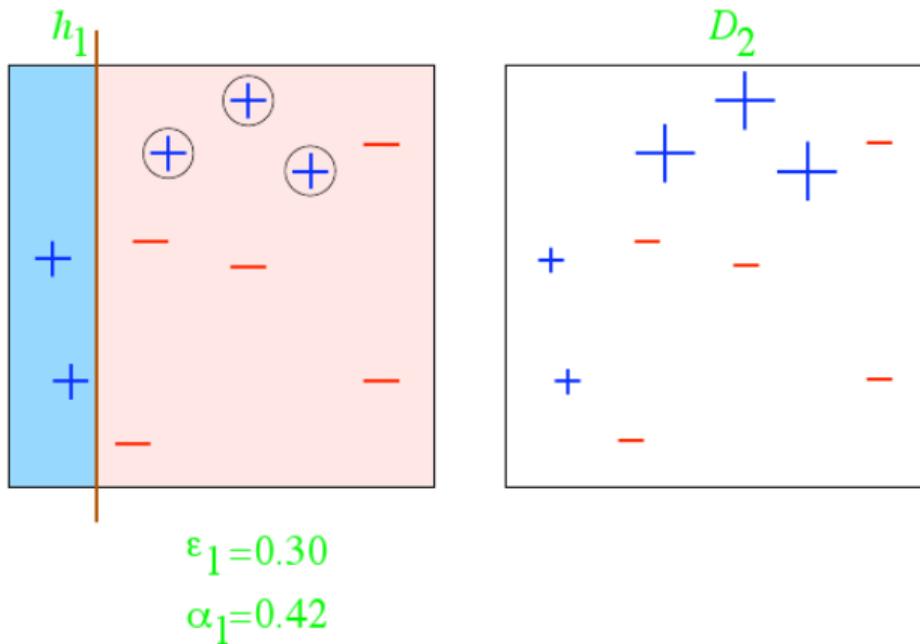
Member classifier with less error are given more weight in final ensemble hypothesis. Final prediction is a weighted combination of each members prediction

# Example



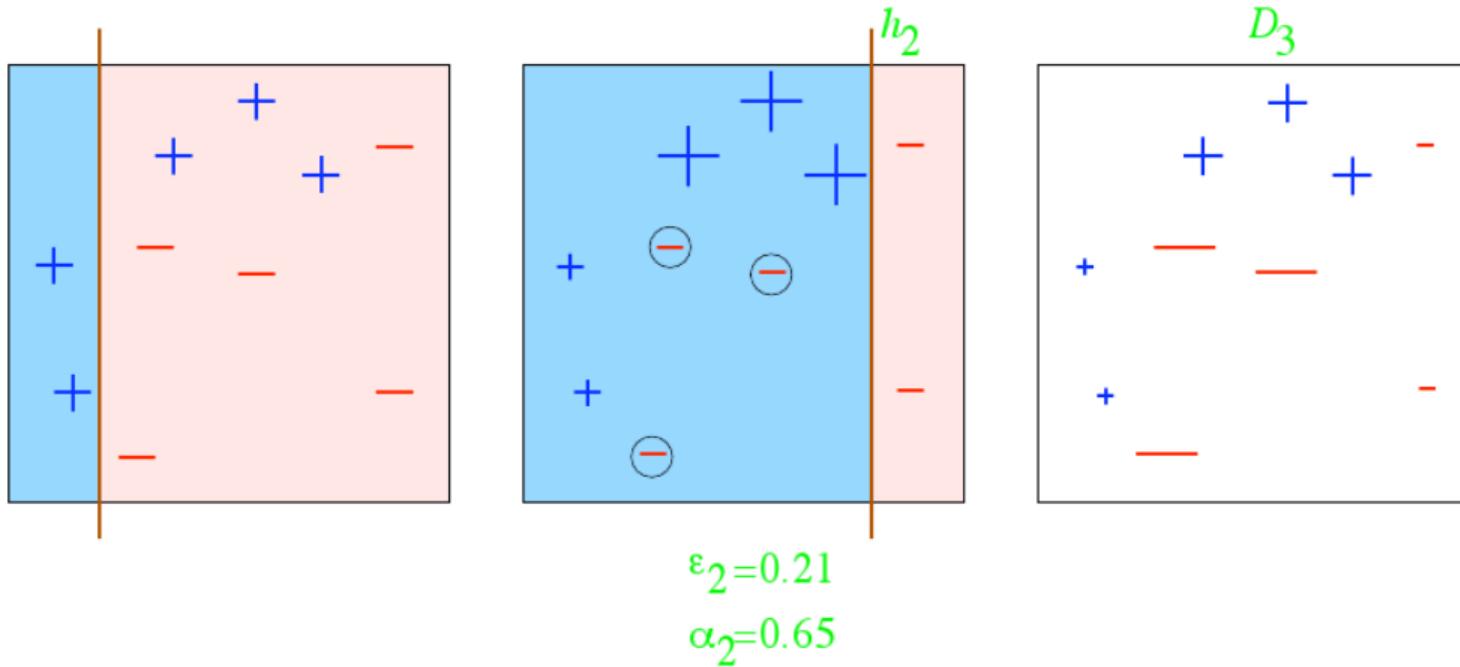
From, Léon Bottou

# Example



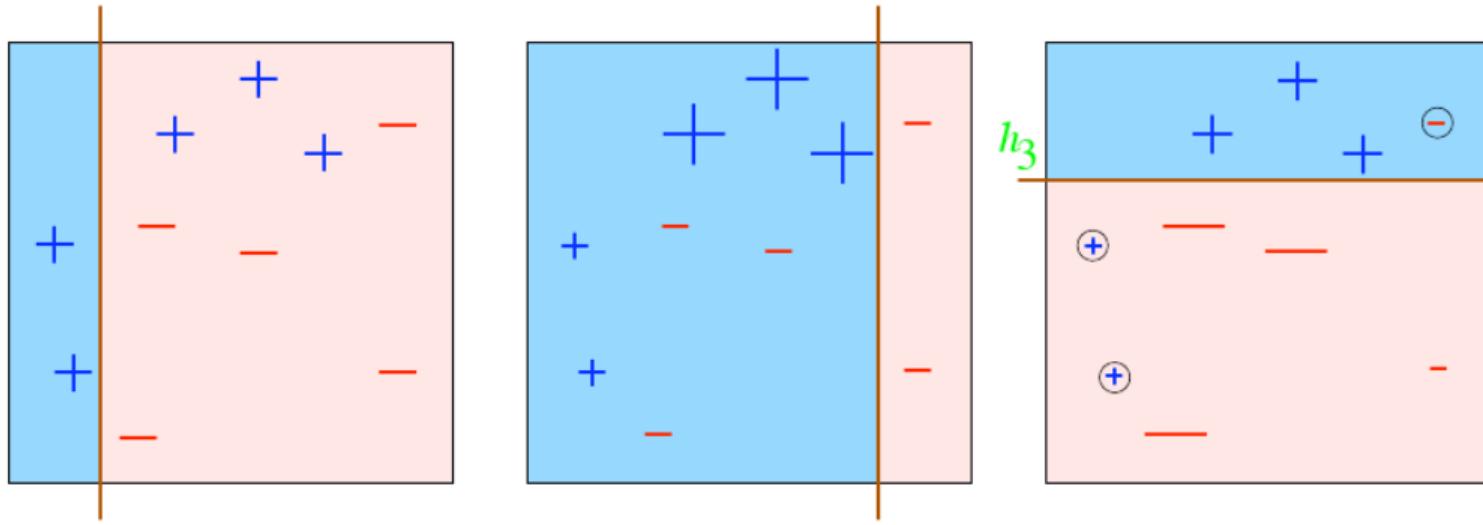
From, Léon Bottou

# Example



From, Léon Bottou

# Example

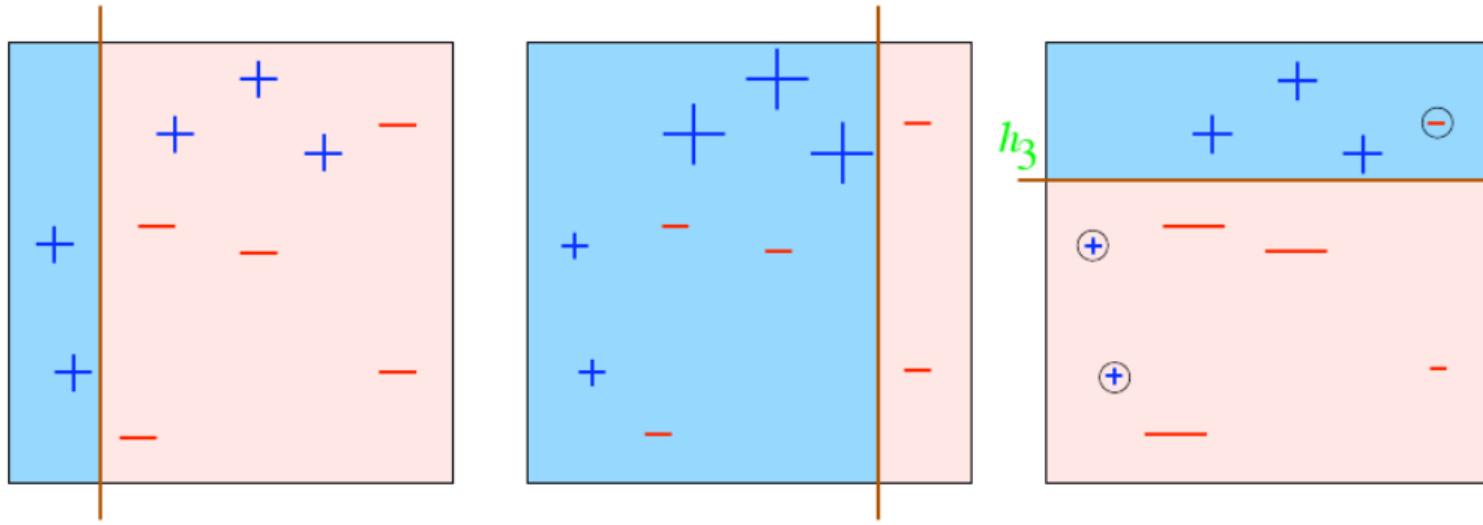


$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

From, Léon Bottou

# Example



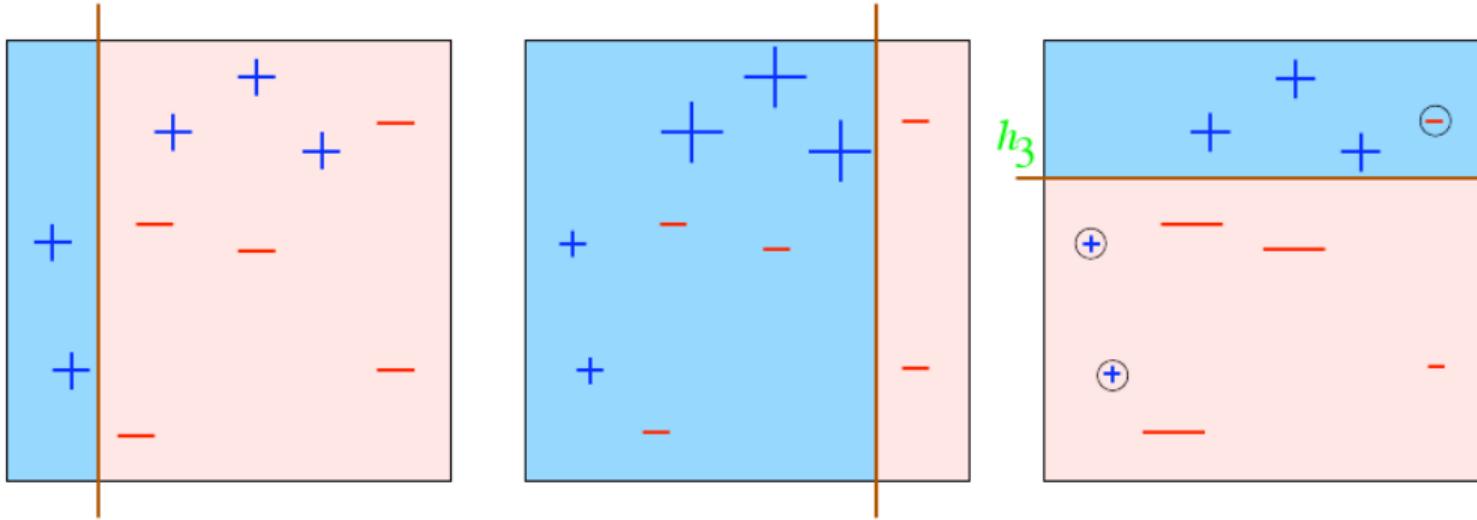
$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

From, Léon Bottou

# Example

How do we combine the results now?



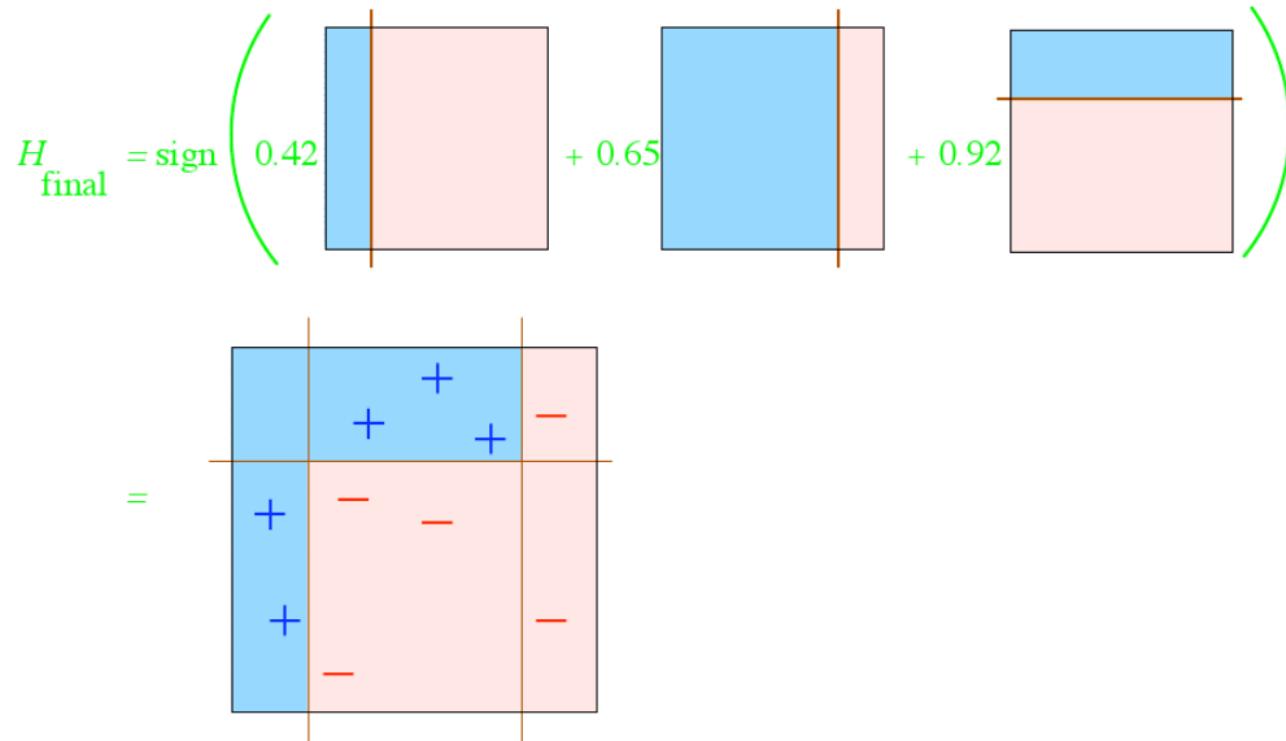
$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

From, Léon Bottou

# Example

How do we combine the results now?



From, Léon Bottou

# AdaBoost base learners

---

- AdaBoost works best with “weak” learners
    - Should not be complex
    - Typically high bias classifiers
    - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
      - Can prove training error goes to 0 in  $O(\log n)$  iterations
  - Examples:
    - Decision stumps (1 level decision trees)
    - Depth-limited decision trees
    - Linear classifiers
-

# AdaBoost in practice

---

## Strengths:

- Fast and simple to program
- No parameters to tune (besides T)
- No assumptions on weak learner

## When boosting can fail:

- Given insufficient data
- Overly complex weak hypotheses
- Can be susceptible to noise
- When there are a large number of outliers

# Fine Tuning Ensembles

---

- Model combination does not always guaranteed to decrease error, unless
  - base-learners are diverse and accurate
- Ignore poor base learners
  - Use accuracy as a cut-off
  - Introduce some pruning with which at each iteration remove poor learners / learners whose absence lead to improvement (if any)
    - Modify iterations to allow both additions / deletions of learners
  - Discarding appropriately leads to better performance

# References

---

The-Morgan-Kaufmann-Series-in-Data-Management-  
Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-  
Mining.-Concepts-and-Techniques-3rd-Edition-  
Morgan-Kaufmann-2011

Bishop - Pattern Recognition And Machine Learning -  
Springer 2006

A Gentle Introduction to Gradient Boosting  
Cheng Li chengli@ccs.neu.edu College of Computer  
and Information Science Northeastern University

[https://www.youtube.com/watch?time\\_continue=647  
&v=LsK-xG1cLYA&feature=emb\\_logo](https://www.youtube.com/watch?time_continue=647&v=LsK-xG1cLYA&feature=emb_logo)



# Thank You!

# A Gentle Introduction to Gradient Boosting

Cheng Li

chengli@ccs.neu.edu

College of Computer and Information Science  
Northeastern University

# Gradient Boosting

- ▶ a powerful machine learning algorithm
- ▶ it can do
  - ▶ regression
  - ▶ classification
  - ▶ ranking
- ▶ won Track 1 of the Yahoo Learning to Rank Challenge

Our implementation of Gradient Boosting is available at  
<https://github.com/cheng-li/pyramid>

# Outline of the Tutorial

- 1 What is Gradient Boosting
- 2 A brief history
- 3 Gradient Boosting for regression
- 4 Gradient Boosting for classification
- 5 A demo of Gradient Boosting
- 6 Relationship between Adaboost and Gradient Boosting
- 7 Why it works

**Note:** This tutorial focuses on the intuition. For a formal treatment, see [Friedman, 2001]

# What is Gradient Boosting

**Gradient Boosting = Gradient Descent + Boosting**

Adaboost

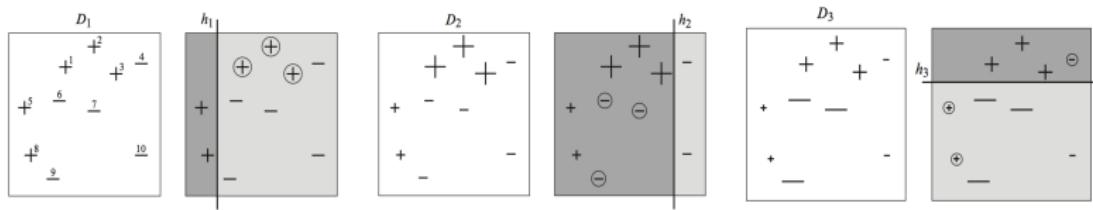


Figure: AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

# What is Gradient Boosting

**Gradient Boosting = Gradient Descent + Boosting**

Adaboost

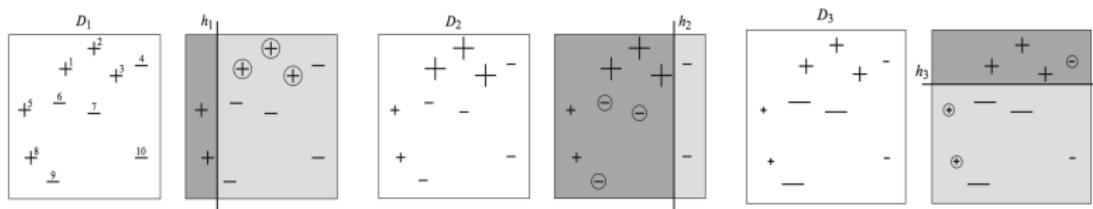


Figure: AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

- ▶ Fit an additive model (ensemble)  $\sum_t \rho_t h_t(x)$  in a forward stage-wise manner.
- ▶ In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- ▶ In Adaboost, “shortcomings” are identified by high-weight data points.

# What is Gradient Boosting

**Gradient Boosting = Gradient Descent + Boosting**

Adaboost

$$H(x) = \sum_t \rho_t h_t(x)$$

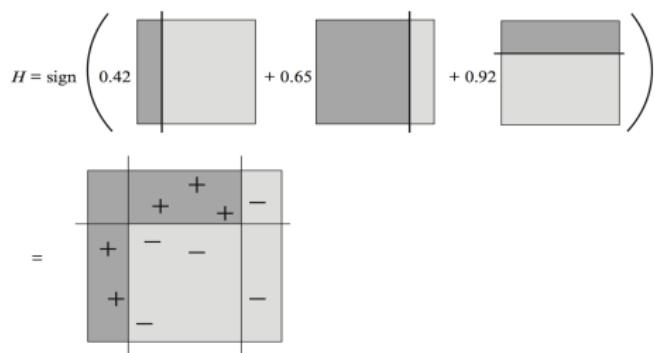


Figure: AdaBoost. Source: Figure 1.2 of [Schapire and Freund, 2012]

# What is Gradient Boosting

**Gradient Boosting = Gradient Descent + Boosting**

## Gradient Boosting

- ▶ Fit an additive model (ensemble)  $\sum_t \rho_t h_t(x)$  in a forward stage-wise manner.
- ▶ In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- ▶ In Gradient Boosting, “shortcomings” are identified by gradients.
- ▶ Recall that, in Adaboost, “shortcomings” are identified by high-weight data points.
- ▶ Both high-weight data points and gradients tell us how to improve our model.

# What is Gradient Boosting

Why and how did researchers invent Gradient Boosting?

# A Brief History of Gradient Boosting

- ▶ Invent Adaboost, the first successful boosting algorithm  
[Freund et al., 1996, Freund and Schapire, 1997]
- ▶ Formulate Adaboost as gradient descent with a special loss function [Breiman et al., 1998, Breiman, 1999]
- ▶ Generalize Adaboost to Gradient Boosting in order to handle a variety of loss functions  
[Friedman et al., 2000, Friedman, 2001]

# Gradient Boosting for Regression

## Gradient Boosting for Different Problems

Difficulty:

regression ==> classification ==> ranking

# Gradient Boosting for Regression

Let's play a game...

You are given  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , and the task is to fit a model  $F(x)$  to minimize square loss.

Suppose your friend wants to help you and gives you a model  $F$ .

You check his model and find the model is good but not perfect.

There are some mistakes:  $F(x_1) = 0.8$ , while  $y_1 = 0.9$ , and

$F(x_2) = 1.4$  while  $y_2 = 1.3\dots$  How can you improve this model?

# Gradient Boosting for Regression

Let's play a game...

You are given  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , and the task is to fit a model  $F(x)$  to minimize square loss.

Suppose your friend wants to help you and gives you a model  $F$ .

You check his model and find the model is good but not perfect.

There are some mistakes:  $F(x_1) = 0.8$ , while  $y_1 = 0.9$ , and

$F(x_2) = 1.4$  while  $y_2 = 1.3\dots$  How can you improve this model?

**Rule of the game:**

- ▶ You are not allowed to remove anything from  $F$  or change any parameter in  $F$ .

# Gradient Boosting for Regression

Let's play a game...

You are given  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , and the task is to fit a model  $F(x)$  to minimize square loss.

Suppose your friend wants to help you and gives you a model  $F$ .

You check his model and find the model is good but not perfect.

There are some mistakes:  $F(x_1) = 0.8$ , while  $y_1 = 0.9$ , and

$F(x_2) = 1.4$  while  $y_2 = 1.3\dots$  How can you improve this model?

**Rule of the game:**

- ▶ You are not allowed to remove anything from  $F$  or change any parameter in  $F$ .
- ▶ You can add an additional model (regression tree)  $h$  to  $F$ , so the new prediction will be  $F(x) + h(x)$ .

# Gradient Boosting for Regression

Simple solution:

You wish to improve the model such that

$$F(x_1) + h(x_1) = y_1$$

$$F(x_2) + h(x_2) = y_2$$

...

$$F(x_n) + h(x_n) = y_n$$

# Gradient Boosting for Regression

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$

$$h(x_2) = y_2 - F(x_2)$$

...

$$h(x_n) = y_n - F(x_n)$$

# Gradient Boosting for Regression

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$

$$h(x_2) = y_2 - F(x_2)$$

...

$$h(x_n) = y_n - F(x_n)$$

Can any regression tree  $h$  achieve this goal perfectly?

# Gradient Boosting for Regression

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$

$$h(x_2) = y_2 - F(x_2)$$

...

$$h(x_n) = y_n - F(x_n)$$

Can any regression tree  $h$  achieve this goal perfectly?

Maybe not....

# Gradient Boosting for Regression

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$

$$h(x_2) = y_2 - F(x_2)$$

...

$$h(x_n) = y_n - F(x_n)$$

Can any regression tree  $h$  achieve this goal perfectly?

Maybe not....

But some regression tree might be able to do this approximately.

# Gradient Boosting for Regression

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$

$$h(x_2) = y_2 - F(x_2)$$

...

$$h(x_n) = y_n - F(x_n)$$

Can any regression tree  $h$  achieve this goal perfectly?

Maybe not....

But some regression tree might be able to do this approximately.

How?

# Gradient Boosting for Regression

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$

$$h(x_2) = y_2 - F(x_2)$$

...

$$h(x_n) = y_n - F(x_n)$$

Can any regression tree  $h$  achieve this goal perfectly?

Maybe not....

But some regression tree might be able to do this approximately.

How?

Just fit a regression tree  $h$  to data

$$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), \dots, (x_n, y_n - F(x_n))$$

# Gradient Boosting for Regression

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$

$$h(x_2) = y_2 - F(x_2)$$

...

$$h(x_n) = y_n - F(x_n)$$

Can any regression tree  $h$  achieve this goal perfectly?

Maybe not....

But some regression tree might be able to do this approximately.

How?

Just fit a regression tree  $h$  to data

$$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), \dots, (x_n, y_n - F(x_n))$$

Congratulations, you get a better model!

# Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$  are called **residuals**. These are the parts that existing model  $F$  cannot do well.

The role of  $h$  is to compensate the shortcoming of existing model  $F$ .

# Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$  are called **residuals**. These are the parts that existing model  $F$  cannot do well.

The role of  $h$  is to compensate the shortcoming of existing model  $F$ .

If the new model  $F + h$  is still not satisfactory,

# Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$  are called **residuals**. These are the parts that existing model  $F$  cannot do well.

The role of  $h$  is to compensate the shortcoming of existing model  $F$ .

If the new model  $F + h$  is still not satisfactory, we can add another regression tree...

# Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$  are called **residuals**. These are the parts that existing model  $F$  cannot do well.

The role of  $h$  is to compensate the shortcoming of existing model  $F$ .

If the new model  $F + h$  is still not satisfactory, we can add another regression tree...

We are improving the predictions of training data, is the procedure also useful for test data?

# Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$  are called **residuals**. These are the parts that existing model  $F$  cannot do well.

The role of  $h$  is to compensate the shortcoming of existing model  $F$ .

If the new model  $F + h$  is still not satisfactory, we can add another regression tree...

We are improving the predictions of training data, is the procedure also useful for test data?

Yes! Because we are building a model, and the model can be applied to test data as well.

# Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$  are called **residuals**. These are the parts that existing model  $F$  cannot do well.

The role of  $h$  is to compensate the shortcoming of existing model  $F$ .

If the new model  $F + h$  is still not satisfactory, we can add another regression tree...

We are improving the predictions of training data, is the procedure also useful for test data?

Yes! Because we are building a model, and the model can be applied to test data as well.

How is this related to gradient descent?

# Gradient Boosting for Regression

## Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

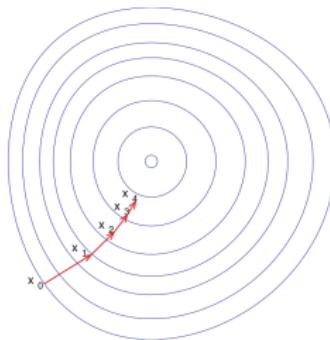


Figure: Gradient Descent. Source:

[http://en.wikipedia.org/wiki/Gradient\\_descent](http://en.wikipedia.org/wiki/Gradient_descent)

# Gradient Boosting for Regression

How is this related to gradient descent?

Loss function  $L(y, F(x)) = (y - F(x))^2/2$

We want to minimize  $J = \sum_i L(y_i, F(x_i))$  by adjusting  $F(x_1), F(x_2), \dots, F(x_n)$ .

Notice that  $F(x_1), F(x_2), \dots, F(x_n)$  are just some numbers. We can treat  $F(x_i)$  as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

# Gradient Boosting for Regression

How is this related to gradient descent?

$$F(x_i) := F(x_i) + h(x_i)$$

$$F(x_i) := F(x_i) + y_i - F(x_i)$$

$$F(x_i) := F(x_i) - 1 \frac{\partial J}{\partial F(x_i)}$$

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

# Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

*residual  $\Leftrightarrow$  negative gradient*

*fit  $h$  to residual  $\Leftrightarrow$  fit  $h$  to negative gradient*

*update  $F$  based on residual  $\Leftrightarrow$  update  $F$  based on negative gradient*

# Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

*residual  $\Leftrightarrow$  negative gradient*

*fit  $h$  to residual  $\Leftrightarrow$  fit  $h$  to negative gradient*

*update  $F$  based on residual  $\Leftrightarrow$  update  $F$  based on negative gradient*

So we are actually updating our model using **gradient descent!**

# Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

*residual  $\Leftrightarrow$  negative gradient*

*fit  $h$  to residual  $\Leftrightarrow$  fit  $h$  to negative gradient*

*update  $F$  based on residual  $\Leftrightarrow$  update  $F$  based on negative gradient*

So we are actually updating our model using **gradient descent!**

It turns out that the concept of **gradients** is more general and useful than the concept of **residuals**. So from now on, let's stick with gradients. The reason will be explained later.

# Gradient Boosting for Regression

## Regression with square Loss

Let us summarize the algorithm we just derived using the concept of gradients. Negative gradient:

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

start with an initial model, say,  $F(x) = \frac{\sum_{i=1}^n y_i}{n}$

iterate until converge:

calculate negative gradients  $-g(x_i)$

fit a regression tree  $h$  to negative gradients  $-g(x_i)$

$F := F + \rho h$ , where  $\rho = 1$

# Gradient Boosting for Regression

## Regression with square Loss

Let us summarize the algorithm we just derived using the concept of gradients. Negative gradient:

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

start with an initial model, say,  $F(x) = \frac{\sum_{i=1}^n y_i}{n}$

iterate until converge:

calculate negative gradients  $-g(x_i)$

fit a regression tree  $h$  to negative gradients  $-g(x_i)$

$F := F + \rho h$ , where  $\rho = 1$

The benefit of formulating this algorithm using gradients is that it allows us to consider other loss functions and derive the corresponding algorithms in the same way.

# Gradient Boosting for Regression

## Loss Functions for Regression Problem

Why do we need to consider other loss functions? Isn't square loss good enough?

# Gradient Boosting for Regression

## Loss Functions for Regression Problem

Square loss is:

- ✓ Easy to deal with mathematically
- ✗ Not robust to outliers

Outliers are heavily punished because the error is squared.

Example:

$y_i$	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

Consequence?

# Gradient Boosting for Regression

## Loss Functions for Regression Problem

Square loss is:

- ✓ Easy to deal with mathematically
- ✗ Not robust to outliers

Outliers are heavily punished because the error is squared.

Example:

$y_i$	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

Consequence?

Pay too much attention to outliers. Try hard to incorporate outliers into the model. Degrade the overall performance.

# Gradient Boosting for Regression

## Loss Functions for Regression Problem

- ▶ Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

# Gradient Boosting for Regression

## Loss Functions for Regression Problem

- ▶ Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

- ▶ Huber loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

# Gradient Boosting for Regression

## Loss Functions for Regression Problem

- ▶ Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

- ▶ Huber loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

$y_i$	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
Square loss	0.005	0.02	0.125	5.445
Absolute loss	0.1	0.2	0.5	3.3
Huber loss( $\delta = 0.5$ )	0.005	0.02	0.125	1.525

# Gradient Boosting for Regression

## Regression with Absolute Loss

Negative gradient:

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = \text{sign}(y_i - F(x_i))$$

start with an initial model, say,  $F(x) = \frac{\sum_{i=1}^n y_i}{n}$

iterate until converge:

calculate gradients  $-g(x_i)$

fit a regression tree  $h$  to negative gradients  $-g(x_i)$

$F := F + \rho h$

# Gradient Boosting for Regression

## Regression with Huber Loss

Negative gradient:

$$\begin{aligned}-g(x_i) &= -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \\&= \begin{cases} y_i - F(x_i) & |y_i - F(x_i)| \leq \delta \\ \delta sign(y_i - F(x_i)) & |y_i - F(x_i)| > \delta \end{cases}\end{aligned}$$

start with an initial model, say,  $F(x) = \frac{\sum_{i=1}^n y_i}{n}$

iterate until converge:

calculate negative gradients  $-g(x_i)$

fit a regression tree  $h$  to negative gradients  $-g(x_i)$

$F := F + \rho h$

# Gradient Boosting for Regression

Regression with loss function  $L$ : general procedure

Give any differentiable loss function  $L$

start with an initial model, say  $F(x) = \frac{\sum_{i=1}^n y_i}{n}$

iterate until converge:

calculate negative gradients  $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$

fit a regression tree  $h$  to negative gradients  $-g(x_i)$

$F := F + \rho h$

# Gradient Boosting for Regression

Regression with loss function  $L$ : general procedure

Give any differentiable loss function  $L$

start with an initial model, say  $F(x) = \frac{\sum_{i=1}^n y_i}{n}$

iterate until converge:

calculate negative gradients  $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$

fit a regression tree  $h$  to negative gradients  $-g(x_i)$

$F := F + \rho h$

In general,

*negative gradients  $\not\Rightarrow$  residuals*

We should follow negative gradients rather than residuals. Why?

# Gradient Boosting for Regression

## Negative Gradient vs Residual: An Example

Huber loss

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

Update by Negative Gradient:

$$h(x_i) = -g(x_i) = \begin{cases} y_i - F(x_i) & |y_i - F(x_i)| \leq \delta \\ \delta sign(y_i - F(x_i)) & |y_i - F(x_i)| > \delta \end{cases}$$

Update by Residual:

$$h(x_i) = y_i - F(x_i)$$

Difference: negative gradient pays less attention to outliers.

# Gradient Boosting for Regression

## Summary of the Section

- ▶ Fit an additive model  $F = \sum_t \rho_t h_t$  in a forward stage-wise manner.
- ▶ In each stage, introduce a new regression tree  $h$  to compensate the shortcomings of existing model.
- ▶ The “shortcomings” are identified by negative gradients.
- ▶ For any loss function, we can derive a gradient boosting algorithm.
- ▶ Absolute loss and Huber loss are more robust to outliers than square loss.

## Things not covered

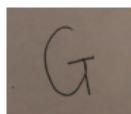
How to choose a proper learning rate for each gradient boosting algorithm. See [Friedman, 2001]

# Gradient Boosting for Classification

## Problem

Recognize the given hand written capital letter.

