

ECE 610
Machine Learning
&
Artificial Intelligence

Spring 2024
David Han
Electrical and Computer Engineering
Drexel University

ECE 610 Machine Learning & AI

- Instructor: David Han (dkh42@drexel.edu)

- Lecture Time: Tue. & Thr. 3:30 – 4:50 PM

- Location: Papadakis room 104

- Office Hour: Wed. 11:00 AM – 12:00 PM

- Location: Bossone Hall room 513b

University Academic Policies

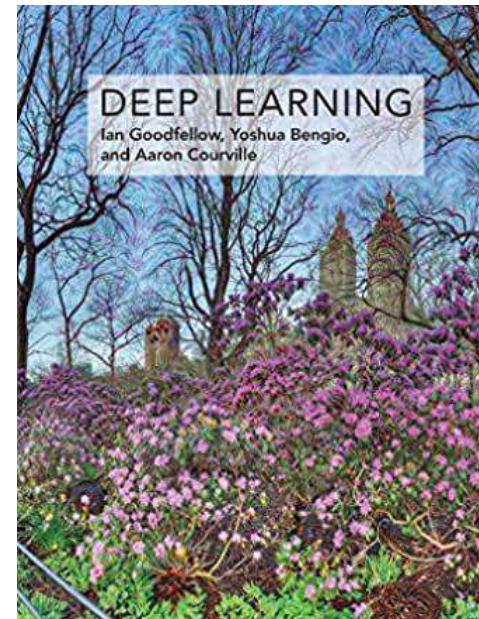
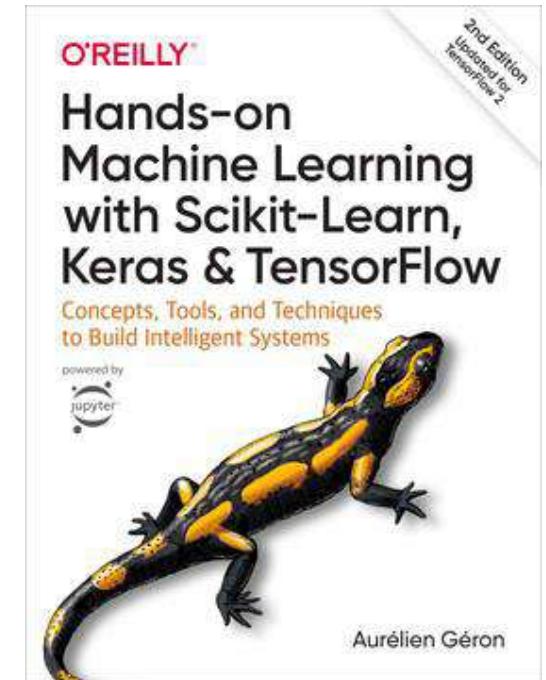
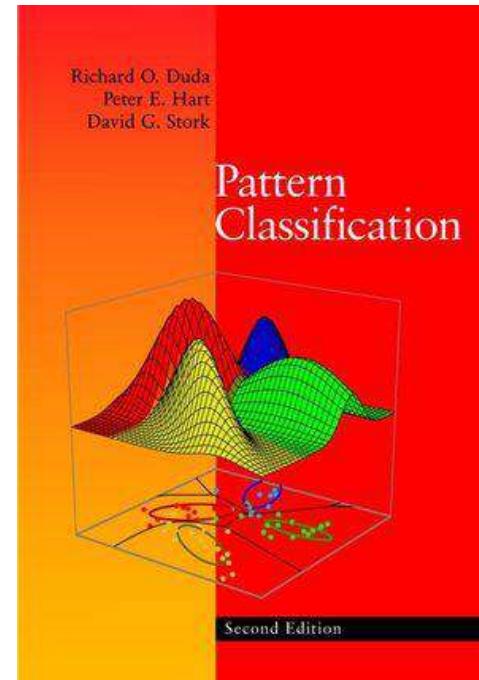
- **Course Change Policy:** The instructor reserves the **right to modify** the course, as necessary, during the term: including policies, evaluations, due dates, course content, schedule, assignments or requirements. All changes will be communicated in **lecture and/or via the course DrexelLearn page**.
- Please review the following relevant university policies.
- **Academic Integrity, Plagiarism, Dishonesty and Cheating Policy:**
- <http://drexel.edu/provost/policies/absence/>
- Disability Resources: Students requesting accommodations due to a disability at Drexel University need to request a current Accommodations Verification Letter (AVL) in the ClockWork database before accommodations can be made. These requests are received by Disability Resources (DR), who then issues the AVL to the appropriate contacts. For additional information, visit the DR website at <http://drexel.edu/oed/disabilityResources/overview/>, or contact DR for more information by phone at 215.895.1401, or by email at disability@drexel.edu.
- Course Add/Drop Policy: <http://www.drexel.edu/provost/policies/course-add-drop>
- Course Withdrawal Policy: <http://drexel.edu/provost/policies/course-withdrawal>

Purpose of this course

- To gain in depth conceptual understanding of modern machine learning methods
 - To gain hands on knowledge of writing basic python-based machine learning codes for solving some representative problems.
 - To obtain key knowledge in machine learning in preparation for graduate level research
-
- **Required background: basic concepts in probability, linear algebra, and elementary python**

Course Textbook

- There is no single source for the course material
- Earlier part of the course will be from “Pattern Classification” by Duda, Hart, and Stork
 - Not necessary to purchase, mostly will be in lecture notes
- Some material will be from “Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow,” by Aurelien Geron
 - Available online from Drexel Library
- Select material will be from “Deep Learning,” by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
 - Available online from Drexel Library
- Additionally, each lecture will be accompanied with lecture slides, which serve as the primary resource for the course
- Lecture slides will be posted on blackboard afterwards



Course Syllabus

- We will use the **BBLearn** system to deliver all information regarding the course, including assignment descriptions, links to online materials, and grades as they are calculated.
- Your assignments are to be submitted through this system in a manner consistent with each assignment description. However, emails should not be sent through this system, but rather directly to the instructor at the email addresses above.
- **Assessment Overview**
 - 5% - Class attendance
 - 25% - Assignments
 - 20% - Midterm Exam
 - 25% - Final Exam
 - 25% - Paper report presentation

Course Outline

- AI landscape
- Linear Algebra & Probability review
- Naïve Bayes Classifier
- Learning Loss
- Artificial Neural Network
- Multi-layer Perceptron
- Convolutional Neural Network
- Sequence Models
- Attention Mechanism
- Generative Models
- Reviews of contemporary ML papers

Machine Learning is every where!

- Language translation services (Google) 
- Speech translation (Google Translate) 
- Song recognition (Shazam) 
- Face recognition (Recognizr)

- Question answering (Google Home, Amazon Alexa, Apple Siri, IBM Watson)
- Board games (IBM Deep Blue, Google DeepMind AlphaGo)



- Driverless cars (Uber, Tesla, Google)
- Deep Fake



Semantic Segmentation for Driverless Car Application



ICNet for Real-Time Semantic Segmentation on High-Resolution Images

Hengshuang Zhao¹ Xiaojuan Qi¹ Xiaoyong Shen¹ Jianping Shi² Jiaya Jia¹

¹The Chinese University of Hong Kong ²SenseTime Group Limited

*Each frame in the video is processed independently at the rate of 30 fps on a 1024*2048 resolution image.*

Also some filters



YouTube, Guardian News, 'I'm not a cat': lawyer gets stuck on Zoom kitten filter during court case,
<https://www.youtube.com/watch?v=lGOofzZOyl8>

Some Exciting New Development

- ChatGPT: Large Language Model (LLM)
- Example:
- User: Explain how a combustion engine works
- ChatGPT: “A combustion engine works by converting the energy stored in fuel into mechanical energy. This is typically done through a four-stroke process in which fuel is mixed with air and then ignited in the combustion chamber. The resulting explosion drives a piston, which moves the engine’s crankshaft and converts the linear motion of the piston into the rotational motion of the crankshaft. This rotational motion can then be used to power a vehicle or other machinery.”
- User: Who won the F1 championship in 2021?
- ChatGPT: “I’m sorry, but I am a large language model trained by OpenAI and I do not have access to current information. My training only goes up until 2021, so I am unable to provide information about events that have occurred after that. My knowledge is limited to what I was trained on and I do not have the ability to browse the internet or access other sources of information. Is there anything else I may be able to help you with?”

ChatGPT

- ChatGPT task examples:
 - Speech and text analysis
 - Translations
 - Explanations of complex issues
 - Writing stories and essays
 - Learn coding
 - Debugging code

What is Machine Learning?

- Science (and art) of programming computers so they can **learn from data**.
- A study that gives computers the ability to learn without being explicitly programmed. —Arthur Samuel, 1959
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. —Tom Mitchell, 1997

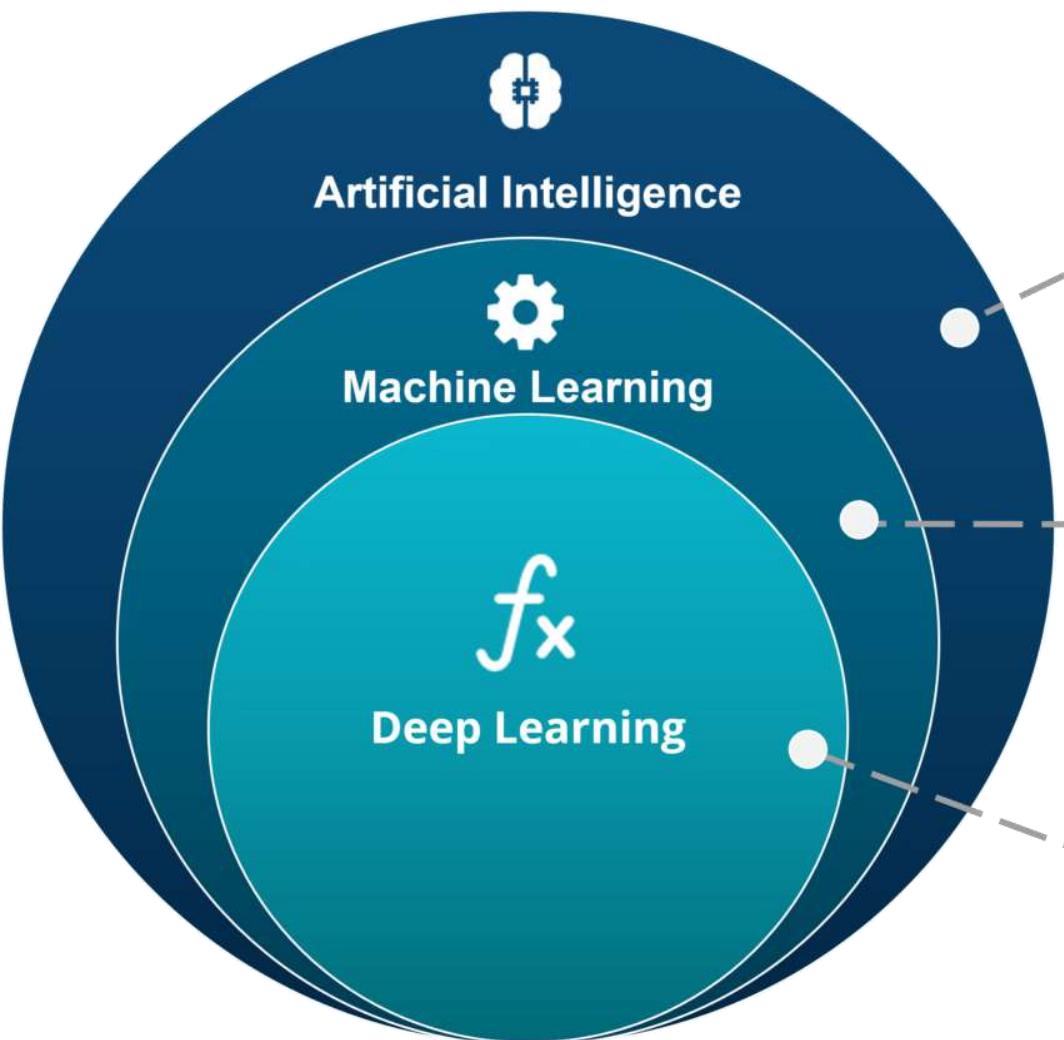
What is Machine Learning?

- Science (and art) of programming computers so they can **learn from data**.
- A study that gives computers the ability to learn **without being explicitly programmed**. —Arthur Samuel, 1959
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. —Tom Mitchell, 1997

What is Machine Learning?

- Science (and art) of programming computers so they can **learn from data**.
- A study that gives computers the ability to learn **without being explicitly programmed**. —Arthur Samuel, 1959
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its **performance on T, as measured by P, improves with experience E**. —Tom Mitchell, 1997

AI vs Machine Learning vs Deep Learning



ARTIFICIAL INTELLIGENCE

A technique which enables machines to mimic human behaviour

MACHINE LEARNING

Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING

Subset of ML which make the computation of multi-layer neural network feasible

Artificial Intelligence Timeline

Artificial Intelligence

Any technique which enables computers to mimic human behavior

Machine Learning

AI techniques that give computers the ability to learn without being explicitly programmed to do so

Deep Learning

A subset of ML which make the computation of multi-layer neural networks feasible

1950's

1960's

1970's

1980's

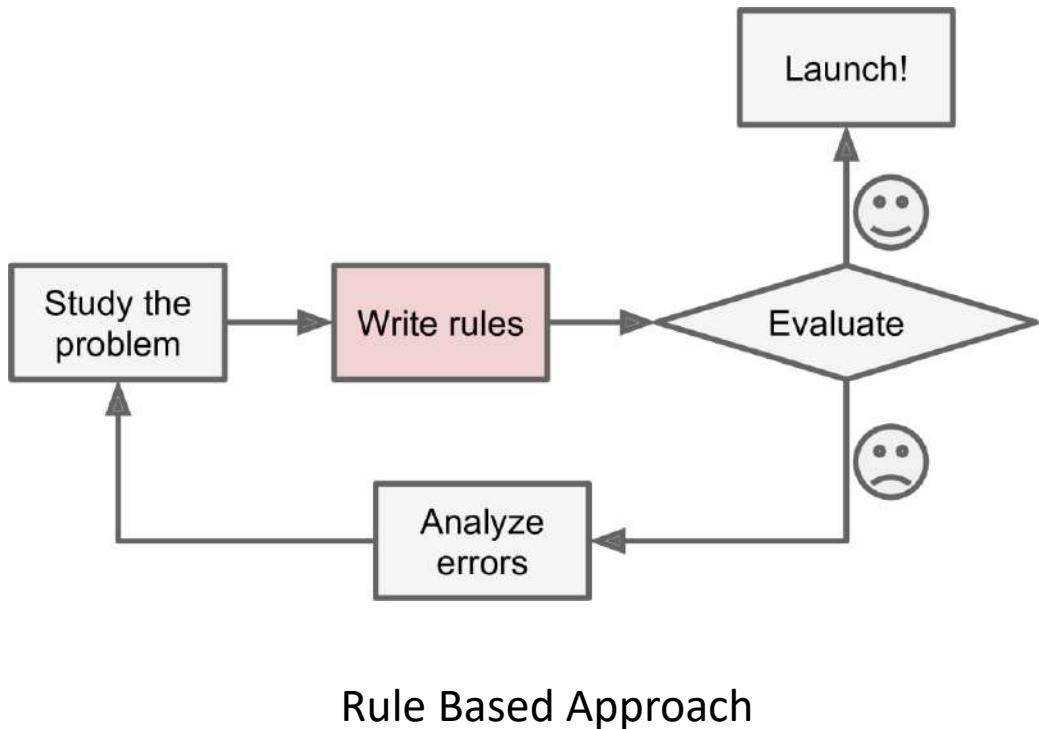
1990's

2000's

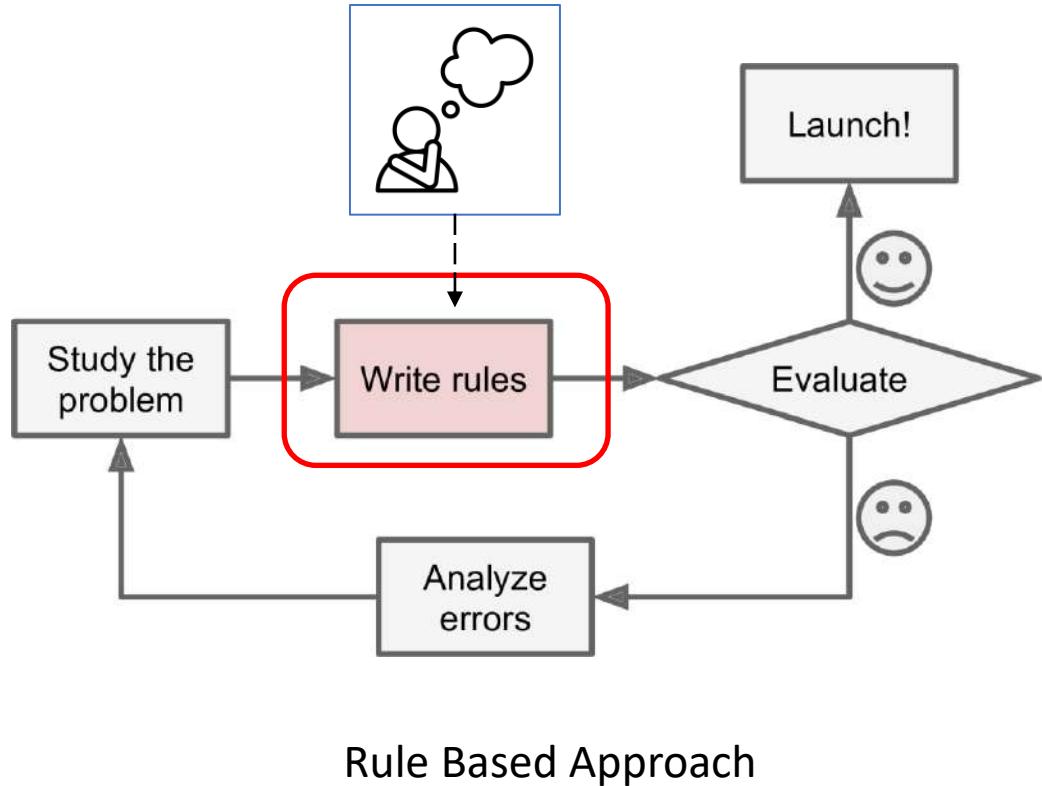
2010's

2020's

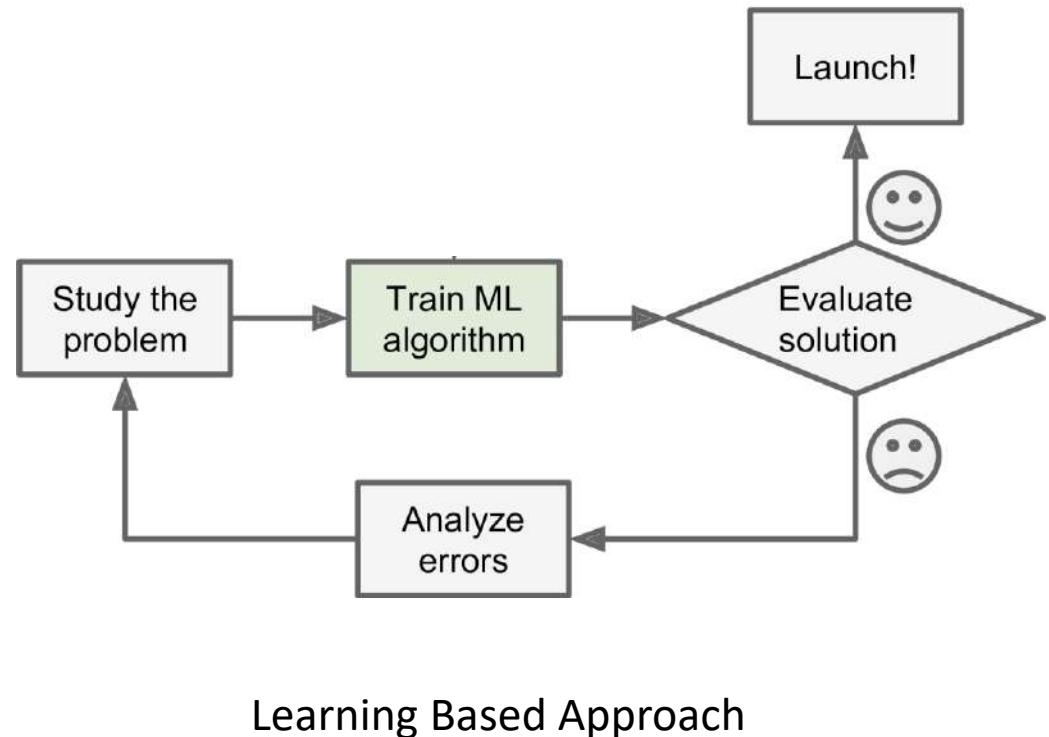
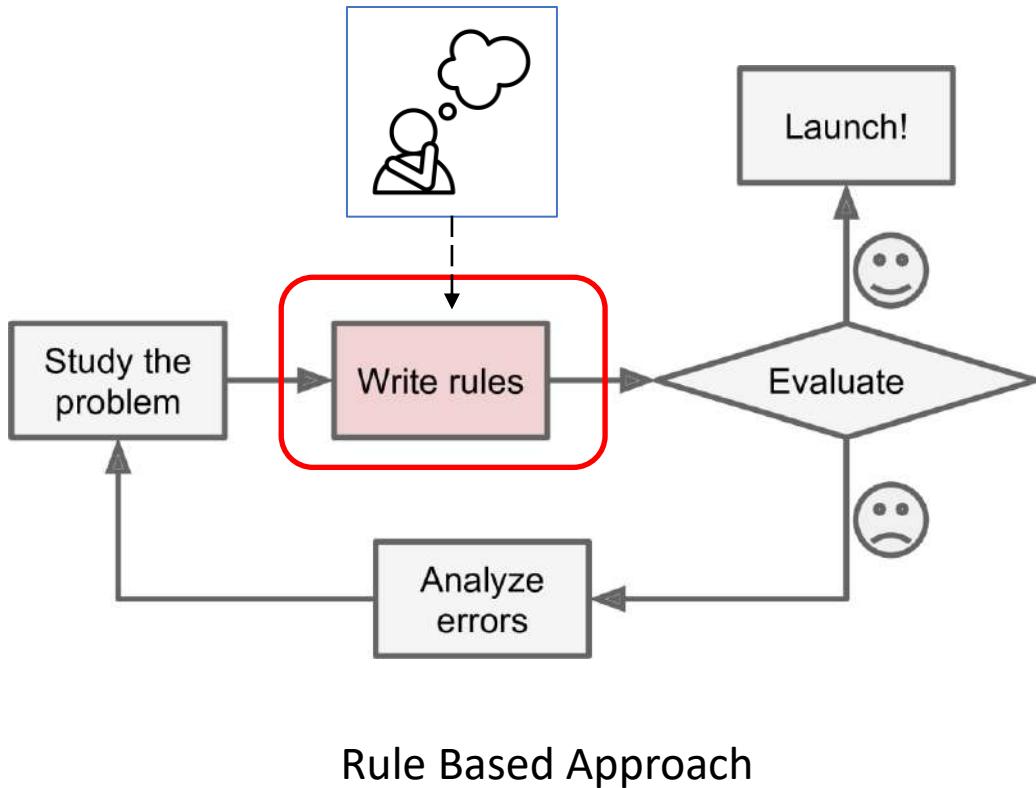
Good Old Fashioned Artificial Intelligence (GOFAI) vs Machine Learning



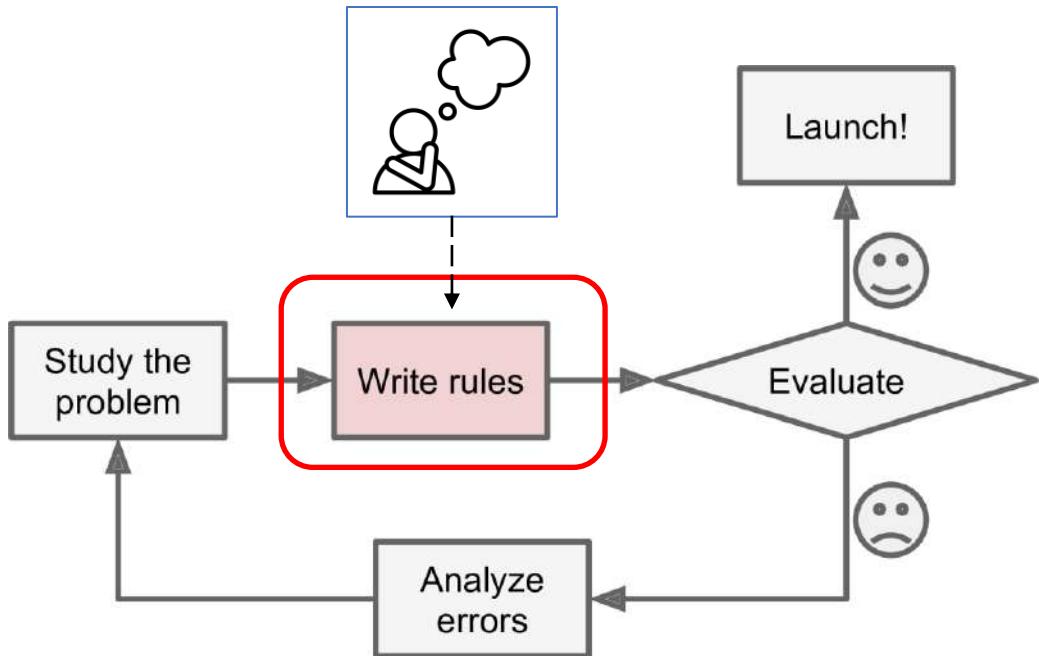
Good Old Fashioned Artificial Intelligence (GOFAI) vs Machine Learning



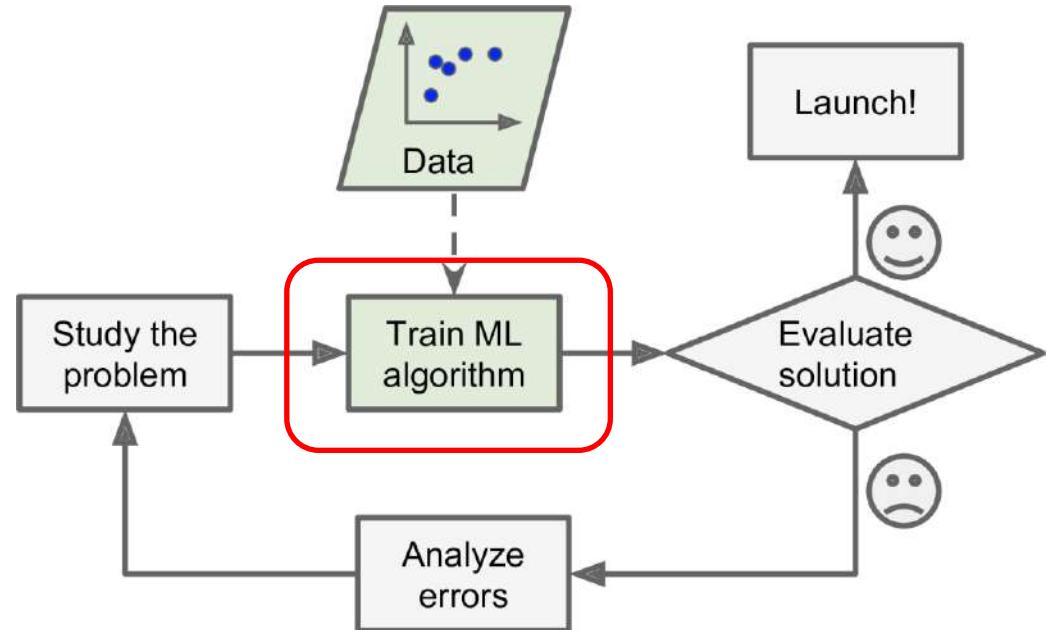
Good Old Fashioned Artificial Intelligence (GOFAI) vs Machine Learning



Good Old Fashioned Artificial Intelligence (GOFAI) vs Machine Learning



Rule Based Approach



Learning Based Approach

Rule based approach vs ML approach

Rule Based Approach (Good Old Fashion AI)

- Deterministic
- Can operate with simple basic information
- Hard to scale as problem complexity grows
- Rules are implemented manually
→ Knowledge Engineering

Learning Based Approach

- Probabilistic
- Constantly evolve, develop and adapt its production in accordance with training information streams
- Learns rules from data
- Easily scaled
- Requires more data for better performance

Rule Based AI: Expressing human knowledge in the form of rules for problem solving

- Rule can be defined as an IF-THEN structure that relates **given information or facts** in the **IF** part to some **action** in the **THEN** part. A rule provides a solution to a problem
- Any rule consists of two parts: the **IF part**, called the **antecedent (premise or condition)** and the **THEN part** called the **consequent (conclusion or action)**.

