



**VIT-AP**  
UNIVERSITY

**Course code:** CSE3008

**Course Title:** Introduction to Machine Learning

**Dr. Usha Rani Gogoi**  
**Assistant Professor Sr. Grade 1**

**[usha.gogoi@vitap.ac.in](mailto:usha.gogoi@vitap.ac.in)**

**SCOPE**

# Contents of Module 1: Introduction

- Learning problems
- perspectives and issues
- concept learning, version spaces and candidate eliminations, inductive bias
- decision tree learning
- Ensemble Learning and Random Forest

# INTRODUCTION

- We know humans learn from past experiences and machines follow instructions given by humans.
- **What if humans can train machines to learn from the past data?** This is what is called Machine Learning; though there is lot more than just learning, it is also about understanding and reasoning.
- The term **Machine Learning** was coined by **Arthur Samuel** in **1959**, an American pioneer in the field of computer gaming and artificial intelligence.
- **Definition:** *According to Arthur Samuel, Machine learning is all about giving computers the ability to learn without being explicitly programmed*".

# INTRODUCTION

- ML learns the data, builds a prediction model and when the new data comes in the model can predict about it.
- **More data>Better model>Higher accuracy**
- **Machine Learning is concerned with computer programs that automatically improve their performance through experience.**
- **Definition:** 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: Learning to Spam Filtering

# Example of Machine Learning

Learning to filter spam

**Spam** is all email the user does not want to receive and has not asked to receive.

**T:** Identify Spam Emails

**P:** % of spam emails that were filtered

% of non-spam emails that were incorrectly filtered-out

**E:** a database of emails that were labelled by users

# Why “Learn”

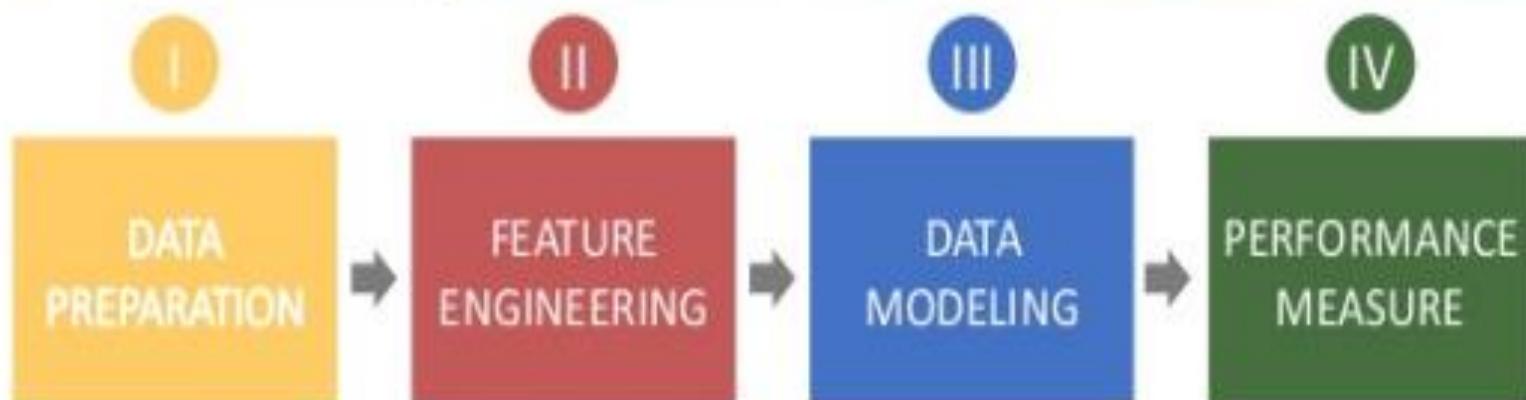
- Machine learning is programming computers to optimize a performance criterion using example data or past experience.
- Learning is used when:
  - Human expertise does not exist (navigating on Mars),
  - Humans are unable to explain their expertise (speech recognition)
  - Solution changes in time (routing on a computer network)
  - Solution needs to be adapted to particular cases (user biometrics)

# Why now?

- Especially with the advent of internet, there is a flood of available data
- Increasing computational power of computers
- Growing progress in available algorithms and theory developed by researchers
- Increasing support from industries.

# Steps of Machine Learning

There are 4 steps to build a machine learning model...



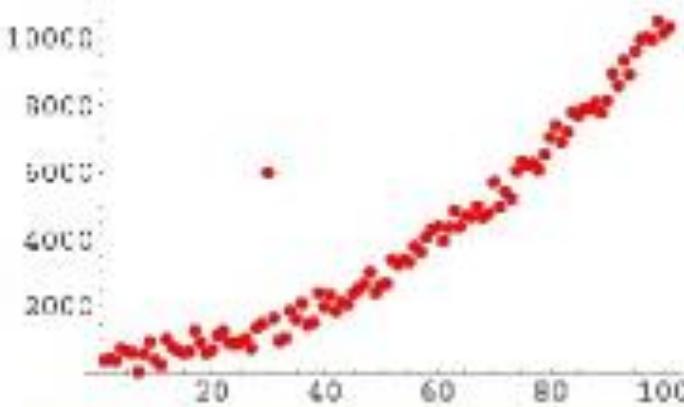
# Data Preparation

## Data Preprocessing

---

Is there anything **wrong** with the data?

- Missing values
- Outliers
- Wrongly-labeled examples
- Biased data
  - Do I have many more samples of one class than the rest?

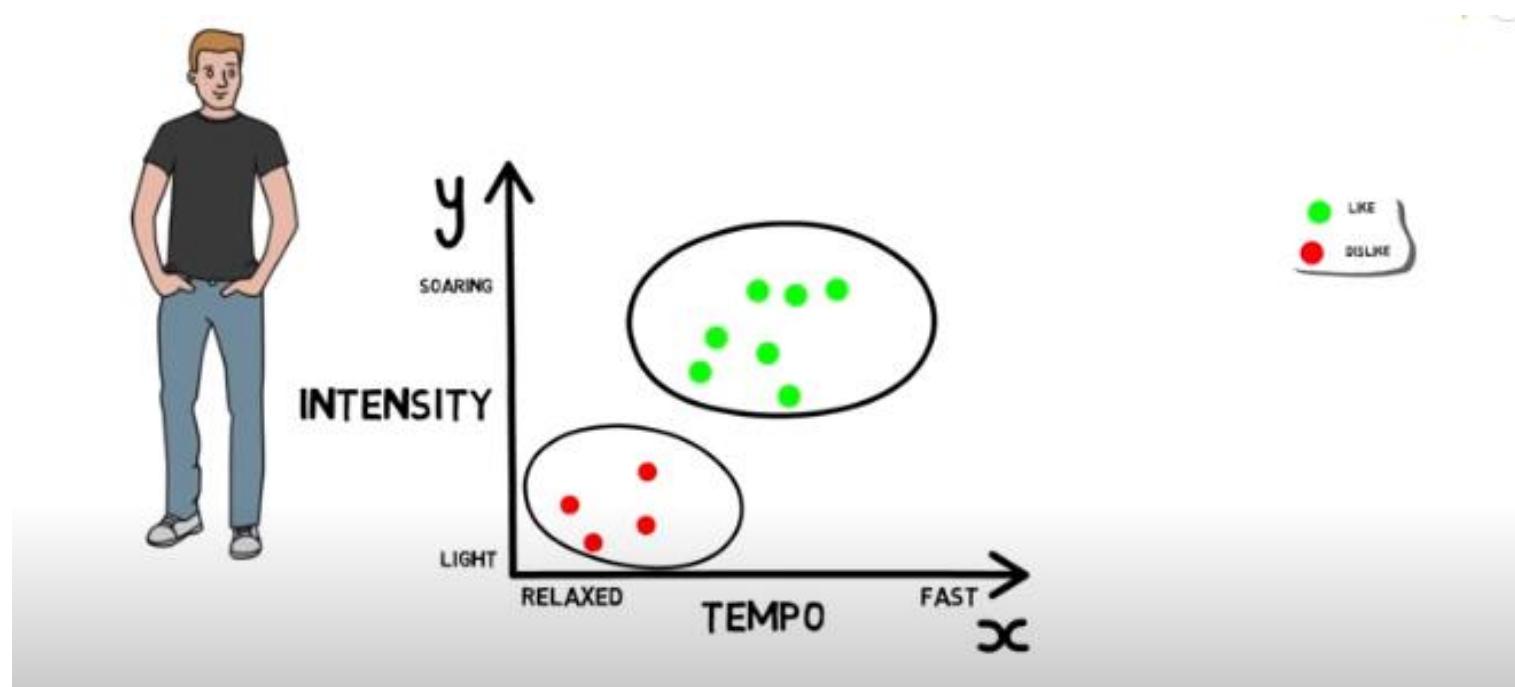


# Illustration

- Suppose Paul is listening to songs and there can be only two choices:
  - Paul likes the song.
  - Paul dislikes the song.
- Say, the decision for the choices are based on following features:
  - Tempo
  - Gender
  - Intensity
  - Gender of voice
- To simplify, amongst these features the most important ones say are Tempo and intensity

# Illustration(Training phase)

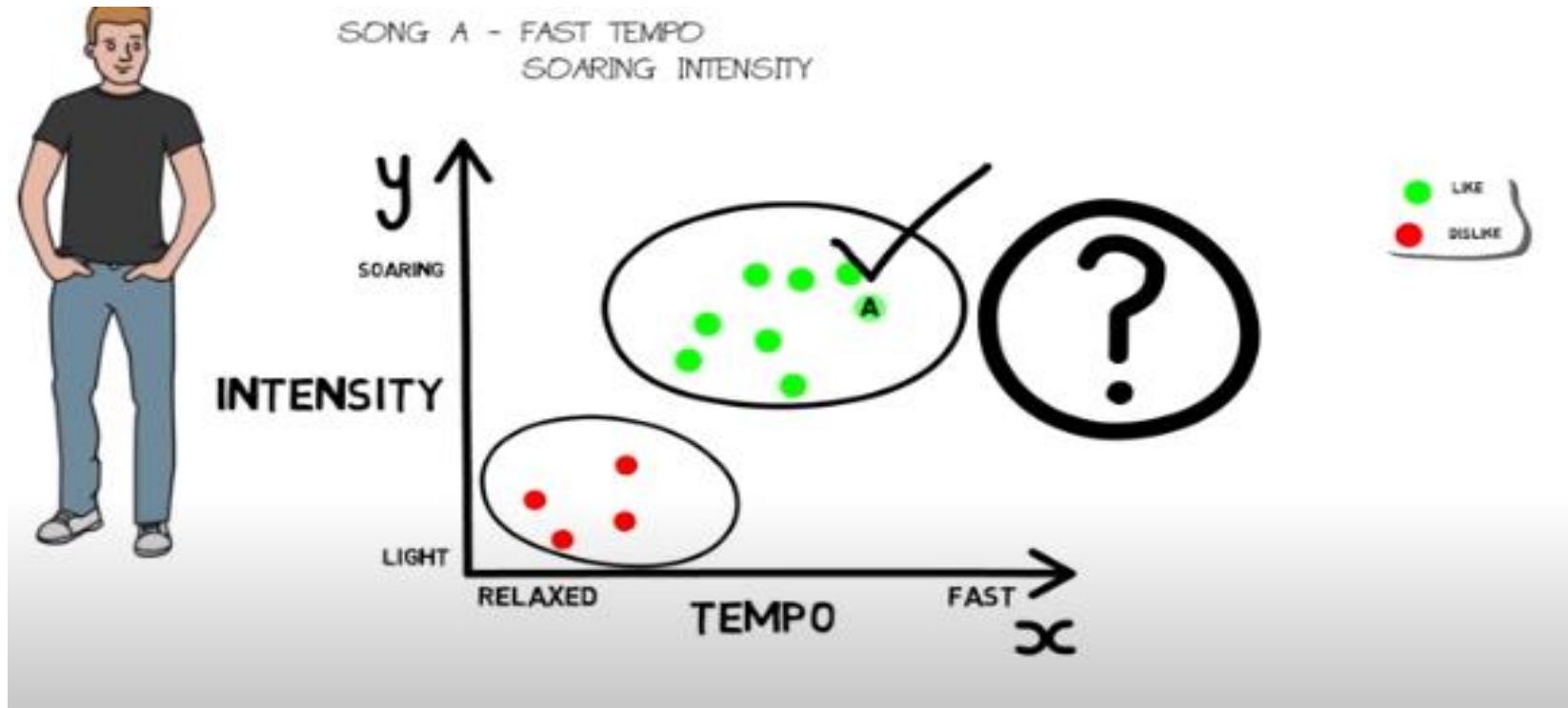
- Let us represent this entire ML problem statement in a 2-D solution space:
  - X-axis represents Temp level ranging from Relaxed to fast
  - Y axis represents Intensity level ranging from Light to soaring
  - As per figure, Paul likes high intensity and fast tempo songs, whereas he dislikes low intensity and relaxed tempo songs. (Assuming this information is obtained by training the model with training datasets)



## Illustration(Testing phase)

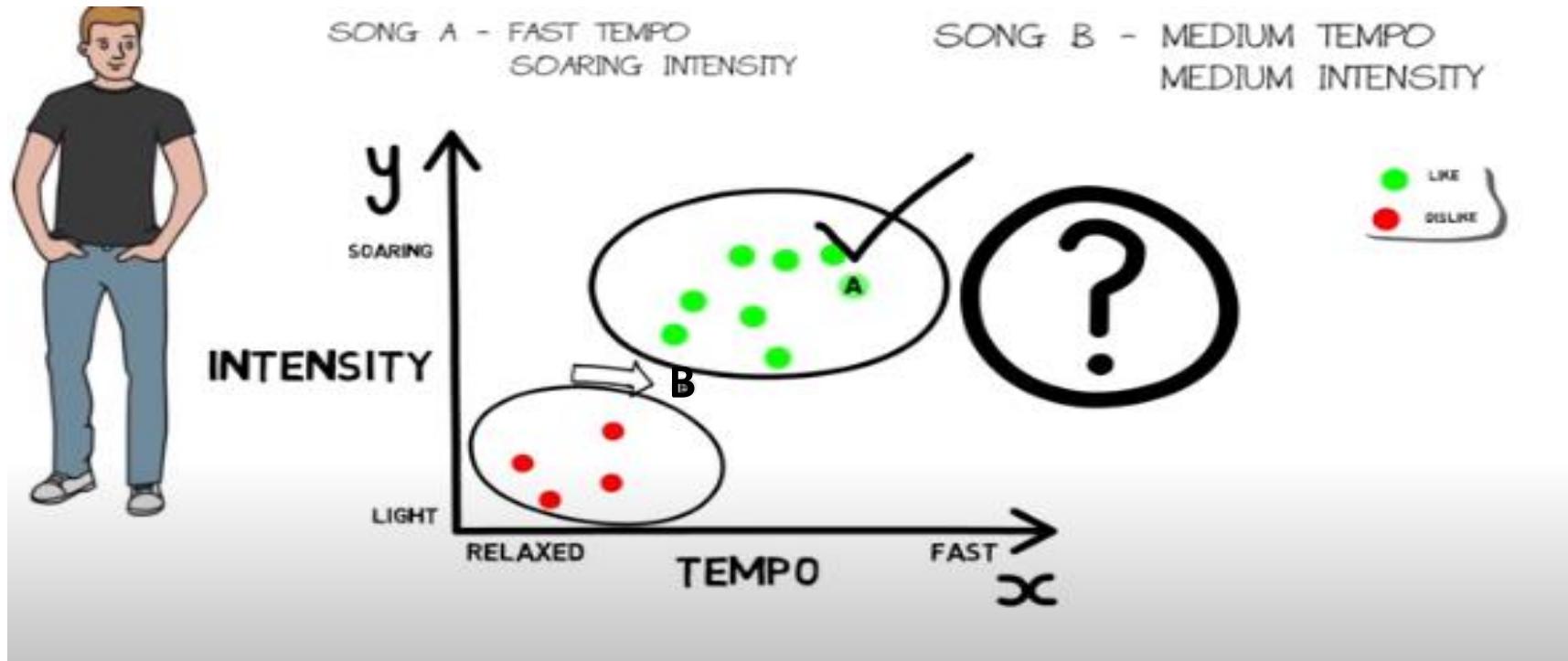
- Let us test the trained model with the testing data. Testing data say contains two songs, and we should decide whether Paul likes the song or dislikes it.
  - Song A: Fast tempo, Soaring intensity
  - Song B: Medium tempo, Medium intensity

# Does Paul like song A?



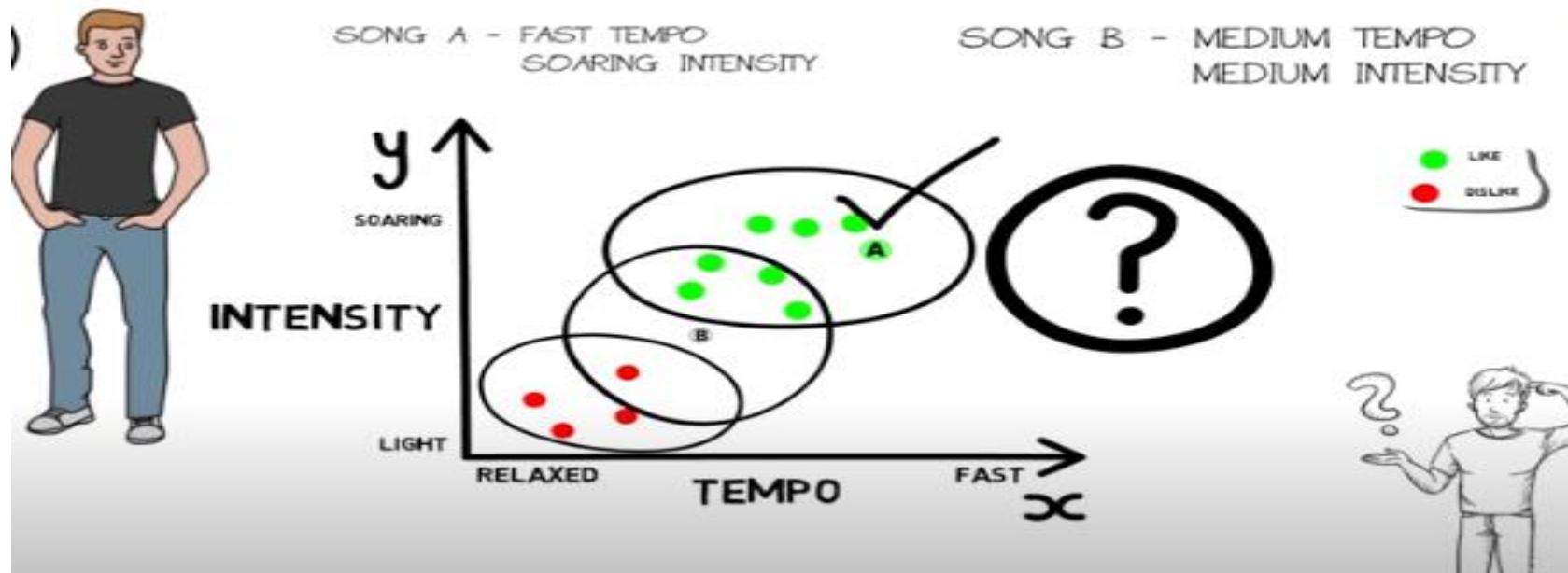
- Based on the trained model using past experience, we can say song A falls in the region as shown in figure.
- Does Paul like song A? → Yes Paul likes the song as it has Fast tempo and soaring intensity.
- Decision for song A was quite easy (best case scenario). But this is not the case always as we will see for Song B in the next slide.

# Does Paul like song B?



- Based on the trained model using past experience, we can say song B falls in the region as shown in figure.
- So this song does not directly belong to a particular region (called class in ML terms) as it is having intermediate characteristics.
- How to decide? → For this we will have to use various algorithms present in Machine learning.

# Does Paul like song B? Continued....



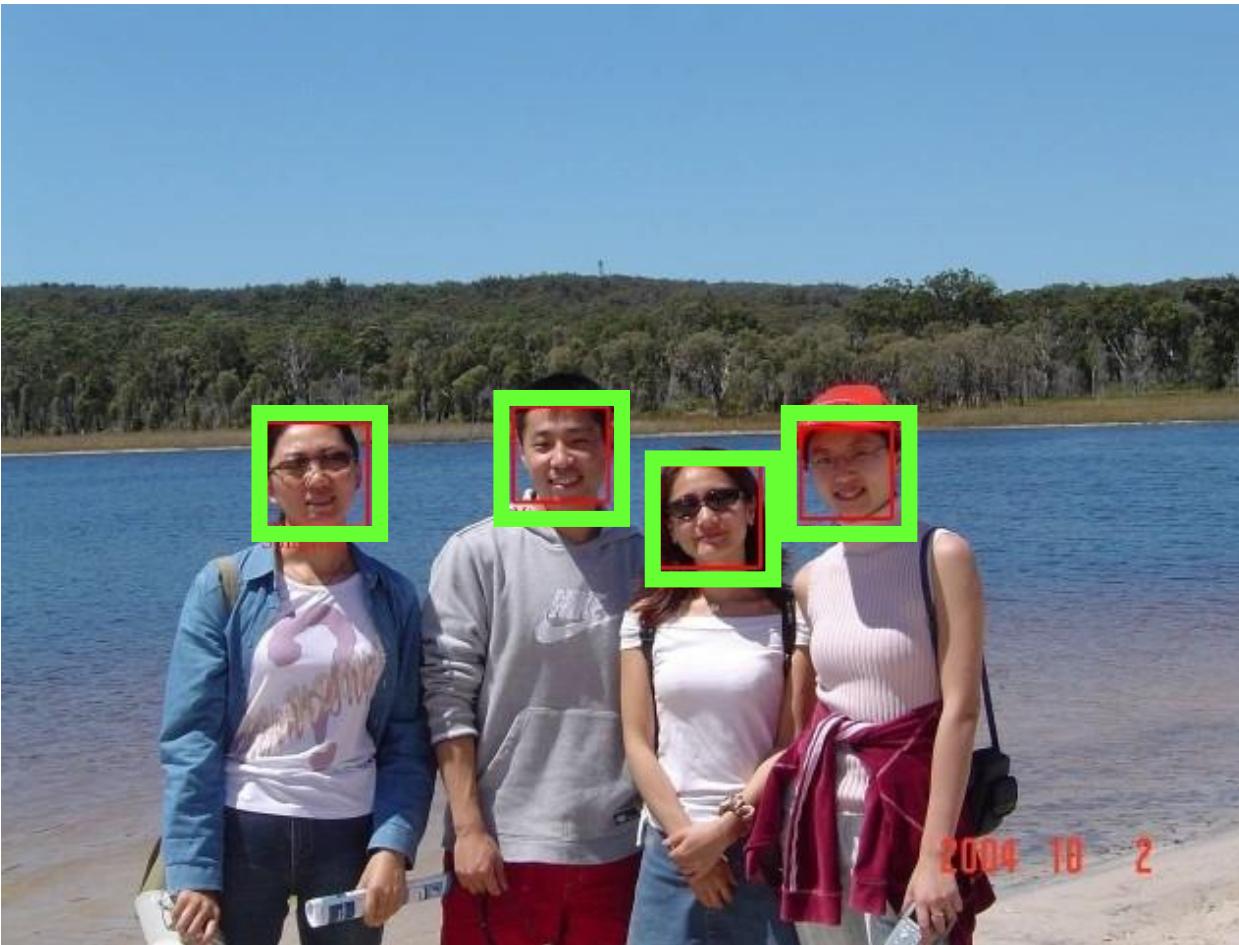
- To decide about Song B we will make use of majority vote concept
- If we draw a circle taking point B as center, we can observe:
  - There are 4 data points supporting Paul likes the song B.
  - There are 1 data points supporting Paul dislikes the song B.
- **So considering major vote, we can decide that Paul likes the Song B.**
- This concept is used in one of the machine learning algorithm known as K-nearest neighbors algorithm

# Application of Machine Learning

- Machine learning is preferred approach to
  - Speech recognition, Natural language processing
  - Computer vision
  - Medical outcomes analysis
  - Robot control
  - Computational biology

# Face detection

discriminating human faces from non faces.



# Signature Recognition

- Recognize signatures by structural similarities which are difficult to quantify.
- Does a signature belongs to a specific person, say Tony Blair, or not?



Tony Blair



Tony Blair



James Gauthier



Eva Ejeris

Authentic signature



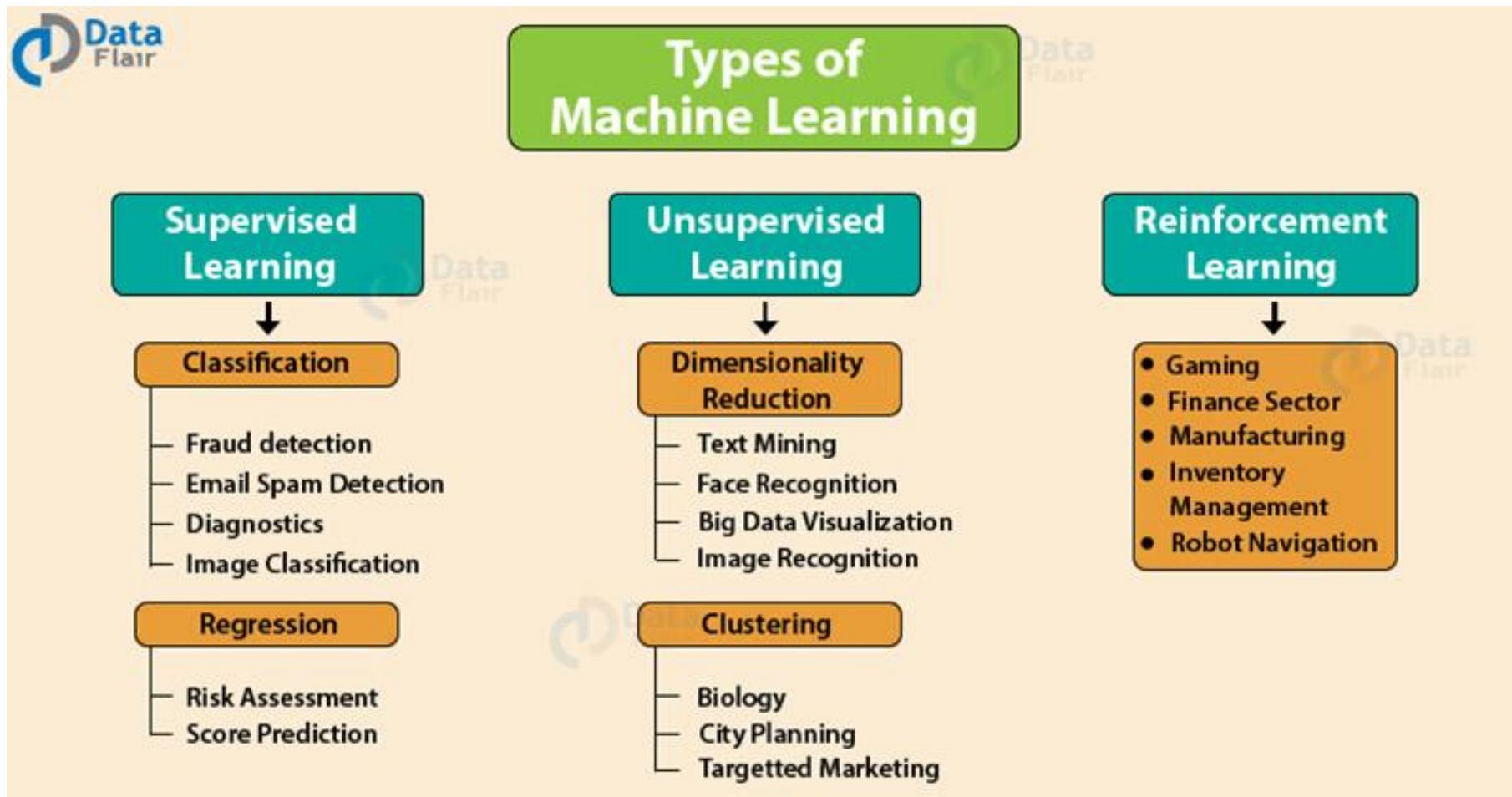
Eva Ejeris

Skilled forgery

# Real Life Example of ML Application

- Suppose that you decide to check out the offer for a vacation . You browse through the travel agency website and search for a hotel. When you look at a specific hotel, just below the hotel description there is a section titled “You might also like these hotels”. **This is a common use case of Machine Learning called “Recommendation Engine”.**
- Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, **Automatic friend tagging suggestion in Facebook**. The technology behind this is machine learning's **face detection and recognition algorithm**.
- Speech Recognition(process of converting voice instructions into text, and it is also known as "**Speech to text** ): **Google assistant, Siri, Cortana, and Alexa** are using speech recognition technology to follow the voice instructions.

# Classification of ML



# Labeled Data & Unlabeled Data

## Labeled Data

- Data + Class Label
- Expensive and Scare
- Training set is represented as

$$X = \{x_i, y_i\}_{i=1}^n$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

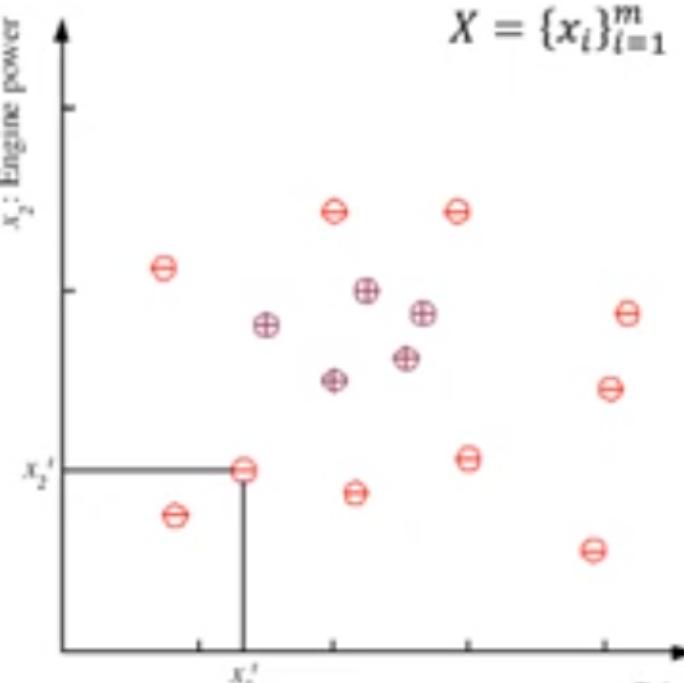
$$y = 1/0$$



## Unlabeled data

- Data only
- Cheap and Abundant
- Training set is represented as

$$X = \{x_i\}_{i=1}^m$$



# Supervised Learning

- In Supervised learning, you train the machine using data which is well "**labeled**." It means some data is already tagged with the correct answer. It can be compared to learning which takes place in the presence of a supervisor or a teacher.
- A supervised learning algorithm learns from labeled training data, helps you to predict outcomes for unforeseen data.
- Example: Say we want to identify a coin using its weight as a Feature.
  - weight: It is the feature.
  - Value: It is the class or label of the coin to be classified.
  - Say the trained model with labels as in table:
  - Testing data: A coin weighing 3 grams: So we can easily classify the testing coin as One Rupee coin based on the label in the train model.
- So Supervised learning always use labeled data to train the machine

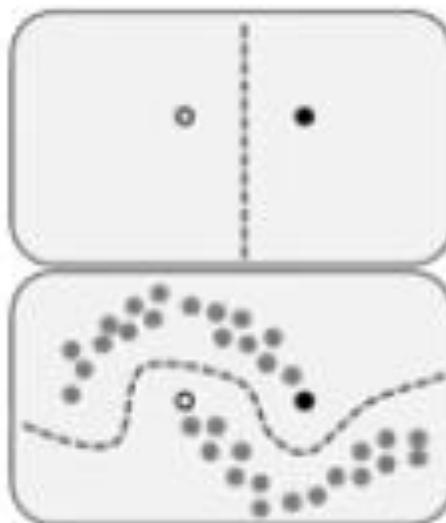
Weight (grams)	value
3	1
5	2
7	5

# Unsupervised Learning

- In supervised learning, the main idea is to learn under supervision, where the supervision signal is named as target value or label. In unsupervised learning, we **lack this kind of signal**. Therefore, we need to find our way without any supervision or guidance. This simply means that we are alone and need to figure out what is what by ourselves.
- **Unsupervised Learning** is a machine learning technique in which the users do not need to supervise the model. Instead, it allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with the unlabelled data.
- NO LABELLED DATA
- Eg: A person identifies an animal based on its observations.

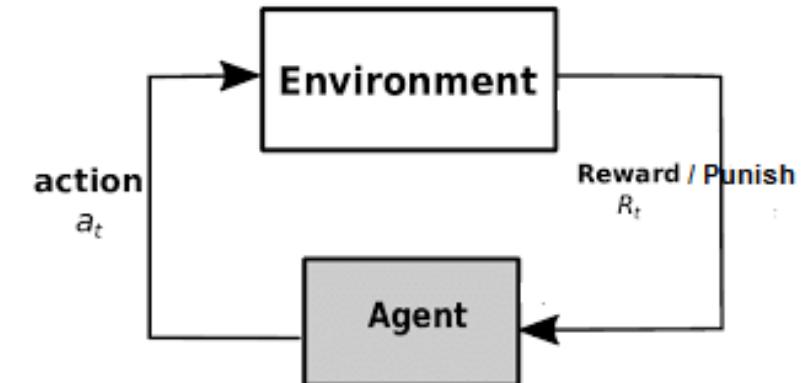
# Semi-supervised Learning

- combines a small amount of labelled data with a large amount of unlabelled data during training
- $X1 = \{x_i, y_i\}_{i=1}^n, X2 = \{x_i\}_{i=1}^m$  where  $m \gg n$



# Reinforcement Learning

- Reinforcement learning is a machine learning models to make a sequence of decisions. In reinforcement learning, an **artificial intelligence faces a game-like situation**.
- The computer employs **trial and error** to come up with a solution to the problem. To get the machine to do what the programmer wants, the **artificial intelligence gets either rewards or penalties for the actions it performs**.
- Its goal is to **maximize the total reward**. Reward based learning.
- It works on principle of feedback.
- The model learns what to do based on **positive experiences**.



# Types of Supervised Learning

- Supervised Learning has been broadly classified into 2 types.
  - **Regression**
  - **Classification**
- **Regression** is the kind of Supervised Learning that learns from the Labelled Datasets and is then able to **predict a continuous-valued output** for the new data given to the algorithm. It is used whenever the output required is a number such as money or height etc. Eg. Linear Regression, Logistic Regression etc.
- **Classification**, on the other hand, is the kind of learning where the algorithm needs to **map the new data that is obtained to any one of the 2 classes** that we have in our dataset. The classes need to be mapped to either 1 or 0 which in real-life translated to ‘Yes’ or ‘No’, ‘Rains’ or ‘Does Not Rain’ and so forth. Eg. Decision tree, Neural network, Naïve Bayes, Ensemble classifier and many more..

# Regression

- Let us say we want to have a system that can predict the price of a used car.
- Inputs are the car attributes—brand, year, engine capacity, mileage, and other information—that we believe affect a car's worth.
- The output is the price of the car. Such problems where the output is a number are *regression* problems

**Example: Price of a used car**

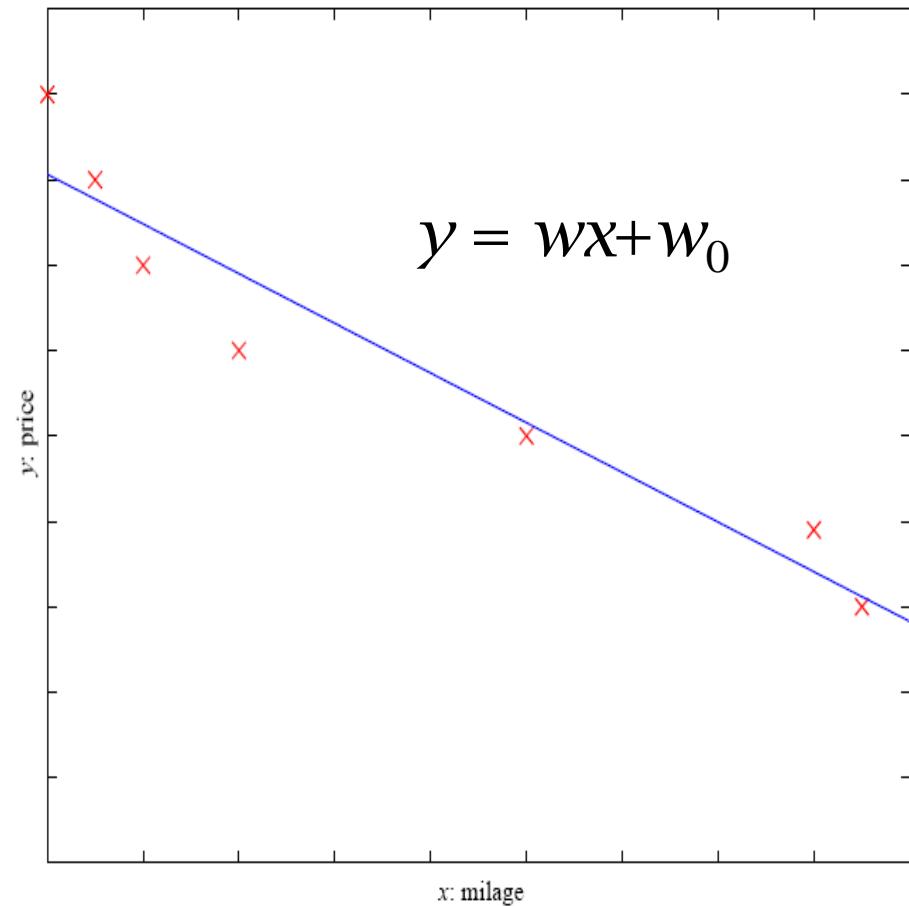
$x$  : car attributes

$y$  : price of the car

$$y = g(x | \theta)$$

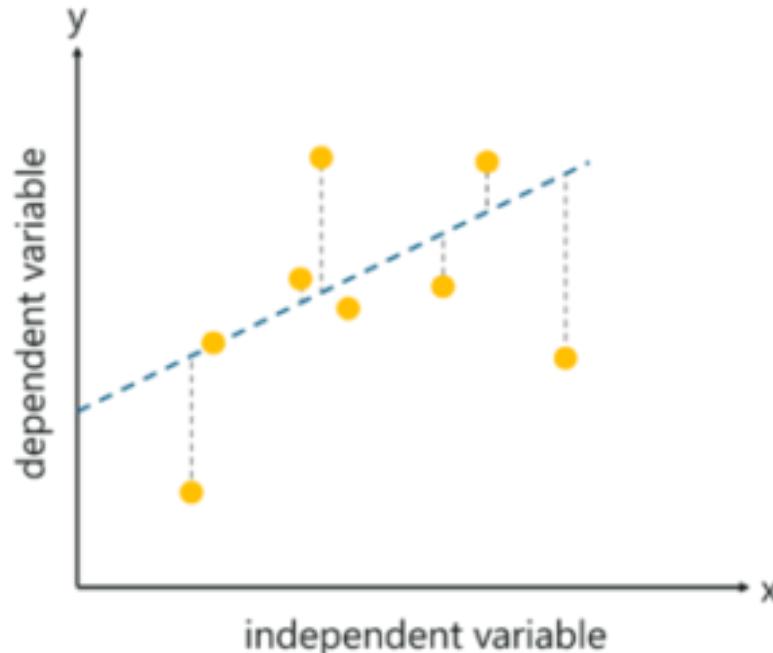
$g(\cdot)$  model,

$\theta$  parameters

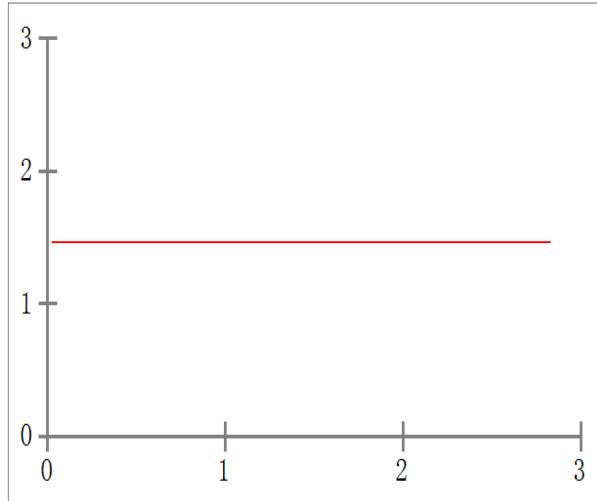


# Linear Regression

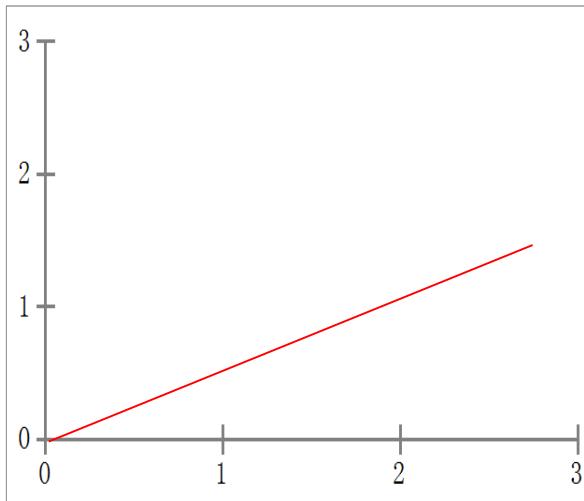
- This algorithm assumes that there is a **linear relationship between the 2 variables**, Input (X) and Output (Y), of the data it has learnt from. The **Input variable** is called the ***Independent Variable*** and the **Output variable** is called the ***Dependent Variable***. When **unseen data** is passed to the algorithm, it uses the function, calculates and **maps the input to a continuous value** for the output.



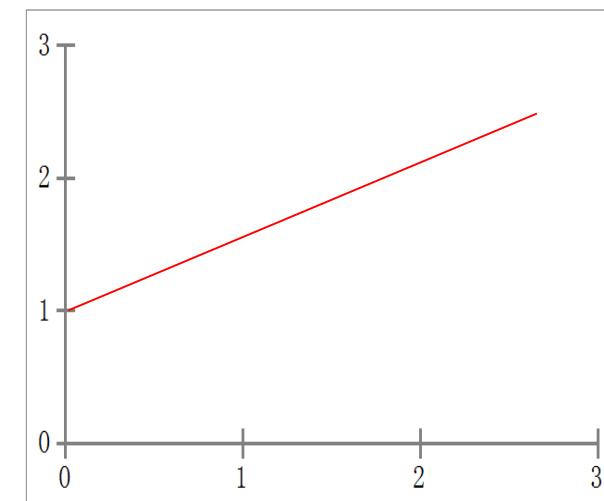
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



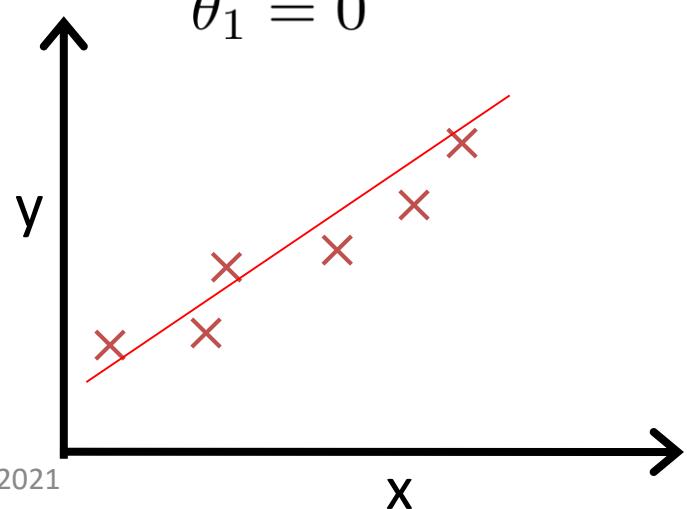
$$\begin{aligned}\theta_0 &= 1.5 \\ \theta_1 &= 0\end{aligned}$$



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



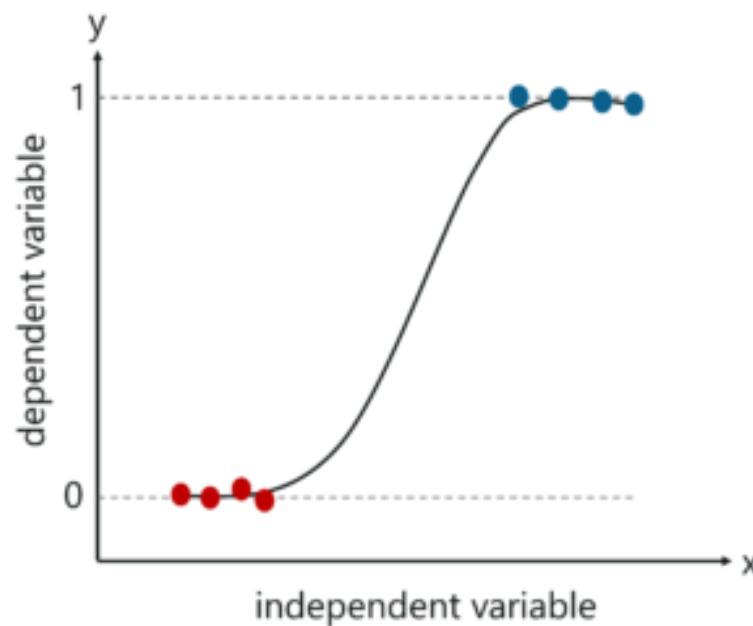
$$\begin{aligned}\theta_0 &= 1 \\ \theta_1 &= 0.5\end{aligned}$$



Idea: Choose  $\theta_0, \theta_1$  so that  $h_{\theta}(x)$  is close to  $y$  for our training  $(x, y)$  examples

# Logistic Regression

- This **algorithm predicts discrete values for the set of Independent variables** that have been passed to it. It does the prediction by mapping the **unseen data to the logit function** that has been programmed into it. The algorithm predicts the **probability of the new data** and so its output lies between the range **of 0 and 1**.



# Traditional Programming Vs. Machine Learning

## Traditional Programming



## Machine Learning



# Concept Learning

- Learning involves acquiring **general concepts** from **specific training** examples. For eg: People learn what an animal actually is (say in terms of its features) by seeing different animals like dogs, cats etc. Here dogs, cats are specific concepts whereas Animal is a general concept.
- So a concept can be viewed as describing some subset of objects or events described over a larger set.
- Alternatively each concept can be thought of as a **Boolean-valued function defined over this larger set**. For e.g: A function defined over all animals, whose value is true for birds and false for other animals) Say that function is defined as below:

$c(x) = x \text{ is a bird?}$

- if ‘x’ is parrot  $\rightarrow c(x) = 1$
- if ‘x’ is cat  $\rightarrow c(x) = \text{false}(0)$

# Definition of Concept Learning

- *Task*: learning a category description (*concept*) from a set of positive and negative training examples.
- *Target function*: a boolean function  $c: X \rightarrow \{0, 1\}$
- *Experience*: a set of training instances  $D: \{\langle x, c(x) \rangle\}$
- A search problem for best fitting hypothesis in a hypotheses space
  - The space is determined by the choice of representation of the hypothesis (all boolean functions or a subset)

## A Concept Learning Task – Enjoy Sport Training Examples

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	YES
2	Sunny	Warm	High	Strong	Warm	Same	YES
3	Rainy	Cold	High	Strong	Warm	Change	NO
4	Sunny	Warm	High	Strong	Warm	Change	YES

  
**ATTRIBUTES**   **CONCEPT**

- A set of example days, and each is described by six attributes.
- The task is to learn to predict the value of EnjoySport for arbitrary day, based on the values of its attribute values.

# A Concept Learning Task

- When learning the target concept, the learner is presented a set of *training examples*, each consisting of an instance  $x$  from  $X$ , along with its target concept value  $c(x)$
- (e.g., the training examples in Table 1). Instances for which  $c(x) = 1$  are called *positive examples*, or members of the target concept.
- Instances for which  $C(X) = 0$  are called *negative examples*, or non-members of the target concept.
- The ordered pair  $(x, c(x))$  is used to describe the training example consisting of the instance  $x$  and its target concept value  $c(x)$ .
- We use the symbol  $D$  to denote the set of available training examples.

# A Concept Learning Task

**Target Concept: Day on which person ABC enjoys his favourite water sport?**

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

- Feature vector/ Hypothesis - Conjunction of constraints on the instance attributes

- For each attribute, the hypothesis will either

- indicate by a “?” that any value is acceptable for this attribute,
- specify a single required value (e.g., *Warm*) for the attribute, or
- indicate by a “Ø” that no value is acceptable.

- Hypothesis <Sunny, Warm, ?, ?, ?, ?> indicates that ABC enjoys his watersport only on Sunny and Warm days

# A Concept Learning Task

The most general hypothesis—that every day is a positive example—is represented by

$$\langle ?, ?, ?, ?, ?, ? \rangle$$

and the most specific possible hypothesis—that *no* day is a positive example—is represented by

$$\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

# Hypothesis satisfaction

- An instance  $x$  *satisfies* an hypothesis  $h$  iff all the constraints expressed by  $h$  are satisfied by the attribute values in  $x$ .

- Example 1:

$x_1: \langle Sunny, Warm, Normal, Strong, Warm, Same \rangle$

$h_1: \langle Sunny, ?, ?, Strong, ?, Same \rangle$       Satisfies? Yes

- Example 2:

$x_2: \langle Sunny, Warm, Normal, Strong, Warm, Same \rangle$

$h_2: \langle Sunny, ?, ?, \emptyset, ?, Same \rangle$       Satisfies? No

# A Concept Learning Task

- Given:
  - Instances  $X$ : Possible days, each described by the attributes
    - $Sky$  (with possible values *Sunny*, *Cloudy*, and *Rainy*),
    - $AirTemp$  (with values *Warm* and *Cold*),
    - $Humidity$  (with values *Normal* and *High*),
    - $Wind$  (with values *Strong* and *Weak*),
    - $Water$  (with values *Warm* and *Cool*), and
    - $Forecast$  (with values *Same* and *Change*).
  - Hypotheses  $H$ : Each hypothesis is described by a conjunction of constraints on the attributes  $Sky$ ,  $AirTemp$ ,  $Humidity$ ,  $Wind$ ,  $Water$ , and  $Forecast$ . The constraints may be “?” (any value is acceptable), “ $\emptyset$ ” (no value is acceptable), or a specific value.
  - Target concept  $c$ :  $EnjoySport : X \rightarrow \{0, 1\}$
  - Training examples  $D$ : Positive and negative examples of the target function (see Table 2.1).
- Determine:
  - A hypothesis  $h$  in  $H$  such that  $h(x) = c(x)$  for all  $x$  in  $X$ .

[ $h(x) = 1$  if  $x$  satisfies  $h$ ;  $h(x) = 0$  if  $x$  does not satisfy  $h$ ]

A target concept:  $c: X \rightarrow \{0, 1\}$  where  $c(x) = 1$  iff  $EnjoySport = Yes$ ;  $c(x) = 0$  iff  $EnjoySport = No$ ;

# The Inductive Learning Assumption/ Hypothesis

- In concept learning, our goal is to find a **hypothesis  $h$  in  $H$  such that  $h(x) = c(x)$  for all  $x$  in  $D$**
- So, we can at best guarantee that the **output hypothesis** fits the target concept over the training data.
- **Inductive Learning Hypothesis / Assumption:** An hypothesis that **approximates well over a sufficiently large set of training examples** will also **approximates the target function (hypothesis)** over other unobserved examples.
- i.e. given a significant training set, the output hypothesis is able to make predictions.

# Concept Learning as Search

- Concept learning can be viewed as the task of **Searching** through a large space of hypotheses ( $H$ ) for a hypotheses that best fits the training examples.
- Hypotheses Space  $H$  for EnjoySport Learning Task?  
 $X = 3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96$  distinct instances  $x$   
 $H = 5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5120$  distinct hypotheses.
- For practical problem, we will have very large or infinite hypothesis space.  
**Solution:** Structuring the search space may help in searching more efficiently.

# General to Specific Ordering of Hypotheses

- Consider:

$$h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ? \rangle$$

$$h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ? \rangle$$

- Any instance classified positive by  $h_1$  will also be classified positive by  $h_2$
- $h_2$  is more general than  $h_1$
- Most general hypothesis:  $\langle ?, ?, ?, ?, ?, ? \rangle$
- Most specific hypothesis:  $\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

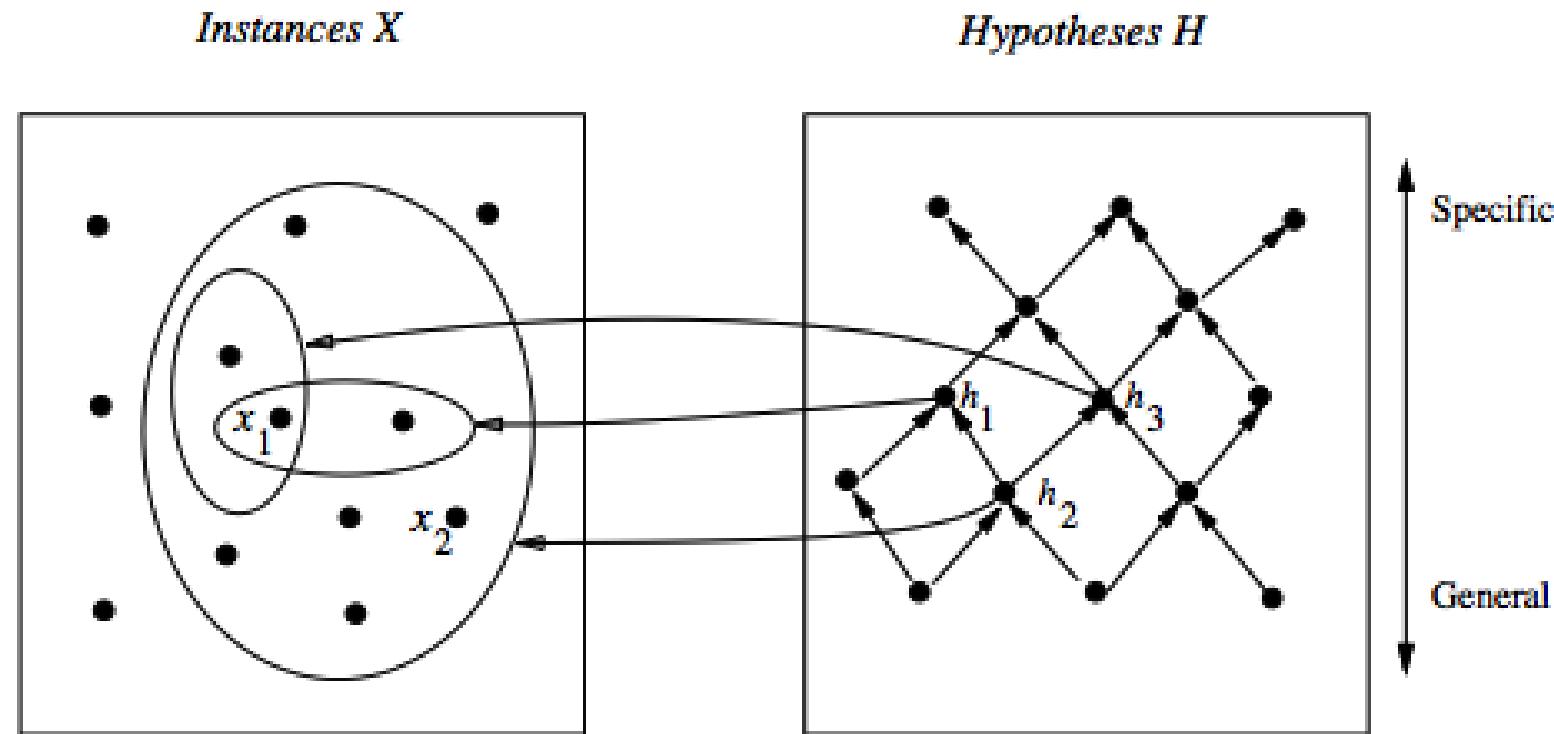
# General to Specific Ordering of Hypotheses

- “**More General than**” Relationship: For any instance  $x$  in  $X$  and hypothesis  $h$  in  $H$ , we say that  **$x$  satisfies  $h$  if  $h(x) = 1$** . Now, consider two hypothesis  $h_j$  and  $h_k$ , then  $h_j$  is said to be more general than  $h_k$  iff any instance that satisfies  $h_k$  also satisfies  $h_j$ .

**Definition:** Let  $h_j$  and  $h_k$  be boolean-valued functions defined over  $X$ . Then  $h_j$  is **more\_general\_than\_or\_equal\_to**  $h_k$  (written  $h_j \geq_g h_k$ ) if and only if

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

# General to specific ordering: induced structure



$x_1 = \langle \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Cool}, \text{Same} \rangle$

$x_2 = \langle \text{Sunny}, \text{Warm}, \text{High}, \text{Light}, \text{Warm}, \text{Same} \rangle$

$h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ? \rangle$

$h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ? \rangle$

$h_3 = \langle \text{Sunny}, ?, ?, ?, \text{Cool}, ? \rangle$

# Find-S: finding the most specific hypothesis

1. Initialize  $h$  to the most specific hypothesis in  $H$
2. For each positive training instance:  
*for each* attribute constraint  $a_i$  in  $h$ :  
    *If* the constraint  $a_i$  is satisfied by  $x$  *then do nothing*  
    *else* replace  $a_i$  in  $h$  by the next more general constraint that is satisfied by  $x$   
        (move towards a more general  $h$ )
3. Output hypothesis  $h$

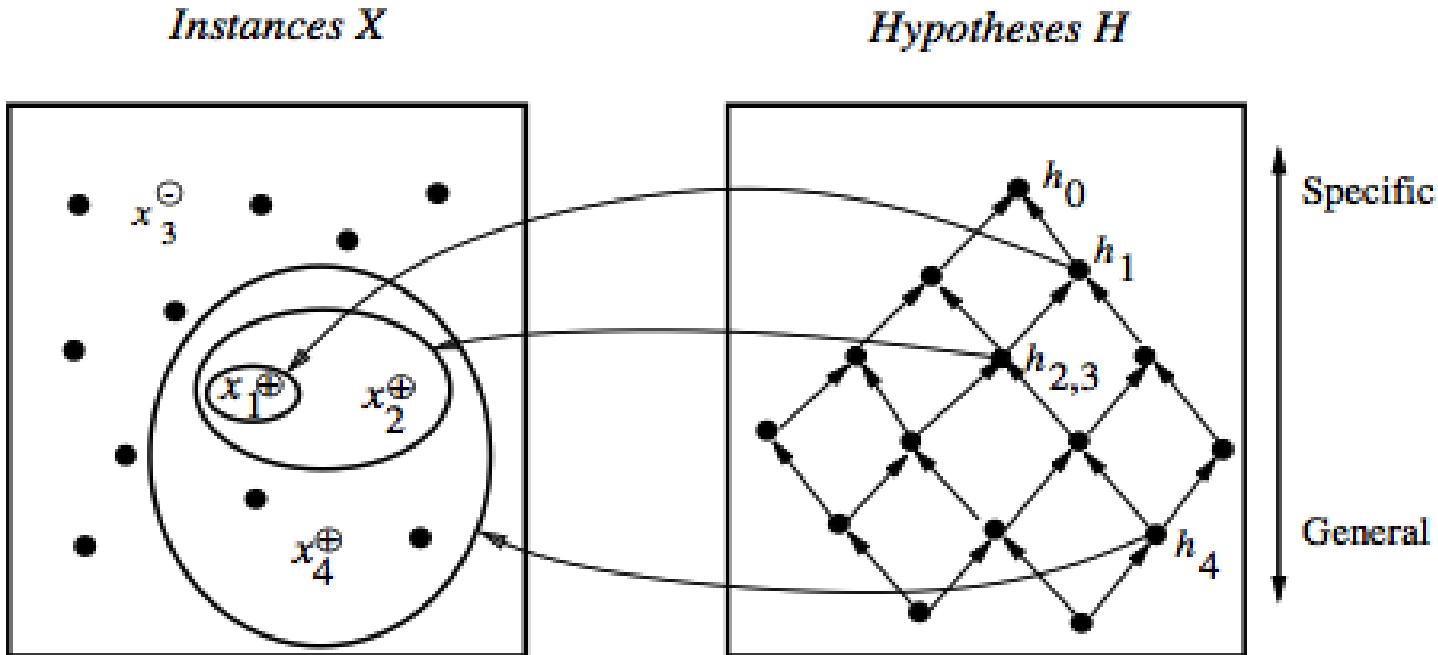
The search moves from hypothesis to hypothesis, to search the most specific hypothesis consistent with the training examples observed upto this point.

As long as we assume that the hypothesis space  $H$  contains a hypothesis that describes the true target concept  $c$  and that the training data contains no error, then the current hypothesis  $h$  never requires a revision in response to a negative example.

# Find-S in Illustration

Target concept: *Days in which person can enjoy water sport*

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	YES
2	Sunny	Warm	High	Strong	Warm	Same	YES
3	Rainy	Cold	High	Strong	Warm	Change	NO
4	Sunny	Warm	High	Strong	Warm	Change	YES



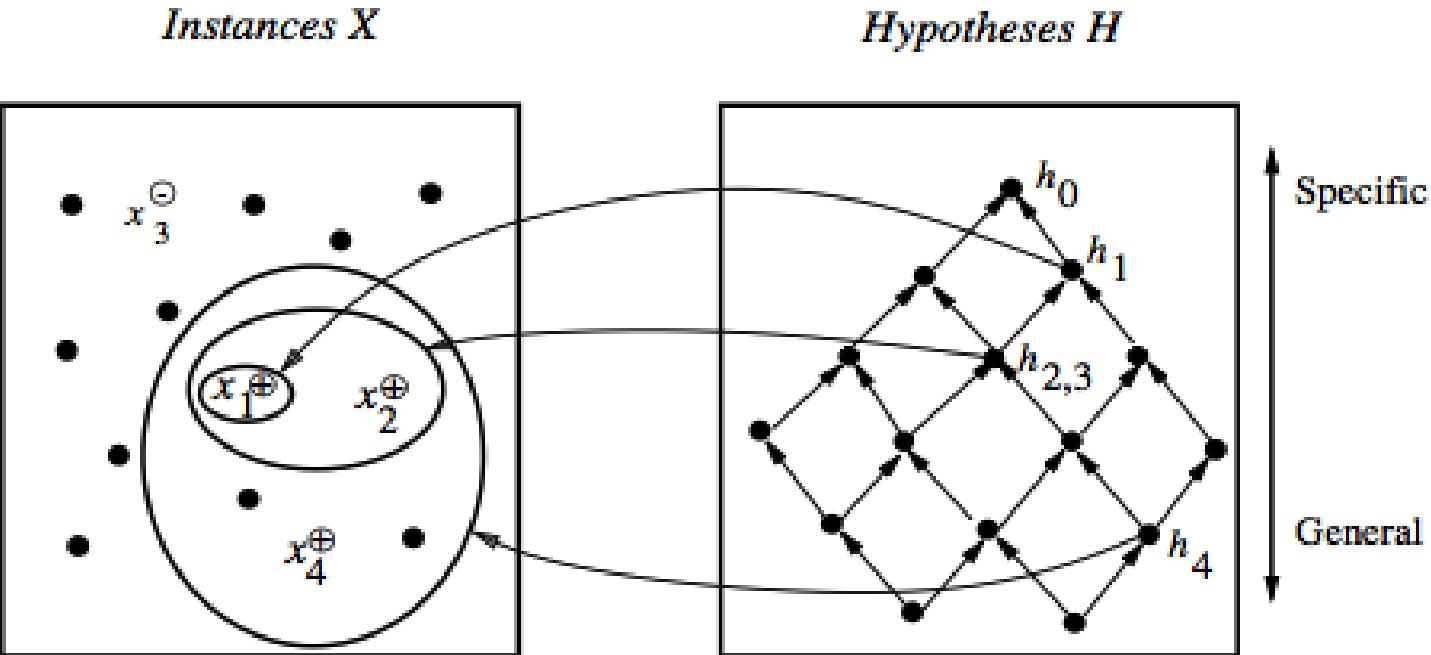
$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, +$   
 $x_2 = \langle \text{Sunny Warm High Strong Warm Same} \rangle, +$   
 $x_3 = \langle \text{Rainy Cold High Strong Warm Change} \rangle, -$   
 $x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$

$h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$   
 $h_1 = \langle \text{Sunny Warm Normal Strong Warm San} \rangle$   
 $h_2 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$   
 $h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$   
 $h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle$

# Find-S in Illustration

Target concept: *Days in which person can enjoy water sport*

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	YES
2	Sunny	Warm	High	Strong	Warm	Same	YES
3	Rainy	Cold	High	Strong	Warm	Change	NO
4	Sunny	Warm	High	Strong	Warm	Change	YES



$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, +$   
 $x_2 = \langle \text{Sunny Warm High Strong Warm Same} \rangle, +$   
 $x_3 = \langle \text{Rainy Cold High Strong Warm Change} \rangle, -$   
 $x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$

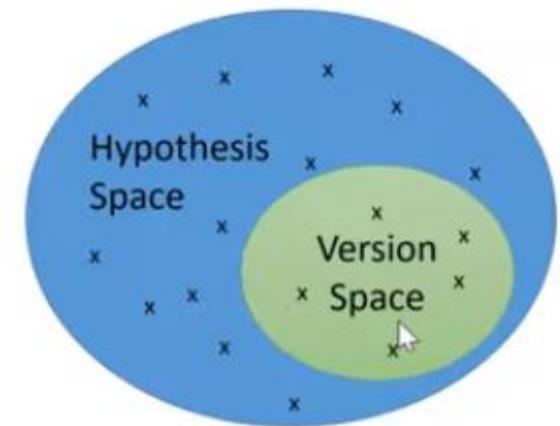
$h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$   
 $h_1 = \langle \text{Sunny Warm Normal Strong Warm San} \rangle$   
 $h_2 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$   
 $h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$   
 $h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle$

# Properties of Find-S

- *Find-S* is guaranteed to output (**find**) the *most specific hypothesis* within  $H$  that is consistent with the *positive* training examples.
- The final hypothesis will also be consistent with the negative examples
- Problems:
  - There can be more than one “most specific hypotheses”
  - We cannot say if the learner converged to the correct target
  - Why choose the most specific?
  - If the training examples are inconsistent, the algorithm can be mislead: no tolerance to rumor.
  - Negative example are not considered

# Candidate Elimination Algorithm

- **The idea:** Output a description of the set of *all hypotheses consistent* with the observed training examples (correctly classify training examples).
- **Version space:** a representation of the set of hypotheses which are *consistent* with  $D$  ( $h(x) = c(x)$  for each example  $\langle x, c(x) \rangle$  in  $D$ )
  1. List-Then-Eliminate Algorithm
  2. A compact representation of Version Space.



# Version Space

- The version space  $VS_{H,D}$  is the subset of the hypothesis from  $H$  *consistent* with the training example in  $D$

$$VS_{H,D} \equiv \{h \in H \mid \text{Consistent}(h, D)\}$$

- An hypothesis  $h$  is consistent with a set of training examples  $D$  iff  $h(x) = c(x)$  for each example in  $D$

$$\text{Consistent}(h, D) \equiv (\forall \langle x, c(x) \rangle \in D) h(x) = c(x)$$

Note: " $x$  satisfies  $h$ " ( $h(x)=1$ ) different from " $h$  consistent with  $x$ "

- In particular when an hypothesis  $h$  is consistent with a negative example  $d = \langle x, c(x)=No \rangle$ , then  $x$  must not satisfy  $h$

# The List-Then-Eliminate Algorithm

Version space as list of hypotheses

1.  $\text{VersionSpace} \leftarrow$  a list containing every hypothesis in  $H$
2. For each training example,  $\langle x, c(x) \rangle$   
    Remove from  $\text{VersionSpace}$  any hypothesis  $h$  for which  $h(x) \neq c(x)$
3. Output the list of hypotheses in  $\text{VersionSpace}$

- **Advantages**

It guarantees to output all hypotheses consistent with the training data.

- **Problems**

The hypothesis space must be finite

Enumeration of all the hypothesis, rather inefficient

# A Compact Representation for Version Space

$S: \{ <\text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ?> \}$

$G: \{ <\text{Sunny}, ?, ?, ?, ?, ?, ?>, <?, \text{Warm}, ?, ?, ?, ?, ?> \}$

**Note:** The output of *Find-S* is just  $\langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ? \rangle$

- Version space represented by its **most general members  $G$**  and **its most specific members  $S$  (boundaries)**

# General and Specific Boundaries

- The *Specific boundary*,  $S$ , of version space  $VS_{H,D}$  is the set of its minimally general (most specific) members

**Note:** any member of  $S$  is satisfied by all **positive** examples, but more specific hypotheses fail to capture some

- The *General boundary*,  $G$ , of version space  $VS_{H,D}$  is the set of its maximally general members

**Note:** any member of  $G$  is satisfied by **no negative** example but more general hypothesis cover some negative example

# Candidate Elimination Algorithm-1

$S \leftarrow$  minimally general hypotheses in  $H$ ,

$G \leftarrow$  maximally general hypotheses in  $H$

Initially any hypothesis is still possible

$$S_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle \quad G_0 = \langle ?, ?, ?, ?, ?, ? \rangle$$

For each training example  $d$ , do:

If  $d$  is a *positive* example:

1. Remove from  $G$  any  $h$  inconsistent with  $d$
2.  $\text{Generalize}(S, d)$

If  $d$  is a *negative* example:

1. Remove from  $S$  any  $h$  inconsistent with  $d$
2.  $\text{Specialize}(G, d)$

**Note:** when  $d = \langle x, \text{No} \rangle$  is a negative example, an hypothesis  $h$  is inconsistent with  $d$  iff  $h$  satisfies  $x$

# Candidate Elimination Algorithm-Continued

*Generalize( $S, d$ ):  **$d$  is positive***

For each hypothesis  $s$  in  $S$  not consistent with  $d$ :

1. Remove  $s$  from  $S$
2. Add to  $S$  all minimal generalizations of  $s$  consistent with  $d$  and having a generalization in  $G$
3. Remove from  $S$  any hypothesis with a more specific  $h$  in  $S$

*Specialize( $G, d$ ):  **$d$  is negative***

For each hypothesis  $g$  in  $G$  not consistent with  $d$ : i.e.  $g$  satisfies  $d$ ,

1. Remove  $g$  from  $G$  *but  $d$  is negative*
2. Add to  $G$  all minimal specializations of  $g$  consistent with  $d$  and having a specialization in  $S$
3. Remove from  $G$  any hypothesis having a more general hypothesis in  $G$

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	YES
2	Sunny	Warm	High	Strong	Warm	Same	YES

# Candidate Elimination Algorithm- Example

$S_0:$

$$\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

$G_0$

$$\langle ?, ?, ?, ?, ?, ? \rangle$$

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	YES
2	Sunny	Warm	High	Strong	Warm	Same	YES

# Candidate Elimination Algorithm- Example

after seeing  $\langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} \rangle +$

$S_0:$

$\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$



$S_1:$

$\langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} \rangle$

$G_0, G_1$

$\langle ?, ?, ?, ?, ?, ? \rangle$

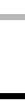
Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	YES
2	Sunny	Warm	High	Strong	Warm	Same	YES

# Candidate Elimination Algorithm- Example

after seeing  $\langle \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Warm}, \text{Same} \rangle +$

$S_1:$

$\langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} \rangle$



$S_2:$

$\langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} \rangle$

$G_1, G_2$

$\langle ?, ?, ?, ?, ?, ? \rangle$

**Note:** The processing of positive training examples may force the  $S$  boundary of the version space to become increasingly general. There is no change in  $G$  boundary.

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
3	Rainy	Cold	High	Strong	Warm	Change	NO
4	Sunny	Warm	High	Strong	Warm	Change	YES

# Candidate Elimination Algorithm- Example

after seeing *Rainy, Cold, High, Strong, Warm, Change* —

$S_2, S_3$ :  $\langle \text{Sunny, Warm, ?, Strong, Warm, Same} \rangle$

$$G_3: \quad \langle Sunny, ?, ?, ?, ?, ?, ? \rangle \langle ?, Warm, ?, ?, ?, ?, ? \rangle \langle ?, ?, ?, ?, ?, ?, Same \rangle$$

$G_2$ :  $\langle ?, ?, ?, ?, ?, ? \rangle$

**Note:** The processing of negative training examples may force the G boundary of the version space to become increasingly specific. There is no change in S boundary.

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
3	Rainy	Cold	High	Strong	Warm	Change	NO
4	Sunny	Warm	High	Strong	Warm	Change	YES

## Candidate Elimination Algorithm- Example

after seeing  $\langle \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Cool Change} \rangle +$

$S_3$

$\langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} \rangle$



$S_4$

$\langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ? \rangle$

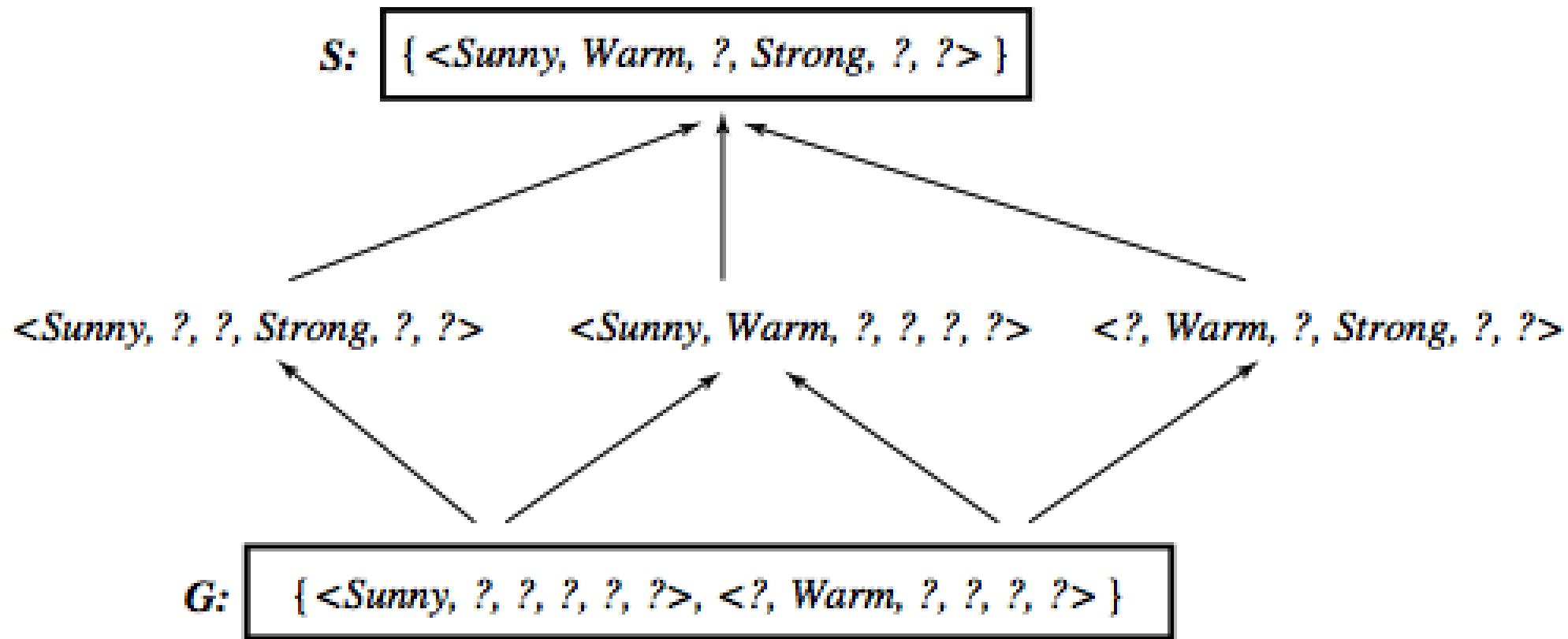
$G_4:$

$\langle \text{Sunny}, ?, ?, ?, ?, ? \rangle \langle ?, \text{Warm}, ?, ?, ?, ? \rangle$

$G_3:$

$\langle \text{Sunny}, ?, ?, ?, ?, ? \rangle \langle ?, \text{Warm}, ?, ?, ?, ? \rangle \langle ?, ?, ?, ?, ?, \text{Same} \rangle$

# Learned Version Space



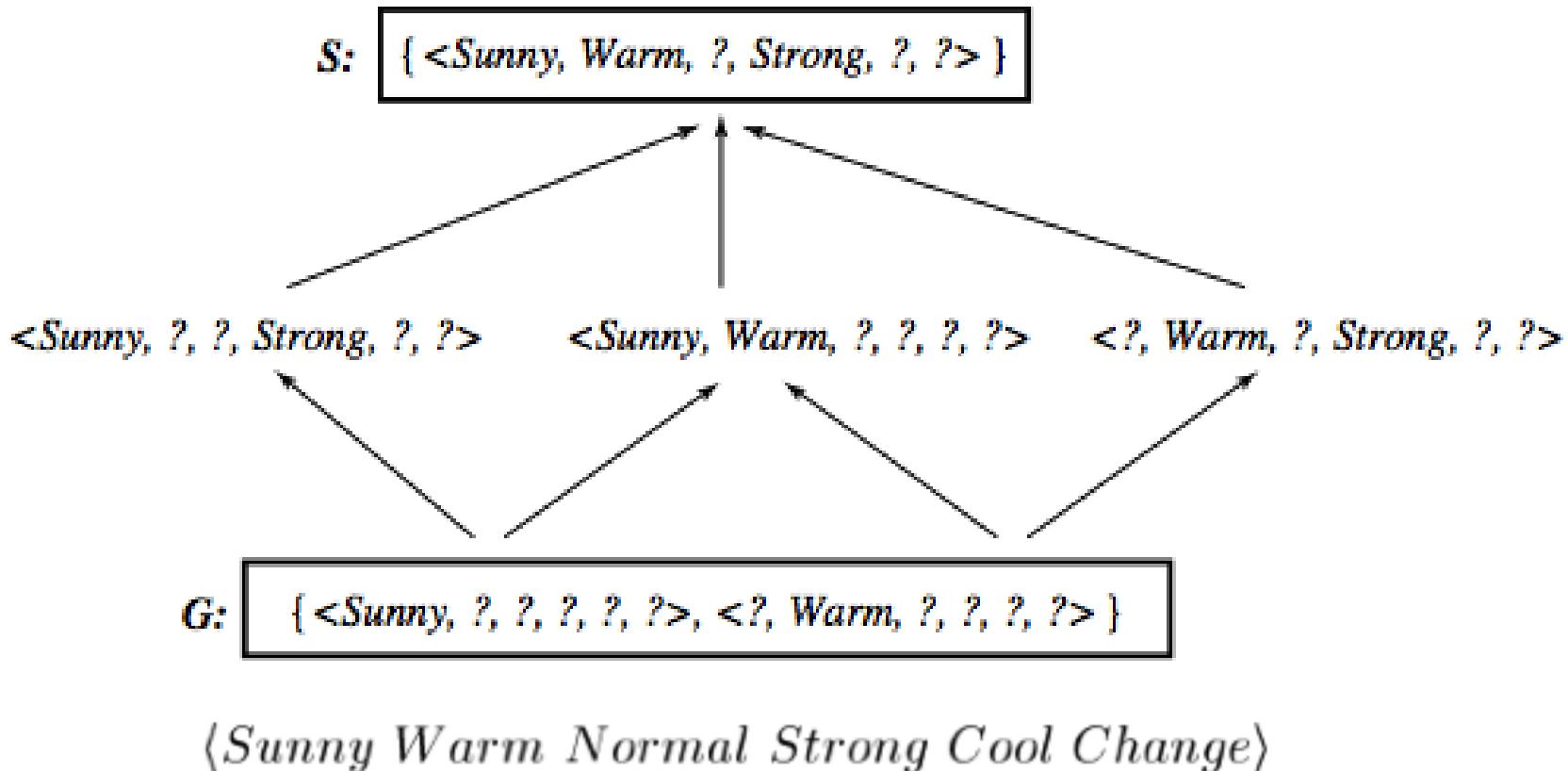
# Observations

- The learned Version Space correctly describes the target concept, provided:
  1. There are no errors in the training examples
  2. There is some hypothesis that correctly describes the target concept
- If  $S$  and  $G$  converge to a single hypothesis the concept is exactly learned
- In case of errors in the training, useful hypothesis are discarded, no recovery possible
- An empty version space means no hypothesis in  $H$  is consistent with training examples

# Ordering on Training examples

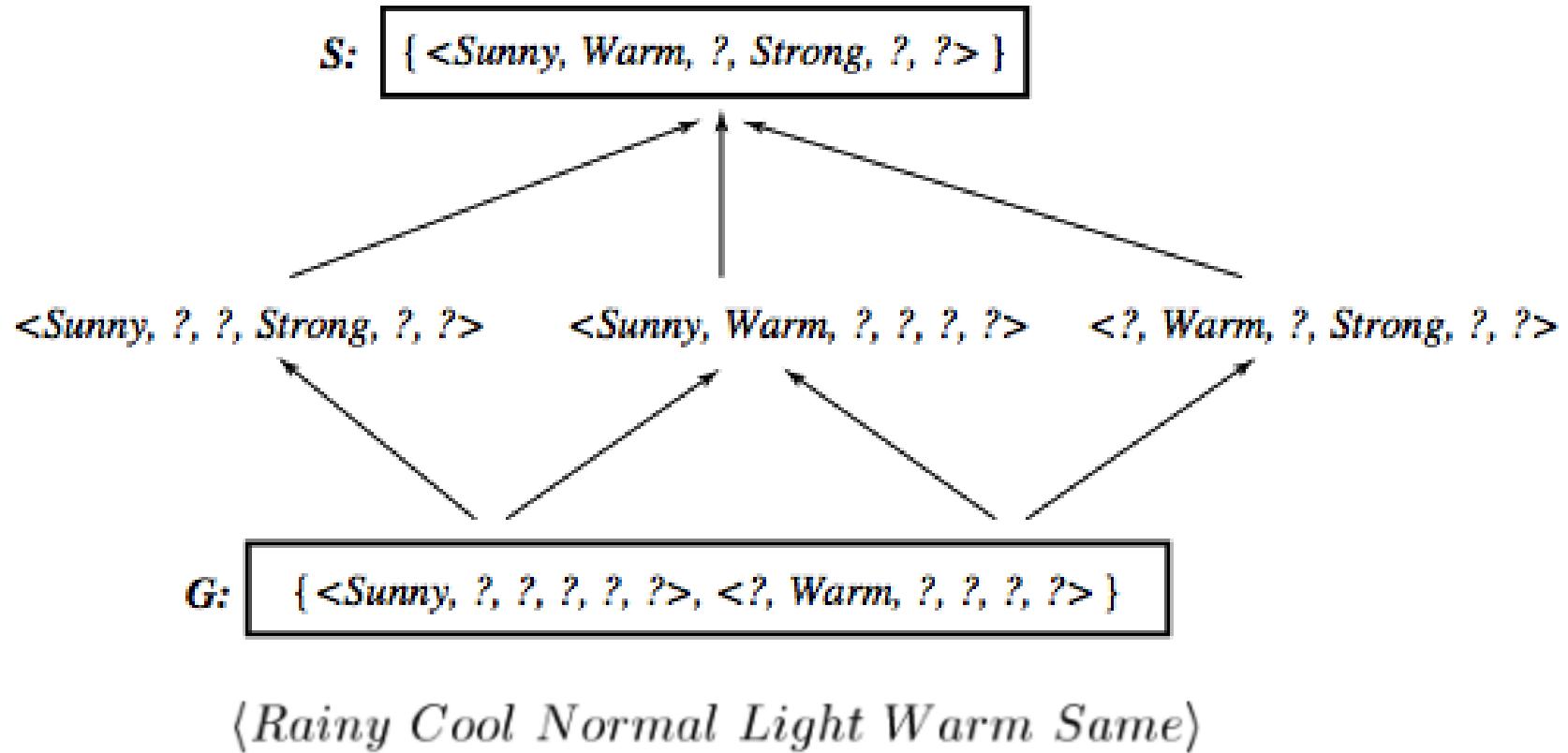
- The learned version space does not change with different orderings of training examples.
- **Optimal strategy (if you are allowed to choose)**
  - Generate instances that satisfy half the hypotheses in the current version space.  
For example:  
 $\langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Light}, \text{Warm}, \text{Same} \rangle$  satisfies 3/6 hyp.
  - Ideally the VS can be reduced by half at each experiment
  - Correct target found in  $\lceil \log_2 |\text{VS}| \rceil$  experiments

# Use of partially learned concepts



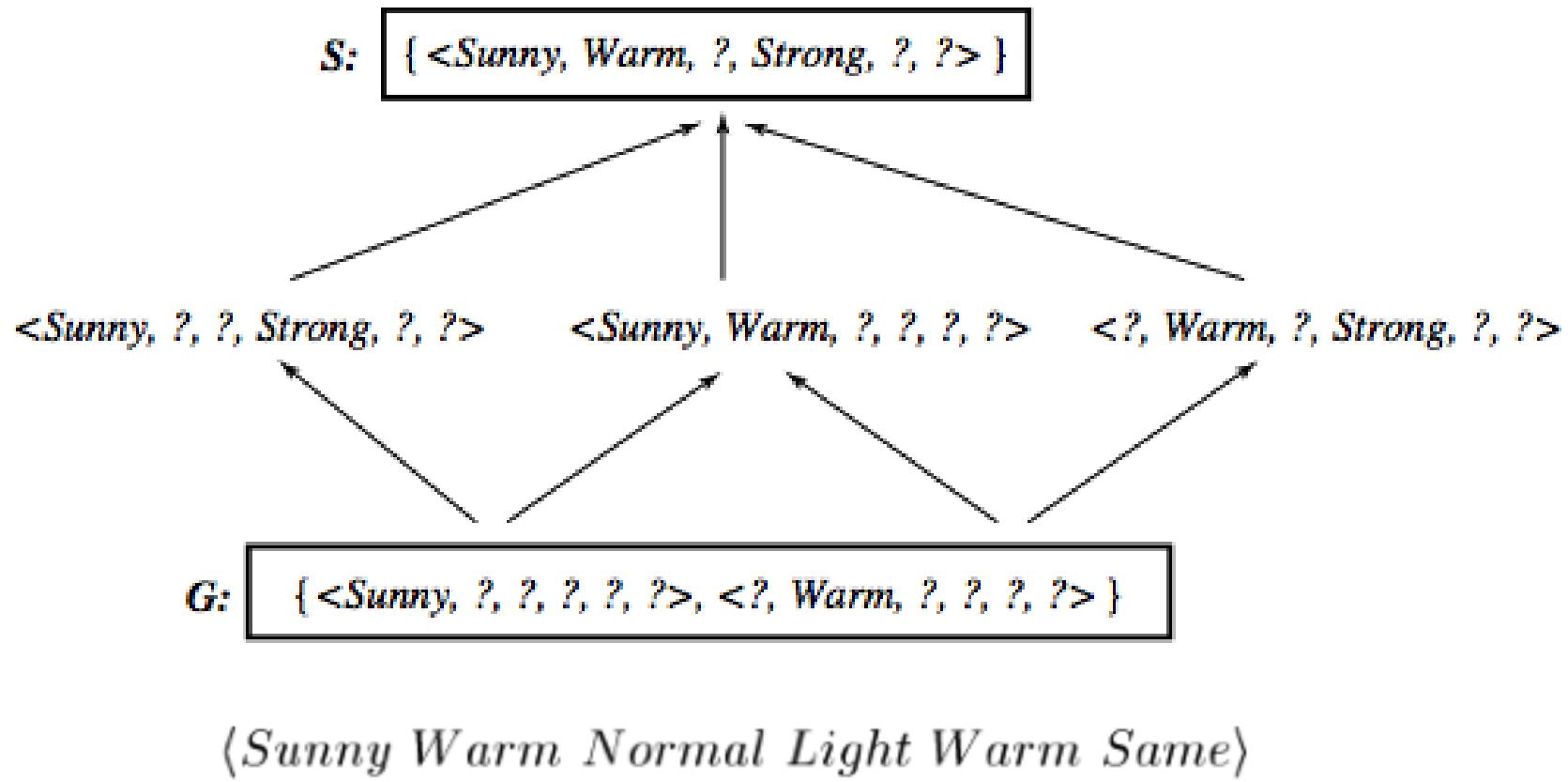
Classified as *positive* by all hypothesis, since satisfies any hypothesis in S

# Classifying new examples



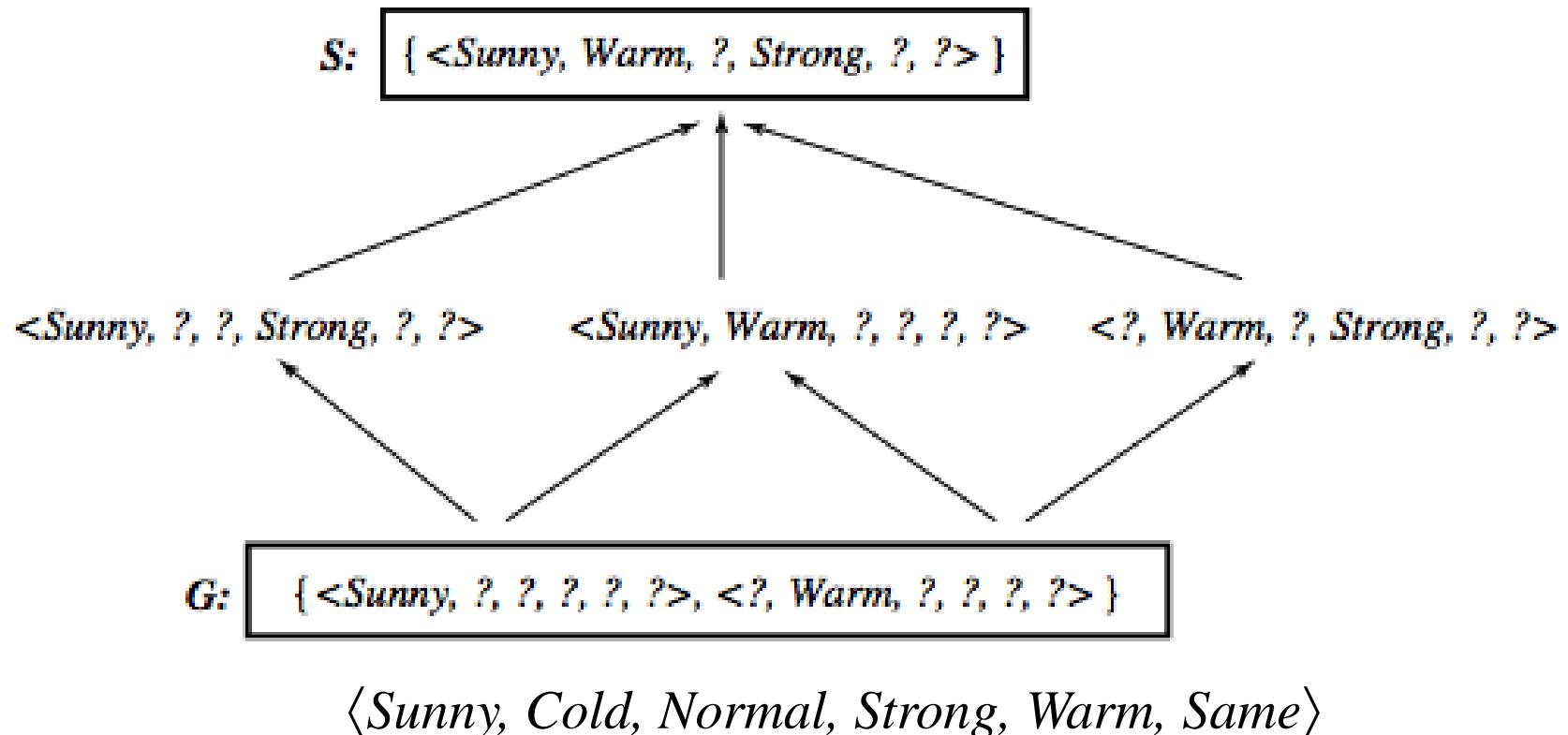
Classified as *negative* by all hypothesis, since does not satisfy any hypothesis in  $G$

# Classifying new examples



Uncertain classification: half hypothesis are consistent, half are not consistent

# Classifying new examples



4 hypothesis not satisfied; 2 satisfied Probably a negative instance. Majority vote?

# Hypothesis Space and Inductive Bias

- What if  $H$  does not contain the target concept?
- Can we improve the situation by extending the hypothesis space?
- Will this influence the ability to generalize?
- These are general questions for inductive inference, addressed in the context of Candidate-Elimination
- Suppose we include in  $H$  every possible hypothesis ... including the ability to represent disjunctive concepts

# Extending the hypothesis space

	<i>Sky</i>	<i>AirTemp</i>	<i>Humidity</i>	<i>Wind</i>	<i>Water</i>	<i>Forecast</i>	<i>EnjoyS</i>
1	<i>Sunny</i>	<i>Warm</i>	<i>Normal</i>	<i>Strong</i>	<i>Cool</i>	<i>Change</i>	<i>YES</i>
2	<i>Cloudy</i>	<i>Warm</i>	<i>Normal</i>	<i>Strong</i>	<i>Cool</i>	<i>Change</i>	<i>YES</i>
3	<i>Rainy</i>	<i>Warm</i>	<i>Normal</i>	<i>Strong</i>	<i>Cool</i>	<i>Change</i>	<i>NO</i>

- No hypothesis in the hypothesis space is consistent with these three examples since, the H/ target is a conjunction of constraints.
- Target concept exists in a different space  $H'$ , including disjunction and in particular the hypothesis ***Sky = Sunny or Sky = Cloudy***
- The most specific hypothesis consistent with the first two examples is  $\langle ?, Warm, Normal, Strong, Cool, Change \rangle$  is too general

# An Unbiased Learner

- Every possible subset of  $X$  is a possible target  
 $|H'| = 2^{|X|}$ , or  $2^{96}$  (vs  $|H| = 973$ , a strong bias)
- This amounts to allowing conjunction, disjunction and negation  
 $\langle \text{Sunny}, ?, ?, ?, ?, ? \rangle \vee \langle \text{Cloudy}, ?, ?, ?, ?, ? \rangle$   
Sunny(Sky) V Cloudy(Sky)
- We are guaranteed that the target concept exists
- No generalization is however possible!!!  
Let's see why ...

# No Generalization without Bias !

- VS after presenting three positive instances  $x_1, x_2, x_3$ , and two negative instances  $x_4, x_5$   
 $S = \{(x_1 \vee x_2 \vee x_3)\}$   
 $G = \{\neg(x_4 \vee x_5)\}$   
... all subsets including  $x_1 x_2 x_3$  and not including  $x_4 x_5$
- We can only classify precisely already seen examples !
- Take a majority vote?
  - Unseen instances, e.g.  $x$ , are classified positive (and negative) by half of the hypothesis
  - For any hypothesis  $h$  that classifies  $x$  as positive, there is a complementary hypothesis  $\neg h$  that classifies  $x$  as negative

# No Generalization without Bias !

- VS after presenting three positive instances  $x_1, x_2, x_3$ , and two negative instances  $x_4, x_5$   
 $S = \{(x_1 \vee x_2 \vee x_3)\}$   
 $G = \{\neg(x_4 \vee x_5)\}$   
... all subsets including  $x_1 x_2 x_3$  and not including  $x_4 x_5$
- We can only classify precisely already seen examples !
- Take a majority vote?
  - Unseen instances, e.g.  $x$ , are classified positive (and negative) by half of the hypothesis
  - For any hypothesis  $h$  that classifies  $x$  as positive, there is a complementary hypothesis  $\neg h$  that classifies  $x$  as negative

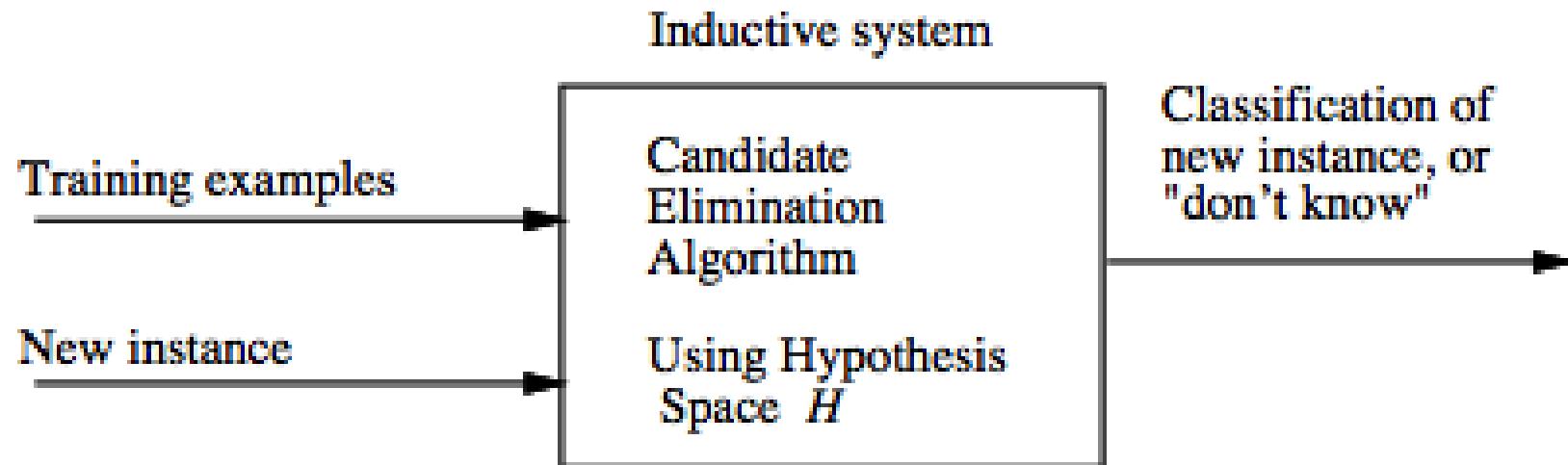
# No inductive inference without a bias

- *A learner that makes no a priori assumptions regarding the identity of the target concept, has no rational basis for classifying unseen instances.*
- The *inductive bias* of a learner are the assumptions that justify its inductive conclusions or the policy adopted for generalization
- The only reason that CEA was able to generalize beyond the observed training examples is that it was biased by the implicit assumption that the target concept could be represented by a conjunction of attribute values.
- Different learners can be characterized by their bias
- See next for a more formal definition of *inductive bias* ...

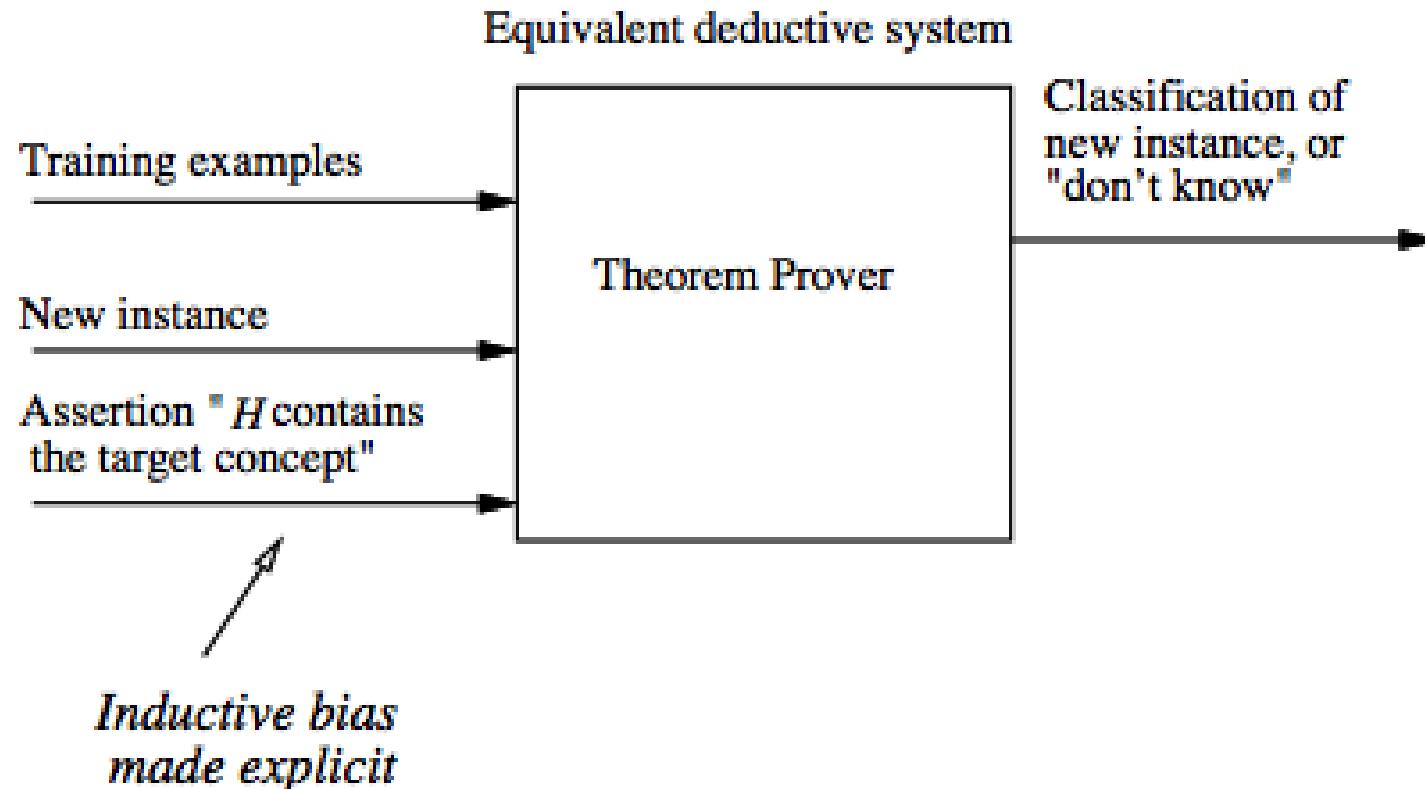
# Inductive bias of Candidate-Elimination

- Assume  $L$  is defined as follows:
  - compute  $VS_{H,D}$
  - classify new instance by [complete agreement](#) of all the hypotheses in  $VS_{H,D}$
- Then the inductive bias of Candidate-Elimination is simply
$$B \equiv (c \in H)$$
- In fact by assuming  $c \in H$ :
  1.  $c \in VS_{H,D}$ , in fact  $VS_{H,D}$  includes all hypotheses in  $H$  consistent with  $D$
  2.  $L(x_i, D_c)$  outputs a classification "by complete agreement", hence any hypothesis, including  $c$ , outputs  $L(x_i, D_c)$

# Inductive system



# Equivalent deductive system



# Each learner has an inductive bias

- Three learner with three different inductive bias:
  1. *Rote learner*: no inductive bias, just stores examples and is able to classify only previously observed examples
  2. *Candidate Elimination*: the concept is a conjunction of constraints.
  3. *Find-S*: the concept is in  $H$  (a conjunction of constraints) plus "all instances are negative unless seen as positive examples" (stronger bias)
    - The stronger the bias, greater the ability to generalize and classify new instances (greater inductive leaps).

# Decision Tree

- Decision tree is a powerful classification method (discrete valued function) that is robust to noisy data.
- Decision trees represent if-then *rules*, which can be understood by humans. In this method a set of training examples is broken down into smaller and smaller subsets while at the same time an associated decision tree get incrementally developed. At the end of the learning process, a decision tree covering the training set is returned.
- The decision tree can be thought of as a set sentences (in Disjunctive Normal Form) written propositional logic.
- Some characteristics of problems that are well suited to Decision Tree Learning are:
  - Attribute-value paired elements
  - Discrete target function
  - Works well with missing or erroneous training data

# Appropriate Problem for Decision Tree (DT) Learning

- When instances are represented by **attribute value-pairs (real valued attribute)**. Instances are represented by a fixed set of attributes (eg. Temperature) and their values( eg. Hot).
- When each attribute takes an small number of disjoint possible values. (Temperature attribute: Hot, Mild, Cold).
- When the target function has **discrete output values**. The descision tree assigns a boolean classification (eg. Yes or no) to each example.
- When the training data may contain some error because the DT learning methods are robust to errors. Errors may be in classification of the training examples or in the attribute values.
- When the training data may contain missing attribute values (eg. If the humidity of the day is known for only some of the training examples).

# Some Characteristics

- Decision tree classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of some instance.
- A decision tree is a tree with the following properties:
  - A inner node represents an attribute
  - An edge represents a test on the attribute of the parent node
  - A leaf represents one of the classes.
- In a path, a node with same label **is never repeated**.
- Decision tree **is not unique**, as different ordering of internal nodes can give different decision tree.

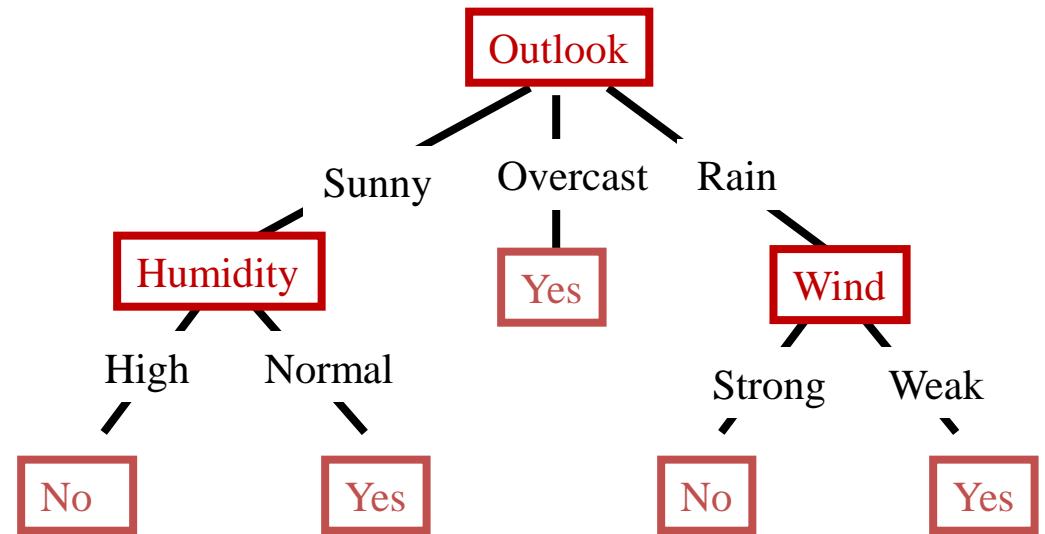
# Decision Tree

- Decision trees represent a **disjunction of conjunctions** of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and
- The tree itself is a disjunction of these conjunctions.

(Outlook = Sunny  $\wedge$  Humidity = Normal)

✓ (Outlook = Overcast)

✓ (Outlook = Rain  $\wedge$  Wind = Weak)



<Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong> is a negative instance

# Decision Tree Learning

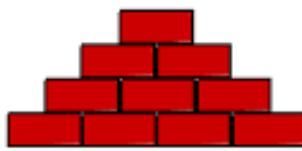
## Building a Decision Tree

1. First test all attributes and select the one that would function as the best root;
2. Break-up the training set into subsets based on the branches of the root node;
3. Test the remaining attributes to see which ones fit best underneath the branches of the root node;
4. Continue this process for all other branches until
  - a. all examples of a subset are of one type
  - b. there are no examples left (return majority classification of the parent)
  - c. there are no more attributes left (default value should be majority classification)

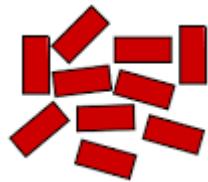
# Which Attribute is "best"?

- We would like to select the attribute that is most useful for classifying examples.
- *Information gain* measures how well a given attribute separates the training examples according to their target classification.
- ID3 uses this *information gain* measure to select among the candidate attributes at each step while growing the tree.
- In order to define information gain precisely, we use a measure commonly used in information theory, called *entropy*
- *Entropy* characterizes the (im)purity of an arbitrary collection of examples.

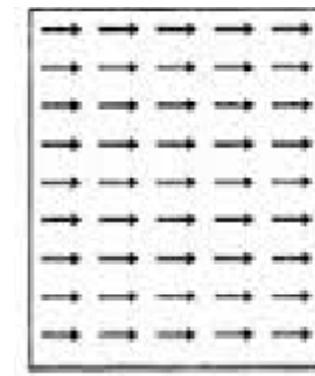
# Concept of Entropy



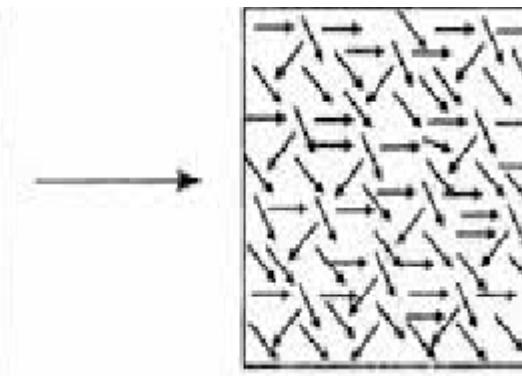
More ordered  
less entropy



Less ordered  
higher entropy



More organized or  
ordered (less probable)



Less organized or  
disordered (more probable)

# Entropy

- Given a collection  $S$ , containing **positive and negative** examples of some target concept, the *entropy of  $S$*  relative to this boolean classification is:

$$\text{Entropy}(S) = -p/(p+n)\log_2(p/(p+n)) - n/(p+n)\log_2(n/(p+n))$$

- $S$  is a sample of training examples
- $p$  is the proportion of positive examples
- $n$  is the proportion of negative examples

# Entropy

**Suppose S is a collection of 14 examples including 9 positive and 5 negative examples.  
If the collection S contains unequal number of positive and negative samples, then entropy  
is between 0 and 1.**

$$\text{Entropy}([9+, 5-] = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.940$$

**It is assumed that  $\log_2(0)$  is 0**

**If all members of S belong to the same class, then entropy(s) = 0.**

$$\text{Entropy}([8+, 0-] = -(8/8) \log_2(8/8) - (0/8) \log_2(0/8) = 0.0$$

$$\text{Entropy}([0+, 8-] = -(0/8) \log_2(0/8) - (8/8) \log_2(8/8) = 0.0$$

**If S contains an equal number of positive and negative examples, then  $\text{entropy}(S) = 1$**

$$\text{Entropy}([8+, 8-] = -(8/16) \log_2(8/16) - (8/16) \log_2(8/16) = 1.0$$

# Information Gain

- **Entropy** is a measure of the impurity in a collection of training examples.
- **Information gain** is a measure of the effectiveness of an attribute in classifying the training data.
- **Information gain** measures the expected reduction in entropy by partitioning the examples according to an attribute. More precisely, the information gain,  $\text{Gain}(S, A)$  of an attribute A, relative to a collection of example S is defined as

$$\text{Gain}(S, A) \equiv \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Where,  $\text{Values}(A)$  is the set of all possible values for attribute A and  $S_v$  is the subset of S for which attribute A has value v.

$$I(\text{attribute } A) = \sum_{i=1}^c (p_i + n_i)/(p + n) \cdot \text{Entropy}(A)$$

# Information Gain

Suppose,  $S$  is a collection of 14 training example day [9+,5-]described by the attributes Wind = {Weak, Strong}

Of these 14 examples, suppose **6 of the positive** and **2 of the negative examples have Wind = Weak** and the remainder have **Wind = Strong**. So the information gain due to sorting the original 14 example by the attribute **Wind** can be calculated as

$$\text{Values(Wind)} = \text{Weak, Strong}$$

$$S = [9+, 5-]$$

$$S_{\text{Weak}} \leftarrow [6+, 2-]$$

$$S_{\text{Strong}} \leftarrow [3+, 3-]$$

**Note:** Information Gain is used by the ID3 to select the best attribute at each step in growing tree.

$$\begin{aligned} \text{Gain}(S, \text{Wind}) &= \text{Entropy}(S) - \sum_{v \in \{\text{Weak}, \text{Strong}\}} \frac{|S_v|}{|S|} \text{Entropy}(S_v) \\ &= \text{Entropy}(S) - (8/14)\text{Entropy}(S_{\text{Weak}}) \\ &\quad - (6/14)\text{Entropy}(S_{\text{Strong}}) \\ &= 0.940 - (8/14)0.811 - (6/14)1.00 \\ &= 0.048 \end{aligned}$$

# Information Gain

Suppose,  $S$  is a collection of 14 training example day [9+,5-]described by the attributes Wind = {Weak, Strong}

Of these 14 examples, suppose **6 of the positive** and **2 of the negative examples have Wind = Weak** and the remainder have **Wind = Strong**. So the information gain due to sorting the original 14 example by the attribute **Wind** can be calculated as

$$\text{Values(Wind)} = \text{Weak, Strong}$$

$$S = [9+, 5-]$$

$$S_{\text{Weak}} \leftarrow [6+, 2-]$$

$$S_{\text{Strong}} \leftarrow [3+, 3-]$$

**Note:** Information Gain is used by the ID3 to select the best attribute at each step in growing tree.

$$\begin{aligned} \text{Gain}(S, \text{Wind}) &= \text{Entropy}(S) - \sum_{v \in \{\text{Weak}, \text{Strong}\}} \frac{|S_v|}{|S|} \text{Entropy}(S_v) \\ &= \text{Entropy}(S) - (8/14)\text{Entropy}(S_{\text{Weak}}) \\ &\quad - (6/14)\text{Entropy}(S_{\text{Strong}}) \\ &= 0.940 - (8/14)0.811 - (6/14)1.00 \\ &= 0.048 \end{aligned}$$

# Which Attribute Best?

Decide which is the better attribute: **Humidity or Wind?**

## Humidity:

$$S = [9+, 5-]$$

$$S(\text{high}) = [3+, 4-]$$

$$S(\text{Normal}) = [6+, 1-]$$

$$\text{Gain}(S, \text{Humidity}) = 0.151$$

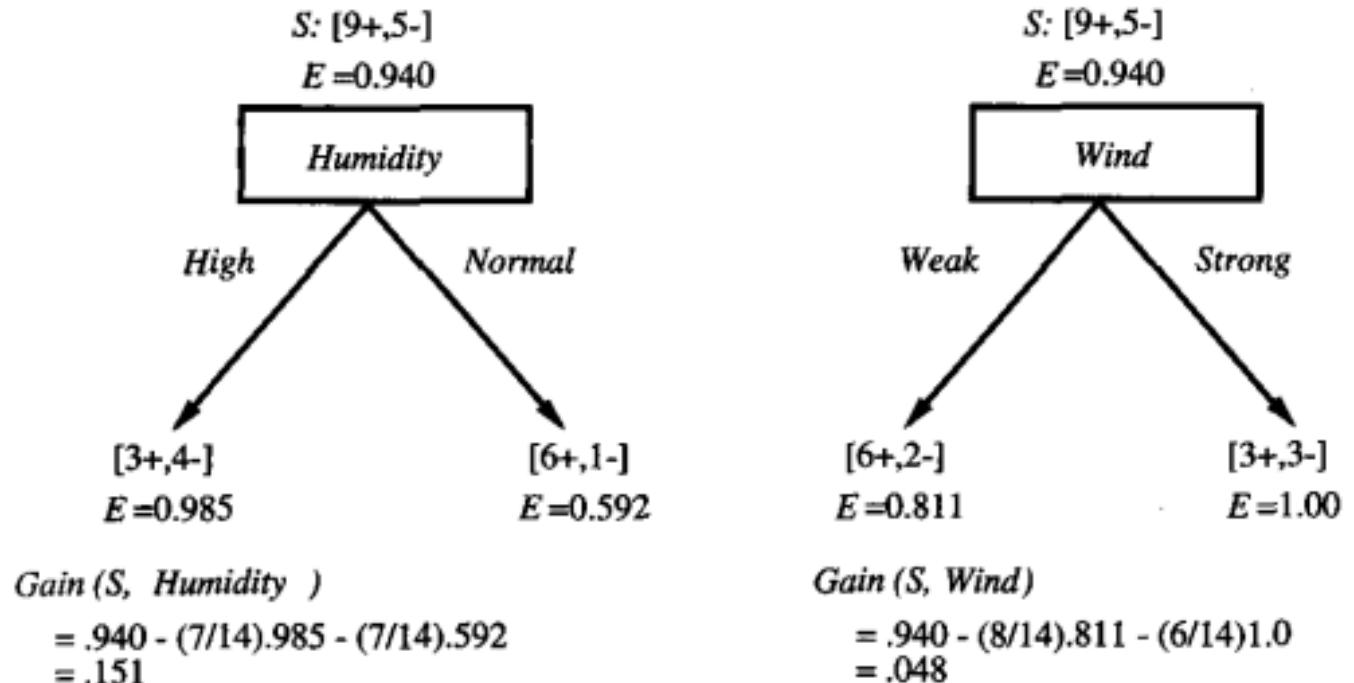
## Wind:

$$S = [9+, 5-]$$

$$S(\text{Weak}) = [6+, 2-]$$

$$S(\text{Strong}) = [3+, 3-]$$

$$\text{Gain}(S, \text{Wind}) = 0.048$$



So, **Humidity provides better information gain than Wind.**

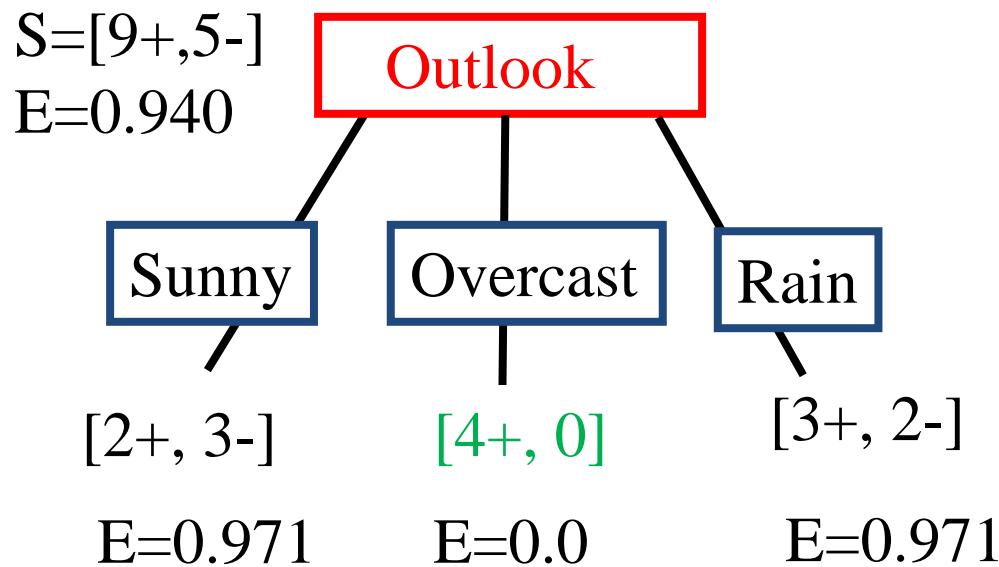
Humidity is better classifier than Wind relative to the target classification.