- ▶ Multi-class classification
- ▶ 26 classes. A,B,C,...,Z

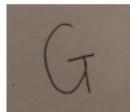


## Data Set

- ▶ <http://archive.ics.uci.edu/ml/datasets/Letter+Recognition>
- ▶ 20000 data points, 16 features

# Gradient Boosting for Classification

## Feature Extraction



1	horizontal position of box	9	mean y variance
2	vertical position of box	10	mean x * y correlation
3	width of box	11	mean of x * x * y
4	height of box	12	mean of x * y * y
5	total number on pixels	13	mean edge count left to right
6	mean x of on pixels in box	14	correlation of x-ege with y
7	mean y of on pixels in box	15	mean edge count bottom to top
8	mean x variance	16	correlation of y-ege with x

Feature Vector= (2, 1, 3, 1, 1, 8, 6, 6, 6, 6, 5, 9, 1, 7, 5, 10)

Label = G

# Gradient Boosting for Classification

## Model

- ▶ 26 score functions (our models):  $F_A, F_B, F_C, \dots, F_Z$ .
- ▶  $F_A(x)$  assigns a score for class A
- ▶ scores are used to calculate probabilities

$$P_A(x) = \frac{e^{F_A(x)}}{\sum_{c=A}^Z e^{F_c(x)}}$$

$$P_B(x) = \frac{e^{F_B(x)}}{\sum_{c=A}^Z e^{F_c(x)}}$$

...

$$P_Z(x) = \frac{e^{F_Z(x)}}{\sum_{c=A}^Z e^{F_c(x)}}$$

- ▶ predicted label = class that has the highest probability

## Loss Function for each data point

Step 1 turn the label  $y_i$  into a (true) probability distribution  $Y_c(x_i)$

For example:  $y_5=G$ ,

$$Y_A(x_5) = 0, Y_B(x_5) = 0, \dots, Y_G(x_5) = 1, \dots, Y_Z(x_5) = 0$$

# Gradient Boosting for Classification

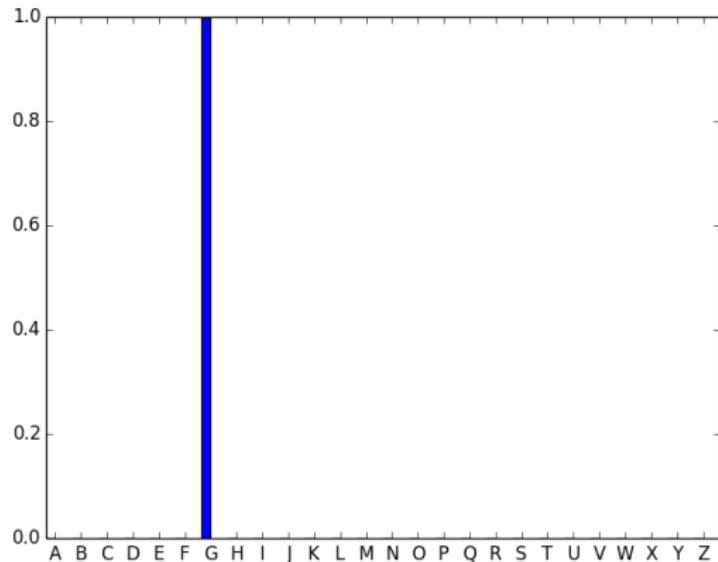


Figure: true probability distribution

## Loss Function for each data point

Step 1 turn the label  $y_i$  into a (true) probability distribution  $Y_c(x_i)$

For example:  $y_5 = G$ ,

$$Y_A(x_5) = 0, Y_B(x_5) = 0, \dots, Y_G(x_5) = 1, \dots, Y_Z(x_5) = 0$$

Step 2 calculate the predicted probability distribution  $P_c(x_i)$  based on the current model  $F_A, F_B, \dots, F_Z$ .

$$P_A(x_5) = 0.03, P_B(x_5) = 0.05, \dots, P_G(x_5) = 0.3, \dots, P_Z(x_5) = 0.05$$

# Gradient Boosting for Classification

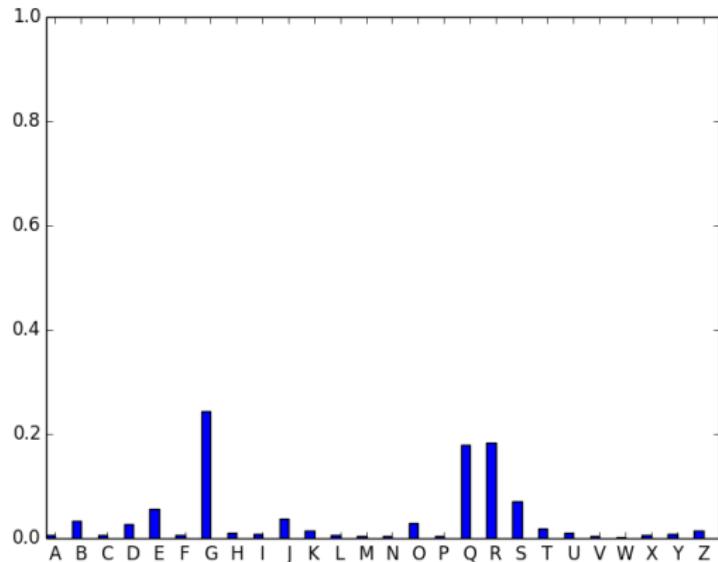


Figure: predicted probability distribution based on current model

## Loss Function for each data point

**Step 1** turn the label  $y_i$  into a (true) probability distribution  $Y_c(x_i)$

For example:  $y_5=G$ ,

$$Y_A(x_5) = 0, Y_B(x_5) = 0, \dots, Y_G(x_5) = 1, \dots, Y_Z(x_5) = 0$$

**Step 2** calculate the predicted probability distribution  $P_c(x_i)$  based on the current model  $F_A, F_B, \dots, F_Z$ .

$$P_A(x_5) = 0.03, P_B(x_5) = 0.05, \dots, P_G(x_5) = 0.3, \dots, P_Z(x_5) = 0.05$$

**Step 3** calculate the difference between the true probability distribution and the predicted probability distribution.  
Here we use KL-divergence

# Gradient Boosting for Classification

## Goal

- ▶ minimize the total loss (KL-divergence)
- ▶ for each data point, we wish the predicted probability distribution to match the true probability distribution as closely as possible

# Gradient Boosting for Classification

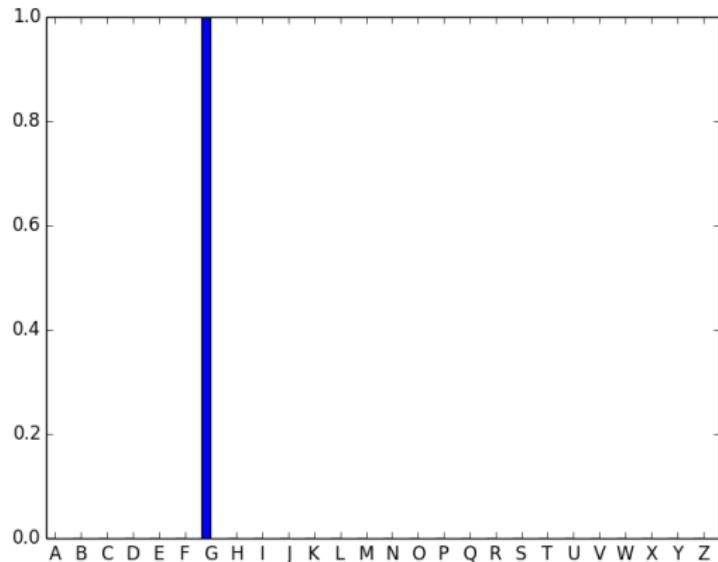


Figure: true probability distribution

# Gradient Boosting for Classification

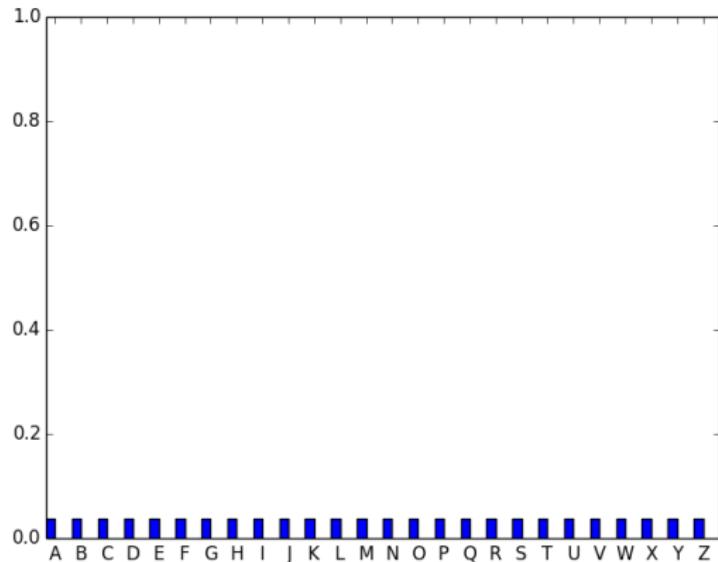


Figure: predicted probability distribution at round 0

# Gradient Boosting for Classification

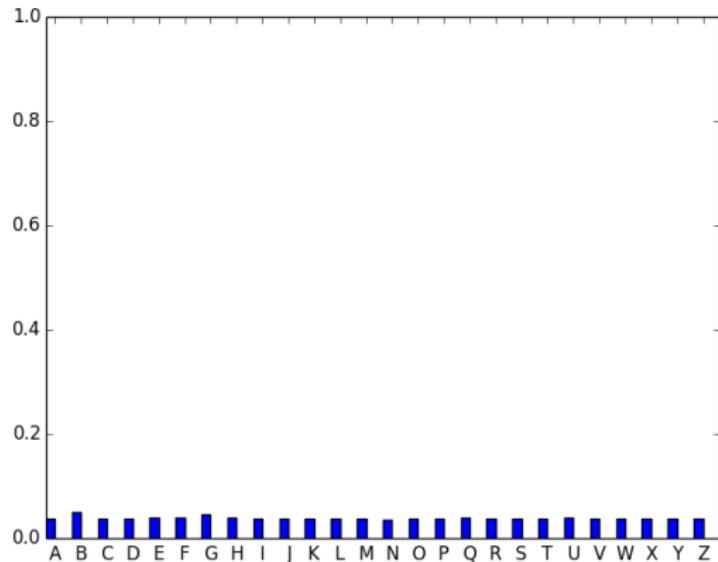


Figure: predicted probability distribution at round 1

# Gradient Boosting for Classification

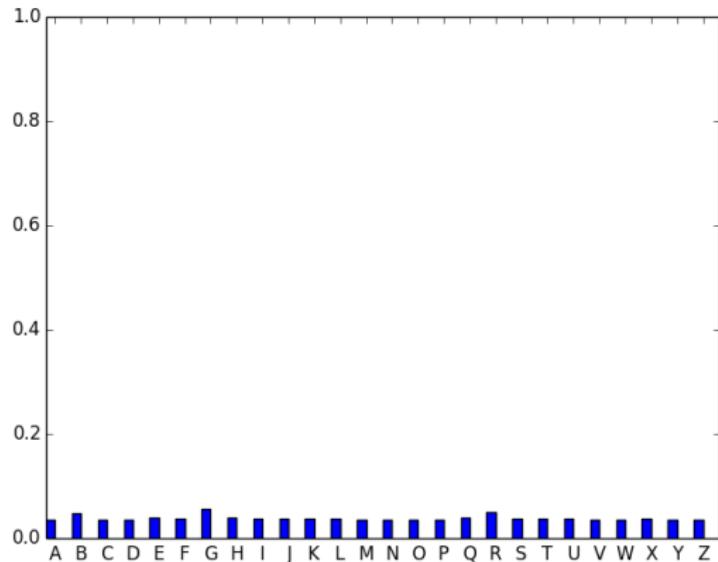


Figure: predicted probability distribution at round 2

# Gradient Boosting for Classification

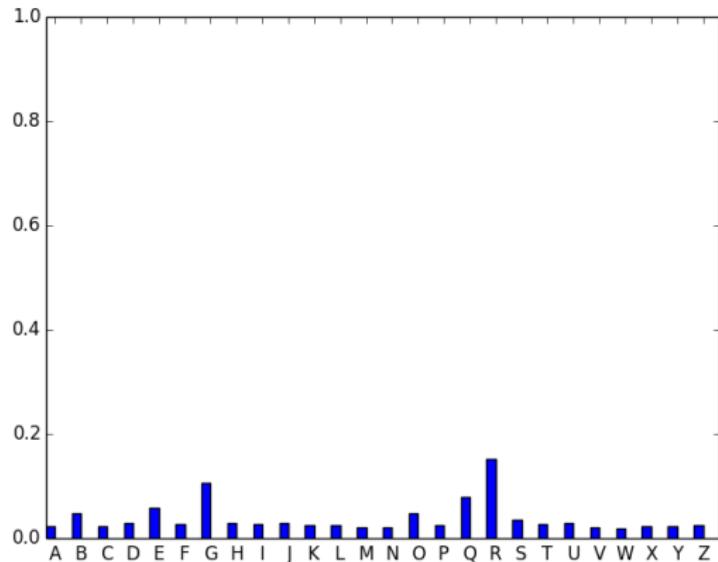


Figure: predicted probability distribution at round 10

# Gradient Boosting for Classification

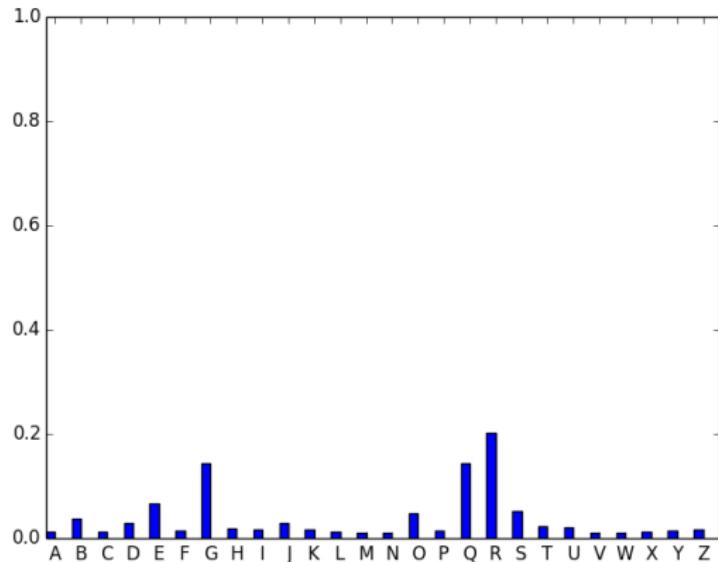


Figure: predicted probability distribution at round 20

# Gradient Boosting for Classification

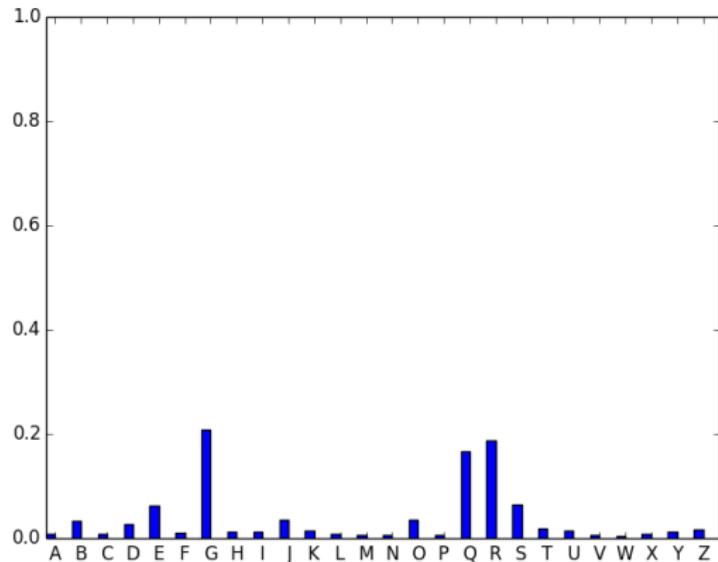


Figure: predicted probability distribution at round 30

# Gradient Boosting for Classification

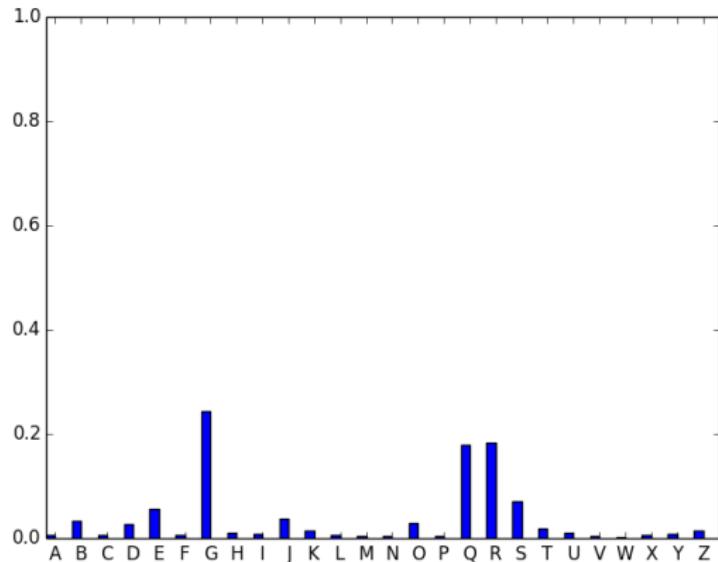


Figure: predicted probability distribution at round 40

# Gradient Boosting for Classification

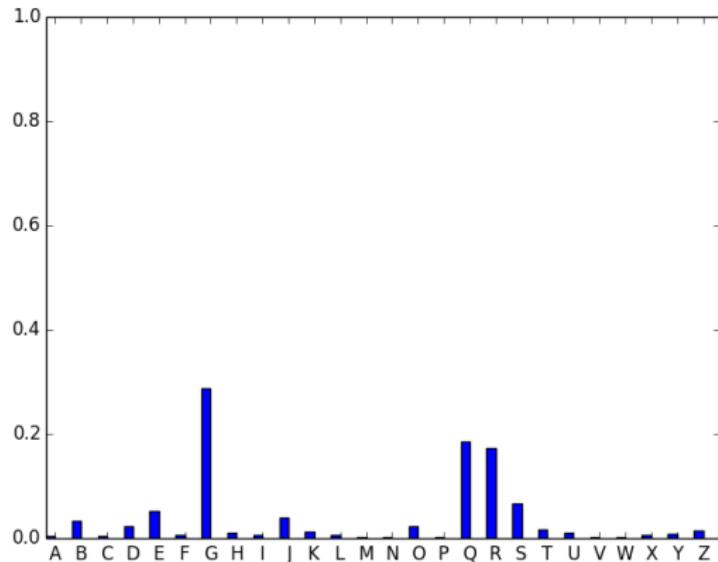


Figure: predicted probability distribution at round 50

# Gradient Boosting for Classification

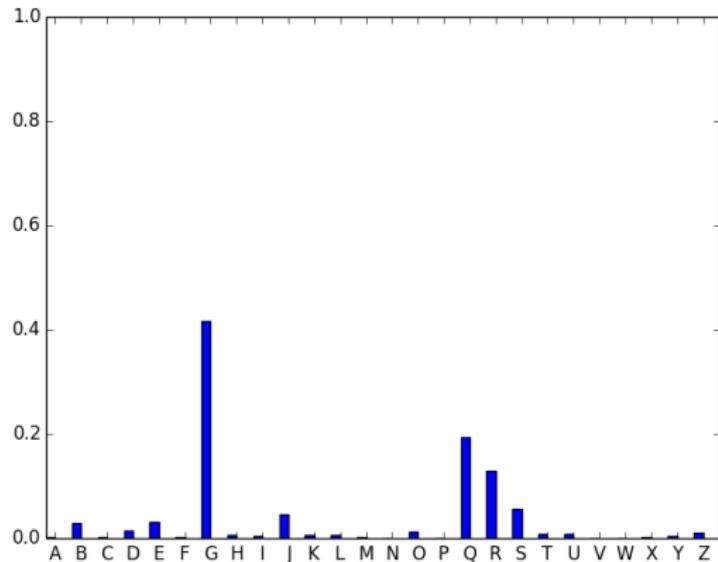


Figure: predicted probability distribution at round 100

# Gradient Boosting for Classification

## Goal

- ▶ minimize the total loss (KL-divergence)
- ▶ for each data point, we wish the predicted probability distribution to match the true probability distribution as closely as possible
- ▶ we achieve this goal by adjusting our models  $F_A, F_B, \dots, F_Z$ .

# Gradient Boosting for Regression: Review

Regression with loss function  $L$ : general procedure

Give any differentiable loss function  $L$

start with an initial model  $F$

iterate until converge:

calculate negative gradients  $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$

fit a regression tree  $h$  to negative gradients  $-g(x_i)$

$F := F + \rho h$

# Gradient Boosting for Classification

## Differences

- ▶  $F_A, F_B, \dots, F_Z$  **vs**  $F$
- ▶ a matrix of parameters to optimize **vs** a column of parameters to optimize

$F_A(x_1)$	$F_B(x_1)$	...	$F_Z(x_1)$
$F_A(x_2)$	$F_B(x_2)$	...	$F_Z(x_2)$
...	...	...	...
$F_A(x_n)$	$F_B(x_n)$	...	$F_Z(x_n)$

- ▶ a matrix of gradients **vs** a column of gradients

$\frac{\partial L}{F_A(x_1)}$	$\frac{\partial L}{F_B(x_1)}$	...	$\frac{\partial L}{F_Z(x_1)}$
$\frac{\partial L}{F_A(x_2)}$	$\frac{\partial L}{F_B(x_2)}$	...	$\frac{\partial L}{F_Z(x_2)}$
...	...	...	...
$\frac{\partial L}{F_A(x_n)}$	$\frac{\partial L}{F_B(x_n)}$	...	$\frac{\partial L}{F_Z(x_n)}$

# Gradient Boosting for Classification

start with initial models  $F_A, F_B, F_C, \dots, F_Z$

iterate until converge:

calculate negative gradients for class A:  $-g_A(x_i) = -\frac{\partial L}{\partial F_A(x_i)}$

calculate negative gradients for class B:  $-g_B(x_i) = -\frac{\partial L}{\partial F_B(x_i)}$

...

calculate negative gradients for class Z:  $-g_Z(x_i) = -\frac{\partial L}{\partial F_Z(x_i)}$

fit a regression tree  $h_A$  to negative gradients  $-g_A(x_i)$

fit a regression tree  $h_B$  to negative gradients  $-g_B(x_i)$

...

fit a regression tree  $h_Z$  to negative gradients  $-g_Z(x_i)$

$$F_A := F_A + \rho_A h_A$$

$$F_B := F_A + \rho_B h_B$$

...

$$F_Z := F_A + \rho_Z h_Z$$

# Gradient Boosting for Classification

start with initial models  $F_A, F_B, F_C, \dots, F_Z$

iterate until converge:

calculate negative gradients for class A:  $-g_A(x_i) = Y_A(x_i) - P_A(x_i)$

calculate negative gradients for class B:  $-g_B(x_i) = Y_B(x_i) - P_B(x_i)$

...

calculate negative gradients for class Z:  $-g_Z(x_i) = Y_Z(x_i) - P_Z(x_i)$

fit a regression tree  $h_A$  to negative gradients  $-g_A(x_i)$

fit a regression tree  $h_B$  to negative gradients  $-g_B(x_i)$

...

fit a regression tree  $h_Z$  to negative gradients  $-g_Z(x_i)$

$$F_A := F_A + \rho_A h_A$$

$$F_B := F_A + \rho_B h_B$$

...

$$F_Z := F_A + \rho_Z h_Z$$

# Gradient Boosting for Classification

round 0

i	y	$Y_A$	$Y_B$	$Y_C$	$Y_D$	$Y_E$	$Y_F$	$Y_G$	$Y_H$	$Y_I$	$Y_J$	$Y_K$	$Y_L$	$Y_M$	$Y_N$	$Y_O$	$Y_P$	$Y_Q$	$Y_R$	$Y_S$	$Y_T$	$Y_U$	$Y_V$	$Y_W$	$Y_X$	$Y_Y$	$Y_Z$	
1	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	I	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	$F_A$	$F_B$	$F_C$	$F_D$	$F_E$	$F_F$	$F_G$	$F_H$	$F_I$	$F_J$	$F_K$	$F_L$	$F_M$	$F_N$	$F_O$	$F_P$	$F_Q$	$F_R$	$F_S$	$F_T$	$F_U$	$F_V$	$F_W$	$F_X$	$F_Y$	$F_Z$	
1	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	$P_A$	$P_B$	$P_C$	$P_D$	$P_E$	$P_F$	$P_G$	$P_H$	$P_I$	$P_J$	$P_K$	$P_L$	$P_M$	$P_N$	$P_O$	$P_P$	$P_Q$	$P_R$	$P_S$	$P_T$	$P_U$	$P_V$	$P_W$	$P_X$	$P_Y$	$P_Z$	
1	T	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	
2	I	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	
3	D	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	
4	N	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	
5	G	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	$Y_A - P_A$	$Y_B - P_B$	$Y_C - P_C$	$Y_D - P_D$	$Y_E - P_E$	$Y_F - P_F$	$Y_G - P_G$	$Y_H - P_H$	$Y_I - P_I$	$Y_J - P_J$	$Y_K - P_K$	$Y_L - P_L$	$Y_M - P_M$	$Y_N - P_N$	$Y_O - P_O$	$Y_P - P_P$	$Y_Q - P_Q$	$Y_R - P_R$	$Y_S - P_S$	$Y_T - P_T$	$Y_U - P_U$	$Y_V - P_V$	$Y_W - P_W$	$Y_X - P_X$	$Y_Y - P_Y$	$Y_Z - P_Z$	
1	T	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
2	I	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
3	D	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
4	N	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
5	G	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

# Gradient Boosting for Classification

$$h_A(x) = \begin{cases} 0.98 & \text{feature 10 of } x \leq 2.0 \\ -0.07 & \text{feature 10 of } x > 2.0 \end{cases}$$

$$h_B(x) = \begin{cases} -0.07 & \text{feature 15 of } x \leq 8.0 \\ 0.22 & \text{feature 15 of } x > 8.0 \end{cases}$$

...

$$h_Z(x) = \begin{cases} -0.07 & \text{feature 8 of } x \leq 8.0 \\ 0.82 & \text{feature 8 of } x > 8.0 \end{cases}$$

$$F_A := F_A + \rho_A h_A$$

$$F_B := F_B + \rho_B h_B$$

...

$$F_Z := F_Z + \rho_Z h_Z$$

# Gradient Boosting for Classification

round 1

i	y	$Y_A$	$Y_B$	$Y_C$	$Y_D$	$Y_E$	$Y_F$	$Y_G$	$Y_H$	$Y_I$	$Y_J$	$Y_K$	$Y_L$	$Y_M$	$Y_N$	$Y_O$	$Y_P$	$Y_Q$	$Y_R$	$Y_S$	$Y_T$	$Y_U$	$Y_V$	$Y_W$	$Y_X$	$Y_Y$	$Y_Z$	
1	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	I	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

i	y	$F_A$	$F_B$	$F_C$	$F_D$	$F_E$	$F_F$	$F_G$	$F_H$	$F_I$	$F_J$	$F_K$	$F_L$	$F_M$	$F_N$	$F_O$	$F_P$	$F_Q$	$F_R$	$F_S$	$F_T$	$F_U$	$F_V$	$F_W$	$F_X$	$F_Y$	$F_Z$	
1	T	-0.08	-0.07	-0.06	-0.07	-0.02	-0.02	-0.08	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	-0.08	-0.08	-0.07	-0.07	-0.02	-0.04	-0.04	0.59	-0.01	-0.07	-0.07	-0.05	-0.07	
2	I	-0.08	0.23	-0.06	-0.07	-0.02	-0.02	0.16	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	-0.08	-0.08	-0.07	-0.07	-0.02	-0.04	-0.04	-0.07	-0.01	-0.07	-0.07	-0.05	-0.06	-0.07
3	D	-0.08	0.23	-0.06	-0.07	-0.02	-0.02	0.16	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	-0.08	-0.08	-0.07	-0.07	-0.02	-0.04	-0.04	-0.07	-0.01	-0.07	-0.07	-0.05	-0.06	-0.07
4	N	-0.08	-0.07	-0.06	-0.07	-0.02	-0.02	0.16	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	0.3	-0.07	-0.07	-0.02	-0.04	-0.04	-0.07	-0.01	-0.07	-0.07	-0.05	-0.06	-0.07	
5	G	-0.08	0.23	-0.06	-0.07	-0.02	-0.02	0.16	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	-0.08	-0.07	-0.07	-0.02	-0.04	-0.04	-0.07	-0.01	-0.07	-0.07	-0.05	-0.06	-0.07	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

i	y	$P_A$	$P_B$	$P_C$	$P_D$	$P_E$	$P_F$	$P_G$	$P_H$	$P_I$	$P_J$	$P_K$	$P_L$	$P_M$	$P_N$	$P_O$	$P_P$	$P_Q$	$P_R$	$P_S$	$P_T$	$P_U$	$P_V$	$P_W$	$P_X$	$P_Y$	$P_Z$
1	T	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
2	I	0.04	0.05	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
3	D	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
4	N	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.05	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
5	G	0.04	0.05	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	$Y_A - P_A$	$Y_B - P_B$	$Y_C - P_C$	$Y_D - P_D$	$Y_E - P_E$	$Y_F - P_F$	$Y_G - P_G$	$Y_H - P_H$	$Y_I - P_I$	$Y_J - P_J$	$Y_K - P_K$	$Y_L - P_L$	$Y_M - P_M$	$Y_N - P_N$	$Y_O - P_O$	$Y_P - P_P$	$Y_Q - P_Q$	$Y_R - P_R$	$Y_S - P_S$	$Y_T - P_T$	$Y_U - P_U$	$Y_V - P_V$	$Y_W - P_W$	$Y_X - P_X$	$Y_Y - P_Y$	$Y_Z - P_Z$	
1	T	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	0.93	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
2	I	-0.04	-0.05	-0.04	-0.04	-0.04	-0.05	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
3	D	-0.04	-0.05	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
4	N	-0.04	-0.04	-0.04	-0.04	-0.04	-0.05	-0.04	-0.04	-0.04	-0.05	-0.04	-0.04	-0.04	0.95	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
5	G	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	0.95	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

# Gradient Boosting for Classification

$$h_A(x) = \begin{cases} 0.37 & \text{feature 10 of } x \leq 2.0 \\ -0.07 & \text{feature 10 of } x > 2.0 \end{cases}$$

$$h_B(x) = \begin{cases} -0.07 & \text{feature 14 of } x \leq 5.0 \\ 0.22 & \text{feature 14 of } x > 5.0 \end{cases}$$

...

$$h_Z(x) = \begin{cases} -0.07 & \text{feature 8 of } x \leq 8.0 \\ 0.35 & \text{feature 8 of } x > 8.0 \end{cases}$$

$$F_A := F_A + \rho_A h_A$$

$$F_B := F_B + \rho_B h_B$$

...

$$F_Z := F_Z + \rho_Z h_Z$$

# Gradient Boosting for Classification

round 2

i	y	$Y_A$	$Y_B$	$Y_C$	$Y_D$	$Y_E$	$Y_F$	$Y_G$	$Y_H$	$Y_I$	$Y_J$	$Y_K$	$Y_L$	$Y_M$	$Y_N$	$Y_O$	$Y_P$	$Y_Q$	$Y_R$	$Y_S$	$Y_T$	$Y_U$	$Y_V$	$Y_W$	$Y_X$	$Y_Y$	$Y_Z$	
1	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	I	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

i	y	$F_A$	$F_B$	$F_C$	$F_D$	$F_E$	$F_F$	$F_G$	$F_H$	$F_I$	$F_J$	$F_K$	$F_L$	$F_M$	$F_N$	$F_O$	$F_P$	$F_Q$	$F_R$	$F_S$	$F_T$	$F_U$	$F_V$	$F_W$	$F_X$	$F_Y$	$F_Z$	
1	T	-0.15	-0.14	-0.12	-0.14	-0.03	0.28	-0.14	-0.04	1.49	-0.07	-0.11	-0.08	-0.14	-0.17	-0.13	-0.13	-0.04	-0.11	-0.07	1.05	0.19	0.25	-0.16	-0.09	0.33	-0.14	
2	I	-0.15	0.16	-0.12	-0.14	-0.03	-0.08	0.33	-0.04	-0.07	-0.07	-0.11	-0.08	-0.14	-0.17	-0.13	-0.13	-0.04	-0.11	-0.07	-0.11	-0.07	-0.15	-0.16	-0.09	-0.13	-0.14	
3	D	-0.15	0.16	-0.12	-0.14	-0.03	-0.08	0.1	-0.04	-0.07	-0.07	-0.11	-0.08	-0.14	-0.17	-0.13	-0.13	-0.04	0.19	0.19	-0.11	-0.11	-0.07	-0.15	-0.16	-0.09	-0.13	-0.14
4	N	-0.15	-0.14	-0.12	-0.14	-0.03	-0.08	0.1	-0.04	-0.07	-0.07	0.46	-0.08	-0.14	0.5	-0.13	-0.13	-0.04	-0.11	-0.07	-0.11	-0.07	-0.15	0.25	-0.09	-0.13	-0.14	
5	G	-0.15	0.16	-0.12	-0.14	-0.03	-0.08	0.33	-0.04	-0.07	-0.07	-0.11	-0.08	-0.14	-0.17	-0.13	-0.13	-0.04	0.19	-0.07	-0.11	-0.07	-0.15	-0.16	-0.09	-0.13	-0.14	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

i	y	$P_A$	$P_B$	$P_C$	$P_D$	$P_E$	$P_F$	$P_G$	$P_H$	$P_I$	$P_J$	$P_K$	$P_L$	$P_M$	$P_N$	$P_O$	$P_P$	$P_Q$	$P_R$	$P_S$	$P_T$	$P_U$	$P_V$	$P_W$	$P_X$	$P_Y$	$P_Z$
1	T	0.03	0.03	0.03	0.03	0.04	0.03	0.03	0.15	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.09	0.04	0.04	0.03	0.03	0.05	0.03
2	I	0.04	0.05	0.04	0.04	0.04	0.04	0.06	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
3	D	0.04	0.05	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
4	N	0.03	0.03	0.03	0.03	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.06	0.04	0.03	0.06	0.03	0.03	0.04	0.04	0.04	0.04	0.04	0.03	0.05	0.04	0.03
5	G	0.03	0.05	0.04	0.04	0.04	0.06	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.03	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.03	0.04	0.04	0.04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	$Y_A - P_A$	$Y_B - P_B$	$Y_C - P_C$	$Y_D - P_D$	$Y_E - P_E$	$Y_F - P_F$	$Y_G - P_G$	$Y_H - P_H$	$Y_I - P_I$	$Y_J - P_J$	$Y_K - P_K$	$Y_L - P_L$	$Y_M - P_M$	$Y_N - P_N$	$Y_O - P_O$	$Y_P - P_P$	$Y_Q - P_Q$	$Y_R - P_R$	$Y_S - P_S$	$Y_T - P_T$	$Y_U - P_U$	$Y_V - P_V$	$Y_W - P_W$	$Y_X - P_X$	$Y_Y - P_Y$	$Y_Z - P_Z$	
1	T	-0.03	-0.03	-0.03	-0.03	-0.04	-0.03	-0.03	-0.15	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	0.91	-0.04	-0.04	-0.03	-0.03	-0.05	-0.03	
2	I	-0.04	-0.05	-0.04	-0.04	-0.04	-0.06	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
3	D	-0.04	-0.05	-0.04	0.96	-0.04	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	
4	N	-0.03	-0.03	-0.03	-0.03	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.06	-0.04	-0.03	0.94	-0.03	-0.03	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.03	-0.05	-0.04	-0.03
5	G	-0.03	-0.05	-0.04	-0.04	-0.04	0.94	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.03	-0.04	-0.04	-0.04	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	-0.03	-0.04	-0.04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

# Gradient Boosting for Classification

round 100

i	y	$Y_A$	$Y_B$	$Y_C$	$Y_D$	$Y_E$	$Y_F$	$Y_G$	$Y_H$	$Y_I$	$Y_J$	$Y_K$	$Y_L$	$Y_M$	$Y_N$	$Y_O$	$Y_P$	$Y_Q$	$Y_R$	$Y_S$	$Y_T$	$Y_U$	$Y_V$	$Y_W$	$Y_X$	$Y_Y$	$Y_Z$	
1	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	I	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
5	G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...		

i	y	$F_A$	$F_B$	$F_C$	$F_D$	$F_E$	$F_F$	$F_G$	$F_H$	$F_I$	$F_J$	$F_K$	$F_L$	$F_M$	$F_N$	$F_O$	$F_P$	$F_Q$	$F_R$	$F_S$	$F_T$	$F_U$	$F_V$	$F_W$	$F_X$	$F_Y$	$F_Z$
1	T	-3.26	-2.7	-2.2	-2.22	-2.48	-0.31	-2.77	-1.19	2.77	0.1	-1.49	-1.02	-1.64	-0.8	-2.4	-3.57	-0.9	-2.45	-0.2	4.61	0.5	-0.71	-1.21	-0.24	0.49	-1.66
2	I	-1.64	-1.09	-2.29	-1.8	0.45	-0.43	2.14	-1.56	1.19	1.09	-1.5	-0.5	-3.64	-3.98	-0.39	-2.3	1.42	-0.59	0.27	-2.88	-1.96	-1.67	-4.38	-2.06	-2.95	-1.76
3	D	-2.45	0.18	-3.01	0.18	-2.79	-1.7	-2.21	0.43	-1.12	0.32	0.67	-2.16	-2.91	-2.76	-1.92	-3.04	-1.47	-0.48	-1.48	-1.25	-2.25	-3.23	-4.38	0.17	-2.95	-2.65
4	N	-3.95	-3.38	-0.22	-0.94	-1.33	-1.38	-1.22	-0.12	-2.33	-3.13	0.58	-0.65	-0.25	2.96	-2.84	-1.82	0.19	0.55	-1.22	-1.25	0.45	-1.8	0.11	-0.69	-1.6	-3.78
5	G	-3.14	-0.04	-2.37	-0.78	0.02	-2.68	2.6	-1.48	-1.93	0.42	-1.44	-1.45	-3.36	-3.98	-0.94	-3.42	1.84	1.44	0.62	-1.25	-1.33	-4.41	-4.71	-2.62	-2.15	-1.09
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

i	y	$P_A$	$P_B$	$P_C$	$P_D$	$P_E$	$P_F$	$P_G$	$P_H$	$P_I$	$P_J$	$P_K$	$P_L$	$P_M$	$P_N$	$P_O$	$P_P$	$P_Q$	$P_R$	$P_S$	$P_T$	$P_U$	$P_V$	$P_W$	$P_X$	$P_Y$	$P_Z$
1	T	0	0	0	0	0.01	0	0	0.13	0.01	0	0	0	0	0	0	0	0.01	0.79	0.01	0	0	0.01	0.01	0	0	
2	I	0.01	0.01	0	0.01	0.06	0.02	0.32	0.01	0.12	0.11	0.01	0.02	0	0	0.03	0	0.16	0.02	0.05	0	0.01	0.01	0	0	0.01	0
3	D	0.01	0.11	0	0.11	0.01	0.02	0.01	0.14	0.03	0.12	0.17	0.01	0	0.01	0.01	0	0.02	0.05	0.02	0.03	0.01	0	0	0.11	0	0.01
4	N	0	0	0.02	0.01	0.01	0.01	0.01	0.03	0	0	0.05	0.02	0.02	0.59	0	0	0.04	0.05	0.01	0.01	0.05	0.01	0.03	0.02	0.01	0
5	G	0	0.03	0	0.01	0.03	0	0.42	0.01	0	0.05	0.01	0	0	0.01	0	0.19	0.13	0.06	0.01	0.01	0	0	0	0	0	0.01
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

i	y	$Y_A - P_A$	$Y_B - P_B$	$Y_C - P_C$	$Y_D - P_D$	$Y_E - P_E$	$Y_F - P_F$	$Y_G - P_G$	$Y_H - P_H$	$Y_I - P_I$	$Y_J - P_J$	$Y_K - P_K$	$Y_L - P_L$	$Y_M - P_M$	$Y_N - P_N$	$Y_O - P_O$	$Y_P - P_P$	$Y_Q - P_Q$	$Y_R - P_R$	$Y_S - P_S$	$Y_T - P_T$	$Y_U - P_U$	$Y_V - P_V$	$Y_W - P_W$	$Y_X - P_X$	$Y_Y - P_Y$	$Y_Z - P_Z$
1	T	-0	-0	-0	-0	-0.01	-0	-0	-0.13	-0.01	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0.01	0.21	-0.01	-0	-0	-0.01	-0.01	-0
2	I	-0.01	-0.01	-0	-0.01	-0.06	-0.02	-0.32	-0.01	0.88	-0.11	-0.01	-0.02	-0	-0	-0.03	-0	-0.16	-0.02	-0.05	-0	-0.01	-0.01	-0	-0	-0.01	-0.01
3	D	-0.01	-0.11	-0	0.89	-0.01	-0.02	-0.01	-0.14	-0.03	-0.12	-0.17	-0.01	-0	-0.01	-0.01	-0	-0.02	-0.05	-0.02	-0.03	-0.01	-0	-0	-0.11	-0	-0.01
4	N	-0	-0	-0.02	-0.01	-0.01	-0.01	-0.01	-0.03	-0	-0	-0.05	-0.02	-0.02	0.41	-0	-0	-0.04	-0.05	-0.01	-0.01	-0.05	-0.01	-0.01	-0.03	-0.02	-0.01
5	G	-0	-0.03	-0	-0.01	-0.03	-0	0.58	-0.01	-0	-0.05	-0.01	-0.01	-0	-0	-0.01	-0	-0.19	-0.13	-0.06	-0.01	-0.01	-0	-0	-0	-0	-0.01
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

## Things not covered

- ▶ How to choose proper learning rates. See [Friedman, 2001]
- ▶ Other possible loss functions. See [Friedman, 2001]

# References I

-  Breiman, L. (1999).  
Prediction games and arcing algorithms.  
*Neural computation*, 11(7):1493–1517.
-  Breiman, L. et al. (1998).  
Arcing classifier (with discussion and a rejoinder by the author).  
*The annals of statistics*, 26(3):801–849.
-  Freund, Y. and Schapire, R. E. (1997).  
A decision-theoretic generalization of on-line learning and an application to boosting.  
*Journal of computer and system sciences*, 55(1):119–139.
-  Freund, Y., Schapire, R. E., et al. (1996).  
Experiments with a new boosting algorithm.  
In *ICML*, volume 96, pages 148–156.

## References II

-  Friedman, J., Hastie, T., and Tibshirani, R. (2000).  
Special invited paper. additive logistic regression: A statistical view of boosting.  
*Annals of statistics*, pages 337–374.
-  Friedman, J. H. (2001).  
Greedy function approximation: a gradient boosting machine.  
*Annals of Statistics*, pages 1189–1232.
-  Schapire, R. E. and Freund, Y. (2012).  
*Boosting: Foundations and Algorithms*.  
MIT Press.