Rules as knowledge representation

IF **the 'traffic light' is green**
THEN **the action is go**



IF **the 'traffic light' is red**
THEN **the action is stop**

Learning Based Approach

How can a machine tell the difference between cars and motorcycles?

By extracting distinct features of each class

Cars



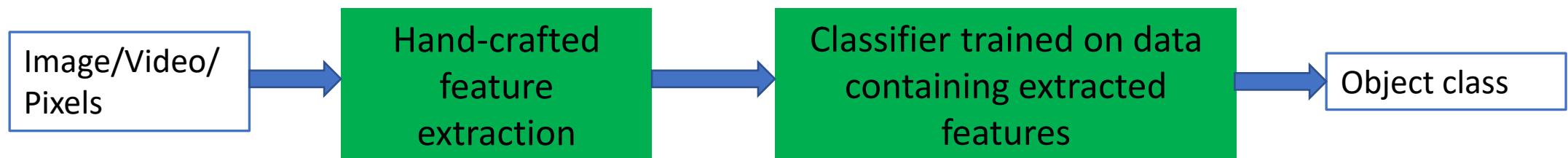
Motor Trend (2019), "The 15 Most Important Cars of the Decade,"
<https://www.motortrend.com/features/the-15-most-important-cars-of-the-decade/>

Motorcycles



Motordynasty (2020), "Top of the line Cruisers in September 2020,"
<https://motordynasty.com/2020/10/28/top-of-the-line-cruisers-in-september-2020/>

Traditional ML based Classification



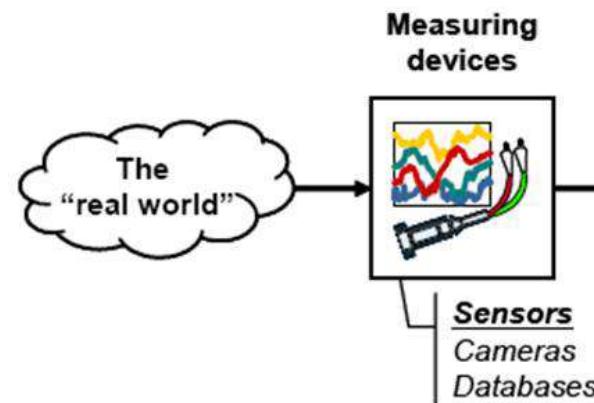
ML Process Before Deep Learning (BDL)

- Components of a basic machine learning system:
- Sensor(s)
- Preprocessor & Feature extraction (manual or automated)
- Classification algorithm
- Training set examples already classified or described



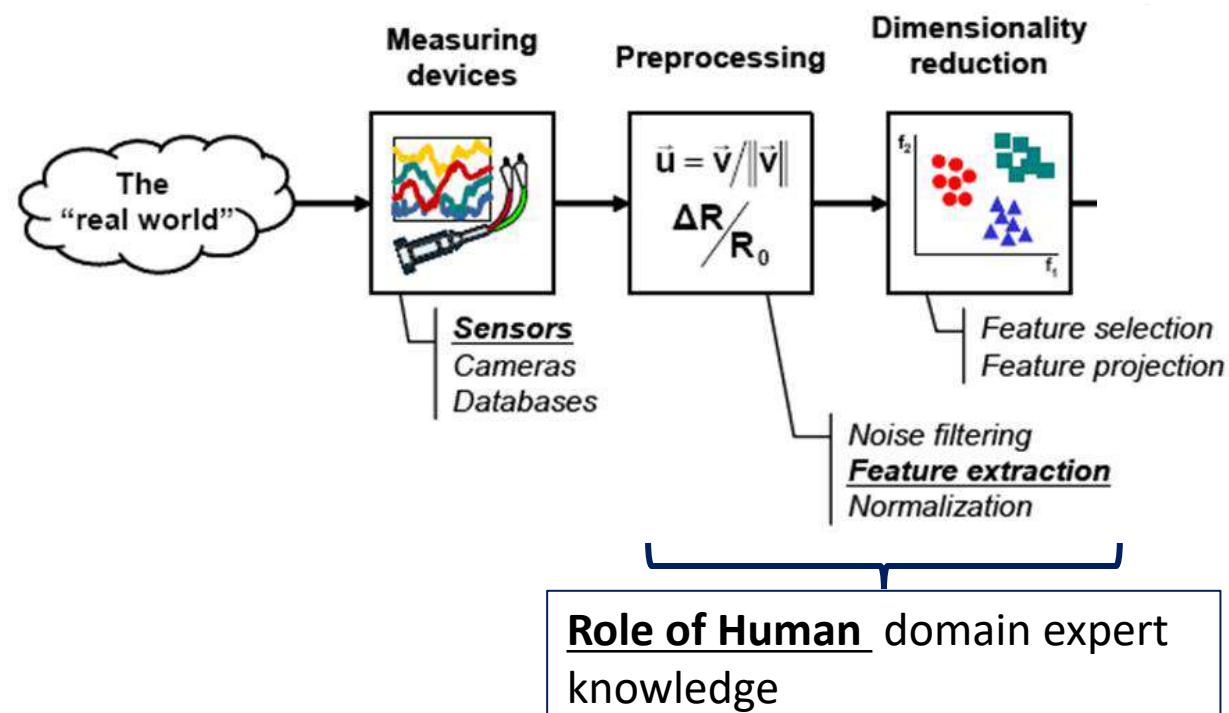
ML Process Before Deep Learning (BDL)

- Components of a basic machine learning system:
- Sensor(s)
- Preprocessor & Feature extraction (manual or automated)
- Classification algorithm
- Training set examples already classified or described



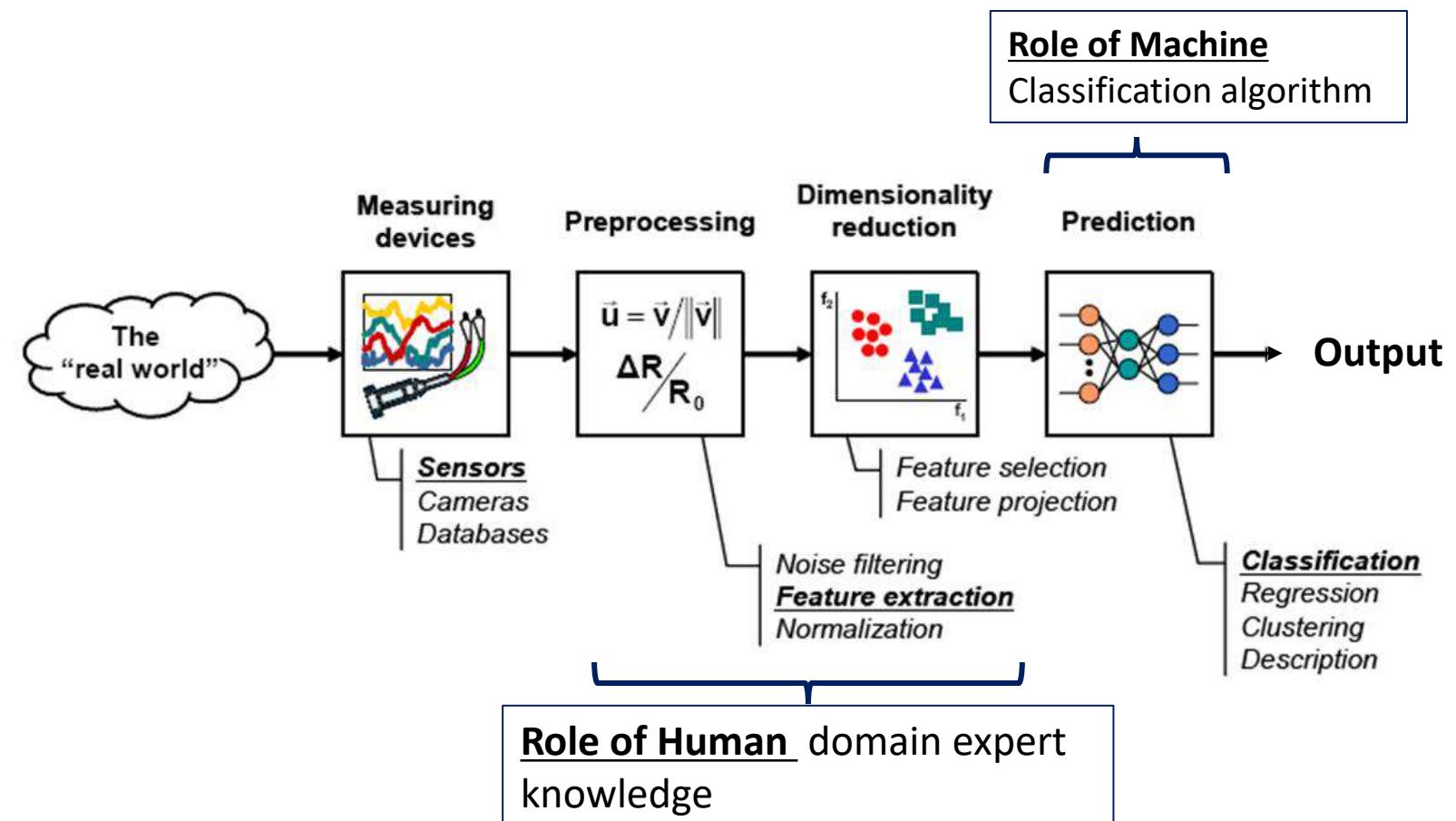
ML Process Before Deep Learning (BDL)

- Components of a basic machine learning system:
- Sensor(s)
- Preprocessor & **Feature extraction** (manual or automated)
- Classification algorithm
- Training set examples already classified or described



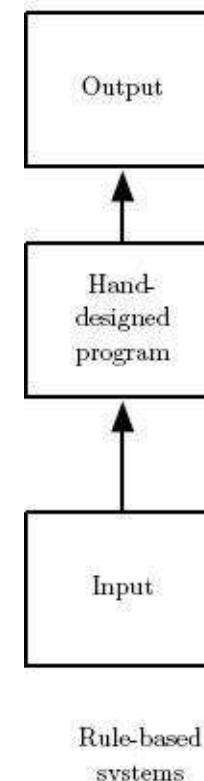
ML Process Before Deep Learning (BDL)

- Components of a basic machine learning system:
- Sensor(s)
- Preprocessor & **Feature extraction** (manual or automated)
- Classification algorithm
- Training set examples already classified or described



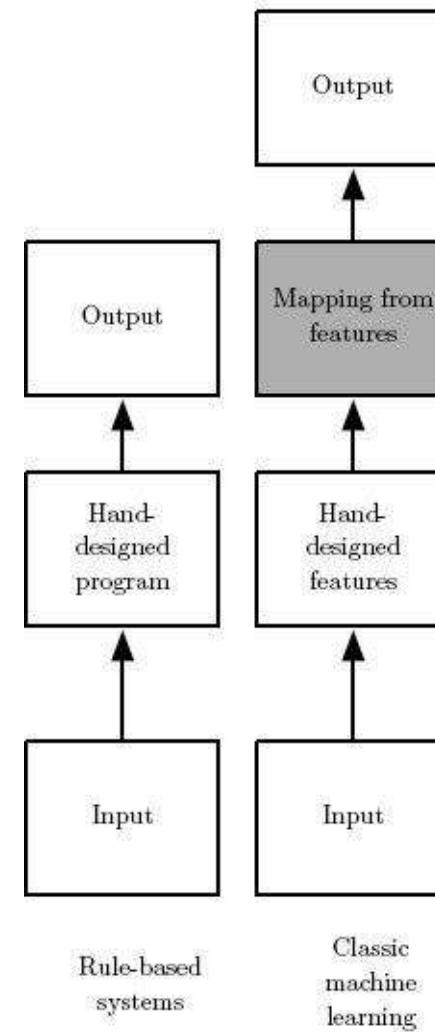
Classical to Modern AI

- Differences and similarities of AI approaches
- Note shaded components learn from data



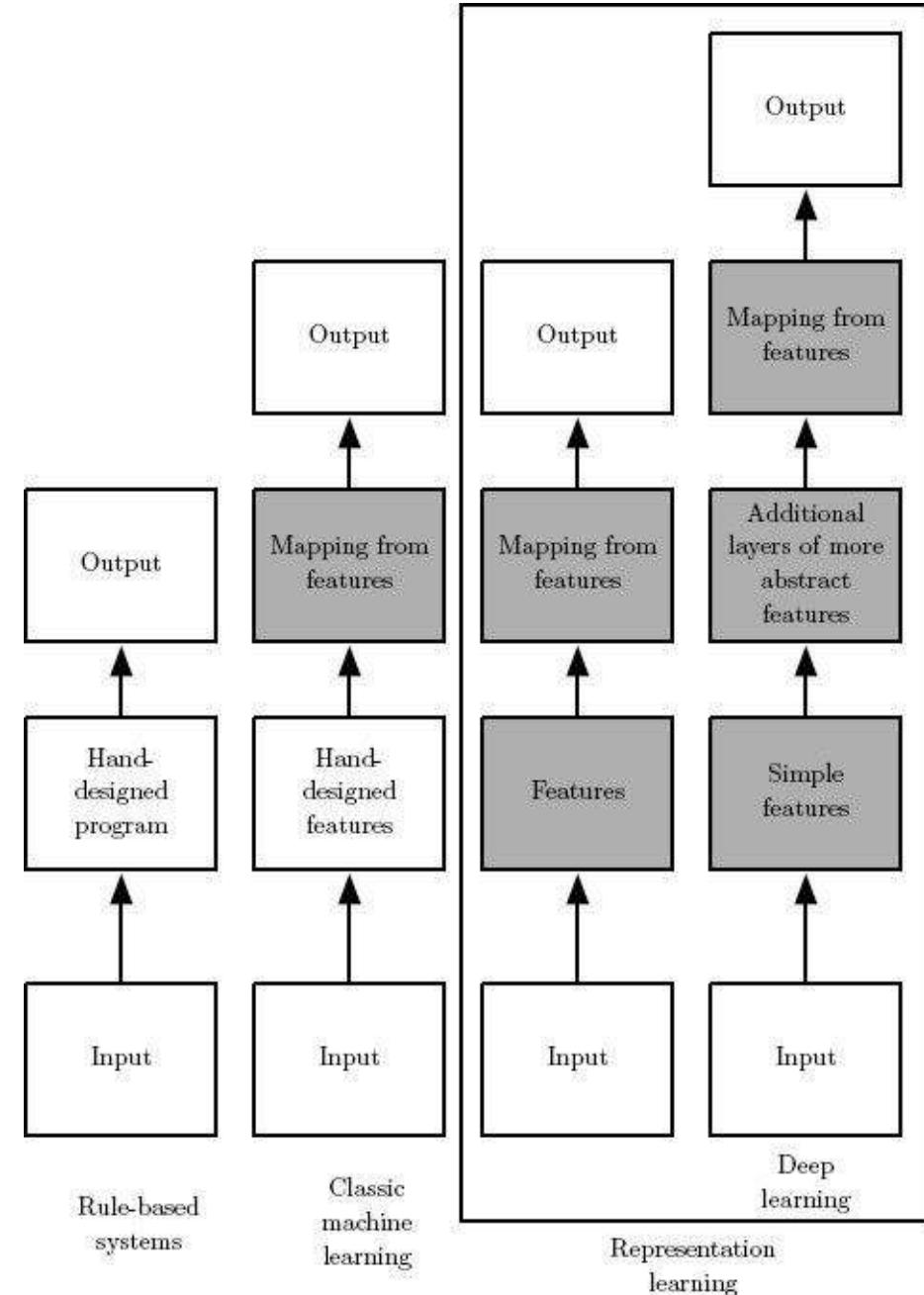
Classical to Modern AI

- Differences and similarities of AI approaches
- Note shaded components learn from data



Classical to Modern AI

- Differences and similarities of AI approaches
- Note shaded components learn from data



Features

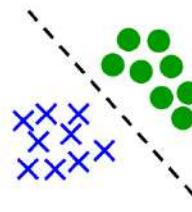
- Feature is any distinctive aspect, quality or characteristic
 - Can be categorical (i.e., color) or numeric (i.e., height)
- Definitions
 - The combination of d features is a d -dim column vector called a **feature vector**
 - The d -dimensional space defined by the feature vector is called **the feature space**
 - Objects are represented as points in feature space; the result is a **scatter plot**

Feature vector

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

Features

- What makes a “good” feature vector?
- The quality of a feature vector is related to its **ability to discriminate** examples from different classes
 - Examples from the **same class** should have **similar feature** values
 - Examples from different classes have different feature values

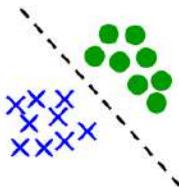


“Good” features

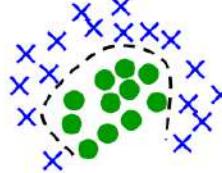


“Bad” features

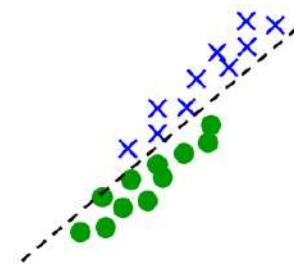
- More feature properties



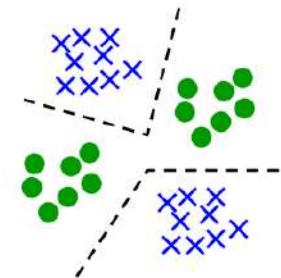
Linear separability



Non-linear separability



Highly correlated features



Multi-modal

Machine Learning: Classification Task

Sorting incoming fish on a conveyor according to species (salmon or sea bass) using optical sensing

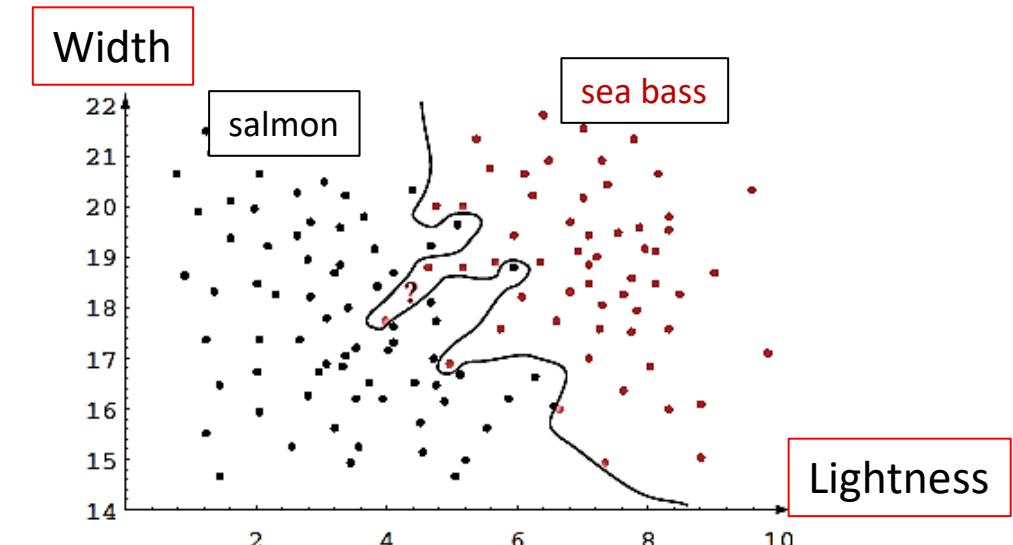
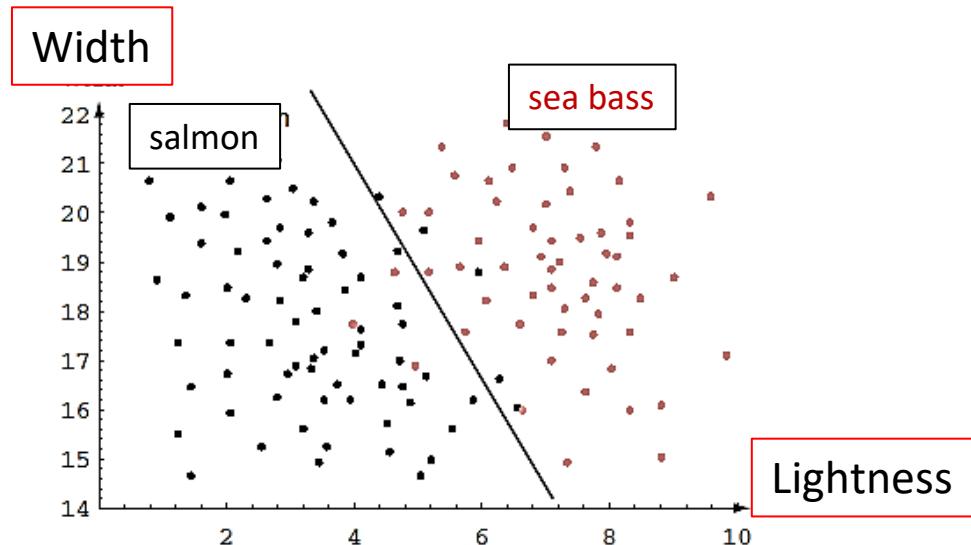


**Salmon or sea bass?
(2 categories or
classes)**

Duda, et. al. (1973), Pattern Classification, 2nd Ed.