# ID3 - Training Examples - [9+,5-]

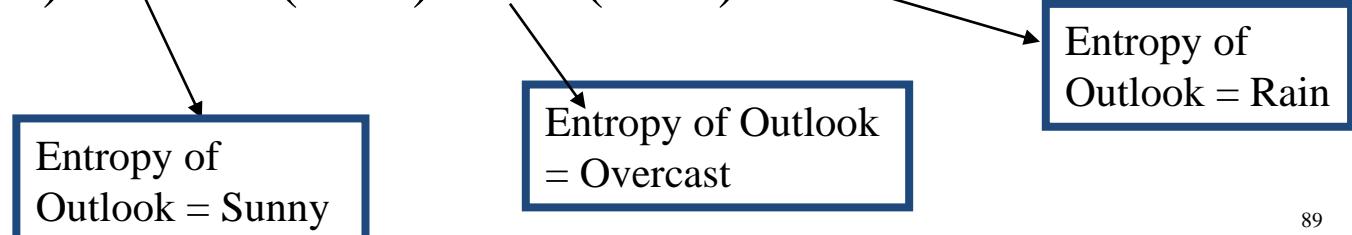
Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	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

## 4 Attributes: Root Node?

$$\text{Entropy}([9+, 5-]) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.940$$



$$\begin{aligned}\text{Gain}(S, \text{Outlook}) &= 0.940 - (5/14)*0.971 - (4/14)*0.0 - (5/14)*0.971 \\ &= \mathbf{0.246}\end{aligned}$$



# Calculate Average Information Entropy

$$I(\text{outlook}) = (P_{\text{sunny}} + N_{\text{sunny}})/(p+n) \text{ (Entropy (outlook = sunny))} + \\ (P_{\text{overcast}} + N_{\text{overcast}})/(p+n) \text{ (Entropy (outlook = overcast))} + \\ (P_{\text{rainy}} + N_{\text{rainy}})/(p+n) \text{ (Entropy (outlook = rainy))}$$

$$I(\text{outlook}) = (2+3)/(9+5) * 0.971 + + (4+0)/(9+5) * 0 + \\ (3+2)/(9+5) * 0.971$$

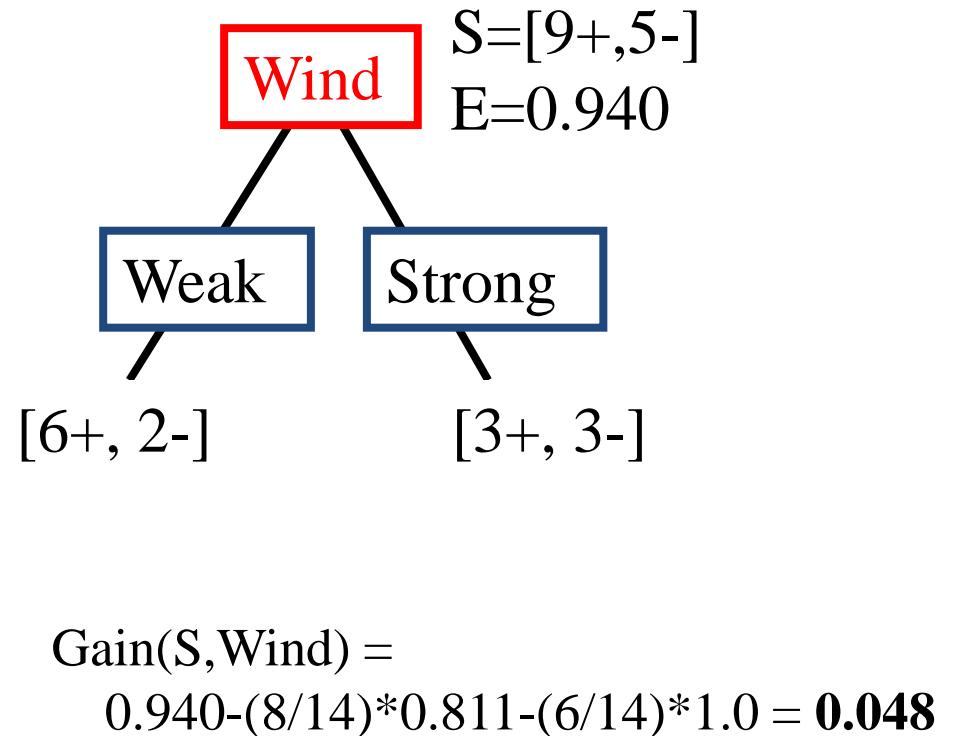
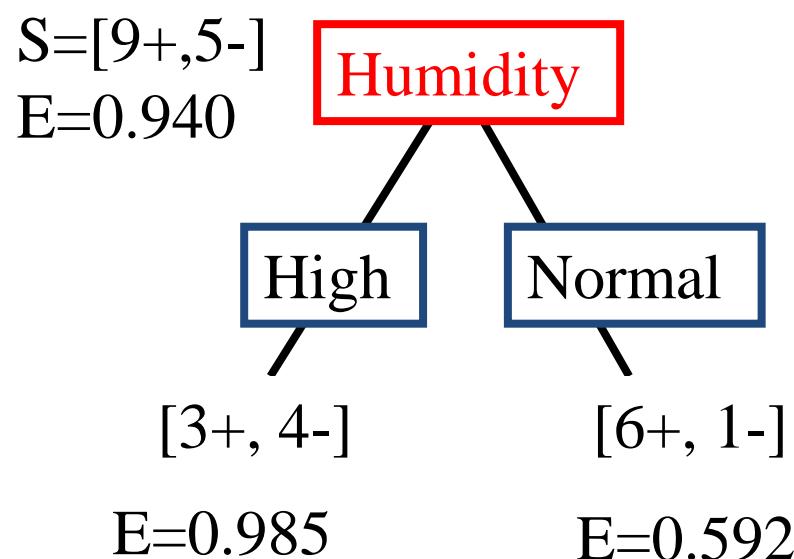
$$\text{Gain}(S, \text{Outlook}) = 0.940 - \underline{(5/14)*0.971} - \underline{(4/14)*0.0} - \underline{(5/14)*0.971}$$

(avg.info.Entropy)

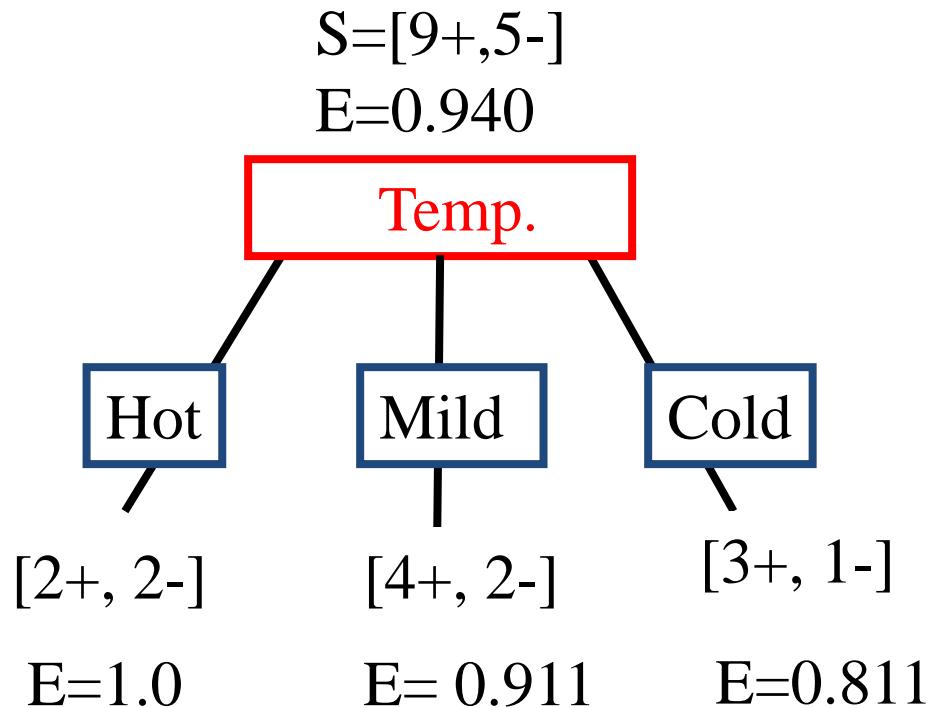
$$= \mathbf{0.247}$$

## 4 Attributes: Root Node?

$$\text{Entropy}([9+, 5-]) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.940$$

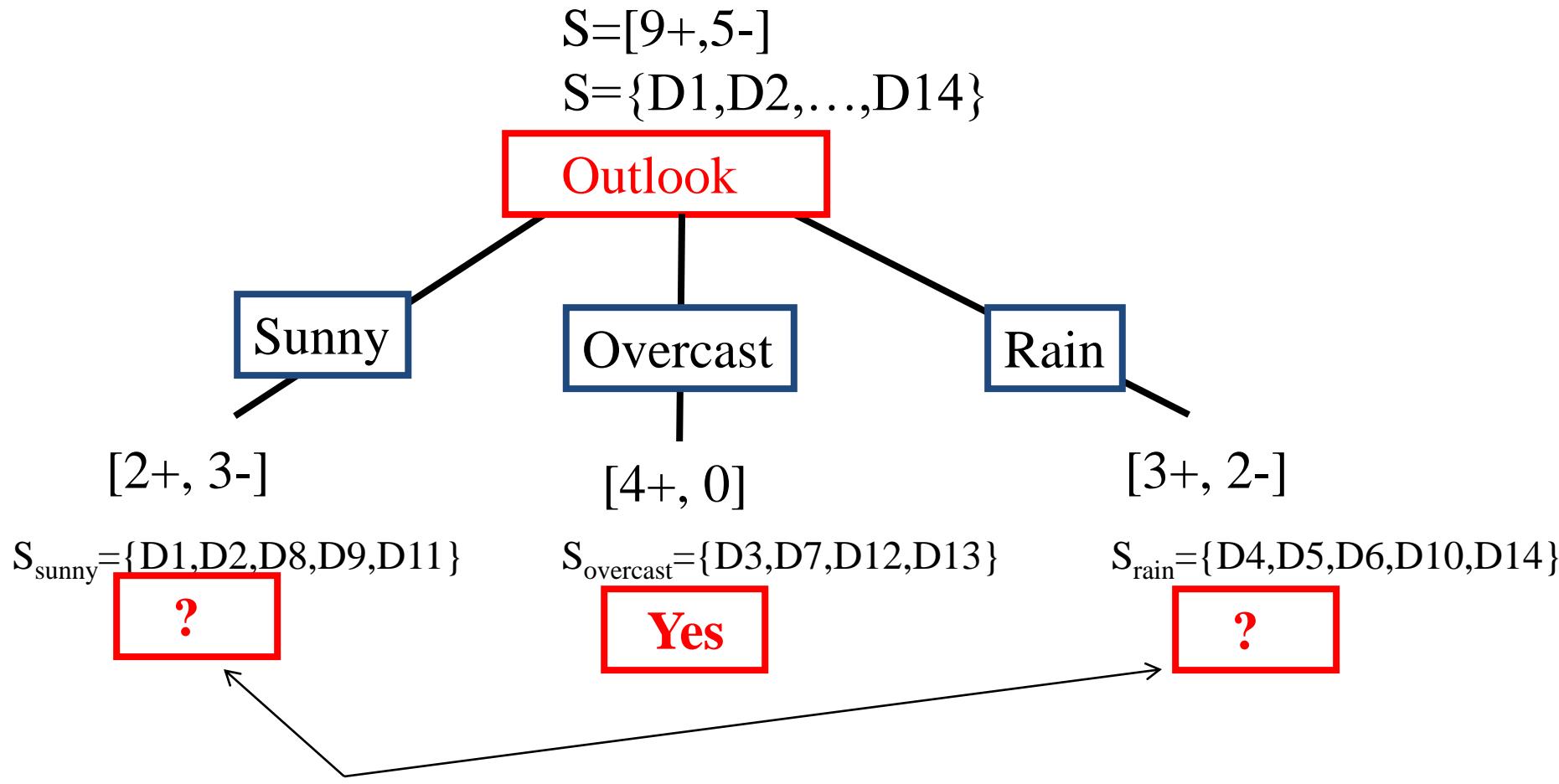


# 4 Attributes: Root Node?



$$\text{Gain}(S, \text{Temperature}) = 0.940 - (4/14)*1.0 - (6/14)*0.911 - (4/14)*0.811 = \mathbf{0.029}$$

# Best Attribute - Outlook



Repeat the same thing for the sub-trees till we get the Tree

When Outlook = Sunny

p= 2 and n=3

Outlook	Temp	Humidity	Windy	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

$$\text{Entropy}(\text{Sunny}) = -2/(2+3) \log_2(2/(2+3)) - 3/(2+3)\log_2(3/(2+3)) = 0.970$$

# ID3 - Ssunny

Outlook	Humidity	Play Tennis
Sunny	High	No
Sunny	High	No
Sunny	High	No
Sunny	Normal	Yes
Sunny	Normal	Yes

Humidity	p	n	Entropy
High	0	3	0
Normal	2	0	0

$$\text{Gain}(S_{\text{sunny}}, \text{Humidity}) = 0.970 - (3/5)0.0 - 2/5(0.0) = \mathbf{0.970}$$

## ID3 - Ssunny

$$\text{Gain}(S_{\text{sunny}}, \text{Temp.}) = 0.970 - (2/5)0.0 - 2/5(1.0) - (1/5)0.0 = 0.570$$

$$\text{Gain}(S_{\text{sunny}}, \text{Wind}) = 0.970 - (2/5)1.0 - 3/5(0.918) = 0.019$$

**So, Hummudity will be selected**

Repeat the same thing for the sub-trees till we get the tree

When Outlook = Rain

p= 3 and n=2

Outlook	Temp	Humidity	Windy	Play Tennis
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Rain	Mild	Normal	Weak	Yes
Rain	Mild	High	Strong	No

$$\text{Entropy}(\text{Rain}) = -\frac{3}{3+2} \log_2(3/(3+2)) - \frac{2}{3+2} \log_2(2/(3+2)) = 0.971$$

Outlook	Humidity	Play Tennis
Rain	High	Yes
Rain	High	No
Rain	Normal	Yes
Rain	Normal	No
Rain	Normal	Yes

Humidity	p	n	Entropy
High	1	1	0
Normal	2	1	0

Average Information Entropy I(Humidity)

$$\begin{aligned}
 \text{Gain}(S_{\text{rain}}, \text{Humidity}) &= \text{Entropy}(S_{\text{rain}}) - I_{\text{humidity}} \\
 &= 0.971 - 0.951 = 0.020
 \end{aligned}$$

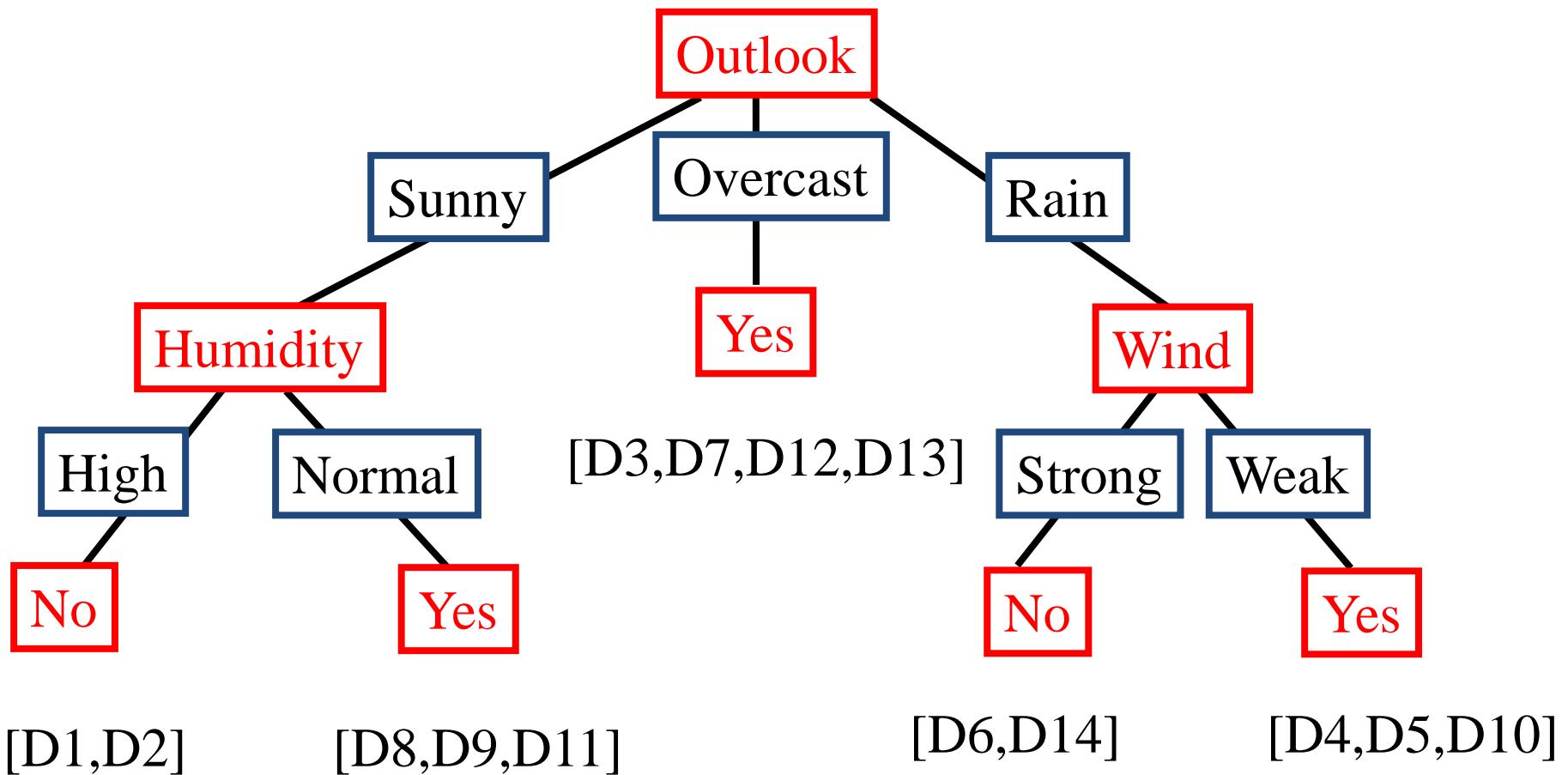
# ID3 - Srainy

$$\begin{aligned}\text{Gain}(S_{\text{rain}}, \text{Temp.}) &= \text{Entropy}(S_{\text{rain}}) - (I_{\text{temperature}}) \\ &= 0.971 - 0.951 = 0.020\end{aligned}$$

$$\begin{aligned}\text{Gain}(S_{\text{rain}}, \text{Wind}) &= \text{Entropy}(S_{\text{rain}}) - (I_{\text{windy}}) \\ &= 0.971 - 0 = \mathbf{0.971}\end{aligned}$$

**So, Wind will be selected**

# ID3 - Result



# Problem-1

## Car Examples

Color	Type	Doors	Tires	Class	
Red	SUV	2	Whitewall	+	
Blue	Minivan	4	Whitewall	-	
Green	Car	4	Whitewall	-	
Red	Minivan	4	Blackwall	-	
Green	Car	2	Blackwall	+	
Green	SUV	4	Blackwall	-	
Blue	SUV	2	Blackwall	-	
Blue	Car	2	Whitewall	+	
Red	SUV	2	Blackwall	-	
Blue	Car	4	Blackwall	-	
Green	SUV	4	Whitewall	+	
Red	Car	2	Blackwall	+	
Green	SUV	2	Blackwall	-	
Green	Minivan	4	Whitewall	-	

# ID3 - Algorithm

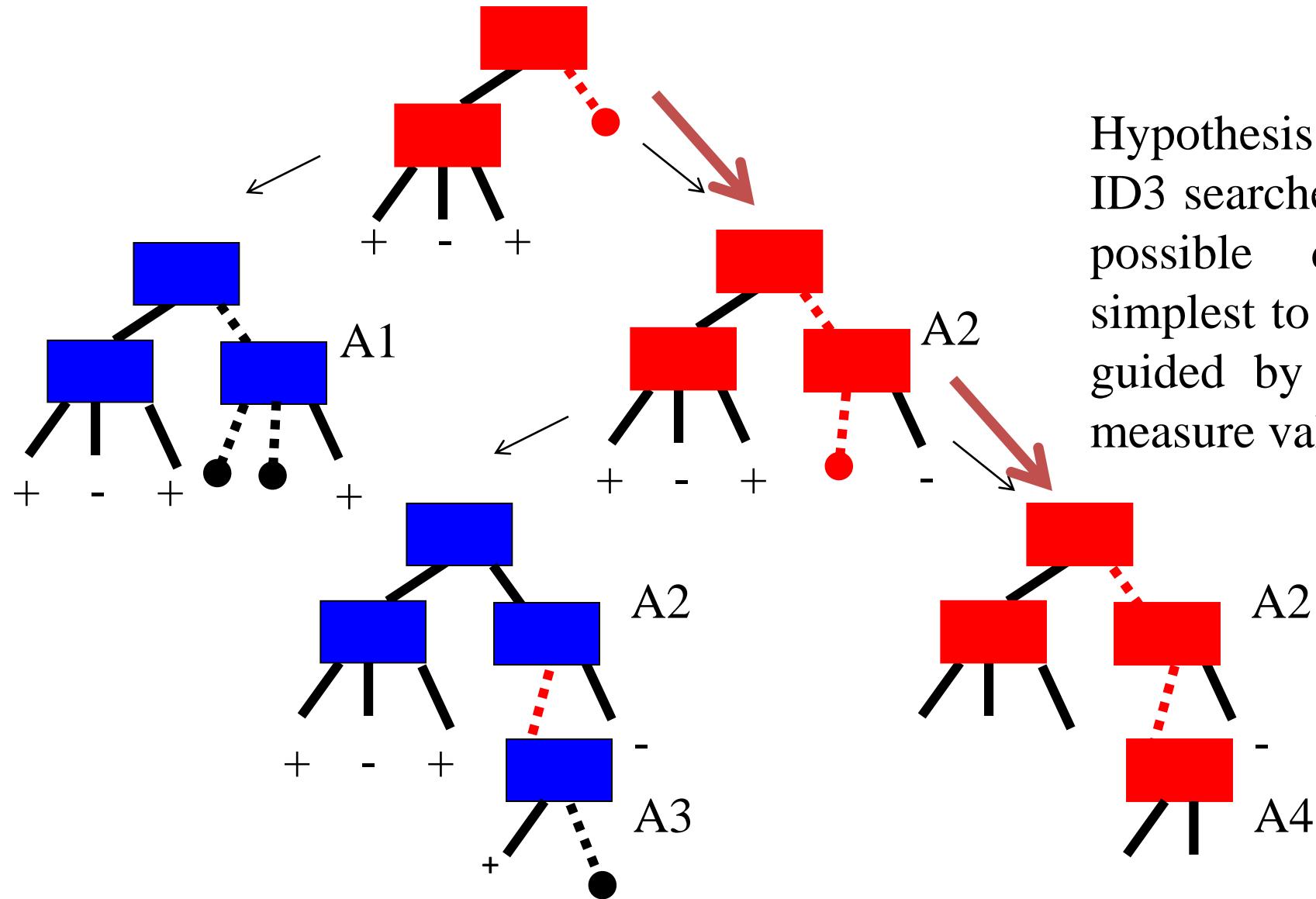
$\text{ID3}(\text{Examples}, \text{TargetAttribute}, \text{Attributes})$

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = -
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *TargetAttribute* in *Examples*
- Otherwise Begin
  - $A \leftarrow$  the attribute from *Attributes* that best classifies *Examples*
  - The decision attribute for *Root*  $\leftarrow A$
  - For each possible value,  $v_i$ , of  $A$ ,
    - Add a new tree branch below *Root*, corresponding to the test  $A = v_i$
    - Let  $\text{Examples}_{v_i}$  be the subset of *Examples* that have value  $v_i$  for  $A$
    - If  $\text{Examples}_{v_i}$  is empty
      - Then below this new branch add a leaf node with label = most common value of *TargetAttribute* in *Examples*
      - Else below this new branch add the subtree  
$$\text{ID3}(\text{Examples}_{v_i}, \text{TargetAttribute}, \text{Attributes} - \{A\})$$
- End
- Return *Root*

# Hypothesis Space Search in Decision Tree Learning (ID3)

- The hypothesis space searched by ID3 is the set of possible decision trees.
- ID3 performs a simple-to complex, hill-climbing search through this hypothesis space.
- Begins with the empty tree, then considers progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data.
- The information gain measure guides the hill-climbing search.

# Hypothesis Space Search in Decision Tree Learning (ID3)



Hypothesis Space search by ID3. ID3 searches through the space of possible decision trees from simplest to increasingly complex , guided by the Information Gain measure values.

# ID3 - Capabilities and Limitations

- ID3's hypothesis space of all decision trees is a **complete** space of finite discrete-valued functions.
  - Every finite discrete-valued function can be represented by some decision tree.
  - Target function is surely in the hypothesis space.
- ID3 maintains only a single current hypothesis, and outputs only a single hypothesis.
  - ID3 loses the capabilities that follow from explicitly representing all consistent hypotheses.
  - ID3 cannot determine how many alternative decision trees are consistent with the available training data.
- No backtracking on selected attributes (greedy search)
  - Local minimal (suboptimal splits)
- Statistically-based search choices
  - Robust to noisy data

# Inductive Bias in ID3

- ID3 search strategy
  - selects in favor of shorter trees over longer ones,
  - selects trees that place the attributes with highest information gain closest to the root.
  - because ID3 uses the information gain heuristic and a hill climbing strategy, it does not always find the shortest consistent tree, and it is biased to shorter trees and the favor trees that place attributes with high information gain closest to the root.

## Inductive Bias of ID3:

- **Shorter trees are preferred over longer trees.**
- **Trees that place high information gain attributes close to the root are preferred over those that do not.**

# Inductive Bias in ID3

- ID3 searches a complete hypothesis space
  - It searches **incompletely** through this space, from simple to complex hypotheses, until its termination condition is met
  - Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy.
  - Its hypothesis space introduces no additional bias.
- Candidate Elimination searches an incomplete hypothesis space
  - It searches this space completely, finding every hypothesis consistent with the training data.
  - Its inductive bias is solely a consequence of the expressive power of its hypothesis representation.
  - Its search strategy introduces no additional bias.

# Inductive Bias in ID3 - Restriction Bias and Preference Bias

- The inductive bias of ID3 is a preference for certain hypotheses over others (e.g. for shorter hypotheses), with no hard restriction on the hypotheses that can be eventually enumerated. This type of bias is called **preference bias** (*search bias* ).
- The inductive bias of Candidate Elimination is in the form of a categorical restriction on the set of hypotheses considered. This type of bias is called **restriction bias** (*language bias* )
- Given that some form of inductive bias is required in order to generalize a unseen data, then which type of inductive bias shall we prefer? A **preference bias** is more desirable than a *restriction bias*, because it allows the learner to work within a complete hypothesis space that is assured to contain the unknown target function.

# Inductive Bias in ID3 - Occam's Razor

**OCCAM'S RAZOR:** Prefer the simplest hypothesis that fits the data.

**Why prefer short hypotheses?**

***Argument in favor:***

- Fewer short hypotheses than long hypotheses
- A short hypothesis that fits the data is unlikely to be a coincidence
- A long hypothesis that fits the data might be a coincidence

***Argument opposed:***

- There are many ways to define small sets of hypotheses
- What is so special about small sets based on *size* of hypothesis

# Issues in Decision Tree Learning - Overfitting

- **Definition of Overfitting:**

Given a hypothesis space  $H$ , a hypothesis  $h \in H$  is said to ***OVERFIT*** the training data if there exists some alternative hypothesis  $h' \in H$ , such that ***h has smaller error than h' over the training examples***, but  $h'$  has a smaller error than  $h$  over the entire distribution of instances (***including the instances beyond the training set***).

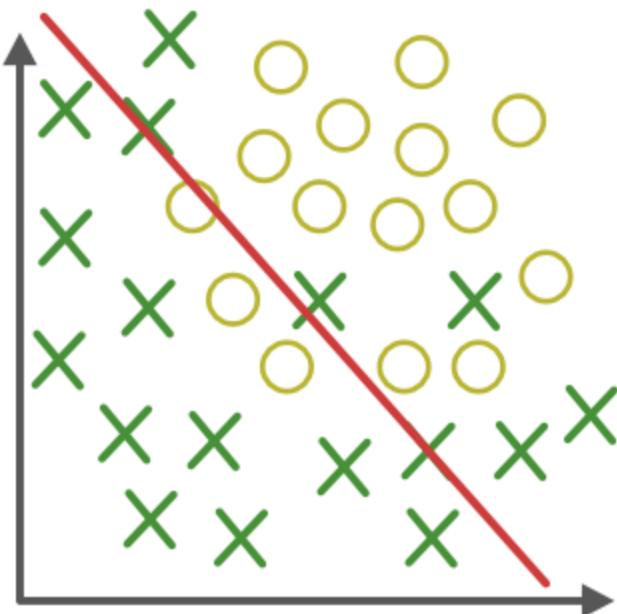
- **Reasons for overfitting:**

- Errors and noise in training examples
- Small number of training samples to produce a representative sample.

# Overfitting

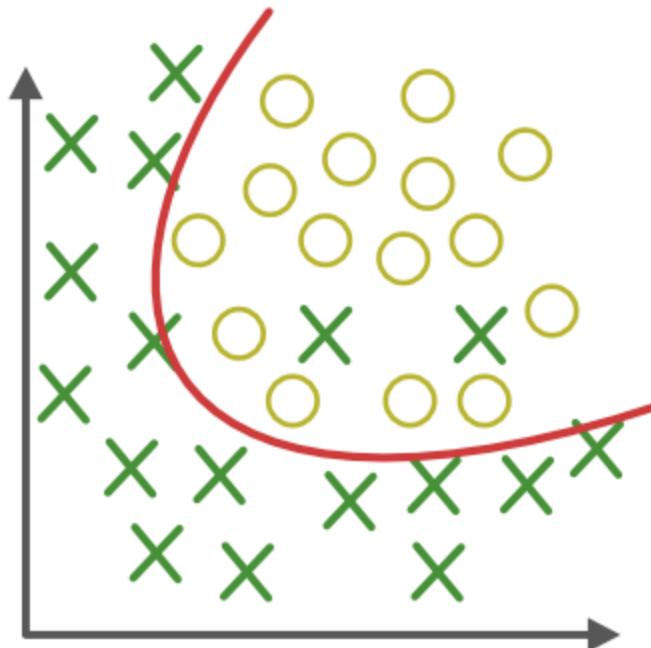
- Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.
- The noise or random fluctuations in the training data is picked up and learned as concepts by the model.
- These concepts are not applicable to new data and negatively impact the models ability to generalize.

# Overfitting

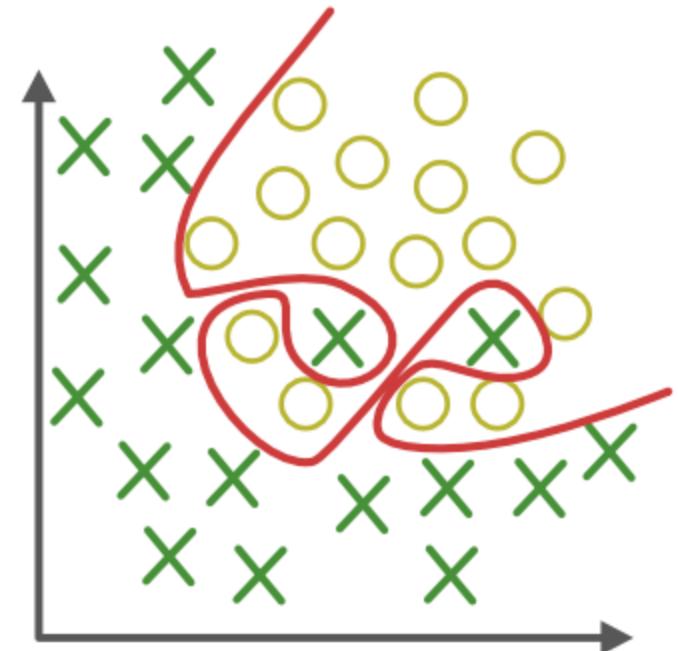


**Under-fitting**

(too simple to explain the variance)



**Appropriate-fitting**

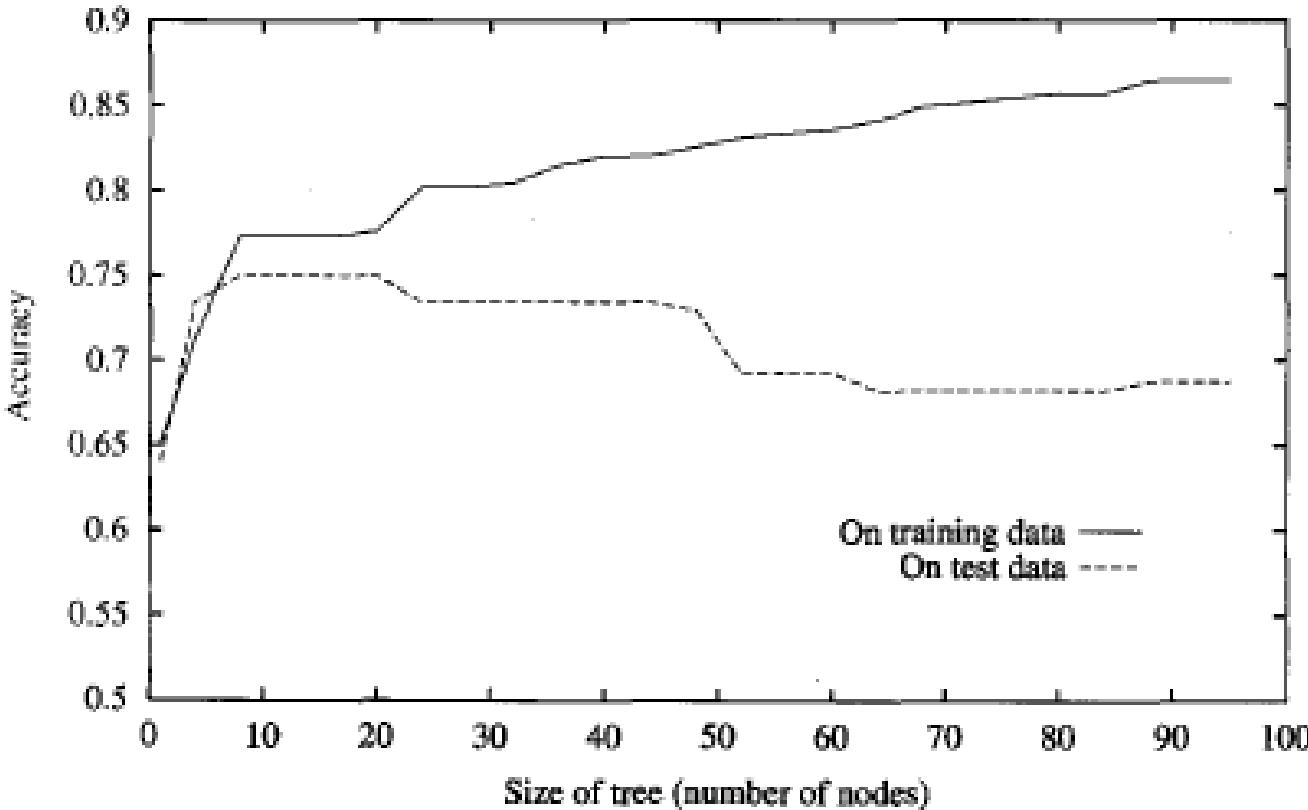


**Over-fitting**

(forcefitting--too good to be true)

DG

# Overfitting



- As ID3 adds new nodes to grow the decision tree, the accuracy of the tree measured over the training examples increases monotonically.
- However, when measured over a set of test examples independent of the training examples, accuracy first increases, then decreases.

# Avoid Overfitting

How can we avoid overfitting?

- Stop growing when data split not statistically significant
  - stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
- Grow full tree then post-prune
  - allow the tree to overfit the data, and then post-prune the tree.
- If the correct tree size is found by stopping early or by post-pruning, a key question is what criterion is to be used to determine the correct final tree size.
  - Use a **separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree.**
  - Use all the available data for training, but apply a statistical test to estimate whether expanding a particular node is likely to produce an improvement beyond the training set.

# Avoid Overfitting: Approach 1

**Approach 1:** Use a **separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree.**

This approach is very common and referred as a **training and validation set approach**. In this approach the data are separated into two sets of examples: **a training set** (used for forming the learned hypothesis) and **a separate validation set** (used for evaluating the accuracy of the learned hypothesis over subsequent data).

**Motivation:** Even though the learner may be misled by **random errors within the training set**, the validation set is unlikely to **exhibit the same random fluctuations**. Therefore the validation set can be expressed to provide a **safety check against the Overfitting the spurious characteristics of the training set**.

**Note:** it is important that the validation set be **large enough to itself** provide a statistically significant sample of instances.

One **common practice** is to withhold the **one third** of the available examples for the validation set using the **other two third** for training.

## Avoid Overfitting - Approach 2

- **Reduced Error Pruning:**

Pruning is one of the techniques that is used to **overcome the Overfitting**. Pruning is a practice which involves the selective removal of certain parts of a tree.

- It reduces the size of a Decision Tree which might slightly **increase your training error** but drastically **decrease your testing error**, hence making it more adaptable.
- It uses a validation set to prevent overfitting.

# Avoid Overfitting - Approach 2

- **Procedure of Reduced Error Pruning**
  1. Consider each node for pruning
  2. Pruning = removing the subtree at that node, make it a leaf and assign the most common class at that node
  3. A node is removed if the resulting tree performs no worse than the original on the validation set.
  4. Nodes are removed iteratively choosing the node whose removal most increases the decision tree accuracy on the graph
  5. Pruning continues until further pruning is harmful
  6. uses training, validation & test sets - effective approach if a large amount of data is available

# Avoid Overfitting - Approach 2

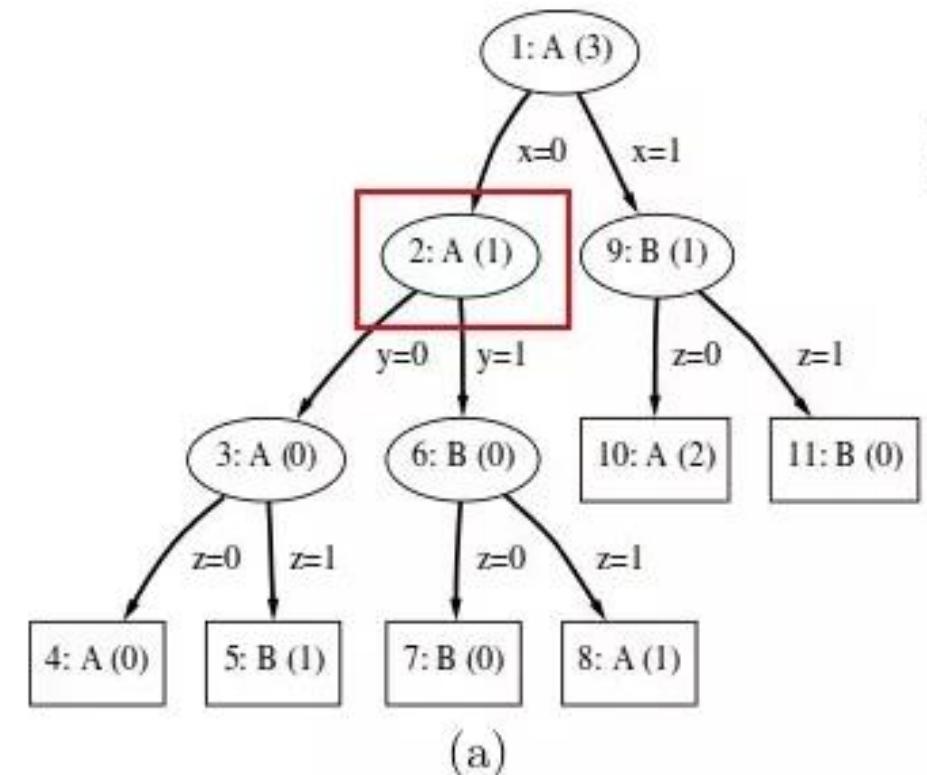
- **Procedure of Reduced Error Pruning**

**Note:** The pruning happens **bottom-up** and **left-to-right**.

Every node notes the number of samples it misclassifies in parenthesis. Eg. the node in marked in red misclassifies one example from the pruning set.

Starting from the bottom its easy to see that **node 3 can be made into a leaf** since it makes lesser errors (on the pruning set) than as a sub-tree - as a sub-tree the classification happens at nodes 4 and 5, and 5 makes one error; node 3 by itself makes no errors.

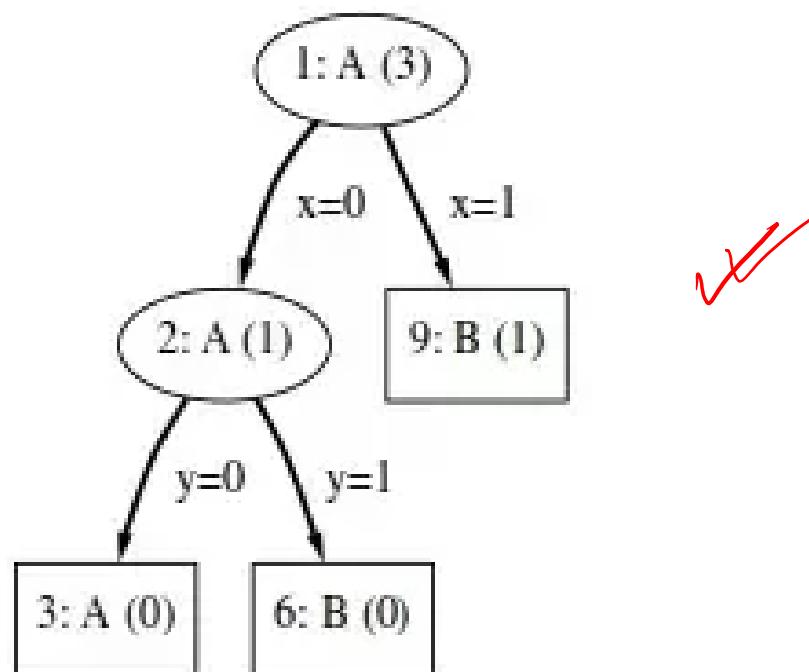
Same goes for 6 and 9. However, 2 cannot be made into a leaf since it makes one error, while as a sub-tree, with the newly-created leaves 3 and 6 it makes no errors.



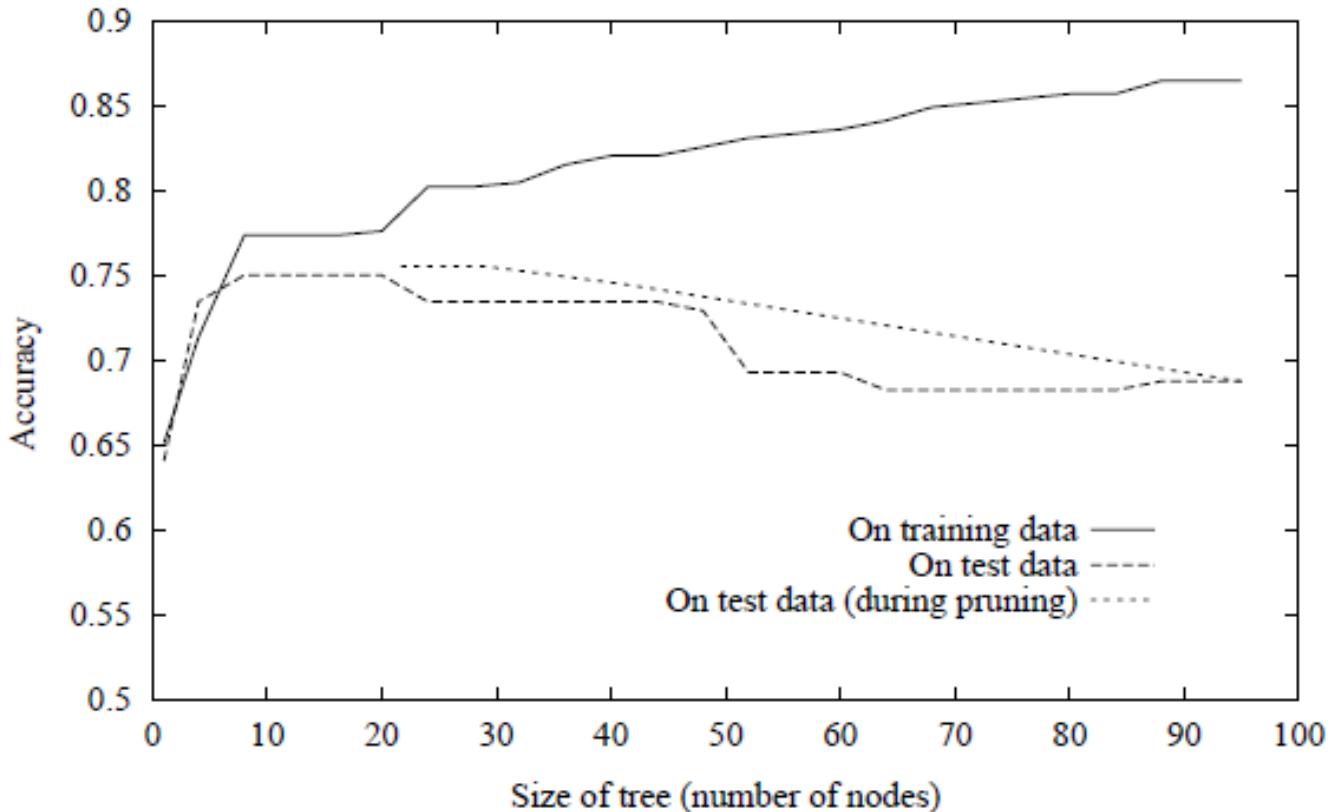
# Avoid Overfitting - Approach 2

- The Final Pruned Tree

Final tree:



# Effect of Reduced Error Pruning



- the accuracy increases over the test set as nodes are pruned from the tree.
- the validation set used for pruning is distinct from both the training and test sets.

# Ensemble Learning

- Suppose you are a movie director and you have created a short movie on a very important and interesting topic. Now, you want to take preliminary feedback (ratings) on the movie before making it public.
- **What are the possible ways by which you can do that?**
  1. *You may ask one of your friends to rate the movie for you.*
  2. *Another way could be by asking 5 colleagues of yours to rate the movie.*
  3. *How about asking 50 people to rate the movie?*
- Diverse group of people are likely to make better decisions as compared to individuals.  
**Similar is true for a diverse set of models in comparison to single models. This diversification in Machine Learning is achieved by a technique called Ensemble Learning.**

# Basic Ensemble Techniques

## (Max Voting, Averaging, Weighted average)

- **Max Voting:** **The max voting method is generally used for classification problems.** In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a ‘vote’. The predictions which we get from the majority of the models are used as the final prediction.
- **Eg.** when you asked 5 of your colleagues to rate your movie (out of 5); we’ll assume three of them rated it as 4 while two of them gave it a 5. Since the majority gave a rating of 4, the final rating will be taken as 4. **You can consider this as taking the mode of all the predictions.**

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
5	4	5	4	4	4

# Basic Ensemble Techniques

- **Averaging:** Multiple predictions are made for each data point in averaging. In this method, **we take an average of predictions from all the models and use it to make the final prediction.** Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.
- **Eg.** when you asked 5 of your colleagues to rate your movie (out of 5); we'll assume three of them rated it as 4 while two of them gave it a 5.

For example, in the below case, the averaging method would take the average of all the values.

i.e.  $(5+4+5+4+4)/5 = 4.4$

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
5	4	5	4	4	4.4

# Basic Ensemble Techniques

- **Weighted Average:** This is an **extension of the averaging method**. **All models are assigned different weights defining the importance of each model for prediction.** For instance, if two of your colleagues are critics, while others have no prior experience in this field, then the answers by these two friends are given more importance as compared to the other people.
- **Eg.** when you asked 5 of your colleagues to rate your movie (out of 5); we'll assume three of them rated it as 4 while two of them gave it a 5.

The result is calculated as  $[(5*0.23) + (4*0.23) + (5*0.18) + (4*0.18) + (4*0.18)] = 4.41$ .

	Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
weight	0.23	0.23	0.18	0.18	0.18	
rating	5	4	5	4	4	4.41

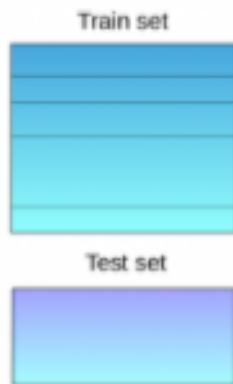
# Advanced Ensemble Techniques

## (Stacking, Blending, Bagging, Boosting)

- **Stacking:** It uses predictions from multiple to build a new model. **This model is used for making predictions on the test set.**

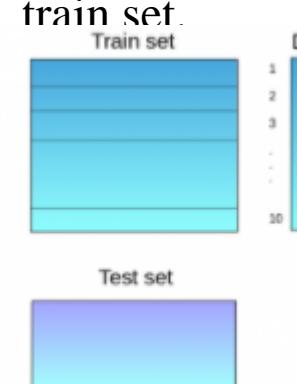
### Step 1:

First Data is divided into train and test set and the train set is further divided into 10 smaller parts.



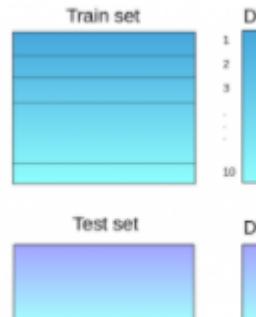
### Step 2:

A base mode say: DT is fitted on 9 parts and predictions are made on 10<sup>th</sup> part. The same process is repeated for each part of train set.



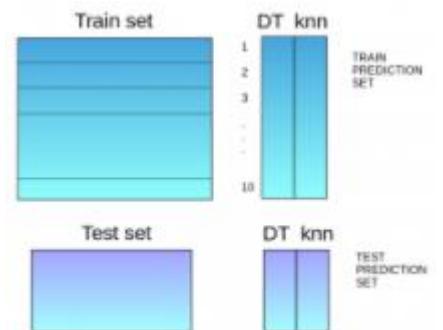
### Step 3:

Using this model fitted on the whole training set, predictions are made on the test data.



### Step 4:

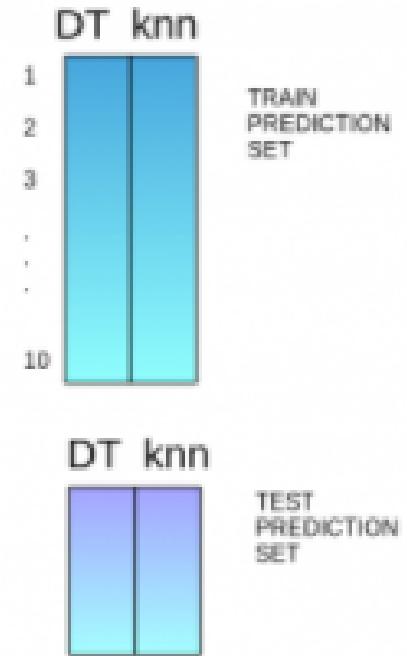
Steps 2 & 3 are repeated for another base model: Say KNN



# Advanced Ensemble Techniques

## Step 5:

The predictions from the train set are used as features to build a new model.



## Step 6:

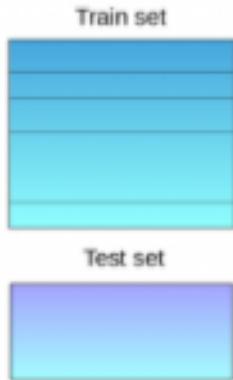
This model is used to make final predictions on the test prediction set.

# Advanced Ensemble Techniques

- **Blending:** follows the same approach as stacking but uses only a holdout (validation) set from the train set to make predictions. In other words, unlike stacking, the predictions are made on the holdout set only. The holdout set and the predictions are used to build a model which is run on the test set.

## Step 1:

First Data is divided into train and test set.



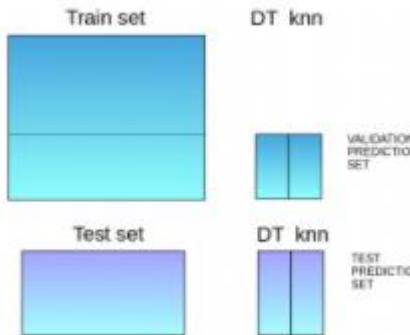
## Step 2:

The train set is split into training and validation sets.



## Step 3:

Base models are fitted on the training set and predictions are made on the validation set and test set.

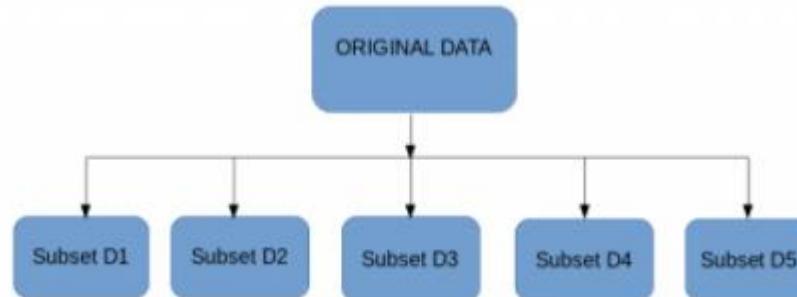


## Step 4:

The validation set and its predictions are used as features to build a new model and this model is used to make final predictions on the test features.

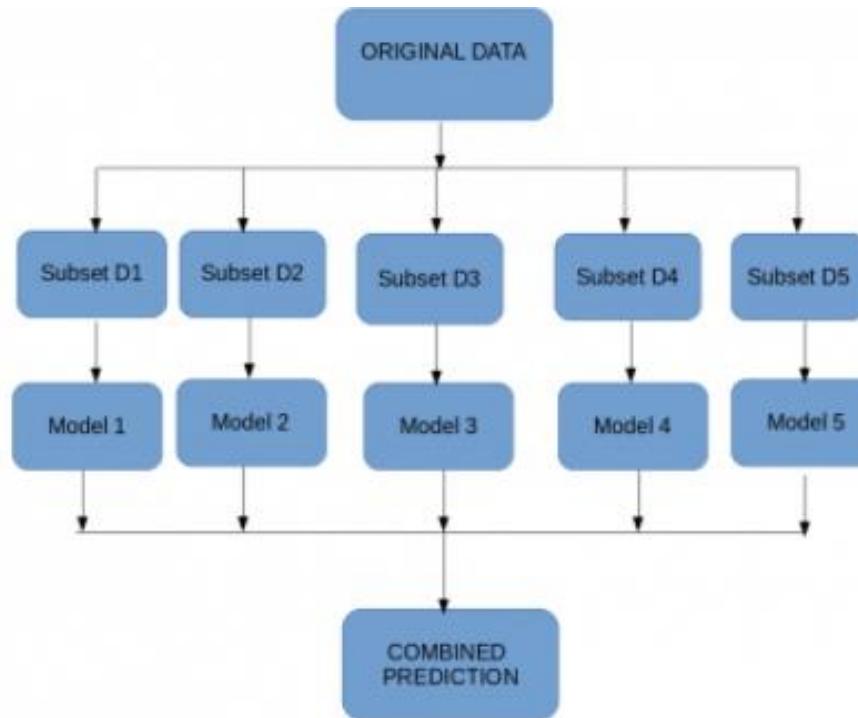
# Advanced Ensemble Techniques

- **Bagging:** The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result.
- ? If we **create all the models on the same set of data and combine it**, will it be useful? There is a high chance that these models will give the **same result since they are getting the same input**. So how can we solve this problem? One of the techniques is **bootstrapping**.
- In Bootstrapping, **some subsets of observations are created** from the original dataset, with replacement. **Bagging /Bootstrap Aggregating 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.



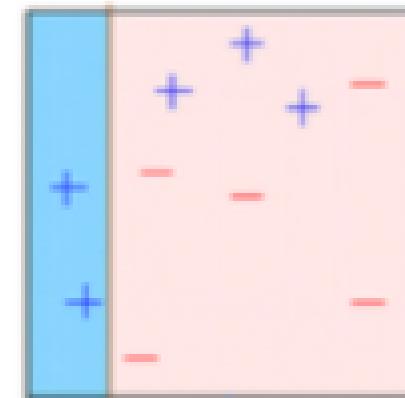
# Advanced Ensemble Techniques

- 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.



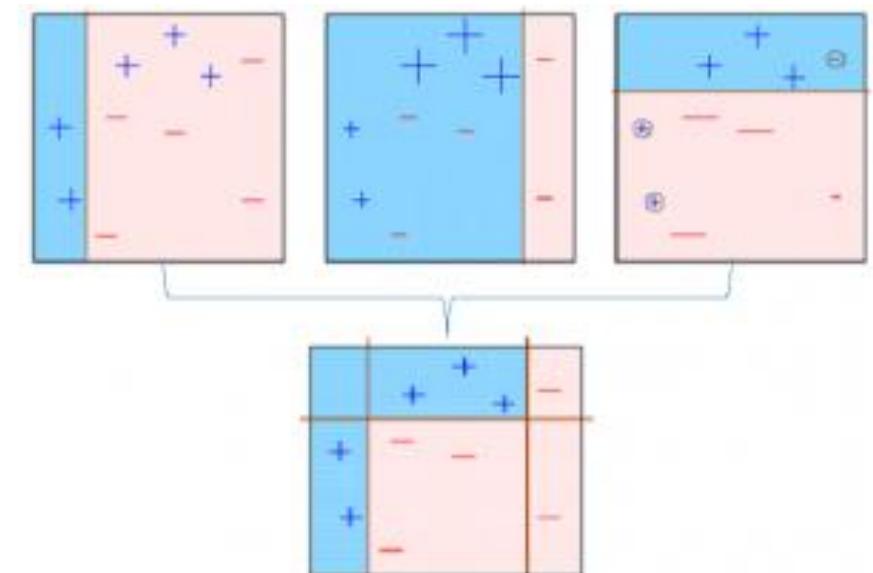
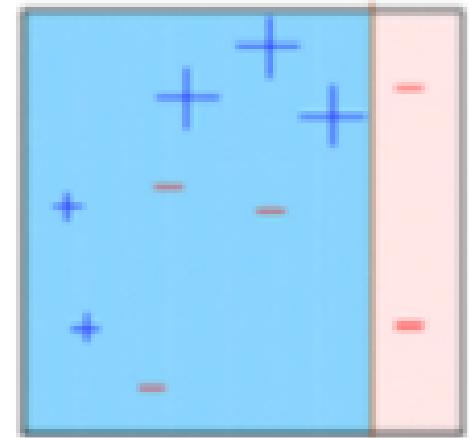
# Advanced Ensemble Techniques

- 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:** 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.
  - 1: A subset is created from the original dataset.
  - 2: Initially, all data points are given equal weights.
  - 3: A base model is created on this subset.
  - 4: This model is used to make predictions on the whole dataset.



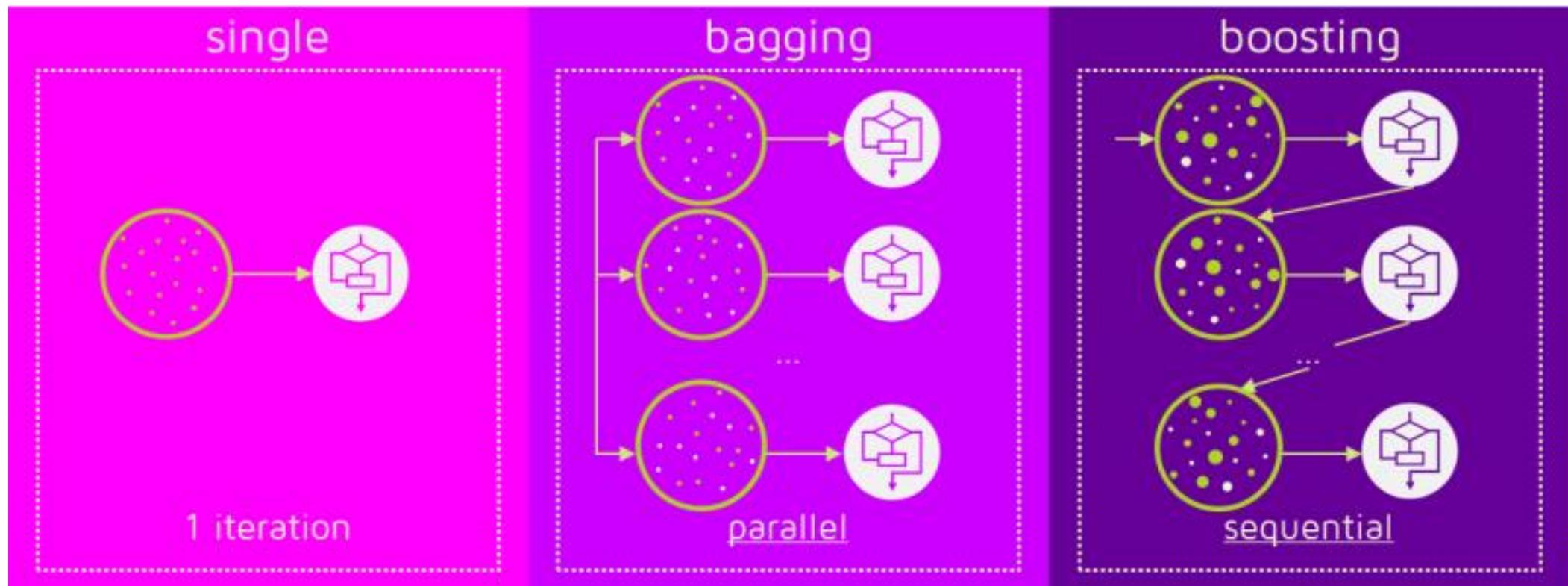
# Advanced Ensemble Techniques

5. Errors are calculated using the actual values and predicted values.
  6. The observations which are incorrectly predicted, are given higher weights.  
(Here, the three misclassified blue-plus points will be given higher weights)
  7. Another model is created and predictions are made on the dataset.  
(This model tries to correct the errors from the previous model)
- 
8. Similarly, multiple models are created, each correcting the errors of the previous model.
  9. The final model (strong learner) is the weighted mean of all the models (weak learners)



Thus, the **boosting algorithm combines a number of weak learners to form a strong learner**. The 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**.

# Advanced Ensemble Techniques

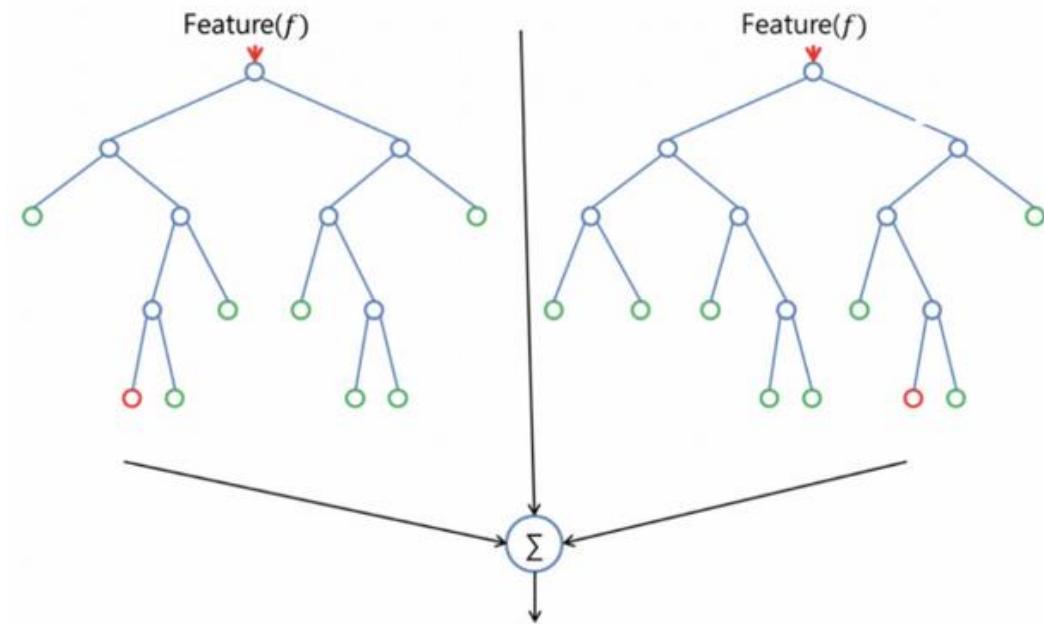


# Algorithms based on Bagging Boosting

- **Bagging and Boosting** are two of the most commonly used techniques in machine learning.
- **Bagging algorithms:**
  - Bagging meta-estimator
  - Random forest
- **Boosting algorithms:**
  - AdaBoost
  - GBM
  - XGBM
  - Light GBM
  - CatBoost

# Random Forest (Decision Tree Forests)

- Random forest is a supervised learning algorithm. **The "forest" it builds, is an ensemble of decision trees**, usually trained with the **"bagging" method**. The general idea of the bagging method is that a combination of learning models increases the overall result.
- **Working Rule: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.**
- There is a **direct relationship between the number of trees in the forest and the results it can get: the larger the number of trees, the more accurate the result.**



# Random Forest

- In a **decision tree** each internal node represents a 'test' on an attribute, **each branch represents the outcome of the test**, and **each leaf node represents a class label** (decision taken after computing all attributes). A node that has no children is a leaf.
- **The output is**
  - the **mode of the predicted classes** ( in case of classification), and
  - the **mean of the prediction value** ( in case of regression) provided by the individual trees.

# How Random Forest works?

- There are **two stages in Random Forest algorithm**, one is **random forest creation**, the other is to **make a prediction from the random forest classifier** created in the first stage.

## Algorithm:

1. **Bagging:** From the full dataset, take a sample , generate a tree and obtain predictions. Repeat with a different sample, from the same dataset. The new tree will typically make different predictions.
2. Each tree is built by considering **only a random subset of the features** for splitting at the nodes. If  $d$  is the **dimensionality of the data**,  $d_1$  features are chosen at random where  $d_1 < d$ .
3. By using the above procedure, **a number of decision trees are built leading to a decision tree forest**.
4. A new pattern is assigned a **class label by using all the decision trees** in the forest, and combined to give the final class label.

**Note: The selection of a random subset of features for each decision tree makes this a random subspace method. This prevents Overfitting.**

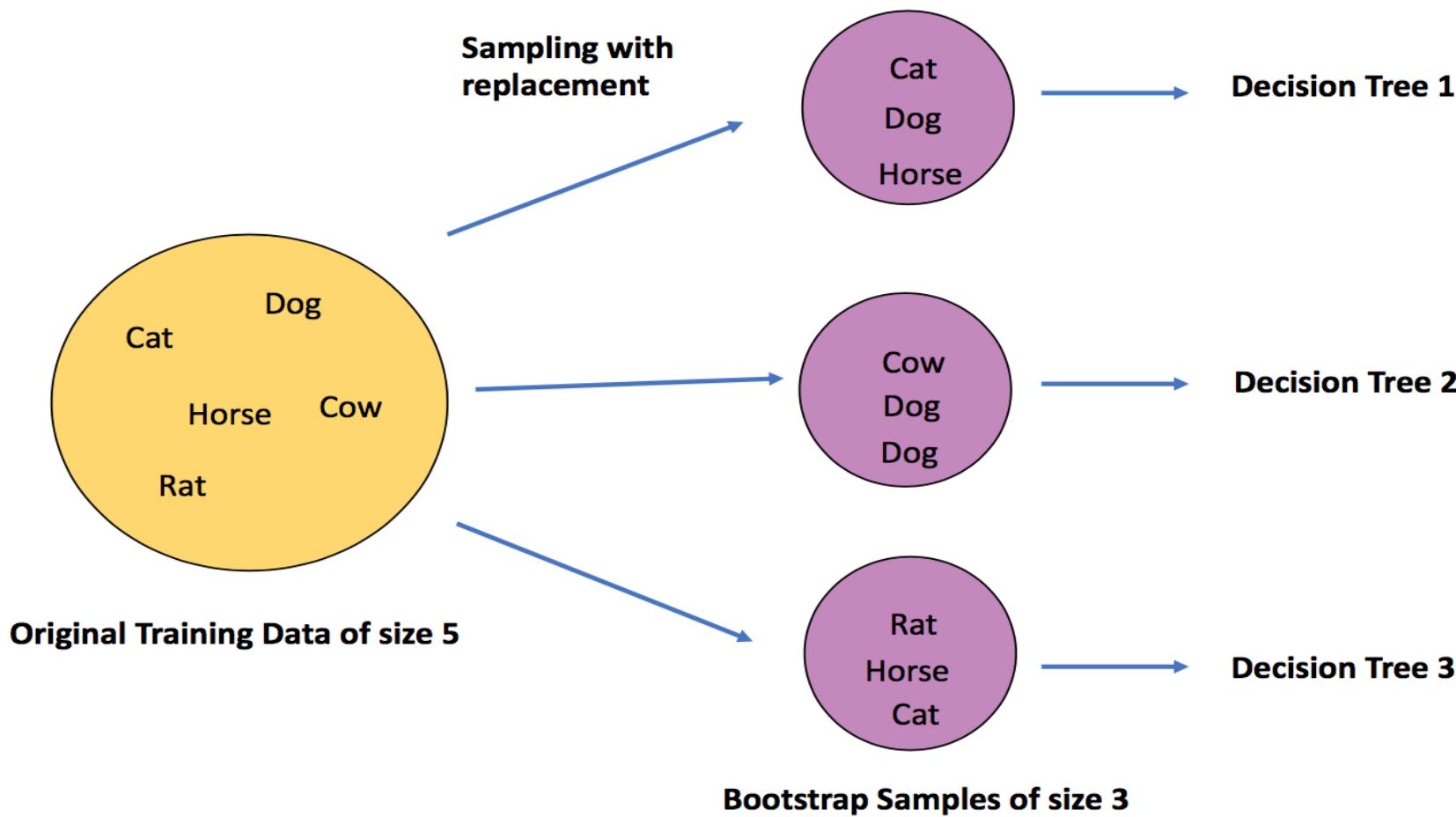
# How Random Forest works?

- In Random forests, **1/3<sup>rd</sup> patterns are left out for building a decision tree** (which are called the “out of bag” (**OOB**) samples) can be used as a test sample for that tree. By using the **OOB samples**, the error rate of prediction for each tree is computed. **The average error rate for all the trees gives the generalization error of the decision tree forest.**
- Every tree grown has a different holdout sample. Every record in the full dataset is “in bag” for some trees(about 2/3<sup>rd</sup>) and “out of bag” for the other trees.
- **Some of the choices to be made for building a random forest are as follows:**
  1. The forest size or the number of decision trees F.
  2. Type of decision to be made at each node: 1) Using a single feature, 2) linear combinations of some features (**oblique split**) or 3) **multivariate split** (nonlinear combination of some features).
  3. Parameter to be used to choose the **best feature** to split at a node. Generally, information gain or gini index is used.

# Types of Sampling

- **Simple random sampling**
  - There is an equal probability of selecting any particular item
- **Sampling without replacement**
  - Once an object is selected, it is removed from the population
- **Sampling with replacement**
  - A selected object is not removed from the population
- **Stratified sampling:**
  - Partition the data set, and draw samples from each partition (proportionally, i.e., approximately the same percentage of the data)

# ‘Out of Bag’ (OOB) Data example



Generation of bootstrap samples with replacement. “Sampling-with-replacement” here means that if a data point is chosen in the first random draw it still remains in the original sample for choosing in another random draw that may follow with an equal probability. This can be seen in the image above as “Dog” is chosen twice in the second bootstrap sample.

# ‘Out of Bag’ (OOB) Data example

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Weak	Yes
Windy	Cold	Low	Weak	Yes

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Weak	Yes
Windy	Cold	Low	Weak	Yes

Bootstrap sample

# ‘Out of Bag’ (OOB) Data example

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Weak	Yes
Windy	Cold	Low	Weak	Yes

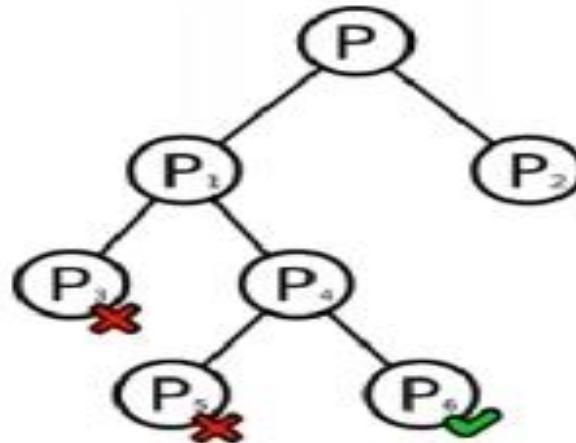
Out of Bag sample

Decision Tree	Prediction
1	YES
3	NO
5	YES
Majority vote : YES	

# Random Forest

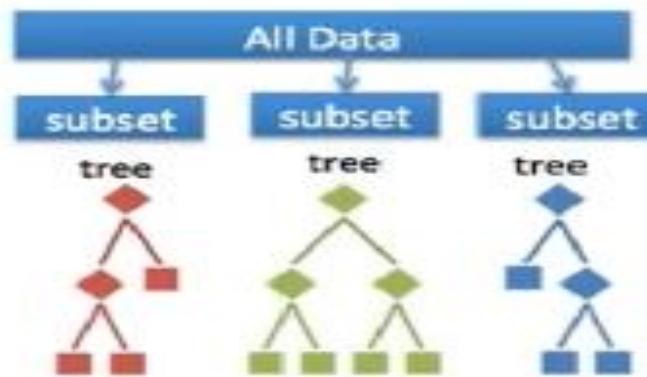
## Decision Tree

- Decision tree builds classification models in the form of a tree structure.
- It breaks down a dataset into smaller and smaller subsets.



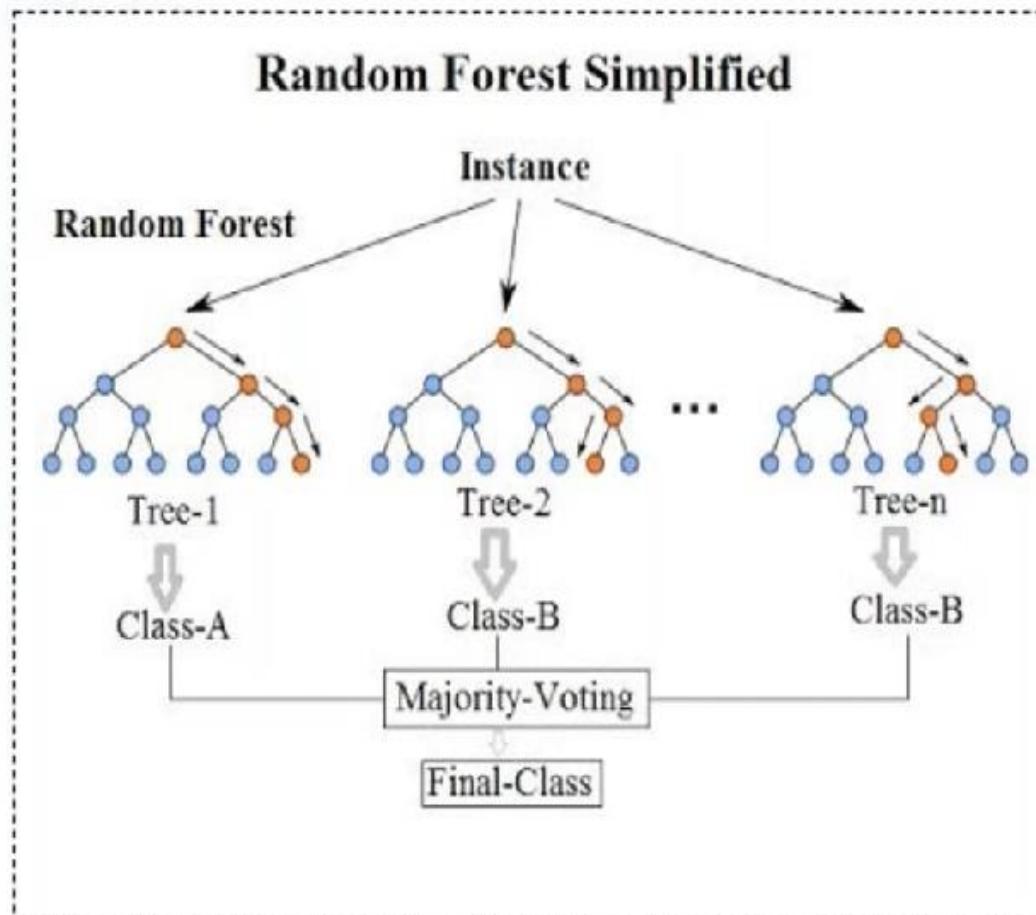
## Random Forest

- Random Forest is an ensemble classifier made using many decision tree models.
- Ensemble models combine the results from different models.



# Random Forest

- Random Forest - a versatile algorithm capable of performing both
  - i) Regression
  - ii) Classification
- It is a type of ensemble learning method
- Commonly used predictive modelling and machine learning technique



# Random Forest Algorithm

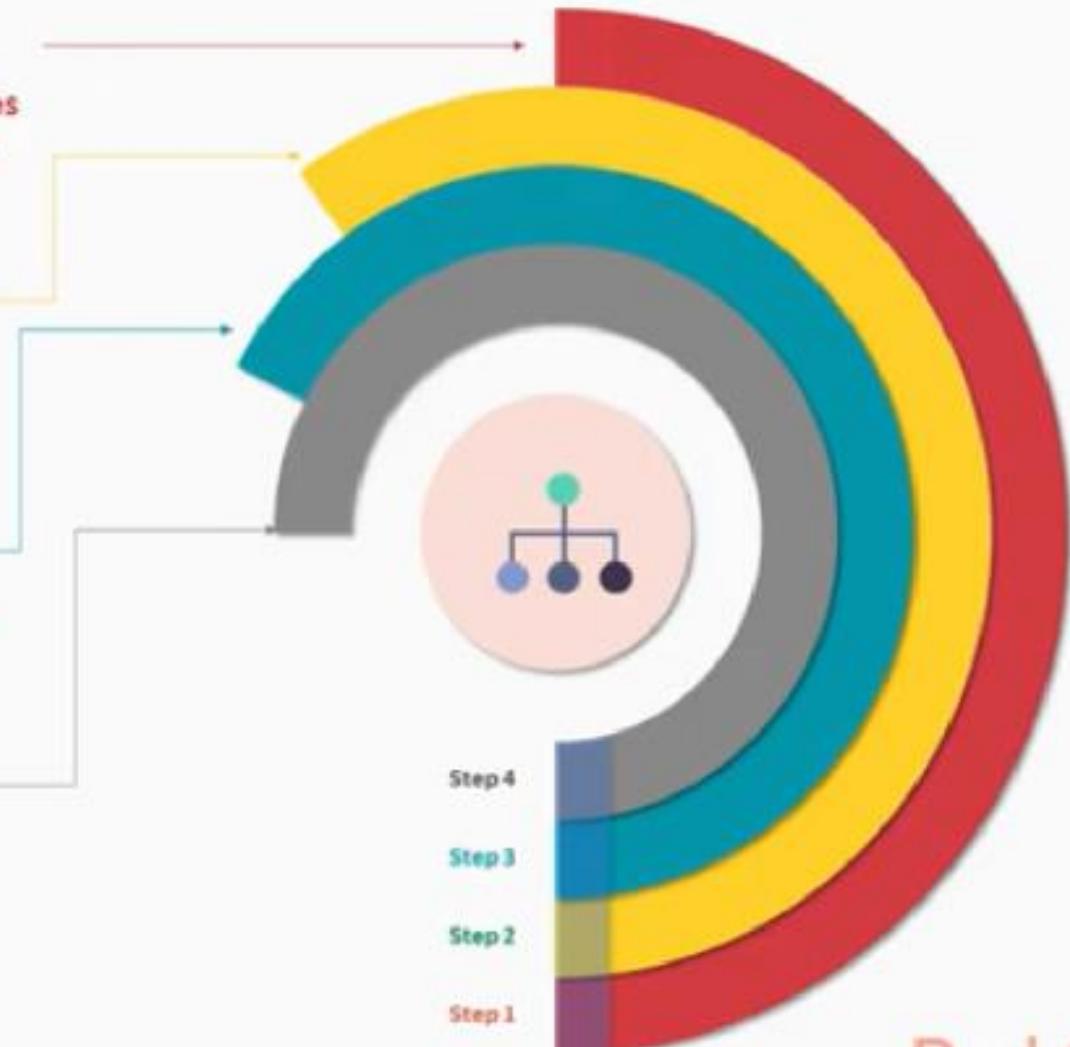
From a very high level, RF algorithm goes through four major steps to build the tree:

Step 1  
Selects k features (columns) from the dataset (table) with a total of m features randomly (where  $k < m$ ). Then, it builds Decision Tree from those k features.

Step 2  
Repeats 'n' times so that you have 'n' Decision Trees built from different random combinations of k features (or a different random sample of the data, called bootstrap sample).

Step 3  
Takes each of the n built Decision Trees and passes a random variable to predict the outcome. Stores the predicted outcome (target), so that you have a total of n outcomes from the n Decision Trees.

Step 4  
Calculates the votes for each predicted target and takes the mode (most frequent target variable). In other words, considers the high voted predicted target as the final prediction from the random forest algorithm.



# ADVANTAGES OF RANDOM FORESTS

- Automatic identification of important predictors
- Each decision tree is independent. Therefore trees can be grown on different cores or different computers, allowing for quicker analysis.
- It can be used for both classification and regression tasks.
- In Random Forest algorithm, if there are enough trees in the forest, the classifier won't overfit the model.
- The Random Forest can handle missing values,
- 4) The Random Forest classifier can be modelled for categorical values.

# SHORTCOMINGS OF RANDOM FORESTS

- Suited for wide datasets with only a **moderate number of rows**. Breiman recommends the use of other tools for larger datasets.
- Large memory needed to store built models.
- Overfitting might be seen with noisy data.

# APPLICATIONS OF RANDOM FORESTS

- Online targeted marketing
- Credit card fraud detection
- Text analytics
- Credit risk and insurance risk
- Retail Sales prediction
- Biological & Medical Research
- Manufacturing Quality Control

**Course code:** CSE3008

**Course Title:** Introduction to Machine Learning

**Module 2: Training Models**

**Dr. Usha Rani Gogoi**

**Assistant Professor Sr. Grade 1**

**[usha.gogoi@vitap.ac.in](mailto:usha.gogoi@vitap.ac.in)**

**SCOPE**

# What & Why

- **What is Regression?**

Formulation of a functional relationship between a set of independent or Explanatory variables (X's) with a Dependent or Response variable (Y).

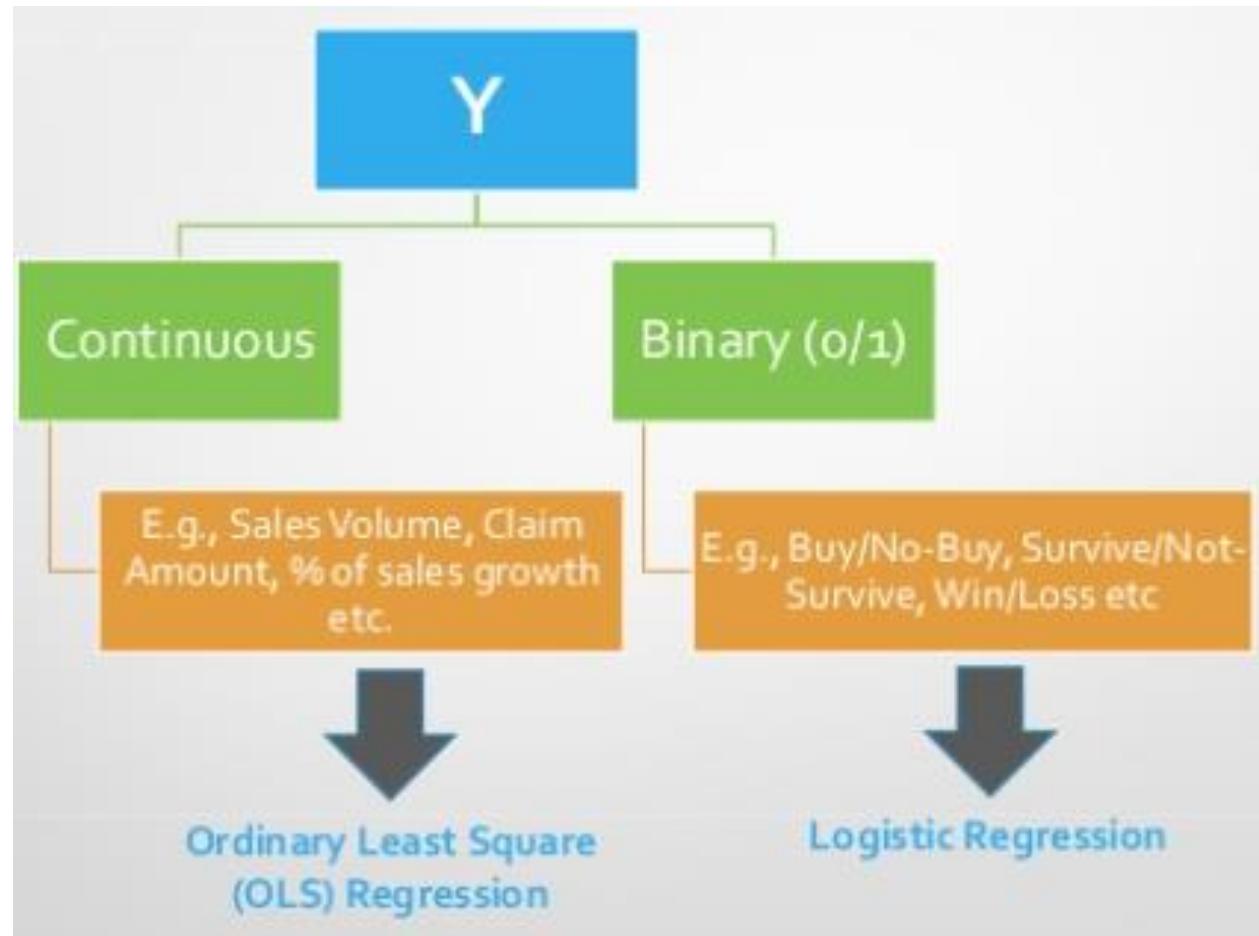
$$Y = f(x)$$

- **Why Regression?**

Knowledge of Y is crucial for decision making.

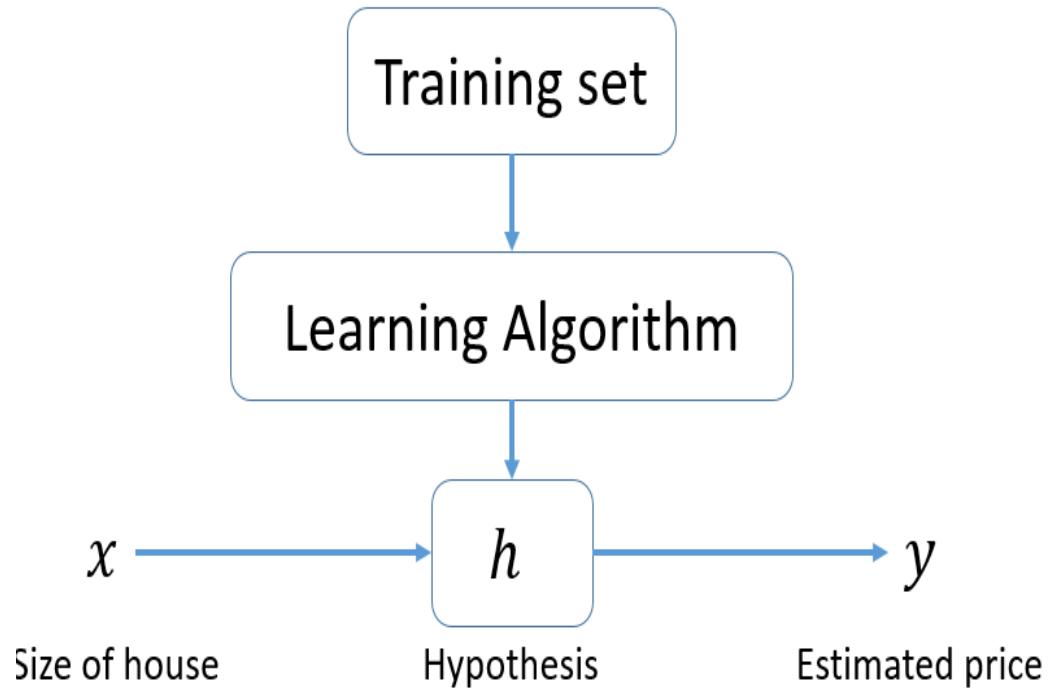
- Will he/she buy or not?
- Shall I offer him/ her the loan or not

# Types of Regression



# Linear Regression

- Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It's used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog). There are two main types: **Simple Regression and Multivariable Regression**.



# Simple Linear Regression (Univariate LR)

- Only one independent variable / descriptor.
- The model is represented with the equation of straight line

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

where,  $\theta_0$  and  $\theta_1$  are coefficients of Regression and  
 $\theta_0$  is the  $y$  – intercept or Zero Condition  
and  $\theta_1$  is the slope or Gradient of the line

- So, in **Linear regression, the model “learns” (read: estimates) two parameters: the intercept and the slope**. The intercept is the level of  $y$  when  $X$  is 0 (i.e. the price of house with 0 square feet is?) and the slope is the rate of predicted increase or decrease in  $y$  for each unit increase in  $X$  (i.e. how much do the price increase per unit increase in square feet).
- Once the model **learns these parameters they** can be used to compute estimated values of  $y$  given new values of  $X$ .

# Training set

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

$m = 47$

- **Notation:**

- $m$  = Number of training examples
- $x$  = Input variable / features
- $y$  = Output variable / target variable
- $(x, y)$  = One training example
- $(x^{(i)}, y^{(i)})$  =  $i^{th}$  training example

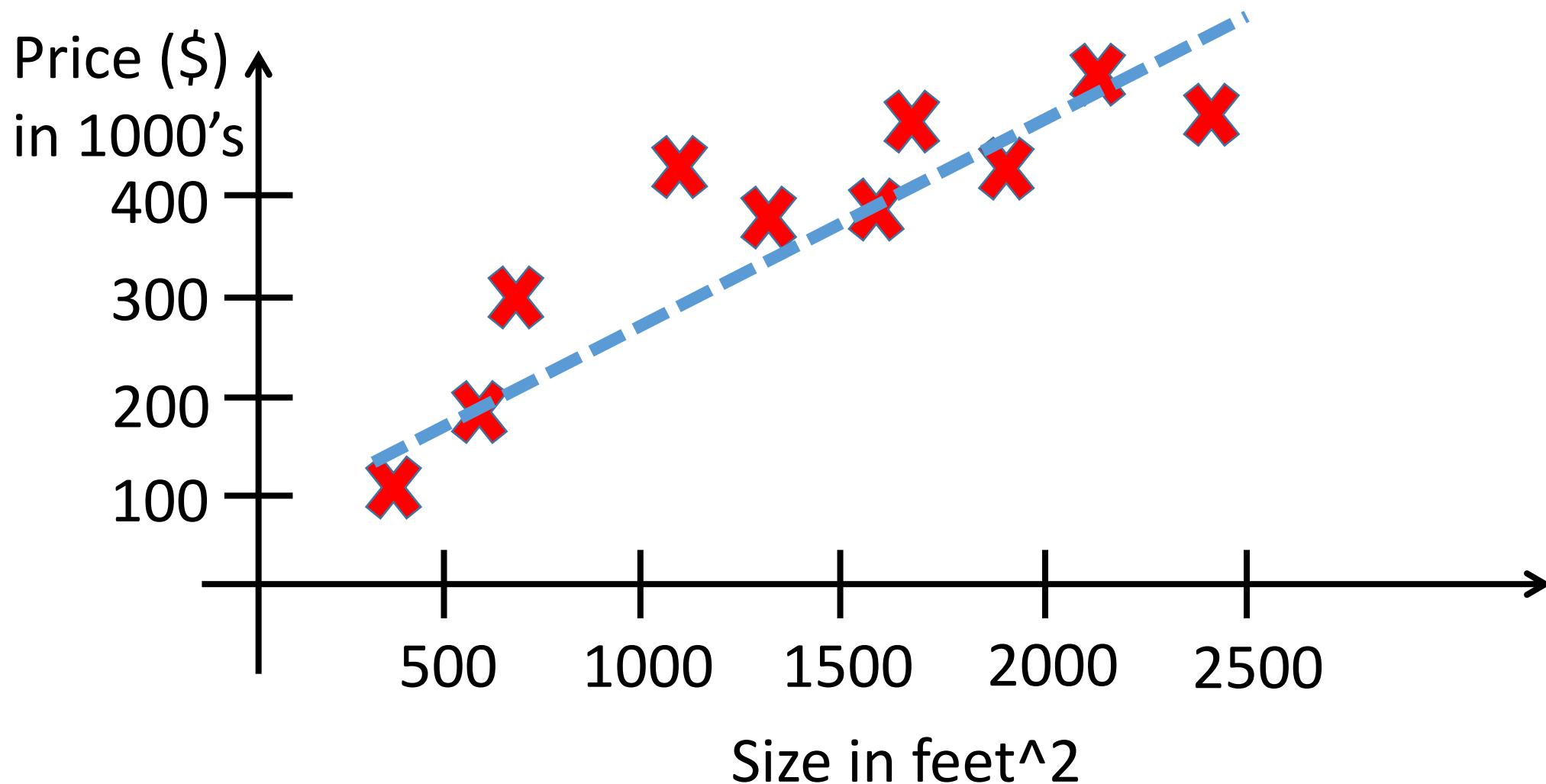
- **Examples:**

$$x^{(1)} = 2104$$

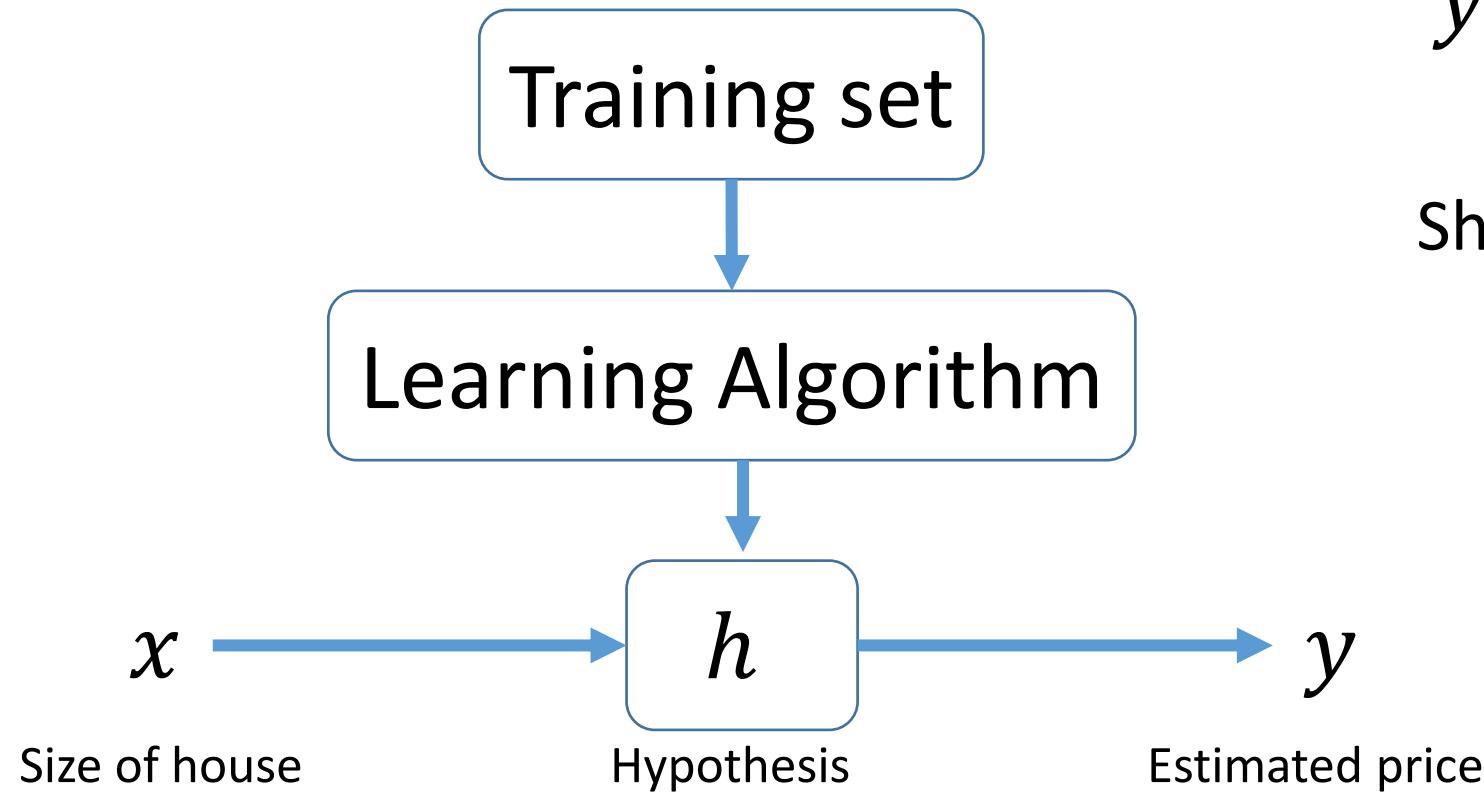
$$x^{(2)} = 1416$$

$$y^{(1)} = 460$$

# House pricing prediction

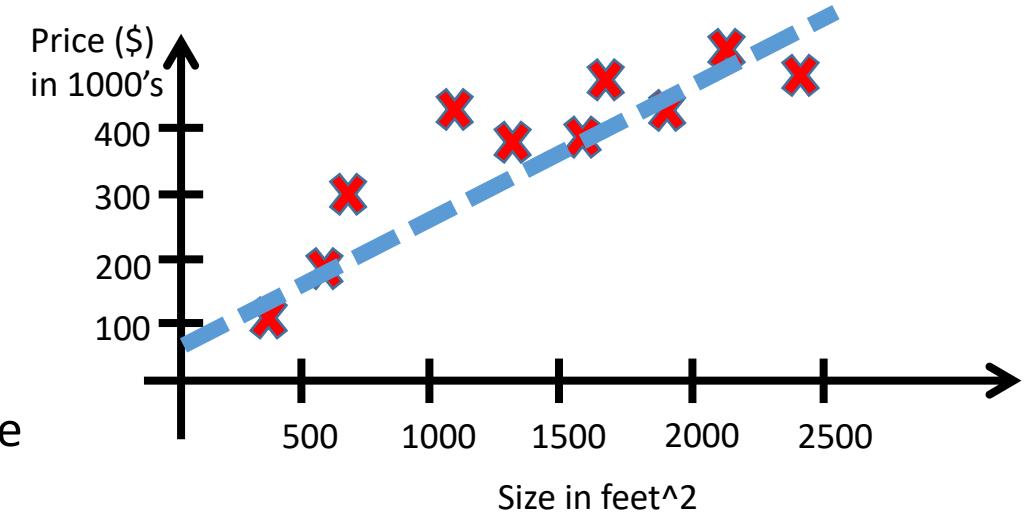


# Model Representation



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

Shorthand  $h(x)$



Univariate linear regression

## Training Set

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

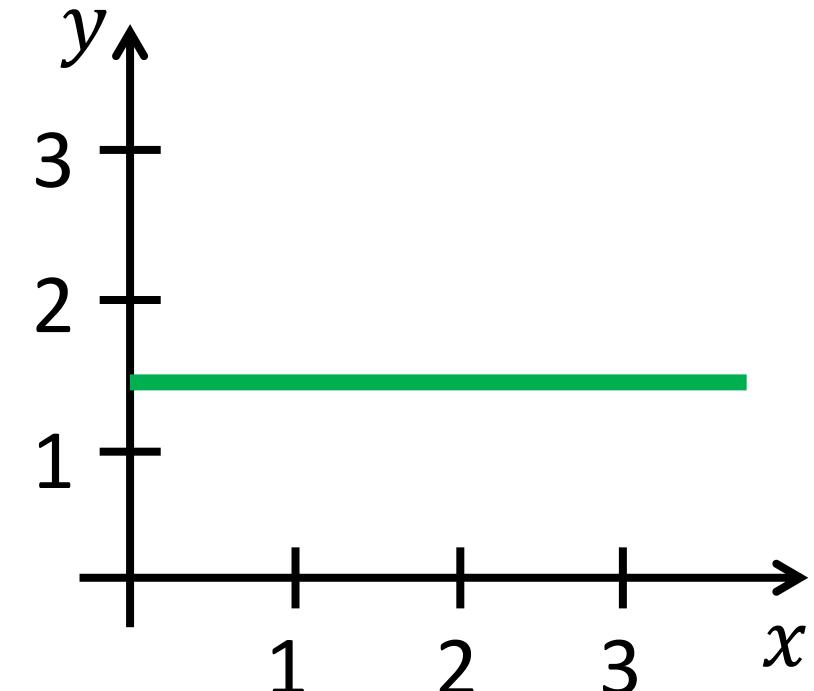
$m = 47$

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

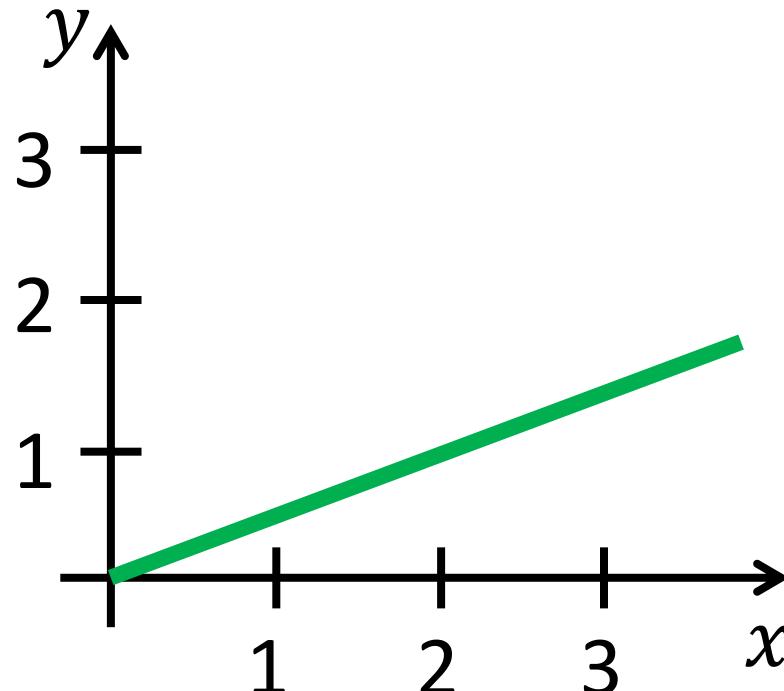
$\theta_0, \theta_1$ : parameters/weights

How to choose  $\theta_i$ 's?

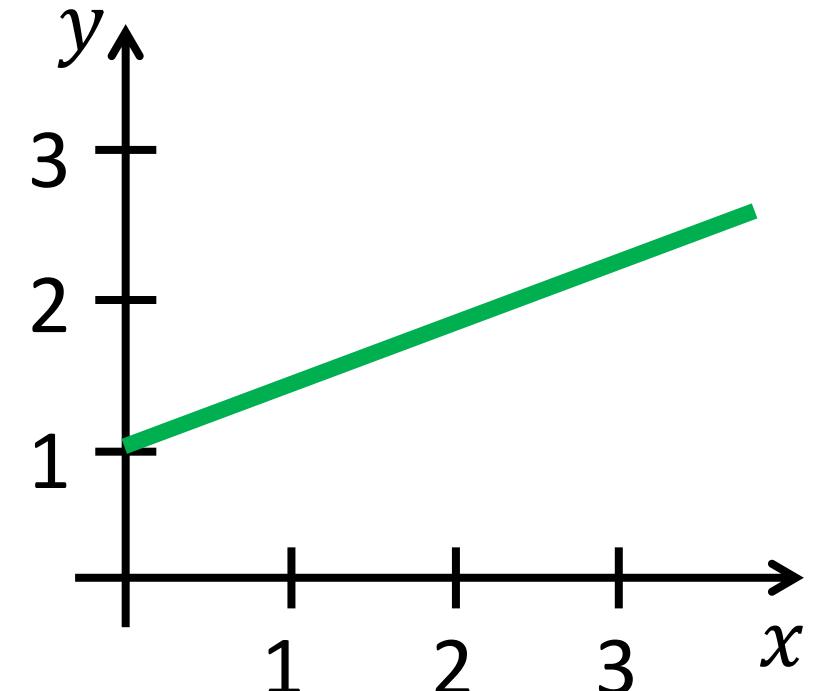
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



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



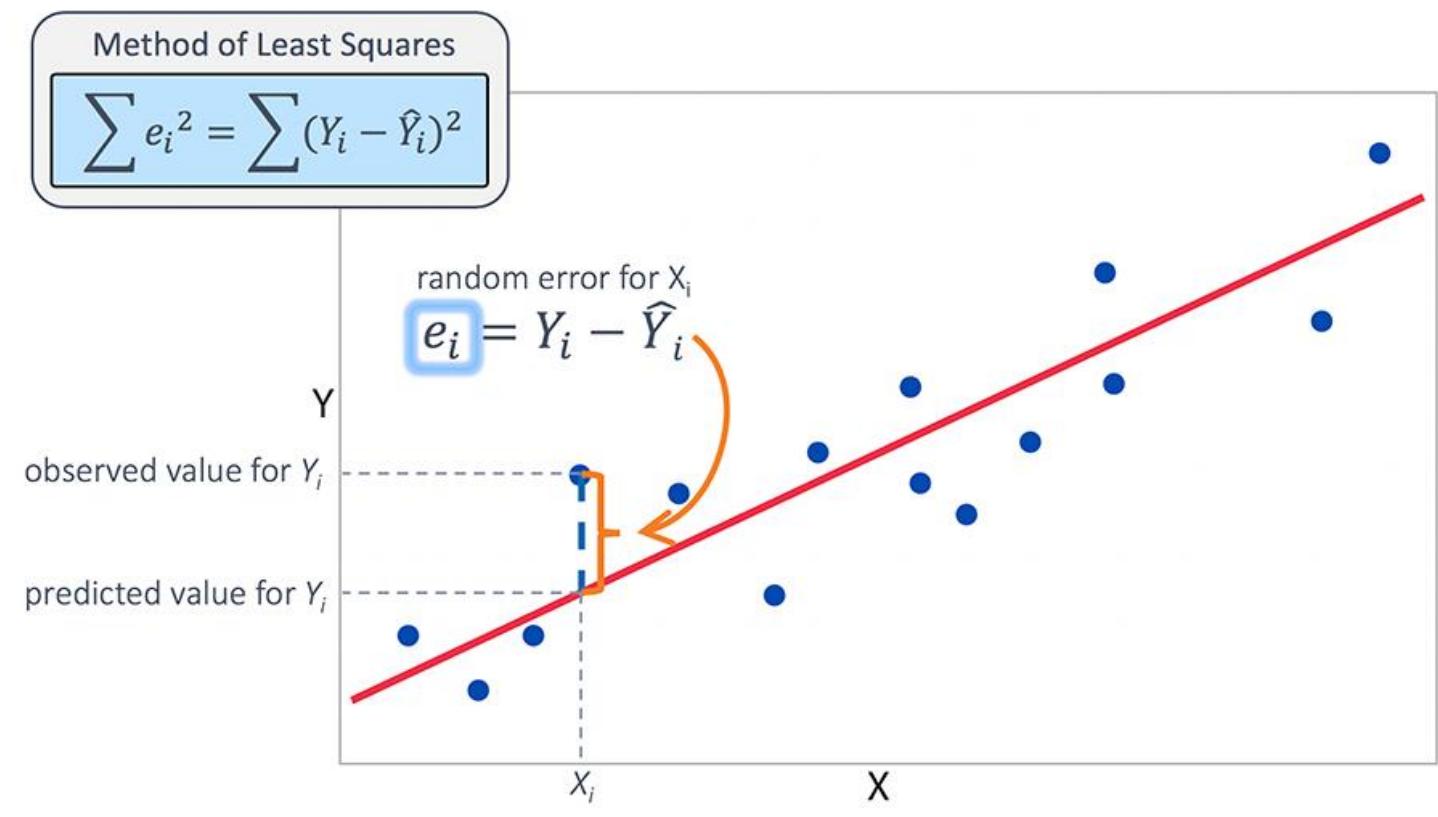
$$\begin{aligned}\theta_0 &= 0 \\ \theta_1 &= 0.5\end{aligned}$$



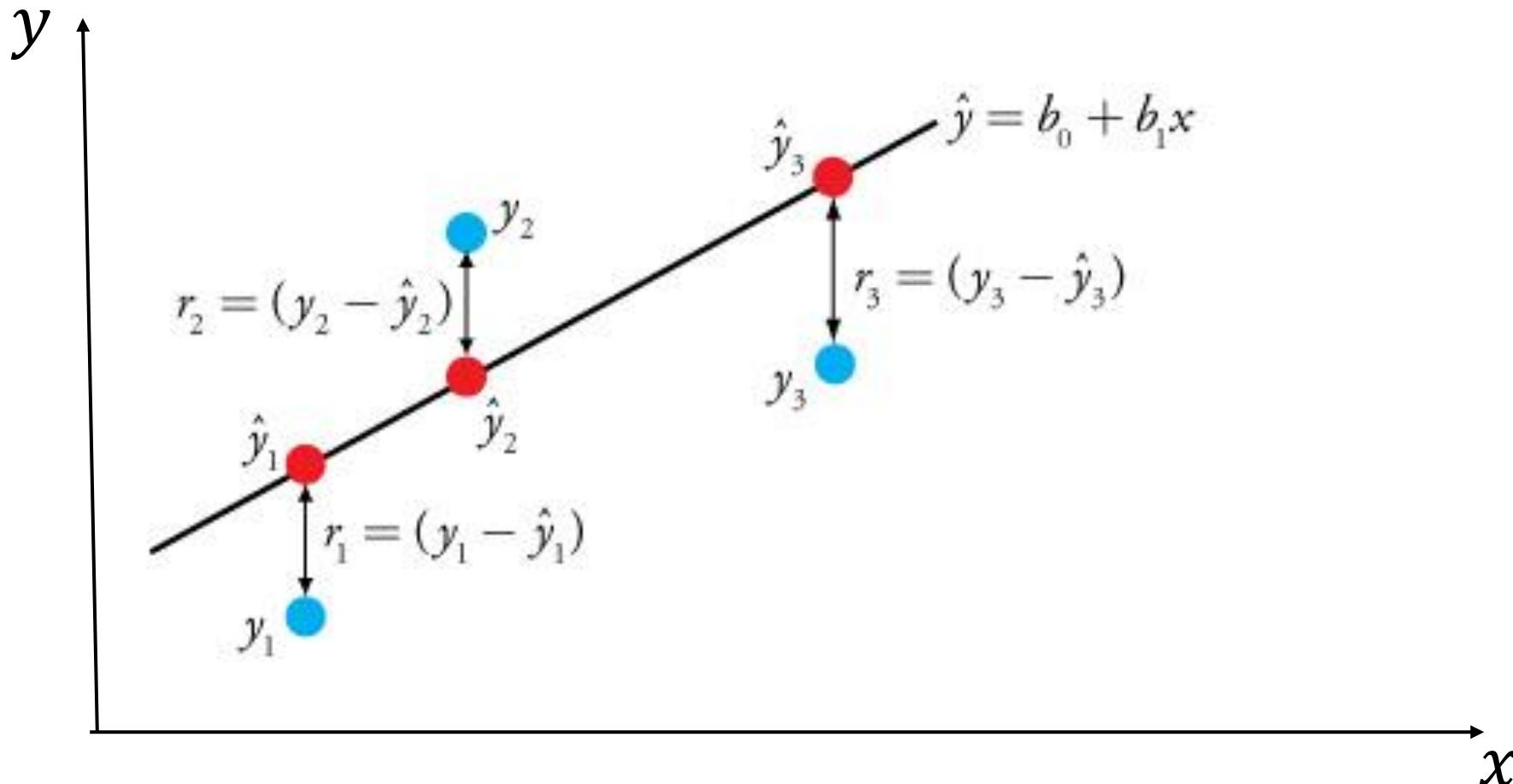
$$\begin{aligned}\theta_0 &= 1 \\ \theta_1 &= 0.5\end{aligned}$$

# Cost Function

- **It helps the learner to correct / change behaviour to minimize mistakes.**
- In ML, cost functions are used to estimate how badly models are performing.
- Hence, cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between X and y.
- This is typically expressed as a difference or distance between the predicted value and the actual value.