**BITS** Pilani  
Pilani Campus

# Support Vector Machines

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)





## Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

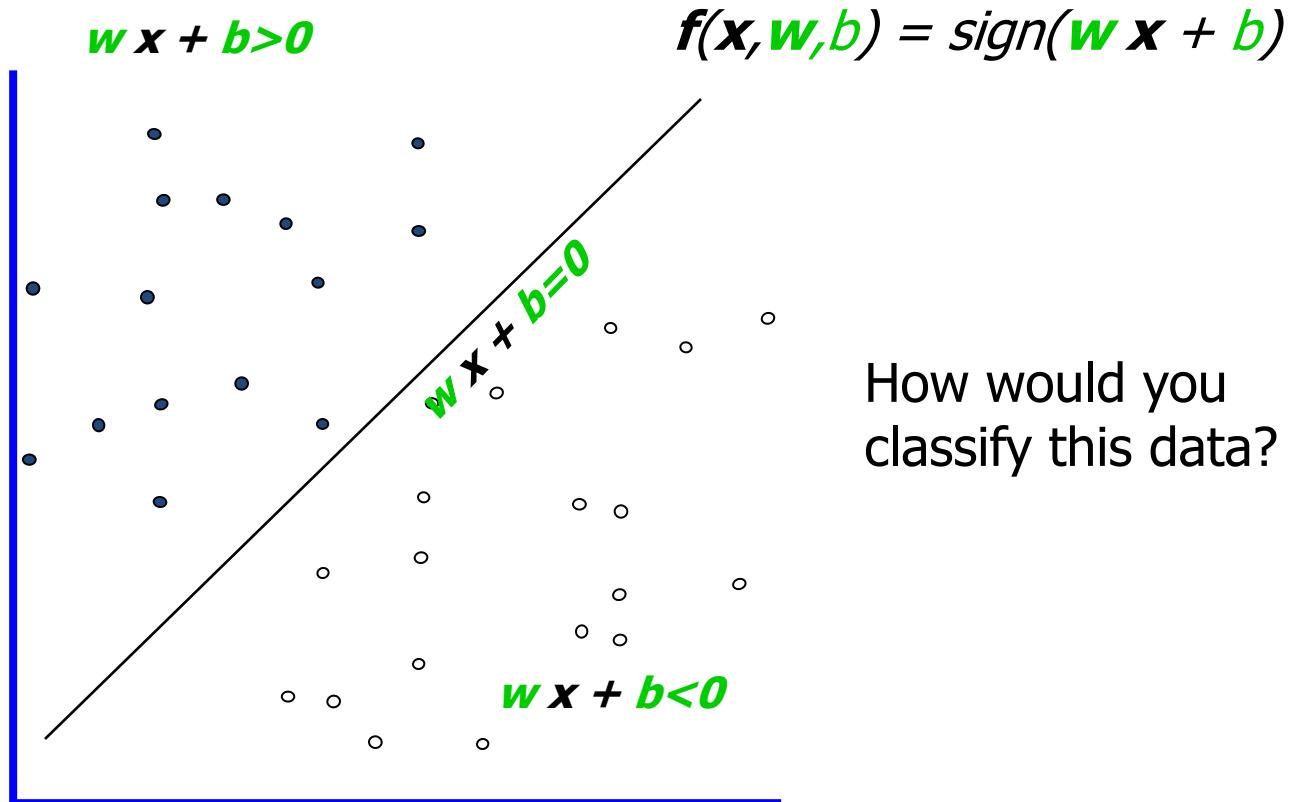
# Topics to be covered

---

- Linear Classifiers
  - Maximum Margin Classification
  - Linear SVM
  - SVM optimization problem
  - Soft Margin SVM
-

# Linear Classifiers

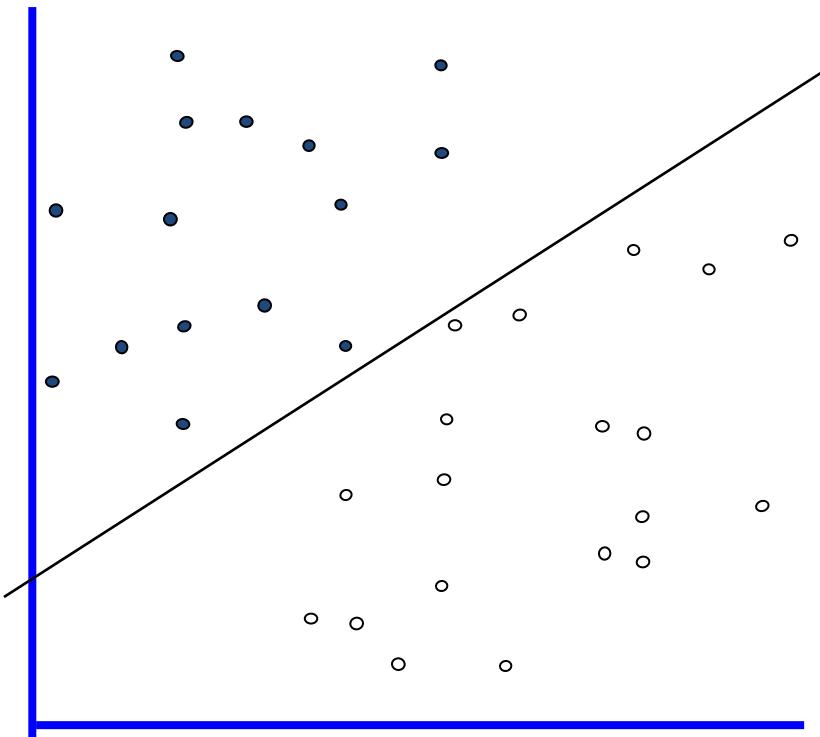
- denotes +1
- denotes -1



# Linear Classifiers

$$\mathbf{f}(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

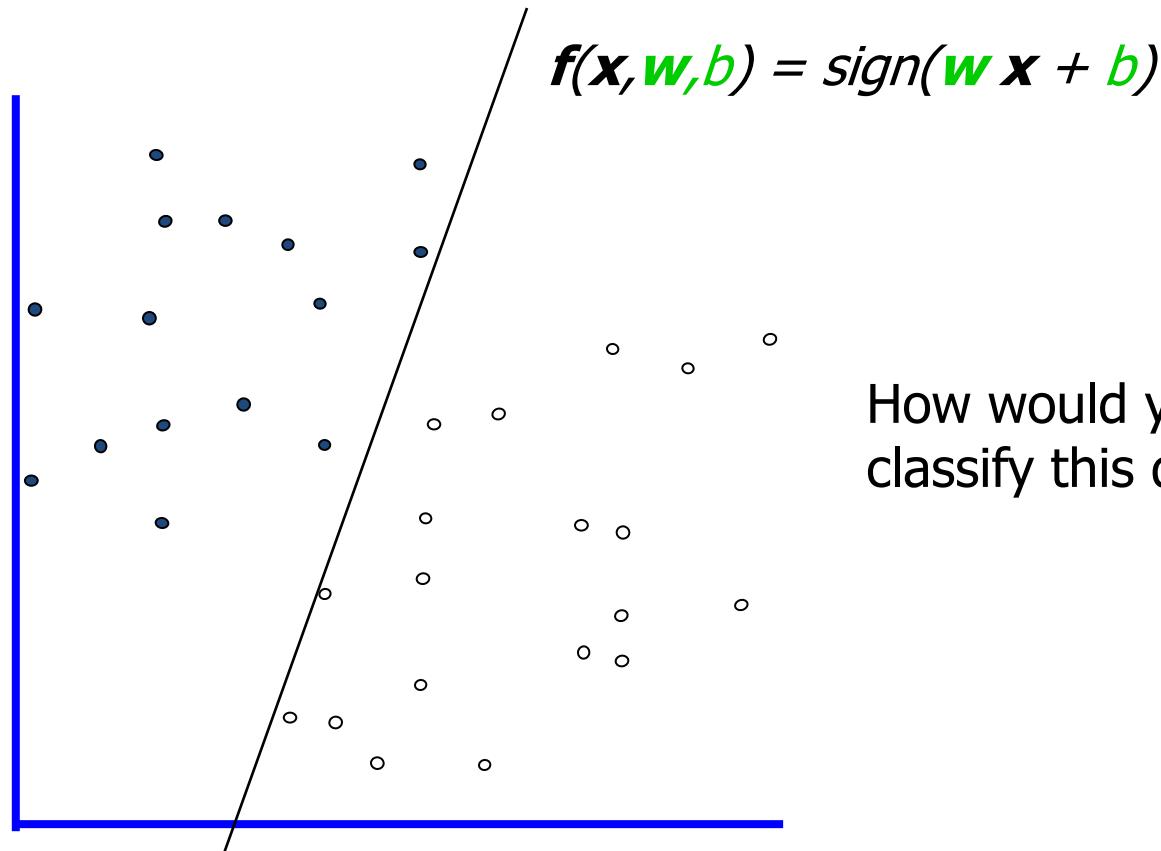
- denotes +1
- denotes -1



How would you  
classify this data?

# Linear Classifiers

- denotes +1
- denotes -1

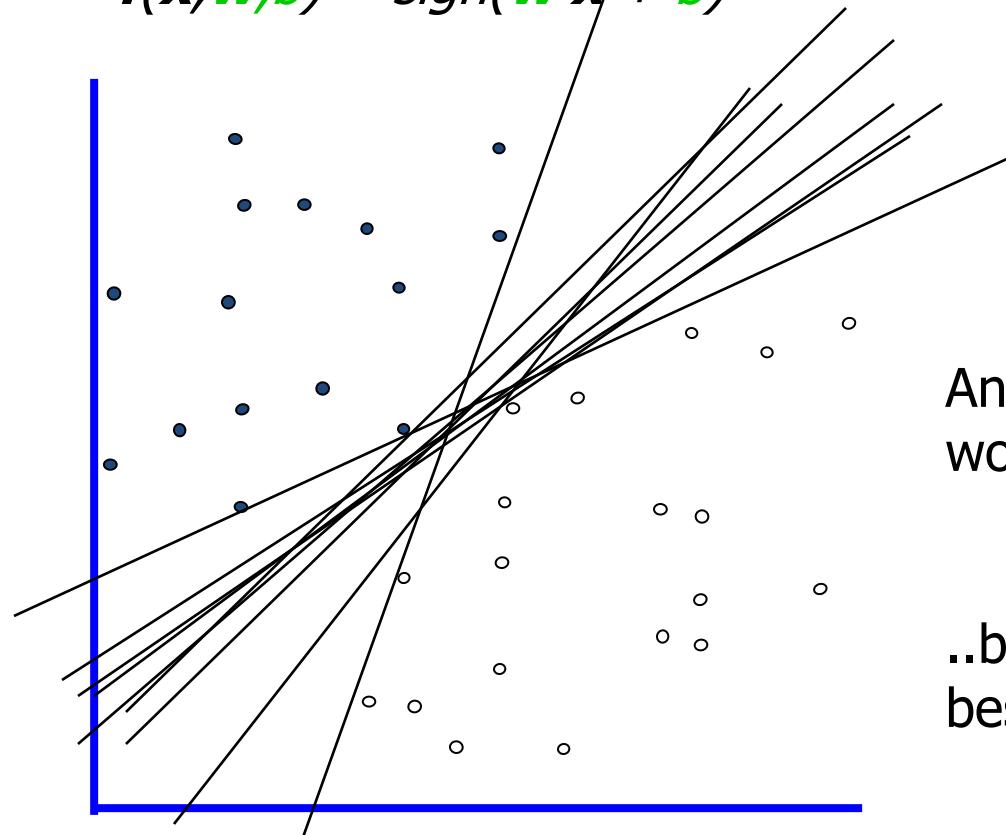


How would you  
classify this data?

# Linear Classifiers

$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

- denotes +1
- denotes -1

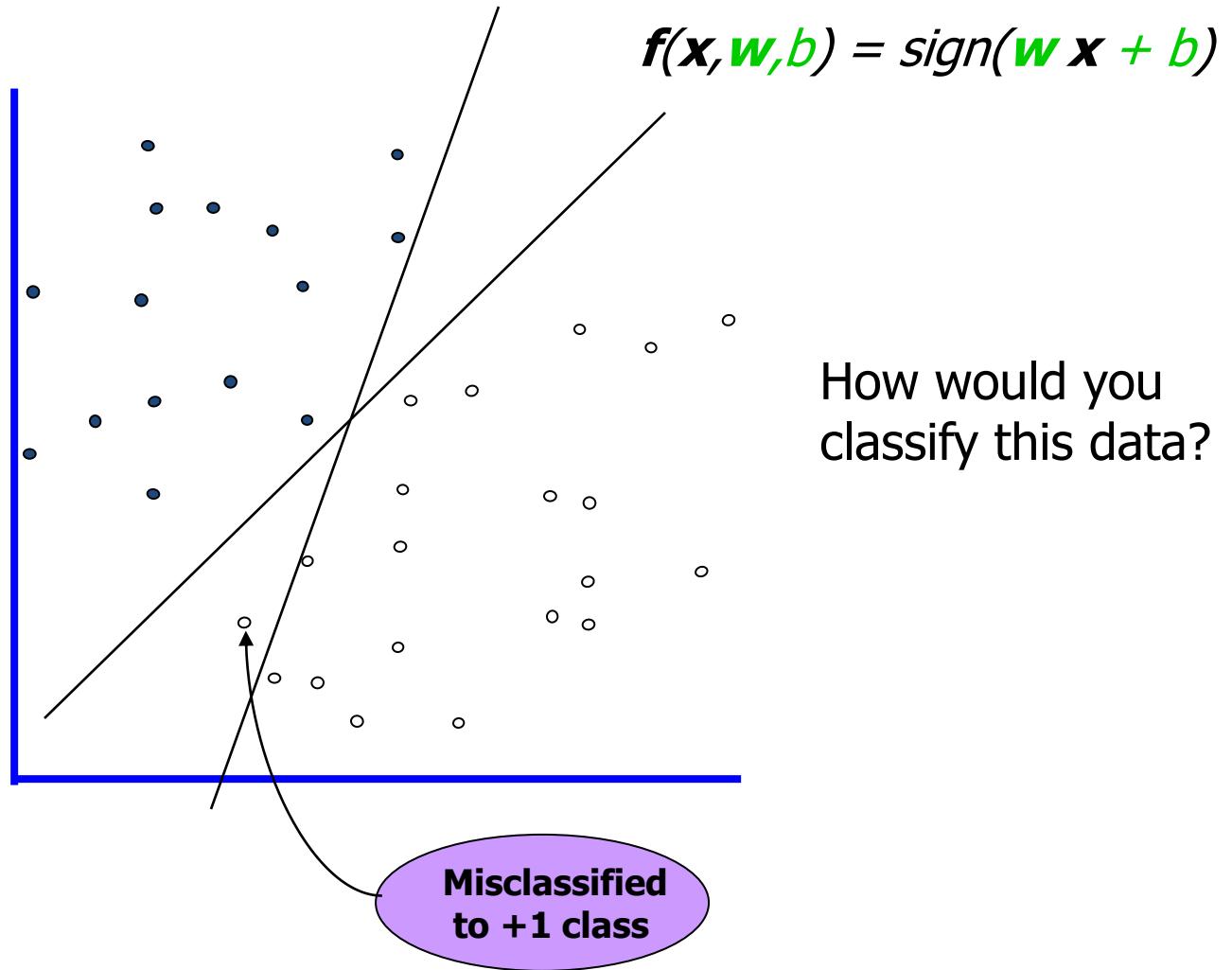


Any of these  
would be fine..

..but which is  
best?

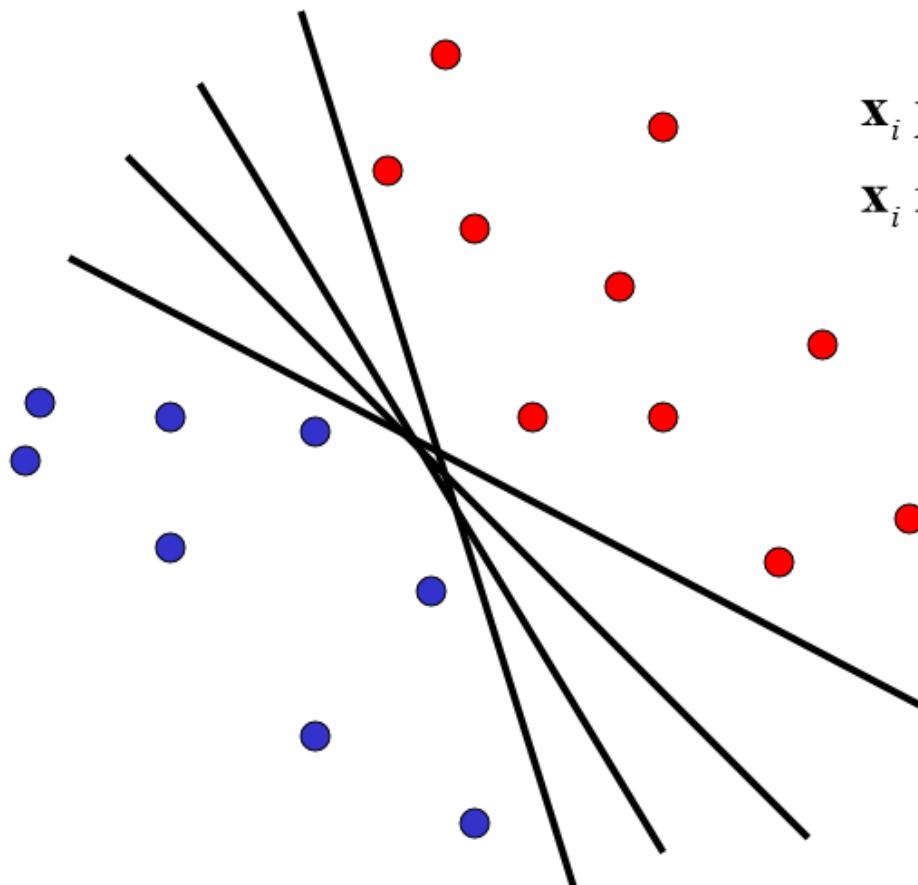
# Linear Classifiers

- denotes +1
- denotes -1



# Linear Classifier

- Find linear function to separate positive and negative examples

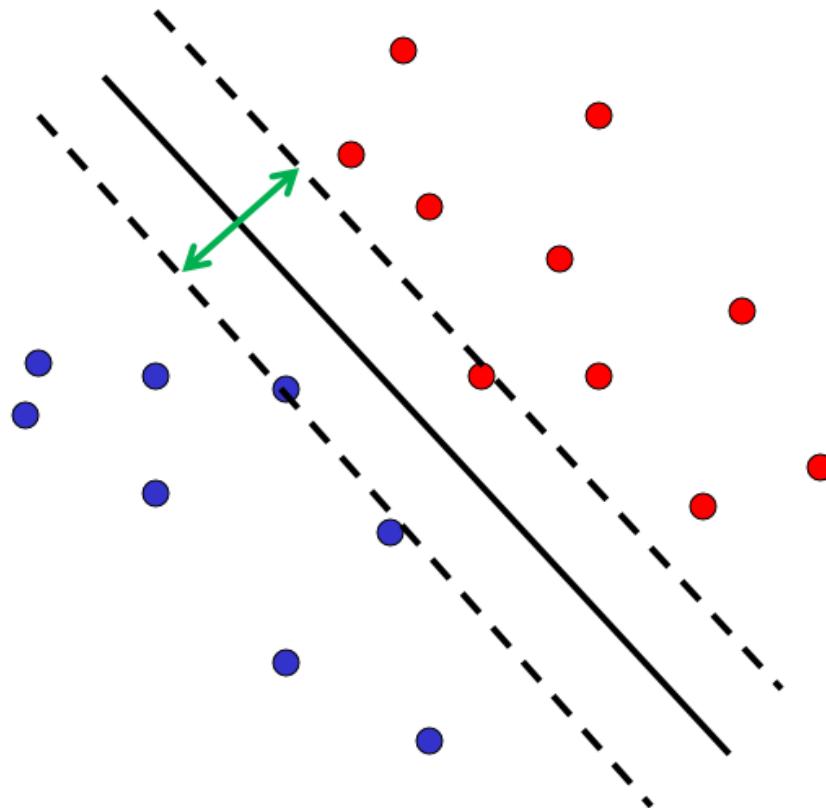


$\mathbf{x}_i$  positive :  $\mathbf{x}_i \cdot \mathbf{w} + b \geq 0$

$\mathbf{x}_i$  negative :  $\mathbf{x}_i \cdot \mathbf{w} + b < 0$

Which line  
is best?

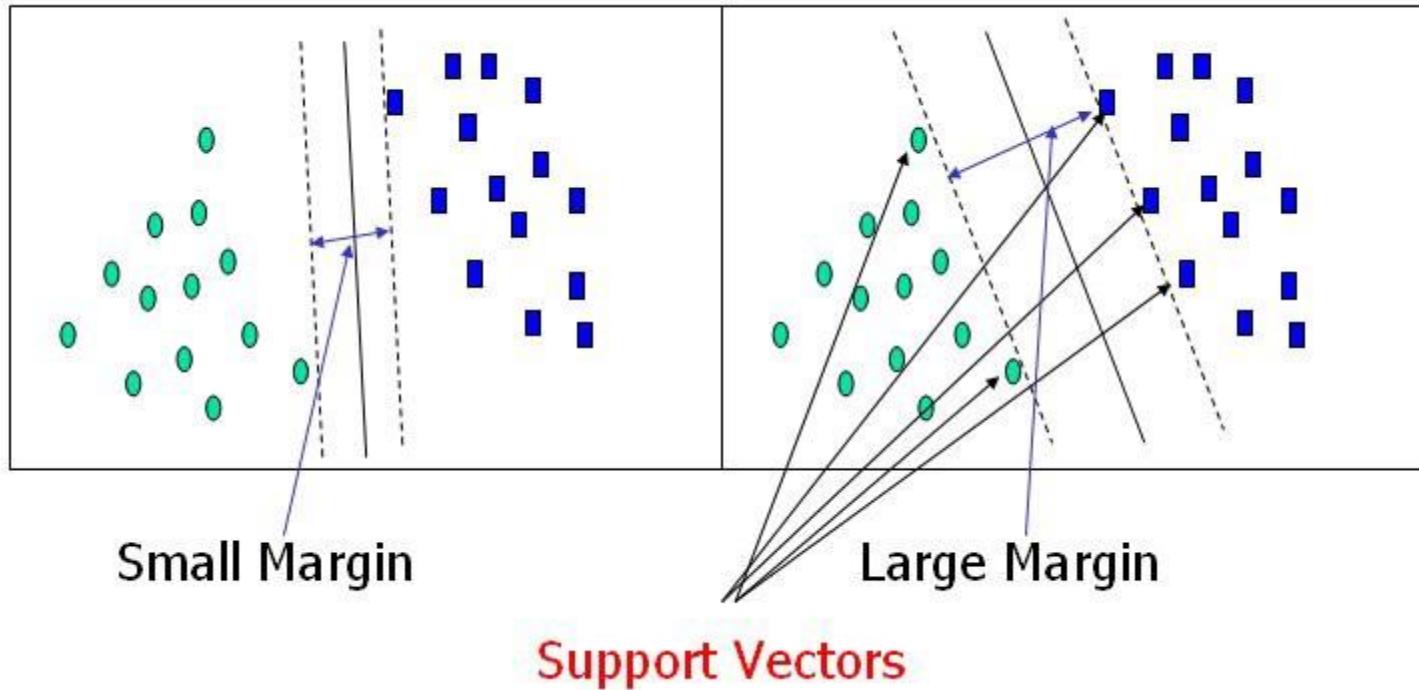
# Linear Classifier



- Discriminative classifier based on *optimal separating line (for 2d case)*
- Maximize the *margin* between the positive and negative training examples

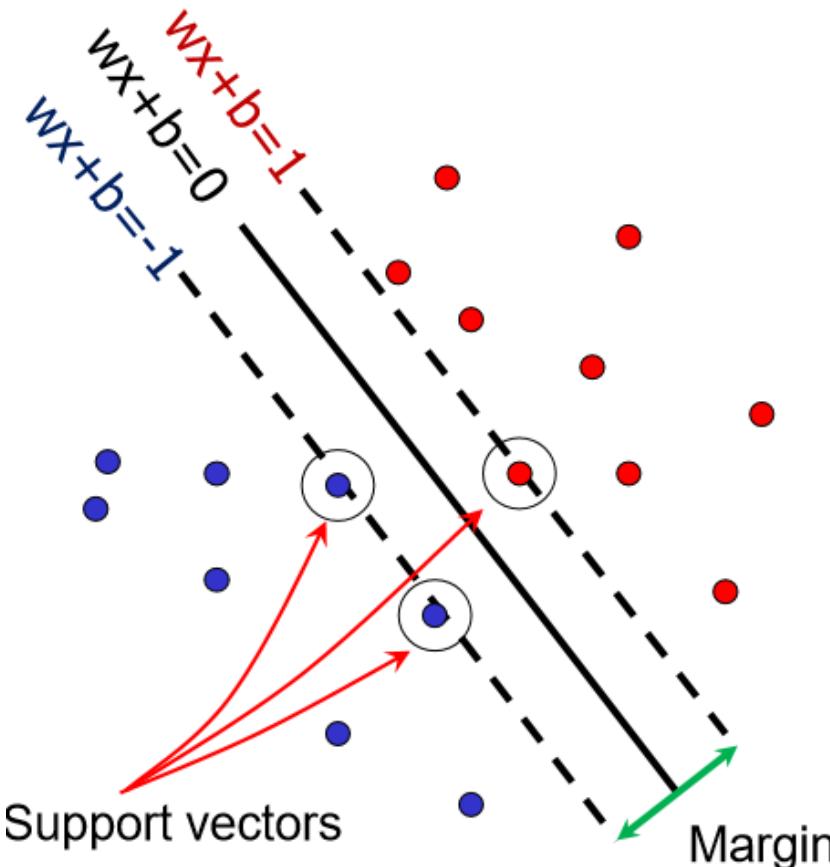
C. Burges, [A Tutorial on Support Vector Machines for Pattern Recognition](#), Data Mining and Knowledge Discovery, 1998

# Large margin and support vectors



# Support Vector Machines

- Want line that maximizes the margin.



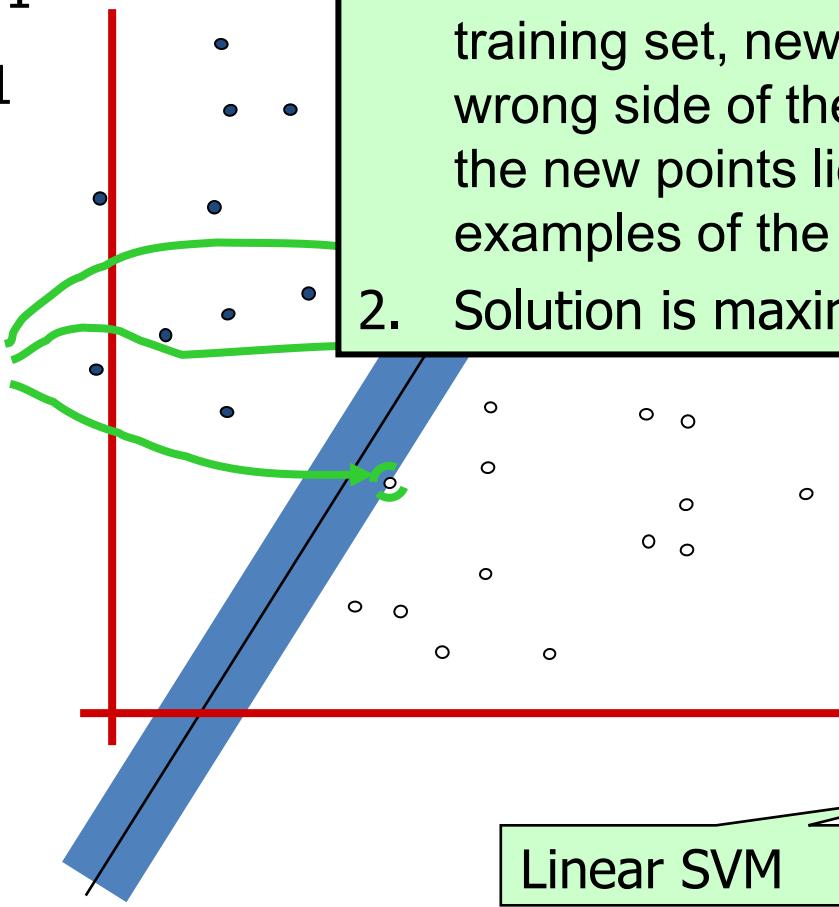
$\mathbf{x}_i$  positive ( $y_i = 1$ ):  $\mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$  negative ( $y_i = -1$ ):  $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support vectors,  $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

# Maximum Margin

• denotes +1  
 • denotes -1  
  
**Support Vectors**  
 are those  
 datapoints that  
 the margin  
 pushes up  
 against



1. If hyperplane is oriented such that it is close to some of the points in your training set, new data may lie on the wrong side of the hyperplane, even if the new points lie close to training examples of the correct class.
2. Solution is maximizing the margin with the, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

# Support Vectors

---

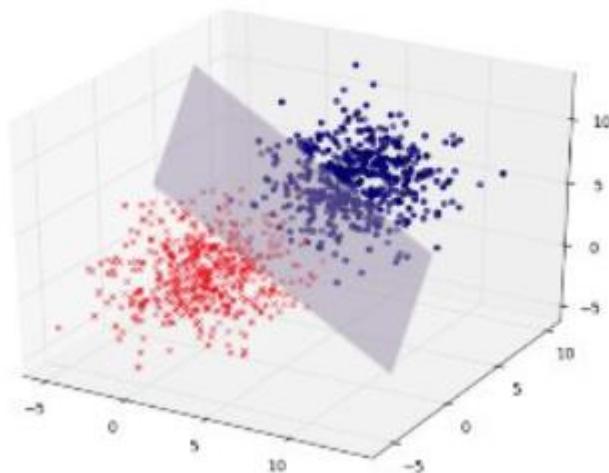
- Geometric description of SVM is that the max-margin hyperplane is completely determined by those points that lie nearest to it.
- Points that lie on this margin are the support vectors.
- The points of our data set which if removed, would alter the position of the dividing hyperplane

# Example

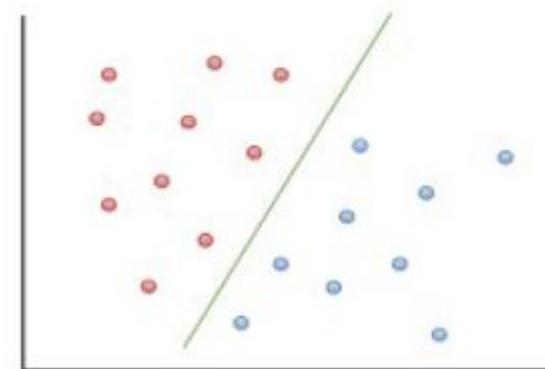
$$\mathbf{w}^T \mathbf{x} = 0$$

$$y = ax + b$$

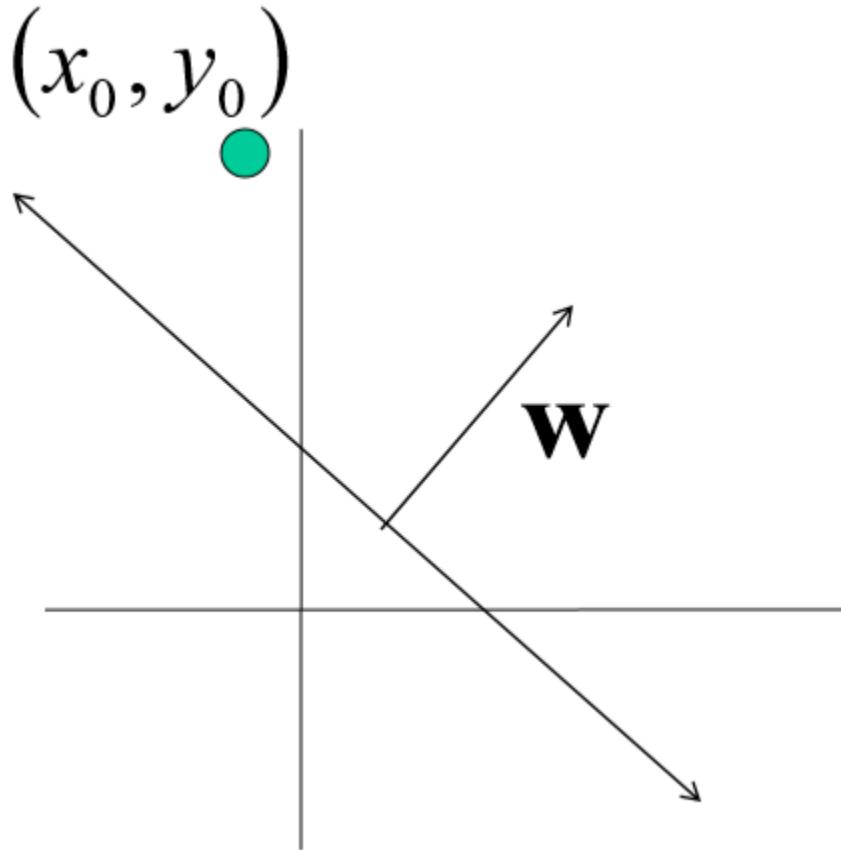
Hyperplane



Line



# Line with 2 features: R2



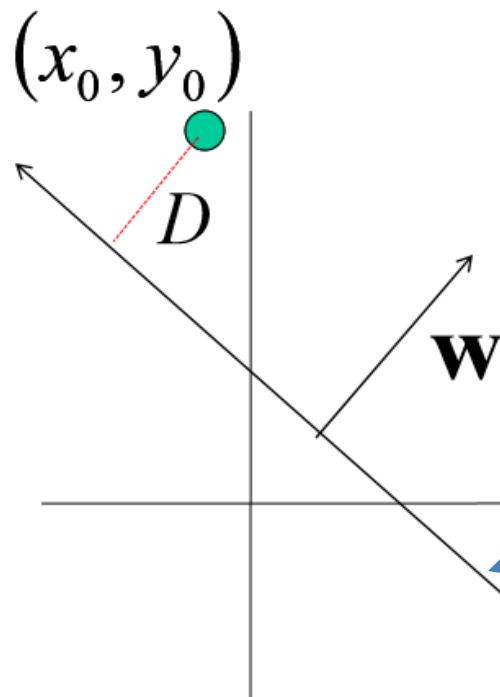
Let  $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$   $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$



$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

# Line with 2 features: R2



Let  $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$   $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

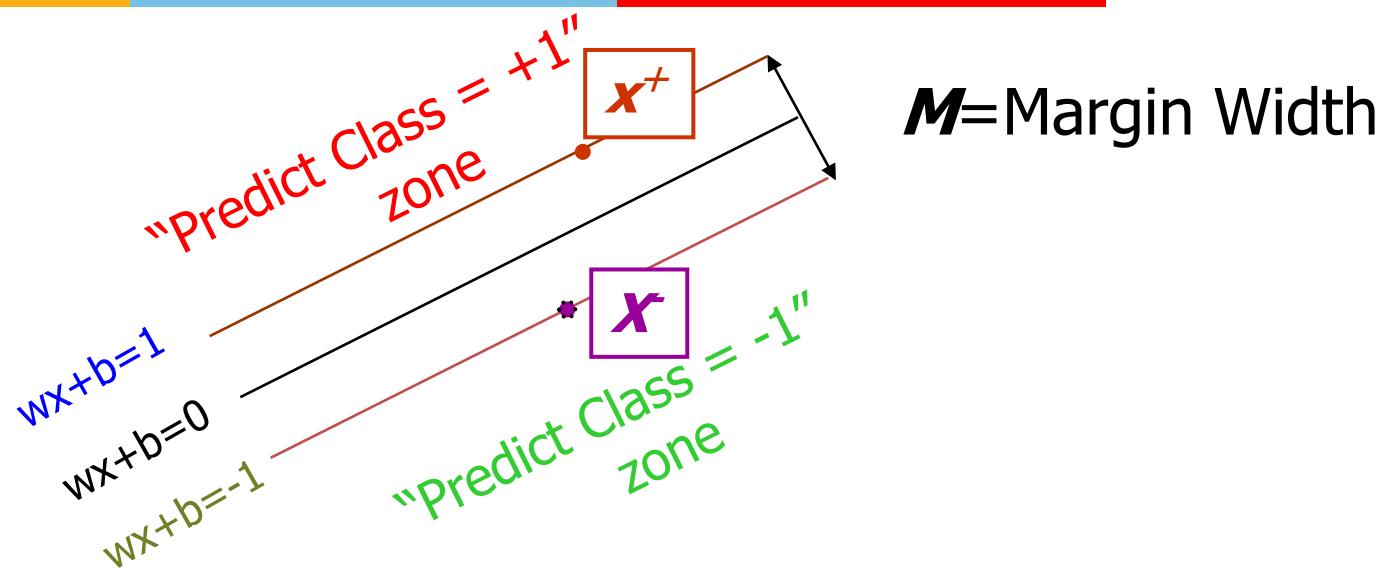
$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

$$D = \frac{|ax_0 + cy_0 + b|}{\sqrt{a^2 + c^2}} = \frac{|\mathbf{w}^\top \mathbf{x} + b|}{\|\mathbf{w}\|}$$

} distance from  
point to line

Kristen Grauman

# Linear SVM Mathematically



Distance between lines given by solving linear equation:

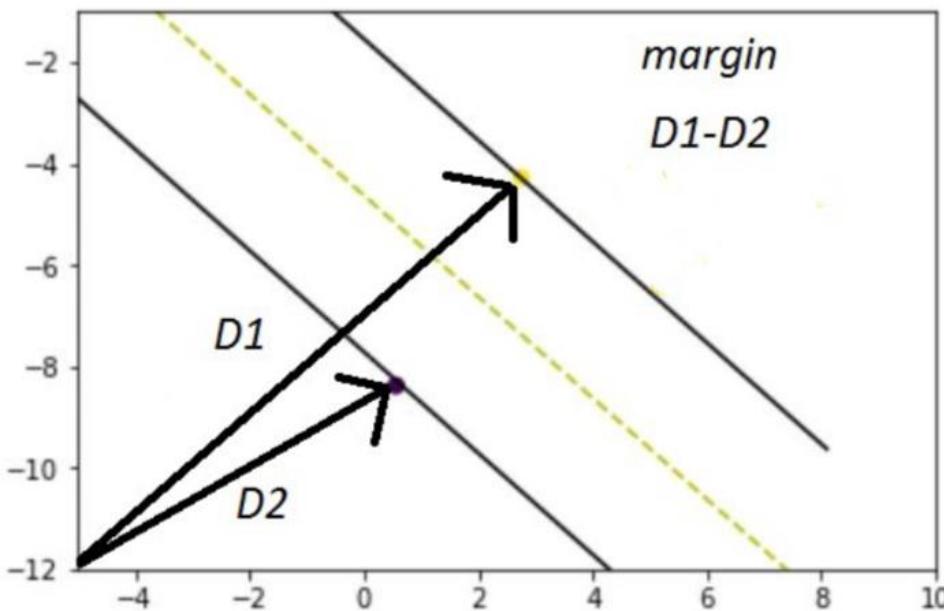
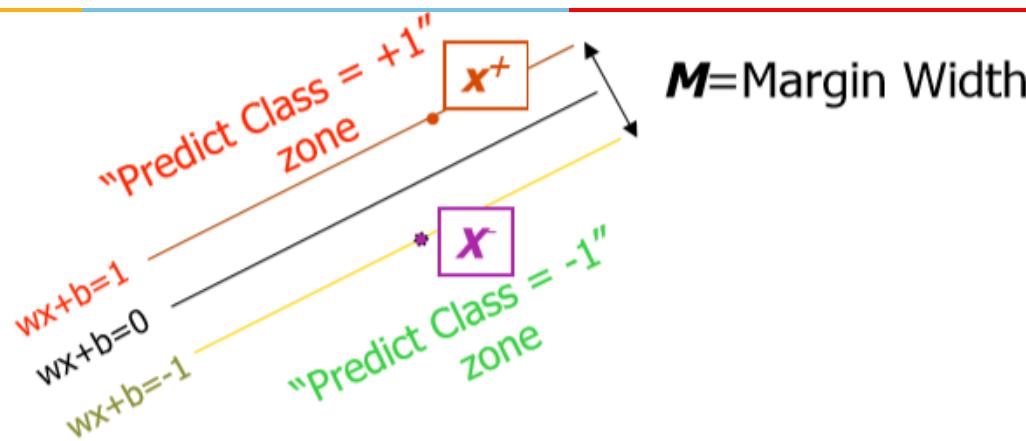
What we know:

- $\mathbf{w} \cdot \mathbf{x}^+ + b = +1$
- $\mathbf{w} \cdot \mathbf{x}^- + b = -1$

$$\text{Maximize margin: } M = \frac{2}{\|\mathbf{w}\|}$$

$$\text{Equivalent to minimize: } \frac{1}{2} \|\mathbf{w}\|^2$$

# Linear SVM Mathematically



$$D1 = w^T x + b = 1 \quad w^T x + b - 1 = 0$$

$$D2 = w^T x + b = -1 \quad w^T x + b + 1 = 0$$

$$w^T x + b - 1 - w^T x + b + 1$$



Solve algebraically

$$\frac{2}{|w|}$$

# Solving the Optimization Problem

---

1. Maximize margin  $2/\|\mathbf{w}\|$
2. Correctly classify all training data points:

$\mathbf{x}_i$  positive ( $y_i = 1$ ):  $\mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$  negative ( $y_i = -1$ ):  $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

*Quadratic optimization problem:*

Find  $\mathbf{w}$  and  $b$  such that

$\Phi(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$  is minimized;

and for all  $\{(\mathbf{x}_i, y_i)\}$ :  $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

# Solving the Optimization Problem

---

Find  $\mathbf{w}$  and  $b$  such that

$\Phi(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2$  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 inequality* constraints.
- All constraints in SVM are linear
- Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.

# Optimization Problem

---

- Optimization problem is typically written:

Minimize  $f(x)$

subject to

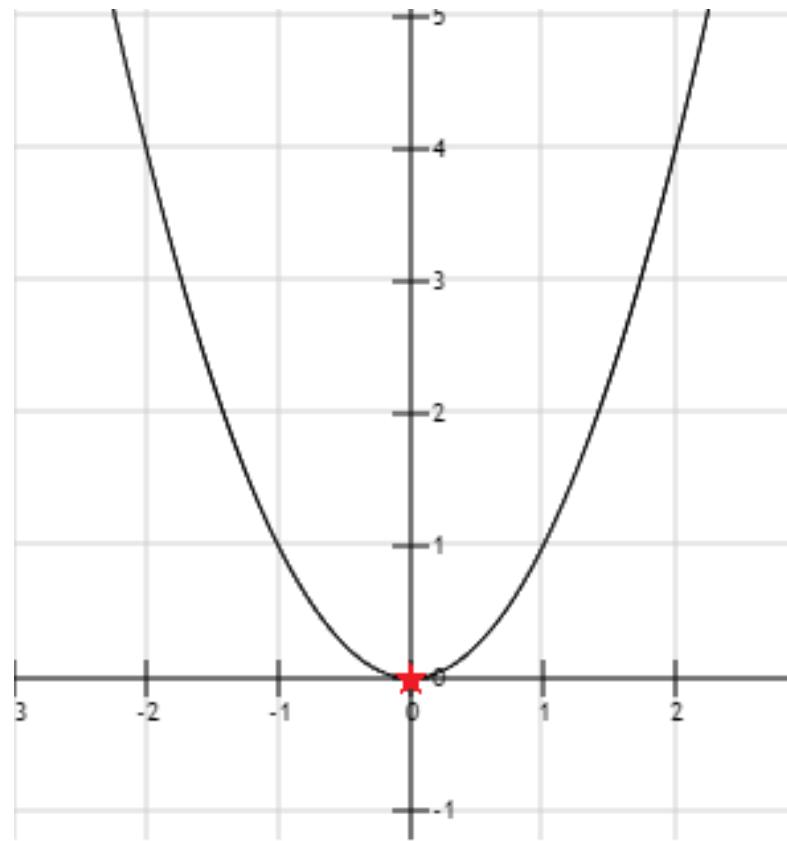
$$g_i(x) = 0, \quad i=1, \dots, p$$

$$h_i(x) \leq 0, \quad i=1, \dots, m$$

- $f(x)$  is called the objective function
  - By changing  $x$  (the optimization variable) we wish to find a value  $x^*$  for which  $f(x)$  is at its minimum.
  - $p$  functions of  $g_i$  define equality constraints and
  - $m$  functions  $h_i$  define inequality constraints.
  - The value we find MUST respect these constraints!
-

# Unconstrained Optimization

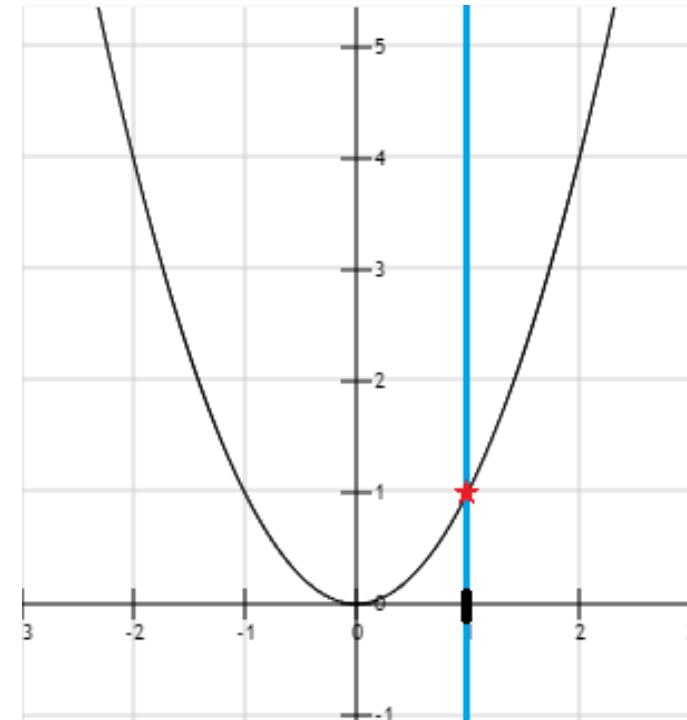
- Minimize  $x^2$



# Constrained Optimization -Equality Constraint

Minimize  $x^2$

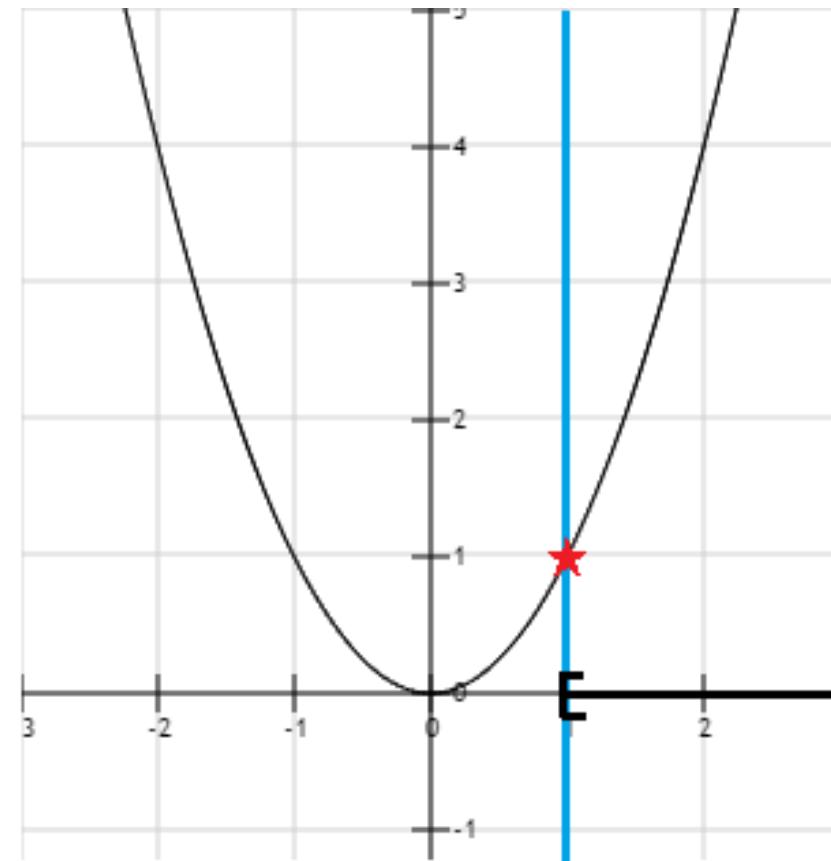
Subject to  $x = 1$



# Constrained Optimization - Inequality Constraint

Minimize  $x^2$

Subject to  $x \geq 1$



# Constrained optimization

---

- We can also have mix equality and inequality constraints together.
- Only restriction is that if we use contradictory constraints, we can end up with a problem which does not have a feasible set

Minimize  $x^2$

Subject to

$$x = 1$$

$$x < 0$$

Impossible for  $x$  to be equal 1 and less than zero at the same

# Constrained optimization

---

- A solution is an assignment of values to variables.
- A feasible solution is an assignment of values to variables such that all the constraints are satisfied.
- The objective function value of a solution is obtained by evaluating the objective function at the given solution.
- An optimal solution (assuming minimization) is one whose objective function value is less than or equal to that of all other feasible solutions.