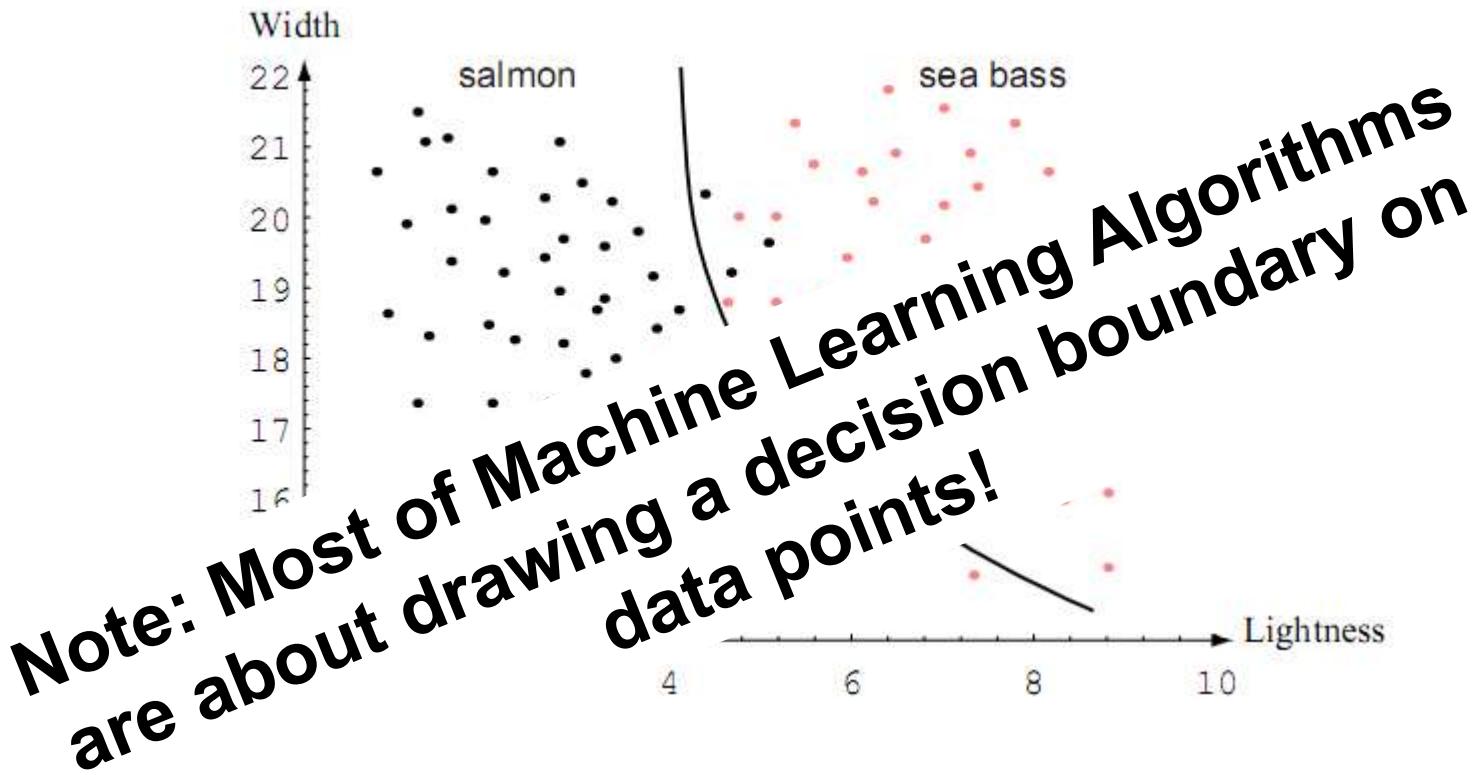
Binary Classification

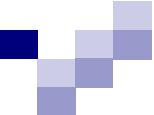
- Use both width and average lightness features for classification. Use a boundary to discriminate given feature space $\mathbf{x} = (x_1 \ x_2)$



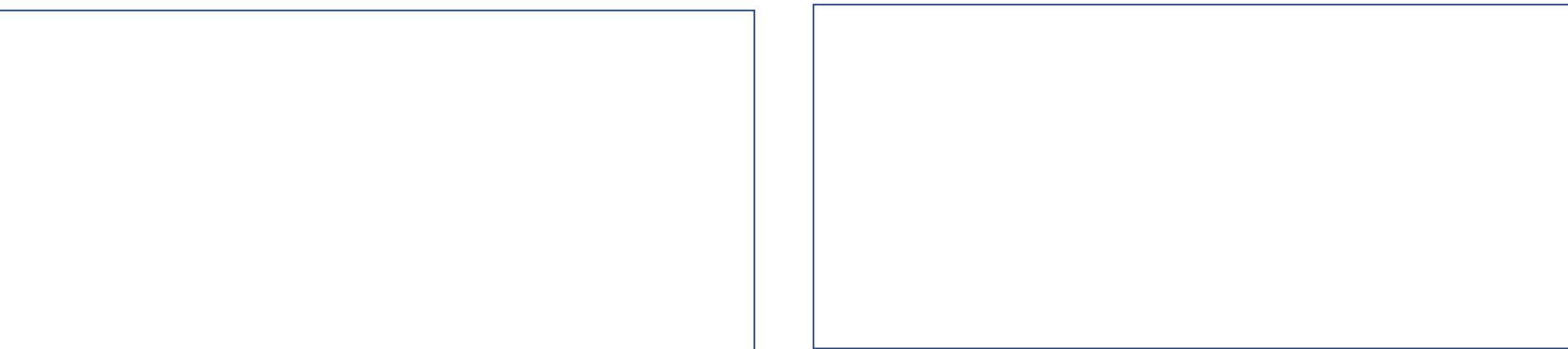
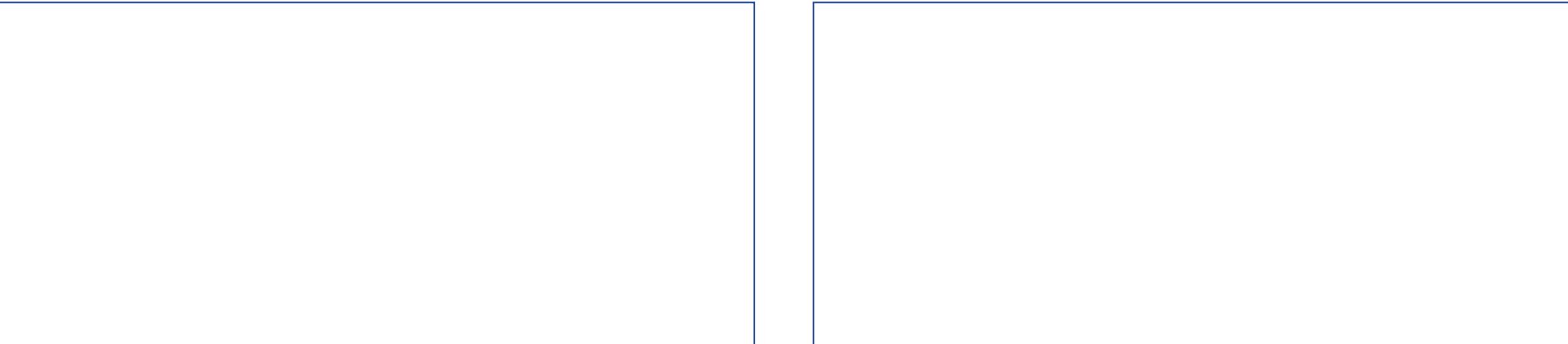
Binary Classification

- Decision boundary with good generalization



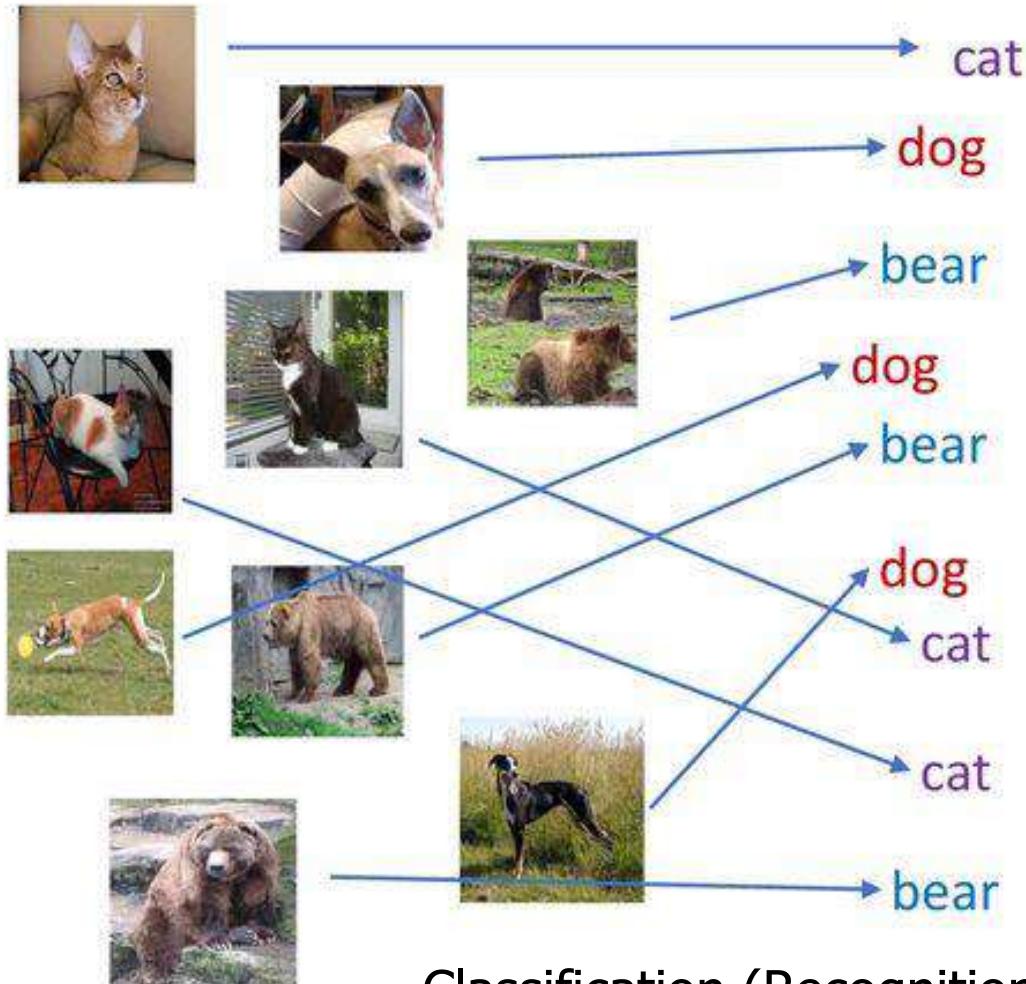


3+1 Types of Machine Learning



Supervised Learning vs Unsupervised Learning

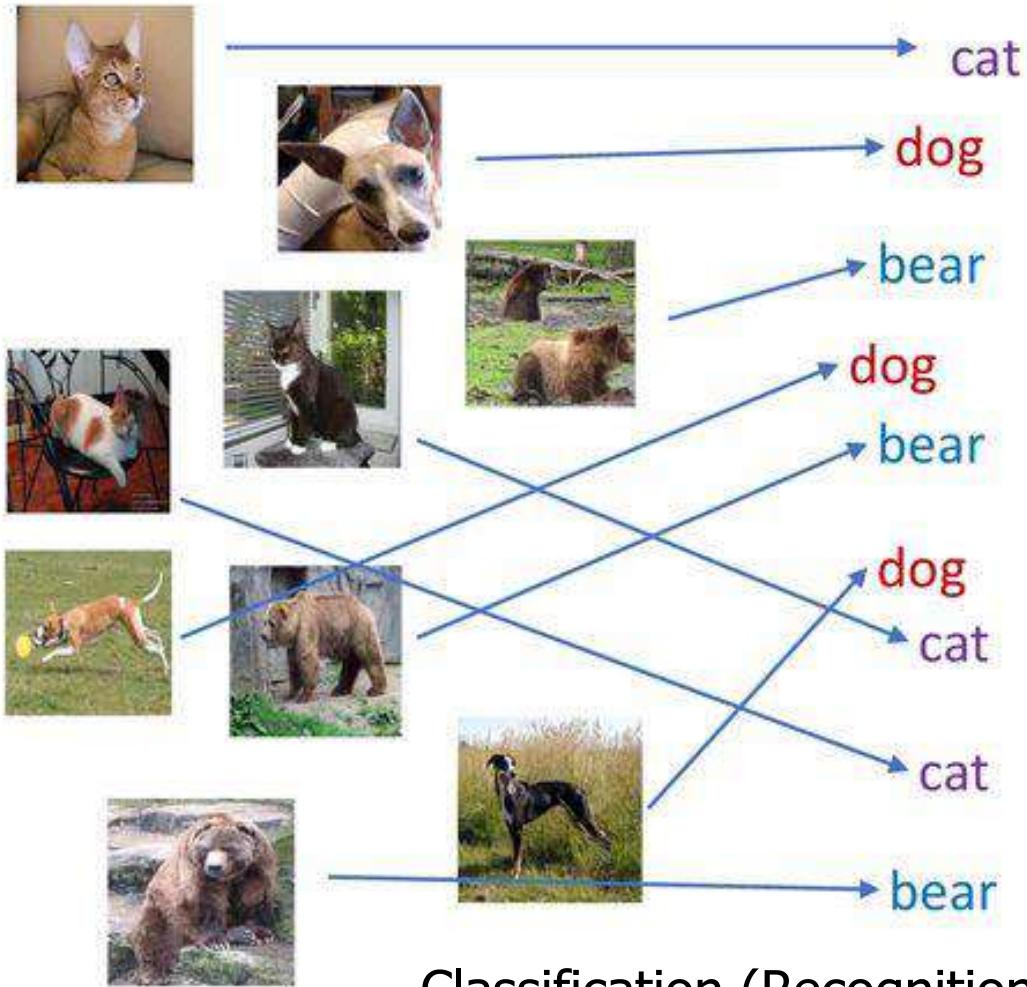
$x \rightarrow y$



Classification (Recognition)
(Supervised Classification)

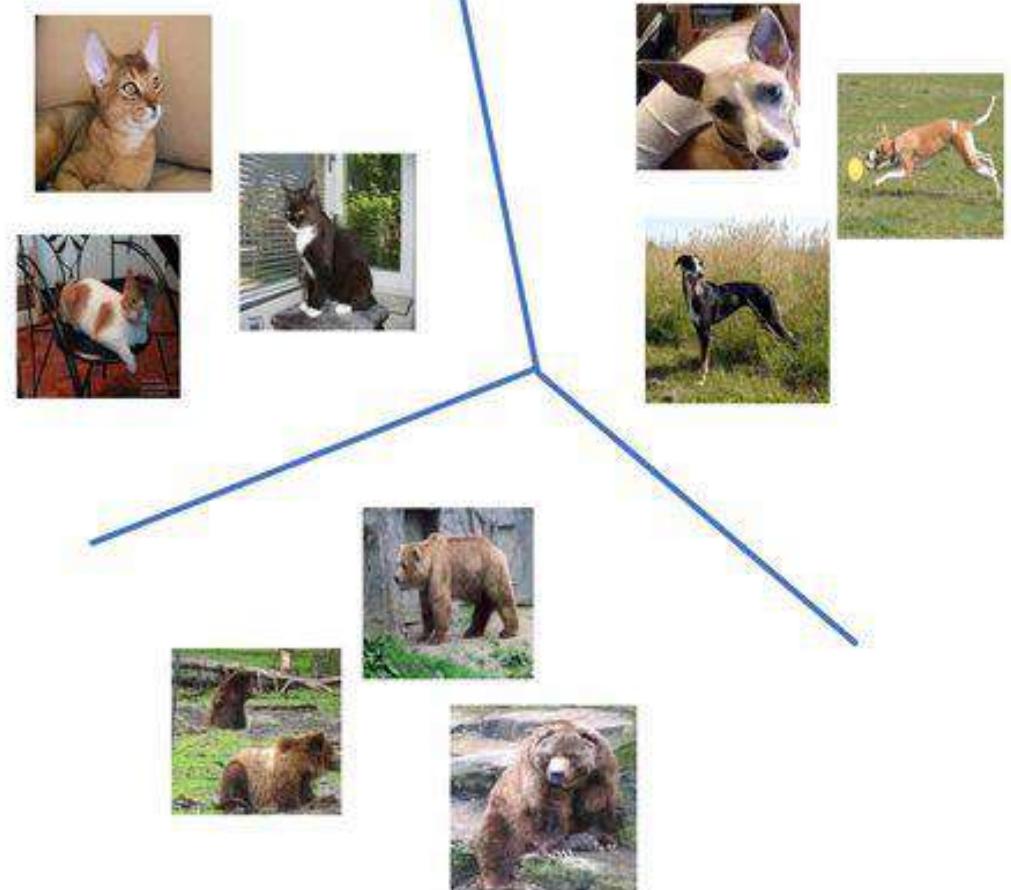
Supervised Learning vs Unsupervised Learning

$x \rightarrow y$



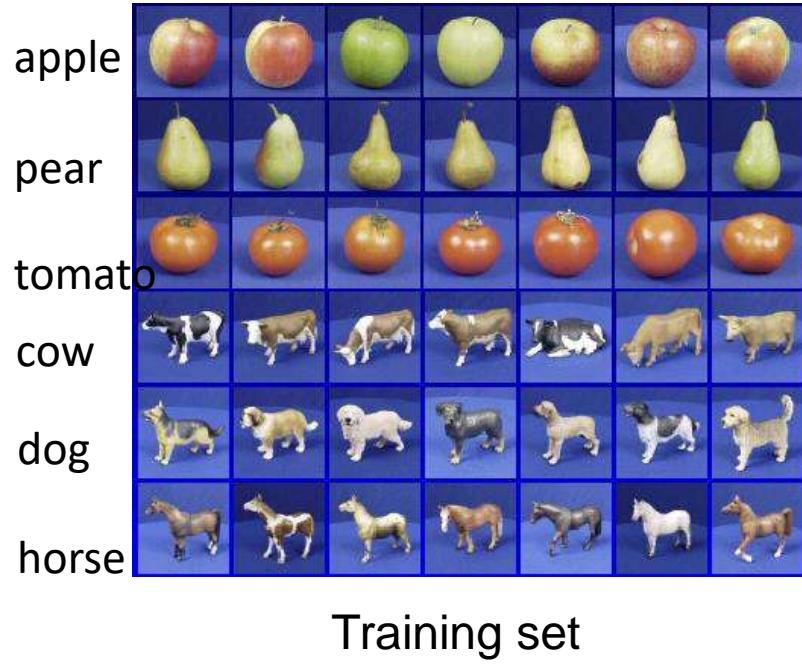
Classification (Recognition)
(Supervised Classification)

x



Clustering
(Unsupervised Classification)

Supervised Learning



$$y = f(x)$$

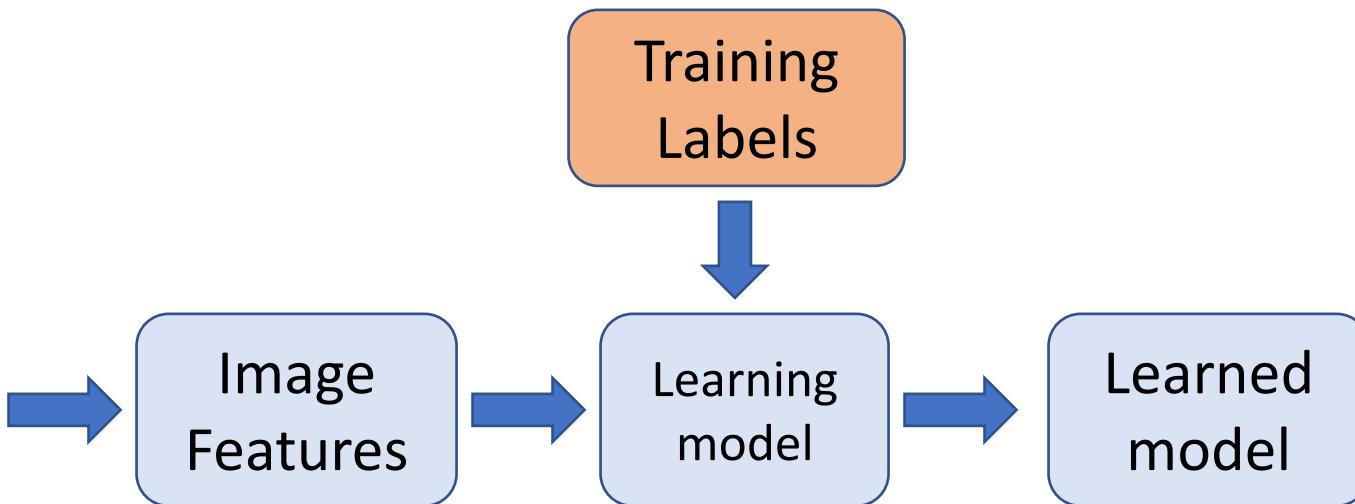
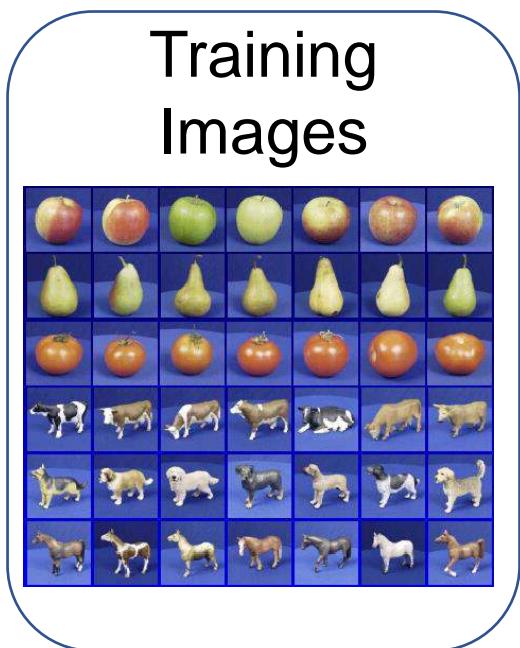
↑ output ↑ prediction function → Image

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

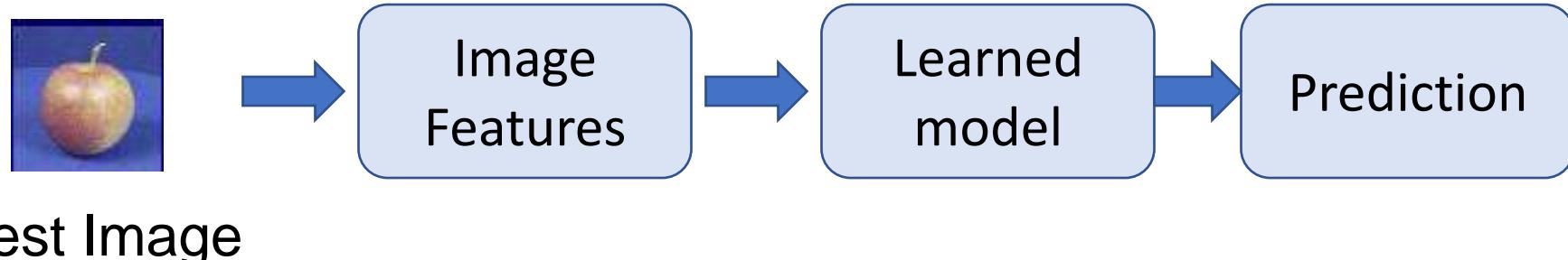
- **Training:** given a *training* set of labeled examples $\{(x_1, y_1), \dots, (x_N, y_N)\}$, estimate the prediction function f by **minimizing** the prediction **error** on the training set
- **Testing:** apply f to a never before seen *test example* x and output the predicted value $y = f(x)$
-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

Supervised Training Steps

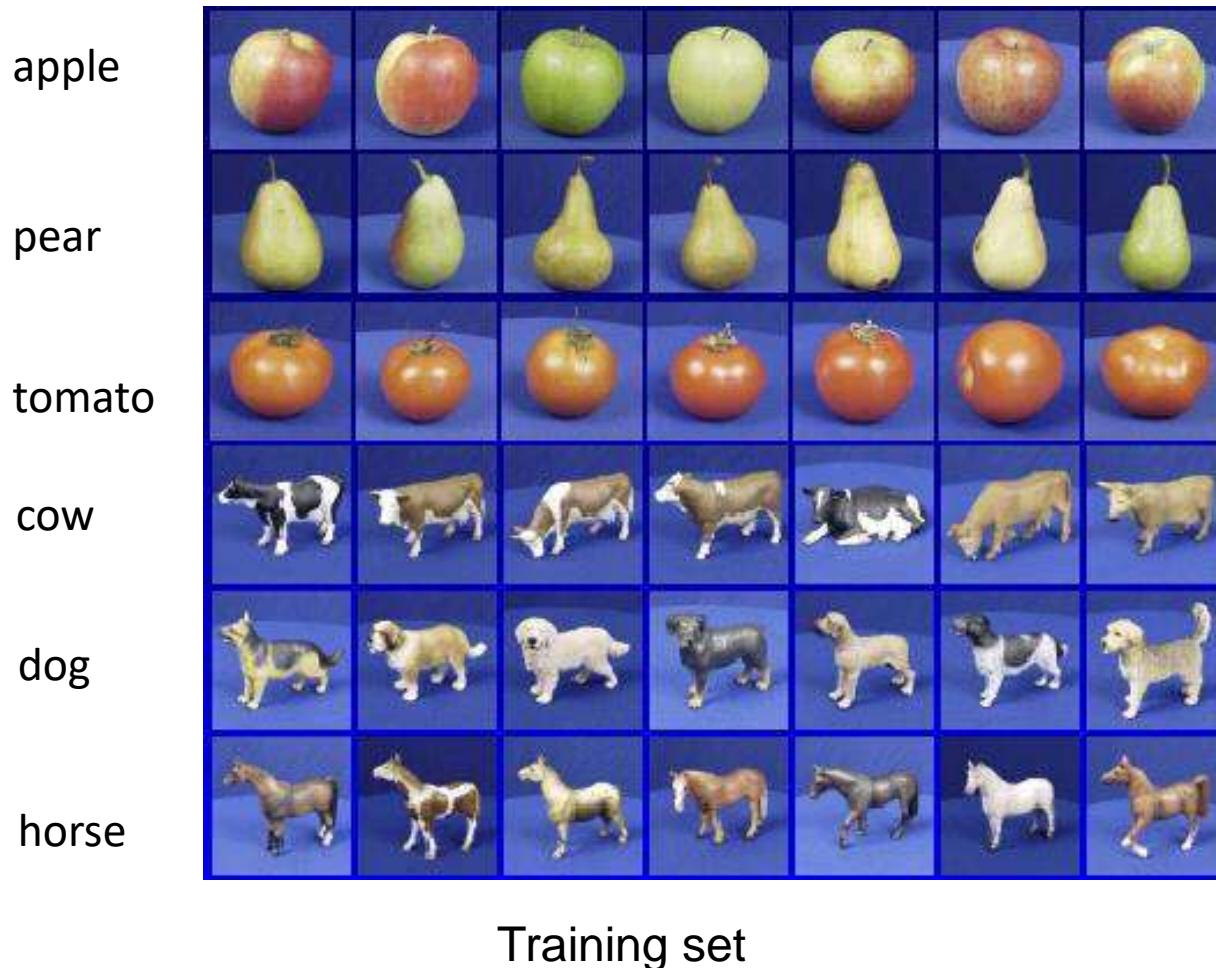
Training



Testing



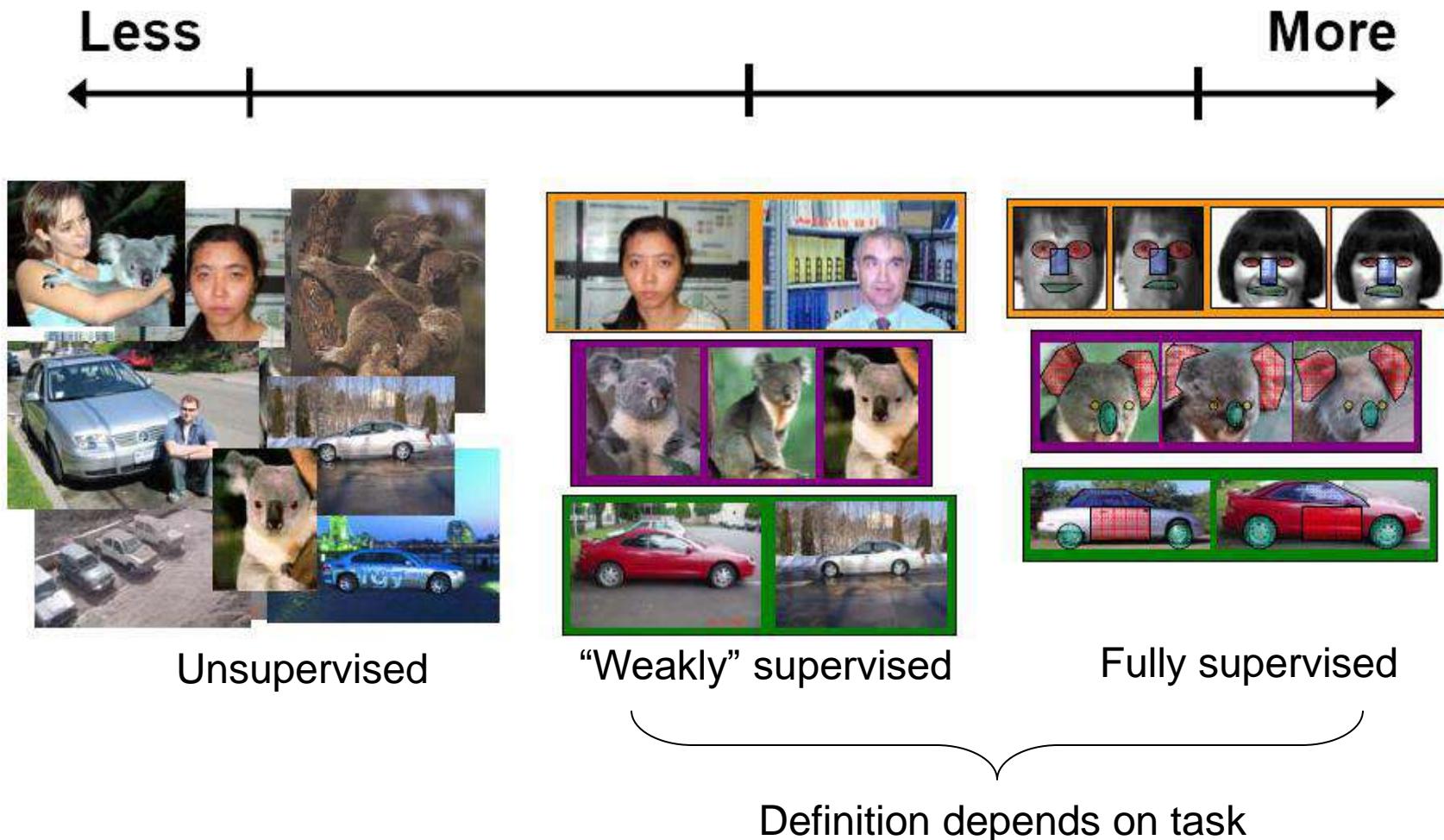
Generalization



Test set (previously unseen)

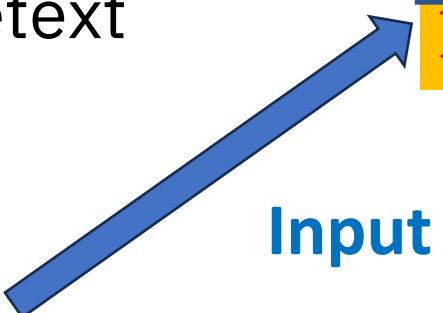
- How well does a learned model generalize from the data it was trained on to a new test set?