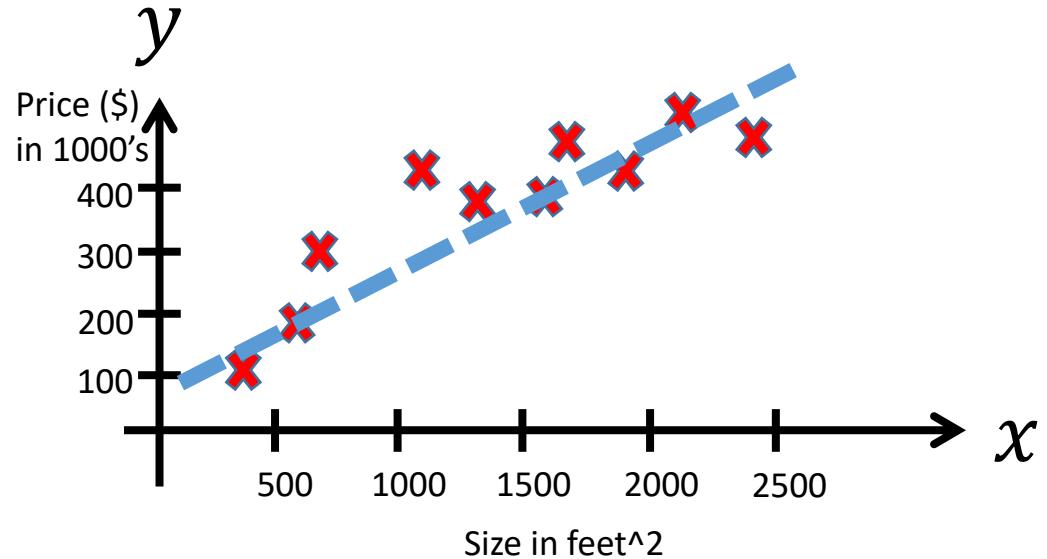
# Cost Function



A lowest cost is desirable. A low costs represents a smaller difference. By minimizing the cost, we are finding the best fit.

# Cost function

- Idea: Choose  $\theta_0, \theta_1$  so that  $h_\theta(x)$  (Predicted value) is close to  $y$  (actual value) for our training example  $(x, y)$



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

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

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

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

# Analysis of Cost function

- Hypothesis:

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



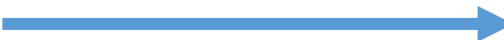
# Simplified

- Hypothesis:

$$h_{\theta}(x) = \theta_1 x \quad \theta_0 = 0$$

- Parameters:

$$\theta_0, \theta_1$$



- Parameters:

$$\theta_1$$

- Cost function:

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

- Cost function:

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

- Goal:

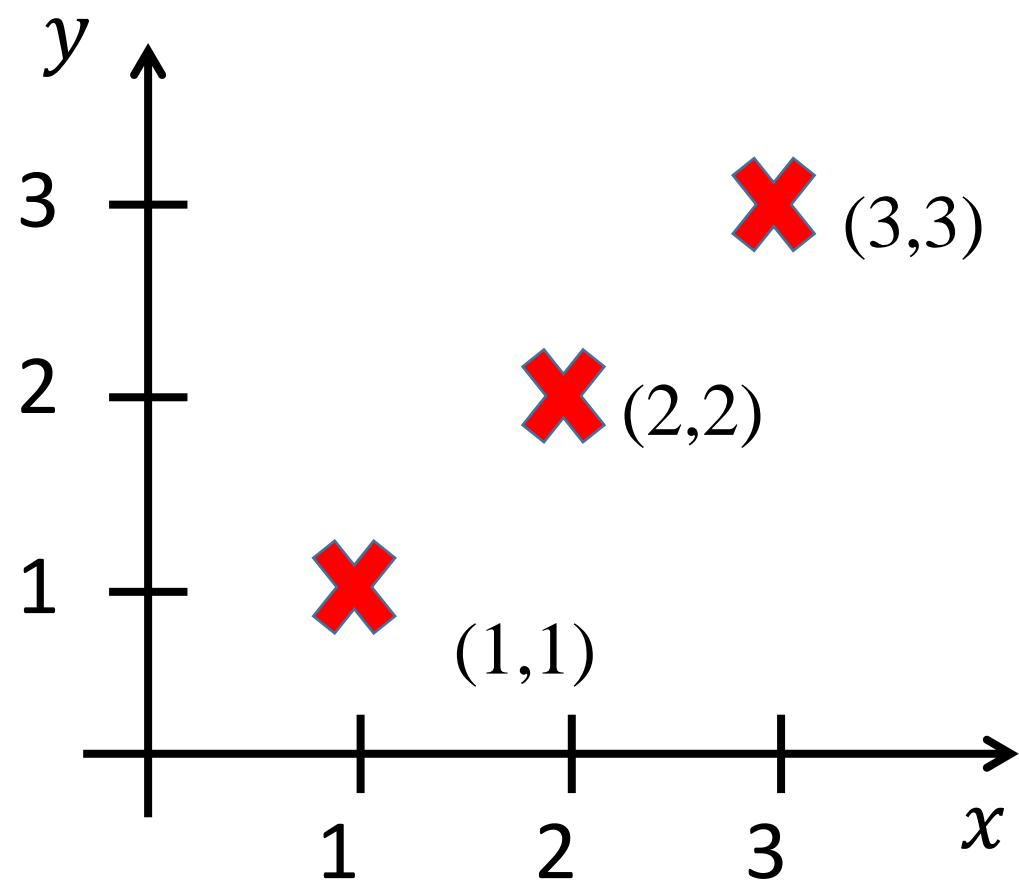
$$\text{minimize } J(\theta_0, \theta_1)$$
$$\theta_0, \theta_1$$



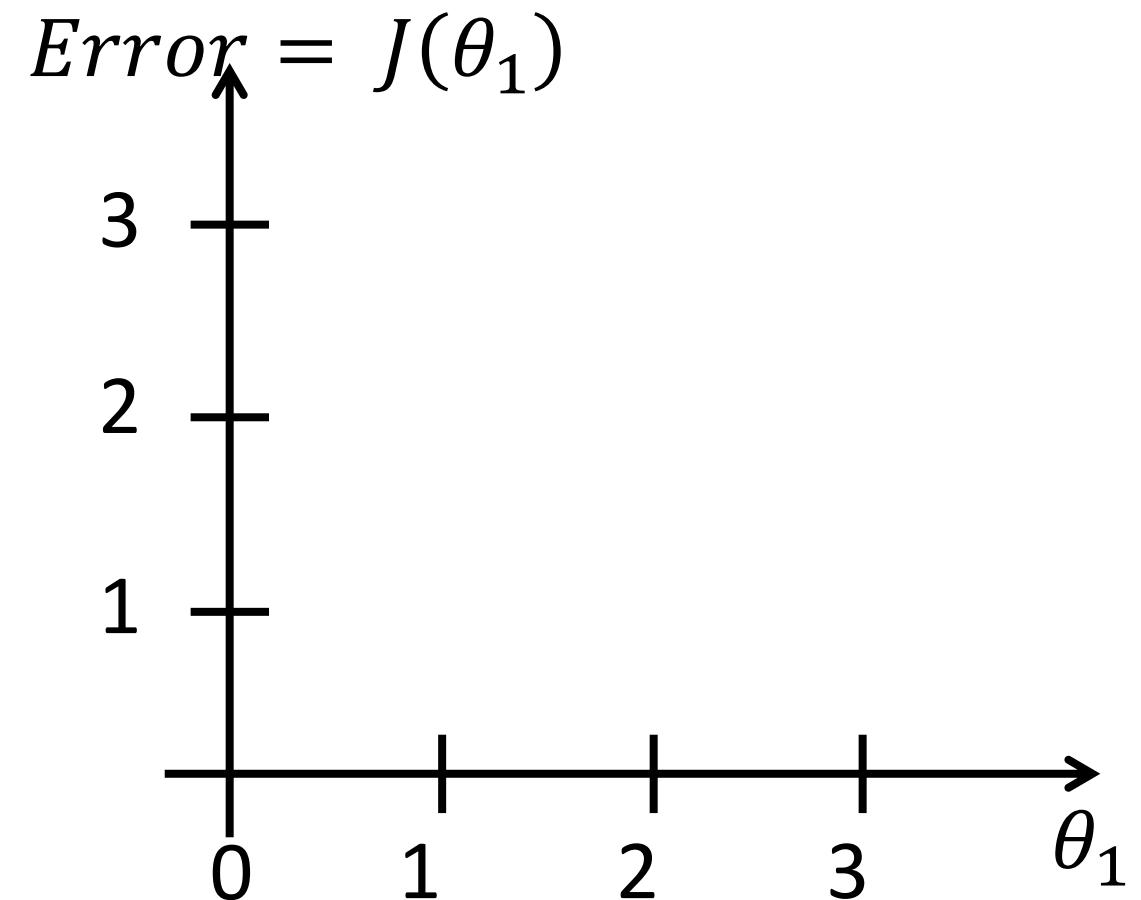
- Goal:

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

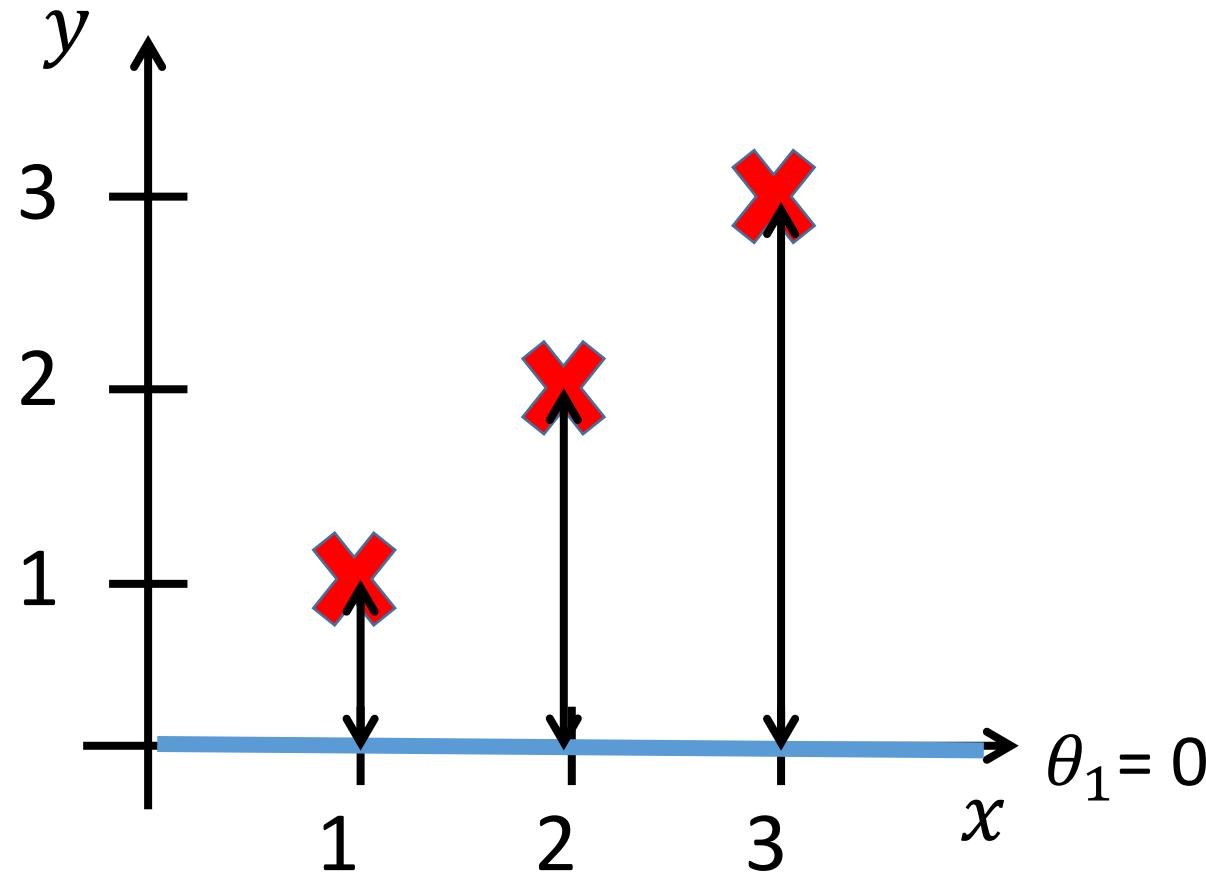
$h_{\theta_1}(x)$ , function of  $x$



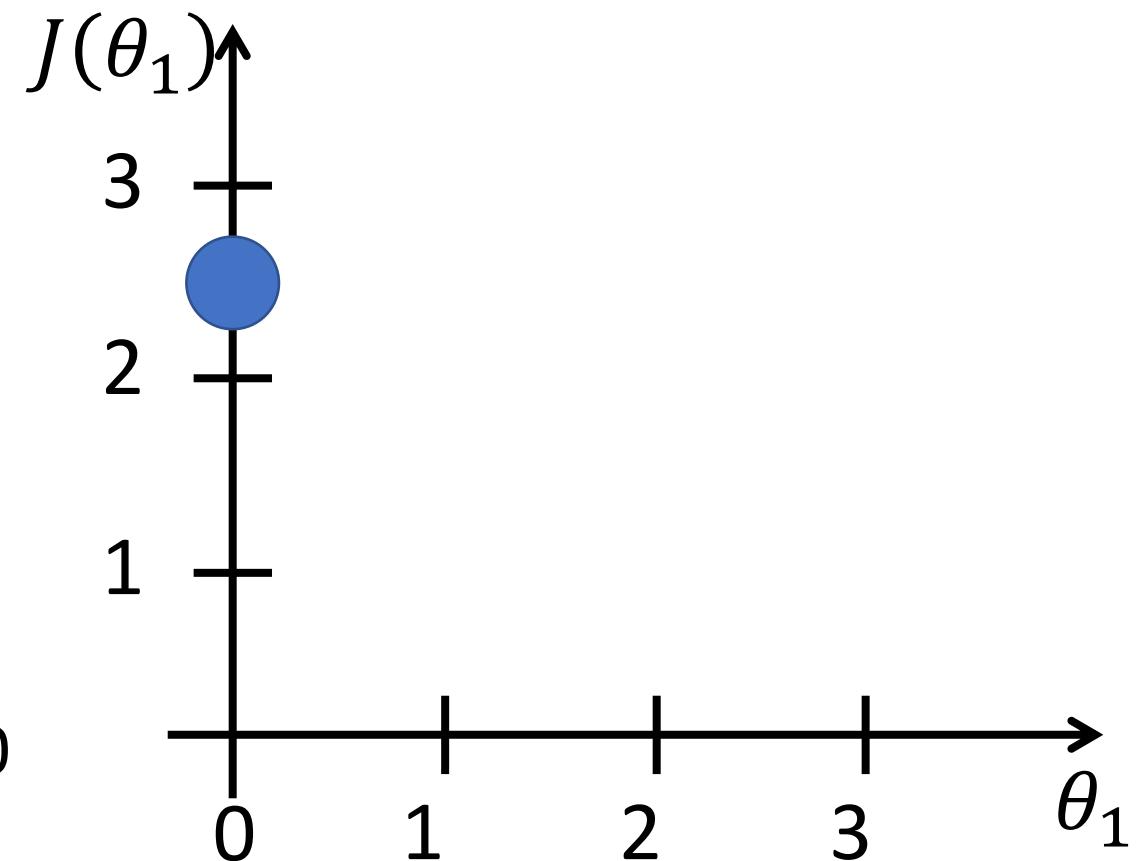
$J(\theta_1)$ , function of  $\theta_1$



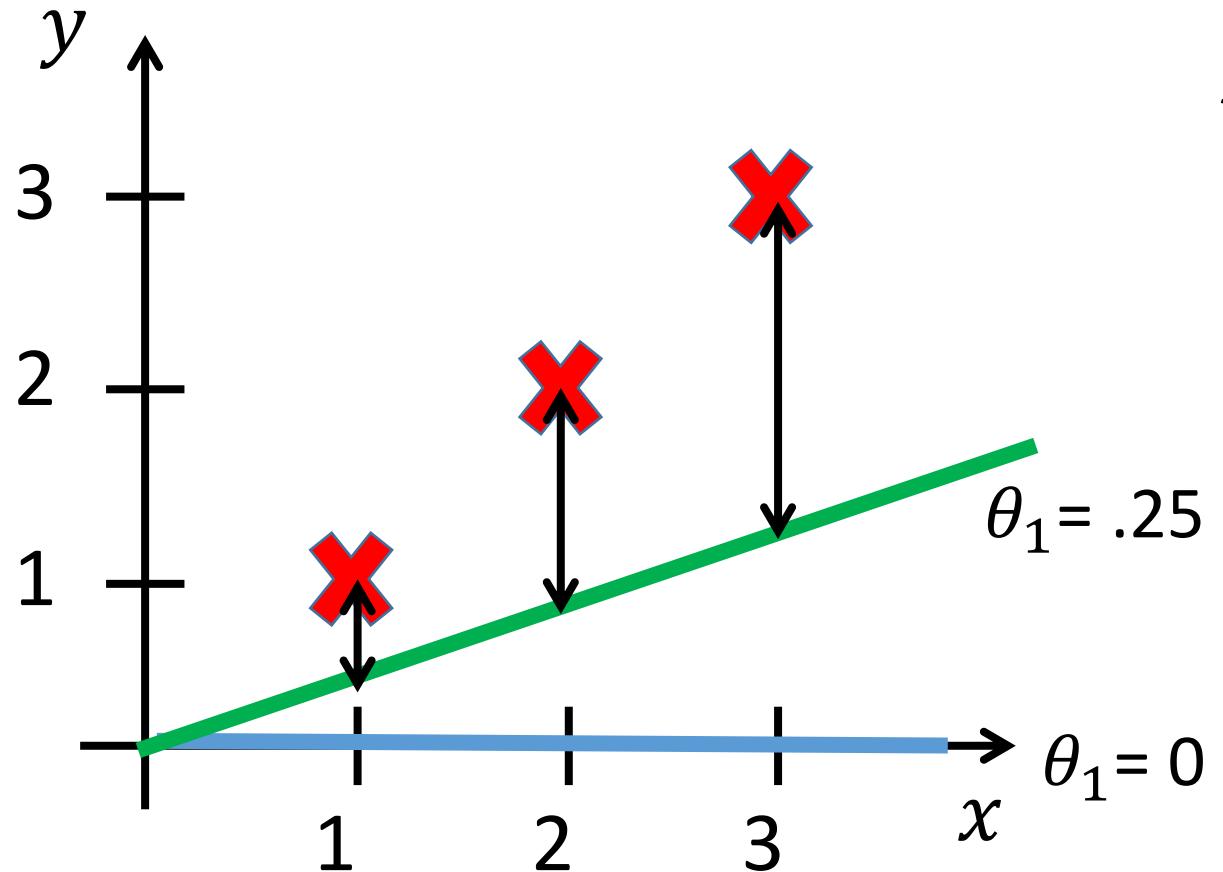
$h_{\theta_1}(x)$ , function of  $x$



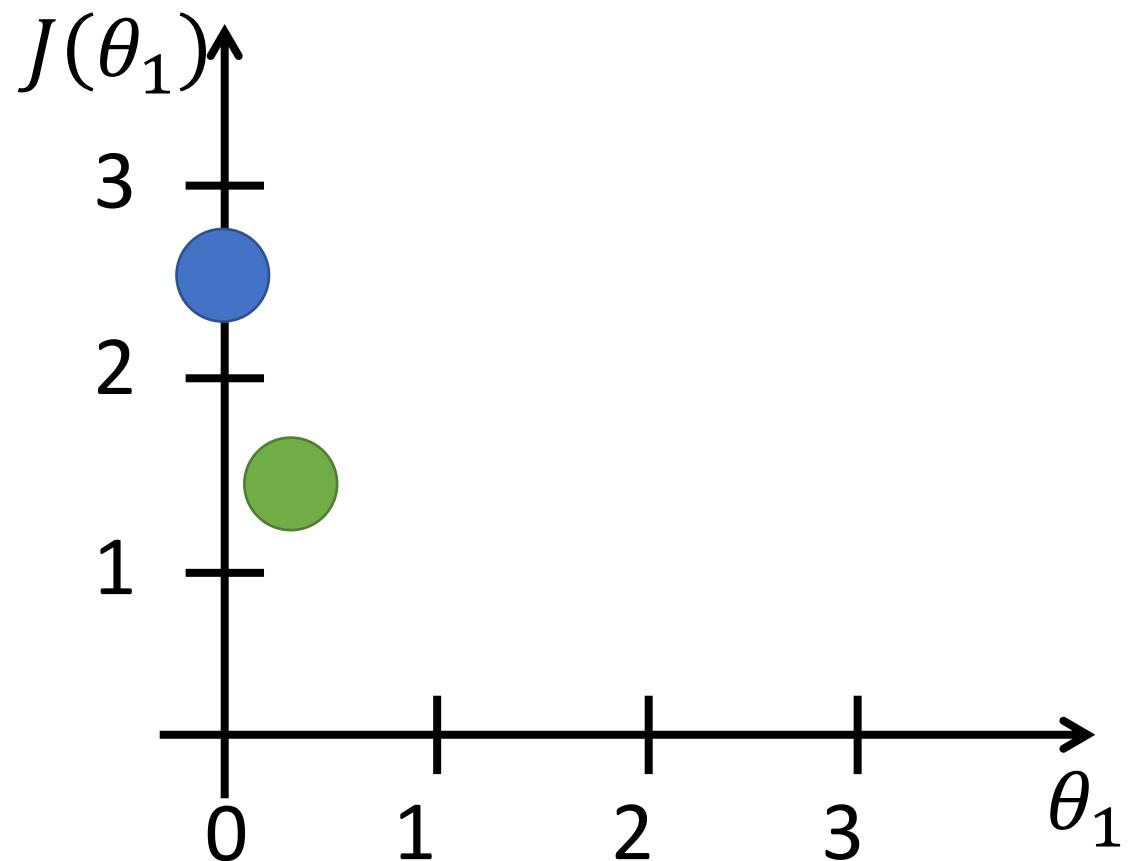
$J(\theta_1)$ , function of  $\theta_1$



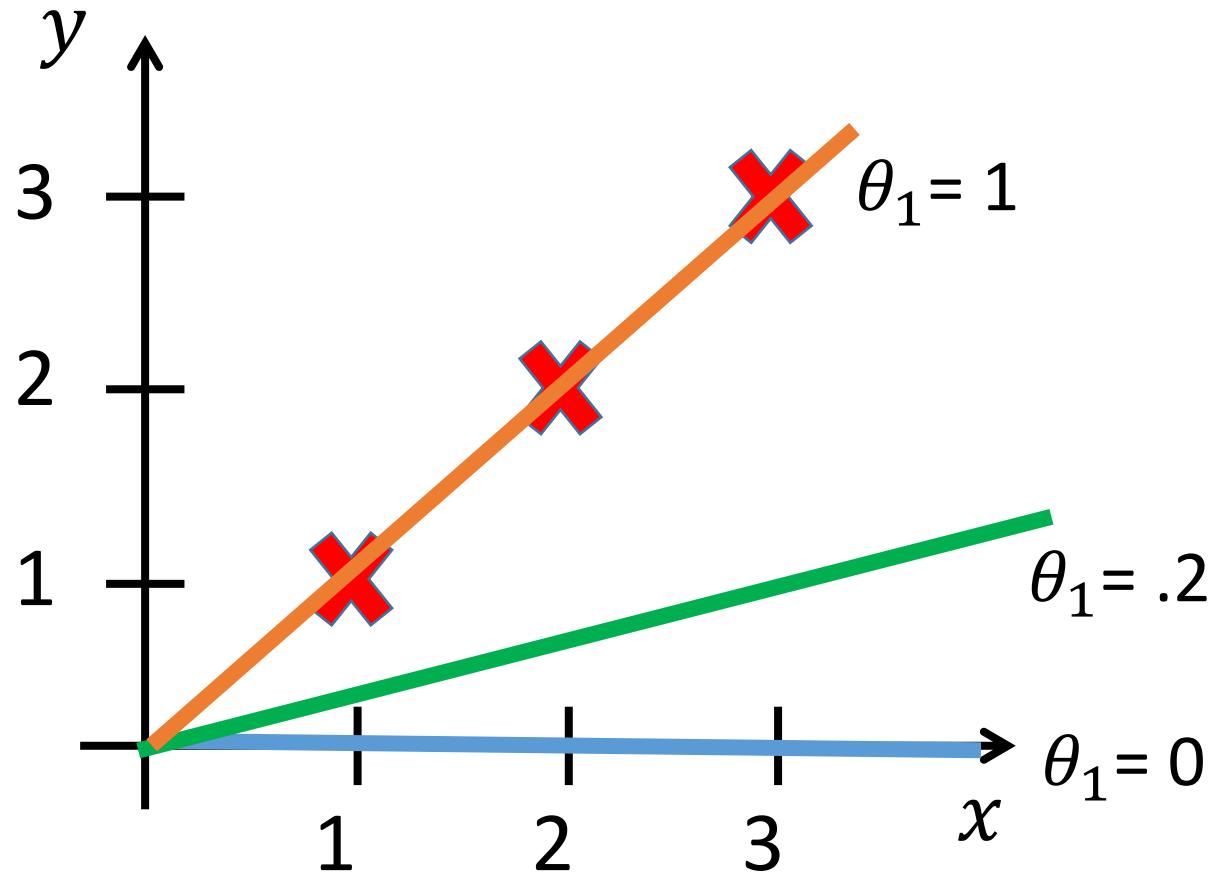
$h_\theta(x)$ , function of  $x$



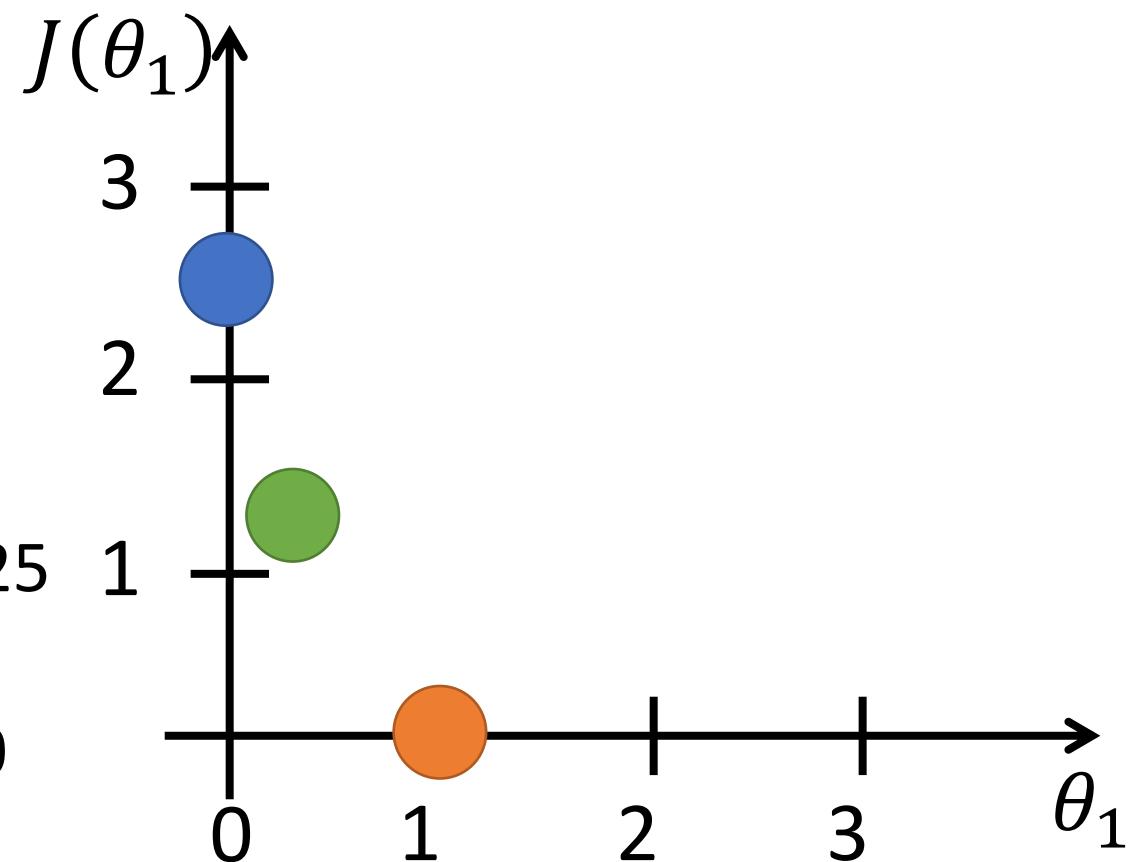
$J(\theta_1)$ , function of  $\theta_1$



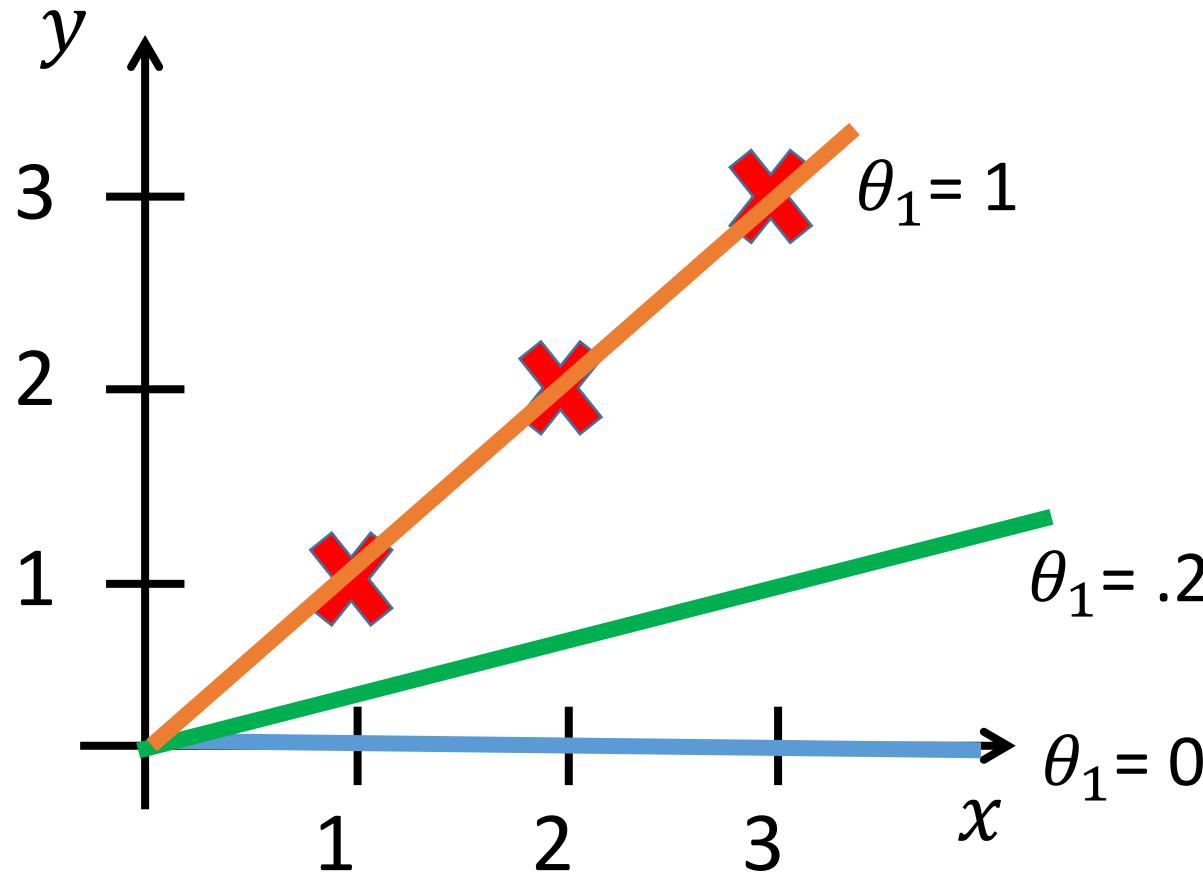
$h_\theta(x)$ , function of  $x$



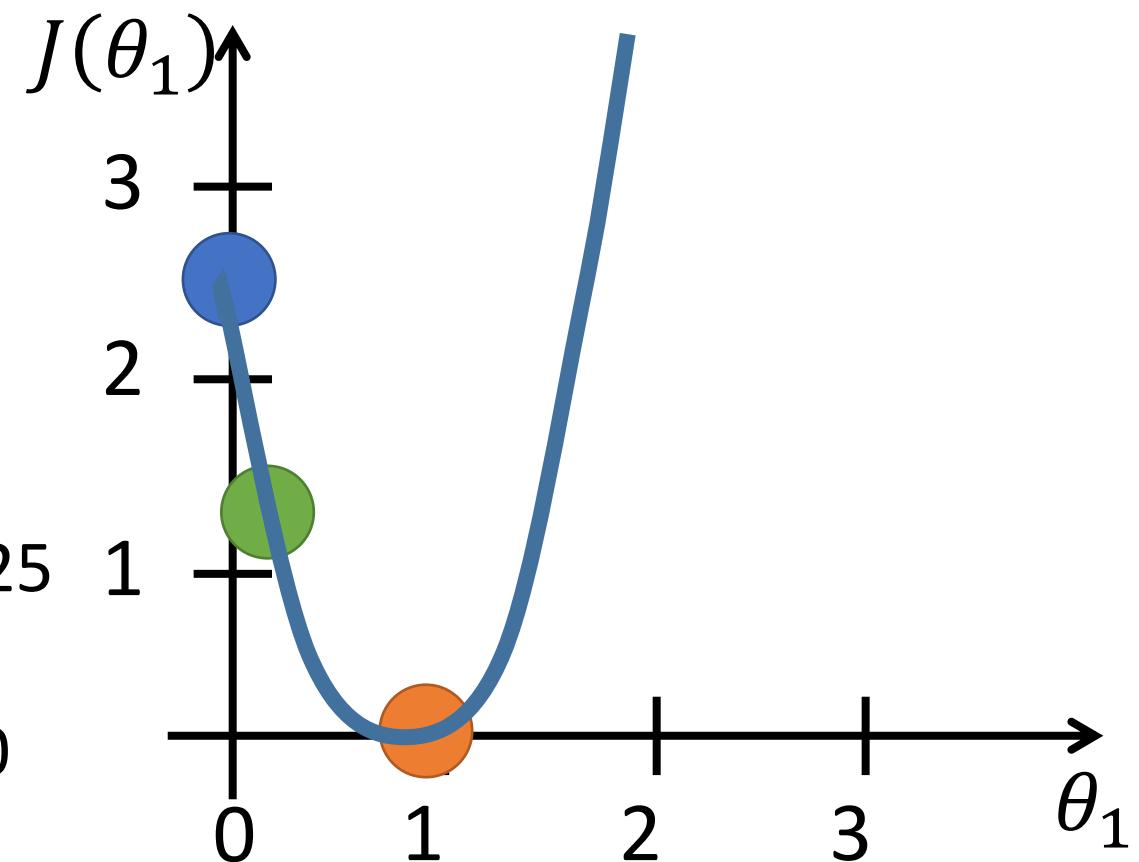
$J(\theta_1)$ , function of  $\theta_1$



$h_\theta(x)$ , function of  $x$



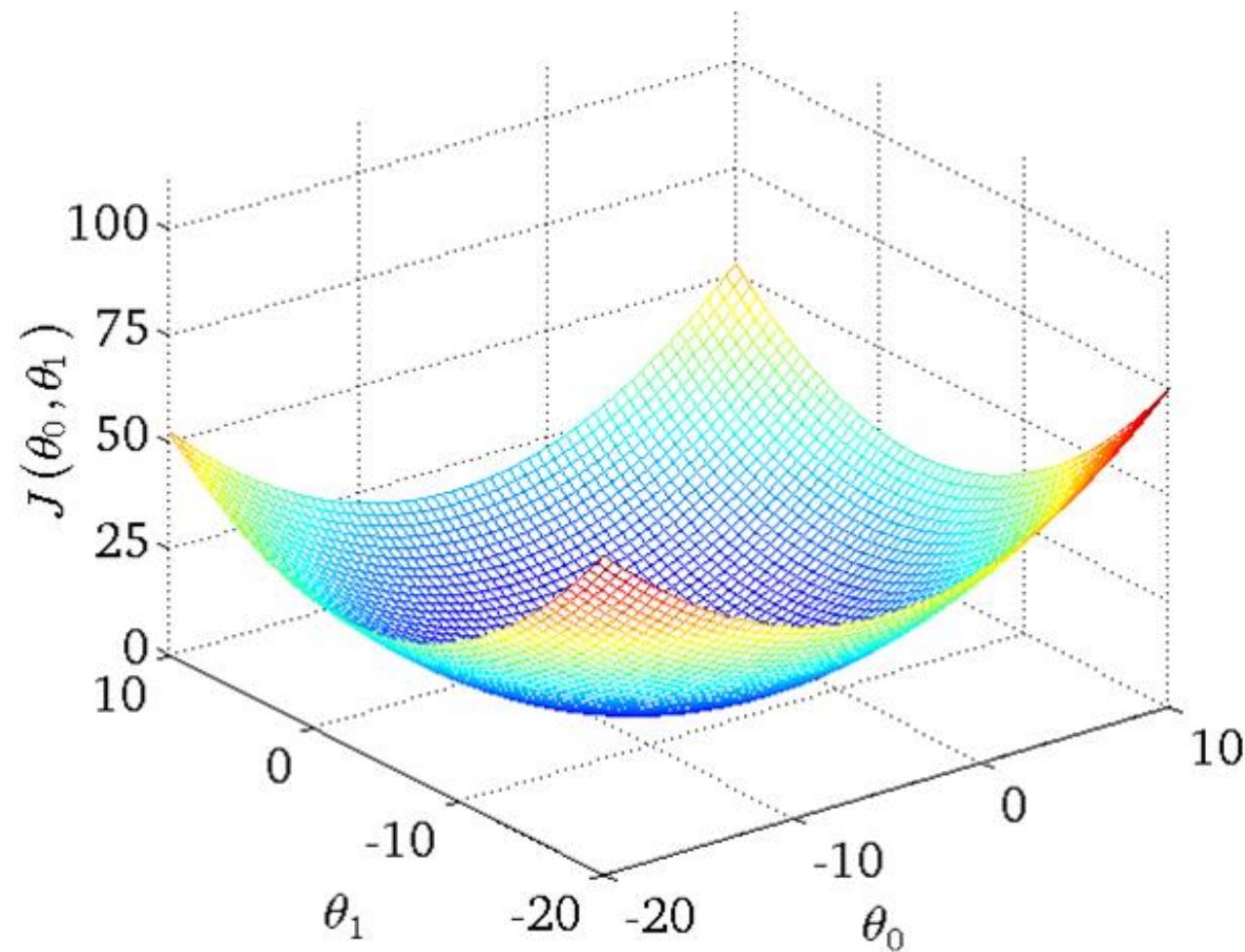
$J(\theta_1)$ , function of  $\theta_1$



If we consider both  $\theta_0, \theta_1$

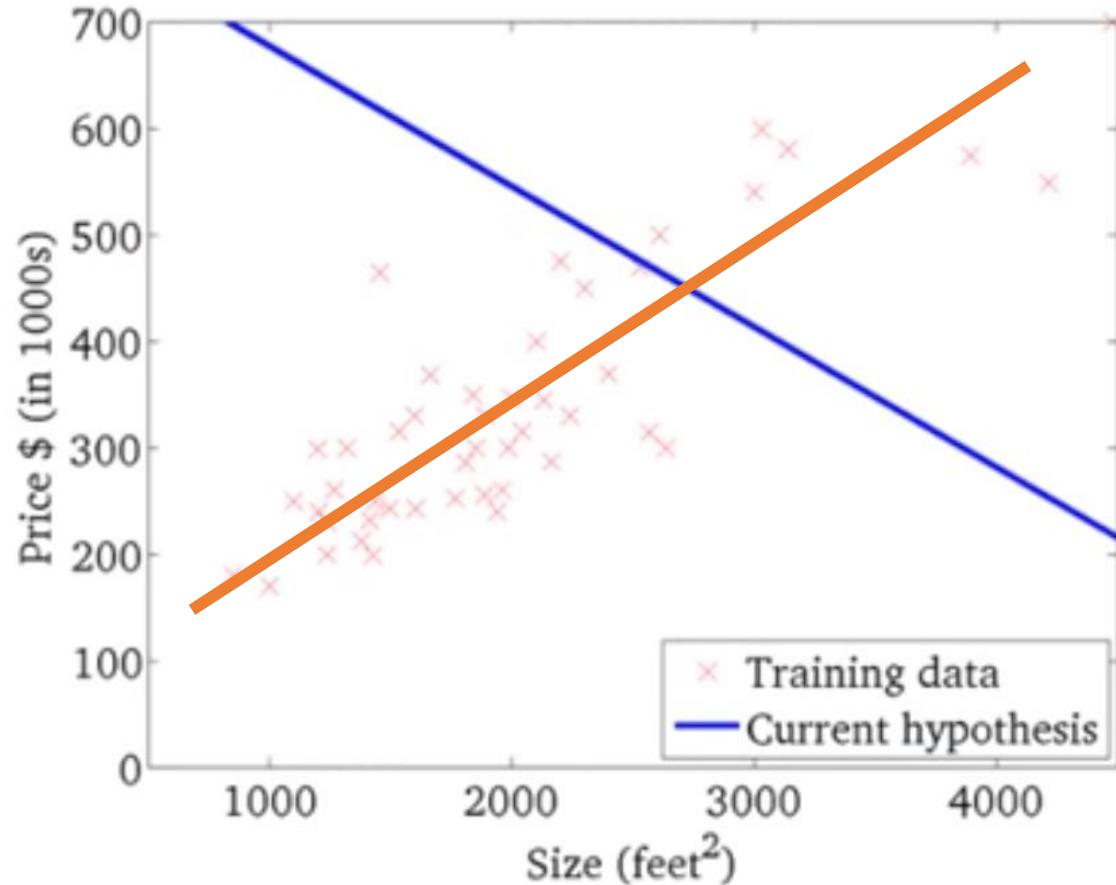
- **Hypothesis:**  $h_{\theta}(x) = \theta_0 + \theta_1 x$
- **Parameters:**  $\theta_0, \theta_1$
- **Cost function:**  $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$
- **Goal:**  $\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1)$

# Error Plan of Cost function with both $\theta_0, \theta_1$



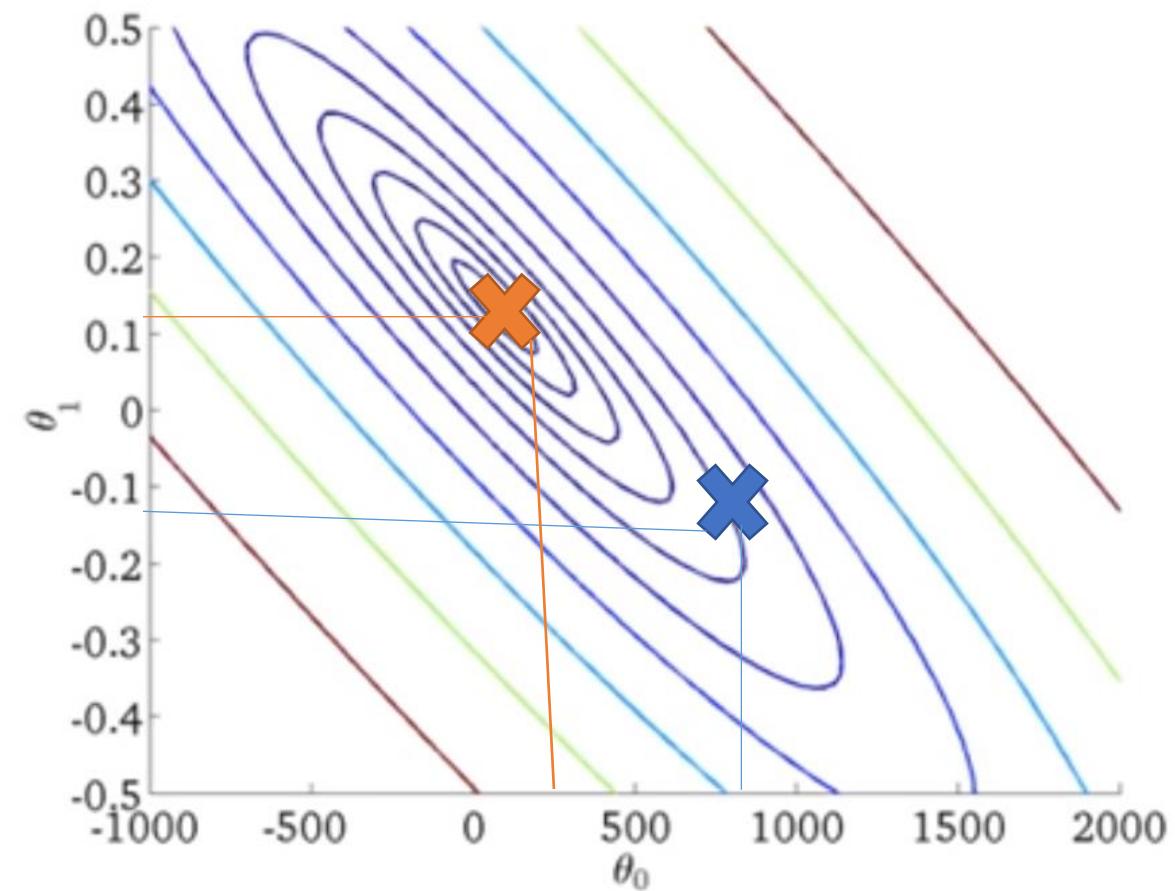
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



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

(function of the parameters  $\theta_0, \theta_1$ )

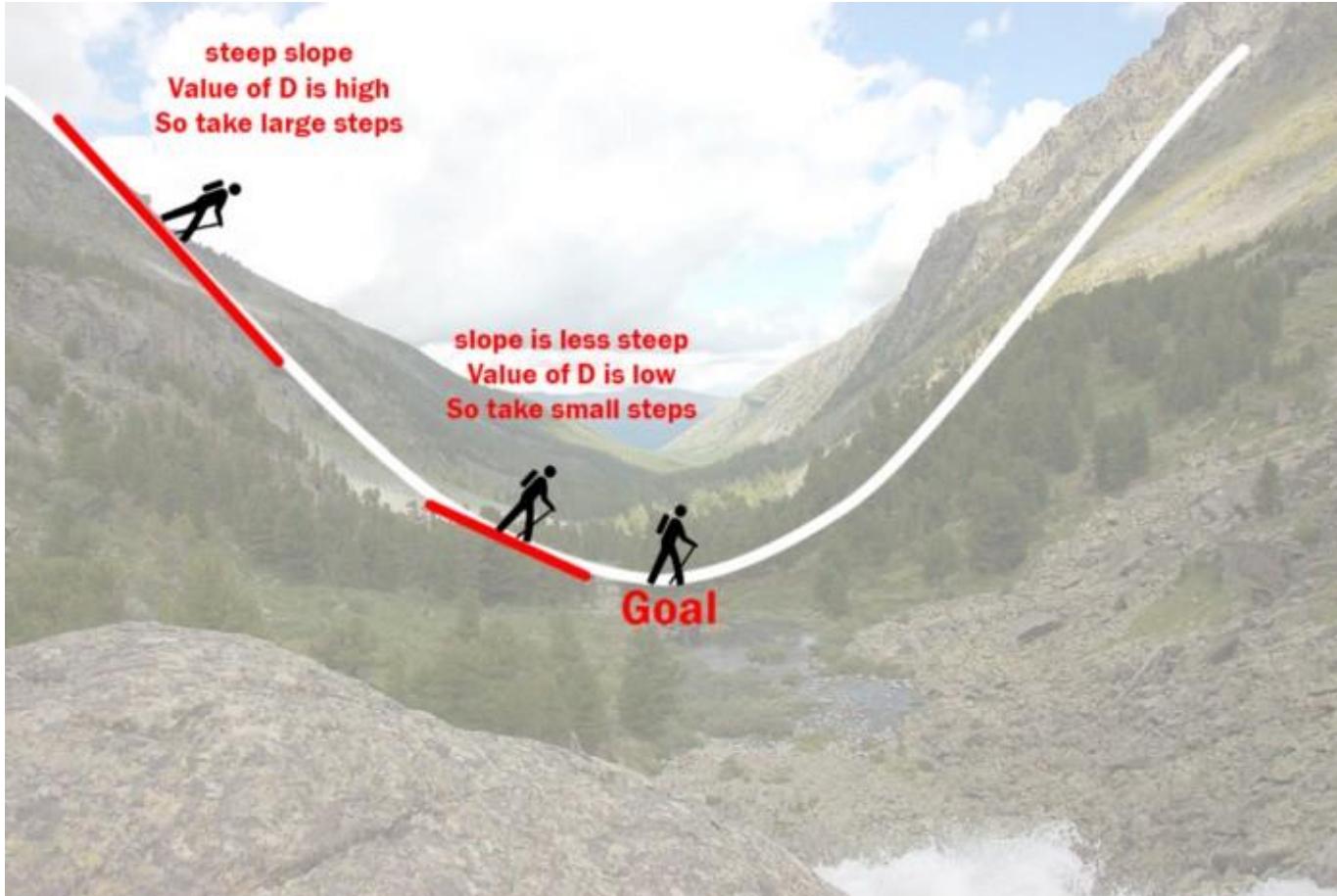


How do we find good  $\theta_0, \theta_1$  that minimize  $J(\theta_0, \theta_1)$ ?

# Gradient Descent

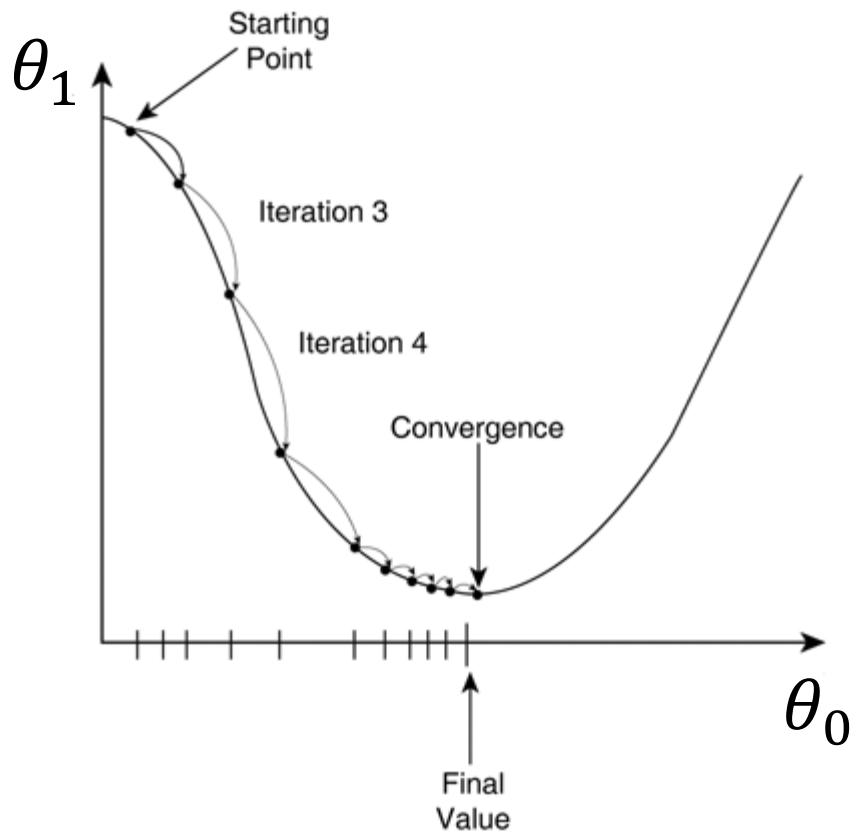
- Gradient descent enables a model to **learn the gradient or *direction*** that the model should take **in order to reduce errors** (differences between actual y and predicted y).
- Direction in the simple linear regression refers to how the **model parameters  $\theta_0, \theta_1$  should be tweaked** or corrected to further reduce the cost function.
- As the model iterates, it gradually **converges towards a minimum where further tweaks to the parameters produce little or zero changes** in the loss — also referred to as convergence.

# Gradient Descent



- Imagine a valley and a person with no sense of direction who wants to get to the bottom of the valley.
- He goes down the slope and takes large steps when the slope is steep and small steps when the slope is less steep.
- He decides his next position based on his current position and stops when he gets to the bottom of the valley which was his goal.

# Gradient Descent



- At this point the model has **optimized the weights** such that they minimize the cost function
- **Gradient** descent identifies the optimal value by taking big steps when it is far away and and takes the baby steps when it is closer to the optimal solution.
- There is a term called **learning rate** — this controls the size of the steps taken by each gradient, i.e. the change in the value in  $\theta_1, \theta_0$
- If this is **too big**, the model might **miss the local minimum of the function**. If it **too small, the model will take a long time to converge**.

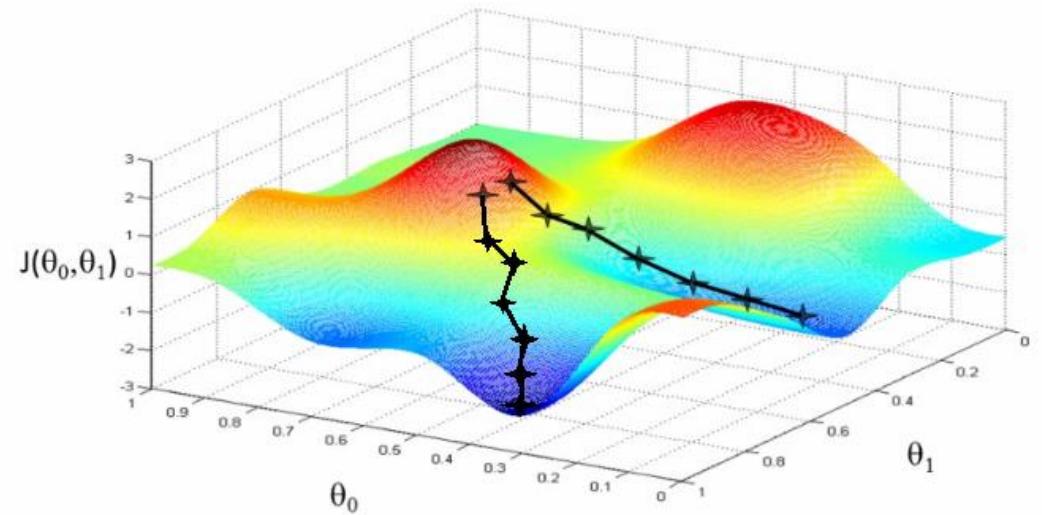
# Gradient descent

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

**Task:**  $\operatorname{argmin}_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

**Solution:**

- Start with some  $\theta_0, \theta_1$
- Keep changing  $\theta_0, \theta_1$  to reduce  $J(\theta_0, \theta_1)$  until we hopefully end up at minimum



# Gradient descent

Repeat until convergence

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

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

$\alpha$ : Learning rate (step size)

$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ : derivative (rate of change)

Slope (gradient) is the first-order derivative of the function  $J(\theta_0, \theta_1)$  at the current point.

# Gradient descent for Linear Regression

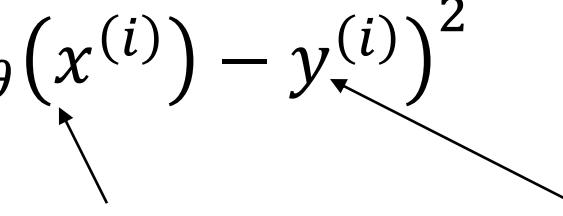
Repeat until convergence

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

- Linear regression model

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

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

A diagram illustrating the cost function. It shows two terms being squared and summed:  $(h_{\theta}(x^{(i)}) - y^{(i)})^2$ . An arrow points from the term  $h_{\theta}(x^{(i)})$  to the text "Predicted value". Another arrow points from the term  $y^{(i)}$  to the text "Actual Value/True value".

**Predicted value**      **Actual Value/True value**

# Computing partial derivative

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

$$\begin{aligned} \text{Now, } \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) &= \frac{\partial}{\partial \theta_j} \left[ \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \right] \\ &= \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \frac{\partial}{\partial \theta_j} (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) \\ &= \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \frac{\partial}{\partial \theta_j} (\theta_1 x^{(i)} - y^{(i)}) \\ &= \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \frac{\partial}{\partial \theta_j} (\theta_1 x^{(i)}) \end{aligned}$$

# Computing partial derivative

- $j = 0$ :  $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$
- $j = 1$ :  $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$
- Therefore  $\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$

## Gradient Descent approach

Step1: Initially, set  $\phi_1 = 0$  and  $\phi_0 = 0$  and  $L$  be the LR.  
 let  $L = 0.0001$  (very small for good accuracy)

Step2: Calculate Partial derivative of loss function with respect to  $\phi_1$ .

$$\begin{aligned} D\phi_1 &= \frac{1}{2m} \sum_{i=0}^m 2(y^{(i)} - (\phi_1 x^{(i)} + \phi_0)) \frac{\partial}{\partial \phi_1} (-\phi_1 x^{(i)}) \\ &= \frac{1}{m} \sum_{i=0}^m (y^{(i)} - (\phi_1 x^{(i)} + \phi_0)) (-x^{(i)}) \end{aligned}$$

$$D\phi_1 = -\frac{1}{m} \sum_{i=0}^m x^{(i)} (y^{(i)} - h_\theta(x^{(i)}))$$

Now, replace the value of  $D\phi_1$  with current values of  $x, y, \phi_1$  &  $\phi_0$ .

Similarly, find the partial derivative of  $\text{with respect to } \varphi_0$

$$\begin{aligned} D\varphi_0 &= \frac{1}{2m} \sum_{i=0}^m 2(y^{(i)} - (\varphi_1 x^{(i)} + \varphi_0)) \frac{\partial}{\partial \varphi_0} (\varphi_0) \\ &= \frac{1}{m} \sum_{i=0}^m (y^{(i)} - \varphi_0 x^{(i)}). \end{aligned}$$

Now, we can obtain value of  $D\varphi_0$  by putting the current values of  $\varphi_0, \varphi_1$ , and  $m$ .

Step 3: Update current values of  $\varphi_0$  and  $\varphi_1$ .

$$\therefore \varphi_1 = \varphi_1 - L * D\varphi_1$$

$$\varphi_0 = \varphi_0 - L * D\varphi_0$$

Step 4: We repeat this process until when our loss function is a very small value or ideally 0. [accuracy 100%]

# Gradient descent for Linear Regression

Repeat until convergence

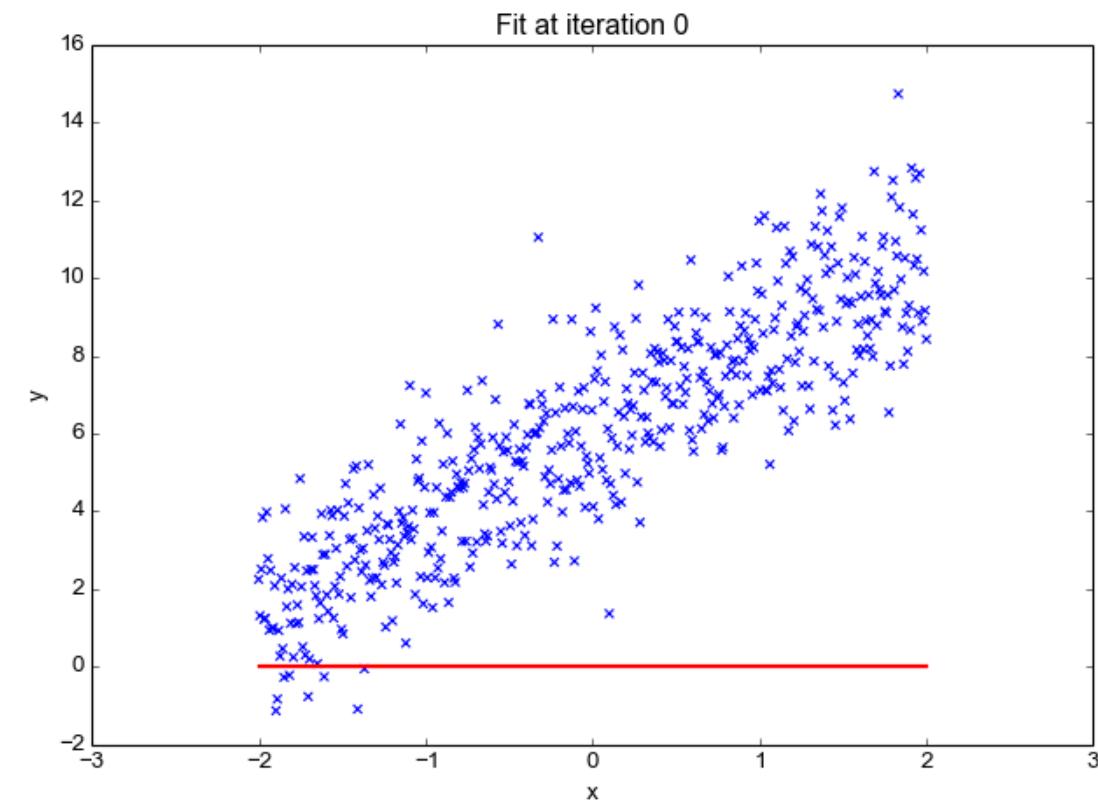
{

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

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

}

Update  $\theta_0$  and  $\theta_1$  simultaneously



# Gradient descent

## Simultaneous Update (Correct)

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

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

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

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

## Incorrect:

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

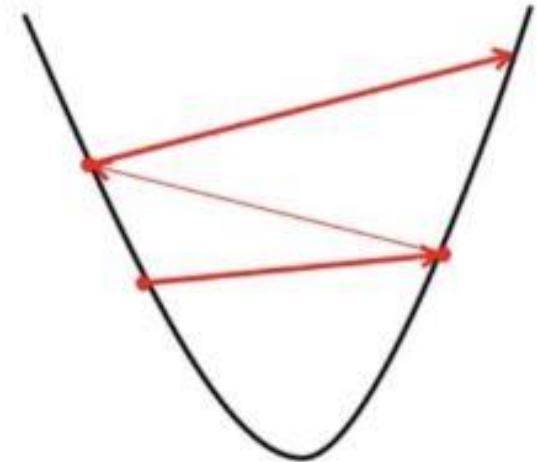
$$\theta_0 := \text{temp0}$$

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

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

# Importance of Learning Rate

- If we choose  $\alpha$  to be very large, Gradient Descent can overshoot the minimum. It may fail to converge or even diverge as instead of minimizing error, it will increase the error.
- If we choose  $\alpha$  to be very small, Gradient Descent will take small steps to reach local minima and will take a longer time to reach minima.

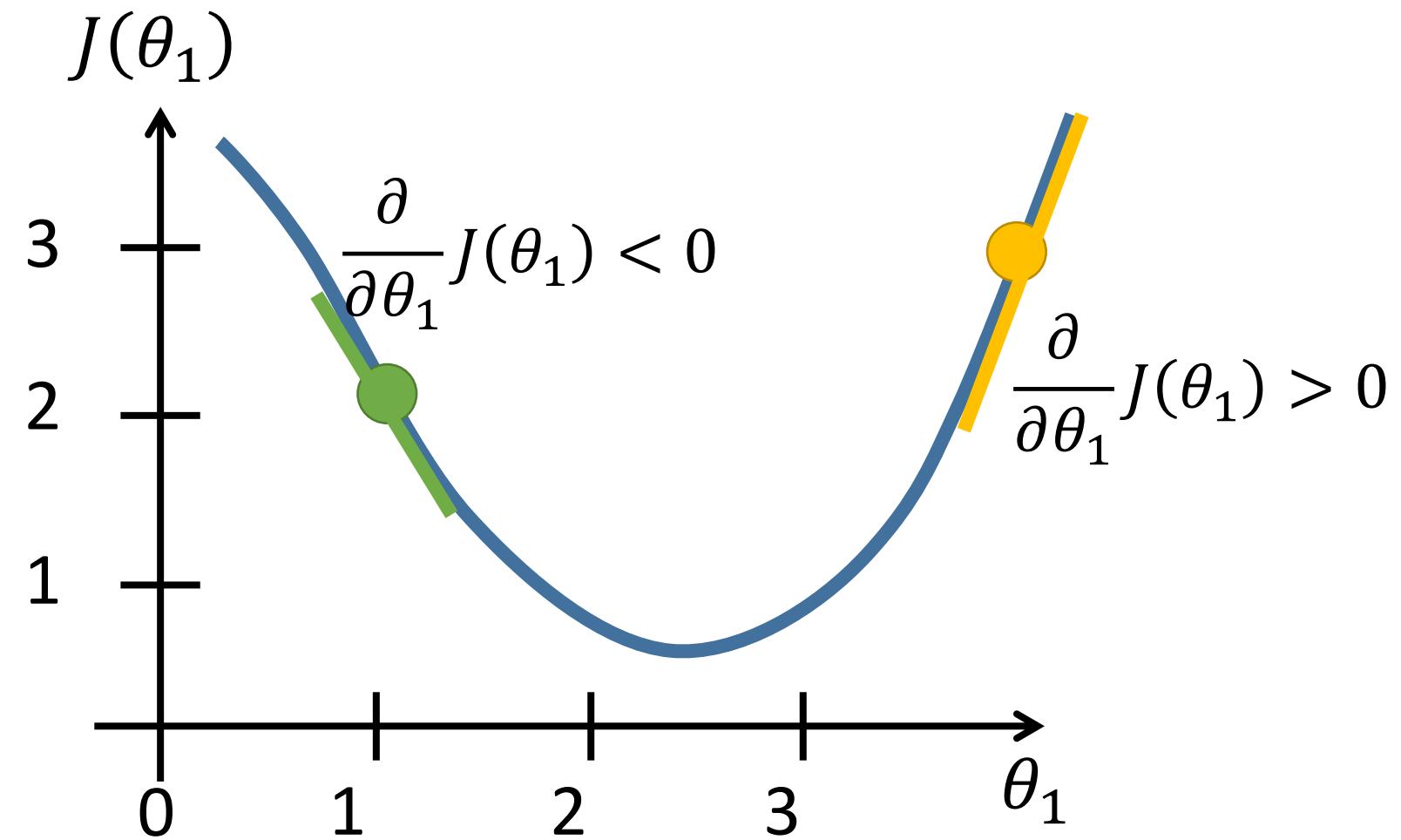


Gradient descent determines the **step size** by multiplying the slope with the learning rate  $\alpha$ . Where,

If **slope of the curve is positive**, then step size is positive value, hence, **value of  $\theta_1$  decreases**.

If slope of the curve is negative, then step size is negative, hence **value of  $\theta_1$  increases**.

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



# Batch gradient descent

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

Repeat until convergence{

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

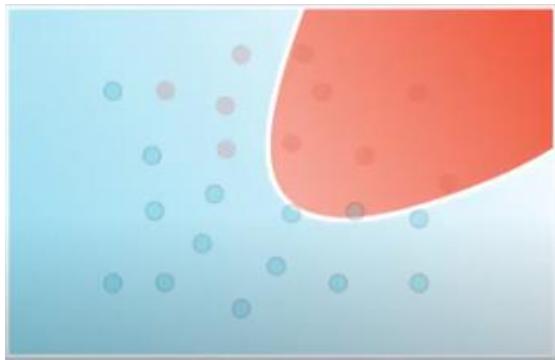
$m$ : Number of training examples

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

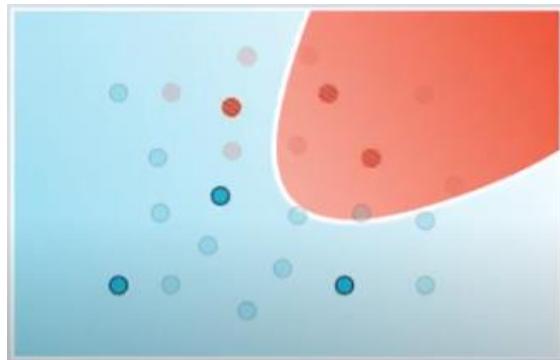
}

# Batch gradient descent

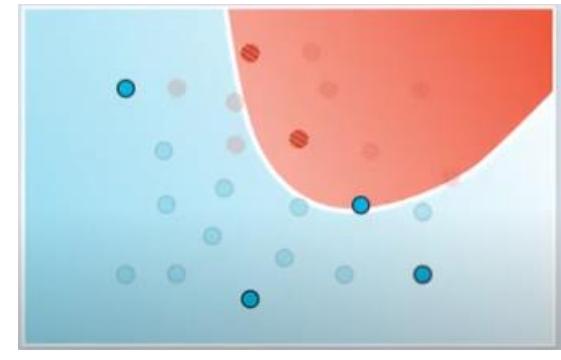
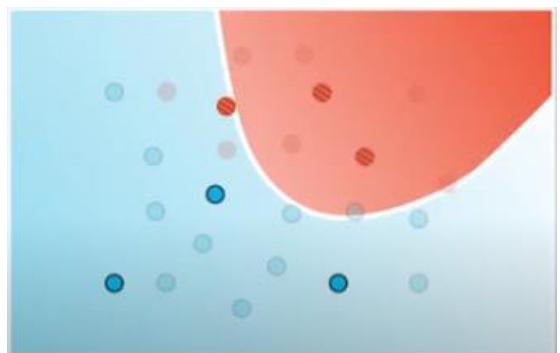
- 24 Data points: Divided into 4 batches of 6 data points in each batch



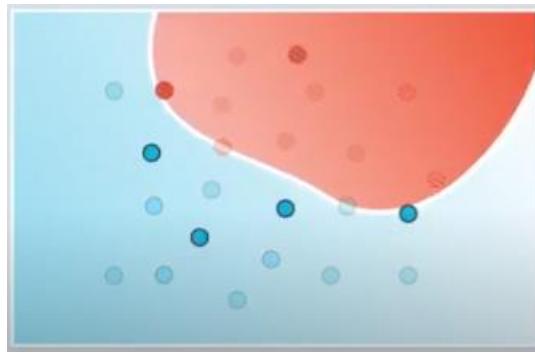
D = Original Training Set  
with 24 samples



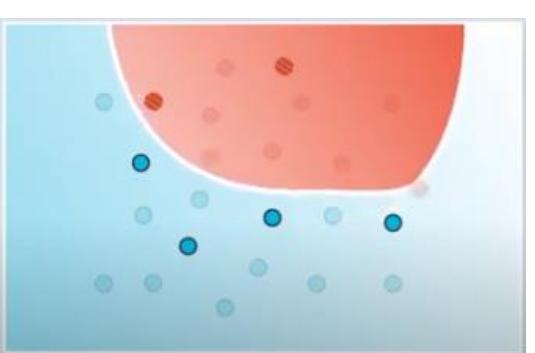
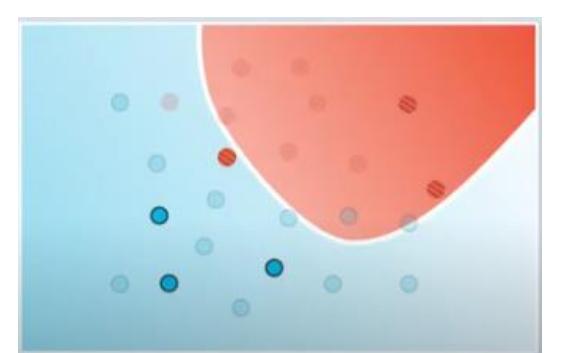
- Batch1
- Compute Error
- Upgrade weights using GD



- Batch2  
Compute Error  
Upgrade weights using GD



Decision Boundary after  
training with Batch3



Decision Boundary after  
training with Batch4

# Multiple features (input/ independent variables)

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

Notation:

$n$  = Number of features

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

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

$$x_3^{(2)} = ?$$

$$x_3^{(4)} = ?$$

# Form of Hypothesis for Multiple Features

Previously:

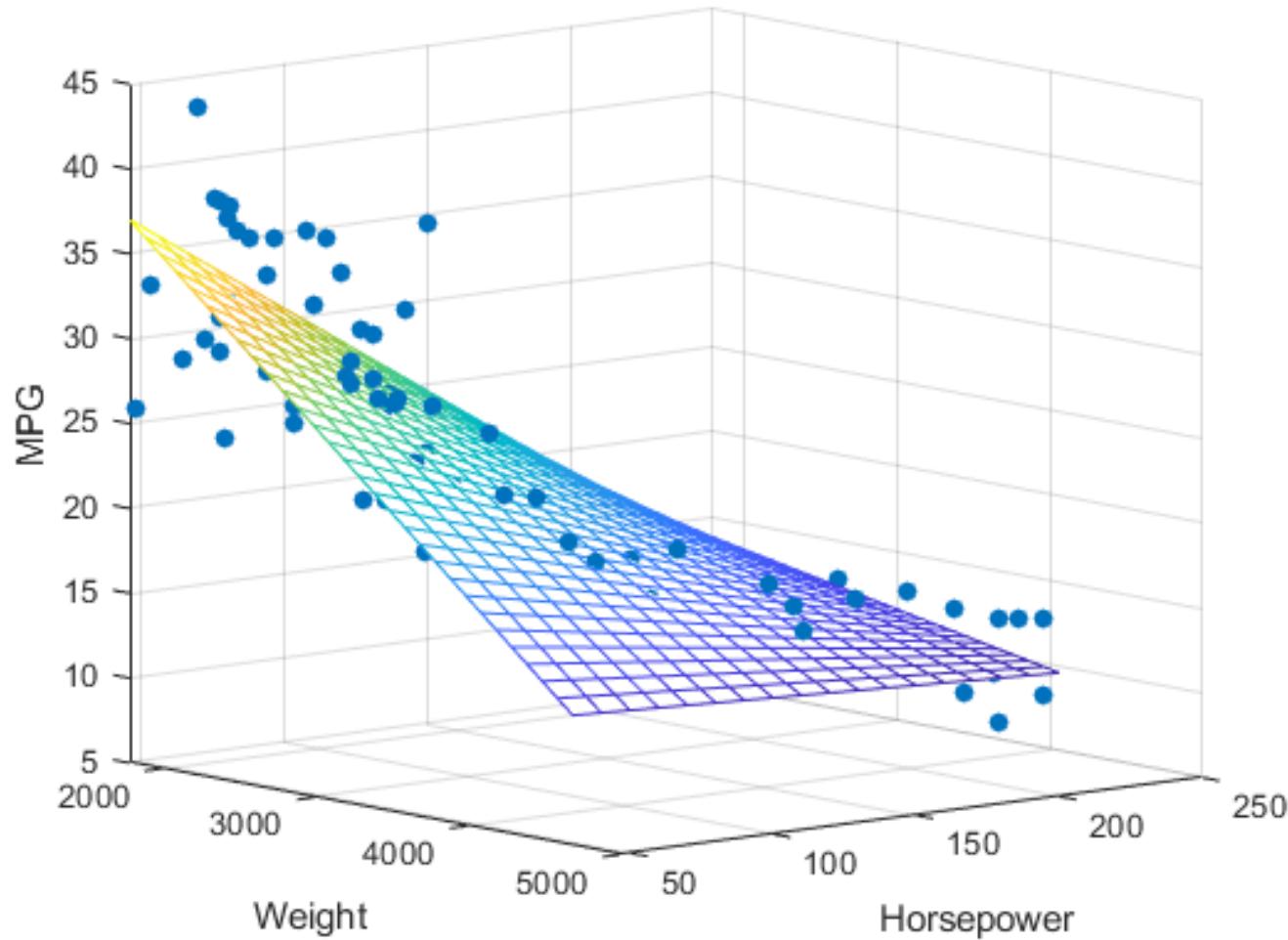
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Now:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Coefficients or Weights

# The Best fitted Plan in Multiple Regression



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

- For convenience of notation, define  $x_0 = 1$   
( $x_0^{(i)} = 1$  for all examples)

$$\bullet \quad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in R^{n+1} \qquad \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in R^{n+1}$$

$$\begin{aligned} \bullet \quad h_{\theta}(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \\ &= \boldsymbol{\theta}^{\top} \mathbf{X} \end{aligned}$$

# Multiple Regression

$$y_1 = \theta_0 + \theta_1 x_{11} + \theta_2 x_{12} + \dots + \theta_n x_{1n}$$

$$y_2 = \theta_0 + \theta_1 x_{21} + \theta_2 x_{22} + \dots + \theta_n x_{2n}$$

⋮

$$y_m = \theta_0 + \theta_1 x_{m1} + \theta_2 x_{m2} + \dots + \theta_n x_{mn}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & & & \vdots \\ 1 & x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} * \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix}$$

$$y = X\theta^T$$

$$h_{\theta}(x) = \theta_0 + 2104\theta_1 + 5\theta_2 + \theta_3 + 45\theta_4 = 460$$

$$h_{\theta}(x) = \theta_0 + 1416\theta_1 + 3\theta_2 + 2\theta_3 + 40\theta_4 = 232$$

$$h_{\theta}(x) = \theta_0 + 1534\theta_1 + 2\theta_2 + 2\theta_3 + 30\theta_4 = 315$$

A data

	Size ( feet <sup>2</sup> ) $x_1$	Number of bedrooms $x_2$	Number of floors $x_3$	age of home (years) $x_4$	Price(\$1000) $h_{\theta}(x) = y$
	2104	5	1	45	460
	1416	3	2	40	232
	1534	2	2	30	315
	...	...	...	...	...

Features (x)      Label (  $h_{\theta}(x) = y$  )

# Gradient descent

- Previously for  $n(\text{degree}) = 1)$

Repeat until convergence

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

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

}

- For  $n (\text{degree}) \geq 1$

Repeat until convergence

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

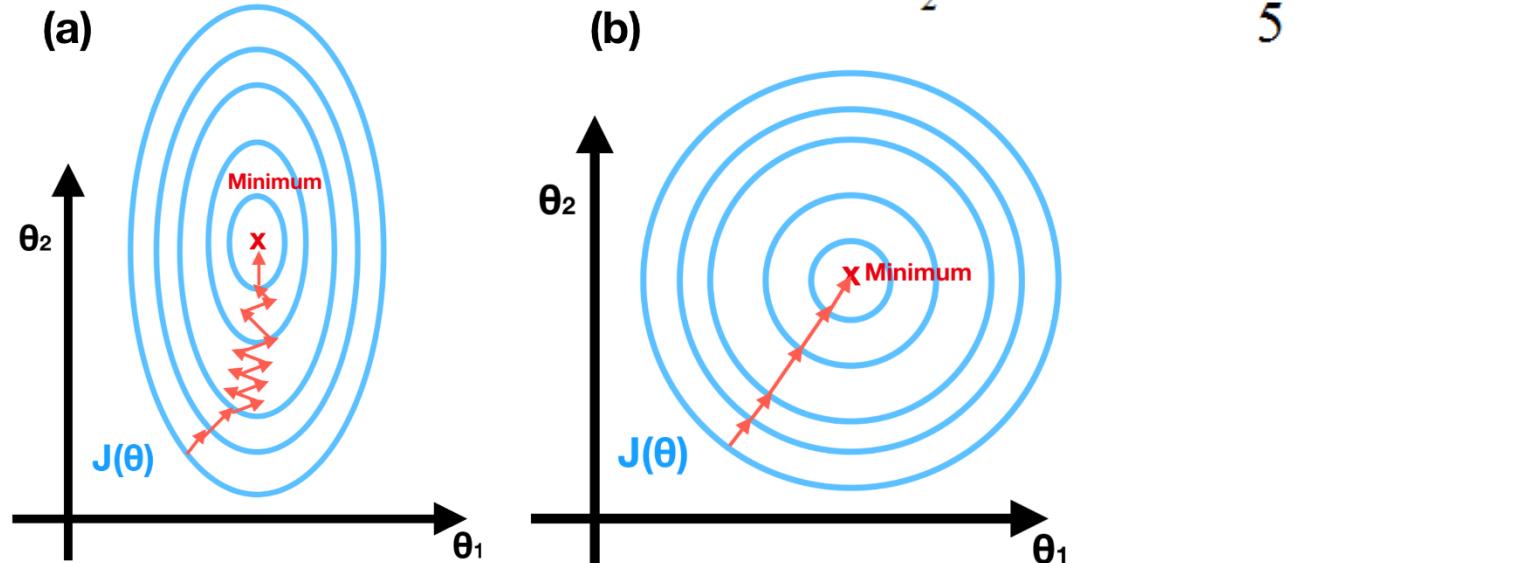
Simultaneously update  
 $\theta_j$ , for  $j = 0, 1, \dots, n$

# Gradient descent in practice: Feature scaling

- **Idea:** Make sure features are on a similar scale (e.g.,  $-1 \leq x_i \leq 1$ ) so that no variable is dominated by the other. **Gradient descent converges much faster with feature scaling than without it.**
- E.g.  $x_1 = \text{size (0-2000 feet}^2)$   
 $x_2 = \text{number of bedrooms (1-5)}$

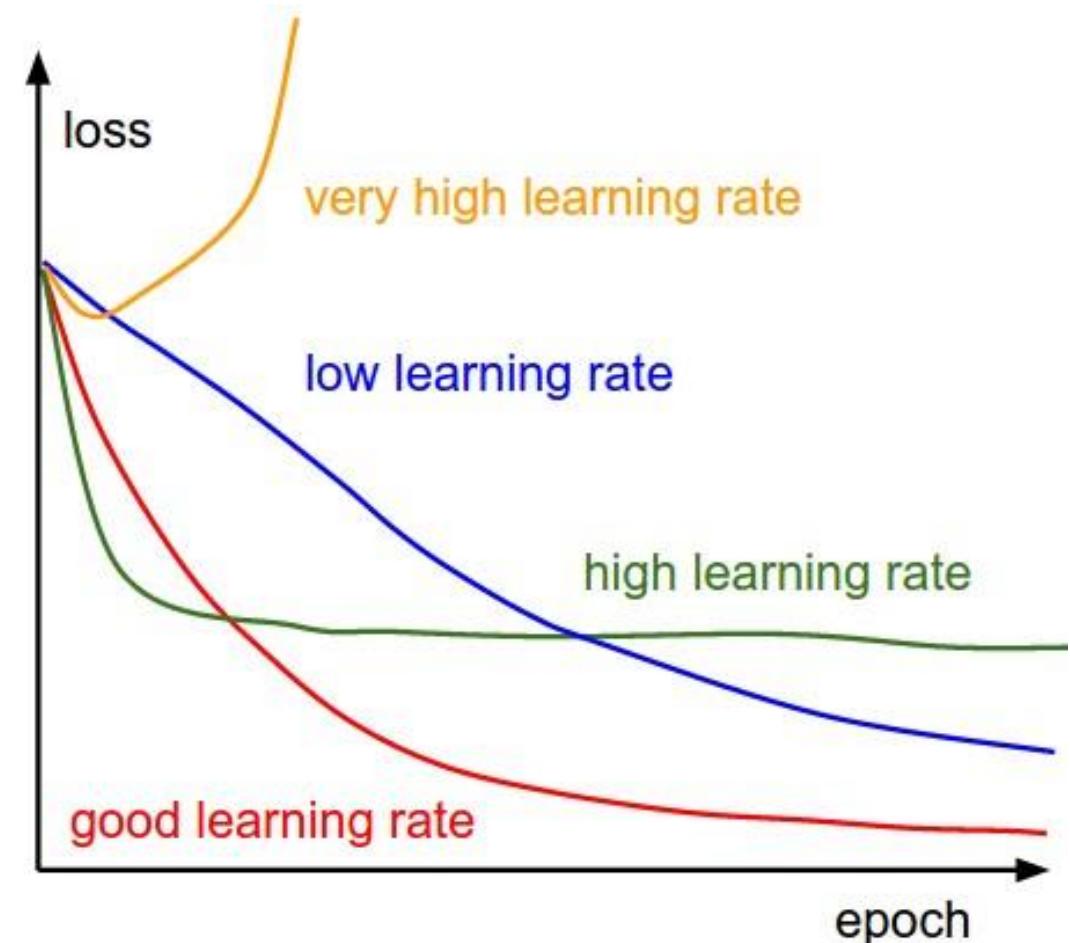
$$x_1 = \frac{\text{size(feet}^2)}{2000}$$

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

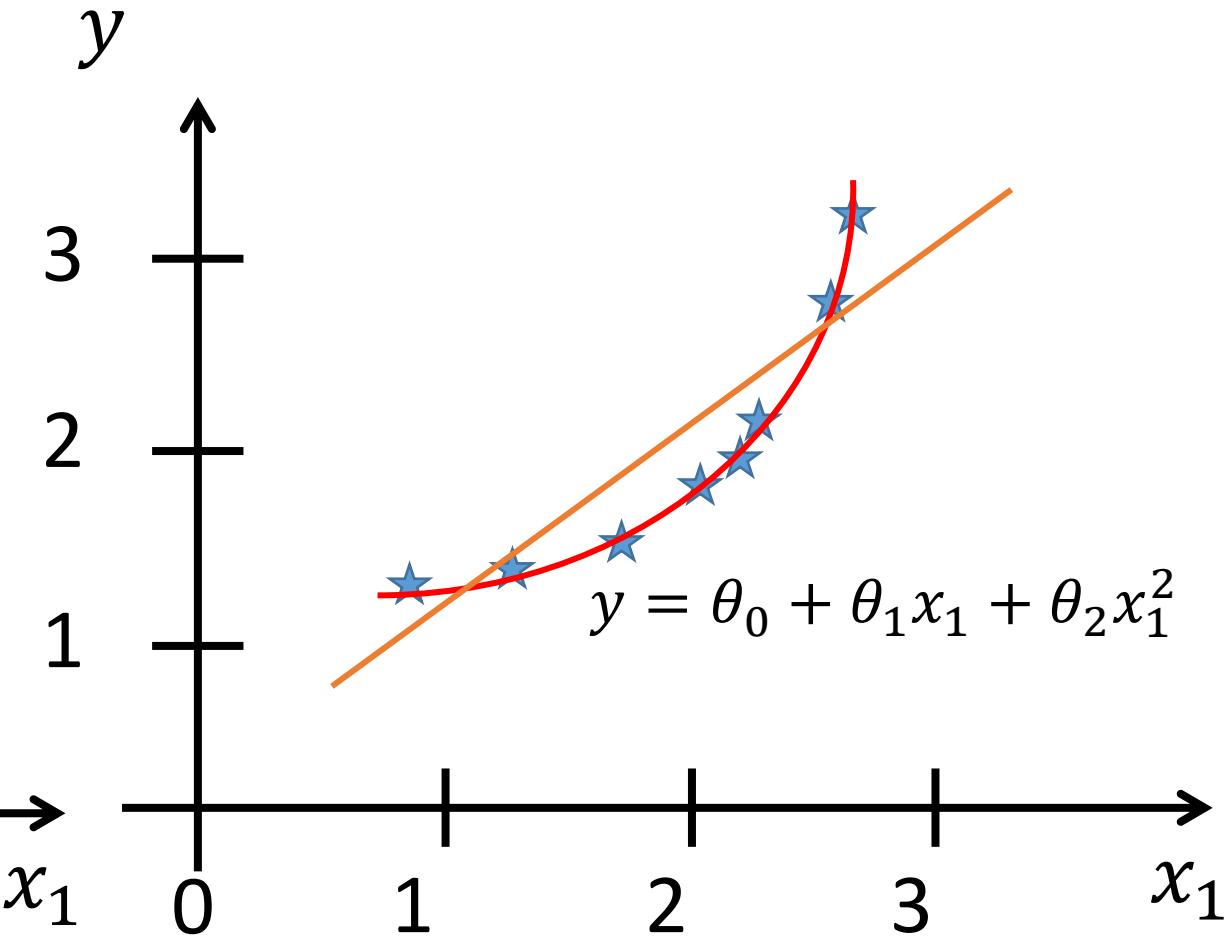
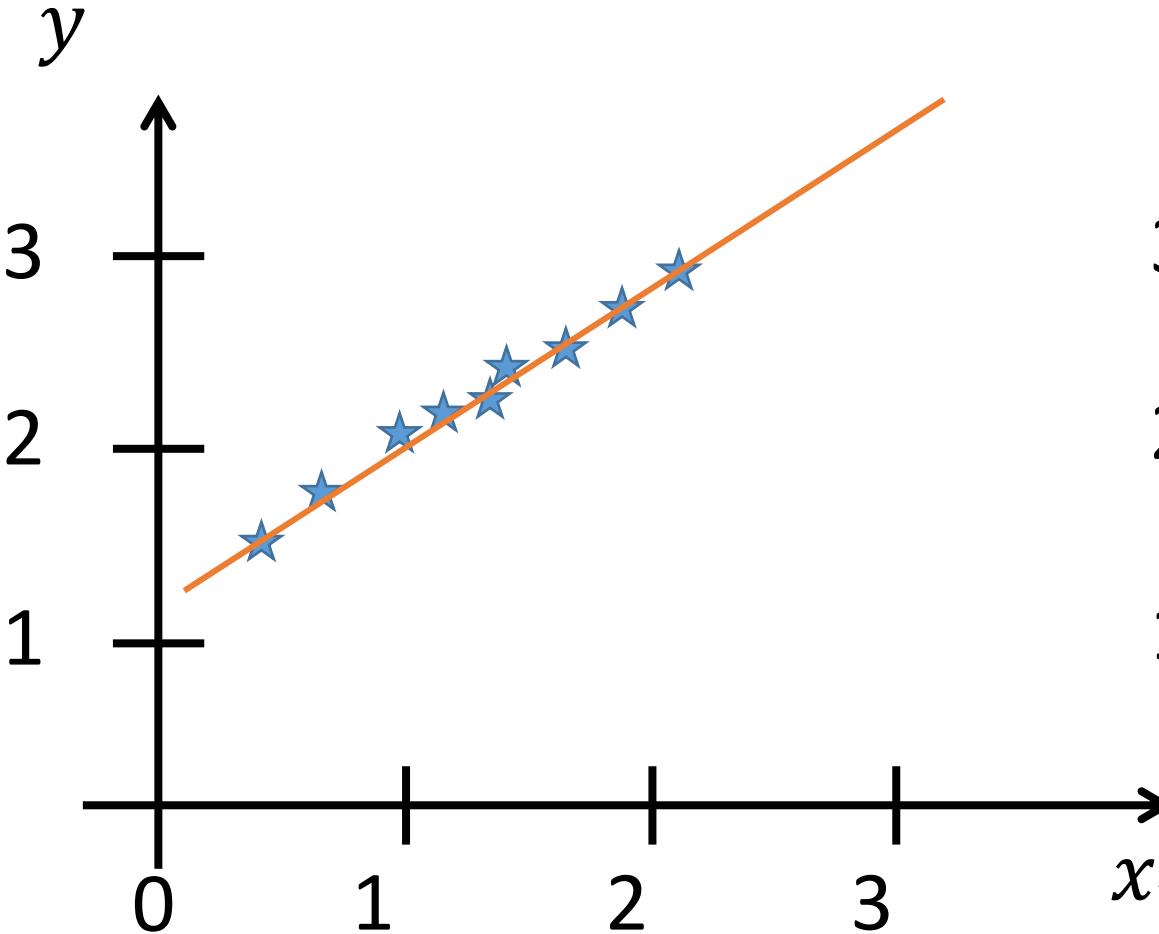


# Gradient descent in practice: Learning rate

- With **low learning rates** the improvements will be linear.
- With **very high learning rates** they will start to look **more exponential**.
- Higher learning rates will **decay the loss faster**, but they **get stuck at worse values of loss** (green line).
- To choose  $\alpha$ :** Try  $0.001, \dots, 0.01, \dots, 0.1, \dots, 1$



# Polynomial Regression (Linear Model for Nonlinear dataset)



# Polynomial Regression

- Polynomial regression is an attempt to create a polynomial function that approximates a set of data points of nonlinear fashion.
- Polynomial regression is one of several methods of curve fitting
- Linear regression is polynomial regression of degree (n) 1

# Polynomial Regression

$$y = h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \theta_0 + \sum_{i=1}^n \theta_i x_i$$

0 degree polynomial

$$y = \theta_0$$

1 degree polynomial

$$y = \theta_0 + \theta_1 x_1$$

2 degree polynomial

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$

n degree polynomial

$$y = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_1^n$$

# Linear/ Multiple Linear/ Polynomial Regression

Simple  
Linear  
Regression

$$y = \theta_0 + \theta_1 x_1$$

Multiple  
Linear  
Regression

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

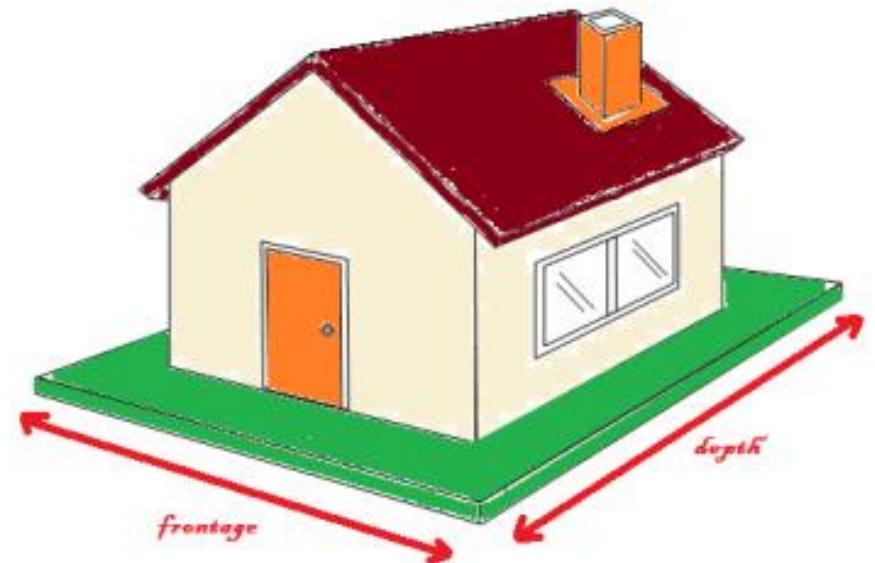
Polynomial  
Linear  
Regression

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \dots + \theta_n x_1^n$$

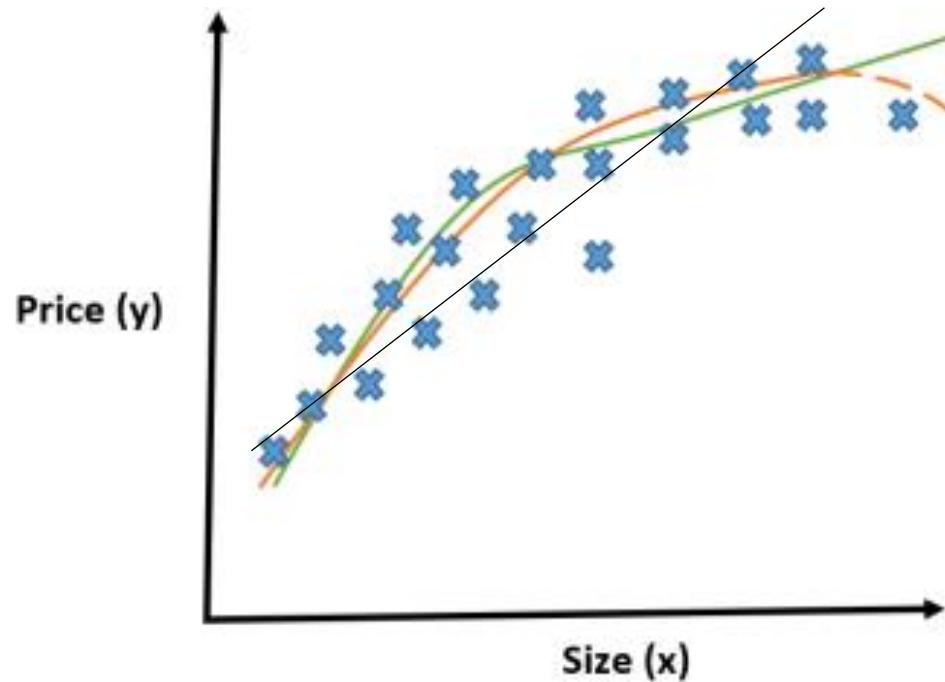
One Variable  $x_1$

# House prices prediction

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



# Application of Polynomial regression



- $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$  (Orange Line)  
 $= \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2$
- $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$  (Green Line)  
 $= \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2 + \theta_3(\text{size})^3$

$$\begin{aligned}x_1 &= (\text{size}) \\x_2 &= (\text{size})^2 \\x_3 &= (\text{size})^3\end{aligned}$$

If we choose our features like this,  
then feature scaling becomes  
increasingly important as

$$\text{size: } 1 - 1\,000$$

$$\text{size}^2: 1 - 1\,000\,000$$

$$\text{size}^3: 1 - 10^9$$

# Things to remember

- **Model representation**

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

- **Cost function**

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

- **Gradient descent for linear regression**

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

- **Features and polynomial regression**

Can combine features; can use different functions to generate features (e.g., polynomial)

# Logistic Regression

- Hypothesis representation
- Cost function
- Logistic regression with gradient descent
- Multi-class classification
- Regularization

# Logistic Regression

- There are many important research topics for which the dependent variable is "limited." For example: voting, morbidity or mortality, and participation data is not continuous or distributed normally.
- Logistic Regression is used to fit a curve to data in which the dependent variable is binary, or dichotomous.
- **dichotomous** (yes/no, True/ False...)
- **categorical** (social class, race, ... )
- **continuous** (age, weight, gestational age, ...)
- **dichotomous categorical response variable  $Y$**  (Success/Failure, Remission/No Remission Survived/Died, Low Birth Weight/Normal Birth Weight, etc)

# Classification

Email: Spam / Not Spam?

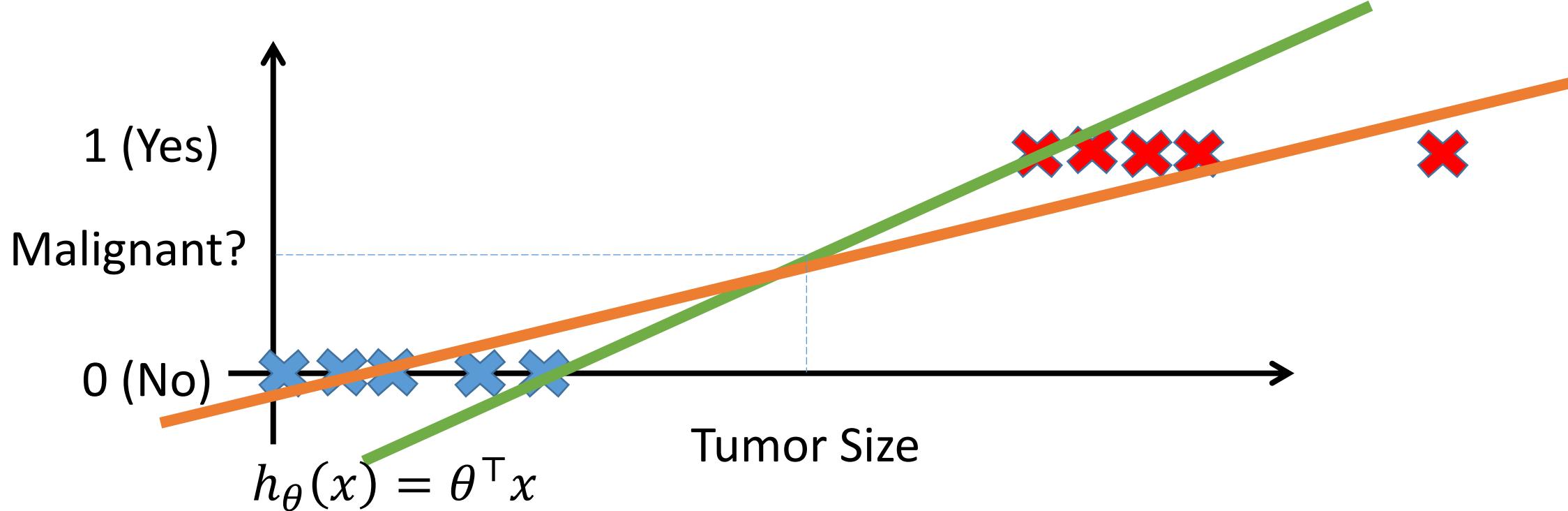
Online Transactions: Fraudulent (Yes / No)?

Tumor: Malignant / Benign ?

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

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

# Logistic Regression



- Threshold classifier output  $h_{\theta}(x)$  at 0.5
  - If  $h_{\theta}(x) \geq 0.5$ , predict " $y = 1$ "
  - If  $h_{\theta}(x) < 0.5$ , predict " $y = 0$ "

# Logistic Regression

Classification:  $y = 1$  or  $y = 0$

$y = h_{\theta}(x) = \theta_0 + \theta_1 x_1$  (from linear regression) can be  $> 1$  or  $< 0$

Logistic Regression uses a very complex cost function  $h_{\theta}(x)$  called **Sigmoid function or ‘Logistic Function’** instead of **Linear Function**.

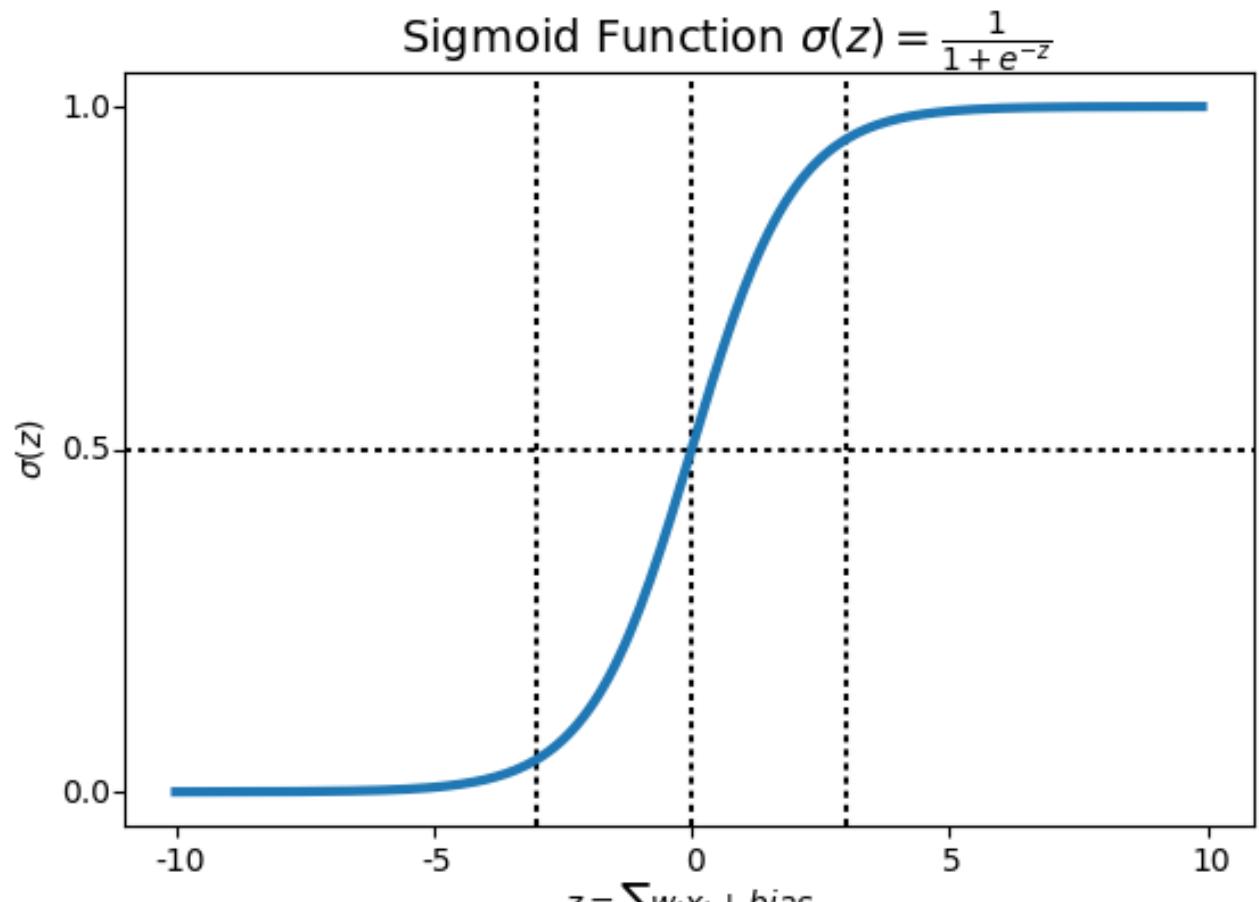
**The hypothesis of Logistic Regression tend to limit the output values between 1 and 0** and it is represented as  $0 \leq h_{\theta}(x) \leq 1$ . So, Linear Regression fails to represent it.

Hence, Logistic regression is actually for **classification**

# Sigmoid Function

In order to map the predicted values to probabilities, Sigmoid function is used. Due to the behavior of the curve, it maps **Any real value into another value between 0 and 1.**

**So, in ML we use Sigmoid to map predictions to probabilities.**



Sigmoid Function Graph

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

# Hypothesis representation in Logistic Regression

- In Linear Regression

$$y = h_{\theta}(x) = \theta_0 + \theta_1 x_1 = (\theta^T x), \quad y > 1 \text{ or } < 0$$

- But, in Logistic we Want  $0 \leq y = h_{\theta}(x) \leq 1$

$$\begin{aligned} h_{\theta}(x) &= \text{sigmoid}(y), \\ &= \text{sigmoid}((\theta^T x)), \end{aligned}$$

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

When our hypothesis ( $h_{\theta}(x)$ ) outputs a number, we treat that value as the estimated probability that  $y=1$  on input  $x$

# Interpretation of Hypothesis Output

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

Example: If  $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$$h_{\theta}(x) = 0.7$$

Tell patient that 70% chance of tumor being malignant

$h_{\theta}(x) = P(y=1|x ; \theta)$  “probability that  $y = 1$ , given  $x$ , parameterized by  $\theta$ ”

Must be true:  $P(y=1|x ; \theta) + P(y=0|x ; \theta) = 1$   
 $P(y=0|x ; \theta) = 1 - P(y=1|x ; \theta)$

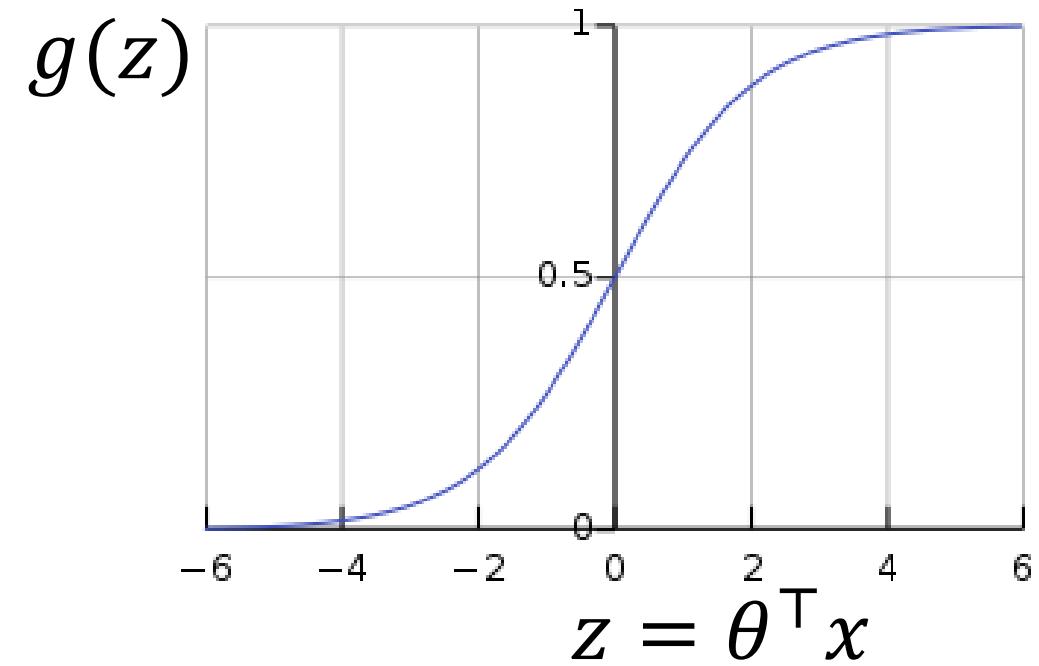
# Decision boundary

We expect a set of outputs or classes based on probability when we pass the inputs through a prediction function and returns a probability score between 0 and 1.