# Lagrange Multipliers

---

- **How do we find the solution to an optimization problem with constraints?**
- Constrained maximization (minimization) problem is rewritten as a Lagrange function whose optimal point is a saddle point, i.e. a global maximum (minimum)
- *Lagrange function use Lagrange multipliers is a strategy for finding the local maxima and minima of a function subject to constraints*

# Constrained to Unconstrained Optimization: Lagrange Multiplier

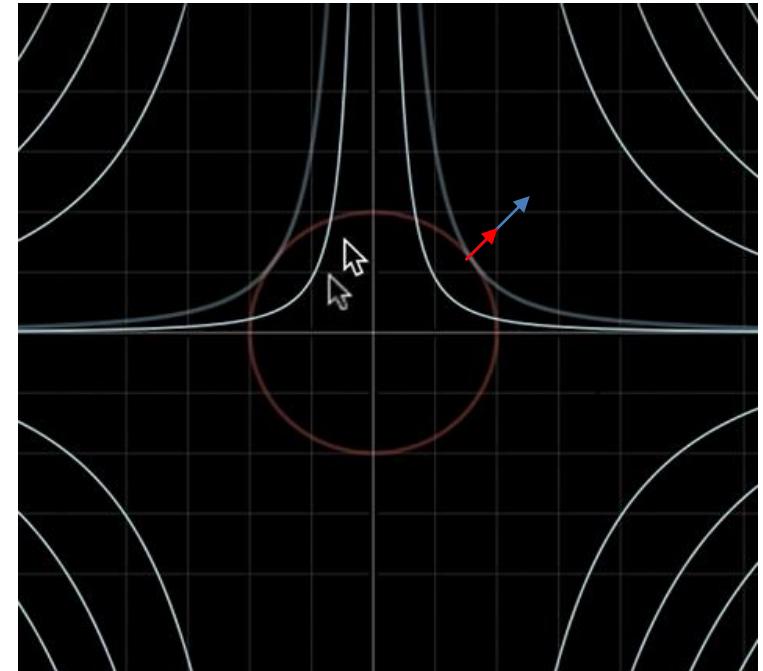
**Maximize**

$$f(x,y) = x^2 y$$

**Subject to**

$$g(x, y) : x^2 + y^2 = 1$$

- Maximum of  $f(x,y)$  under constraint  $g(x, y)$  is obtained when their gradients point to same direction (when they are tangent to each other).
  - Introduce a Lagrange multiplier  $\lambda$  for the equality constraint
  - Mathematically,
- $$\nabla f(x,y) = \lambda \nabla g(x,y)$$



# Example:

$$\max_{x,y} xy \text{ subject to } x + y = 6$$

- Introduce a Lagrange multiplier  $\lambda$  for constraint
- Construct the Lagrangian

$$L(x, y) = xy - \lambda(x + y - 6)$$

- Stationary points

$$\frac{\partial L(x, y)}{\partial \lambda} = x + y - 6 = 0$$

$$\begin{cases} \frac{\partial L(x, y)}{\partial x} = y - \lambda = 0 \\ \frac{\partial L(x, y)}{\partial y} = x - \lambda = 0 \end{cases} \Rightarrow x = y = \lambda$$

$$\Rightarrow x = y = 3$$

x and y values remain same even if you take  $+\lambda$  or  $-\lambda$  for equality constraint

$$\begin{aligned} 2x &= 6 \\ x &= y = 3 \\ \lambda &= 3 \end{aligned}$$

# Karush–Kuhn–Tucker (KKT) theorem

---

- KKT approach to nonlinear programming (quadratic) generalizes the method of Lagrange multipliers, which allows only equality constraints.
- KKT allows inequality constraints

# Karush-Kuhn-Tucker (KKT) conditions

---

- Start with

$\max f(x)$  subject to

$g_i(x) = 0$  and  $h_j(x) \geq 0$  for all  $i, j$

- Make the Lagrangian function

$$\mathcal{L} = f(x) - \sum_i \lambda_i g_i(x) - \sum_j \mu_j h_j(x)$$

- Take gradient and set to 0 – but other conditions also.

# KKT conditions

---

- Make the Lagrangian function

$$\mathcal{L} = f(x) - \sum_i \lambda_i g_i(x) - \sum_j \mu_j h_j(x)$$

- Necessary conditions to have a minimum are

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0$$

$$g_i(x^*) = 0 \text{ for all } i$$

$$h_j(x^*) \geq 0 \text{ for all } j$$

$$\mu_j \geq 0 \text{ for all } j$$

$$\mu_j^* h_j(x^*) = 0 \text{ for all } j$$

---

# KKT conditions

---

- Used to solve Inequality Constraints of the form  
 $h_j(\mathbf{x}) \leq 0$  for  $j = 1, 2, \dots, m$

- Using Lagrangian

$$L = f(\mathbf{x}) + \sum_{i=1}^q \lambda_i h_i(\mathbf{x}).$$

- With the constraints, called KKT conditions given as:

$$\begin{aligned}\frac{\partial L}{\partial x_i} &= 0, \quad \forall i = 1, 2, \dots, d & \lambda_i &\geq 0, \quad \forall i = 1, 2, \dots, q \\ h_i(\mathbf{x}) &\leq 0, \quad \forall i = 1, 2, \dots, q & \lambda_i h_i(\mathbf{x}) &= 0, \quad \forall i = 1, 2, \dots, q\end{aligned}$$

---

# Solving the Optimization Problem

Find  $w$  and  $b$  such that

$\Phi(w) = \frac{1}{2}||w||^2$  is minimized;

and for all  $\{(x_i, y_i)\}$ :  $y_i (w^T x_i + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear inequality* constraints.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier  $\alpha_i$*  is associated with every constraint in the primal problem

# Solving the Optimization Problem

---

- The solution involves constructing a *dual problem* where a *Lagrange multiplier*  $\alpha_i$  is associated with every constraint in the primary problem:

$$L(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i (w^T x_i + b) - 1]$$

- Taking partial derivative with respect to  $w$ ,  $\frac{\partial L}{\partial w} = 0$** 
  - $w - \sum \alpha_i y_i x_i = 0$
  - $w = \sum \alpha_i y_i x_i$
- Taking partial derivative with respect to  $b$ ,  $\frac{\partial L}{\partial b} = 0$** 
  - $-\sum \alpha_i y_i = 0$
  - $\sum \alpha_i y_i = 0$

# Solving the Optimization Problem

$$L(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i (w \cdot x_i + b) - 1]$$

- Expanding above equation:

$$L(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum \alpha_i y_i w \cdot x_i + \sum \alpha_i y_i b + \sum \alpha_i$$

- Substituting  $w = \sum \alpha_i y_i x_i$  and  $\sum \alpha_i y_i = 0$  in above equation

$$L(w, b, \alpha_i) = \frac{1}{2} (\sum_i \alpha_i y_i x_i) (\sum_j \alpha_j y_j x_j) - (\sum_i \alpha_i y_i x_i) (\sum_j \alpha_j y_j x_j) + \sum \alpha_i$$

$$L(w, b, \alpha_i) = \sum \alpha_i - \frac{1}{2} (\sum_i \alpha_i y_i x_i) (\sum_j \alpha_j y_j x_j)$$

$$L(w, b, \alpha_i) = \sum \alpha_i - \frac{1}{2} (\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i \cdot x_j)$$

# Solving the Optimization Problem

---

- Solution:  $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$



# Solving the Optimization Problem

---

- Solution:  $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$   
 $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$  (for any support vector)

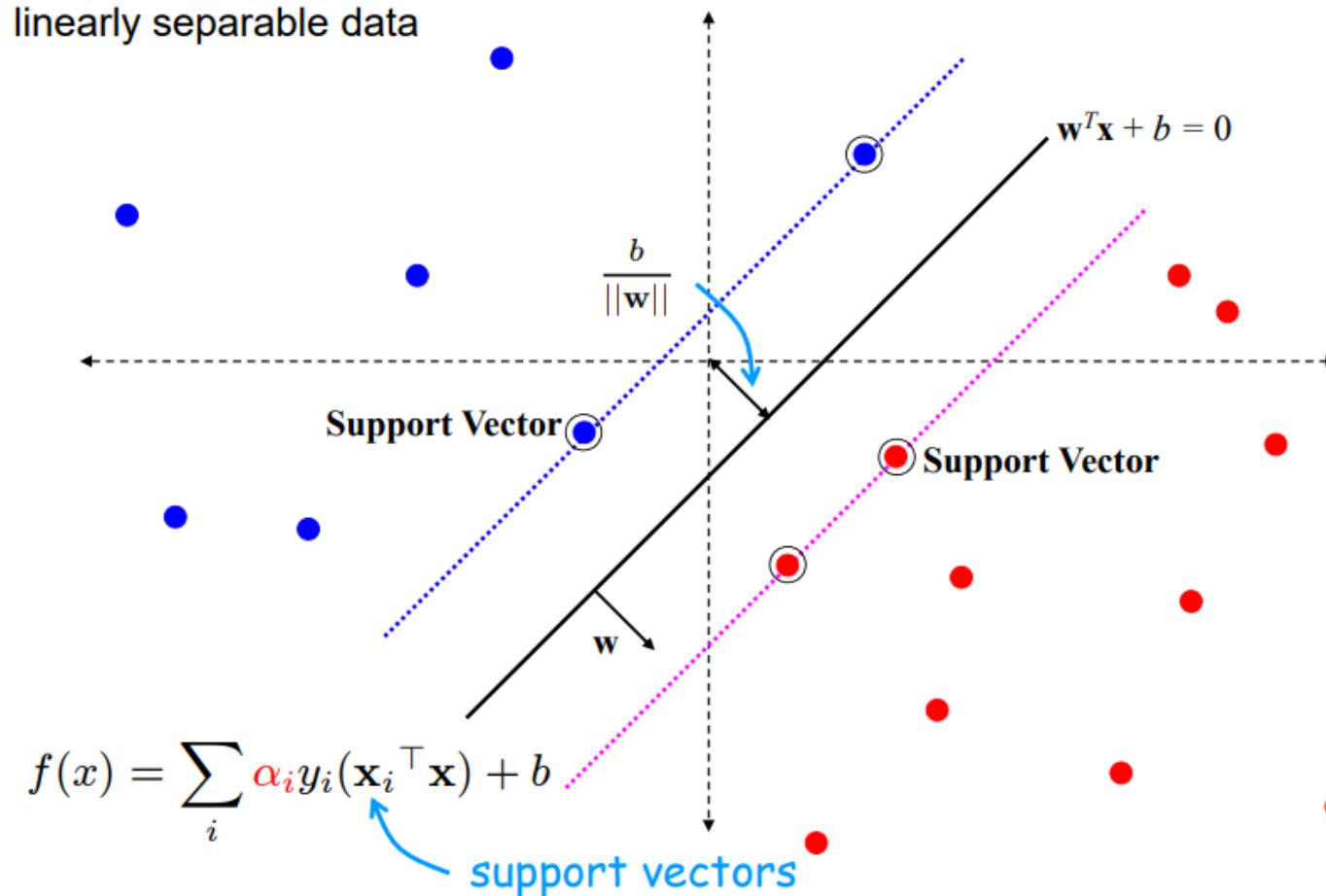
- Classification function:

$$\begin{aligned} f(\mathbf{x}) &= \text{sign} (\mathbf{w} \cdot \mathbf{x} + b) \\ &= \text{sign} \left( \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b \right) \end{aligned}$$

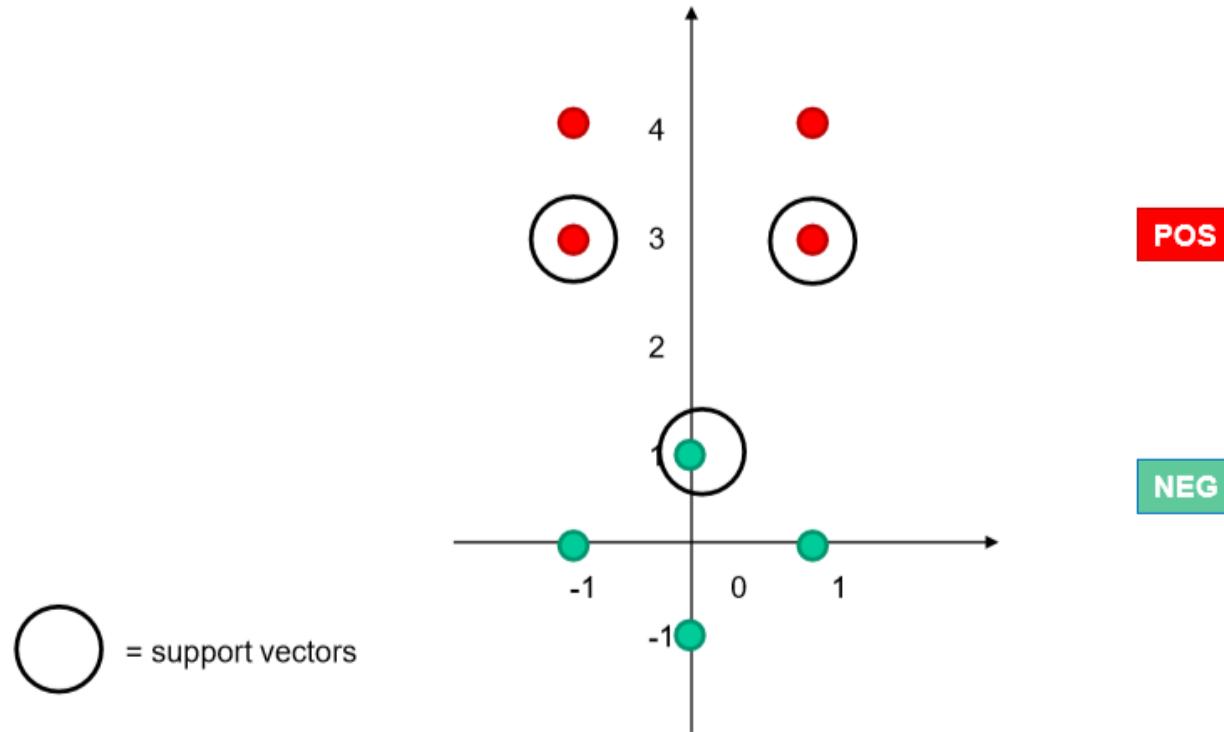
If  $f(\mathbf{x}) < 0$ , classify as negative, otherwise classify as positive.

- Notice that it relies on an *inner product* between the test point  $\mathbf{x}$  and the support vectors  $\mathbf{x}_i$
- (Solving the optimization problem also involves computing the inner products  $\mathbf{x}_i \cdot \mathbf{x}_j$  between all pairs of training points)

# Substituting w in support vectors function



# Example



= support vectors

Example adapted from Dan Ventura

# Solving for $\alpha$

---

- We know that for the support vectors,  $f(x) = 1$  or  $-1$  exactly
- Add a 1 in the feature representation for the bias
- The support vectors have coordinates and labels:
  - $x_1 = [0 \ 1 \ 1]$ ,  $y_1 = -1$
  - $x_2 = [-1 \ 3 \ 1]$ ,  $y_2 = +1$
  - $x_3 = [1 \ 3 \ 1]$ ,  $y_3 = +1$
- Thus we can form the following system of linear equations:

# Solving for $\alpha$

---

- System of linear equations:

$$\alpha_1 y_1 \operatorname{dot}(x_1, x_1) + \alpha_2 y_2 \operatorname{dot}(x_1, x_2) + \alpha_3 y_3 \operatorname{dot}(x_1, x_3) = y_1$$

$$\alpha_1 y_1 \operatorname{dot}(x_2, x_1) + \alpha_2 y_2 \operatorname{dot}(x_2, x_2) + \alpha_3 y_3 \operatorname{dot}(x_2, x_3) = y_2$$

$$\alpha_1 y_1 \operatorname{dot}(x_3, x_1) + \alpha_2 y_2 \operatorname{dot}(x_3, x_2) + \alpha_3 y_3 \operatorname{dot}(x_3, x_3) = y_3$$

$$-2 * \alpha_1 + 4 * \alpha_2 + 4 * \alpha_3 = -1$$

$$-4 * \alpha_1 + 11 * \alpha_2 + 9 * \alpha_3 = +1$$

$$-4 * \alpha_1 + 9 * \alpha_2 + 11 * \alpha_3 = +1$$

- Solution:  $\alpha_1 = 3.5$ ,  $\alpha_2 = 0.75$ ,  $\alpha_3 = 0.75$
-

# Solving for w and b

---

We know  $w = \alpha_1 y_1 x_1 + \dots + \alpha_N y_N x_N$  where  $N = \# \text{ SVs}$

$$\text{Thus } w = -3.5 * [0 \ 1 \ 1] + 0.75 [-1 \ 3 \ 1] + 0.75 [1 \ 3 \ 1] = \\ [0 \ 1 \ -2]$$

Separating out weights and bias, we have:  $w = [0 \ 1]$  and  $b = -2$

For SVMs, we used this eq for a line:  $ax + cy + b = 0$   
where  $w = [a \ c]$

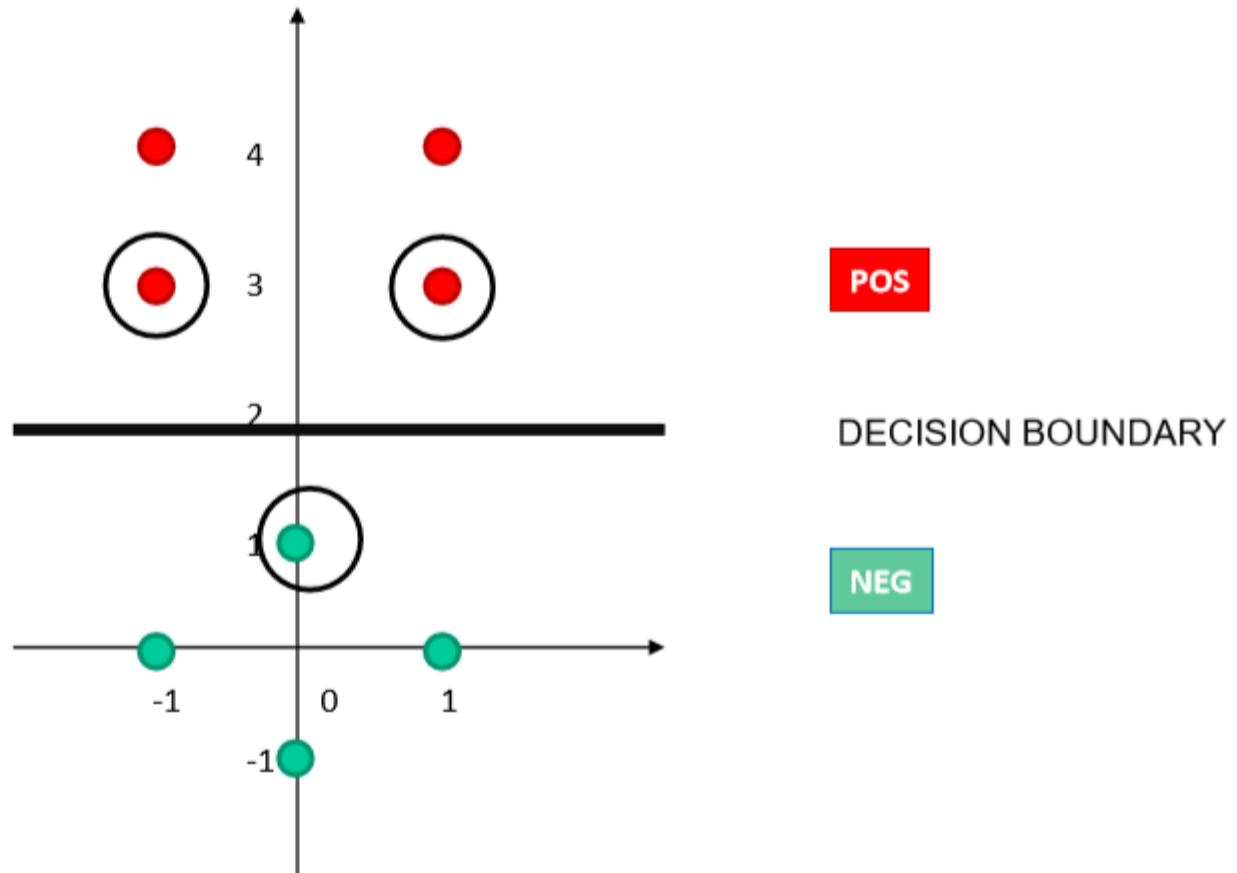
$$\text{Thus } ax + b = -cy \rightarrow y = (-a/c)x + (-b/c)$$

$$\text{Thus y-intercept is } -(-2)/1 = 2$$

The decision boundary is perpendicular to  $w$  and it has slope  $-0/1 = 0$

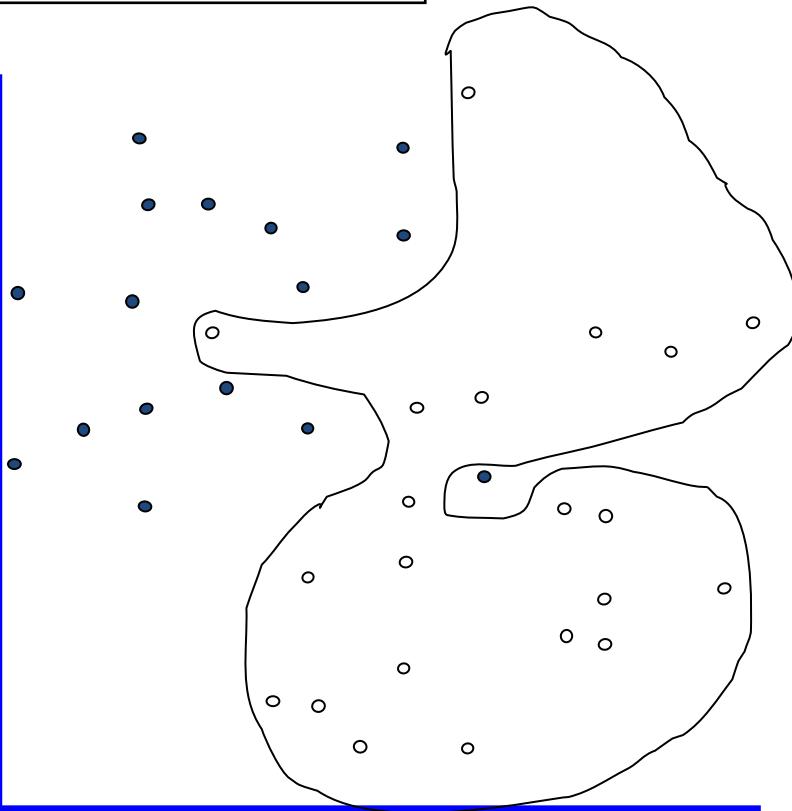
---

# Decision boundary



# 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?**

# Soft Margin Classification

---

***Slack variables  $\xi_i$  can be added to allow misclassification of difficult or noisy examples.***

What should our quadratic optimization criterion be?

Minimize

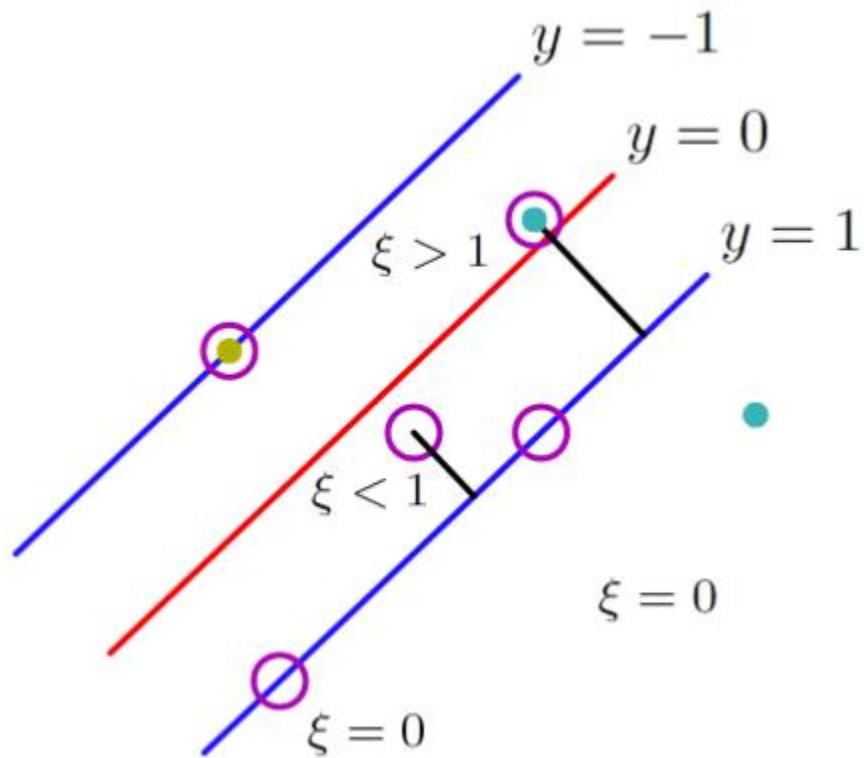
$$\frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_{k=1}^R \xi_k$$

# Slack Variable

---

- **Slack variable** as giving the classifier some leniency when it comes to moving around points near the **margin**.
- When  $C$  is large, larger slacks penalize the objective function of SVM's more than when  $C$  is small.

# Soft margin example



# Soft Margin

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

The  $\mathbf{w}$  that minimizes...

Maximize margin      Minimize misclassification

Misclassification cost      # data samples      Slack variable

subject to     $y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i,$

$\xi_i \geq 0, \quad \forall i = 1, \dots, N$

# Hard Margin versus Soft Margin

- **Hard Margin:**

Find  $\mathbf{w}$  and  $b$  such that

$$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} \text{ is minimized and for all } \{(\mathbf{x}_i, y_i)\}$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

- **Soft Margin 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.**

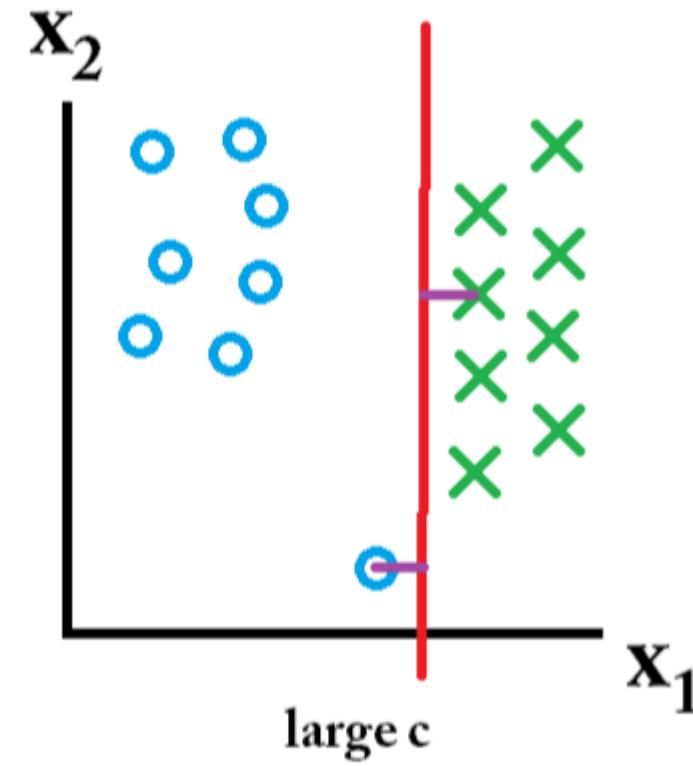
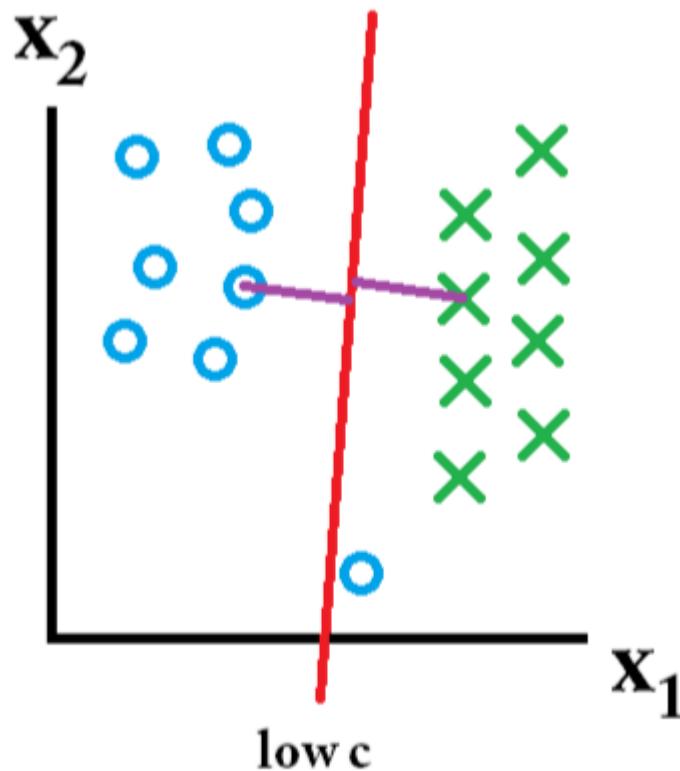
# Value of C parameter

---

- C parameter tells the SVM optimization how much you want to avoid misclassifying each training example.
  - For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.
  - Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.
-

# Effect of Margin size v/s misclassification cost

Training set

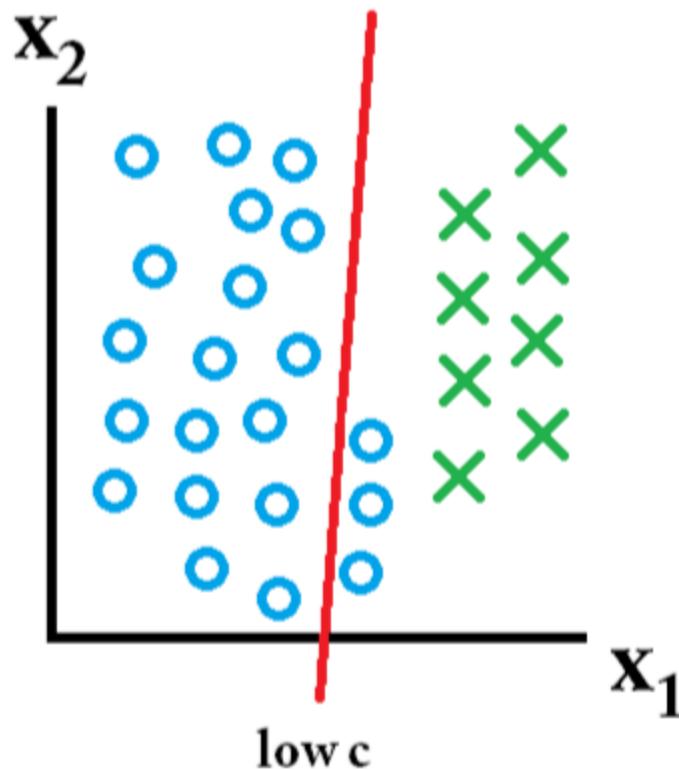


Misclassification ok, want large margin

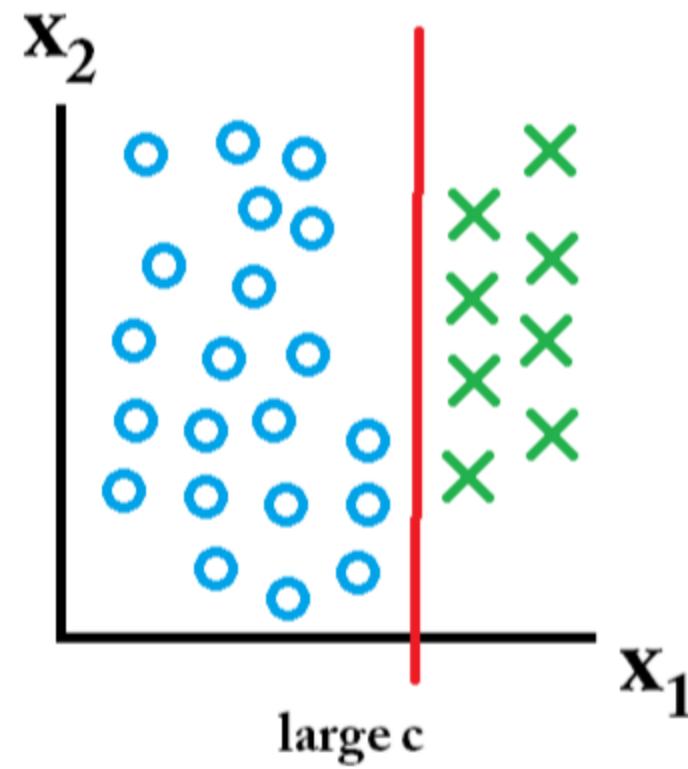
Misclassification not ok

# Effect of Margin size v/s misclassification cost

Including test set A



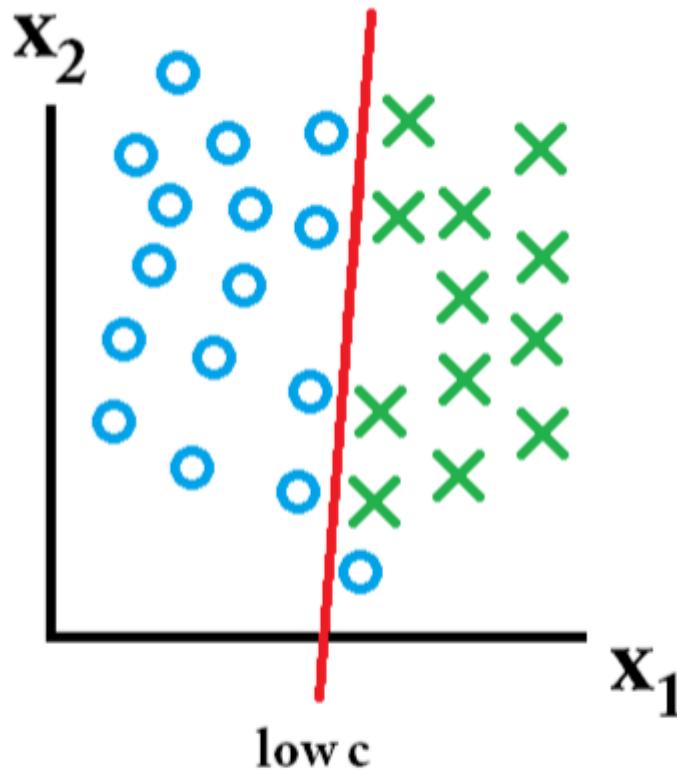
Misclassification ok, want large margin



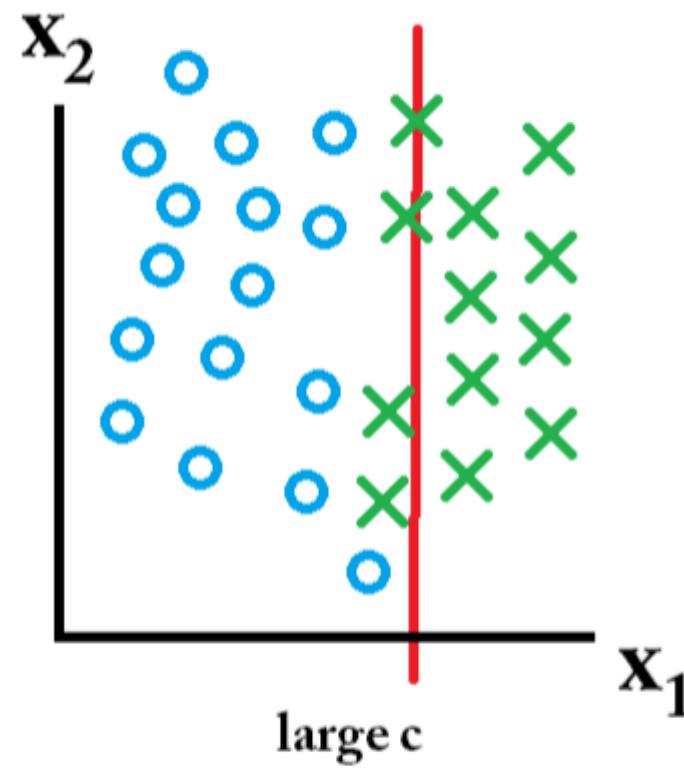
Misclassification not ok

# Effect of Margin size v/s misclassification cost

Including test set B



Misclassification ok, want large margin



Misclassification not ok

# 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  $\alpha_i$ ,

$$f(x) = \sum \alpha_i y_i x_i^T x + b$$

# Good Web References for SVM

---

- **Text categorization with Support Vector Machines: learning with many relevant features** - T. Joachims, ECML
- **A Tutorial on Support Vector Machines for Pattern Recognition**, Kluwer Academic Publishers - Christopher J.C. Burges
- <http://www.cs.utexas.edu/users/mooney/cs391L/>
- <https://www.coursera.org/learn/machine-learning/home/week/7>
- <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- <https://data-flair.training/blogs/svm-kernel-functions/>
- [MIT 6.034 Artificial Intelligence, Fall 2010](https://mit-6.034-artificial-intelligence-fall-2010.readthedocs.io/en/latest/lectures/lec10.html)
- <https://stats.stackexchange.com/questions/30042/neural-networks-vs-support-vector-machines-are-the-second-definitely-superior>
- <https://www.sciencedirect.com/science/article/abs/pii/S0893608006002796>
- <https://medium.com/deep-math-machine-learning-ai/chapter-3-support-vector-machine-with-math-47d6193c82be>
- [Radial basis kernel](#)



---

# Thank You



**BITS** Pilani  
Pilani Campus

# Support Vector Machines

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)





## Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

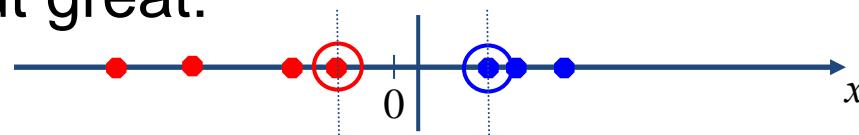
# Topics to be covered

---

- Nonlinear SVM
  - Kernel Trick
  - SVM Kernels
  - Multi-Class Problem
  - SVM vs Logistic Regression
  - SVM Applications
-

# Non-linear SVMs

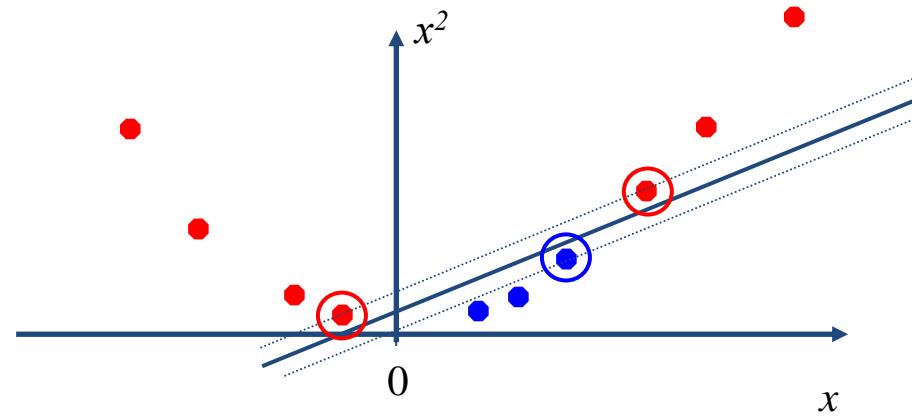
- Datasets that are linearly separable with some noise soft margin work out great:



- But what are we going to do if the dataset is just too hard?