Spectrum of supervision



Self Supervised Learning

- A machine learning process where the model trains itself to learn one part of the input from another part of the input.
- Also known as predictive or pretext learning.
- Examples:
 - Grab a text from some existing source
 - Predict any part of the input from any other part
 - Predict the future from the past
 - Predict the future from the recent past
 - Predict the past from the present
 - Predict the top from the bottom
 - Predict the occluded from the visible



Men in sandals push bicycles overloaded with bags of coal down the highway, while on the back roads close to Hazaribagh, women carry buckets of the stuff on their heads.

Men in sandals push bicycles overloaded with bags of coal down the highway, while on the back roads close to Hazaribagh, women carry buckets of the stuff on their heads.

Output: buckets

Machine Learning Algorithms

Naïve Bayes

Bayesian network

K-nearest neighbor

SVM

Logistic regression

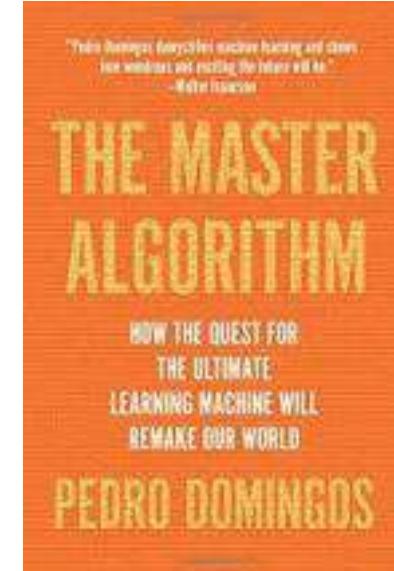
Neural networks

Reinforcement Learning

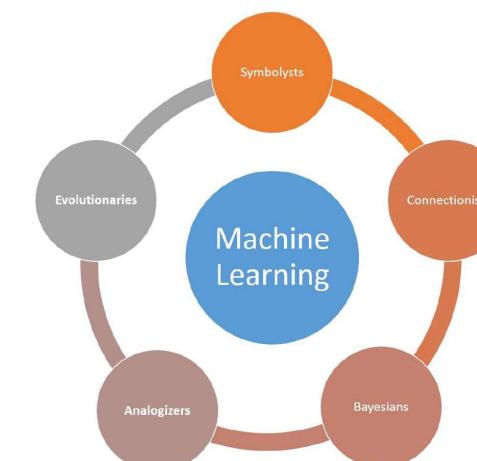
Randomized Forests

Boosted Decision Trees RBMs

Etc.

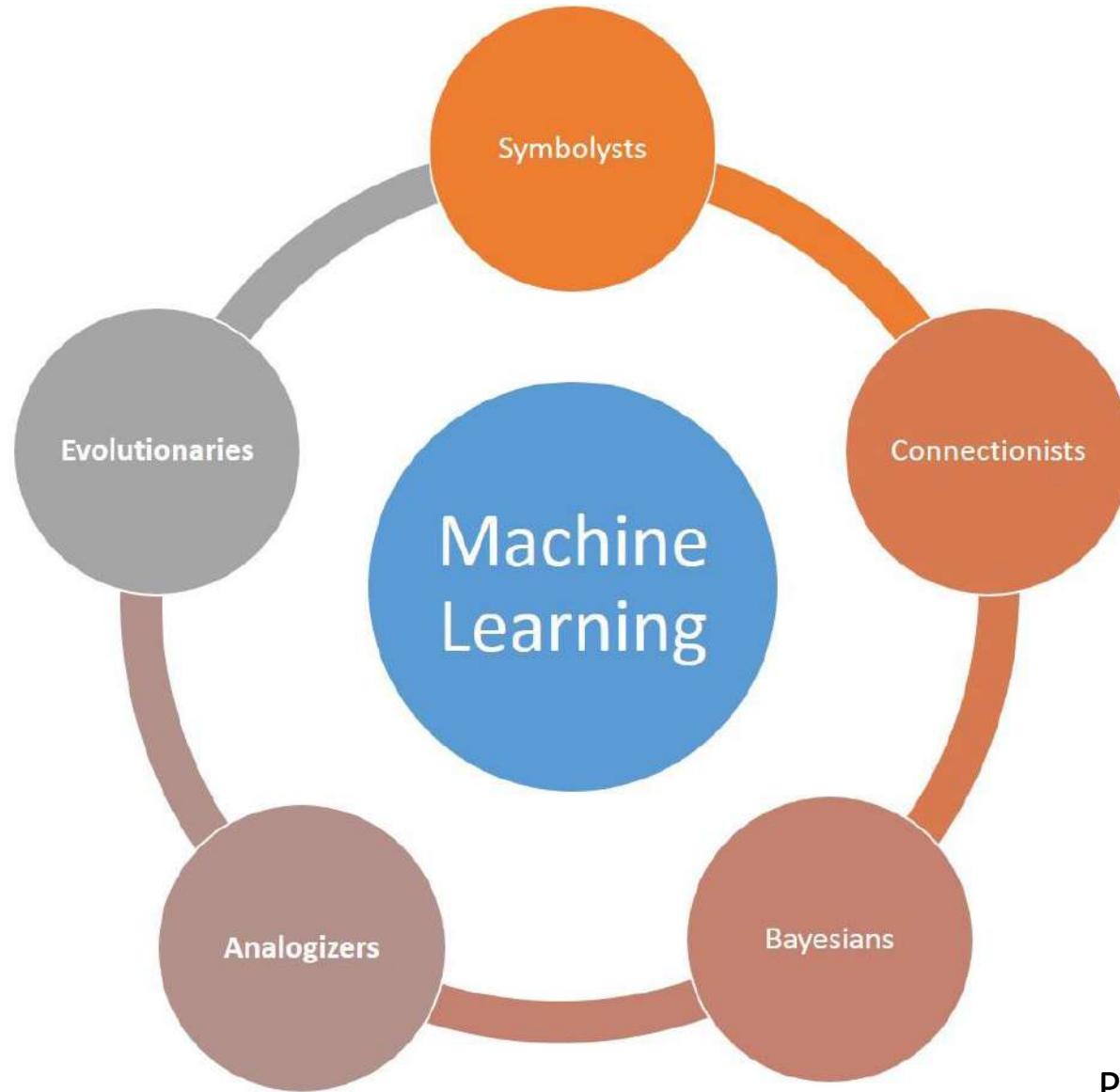


Suggested reading for entertainment: “The Master Algorithm” by Pedro Domingos



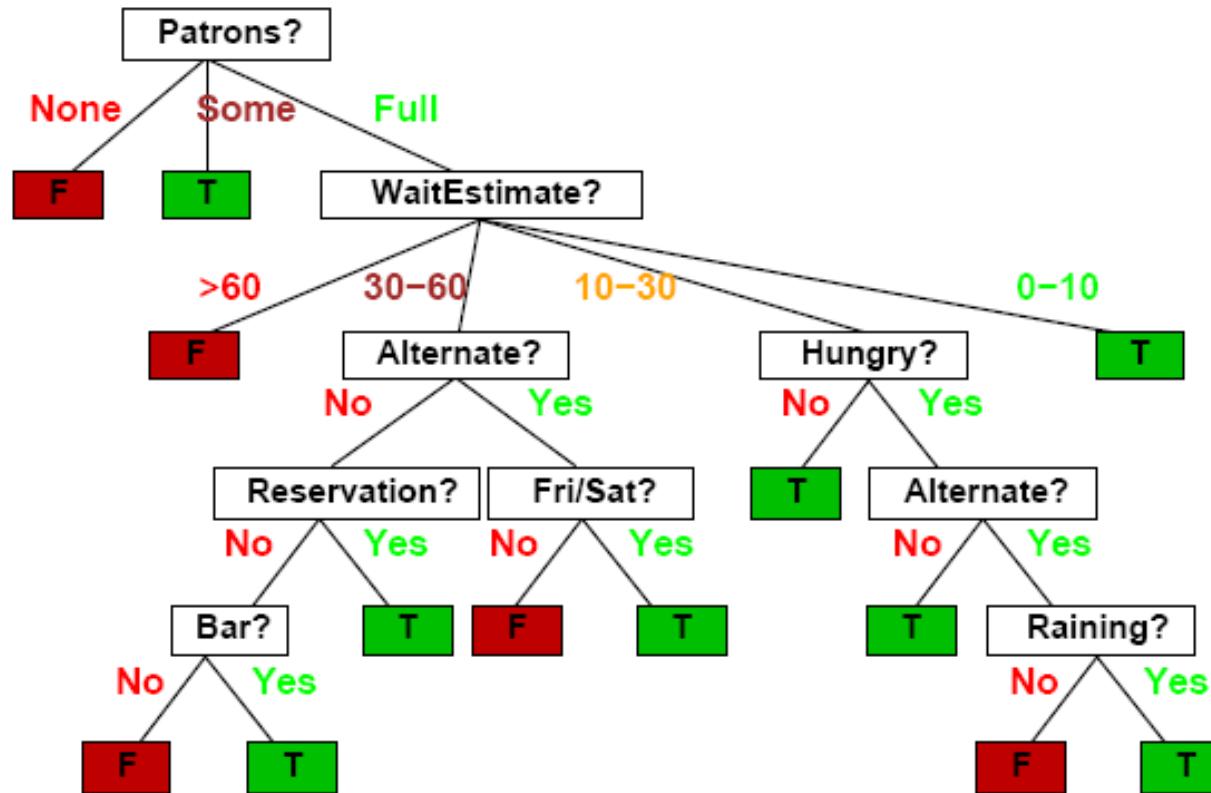
Pedro Domingo (2015), “Master Algorithm

The Five Tribes by Pedro Domingos



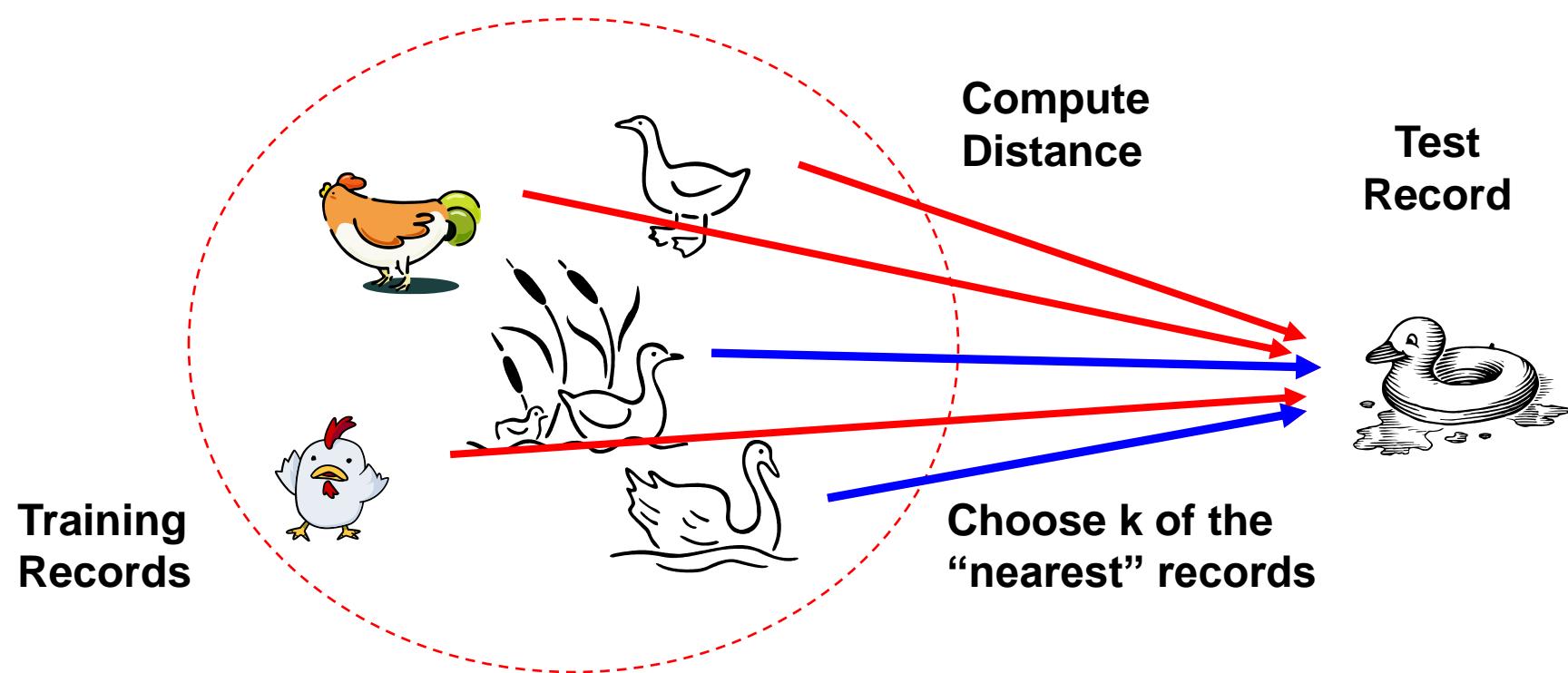
Symbolist: Decision Trees

Should I wait at this restaurant?



Analogizer: Nearest Neighbor Approach

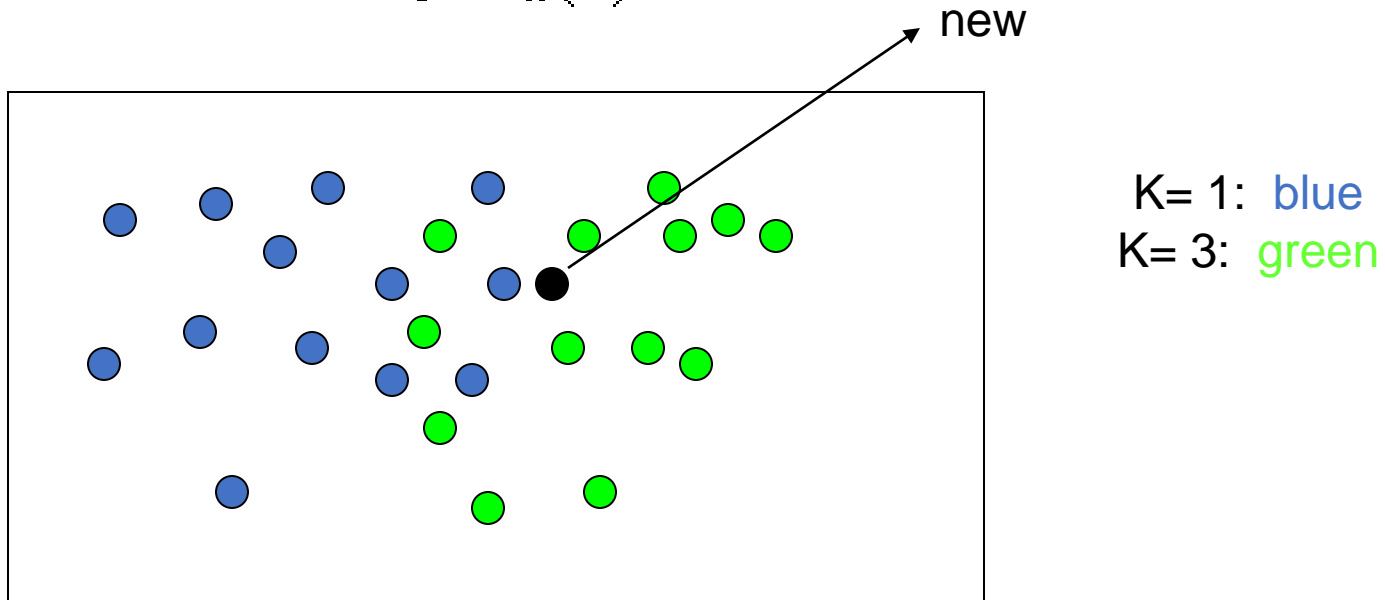
- If it walks like a duck, quacks like a duck, then it's probably a duck



Analogizer: Kth Nearest Neighbor (KNN)

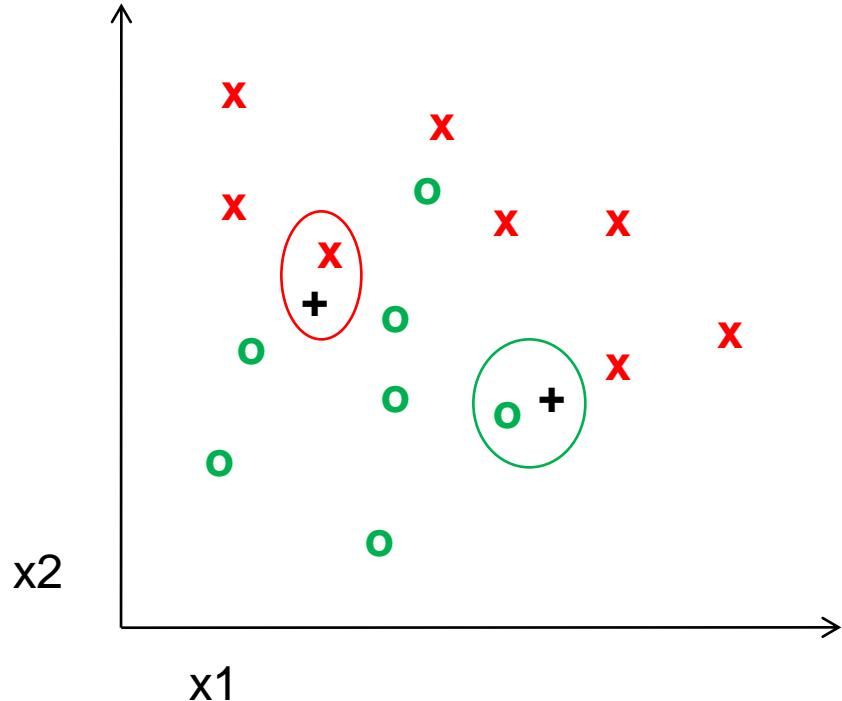
- Majority vote within the kth Nearest Neighbors (KNN)

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

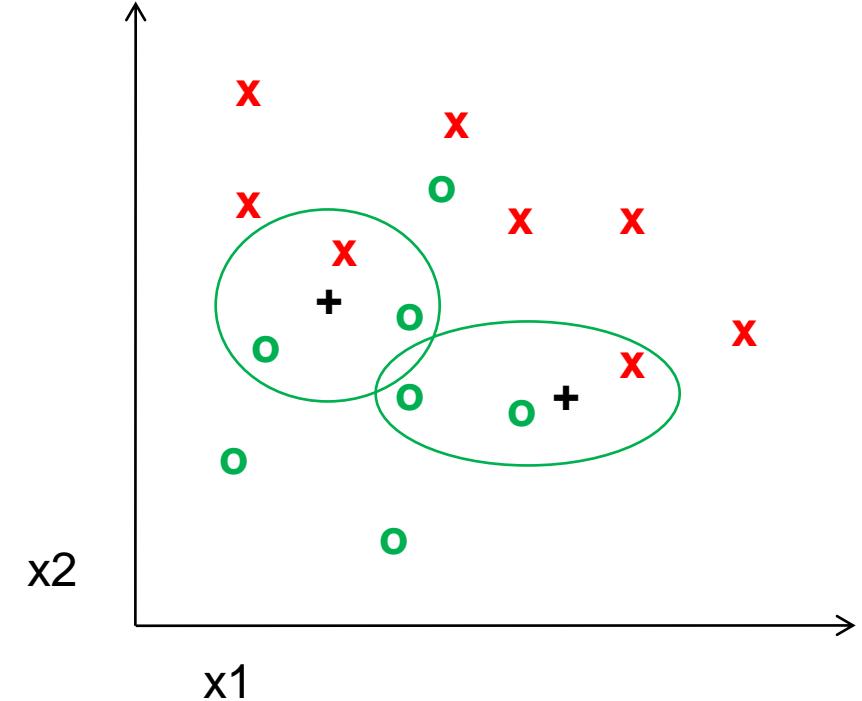


Analogizer: Kth Nearest Neighbor (KNN)

1-nearest neighbor

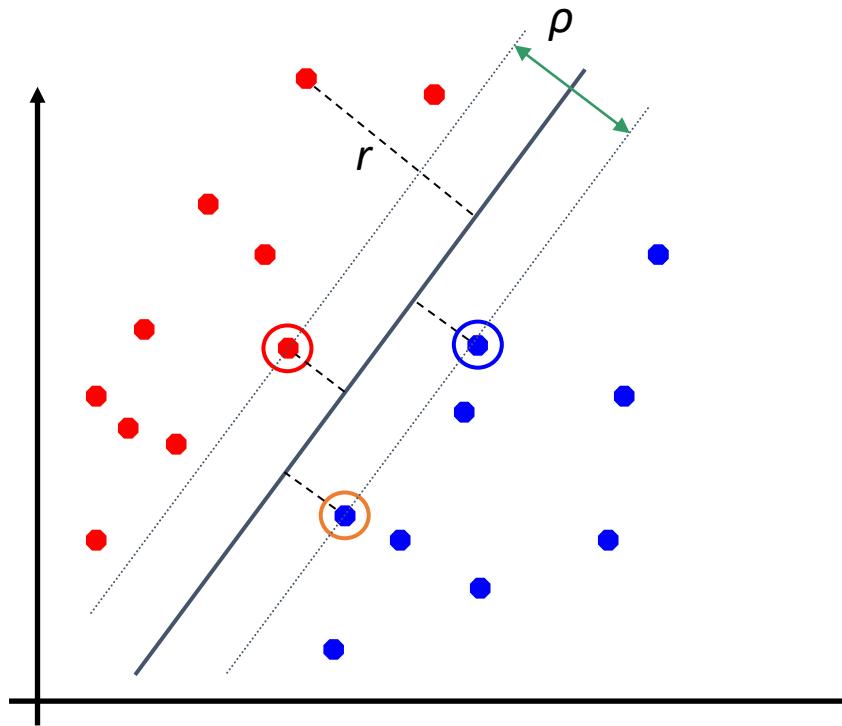


3-nearest neighbor



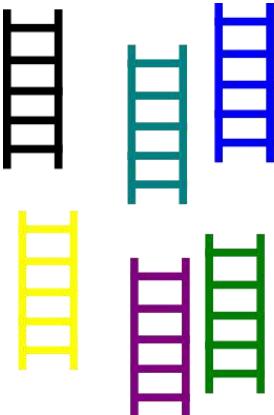
Analogizer: Support Vector Machine (SVM)

- Distance from example \mathbf{x}_i to the separator is $r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are **support vectors**.
- **Margin** ρ of the separator is the distance between support vectors.