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

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

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

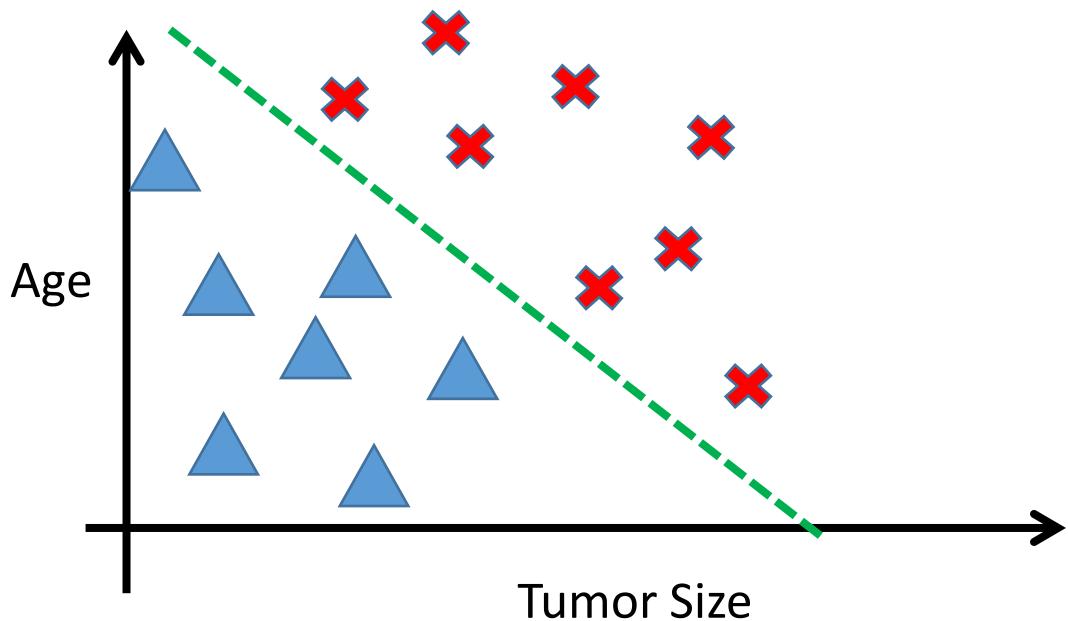


$$z = \theta^T x \geq 0$$

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

$$z = \theta^T x < 0$$

# Linear Decision boundary



*How to fit (find) Parameter  $\theta$ ?*

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

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

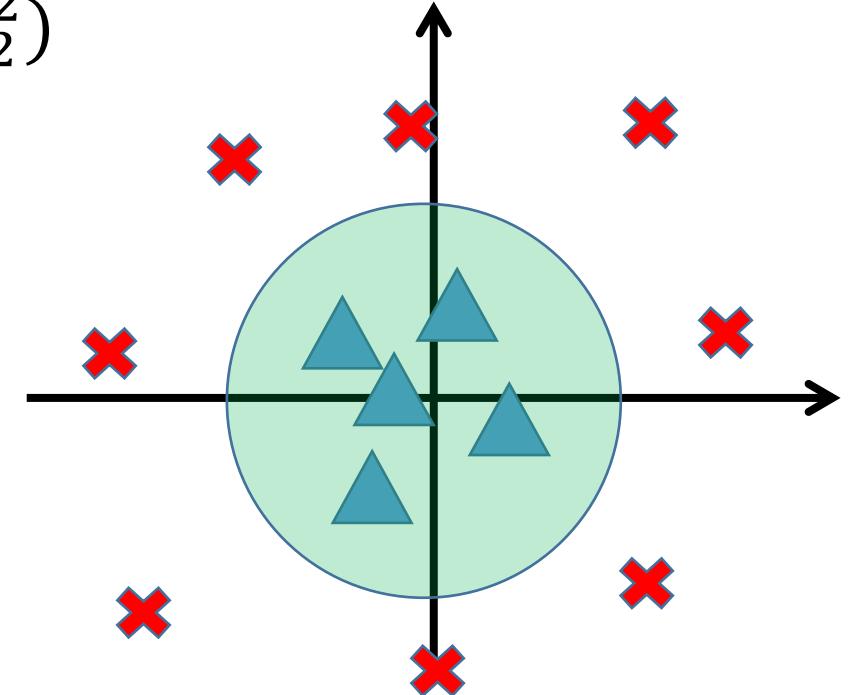
E.g.,  $\theta_0 = -3, \theta_1 = 1, \theta_2 = 1$

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

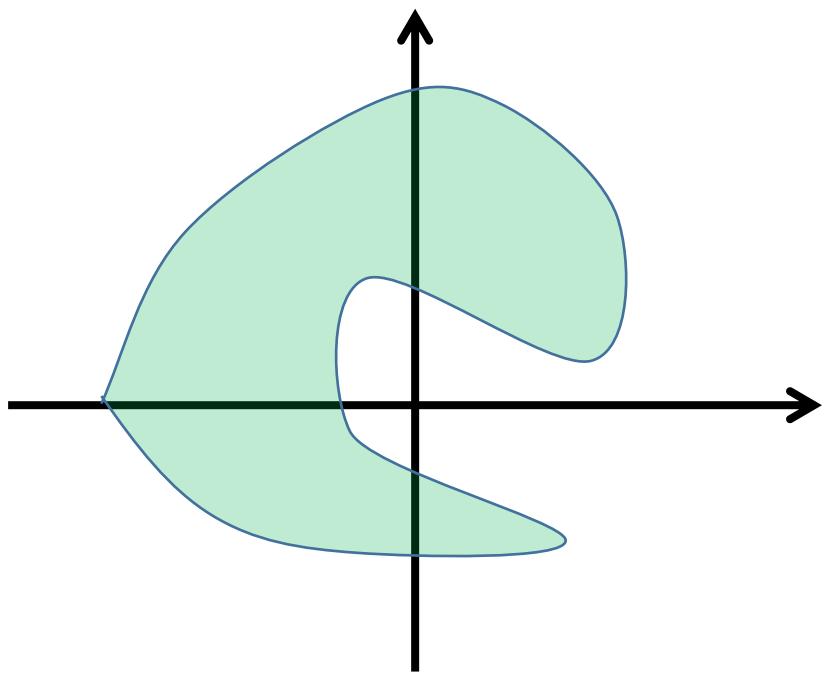
• The decision boundary is a **property of the hypothesis** Means we can create the boundary with the hypothesis and parameters without any data.

# Non Linear Decision boundary

- Get logistic regression to fit a complex non-linear data set
  - Like **polynomial regress** add higher order terms
- $h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$
- E.g.,  $\theta_0 = -1, \theta_1 = 0, \theta_2 = 0, \theta_3 = 1, \theta_4 = 1$
- Predict “ $y = 1$ ” if  $-1 + x_1^2 + x_2^2 \geq 0$



# Non Linear Decision boundary



- By adding more polynomial terms to simple hypothesis function, we will have more complex boundaries,
- $$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$
- The decision boundary is the property of the hypothesis and the parameters ( $\theta$ ) vector.

# Cost Function

Training set with  $m$  examples

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

Each example has is  $n+1$  length column vector

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

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

How to choose parameters  $\theta$ ?

# Cost function for Linear Regression

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \end{aligned}$$

cost that the learning algorithm will pay if the outcome is  $h_\theta(x)$  and the actual outcome is  $y$

Cost of Individual sample

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

# Limitation of Cost Function

- This Cost function for logistic regression will give a **non-convex function** for parameter optimization
- **Non convex?** In Logistic Regression, the hypothesis function  $h_{\theta}(x)$  is a non-linear sigmoid function of  $h_{\theta}(x)$  )
- If you use  $h_{\theta}(x)$  into the Cost() function, and plot the  $J(\theta)$ , then we will find many local optimum -> *non convex function*
- But, lots of local minima mean gradient descent may not find the global optimum - may get stuck in a local minimum. So, our objective is to obtain a convex function so that gradient descent can be applied for getting the a global minimum

# Logistic Regression cost function: Convex

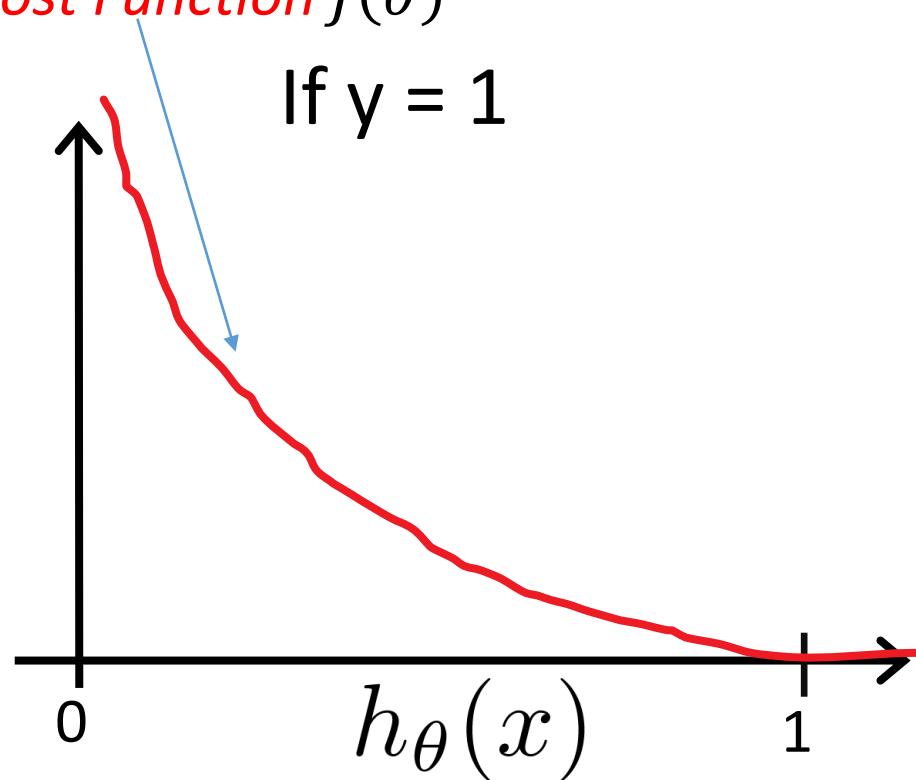
Different Cost Function

- To apply Gradient Descent, we need another CONVEX 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  $J(\theta)$

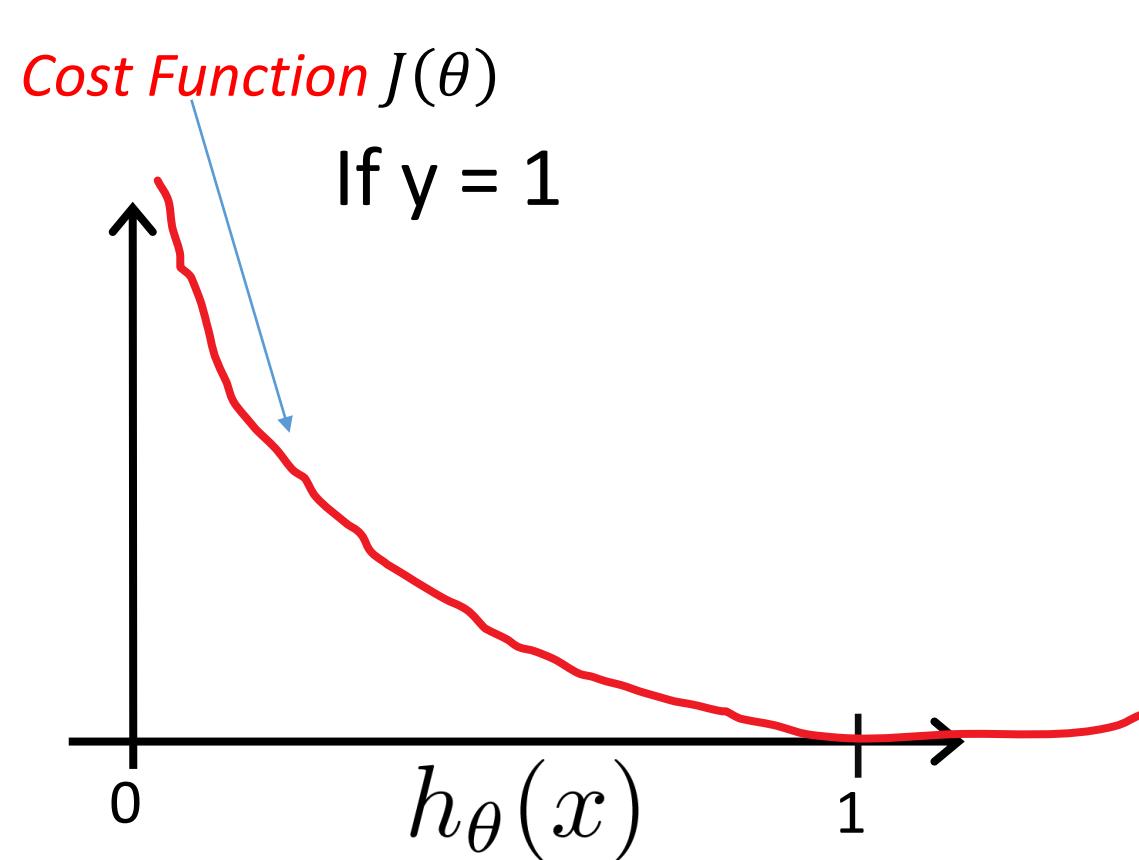
If  $y = 1$



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

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

# Logistic Regression cost function: Convex



## Important Property of Cost Function

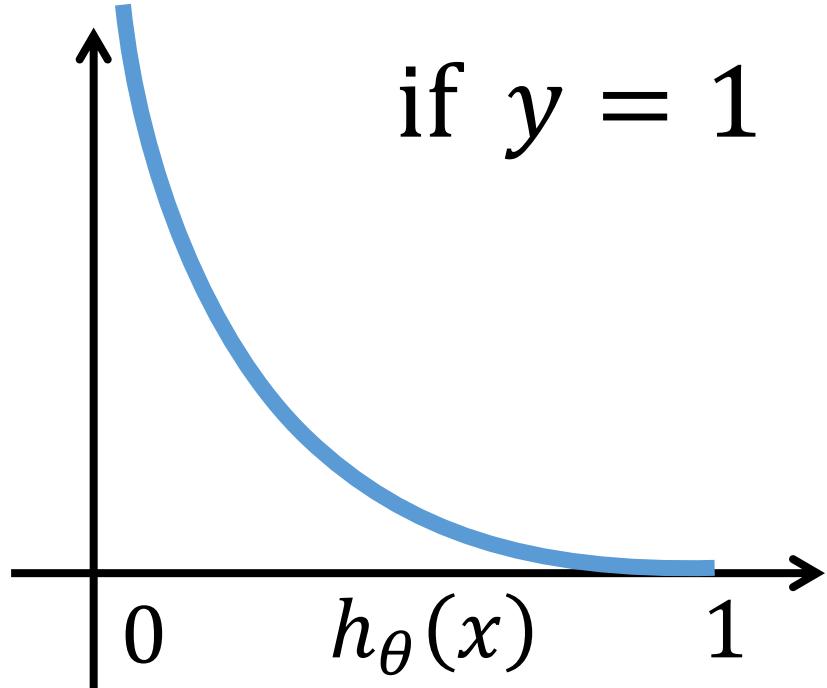
For  $Y = 1$

If  $h_\theta(x) = 1$  (Hypothesis predicts exactly 1 which is correct), corresponds to 0

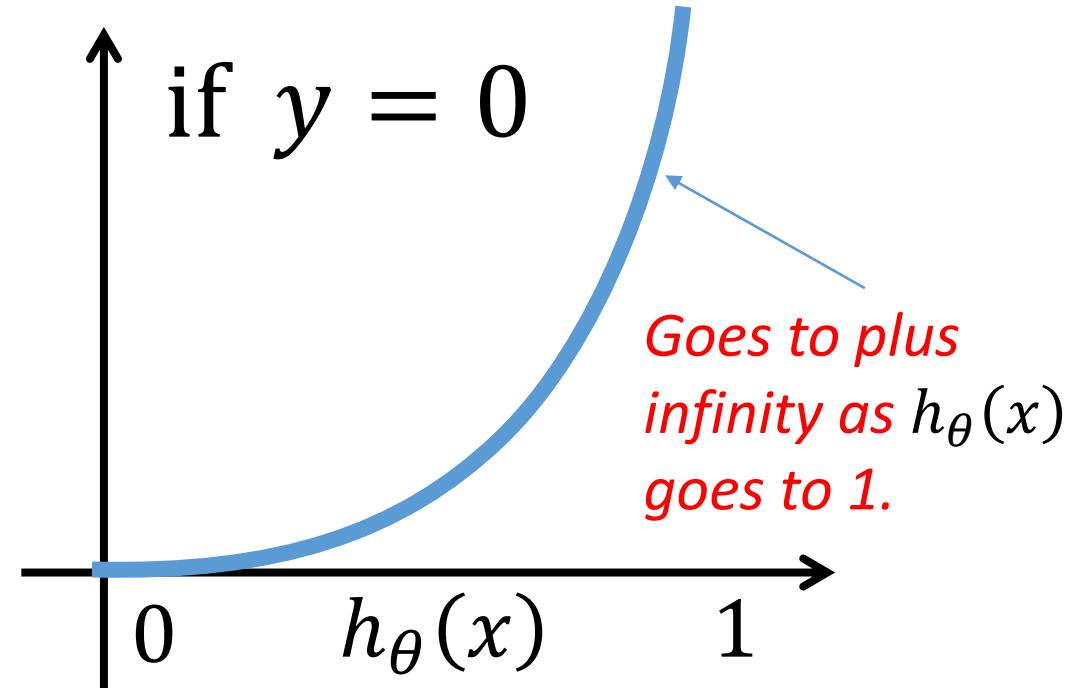
As,  $h\theta(x)$  goes to 0, then Cost goes to infinity.

# Cost function for Logistic Regression

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



if  $y = 1$



*Goes to plus infinity as  $h_\theta(x)$  goes to 1.*

# Simplified Cost function

*Cost for a Single Example*

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

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

*Binary Classification*

$$\cdot \text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

- If  $y = 1$ :  $\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x))$

- If  $y = 0$ :  $\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x))$

# Logistic regression Cost Function

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

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

**Learning:** fit parameters  $\theta$

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

**Prediction:** given new  $x$

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

# Minimizing Cost Function -Gradient descent

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

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

**Good news:** Convex function!

Repeat {

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

(Simultaneously update all  $\theta_j$ )

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

## Gradient descent for Linear Regression

Repeat {

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

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

}

## Gradient descent for Logistic Regression

Repeat {

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

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

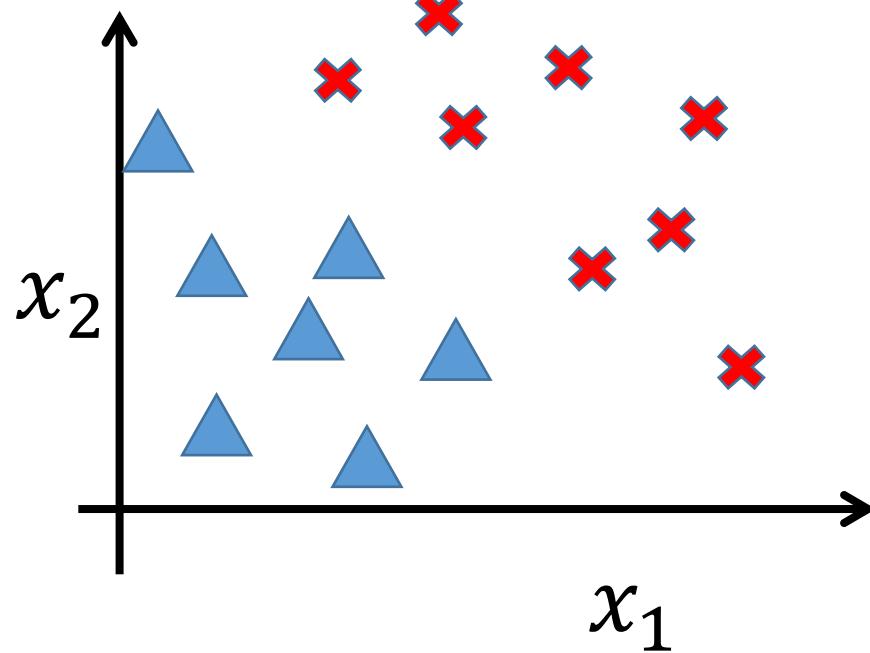
}

## Application of Logistic Regression in Multi-class classification

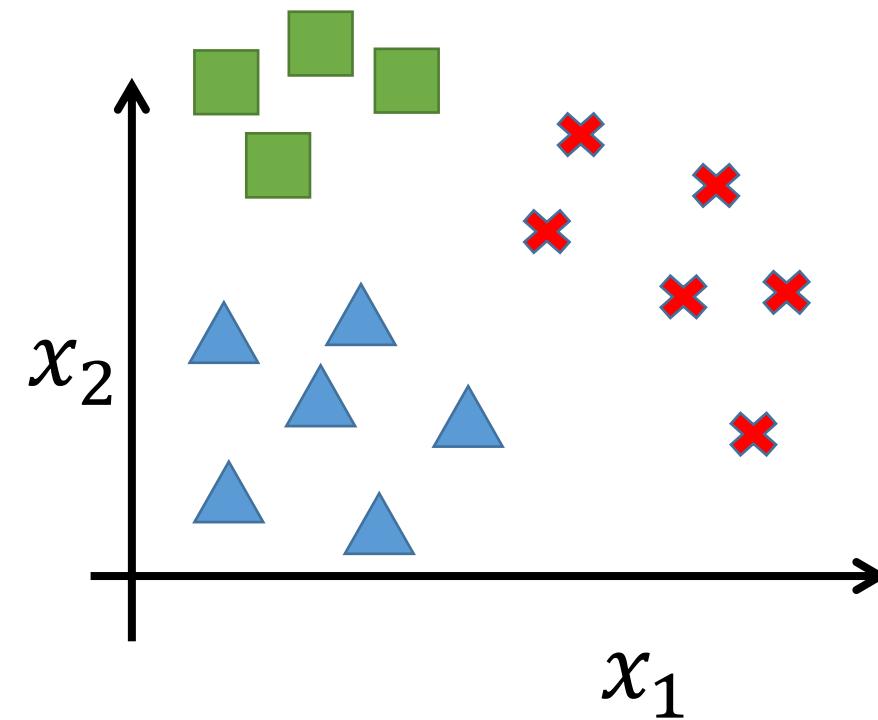
- Email foldering / tagging: Work, Friends, Family, Hobby
- Medical diagrams: Not ill, Cold, Flu
- Weather: Sunny, Cloudy, Rain, Snow

# Binary Classification Vs Multiclass Classification

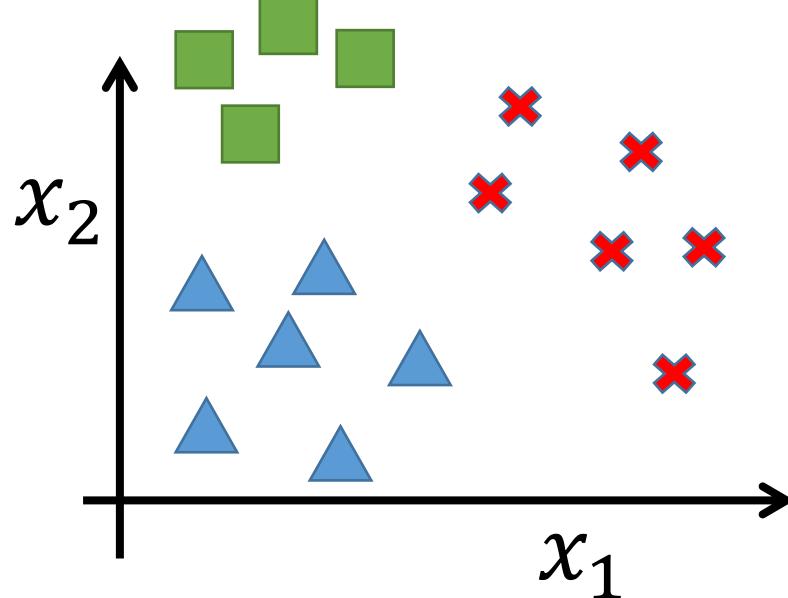
Binary classification



Multiclass classification



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

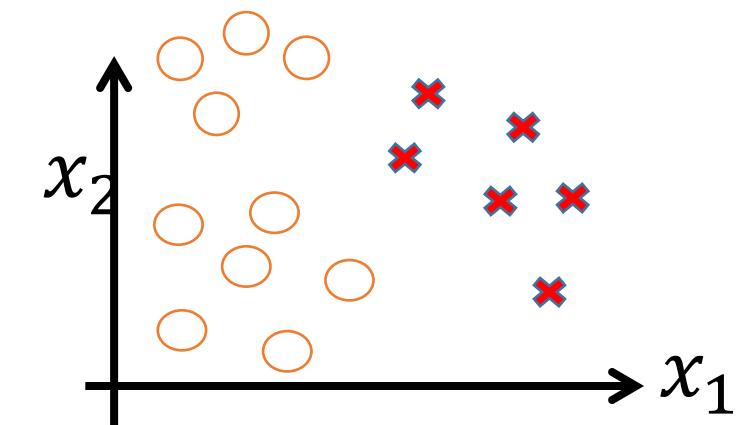
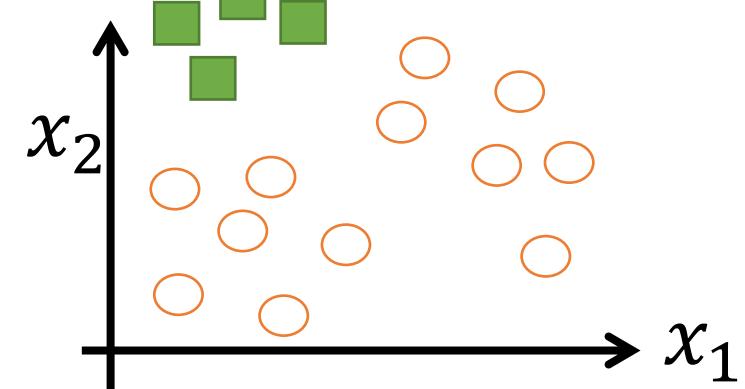
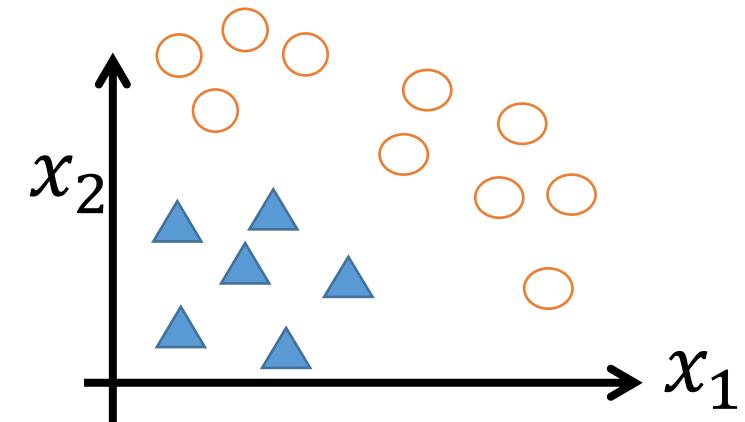
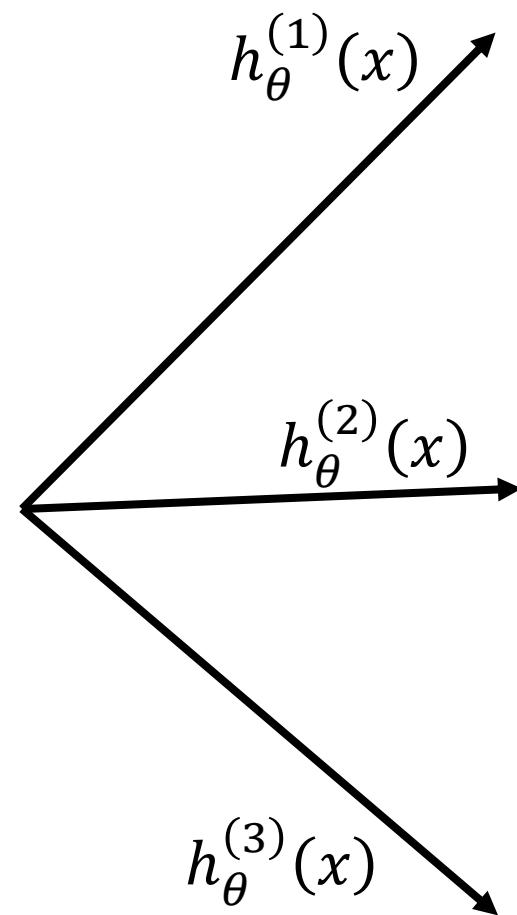


Class 1: ▲

Class 2: ■

Class 3: ✕

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



## One-vs-all

- Train a logistic regression classifier  $h_{\theta}^{(i)}(x)$  for each class  $i$  to predict the probability that  $y = i$
- Given a new input  $x$ , pick the class  $i$  that maximizes the probability that  $h_{\theta}^{(i)}(x) = 1$ 
$$\max_i h_{\theta}^{(i)}(x)$$

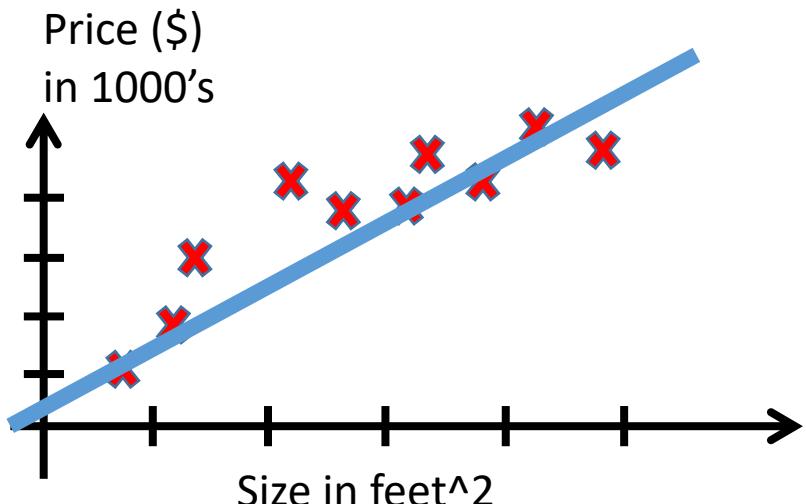
## Things to remember

- Hypothesis representation 
$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$
- Cost function 
$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases}$$
- Logistic regression with gradient descent 
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$
- Multi-class classification 
$$\max_i h_{\theta}^{(i)}(x)$$

# Regularization

- Overfitting
- Cost function
- Regularized linear regression
- Regularized logistic regression

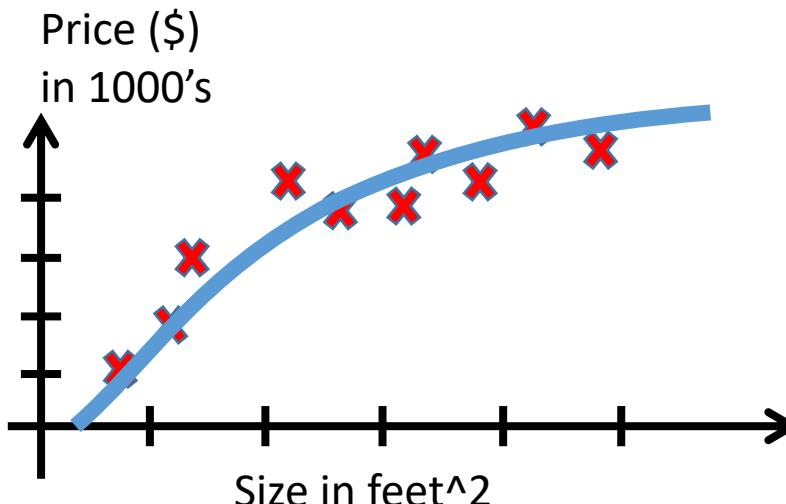
# Overfitting



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

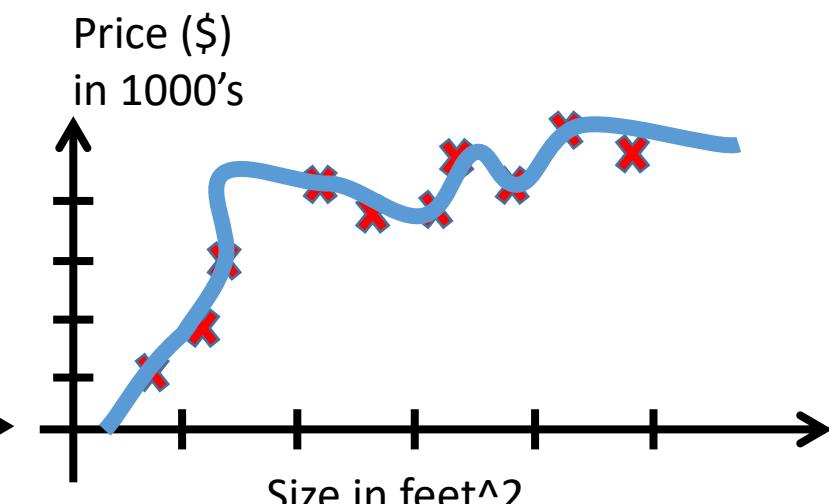
Underfitting

**High bias**



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

Just right



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

Overfitting

**High variance**

# Overfitting: Definition

- If we have too many features (i.e. complex model), the learned hypothesis may fit the training set very well

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$$

but fail to generalize to new examples  
(predict prices on new examples).

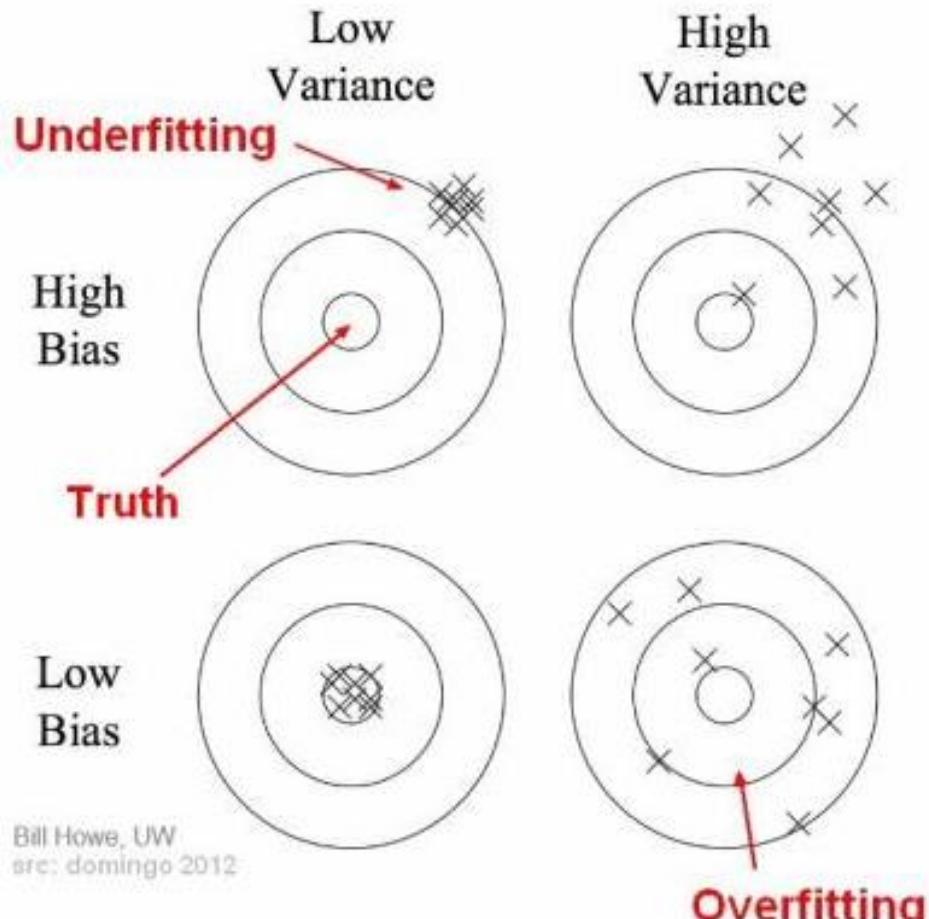
# Overfitting: Bias and Variance (Prediction Error)

- If we're fitting a straight line to the data, we have a strong preconception that **there should be a linear fit**. But, in this case, this is not correct and a straight line can not help to solve the problem. This restriction or preconception is called Bias.
- A type of error called variance occurs due to a model's sensitivity to small fluctuations in the training set.
- **Note:** if we have too many features then the learned hypothesis may give a cost function of exactly zero. But this tries too **hard to fit the training set and hence, fails to provide a *general* solution - unable to generalize**.

# Bias-Variance Tradeoff

- **Bias:** Bias is the **difference between the average prediction of our model and the correct value** which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. High bias model leads to high error on training and test data. It Decreases with more complex model.
- **Variance:** Variance is the variability of model prediction for a given data point. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.
- **Bias –Variance Tradeoff:** So, If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the **right/good balance without overfitting and underfitting the data.**

# Bias-Variance Tradeoff



**Bias and variance using bulls-eye diagram**

The **centre of the target** is a model that perfectly predicts correct values. As we move away from the bulls-eye our predictions become get worse and worse.

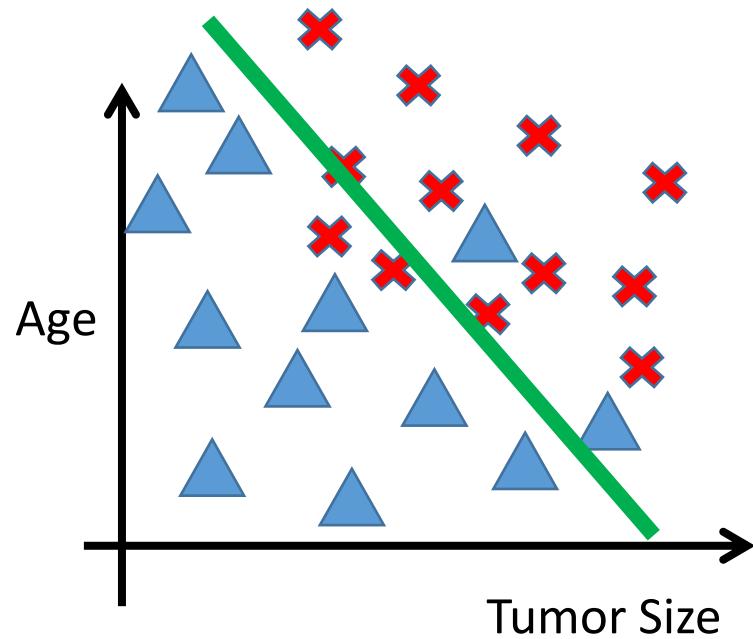
Underfitting happens when a model unable to capture (model is too simple to capture the pattern) the underlying pattern of the data. These models usually have high bias and low variance.

It happens i) have very less amount of data to build an accurate model, ii) Trying to build a linear model with a nonlinear data.

Overfitting happens when our model captures the noise along with the underlying pattern in data.

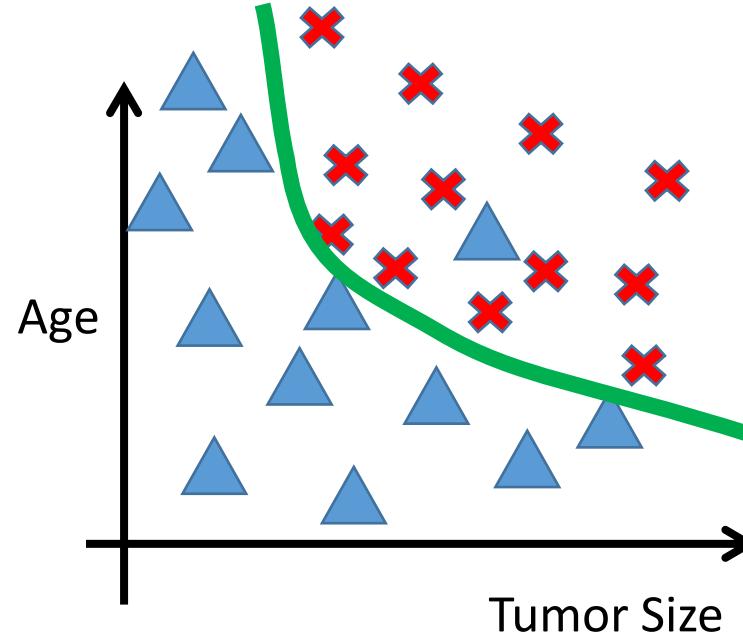
It happens i) when we train our model a lot over noisy dataset. These models have low bias and high variance. These models are very complex like Decision trees which are prone to overfitting.

# Overfitting (Logistic Regression)

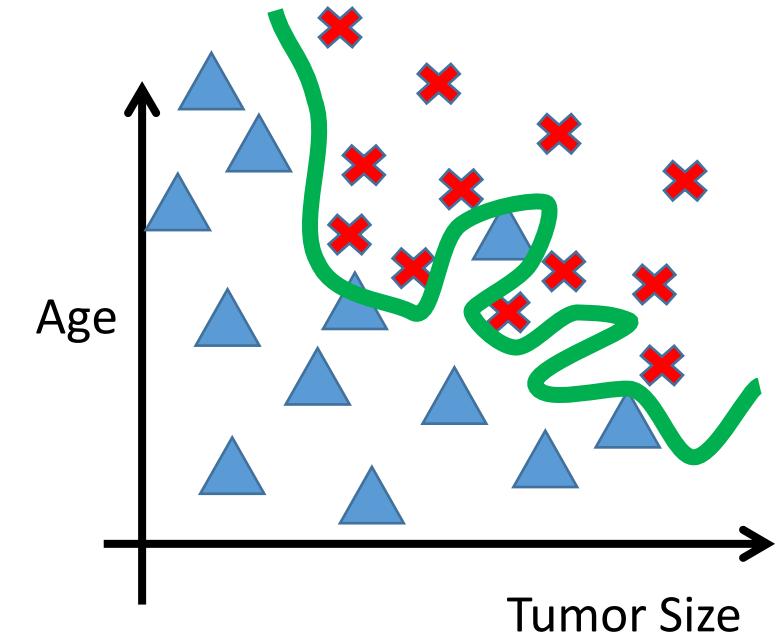


$$h_{\theta}(x) = g(\theta_0 + \theta_1 x + \theta_2 x_2)$$

Underfitting



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

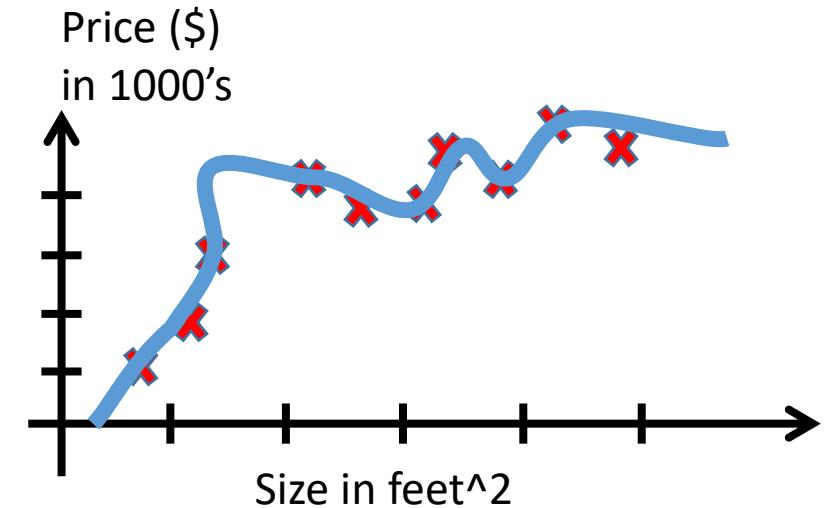


$$h_{\theta}(x) = g(\theta_0 + \theta_1 x + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \theta_6 x_1^3 x_2 + \theta_7 x_1 x_2^3 + \dots)$$

Overfitting

# Addressing Overfitting

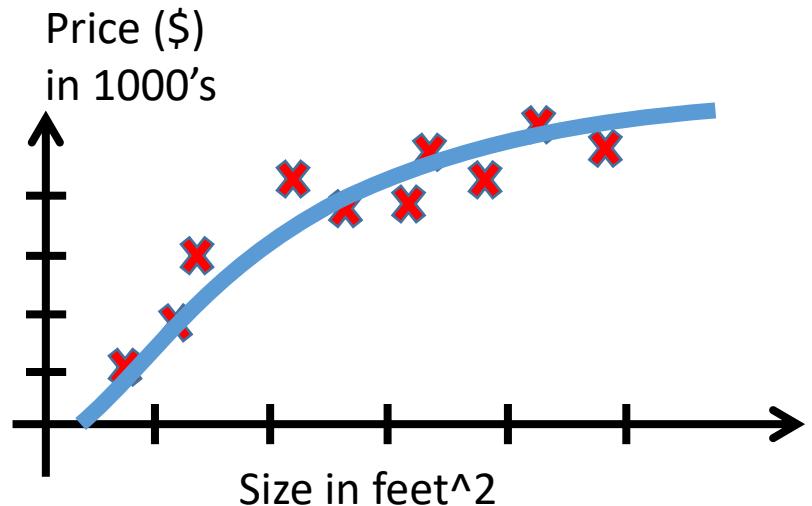
- How to decide **Overfitting?**
- Plotting Hypothesis – Too curvy?
- But, we have lots of feature in a polynomial, so plotting and visualizing the data to decide what feature to keep or drop – Challenging.



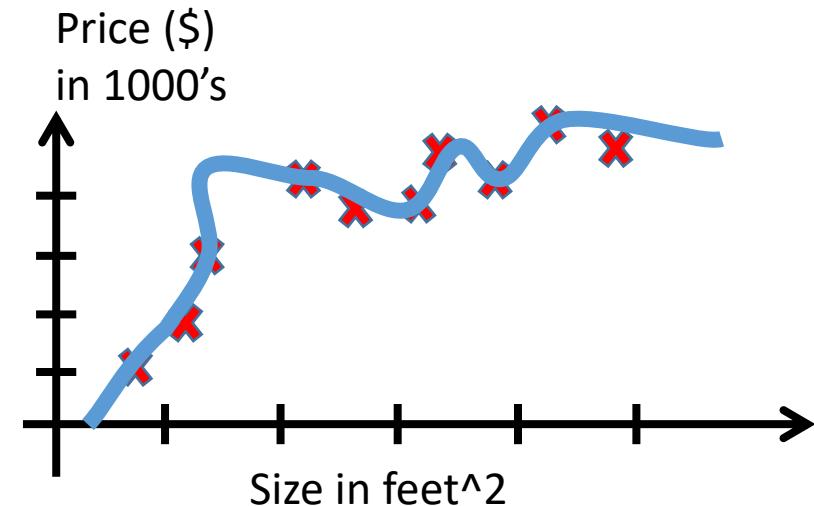
# Addressing overfitting

- **1. Reduce number of features.**
  - Manually select which features to keep.
  - Model selection algorithm (which features to keep and which features to remove from 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  $y$ .

# Cost function Optimization for Regularization



$$h_\theta(x) = \theta_0 + \theta_1x + \theta_2x^2$$



$$h_\theta(x) = \theta_0 + \theta_1x + \theta_2x^2 + \theta_3x^3 + \theta_4x^4$$

- Suppose we penalize and make  $\theta_3, \theta_4$  really small.

$$\min_{\theta} J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000 \theta_3^2 + 1000 \theta_4^2$$

# Regularization.

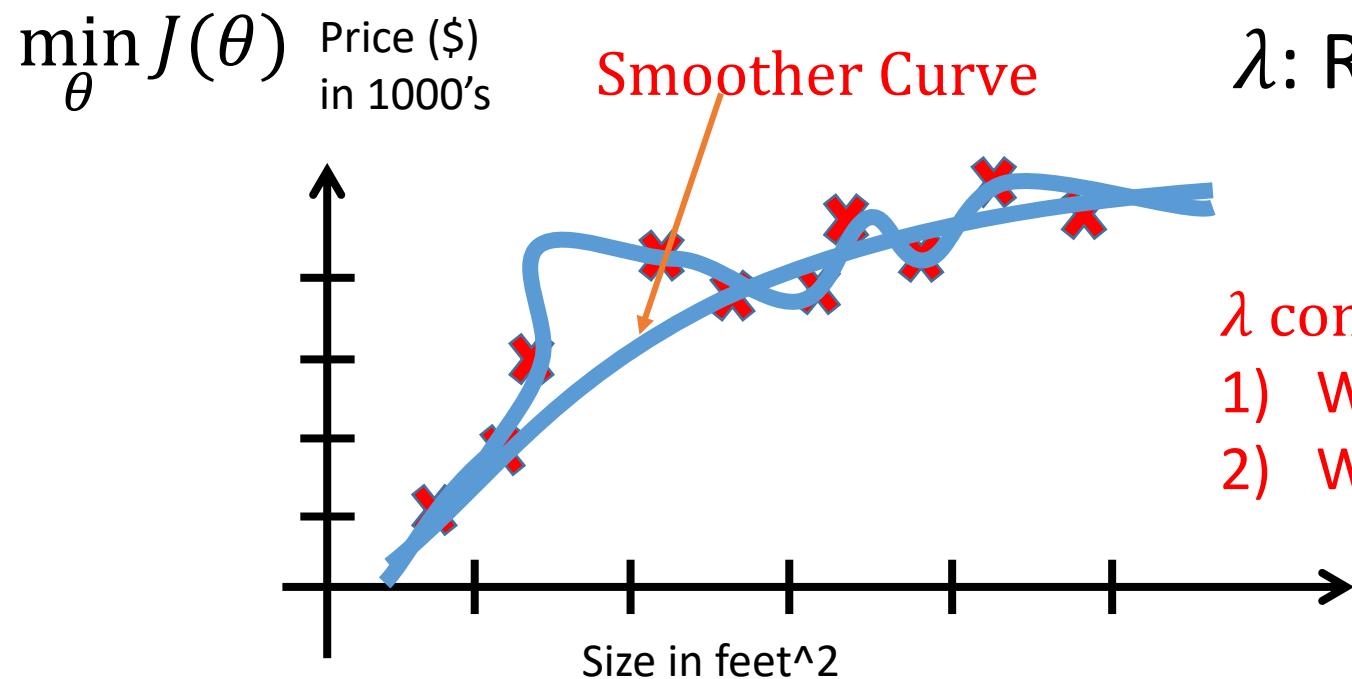
- Small values for parameters  $\theta_1, \theta_2, \dots, \theta_n$ 
  - “Simpler” hypothesis
  - Less prone to overfitting
- Housing:
  - Features:  $x_1, x_2, \dots, x_{100}$
  - Parameters:  $\theta_0, \theta_1, \theta_2, \dots, \theta_{100}$

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

Cannot Penalize  $\theta_0$ ,  
hence starts from  $\theta_1$

# Regularization

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



$\lambda$ : Regularization parameter

$\lambda$  controls a tradeoff between our two goals:

- 1) Want to fit the training set well
- 2) Want to keep the parameters small

# Question

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

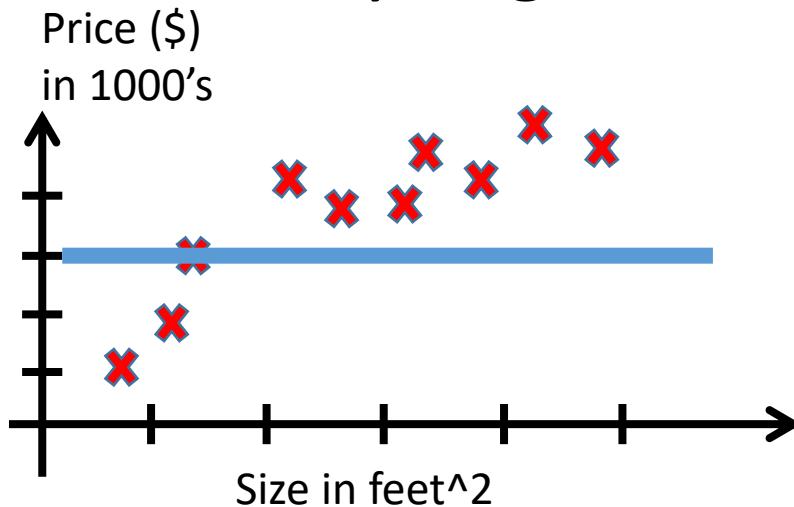
What if  $\lambda$  is set to an extremely large value (say  $\lambda = 10^{10}$ )?

1. Algorithm works fine; setting to be very large can't hurt it
2. Algorithm fails to eliminate overfitting.
3. Algorithm results in underfitting. (Fails to fit even training data well).
4. Gradient descent will fail to converge.

# Question

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

What if  $\lambda$  is set to an extremely large value (say  $\lambda = 10^{10}$ )?



$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n = \theta^\top x$$

# Regularized Linear Regression

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

$n$ : Number of features

$\theta_0$  is not penalized

# Gradient Descent (Previously)

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \quad (j = 0)$$

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

}

# Gradient Descent (Regularized)

Repeat {

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

Big term

$$\theta_j := \theta_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right], \quad j = 1, 2, 3, \dots, n$$

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

}

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

# Comparison

$1 - \alpha \frac{\lambda}{m} < 1$ : Weight decay

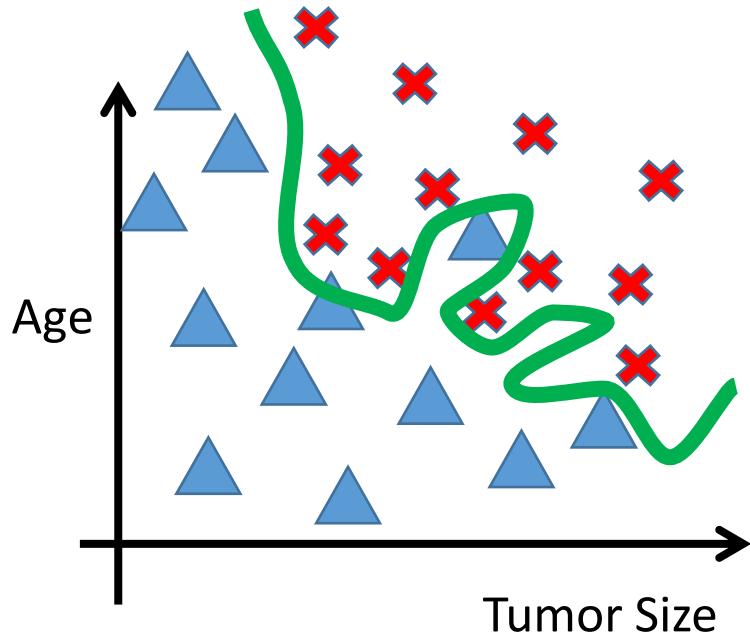
## Regularized linear regression

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

## Un-regularized linear regression

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

# Regularized Logistic Regression



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \theta_6 x_1^3 x_2 + \theta_7 x_1 x_2^3 + \dots)$$

- Cost function:

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

# Gradient descent (Regularized)

Repeat {

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

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

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

}

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

## $|\theta|_1$ : Lasso regularization

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

LASSO: Least Absolute Shrinkage and Selection Operator

# Terminology

## Regularization function

$$\|\theta\|_2^2 = \sum_{j=1}^n \theta_j^2$$

$$\|\theta\|_1 = \sum_{j=1}^n |\theta_j|$$

$$\alpha \|\theta\|_1 + (1 - \alpha) \|\theta\|_2^2$$

## Name

Tikhonov regularization

Ridge regression

LASSO regression

Elastic net regularization

## Solver

Close form

Proximal gradient descent, least angle regression

Proximal gradient descent

**Course code:** CSE3008

**Course Title:** Introduction to Machine Learning

**Module 2: Training Models**

**Topic 2: Instance Based Learning**

**Dr. Usha Rani Gogoi**

**Assistant Professor Sr. Grade 1**

**[usha.gogoi@vitap.ac.in](mailto:usha.gogoi@vitap.ac.in)**

**SCOPE**

# Content

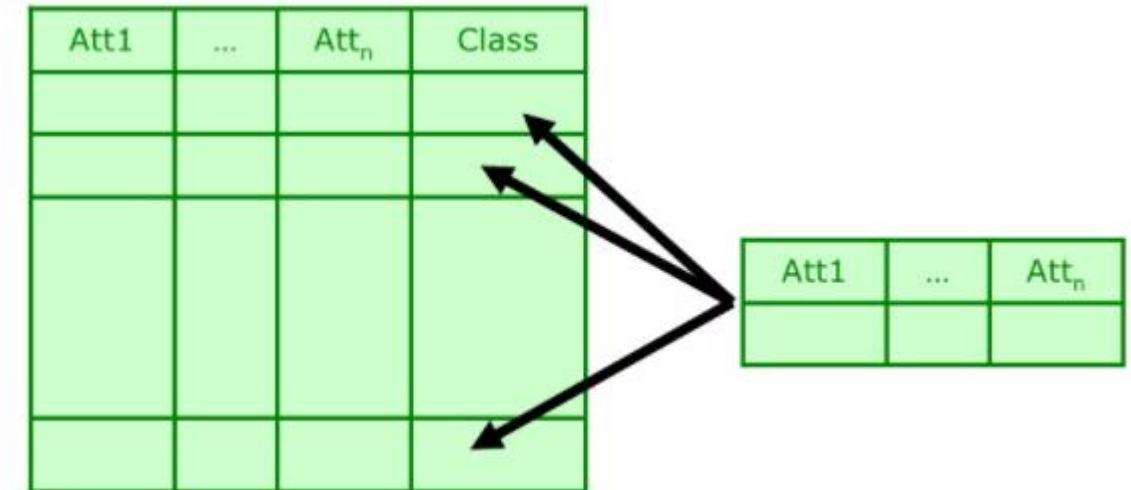
- Instance Based Learning
- K-Nearest Neighbor (kNN)
  - Distance metric
  - Pros/Cons of nearest neighbor
- Validation, cross-validation, hyperparameter tuning

# Instance Based Learning

- Instance based learning is often termed lazy learning as there is typically no “**Transformation**” of training instances into more general “**statements**”.
- Instead the presented training data is simply stored and when new query instance is encountered, a set of similar, related instances is retrieved from memory and used to classify the new query instance.
- Hence, instance based learners never form an **explicit general hypothesis regarding the target function**. They simply compute the classification of each new query instance as needed.
- Instance themselves represent the knowledge. A **similarity (distance)** function defines what is “learned”

# Instance Based Classifier

- Store the training records.
- Use the training records to predict the class label of unseen cases.



# Instance Based Classifier

- **Rote Learner**
  - Memorize entire training data
  - Performs classification only if attributes of record match one of the training examples exactly.
- **Nearest Neighbor**
  - Uses k “closest” points (nearest neighbours) for performing classification.

# When to use Nearest Neighbourhood Algorithm

- Instances map to points in  $R^n$
- Less than 20 attributes per instance
- Lots of training data

# Nearest neighbor classifier

- **Training data**

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

- **Learning**

Do nothing.

- **Testing**

$$h(x) = y^{(k)}, \text{ where } k = \operatorname{argmin}_i D(x, x^{(i)})$$

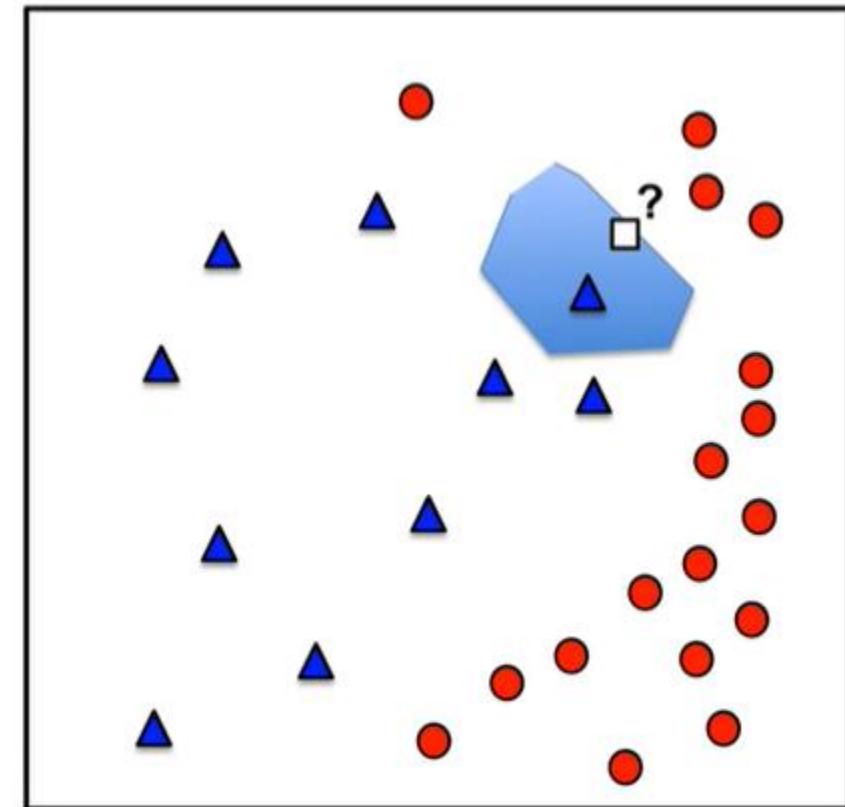
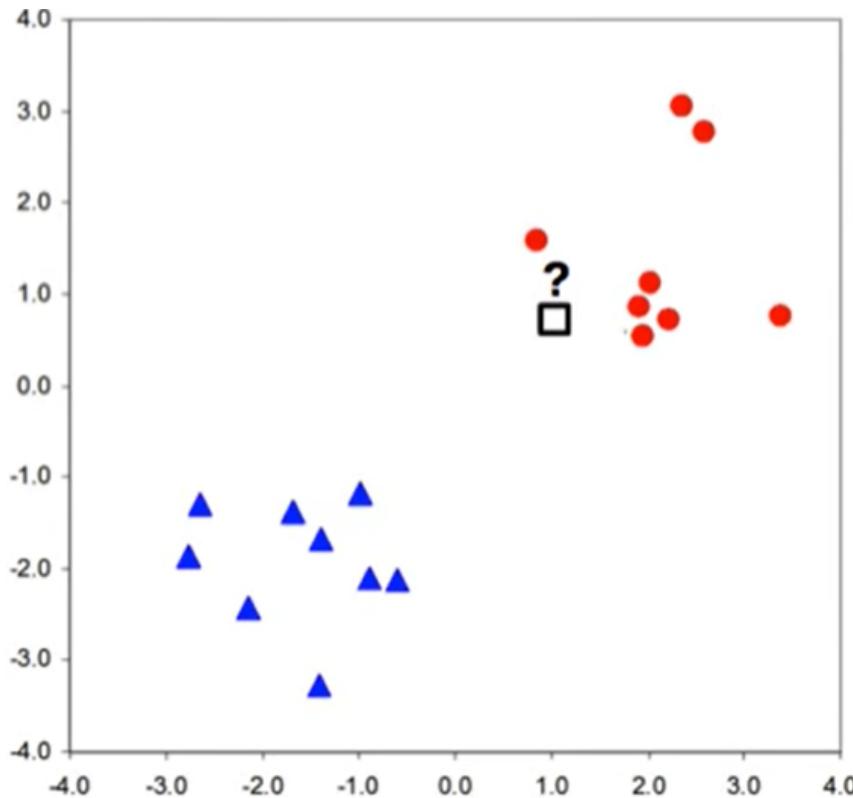


# KNN Approach

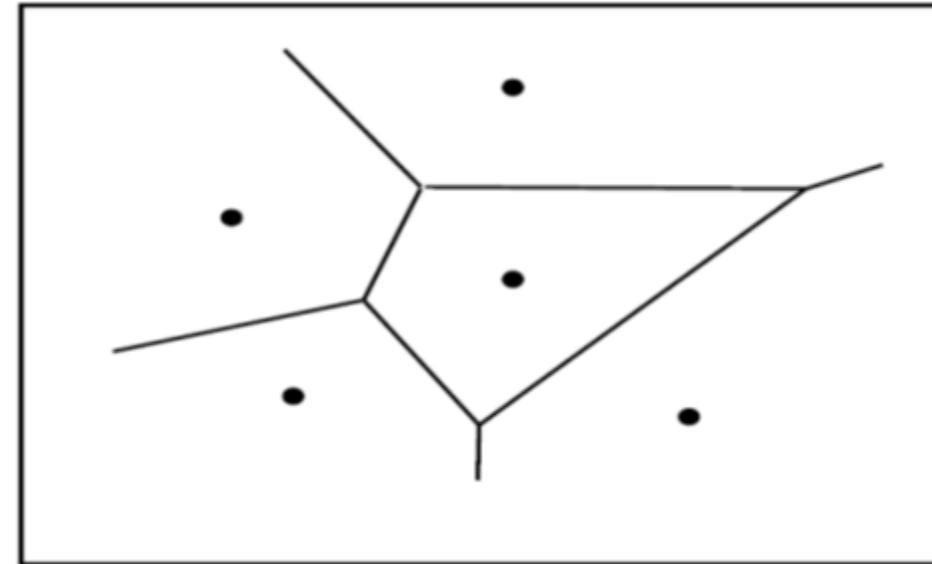
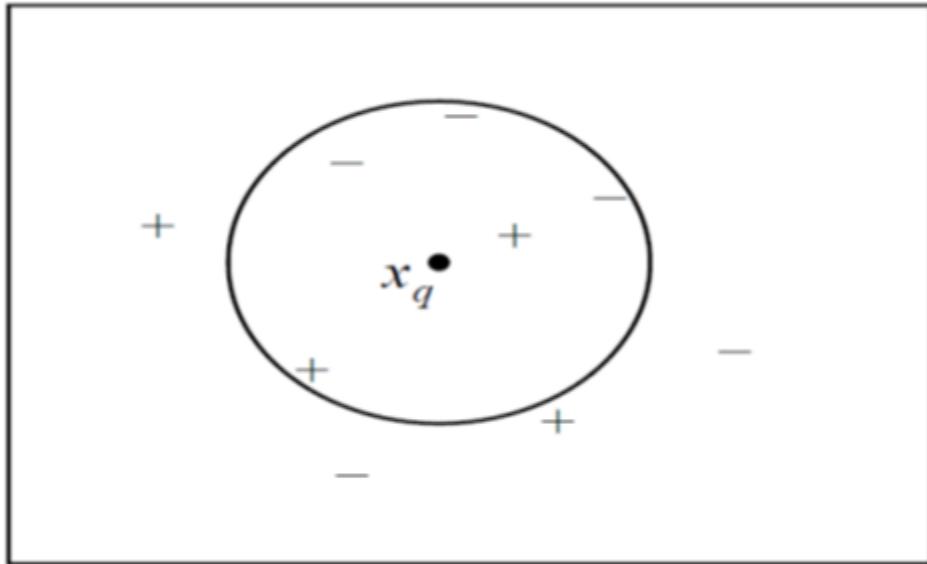
- The simplest, most used instance based learning algorithm is the k-NN algorithm.
- K-NN assumes that all instances are points in some n-dimensional space and defines neighbors in terms of distance (Usually Euclidean in R-Space)
- K is the number of neighbors considered to make a decision.

# Intuition for KNN

Nearby things should have the same class.

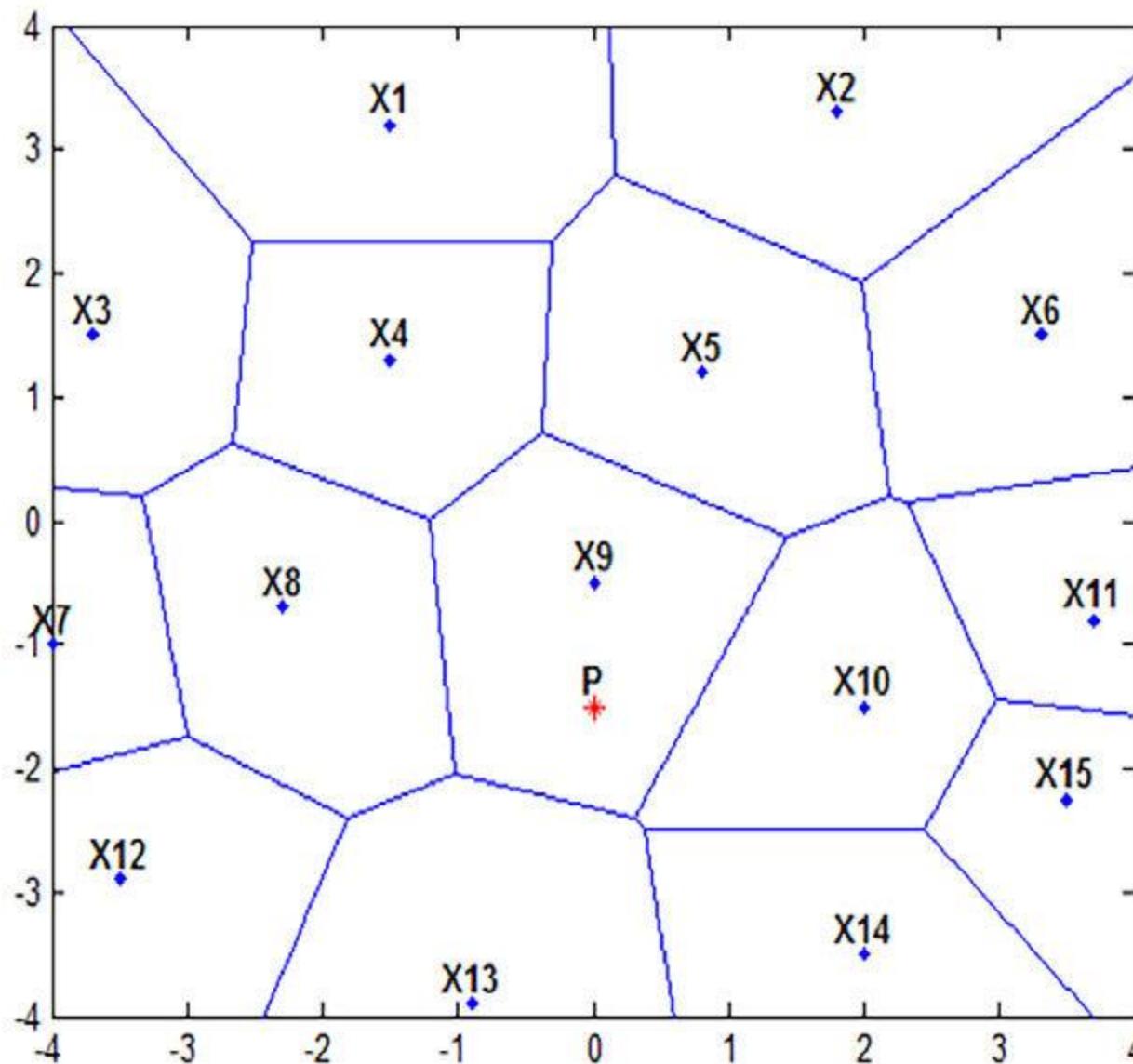


# Voronoi Diagram



k-NEAREST NEIGHBOR A set of positive and negative training examples is shown on the left, along with a query instance  $x_q$ , to be classified. The 1-NEAREST NEIGHBOR algorithm classifies  $x_q$ , positive, whereas 5-NEAREST NEIGHBOR classifies it as negative. On the right is the decision surface induced by the 1-NEAREST NEIGHBOR algorithm for a typical set of training examples. The convex polygon surrounding each training example indicates the region of instance space closest to that point

# Graphic Depiction



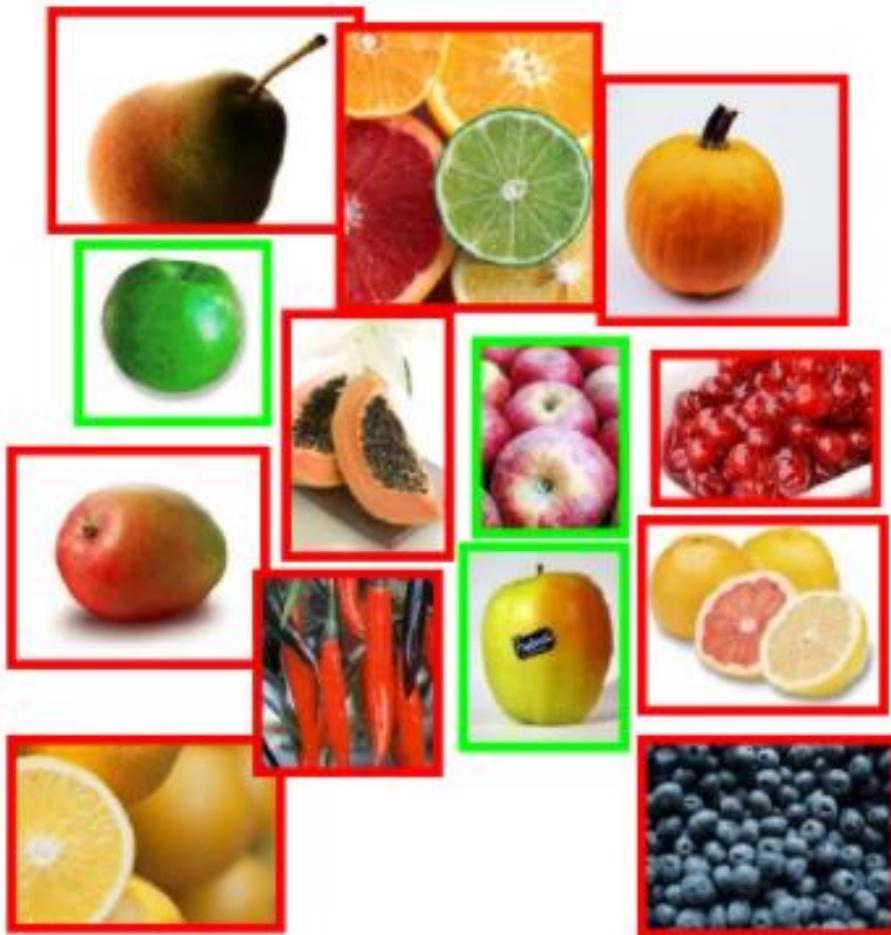
- All possible points within a sample's Voronoi cell are the **Nearest Neighbouring points** for that sample.
- For any sample, the nearest sample is determined by the closest Voronoi cell edge.

## Basic Idea

- The k-NN classification rule is to assign to a test sample the majority category label of its  $k$  nearest training samples.
- In practice,  $k$  is usually chosen to be odd, so as to avoid ties.
- The  $k = 1$  rule is generally called the nearest neighbour classification rule.

# KNN Approach

Apples again!



Is this an apple?

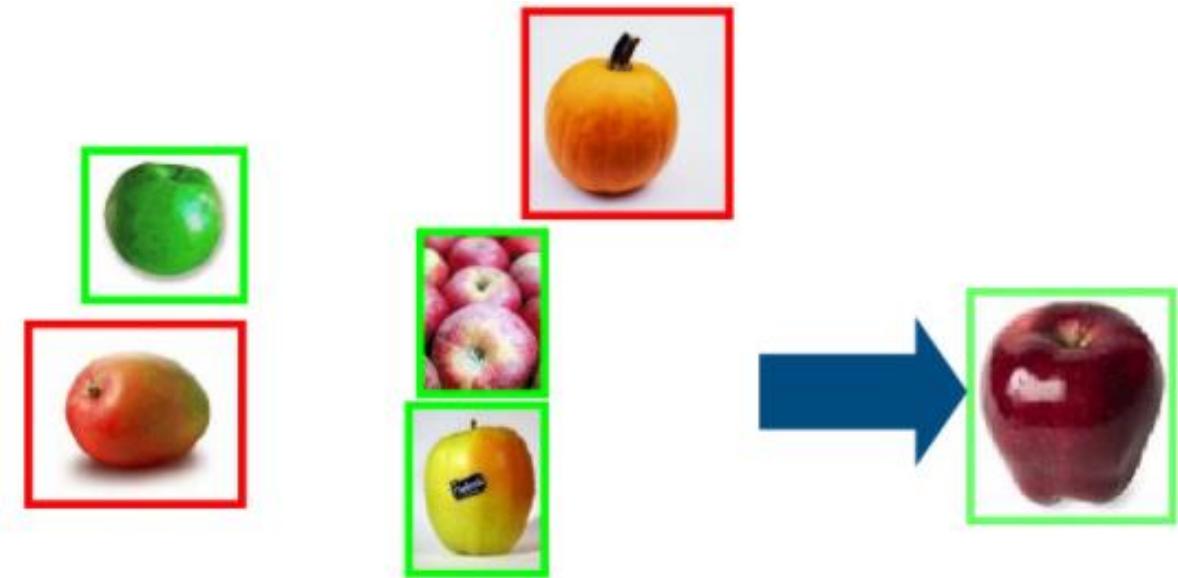


To decide whether **an unseen fruit is an apple**, consider the **k fruits that are more similar** to the unknown fruit.

Then, classify the unknown fruit using the most frequent class.

# An Simple Way to Classify a New Fruit

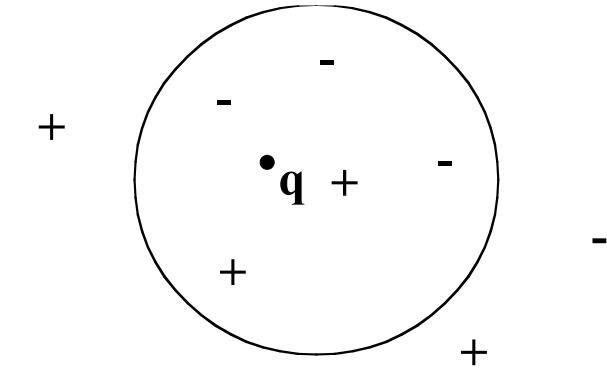
- If  $k=5$ , these 5 fruits are the most similar ones to the unclassified example.
- Since, the majority of the fruits are apples, we decide that the unknown fruit is an apple



# k-NN Algorithm

- For each training instance  $t = (x, f(x))$ 
  - Add  $t$  to the set  $Tr\_instances$
- Given a query instance  $q$  to be classified
  - Let  $x_1, \dots, x_k$  be the  $k$  training instances in  $Tr\_instances$  nearest to  $q$
  - Return

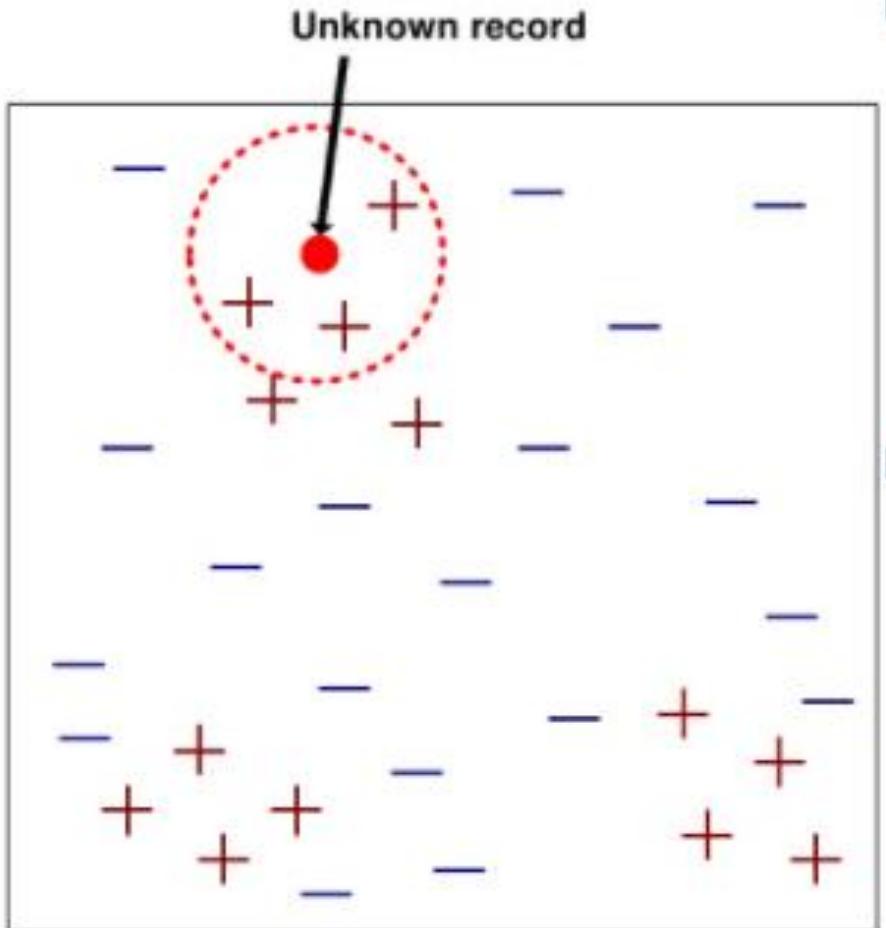
$$\hat{f}(q) = \arg \max_{v \in V} \sum_{i=1}^k \delta(v, f(x_i))$$



Among these  $k$  neighbors,  
count the number of the  
data points in each category.

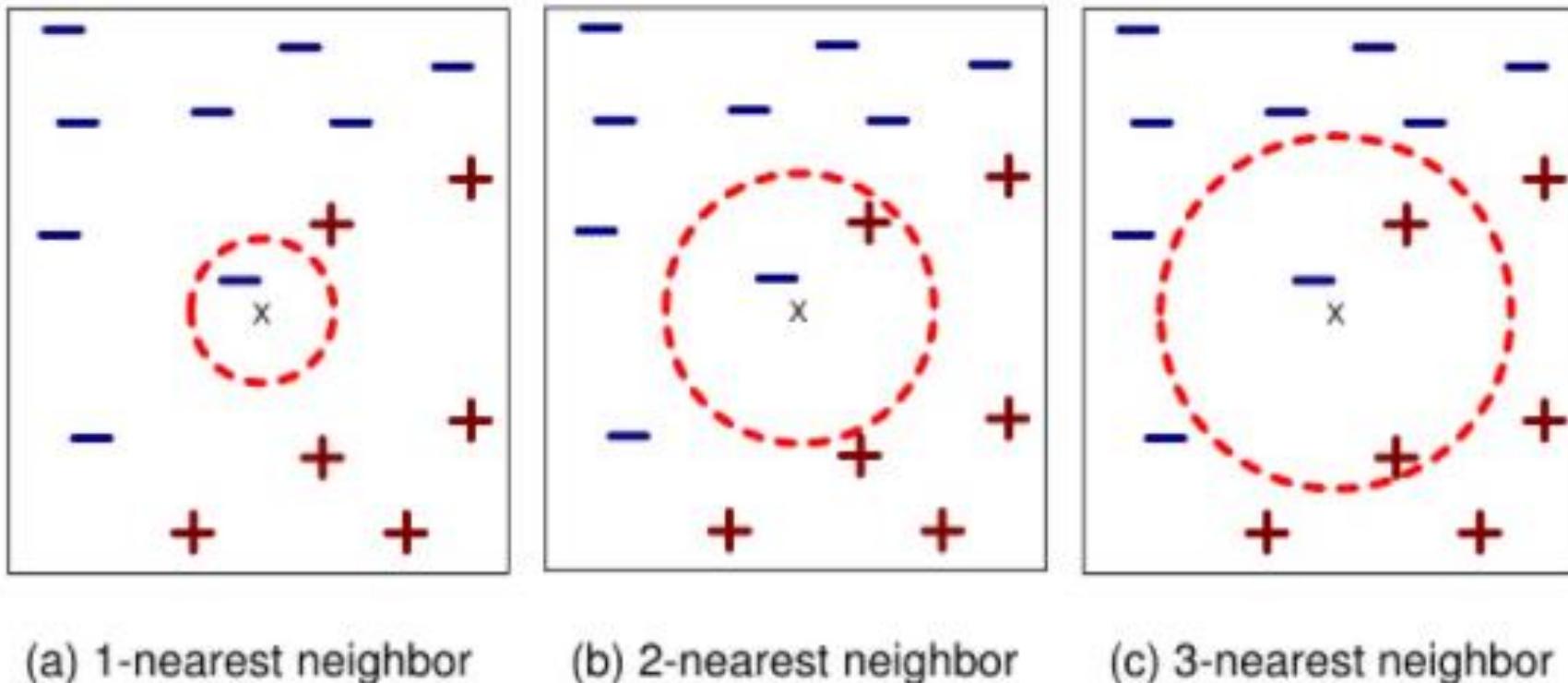
- Where  $V$  is the finite set of target class values, and  $\delta(a,b)=1$  if  $a=b$ , and 0 otherwise (Kronecker function)
- Intuitively, the  $k$ -NN algorithm assigns to each new query instance the majority class among its  $k$  nearest neighbors

# Requisite of k-NN Algorithm



- **Requires 3 Things**
  - i) The set of stored records
  - ii) Distance Metric to compute distance between records.
  - iii) The value of  $k$ , the number of nearest neighbors to retrieve.
  
- **To classify an unknown record:**
  - i) Compute distance to other training records
  - ii) Identify  $k$  nearest neighbors
  - iii) Use class labels of nearest neighbors to determine the class label of unknown record (e.g. by taking majority vote)

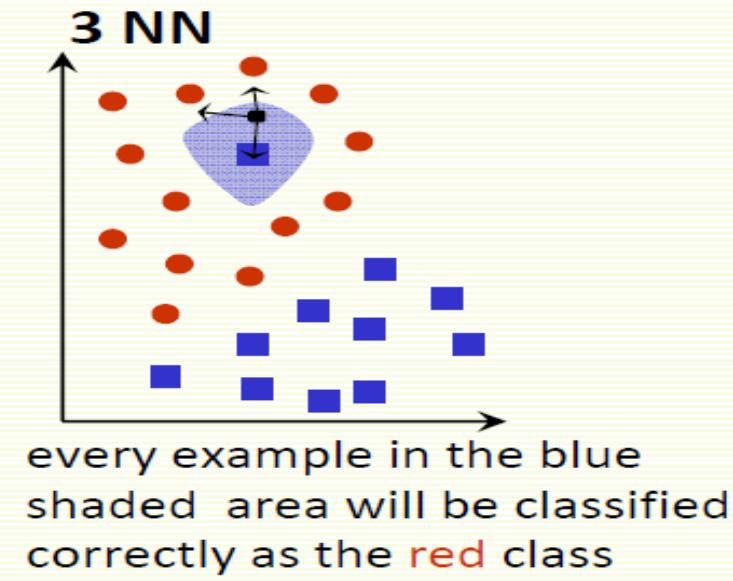
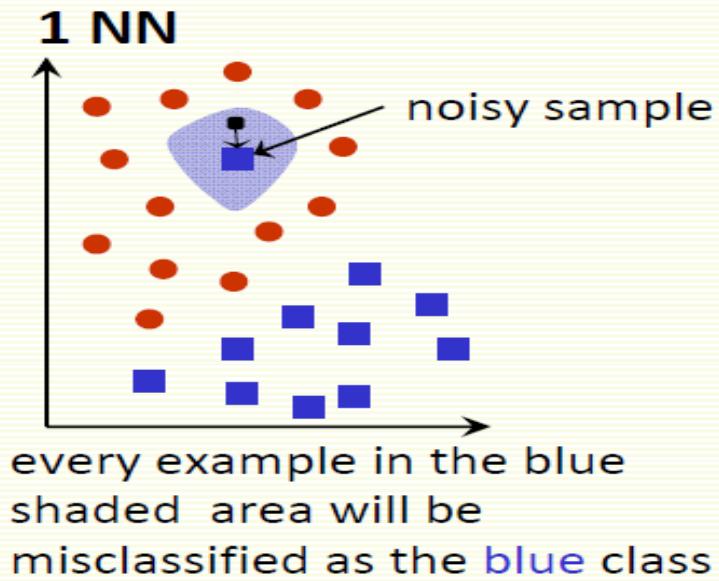
# Requisite of k-NN Algorithm



K-nearest neighbors of a record  $x$  are data points that have the  $k$  smallest distance to  $x$

# kNN: How to Choose k?

- Rule of thumb is  $k = \sqrt{n}$ ,  $n$  is number of examples
  - interesting theoretical properties
- In practice,  $k = 1$  is often used for efficiency, but can be sensitive to “noise”



# Example

- We have data from the questionnaires survey and objective testing with two attributes (acid durability and strength) to classify whether a special paper tissue is good or not. Here are four training samples.

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Y = Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Now the factory produces a new paper tissue that passes the laboratory test with  $X1 = 3$  and  $X2 = 7$ . Guess the classification of this new tissue.

# Distance Metrics

Minkowsky:	Euclidean:	Manhattan / city-block:
$D(\mathbf{x}, \mathbf{y}) = \left( \sum_{i=1}^m  x_i - y_i ^r \right)^{1/r}$	$D(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$	$D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m  x_i - y_i $
<b>Camberra:</b> $D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m \frac{ x_i - y_i }{ x_i + y_i }$	<b>Chebychev:</b> $D(\mathbf{x}, \mathbf{y}) = \max_{i=1}^m  x_i - y_i $	
<b>Quadratic:</b> $D(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T Q (\mathbf{x} - \mathbf{y}) = \sum_{j=1}^m \left( \sum_{i=1}^m (x_i - y_i) q_{ji} \right) (x_j - y_j)$ Q is a problem-specific positive definite $m \times m$ weight matrix		
<b>Mahalanobis:</b> $D(\mathbf{x}, \mathbf{y}) = [\det V]^{1/m} (\mathbf{x} - \mathbf{y})^T V^{-1} (\mathbf{x} - \mathbf{y})$		V is the covariance matrix of $A_1..A_m$ , and $A_j$ is the vector of values for attribute $j$ occurring in the training set instances 1..n.
<b>Correlation:</b> $D(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^m (x_i - \bar{x}_i)(y_i - \bar{y}_i)}{\sqrt{\sum_{i=1}^m (x_i - \bar{x}_i)^2 \sum_{i=1}^m (y_i - \bar{y}_i)^2}}$		$\bar{x}_i = \bar{y}_i$ and is the average value for attribute $i$ occurring in the training set.
<b>Chi-square:</b> $D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m \frac{1}{sum_i} \left( \frac{x_i}{size_x} - \frac{y_i}{size_y} \right)^2$		$sum_i$ is the sum of all values for attribute $i$ occurring in the training set, and $size_x$ is the sum of all values in the vector $\mathbf{x}$ .
<b>Kendall's Rank Correlation:</b> $D(\mathbf{x}, \mathbf{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^m \sum_{j=1}^{i-1} sign(x_i - x_j) sign(y_i - y_j)$ sign( $x$ )=-1, 0 or 1 if $x < 0$ , $x = 0$ , or $x > 0$ , respectively.		

Figure 1. Equations of selected distance functions.  
( $\mathbf{x}$  and  $\mathbf{y}$  are vectors of  $m$  attribute values).

- **Step 1 : Initialize and Define k.**

Lets say,  $k = 3$

(Always choose k as an odd number if the number of attributes is even to avoid a tie in the class prediction)

- **Step 2 : Compute the distance between input sample and training sample**

- Co-ordinate of the input sample is (3,7).

- Instead of calculating the Euclidean distance, we calculate the Squared Euclidean distance.

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Squared Euclidean distance
7	7	$(7-3)^2 + (7-7)^2 = 16$
7	4	$(7-3)^2 + (4-7)^2 = 25$
3	4	$(3-3)^2 + (4-7)^2 = 09$
1	4	$(1-3)^2 + (4-7)^2 = 13$

- Step 3 : Sort the distance and determine the nearest neighbours based of the K<sup>th</sup> minimum distance :**

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Squared Euclidean distance	Rank minimum distance	Is it included in 3-Nearest Neighbour?
7	7	16	3	Yes
7	4	25	4	No
3	4	09	1	Yes
1	4	13	2	Yes

- Step 4 : **Take 3-Nearest Neighbours:**
- Gather the category Y of the nearest neighbours.

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Squared Euclidean distance	Rank minimum distance	Is it included in 3-Nearest Neighbour?	Y = Category of the nearest neighbour
7	7	16	3	Yes	Bad
7	4	25	4	No	-
3	4	09	1	Yes	Good
1	4	13	2	Yes	Good

- Step 5 : **Apply simple majority**
- Use simple majority of the category of the nearest neighbours as the prediction value of the query instance.
- We have 2 “good” and 1 “bad”. Thus we conclude that the new paper tissue that passes the laboratory test with  $X_1 = 3$  and  $X_2 = 7$  is included in the “good” category.

# Scale Effects

- Different features may have different measurement scales
  - E.g., patient weight in kg (range [50,200]) vs. blood protein values in ng/dL (range [-3,3])
- Consequences
  - Patient weight will have a much greater influence on the distance between samples
  - May bias the performance of the classifier
- Hence, features may have to be scaled to prevent distance measures from being dominated by one of the features.

# Standardization

- Transform raw feature values into z-scores

$$z_{ij} = \frac{x_{ij} - m_j}{s_j}$$

- $x_{ij}$  is the value for the  $i^{th}$  sample and  $j^{th}$  feature
- $m_j$  is the average of all  $x_{ij}$  for feature  $j$
- $s_j$  is the standard deviation of all  $x_{ij}$  over all input samples
- Range and scale of z-scores should be similar (providing distributions of raw feature values are alike)

# Application of KNN

- $k$ -NN works well on many practical problems and is fairly noise tolerant (depending on the value of  $k$ )
  - If  $k$  is too small, then model is sensitive to noise points
  - If  $k$  is too large, neighborhood may include points from other classes
    - Used in classification
    - Used to get missing values
    - Used in pattern recognition
    - Used in gene expression
    - Used in protein-protein prediction
    - Used to get 3D structure of protein
    - Used to measure document similarity

## Advantages & How is kNN Incremental?

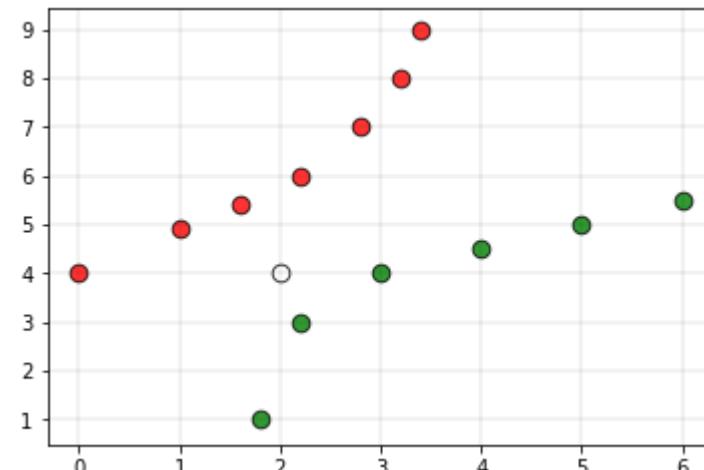
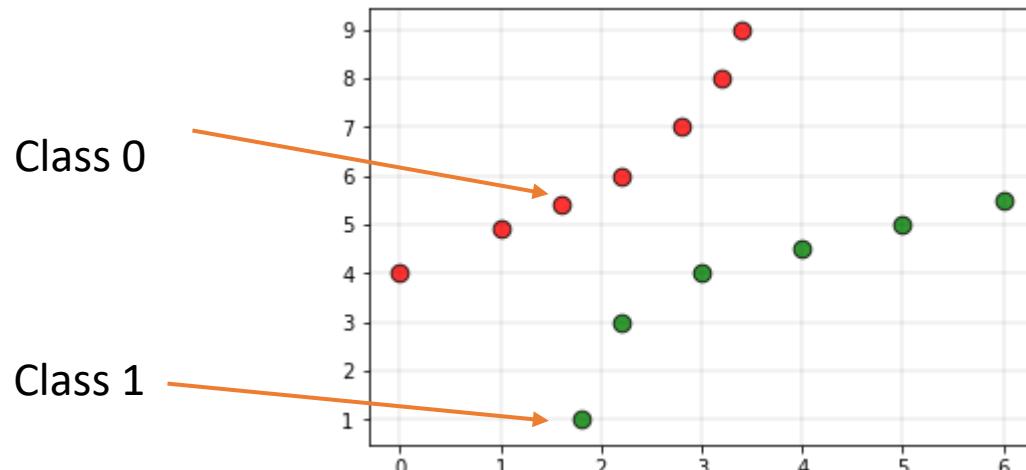
- KNN is very **easy to implement**. There are only two parameters: the value of K and the distance function required to implement KNN
- **No Training Period** for KNN for which it does not learn anything in the training period. All training instances are stored
- Since requires no training, **new data can be added seamlessly** which will not impact the accuracy of the algorithm.
- Adding a new training instance only affects the computation of neighbors, which is done at execution time (i.e., lazily)

# Disadvantages

- **Does not work well with high dimensions:** The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension. Hence, it is called that k-NN is subject to the curse of dimensionality (i.e., presence of many irrelevant attributes)
- k-NN needs adequate distance measure
- **Doesn't work well with a large dataset.** The cost of calculating distance between a new point and each existing point is very high which in turn degrades the performance of the algorithm.
- **Need feature scaling:** We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.

# Distance-weighted k-NN

- Distance Weighted kNN is a modified version of [k nearest neighbors](#). One of the many issues that affect the performance of the kNN algorithm is the choice of the hyperparameter  $k$ .
- Another issue is the approach to combining the class labels. The simplest method is to take the majority vote, but this can be a problem if the nearest neighbors vary widely in their distance and the closest neighbors more reliably indicate the class of the object.



# Distance-weighted k-NN

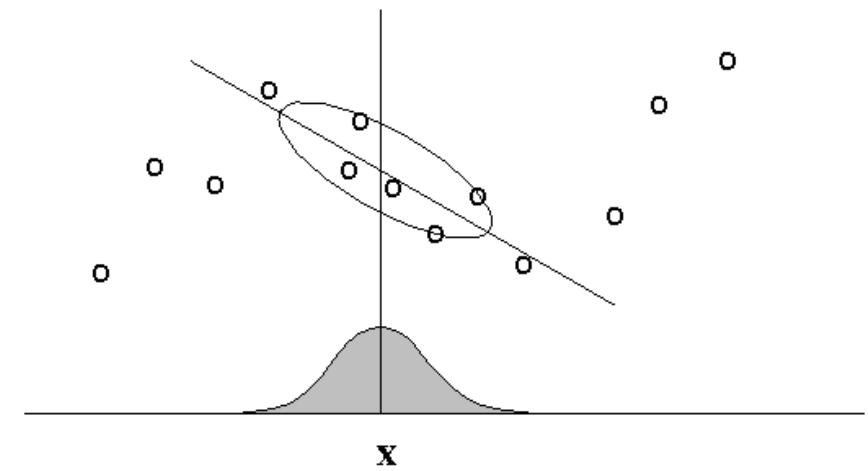
- In **distance weighted kNN**, the nearest k points are given a weight using a function called as the **kernel function**.
- The intuition behind weighted kNN, is to give more weight to the points which are nearby and less weight to the points which are farther away.
- Any function can be used as a kernel function for the weighted knn classifier whose value decreases as the distance increases. The simple function which is used is the **inverse distance function**.
  - Replace

$$\hat{f}(q) = \arg \max_{v \in V} \sum_{i=1}^k d(v, f(x_i)) \quad \text{by:}$$

$$\hat{f}(q) = \arg \max_{v \in V} \sum_{i=1}^k \frac{1}{d(x_i, x_q)^2} d(v, f(x_i)) \quad \text{Where, } \frac{1}{d(x_i, x_q)^2} = W_i$$

# Locally Weighted Linear Regression

- kNN form local approximation for each query point i.e., **approximates the target function  $f(x)$  at the single query point  $x = x_q$**
- Locally weighted regression is a generalization of this approach. It constructs an explicit approximation to  $f$  over a local region surrounding  $x_q$ .
- Locally weighted regression uses nearby or distance weighted training examples to form this local approximation  $f$ .
- For example we might approximate the target function in the neighborhood surrounding  $x_q$ , using a linear function, a quadratic function, a multilayer neural network or some other functional form.
- The phrase **locally weighted regression** is called **local** because the function is approximated based only on data near the query point  $x_q$ , **weighted** because the contribution of each training example is weighted by its distance from the query point and **regression** because this is the term used widely in the statistical learning community for the problem of approximating the real valued function/



**Course code:** CSE3008

**Course Title:** Introduction to Machine Learning

**Module 3: Support Vector Machine**

**Dr. Usha Rani Gogoi**

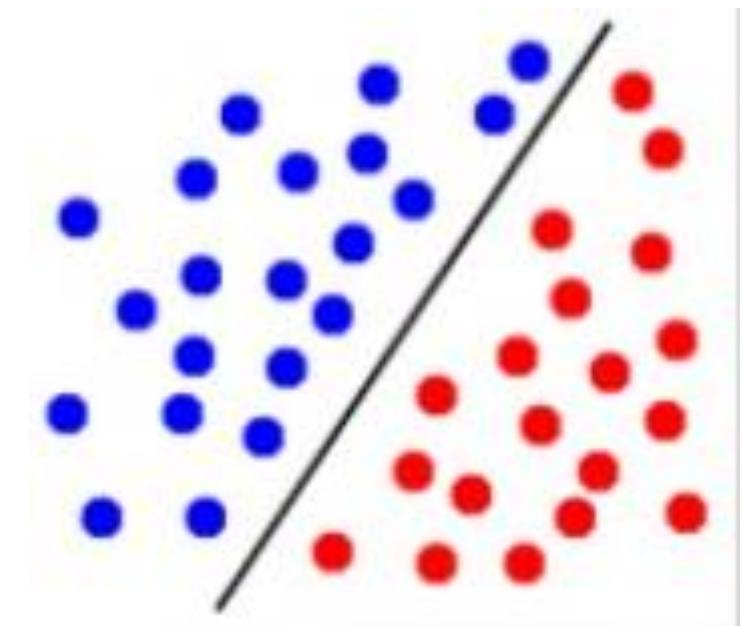
**Assistant Professor Sr. Grade 1**

**[usha.gogoi@vitap.ac.in](mailto:usha.gogoi@vitap.ac.in)**

**SCOPE**

# Linear Separability

- Two sets of points in  $p$ -dimensional space are said to be linearly separable if they can be separated using a  $p-1$  dimensional hyperplane.
- For example, The two sets of 2D data in the image are separated by a single straight line (1D hyperplane), and hence are linearly separable.
- The hyperplane that separates the two sets of data is called the linear discriminant.
- For every linearly separable data, there exists infinite number of separating hyperplanes. Hence, we must choose the most suitable one for classification

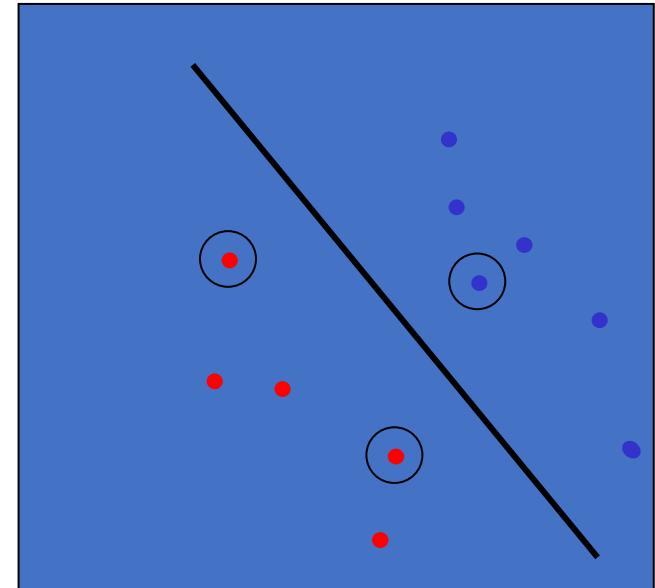
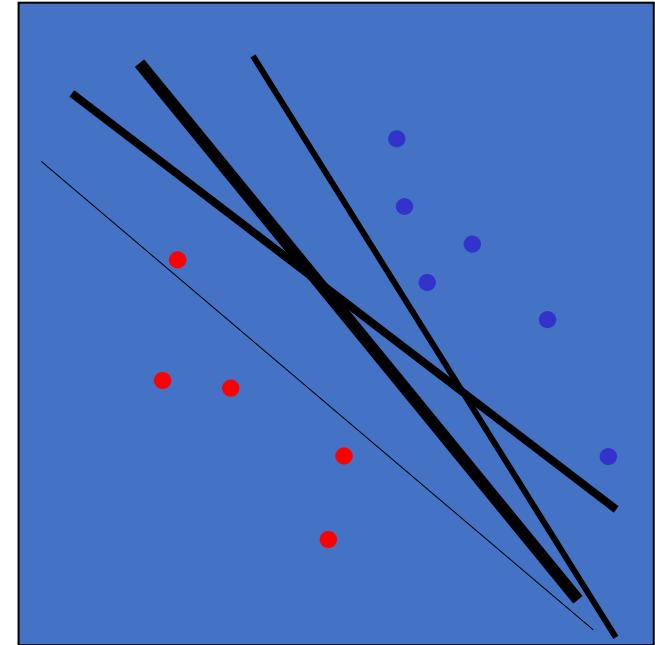


# Support Vector Machine (SVM)

- If we have a big data set that needs a complicated model, then we can use Support Vector machine.
- It prevents overfitting when using big sets of features.
- In ML, SVM are supervised learning models with associated learning algorithms that analyze data and recognize patterns used for classification and regression analysis.
- It is a Non-Probabilistic binary linear classifier. However, it supports non-linear classification using the kernel trick.

# Support Vector Machine (SVM)

- Suppose we use a big set of features to ensure that the two classes are linearly separable. What is the best separating line to use?
- The line that maximizes the minimum margin is a good bet.
- This maximum-margin separator is determined by a subset of the datapoints.
  - Datapoints in this subset are called “support vectors”.
  - It will be useful computationally if only a small fraction of the datapoints are support vectors, because we use the support vectors to decide which side of the separator a test case is on.