- How about... mapping data to a higher-dimensional space:



# The “Kernel Trick”



- The linear classifier relies on dot product between vectors
  - $x_i^T \cdot x_j$
- If every data point is mapped into high-dimensional space via some transformation  $\Phi: x \rightarrow \phi(x)$ , the dot product becomes:
$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$
- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.

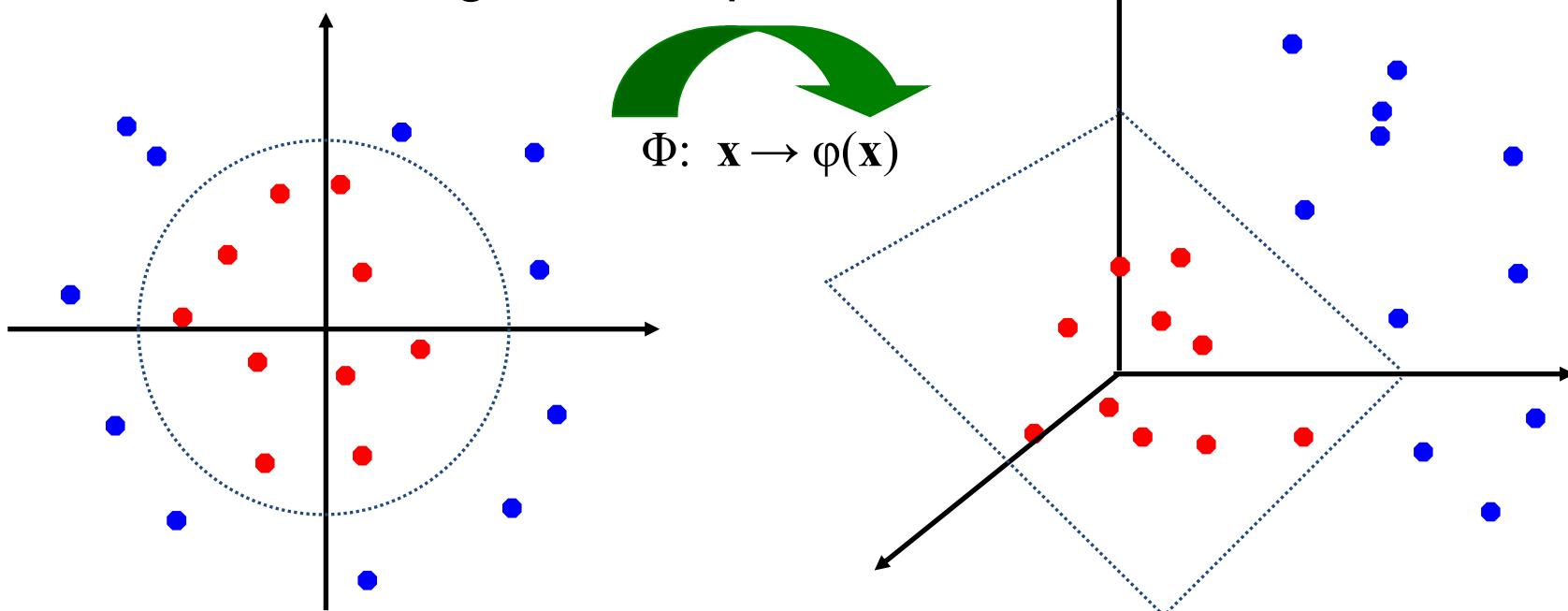
# SVM Kernels

---

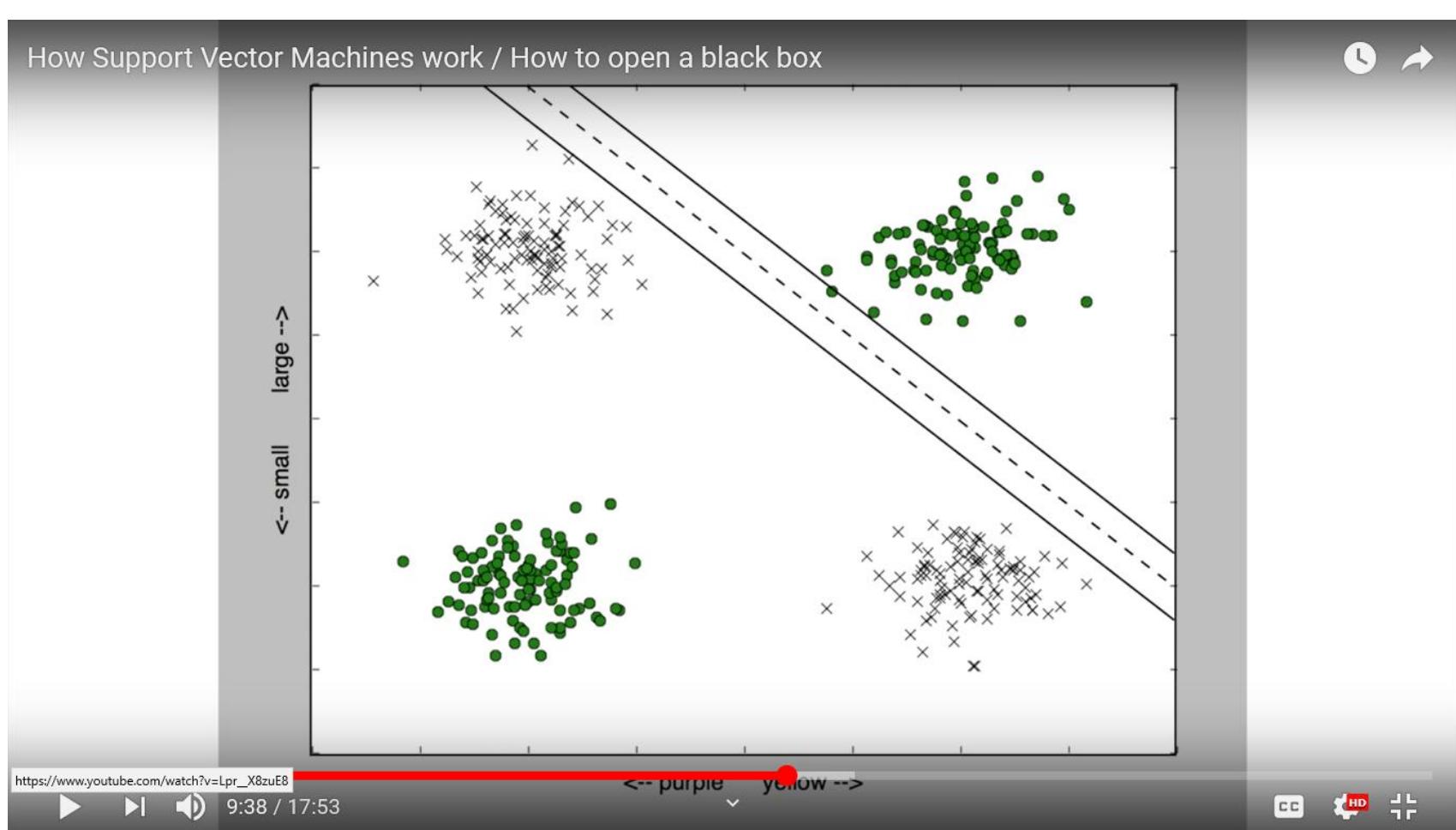
- SVM algorithms use a set of mathematical functions that are defined as the kernel.
- Function of kernel is to take data as input and transform it into the required form.
- Different SVM algorithms use different types of kernel functions. Example *linear, nonlinear, polynomial, and sigmoid etc.*

# 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:

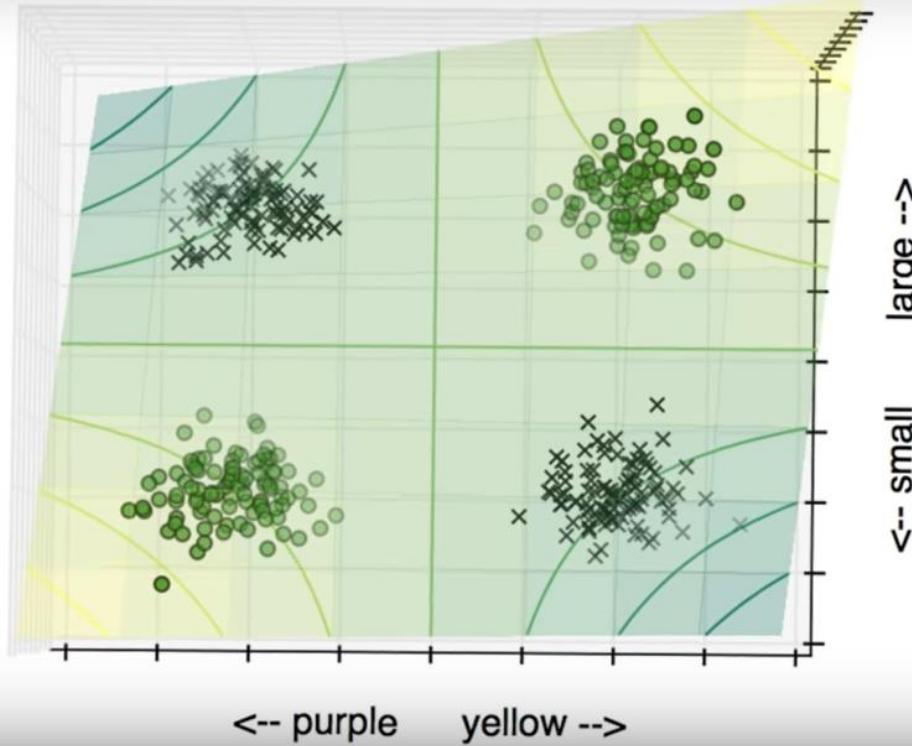


# Non-linear SVMs



# Non-linear SVMs: Feature spaces

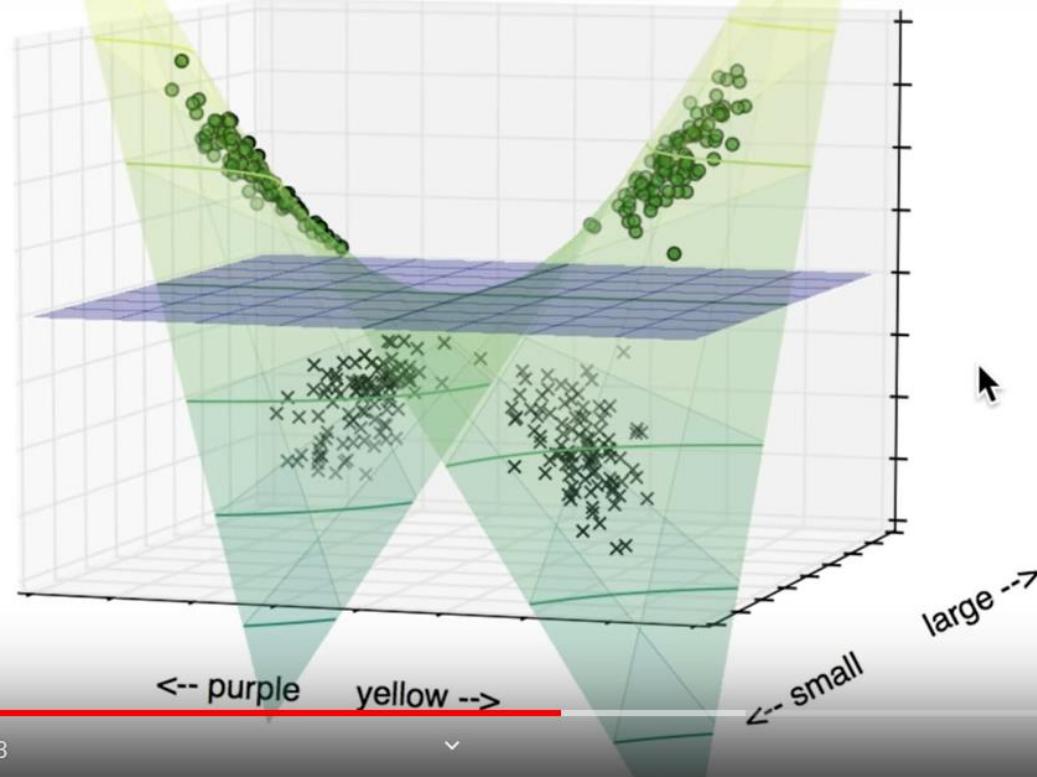
How Support Vector Machines work / How to open a black box



# Non-linear SVMs: Feature spaces



How Support Vector Machines work / How to open a black box



▶ ▶ 🔍 10:08 / 17:53



# What Functions are Kernels?

---

- For some functions  $K(x_i, x_j)$  checking that  $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$  can be cumbersome.
- Mercer's theorem:  
*Every positive-semidefinite symmetric function is a kernel*

# What Functions are Kernels?

---

1) We can *construct kernels from scratch*:

- For any  $\varphi : \mathcal{X} \rightarrow \mathbb{R}^m$ ,  $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$  is a kernel.
- If  $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  is a *distance function*, i.e.
  - $d(x, x') \geq 0$  for all  $x, x' \in \mathcal{X}$ ,
  - $d(x, x') = 0$  only for  $x = x'$ ,
  - $d(x, x') = d(x', x)$  for all  $x, x' \in \mathcal{X}$ ,
  - $d(x, x') \leq d(x, x'') + d(x'', x')$  for all  $x, x', x'' \in \mathcal{X}$ ,

then  $k(x, x') := \exp(-d(x, x'))$  is a kernel.

2) We can *construct kernels from other kernels*:

- if  $k$  is a kernel and  $\alpha > 0$ , then  $\alpha k$  and  $k + \alpha$  are kernels.
- if  $k_1, k_2$  are kernels, then  $k_1 + k_2$  and  $k_1 \cdot k_2$  are kernels.

# 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

- The solution is:

$$f(\mathbf{x}) = \sum \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- Optimization techniques for finding  $\alpha_i$ 's remain the same!

# Non-linear SVM using kernel

---

1. Select a kernel function.
2. Compute pairwise kernel values between labeled examples.
3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.

# 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.

# Multi-Class Problem

---

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

# Multi-Class Problem

---

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

# Multi-Class Problem

---

## One-vs-all (a.k.a. one-vs-others)

- Train C classifiers
- In each, pos = data from class  $i$ , neg = data from classes other than  $i$
- The class with the most confident prediction wins
- Example:
  - You have 4 classes, train 4 classifiers
  - 1 vs others: score 3.5
  - 2 vs others: score 6.2
  - 3 vs others: score 1.4
  - 4 vs other: score 5.5
  - Final prediction: class 2
- Issues?

# Multi-Class Problem

---

## One-vs-one (a.k.a. all-vs-all)

- Train  $C(C-1)/2$  binary classifiers (all pairs of classes)
- They all vote for the label
- Example:
  - You have 4 classes, then train 6 classifiers
  - 1 vs 2, 1 vs 3, 1 vs 4, 2 vs 3, 2 vs 4, 3 vs 4
  - Votes: 1, 1, 4, 2, 4, 4
  - Final prediction is class 4

# SVM versus Logistic Regression

- When viewed from the point of view of regularized empirical loss minimization, SVM and logistic regression appear quite similar:

$$\text{SVM: } \sum_{i=1}^n \left(1 - y_i [w_0 + \mathbf{x}_i^T \mathbf{w}_1]\right)^+ + \|\mathbf{w}_1\|^2/2$$

$$\text{Logistic: } \sum_{i=1}^n \underbrace{-\log P(y_i|\mathbf{x}, \mathbf{w})}_{-\log \sigma(y_i [w_0 + \mathbf{x}_i^T \mathbf{w}_1])} + \|\mathbf{w}_1\|^2/2$$

where  $\sigma(z) = (1 + \exp(-z))^{-1}$  is the logistic function.

# SVM versus Logistic Regression

- The difference comes from how we penalize “errors”:

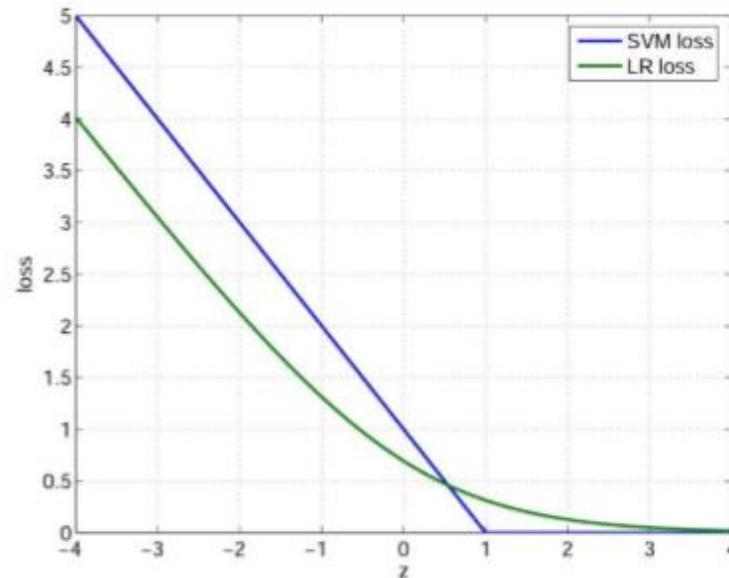
Both: 
$$\sum_{i=1}^n \text{Loss}\left(\overbrace{y_i [w_0 + \mathbf{x}_i^T \mathbf{w}_1]}^z\right) + \|\mathbf{w}_1\|^2/2$$

- SVM:

$$\text{Loss}(z) = (1 - z)^+$$

- Regularized logistic reg:

$$\text{Loss}(z) = \log(1 + \exp(-z))$$

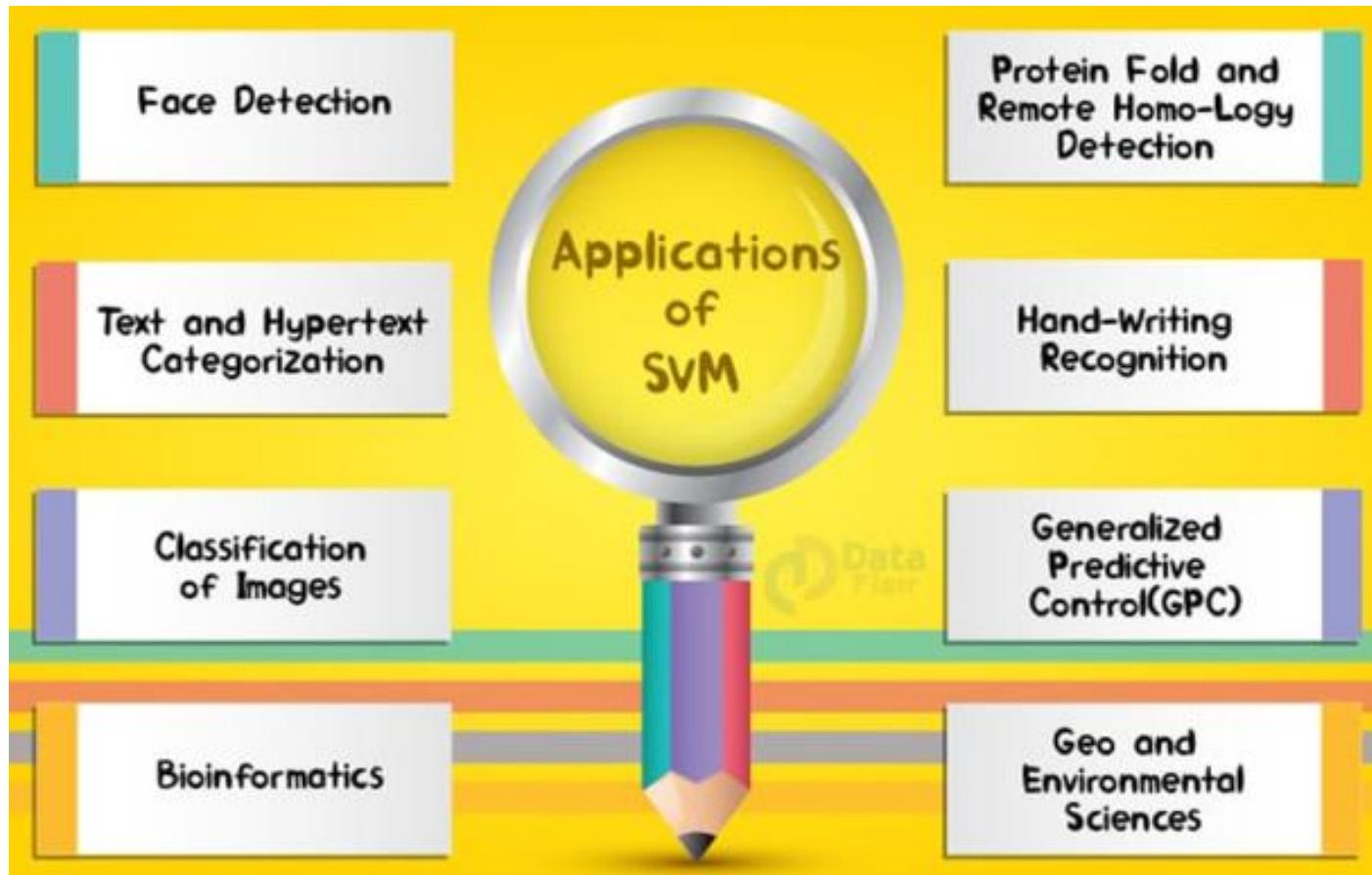


# Properties of SVM

- **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
  - Therefore SVM also called sparse kernel machine.
- **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



# Application : Text Categorization

---

- Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content. 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

# Text Categorization using SVM

---

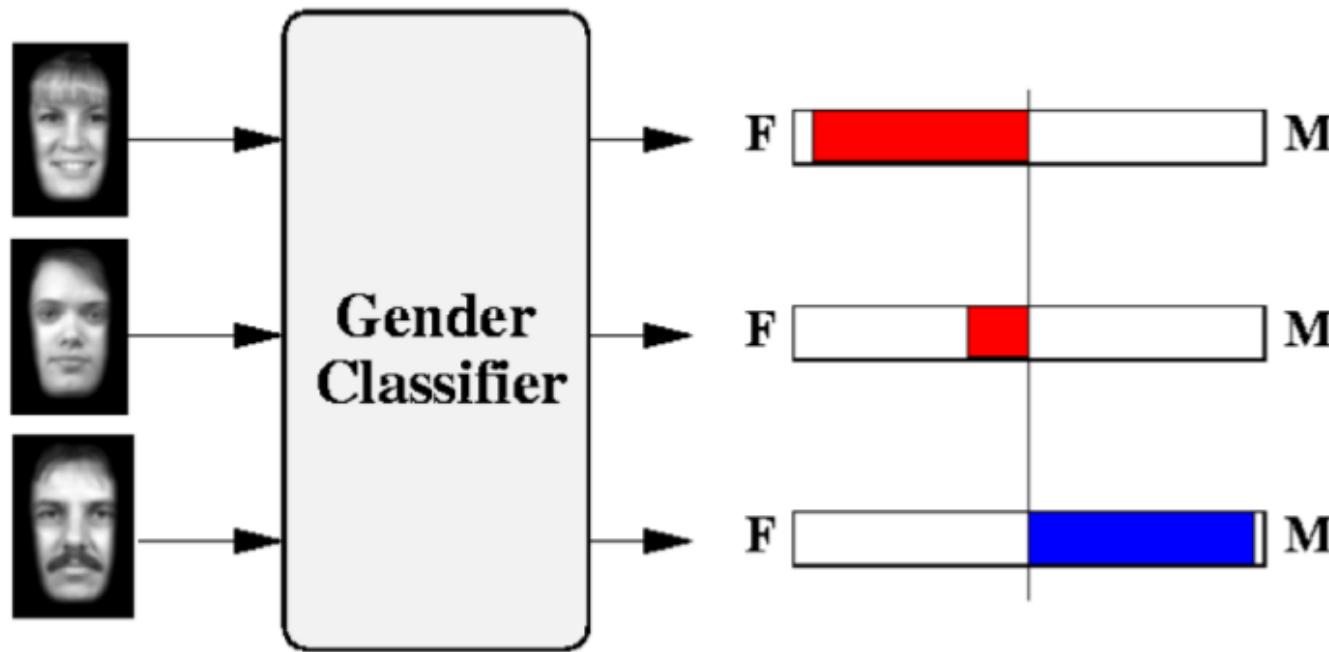
- The distance between two documents is  $\phi(x) \cdot \phi(z)$
- $K(x,z) = \phi(x) \cdot \phi(z)$  is a valid kernel, SVM can be used with  $K(x,z)$  for discrimination.
- Why SVM?
  - High dimensional input space
  - Few irrelevant features (dense concept)
  - Sparse document vectors (sparse instances)
  - Text categorization problems are linearly separable

# Using SVM

---

1. Select a kernel function.
  2. Compute pairwise kernel values between labeled examples.
  3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
  4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.
-

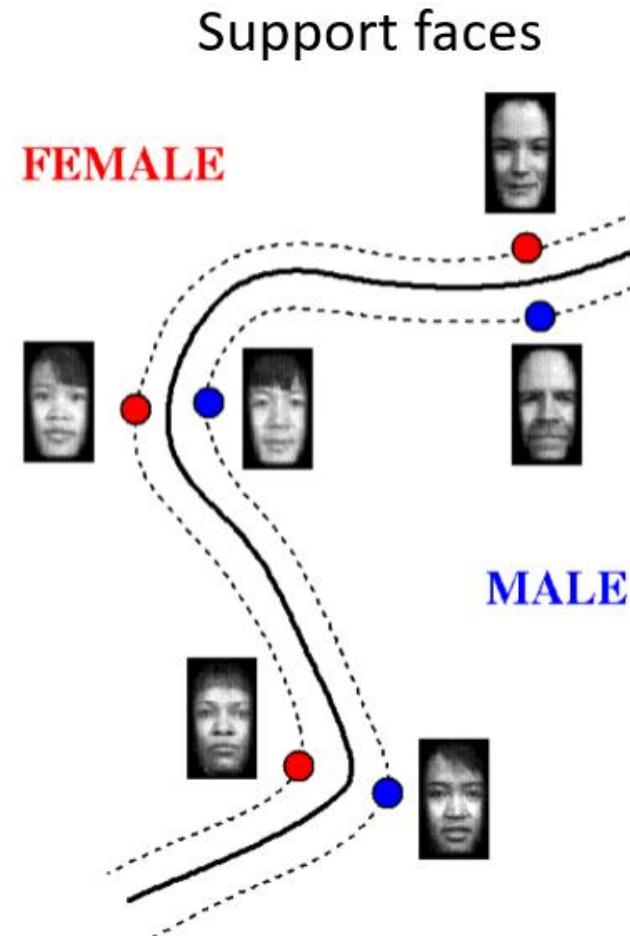
# Learning Gender from image with SVM



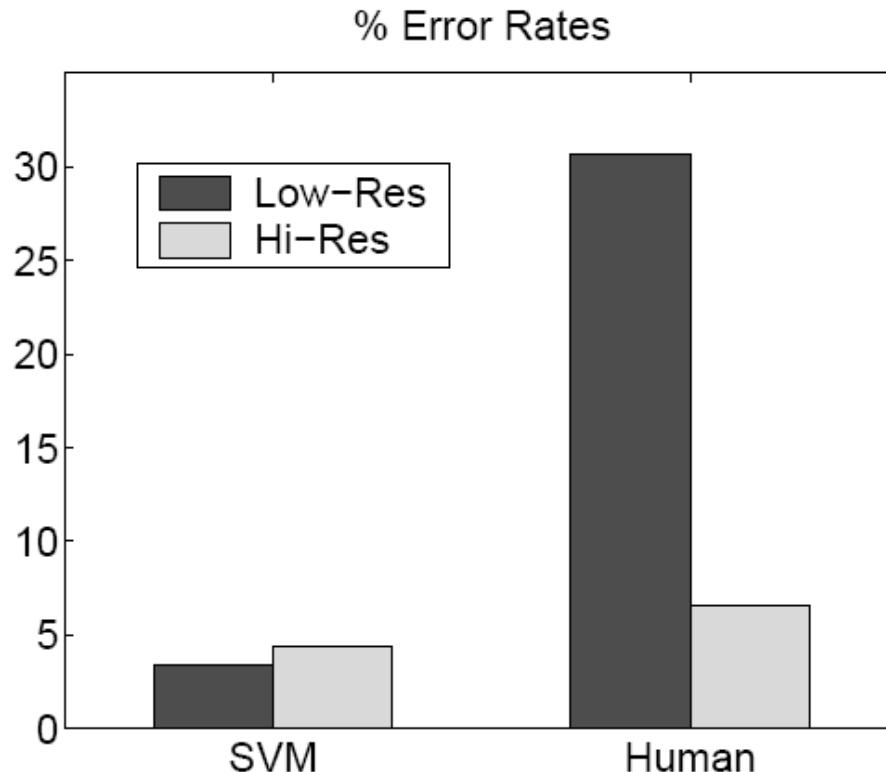
Moghaddam and Yang, Learning Gender with Support Faces, TPAMI 2002

Moghaddam and Yang, Face & Gesture 2000

# Support faces



# Accuracy of SVM Classifier



- SVMs performed better than humans, at either resolution

**Figure 6. SVM vs. Human performance**

# Some Issues

---

- **Sensitive to noise**
    - A relatively small number of mislabeled examples can dramatically decrease the performance
  - **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.  $\sigma$  in Gaussian kernel
    - $\sigma$  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
-

# Good Web References for SVM

- **Text categorization with Support Vector Machines: learning with many relevant features** - T. Joachims, ECML
- **A Tutorial on Support Vector Machines for Pattern Recognition**, Kluwer Academic Publishers - Christopher J.C. Burges
- <http://www.cs.utexas.edu/users/mooney/cs391L/>
- <https://www.coursera.org/learn/machine-learning/home/week/7>
- <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- <https://data-flair.training/blogs/svm-kernel-functions/>
- [MIT 6.034 Artificial Intelligence, Fall 2010](#)
- <https://stats.stackexchange.com/questions/30042/neural-networks-vs-support-vector-machines-are-the-second-definitely-superior>
- <https://www.sciencedirect.com/science/article/abs/pii/S0893608006002796>
- <https://medium.com/deep-math-machine-learning-ai/chapter-3-support-vector-machine-with-math-47d6193c82be>
- [Radial basis kernel](#)
- <http://www.engr.mun.ca/~baxter/Publications/LagrangeForSVMs.pdf>



---

# Thank You



**BITS** Pilani  
Pilani Campus

# Unsupervised Learning

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Bishop and many others who made their course materials freely available online.

# Topics to be covered



Ref: Christopher Bishop: Chapter 9

---

- Unsupervised learning
- Clustering
- K-means Clustering
- Gaussian Mixture Models
- EM algorithm

# Unsupervised Learning

- We only use the features X, not the labels Y
- This is useful because we may not have any labels but we can still detect patterns
- For example:
  - We can detect that news articles revolve around certain topics, and group them accordingly
  - Discover a distinct set of objects appear in a given environment, even if we don't know their names, then ask humans to label each group
  - Identify health factors that correlate with a disease

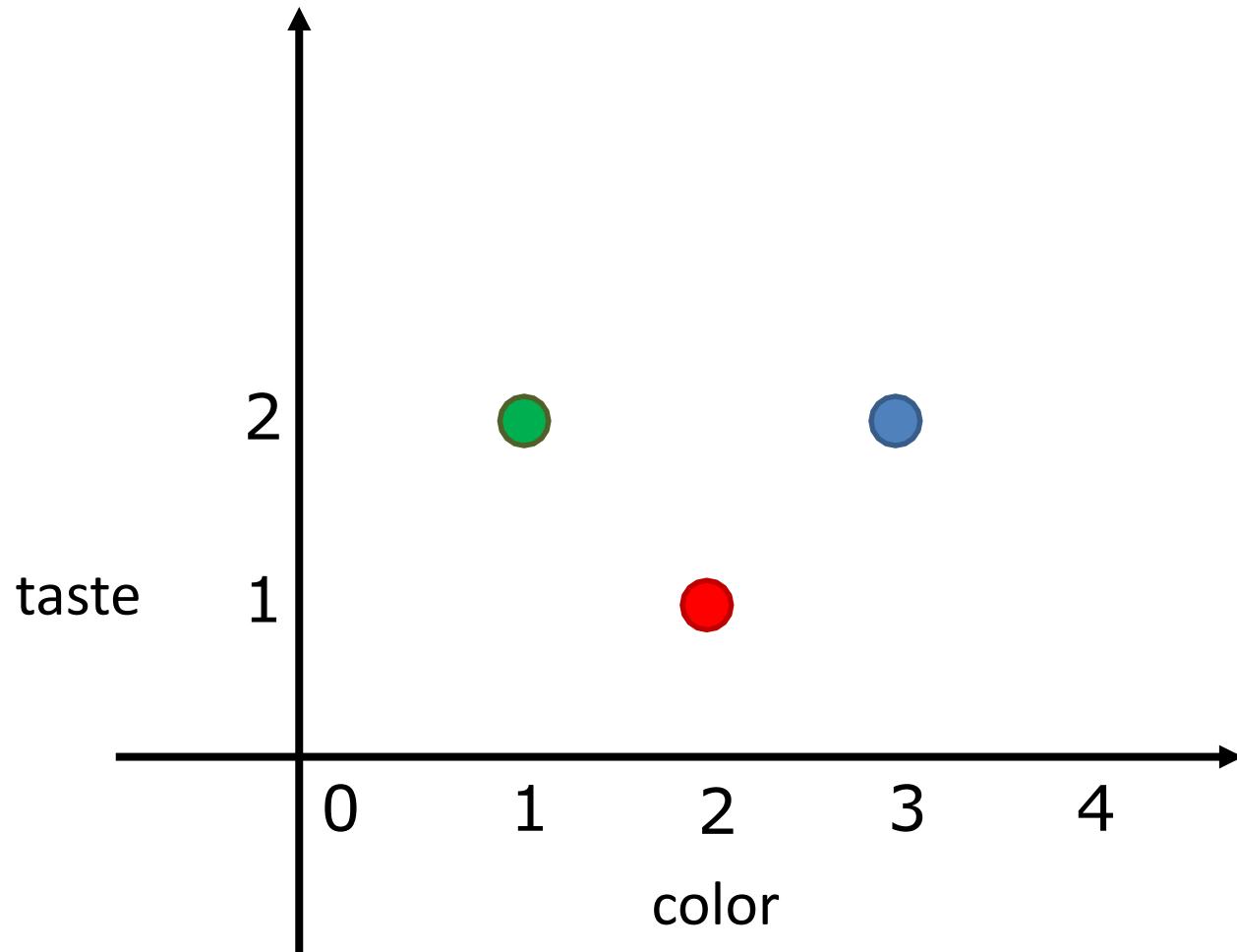
# What is clustering?

- Grouping items that “belong together” (i.e. have similar features)

# Feature representation ( $x$ )

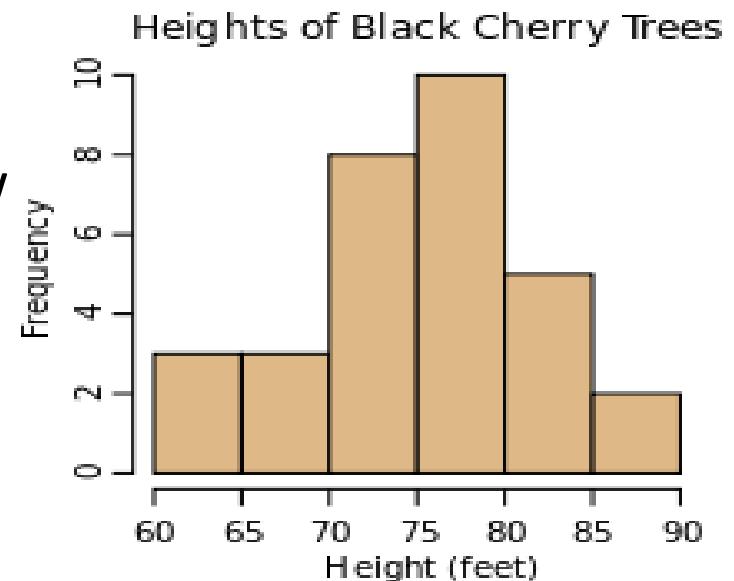
- A vector representing measurable characteristics of a data sample we have
- E.g. a glass of juice can be represented via its color = {yellow=1, red=2, green=3, purple=4} and taste = {sweet=1, sour=2}
- For a given glass  $i$ , this can be represented as a vector:  $x_i = [3 \ 2]$  represents sour green juice
- For  $D$  features, this defines a  $D$ -dimensional space where we can measure similarity between samples

# Feature representation ( $x$ )



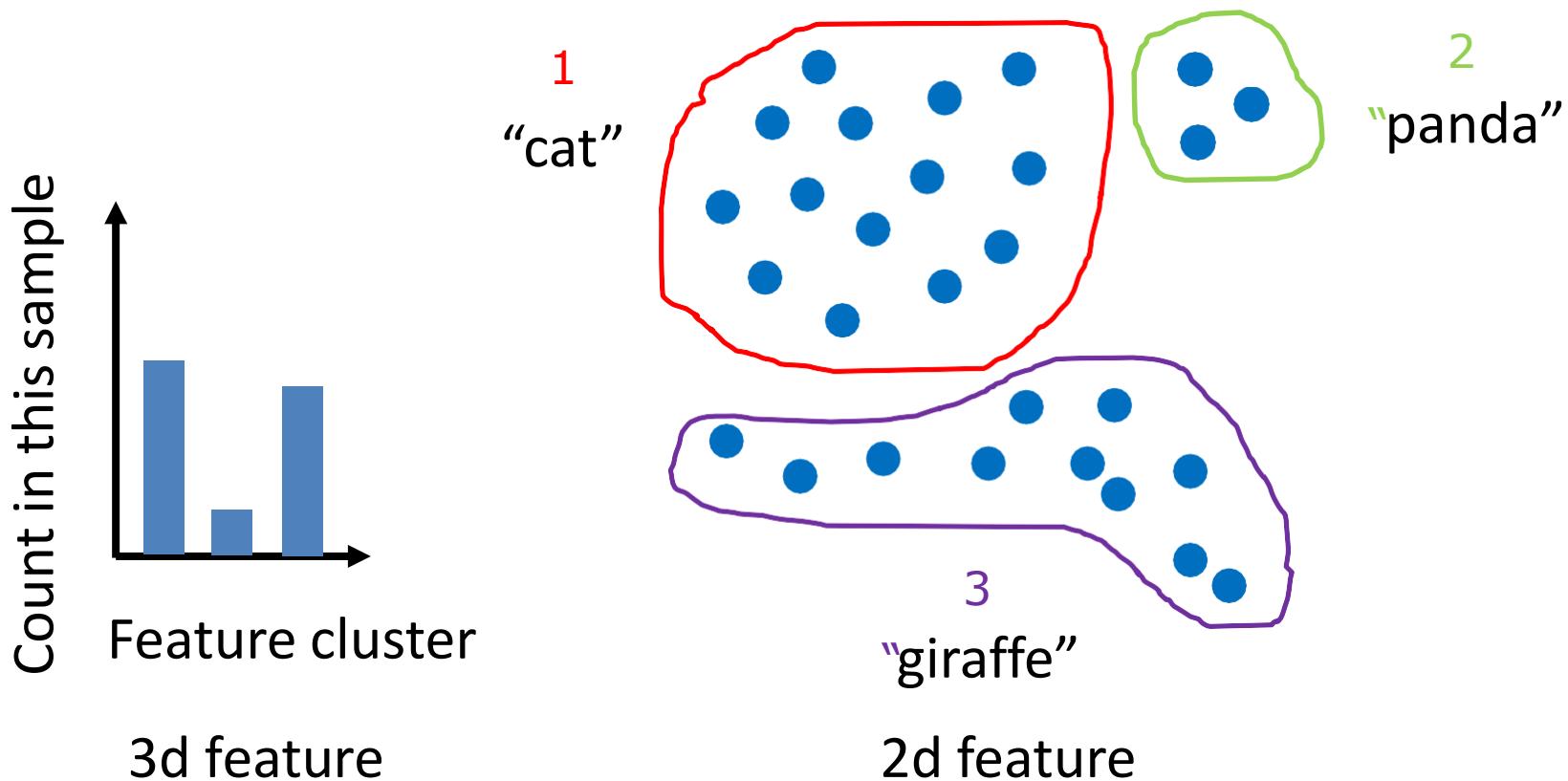
# Why do we cluster?