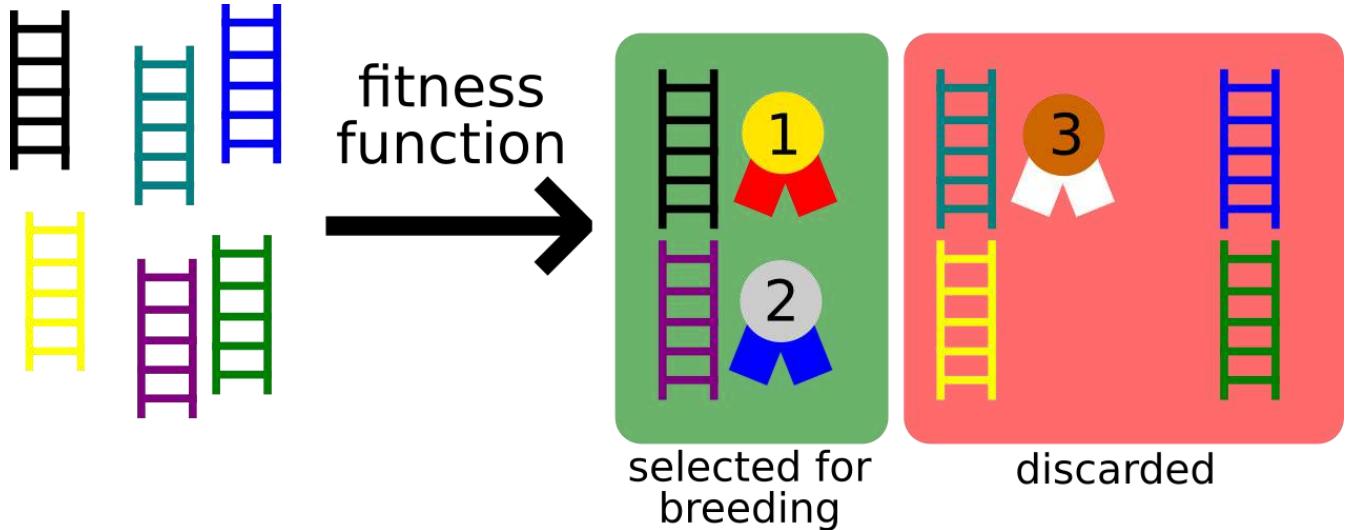
Evolutionaries

- Algorithm selected by an evolutionary process (Genetic Algorithm)
- The process as a whole is typically slow.



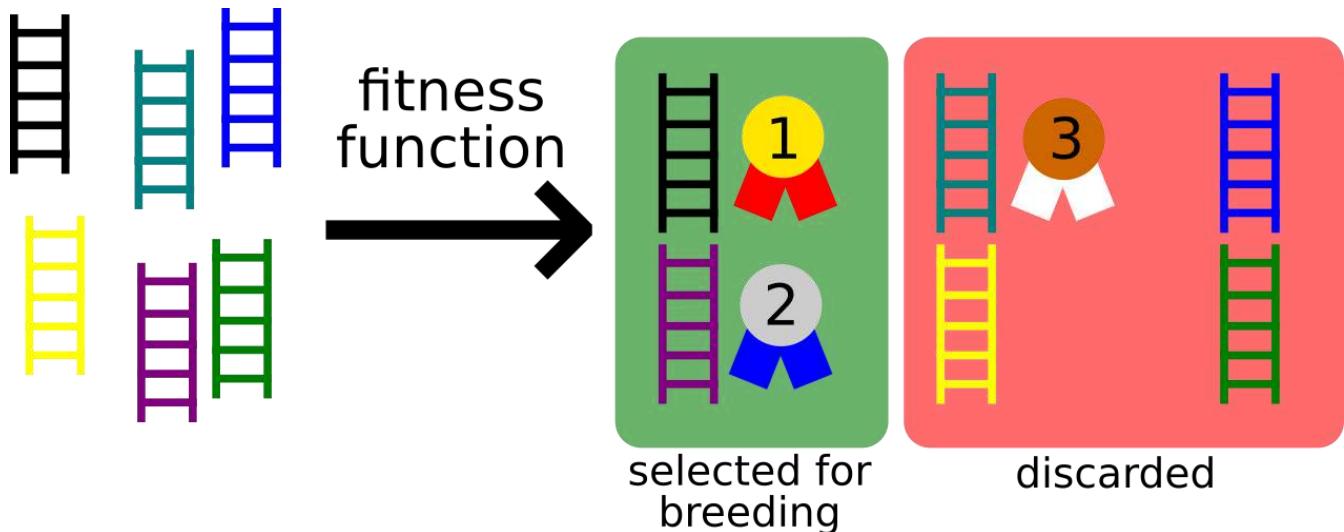
Evolutionaries

- Algorithm selected by an evolutionary process (Genetic Algorithm)
- The process as a whole is typically slow.

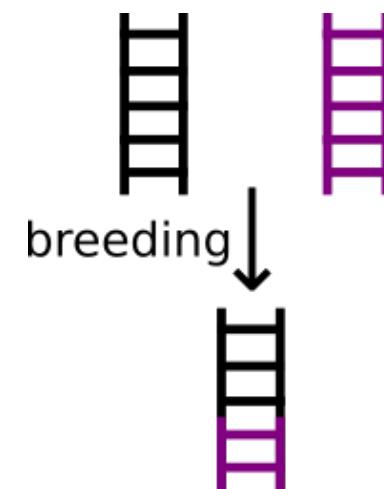


Evolutionaries

- Algorithm selected by an evolutionary process (Genetic Algorithm)

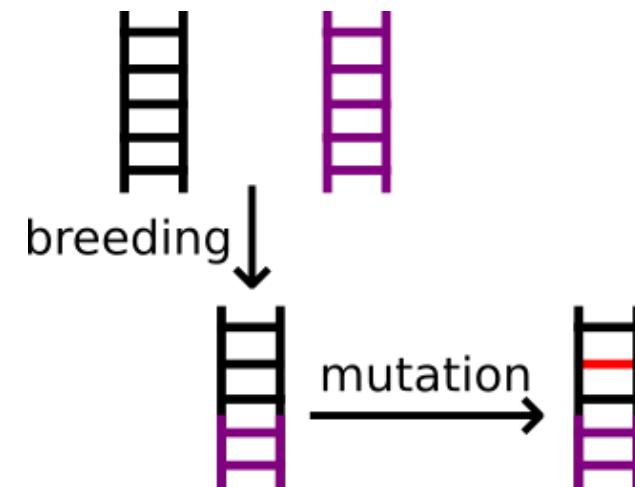
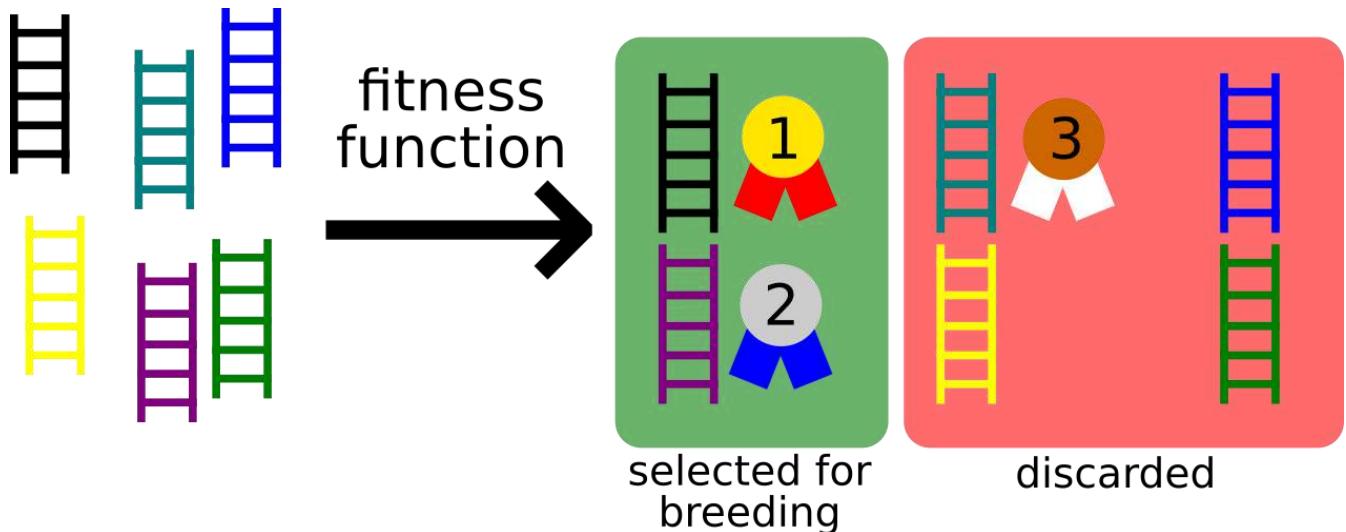


- The process as a whole is typically slow.



Evolutionaries

- Algorithm selected by an evolutionary process (Genetic Algorithm)
- The process as a whole is typically slow.



Bayesian: Probabilistic approach

- Bayes' theorem: Outputs the most probable hypothesis $w \in W$, given the **data x** and **knowledge about prior probabilities** of hypotheses in w

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

- $p(w/x)$: posterior probability of w . probability that w holds given **data x** ; confidence that w **holds given x**
- $p(w)$: **prior probability of w** . Background knowledge we have about that w is a correct hypothesis
- $p(x)$: prior probability that training data x will be observed
- $p(x/w)$: **likelihood of x** . probability of observing x given w holds



Thomas Bayes
1701 – 7 April 1761

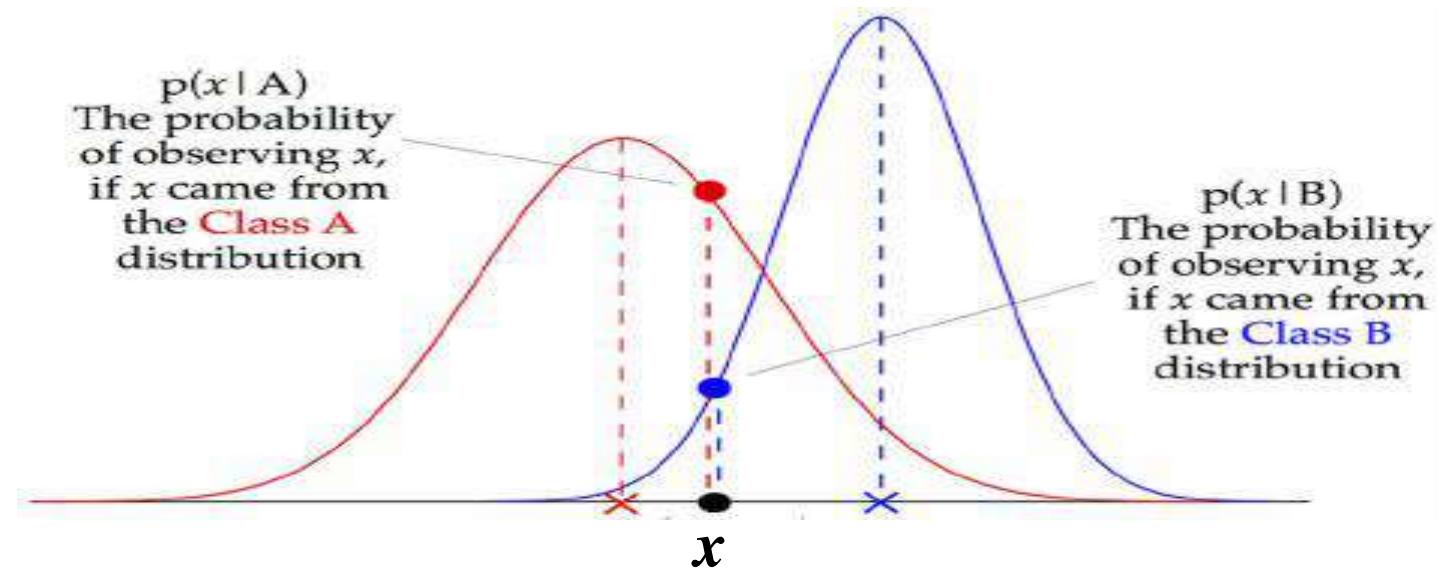
Bayesian

- Bayes theorem allows converting a priori estimate to posteriori (e.g. measurement conditioned) probability density function (pdf) of state of nature via:

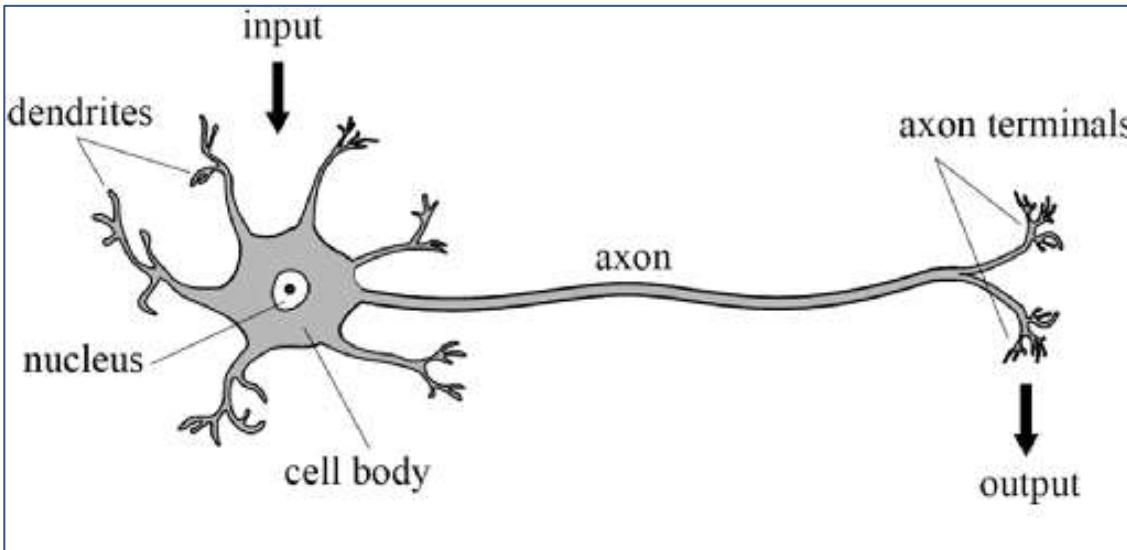
$$P(\text{salmon} | \text{width}) = \frac{[p(\text{width} | \text{salmon})P(\text{salmon})]}{p(\text{width})}$$

From the training set,
Posterior Probability of
salmon for a given width can
be generated

$$P(\text{salmon} | \text{width})$$



Connectionist: Neuronal Network Approach

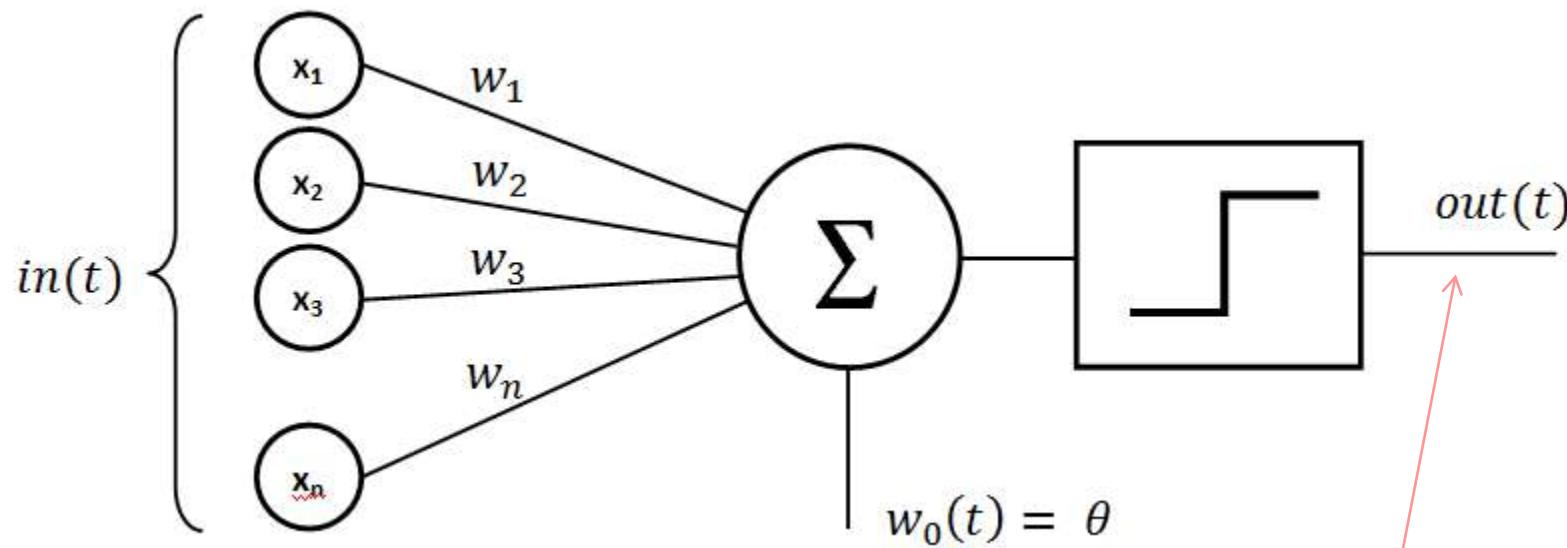


- A neuron has
 - Branching input (dendrites)
 - Branching output (the axon)
- Information moves from the dendrites to the axon via the cell body
- Axon connects to dendrites via synapses
 - Synapses vary in strength
 - Synapses may be excitatory or inhibitory

Neves, et. al. (2018), "A new approach to damage detection in bridges using machine learning"

Basic Perceptron

(Frank Rosenblatt, 1950s and early 60s)

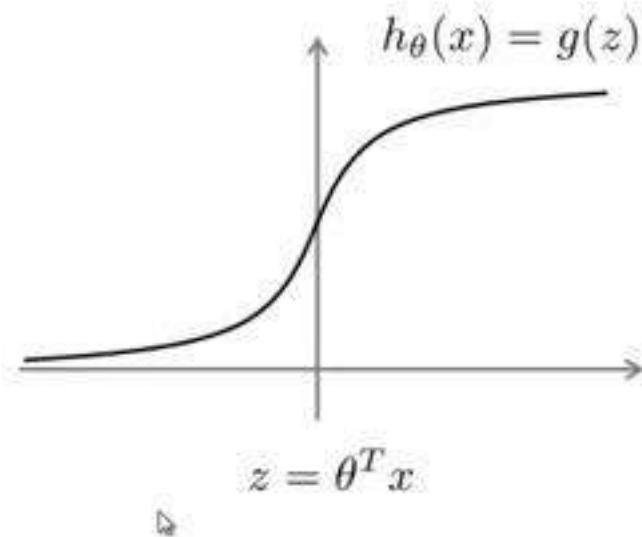


- An Artificial Neuron (AN) is a non-linear parameterized function with restricted output range

$$O = \begin{cases} 1 & \text{if } \sum_i w_i x_i + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Activation Functions – Sigmoid Function

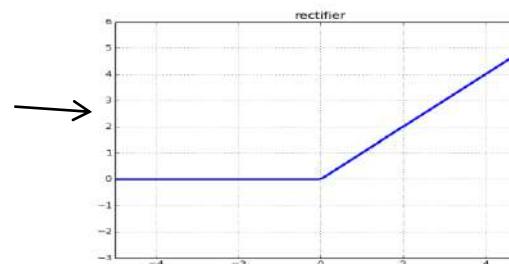
$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$



Recent successes (e.g., Baidu Deep Speech) use
(clipped) Rectifier Linear Units:

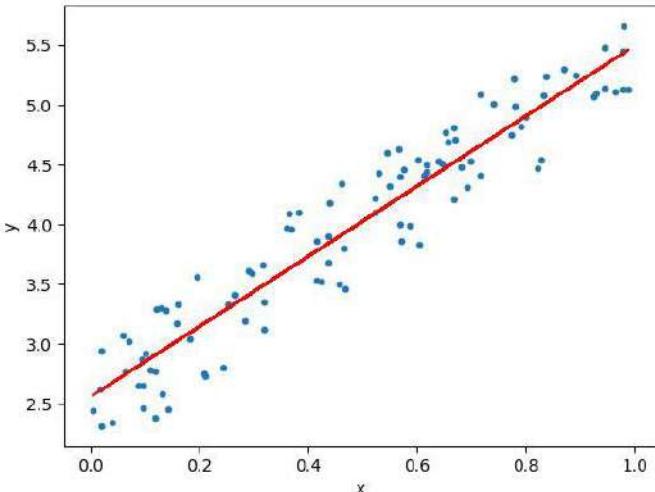
$$f(x) = \max(0, x) \text{ -- rectifier}$$

$$f(x) = \min(\max(0, x), clip)$$



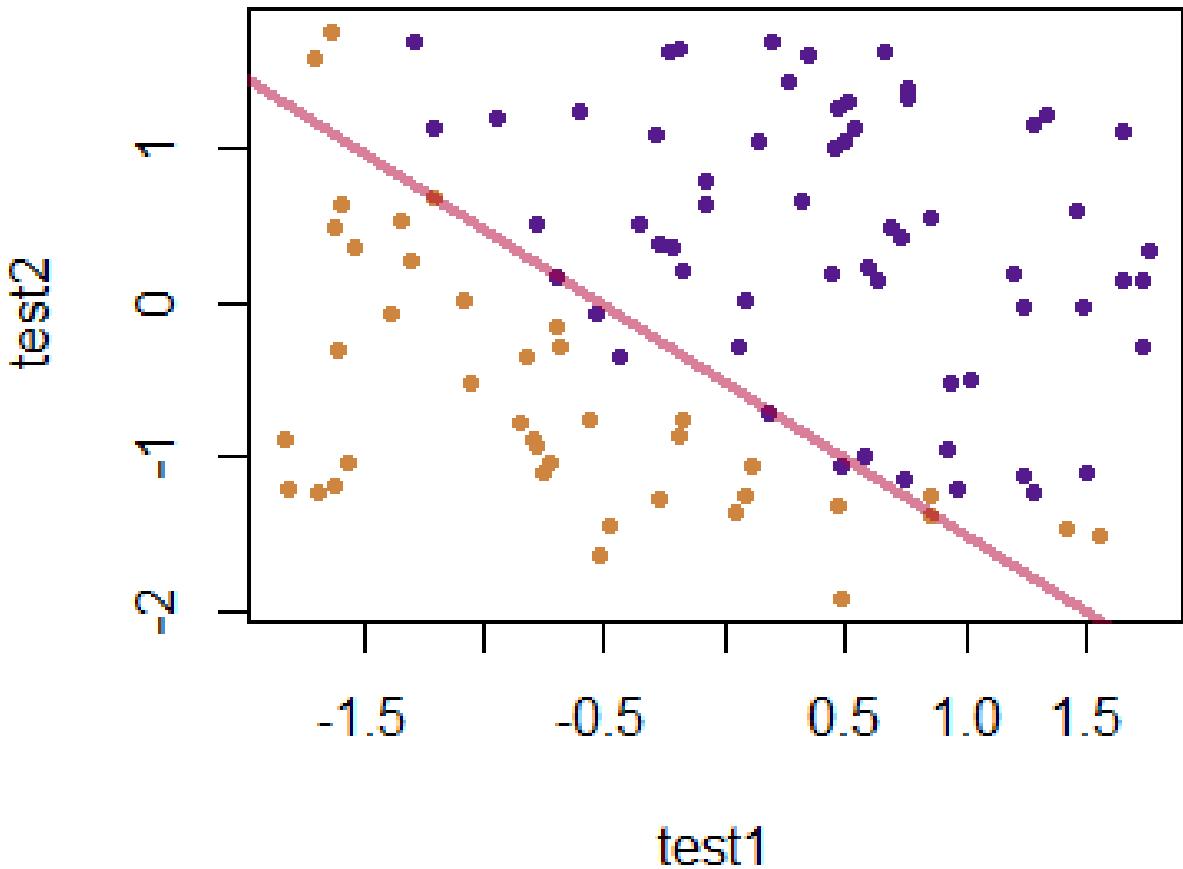
Neural Network for Classification

- Separates different classes
- Logistic Regression (Perceptron)