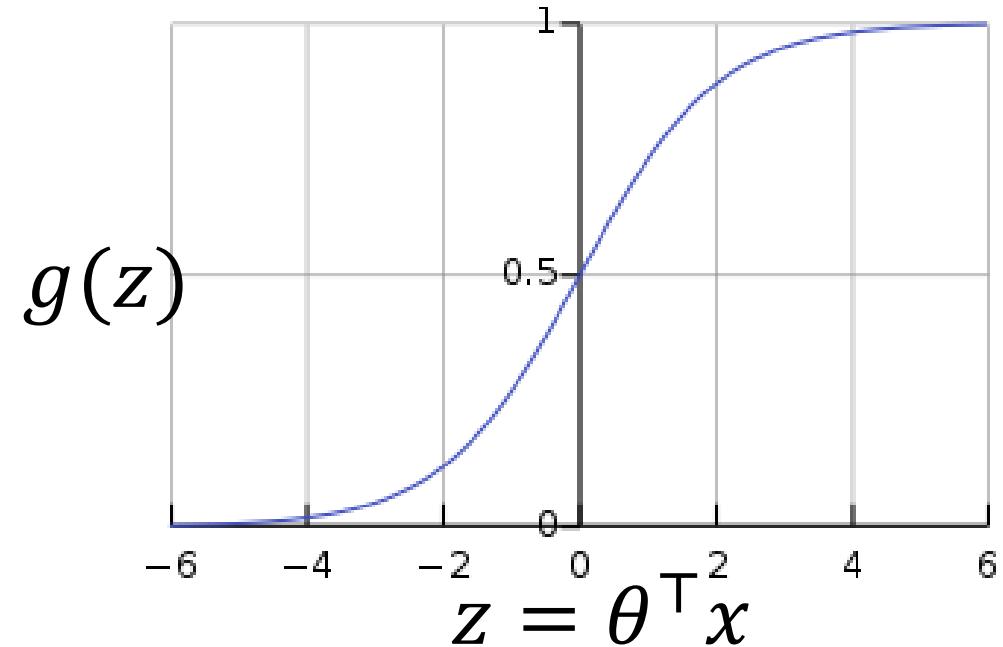
# Support Vector Machine

- Cost function
- Large margin classification
- Kernels
- Using an SVM

# Logistic Regression

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

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



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

$$z = \theta^T x \geq 0$$

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

$$z = \theta^T x < 0$$

## Alternative view

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

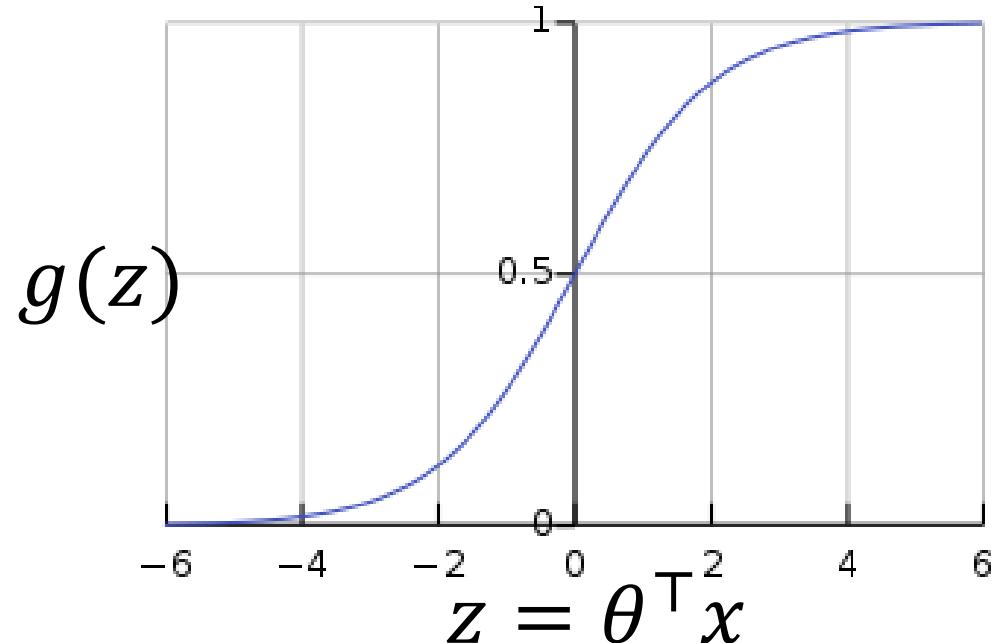
$$g(z) = \frac{1}{1 + e^{-z}}$$

If “y = 1”, we want  $h_{\theta}(x) \approx 1$

$$z = \theta^T x \gg 0$$

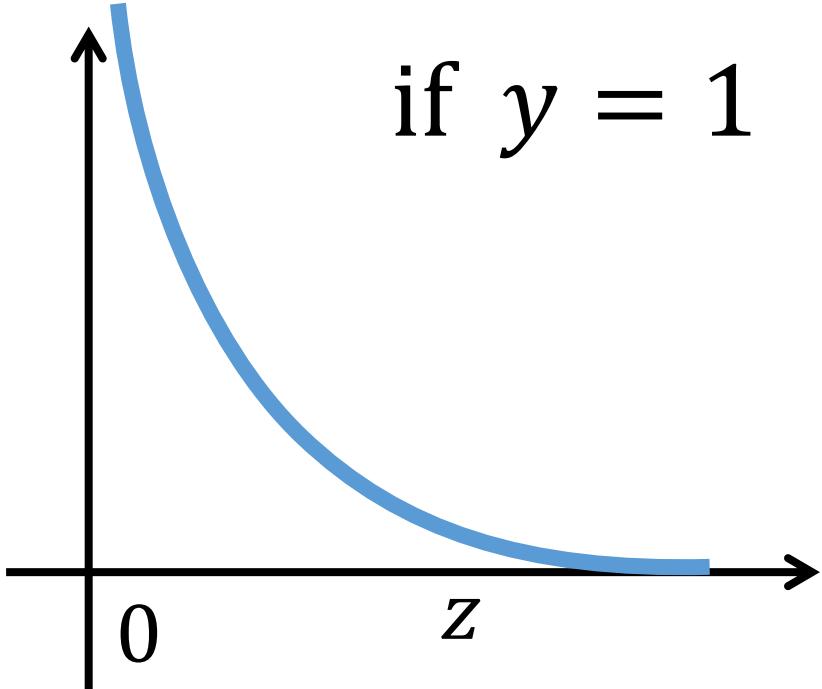
If “y = 0”, we want  $h_{\theta}(x) \approx 0$

$$z = \theta^T x \ll 0$$

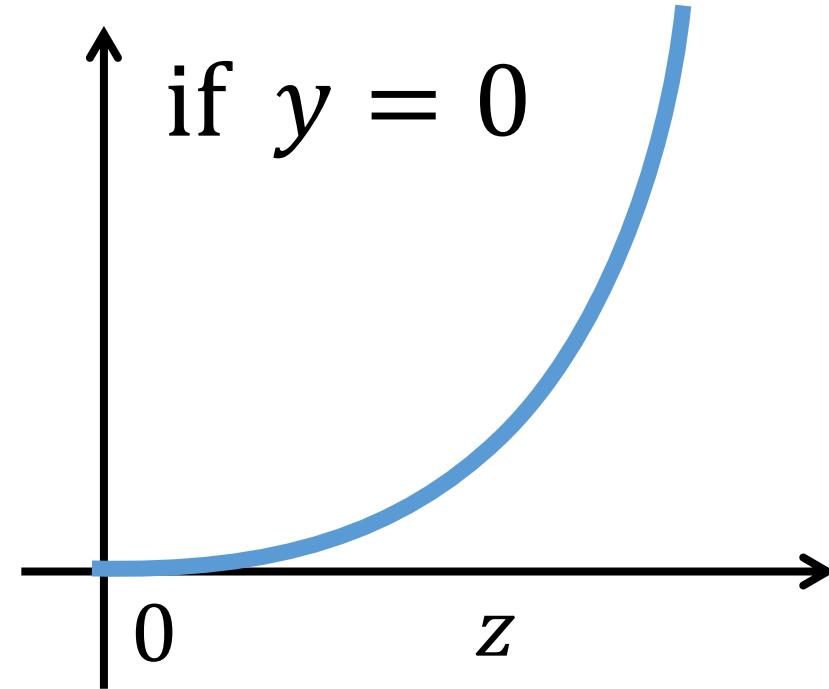


# Cost function for Logistic Regression

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



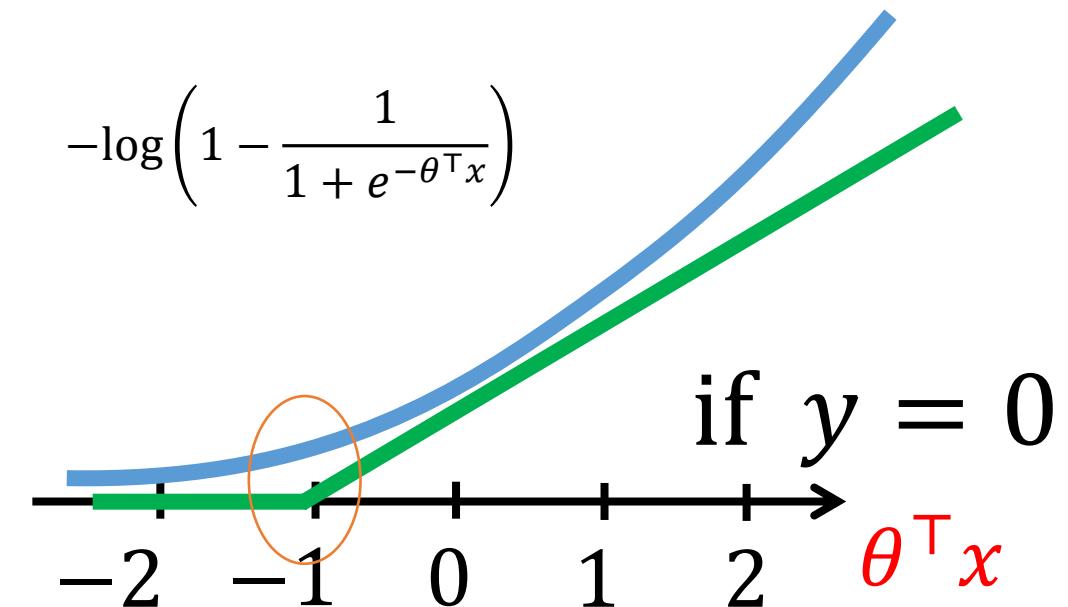
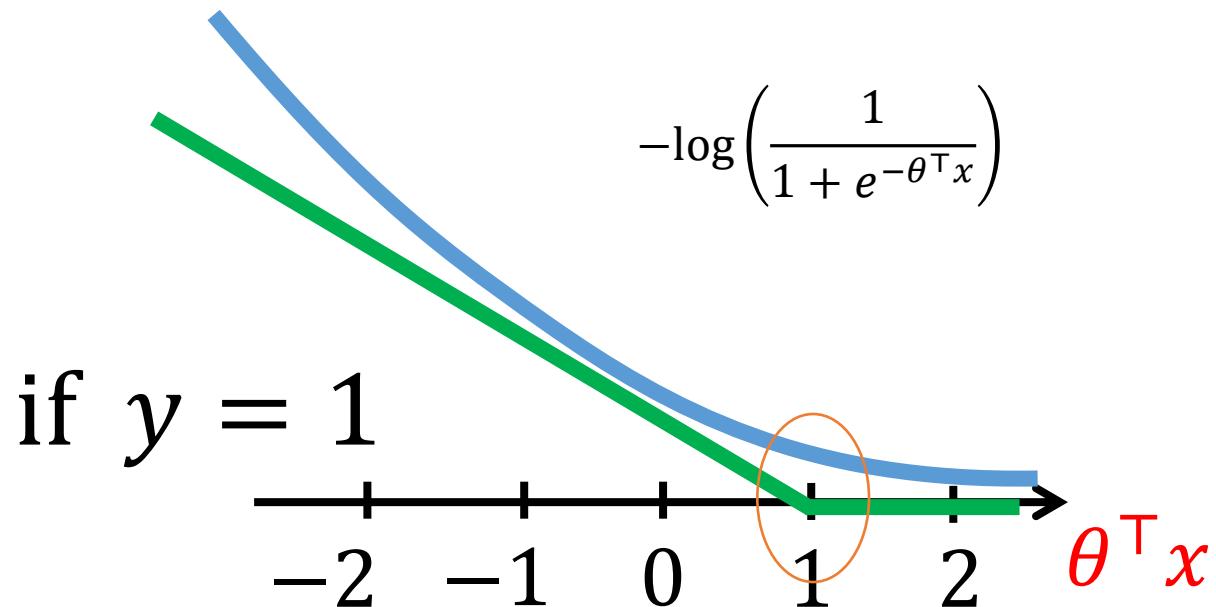
if  $y = 1$



if  $y = 0$

# SVM Cost Function From Logistic Regression

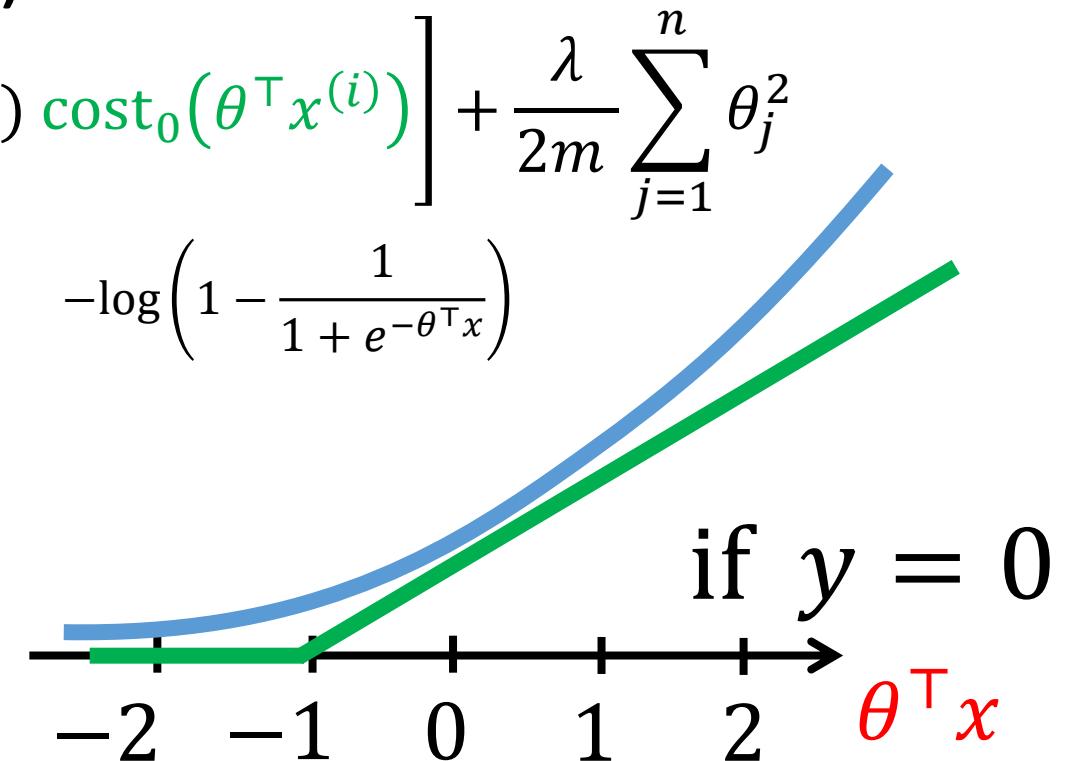
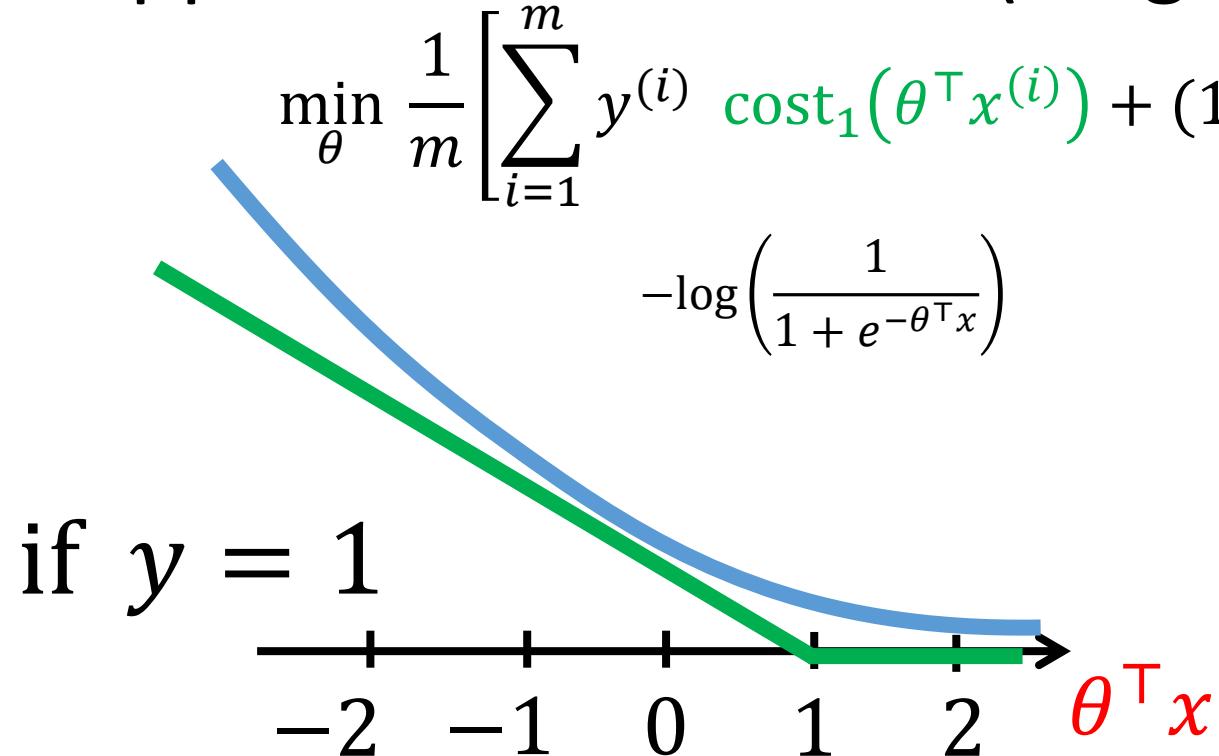
- $\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$   
 $= y \left( -\log\left(\frac{1}{1 + e^{-\theta^T x}}\right) \right) + (1 - y) \left( -\log\left(1 - \frac{1}{1 + e^{-\theta^T x}}\right) \right)$



# Regularized Logistic regression (logistic loss)

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

## Support vector machine (hinge loss)



# Optimization objective for SVM

$$\min_{\theta} \frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \operatorname{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \operatorname{cost}_0(\theta^\top x^{(i)}) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

- 1) Remove  $\frac{1}{m}$
- 2) Multiply  $C = \frac{1}{\lambda}$



$$\min_{\theta} C \left[ \sum_{i=1}^m y^{(i)} \operatorname{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \operatorname{cost}_0(\theta^\top x^{(i)}) \right] + \sum_{j=1}^n \theta_j^2$$

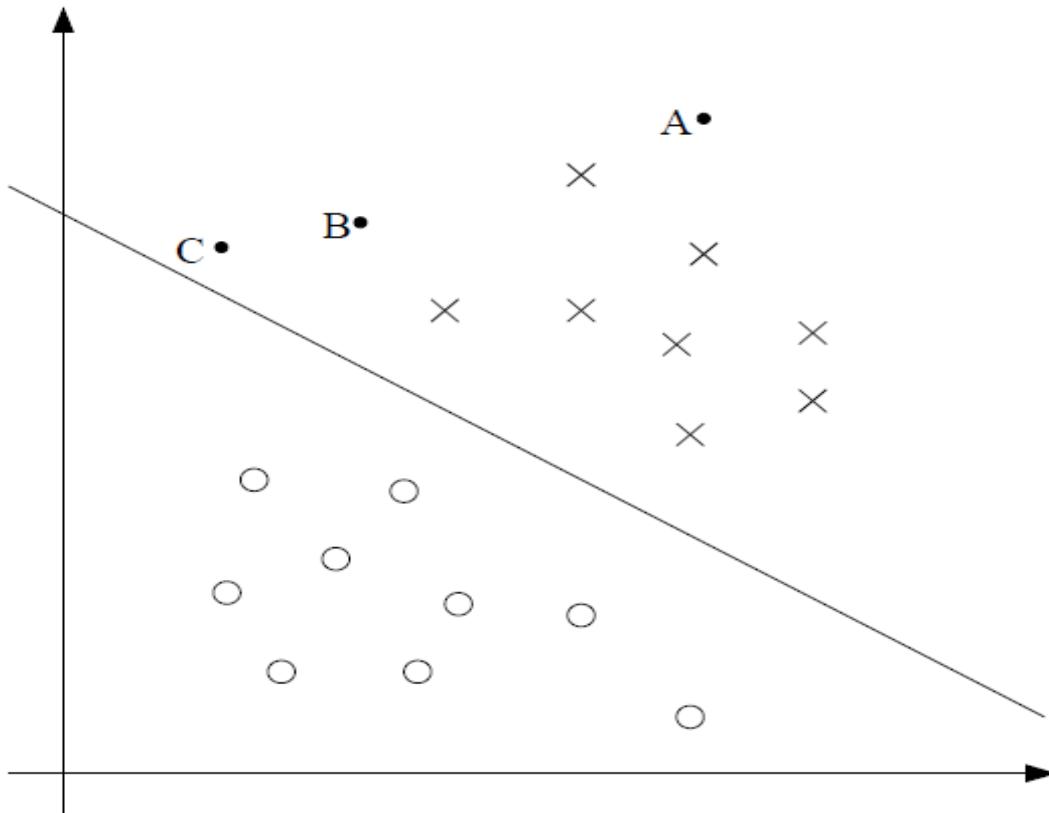
# Hypothesis of SVM

$$\min_{\theta} C \left[ \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \sum_{j=1}^n \theta_j^2$$

- **Hypothesis:** Unlike logistic,  $h_\theta(x)$  does not give probability, but instead we get a direct prediction 0 and 1. That means,

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^\top x \geq 0 \\ 0 & \text{if } \theta^\top x < 0 \end{cases}$$

# Margins: Intuition



Consider the following figure, in which x's represent positive training examples, o's denote negative training examples, a decision boundary (this is the line given by the equation  $\theta^T x = 0$ , and is also called the separating hyperplane) is also shown, and three points have also been labelled A, B and C.

# Margins: Intuition

- Notice that the point A is very far from the decision boundary. If we are asked to make a prediction for the value of  $y$  at A, it seems we should be quite confident that  $y = 1$  there.
- Conversely, the point C is very close to the decision boundary, and while it's on the side of the decision boundary on which we would predict  $y = 1$ , it seems likely that just a small change to the decision boundary could easily have caused our prediction to be  $y = 0$ .
- Hence, we're much more confident about our prediction at A than at C.
- The point B lies in-between these two cases, and more broadly, we see that if a point is far from the separating hyperplane, then we may be significantly more confident in our predictions.

# Notation

- For SVM we will be considering a linear classifier for a binary classification problem with labels  $y$  and features  $x$ .  $y \in \{-1, 1\}$  (instead of  $\{0, 1\}$ ) to denote the class labels.
- Also, rather than parameterizing our linear classifier with the vector  $\theta$ , we will use parameters  $w$ ,  $b$ , and write our classifier as

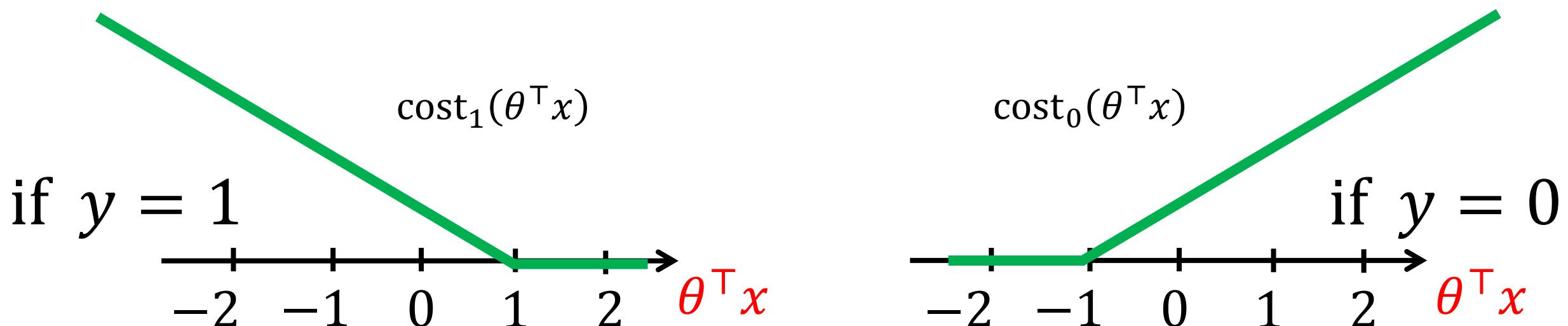
$$h_{w,b}(x) = g(w^T x + b).$$

Here,  $g(z) = 1$  if  $z \geq 0$ ,  
 $g(z) = -1$  otherwise.

- This “ $w, b$ ” notation allows us to explicitly treat the intercept term  $b$  separately from the other parameters. (We also drop the convention we had previously of letting  $x_0 = 1$  be an extra coordinate in the input feature vector.)
- Thus,  $b$  takes the role of what was previously  $\theta_0$ , and  $w$  takes the role of  $[\theta_1 \dots \theta_n]^T$
- From our definition of  $g$  above, our classifier will directly predict either 1 or -1.

# SVM, A Large Margin Classifier

$$\min_{\theta} C \left[ \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



If “ $y = 1$ ”, we want  $\theta^\top x \geq 1$  (not just  $\geq 0$ )

If “ $y = 0$ ”, we want  $\theta^\top x \leq -1$  (not just  $< 0$ )

# SVM, A Large Margin Classifier

So, if  $y = 1$ ,  $\text{cost}_1(z) = 0$  only when  $z \geq 1$

if  $y = 0$ ,  $\text{cost}_0(z) = 0$  only when  $z \leq -1$

So, instead of predicting a sample as positive when  $z \geq 0$ , SVM wants a bit more than this. That means it does not want to just get it right, but have the value be quite a bit bigger than 0 . So, it throws a extra safety margin factor

# Consequence in Optimizing SVM

$$\min_{\theta} C \left[ \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

- Let's say we have a very large  $C$  ...

- Whenever  $y^{(i)} = 1$ :

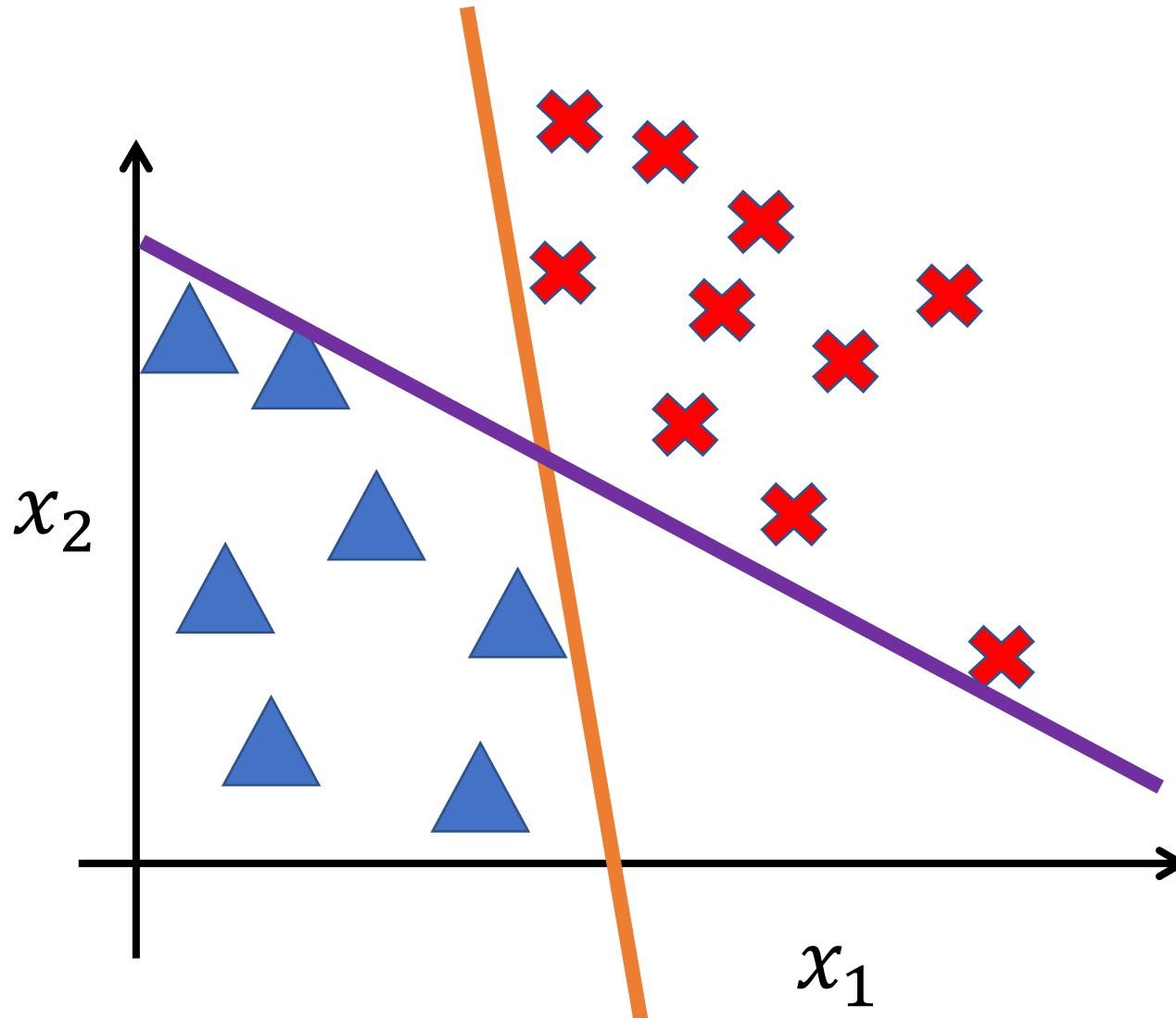
$$\theta^\top x^{(i)} \geq 1$$

- Whenever  $y^{(i)} = 0$ :

$$\theta^\top x^{(i)} \leq -1$$

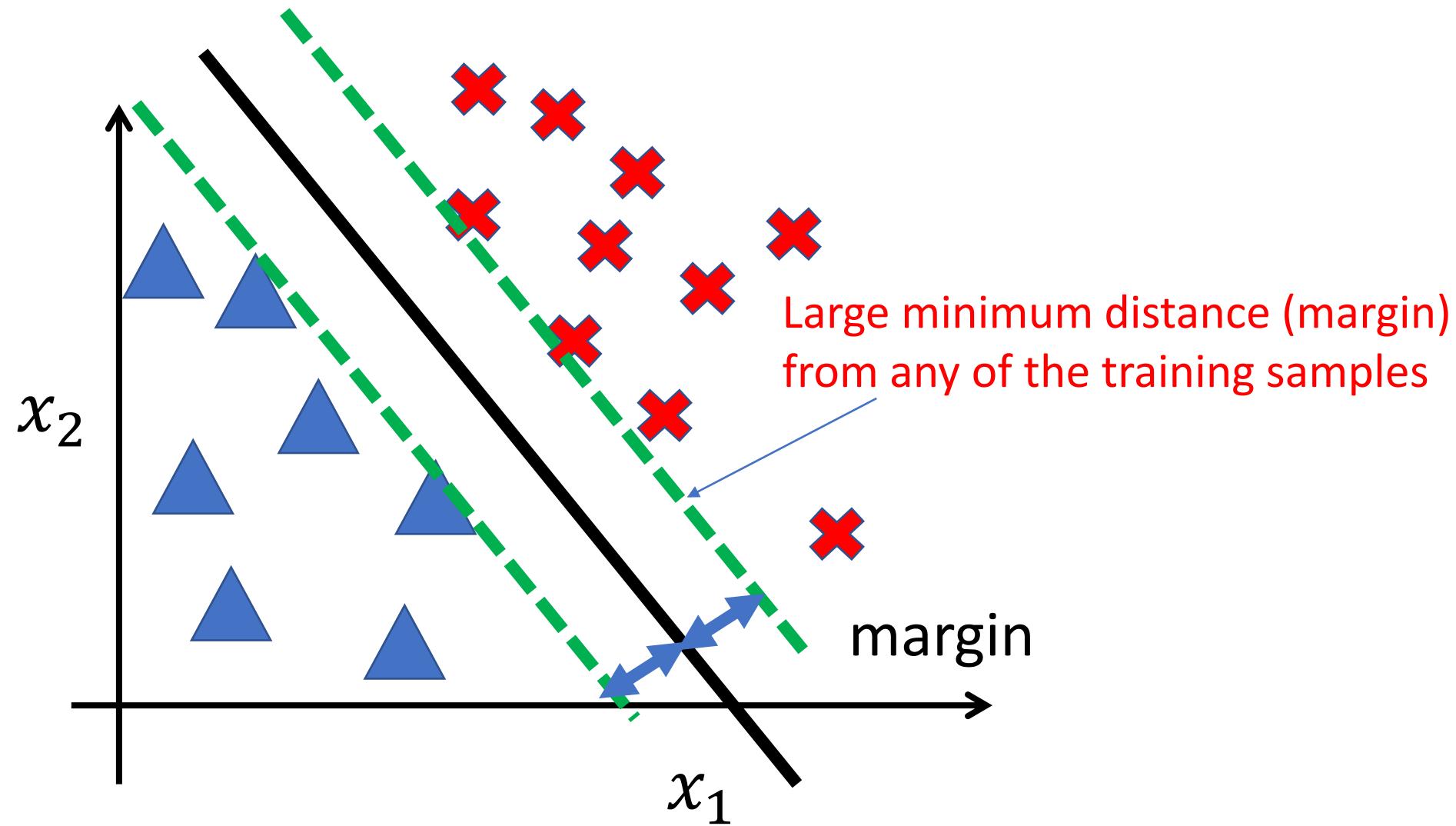
$$\begin{aligned} & \min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 \\ \text{s. t. } & \theta^\top x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1 \\ & \theta^\top x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0 \end{aligned}$$

# SVM decision boundary: Linearly separable case

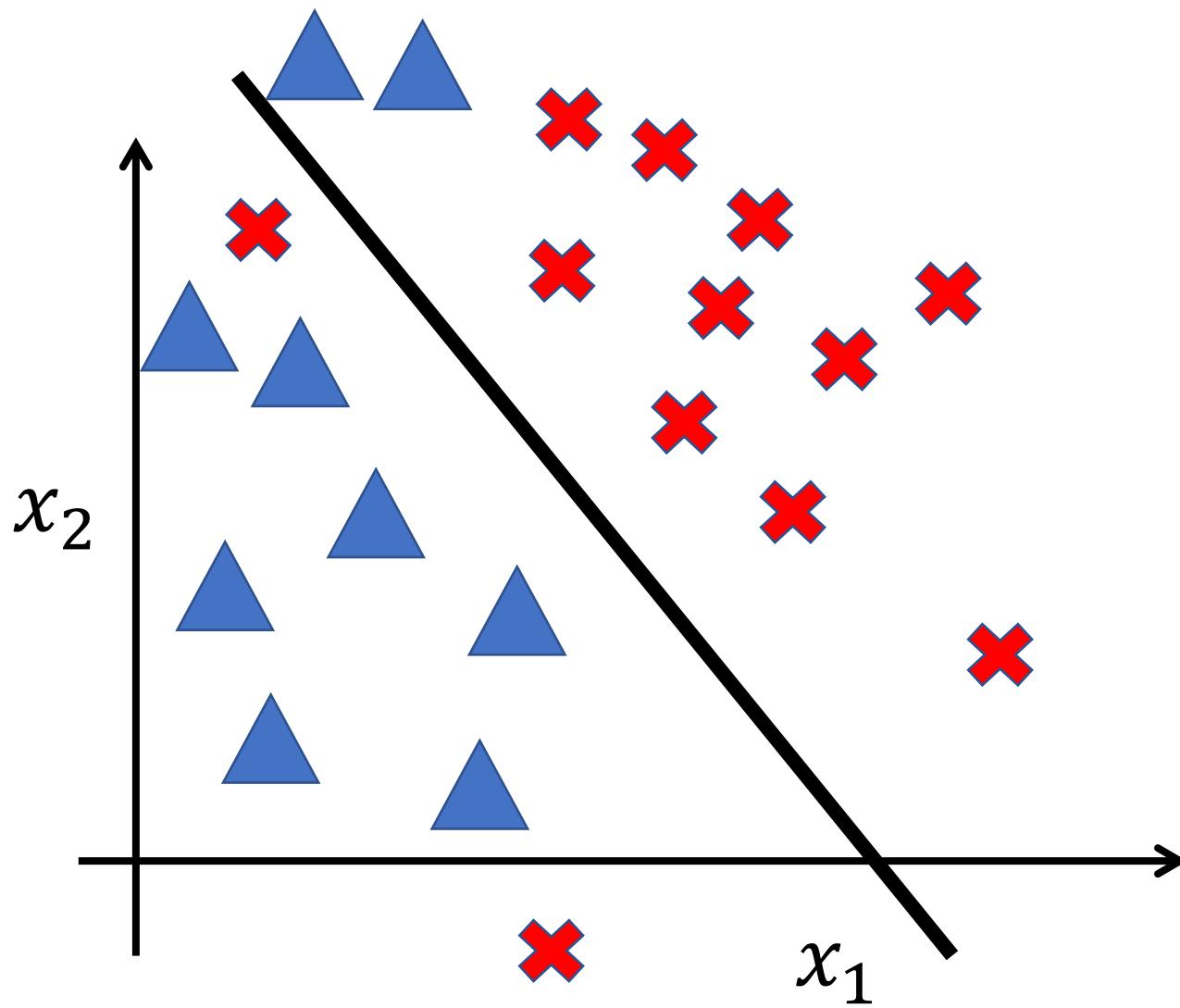


# SVM decision boundary: Linearly separable case

**Margin:** The distance from the decision surface to the closest datapoints determines the margin of the classifier

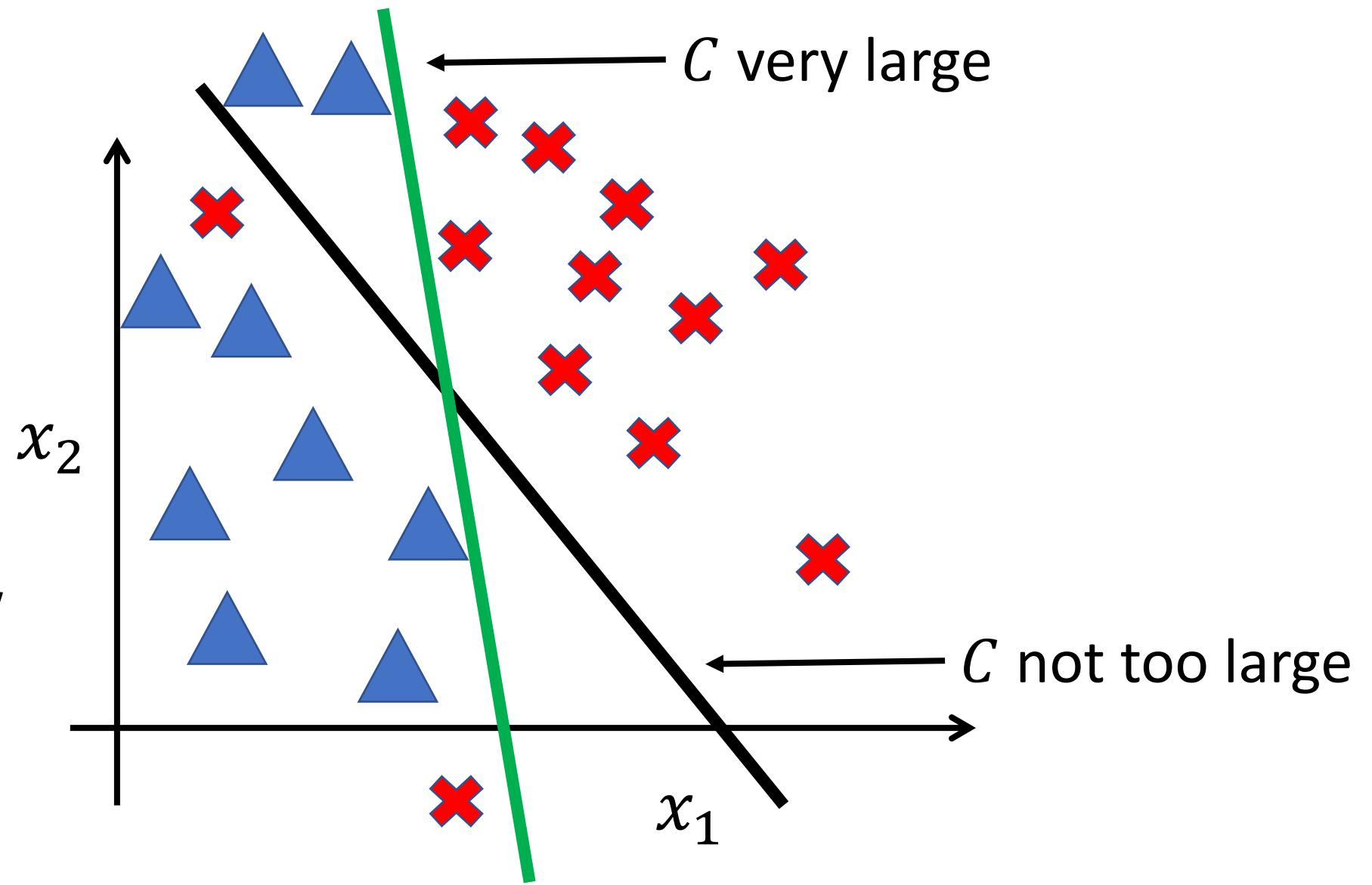


# Large margin classifier in the presence of outlier



# Large margin classifier in the presence of outlier

SVM being a large margin classifier is only really relevant when you have no outliers and data are linearly separable. We may ignore a few outliers



# Requirement of Maximum Margin Hyperplane

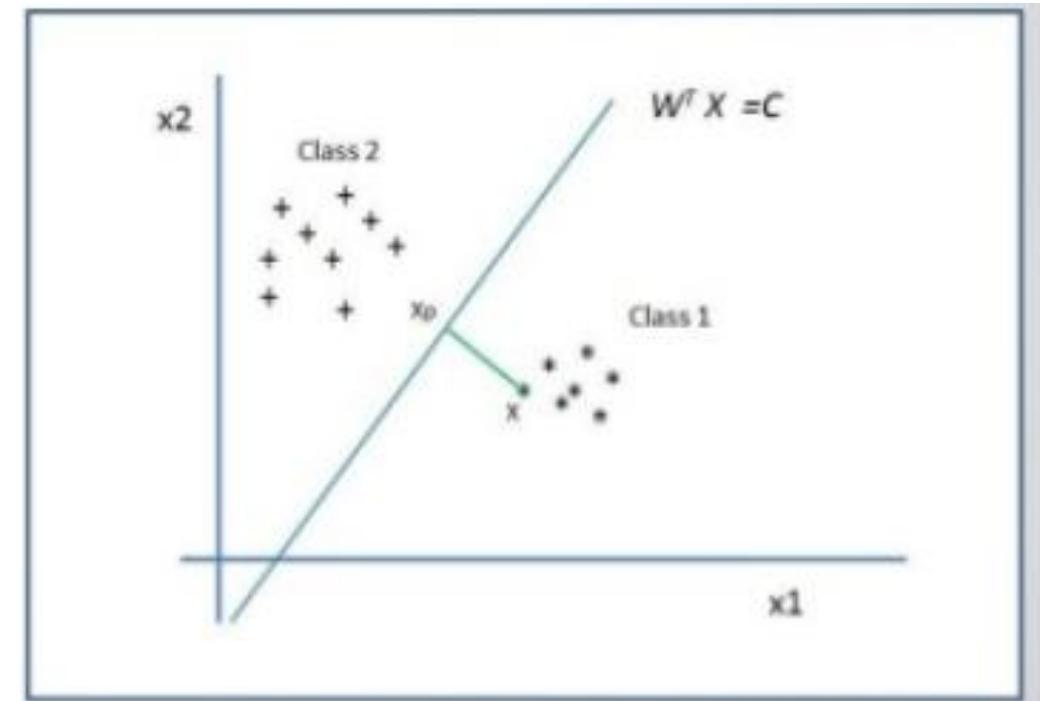
- We can compute the **(perpendicular)** distance from each observation in the data set to a given separating hyperplane; **the smallest such distance is the minimal distance from the observations to the hyperplane, and is known as the margin.** The maximal margin hyperplane is the separating hyperplane for which the margin is largest.

- Finding the Shortest Margin Distance**

Find  $X_p$

Such that  $\|X_p - X\|$  is minimum

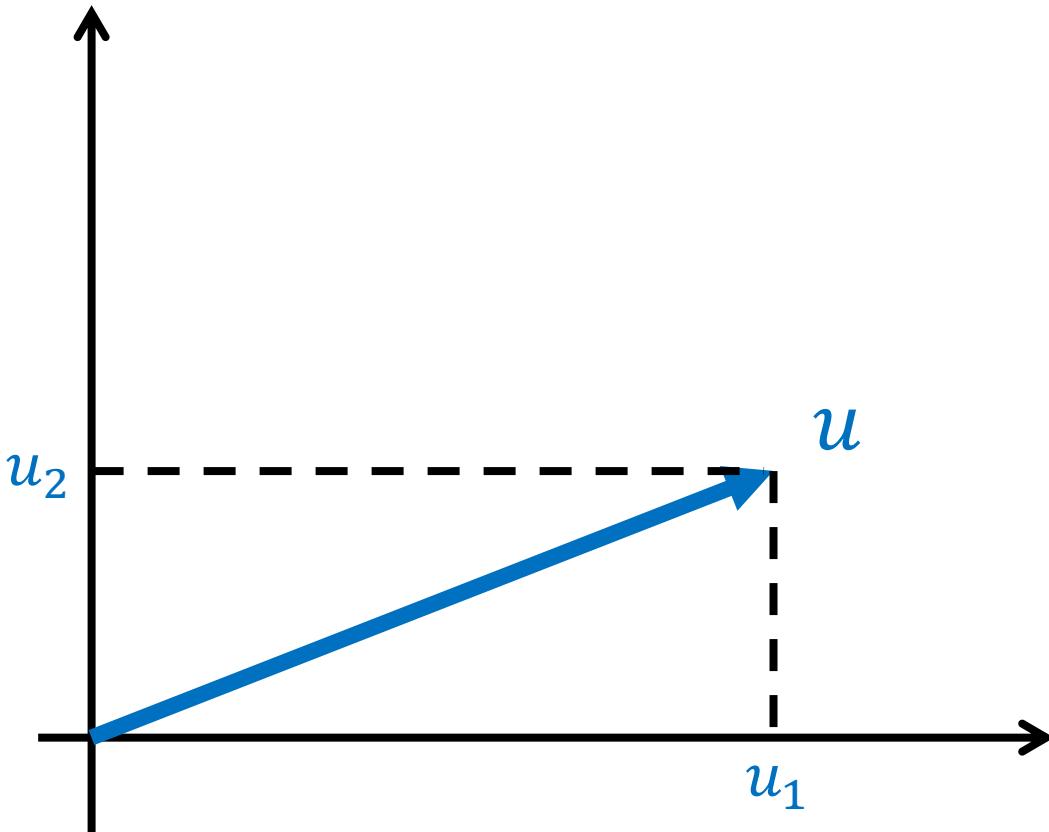
- Requirement:** Supposing we have a maximal margin hyperplane for a data set and want to predict the class for a new observation, we compute the distance from the hyperplane.



# Requirement of Maximum Margin Hyperplane

- The more the distance from the hyperplane the more confident we are that the sample belongs to that class.
- Thus the hyperplane with the farthest smallest distance from the training observation would be the most suitable.

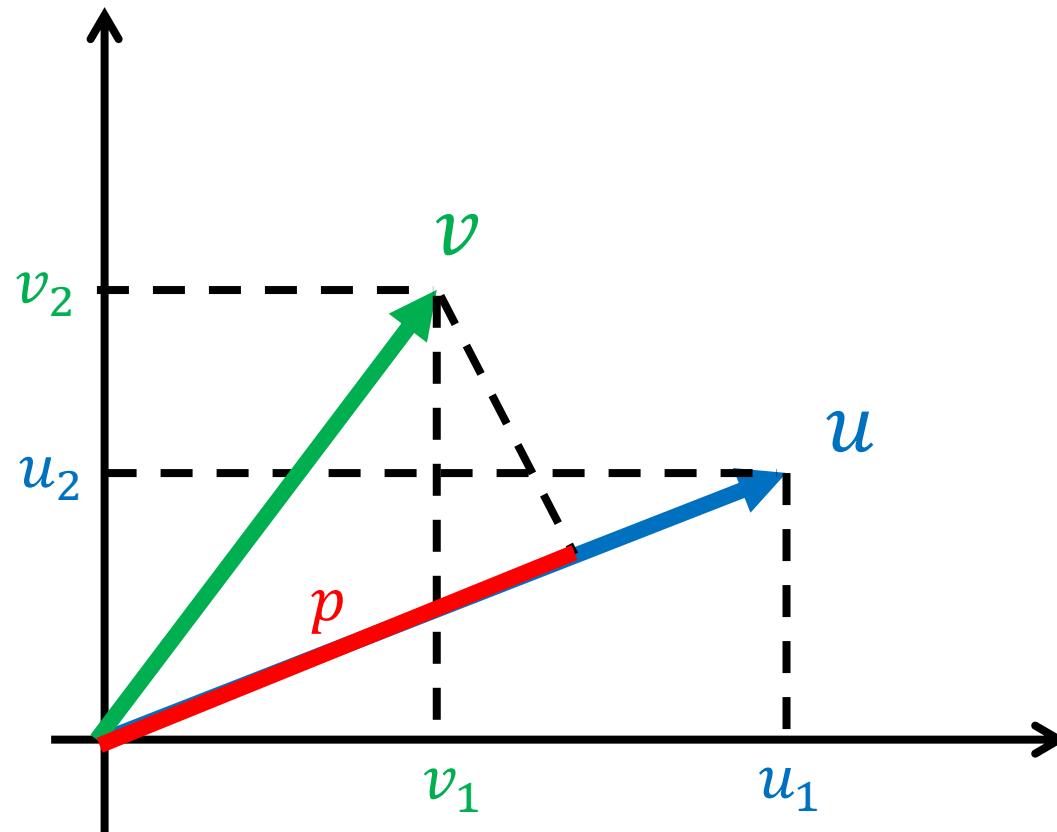
# Vector inner product



$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$\|u\| = \text{length of vector } u$   
 $= \text{Norm of } u$   
 $= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$

# Vector inner product



$p$  = length of projection of  $v$  onto  $u$   
 $p$  can be negative also.

Inner Product of  $u$  and  $v$

$$\begin{aligned} u^T v &= p \cdot \|u\| \\ &= u_1 v_1 + u_2 v_2 \end{aligned}$$

Reverse is also true. i.e.,

$$v^T u = v_1 u_1 + v_2 u_2$$

$$v^T u = u^T v$$

# SVM decision boundary

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

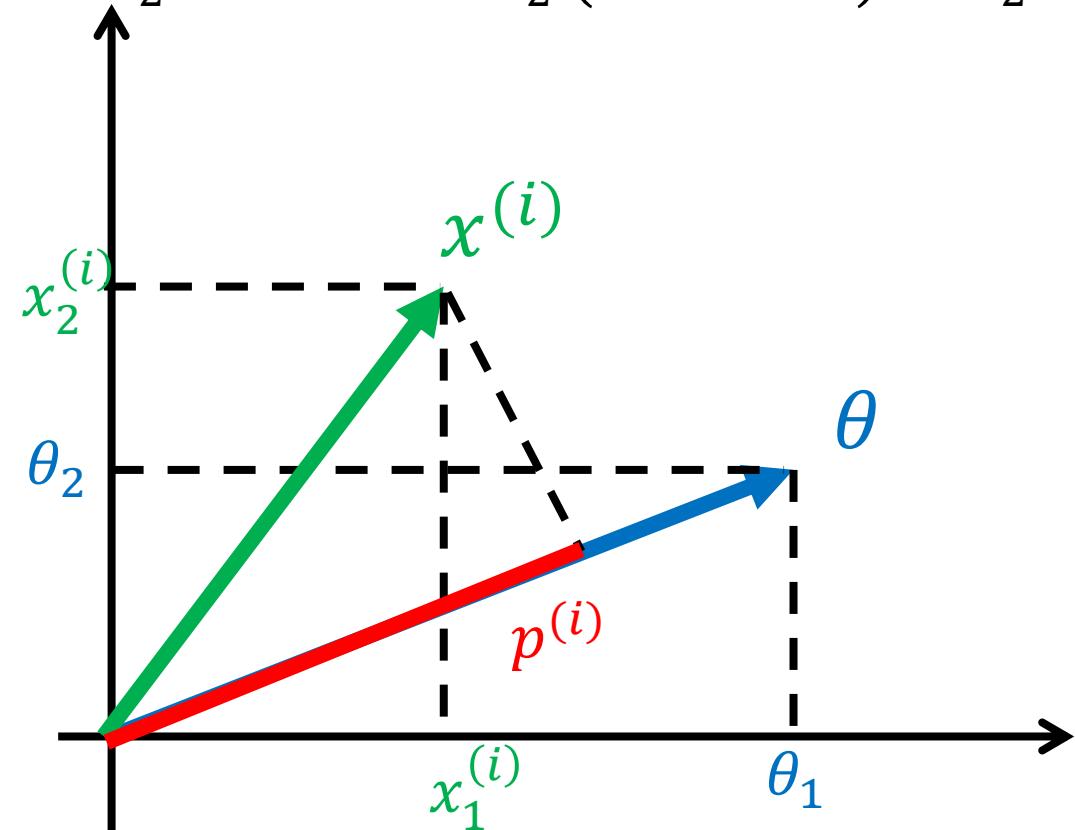
$$\text{s. t. } \begin{aligned} \theta^\top x^{(i)} &\geq 1 & \text{if } y^{(i)} = 1 \\ \theta^\top x^{(i)} &\leq -1 & \text{if } y^{(i)} = 0 \end{aligned}$$

What's  $\theta^\top x^{(i)}$ ?

$$\theta^\top x^{(i)} = p^{(i)} \|\theta\|^2$$

Simplification:  $\theta_0 = 0, n = 2$

$$\frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} (\theta_1^2 + \theta_2^2) = \frac{1}{2} (\sqrt{\theta_1^2 + \theta_2^2})^2 = \frac{1}{2} \|\theta\|^2$$

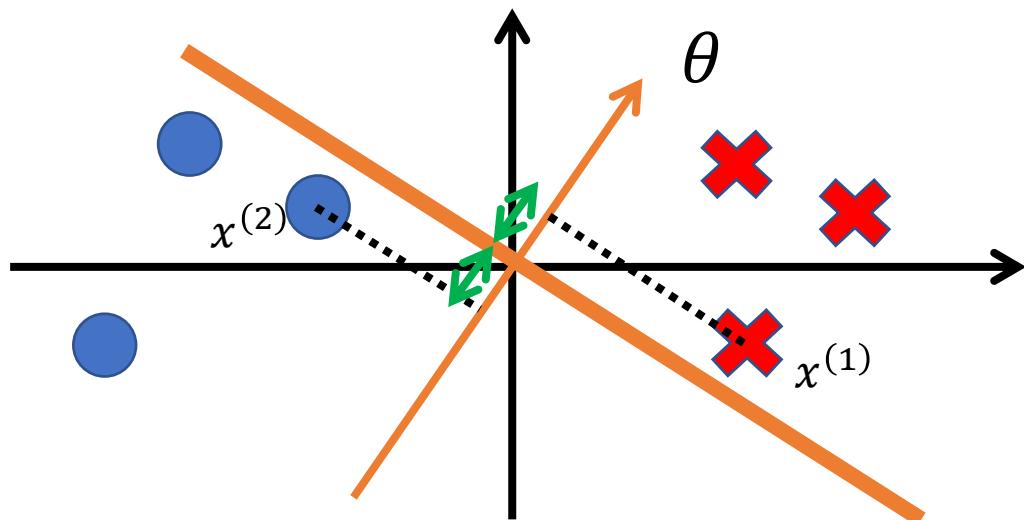


# SVM decision boundary

$$\min_{\theta} \frac{1}{2} \|\theta\|^2$$

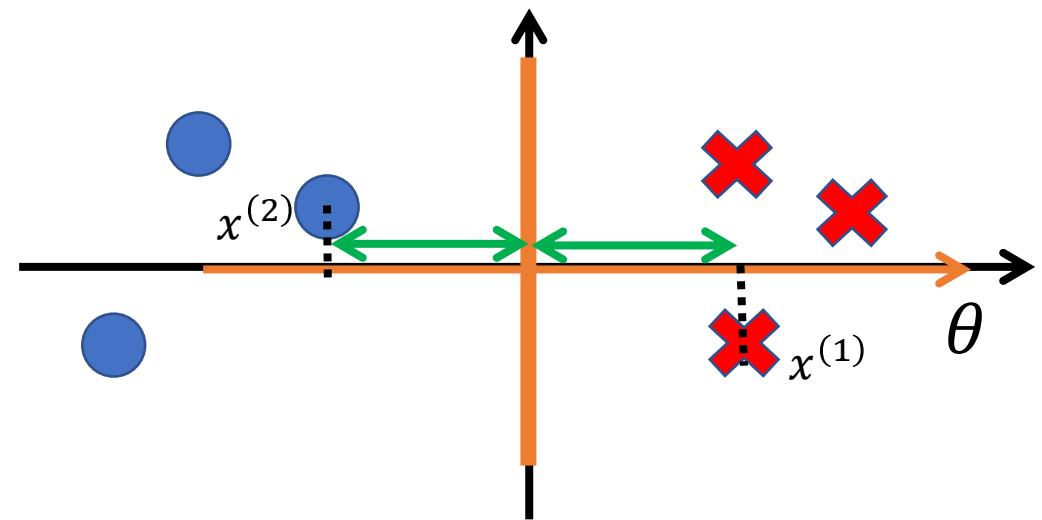
$$\text{s.t. } p^{(i)} \|\theta\|^2 \geq 1 \quad \text{if } y^{(i)} = 1$$
$$p^{(i)} \|\theta\|^2 \leq -1 \quad \text{if } y^{(i)} = 0$$

$p^{(1)}, p^{(2)}$  small  $\rightarrow \|\theta\|^2$  large

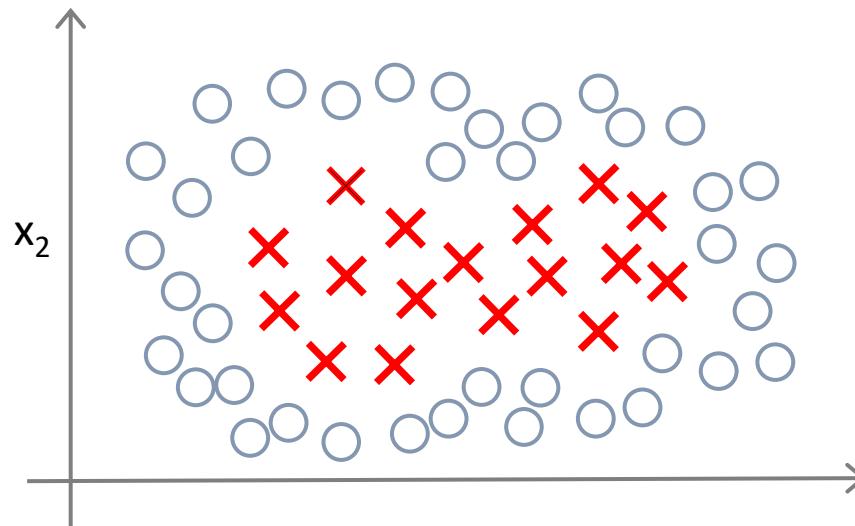


Simplification:  $\theta_0 = 0, n = 2$

$p^{(1)}, p^{(2)}$  large  $\rightarrow \|\theta\|^2$  can be small



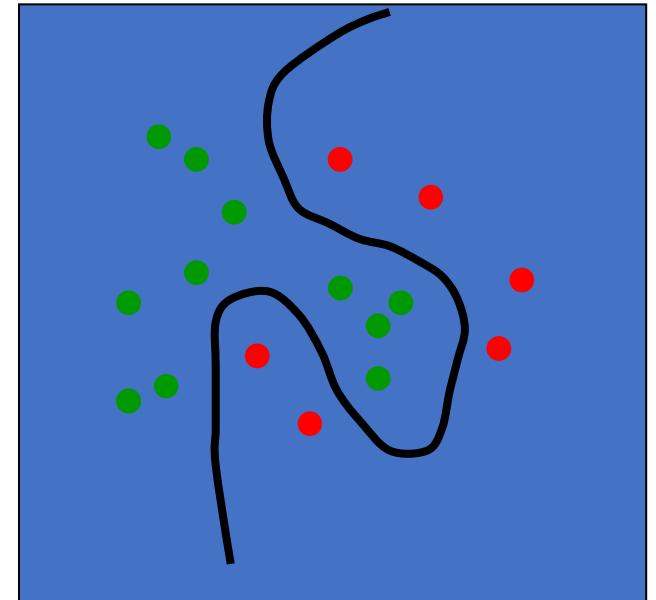
# SVM: Introducing Nonlinearity



**Need to generate new features!**

# How to make a plane curved

- Fitting hyperplanes as separators is mathematically easy.
  - The mathematics is linear.
- By replacing the raw input variables with a **much larger set of features** we get a nice property:
  - A planar separator in the high-dimensional space of feature vectors is a **curved separator** in the low dimensional space of the raw input variables.



A planar separator in a 20-D feature space projected back to the original 2-D space

# A potential problem and a magic solution

- If we map the input vectors into a **very high-dimensional feature space**, surely the task of finding the maximum-margin separator becomes computationally intractable?
  - The mathematics is all linear, which is good, but the vectors have a huge number of components.
  - So taking the scalar product of two vectors is very expensive.
- The way to keep things tractable is to use **“the kernel trick”**
- The kernel trick makes your brain hurt when you first learn about it, but its actually very simple.

# What the kernel trick achieves

- All of the computations that we need to do to **find the maximum-margin separator can be expressed in terms of scalar products between pairs of datapoints (in the high-dimensional feature space)**.
- These **scalar products are the only part of the computation that depends on the dimensionality of the high-dimensional space.**
  - So if we had a fast way to do the scalar products we would not have to pay a price for solving the learning problem in the high-D space.
- The kernel trick is just a magic way of doing scalar products a whole lot faster than is usually possible.
  - It relies on choosing a way of mapping to the high-dimensional feature space that allows fast scalar products.

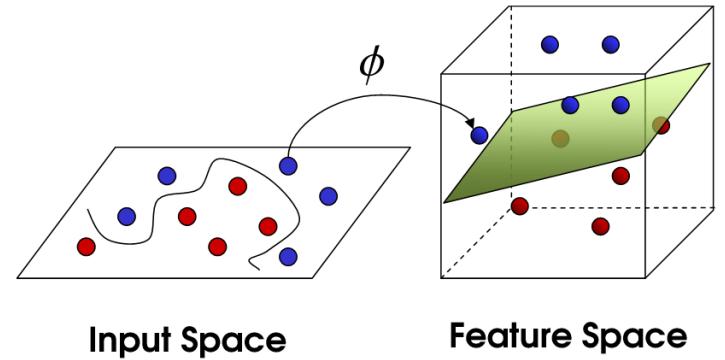
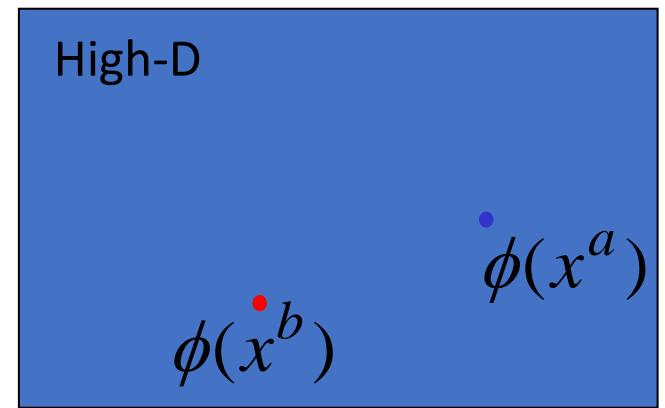
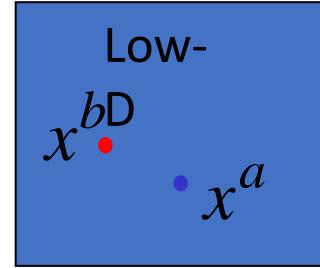
# The kernel trick

- For many mappings from a low-D space to a high-D space, there is a simple operation on two vectors in the low-D space that can be used to compute the scalar product of their two images in the high-D space.

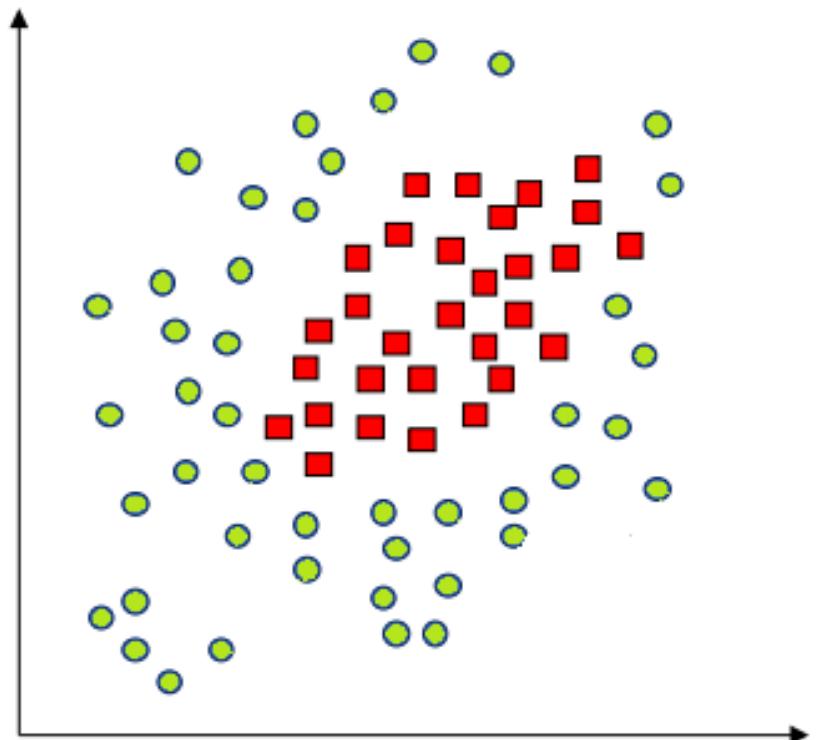
$$K(x^a, x^b) = \phi(x^a) \cdot \phi(x^b)$$

↑  
Letting the  
kernel do  
the work

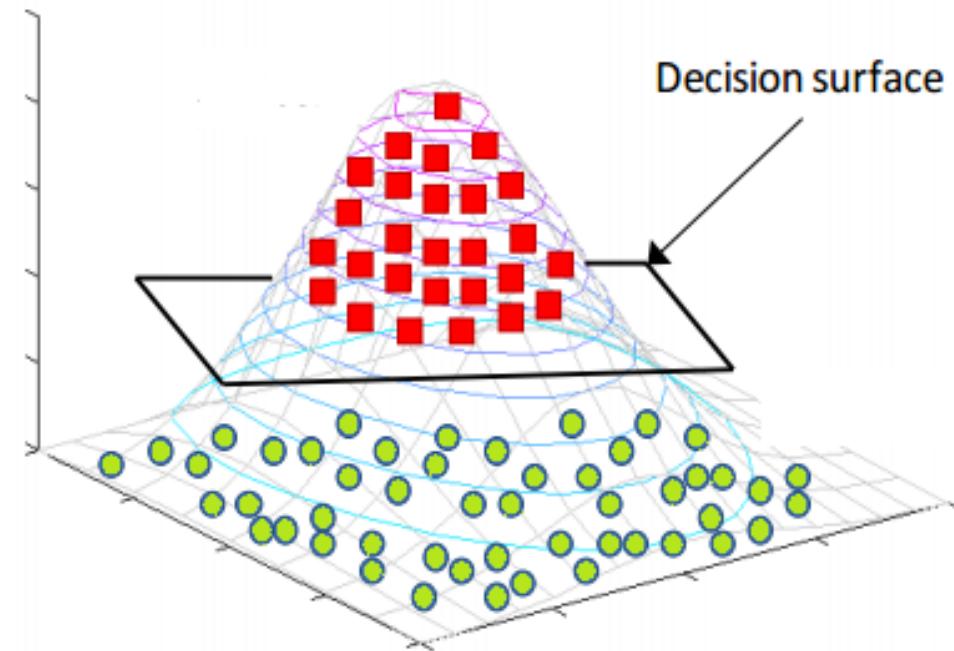
↑  
doing the scalar  
product in the  
obvious way



# Kernels

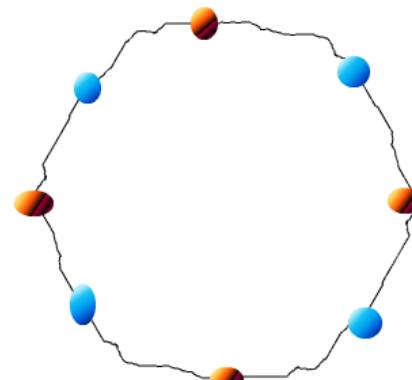


kernel

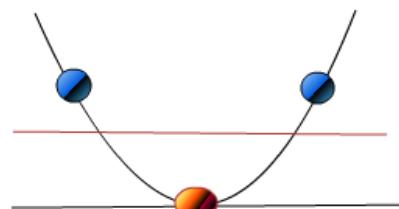


Decision surface

# Kernel trick ?



Inseparable data! T \_\_\_\_\_ T  
What to do? 😞



- There is no straight line (hyper plane in 2 dimensions) which can separate red and blue dots.

- Generate more features!

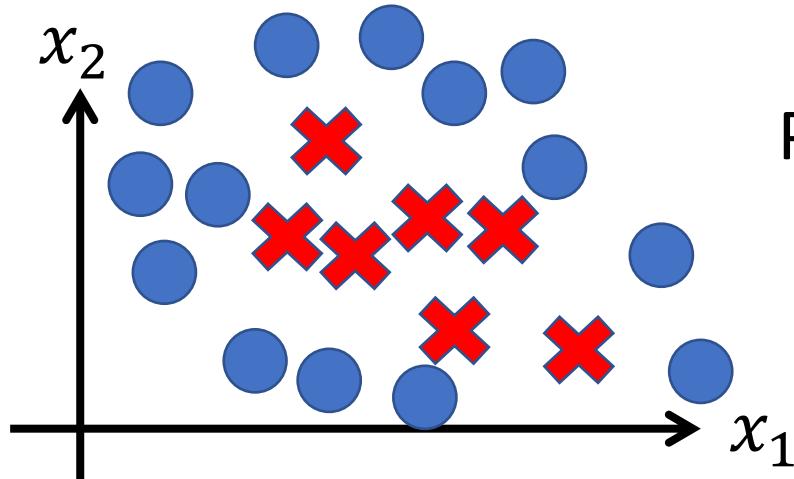
- project all points up to a two dimensional space using the mapping

$$x \rightarrow (x, x^2)$$

- We can indeed find a hyper plane to separate data with SVM.

The mapping  $x \rightarrow (x, x^2)$  in this case is called **KERNEL FUNCTION**

# Non-linear decision boundary



Predict  $y = 1$  if

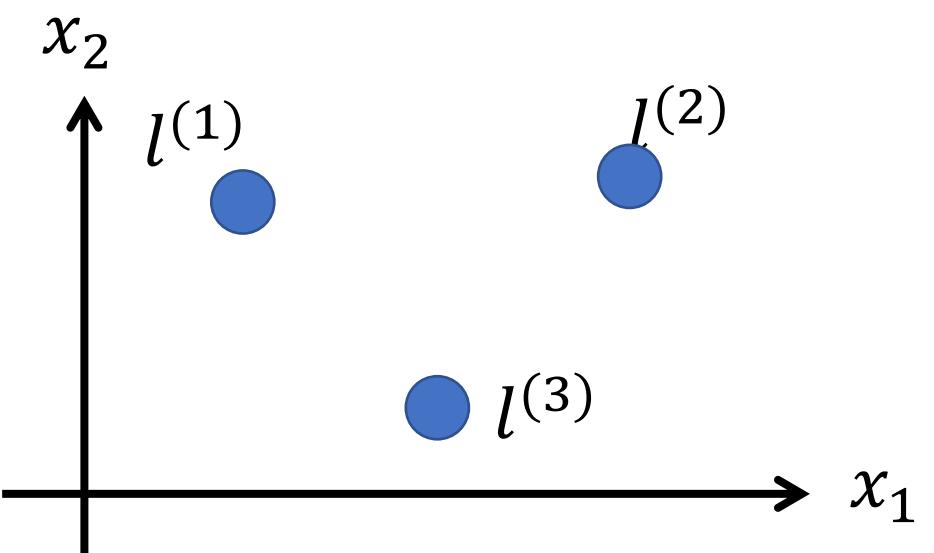
$$\begin{aligned}\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 \\ + \theta_4 x_1^2 + \theta_5 x_2^2 + \dots \geq 0\end{aligned}$$

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \dots$$

$$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, \dots$$

Is there a different/better choice of the features  $f_1, f_2, f_3, \dots$ ?

# Kernel



Given  $x$ , compute new features depending on proximity to landmarks  $l^{(1)}, l^{(2)}, l^{(3)}$

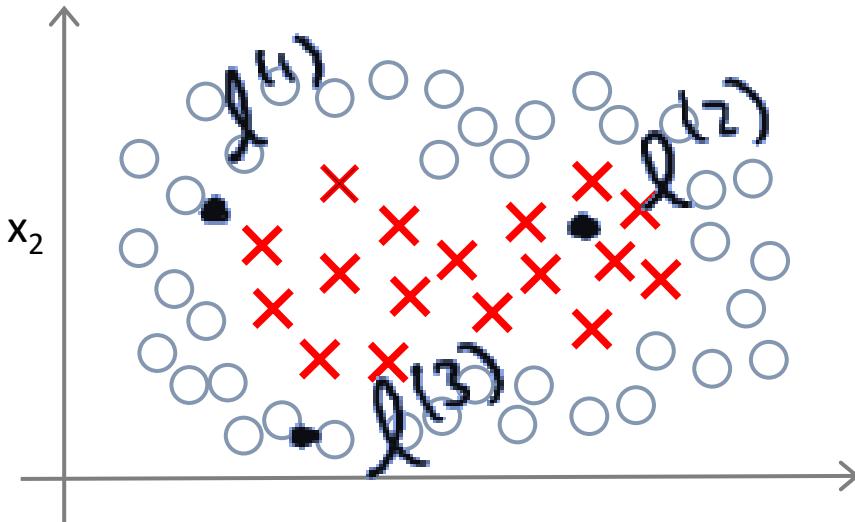
$$\begin{aligned}f_1 &= \text{similarity}(x, l^{(1)}) \\f_2 &= \text{similarity}(x, l^{(2)}) \\f_3 &= \text{similarity}(x, l^{(3)})\end{aligned}$$

Gaussian kernel

$$\text{similarity}(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$

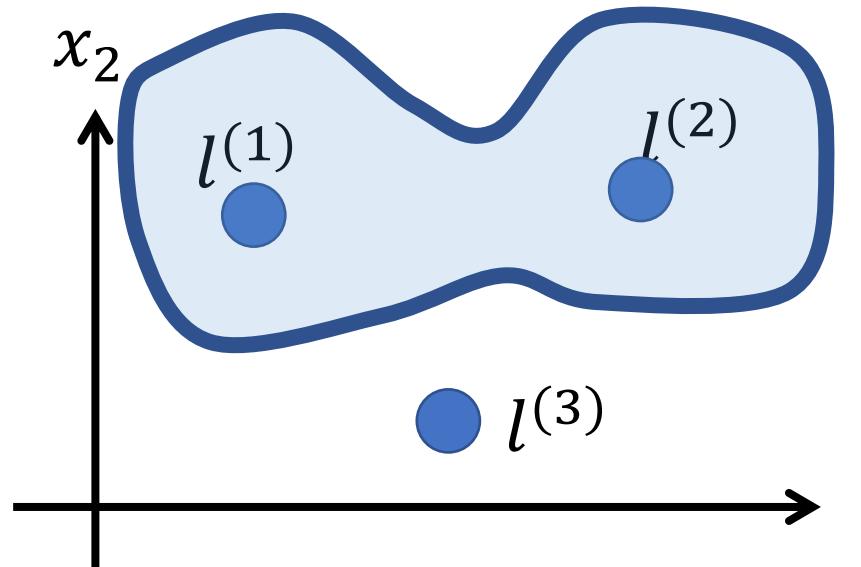
# Kernel trick in practice

$$f_i = \text{similarity}(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$



If  $x \approx l^{(1)}$  : f is approx 1

If  $x$  if far from  $l^{(1)}$  : f is approx 0



Predict  $y = 1$  if

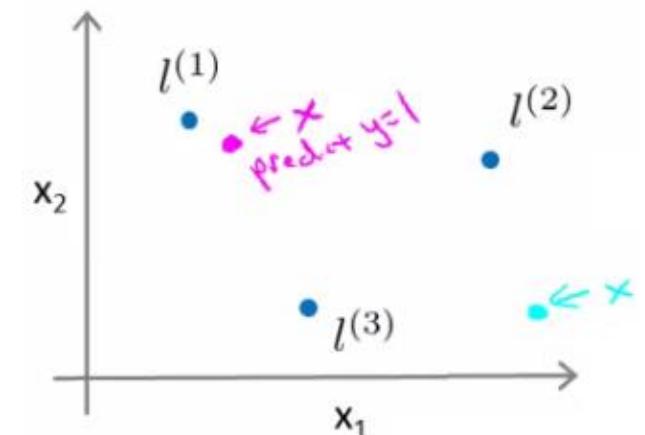
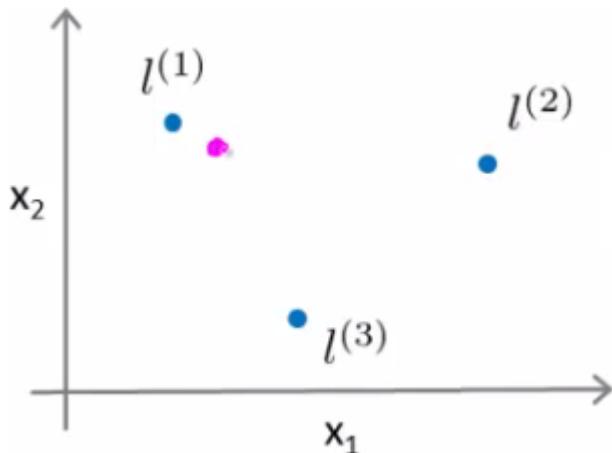
$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

Ex:  $\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$

$$f_1 = \text{similarity}(x, l^{(1)})$$

$$f_2 = \text{similarity}(x, l^{(2)})$$

$$f_3 = \text{similarity}(x, l^{(3)})$$



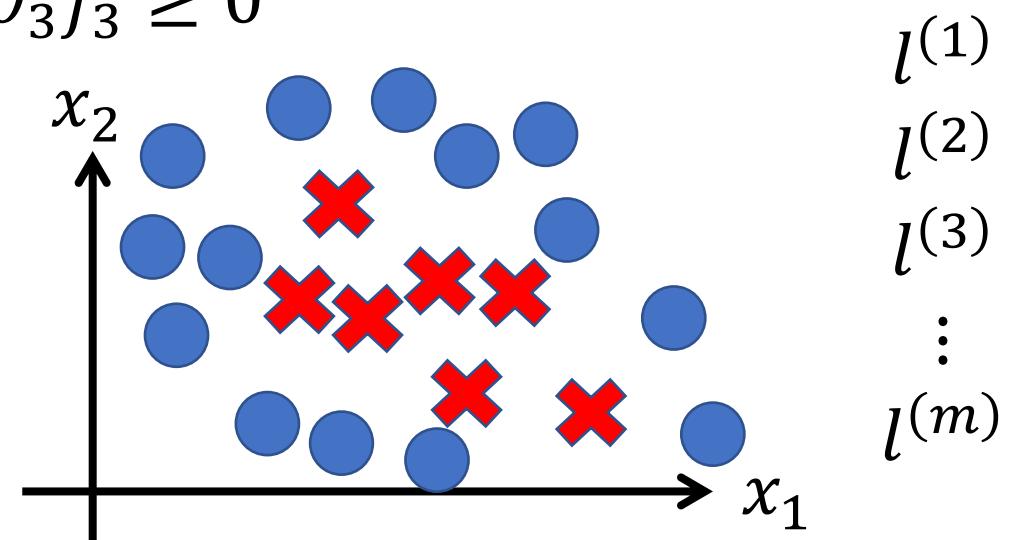
# Choosing the landmarks

- Given  $x$

$$f_i = \text{similarity}(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$

Predict  $y = 1$  if  $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get  $l^{(1)}, l^{(2)}, l^{(3)}, \dots$ ?



# SVM with kernels

- Given  $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$
- Choose  $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, l^{(3)} = x^{(3)}, \dots, l^{(m)} = x^{(m)}$
- Given example  $x$ :
  - $f_1 = \text{similarity}(x, l^{(1)})$
  - $f_2 = \text{similarity}(x, l^{(2)})$
  - ...
- For training example  $(x^{(i)}, y^{(i)})$ :
  - $x^{(i)} \rightarrow f^{(i)}$
  - $f_1^{(i)} = \text{similarity}(x^{(i)}, l^1)$
  - $f_2^{(i)} = \text{similarity}(x^{(i)}, l^2) \quad \dots \dots \quad f_m^{(i)} = \text{similarity}(x^{(i)}, l^m)$

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$$

# SVM with kernels

- Hypothesis: Given  $x$ , compute features  $f \in \mathbb{R}^{m+1}$ 
  - Predict  $y = 1 \text{ if } \theta^\top f \geq 0$

- **Training (original)**

$$\min_{\theta} C \left[ \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

- **Training (with kernel)**

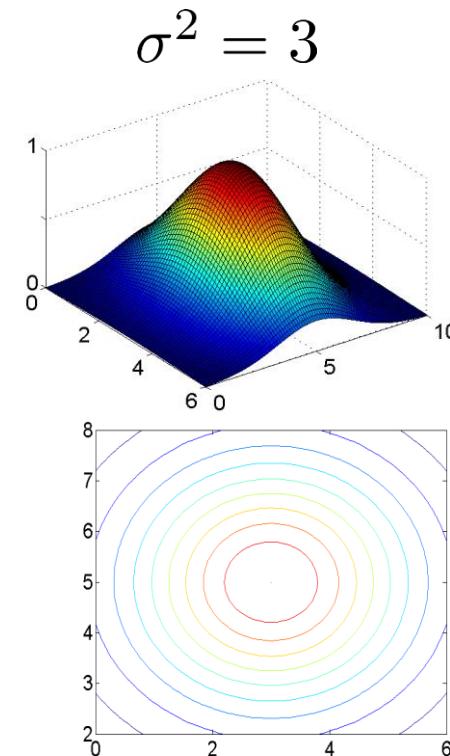
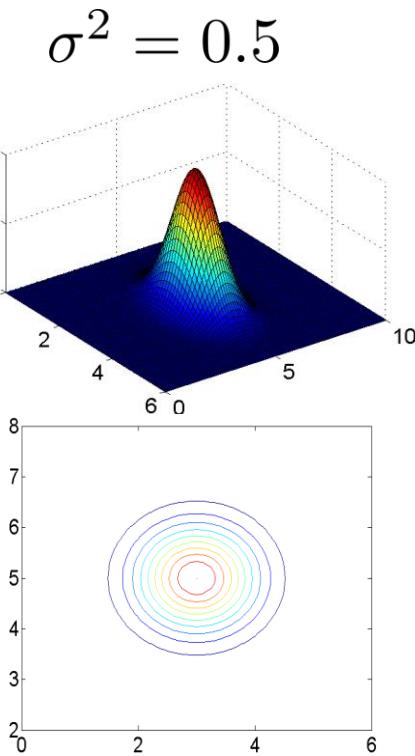
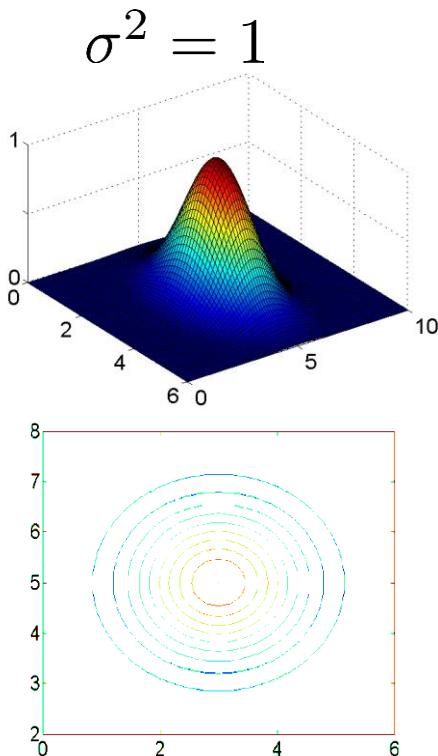
$$\min_{\theta} C \left[ \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^m \theta_j^2$$

# SVM parameters

- $C \left( = \frac{1}{\lambda} \right)$ 
  - Large  $C$ : Lower bias, high variance
  - Small  $C$ : Higher bias, low variance
  - may lead to overfitting
  - may lead to underfitting
- $\sigma^2$
- Large  $\sigma^2$ : features  $f_i$  vary more smoothly.
  - Higher bias, lower variance
- Small  $\sigma^2$ : features  $f_i$  vary less smoothly.
  - Lower bias, higher variance

# SVM with radial basis kernels: sigma

$$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x-l^{(1)}\|^2}{2\sigma^2}\right)$$



# Implementation of SVM

- SVM software package (e.g., liblinear, libsvm) to solve for  $\theta$
- Need to specify:
  - Choice of parameter  $C$ .
  - Choice of kernel (similarity function):
- Linear kernel:

Predict  $y = 1 \text{ if } \theta^\top x \geq 0$

When to use?

If  $n$  is large and  $m$  is small.

Not enough data – risk overfitting in high dimensional feature space

# Implementation of SVM

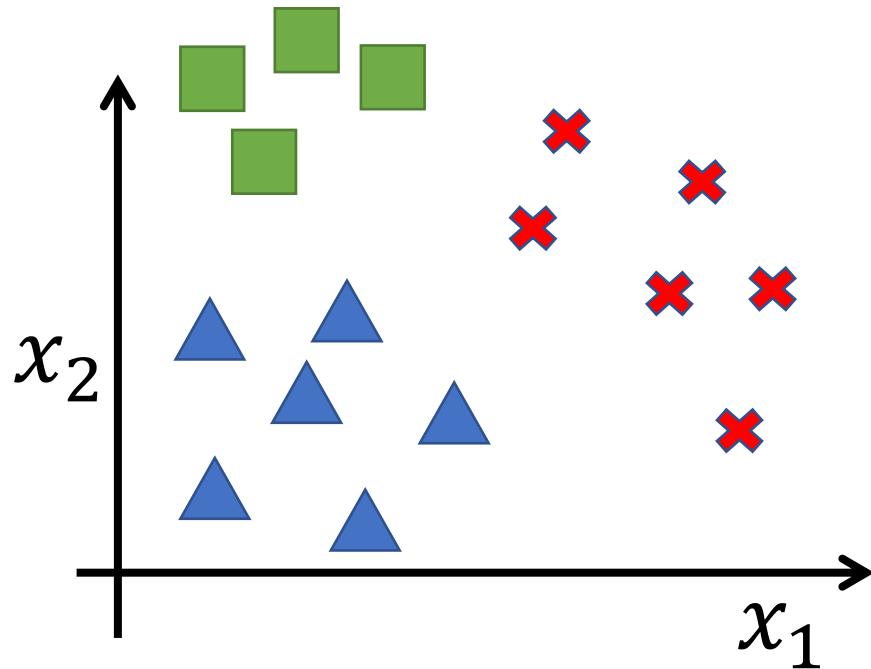
- Gaussian kernel:
  - Most commonly used kernel.
  - Need to choose  $\sigma^2$ . Need proper feature scaling
- $f_i = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$ , where  $l^{(i)} = x^{(i)}$
- When to use?

When n is small and m is large

# Kernel (similarity) functions

- Note: not all similarity functions make valid kernels.
- Many off-the-shelf kernels available:
  - Polynomial kernel
  - String kernel
  - Chi-square kernel
  - Histogram intersection kernel

# Multi-class classification

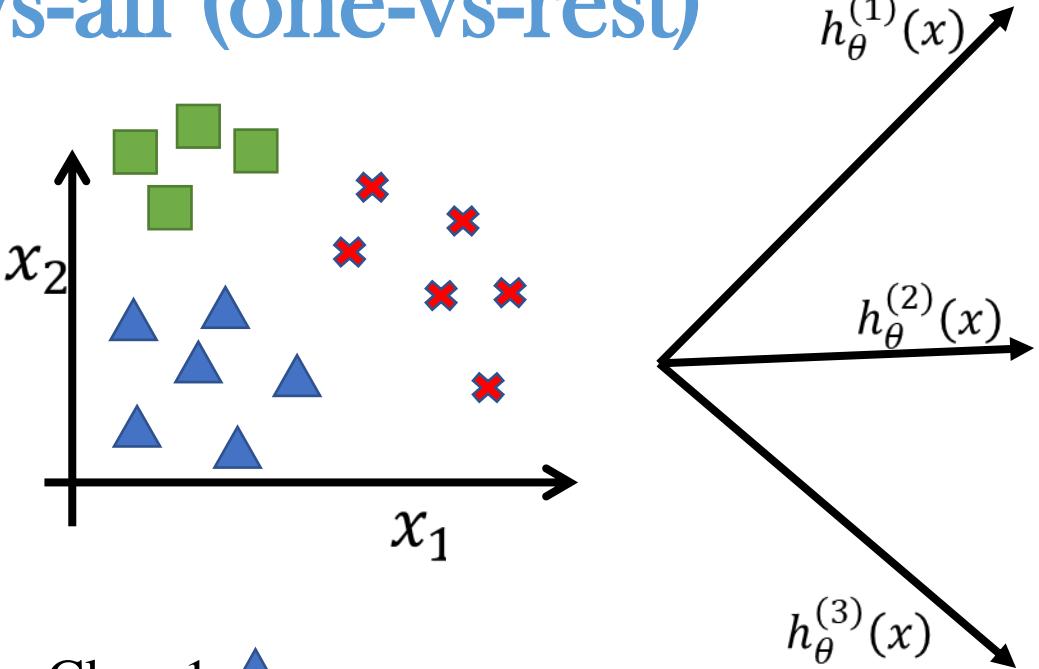


Many packages have built in multi class classification packages.

Otherwise we can use One Vs. All method

- Use one-vs.-all method. Train  $K$  SVMs, one to distinguish  $y = i$  from the rest, get  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$
- Pick class  $i$  with the largest  $\theta^{(i)^\top} x$

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

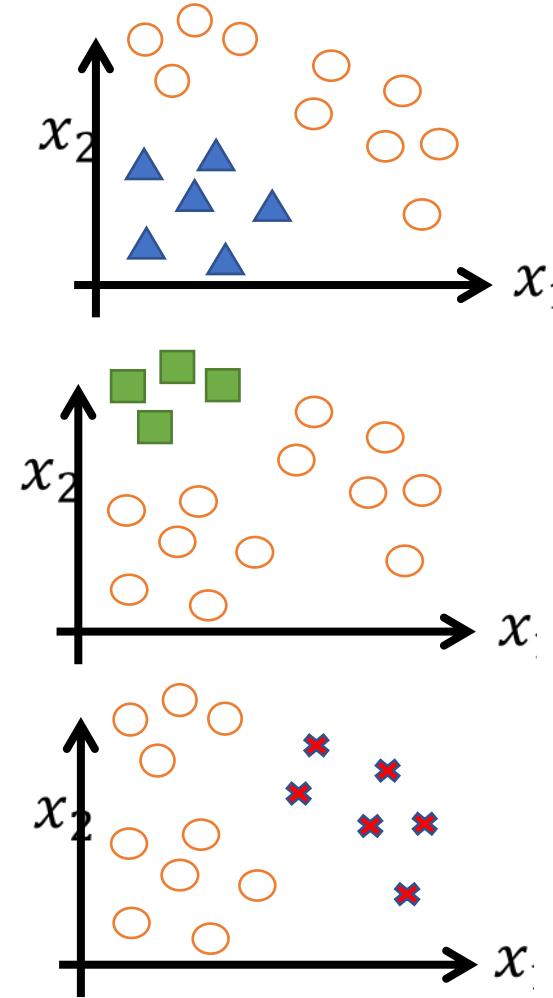


Class 1: ▲

Class 2: ■

Class 3: ✕

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



# Logistic regression vs. SVMs

- $n$  = number of features ( $x \in \mathbb{R}^{n+1}$ ),  $m$  = number of training examples
1. **If  $n$  is large (relative to  $m$ ):** ( $n = 10,000, m = 10 - 1000$ )  
→ Use logistic regression or SVM without a kernel (“linear kernel”)
  2. **If  $n$  is small,  $m$  is intermediate:** ( $n = 1 - 1000, m = 10 - 10,000$ )  
→ Use SVM with Gaussian kernel
  3. **If  $n$  is small,  $m$  is large:** ( $n = 1 - 1000, m = 50,000+$ )  
→ Create/add more features, then use logistic regression or linear SVM

# Things to remember

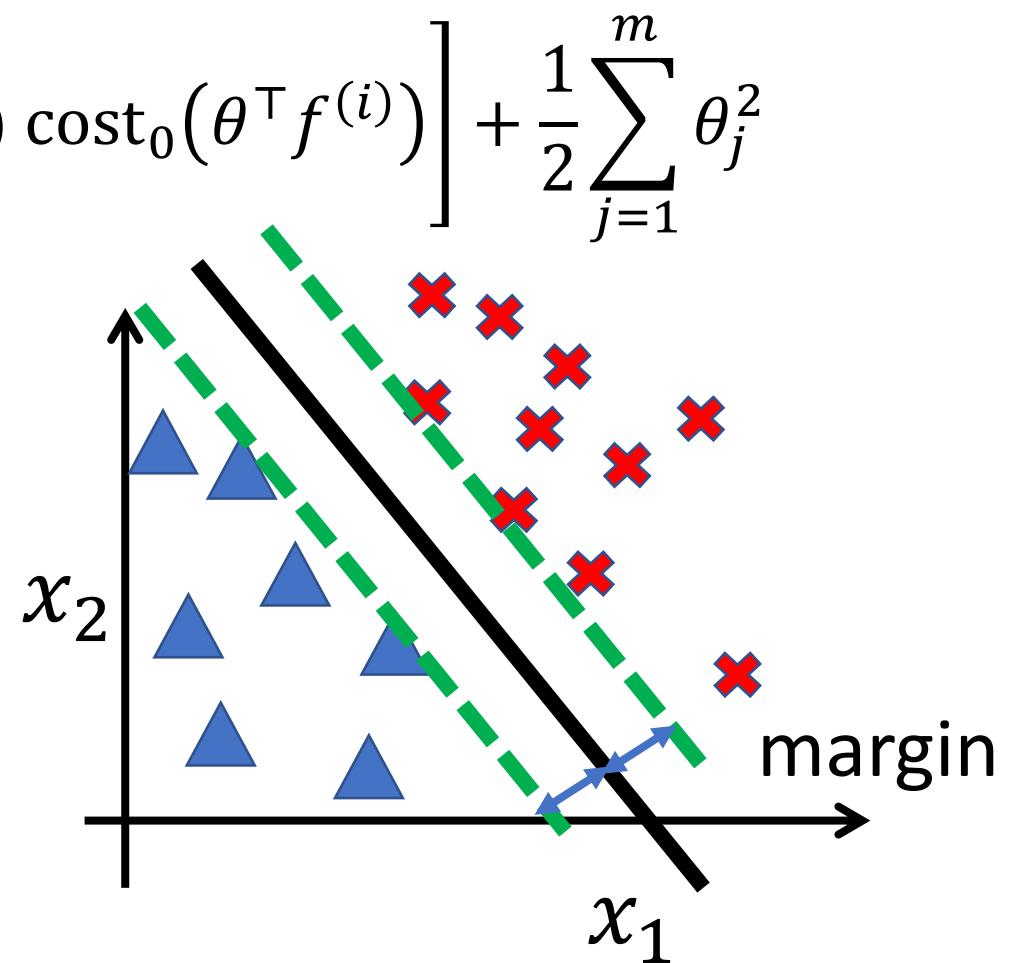
- Cost function

$$\min_{\theta} C \left[ \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^m \theta_j^2$$

- Large margin classification

- Kernels

- Using an SVM



# Example: Linear SVM

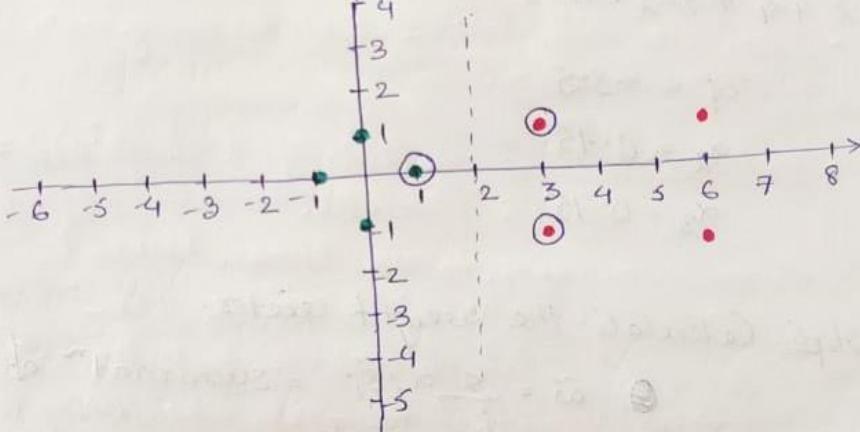
## Linear SVM Example

Suppose we have 8 datapoints.

$$4 \text{ are +vely labelled} \rightarrow \left\{ \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ -1 \end{pmatrix}, \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 6 \\ -1 \end{pmatrix} \right\}$$

$$4 \text{ are -vely labelled} \rightarrow \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right\}$$

Step1: Plot the datapoints in feature space



Step2: find the Support vectors [nearest datapts from both the sides].

$$\begin{aligned} s_1 &\rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ s_2 &\rightarrow \begin{pmatrix} 3 \\ 1 \end{pmatrix} \\ s_3 &\rightarrow \begin{pmatrix} 3 \\ -1 \end{pmatrix} \end{aligned} \quad \text{are support vectors.}$$

$$s_3 \rightarrow \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Step3: For each support, augmentation is done by adding 1 as a bias input.

$$\text{So, } s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ then } \tilde{s}_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

$$\text{likewise, } s_2 = \begin{pmatrix} 3 \\ 1 \end{pmatrix} \text{ then } \tilde{s}_2 = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$$

$$s_3 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \text{ then } \tilde{s}_3 = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}$$

Step4: Now, we need to calculate 3 variables  $\alpha_1, \alpha_2, \alpha_3$  to calculate the eight vectors  $\mathbf{w}$ .

$$\alpha_1 \tilde{s}_1 \cdot \tilde{s}_1 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_1 + \alpha_3 \tilde{s}_3 \cdot \tilde{s}_1 = -1 \quad [s_1 \rightarrow \text{+ve data}]$$

$$\alpha_1 \tilde{s}_1 \cdot \tilde{s}_2 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_2 + \alpha_3 \tilde{s}_3 \cdot \tilde{s}_2 = +1 \quad [s_2 \rightarrow \text{+ve data}]$$

$$\alpha_1 \tilde{s}_1 \cdot \tilde{s}_3 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_3 + \alpha_3 \tilde{s}_3 \cdot \tilde{s}_3 = +1 \quad [s_3 \rightarrow \text{+ve data}]$$

$$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = -1$$

$$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} = +1$$

$$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} = +1$$

$$\begin{aligned}\Rightarrow 2\alpha_1 + 4\alpha_2 + 4\alpha_3 &= 1 \\ \Rightarrow 4\alpha_1 + 11\alpha_2 + 9\alpha_3 &= 1 \\ \Rightarrow 4\alpha_1 + 9\alpha_2 + 11\alpha_3 &= 1\end{aligned}$$

$$\alpha_1 = -3.5$$

$$\alpha_2 = 0.75$$

$$\alpha_3 = 0.75$$

Step 5: Calculate the weight vector.

(\*)  $\tilde{\omega} = \sum_i \alpha_i \tilde{s}_i$  = summat<sup>n</sup> of  $\alpha_i s_i$  where  
 $i \rightarrow \# \text{nearest datapoint}$

$$= -3.5 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix}$$

Step 6: But remember that our vectors are augmented with a bias. So, we can equal the last entry in  $\tilde{\omega}$  as the hyperplane offset  $b$  and write the separating hyperplane equation

$$y = \omega x + b,$$

where  $\omega = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$  and  
 $b = -2$ .

Thus, the equation of hyperplane is obtained from training samples

**Course code:** CSE3008

**Course Title:** Introduction to Machine Learning

**Module 4: Bayesian and Computational Learning**

**Dr. Usha Rani Gogoi**

**Assistant Professor Sr. Grade 1**

**[usha.gogoi@vitap.ac.in](mailto:usha.gogoi@vitap.ac.in)**

**SCOPE**

# Content

- Probability basics
- Estimating parameters from data
  - Maximum likelihood (ML)
  - Maximum a posteriori estimation (MAP)
- Naïve Bayes

# Random Variables

- A Random Variable is a set of **possible values** from a random experiment. A random variable  $x$  takes on a defined set of values with different probabilities.
  - Example: Rolling a dice: we could get 6 possible outcomes (random), each of which occur with probability one-sixth.
  - Example: Tossing a coin: we could get Heads or Tails. Let **Heads=0** and **Tails=1** and we have a Random Variable "X":
    - In Short  $X = \{0, 1\}$
    - Thus, we have an **experiment** (such as tossing a coin)
    - We give **values** to each event.
    - The **set of values** is a **Random Variable**

<i>Random Variable</i>	<i>Possible Values</i>	<i>Random Events</i>
$X = \begin{cases} 0 \\ 1 \end{cases}$		

# Sample Space and Probability

- A Random Variable's set of values is the Sample Space.

- Example: Throw a dice once

Random Variable  $X$  = "The score shown on the top face".

$X$  could be 1, 2, 3, 4, 5 or 6

So the Sample Space is {1, 2, 3, 4, 5, 6}

- Probability is how frequently we expect different outcomes to occur if we repeat the experiment over and over ("frequentist" view)

- Example: Throw a dice once

$$X = \{1, 2, 3, 4, 5, 6\}$$

In this case they are all equally likely, so the probability of any one is  $1/6$

$$P(X = 1) = 1/6$$

$$P(X = 2) = 1/6$$

$$P(X = 3) = 1/6$$

$$P(X = 4) = 1/6$$

$$P(X = 5) = 1/6$$

$$P(X = 6) = 1/6$$

- Note that the sum of the probabilities = 1, as it should be.

# Example:

- Example: How many heads when we toss 3 coins?

$X$  = "The number of Heads" is the Random Variable.

In this case, there could be 0 Heads (if all the coins land Tails up), 1 Head, 2 Heads or 3 Heads. So the Sample Space = {0, 1, 2, 3}

But this time the outcomes are NOT all equally likely. The three coins can land in eight possible ways:

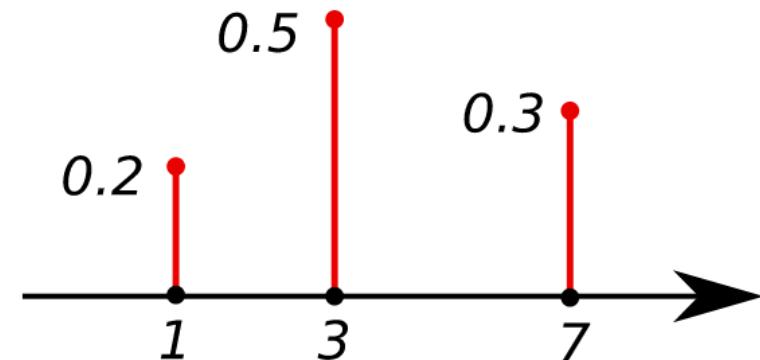
Looking at the table we see just **1 case of Three Heads**, but **3 cases of Two Heads**, **3 cases of One Head**, and **1 case of Zero Heads**. So:

- $P(X = 3) = 1/8$
- $P(X = 2) = 3/8$
- $P(X = 1) = 3/8$
- $P(X = 0) = 1/8$

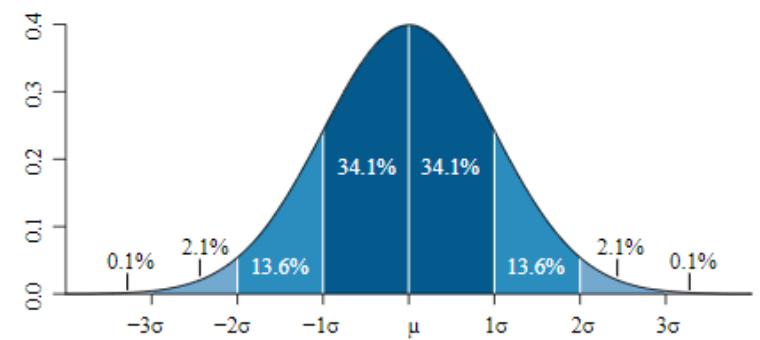
X = "Number of Heads"	
HHH	 3
HHT	 2
HTH	 2
HTT	 1
THH	 2
THT	 1
TTH	 1
TTT	 0

# Random variables can be discrete or continuous

- **Discrete** random variables have a countable number of outcomes (no in between values)
  - Examples: Dead/alive, treatment/placebo, dice, counts, etc.



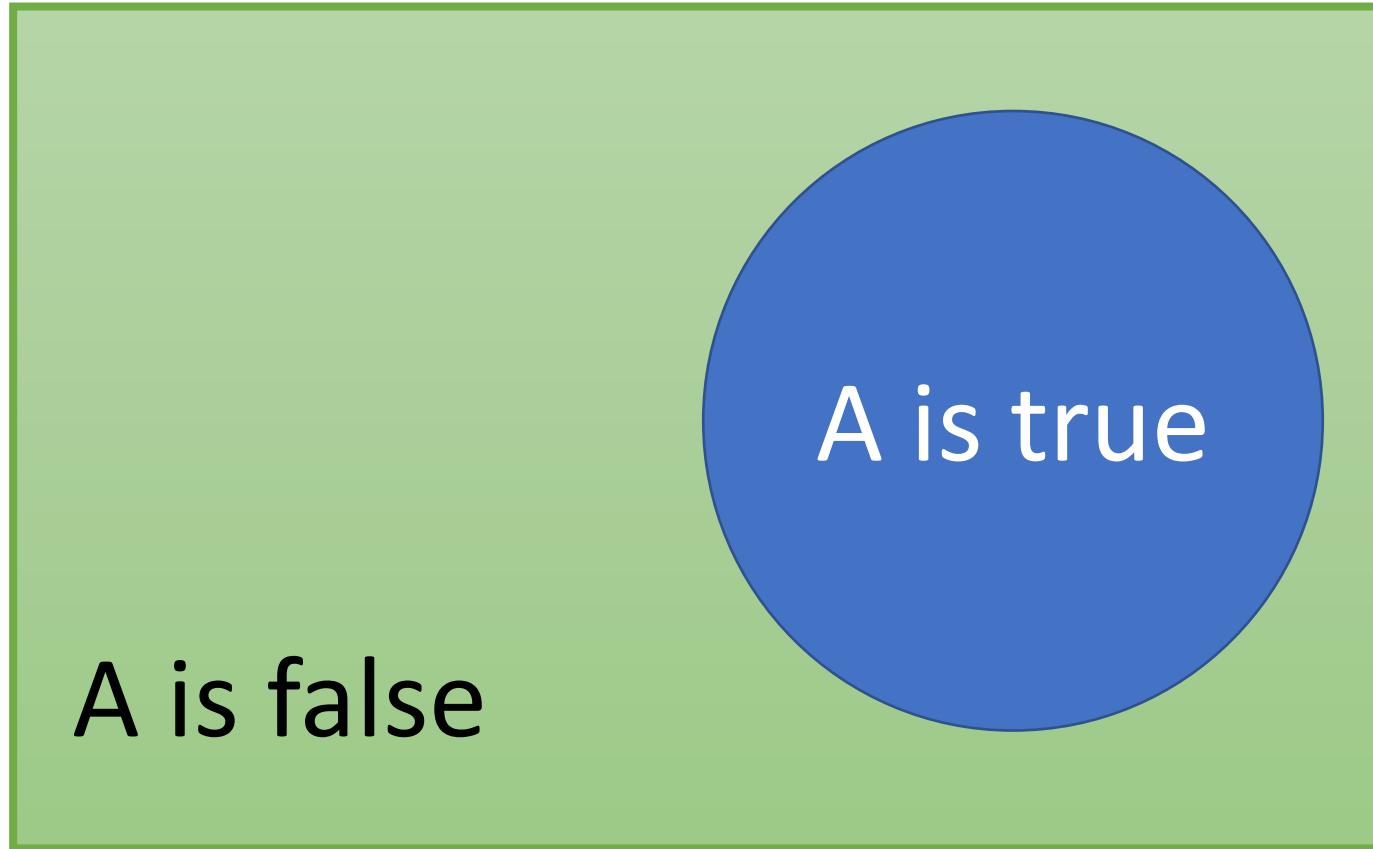
- **Continuous** random variables have an infinite continuum of possible values.
  - Examples: blood pressure, weight, the speed of a car, the real numbers from 1 to 6.



# Probability functions

- A probability function is used to describe the probability distribution of a random variable  $X$  i.e. PF maps the possible values of  $x$  against their respective probabilities of occurrence,  $p(x)$ .
- $p(x)$  is a number from 0 to 1.0.
- The area under a probability function is always 1.

# Visualizing probability $P(A)$

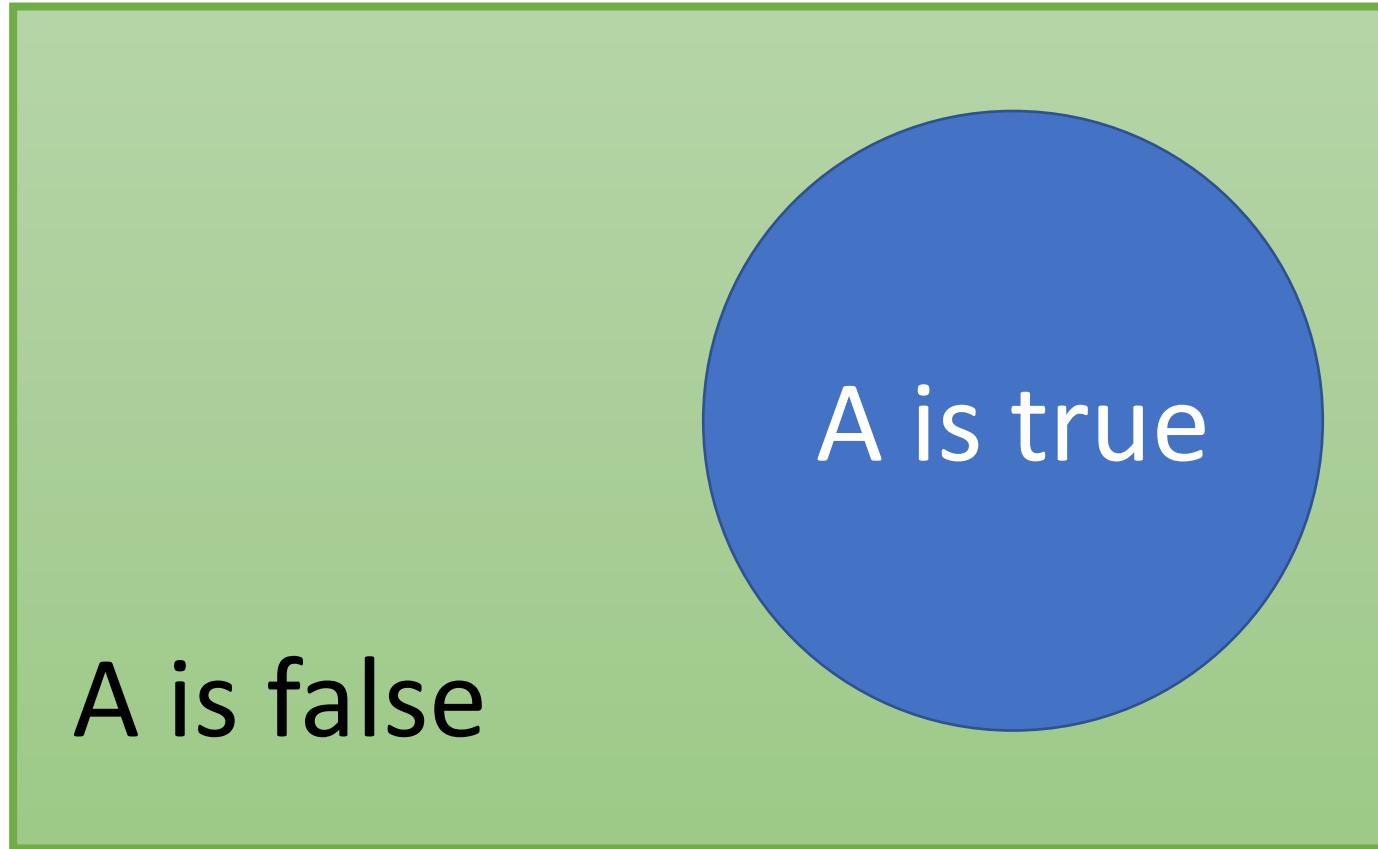


Sample space

Area = 1

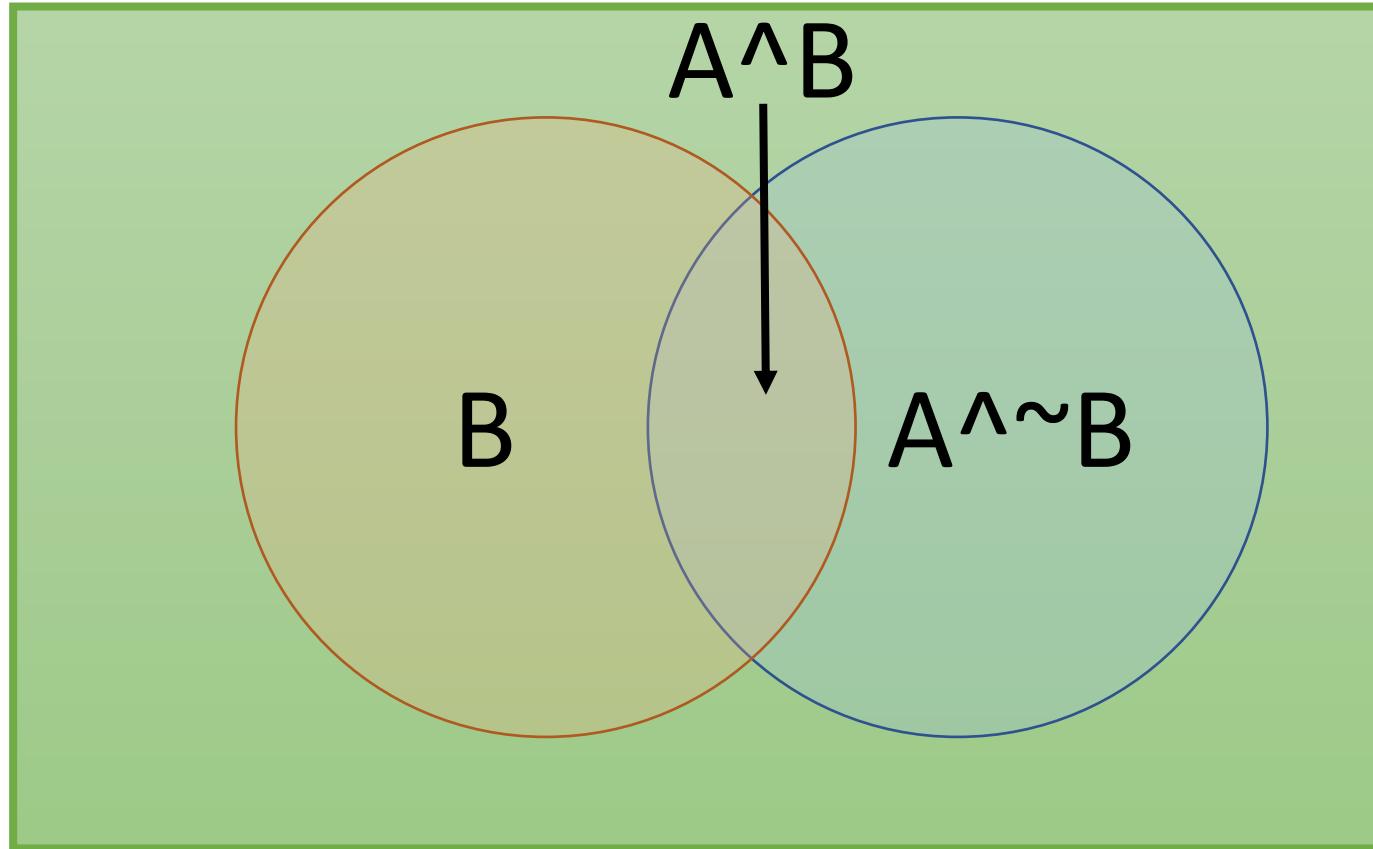
$$P(A) = \text{Area of the blue circle}$$

# Visualizing probability $P(A) + P(\sim A)$



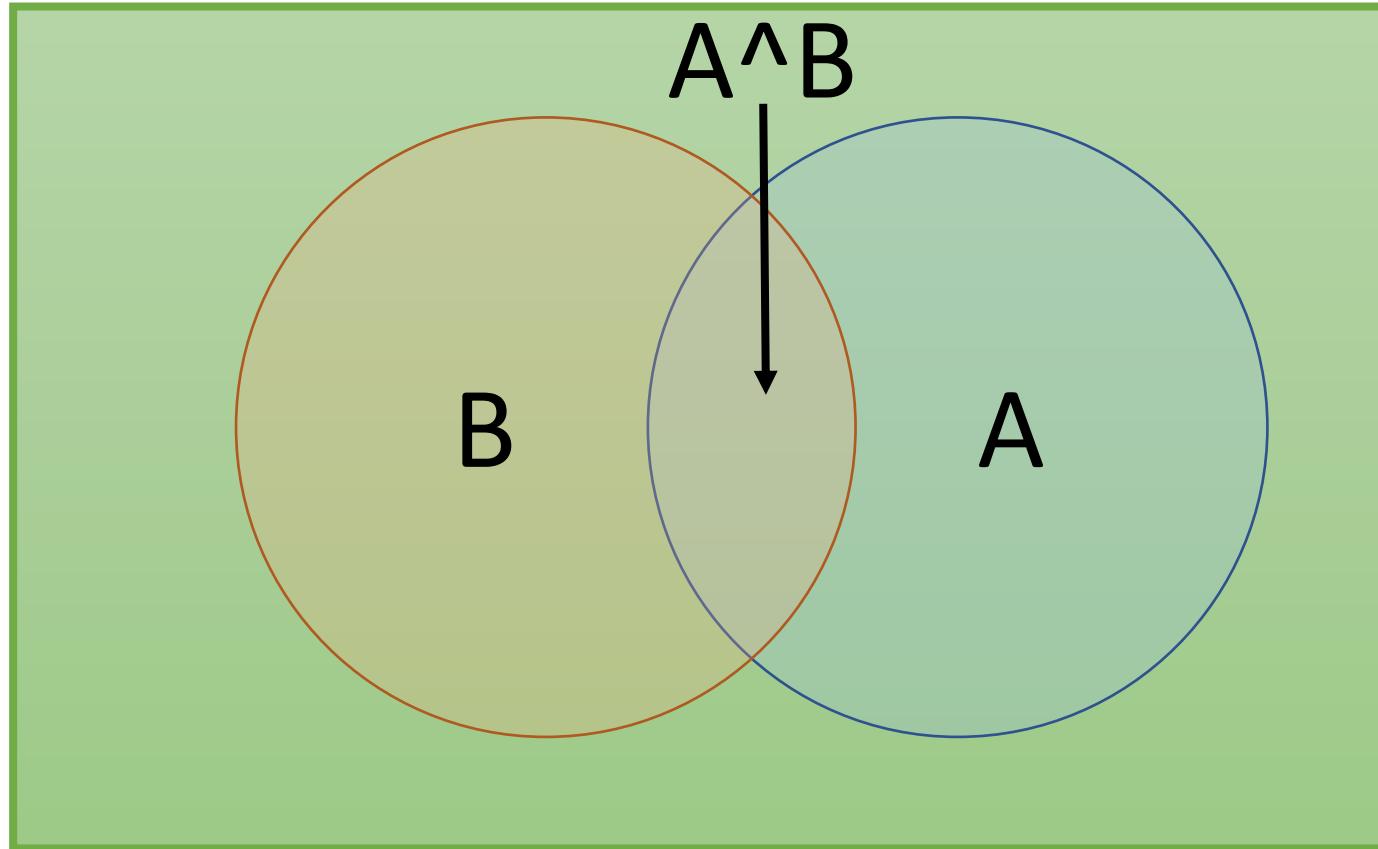
$$P(A) + P(\sim A) = 1$$

# Visualizing probability $P(A)$



$$P(A) = P(A \wedge B) + P(A \wedge \sim B)$$

# Visualizing Conditional Probability

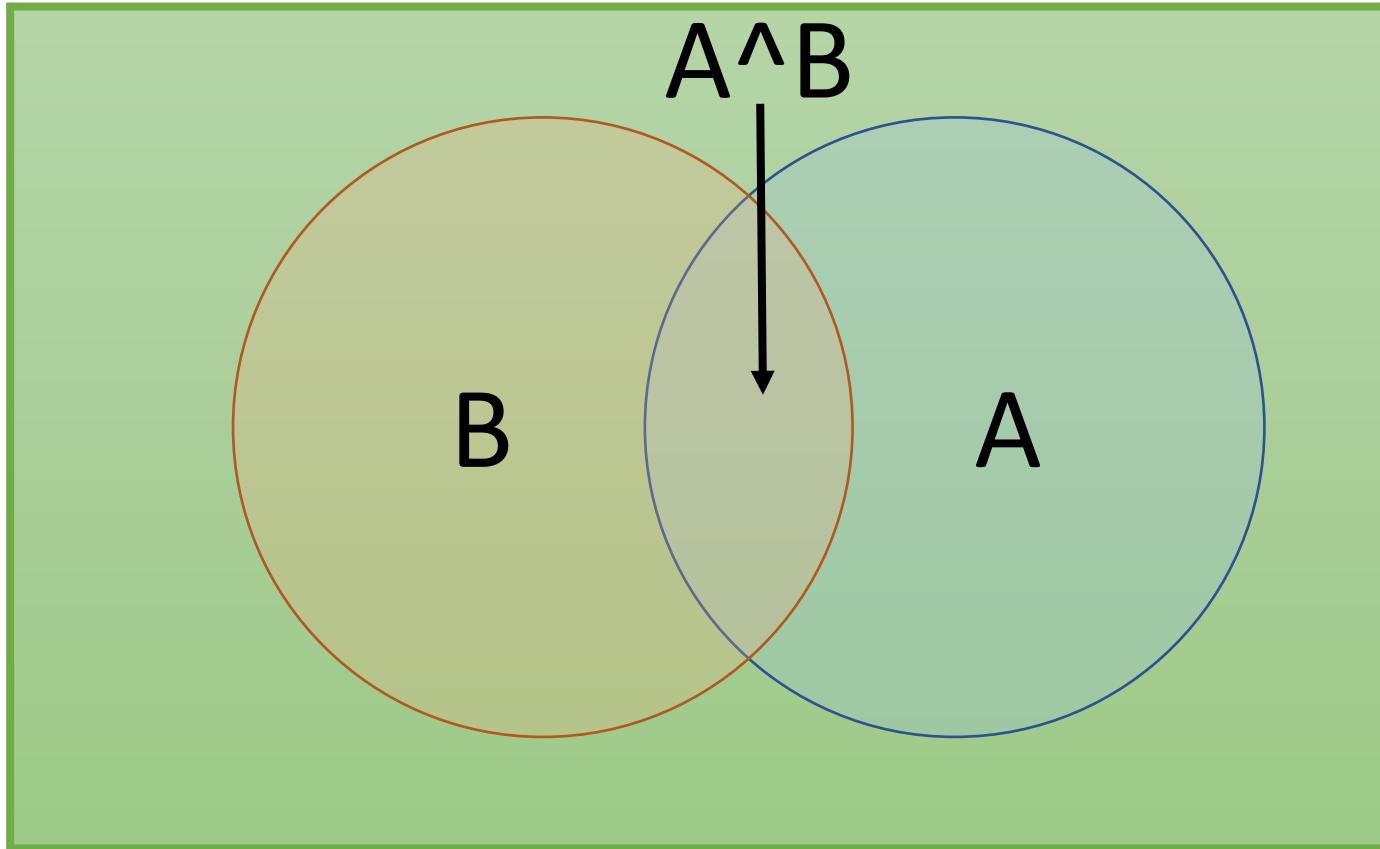


$$P(A|B) = P(A \wedge B)/P(B)$$

Corollary: The chain rule

$$\begin{aligned} P(A \wedge B) &= P(A|B)P(B) \\ &= P(B|A)P(A) \end{aligned}$$

# Bayes rule



Thomas Bayes

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

Corollary: The chain rule

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

# Other forms of Bayes rule

**Form 1:**  $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

**Form 2:**  $P(A|B^{\wedge}X) = \frac{P(B|A^{\wedge}X)P(A^{\wedge}X)}{P(B^{\wedge}X)}$

**Form 3:**  $P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\sim A)P(\sim A)}$

## Proof for Forms:

$$\textbf{Form 1: } P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

$$\textbf{Form 3: } P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{P(B|A)P(A)}{P(B \wedge A) + P(B \wedge \sim A)}$$

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

# Applying Bayes Rule

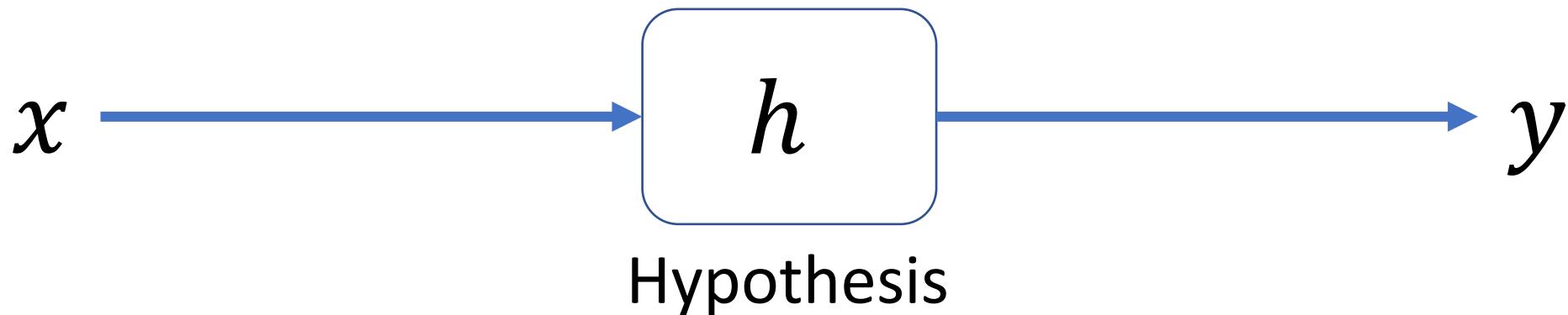
$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\sim A)P(\sim A)}$$

- $A = \text{you have the flu}$ ,       $B = \text{you just coughed}$
- **Assume:**
  - $P(A) = 0.05$
  - $P(B|A) = 0.8$
  - $P(B|\sim A) = 0.2$
- **What is  $P(\text{flu} \mid \text{cough}) = P(A|B)$ ?** = 0.17

$$P(A|B) = \frac{0.8 \times 0.05}{0.8 \times 0.05 + 0.2 \times 0.95} \sim 0.17$$

# Function Approximation

Why we are learning this?