- Counting
  - Feature histograms: by grouping similar features and counting how many of each a data sample has
- Summarizing data
  - Look at large amounts of data
  - Represent a large continuous vector with the cluster number
- Prediction
  - Data points in the same cluster may have the same labels
  - Ask a human to label the clusters

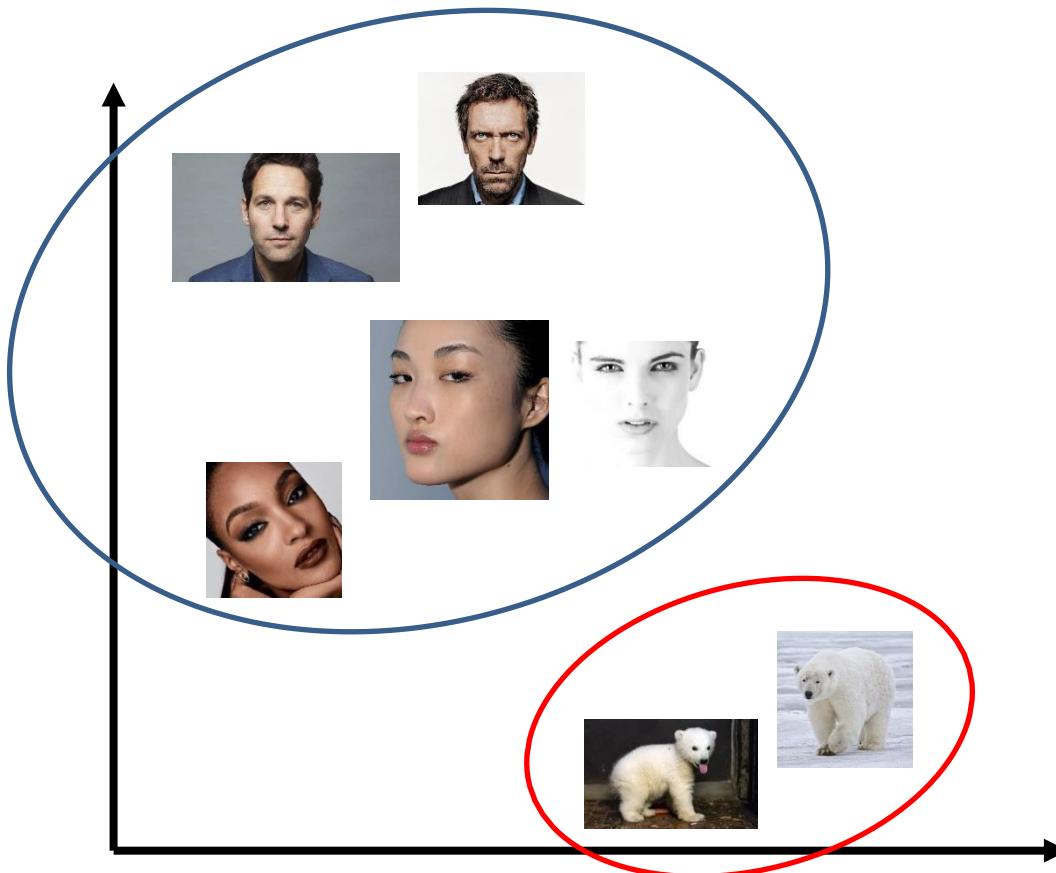


# Why do we cluster?

- Two uses of clustering in one application
- Cluster, then ask human to label groups
- Compute a histogram to summarize the data



# Unsupervised discovery

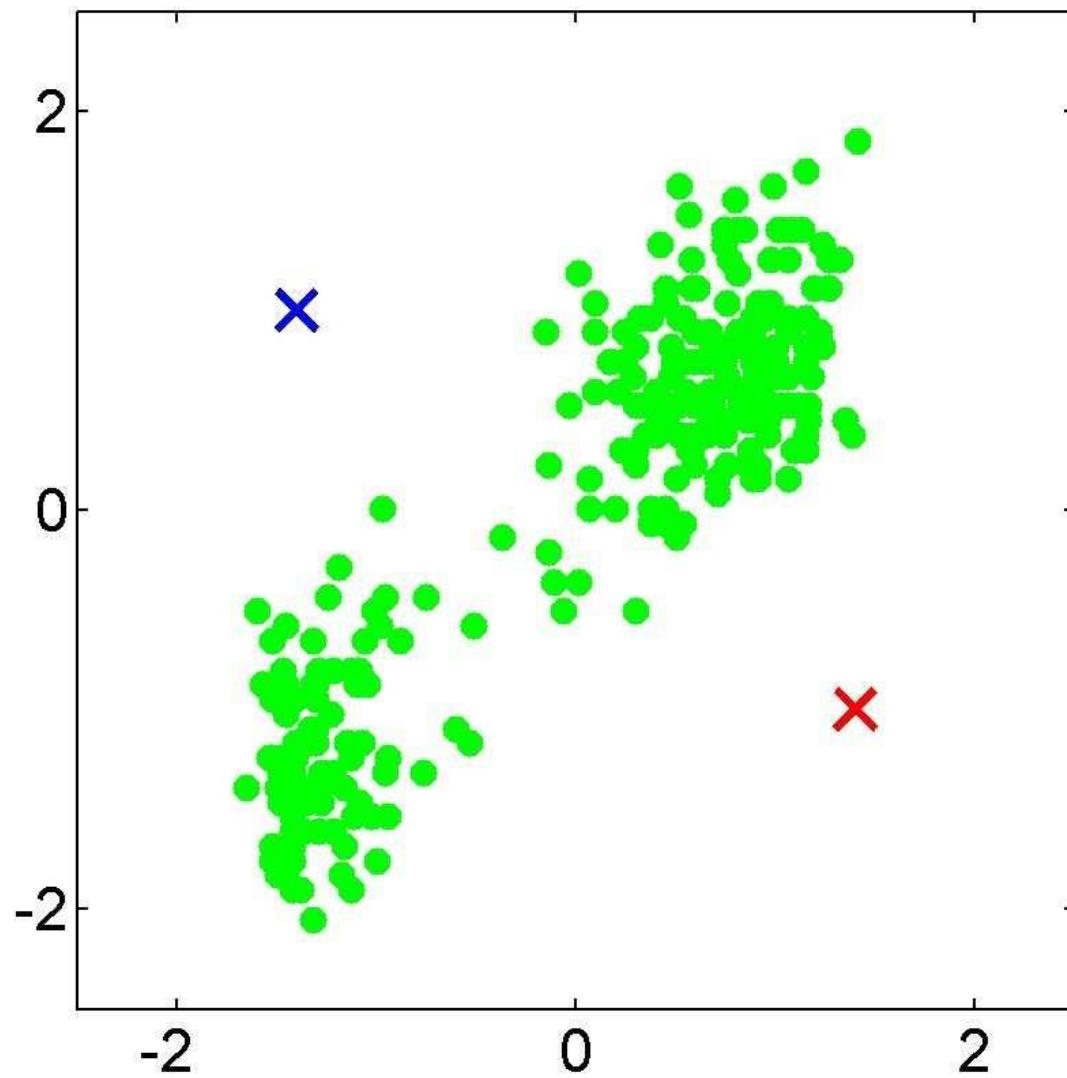


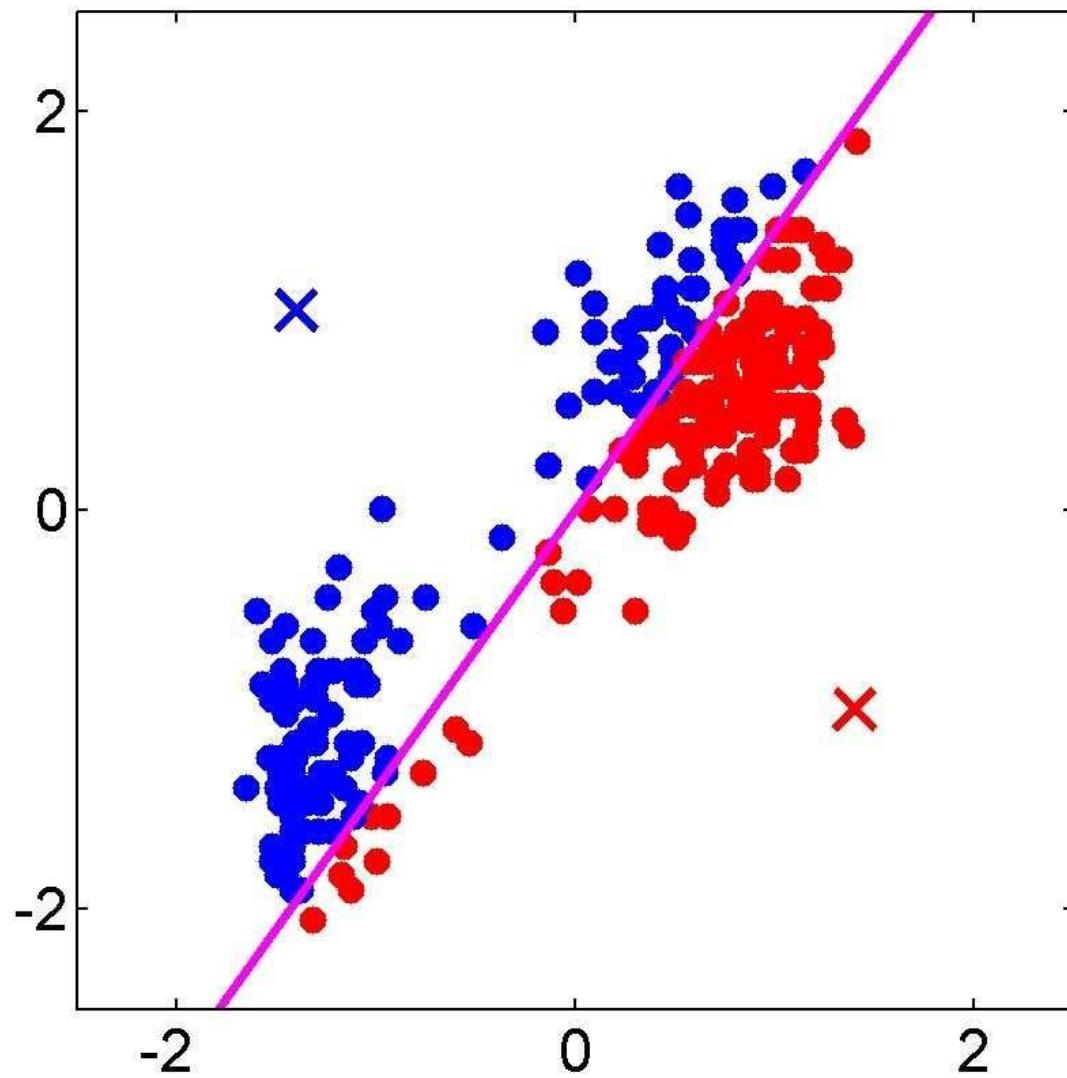
# Clustering algorithms

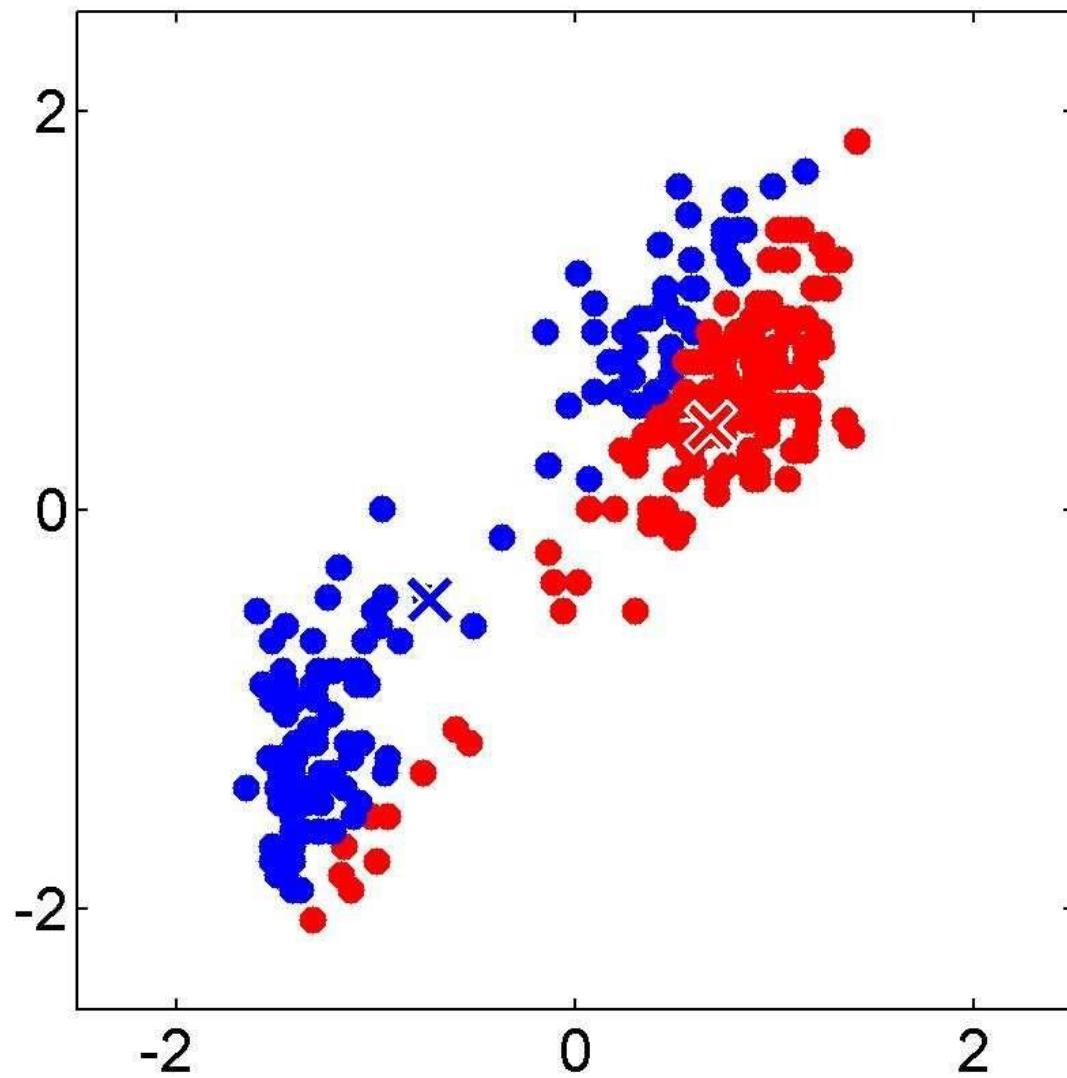
- In depth
  - K-means (iterate between finding centers and assigning points)
  - Gaussian Mixture Models (GMMs)

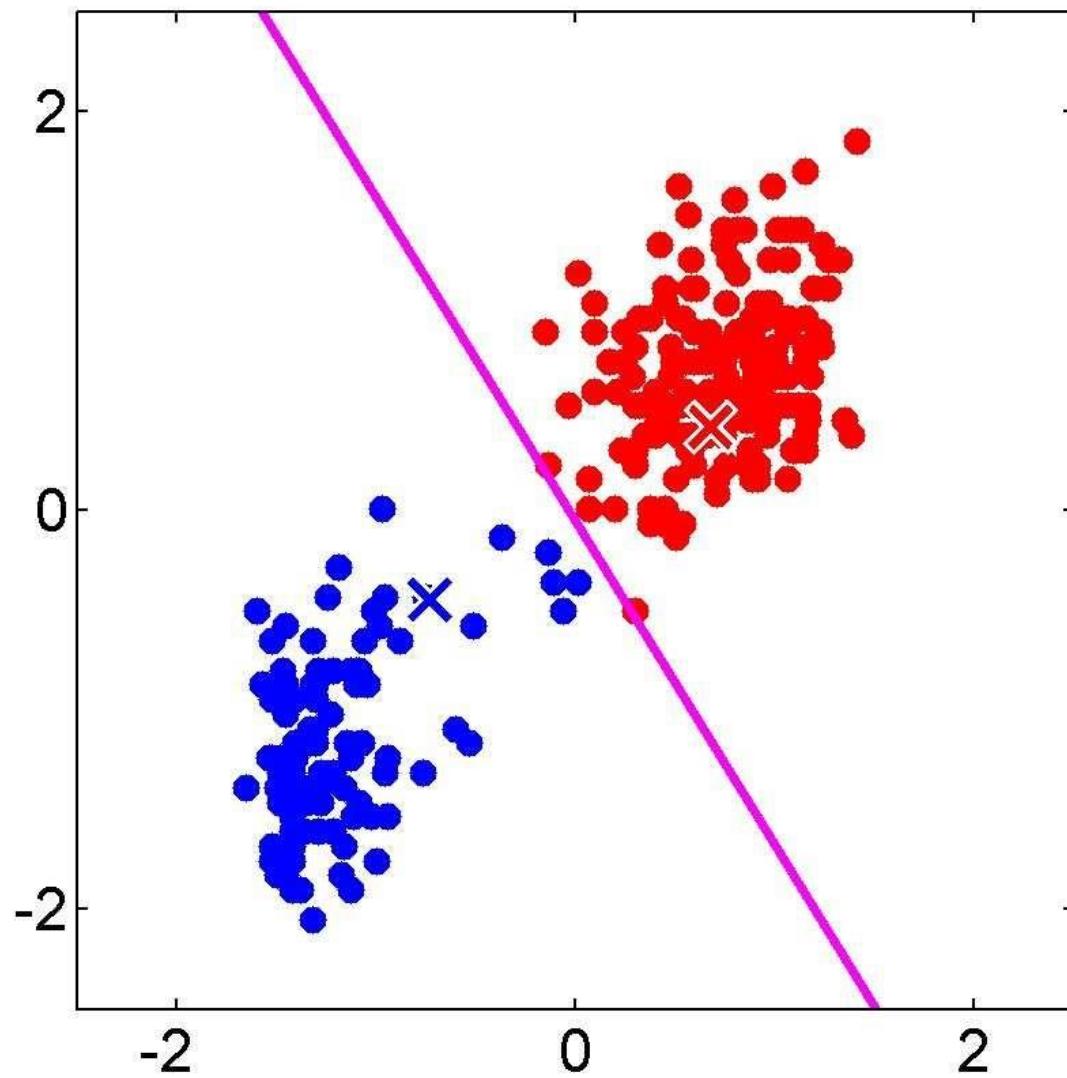
# K-means Algorithm

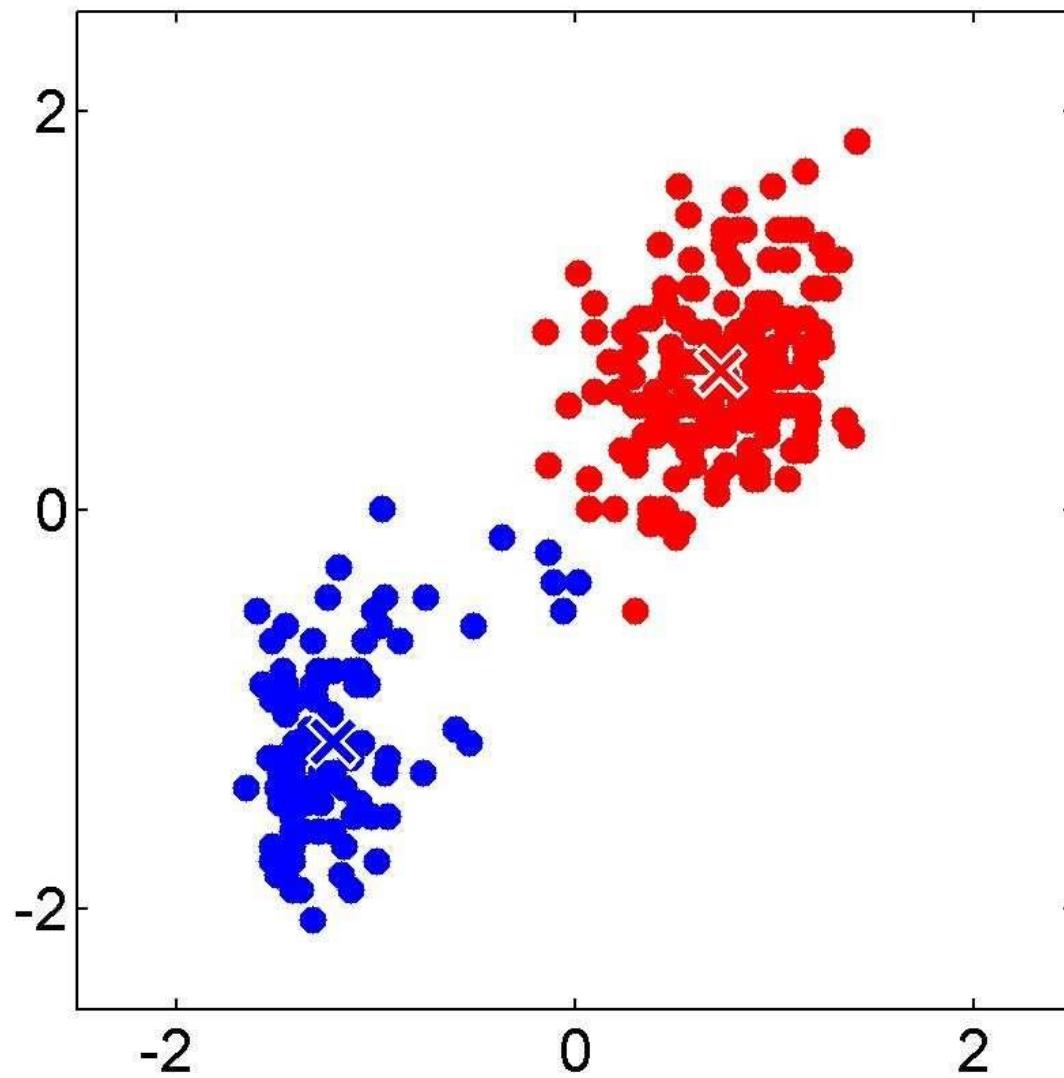
- Goal: represent a data set in terms of  $K$  clusters each of which is summarized by a prototype  $\mu_k$
- Initialize prototypes, then iterate between two phases:
  - E-step: assign each data point to nearest prototype
  - M-step: update prototypes to be the cluster means
- Simplest version is based on Euclidean distance
  - re-scale Old Faithful data

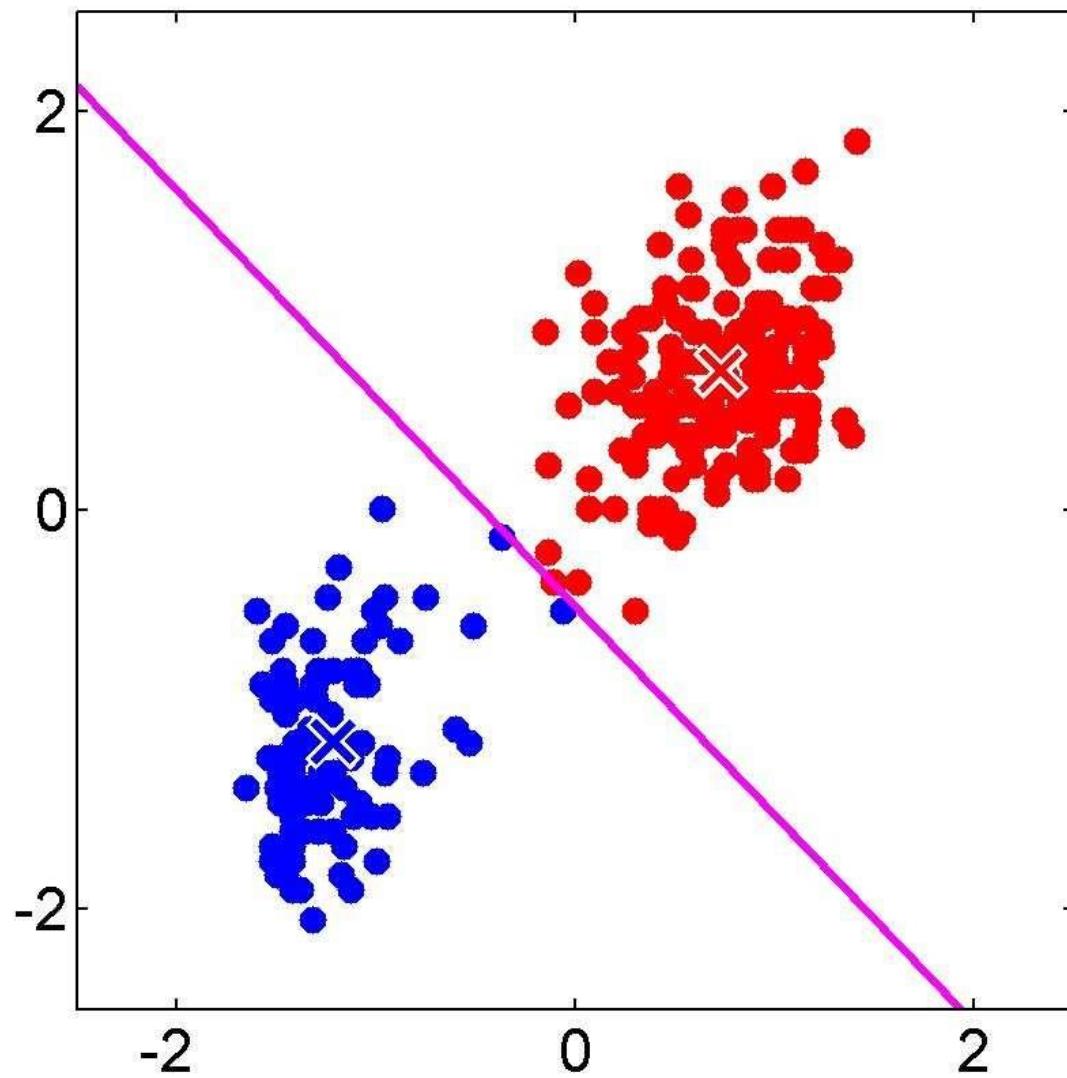


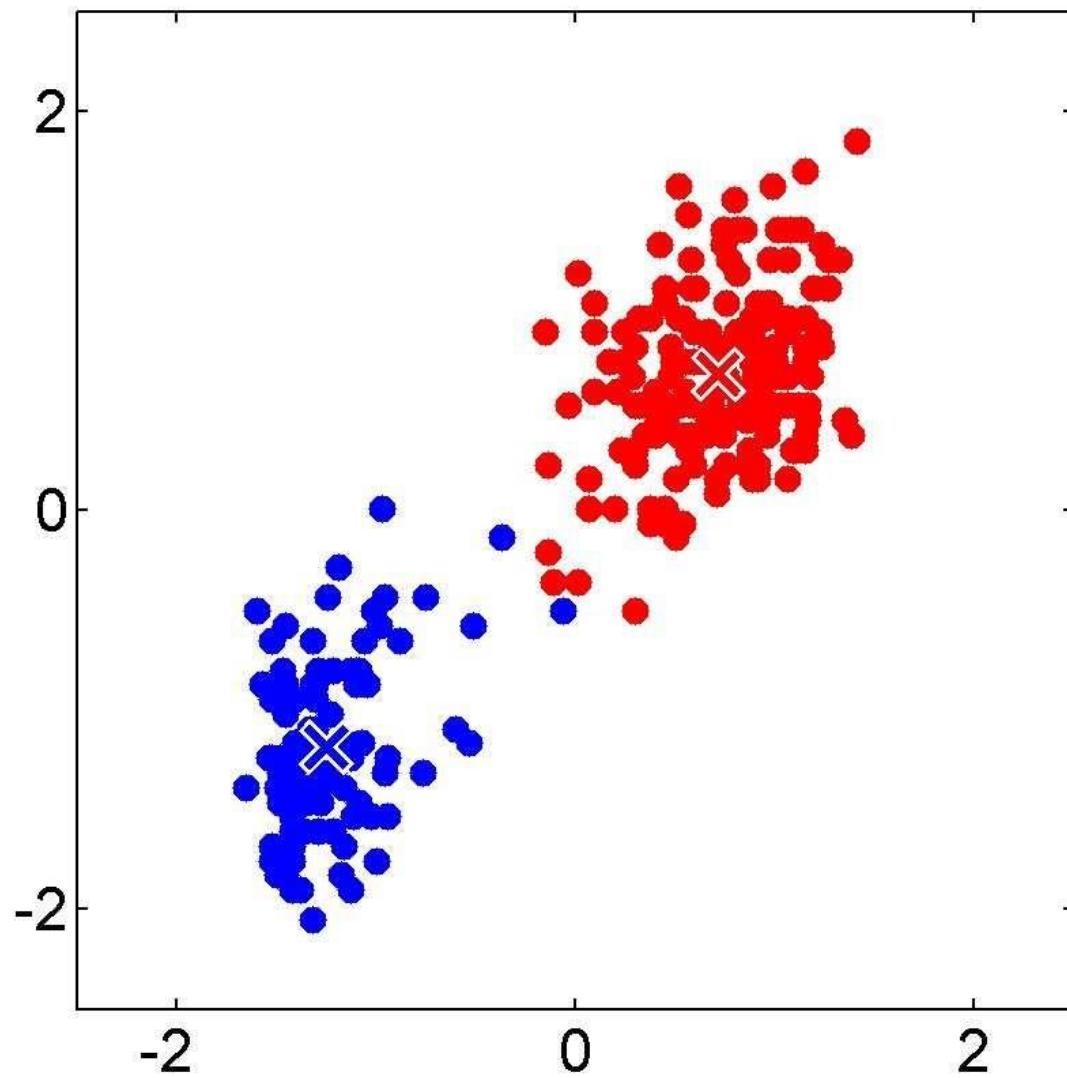


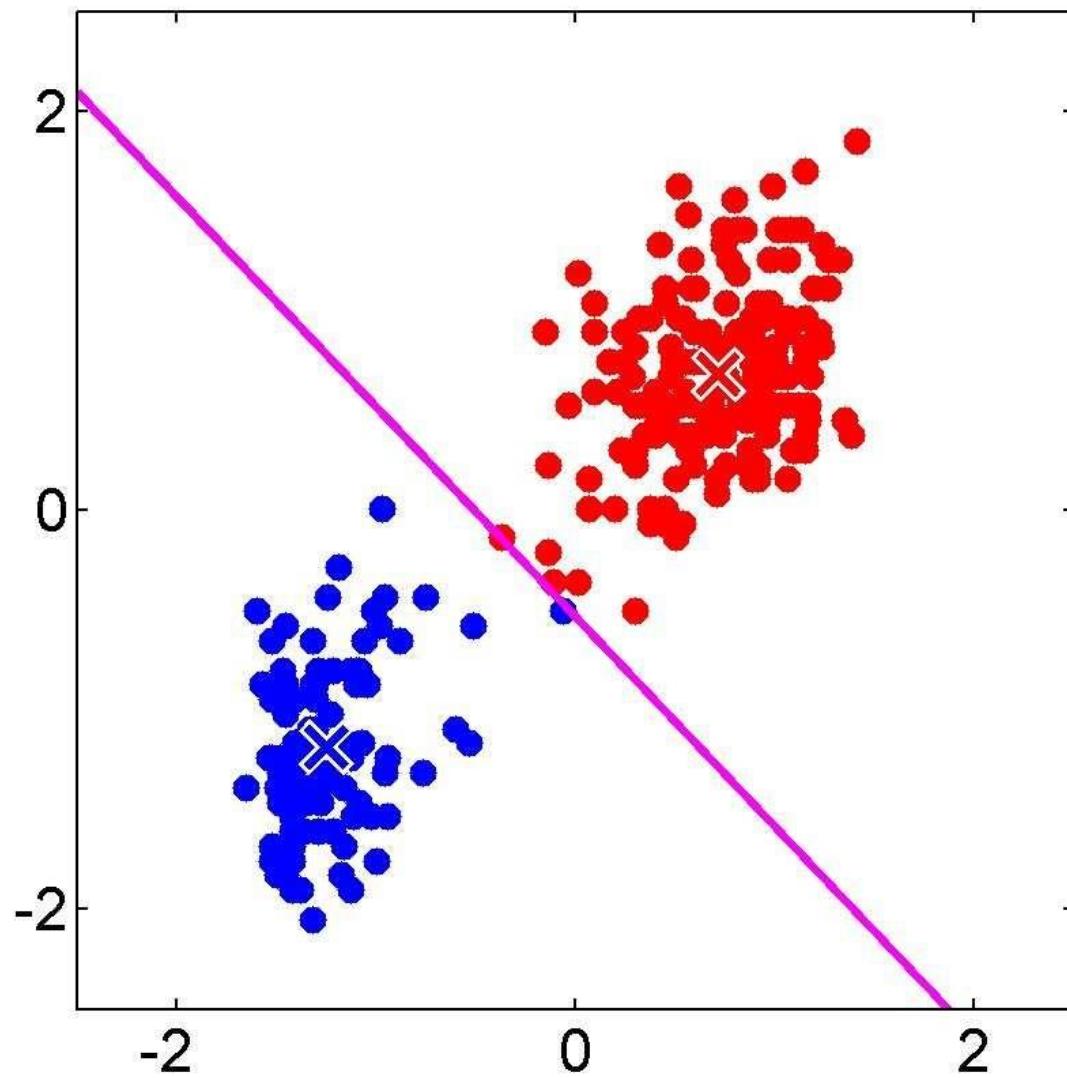


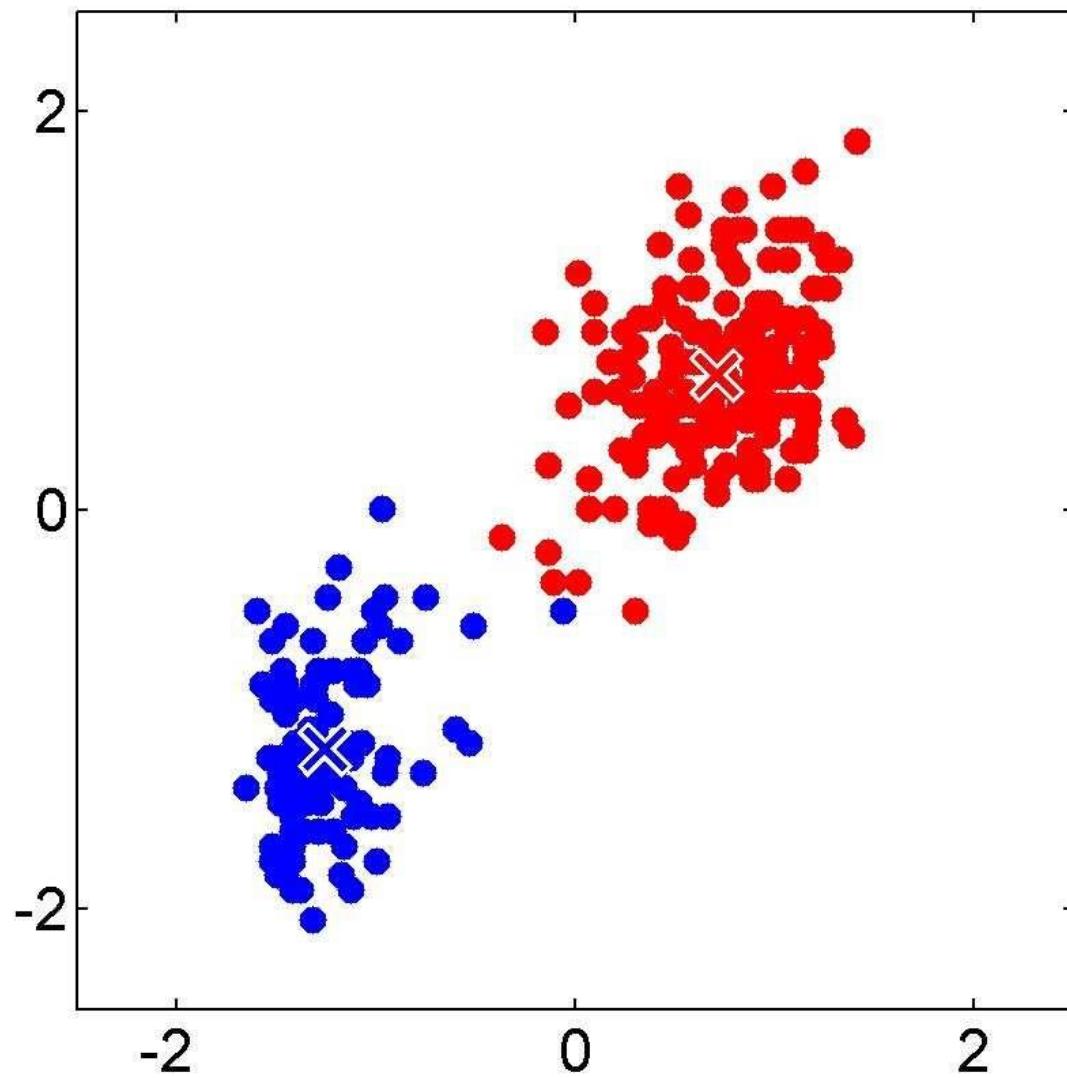












# 1 of K coding mechanism

- For each data point  $x_n$ , we introduce a set of binary indicator variables  $r_{nk} \in \{0, 1\}$  such that  $\sum_k r_{nk} = 1$

where  $k = 1, \dots, K$  describing which of the  $K$  clusters the data point  $x_n$  is assigned to, so that if data point  $x_n$  is assigned to cluster  $k$  then  $r_{nk} = 1$ , and  $r_{nj} = 0$  for  $j$  not equal to  $k$ .

- Example: 5 data points and 3 clusters

$$(r_{nk}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

# K-means Cost Function

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|x_n - \mu_k\|^2$$

Diagram illustrating the K-means Cost Function:

- The cost function  $J$  is the sum of weighted squared distances between data points and prototypes.
- The term  $r_{nk}$  is labeled "responsibilities", indicating the weight or probability of assigning data point  $n$  to prototype  $k$ .
- The term  $\|x_n - \mu_k\|^2$  is labeled "data" and "prototypes", representing the squared Euclidean distance between data point  $n$  and prototype  $k$ .