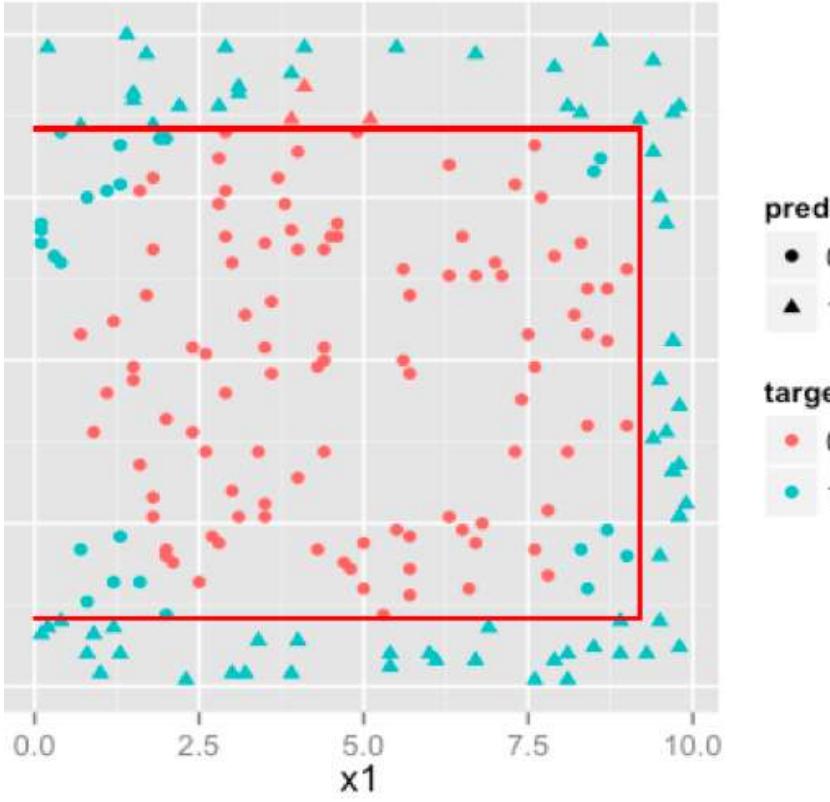


Linear Regression

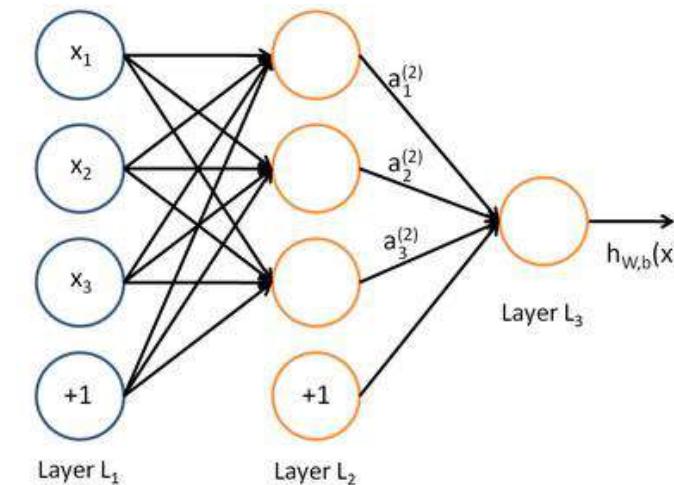
College admissions



For More Complex Boundaries

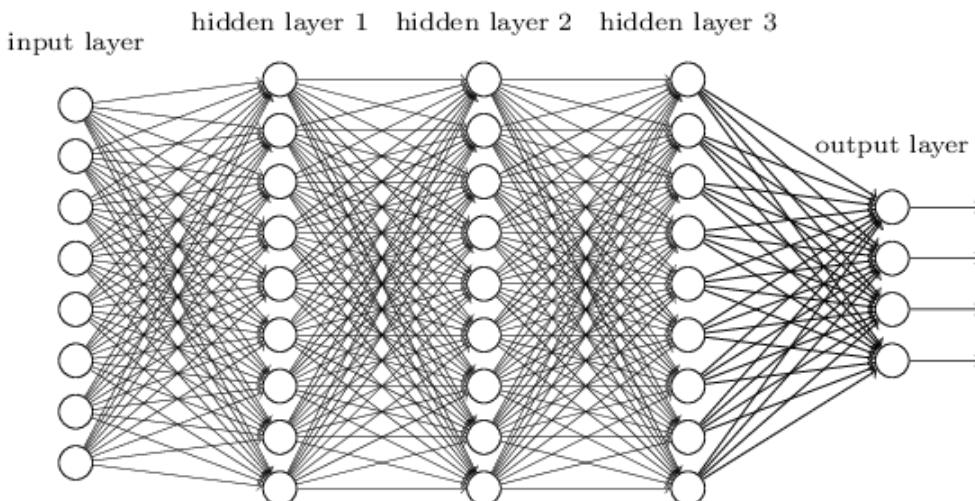


- More complex boundaries can be constructed using multilayer perceptrons
 - Add additional linear boundaries by adding a hidden layer

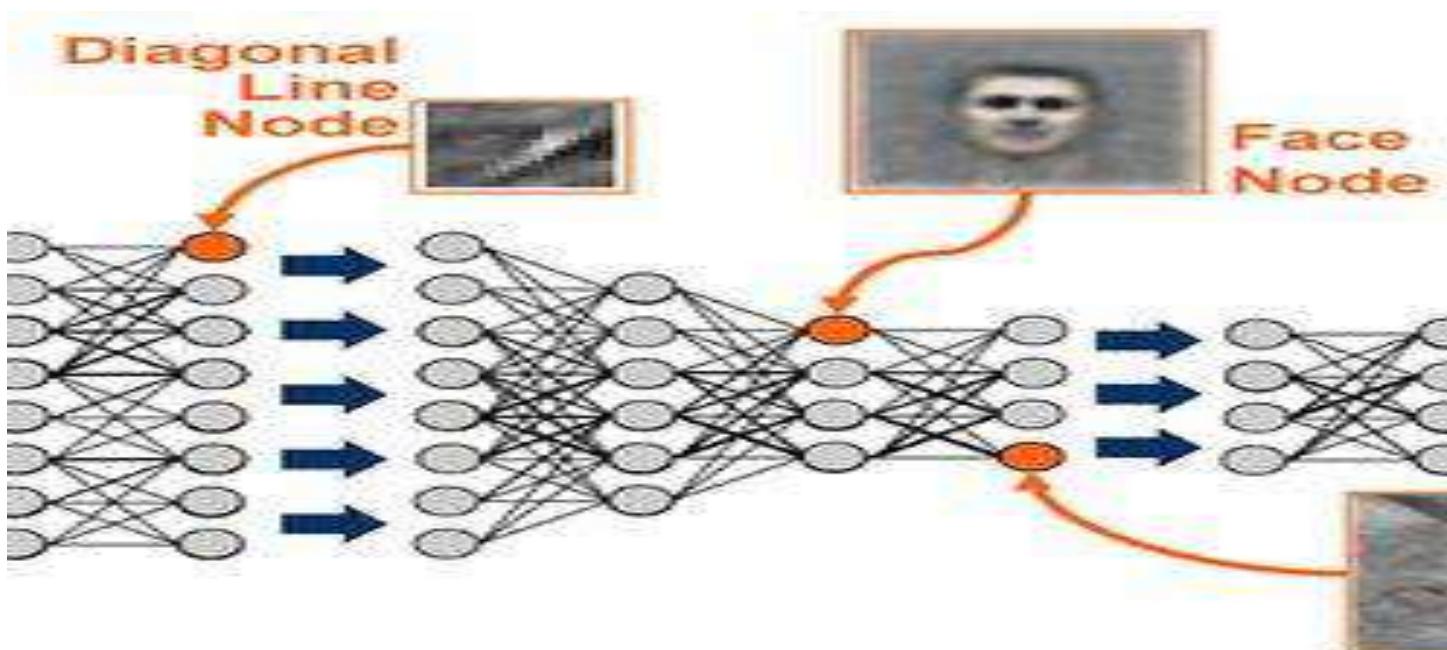


Multi-layer Perceptron

Deep Learning/Deep Neural Net (DNN)



- Not Possible Previously Because:
- Requires a large set of data
- Involves a large set of weight computation
- Vanishing Gradient Problem



Convolutional Neural Net

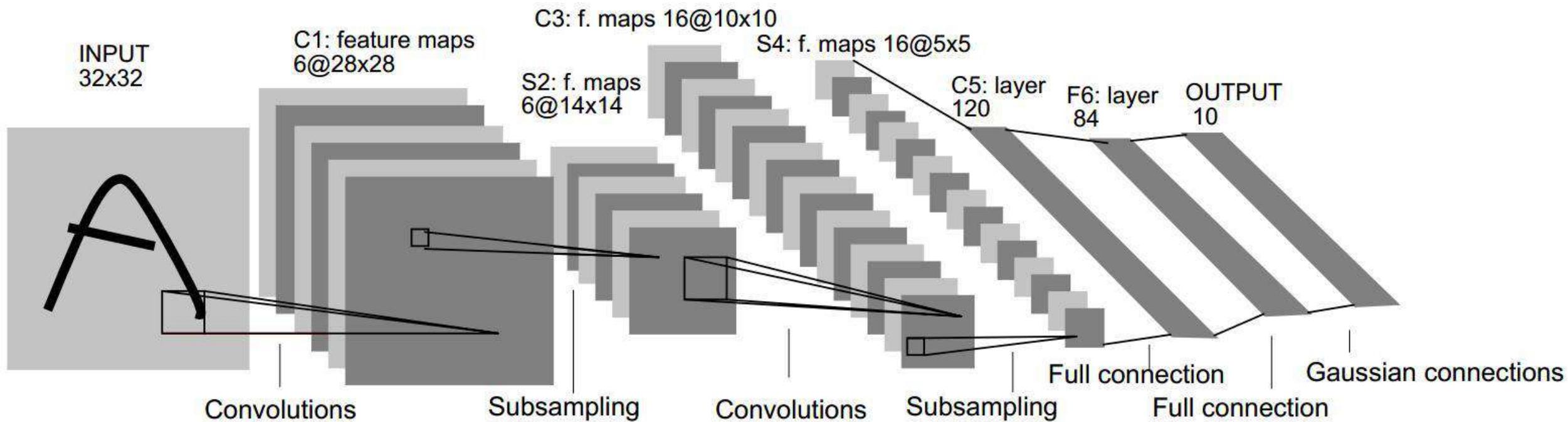


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Machine Learning Tasks

Computer Vision Use Cases

Image Classification



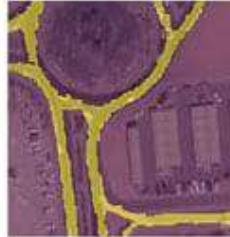
Object Detection



Semantic Segmentation

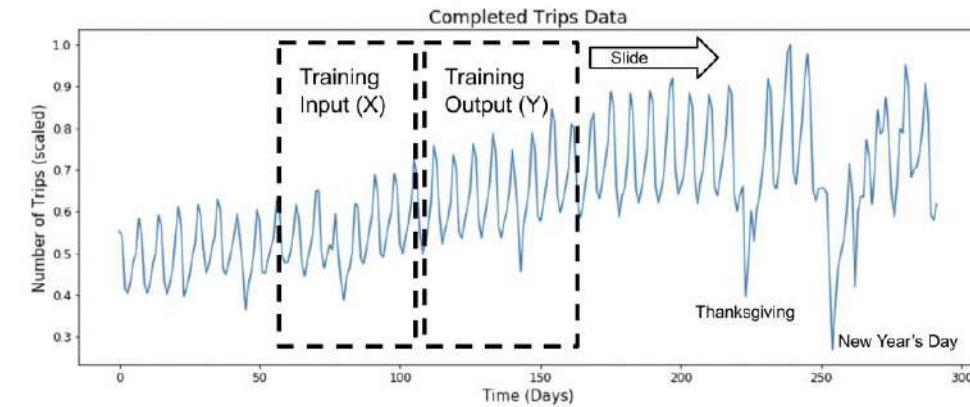
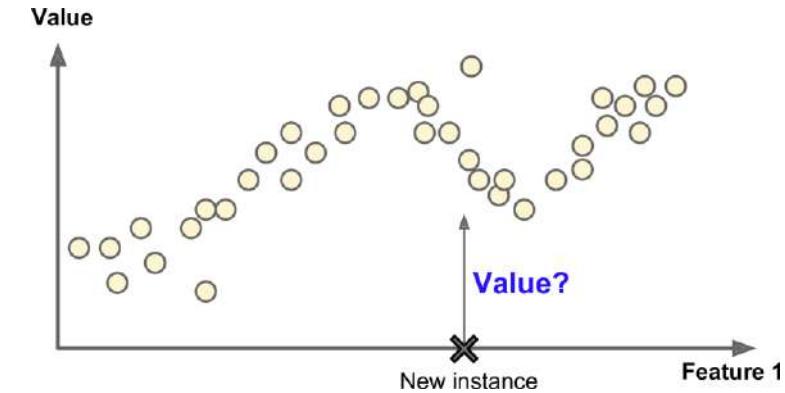


Instance Segmentation



"Where Deep Learning Meets GIS" by Rohit Singh, June 2019

Regression



"Creating a Model for Weather Forecasting Using Linear Regression" by Ashan Lakmal

Review of Probability and Statistics

ECE 610

**David Han
Drexel University**

Probability in our lives



"And now the 7-day forecast..."



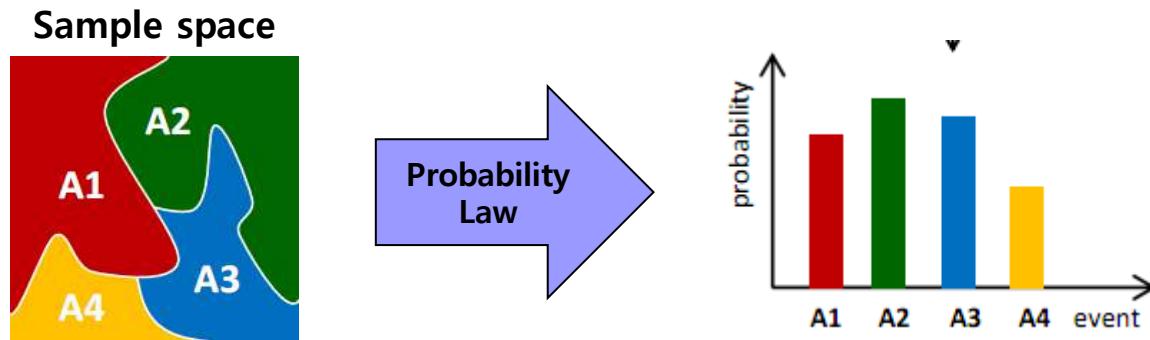
StudiousGuy, "8 Real Life Examples of Probability," <https://studiousguy.com/8-real-life-examples-of-probability/>



The Guardian (2020), "As restaurants reopen, what will eating out be like in the age of coronavirus us?" <https://www.theguardian.com/us-news/2020/jun/02/restaurants-reopening-eating-out-coronavirus-safety>

Basic Probability Concepts

- Definitions (informal)
- Probabilities are **numbers assigned** to events that indicate “*how likely*” it is that the **event will occur** when a random experiment is performed
- The **sample space S** of a random experiment is the **set of all possible outcomes**

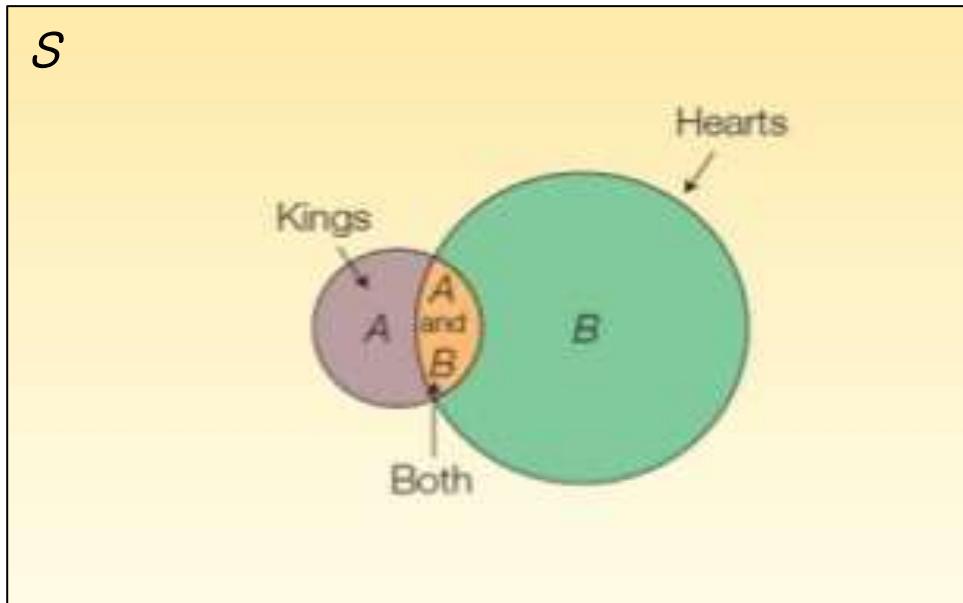


- **Axioms of probability**
- **Axiom I :** $0 \leq P[A_i]$
- **Axiom II :** $P[S] = 1$
- **Axiom III :** if $A_i \cap A_j = \emptyset$, then $P[A_i \cup A_j] = P[A_i] + P[A_j]$

Joint Probability

- Assume that there are two events, A and B , on the sample space S .
- Joint probability
 - The probability for two events which intersect in the sample space. (events that occur together)
 - Example: a deck of cards

$$P(A \cap B) = P(A) + P(B) - P(A \cup B)$$
$$\Leftrightarrow P(A \cup B) = P(A) + P(B) - P(A \cap B) \leq P(A) + P(B)$$



More Properties of Probability

□ **Property 1 :** $P[A^C] = 1 - P[A]$

□ **Property 2 :** $P[A] \leq 1$

□ **Property 3 :** $P[\emptyset] = 0$

□ **Property 4 :** given $\{A_1, A_2, \dots, A_N\}$, if $\{A_i \cap A_j = \emptyset, \forall i, j\}$, then $P\left[\bigcup_{k=1}^N A_k\right] = \sum_{k=1}^N P[A_k]$

□ **Property 5 :** $P[A_1 \cup A_2] = P[A_1] + P[A_2] - P[A_1 \cap A_2]$

□ **Property 6 :** if $A_1 \subset A_2$, then $P[A_1] \leq P[A_2]$

Note: $\bigcup_{n=1}^N A_n \equiv A_1 \cup A_2 \cup \dots \cup A_N$

Probability as a Relative Frequency

- Probability is defined as a “*relative frequency*” of occurrence of some event
- If the “fair” coin is flipped many times (say n) and heads shows up n_H times out of the n flips, then

$$\lim_{n \rightarrow \infty} (n_H / n) = P(H)$$

- Here, the ratio (n_H / n) is the *relative frequency* (or average number of successes) for this event.
- *statistical regularity*
 - relative frequencies → a fixed value (a probability) as $n \rightarrow$ large.
 - Based on observations

Basic Probability Concepts

- An experiment consist of obtaining a number x by spinning the pointer on a “fair” wheel of chance that is labeled from 0 to 100 points.
 - $S = \{0 < x \leq 100\}$

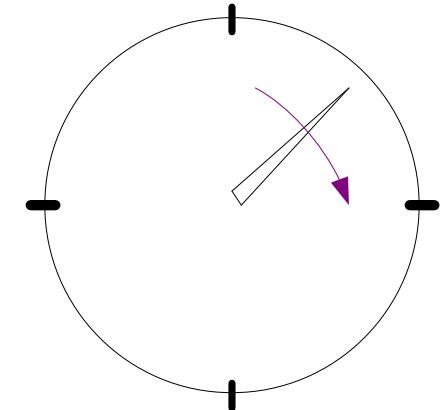
- Probability of the pointer falling between any two numbers

$$x_2 \geq x_1$$

$$P(x_1 < x \leq x_2) = \frac{x_2 - x_1}{100}$$

- Axiom 1 : Because of $x_2 \geq x_1$, Ok!
- Axiom 2 : Applied to $x_2=100$ and $x_1=0$
- Axiom 3

- Break the wheel's periphery into N contiguous segment A_n



$$P\left(\bigcup_{n=1}^N A_n\right) = \sum_{n=1}^N P(A_n) = \sum_{n=1}^N \frac{1}{N} = 1 = P(S)$$

Basic Probability Concepts

- Observe the sum of the number showing up two dice that is thrown.

- Sample space

- Total $36(=6^2)$ points.

(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)	<i>A</i>
(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)	<i>B</i>
(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)	
(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)	
(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)	
(6, 1)	(6, 2)	(6, 3)	(6, 4)	(6, 5)	(6, 6)	<i>C</i>

- Definition of events

- $A = \{sum = 7\}$, $B = \{8 < sum \leq 11\}$, and $C = \{10 < sum\}$

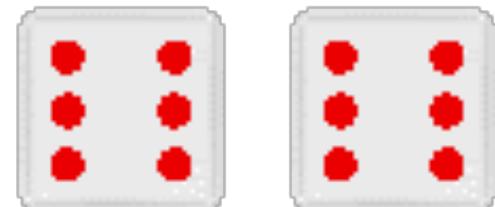
- $A_{ij} = \{\text{sum for outcome } (i, j) = i + j\}$

- where i is row location, j is the column location in the table.

- Probability

- $P(A) = P\left(\bigcup_{i=1}^6 A_{i,7-i}\right) = \sum_{i=1}^6 P(A_{i,7-i}) = 6\left(\frac{1}{36}\right) = \frac{1}{6}$

$$P(B) = 9\left(\frac{1}{36}\right) = \frac{1}{4}, \quad P(C) = 3\left(\frac{1}{36}\right) = \frac{1}{12}$$



Examples

- There are 80 resistor.

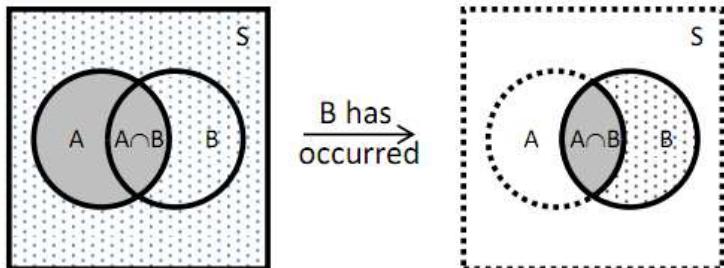
Ohm(Ω)	10 Ω	22 Ω	27 Ω	47 Ω
Num.	18	12	33	17

- $P(\text{draw } 10\Omega) = 18/80$, $P(\text{draw } 22\Omega) = 12/80$
 $P(\text{draw } 27\Omega) = 33/80$, $P(\text{draw } 47\Omega) = 17/80$
- Suppose a 22Ω resistor is drawn from the box at first, what are the probabilities of drawing a second resistor of any one of the four values?
 - $P(.| \Omega)$: The probability on the second drawing are now *conditional* on the outcome of the first drawing.

$$P(\text{draw } 10\Omega | 22\Omega) = 18/79, \quad P(\text{draw } 22\Omega | 22\Omega) = 11/79$$
$$P(\text{draw } 27\Omega | 22\Omega) = 33/79, \quad P(\text{draw } 47\Omega | 22\Omega) = 17/79$$

Conditional Probability

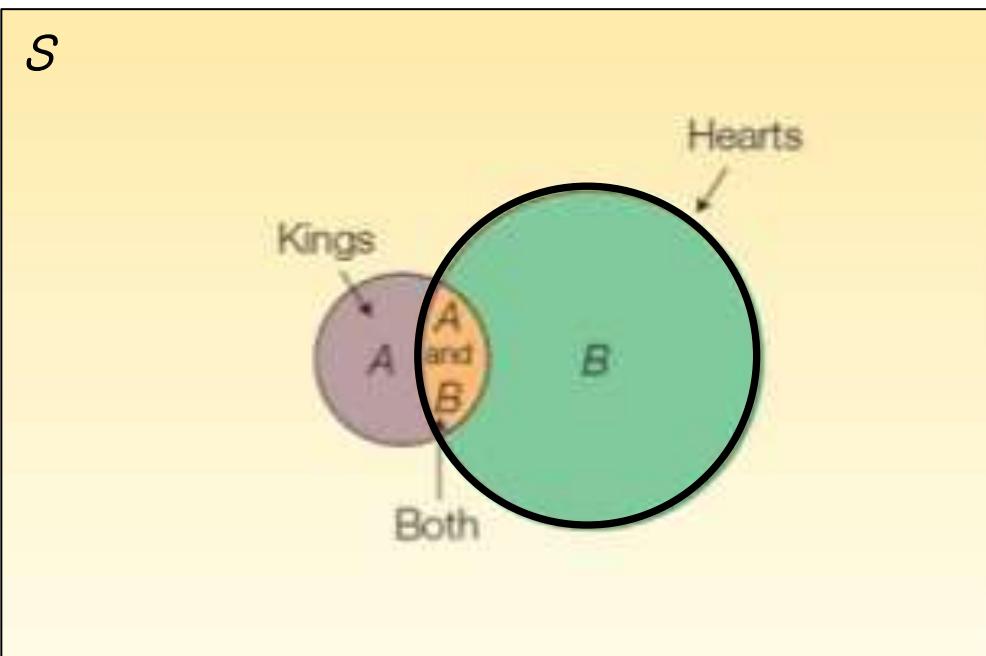
- If A and B are two events, the probability of event A when we already know that event B has occurred is **P[A|B]**
 - the "conditional probability of A conditioned on B", or simply
 - the "probability of A given B"
- Interpretation
 - The new evidence "B has occurred" has the following effects
 - The original sample space S (the square) becomes B (the rightmost circle)
 - The event A becomes $A \cap B$
 - $P[B]$ simply re-normalizes the probability of events that occur jointly with B



$$P[A|B] = \frac{P[A \cap B]}{P[B]} \text{ for } P[B] > 0$$

Conditional Probability Example

- ❑ Conditional probability in a card game
- ❑ Event *B*: hearts
- ❑ Event *A*: kings
 - The conditional prob. of drawing a king, given the cards are all hearts



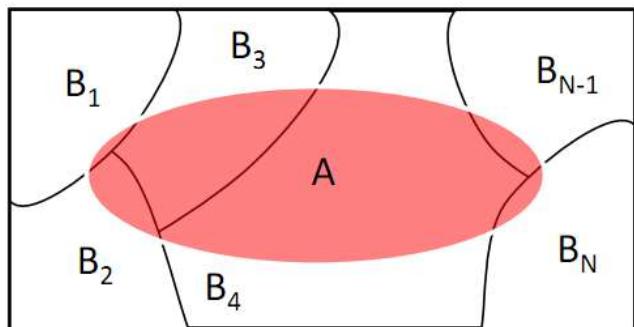
$$P(A|B) \equiv \frac{P(A \cap B)}{P(B)}$$

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

Theorem of Total Probability

- Let B_1, B_2, \dots, B_N be a partition of S (mutually exclusive that totals to S)
- Any event A can be represented as

$$A = A \cap S = A \cap (B_1 \cup B_2 \cup \dots \cup B_N) = (A \cap B_1) \cup (A \cap B_2) \cup \dots \cup (A \cap B_N)$$