Learn  $P(Y|X)$

# Joint distribution

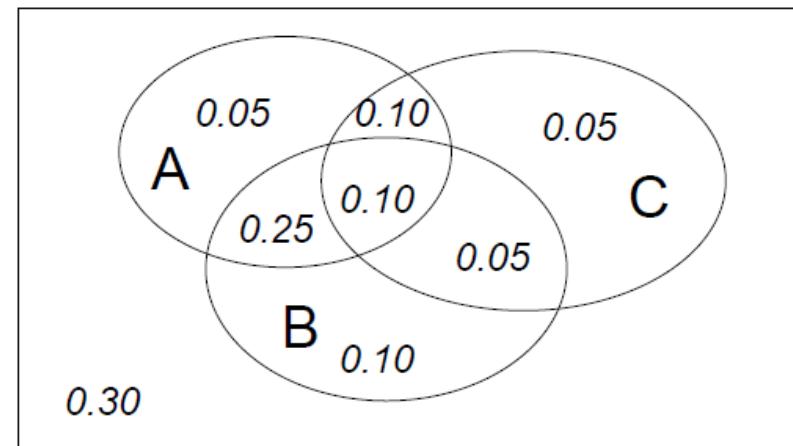
- Making a joint distribution of M variables

1. Make a truth table listing all combinations

A	B	C	Prob
0	0	0	0.30
0	0	1	0.05
0	1	0	0.10
0	1	1	0.05
1	0	0	0.05
1	0	1	0.10
1	1	0	0.25
1	1	1	0.10

2. For each combination of values, say how probable it is

3. Probability must sum to 1



# Using Joint Distribution

- Once you have the JD you can ask for the probability of **any** logical expression involving these variables
- $P(E) = \sum_{\text{rows matching } E} P(\text{row})$
- $P(E_1 | E_2) = \frac{\sum_{\text{rows matching } E_1 \text{ and } E_2} P(\text{row})}{\sum_{\text{rows matching } E_2} P(\text{row})}$



# Using the Joint

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
	v1:40.5+	poor	0.134106
		rich	0.105933

$$P(\text{Poor Male}) = 0.4654$$

$$P(E) = \sum_{\text{rows matching } E} P(\text{row})$$

# Inference with the Joint

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
v1:40.5+		poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
v1:40.5+		poor	0.134106
		rich	0.105933

$$P(E_1 | E_2) = \frac{P(E_1 \wedge E_2)}{P(E_2)} = \frac{\sum_{\text{rows matching } E_1 \text{ and } E_2} P(\text{row})}{\sum_{\text{rows matching } E_2} P(\text{row})}$$

$$P(\text{Male} | \text{Poor}) = 0.4654 / 0.7604 = 0.612$$

# Learning and the Joint Distribution

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
	v1:40.5+	poor	0.134106
		rich	0.105933

Suppose we want to learn the function  $f: \langle G, H \rangle \rightarrow W$

Equivalently,  $P(W | G, H)$

Solution: learn joint distribution from data, calculate  $P(W | G, H)$

e.g.,  $P(W=\text{rich} | G = \text{female}, H = 40.5-) =$

# The solution to learn $P(Y|X)$ ?

- Main problem: learning  $P(Y|X)$  may require more data than we have
- Say, learning a joint distribution with 100 attributes
- # of rows in this table?  $2^{100} \geq 10^{30}$
- # of people on earth?  $10^9$



# What should we do?

1. Be smart about  
**how we estimate probabilities from sparse data**
  - Maximum likelihood estimates (MLE)
  - Maximum a posteriori estimates (MAP)
  
2. Be smart about  
**how to represent joint distributions**
  - Bayes network, graphical models

- Probability basics
- Estimating parameters from data
  - Maximum likelihood (MLE)
  - Maximum a posteriori (MAP)
- Naive Bayes

# Estimating the probability



$$X = 1 \quad X = 0$$

- Flip the coin repeatedly, observing
    - It turns heads  $\alpha_1$  times
    - It turns tails  $\alpha_0$  times
  - Your estimate for  $P(X = 1)$  is?
- 
- Case A: 100 flips: 51 Heads ( $X = 1$ ), 49 Tails ( $X = 0$ )  
 $P(X = 1) = ?$
  - Case B: 3 flips: 2 Heads ( $X = 1$ ), 1 Tails ( $X = 0$ )  
 $P(X = 1) = ?$

# Two principles for estimating parameters

- **Maximum Likelihood Estimate (MLE)**

Choose  $\theta$  that maximizes probability of observed data

$$\hat{\theta}^{\text{MLE}} = \operatorname{argmax}_{\theta} P(\text{Data}|\theta) = \frac{\alpha_1}{\alpha_1 + \alpha_0}$$

- **Maximum a posteriori estimation (MAP)**

Choose  $\theta$  that is most probable given prior probability and data

$$\hat{\theta}^{\text{MAP}} = \operatorname{argmax}_{\theta} P(\theta|D) = \operatorname{argmax}_{\theta} \frac{P(\text{Data}|\theta)P(\theta)}{P(\text{Data})}$$

# Maximum likelihood estimate



- Each flip yields Boolean value for  $X$

$$X \sim \text{Bernoulli}: P(X) = \theta^X(1 - \theta)^{1-X}$$

$$X = 1 \quad X = 0$$

$$P(X = 1) = \theta$$

$$P(X = 0) = 1 - \theta$$

- Data set  $D$  of independent, identically distributed (iid) flips, produces  $\alpha_1$  ones,  $\alpha_0$  zeros

$$P(D|\theta) = P(\alpha_1, \alpha_0|\theta) = \theta^{\alpha_1}(1 - \theta)^{\alpha_0}$$

$$\hat{\theta} = \operatorname{argmax}_{\theta} P(D|\theta) = \frac{\alpha_1}{\alpha_1 + \alpha_0}$$

$$\hat{\theta} = \arg \max_{\theta} \ln(P(D|\theta)) \rightarrow \text{Max LE for } \theta$$

$$= \arg \max_{\theta} \ln \theta^{\alpha_1} (1-\theta)^{\alpha_0}$$

derivative = 0       $\frac{d}{d\theta} \ln P(D|\theta) = 0$

$$\frac{\partial}{\partial \theta} \ln n \approx \frac{1}{\lambda}$$

$$\theta = \frac{\partial}{\partial \theta} \ln P(D|\theta) = \frac{\partial}{\partial \theta} \ln [\theta^{\alpha_1} (1-\theta)^{\alpha_0}] = \frac{1}{\theta} [\alpha_1 \ln \theta + \alpha_0 \ln (1-\theta)]$$

$$\theta = \alpha_1 \frac{1}{\theta} \ln \theta + \alpha_0 \frac{1}{1-\theta} \ln (1-\theta)$$

$$\theta = \alpha_1 \frac{1}{\theta} + \alpha_0 \frac{2 \ln (1-\theta)}{2(1-\theta)} \cdot \frac{1}{\theta}$$

$$\theta = \frac{\alpha_1}{\theta} + \alpha_0 \frac{1}{(1-\theta)} \cdot (-1)$$

$$\theta = \frac{\alpha_1}{\theta} - \frac{\alpha_0}{1-\theta}$$

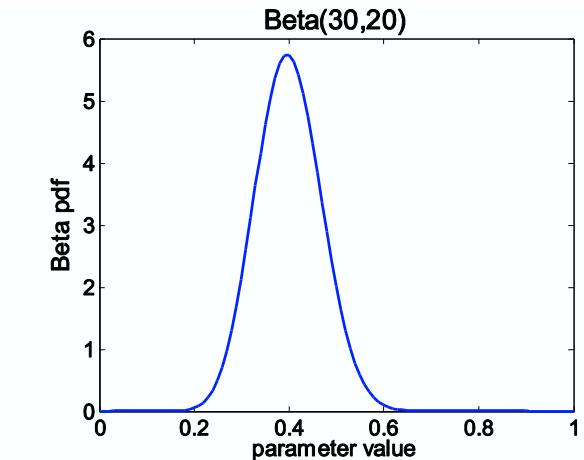
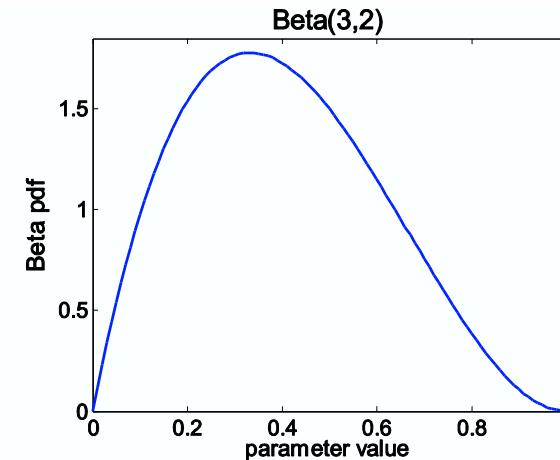
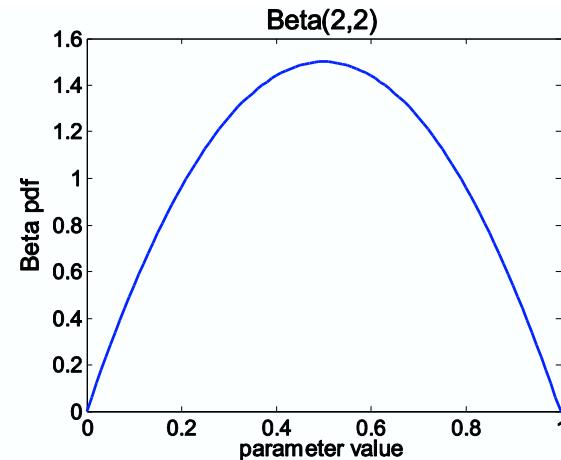
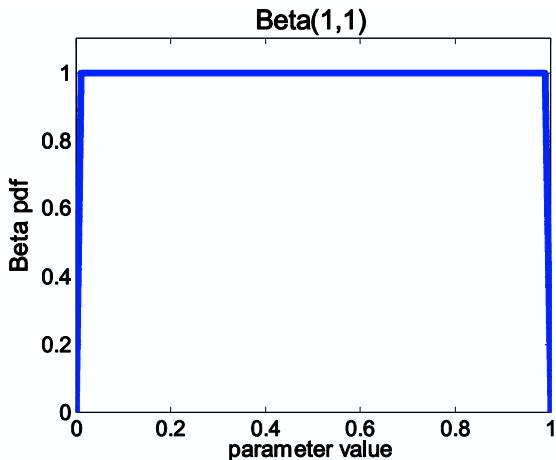
$$\boxed{\theta = \frac{\alpha_1}{\alpha_1 + \alpha_0}}$$

# Maximum Likelihood Estimation

- **Goal :** Find the parameter  $p$  that maximizes the likelihood of seeing all training samples.
  - Example: 6H, 4T
  - $P(H) = p$ ,  $P(T) = 1-p$
- 
- $L(p) = p^6(1-p)^4$  likelihood  $\longrightarrow$  Find the total likelihood
  - $\log L(p) = 6 \log(p) + 4 \log(1-p)$   $\longrightarrow >$  Take log likelihood
  - $d(\log L(p))/dp = 6/p - 4/(1-p) = 0$   $\longrightarrow \rightarrow$  Take derivative
  - $P = 6/10$   $\longrightarrow \rightarrow$  Solve

# Beta prior distribution $P(\theta)$

- $P(\theta) = Beta(\beta_1, \beta_0) = \frac{1}{B(\beta_1, \beta_0)} \theta^{\beta_1-1} (1 - \theta)^{\beta_0-1}$
- $\beta$  makes the probability integrated up to one and a well formed distribution function( a constant and a kind of normalization)



# Maximum likelihood estimate



$X = 1 \quad X = 0$

- Data set  $D$  of iid flips,  
produces  $\alpha_1$  ones,  $\alpha_0$  zeros

$$P(\text{Data}|\theta) = P(\alpha_1, \alpha_0|\theta) = \theta^{\alpha_1}(1-\theta)^{\alpha_0}$$

- Assume prior (Conjugate prior: Closed form representation of posterior)
- 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

$$P(\theta) = \text{Beta}(\beta_1, \beta_0) = \frac{1}{B(\beta_1, \beta_0)} \theta^{\beta_1-1} (1-\theta)^{\beta_0-1}$$

$$\hat{\theta} = \operatorname{argmax}_{\theta} P(D|\theta) P(\theta) = \frac{\alpha_1 + \beta_1 - 1}{(\alpha_1 + \beta_1 - 1) + (\alpha_0 + \beta_0 - 1)}$$

# Some terminology

- **Likelihood function**  $P(Data|\theta)$
- **Prior**  $P(\theta)$
- **Posterior**  $P(\theta|Data)$
- Conjugate prior:  
**Prior**  $P(\theta)$  is the conjugate prior for a **likelihood function**  $P(Data|\theta)$  if the **prior**  $P(\theta)$  and the **posterior**  $P(\theta|Data)$  have the same form.
- Example (coin flip problem)
  - **Prior**  $P(\theta)$ : Beta( $\beta_1, \beta_0$ )    **Likelihood**  $P(Data|\theta)$ : Binomial  $\theta^{\alpha_1}(1 - \theta)^{\alpha_0}$
  - **Posterior**  $P(\theta|Data)$ : Beta( $\alpha_1 + \beta_1, \alpha_0 + \beta_0$ )

# How many parameters?

- Suppose  $X = [X_1, \dots, X_n]$ , where  $X_i$  and  $Y$  are Boolean random variables

To estimate  $P(Y|X_1, \dots, X_n)$

When  $n = 2$  (Gender, Hours-worked)?

When  $n = 30$ ? (i. e.,  $P(Y|X_1, \dots, X_{30}) = ?$ )

Let's learn classifiers by learning  $P(Y|X)$

Consider  $Y = \text{Wealth}$ ,  $X = \langle \text{Gender}, \text{HoursWorked} \rangle$



Gender	HrsWorked	P(rich   G,HW)	P(poor   G,HW)
F	<40.5	.09	.91
F	>40.5	.21	.79
M	<40.5	.23	.77
M	>40.5	.38	.62

# Can we reduce paras using Bayes rule?

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

- How many parameters for  $P(X_1, \dots, X_n | Y)$ ?  
 $(2^n - 1) \times 2$
- How many parameters for  $P(Y)$ ?  
 $(2^n - 1) \times 2 + 1$

# Contents

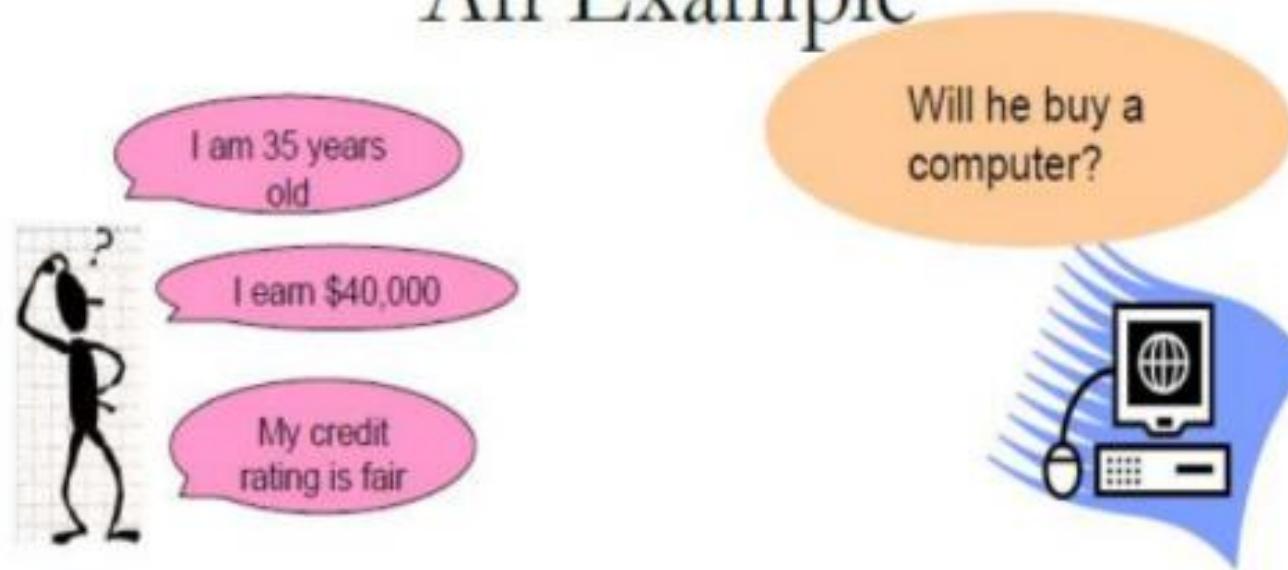
- Probability basics
- Estimating parameters from data
  - Maximum likelihood (ML)
  - Maximum a posteriori estimation (MAP)
- **Naive Bayes**

# Naïve Bayes Classifier/ Bayesian Classifier

- Statistical Method for classification.
- Supervised Learning Method.
- Assumes an underlying probabilistic model, the Bayes theorem.
- Can solve problems involving both categorical and continuous valued attributes.
- Named after Thomas Bayes, who proposed the Bayes Theorem

# Naïve Bayes Classifier/ Bayesian Classifier

## An Example



- X : 35 years old customer with an income of \$40,000 and fair credit rating.
- H : Hypothesis that the customer will buy a computer.

# Naïve Bayes Classifier/ Bayesian Classifier

- The Bayes Theorem:  $P(H|X) = P(X|H) P(H) / P(X)$
- $P(H|X)$  : Probability that the customer will buy a computer given that we know his age, credit rating and income. (Posterior Probability of H)
- $P(H)$  : Probability that the customer will buy a computer regardless of age, credit rating, income (Prior Probability of H)
- $P(X|H)$  : Probability that the customer is 35 yrs old, have fair credit rating and earns \$40,000, given that he has bought our computer (Posterior Probability of X)
- $P(X)$  : Probability that a person from our set of customers is 35 yrs old, have fair credit rating and earns \$40,000. (Prior Probability of X)

# Naïve Bayes Classifier/ Bayesian Classifier

- ▶ D : Set of tuples
  - Each Tuple is an ‘n’ dimensional attribute vector
  - $X : (x_1, x_2, x_3, \dots, x_n)$
  - where  $x_i$  is the value of attribute  $A_i$
- ▶ Let there are ‘m’ Classes :  $C_1, C_2, C_3, \dots, C_m$
- ▶ Bayesian classifier predicts  $X$  belongs to Class  $C_i$  iff
  - $P(C_i|X) > P(C_j|X)$  for  $1 \leq j \leq m, j \neq i$
- ▶ Maximum Posteriori Hypothesis
  - $$P(C_i|X) = \frac{P(X|C_i) P(C_i)}{P(X)}$$
  - Maximize  $P(X|C_i) P(C_i)$  as  $P(X)$  is constant

# Naïve Bayes Classifier/ Bayesian Classifier

- With many attributes, it is computationally expensive to evaluate  $P(X|C_i)$
- Naïve Assumption of “class conditional independence”

$$\begin{aligned} P(X | C_i) &= P(x_1, x_2, \dots, x_n | C_i) \\ &= P(x_1 | C_i) * P(x_2 | C_i) * \dots * P(x_n | C_i) \\ &= \prod_{k=1}^n P(x_k | C_i) \end{aligned}$$

# Classification by likelihood

- Suppose we have two classes  $C_1$  and  $C_2$ .
- Compute the likelihoods  $P(D|C_1)$  and  $P(D|C_2)$ .
- To classify test data  $D'$  assign it to class  $C_1$  if  $P(D|C_1)$  is greater than  $P(D|C_2)$  and  $C_2$  otherwise.

# Naïve Bayes

NB Classifier assumes that **all the features are unrelated to each other. Presence or absence of a feature does not influence the presence or absence of any other feature.**

In real dataset, we test a hypothesis given multiple evidence (features). So, calculations become complicated. To simplify the work, the feature independence approach is used to uncouple multiple evidence and treat each as an independent one.

- 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  $i \neq j$

# 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:

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

# Applying conditional independence

- Naïve Bayes assumes  $X_i$  are conditionally independent given  $Y$   
e.g.,  $P(X_1|X_2, Y) = P(X_1|Y)$

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

General form:  $P(X_1, \dots, X_n|Y) = \prod_{j=1}^n P(X_j|Y)$  2<sup>n-1</sup> for Y = 1 and 2<sup>n-1</sup> for Y = 0

How many parameters to describe  $P(X_1, \dots, X_n|Y)$ ?  $P(Y)$ ?

- Without conditional indep assumption?  $2(2^{n-1})+1$
- With conditional indep assumption?  $2n+1$

# 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  $Y$

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

# Naïve Bayes algorithm -discrete X<sub>i</sub>

- For each value  $y_k$

Estimate  $\pi_k = P(Y = y_k)$  ( Prior Prob.)

For each value  $x_{ij}$  of each attribute  $X_i$

Estimate  $\theta_{ijk} = P(X_i = x_{ijk}|Y = y_k)$

- Classify  $X^{\text{test}}$

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

$$\hat{Y} \leftarrow \operatorname{argmax}_{y_k} \pi_k \prod_i \theta_{ijk}$$

## Estimating parameters: discrete $Y, X_i$

- Maximum likelihood estimates (MLE)

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

Where D = Number of items in data set D for which  $Y = y_k$

# Example : Play Tennis

*PlayTennis: training examples*

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: Play Tennis

- Step 1: Convert the data into frequency table

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	2	3
Sunny	3	2
Grand Total	5	9

- Step 2: Create Likelihood table by finding the prior probabilities like probability of playing,  $P(\text{Yes}) = 0.64$ ,  $P(\text{No}) = 0.36$ .

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Sunny	3	2	=5/14	0.36
Rainy	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

# Example : Play Tennis

Step 3: Compute the conditional probability of individual attribute

Outlook	Play=Yes	Play=No
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Temperature	Play=Yes	Play=No
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Humidity	Play=Yes	Play=No
High	3/9	4/5
Normal	6/9	1/5

Wind	Play=Yes	Play=No
Strong	3/9	3/5
Weak	6/9	2/5

# Example: Play Tennis

- Test Phase

- Given a new instance,

$\mathbf{x}' = (\text{Outlook}=\text{Sunny}, \text{Temperature}=\text{Cool}, \text{Humidity}=\text{High}, \text{Wind}=\text{Strong})$

- Look up tables

$$P(\text{Outlook}=\text{Sunny} | \text{Play}=\text{Yes}) = 2/9$$

$$P(\text{Temperature}=\text{Cool} | \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Humidity}=\text{High} | \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Wind}=\text{Strong} | \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Play}=\text{Yes}) = 9/14$$

$$P(\text{Outlook}=\text{Sunny} | \text{Play}=\text{No}) = 3/5$$

$$P(\text{Temperature}=\text{Cool} | \text{Play}=\text{No}) = 1/5$$

$$P(\text{Humidity}=\text{High} | \text{Play}=\text{No}) = 4/5$$

$$P(\text{Wind}=\text{Strong} | \text{Play}=\text{No}) = 3/5$$

$$P(\text{Play}=\text{No}) = 5/14$$

- MAP rule

$$P(\text{Yes} | \mathbf{x}') = [P(\text{Sunny} | \text{Yes})P(\text{Cool} | \text{Yes})P(\text{High} | \text{Yes})P(\text{Strong} | \text{Yes})]P(\text{Play}=\text{Yes}) = 0.0053$$

$$P(\text{No} | \mathbf{x}') = [P(\text{Sunny} | \text{No})P(\text{Cool} | \text{No})P(\text{High} | \text{No})P(\text{Strong} | \text{No})]P(\text{Play}=\text{No}) = 0.0206$$

Given the fact  $P(\text{Yes} | \mathbf{x}') < P(\text{No} | \mathbf{x}')$ , we label  $\mathbf{x}'$  to be "No".

How to classify the new record X = (Refund='Yes', Status = 'Single', Taxable Income =80K)

<i>Tid</i>	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

# Example of Naïve Bayes Classifier

Given a Test Record:  $X = (\text{Refund} = \text{Yes}, \text{Status} = \text{Single}, \text{Income} = 120\text{K})$

naive Bayes Classifier:

$$P(\text{Refund}=\text{Yes} | \text{No}) = 3/7$$

$$P(\text{Refund}=\text{No} | \text{No}) = 4/7$$

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

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

$$P(\text{Marital Status}=\text{Single} | \text{No}) = 2/7$$

$$P(\text{Marital Status}=\text{Divorced} | \text{No}) = 1/7$$

$$P(\text{Marital Status}=\text{Married} | \text{No}) = 4/7$$

$$P(\text{Marital Status}=\text{Single} | \text{Yes}) = 2/7$$

$$P(\text{Marital Status}=\text{Divorced} | \text{Yes}) = 1/7$$

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

For taxable income:

If class=No: sample mean=110  
sample variance=2975

If class=Yes: sample mean=90  
sample variance=25

$$\begin{aligned} \square P(X | \text{Class}=\text{No}) &= P(\text{Refund}=\text{Yes} | \text{Class}=\text{No}) \times P(\text{Married}=\text{Single} | \text{Class}=\text{No}) \\ &\quad \times P(\text{Income}=120\text{K} | \text{Class}=\text{No}) \\ &= 3/7 * 2/7 * 0.0062 = 0.00075 \end{aligned}$$

$$\begin{aligned} \square P(X | \text{Class}=\text{Yes}) &= P(\text{Refund}=\text{No} | \text{Class}=\text{Yes}) \times P(\text{Married}=\text{Single} | \text{Class}=\text{Yes}) \\ &\quad \times P(\text{Income}=120\text{K} | \text{Class}=\text{Yes}) \\ &= 0 * 2/3 * 0.01 = 0 \end{aligned}$$

- $P(\text{No}) = 0.3, P(\text{Yes}) = 0.7$

Since  $P(X | \text{No})P(\text{No}) > P(X | \text{Yes})P(\text{Yes})$  Therefore  $P(\text{No} | X) > P(\text{Yes} | X)$   
 $\Rightarrow \text{Class} = \text{No}$

- Normal distribution:

$$P(A_i | c_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(A_i - \mu_j)^2}{2\sigma_{ij}^2}}$$

- One for each  $(A_i, c_j)$  pair

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

# Naïve Bayes: Subtlety #1

- Often the  $X_i$  are not really conditionally independent
- Naïve Bayes often works pretty well anyway
  - Often the right classification, even when not the right probability [Domingos & Pazzani, 1996]
- What is the effect on estimated  $P(Y|X)$ ?
  - What if we have two copies:  $X_i = X_k$

$$P(Y = y_k | X_1, \dots, X_n) \propto P(Y = y_k) \prod_i P(X_i | Y = y_k)$$

# Naïve Bayes: Subtlety #2

MLE estimate for  $P(X_i|Y = y_k)$  might be zero.

(for example,  $X_i$  = birthdate.  $X_i$  = Feb\_4\_1995)

- Why worry about just one parameter out of many?

$$P(Y = y_k | X_1, \dots, X_n) \propto P(Y = y_k) \prod_i P(X_i | Y = y_k)$$

- What can we do to address this?

- MAP estimates (adding “imaginary” examples)

# Estimating parameters: discrete $Y, X_i$

- Maximum likelihood estimates (MLE)

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

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

- MAP estimates (Dirichlet priors):

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\} + (\beta_k - 1)}{|D| + \sum_m (\beta_m - 1)}$$

$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} | Y = y_k) = \frac{\#D\{X_i = x_{ij}, Y = y_k\} + (\beta_k - 1)}{\#D\{Y = y_k\} + \sum_m (\beta_m - 1)}$$

# Things to remember

- Probability basics
- Estimating parameters from data
  - Maximum likelihood (ML)  $\text{maximize } P(\text{Data}|\theta)$
  - Maximum a posteriori estimation (MAP)  $\text{maximize } P(\theta|\text{Data})$
- Naive Bayes
$$P(Y = y_k | X_1, \dots, X_n) \propto P(Y = y_k) \prod_i P(X_i | Y = y_k)$$

# Bayesian networks

- A simple, graphical model for depicting probabilistic relationships among a set of variables. It encodes conditional independence relationships between the variables in the graph structure.
- Provides a compact representation of Joint probability distribution over the variables.
- A problem domain is modeled by a list of variables  $X_1, \dots, X_n$ .
- Knowledge about the problem domain is represented by a joint probability  $P(X_1, \dots, X_n)$
- Directed links represent causal direct influences. Each node has a conditional probability table quantifying the effects from the parents.
- No directed cycles

# Bayesian networks

- A Bayesian network specifies a joint distribution in a structured form
- Represent dependence/independence via a directed graph
  - Nodes = random variables
  - Edges = direct dependence
- Structure of the graph  $\Leftrightarrow$  Conditional independence relations  
In general,

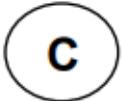
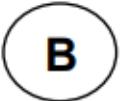
$$p(X_1, X_2, \dots, X_N) = \prod p(X_i | \text{parents}(X_i))$$

The full joint distribution

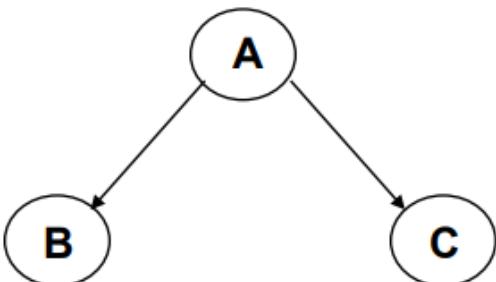
The graph-structured approximation

- Requires that graph is acyclic (no directed cycles)
- 2 components to a Bayesian network
  - The graph structure (conditional independence assumptions)
  - The numerical probabilities (for each variable given its parents)

# Terminology



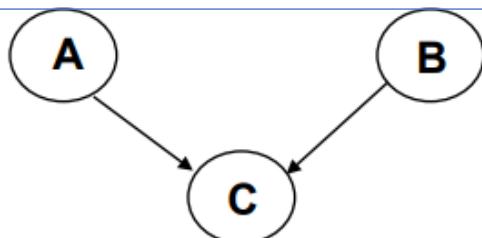
**Marginal Independence:**  
 $p(A,B,C) = p(A) p(B) p(C)$



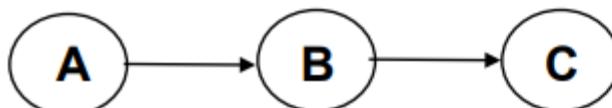
**Conditionally independent effects:**  
 $p(A,B,C) = p(B|A)p(C|A)p(A)$

**B and C are conditionally independent  
Given A**

e.g., A is a disease, and we model  
B and C as conditionally independent  
symptoms given A



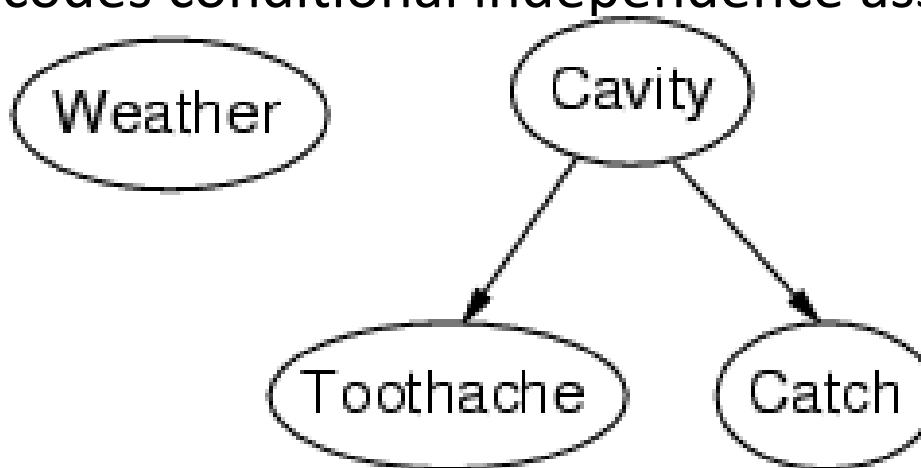
**Independent Causes:**  
 $p(A,B,C) = p(C|A,B)p(A)p(B)$



**Markov dependence:**  
 $p(A,B,C) = p(C|B) p(B|A)p(A)$

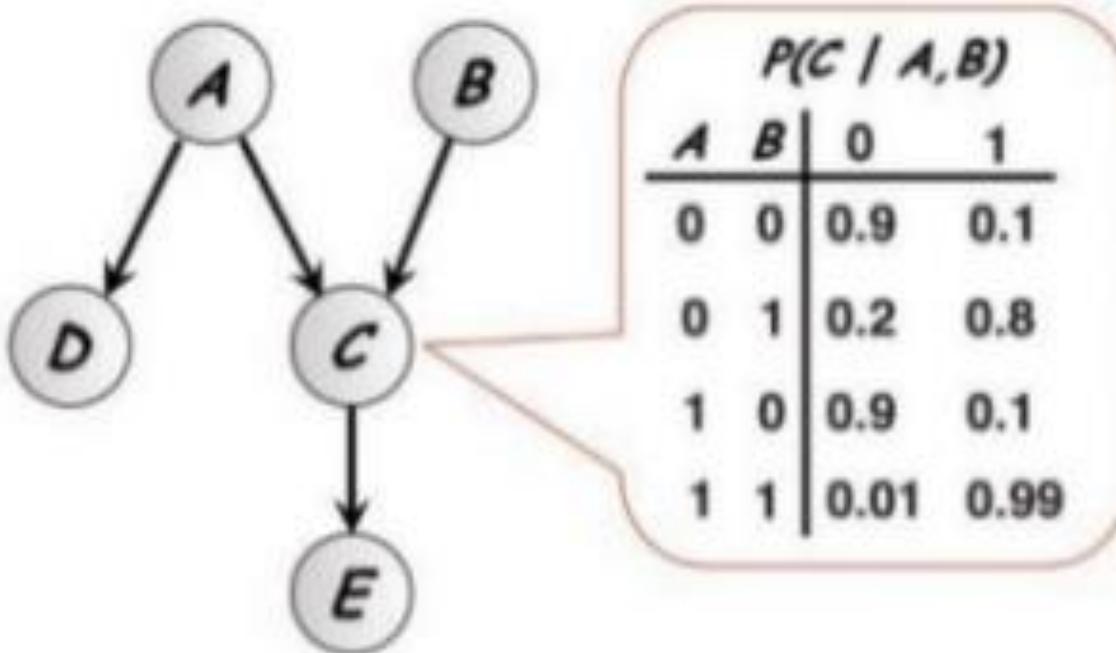
# Example

- Topology of network encodes conditional independence assertions:



- *Weather* is independent of the other variables
- *Toothache* and *Catch* are conditionally independent given *Cavity*

# Example



- Directed edges => direct dependence
- Absence of an edge => conditional independence

$$P(A, B, C, D, E) = P(A)P(B)P(C | A, B)P(D | A)P(E | C)$$

# Why Bayesian networks?

Why do we need graphs?

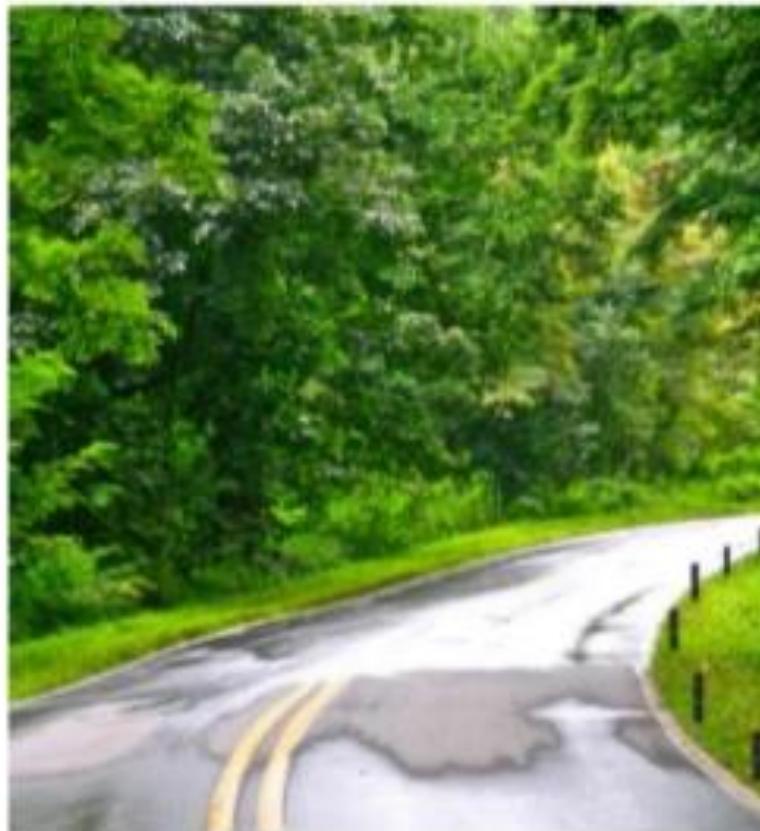


Figure: Motivating Example

# Why Bayesian networks?



Figure: Motivating Example

Variables in the study:

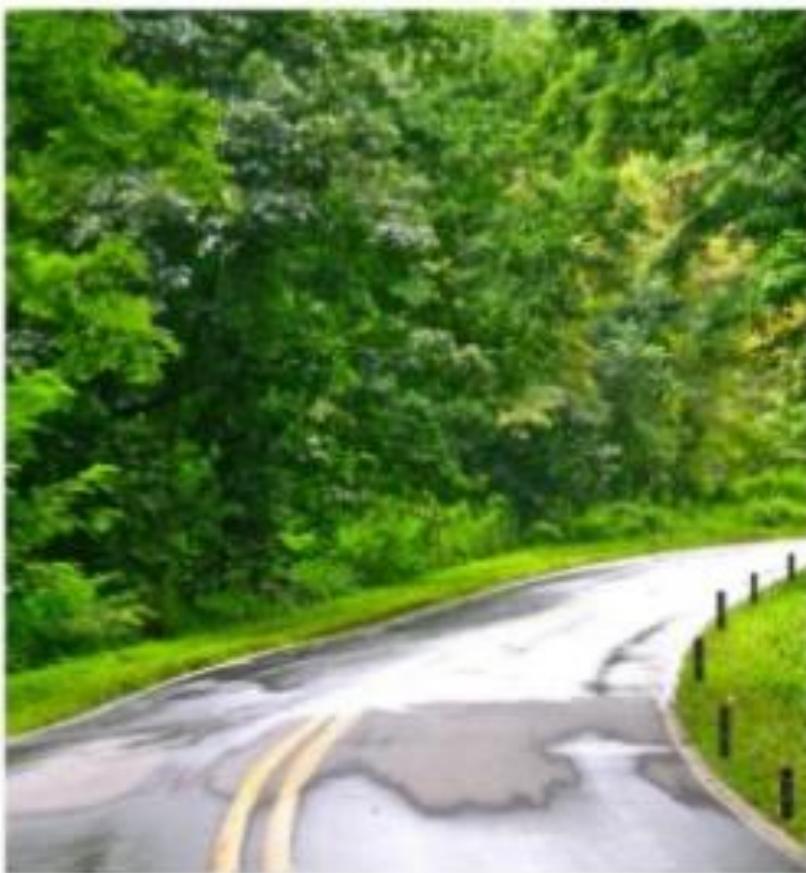
- ▶ Season
- ▶ Sprinkler
- ▶ Rain
- ▶ Wetness of pavement(Wet)
- ▶ Slipperiness of pavement(Slippery)

Assuming binary states for all the variables

Ex) Season: dry or rainy

Ex) Sprinkler: ON or OFF

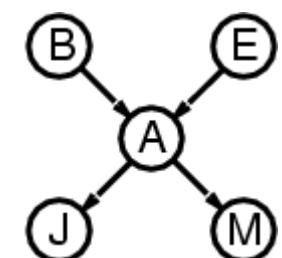
# Why Bayesian networks?



# Variables	Table size
5	32
6	64
7	128
8	256
9	512
10	1,024
20	1,048,576
30	1,073,741,824

# Number of Probability in BN/ Compactness

- Consider  $n$  binary variables
- Unconstrained joint distribution requires  $O(2^n)$  probabilities
- If we have a Bayesian network, with a maximum of  $k$  parents for any node, then we need  $O(n \cdot 2^k)$  probabilities
- Example
  - Full unconstrained joint distribution
    - $n = 30$ : need  $10^9$  probabilities for full joint distribution
  - Bayesian network
    - $n = 30, k = 4$ : need 480 probabilities
- For burglary net,  $1 + 1 + 4 + 2 + 2 = 10$  numbers (vs.  $2^5 - 1 = 31$ )



# Example

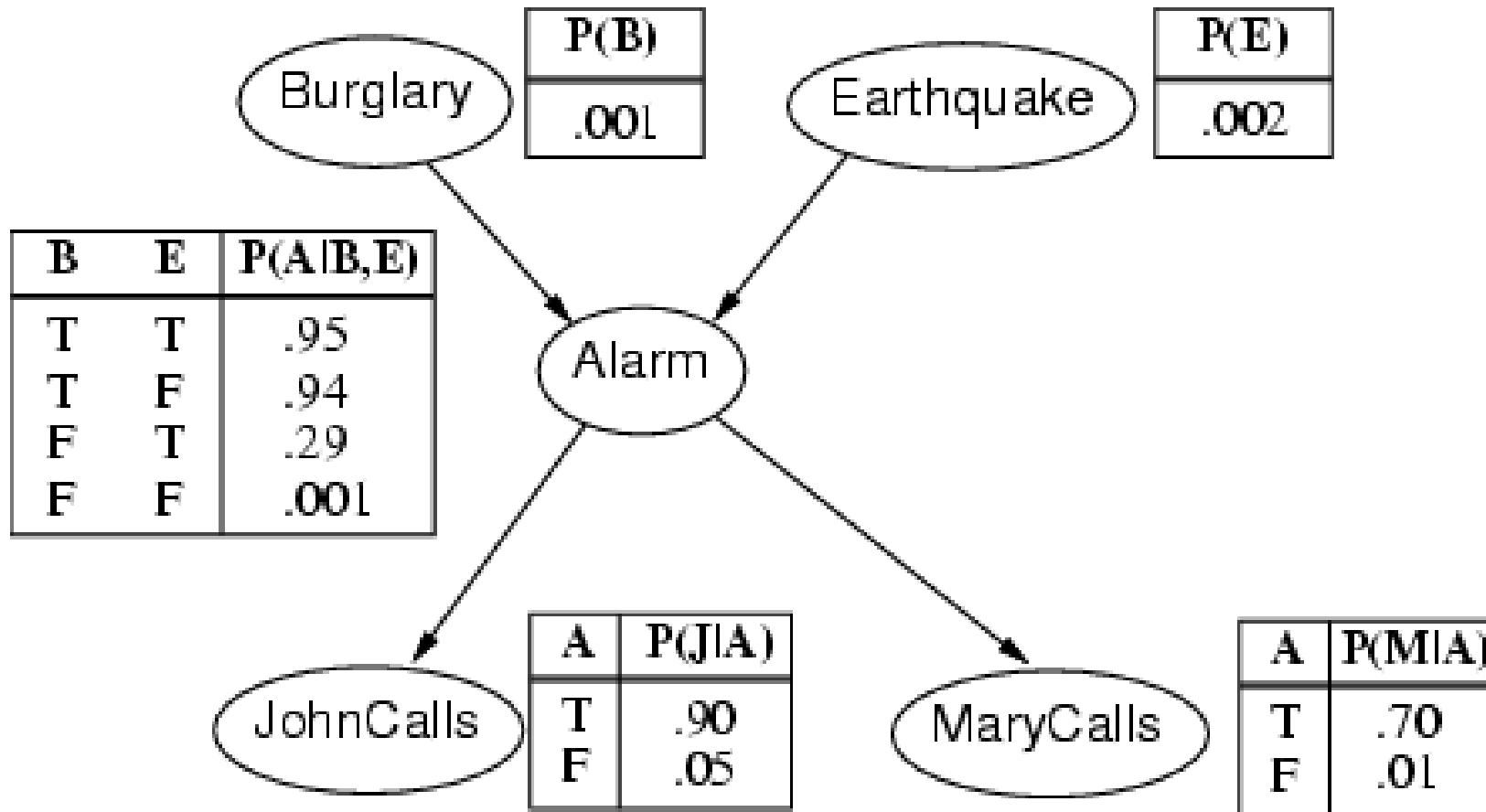
- You have a new burglar alarm installed at home.
- It is fairly reliable at detecting burglary, but also sometimes responds to minor earthquakes.
- You have two neighbors, John and Merry , who promised to call you at work when they hear the alarm.
- John always calls when he hears the alarm, but sometimes confuses telephone ringing with the alarm and calls too.
- Merry likes loud music and sometimes misses the alarm.
- Given the evidence of who has or has not called, we would like to estimate the probability of a burglary.

# Example

Consider the following 5 binary variables:

- $B$  = a burglary occurs at your house
- $E$  = an earthquake occurs at your house
- $A$  = the alarm goes off
- $J$  = John calls to report the alarm
- $M$  = Mary calls to report the alarm
  
- What is  $P(B | M, J)$  ? (for example)
  
- We can use the full joint distribution to answer this question
  - Requires  $2^5 = 32$  probabilities
  - Can we use prior domain knowledge to come up with a Bayesian network that requires fewer probabilities?

## Example contd.

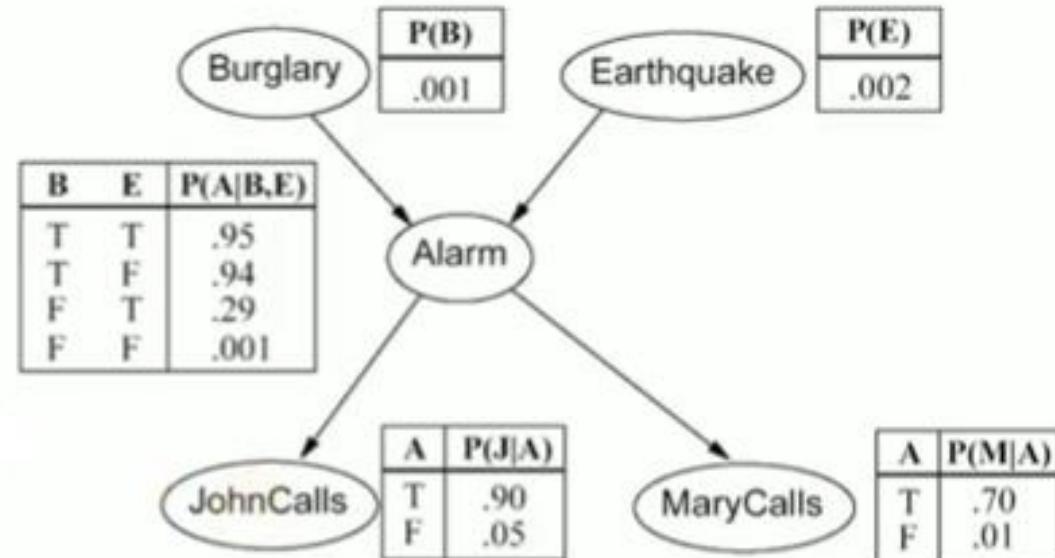


## Example contd.

- What is the probability that the alarm has sounded but neither a burglary nor an earthquake has occurred, and both John and Merry call?

**Solution:**

$$\begin{aligned} P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) &= P(j | a) P(m | a) P(a | \neg b, \neg e) P(\neg b) P(\neg e) \\ &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 \\ &= 0.00062 \end{aligned}$$



## Example contd.

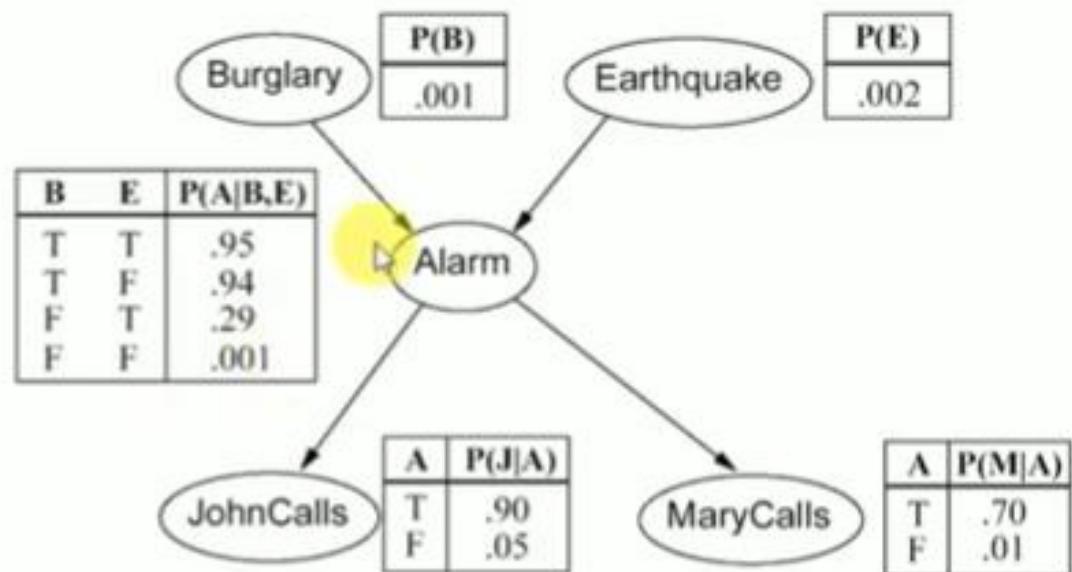
2. What is the probability that John call?

Solution:

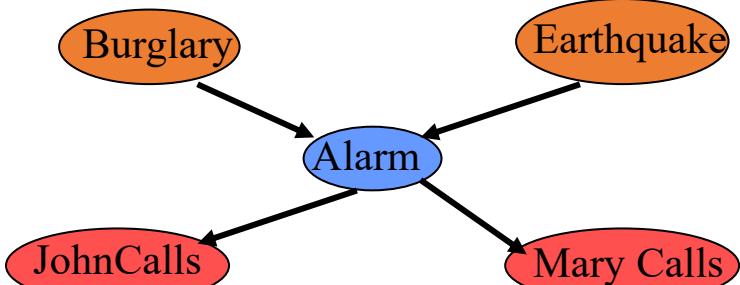
$$P(j) = P(j | a) P(a) + P(j | \neg a) P(\neg a)$$

$$\begin{aligned} &= P(j | a) \{ P(a | b, e) * P(b, e) + P(a | \neg b, e) * P(\neg b, e) + P(a | b, \neg e) * P(b, \neg e) + P(a | \neg b, \neg e) * P(\neg b, \neg e) \} \\ &+ P(j | \neg a) \{ P(\neg a | b, e) * P(b, e) + P(\neg a | \neg b, e) * P(\neg b, e) + P(\neg a | b, \neg e) * P(b, \neg e) + P(\neg a | \neg b, \neg e) * \\ &\quad P(\neg b, \neg e) \} \end{aligned}$$

$$= 0.90 * 0.00252 + 0.05 * 0.9974 = 0.0521$$



# Belief Network Example (cont.)



P(B)
.001

P(E)
.002

A	P(J)
T	.90
F	.05

A	P(M)
T	.70
F	.01

B	E	P(A)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

3. Probability of false notification?

Alarm sounded and both people call, but there was no burglary or earthquake

$$P(J \wedge M \wedge A \wedge \sim B \wedge \sim E)$$

$$P(J | A)P(M | A)P(A | \sim B \wedge \sim E)P(\sim B)P(\sim E)$$

$$.9 * .7 * .001 * .999 * .998 = .00062$$

# Bayesian belief networks (BBNs)

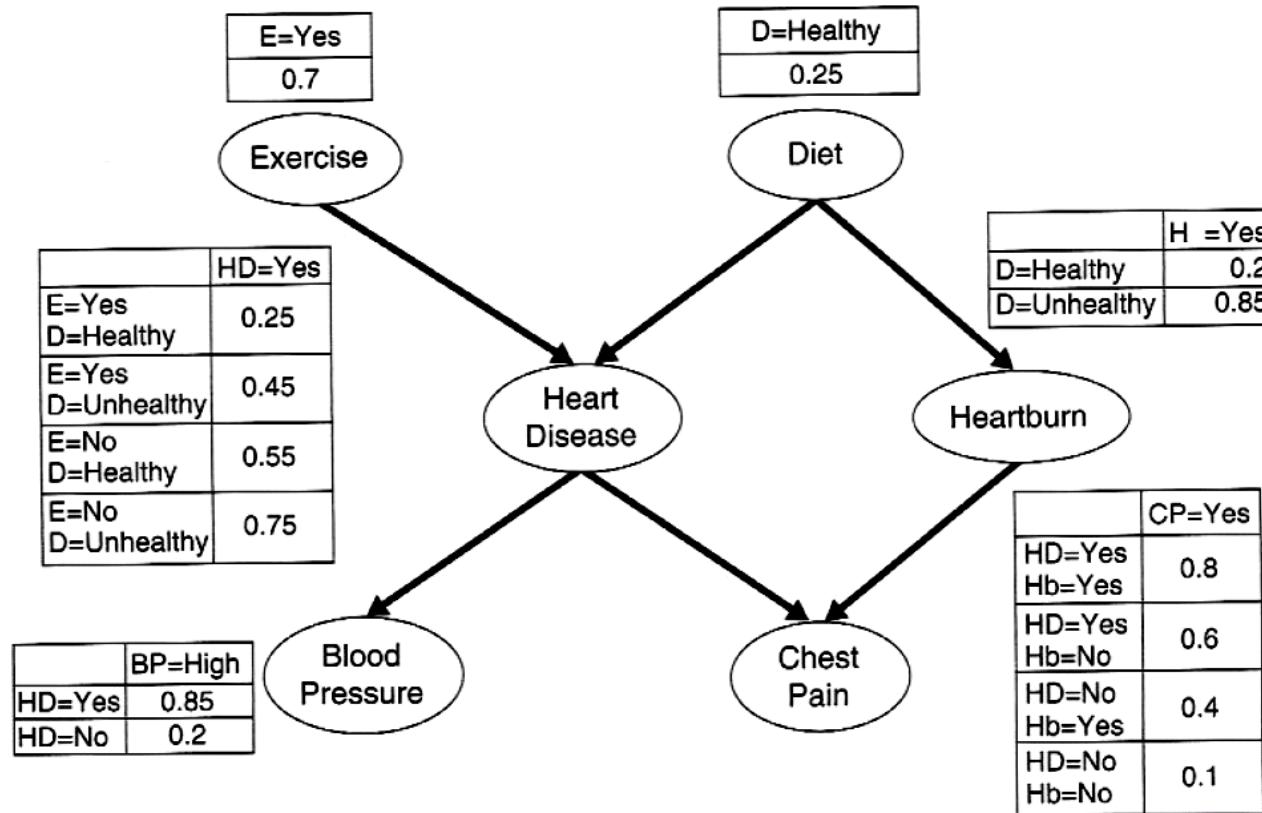
## Bayesian belief networks

- Represent the full joint distribution over the variables more compactly using the product of local conditionals.
- **But how did we get to local parameterizations?**

## Answer:

- **Graphical structure encodes conditional and marginal independences** among random variables
- **A and B are independent**  $P(A, B) = P(A)P(B)$
- **A and B are conditionally independent given C**
$$P(A | C, B) = P(A | C)$$
$$P(A, B | C) = P(A | C)P(B | C)$$
- **The graph structure implies the decomposition !!!**

# Another example



Once the right topology has been found, the probability table associated with each node is determined. Estimating such probabilities is fairly straightforward and is similar to the approach used by naive Bayes classifiers.

# High Blood Pressure

- Suppose we get to know that the new patient has high blood pressure.
- What's the probability he has heart disease under this condition?

# BBNs built in practice

- **In various areas:**
  - Intelligent user interfaces (Microsoft)
  - Troubleshooting, diagnosis of a technical device
  - Medical diagnosis:
    - Pathfinder (Intellipath)
    - CPSC
    - Munin
    - QMR-DT
  - Collaborative filtering
  - Military applications
  - Insurance, credit applications

# Summary

- Bayesian networks provide a natural representation for (causally induced) conditional independence
- Topology + CPTs = compact representation of joint distribution
- Generally easy for domain experts to construct

# Expectation Maximization

- EM algorithm provides a general approach to learning in presence of unobserved variables.
- In many practical learning settings, only a subset of relevant features or variables might be observable. – Eg: Hidden Markov, Bayesian Belief Networks
- Estimation: Estimate the expectation from some random data
- Maximization: Whatever is estimated should be maximized to find the best result.
- From given data EM learn a theory which tells that how each example to be classified and how to predict the feature value of each class.

# Expectation Maximization

Suppose you have 2 coins, A and B, each with a certain bias of landing heads,  $\theta_A, \theta_B$ .

Given data sets  $X_A = \{x_{1,A}, \dots, x_{m_A,A}\}$  and  $X_B = \{x_{1,B}, \dots, x_{m_B,B}\}$

Where  $x_{i,j} = \begin{cases} 1 & ; if \text{ heads} \\ 0 & ; otherwise \end{cases}$

No hidden variables – easy solution.  $\theta_j = \frac{1}{m_j} \sum_{i=1}^{m_j} x_{i,j}$  ; sample mean

# Example

- Assume that we have two coins, C1 and C2
- Assume the bias of C1 is  $\theta_1$   
(i.e., probability of getting heads with C1)
- Assume the bias of C2 is  $\theta_2$   
(i.e., probability of getting heads with C2)
- We want to find  $\theta_1$ ,  $\theta_2$  by performing a number of trials  
(i.e., coin tosses)

# Example

First experiment

- We choose 5 times one of the coins.
- We toss the chosen coin 10 times

	H T T T H H T H T H
	H H H H T H H H H H
	H T H H H H H T H H
	H T H T T T H H T T
	T H H H T H H H T H

$$\theta_1 = \frac{\text{number of heads using } C1}{\text{total number of flips using } C1}$$

$$\theta_2 = \frac{\text{number of heads using } C2}{\text{total number of flips using } C2}$$

# Example



H T T T H H T H T H

H H H H T H H H H H

H T H H H H H T H H

H T H T T T H H T T

T H H H T H H H T H

Coin A	Coin B
	5 H, 5 T
9 H, 1 T	
8 H, 2 T	
	4 H, 6 T
7 H, 3 T	
24 H, 6 T	9 H, 11 T

$$\theta_1 = \frac{24}{24 + 6} = 0.8$$

$$\theta_2 = \frac{9}{9 + 11} = 0.45$$

# Example with Hidden Variable

- What if you were given the same dataset of coin flip results, but no coin identities defining the datasets?

Here:  $X = \{x_1, \dots, x_m\}$ ; the observed variable

$$Z = \begin{Bmatrix} z_{1,1} & \dots & z_{m,1} \\ \dots & z_{i,j} & \dots \\ z_{1,k} & \dots & z_{m,k} \end{Bmatrix} \text{ where } z_{i,j} = \begin{cases} 1 & ; \text{if } x_i \text{ is from } j^{\text{th}} \text{ coin} \\ 0 & ; \text{otherwise} \end{cases}$$

But Z is not known. (Ie: 'hidden' / 'latent' variable)

# Example with Hidden Variable

Assume a more challenging problem

H T T T H H T H T H

H H H H T H H H H H

H T H H H H T H T H

H T H T T T H H T T

T H H H T H H H T H

- We do not know the identities of the coins used for each set of tosses (we treat them as hidden variables).

# Example with Hidden Variable

- 0) Initialize some arbitrary hypothesis of parameter values ( $\theta$ ):

$$\theta = \{ \theta_1, \dots, \theta_k \} \quad \text{coin flip example: } \theta = \{\theta_A, \theta_B\} = \{0.6, 0.5\}$$

- 1) Expectation (E-step)

$$E[z_{i,j}] = \frac{p(x = x_i | \theta = \theta_j)}{\sum_{n=1}^k p(x = x_i | \theta = \theta_n)}$$

If  $z_{i,j}$  is known:

- 2) Maximization (M-step)

$$\theta_j = \frac{\sum_{i=1}^m E[z_{i,j}] x_i}{\sum_{i=1}^m E[z_{i,j}]}$$

$$\theta_j = \frac{\sum_{i=1}^{m_j} x_i}{m_j}$$

# Example with Hidden Variable

$\theta = P(\text{up}), \ 1-\theta = P(\text{down})$

Observe:

Likelihood of the observation sequence depends on  $\theta$ :

$$\begin{aligned} l(\theta) &= \theta(1 - \theta)\theta(1 - \theta)\theta\theta\theta\theta\theta\theta\theta\theta \\ &= \theta^8(1 - \theta)^2 \end{aligned}$$



# Example with Hidden Variable

$$L(C) = \Theta^k (1 - \Theta)^{n-k}$$

Likelihood For first coin Flips

$$L(A) = 0.6^5 (1 - 0.6)^{10-5} = 0.0007963$$

$$L(B) = 0.5^5 (1 - 0.5)^{10-5} = 0.0009766$$

$$P(A) = L(A)/[L(A)+L(B)] = 0.0007963/(0.0007963+0.0009766) = 0.45$$

$$P(B) = L(B)/[L(A)+L(B)] = 0.0009766/(0.0007963+0.0009766) = 0.55$$

**Estimate Likely No of Heads and Tails for First Toss**

**For A:** H = 0.45\*5 = 2.2, T = 0.45\*5 = 2.2

**For B:** H = 0.55\*5 = 2.8, T = 0.55\*5 = 2.8

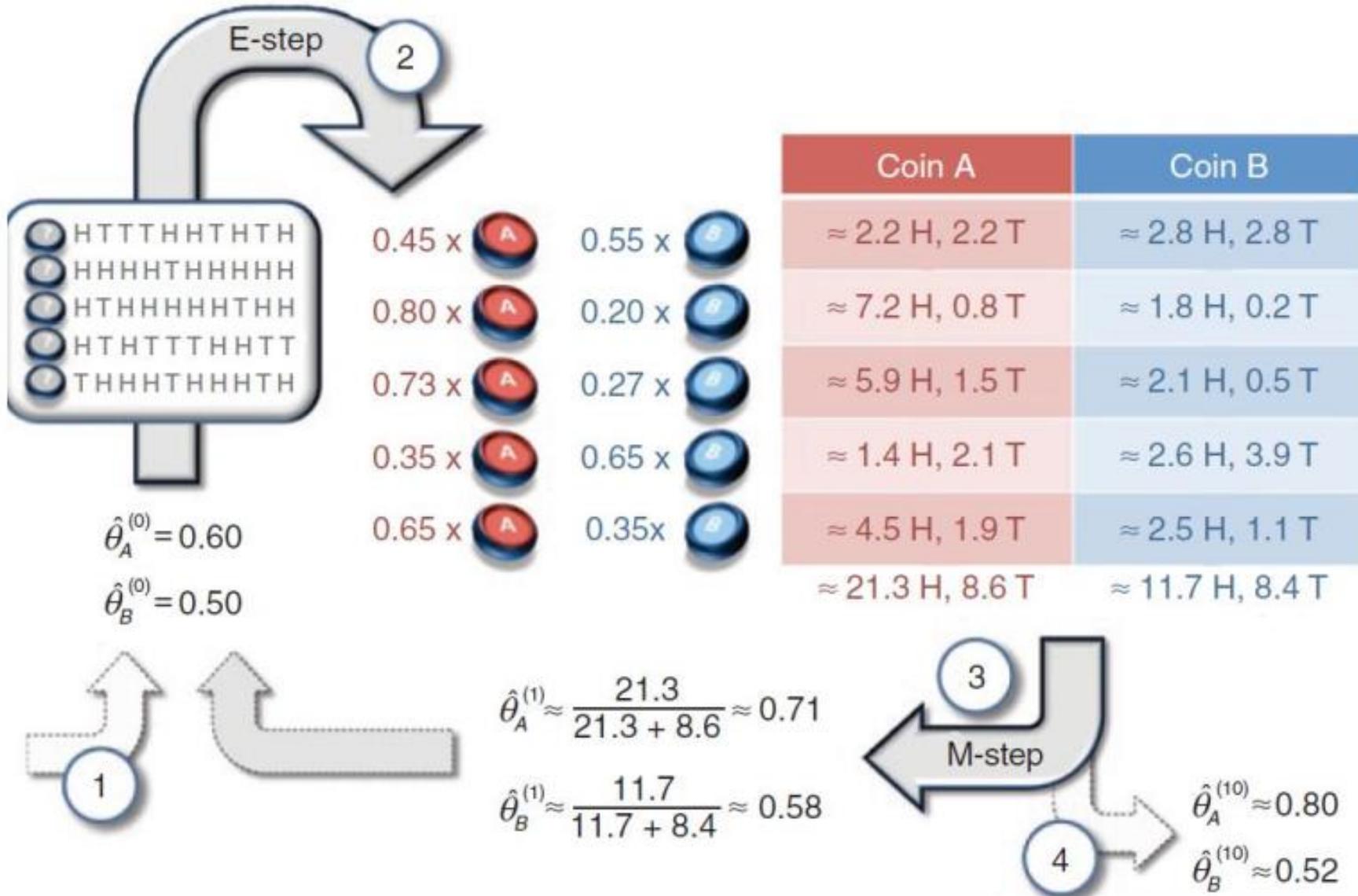
# Example with Hidden Variable

In similar fashion find probability of all coins with all flips. It will be as follows:

$L(H)$ : Likely no of heads       $L(T)$ : Likely no of tails

	Iteration 1->											Coin A		Coin B		
												P(A)	P(B)	L(H)	L(T)	L(H)
B	H	T	T	T	H	H	T	H	T	H	0.45	0.55	2.2	2.2	2.8	2.8
A	H	H	H	H	T	H	H	H	H	H	0.80	0.20	7.2	0.8	1.8	0.2
A	H	T	H	H	H	H	H	T	H	H	0.73	0.27	5.9	1.5	2.1	0.5
B	H	T	H	T	T	T	H	H	T	T	0.35	0.65	1.4	2.1	2.6	3.9
A	T	H	MAT	C	H	H	T	H	H	H	0.65	0.35	4.5	1.9	2.5	1.1

# Example with Hidden Variable



# Expectation Maximization

1. Choose starting parameters
2. Estimate probability using these parameters that each data set ( $x_i$ ) came from  $j^{th}$  coin ( $E[z_{i,j}]$ )
3. Use these probability values ( $E[z_{i,j}]$ ) as weights on each data point when computing a new  $\theta_j$  to describe each distribution
4. Summate these expected values, use maximum likelihood estimation to derive new parameter values to repeat process

**Course code:** CSE3008

**Course Title:** Introduction to Machine Learning

**Module 5: Artificial Neural Network**

**Dr. Usha Rani Gogoi**

**Assistant Professor Sr. Grade 1**

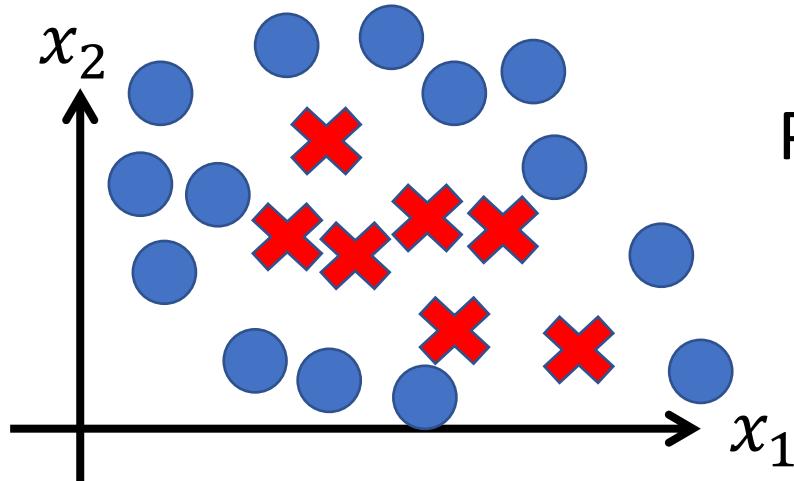
**[usha.gogoi@vitap.ac.in](mailto:usha.gogoi@vitap.ac.in)**

**SCOPE**

# Neural Networks

- Why neural networks?
- Model representation
- Examples and intuitions
- Multi-class classification
- Back propagation algorithm
- Implementation & Applications

# Non-linear classification



$x_1$  = size

$x_2$  = #bedrooms

$x_3$  = #floors

$x_4$  = age

...

$x_{100}$

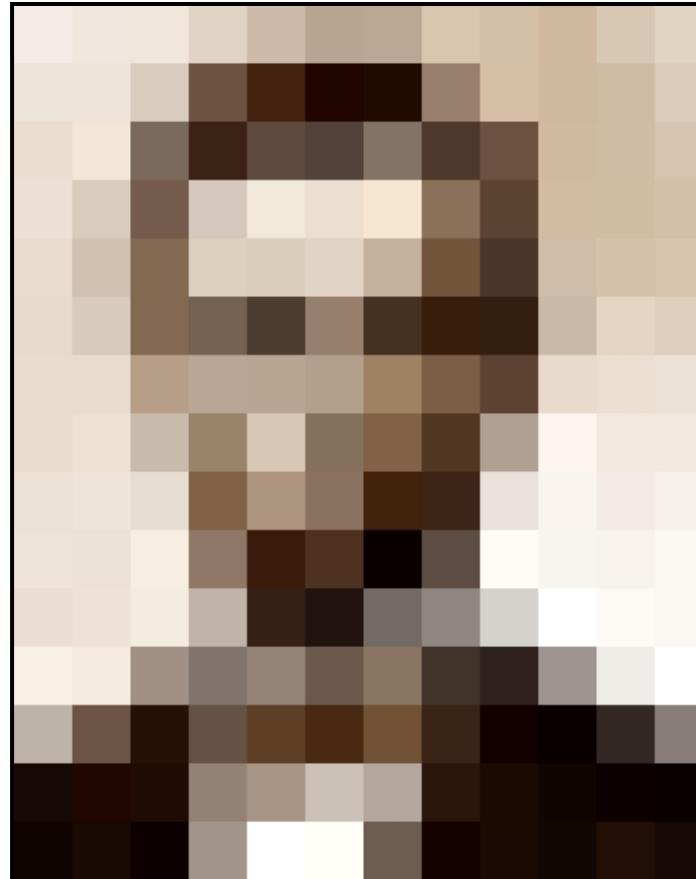
Predict  $y = 1$  if

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \dots) \geq 0$$

# Quadratic features?

# Cubic features?

# What humans see



# What computers see

243	239	240	225	206	185	188	218	211	206	216	225
242	239	218	110	67	31	34	152	213	206	208	221
243	242	123	58	94	82	132	77	108	208	208	215
235	217	115	212	243	236	247	139	91	209	208	211
233	208	131	222	219	226	196	114	74	208	213	214
232	217	131	116	77	150	69	56	52	201	228	223
232	232	182	186	184	179	159	123	93	232	235	235
232	236	201	154	216	133	129	81	175	252	241	240
235	238	230	128	172	138	65	63	234	249	241	245
237	236	247	143	59	78	10	94	255	248	247	251
234	237	245	193	55	33	115	144	213	255	253	251
248	245	161	128	149	109	138	65	47	156	239	255
190	107	39	102	94	73	114	58	17	7	51	137
23	32	33	148	168	203	179	43	27	17	12	1
17	26	17	160	255	255	109	72	26	19	35	24

# Computer Vision: Car detection



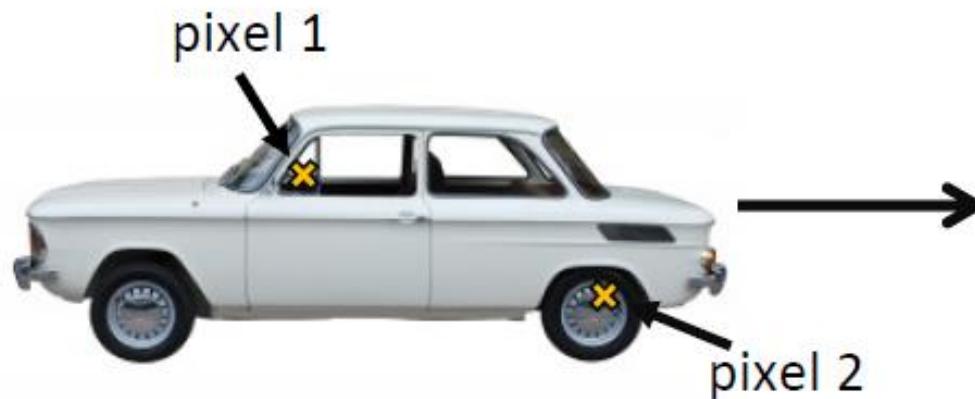
Cars



Not a car

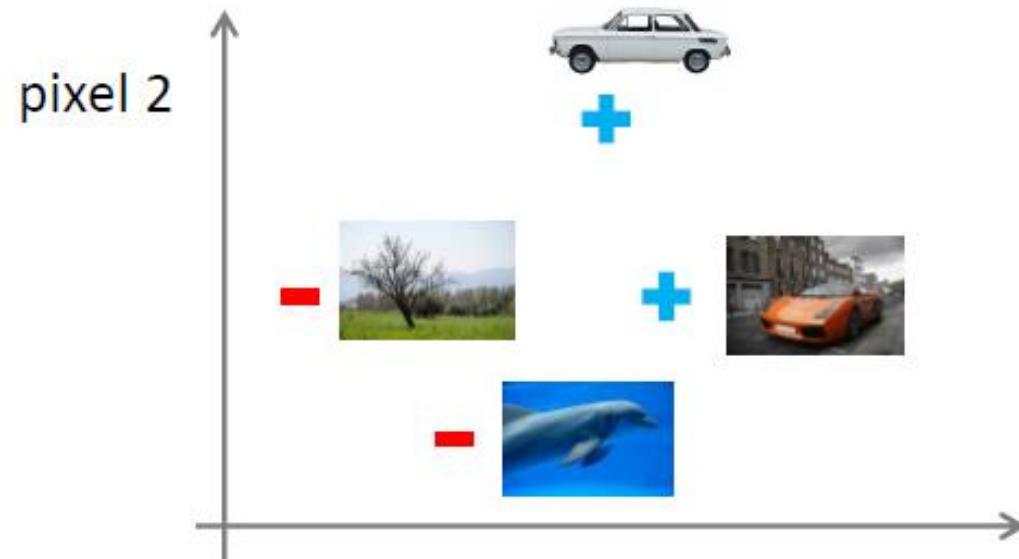
Testing:





Learning  
Algorithm

50x50 pixel images -> 2500 pixels



- + Cars
- "Non"-Cars

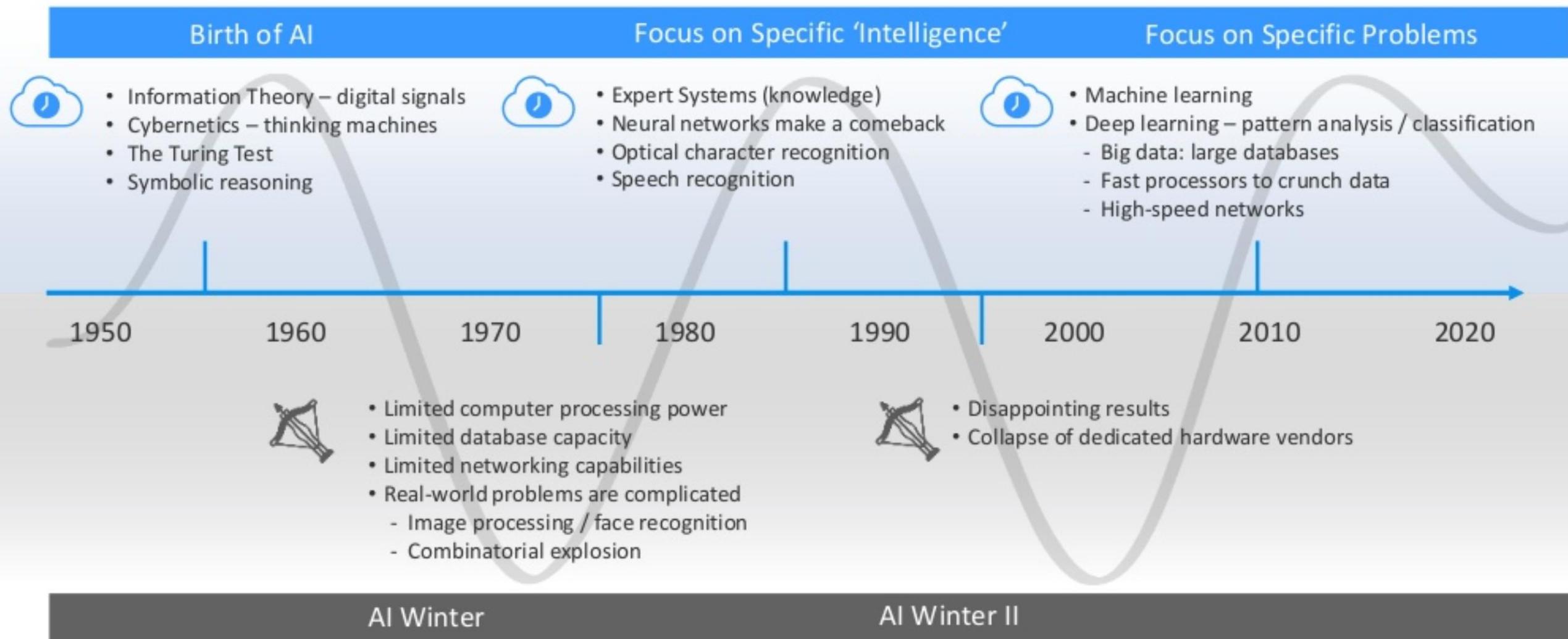
$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

Quadratic features  $(x_i x_j) \sim$   
3 million features

# Neural Networks

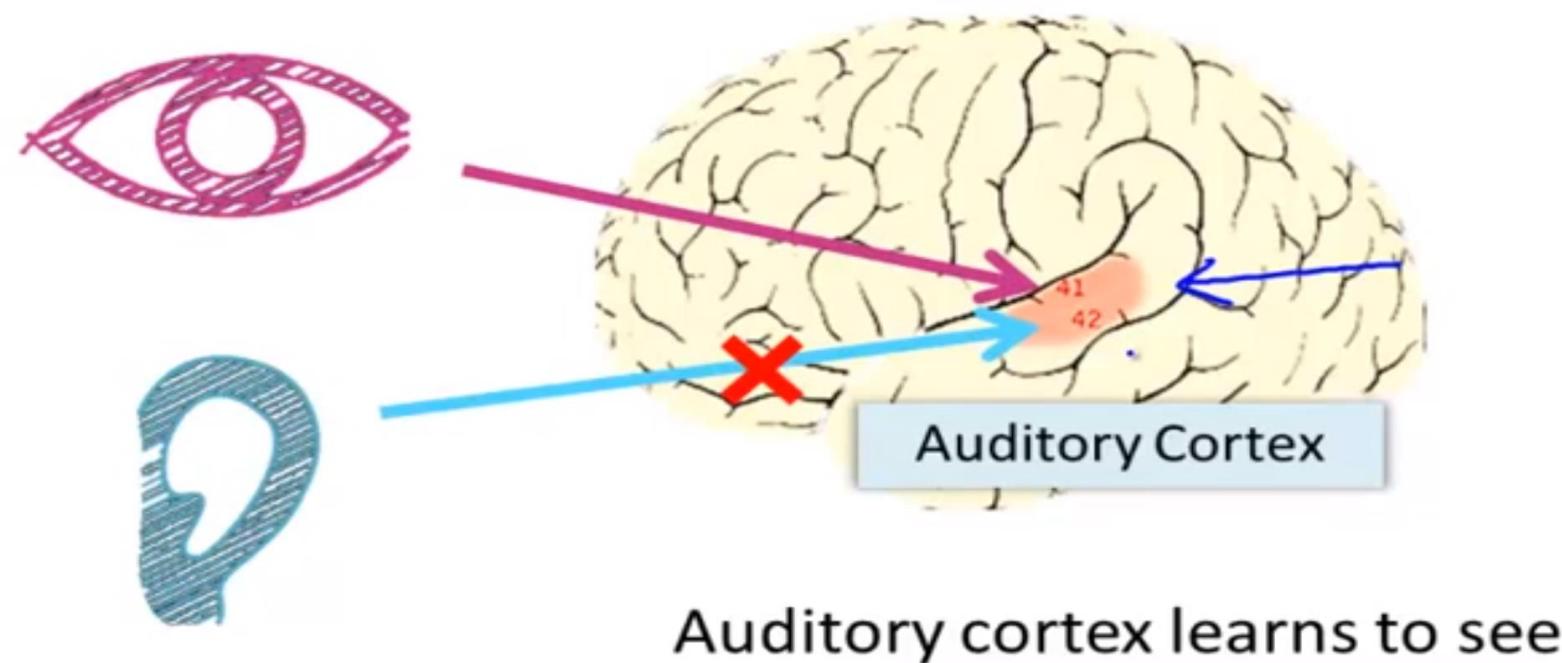
- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications

# An AI Timeline



# The “one learning algorithm” hypothesis

- Auditory cortex --> takes sound signals If you cut the wiring from the ear to the auditory cortex
- Re-route optic nerve to the auditory cortex
- Auditory cortex learns to see

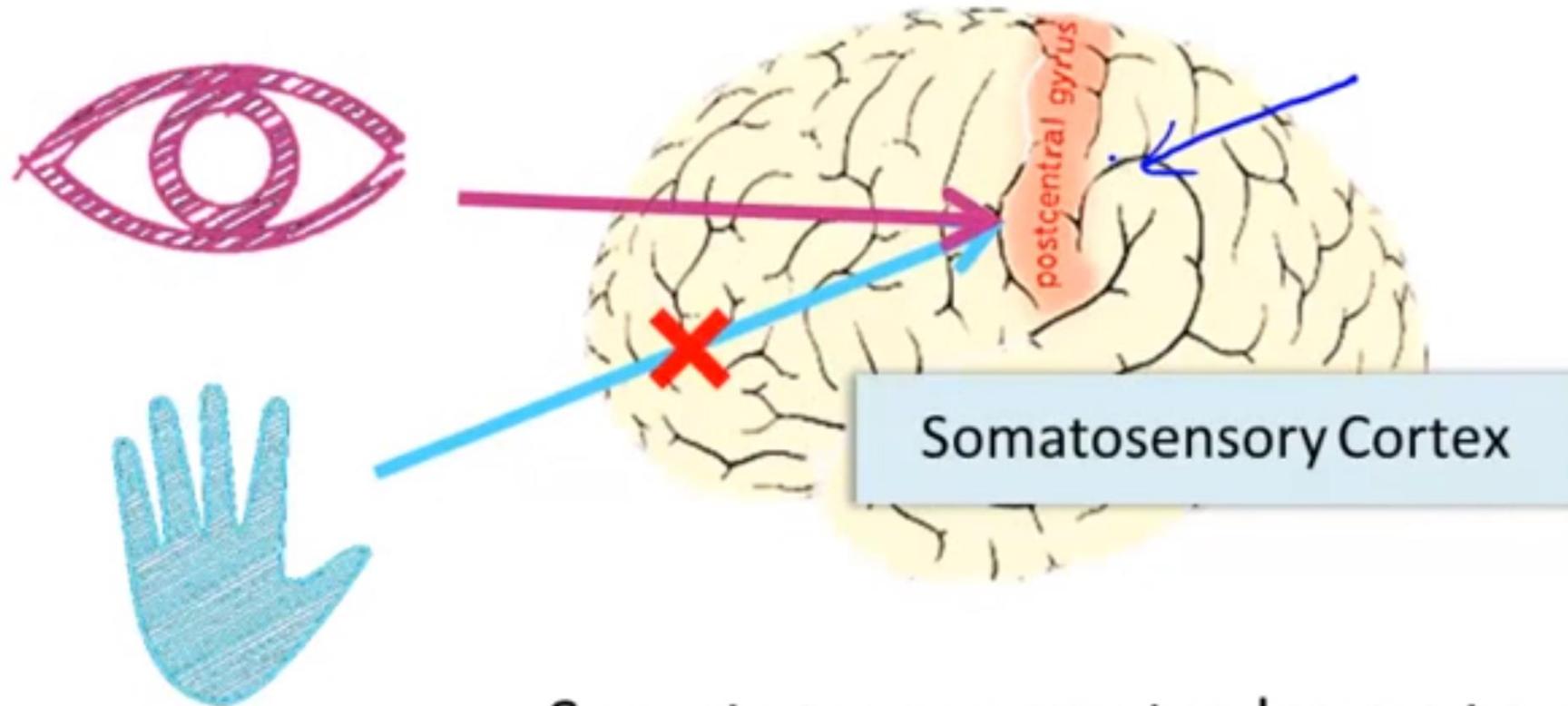


# Auditory cortex

- The auditory cortex plays a critical role in our ability to perceive sound. It is thought to be integral to our perception of the fundamental aspects of an auditory stimulus, like the pitch of the sound. It is also important in other aspects of sound processing, like determining where in space a sound originates from as well as identifying what might be producing the sound.
- The research involved "rewiring" brains in very young mammals, so that inputs from the eye were directed to brain structures that normally process hearing. The animal's auditory cortex successfully interpreted input from its eyes. But it didn't do the job as well as the primary visual cortex would have.

# The “one learning algorithm” hypothesis

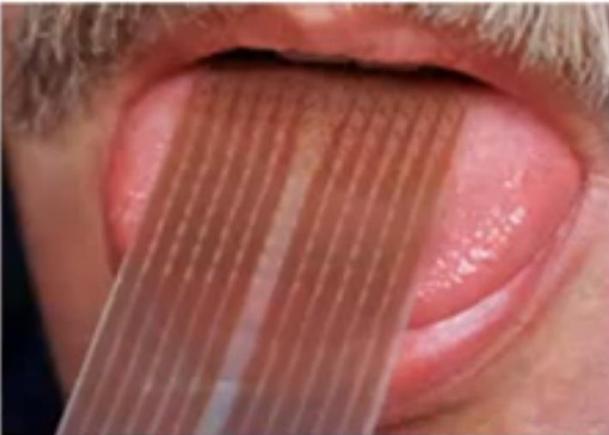
- Somatosensory context (touch processing)
- If you rewrite optic nerve to somatosensory cortex then it learns to see



# The “one learning algorithm” hypothesis

- The somatosensory cortex is a part of your brain that receives and processes sensory information from the entire body.
- This part of the brain is essential for receiving sensory information from the body and processing it to initiate important movements that are required to deal a particular situation. It receives sensations of touch, pain, and vibration from the entire body.

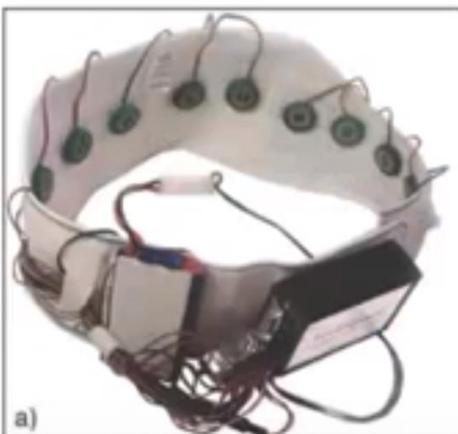
# Sensor representations in the brain



Seeing with your tongue



Human echolocation (sonar)



Haptic belt: Direction sense

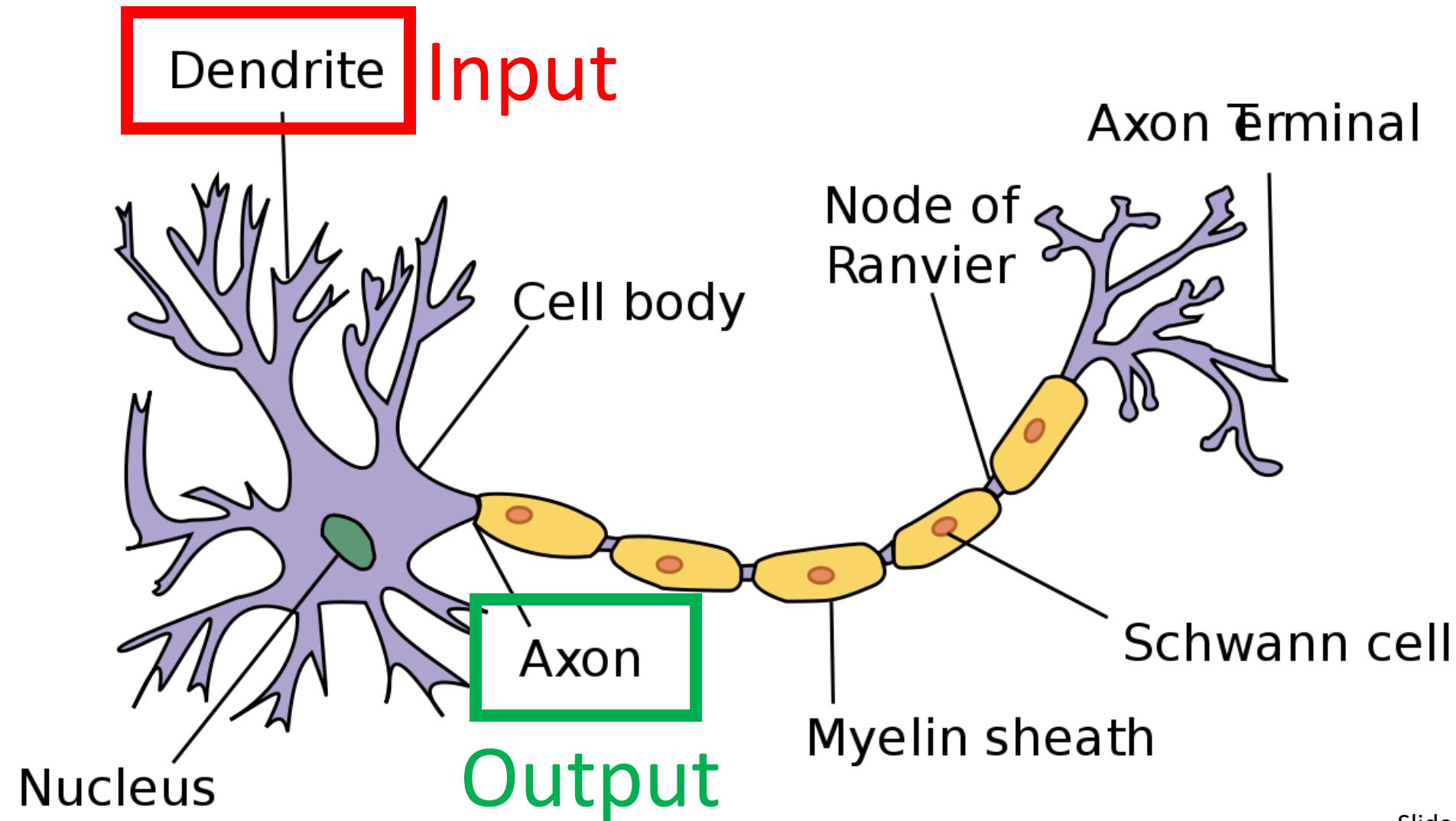


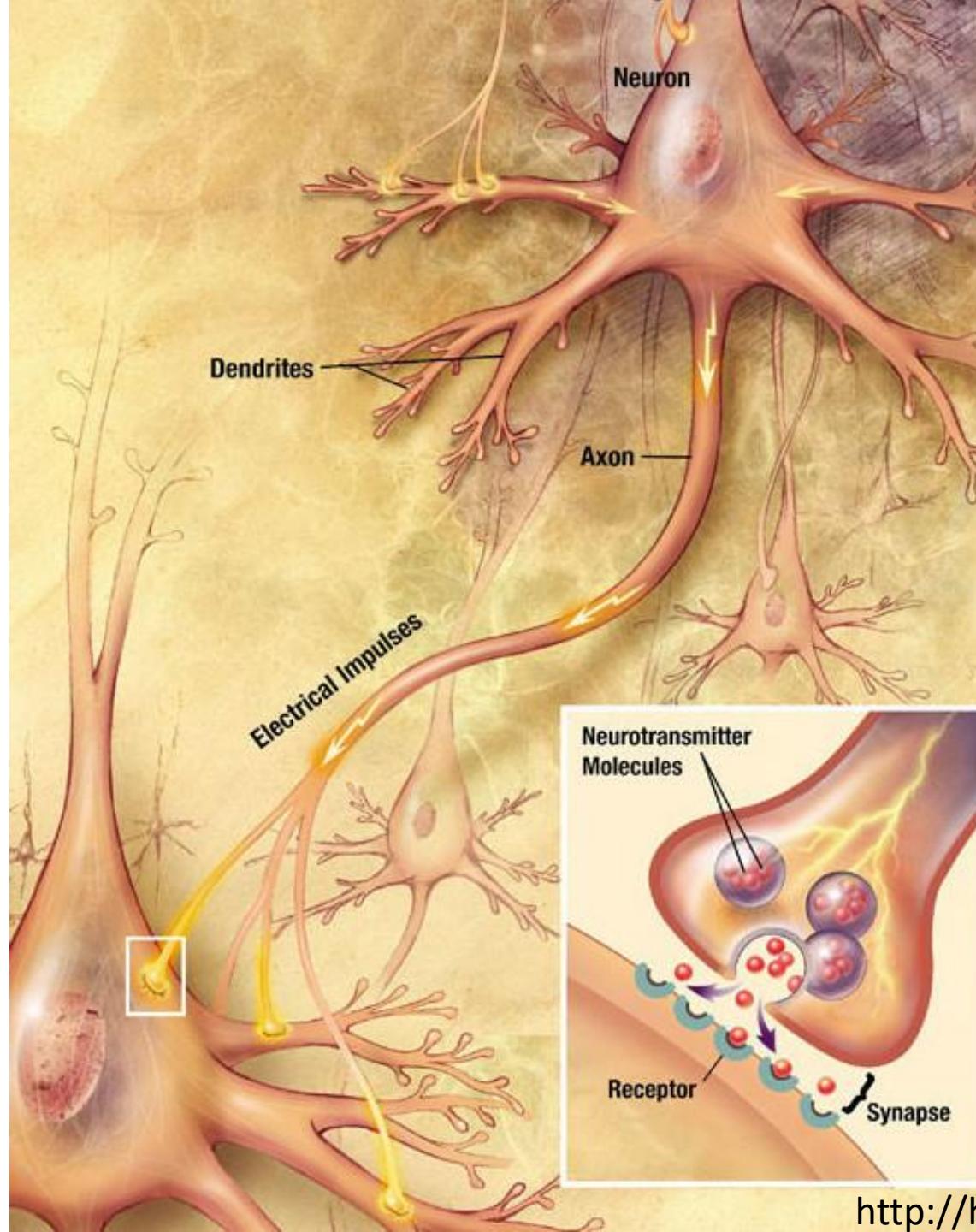
Implanting a 3<sup>rd</sup> eye

# Neural Networks

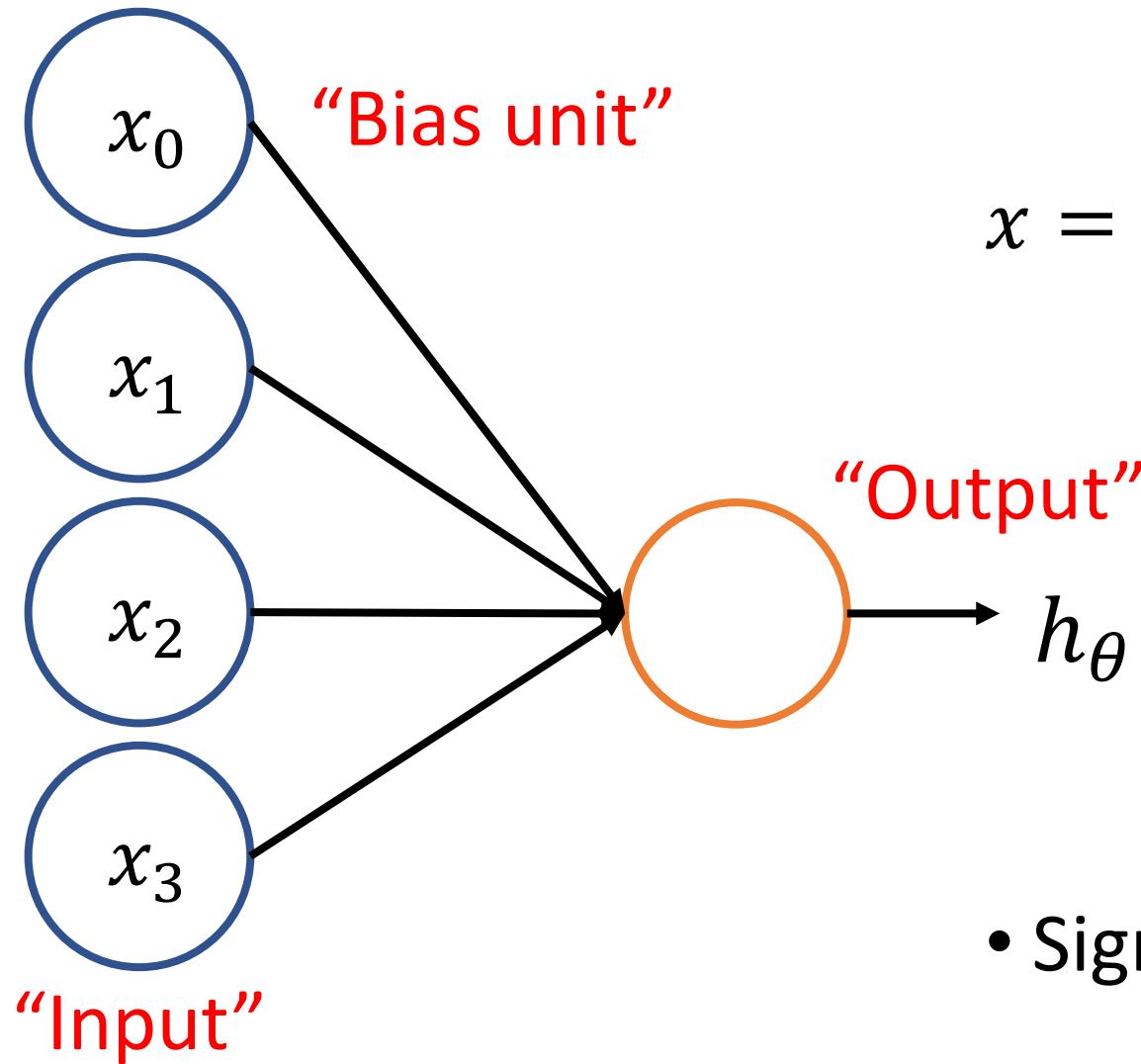
- Why neural networks?
- **Model representation**
- Examples and intuitions
- Multi-class classification

# A single neuron in the brain





# An artificial neuron: Logistic unit



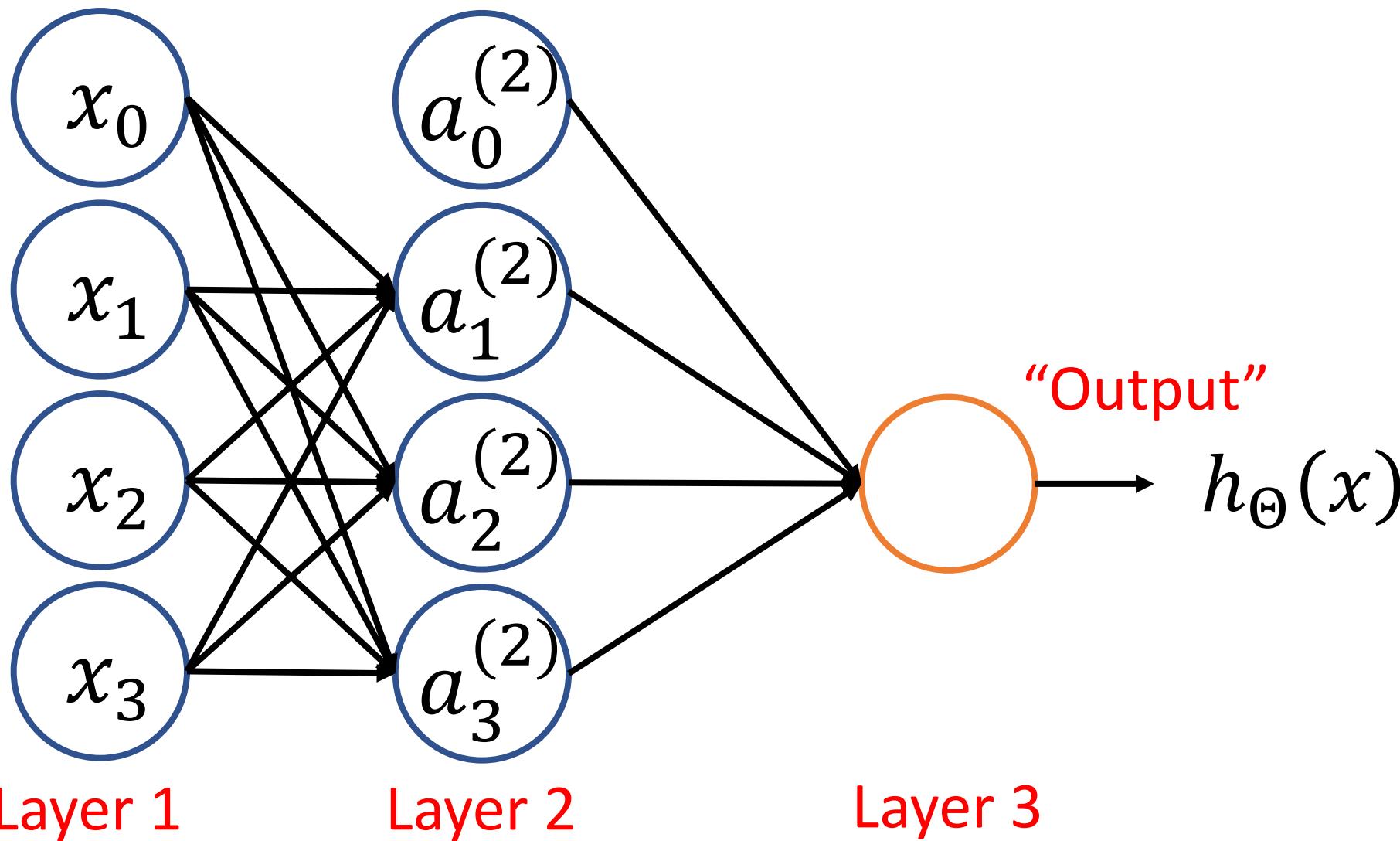
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

"Weights"  
"Parameters"

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

- Sigmoid (logistic) activation function

# Neural network



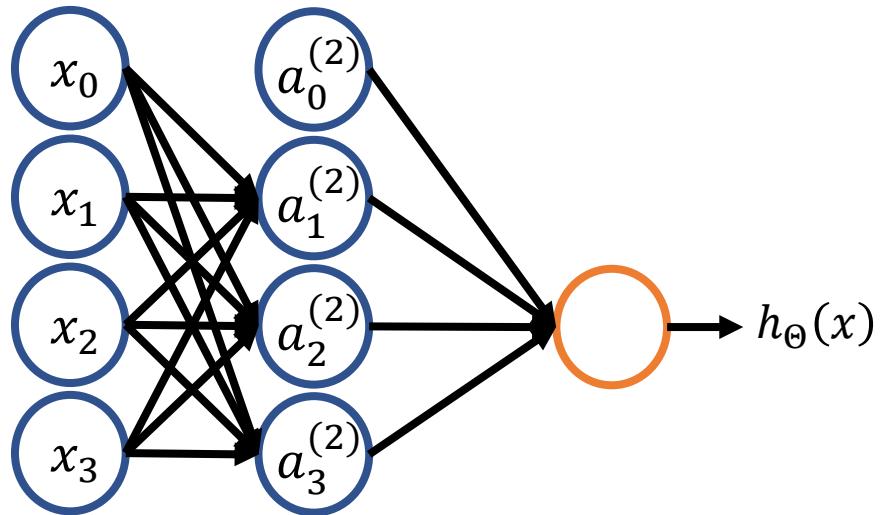
Layer 1

Layer 2

Layer 3

Slide credit: Andrew Ng

# Neural network



$a_i^{(j)}$  = “activation” of unit  $i$  in layer  $j$   
 $\Theta^{(j)}$  = matrix of weights controlling  
function mapping from layer  $j$  to layer  $j + 1$

If networks has :  
 $s_j$  unit in layer  $j$   
 $s_{j+1}$  units in layer  $j + 1$

Size of  $\Theta^{(j)}$ ?  
 $s_{j+1} \times (s_j + 1)$

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

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

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

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

# Neural network

- Then  $\Theta^j$  will be of dimensions  $[s_{j+1} \times s_j + 1]$ 
  - Because
    - $s_{j+1}$  is equal to the number of units in layer  $(j + 1)$
    - is equal to the number of units in layer  $j$ , plus an additional unit
- **Looking at the  $\Theta$  matrix**
  - Column length is the number of units in the following layer
  - Row length is the number of units in the current layer + 1 (because we have to map the bias unit)
  - So, if we had two layers - 110 and 15 units in each
    - Then  $\Theta^j$  would be =  $[15 \times 111]$
- **What are the computations ?**
- We have to calculate the activation for each node. That activation depends on
  - The input(s) to the node
  - The parameter associated with that node (from the  $\Theta$  vector associated with that layer)

# Neural network

- We calculate each of the layer-2 activations based on the input values with the bias term (which is equal to 1) i.e.  $x_0$  to  $x_3$ .
- We then calculate the final hypothesis (i.e. the single node in layer 3) using exactly the same logic, except in input is not  $x$  values, but the activation values from the preceding layer.
- The activation value on each hidden unit (e.g.  $a_1^{(2)}$ ) is equal to the sigmoid function applied to the linear combination of inputs
  - Three input units
    - So  $\Theta^{(1)}$  is the matrix of parameters governing the mapping of the input units to hidden units
      - $\Theta^{(1)}$  here is a  $[3 \times 4]$  dimensional matrix
    - Three hidden units
      - Then  $\Theta^{(2)}$  is the matrix of parameters governing the mapping of the hidden layer to the output layer
        - $\Theta^{(2)}$  here is a  $[1 \times 4]$  dimensional matrix (i.e. a row vector)
    - One output unit

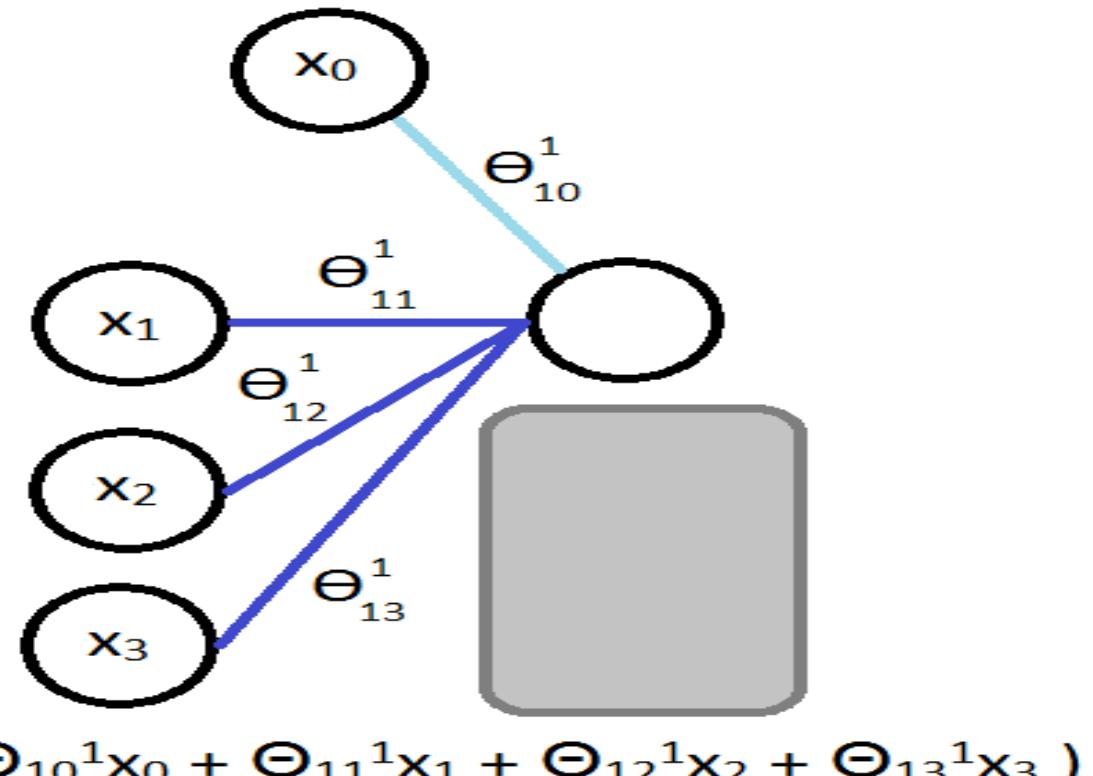
# Neural network

- Every input/activation goes to every node in following layer, which means each "layer transition" uses a matrix of parameters  $\Theta_{ji}^l$  with the following significance

j ranges from 1 to the number of units in layer  $l+1$

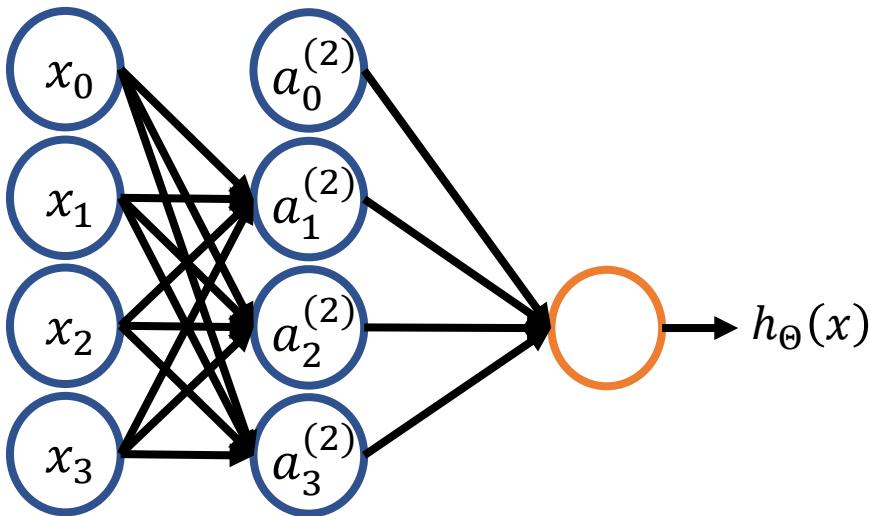
i ranges from 0 to the number of units in layer  $l$