# Minimizing the Cost Function

- E-step: minimize  $J$  w.r.t.  $r_{nk}$

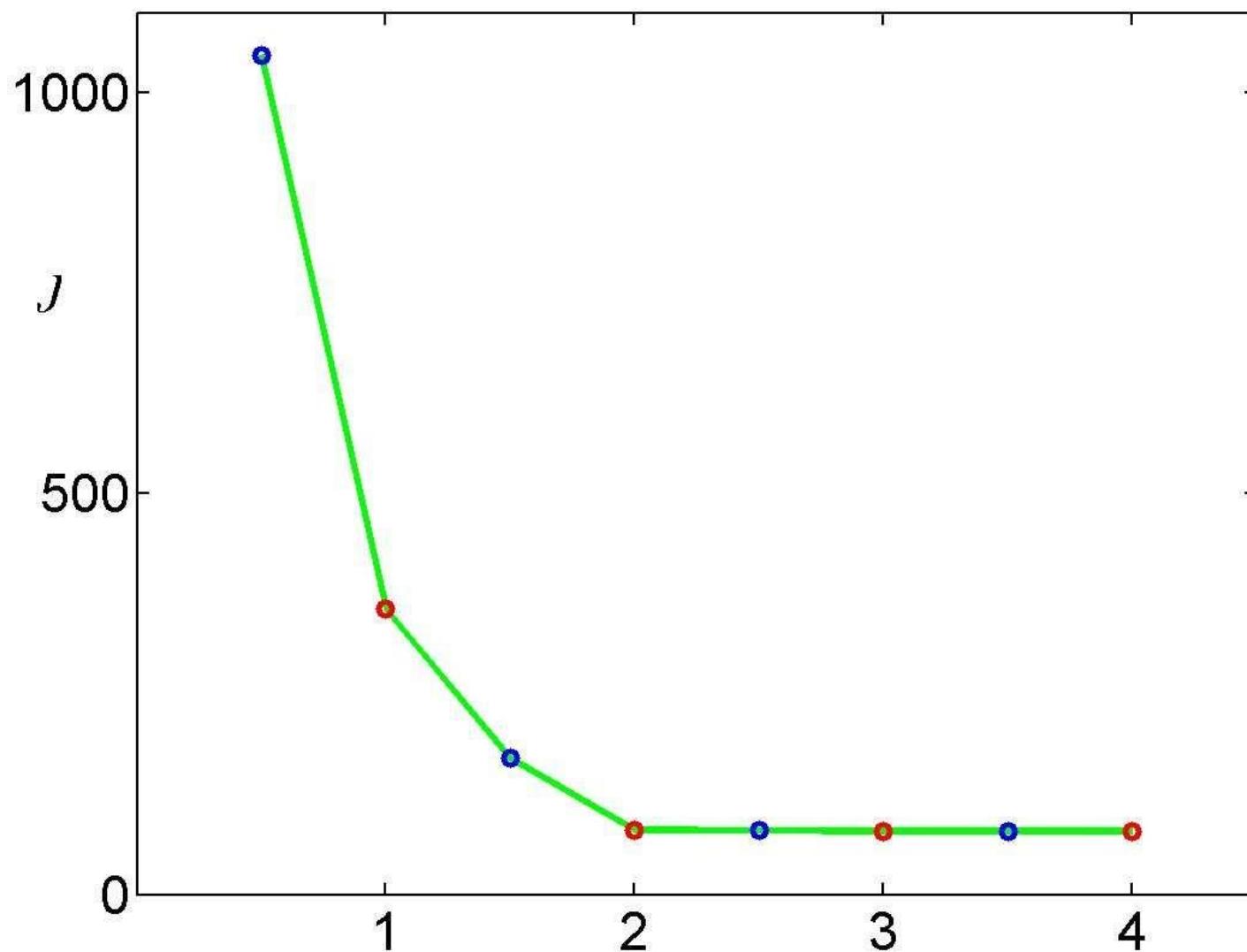
$$\bar{J} = \sum_n \sum_k r_{nk} \|x_n - \mu_k\|^2$$

- M-step: minimize  $J$  w.r.t.  $\mu_k$

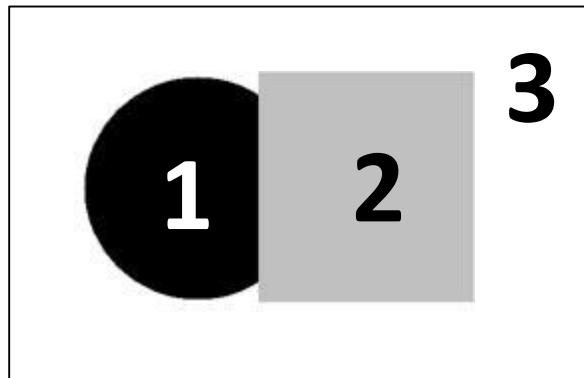
$$O = \sum_{n=1}^N r_{nj} (x_n - \mu_j)$$

$$\mu_j = \frac{\sum_n r_{nj} x_n}{\sum_n r_{nj}}$$

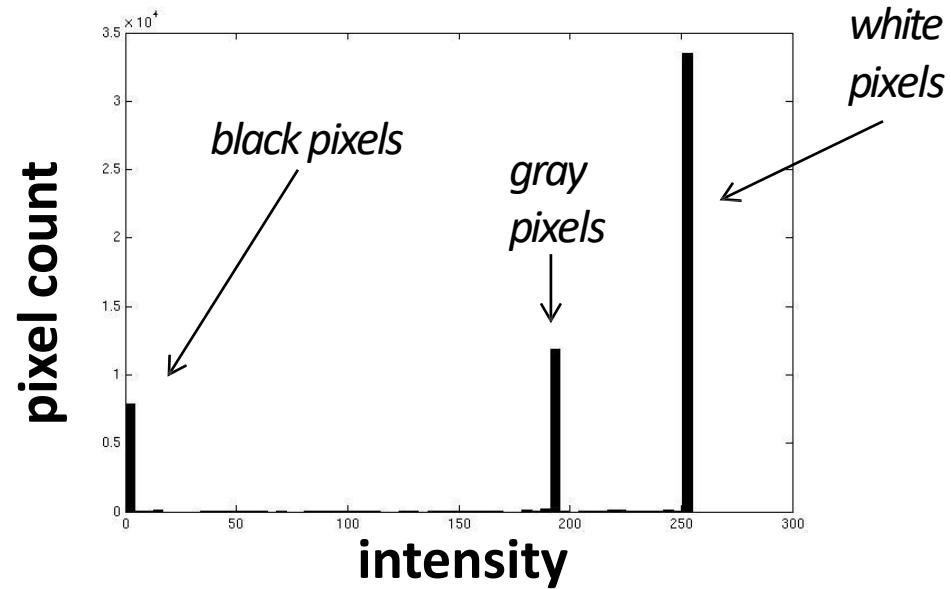
- Convergence guaranteed since there is a finite number of possible settings for the responsibilities



# Image segmentation: toy example



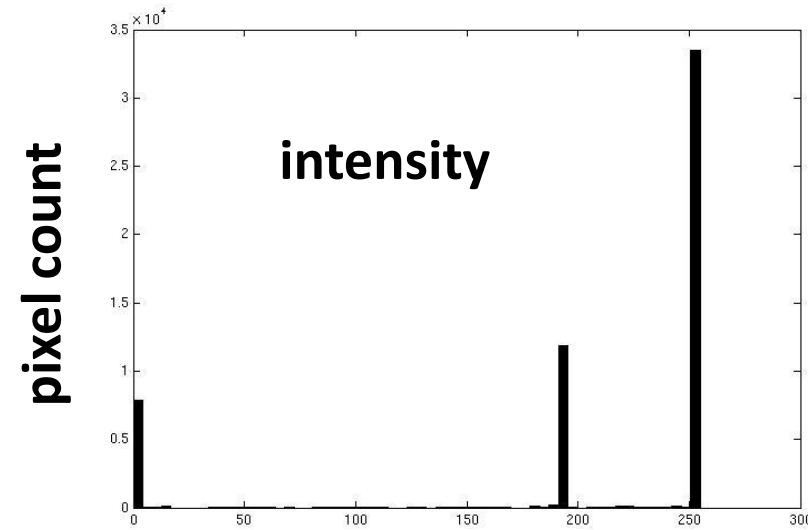
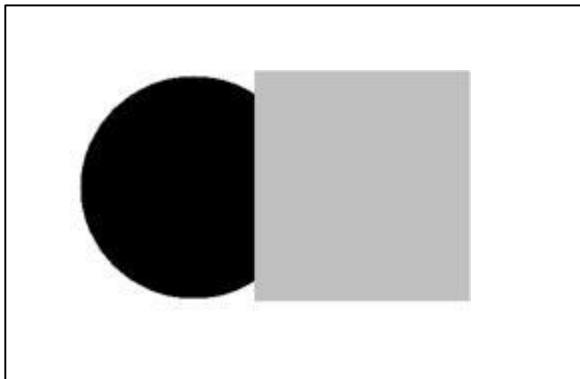
input image



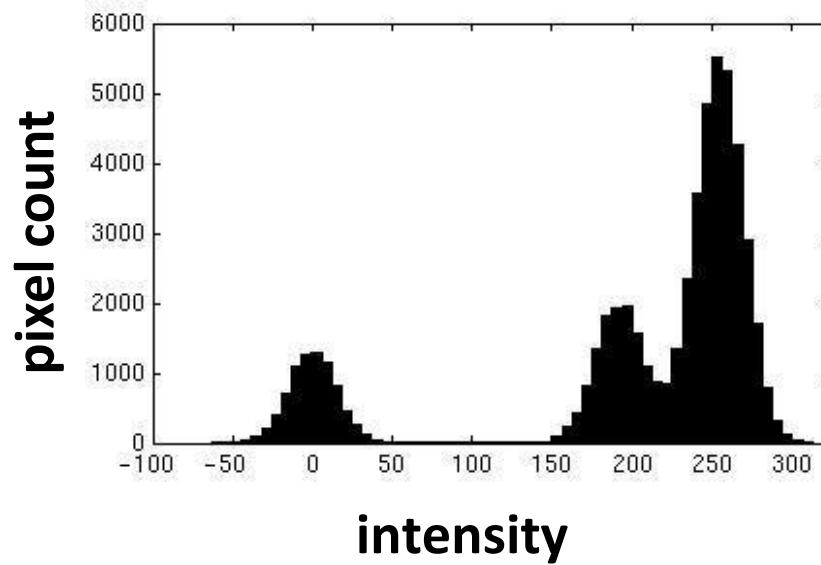
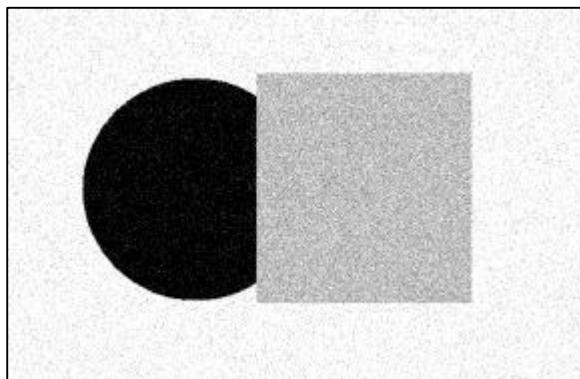
- These intensities define the three groups.
- We could label every pixel in the image according to which of these primary intensities it is.
  - i.e., *segment* the image based on the intensity feature.
- What if the image isn't quite so simple?

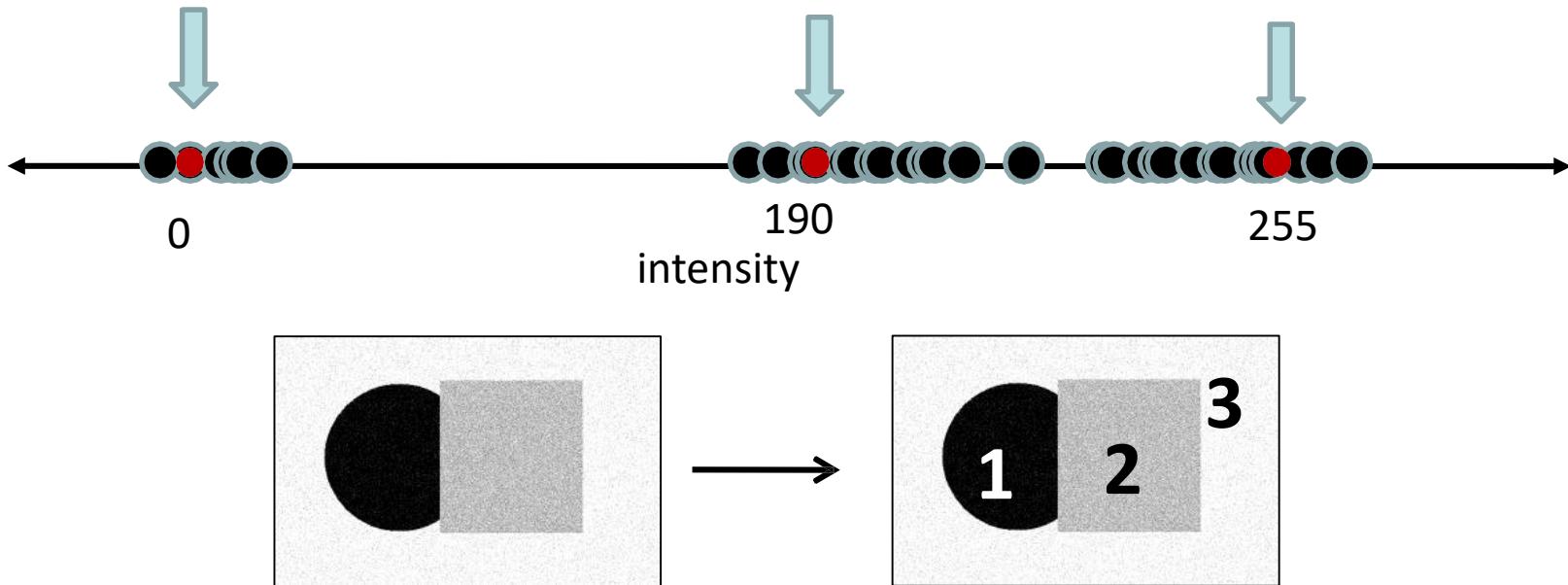
- Now how to determine the three main intensities that define our groups?
- We need to *cluster*.

**input image**



**input image**



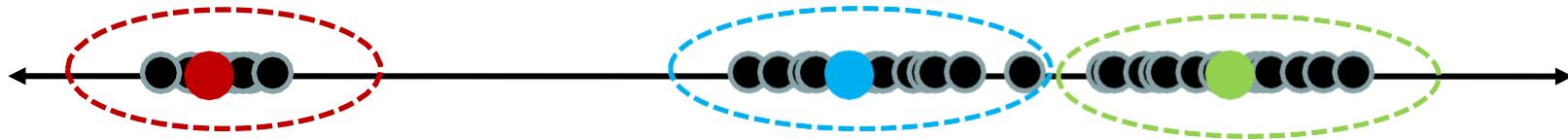


- Goal: choose three “centers” as the **representative** intensities, and label every pixel according to which of these centers it is nearest to.
- Best cluster centers are those that minimize SSD between all points and their nearest cluster center  $c_i$ :

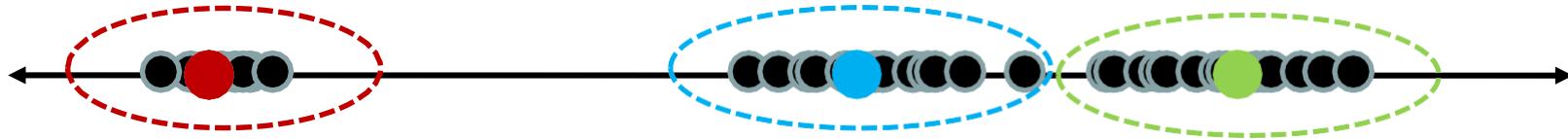
$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \|p - c_i\|^2$$

# Clustering

- With this objective, it is a “chicken and egg” problem:
  - If we knew the **cluster centers**, we could allocate points to groups by assigning each to its closest center.



- If we knew the **group memberships**, we could get the centers by computing the mean per group.



# K-means clustering

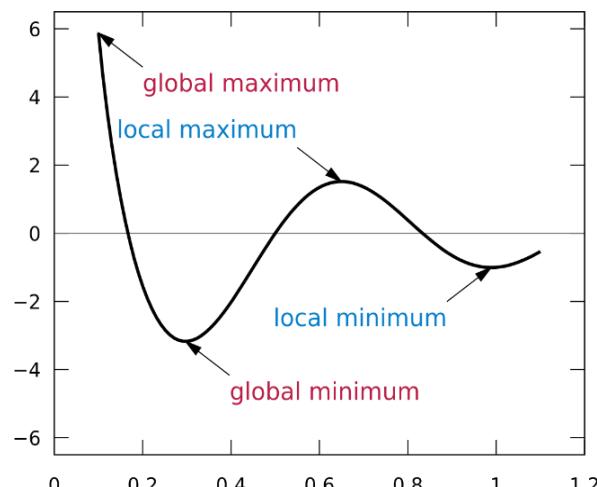
- Basic idea: randomly initialize the  $k$  cluster centers, and iterate between the two steps we just saw.
  1. Randomly initialize the **cluster centers**,  $c_1, \dots, c_K$
  2. Given **cluster centers**, determine **points** in each cluster
    - For each point  $p$ , find the closest  $c_i$ . Put  $p$  into **cluster i**
  3. Given **points** in each **cluster**, solve for  $c_i$ 
    - Set  $c_i$  to be the mean of **points** in **cluster i**
  4. If  $c_i$  have changed, repeat Step 2



## Properties

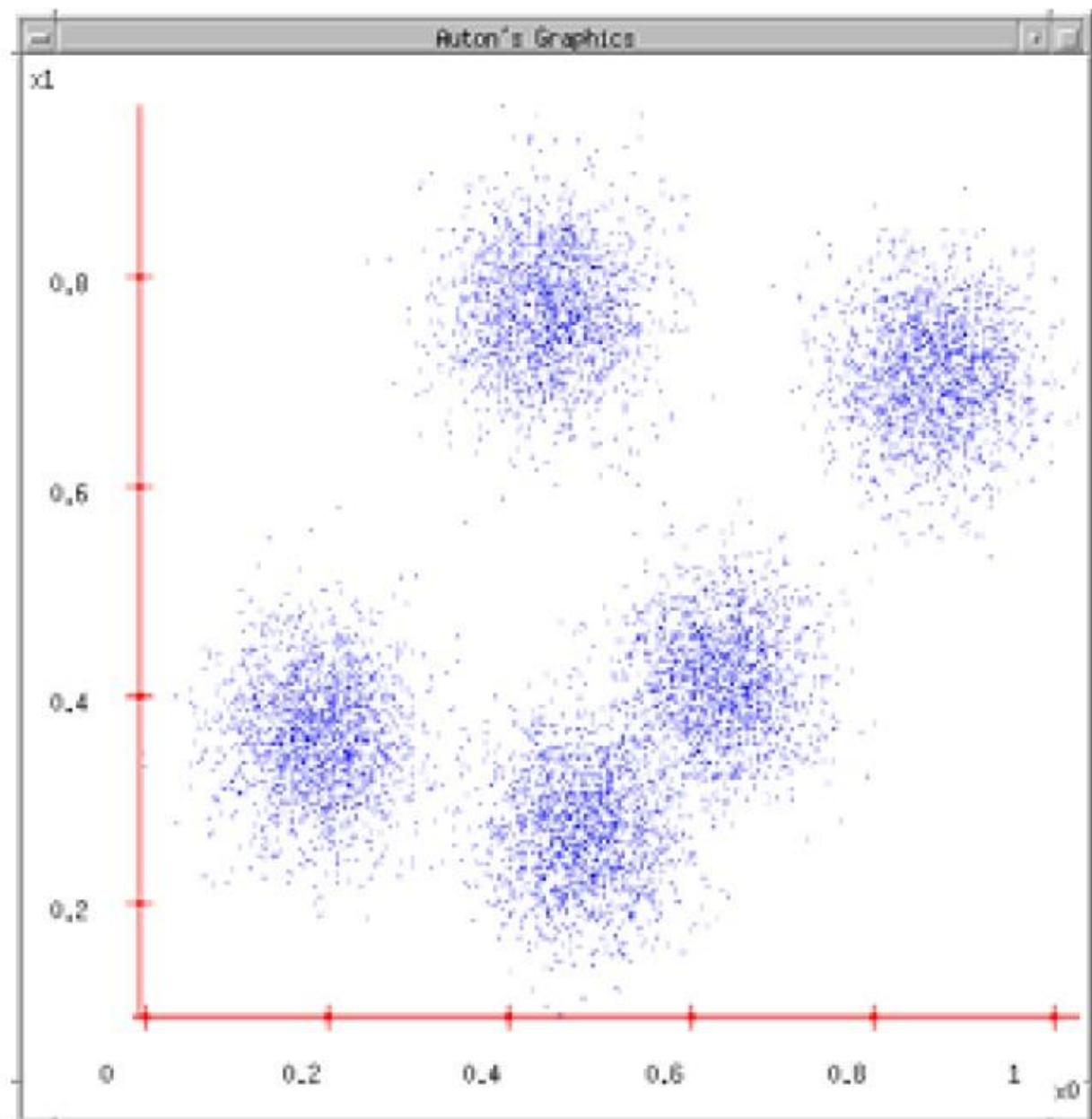
- Will always converge to some solution
- Can be a “local minimum” of objective:

$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \|p - c_i\|^2$$



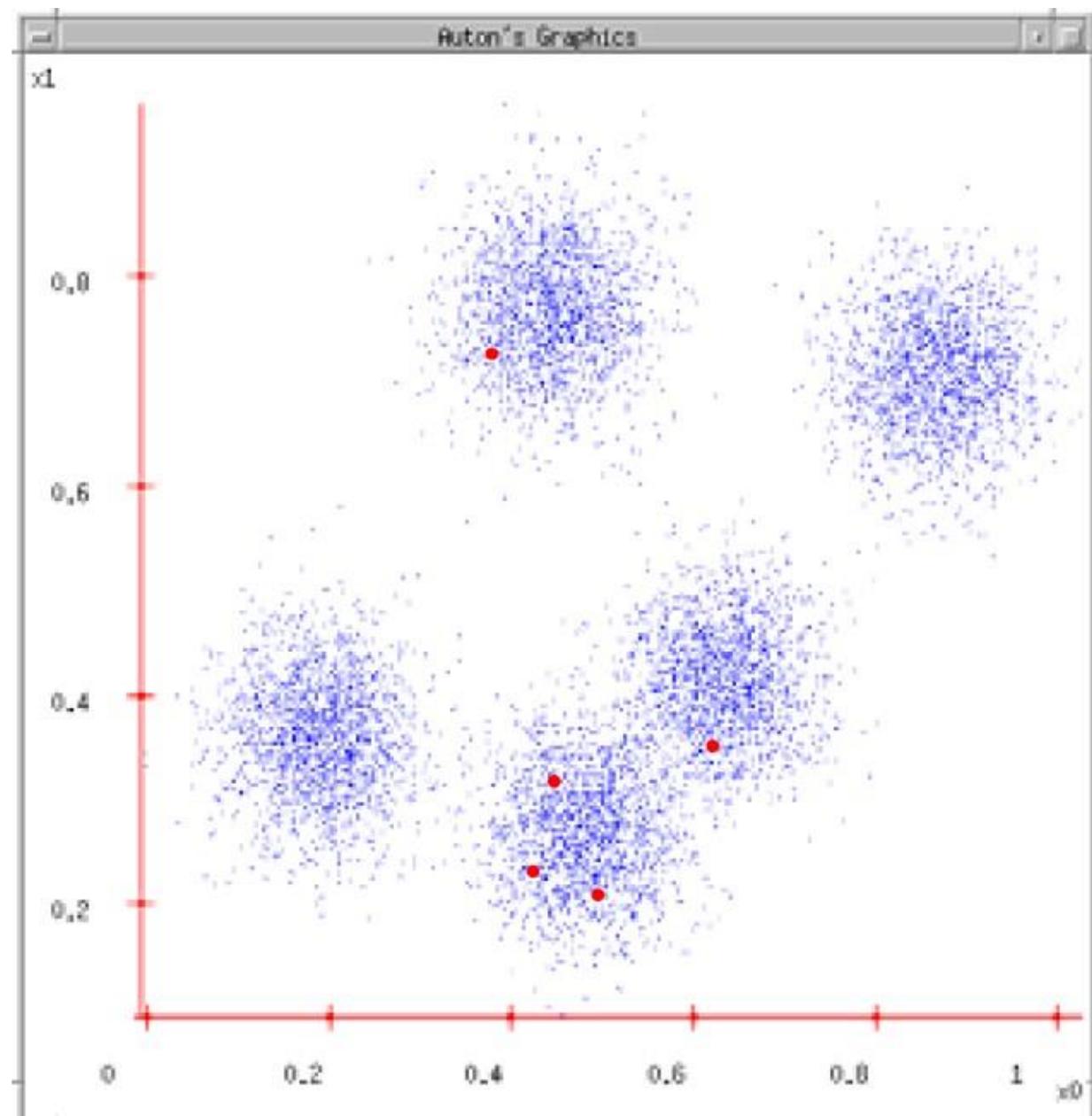
# K-means

1. Ask user how many clusters they'd like.  
*(e.g. k=5)*



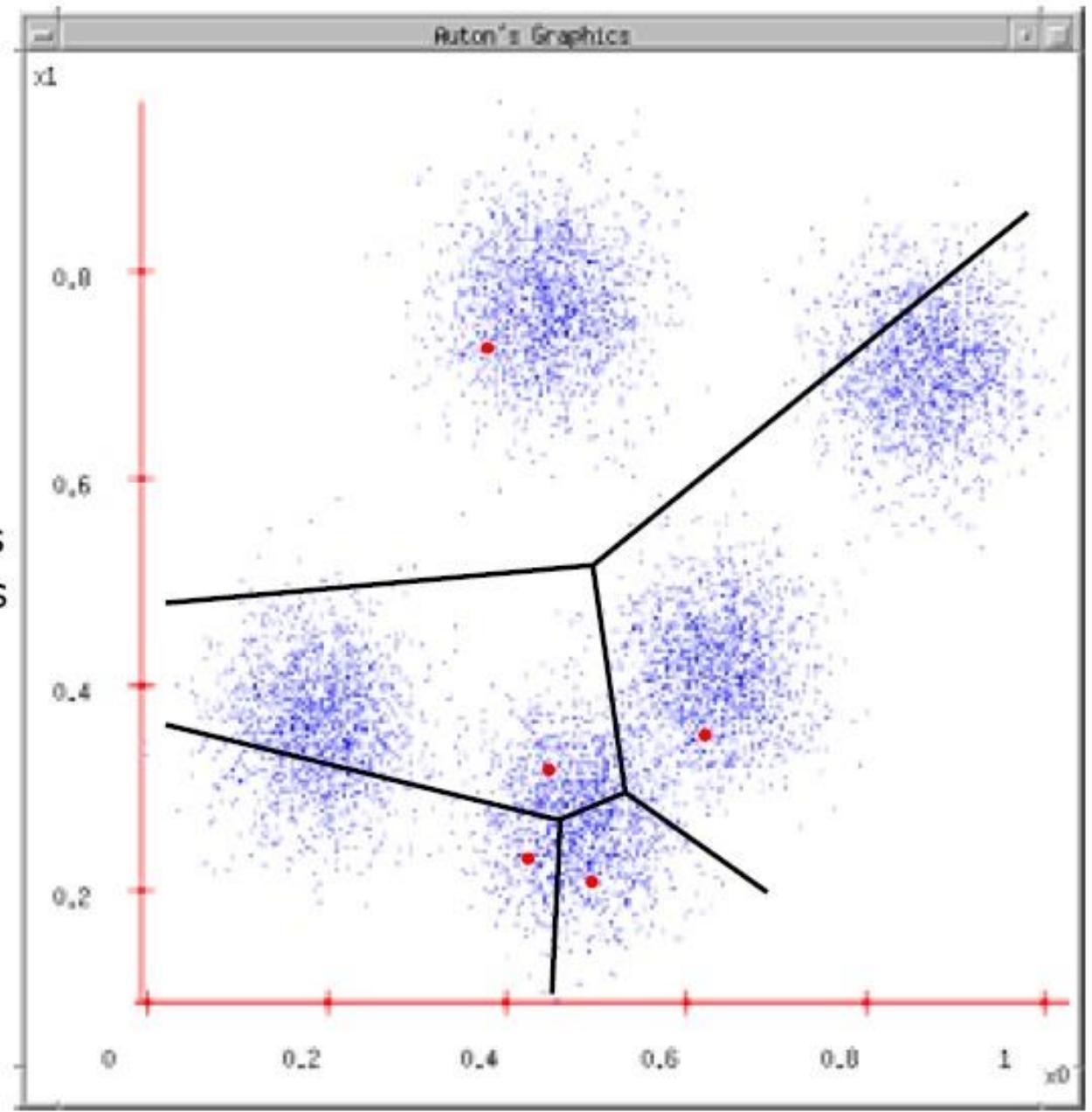
# K-means

1. Ask user how many clusters they'd like.  
*(e.g. k=5)*
2. Randomly guess k cluster Center locations



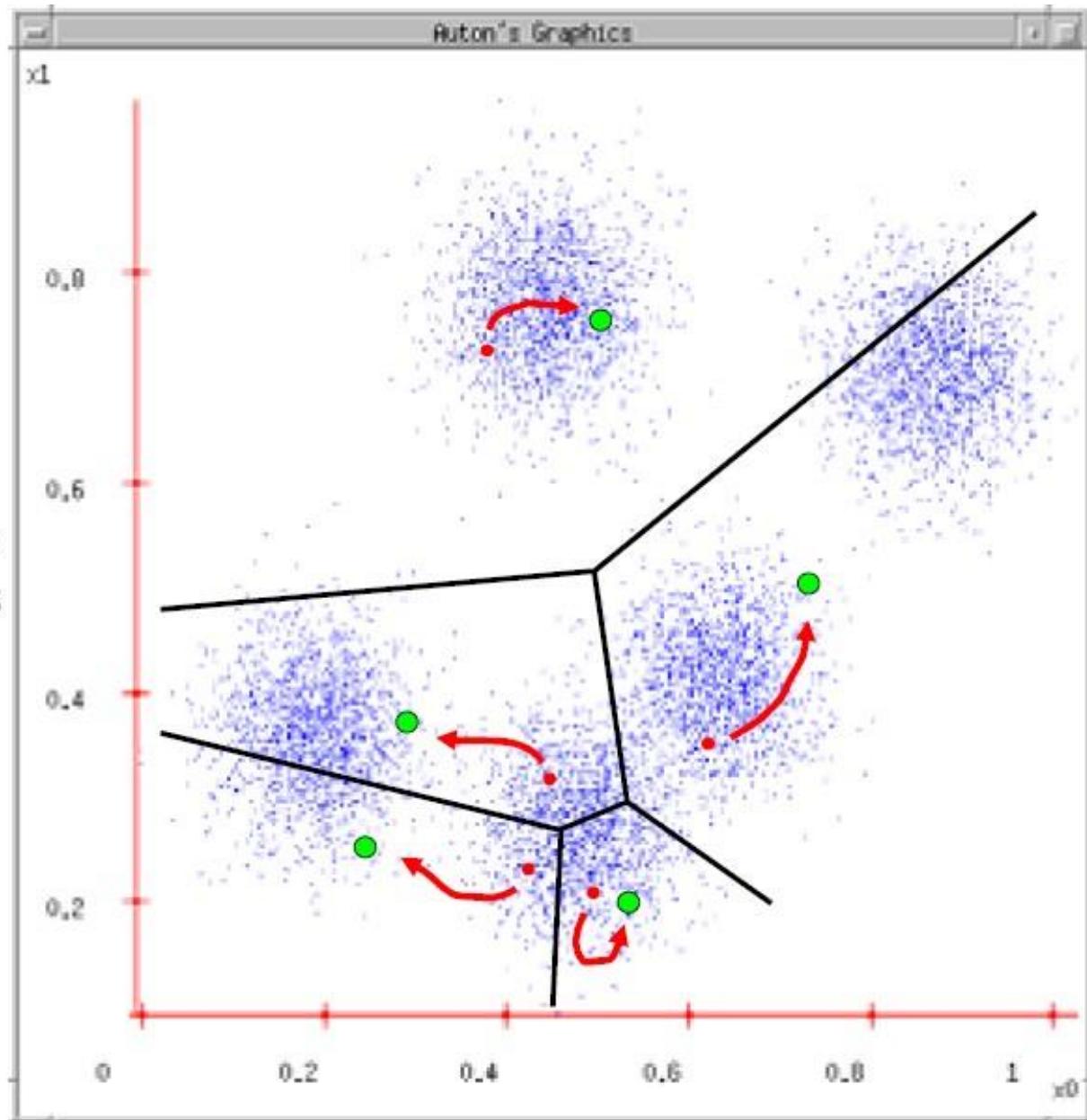
# K-means

1. Ask user how many clusters they'd like.  
*(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



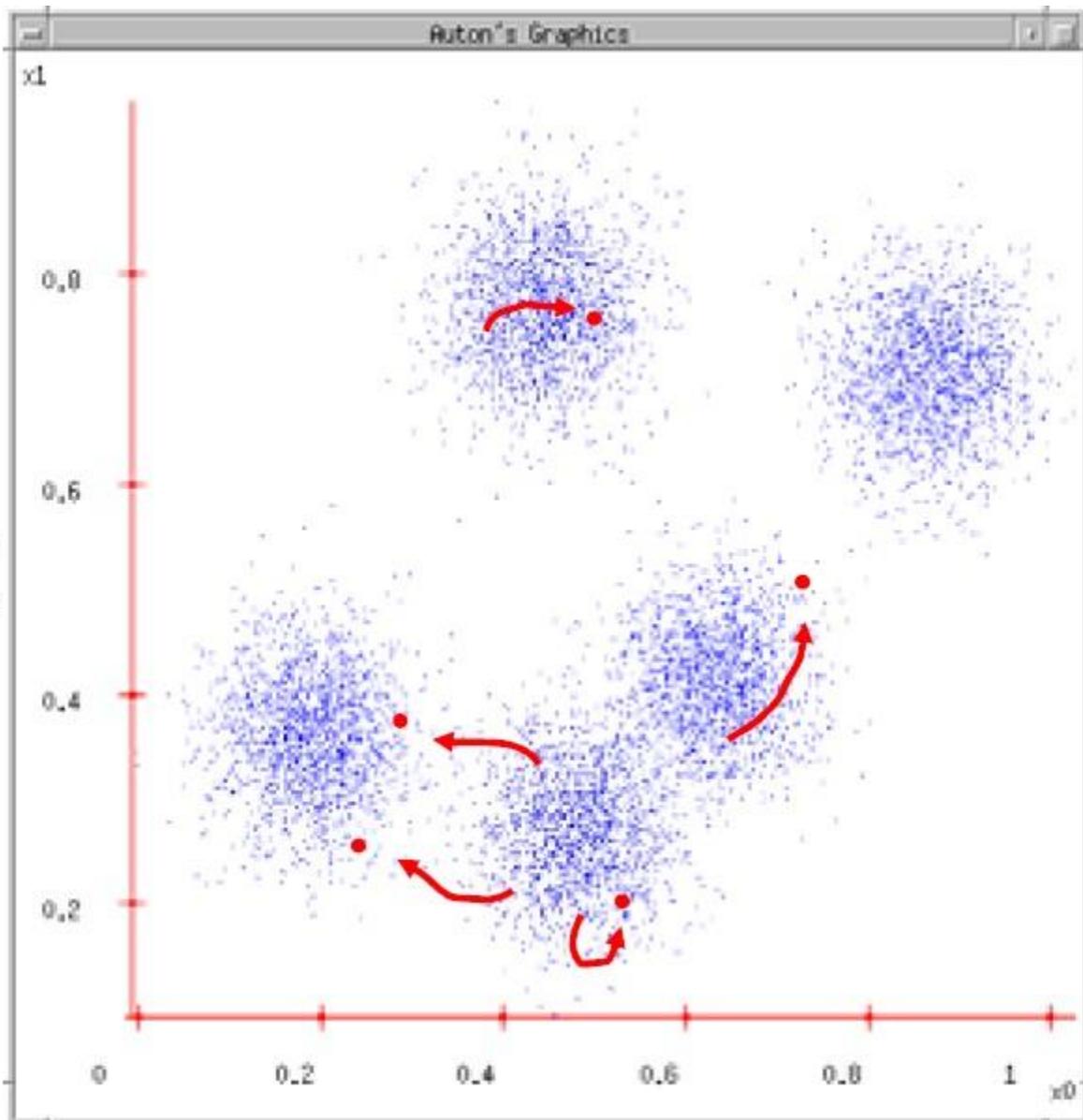
# K-means

1. Ask user how many clusters they'd like.  
*(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



# K-means

1. Ask user how many clusters they'd like.  
*(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns...
5. ...and jumps there
6. ...Repeat until terminated!



# K-means clustering

- Visualization

<https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>

- Java demo

[http://home.dei.polimi.it/matteucc/Clustering/tutorial\\_html/AppletKM.html](http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html)

- Matlab demo

[http://www.cs.pitt.edu/~kovashka/cs1699\\_fa15/kmeans\\_demo.m](http://www.cs.pitt.edu/~kovashka/cs1699_fa15/kmeans_demo.m)

# Time Complexity

- Let  $n$ = number of instances,  $d$ = dimensionality of the features,  $k$ = number of clusters
- Assume computing distance between two instances is  $O(d)$
- Reassigning clusters:
  - $O(kn)$  distance computations, or  $O(knd)$
- Computing centroids:
  - Each instance vector gets added once to a centroid:  $O(nd)$
- Assume these two steps are each done once for a fixed number of iterations  $I$ :  $O(Iknd)$ 
  - Linear in all relevant factors

# Another way of writing objective

- **K-means:** Let  $r_{nk} = 1$  if instance  $n$  belongs to cluster  $k$ , 0 otherwise

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}}$$

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- k-means the centre of a cluster is not necessarily one of the input data points (it is the average between the points in the cluster).