- Since B_i are mutually exclusive, then

$$P[A] = P[A \cap B_1] + P[A \cap B_2] + \dots + P[A \cap B_N]$$

- Recall $P(A|B)P(B) = P(A \cap B)$

- And, therefore $P[A] = P[A | B_1]P[B_1] + \dots + P[A | B_N]P[B_N]$

$$= \sum_{k=1}^N P[A | B_k]P[B_k]$$

Bayes Theorem

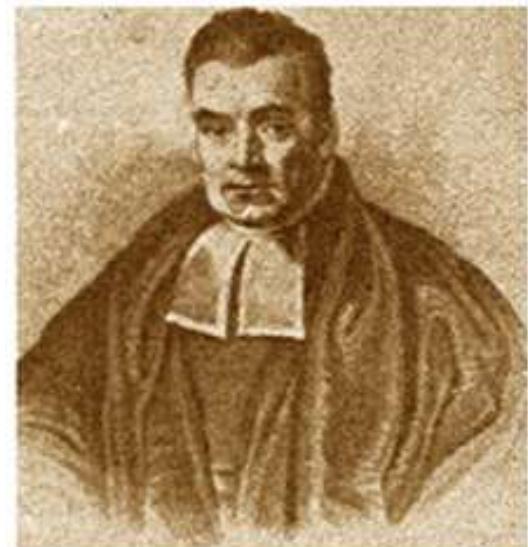
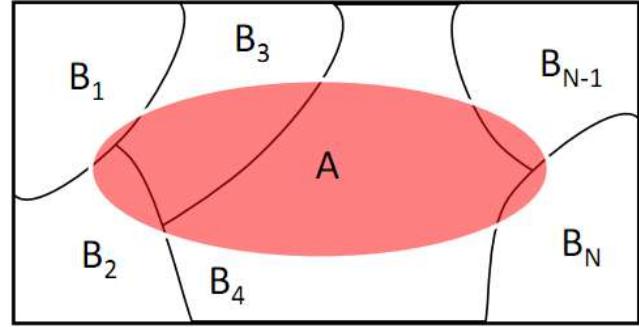
□ Problem formulation

- Assume $\{B_1, B_2, \dots, B_N\}$ is a partition of S.
- Given that event A occurs.
- What is the probability of event B_j ?
- Using the definition of conditional probability and the Theorem of total probability we obtain

$$P[B_j | A] = \frac{P[A \cap B_j]}{P[A]} = \frac{P[A | B_j]P[B_j]}{\sum_{k=1}^N P[A | B_k]P[B_k]}$$

□ This is known as Bayes Theorem or Bayes Rule, and is (one of) the most useful relations in probability and statistics

- Bayes Theorem is definitely the fundamental relation in Statistical Pattern Recognition



Rev. Thomas Bayes (1702-1761)

Bayes Theorem and Statistical Pattern Recognition

- When used for pattern classification, BT is generally expressed as

$$P[\omega_j | x] = \frac{P[x | \omega_j]P[\omega_j]}{\sum_{k=1}^N P[x | \omega_k]P[\omega_k]} = \frac{P[x | \omega_j]P[\omega_j]}{P[x]}$$

- where ω_j is the j -th class (e.g., car, motorcycle, ..) and x is the feature/observation vector (e.g., image of a car, motorcycle, ...)
- A typical decision rule is to choose class ω_j with highest $P[\omega_j | x]$
 - Intuitively, we choose the class that is more "likely" given observation x
- Each term in the Bayes Theorem has a special name

- $P[\omega_j]$: Prior probability (of class ω_j)
- $P[\omega_j | x]$: Posterior probability (of class ω_j given the observation x)
- $P[x | \omega_j]$: likelihood (probability of observation x given class ω_j)
- $P[x]$: normalization constant (does not affect the decision)

Exercise

- Consider a clinical problem where we need to decide if a patient has a particular medical condition on the basis of an imperfect test
 - Someone with the condition may go undetected (false-negative)
 - Someone free of the condition may yield a positive result (false-positive)
- Nomenclature
 - The true-negative rate $P(NEG|\sim COND)$ of a test is called its **SPECIFICITY**
 - The true-positive rate $P(POS|COND)$ of a test is called its **SENSITIVITY**
- Problem
 - Assume a population of **10,000** with a **1%** prevalence for the condition
 - Assume that we design a test with **98%** specificity and **90%** sensitivity
 - Assume you take the test, and the result comes out **POSITIVE**
 - **What is the probability that you have the condition(having the disease)?**
- Solution
 - Solution A : Fill in the joint frequency table next slide, or
 - Solution B : Apply Bayes rule

Exercise : Solution B

Applying Bayes rule

Assume a population of **10,000** with a **1%** prevalence for the condition

Assume that we design a test with **98%** specificity (True Neg) and **90%** sensitivity (Tru Pos)

Assume you take the test, and the result comes out POSITIVE

$$P[\omega_j | x] = \frac{P[x | \omega_j]P[\omega_j]}{\sum_{k=1}^N P[x | \omega_k]P[\omega_k]} = \frac{P[x | \omega_j]P[\omega_j]}{P[x]}$$

$$\begin{aligned} P[\text{COND} | \text{POS}] &= \frac{P[\text{POS} | \text{COND}]P[\text{COND}]}{P[\text{POS}]} \\ &= \frac{P[\text{POS} | \text{COND}]P[\text{COND}]}{P[\text{POS} | \text{COND}]P[\text{COND}] + P[\text{POS} | \sim \text{COND}]P[\sim \text{COND}]} \\ &= \frac{0.90 \times 0.01}{0.90 \times 0.01 + (1 - 0.98) \times 0.99} \\ &= 0.3125 \end{aligned}$$

	Test is Positive	Test is Negative	Row Total
Has Condition	True-positive P(POS COND) 100×0.90	False-negative P(NEG COND) 100×(1-0.90)	100
Free of Condition	False-positive P(POS ~COND) 9,900×(1-0.98)	True-negative P(NEG ~COND) 9,900×0.98	9,900
Column Total	288	9,712	10,000

Joint And Conditional Probability

- EXAMPLE: Binary communication system
- Send one of two possible symbol (0 or 1) over a channel.
- Denote (for $i = 1, 2$)

- B_i : "the symbol before the channel"
- A_i : "the symbol after the channel"



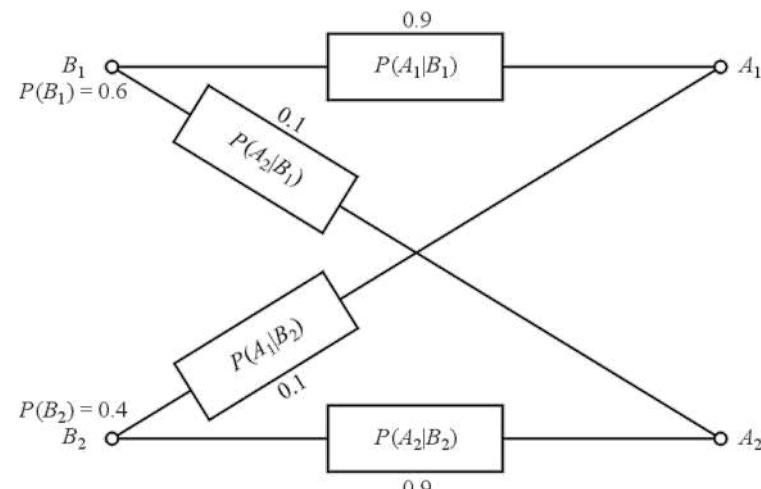
- Assume

- $P(B_1) = 0.6, \quad P(A_1 | B_1) = 0.9, \quad P(A_1 | B_2) = 0.1$
- $P(B_2) = 0.4, \quad P(A_2 | B_1) = 0.1, \quad P(A_2 | B_2) = 0.9$

- Calculate rest of probability.

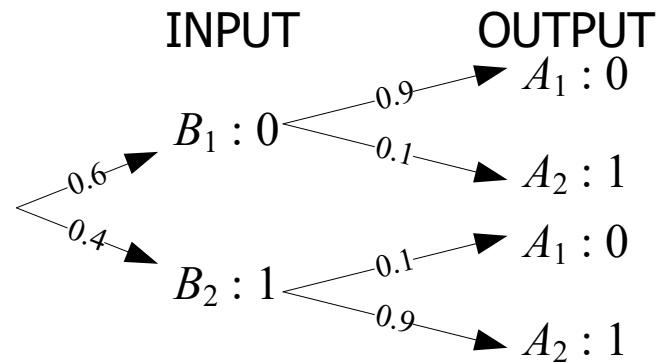
$$\begin{aligned}P(A_1) &= P(A_1 | B_1)P(B_1) + P(A_1 | B_2)P(B_2) \\&= 0.9(0.6) + 0.1(0.4) = 0.58\end{aligned}$$

$$\begin{aligned}P(A_2) &= P(A_2 | B_1)P(B_1) + P(A_2 | B_2)P(B_2) \\&= 0.1(0.6) + 0.9(0.4) = 0.42\end{aligned}$$



Joint And Conditional Probability

- ❑ EXAMPLE
- ❑ Conditional Probability
 - Using the Bayes' theorem,



$$P(B_1 | A_1) = \frac{P(A_1 | B_1)P(B_1)}{P(A_1)} = \frac{0.9(0.6)}{0.58} = \frac{0.54}{0.58} \approx 0.931$$

$$P(B_2 | A_2) = \frac{P(A_2 | B_2)P(B_2)}{P(A_2)} = \frac{0.9(0.4)}{0.42} = \frac{0.36}{0.42} \approx 0.857$$

$$P(B_1 | A_2) = \frac{P(A_2 | B_1)P(B_1)}{P(A_2)} = \frac{0.1(0.6)}{0.42} = \frac{0.06}{0.42} \approx 0.143$$

$$P(B_2 | A_1) = \frac{P(A_1 | B_2)P(B_2)}{P(A_1)} = \frac{0.1(0.4)}{0.58} = \frac{0.04}{0.58} \approx 0.069$$

Independent Events

- Let two events A and B with **nonzero probability**.
- (*Statistically*) **Independent**
- **Definition**
 - The probability of occurrence of one event is not affected by the occurrence of the other event.
- **Mathematical representation**
$$P(A | B) = P(A)$$
$$\Leftrightarrow P(B | A) = P(B)$$
$$\Leftrightarrow P(A \cap B) = P(A)P(B)$$

$$P(A | B) \equiv \frac{P(A \cap B)}{P(B)}$$
- If two events are **mutually exclusive**, $P(A \cap B) = 0 \neq P(A)P(B)$
 - So, two events cannot be both *mutually exclusive* and *statistically independent*.
 - i.e. For two events to be **independent**, they must **have an intersection**

Independent Events

- EXAMPLE: One card is selected from an ordinary 52-card deck.
- Define events
 - A : "select a king", B : "select a jack or queen", C : "select a heart"
- Probabilities
 - $P(A) = 4/52$, $P(B) = 8/52$, $P(C) = 13/52$

- Check the independence by pairs.
 - $P(A \cap B) = 0 \neq P(A)P(B)$ Mutually Exclusive

$$P(A \cap C) = \frac{1}{52} = P(A)P(C) = \frac{1}{52}$$

$$P(B \cap C) = \frac{2}{52} = P(B)P(C) = \frac{2}{52}$$

- So, A and C are independent as a pair, as are B and C .

Independent Events

- EXAMPLE: Draw four cards from an ordinary 52-card deck.
- Define events
 - A_i : "draw an ace consecutively i times" (for $i=1,2,3,4$)
- Check the independent events in the either replaced trials or not.
- Assuming that each trial is replaced.
 - They are independent.
$$\begin{aligned} P(A_1 \cap A_2 \cap A_3 \cap A_4) &= P(A_1)P(A_2)P(A_3)P(A_4) \\ &= (4/52)^4 \approx 3.50(10^{-5}) \end{aligned}$$
- Assuming that each trial is not replaced.
 - They are not independent events.

$$\begin{aligned} P(A_1 \cap A_2 \cap A_3 \cap A_4) &\neq P(A_1) P(A_2) P(A_3) P(A_4) \\ &= P(A_1)P(A_2 \cap A_3 \cap A_4 | A_1) \\ &= P(A_1)P(A_2 | A_1)P(A_3 \cap A_4 | A_1 \cap A_2) \\ &= P(A_1)P(A_2 | A_1)P(A_3 | A_1 \cap A_2)P(A_4 | A_1 \cap A_2 \cap A_3) \\ &= \frac{4}{52} \cdot \frac{3}{51} \cdot \frac{2}{50} \cdot \frac{1}{49} \approx 3.69(10^{-6}) \end{aligned}$$

Independent Events

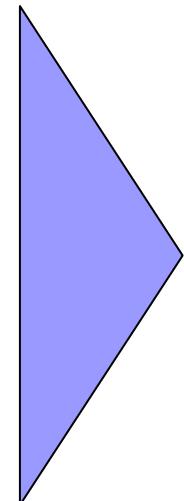
- EXAMPLE: A pair of fair dice is rolled, the values of their faces are added, and the following events are defined.

- **Define events**

- A: "roll an odd sum"
 - B: "roll a sum that is an integer multiple of 3"

- a) **Are events A and B statistically independent?**

						Sum of faces	Probability
1,1						2	1/36
1,2	2,1					3	2/36
1,3	2,2	3,1				4	3/36
1,4	2,3	3,2	4,1			5	4/36
1,5	2,4	3,3	4,2	5,1		6	5/36
1,6	2,5	3,4	4,3	5,2	6,1	7	6/36
	2,6	3,5	4,4	5,3	6,2	8	5/36
		3,6	4,5	5,4	6,3	9	4/36
			4,6	5,5	6,4	10	3/36
				5,6	6,5	11	2/36
					6,6	12	1/36



- b) **Note all the events are mutually exclusive. Total probability axiom applies.**

Independent Events

- **Example (cont.)**
- $P(A) = 2/36 + 4/36 + 6/36 + 4/36 + 2/36 = 18/36 = 1/2$
- $P(B) = 2/36 + 5/36 + 4/36 + 1/36 = 12/36 = 1/3$

- $P(A \cap B) = ?$
- Find events of $A \cap B \rightarrow$ Sum = odd & multiples of 3

- $P(A \cap B) = 2/36 + 4/36 = 6/36 = 1/6$
- Is $P(A \cap B) = P(A)P(B)$?
 - $1/6 = 1/2 \cdot 1/3$
- Thus A & B are independent!

- What is probability of event $A \cup B$?
- $P(A \cup B) = P(A) + P(B) - P(A \cap B) = P(A) + P(B) - P(A)P(B)$
 $= 1/2 + 1/3 - 1/6 = 2/3$

Random Variables

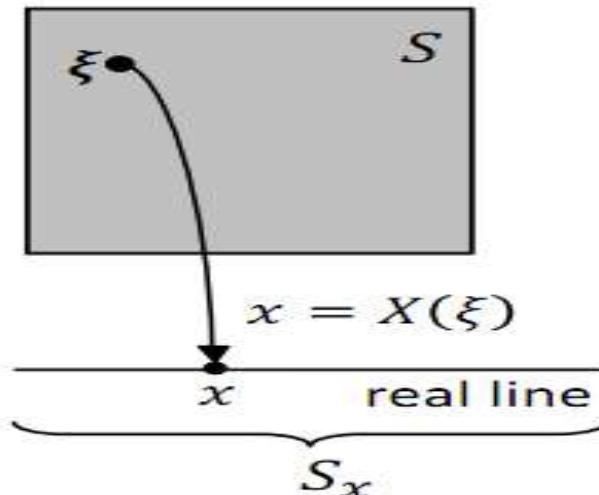
- When we perform a random experiment we are usually interested in some **measurement** or **numerical attribute** of the outcome
 - e.g., weights in a population of subjects, execution times when benchmarking CPUs, shape parameters when performing ATR
- These examples lead to the concept of random variable
- A random variable **X** is a function that assigns a **real number** $X(\xi)$ to each outcome ξ in the sample space of a random experiment
- $X(\xi)$ maps from all possible outcomes in sample space onto the real line



Random Variables

- The function that assigns values to each outcome is fixed and deterministic, i.e., as in the rule “count the number of heads in three coin tosses”
 - Randomness in it is due to the underlying randomness of the outcome ξ of the experiment

- Random variables can be
 - Discrete, e.g., the resulting number after rolling a dice
 - Continuous, e.g., the weight of a sampled individual



Random Variables

EXAMPLE 2.1-1

- An experiment consist of rolling a die and flipping a coin.
- (1) a coin head (H) outcome corresponds to positive values of X that are equal to numbers that show up on the die.
- (2) a coin tail (T) outcome correspond to negative values of X that are equal in magnitude to twice the number that shows on the die.

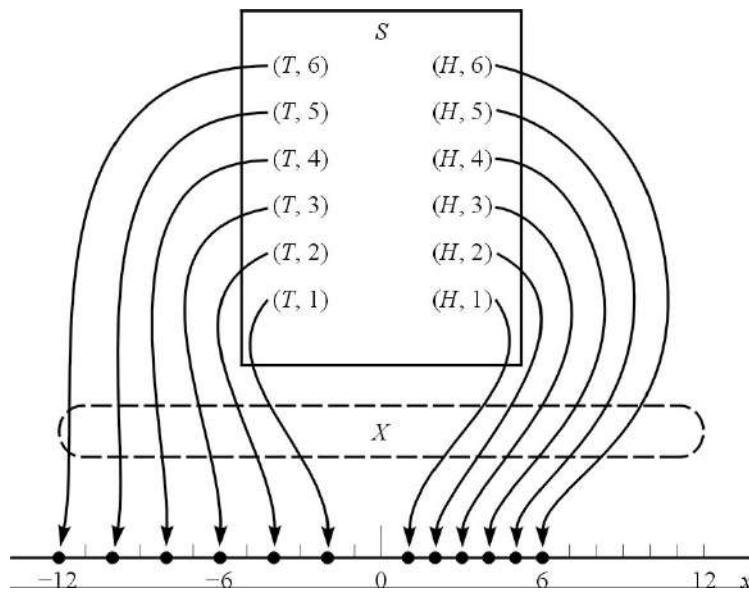


FIGURE 2.1-1
A Random variable
mapping of a sample
space.

Random Variables

□ EXAMPE 2.1-2

- An experiment where the pointer on a wheel of chance is spun.
- The possible outcomes are the numbers from 0 to 12 marked on the wheel. (The sample space : $\{0 < s \leq 12\}$.
- a random variable : $X = X(s) = s^2$
- Points in S now map onto the real line as the set $\{0 < s \leq 144\}$.

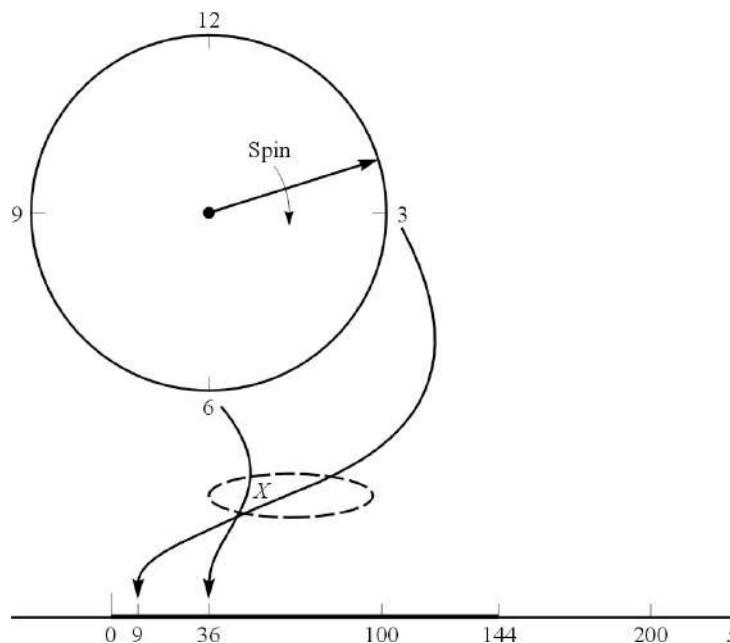


FIGURE 2.1-2
Mapping applicable to Example 2.1-2.

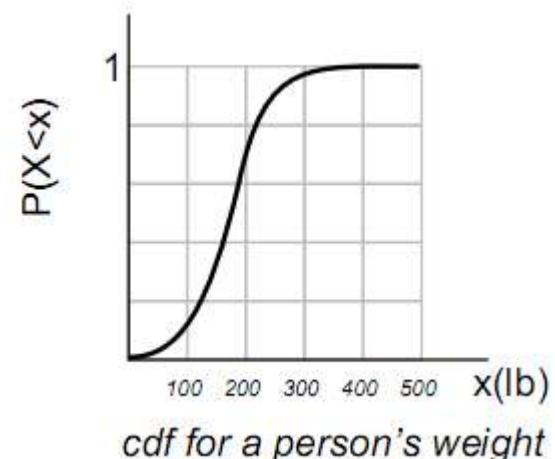
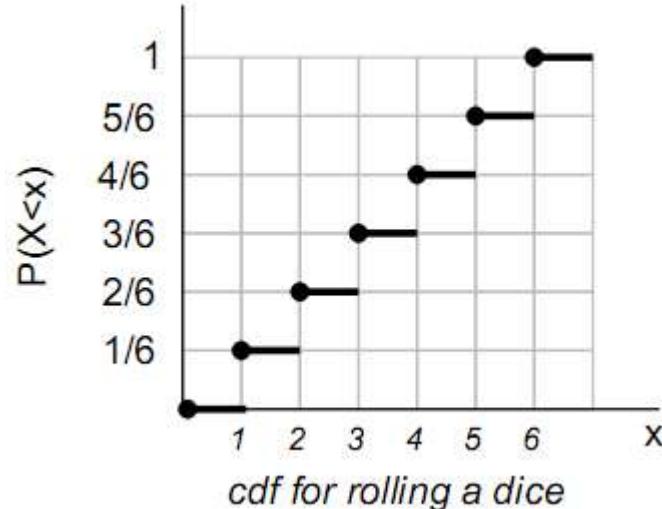
Cumulative Distribution Function (cdf)

- The cumulative distribution function $F_X(x)$ of a random variable X is defined as the probability of the event $\{X \leq x\}$

$$F_X(x) = P[X \leq x] \quad \text{for } -\infty < x < +\infty$$

- Properties of the cdf

- $0 \leq F_X(x) \leq 1$
- $\lim_{x \rightarrow \infty} F_X(x) = 1$
- $\lim_{x \rightarrow -\infty} F_X(x) = 0$
- $F_X(a) \leq F_X(b)$ if $a \leq b$
-



Probability Density Function (pdf)

- The probability density function $f_X(x)$ of a continuous random variable X , if it exists, is defined as the derivative of $F_X(x)$

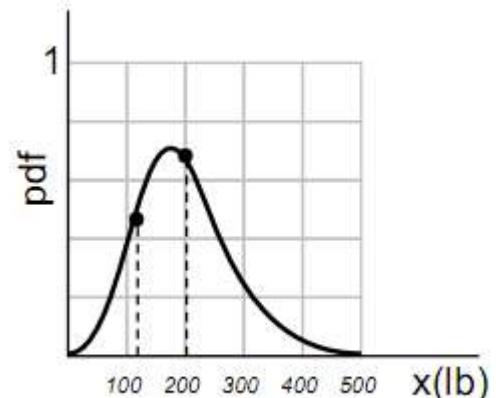
$$f_X(x) = \frac{dF_X(x)}{dx}$$

- For discrete random variables, the equivalent to the pdf is the probability mass function

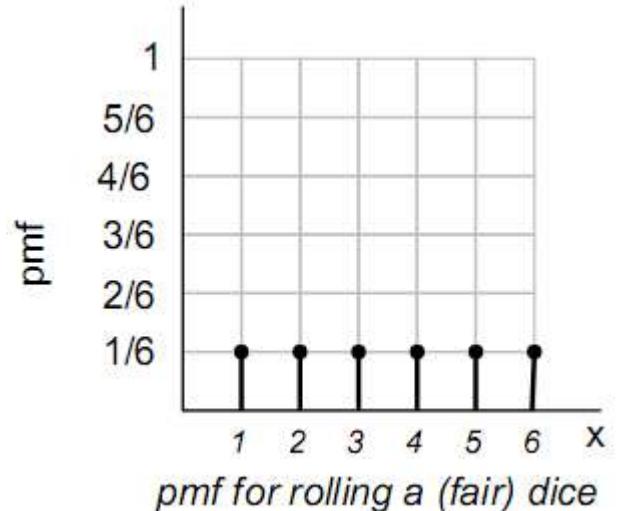
$$f_X(x) = \frac{\Delta F_X(x)}{\Delta x}$$

- Properties of the pdf

- $f_X(x) \geq 0$
- $P[a < x < b] = \int_a^b f_X(x) dx$
- $F_X(x) = \int_{-\infty}^x f_X(x) dx$
- $\int_{-\infty}^{\infty} f_X(x) dx = 1$



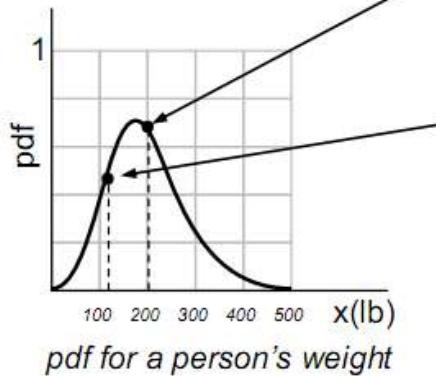
pdf for a person's weight



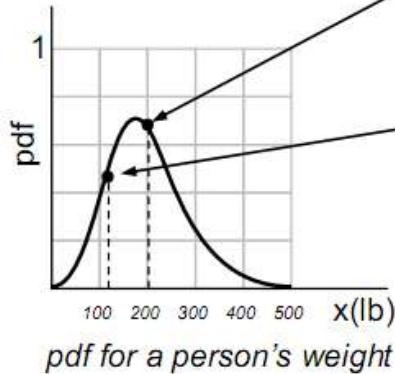
pmf for rolling a (fair) dice

Probability Density Function Vs. Probability

- What is the probability of somebody weighting 200 lb?
 - According to the pdf, this is about 0.62
 - This number seems reasonable, right?



Probability Density Function Vs. Probability



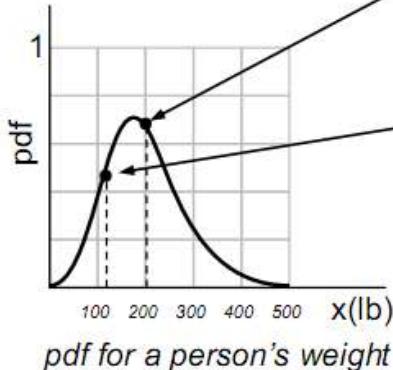
- What is the probability of somebody weighting 200 lb?