$l$  is the layer you're moving FROM



- For example  $\Theta_{13}^1 =$  means
  - 1 - we're mapping to node 1 in layer  $l+1$
  - 3 - we're mapping from node 3 in layer  $l$
  - 1 - we're mapping from layer 1

# Neural network



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

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

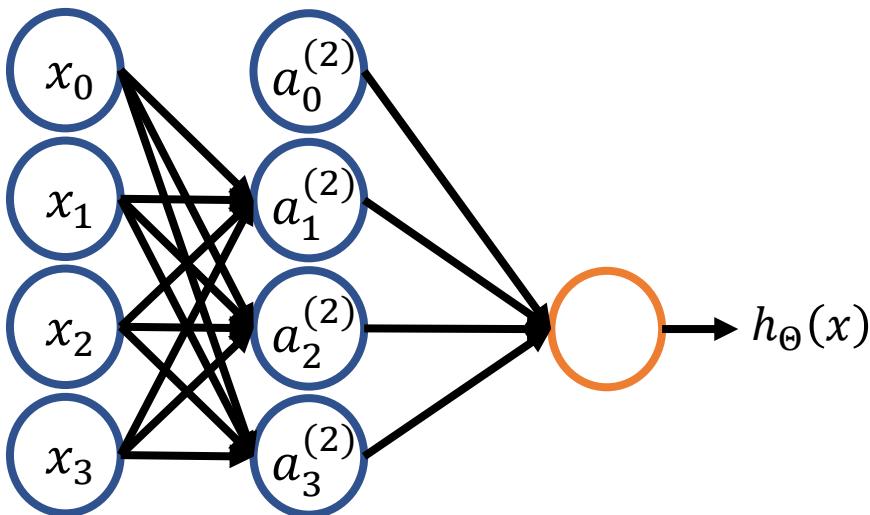
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g(z_2^{(2)})$$

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

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

“Pre-activation”

# Neural network



$$a_1^{(2)} = g(z_1^{(2)})$$

$$a_2^{(2)} = g(z_2^{(2)})$$

$$a_3^{(2)} = g(z_3^{(2)})$$

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

“Pre-activation”

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

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

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

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

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

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

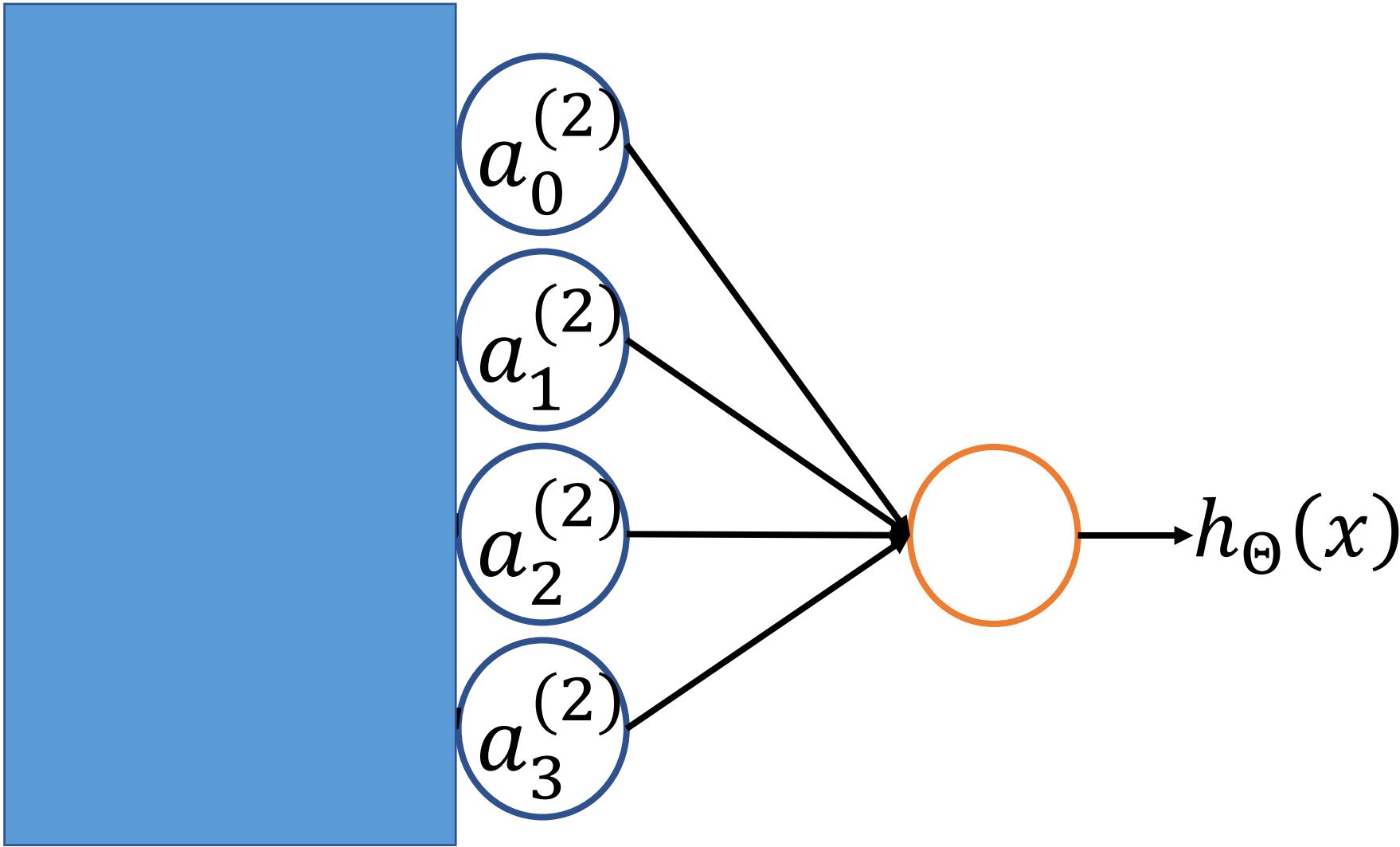
# Forward Propagation

- This process is also called **forward propagation**
  - Start off with activations of input unit
    - i.e. the  $x$  vector as input
  - Forward propagate and calculate the activation of each layer sequentially
  - This is a vectorised version of this implementation

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

# Neural network learning its own features

Diagram below looks a lot like logistic regression



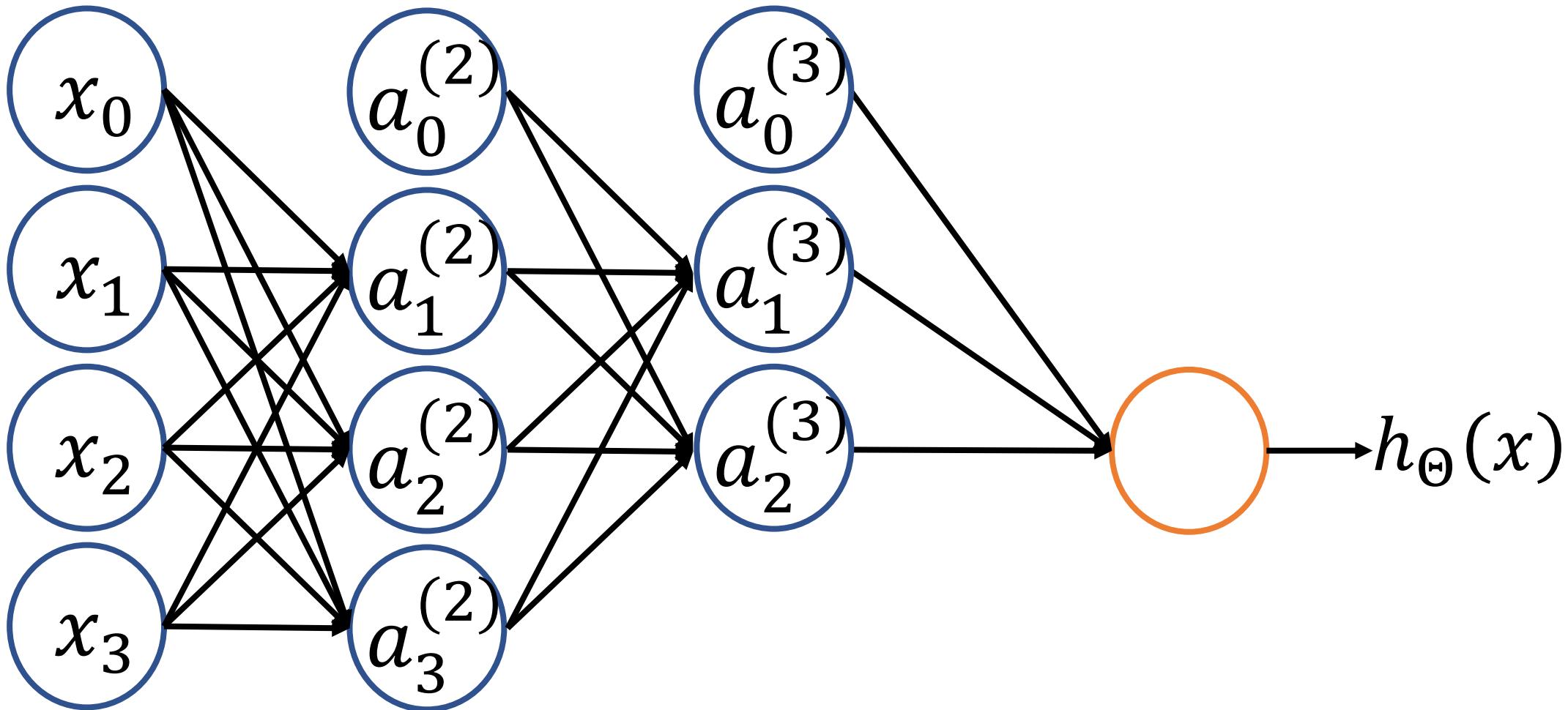
# Neural network learning its own features

- Layer 3 is a logistic regression node
  - The hypothesis output =  $g(\Theta_{10}^2 a_0^2 + \Theta_{11}^2 a_1^2 + \Theta_{12}^2 a_2^2 + \Theta_{13}^2 a_3^2)$
  - This is just logistic regression
    - The only difference is, instead of input a feature vector, the features are just values calculated by the hidden layer
- The features  $a_1^2$ ,  $a_2^2$ , and  $a_3^2$  are calculated/learned - not original features

# Neural network learning its own features

- So the mapping from layer 1 to layer 2 (i.e. the calculations which generate the  $a^2$  features) is determined by another set of parameters -  $\Theta^1$
- So instead of being constrained by the original input features, a neural network can learn its own features to feed into logistic regression.
- Depending on the  $\Theta^1$  parameters you can learn some interesting things
  1. Flexibility to learn whatever features it wants to feed into the final logistic regression calculation.
  2. So, if we compare this to previous logistic regression, you would have to calculate your own exciting features to define the best way to classify or describe something.
  3. NN allows the hidden layers to do that, so we feed the hidden layers our input values, and let them learn whatever gives the best final result to feed into the final output layer

# Other network architectures



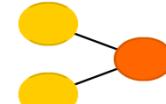
*A mostly complete chart of*

# Neural Networks

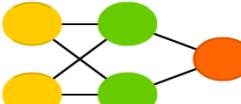
©2016 Fjodor van Veen - [asimovinstitute.org](http://asimovinstitute.org)

- (○) Backfed Input Cell
- (○) Input Cell
- (△) Noisy Input Cell
- (●) Hidden Cell
- (○) Probabilistic Hidden Cell
- (△) Spiking Hidden Cell
- (●) Output Cell
- (○) Match Input Output Cell
- (●) Recurrent Cell
- (○) Memory Cell
- (△) Different Memory Cell
- (●) Kernel
- (○) Convolution or Pool

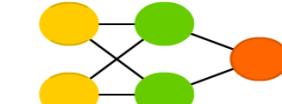
Perceptron (P)



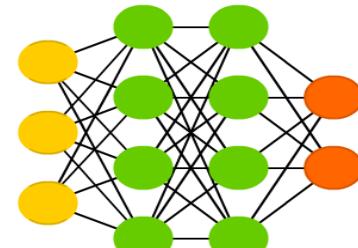
Feed Forward (FF)



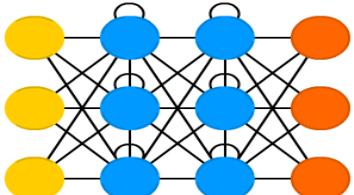
Radial Basis Network (RBF)



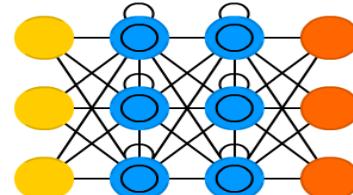
Deep Feed Forward (DFF)



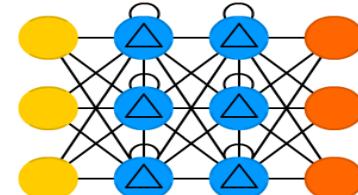
Recurrent Neural Network (RNN)



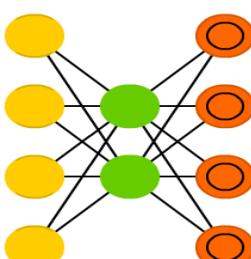
Long / Short Term Memory (LSTM)



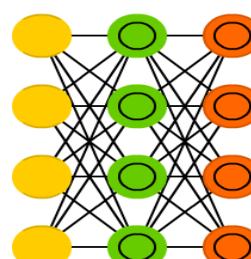
Gated Recurrent Unit (GRU)



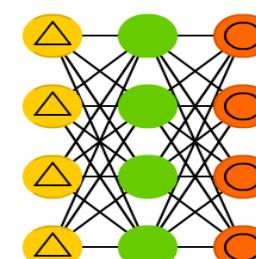
Auto Encoder (AE)



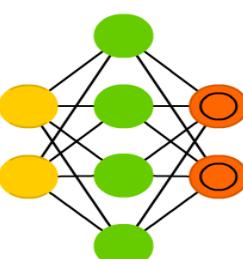
Variational AE (VAE)



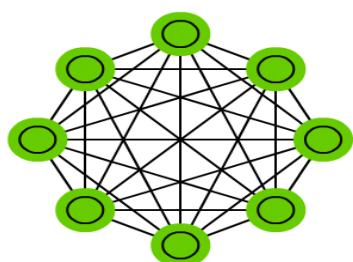
Denoising AE (DAE)



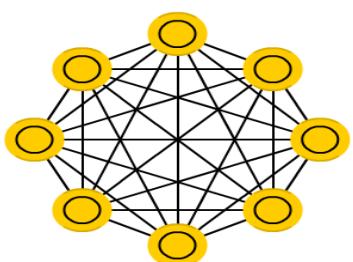
Sparse AE (SAE)



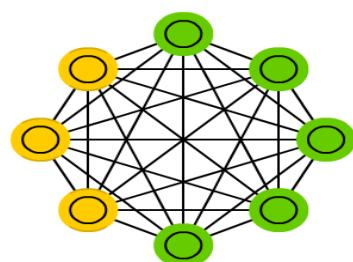
Markov Chain (MC)



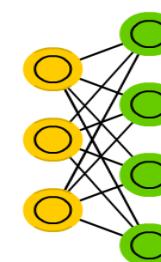
Hopfield Network (HN)



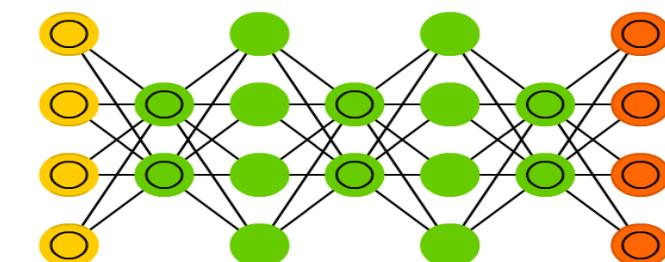
Boltzmann Machine (BM)



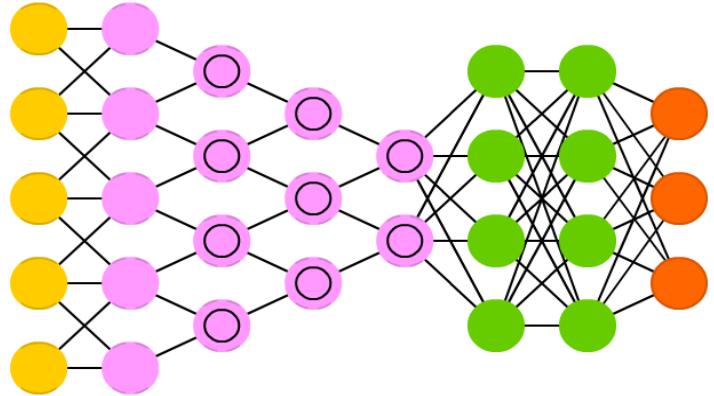
Restricted BM (RBM)



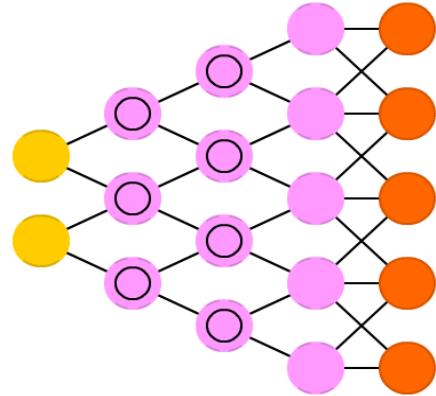
Deep Belief Network (DBN)



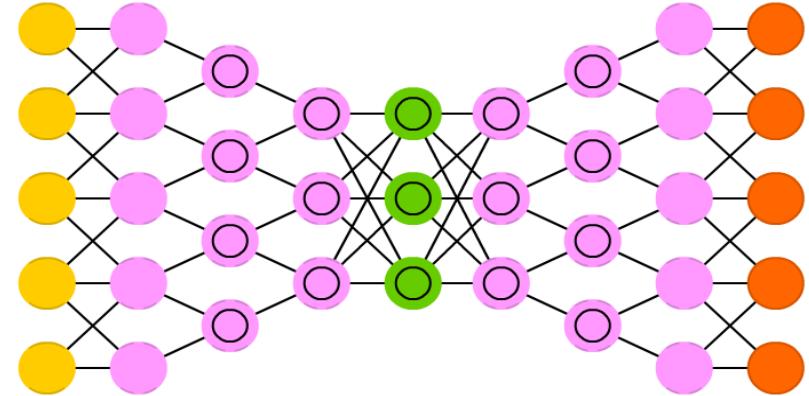
Deep Convolutional Network (DCN)



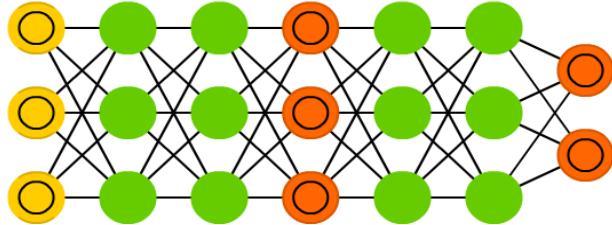
Deconvolutional Network (DN)



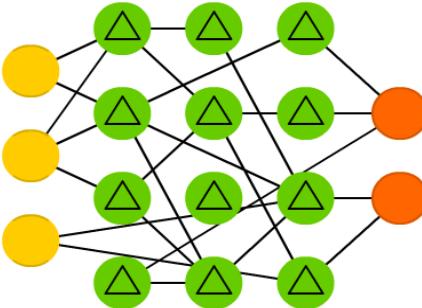
Deep Convolutional Inverse Graphics Network (DCIGN)



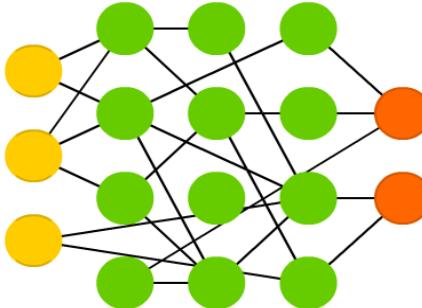
Generative Adversarial Network (GAN)



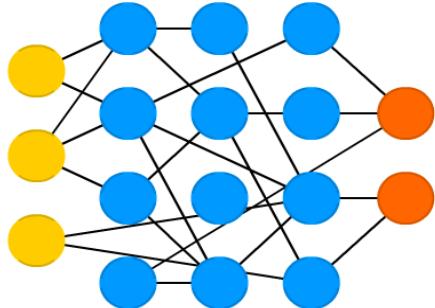
Liquid State Machine (LSM)



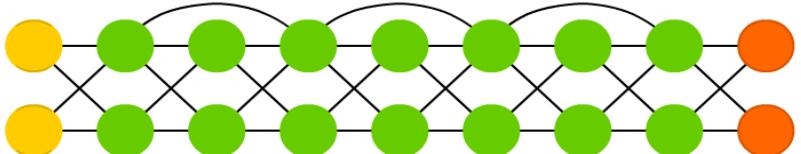
Extreme Learning Machine (ELM)



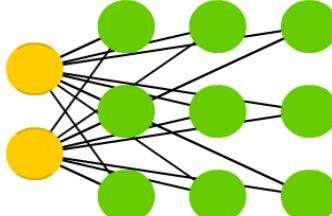
Echo State Network (ESN)



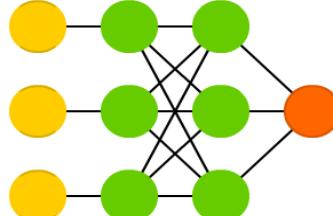
Deep Residual Network (DRN)



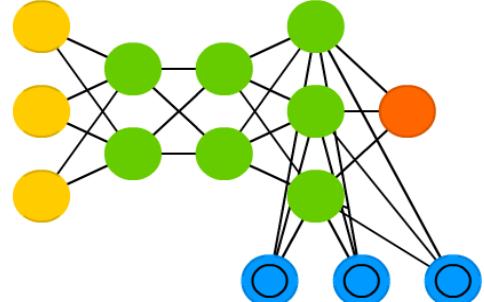
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)

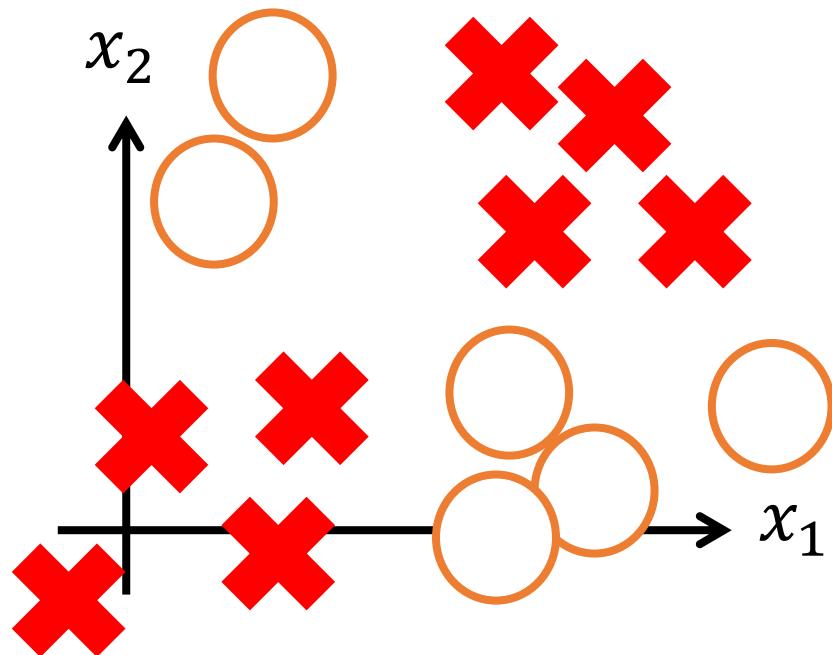
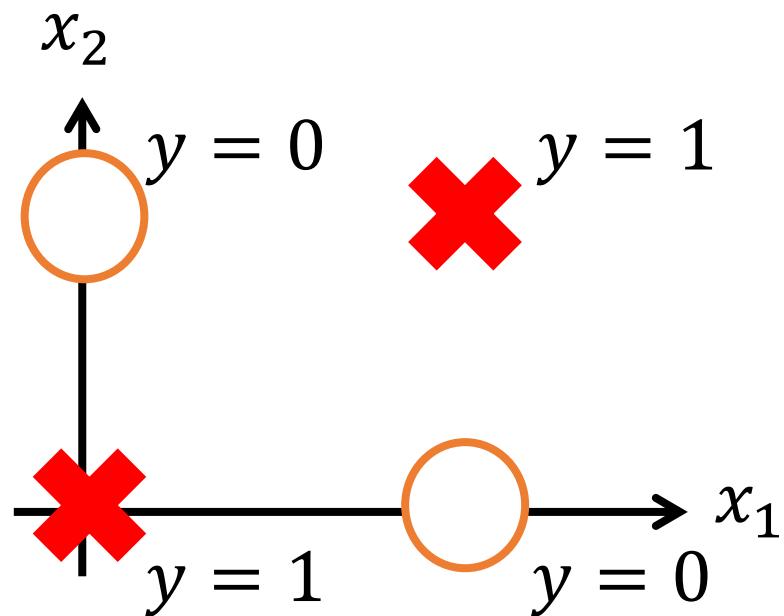


# Neural Networks

- Why neural networks?
- Model representation
- **Examples and intuitions**
- Multi-class classification

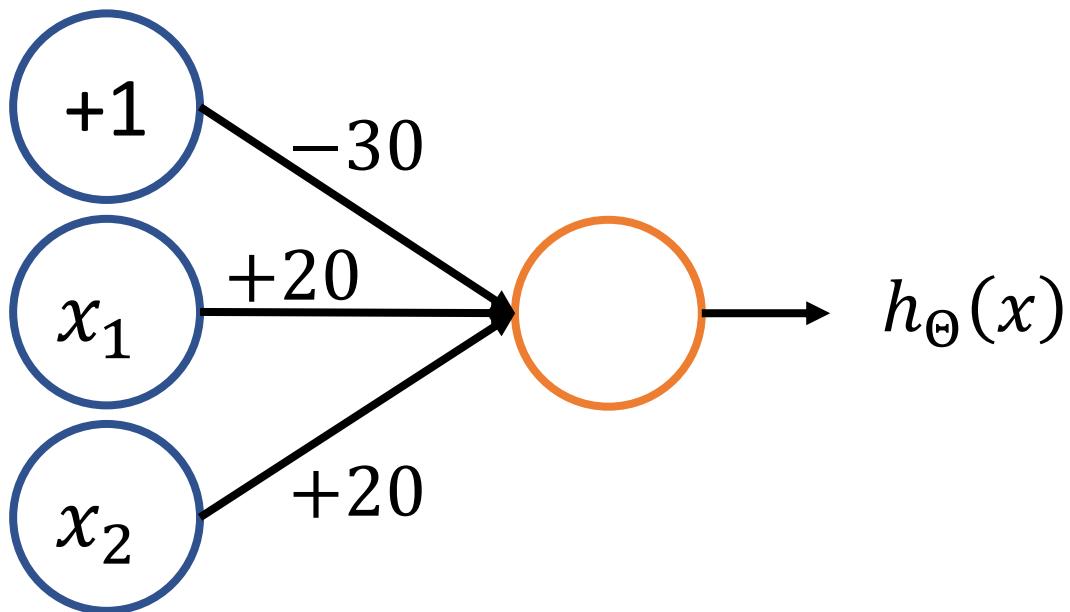
# A complex Non-linear classification example using Neural Networks: XOR/XNOR

- $x_1, x_2$  are binary (0 or 1)
- $y = XOR(x_1, x_2)$

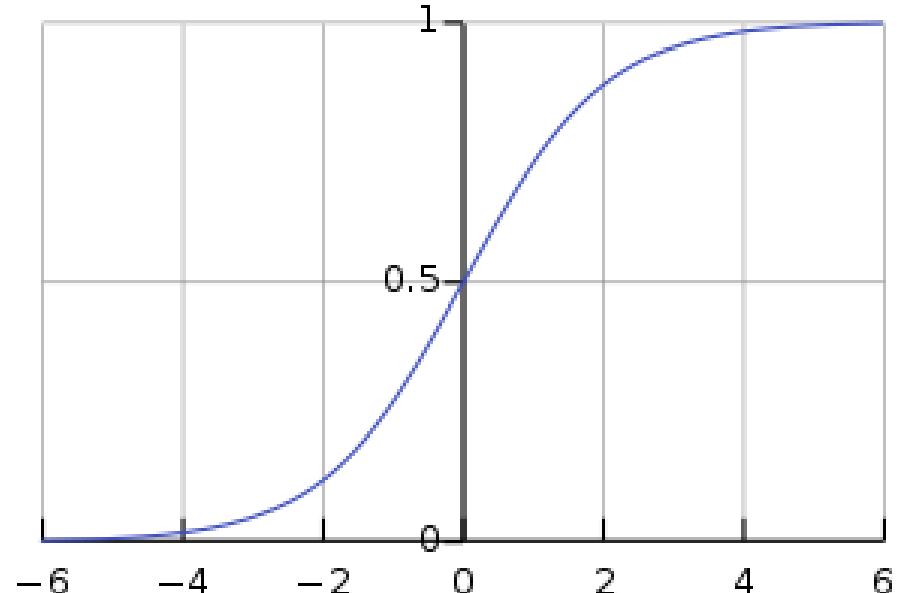


## Simple example: AND

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$



$$h_{\Theta}(x) = g(-30 + 20x_1 + 20x_2)$$



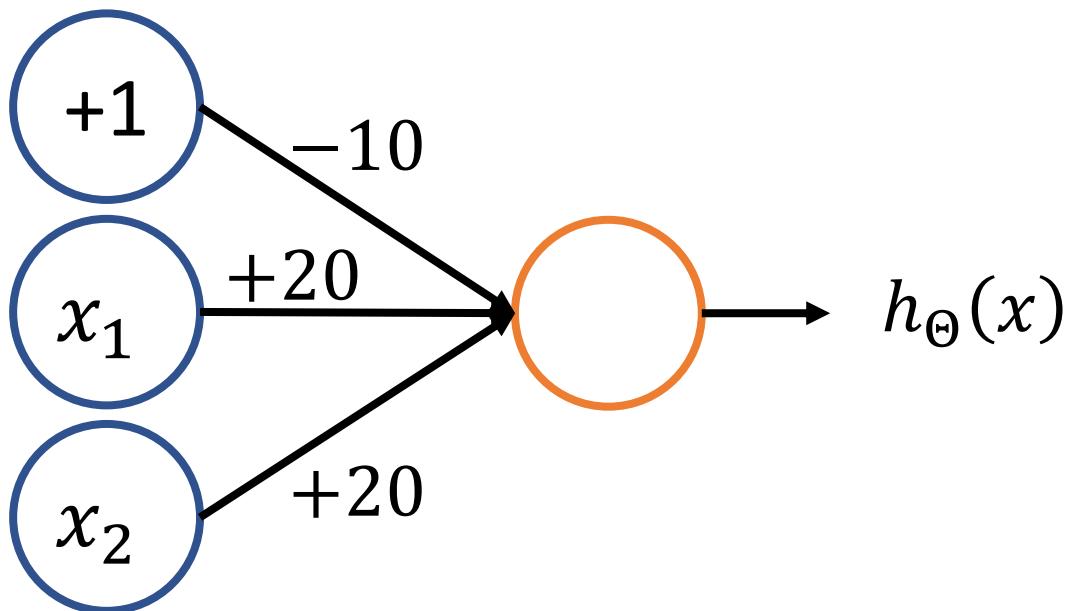
$x_1$	$x_2$	$h_{\Theta}(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

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

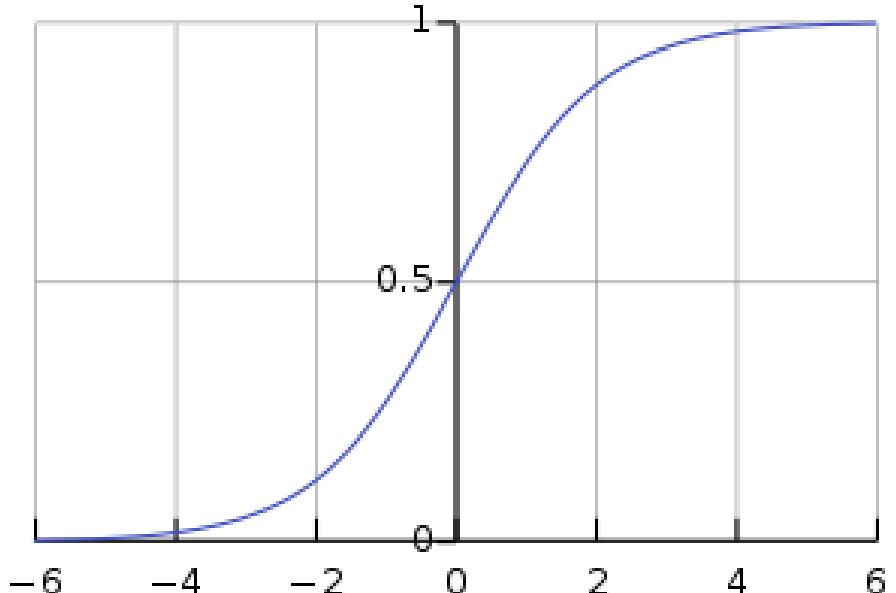
Slide credit: Andrew Ng

## Simple example: OR

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ OR } x_2$



$$h_{\Theta}(x) = g(-10 + 20x_1 + 20x_2)$$



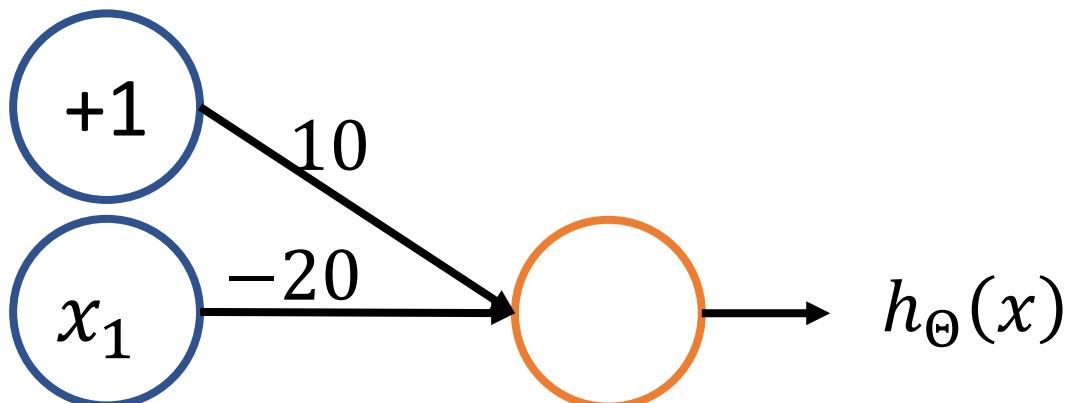
$x_1$	$x_2$	$h_{\Theta}(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

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

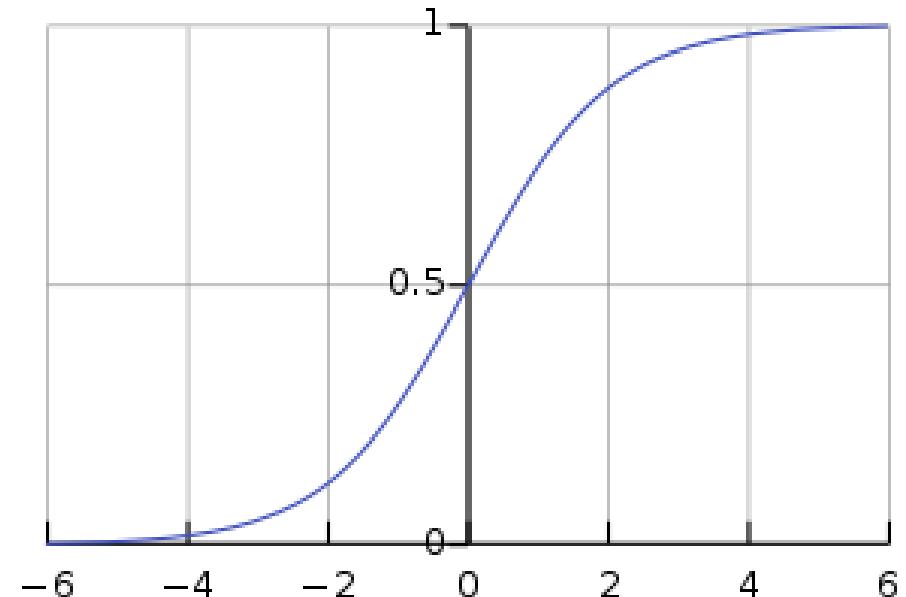
Slide credit: Andrew Ng

## Simple example: NOT

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$



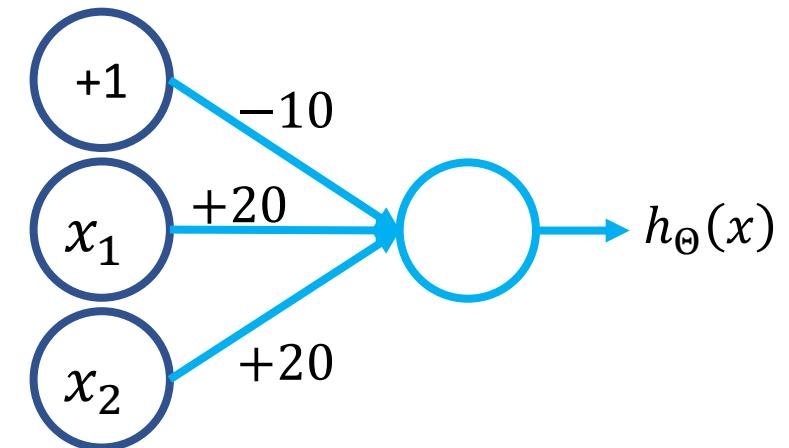
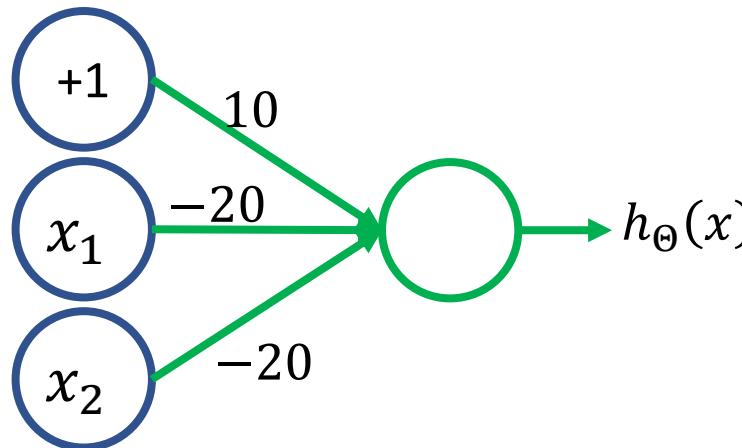
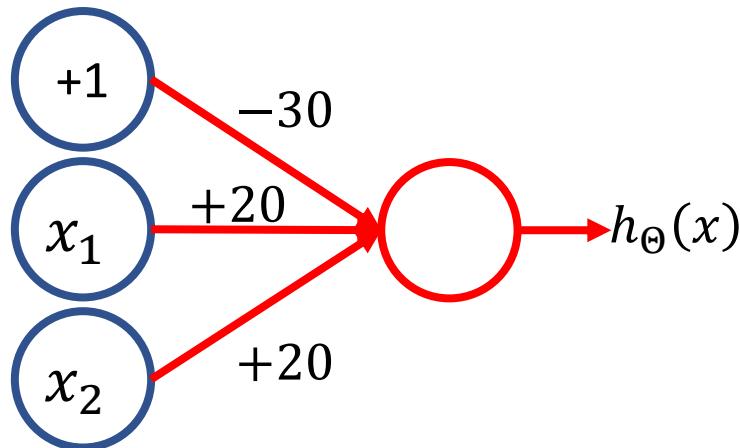
$$h_{\Theta}(x) = g(10 - 20x_1)$$



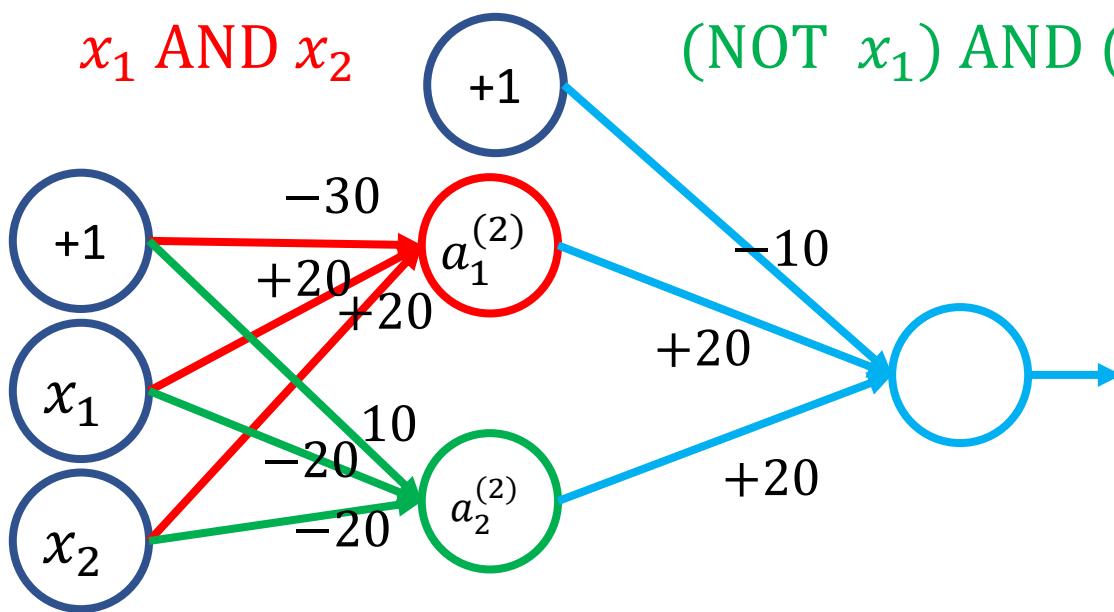
$x_1$	$h_{\Theta}(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

$$h_{\Theta}(x) \approx \text{NOT } x_1$$

## Putting it together: $x_1$ XNOR $x_2$



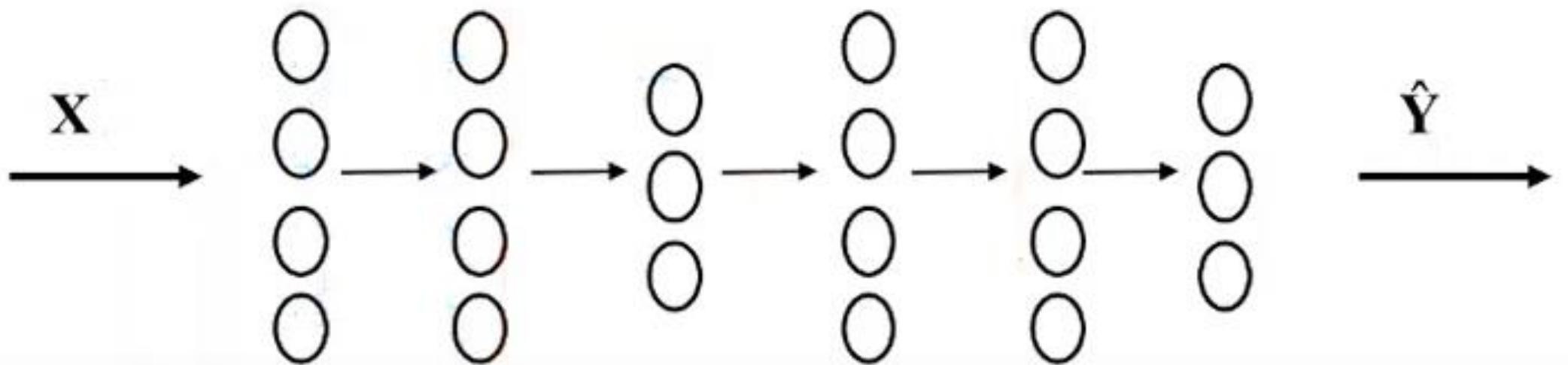
$x_1$  AND  $x_2$



(NOT  $x_1$ ) AND (NOT  $x_2$ )

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

# Layers



# Neural Networks

- Why neural networks?
- Model representation
- Examples and intuitions
- **Multi-class classification**

# Multiple output units: One-vs-all



Pedestrian



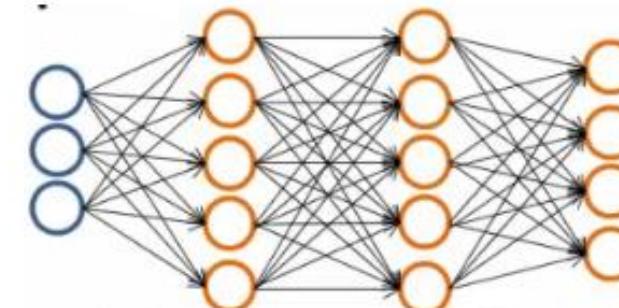
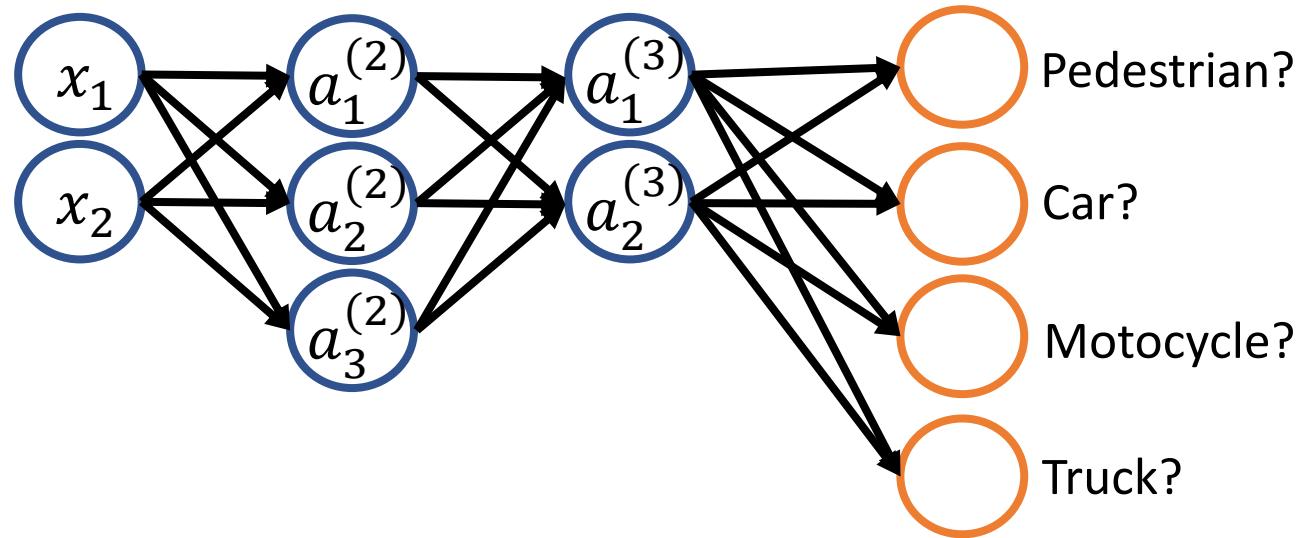
Car



Motorcycle

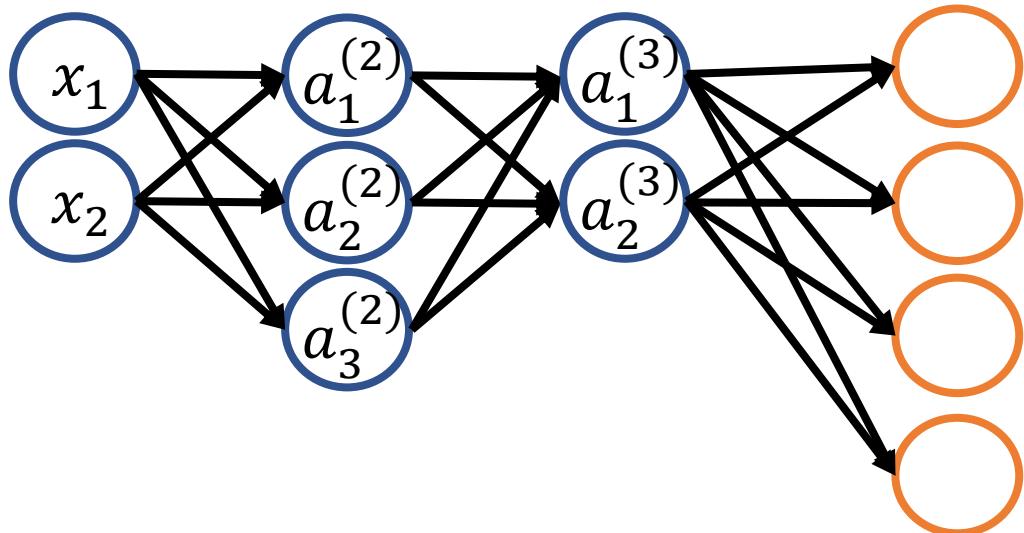


Truck



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

# Multiple output units: One-vs-all



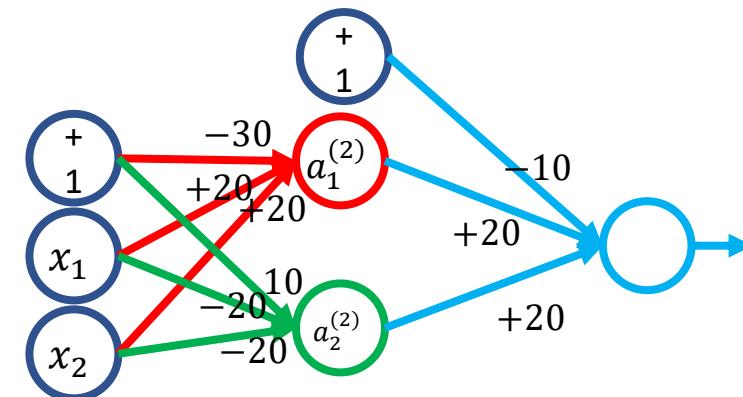
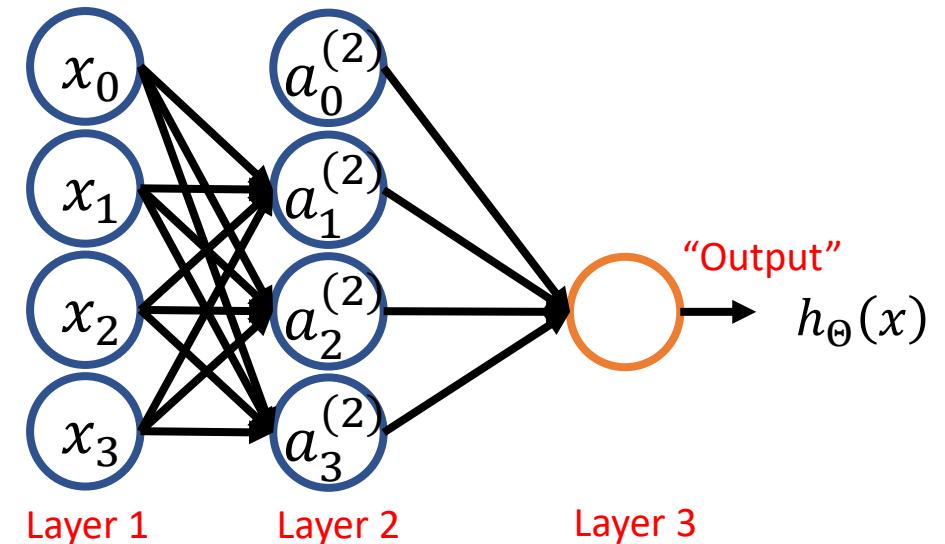
$$h_{\Theta}(x) \in R^4$$

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

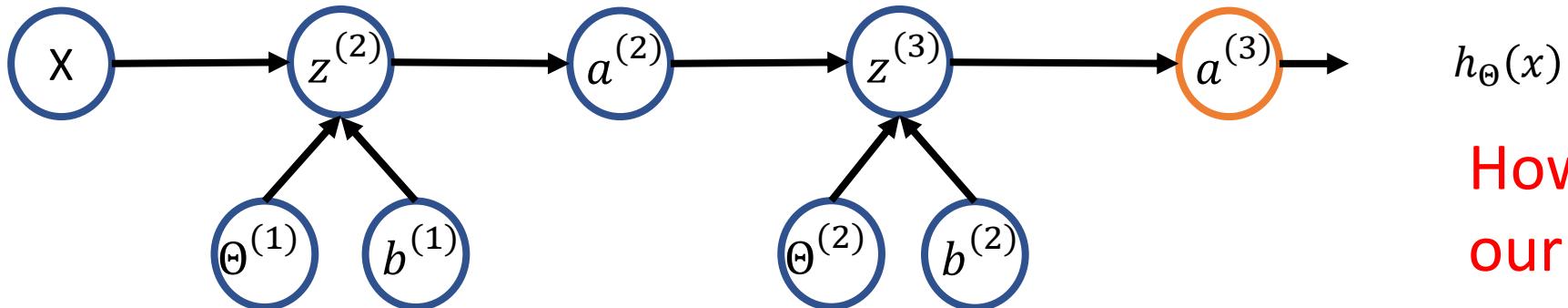
$$y^{(i)} \text{ one of } \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

# Things to remember

- Why neural networks?
- Model representation
- Examples and intuitions
- Multi-class classification



# Flow graph - Forward propagation



How do we evaluate  
our prediction?

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

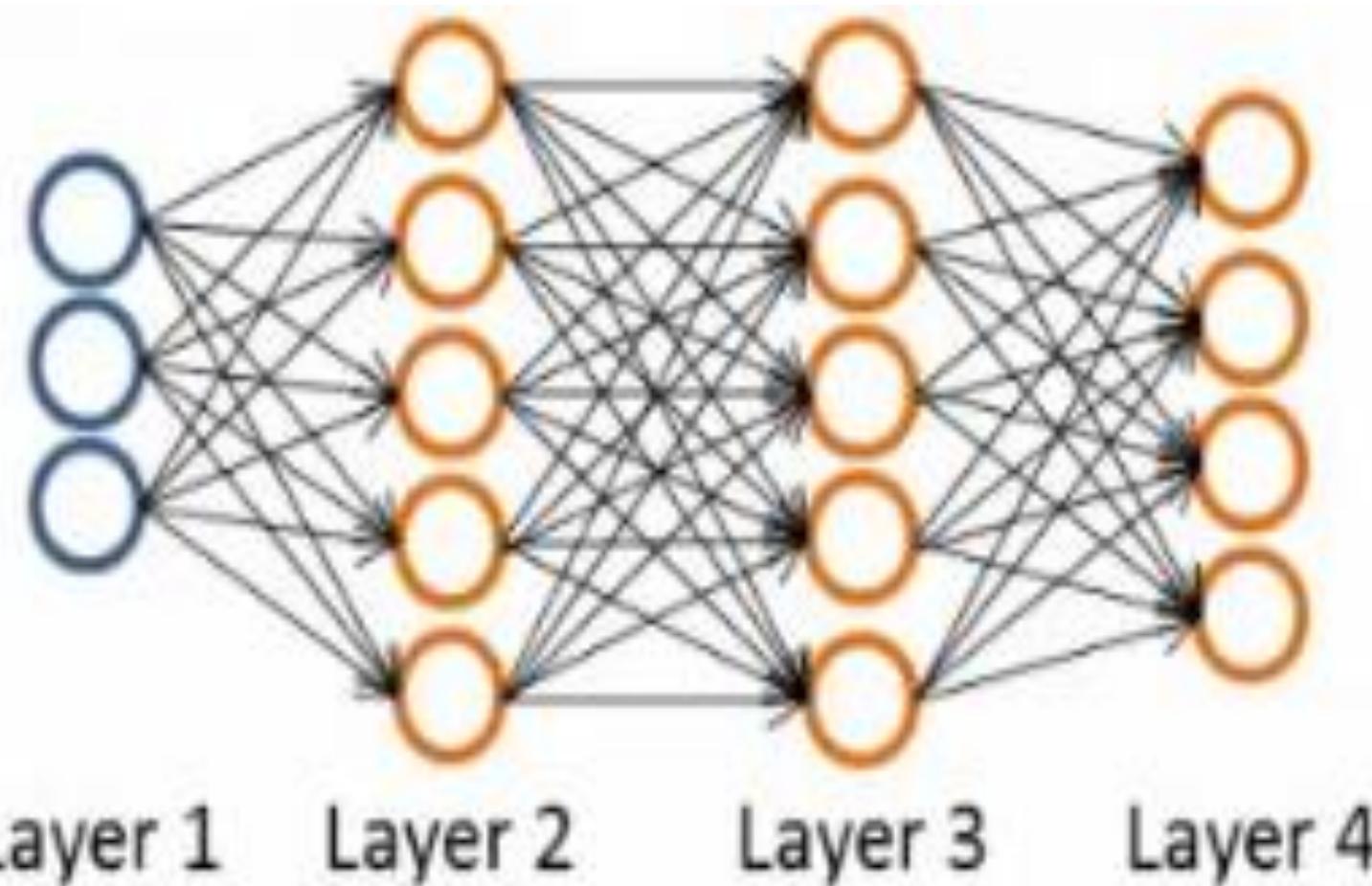
$$a^{(2)} = g(z^{(2)})$$

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

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

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

# Neural Network cost function



# Cost function

Logistic regression:

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

Neural network:

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

$$J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right]$$

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

# Cost function

- Our cost function now outputs a  $k$  dimensional vector
  - $h_{\theta}(x)$  is a  $k$  dimensional vector, so  $h_{\theta}(x)_i$  refers to the  $i$ th value in that vector
- Cost function  $J(\Theta)$  is
  - $[-1/m]$  times a sum of a similar term to which we had for logic regression
  - But now this is also a sum from  $k = 1$  through to  $K$  ( $K$  is number of output nodes)
    - Summation is a sum over the  $k$  output units - i.e. for each of the possible classes
    - So if we had 4 output units then the sum is  $k = 1$  to 4 of the logistic regression over each of the four output units in turn

# Cost function

$$J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] \\ + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

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

Need to compute:

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

- This is a massive regularization summation term, it's a fairly straightforward triple nested summation.
- This is also called a weight decay term ,as before, the lambda value determines the important of the two halves
- The regularization term is similar to that in logistic regression. So, we have a cost function, but how do we minimize this

# Gradient computation

Given one training example  $(x, y)$

$$a^{(1)} = x$$

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

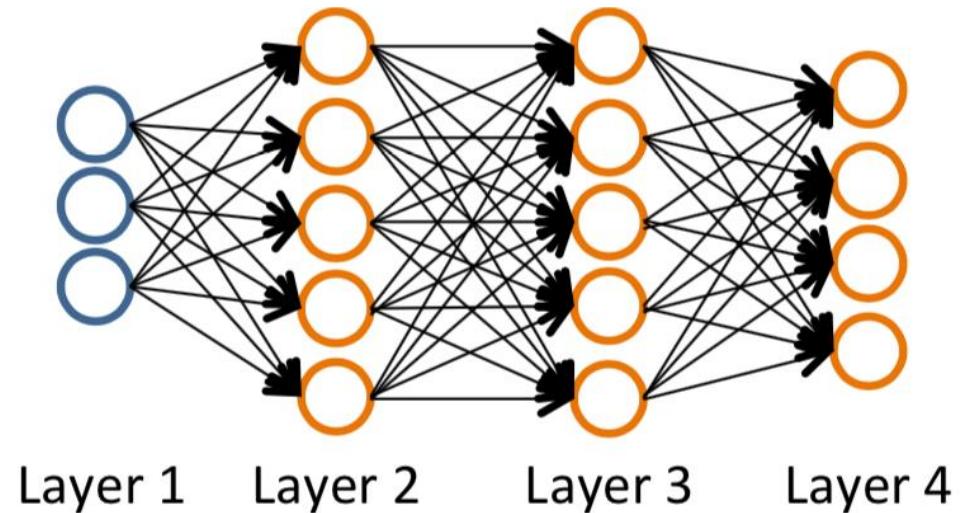
$$a^{(2)} = g(z^{(2)}) \text{ (add } a_0^{(2)})$$

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

$$a^{(3)} = g(z^{(3)}) \text{ (add } a_0^{(3)})$$

$$z^{(4)} = \Theta^{(3)} a^{(3)}$$

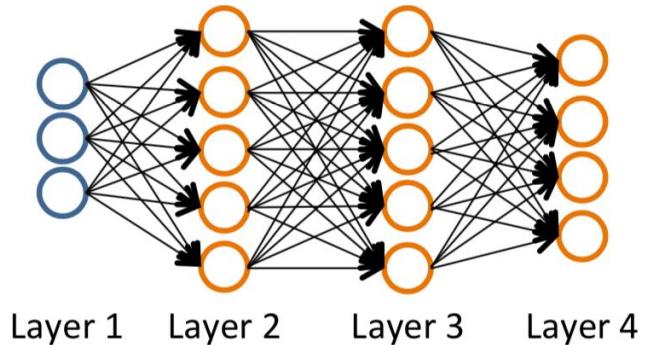
$$a^{(4)} = g(z^{(4)}) = h_{\Theta}(x)$$



# Gradient computation: Backpropagation

Intuition:  $\delta_j^{(l)}$  = “error” of node  $j$  in layer  $l$

For each output unit (layer  $L = 4$ )



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

$$\delta^{(3)} = \delta^{(4)} \frac{\partial \delta^{(4)}}{\partial z^{(3)}} = \delta^{(4)} \frac{\partial \delta^{(4)}}{\partial a^{(4)}} \frac{\partial a^{(4)}}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}}$$

$$\delta^{(3)} = (\Theta^3)^T \delta^4 \cdot * (a^{(3)} \cdot * (1 - a^{(3)}))$$

$$\delta^{(2)} = (\Theta^2)^T \delta^2 \cdot * (a^{(2)} \cdot * (1 - a^{(2)}))$$

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

$$a^{(3)} = g(z^{(3)})$$

$$z^{(4)} = \Theta^{(3)} a^{(3)}$$

$$a^{(4)} = g(z^{(4)})$$

# Why do we do this?

- We do all this to get all the  $\delta$  terms, and we want the  $\delta$  terms because through a very complicated derivation you can use  $\delta$  to get the partial derivative of  $\Theta$  with respect to individual parameters (if you ignore regularization, or regularization is 0)

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

- By doing back propagation and computing the delta terms you can then compute the **partial derivative terms**
- We need the partial derivatives to minimize the cost function!

# Backpropagation algorithm

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

Set  $\Theta^{(1)} = 0$

For  $i = 1$  to  $m$

    Set  $a^{(1)} = x$

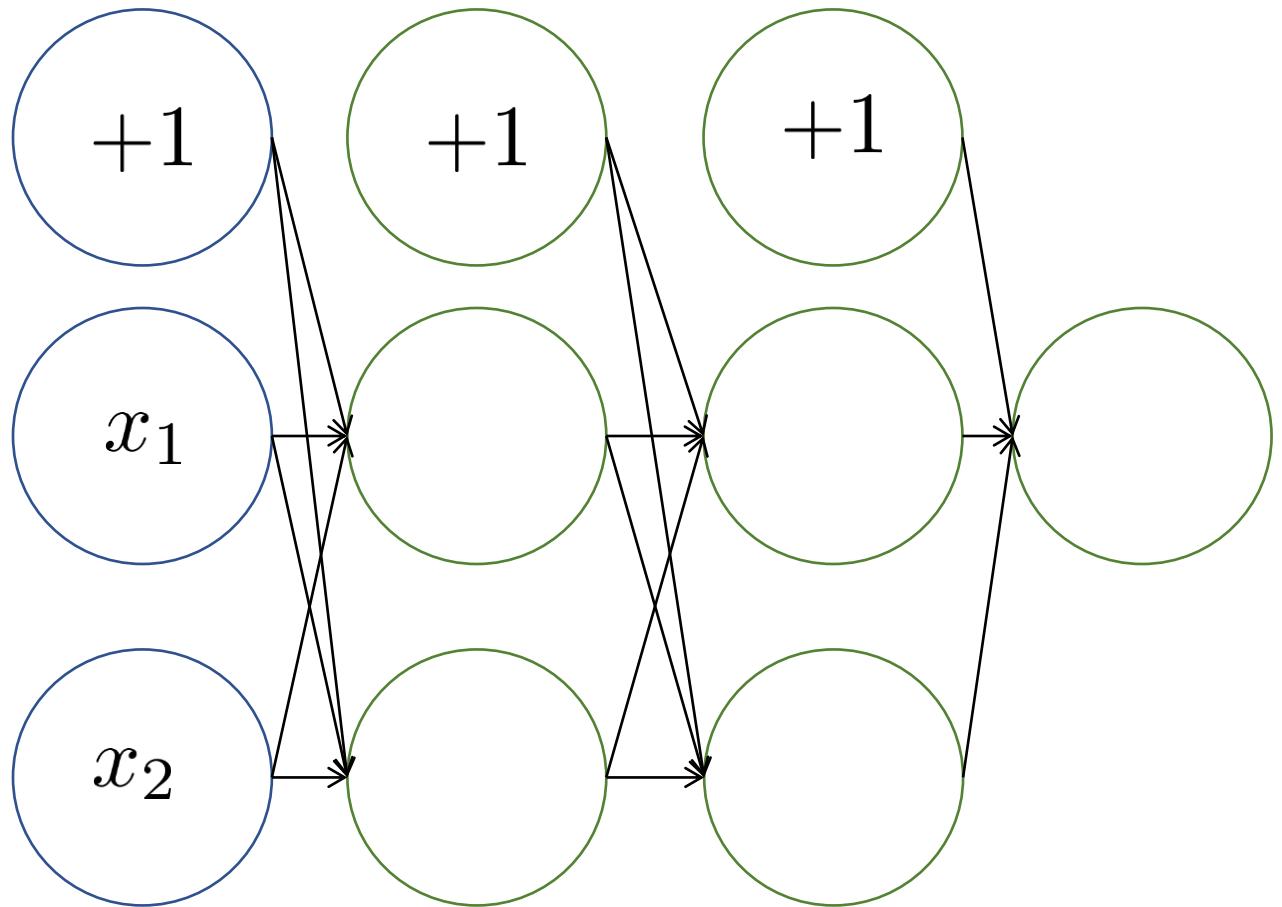
    Perform forward propagation to compute  $a^{(l)}$  for  $l = 2..L$

    use  $y^{(i)}$  to compute  $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute  $\delta^{(L-1)}, \delta^{(L-2)} \dots \delta^{(2)}$

$\Theta^{(l)} = \Theta^{(l)} - a^{(l)} \delta^{(l+1)}$

# Forward Propagation



# What is backpropagation doing?

$$J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \log(h_\Theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\Theta(x^{(i)})) \right] \\ + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

Focusing on a single example  $x^{(i)}$ ,  $y^{(i)}$ , the case of 1 output unit, and ignoring regularization ( $\lambda = 0$ ),

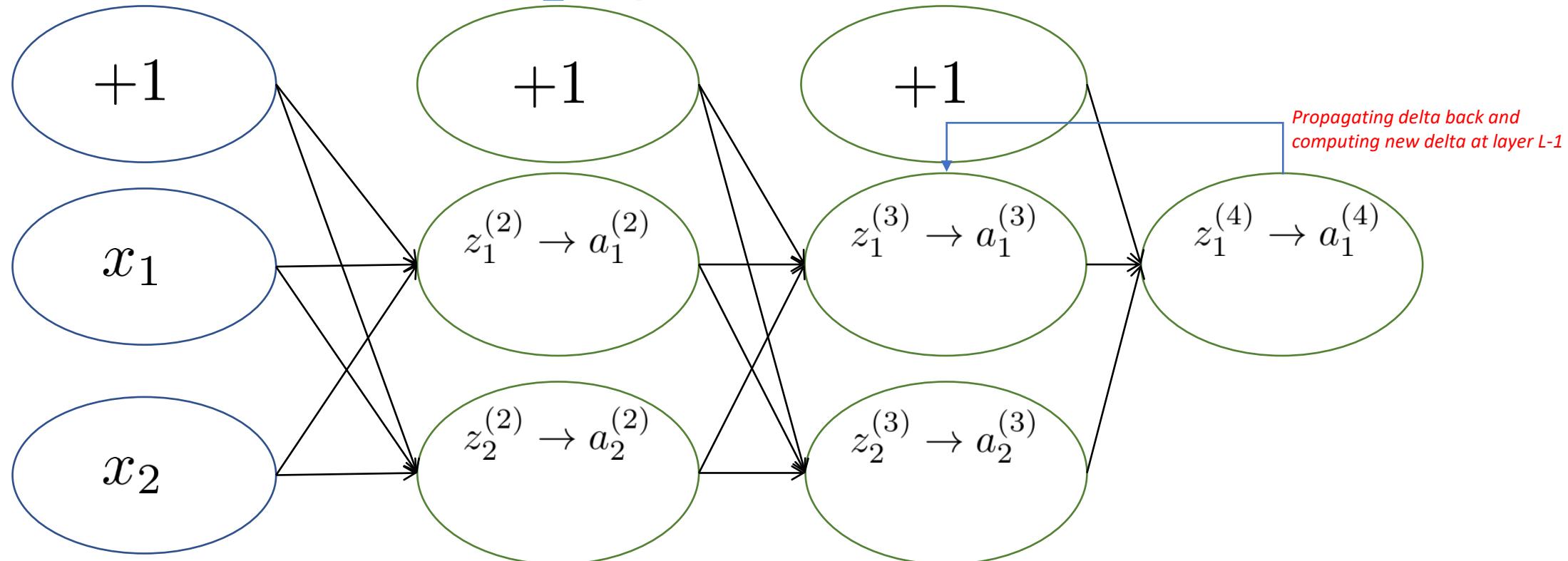
You can think of cost function as a mean square error function to get a better intuition of back propagation algorithm

(Think of  $\text{cost}(i) \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$ )

i.e. how well is the network doing on example i?

$$\text{cost}(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$

# Backward Propagation



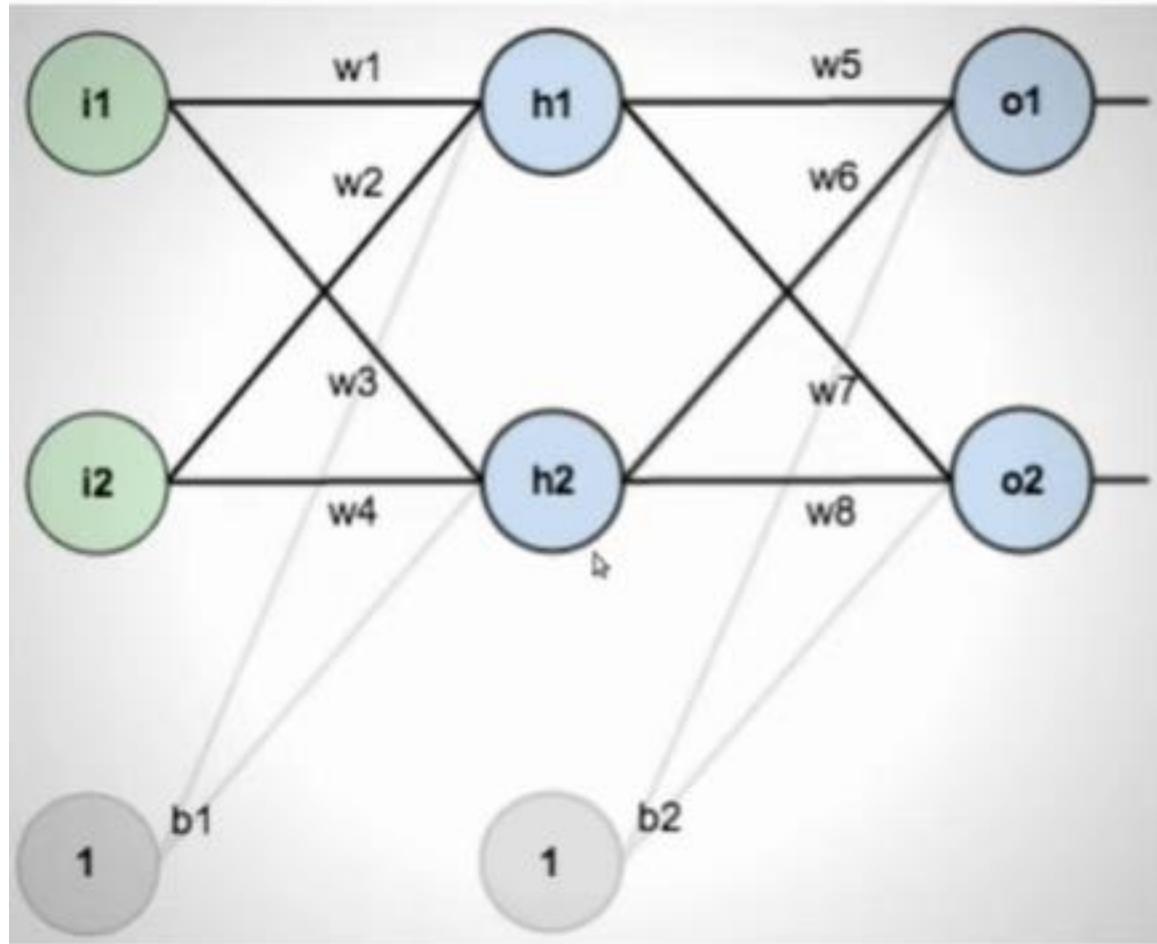
$\delta_j^{(l)}$  = “error” of cost for  $a_j^{(l)}$  (unit  $j$  in layer  $l$ ).

Formally,  $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(i)$  (for  $j \geq 0$ ), where  
 $\text{cost}(i) = y^{(i)} \log h_{\Theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\Theta}(x^{(i)}))$

# Initialization

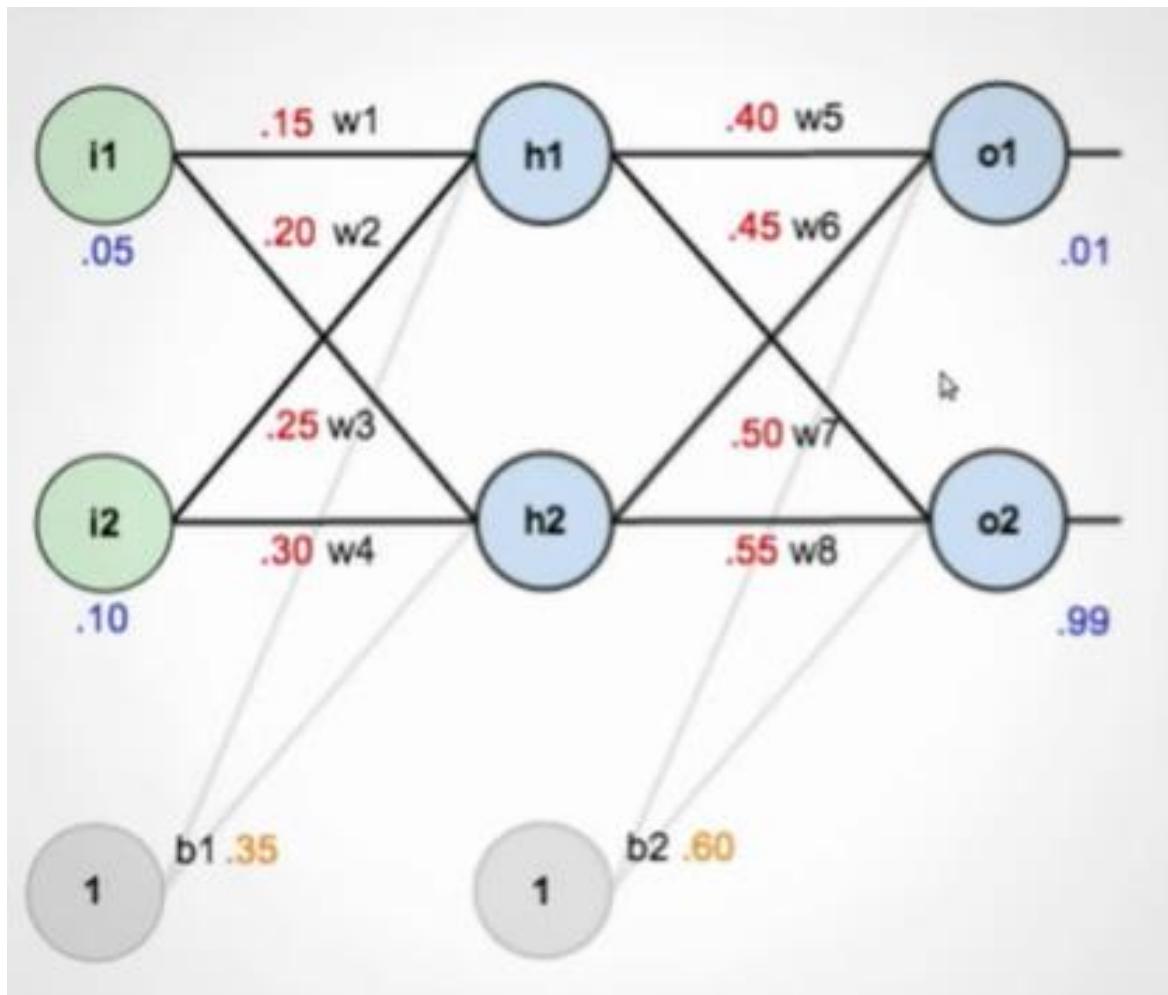
- For bias
  - Initialize all to 1
- For weights
  - Can't initialize all weights to the same value
    - we can show that all hidden units in a layer will always behave the same
    - need to break symmetry
  - Recipe:  $U[-b, b]$ 
    - the idea is to sample around 0 but break symmetry

# Example: Forward Propagation



Why Bias Important?

# Example: Forward Propagation algorithm



$$i_1 = 0.05, i_2 = 0.10$$

$$w_1 = 0.15, w_2 = 0.20$$

$$w_3 = 0.25, w_4 = 0.30$$

$$b_1 = 0.3$$

$$w_5 = 0.4, w_6 = 0.45$$

$$w_7 = 0.5, w_8 = 0.55$$

$$b_2 = 0.6$$

$$o_1 = 0.01, o_2 = 0.99$$

# Example: Forward Propagation algorithm

First Step: Forward Pass [ $i_1 \rightarrow o_1$ , and  $i_2 \rightarrow o_2$ ]

Hidden Layer 1's units Calculation

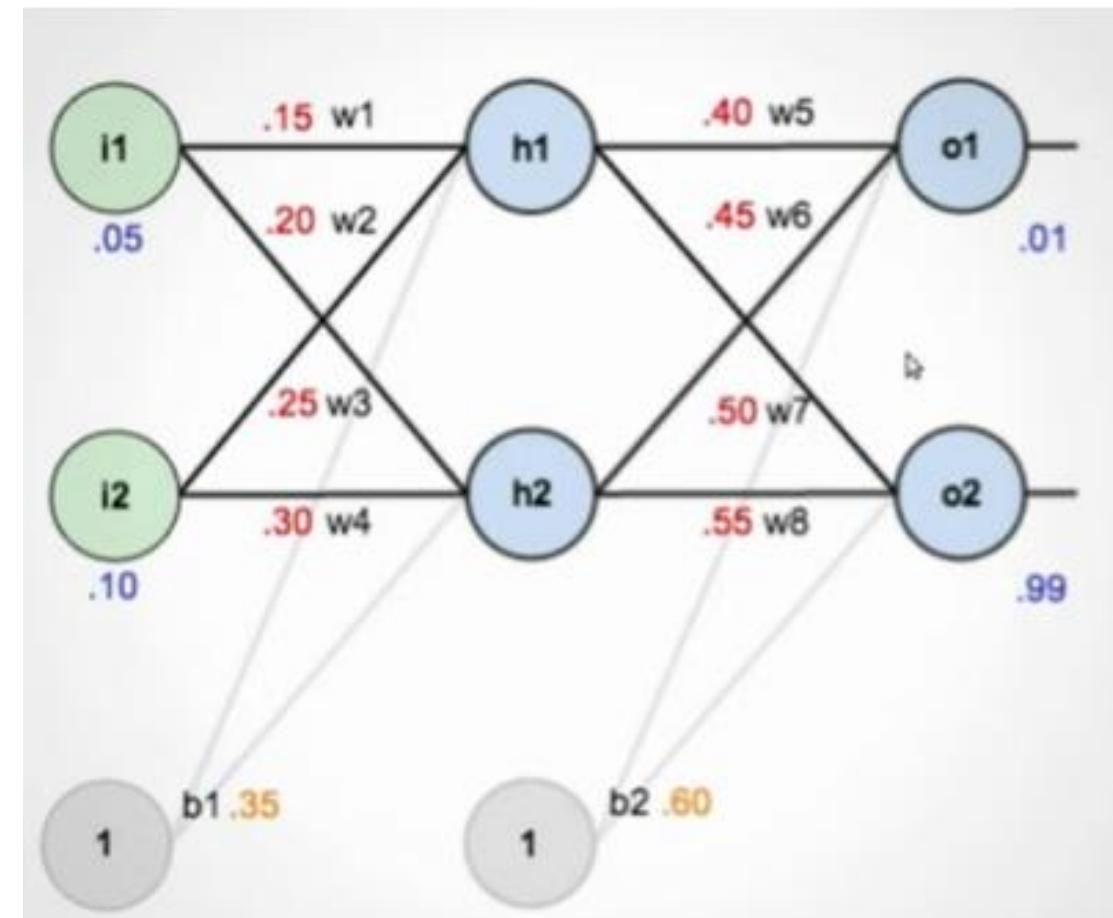
$$\begin{aligned} h_1 &= w_1 * i_1 + w_2 * i_2 + b_1 * 1 \\ (\text{in}) &= 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 \\ &= 0.3775 \end{aligned}$$

$$\begin{aligned} h_2 &= w_3 * i_1 + w_4 * i_2 + b_2 * 1 \\ (\text{in}) &= 0.25 * 0.05 + 0.30 * 0.1 + 0.35 * 1 \\ &= 0.3925 \end{aligned}$$

Now, Apply activation functn: logistic function / Sigmoid function

$$h_1 = \frac{1}{1+e^{-0.3775}} = 0.5932$$

$$(act) \quad h_2 = \frac{1}{1+e^{-0.3925}} = 0.5968$$



### Output layer's error Computation

$$O_1 = h_1 * \omega_5 + h_2 * \omega_6 + b_2 * 1$$

$$(in) = \cancel{.5932} * \overset{(out)}{.5932} * .40 + \overset{(out)}{.5968} * .45 + .6 * 1$$

$$= 1.10590$$

$$O_2 = h_1 * \omega_7 + h_2 * \omega_8 + b_2 * 1$$

$$(in) = \overset{(out)}{.5932} * .50 + \overset{(out)}{.5968} * .55 + .6 * 1$$

$$= 1.2248$$

Now, apply activation function: sigmoid function

$$O_1 = \overset{(out)}{.75136} \left( = \frac{1}{1+e^{-1.1059}} \right)$$

$$O_2 = \overset{(out)}{.7729} \left( = \frac{1}{1+e^{-1.2248}} \right)$$

### Step 2: Calculate Error

$$\epsilon_{\text{total}} = \sum \frac{1}{2} (\text{target} - \text{output})^2 = \frac{1}{2} (\text{target}_{O_1} - O_1 \text{ (out)})^2 + \frac{1}{2} (\text{target}_{O_2} - O_2 \text{ (out)})^2$$

$$\epsilon_{O_1} = \frac{1}{2} (\text{target}_{O_1} - O_1 \text{ (out)})^2 = \frac{1}{2} (0.01 - \overset{(out)}{.75136})^2 = 0.2748 \gg 0$$

$$\epsilon_{O_2} = \frac{1}{2} (\text{target}_{O_2} - O_2 \text{ (out)})^2 = \frac{1}{2} (0.99 - \overset{(out)}{.7729})^2 = 0.02357$$

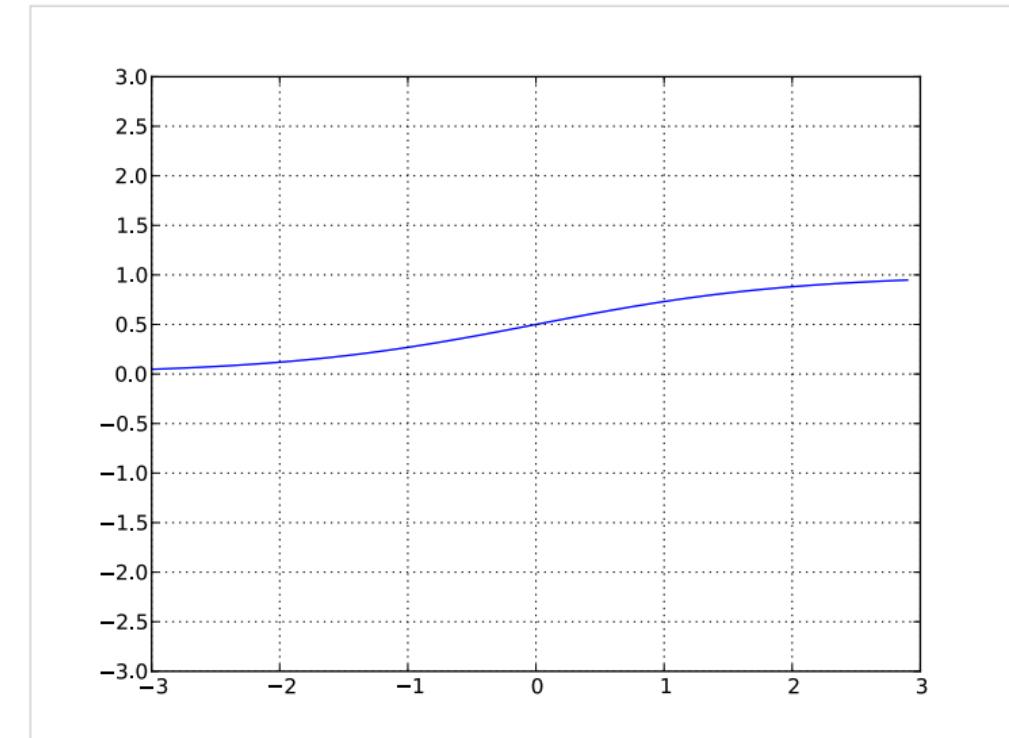
$$\therefore \epsilon_{\text{total}} = \epsilon_{O_1} + \epsilon_{O_2} = 0.2748 + 0.02357 = 0.29837$$

# Activation - Sigmoid

- Partial derivative

$$g'(x) = g(x)(1 - g(x))$$

- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing



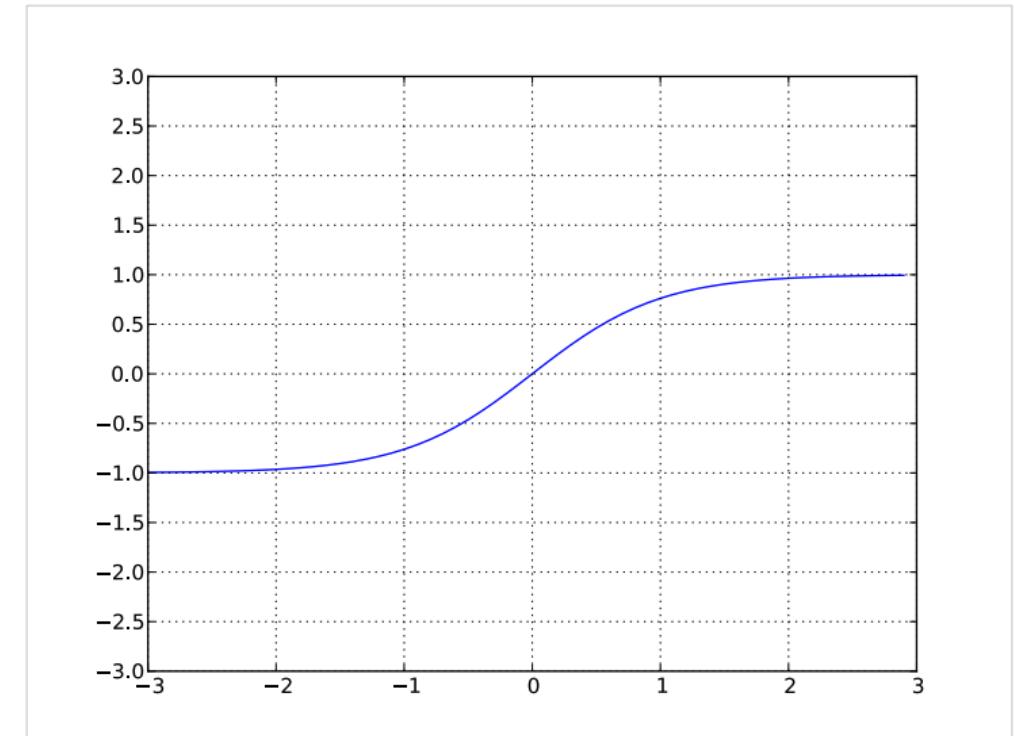
$$g(x) = \frac{1}{1 + e^{-x}}$$

Slide credit: Hugo Larochelle

# Activation - hyperbolic tangent (tanh)

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

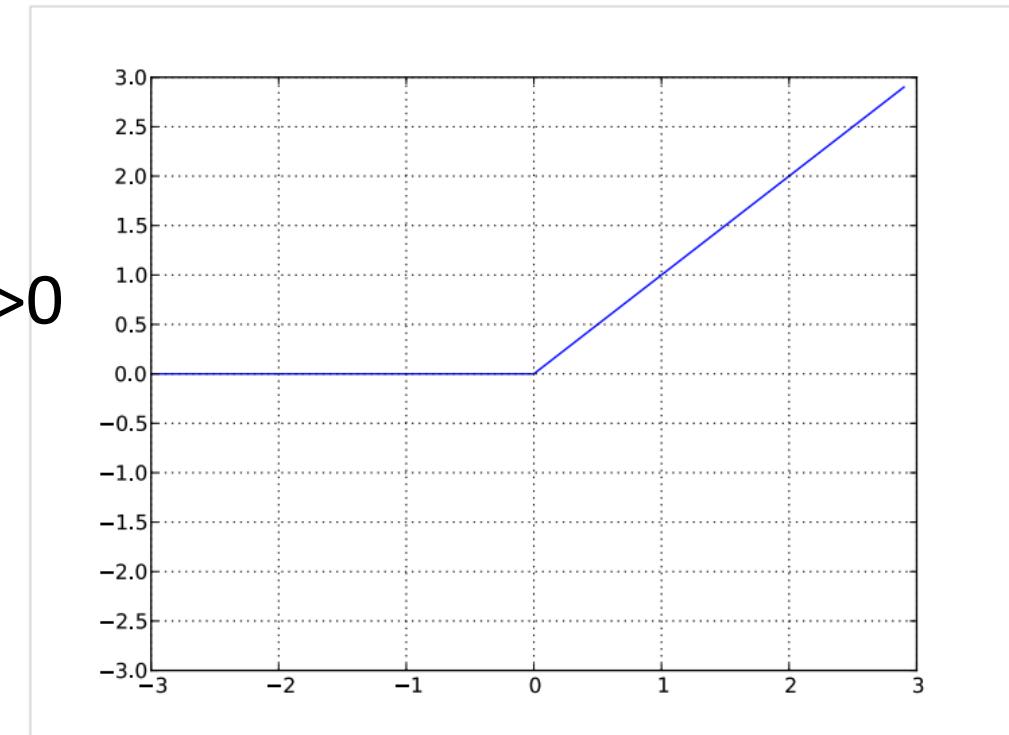
- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing
- Partial derivative  $g'(x) = 1 - g(x)^2$



# Activation - rectified linear(relu)

$$g(x) = \text{relu}(x) = \max(0, x)$$

- It has constant derivative value for all i/p >0
- Computationally efficient as it allows the network to converge very quickly.
- It allows backpropagation.



# SoftMax Function

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^m e^{z_k}}$$

- It is similar to the Sigmoid function, with the only difference being that the output are normalized to sum upto 1.
- The calculated probabilities will be in the range of 0 to 1.
- Able to handle multiclass classification problem.

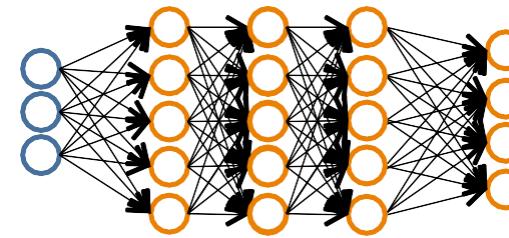
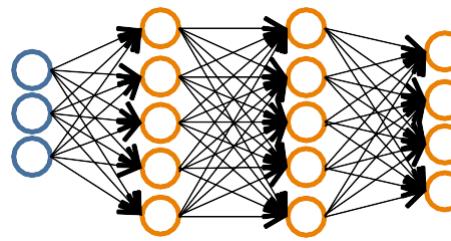
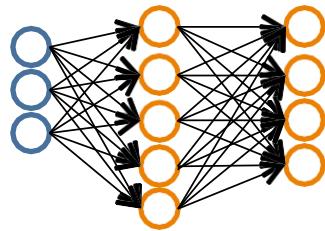
$$z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1.6 \\ 0.55 \\ 0.98 \end{bmatrix}$$

Softmax

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0.51 \\ 0.18 \\ 0.31 \end{bmatrix}$$

# Putting it together

Pick a network architecture



- No. of input units: Dimension of features
- No. output units: Number of classes
- Reasonable default: 1 hidden layer, or if  $>1$  hidden layer, have same no. of hidden units in every layer (usually the more the better)
- Grid search

# Putting it together

## Early stopping

- Use a validation set performance to select the best configuration
- To select the number of epochs, stop training when validation set error increases



# Other tricks of the trade

- Normalizing your (real-valued) data
- Decaying the learning rate
  - as we get closer to the optimum, makes sense to take smaller update steps
- mini-batch
  - can give a more accurate estimate of the risk gradient
- Momentum
  - can use an exponential average of previous gradients

# Dropout

- Dropout is a regularization technique which prevents overfitting of the network.
- During training, a certain number of neurons in the hidden layer is randomly dropped.
- So, training happens on several architecture of NN on different combinations of the neurons.
- Idea: Each hidden unit is set to 0 with probability 0.5

# DEEP LEARNING

- Deep learning is a sub field of Machine Learning that very closely tries to mimic human brain's working using neurons.
- These techniques focus on building Artificial Neural Networks (ANN) using several hidden layers.
- There are variety of deep learning networks such as Multilayer Perceptron ( MLP), Autoencoders (AE), Convolution Neural Network (CNN), Recurrent Neural Network (RNN).

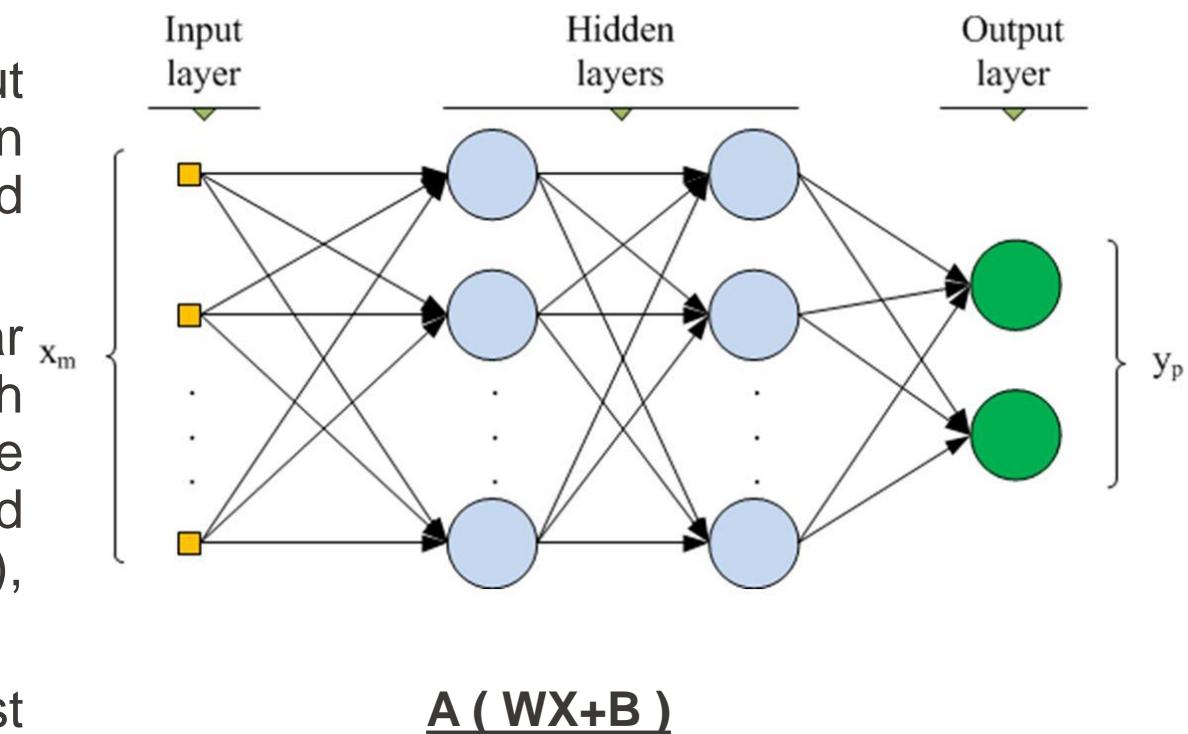
# Why DEEP LEARNING is Growing?

- Processing power needed for Deep learning is readily becoming available using GPUs, Distributed Computing and powerful CPUs
- Moreover, as the data amount grows, Deep Learning models seem to outperform Machine Learning models
- Focus on customization and real time decisioning
- Uncover hard to detect patterns (using traditional techniques). Find latent features (super variables) without significant manual feature engineering.

# Building Blocks of Deep Learning

- **Multilayer Perceptron (MLP)**

- These are the most basic networks and feed forward the inputs to create output.
- They consist of an input layer and an output layer and many interconnected hidden layers and neurons between the input and the output layers.
- They generally use some non linear activation function such as Relu or Tanh and compute the losses (the difference between the true output and computed output) such as Mean Square Error ( MSE), Logloss.
- This loss is backward propagated to adjust the weights and training to minimize the losses or make the models more accurate.



# Autoencoders

- The concept of the AE is quite simple- the input vectors are used to compute the output vectors, but output vectors are same as the input vectors.
- Autoencoders are neural networks that learn to efficiently compress and encode data then learn to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible.
- Therefore, autoencoders reduce the dimensionality of the input data i.e. reducing the number of features that describe input data.

# Convolution Neural Network

- Convolution Neural Networks (CNN) significantly enhances the capabilities of the feed forward network such as MLP by inserting convolution layers.
- They are particularly suitable for spatial data, object recognition and image analysis using multidimensional neurons structures.
- CNNs use convolutions ( a linear operation) rather than matrix multiplication as in MLP
- Typically a CNN will have three stages- convolution stage, detector layer ( non linear activator) and pooling layer

# Convolution Neural Network

- **Convolution Layer-** The most important component in the CNN. The layer has Kernels ( learnable filters) and the input x and y dimensions are convoluted ( dot product) to generate feature map
- **Detector Layer-** The feature maps are passed to this stage using a not linear activation function such as ReLU activation function to accentuate the non linear components of the feature maps
- **Pooling Layer-** A pooling layer such as “max pooling” summarizes (sub-sampling) the responses from several inputs from the previous layer and serves to reduce the size of the spatial representation.

# Filters

- **Filters or Feature detectors** are like weight matrix which we multiply to the input image to generate the feature maps or the activation maps.
- **Feature detectors or filters** help identify different features present in an image like edges, vertical lines, horizontal lines, bends, etc.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

Filter
1 0 1
0 1 0
1 0 1

Filter
4 0 0
0 0 0
0 0 0

- So, Convolution of Filter, reduces the image size

# Filters

INPUT IMAGE						
18	54	51	239	244	188	
55	121	75	78	95	88	
35	24	204	113	109	221	
3	154	104	235	25	130	
15	253	225	159	78	233	
68	85	180	214	245	0	

WEIGHT		
1	0	1
0	1	0
1	0	1

429

INPUT IMAGE						
18	54	51	239	244	188	
55	121	75	78	95	88	
35	24	204	113	109	221	
3	154	104	235	25	130	
15	253	225	159	78	233	
68	85	180	214	245	0	

WEIGHT		
1	0	1
0	1	0
1	0	1

429 505 686 856

INPUT IMAGE						
18	54	51	239	244	188	
55	121	75	78	95	88	
35	24	204	113	109	221	
3	154	104	235	25	130	
15	253	225	159	78	233	
68	85	180	214	245	0	

WEIGHT		
1	0	1
0	1	0
1	0	1

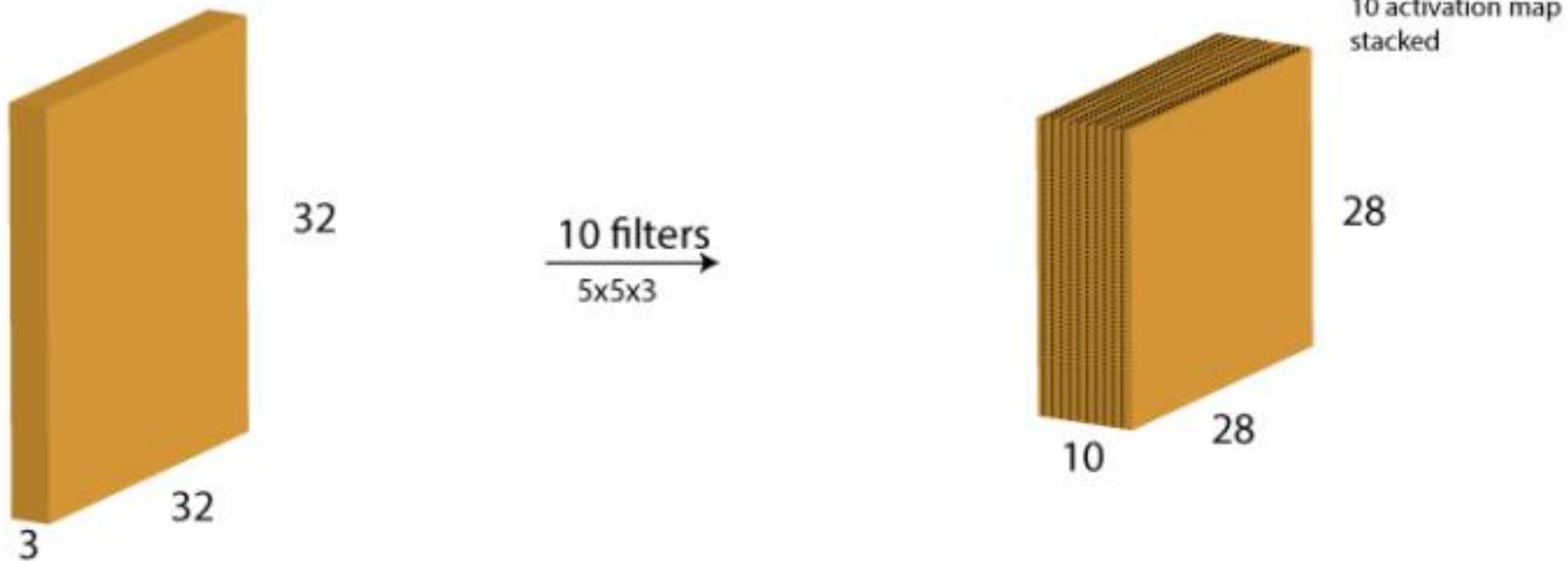
429 505 686 856  
261 792 412 640

INPUT IMAGE						
18	54	51	239	244	188	
55	121	75	78	95	88	
35	24	204	113	109	221	
3	154	104	235	25	130	
15	253	225	159	78	233	
68	85	180	214	245	0	

WEIGHT		
1	0	1
0	1	0
1	0	1

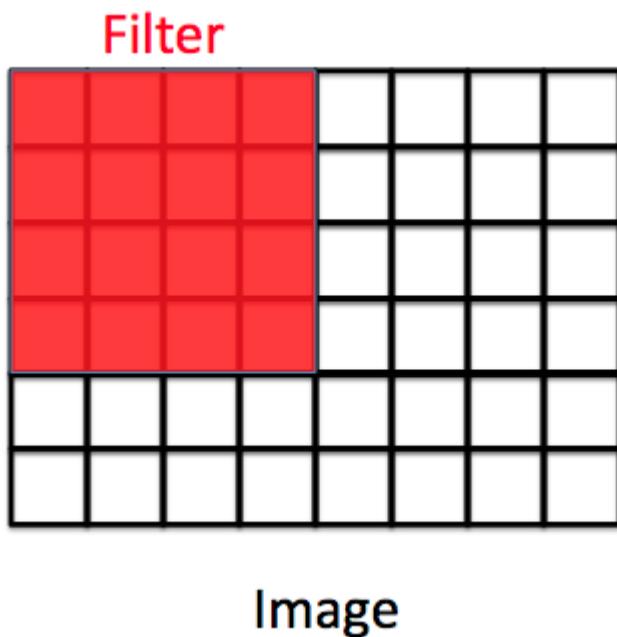
429 505 686 856  
261 792 412 640  
633

# Filters

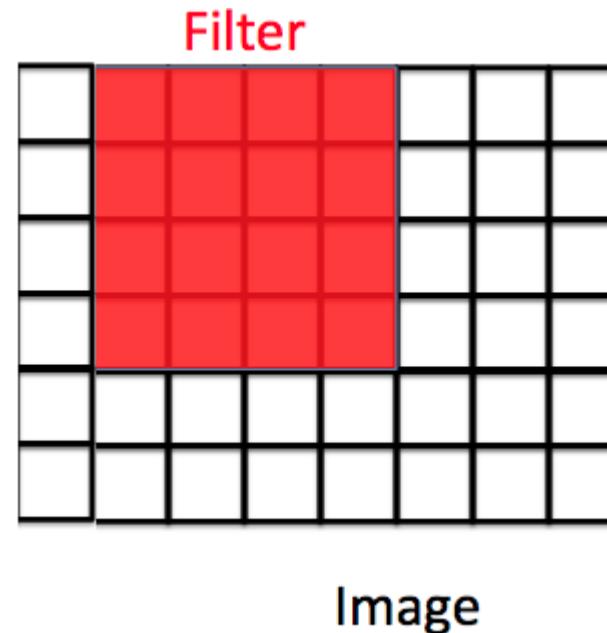


# Filters

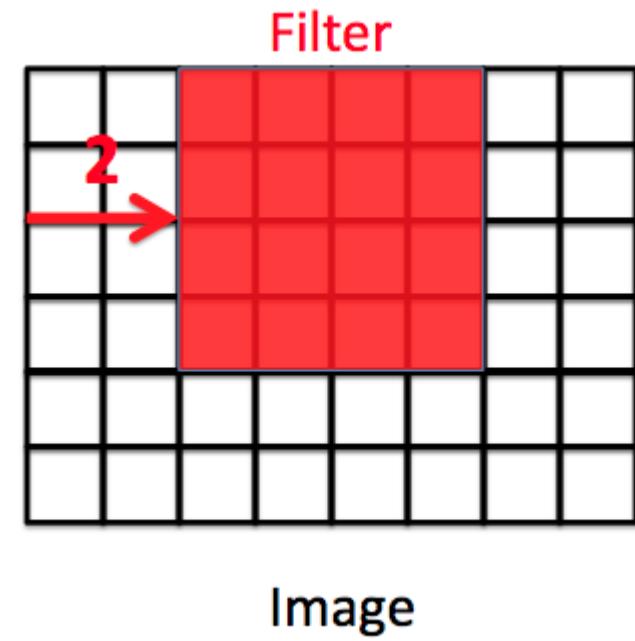
- **Stride:** defines how far the filter moves from one position to the next position by “stride”



If stride = 1, the filter will move one pixels.



If stride = 2, the filter will move two pixels.



# Application of Edge Detection Filters

0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255

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

0	0	255	255	0	0
0	0	255	255	0	0
0	0	255	255	0	0
0	0	255	255	0	0
0	0	255	255	0	0

# Convolution Neural Network

- **Pooling:** This is basically done to reduce the number of parameters to learn and prevents overfitting. Thus, it reduces the amount of computation performed in the network.
- Pooling is then applied over the feature maps for invariance to translation.
- Types of Pooling: **Max Pooling, Min Pooling and Average Pooling**
- Max pooling provides better performance compared to min or average pooling.

# Convolution Neural Network

- Max Pooling:

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

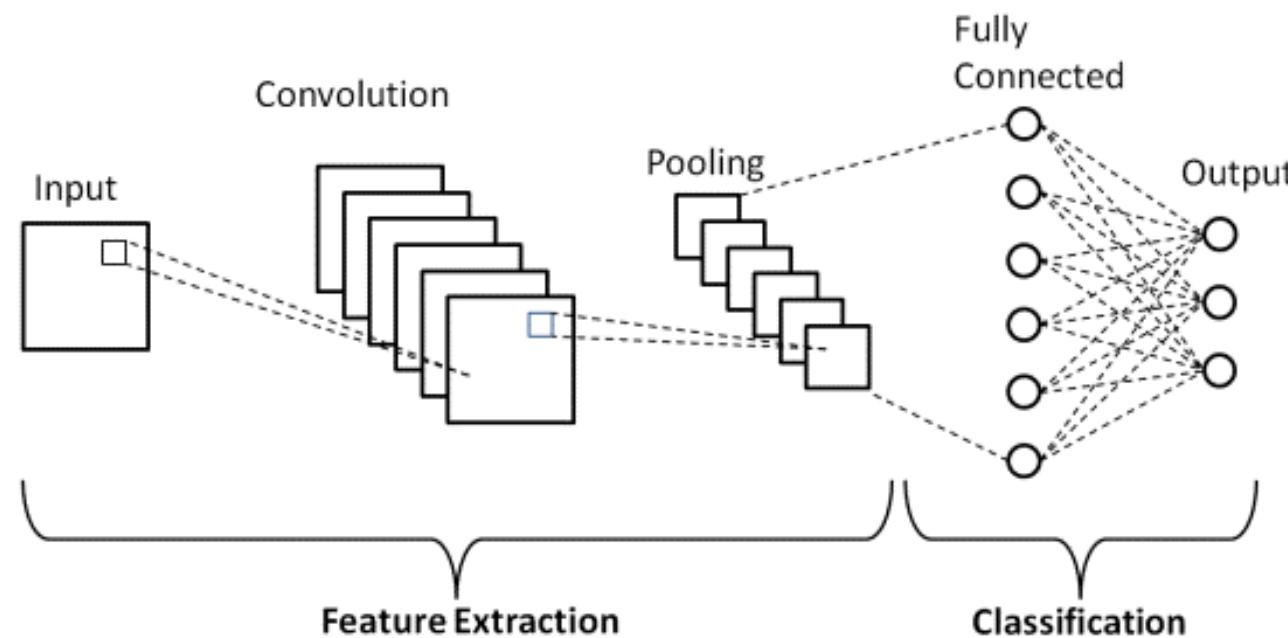
Max Pool  
→

Filter -  $(2 \times 2)$   
Stride -  $(2, 2)$

9	7
8	6

# Fully Connected Layer

- This layer basically takes an input from its preceding layer and outputs an N dimensional vector where N is the number of classes.
- So, it is called flatten all the input and pass these flattened inputs to a deep neural network that outputs the class of the object.