- **K-medoids (more general distances):**

$$\tilde{J} = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \mathcal{V}(\mathbf{x}_n, \boldsymbol{\mu}_k)$$

- k-medoids chooses data points as centers (medoids or exemplars) and can be used with arbitrary distances
- k-medoids more robust to noise and outliers as compared to [k-means](#) because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

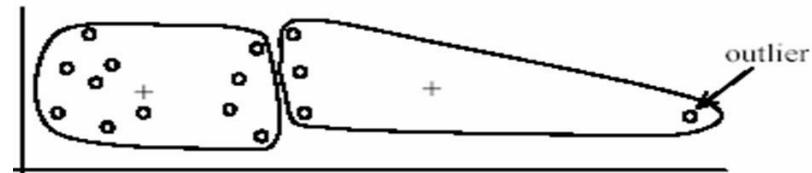
# K-means: pros and cons

## Pros

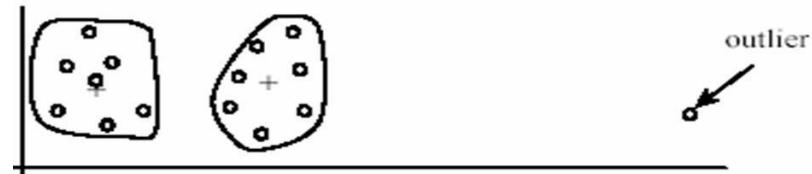
- Simple, fast to compute
- Converges to local minimum of within-cluster squared error

## Cons/issues

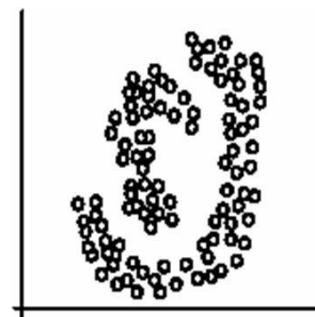
- Setting k?
- Sensitive to initial centers
  - Use heuristics or output of another method
- Sensitive to outliers
- Detects spherical clusters



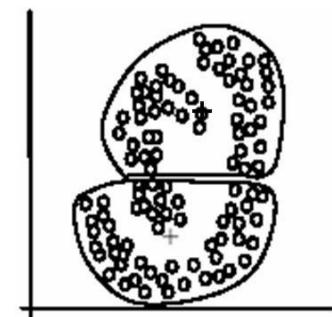
(A): Undesirable clusters



(B): Ideal clusters



(A): Two natural clusters

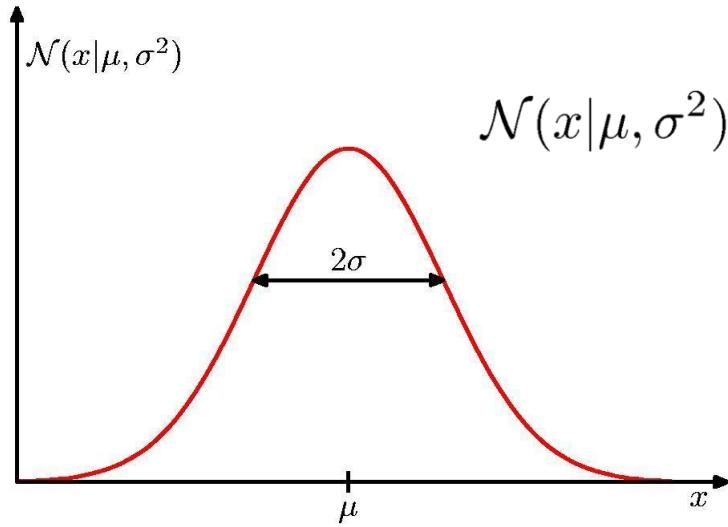


(B):  $k$ -means clusters

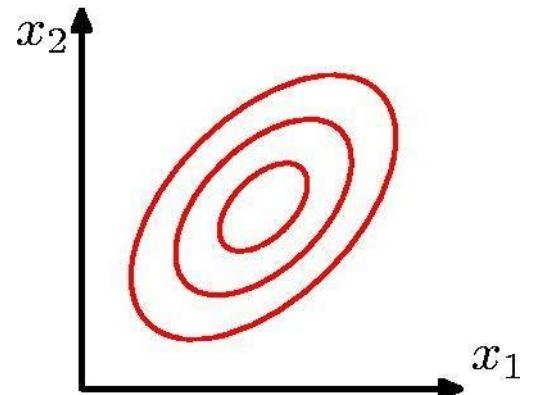
# Probabilistic Clustering

- Represent the probability distribution of the data as a *mixture model*
  - captures uncertainty in cluster assignments
  - gives model for data distribution
  - Bayesian* mixture model allows us to determine  $K$
- Consider mixtures of *Gaussians*

# Review: Gaussian Distribution



$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp \left\{ -\frac{1}{2\sigma^2}(x - \mu)^2 \right\}$$



$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

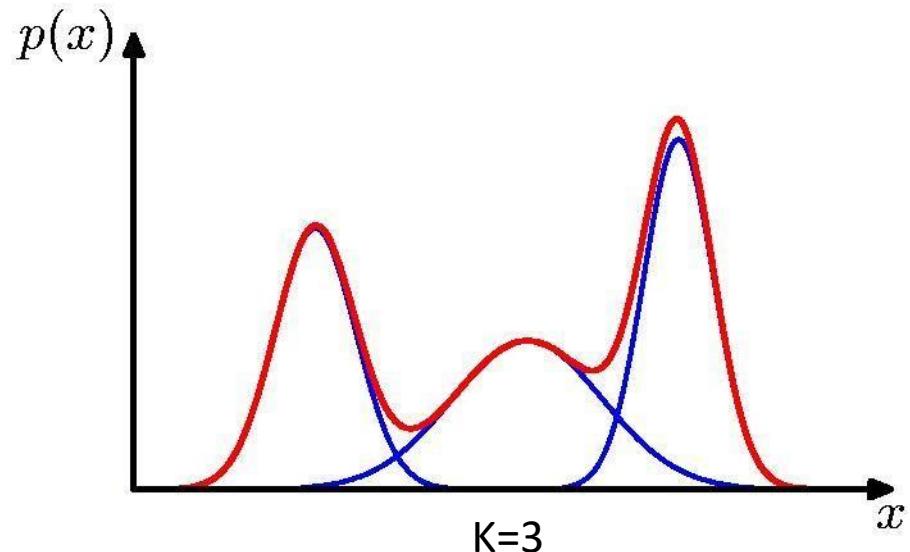
# Mixtures of Gaussians

- Combine simple models into a complex model:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

↑  
Component  
Mixing coefficient

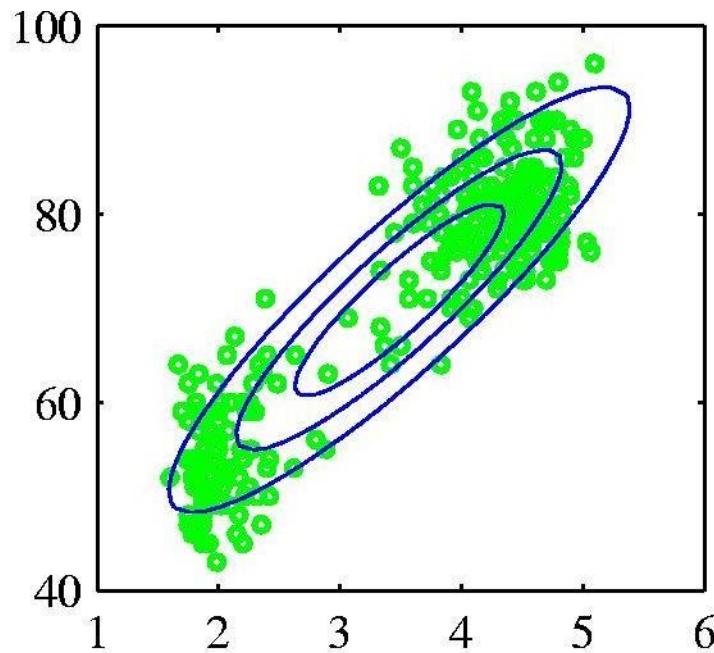
$$\forall k : \pi_k \geq 0 \quad \sum_{k=1}^K \pi_k = 1$$



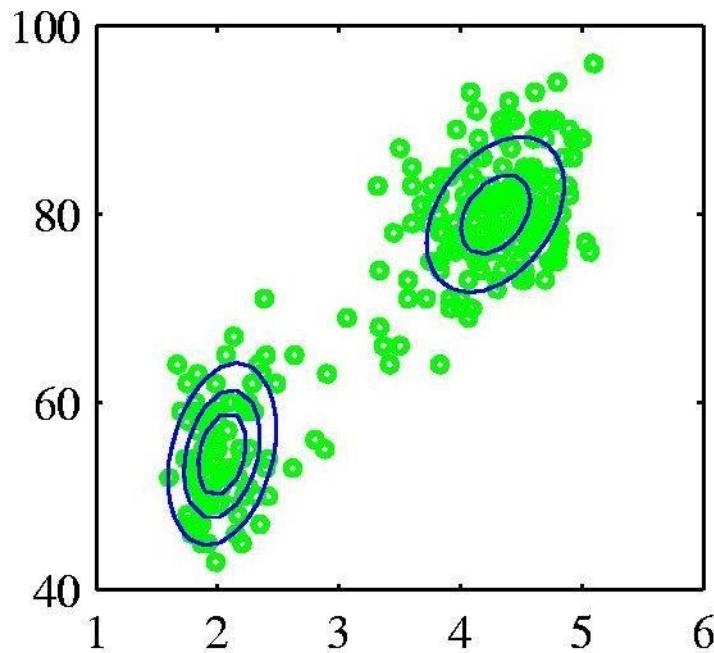
- Find parameters through EM (Expectation Maximization) algorithm

# Probabilistic version: Mixtures of Gaussians

- Old Faithful data set



Single Gaussian

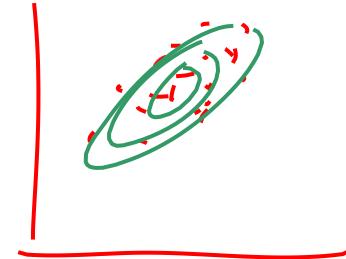


Mixture of two  
Gaussians

# Gaussian Mixture Model

- Data set

$$D = \{\mathbf{x}_n\} \quad n = 1, \dots, N$$



- Consider first a single Gaussian
- Assume observed data points generated independently

$$p(D|\boldsymbol{\mu}, \Sigma) = \prod_{n=1}^N \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}, \Sigma)$$

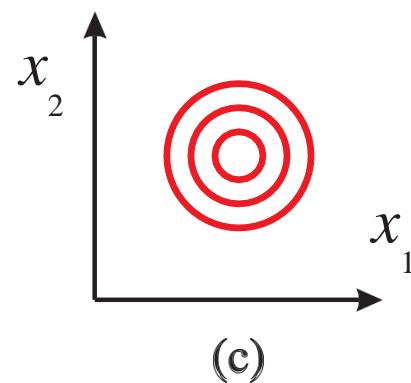
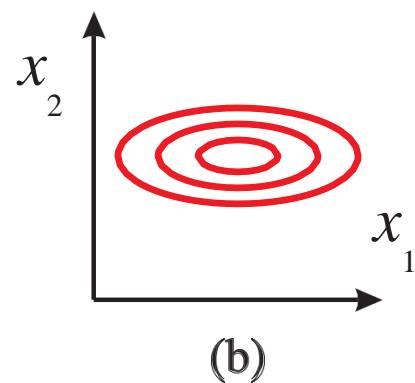
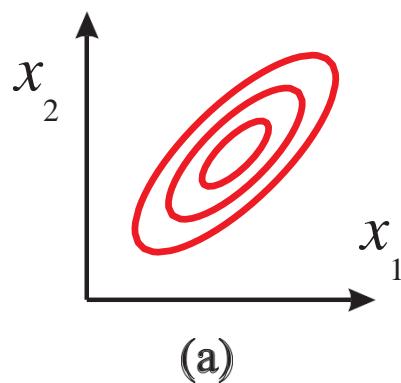
- Viewed as a function of the parameters, this is known as the *likelihood function*

# The Gaussian Distribution

- Multivariate Gaussian

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

**mean**                   **covariance**



# Gaussian Mixture Model

---

- K-dimensional binary random variable  $z$  having a 1-of-K representation in which a particular element  $z_k$  is equal to 1 and all other elements are equal to 0.
- The values of  $z_k$  therefore satisfy  $z_k \in \{0,1\}$
- K possible states for the vector  $z$  according to which element is nonzero.
- Joint distribution  $p(x,z)$  in terms of a marginal distribution  $p(z)$  and a conditional distribution  $p(x|z)$ ,
- Marginal distribution over  $z$  is specified in terms of the mixing coefficients  $\pi_k$ , such that  $p(z_k = 1) = \pi_k$

# Sampling from the Gaussian

- To generate a data point:
  - first pick one of the components with probability  $\pi_k$
  - then draw a sample  $\mathbf{x}_n$  from that component
- Repeat these two steps for each new data point

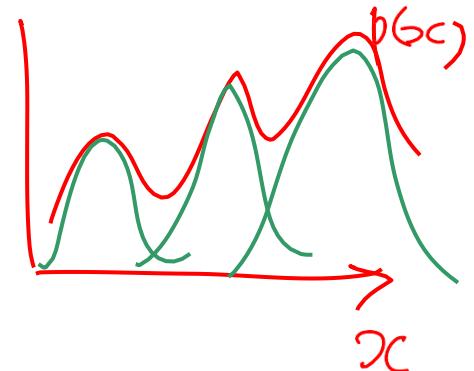
# Fitting the Gaussian Mixture

- We wish to invert this process – given the data set, find the corresponding parameters:
  - mixing coefficients
  - means
  - covariances
- If we knew which component generated each data point, the maximum likelihood solution would involve fitting each component to the corresponding cluster
- Problem: the data set is unlabelled
- We shall refer to the labels as *latent* (= hidden) variables

# Gaussian Mixture Model

- Linear super-position of Gaussians

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$



- Normalization and positivity require

$$\sum_{k=1}^K \pi_k = 1 \quad 0 \leq \pi_k \leq 1$$

- Can interpret the mixing coefficients as prior probabilities

$$p(\mathbf{x}) = \sum_{k=1}^K p(k)p(\mathbf{x}|k)$$

# Gaussian Mixture Model

- $\mathbf{z}$  uses a 1-of- $K$  representation, we can also write this distribution in the form

$$p(\mathbf{z}) = \prod_{k=1}^K \pi_k^{z_k}$$

- Conditional distribution of  $\mathbf{x}$  given a particular value for  $\mathbf{z}$  is a Gaussian

$$p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^K \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

- Joint distribution is given by  $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$ , and the marginal distribution of  $\mathbf{x}$  is then obtained by summing the joint distribution over all possible states of  $\mathbf{z}$  to give

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

# Gaussian Mixture Model

---

- Conditional probability of  $z$  given  $x$
- use  $\gamma(z_k)$  to denote  $p(z_k = 1 | x)$ , whose value can be found using Bayes' theorem

$$\begin{aligned}\gamma(z_k) \equiv p(z_k = 1 | x) &= \frac{p(z_k = 1)p(x|z_k = 1)}{\sum_{j=1}^K p(z_j = 1)p(x|z_j = 1)} \\ &= \frac{\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)}.\end{aligned}$$

- $\pi_k$  as the prior probability of  $z_k = 1$ , and the quantity  $\gamma(z_k)$  as the corresponding posterior probability once we have observed  $x$ .
-

# Maximum Likelihood

---

Log of likelihood function:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

- Maximizing the log likelihood function for a Gaussian mixture model turns out to be a more complex problem than for the case of a single Gaussian.
- The difficulty arises from the presence of the summation over k that appears inside the logarithm, so that the logarithm function no longer acts directly on the Gaussian.

# GMM Problems and Solutions

- How to maximize the log likelihood
  - solved by expectation-maximization (EM) algorithm
- How to avoid singularities in the likelihood function
  - solved by a Bayesian treatment
- How to choose number  $K$  of components
  - also solved by a Bayesian treatment

# Expectation Maximization (EM) Algorithm

---

- Conditions for MLE: Setting the derivatives of  $\ln p(\mathbf{X}|\pi, \mu, \Sigma)$  with respect to the means  $\mu_k$  of the Gaussian components to zero, we obtain

$$0 = - \sum_{n=1}^N \underbrace{\frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}}_{\gamma(z_{nk})} \boldsymbol{\Sigma}_k (\mathbf{x}_n - \boldsymbol{\mu}_k)$$

rearranging we obtain:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n$$

where we have defined

$$N_k = \sum_{n=1}^N \gamma(z_{nk})$$

# Expectation Maximization (EM) Algorithm

---

- $\mu_k$  for the kth Gaussian component is obtained by taking a weighted mean of all of the points in the data set
- Weighting factor for data point  $x_n$  is given by the posterior probability  $\gamma(z_{nk})$  that component k was responsible for generating  $x_n$
- If we set the derivative of  $\ln p(X|\pi, \mu, \Sigma)$  with respect to  $\Sigma_k$  to 0, and follow a similar line of reasoning, making use of the result for the maximum likelihood solution for the covariance matrix of a single Gaussian

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$$

# Expectation Maximization (EM) Algorithm



- Maximize  $\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$  with respect to the mixing coefficients  $\pi_k$  with constraint  $\sum_{k=1}^K \pi_k = 1 \quad 0 \leq \pi_k \leq 1$
- Using a Lagrange multiplier and maximizing the following quantity

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \lambda \left( \sum_{k=1}^K \pi_k - 1 \right)$$

which gives

$$0 = \sum_{n=1}^N \frac{\mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda$$

- If we now multiply both sides by  $\pi_k$  and sum over  $k$ , we find  $\lambda = -N$ .
- Rearranging we obtain

$$\pi_k = \frac{N_k}{N}$$

# Expectation Maximization (EM) Algorithm

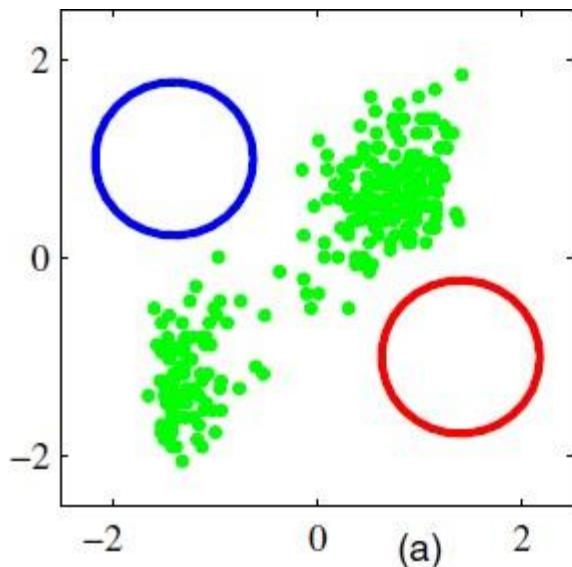


- We first choose some initial values for the means, covariances, and mixing coefficients.
- Then we alternate between the following two updates that we shall call the E step and the M step
- In the expectation step, or E step, we use the current values for the parameters to evaluate the posterior probabilities,
- We then use these probabilities in the maximization step, or M step, to re-estimate the means, covariances, and mixing
- In practice, the algorithm is deemed to have converged when the change in the log likelihood function, or alternatively in the parameters, falls below some threshold

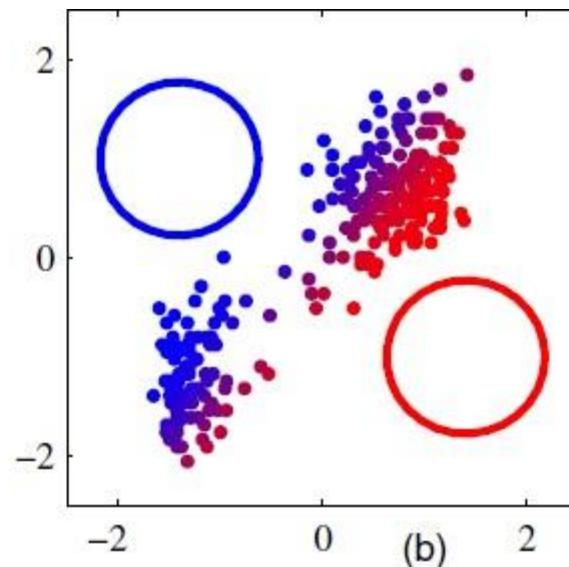
# EM Algorithm – Informal Derivation

- The solutions are not closed form since they are coupled
- Suggests an iterative scheme for solving them:
  - make initial guesses for the parameters
  - alternate between the following two stages:
    1. E-step: evaluate responsibilities
    2. M-step: update parameters using ML results
- Each EM cycle guaranteed not to decrease the likelihood

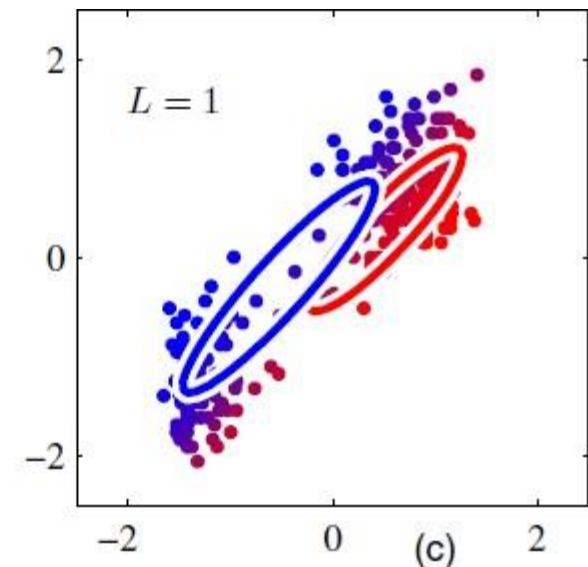
Initialization



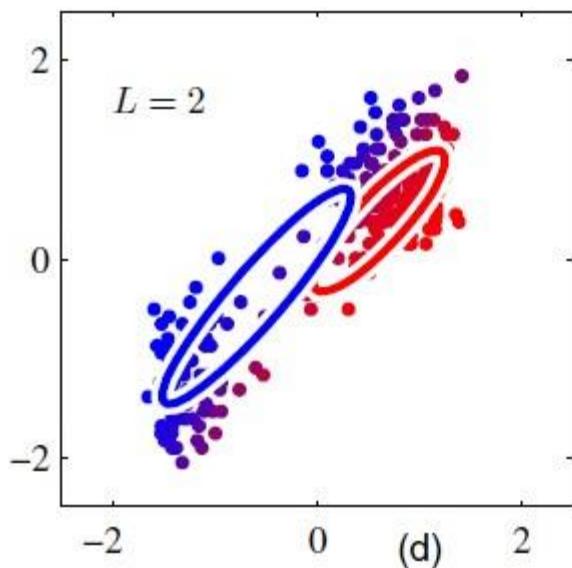
E step



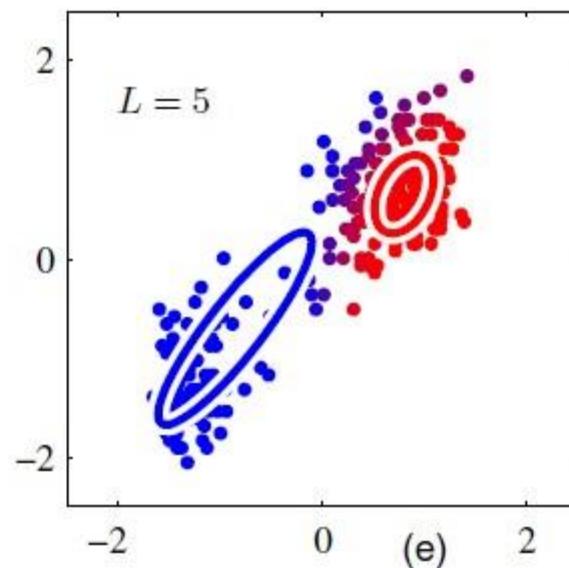
M step



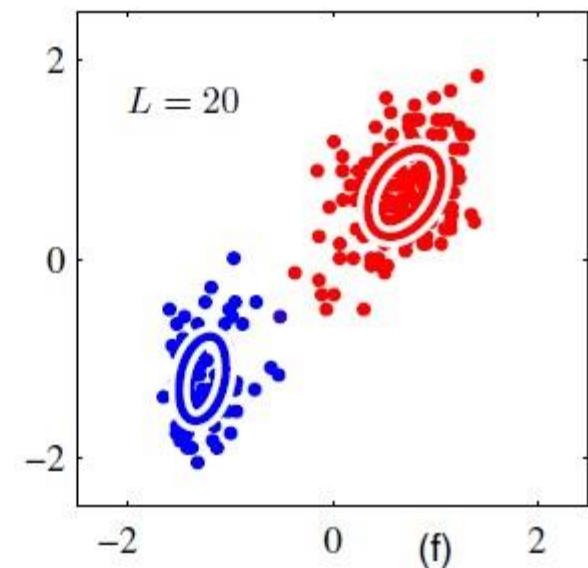
$L = 2$



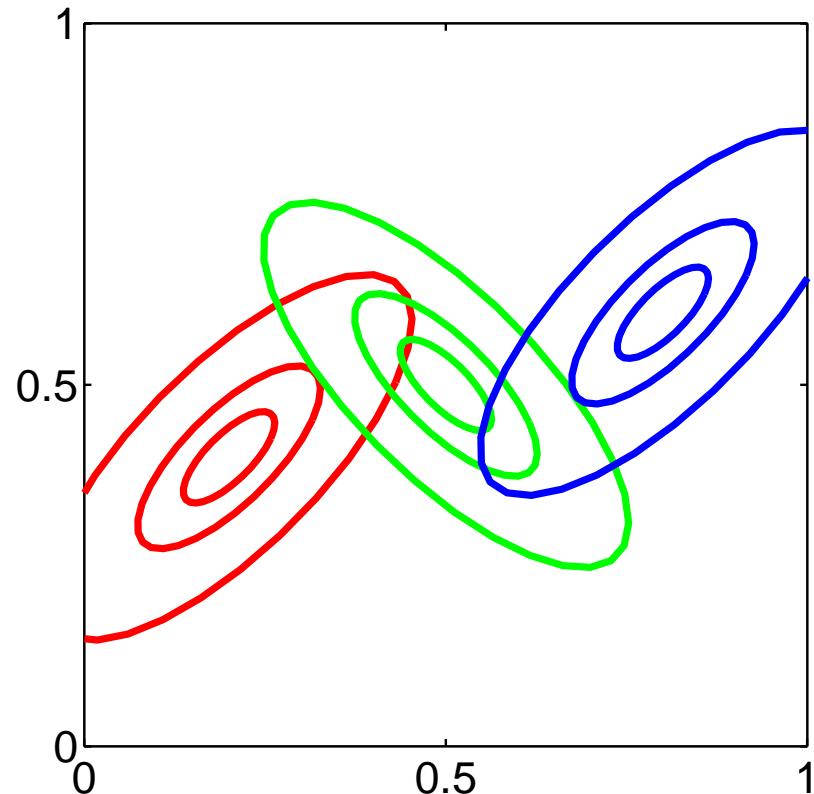
$L = 5$



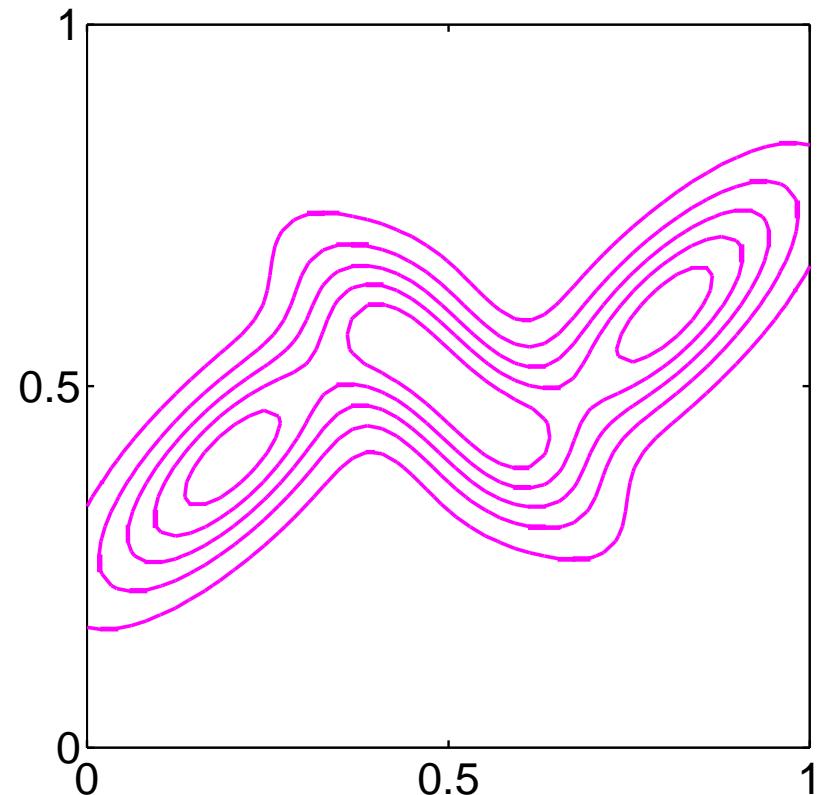
$L = 20$



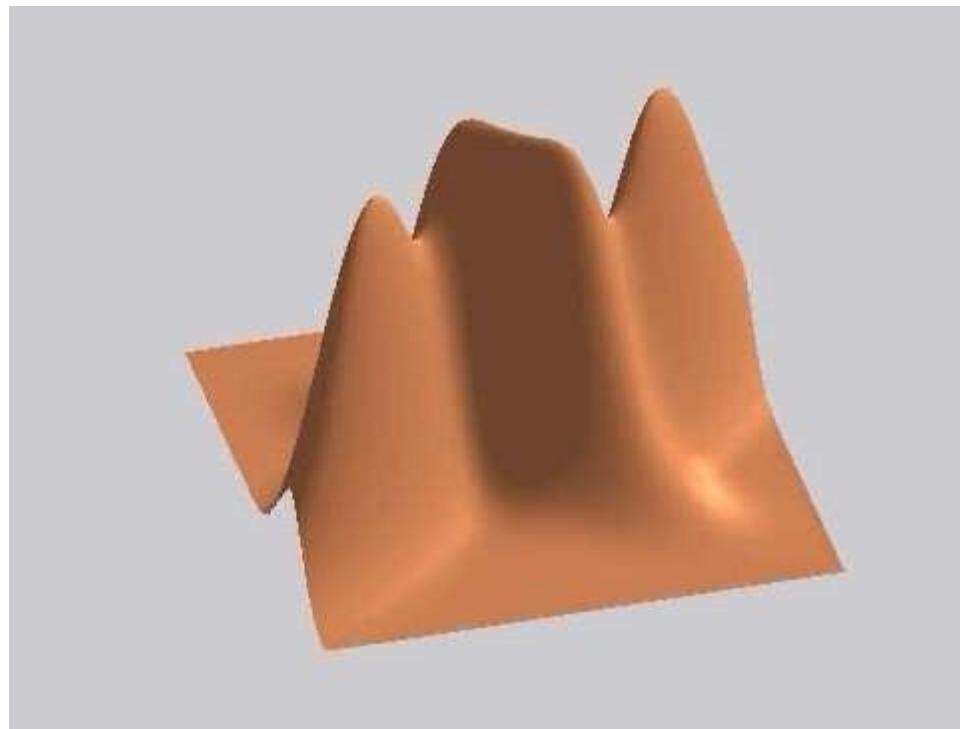
# Example: Mixture of 3 Gaussians



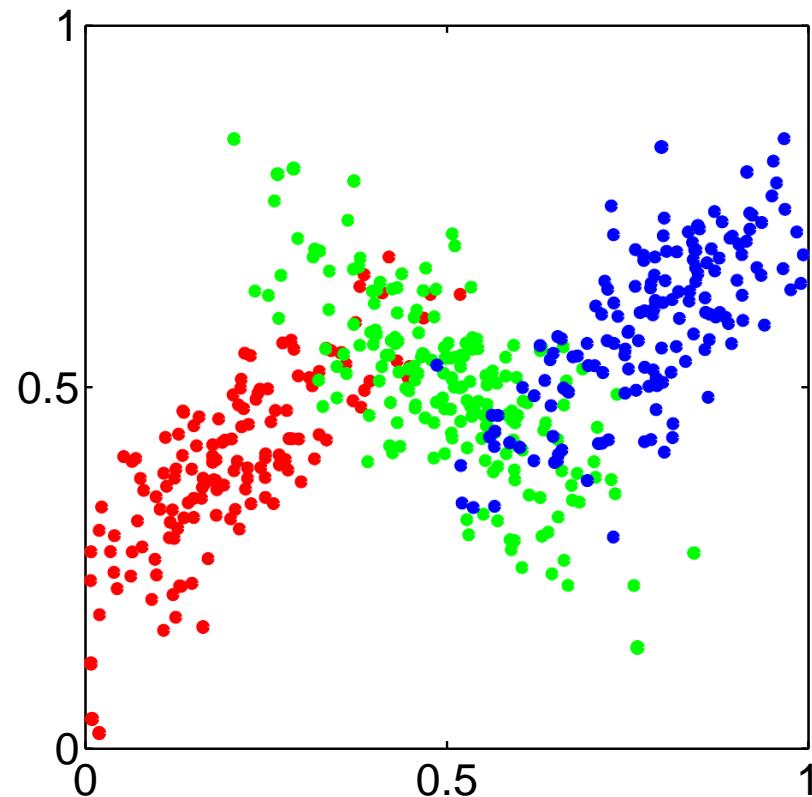
# Contours of Probability Distribution



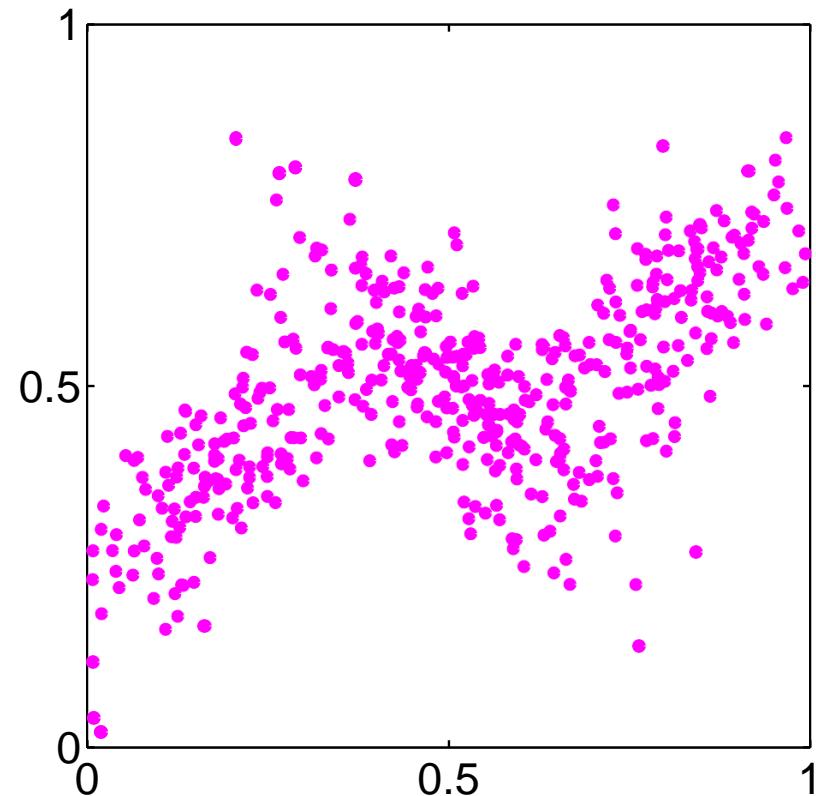
# Surface Plot



# Synthetic Data Set



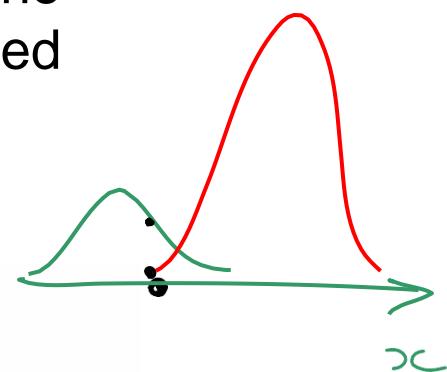
# Synthetic Data Set Without Labels



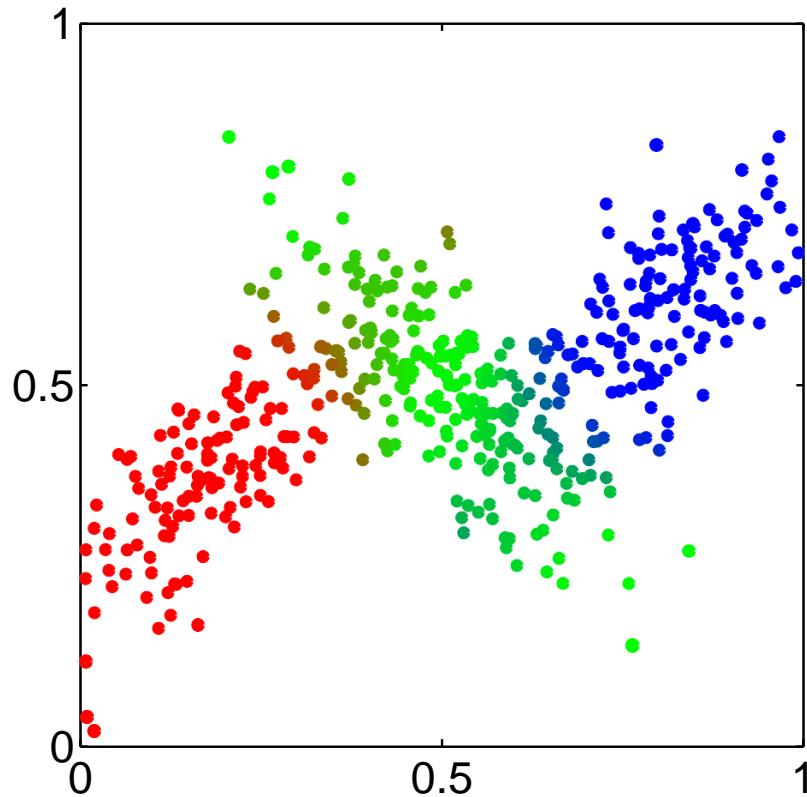
# Posterior Probabilities

- We can think of the mixing coefficients as prior probabilities for the components
- For a given value of  $\mathbf{x}$  we can evaluate the corresponding posterior probabilities, called *responsibilities*
- These are given from Bayes' theorem by

$$\begin{aligned}\gamma_k(\mathbf{x}) \equiv p(k|\mathbf{x}) &= \frac{p(k)p(\mathbf{x}|k)}{p(\mathbf{x})} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}\end{aligned}$$



# Posterior Probabilities (colour coded)



# EM algorithm for GMM

---

1. Initialize the means  $\mu_k$ , covariances  $\Sigma_k$  and mixing coefficients  $\pi_k$ , and evaluate the initial value of the log likelihood.
2. **E step:** Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

# EM algorithm for GMM

3. **M step:** Re-estimate the parameters using the current responsibilities

$$\begin{aligned}
 \boldsymbol{\mu}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n \\
 \boldsymbol{\Sigma}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T \\
 \pi_k^{\text{new}} &= \frac{N_k}{N} \quad \text{where } N_k = \sum_{n=1}^N \gamma(z_{nk})
 \end{aligned}$$

4. Evaluate the log likelihood

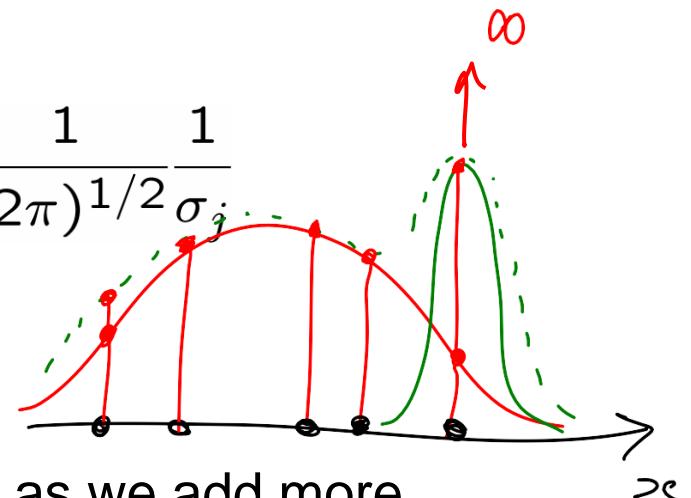
$$\ln p(\mathbf{X} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

# Over-fitting in Gaussian Mixture Models

- Singularities in likelihood function when a component ‘collapses’ onto a data point:

$$\mathcal{N}(\mathbf{x}_n | \mathbf{x}_n, \sigma_j^2 \mathbf{I}) = \frac{1}{(2\pi)^{1/2}} \frac{1}{\sigma_j}$$

then consider  $\sigma_j \rightarrow 0$



- Likelihood function gets larger as we add more components (and hence parameters) to the model
  - not clear how to choose the number  $K$  of components

# Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based  
on **intensity** similarity



Feature space: intensity value (1-d)

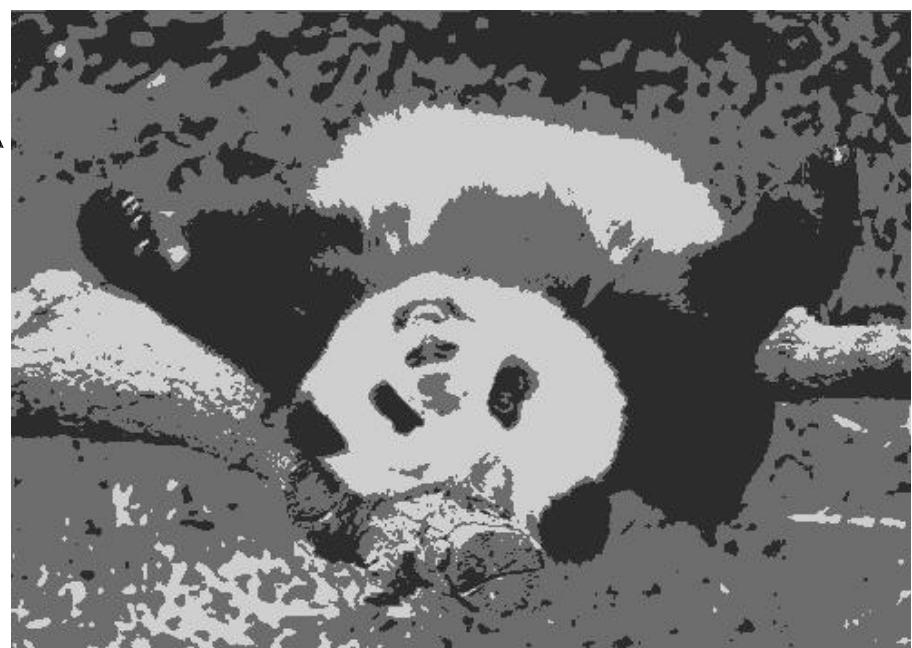


K=2



*quantization of the feature space;  
segmentation label map*

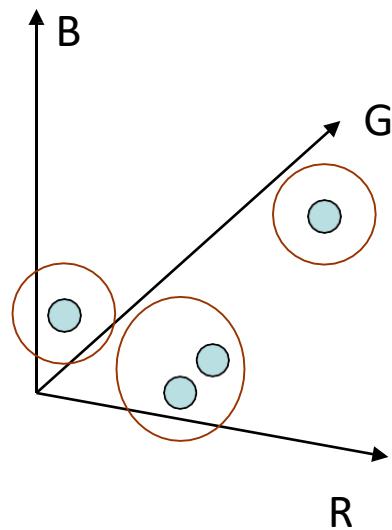
K=3



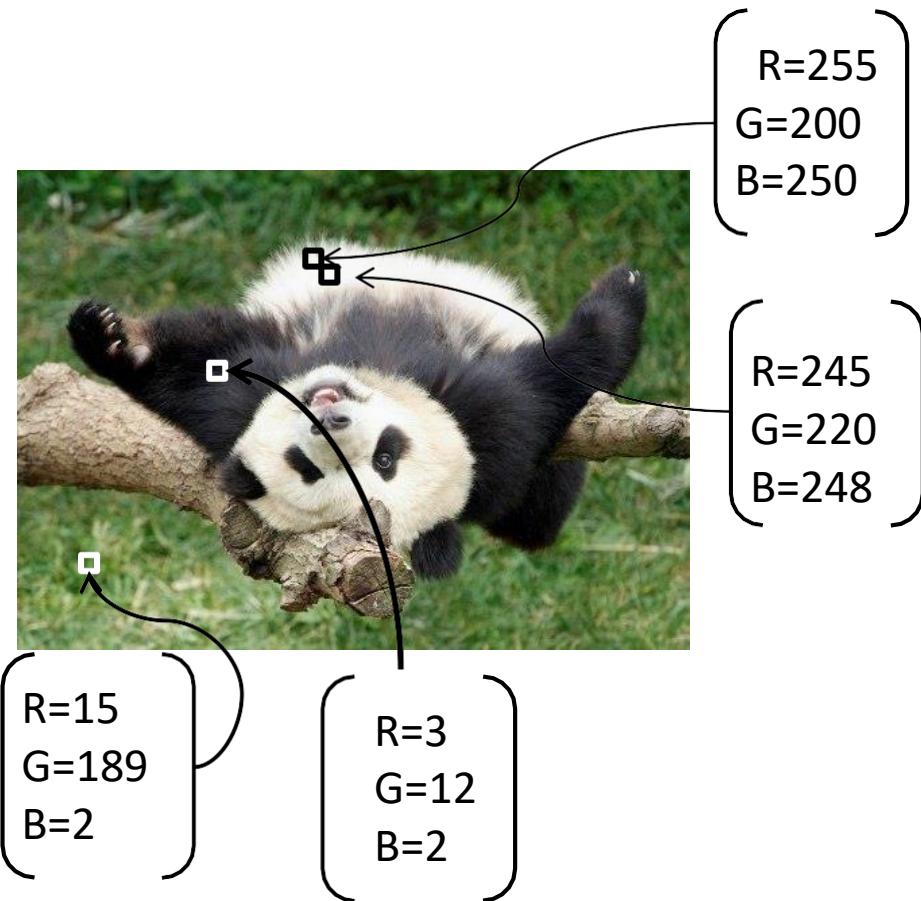
# Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **color** similarity



Feature space: color value (3-d)

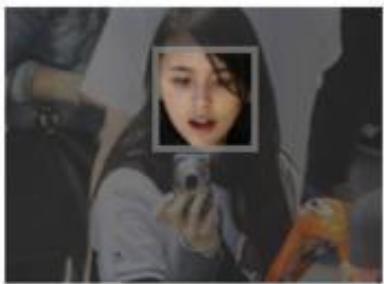


# Application: Face Recognition

facebook  3 Search Home

## Who's in These Photos?

The photos you uploaded were grouped automatically so you can quickly label and notify friends in these pictures. (Friends can always untag themselves.)



Who is this?



Who is this?



Who is this?



Who is this?



Who is this?



Who is this?

# References

---

Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition

<https://www.youtube.com/watch?v=TG6Bh-NFhA0>

<https://www.youtube.com/watch?v=qMTuMa86NzU>