# Example: Convolution Neural Network



# Key points about Convolution layers and Filters

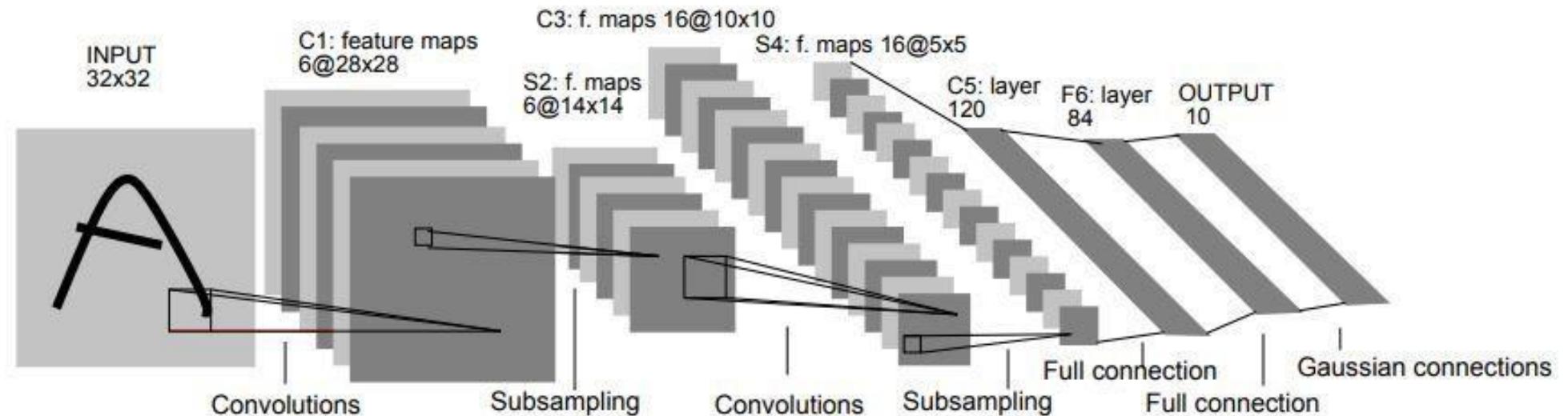
- **The depth of a filter in a CNN must match the depth of the input image.** The number of color channels in the filter must remain the same as the input image.
- **Different Conv2D filters are created for each of the three channels** for a color image.
- **Filters for each layer are randomly initialized based on either Normal or Gaussian distribution.**
- **Initial layers of a convolutional network extract lower-level features from the image, so use fewer filters.** As we build further deeper layers, we increase the number of filters to twice or thrice the size of the filter of the previous layer.
- **Filters of the deeper layers learn more features but are computationally very intensive.**

# LeNet-5 CNN Architecture

- In 1998, the LeNet-5 architecture was introduced in a research paper titled “Gradient-Based Learning Applied to Document Recognition” by Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner.
- It is one of the earliest and most basic CNN architecture.
- **Architecture:**
  - It consists of **7 layers**.
  - The first layer consists of an input image with dimensions of **32x32**.
  - It is convolved with 6 filters of size  $5 \times 5$  resulting in dimension of **28x28x6**.
  - The second layer is a Pooling operation which filter size  $2 \times 2$  and stride of 2. Hence the resulting image dimension will be **14x14x6**.
  - The third layer involves in a convolution operation with 16 filters of size  $5 \times 5$ .
  - The fourth layer is a pooling layer with similar filter size of  $2 \times 2$  and stride of 2. Thus, the resulting image dimension will be reduced to **5x5x16**.

# LeNet-5 CNN Architecture

- The fifth layer is a fully connected convolutional layer with **120** filters each of **size 5x5**. In this layer, each of the 120 units in this layer will be connected to **the 400 (5x5x16)** units from the previous layers.
- The sixth layer is also a fully connected layer with **84** units.
- The final seventh layer will be a **softmax** output layer with ‘n’ possible classes depending upon the number of classes in the dataset.



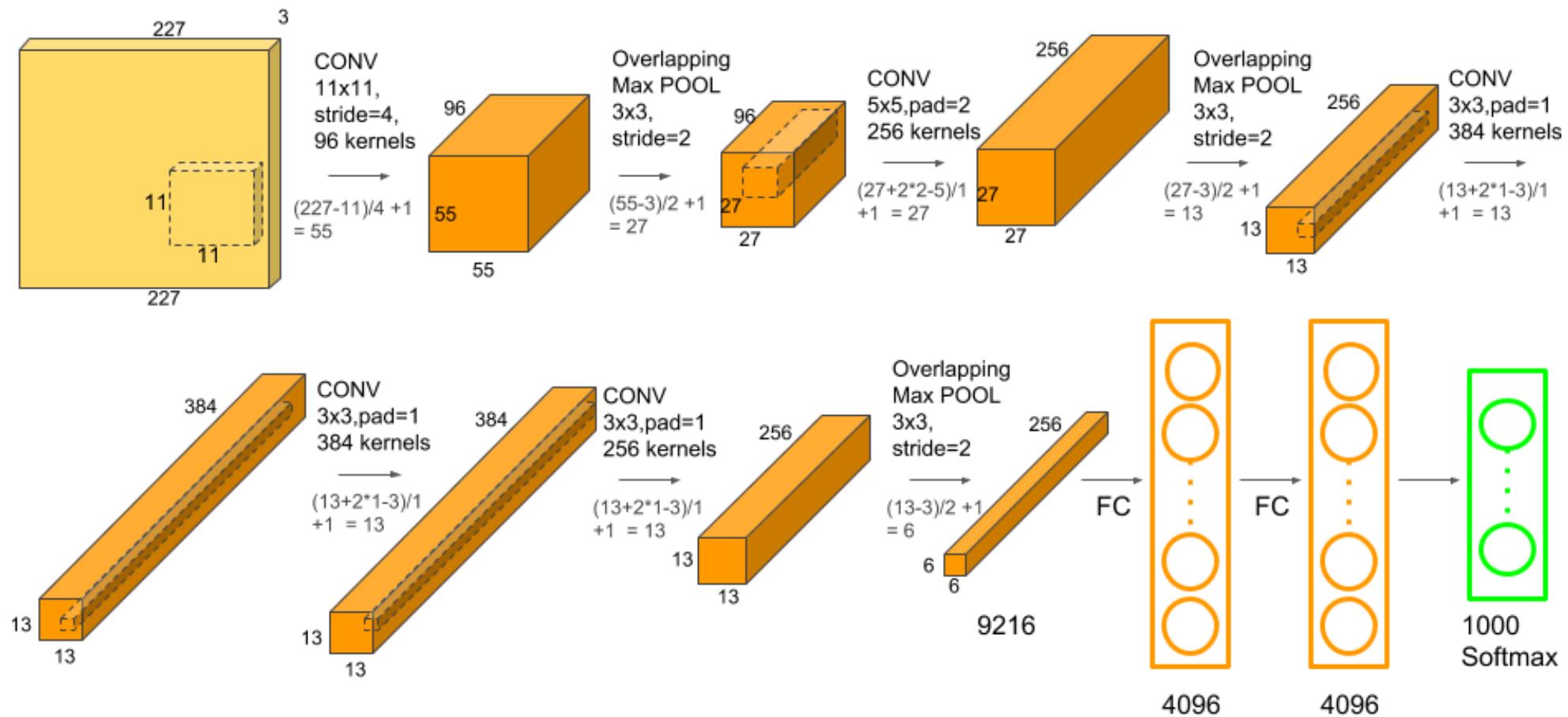
# AlexNet CNN Architecture

- AlexNet is one of the most popular neural network architectures to date. It was proposed by Alex Krizhevsky for the ImageNet Large Scale Visual Recognition Challenge ([ILSVRC](#)), and is based on convolutional neural networks.
- The challenge was to develop a Deep Convolutional Neural Network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 dataset into more than 1000 different categories.
- **Architecture:**
  - It consists of 8 layers in total, out of which the first 5 are convolutional layers and the last 3 are fully-connected.
  - The first two convolutional layers are connected to overlapping max-pooling layers to extract a maximum number of features. The third, fourth, and fifth convolutional layers are directly connected to the fully-connected layers.

# AlexNet CNN Architecture

- All the outputs of the convolutional and fully-connected layers are connected to ReLu non-linear activation function.
- The final output layer is connected to a softmax activation layer, which produces a distribution of 1000 class labels.
- The input dimensions of the network are  $(256 \times 256 \times 3)$ , meaning that the input to AlexNet is an RGB (3 channels) image of  $(256 \times 256)$  pixels.
- There are more than 60 million parameters and 650,000 neurons involved in the architecture.
- To reduce overfitting during the training process, the network uses dropout layers. The neurons that are “dropped out” do not contribute to the forward pass and do not participate in backpropagation. These layers are present in the first two fully-connected layers.

# AlexNet CNN Architecture



# Other CNN Architecture

- VGG16
- VGG19
- RasNet-18
- ResNet-50
- GoogleNet etc.

# Challenges of Deep Learning

- Works better with large amount of data •
- Some models are very hard to train, may take weeks or months
- Overfitting
- Black box and hence may have regulatory challenges

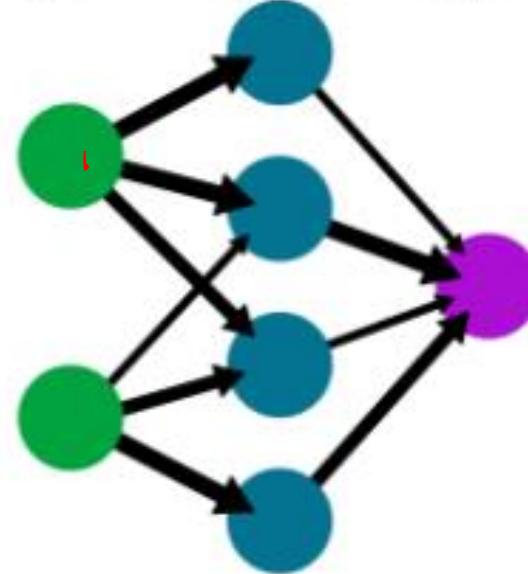
# Backpropagation Algorithm

## WHAT IS BACKPROPAGATION?

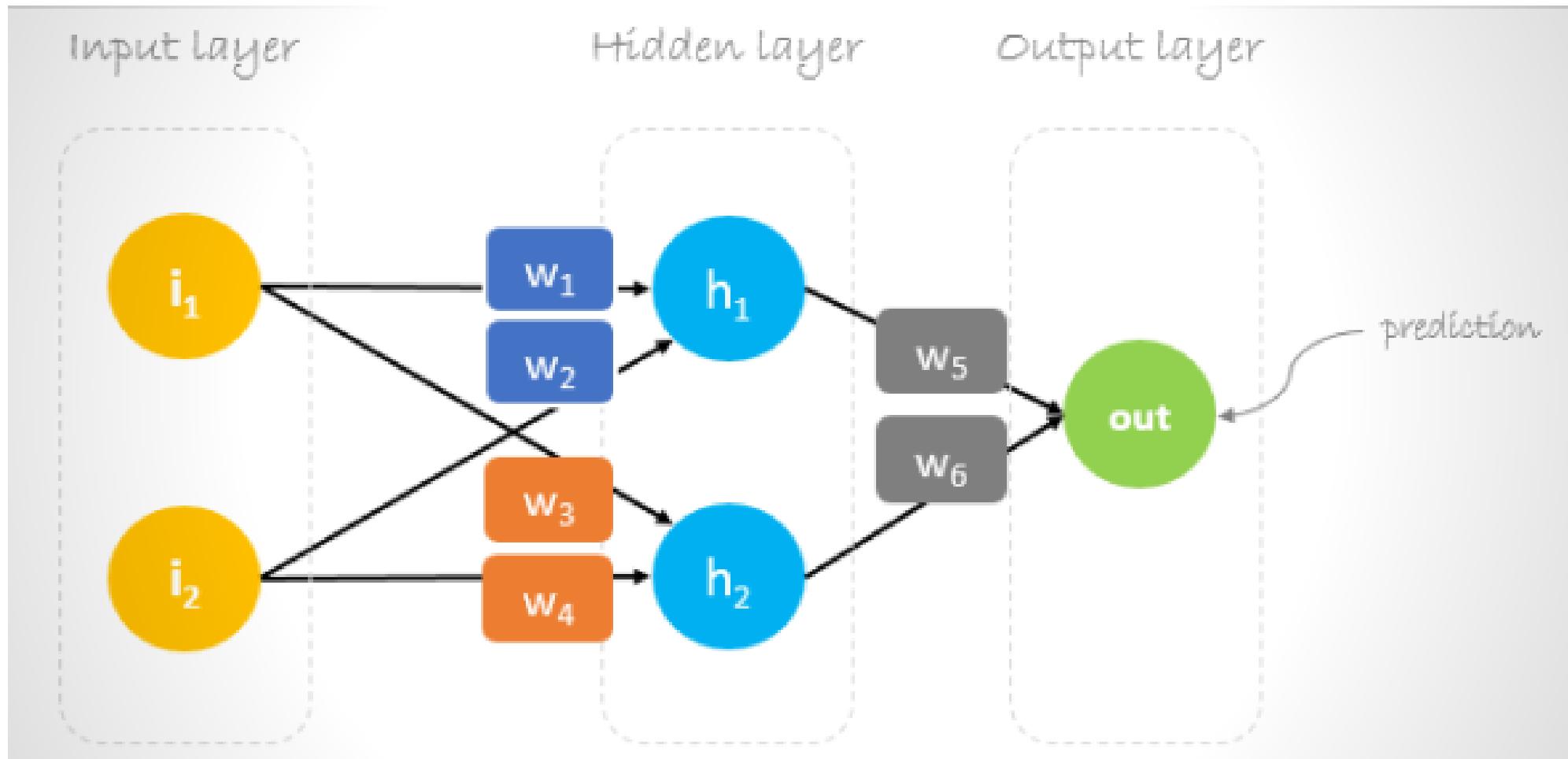
Backpropagation is a supervised learning algorithm, for training Neural Networks.

A simple neural network

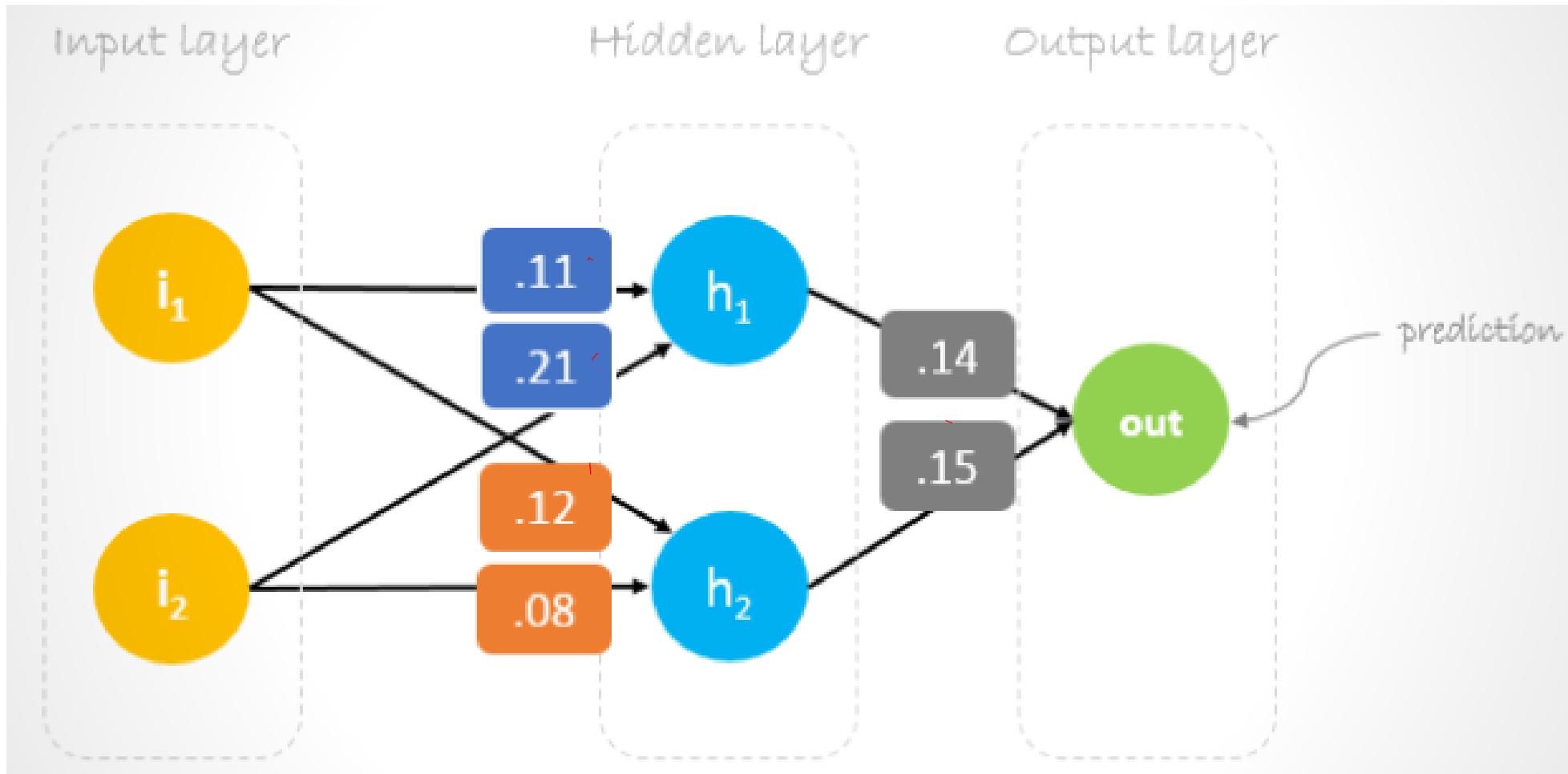
input layer      hidden layer      output layer

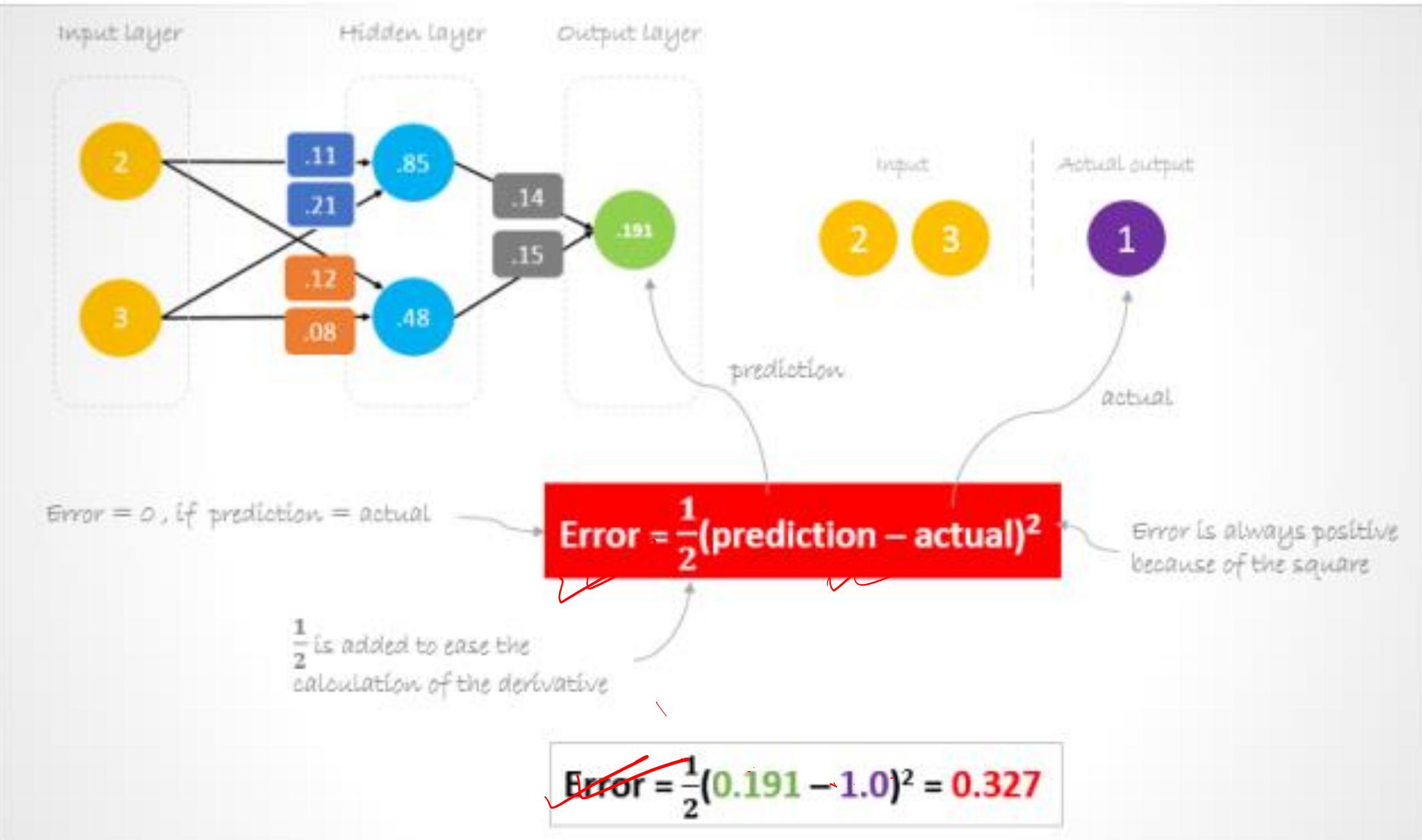


# WHY WE NEED BACKPROPAGATION?



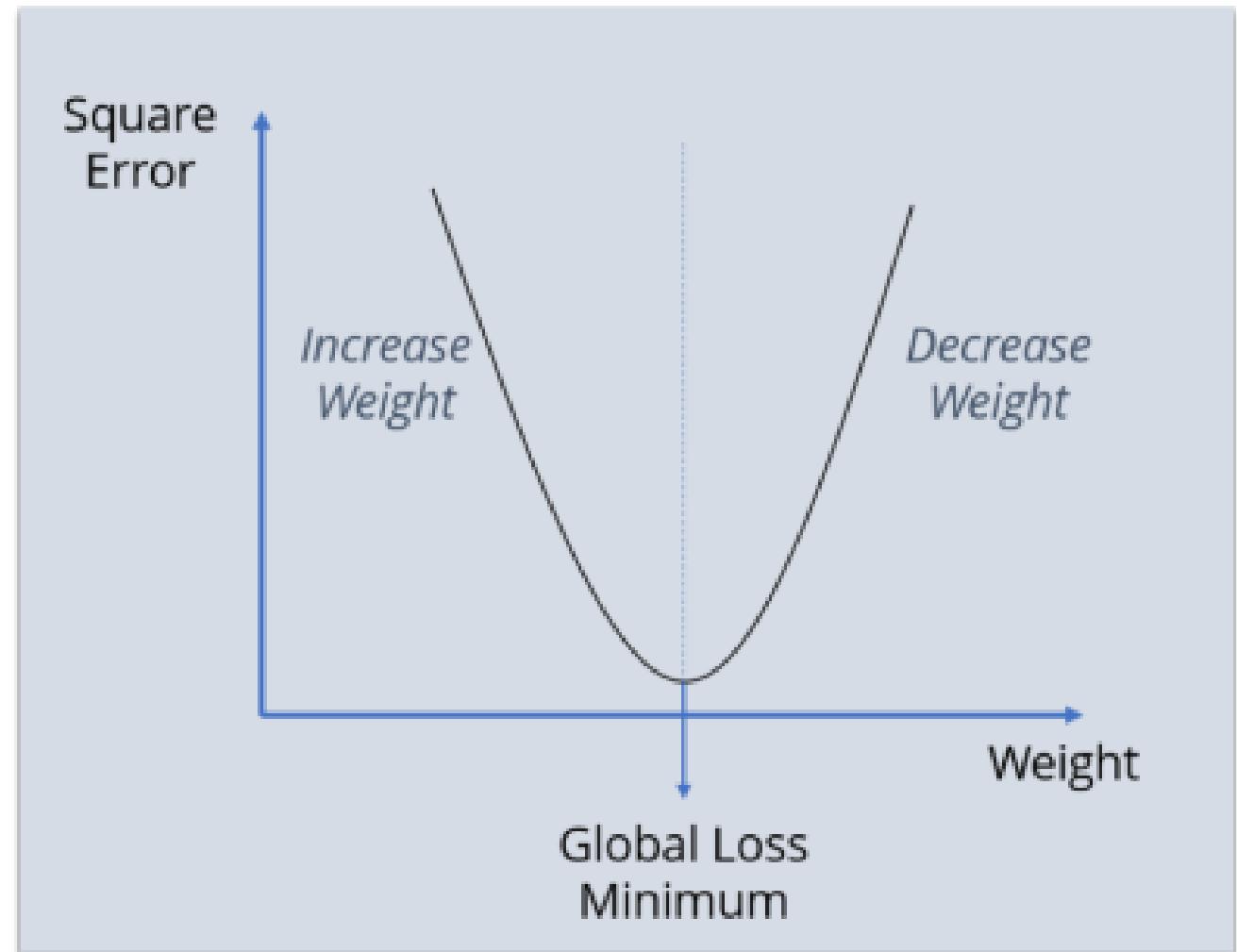
# WHY WE NEED BACKPROPAGATION?



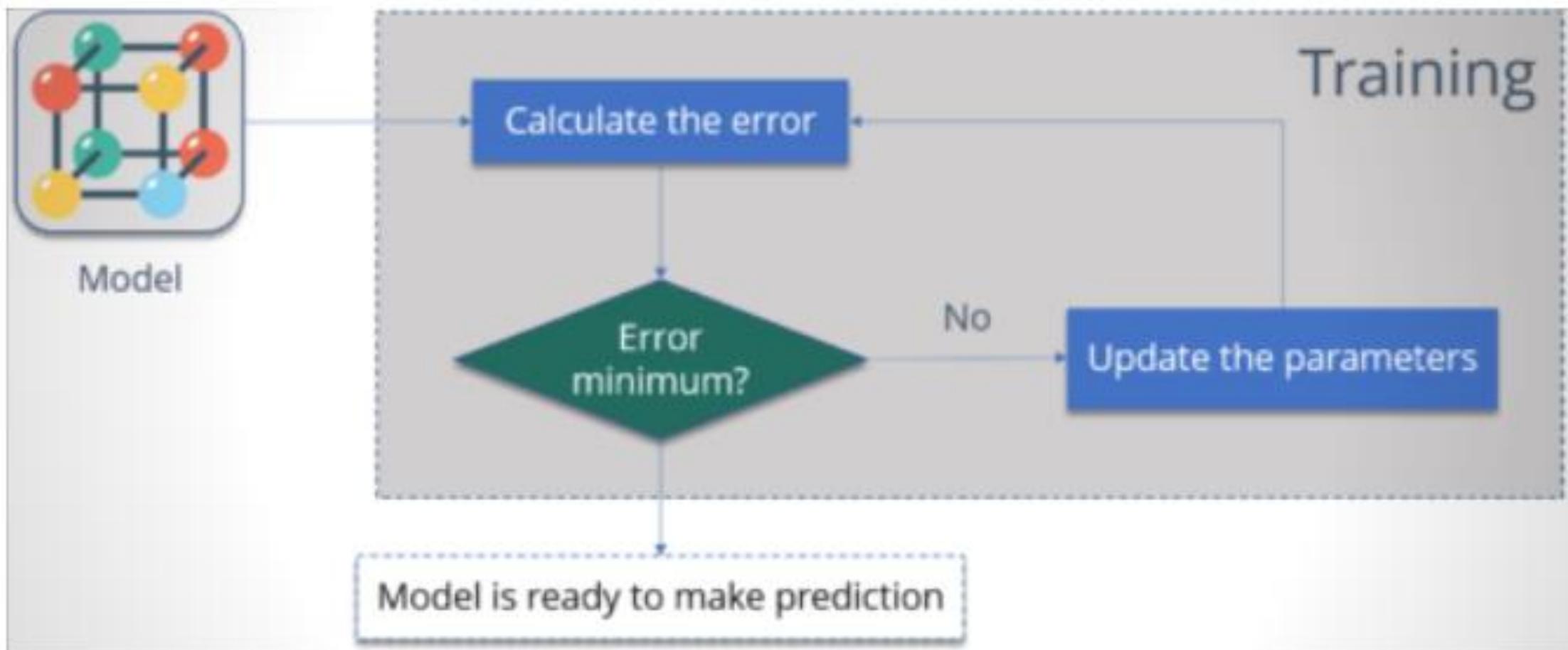


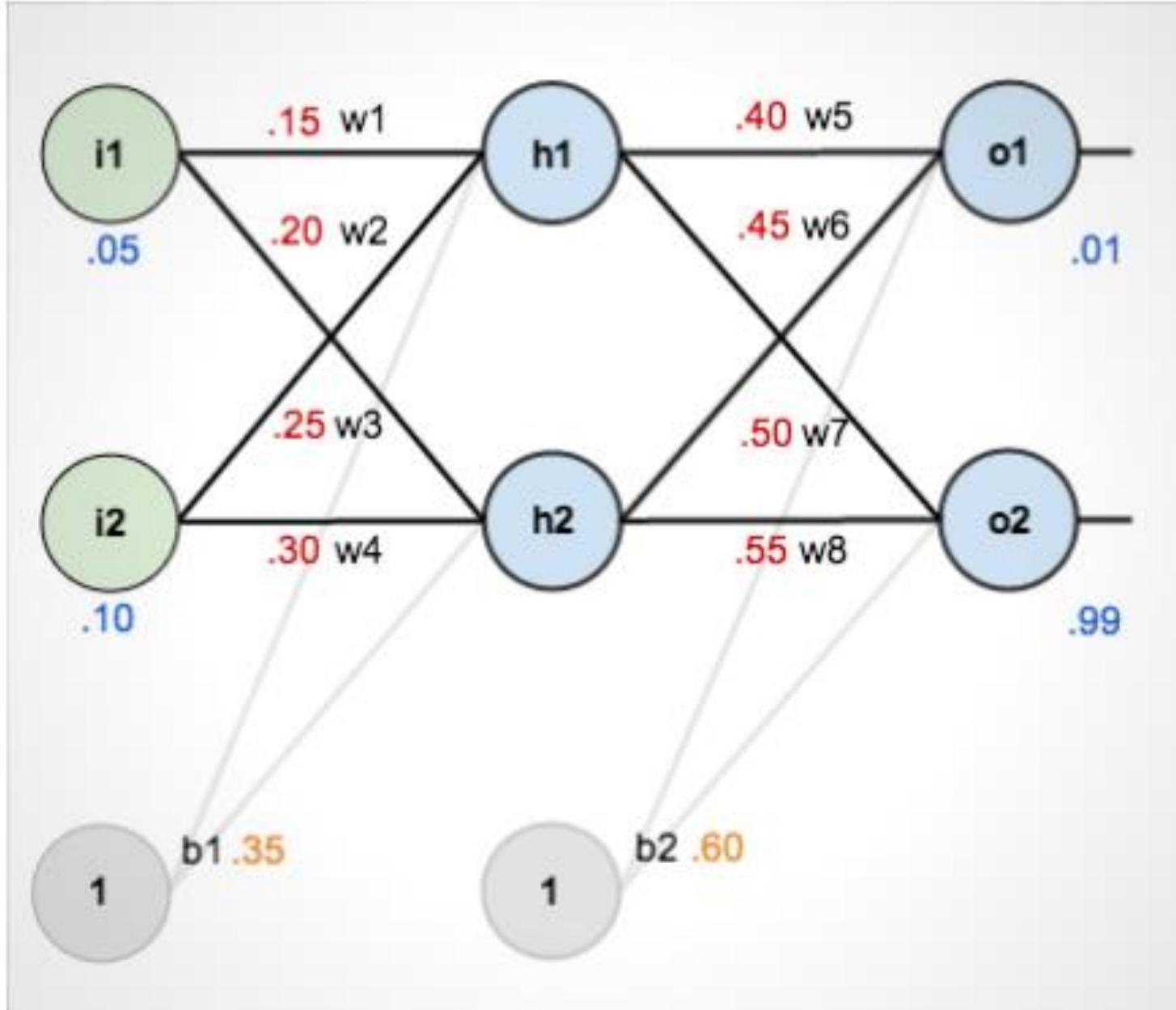
# GRADIENT DESCENT

- Update the weights using gradient descent.
- Gradient descent is used for finding the minimum of a function.
- In our case we want to minimize the error function.



## OVERVIEW:





$$i_1=0.05, i_2=0.10$$

$$w_1=0.15, w_2=0.20$$

$$w_3=0.25, w_4=0.30$$

$$b_1=0.3$$

$$w_5=0.4, w_6=0.45$$

$$w_7=0.5, w_8=0.55$$

$$b_2=0.6$$

$$o_1=0.01, o_2=0.99$$

## FORWARD PASS

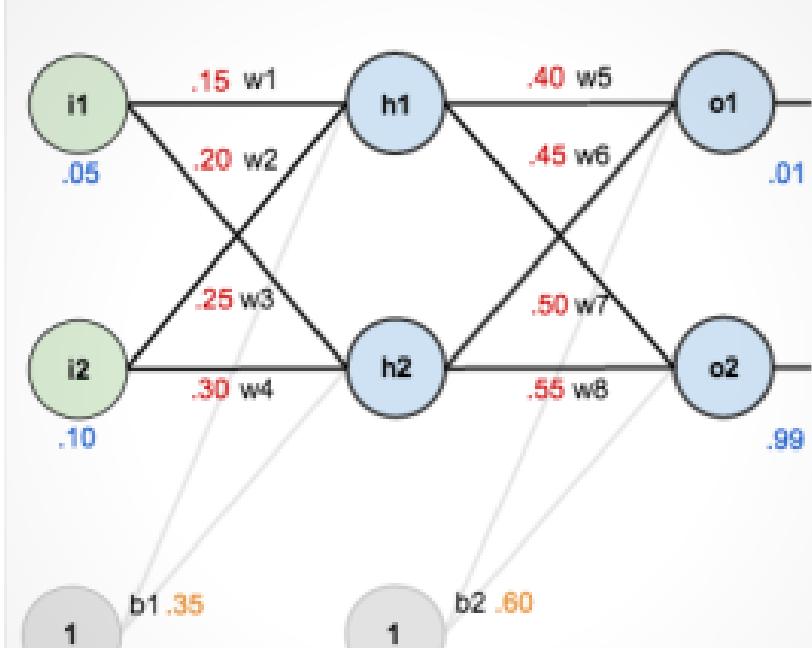
$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

Squash it using logistic function to get output

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$



## FORWARD PASS

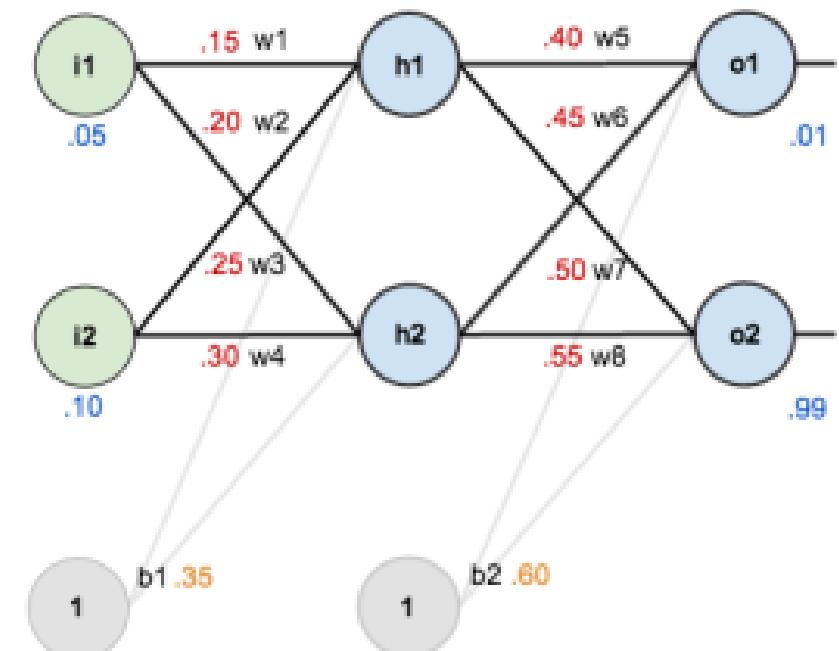
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

And carrying out the same process for  $o_2$  we get:

$$out_{o2} = 0.772928465$$



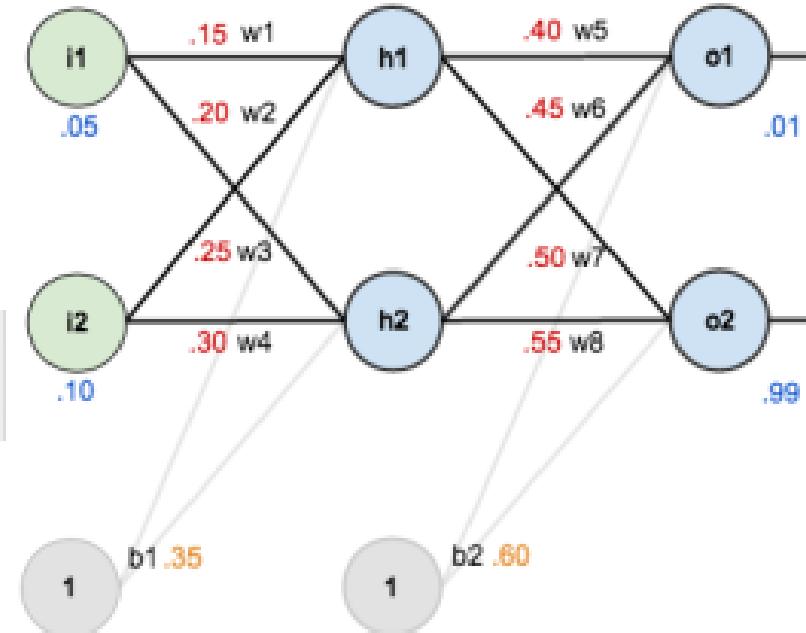
## CALCULATING TOTAL ERROR

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$



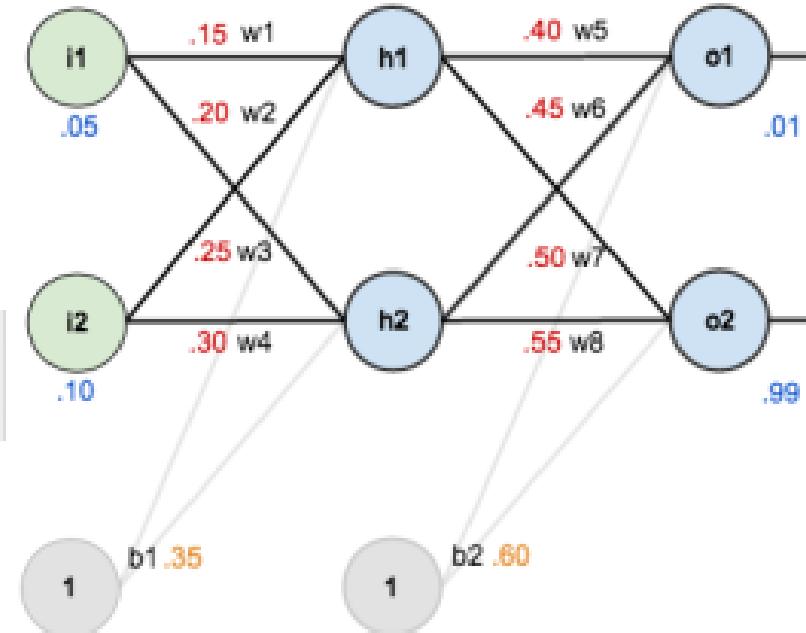
## CALCULATING TOTAL ERROR

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

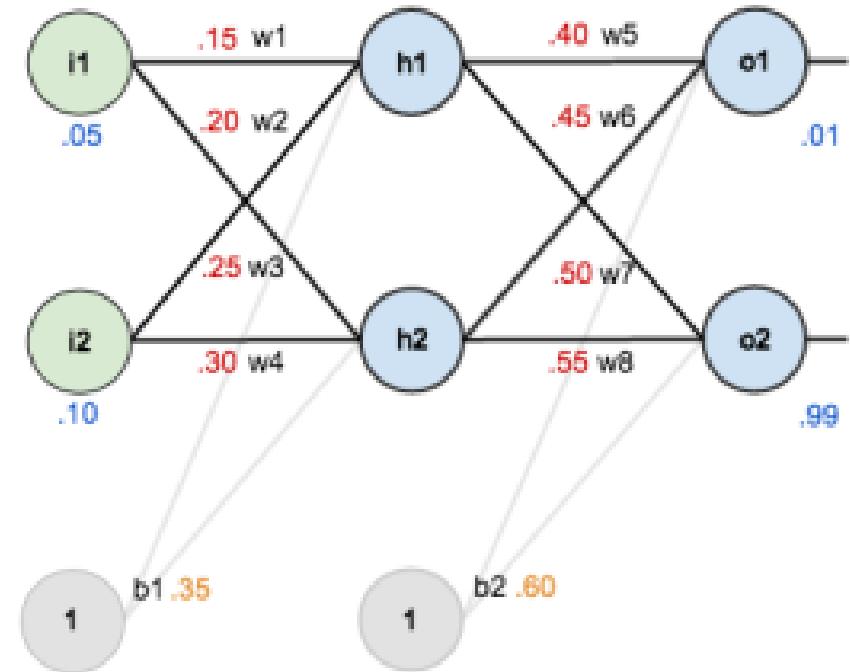
$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$



# THE BACKWARDS PASS

- Update the weights using gradient descent.
- So that they cause the actual output to be closer the target output.
- Thereby minimizing the error for each output neuron and the network as a whole.



## OUTPUT LAYER:

Consider w5.

$\frac{\partial E_{total}}{\partial w_5}$  <sup>total</sup>  
partial derivative of  
E wrt w5.

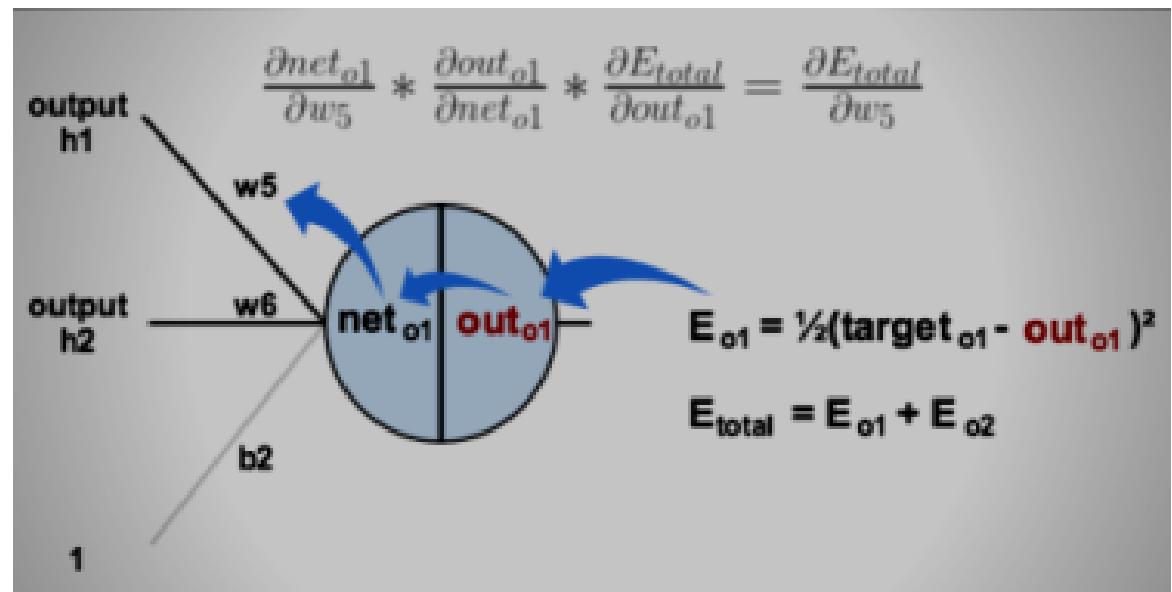
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\delta_{o1} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial net_{o1}}$$

$$E_{total} = E_{o1} + E_{o2} \quad E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} \quad \frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1})$$

$$f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x},$$

$$\frac{d}{dx} f(x) = \frac{e^x \cdot (1 + e^x) - e^x \cdot e^x}{(1 + e^x)^2} = \frac{e^x}{(1 + e^x)^2} = f(x)(1 - f(x)).$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

- Update the weight.
- To decrease the error, we then subtract this value from the current weight.

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$\eta$  := learning rate (eta)

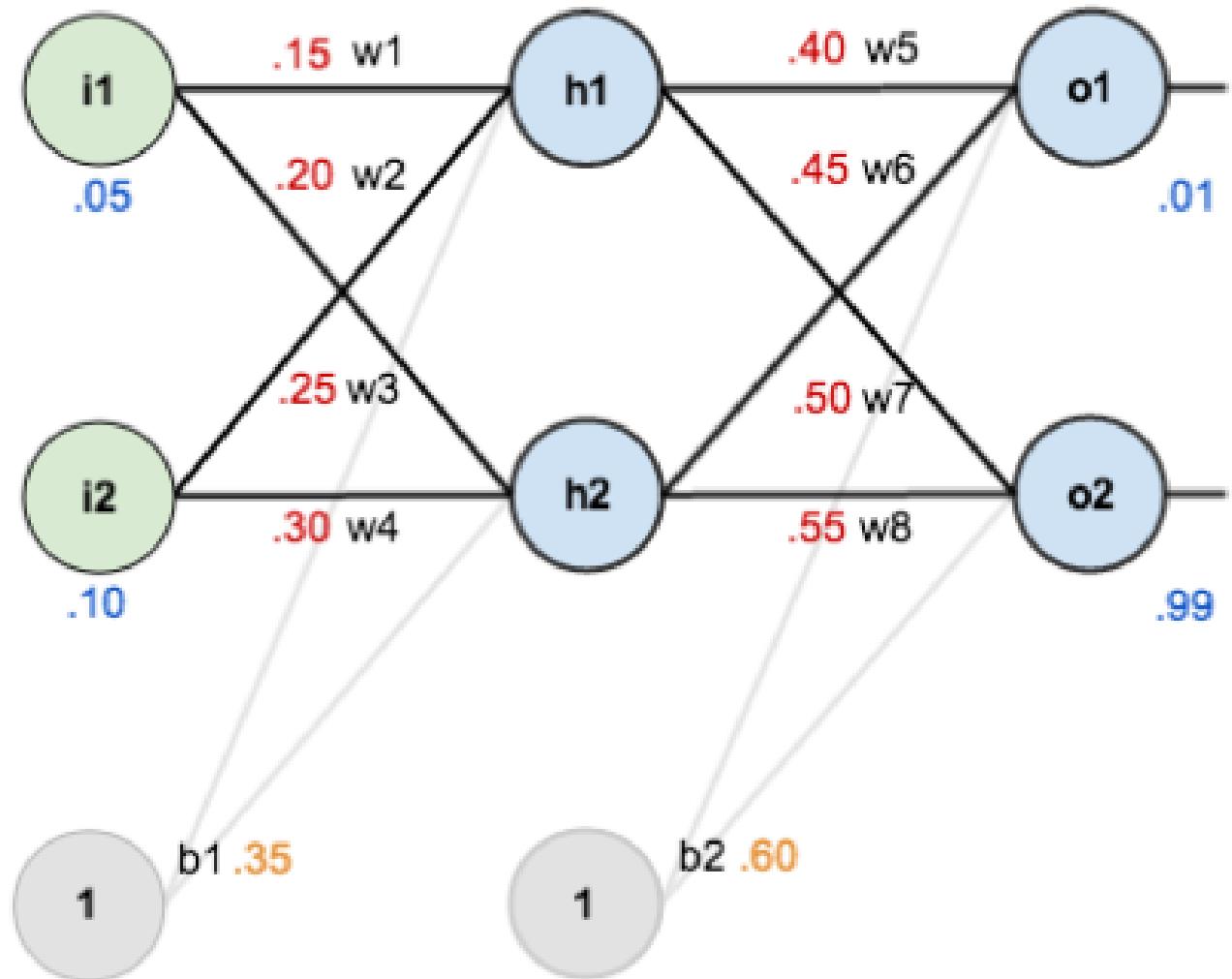
$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$



We perform the *actual* updates in the neural network after we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).



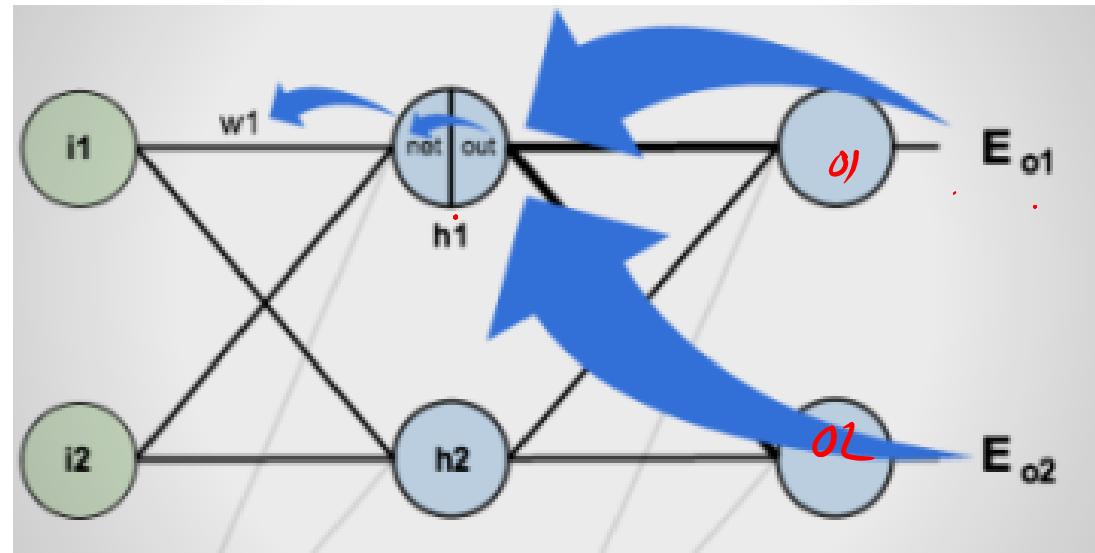
## HIDDEN LAYER:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$



$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}}$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

We can now update  $w_1$ :

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Repeating this for  $w_2$ ,  $w_3$ , and  $w_4$

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

- Finally, we've updated all of our weights!
- When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109.
- After this first round of backpropagation, the total error is now down to 0.291027924.
- It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.0000351085.

**Course code:** CSE3008

**Course Title:** Introduction to Machine Learning

**Module 6: Hidden Markov Model**

**Dr. Usha Rani Gogoi**

**Assistant Professor Sr. Grade 1**

**[usha.gogoi@vitap.ac.in](mailto:usha.gogoi@vitap.ac.in)**

**SCOPE**

# Probabilistic Reasoning

- Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.
- We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.
- **Need of probabilistic reasoning in AI:**
  - When there are unpredictable outcomes.
  - When specifications or possibilities of predicates becomes too large to handle.
  - When an unknown error occurs during an experiment.

# Probabilistic Reasoning

- In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as

"It will rain today,"

"behavior of someone for some situations,"

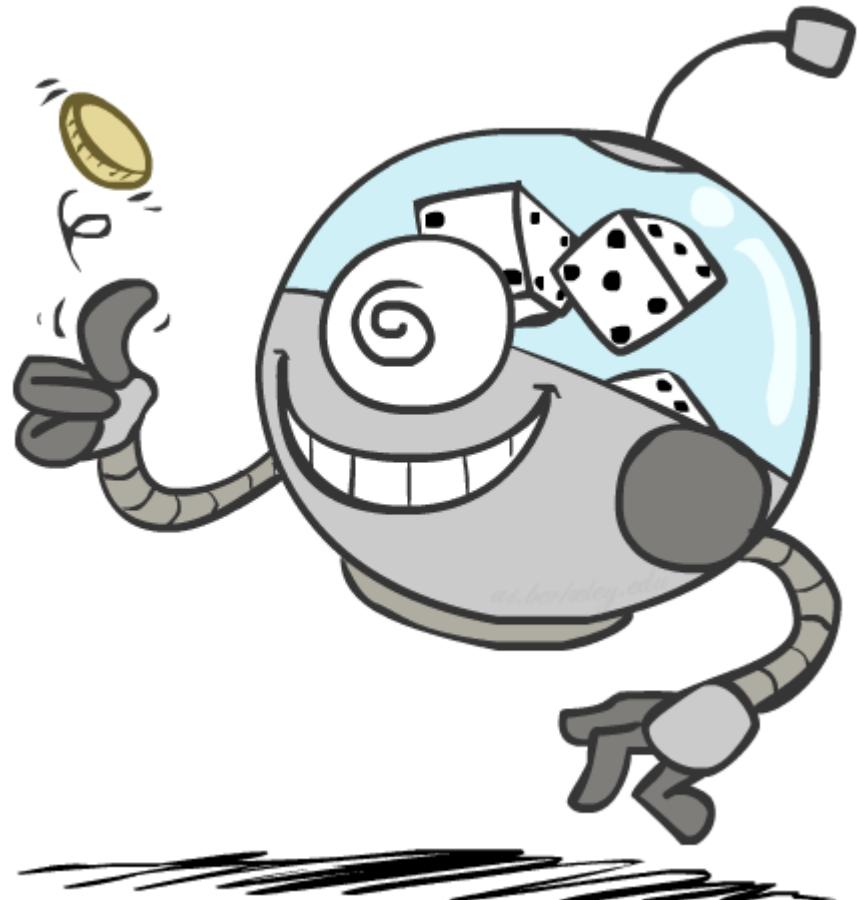
"A match between two teams or two players."

These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

- Diagnosis
- Speech recognition
- Tracking objects
- Robot mapping
- Genetics
- ... lots more!

# We need Knowledge on

- **Probability**
  - Random Variables
  - Joint and Marginal Distributions
  - Conditional Distribution
  - Product Rule, Chain Rule, Bayes' Rule
  - Inference
  - Independence

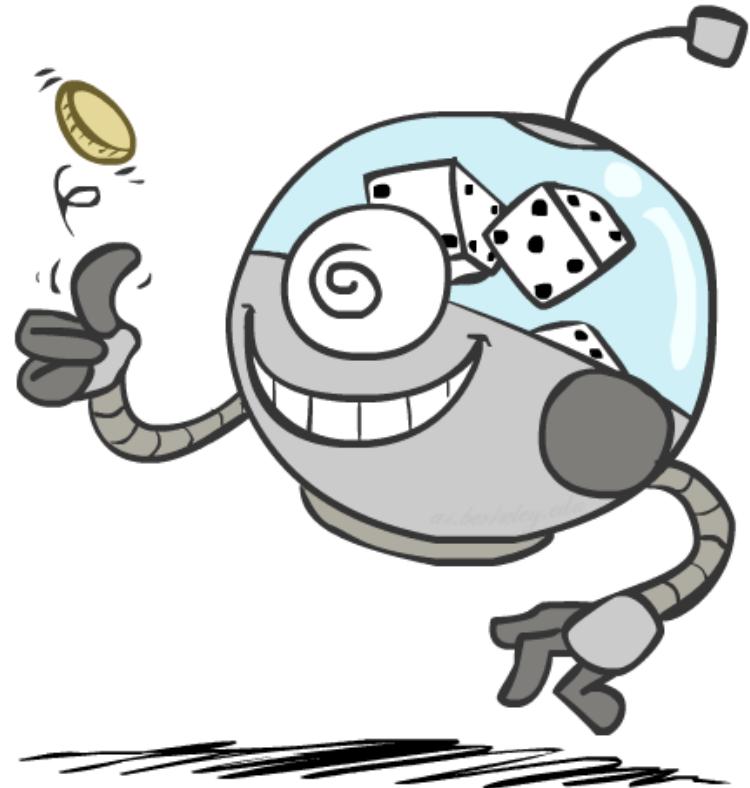


# Uncertainty

- General situation:
  - **Observed variables (evidence):** Agent knows certain things about the state of the variable. That is a observed variable can be directly measured and you have measurements for them in your dataset,(e.g., sensor readings or symptoms or lab tests)
  - **Unobserved variables:** Agent needs to reason about other aspects (e.g. what disease is present, intelligence)
  - **Model:** Agent knows something about how the known variables relate to the unknown variables
- Probabilistic reasoning gives us a framework for managing our beliefs and knowledge

# Random Variables

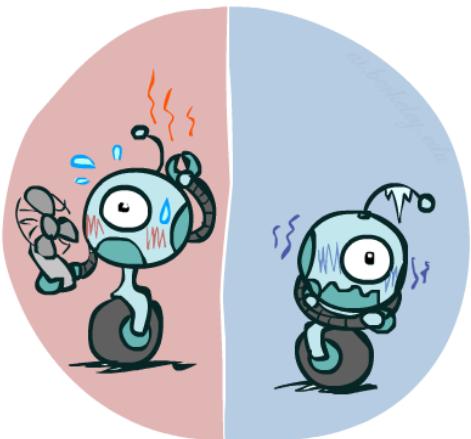
- A random variable is some aspect of the world about which we (may) have uncertainty
  - $R$  = Is it raining?
  - $T$  = Is it hot or cold?
  - $D$  = How long will it take to drive to work?
  - $L$  = Where is the ghost?
- We denote random variables with capital letters
- Random variables have domains
  - $R$  in {true, false} (often write as  $\{+r, -r\}$ )
  - $T$  in {hot, cold}
  - $D$  in  $[0, \infty)$
  - $L$  in possible locations, maybe  $\{(0,0), (0,1), \dots\}$



# Probability Distributions

- Associate a probability with each value

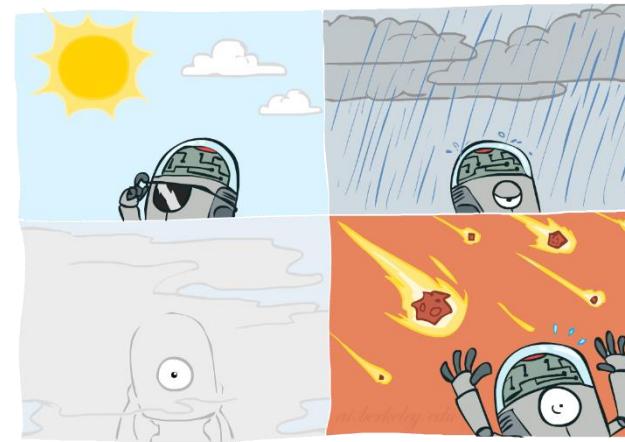
- Temperature:



$P(T)$

T	P
hot	0.5
cold	0.5

- Weather:



$P(W)$

W	P
sun	0.6
rain	0.1
fog	0.3
meteor	0.000 1

# Probability Distributions

- Unobserved random variables have distributions

T	P
hot	0.5
cold	0.5

W	P
sun	0.6
rain	0.1
fog	0.3
meteor	0.0

- A distribution is a TABLE of probabilities of values
- A probability (lower case value) is a single number
- Must have:  $P(W = rain) = 0.1$  and

$$\forall x \ P(X = x) \geq 0$$

$$\sum_x P(X = x) = 1$$

Shorthand notation:

$$P(hot) = P(T = hot),$$

$$P(cold) = P(T = cold),$$

$$P(rain) = P(W = rain),$$

...

OK if all domain entries are unique

# Joint Distributions

- A *joint distribution* over a set of random variables:  $X_1, X_2, \dots, X_n$  specifies a real number for each assignment (or *outcome*):

$$P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$$

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

- Must obey:  $P(x_1, x_2, \dots, x_n) \geq 0$

$$\sum_{(x_1, x_2, \dots, x_n)} P(x_1, x_2, \dots, x_n) = 1$$

$$P(T, W)$$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

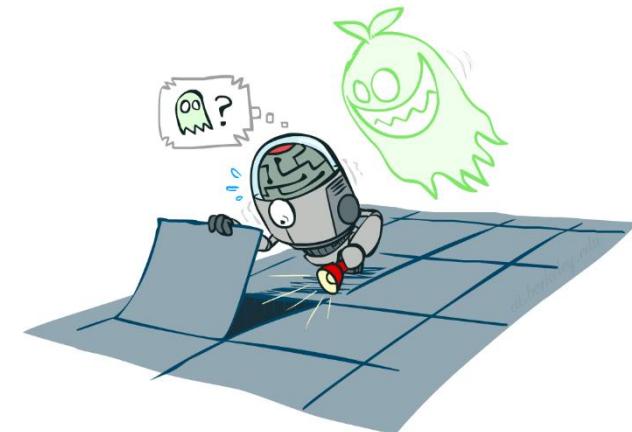
- Size of distribution if  $n$  variables with domain sizes  $d$ ?
  - For all but the smallest distributions, impractical to write out!

# Probabilistic Models

- A probabilistic model is a joint distribution over a set of random variables
- Probabilistic models:
  - (Random) variables with domains
  - Assignments are called *outcomes*
  - Joint distributions: say whether outcomes are likely
  - *Normalized*: sum to 1.0
  - Ideally: only certain variables directly interact
- Constraint satisfaction problems:
  - Variables with domains
  - Constraints: state whether assignments are possible
  - Ideally: only certain variables directly interact

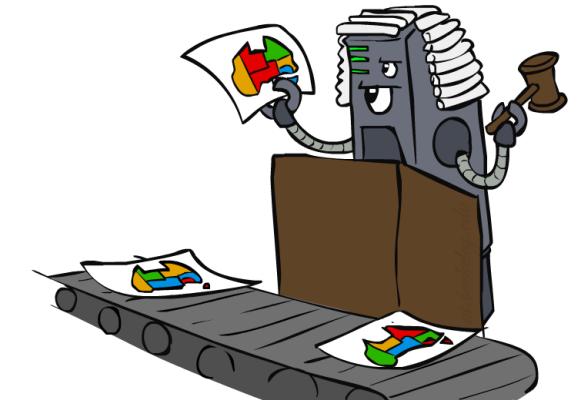
Distribution over T,W

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3



Constraint over T,W

T	W	P
hot	sun	T
hot	rain	F
cold	sun	F
cold	rain	T



# Events

- An *event* is a set  $E$  of outcomes

$$P(E) = \sum_{(x_1 \dots x_n) \in E} P(x_1 \dots x_n)$$

- From a joint distribution, we can calculate the probability of any event
  - Probability that it's hot AND sunny? 0.4
  - Probability that it's hot? 0.5
  - Probability that it's hot OR sunny? 0.7
- Typically, the events we care about are *partial assignments*, like  $P(T=\text{hot})$

$P(T, W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

# Quiz: Events

- $P(+x, +y) ?$

$$P(X, Y)$$

- $P(+x) ?$

- $P(-y \text{ OR } +x) ?$

X	Y	P
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

# Marginal Distributions

- Marginal distributions are sub-tables which eliminate variables
- Marginalization (summing out): Combine collapsed rows by adding

$P(T, W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3



$$P(t) = \sum_s P(t, s)$$



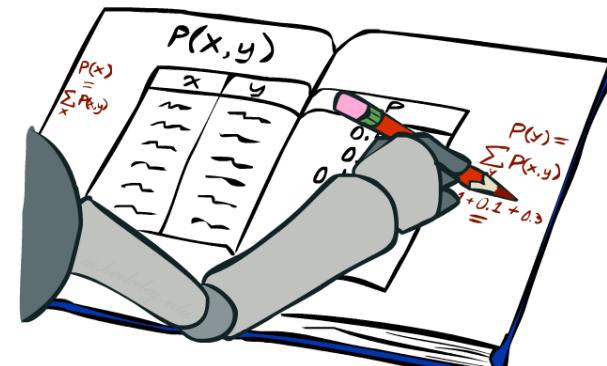
$$P(s) = \sum_t P(t, s)$$

$$P(X_1 = x_1) = \sum_{x_2} P(X_1 = x_1, X_2 = x_2)$$

$P(T)$	
T	P
hot	0.5
cold	0.5

$P(W)$

$P(W)$	
W	P
sun	0.6
rain	0.4



# Quiz: Marginal Distributions

$P(X, Y)$

X	Y	P
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

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

$$P(y) = \sum_x P(x, y)$$

$P(X)$

X	P
+x	
-x	

$P(Y)$

Y	P
+y	
-y	

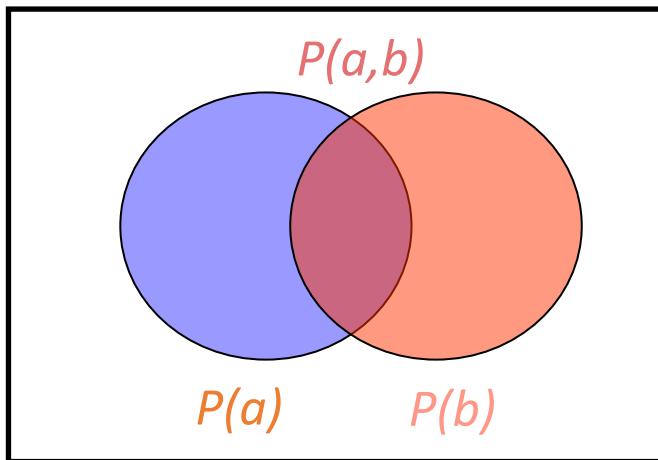
# Conditional Probabilities

- A simple relation between joint and conditional probabilities
  - In fact, this is taken as the *definition* of a conditional probability

$$P(a|b) = \frac{P(a,b)}{P(b)}$$

$P(T, W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3



$$P(W = s | T = c) = \frac{P(W = s, T = c)}{P(T = c)} = \frac{0.2}{0.5} = 0.4$$

$$\begin{aligned} &= P(W = s, T = c) + P(W = r, T = c) \\ &= 0.2 + 0.3 = 0.5 \end{aligned}$$

# Quiz: Conditional Probabilities

$$P(X, Y)$$

- $P(+x \mid +y) ?$

X	Y	P
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

- $P(-x \mid +y) ?$

- $P(-y \mid +x) ?$

# Conditional Distributions

- Conditional distributions are probability distributions over some variables given fixed values of others

Conditional Distributions

$$P(W|T)$$

$P(W T = hot)$	
W	P
sun	0.8
rain	0.2

$P(W T = cold)$	
W	P
sun	0.4
rain	0.6

Joint Distribution

$$P(T, W)$$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

# Problem

- Suppose there is a XYZ cable network that took a survey of 500 subscribers to determine people favourite show?

	Male	Female
BigBoss	80	120
KBC	100	25
Others	50	125

- What is the probability of an XYZ subscriber being Male?
- What is the probability of an XYZ subscriber being Male and preferring KBC?
- Rita just got an XYZ subscription. What is the chance that her favourite show will be Bigboss?

# Normalization Trick

$P(T, W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$\begin{aligned} P(W = s|T = c) &= \frac{P(W = s, T = c)}{P(T = c)} \\ &= \frac{P(W = s, T = c)}{P(W = s, T = c) + P(W = r, T = c)} \\ &= \frac{0.2}{0.2 + 0.3} = 0.4 \end{aligned}$$



$$\begin{aligned} P(W = r|T = c) &= \frac{P(W = r, T = c)}{P(T = c)} \\ &= \frac{P(W = r, T = c)}{P(W = s, T = c) + P(W = r, T = c)} \\ &= \frac{0.3}{0.2 + 0.3} = 0.6 \end{aligned}$$

$P(W|T = c)$

W	P
sun	0.4
rain	0.6

# Normalization Trick

$$\begin{aligned} P(W = s|T = c) &= \frac{P(W = s, T = c)}{P(T = c)} \\ &= \frac{P(W = s, T = c)}{P(W = s, T = c) + P(W = r, T = c)} \\ &= \frac{0.2}{0.2 + 0.3} = 0.4 \end{aligned}$$

$$\begin{aligned} P(W = r|T = c) &= \frac{P(W = r, T = c)}{P(T = c)} \\ &= \frac{P(W = r, T = c)}{P(W = s, T = c) + P(W = r, T = c)} \\ &= \frac{0.3}{0.2 + 0.3} = 0.6 \end{aligned}$$

$P(T, W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

**SELECT** the joint probabilities matching the evidence  
→

$P(c, W)$

T	W	P
cold	sun	0.2
cold	rain	0.3

**NORMALIZE** the selection  
(make it sum to one)

$P(W|T = c)$

W	P
sun	0.4
rain	0.6

Joint distribution  
to

- Why does this work? Sum of selection is  $P(\text{evidence})!$  ( $P(T=c)$ , here)

Conditional distribution

$$P(x_1|x_2) = \frac{P(x_1, x_2)}{P(x_2)} = \frac{P(x_1, x_2)}{\sum_{x_1} P(x_1, x_2)}$$

# Quiz: Normalization Trick

- $P(X | Y=-y) ?$

$P(X, Y)$

X	Y	P
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

**SELECT** the joint probabilities matching the evidence



**NORMALIZE** the selection  
(make it sum to one)



# To Normalize

- (Dictionary) To bring or restore to a normal condition

All entries sum to ONE

- Procedure:

- Step 1: Compute  $Z = \text{sum over all entries}$
- Step 2: Divide every entry by  $Z$

- Example 1

W	P
sun	0.2
rain	0.3

Normalize  Z = 0.5

W	P
sun	0.4
rain	0.6

- Example 2

T	W	P
hot	sun	20
hot	rain	5
cold	sun	10
cold	rain	15

Normalize  Z = 50

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

# Probabilistic Inference

- **Probabilistic inference** compute a desired probability of one or more random variables from other known probabilities (e.g. inferring the conditional probability from joint probability).
- We generally compute conditional probabilities
  - $P(\text{on time} \mid \text{no reported accidents}) = 0.90$
  - These represent the agent's *beliefs* given the evidence
- Probabilities change with new evidence:
  - $P(\text{on time} \mid \text{no accidents, 5 a.m.}) = 0.95$
  - $P(\text{on time} \mid \text{no accidents, 5 a.m., raining}) = 0.80$
  - Observing new evidence causes *beliefs to be updated*

# Inference by Enumeration

- General case:

- Evidence variables:  $E_1 \dots E_k = e_1 \dots e_k$
- Query\* variable:  $Q$
- Hidden variables:  $H_1 \dots H_r$

$$\left. \begin{array}{l} E_1 \dots E_k = e_1 \dots e_k \\ Q \\ H_1 \dots H_r \end{array} \right\} X_1, X_2, \dots, X_n$$

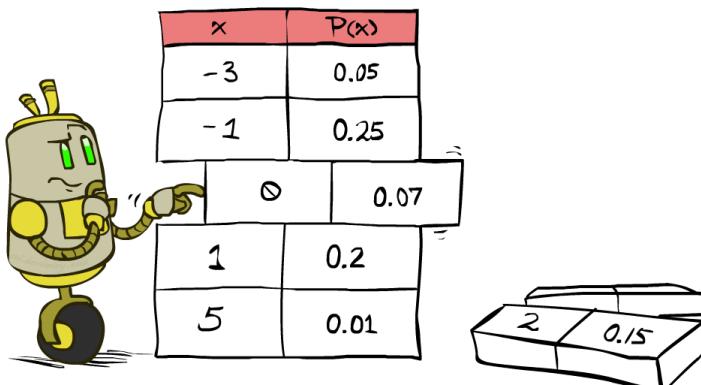
*All variables*

\* Works fine with multiple query variables, too

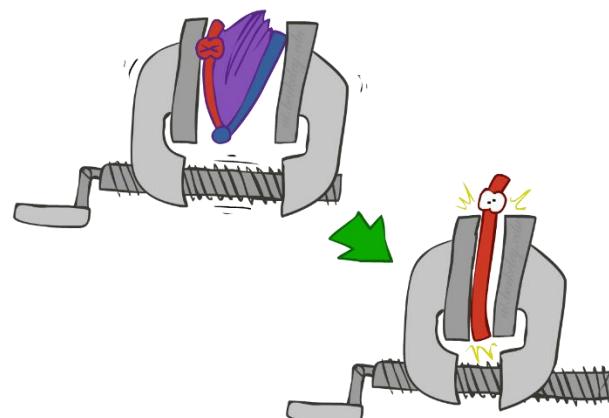
- We want:

$$P(Q|e_1 \dots e_k)$$

- Step 1: Select the entries consistent with the evidence



- Step 2: Sum out H to get joint of Query and evidence



- Step 3: Normalize

$$\times \frac{1}{Z}$$

Note: We get a conditional distribution after normalization which is the actual result

$$P(Q, e_1 \dots e_k) = \sum_{h_1 \dots h_r} \underbrace{P(Q, h_1 \dots h_r, e_1 \dots e_k)}_{X_1, X_2, \dots, X_n}$$

$$P(Q|e_1 \dots e_k) = \frac{1}{Z} P(Q, e_1 \dots e_k)$$

$$Z = \sum_q P(Q, e_1 \dots e_k)$$

# Inference by Enumeration

P(sun)?

S	T	W	P
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

# Inference by Enumeration

$P(\text{sun} \mid \text{winter})?$

S	T	W	P
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

# Inference by Enumeration

$P(\text{sun} \mid \text{winter, hot})?$

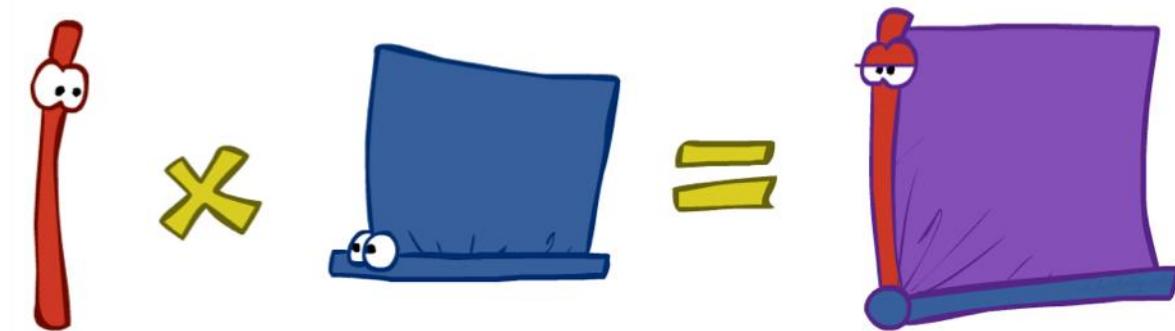
S	T	W	P
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

# The Product Rule

- Sometimes have conditional distributions but want the joint

$$P(y)P(x|y) = P(x, y) \quad \longleftrightarrow$$

$$P(x|y) = \frac{P(x, y)}{P(y)}$$



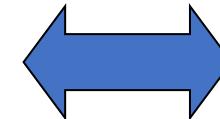
# The Product Rule

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

- Example:

R	P
sun	0.8
rain	0.2

D	W	P
wet	sun	0.1
dry	sun	0.9
wet	rain	0.7
dry	rain	0.3



$$P(D, W)$$

D	W	P
wet	sun	0.08
dry	sun	0.72
wet	rain	0.14
dry	rain	0.06

# The Chain Rule

- More generally, we can always write any joint distribution as an incremental product of conditional distributions

$$P(\underline{x_1}, \underline{\cancel{x_2}}, x_3) = P(x_1)P(x_2|x_1)P(\cancel{x_3}|\underline{x_1}, \underline{x_2})$$

$$P(x_1, x_2, \dots, x_n) = \prod_i P(x_i | x_1 \dots x_{i-1})$$

# Bayes' Rule

- Two ways to factor a joint distribution over two variables:

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

That's my rule!



- Dividing, we get:

$$P(x|y) = \frac{P(y|x)}{P(y)}P(x)$$

# Inference with Bayes' Rule

- Example: Diagnostic probability from causal probability:

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})P(\text{cause})}{P(\text{effect})}$$

- Example:

- M: meningitis, S: stiff neck

$$\left. \begin{array}{l} P(+m) = 0.0001 \\ P(+s|m) = 0.8 \\ P(+s|-m) = 0.01 \end{array} \right\} \text{Example givens}$$

$$P(+m|+s) = \frac{P(+s|m)P(+m)}{P(+s)} = \frac{P(+s|m)P(+m)}{P(+s|m)P(+m) + P(+s|-m)P(-m)} = \frac{0.8 \times 0.0001}{0.8 \times 0.0001 + 0.01 \times 0.999}$$

# Quiz: Bayes' Rule

$$P(D|W)$$

- Given:  $P(W)$

R	P
sun	0.8
rain	0.2

D	W	P
wet	sun	0.1
dry	sun	0.9
wet	rain	0.7
dry	rain	0.3

- What is  $P(W | \text{dry})$  ?

# Probability Recap

- Conditional probability

$$P(x|y) = \frac{P(x,y)}{P(y)}$$

- Product rule

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

- Chain rule

$$\begin{aligned} P(X_1, X_2, \dots, X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\dots \\ &= \prod_{i=1}^n P(X_i|X_1, \dots, X_{i-1}) \end{aligned}$$

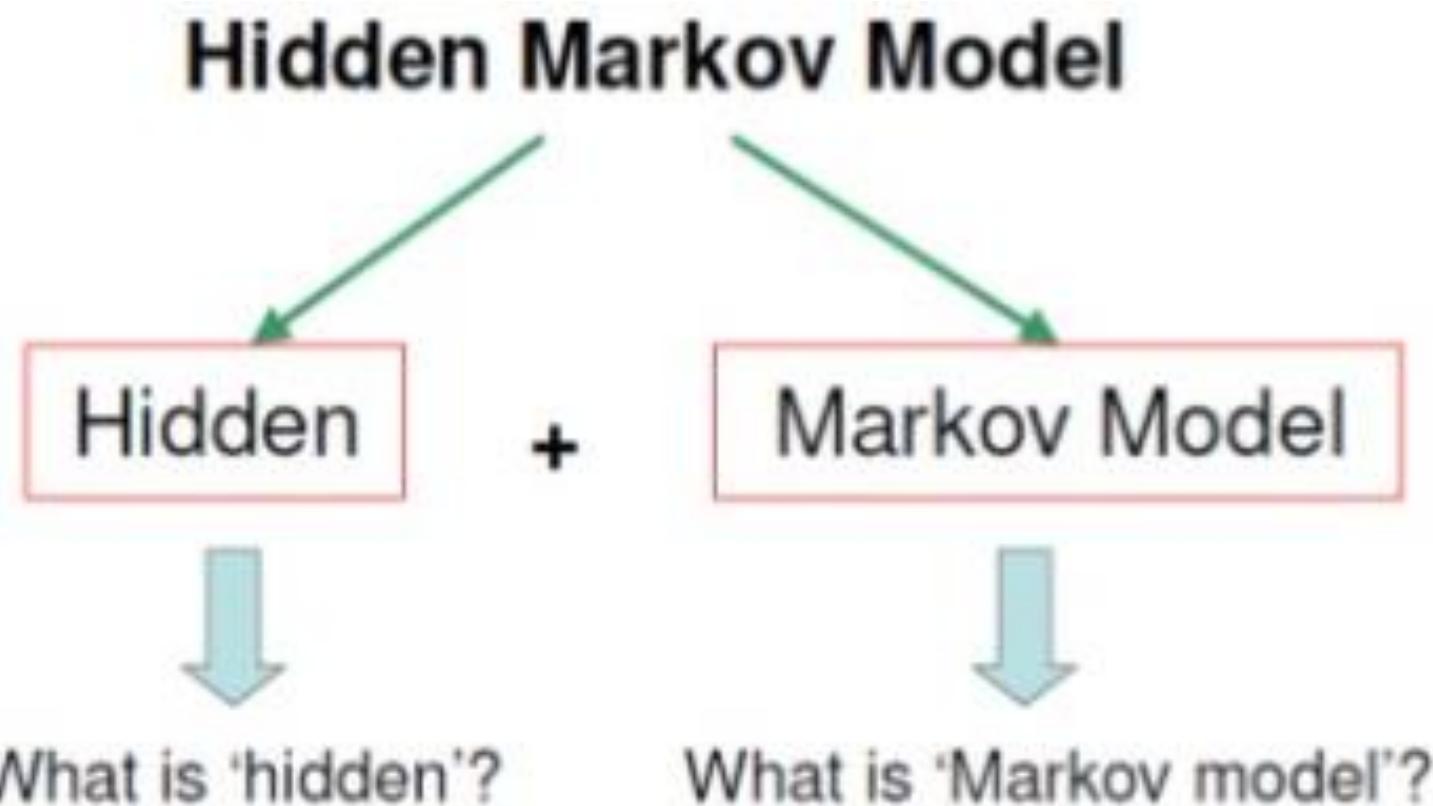
- X, Y independent if and only if:  $\forall x, y : P(x,y) = P(x)P(y)$

- X and Y are conditionally independent given Z if and only if:  $X \perp\!\!\!\perp Y | Z$   
 $\forall x, y, z : P(x,y|z) = P(x|z)P(y|z)$

# Hidden Markov Models

- A Hidden Markov Model is a statistical model, in which the system being modelled is assumed to be a Markov Process (Memory less process: its future and past are independent ) with Hidden States.
- It has a Set of states each of which has limited number of transitions and emissions.
- Each transition between states has an assigned probability.
- Each model starts from the **START** state and ends in **END** state.

# Hidden Markov Models



# Markov Models

- Talk about weather,
- Assume there are three types of weather:

- Sunny,



- Rainy,



- Foggy.



- Weather prediction is about what would be the weather tomorrow,
  - Based on the observations on the past.

# Markov Models

- Weather at day n is  $q_n \in \{\text{sunny}, \text{rainy}, \text{foggy}\}$



- $q_n$  depends on the known weathers of the past days ( $q_{n-1}, q_{n-2}, \dots$ )
- We want to find that:

$$P(q_n | q_{n-1}, q_{n-2}, \dots, q_1)$$

- means given the past weathers what is the probability of any possible weather of today.

# Markov Models

- For example:
  - if we knew the weather for last three days was:



- the probability that tomorrow would be is:

$$P(q_4 = \text{cloudy} | q_3 = \text{rainy}, q_2 = \text{sunny}, q_1 = \text{sunny})$$

- **Markow Models and Assumption (cont.):**
  - Therefore, make a simplifying assumption **Markov assumption**:

# Markov Models

- For sequence:  $\{q_1, q_2, \dots, q_n\}$

$$P(q_n | q_{n-1}, q_{n-2}, \dots, q_1) = P(q_n | q_{n-1})$$

- the weather of tomorrow only depends on today**  
(first order Markov model)

Today's weather	Tomorrow's weather		
	0.8	0.05	0.15
	0.2	0.6	0.2
	0.2	0.3	0.5

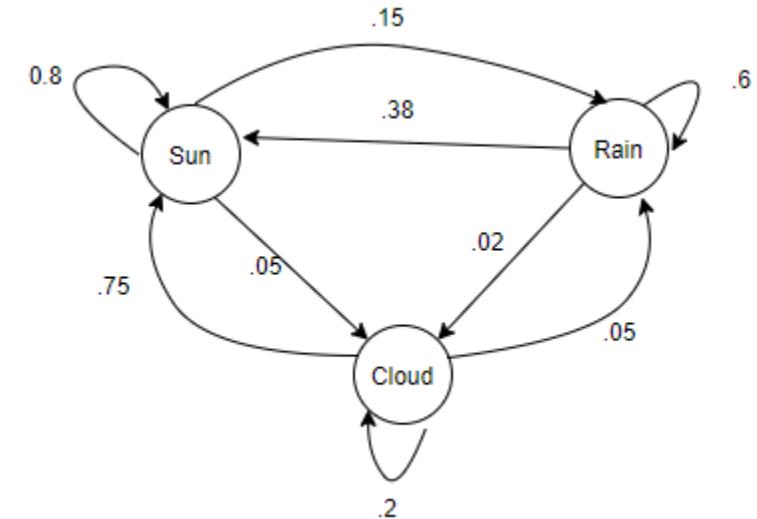
# Markov Models

- State Space = {S, R, C}
- Initial State Distribution = {0.7, 0.25, 0.05}
- Q1. Given that today the weather is sunny, what is the probability that tomorrow is sunny and day after tomorrow is rainy.

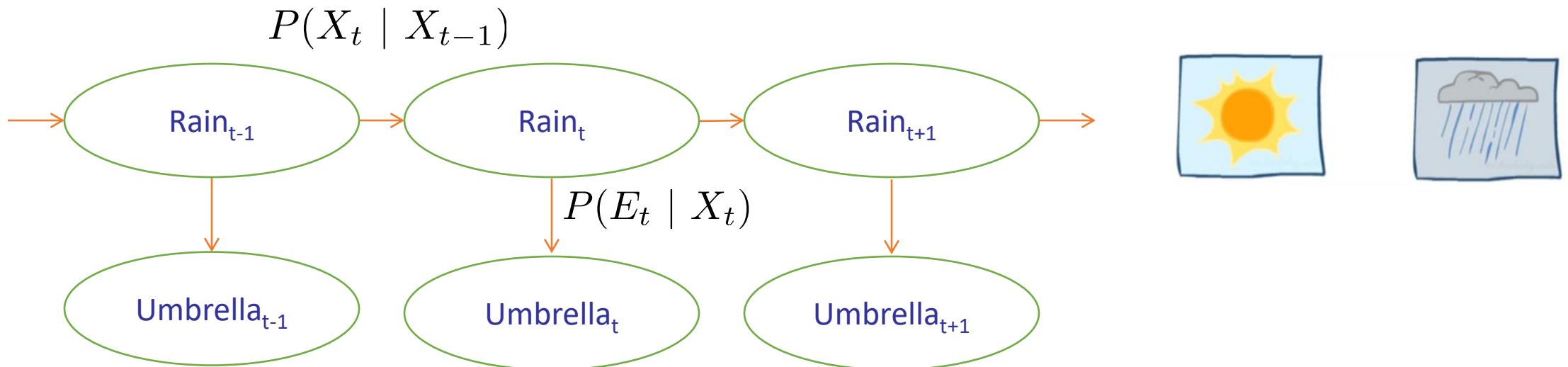
**Ans:  $P(t3 = \text{Rainy}, t2 = \text{Sunny} | t1 = \text{sunny}) = P(t3 = \text{Rainy} | t2=\text{Sunny}) * P(t2=\text{sunny} | t1=\text{sunny})$**

- Q2. If today is cloudy and Yesterday was rainy, what is the probability tomorrow would be sunny?

**Ans:  $P(t3 = \text{Sunny} | t2 = \text{cloudy} | t1 = \text{rainy}) = P(t3 = \text{sunny} | t2=\text{cloudy}) * P(t2=\text{cloudy} | t1=\text{rainy})$**



# Example: Weather HMM



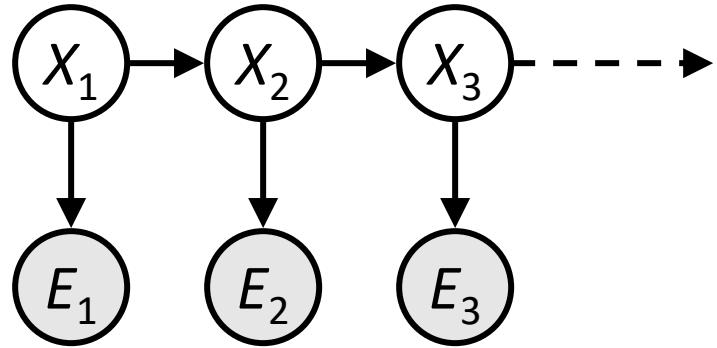
- An HMM is defined by:
  - Initial distribution:
  - Transitions:
  - Emissions:

$$\begin{aligned} &P(X_1) \\ &P(X_t \mid X_{t-1}) \\ &P(E_t \mid X_t) \end{aligned}$$

$R_t$	$R_{t+1}$	$P(R_{t+1} \mid R_t)$
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

$R_t$	$U_t$	$P(U_t \mid R_t)$
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

# Joint Distribution of an HMM

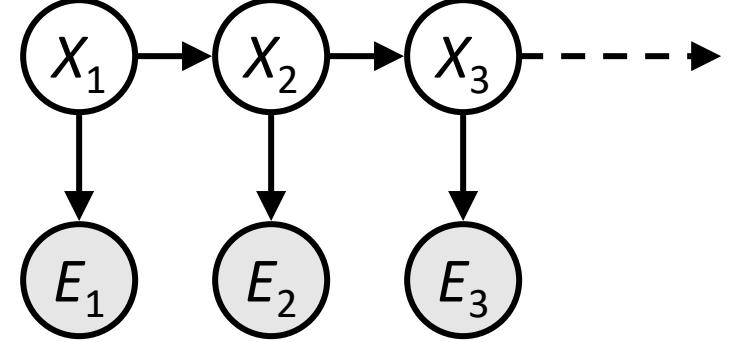


- **Joint distribution:**

$$P(X_1, E_1, X_2, E_2, X_3, E_3) = P(X_1)P(E_1|X_1)P(X_2|X_1)P(E_2|X_2)P(X_3|X_2)P(E_3|X_3)$$

- More generally:

$$P(X_1, E_1, \dots, X_T, E_T) = P(X_1)P(E_1|X_1) \prod_{t=2}^T P(X_t|X_{t-1})P(E_t|X_t)$$



# Chain Rule and HMMs

- From the chain rule, *every* joint distribution over  $X_1, E_1, X_2, E_2, X_3, E_3$  can be written as:

$$\begin{aligned}
 P(X_1, E_1, X_2, E_2, X_3, E_3) = & P(X_1)P(E_1|X_1)P(X_2|X_1, E_1)P(E_2|X_1, E_1, X_2) \\
 & P(X_3|X_1, E_1, X_2, E_2)P(E_3|X_1, E_1, X_2, E_2, X_3)
 \end{aligned}$$

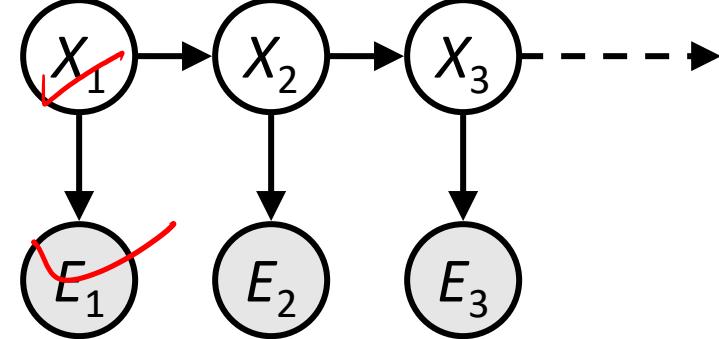
- Assuming* that

$$X_2 \perp\!\!\!\perp E_1 \mid X_1, \quad E_2 \perp\!\!\!\perp X_1, E_1 \mid X_2, \quad X_3 \perp\!\!\!\perp X_1, E_1, E_2 \mid X_2, \quad E_3 \perp\!\!\!\perp X_1, E_1, X_2, E_2 \mid X_3$$

gives us the expression posited on the previous slide:

$$P(X_1, E_1, X_2, E_2, X_3, E_3) = P(X_1)P(E_1|X_1)P(X_2|X_1)P(E_2|X_2)P(X_3|X_2)P(E_3|X_3)$$

# Chain Rule and HMMs



- From the chain rule, *every* joint distribution over  $X_1, E_1, \dots, X_T, E_T$  can be written as:

$$P(X_1, E_1, \dots, X_T, E_T) = P(X_1)P(E_1|X_1) \prod_{t=2}^T P(X_t|X_1, E_1, \dots, X_{t-1}, E_{t-1})P(E_t|X_1, E_1, \dots, X_{t-1}, E_{t-1}, X_t)$$

- Assuming that for all  $t$ :

- State independent of all past states and all past evidence given the previous state, i.e.:

$$X_t \perp\!\!\!\perp X_1, E_1, \dots, X_{t-2}, E_{t-2}, E_{t-1} \mid X_{t-1}$$

- Evidence is independent of all past states and all past evidence given the current state, i.e.:

$$E_t \perp\!\!\!\perp X_1, E_1, \dots, X_{t-2}, E_{t-2}, X_{t-1}, E_{t-1} \mid X_t$$

gives us the expression posited on the earlier slide:

$$P(X_1, E_1, \dots, X_T, E_T) = P(X_1)P(E_1|X_1) \prod_{t=2}^T P(X_t|X_{t-1})P(E_t|X_t)$$

# HMM components: Here ( $X_i = Q_i$ ) and ( $E_i = O_i$ )

$Q = q_1 q_2 \dots q_N$	a set of $N$ <b>states</b>
$A = a_{11} \dots a_{ij} \dots a_{NN}$	a <b>transition probability matrix</b> $A$ , each $a_{ij}$ representing the probability of moving from state $i$ to state $j$ , s.t. $\sum_{j=1}^N a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of $T$ <b>observations</b> , each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
$B = b_i(o_t)$	a sequence of <b>observation likelihoods</b> , also called <b>emission probabilities</b> , each expressing the probability of an observation $o_t$ being generated from a state $i$
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an <b>initial probability distribution</b> over states. $\pi_i$ is the probability that the Markov chain will start in state $i$ . Some states $j$ may have $\pi_j = 0$ , meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

# Three Problems of HMM

An influential tutorial by Rabiner (1989), based on tutorials by Jack Ferguson in the 1960s, introduced the idea that hidden Markov models should be characterized by **three fundamental problems**:

**Problem 1 (Likelihood):** Given an HMM  $\lambda = (A, B)$  and an observation sequence  $O$ , determine the likelihood  $P(O|\lambda)$ .

**Problem 2 (Decoding):** Given an observation sequence  $O$  and an HMM  $\lambda = (A, B)$ , discover the best hidden state sequence  $Q$ .

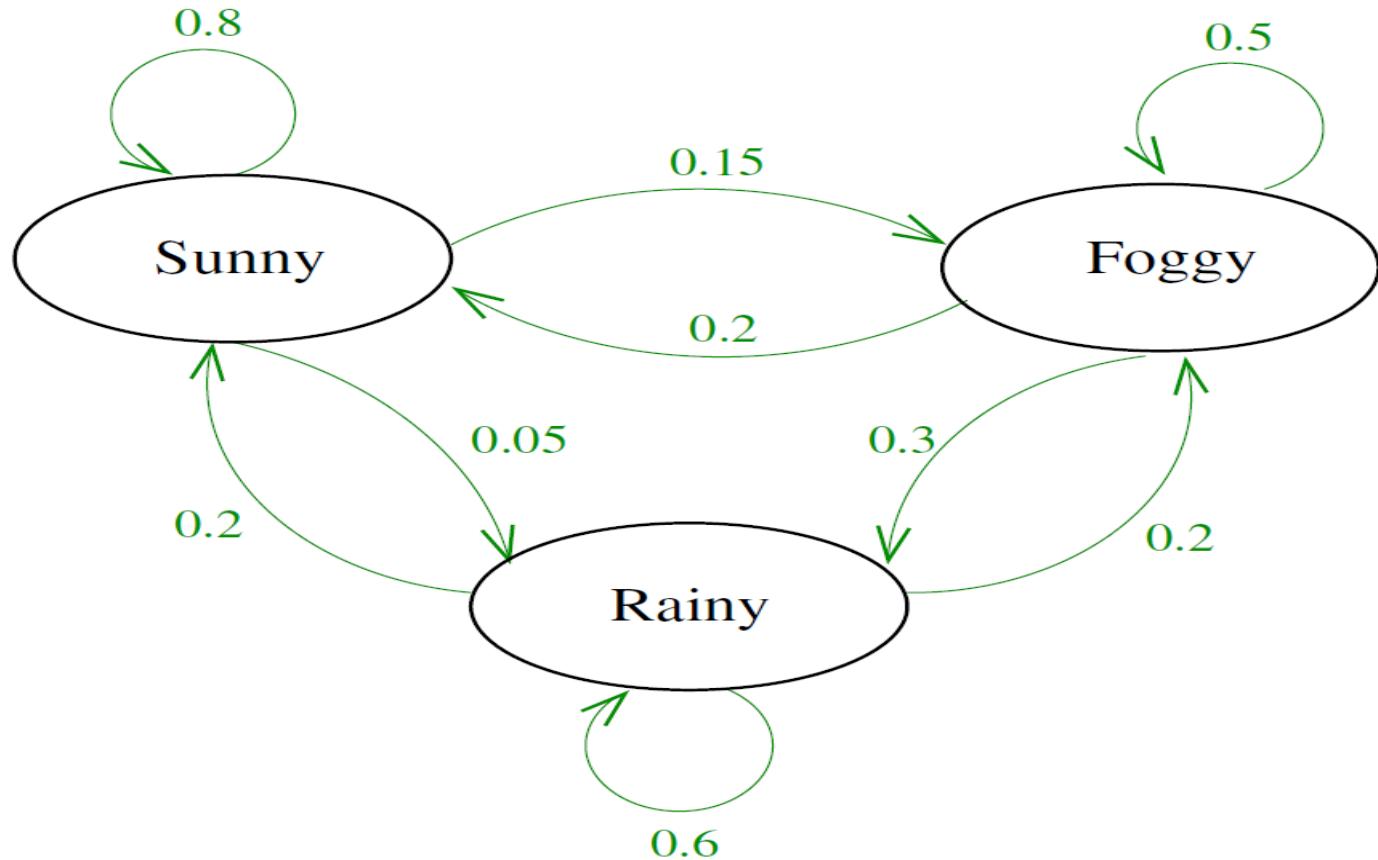
**Problem 3 (Learning):** Given an observation sequence  $O$  and the set of states in the HMM, learn the HMM parameters  $A$  and  $B$ .

# HMM Example

Today's weather	Tomorrow's weather		
			
	0.8	0.05	0.15
	0.2	0.6	0.2
	0.2	0.3	0.5

Probabilities  $p(q_{n+1}|q_n)$  of tomorrow's weather based on today's weather

# Markov model with Graz weather with state transition probabilities



# Probability of Umbrella

Weather	Probability of umbrella
Sunny	0.1
Rainy	0.8
Foggy	0.3

## Example 1

1. Suppose the day you were locked in it was sunny. The next day, the caretaker carried an umbrella into the room. You would like to know, what the weather was like on this second day.

First we calculate the likelihood for the second day to be sunny:

$$\begin{aligned}L(q_2 = \text{☀️}|q_1 = \text{☀️}, x_2 = \text{☂️}) &= P(x_2 = \text{☂️}|q_2 = \text{☀️}) \cdot P(q_2 = \text{☀️}|q_1 = \text{☀️}) \\&= 0.1 \cdot 0.8 = 0.08,\end{aligned}$$

then for the second day to be rainy:

$$\begin{aligned}L(q_2 = \text{🌧️}|q_1 = \text{☀️}, x_2 = \text{☂️}) &= P(x_2 = \text{☂️}|q_2 = \text{🌧️}) \cdot P(q_2 = \text{🌧️}|q_1 = \text{☀️}) \\&= 0.8 \cdot 0.05 = 0.04,\end{aligned}$$

and finally for the second day to be foggy:

$$\begin{aligned}L(q_2 = \text{🌫️}|q_1 = \text{☀️}, x_2 = \text{☂️}) &= P(x_2 = \text{☂️}|q_2 = \text{🌫️}) \cdot P(q_2 = \text{🌫️}|q_1 = \text{☀️}) \\&= 0.3 \cdot 0.15 = 0.045.\end{aligned}$$

Thus, although the caretaker did carry an umbrella, it is most likely that on the second day the weather was sunny.

## Example 2

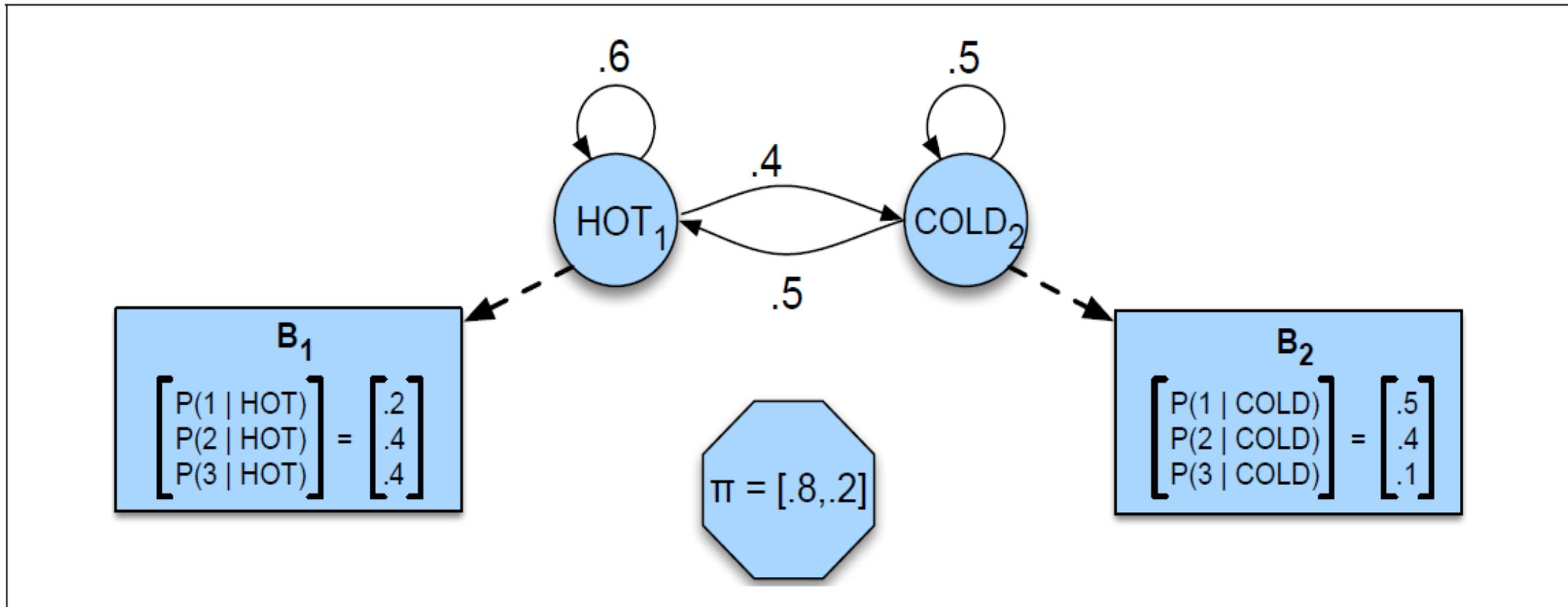
2. Suppose you do not know how the weather was when you were locked in. The following three days the caretaker always comes without an umbrella. Calculate the likelihood for the weather on these three days to have been  $\{q_1 = \text{☀️}, q_2 = \text{☁️}, q_3 = \text{☀️}\}$ . As you do not know how the weather is on the first day, you assume the 3 weather situations are equi-probable on this day (cf. footnote on page 2), and the *prior probability* for sun on day one is therefore  $P(q_1 = \text{☀️}|q_0) = P(q_1 = \text{☀️}) = 1/3$ .

$$\begin{aligned} L(q_1 = \text{☀️}, q_2 = \text{☁️}, q_3 = \text{☀️} | x_1 = \text{✖}, x_2 = \text{✖}, x_3 = \text{✖}) &= \\ P(x_1 = \text{✖} | q_1 = \text{☀️}) \cdot P(x_2 = \text{✖} | q_2 = \text{☁️}) \cdot P(x_3 = \text{✖} | q_3 = \text{☀️}) \cdot \\ P(q_1 = \text{☀️}) \cdot P(q_2 = \text{☁️} | q_1 = \text{☀️}) \cdot P(q_3 = \text{☀️} | q_2 = \text{☁️}) &= \\ 0.9 \cdot 0.7 \cdot 0.9 \cdot 1/3 \cdot 0.15 \cdot 0.2 &= 0.0057 \end{aligned}$$

# ICE Cream Problem

- Imagine that you are a climatologist in the year 2799 studying the history of global warming. You cannot find any records of the weather in Amaravati, for the summer of 2020, but you do find Jason Eisner's diary, which lists how many ice creams Jason ate every day that summer.
- Our goal is to use these observations to estimate the temperature every day.
- We'll simplify this weather task by assuming there are only two kinds of days: cold (C) and hot (H).
- So the Eisner task is as follows: Given a sequence of observations  $O$  (each an integer representing the number of ice creams eaten on a given day) find the 'hidden' sequence  $Q$  of weather states (H or C) which caused Jason to eat the ice cream.
- Figure A.2 shows a sample HMM for the ice cream task.
- The two hidden states (H and C) correspond to hot and cold weather, and the observations (drawn from the alphabet  $O = \{1,2,3\}$ ) correspond to the number of ice creams eaten by Jason on a given day

# ICE Cream Problem



**Figure A.2** A hidden Markov model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables).

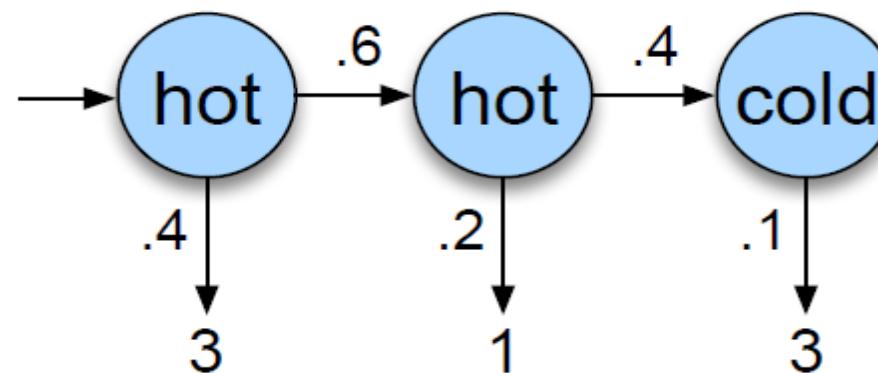
## Question ???

- Compute the probability of ice-cream events 3 1 3 instead by summing over all possible weather sequences, weighted by their probability.
- First, let's compute the joint probability of being in a particular weather sequence  $Q$  and generating a particular sequence  $O$  of ice-cream events.

The computation of the joint probability of our ice-cream observation 3 1 3 and one possible hidden state sequence **hot hot cold** is shown below

$$P(O, Q) = P(O|Q) \times P(Q) = \prod_{i=1}^T P(o_i|q_i) \times \prod_{i=1}^T P(q_i|q_{i-1})$$

$$\begin{aligned} P(3\ 1\ 3, \text{hot hot cold}) &= P(\text{hot|start}) \times P(\text{hot|hot}) \times P(\text{cold|hot}) \\ &\quad \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \end{aligned}$$



**Figure A.4** The computation of the joint probability of the ice-cream events 3 1 3 and the hidden state sequence *hot hot cold*.

- $N = \text{No. of hidden states} = 2$
  - $T = \text{Given no.of observations} = 3$
  - $M = N^T = 2^3 = 8 \text{ possibilities (HHC, CCC, HCH... )}$
- 
- $N= 7$
  - $T = 10$
  - $N^T = 7^{10}$

# How to solve this : Decoding: The Viterbi Algorithm

Each cell of the forward algorithm trellis  $\alpha_t(j)$  represents the probability of being in state  $j$  after seeing the first  $t$  observations, given the automaton  $\lambda$ . The value of each cell  $\alpha_t(j)$  is computed by summing over the probabilities of every path that could lead us to this cell. Formally, each cell expresses the following probability:

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda) \quad (\text{A.11})$$

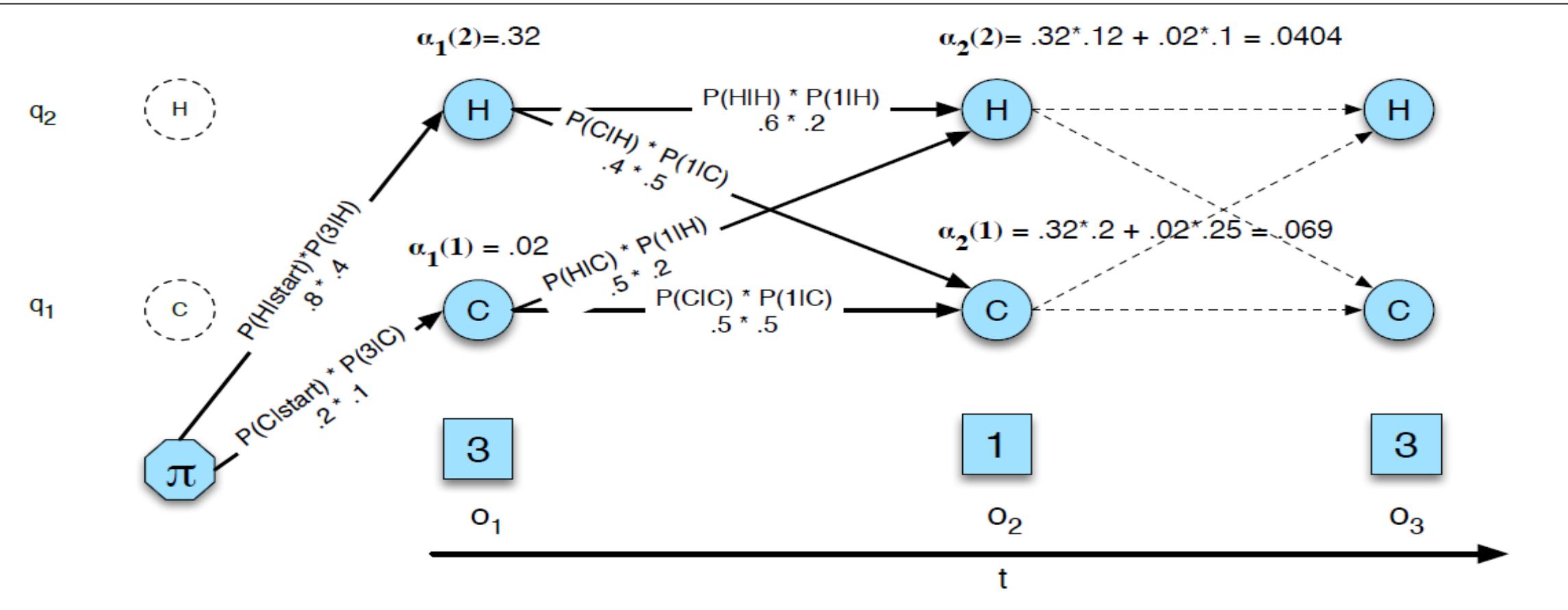
Here,  $q_t = j$  means “the  $t^{\text{th}}$  state in the sequence of states is state  $j$ ”. We compute this probability  $\alpha_t(j)$  by summing over the extensions of all the paths that lead to the current cell. For a given state  $q_j$  at time  $t$ , the value  $\alpha_t(j)$  is computed as

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t) \quad (\text{A.12})$$

The three factors that are multiplied in Eq. A.12 in extending the previous paths to compute the forward probability at time  $t$  are

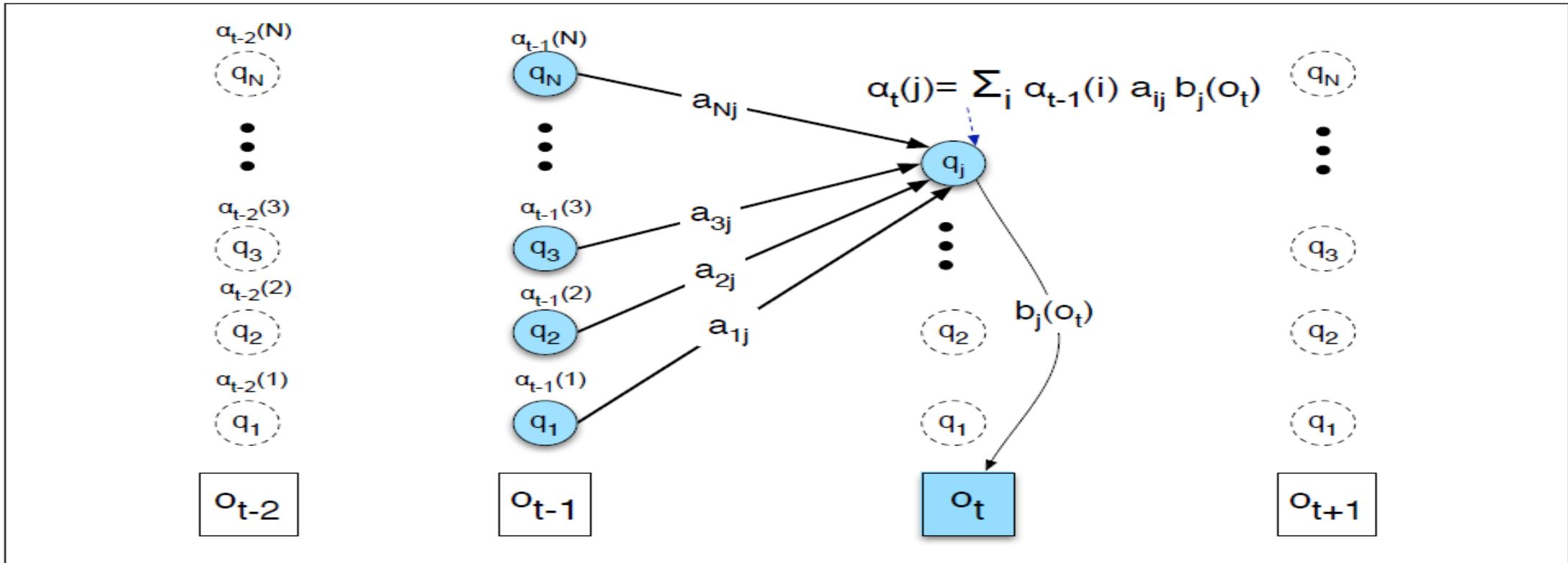
$\alpha_{t-1}(i)$	the <b>previous forward path probability</b> from the previous time step
$a_{ij}$	the <b>transition probability</b> from previous state $q_i$ to current state $q_j$
$b_j(o_t)$	the <b>state observation likelihood</b> of the observation symbol $o_t$ given the current state $j$

# Forward Trellis



**Figure A.5** The forward trellis for computing the total observation likelihood for the ice-cream events 3 1 3. Hidden states are in circles, observations in squares. The figure shows the computation of  $\alpha_t(j)$  for two states at two time steps. The computation in each cell follows Eq. A.12:  $\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$ . The resulting probability expressed in each cell is Eq. A.11:  $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$ .

# Likelihood Computation: The forward algorithm



**Figure A.6** Visualizing the computation of a single element  $\alpha_t(i)$  in the trellis by summing all the previous values  $\alpha_{t-1}$ , weighted by their transition probabilities  $a$ , and multiplying by the observation probability  $b_i(o_t)$ . For many applications of HMMs, many of the transition probabilities are 0, so not all previous states will contribute to the forward probability of the current state. Hidden states are in circles, observations in squares. Shaded nodes are included in the probability computation for  $\alpha_t(i)$ .

# The Forward Algorithm

```
function FORWARD(observations of len  $T$ , state-graph of len  $N$ ) returns forward-prob
    create a probability matrix forward[ $N,T$ ]
    for each state  $s$  from 1 to  $N$  do ; initialization step
         $\text{forward}[s,1] \leftarrow \pi_s * b_s(o_1)$ 
    for each time step  $t$  from 2 to  $T$  do ; recursion step
        for each state  $s$  from 1 to  $N$  do
             $\text{forward}[s,t] \leftarrow \sum_{s'=1}^N \text{forward}[s',t-1] * a_{s',s} * b_s(o_t)$ 
     $\text{forwardprob} \leftarrow \sum_{s=1}^N \text{forward}[s,T]$  ; termination step
    return forwardprob
```

**Figure A.7** The forward algorithm, where  $\text{forward}[s,t]$  represents  $\alpha_t(s)$ .

## Steps in Forward Algorithm

1. Initialization:

$$\alpha_1(j) = \pi_j b_j(o_1) \quad 1 \leq j \leq N$$

2. Recursion:

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i)$$