- According to the pdf, this is about 0.62
- This number seems reasonable, right?

- Now, what is the probability of somebody weighting 124.876 lb?

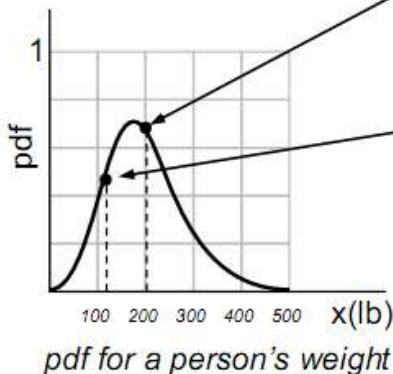
- According to the pdf, this is about 0.43
- But, intuitively, we know that the probability should be zero (or very, very small)

Probability Density Function Vs. Probability



- What is the probability of somebody weighting 200 lb?
 - According to the pdf, this is about 0.62
 - This number seems reasonable, right?
- Now, what is the probability of somebody weighting 124.876 lb?
 - According to the pdf, this is about 0.43
 - But, intuitively, we know that the probability should be zero (or very, very small)
- How do we explain this paradox?
 - The pdf DOES NOT define a probability, but a probability DENSITY!
 - To obtain the actual probability we must integrate the pdf in an interval
 - So we should have asked the question: what is the probability of somebody weighting 124.876 lb plus or minus 2 lb?

Probability Density Function Vs. Probability



- What is the probability of somebody weighting 200 lb?

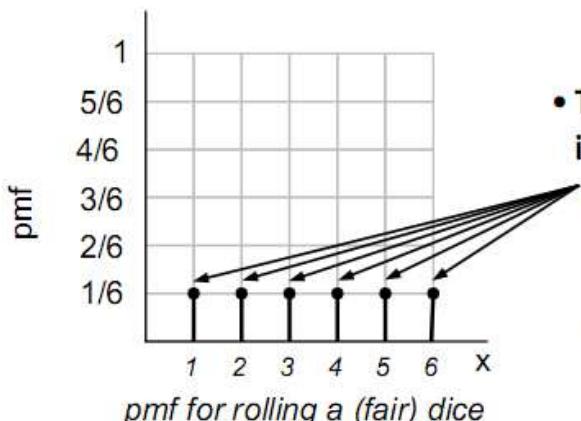
- According to the pdf, this is about 0.62
- This number seems reasonable, right?

- Now, what is the probability of somebody weighting 124.876 lb?

- According to the pdf, this is about 0.43
- But, intuitively, we know that the probability should be zero (or very, very small)

- How do we explain this paradox?

- The pdf DOES NOT define a probability, but a probability DENSITY!
- To obtain the actual probability we must integrate the pdf in an interval
- So we should have asked the question: what is the probability of somebody weighting 124.876 lb plus or minus 2 lb?



- The probability mass function is a ‘true’ probability (reason why we call it a ‘mass’ as opposed to a ‘density’)

- The pmf is indicating that the probability of any number when rolling a fair dice is the same for all numbers, and equal to 1/6, a very legitimate answer

- The pmf DOES NOT need to be integrated to obtain the probability (it cannot be integrated in the first place)

Statistical Characterization of Random Variables

- The cdf or the pdf are **SUFFICIENT** to fully characterize a RV
- However, a RV can be **PARTIALLY** characterized with other measures
- Expectation (center of mass of a density)

$$E[X] = \mu = \int_{-\infty}^{\infty} xf_X(x)dx$$

- Variance (spread about the mean)

$$VAR[X] = \sigma^2 = E[(X - E[X])^2] = \int_{-\infty}^{\infty} (x - \mu)^2 f_X(x)dx$$

- Standard deviation

$$STD[X] = \sigma = VAR[X]^{1/2}$$

Random Vectors

- An extension of the concept of a random **variable**
- A random **vector** \mathbf{X} is a function that assigns a **vector** of real numbers to each outcome ξ in sample space S
- Denote a random vector by a **column vector**
- The notions of cdf and pdf are replaced by 'joint cdf' and 'joint pdf'
- Given random vector $\mathbf{X} = [x_1, x_2, \dots, x_N]^T$ we define the joint cdf as

$$F_{\mathbf{X}}(\mathbf{x}) = P_{\mathbf{X}}[\{X_1 \leq x_1\} \cap \{X_2 \leq x_2\} \cap \dots \cap \{X_N \leq x_N\}]$$

- And the joint pdf as

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{\partial^N F_{\mathbf{X}}(\mathbf{x})}{\partial x_1 \partial x_2 \cdots \partial x_N}$$

Statistical Characterization of Random Vectors

- A random vector is also fully characterized by its joint cdf or joint pdf
- Alternatively, we can (partially) describe a random vector with measures similar to those defined for scalar random variables
- Mean vector

$$E[\mathbf{X}] = \boldsymbol{\mu} = [E[X_1], E[X_2], \dots, E[X_N]]^T = [\mu_1, \mu_2, \dots, \mu_N]^T$$

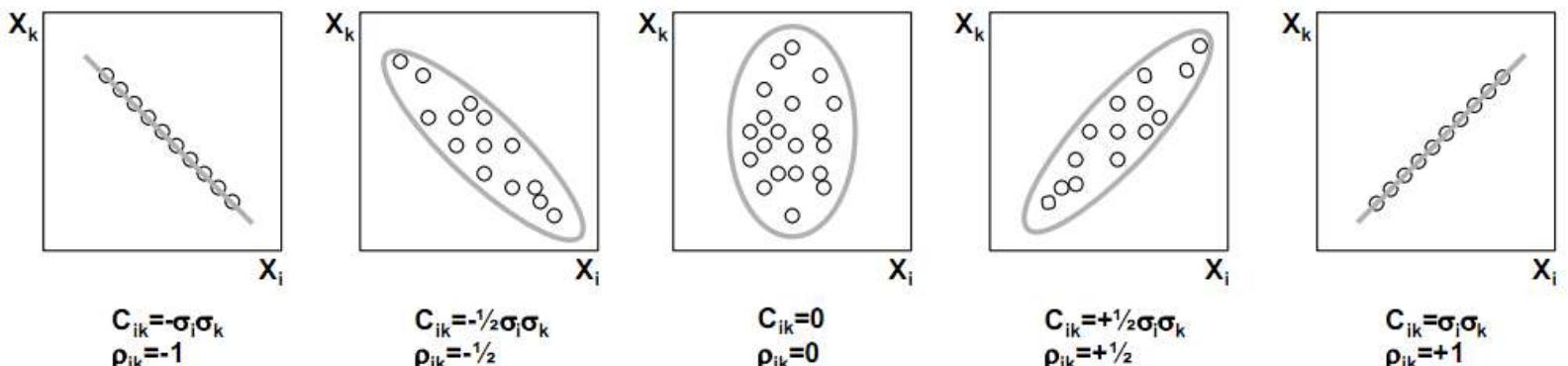
- Covariance matrix

$$COV[\mathbf{X}] = \boldsymbol{\Sigma} = E[(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^T]$$

$$= \begin{bmatrix} E[(x_1 - \mu_1)^2] & \cdots & E[(x_1 - \mu_1)(x_N - \mu_N)] \\ \vdots & \ddots & \vdots \\ E[(x_N - \mu_N)(x_1 - \mu_1)] & \cdots & E[(x_N - \mu_N)^2] \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & \cdots & c_{1N} \\ \vdots & \ddots & \vdots \\ c_{N1} & \cdots & \sigma_N^2 \end{bmatrix}$$

Covariance Matrix

- The covariance matrix indicates the tendency of each pair of features (dimensions in a random vector) to vary together, i.e., to co-vary*
- The covariance has several important properties
 - If x_i and x_k tend to increase together, then $c_{ik} > 0$
 - If x_i tends to decrease when x_k increases, then $c_{ik} < 0$
 - If x_i and x_k are uncorrelated, then $c_{ik} = 0$
 - $c_{ii} = \sigma_i^2 = \text{VAR}[x_i]$
- The covariance terms can be expressed as $c_{ii} = \sigma_i^2$ and $c_{ik} = \rho_{ik}\sigma_i\sigma_k$
 - where ρ_{ik} is called the correlation coefficient



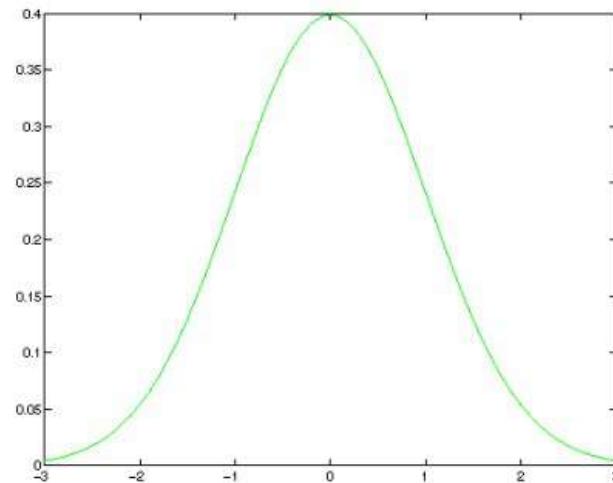
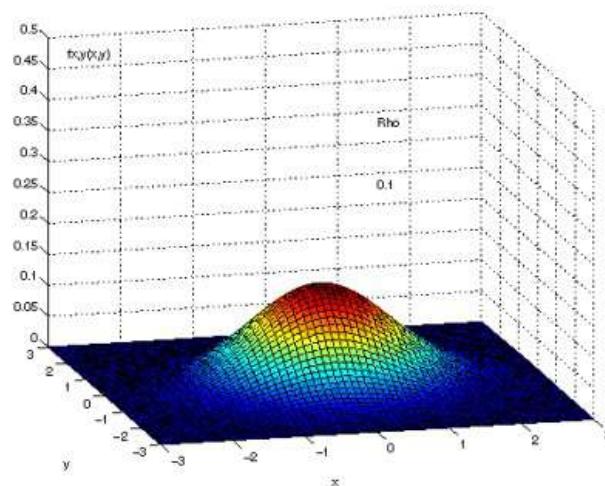
The Normal or Gaussian Distribution

- The multivariate Normal distribution $N(\mu, \Sigma)$ is defined as

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{X} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{X} - \boldsymbol{\mu})\right]$$

- For a single dimension, this expression is reduced to

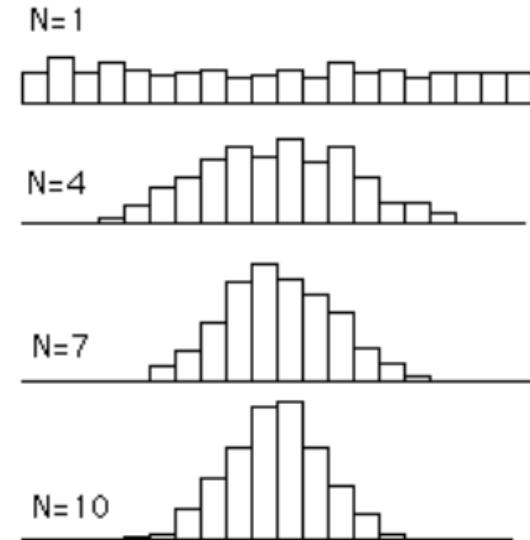
$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$



Central Limit Theorem

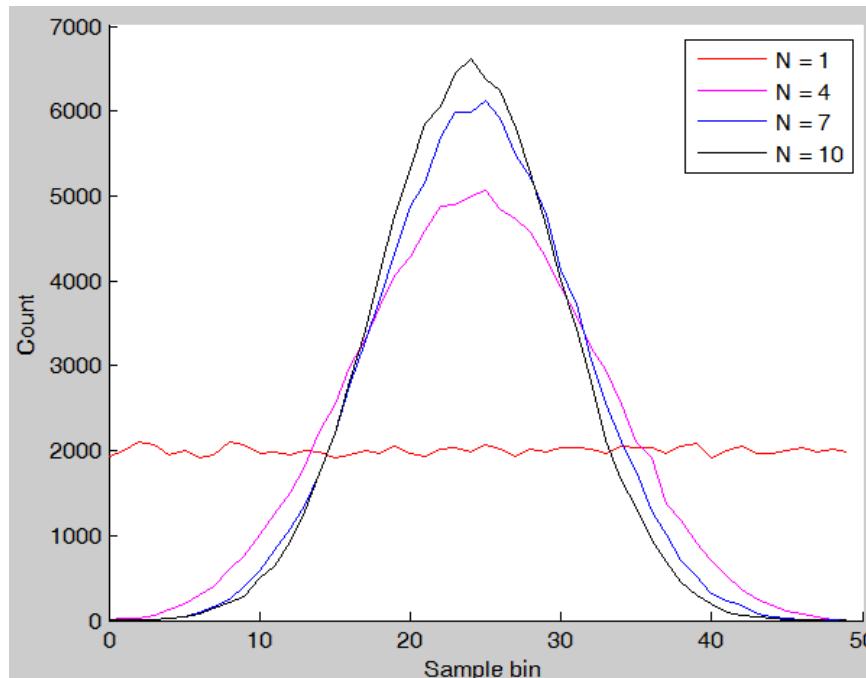
- The probability distribution function of the sum of a large number of random variables approaches a gaussian distribution.
- Example: 500 experiments are performed using a uniform distribution

- $N = 1$
 - One sample is drawn from the distribution and its mean is recorded (500 times)
 - The histogram resembles a uniform distribution, as one would expect
- $N = 4$
 - Four samples are drawn and the mean of the four samples is recorded (500 times)
 - The histogram starts to look more Gaussian
- As N grows, the shape of the histograms resembles a Normal distribution more closely



Implementation: Central Limit Theorem

- ❑ 100000 experiments are performed using a uniform distribution
- ❑ The number of samples is increased to 10 from 1 ($N = 1, 4, 7, 10$)
- ❑ As N grows, the shape of the histograms resembles a Normal distribution more closely



Independent Events

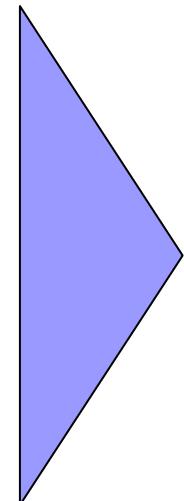
- EXAMPLE: A pair of fair dice is rolled, the values of their faces are added, and the following events are defined.

- **Define events**

- A: "roll an odd sum"
 - B: "roll a sum that is an integer multiple of 3"

- a) **Are events A and B statistically independent?**

						Sum of faces	Probability
1,1						2	1/36
1,2	2,1					3	2/36
1,3	2,2	3,1				4	3/36
1,4	2,3	3,2	4,1			5	4/36
1,5	2,4	3,3	4,2	5,1		6	5/36
1,6	2,5	3,4	4,3	5,2	6,1	7	6/36
	2,6	3,5	4,4	5,3	6,2	8	5/36
		3,6	4,5	5,4	6,3	9	4/36
			4,6	5,5	6,4	10	3/36
				5,6	6,5	11	2/36
					6,6	12	1/36



- b) **Note all the events are mutually exclusive. Total probability axiom applies.**

Bayesian Classifier

ECE 610

David K. Han

Bayesian Decision Theory

- A fundamental statistical approach to ML classification.
- Allows us to combine observed data and prior knowledge
- Provides practical learning algorithms
- It is a generative (model based) approach, which offers a useful conceptual framework
 - Any kind of objects (e.g. time series, trees, etc.) can be classified, based on a **probabilistic model** specification
- Based on **tradeoffs** between
 - decisions using **probability** and the costs
- Assumes relevant probability values are **known**.

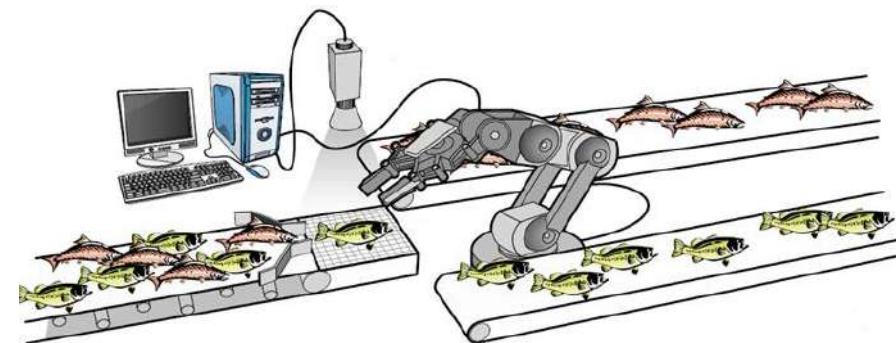
Bayesian Decision Theory

- Example:
- Designing a classifier to **separate two kinds of fish**: sea bass and salmon
- Let ω denote the state of nature
 - $\omega = \omega_1$ for sea bass
 - $\omega = \omega_2$ for salmon
 - Thus, ω needs to be described probabilistically
- **A priori probability**: Representative of **existing knowledge** on the problem
 - $P(\omega_1)$ is the probability that next fish is sea bass
 - $P(\omega_2)$ is the probability that it is salmon



Bayesian Decision Theory

- If forced to make a decision **without observing** the fish.
- Use the following **decision rule**
 - Decide ω_1 if $P(\omega_1) > P(\omega_2)$; otherwise decide ω_2 .
 - End up deciding the fish to be **one type always** unless the probabilities are the same.
- This strategy may be **acceptable** so long as
 - $P(\omega_1) \gg P(\omega_2)$ or $P(\omega_1) \ll P(\omega_2)$
- Probability of error by this strategy:
 - $P(E) = \min(P(\omega_1), P(\omega_2))$.
- An alternative: Utilize **observation** → data
 - Consider x (**lightness of the fish**) to be a continuous random variable whose distribution depends on the state of nature (salmon or sea bass)
 - Expressed as $p(x|\omega_1)$ → class-conditional probability density function for state of nature being ω_1



Bayesian Decision Theory

- Difference between $p(x|\omega_1)$ and $p(x|\omega_2)$: difference in lightness between sea bass and salmon
- Suppose both the prior probabilities $P(\omega_j)$ and the conditional densities $p(x|\omega_j)$ known. The joint probability density can be written as

$$p(w_j, x) = P(w_j | x)p(x) = p(x | w_j)P(w_j)$$

- Rearranging terms lead to Bayes Theorem

$$P(w_j | x) = \frac{p(x | w_j)P(w_j)}{p(x)}$$

- Where $p(x) = \sum_{j=1}^2 p(x | w_j)P(w_j)$
- In plain English: $posterior = \frac{likelihood \times prior}{evidence}$

Bayesian Decision Theory

- By measuring x , we can **convert** the **prior** probability, $P(\omega_j)$, into a **posterior probability**, $P(\omega_j|x)$.
- More specifically, for ω_1

$$P(w_1|x) = \frac{p(x|w_1)P(w_1)}{p(x)} = \frac{p(x|w_1)P(w_1)}{\sum_{j=1}^2 p(x|w_j)P(w_j)} = \frac{p(x|w_1)P(w_1)}{p(x|w_1)P(w_1) + p(x|w_2)P(w_2)}$$

- For ω_2
- Evidence $p(x)$, viewed as a **scale factor**, is **identical** for both ω_1 and ω_2 , it is often ignored.

Bayesian Decision Theory: Caution

- By measuring x , we can convert the prior probability, $P(\omega_j)$, into a posterior probability, $P(\omega_j|x)$.
- Note however that $P(\omega_j|x)$ is an estimate of the state being ω_j given the observation x .
- Recall

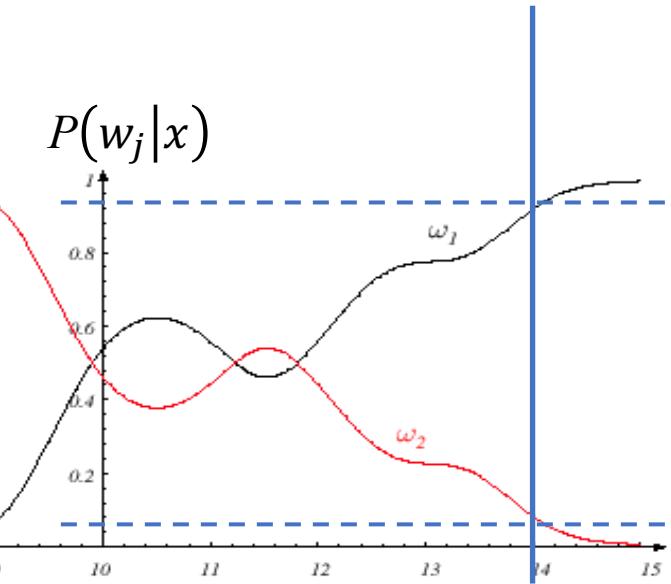
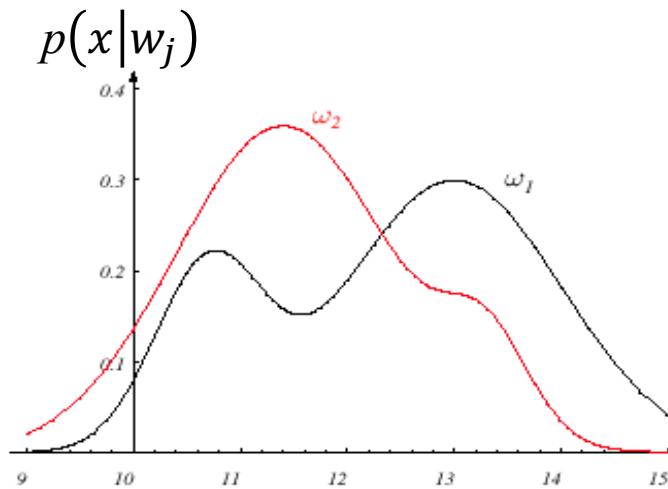
“I have a dream that my four little children will one day live in a nation where they will not be judged by the color of their skin but by the content of their character.”

- The observation used to estimate the state is not the actual content of the state



Bayesian Decision Theory

- Let's plot likelihood and posterior for a given prior probability: $P(\omega_1) = 2/3$ and $P(\omega_2) = 1/3$

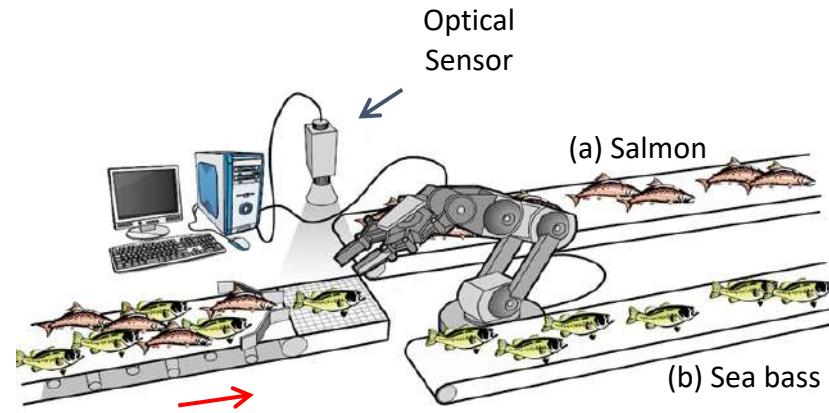


$$P(w_j|x) = \frac{p(x|w_j)P(w_j)}{p(x)}$$

- For every value of x , the posteriors sum to 1.0. $\rightarrow P(\omega_1|x) + P(\omega_2|x) = 1$
- At $x=14$, the probability it is in category ω_2 is 0.08, and for category ω_1 is 0.92.

Implementing a Bayesian classifier

- Sorting Fish: incoming fish are sorted according to species using optical sensing (sea bass ω_1 or salmon ω_2 ?)
- Collect data on sea bass and salmon according to their distinctive features
- Feature : x (length or lightness)

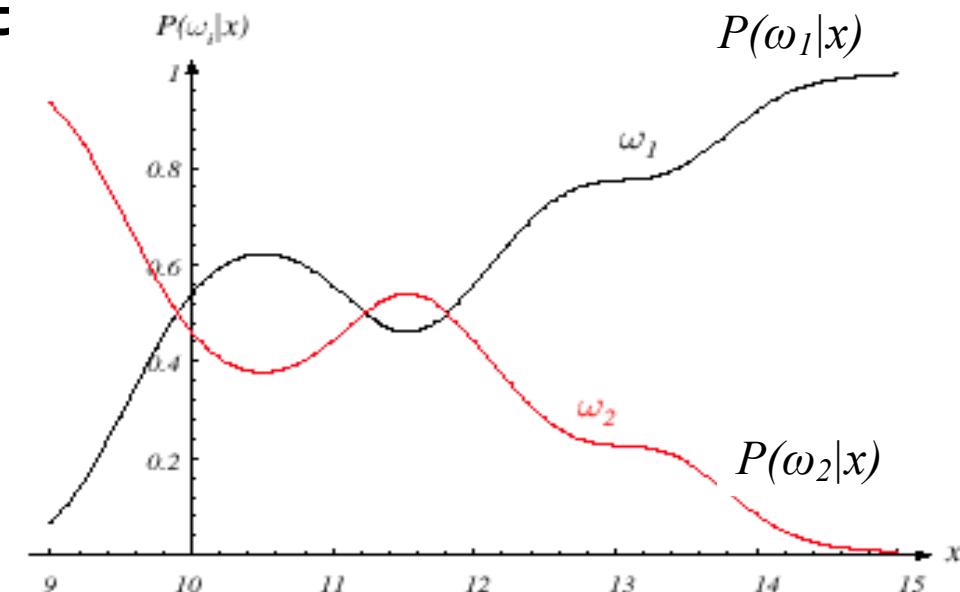


- Step
- Estimate likelihood $p(x|\omega_1)$ and $p(x|\omega_2)$ from observation data.
- Feature extraction using optical sensing.(length and lightness of fish)
- Compare
 - If $p(x|\omega_1)P(\omega_1) > p(x|\omega_2) P(\omega_2)$, fish = Sea bass
 - If $p(x|\omega_1)P(\omega_1) < p(x|\omega_2) P(\omega_2)$, fish = Salmon

Maximum A Posteriori (MAP) Inference

- Bayes' Decision Rule

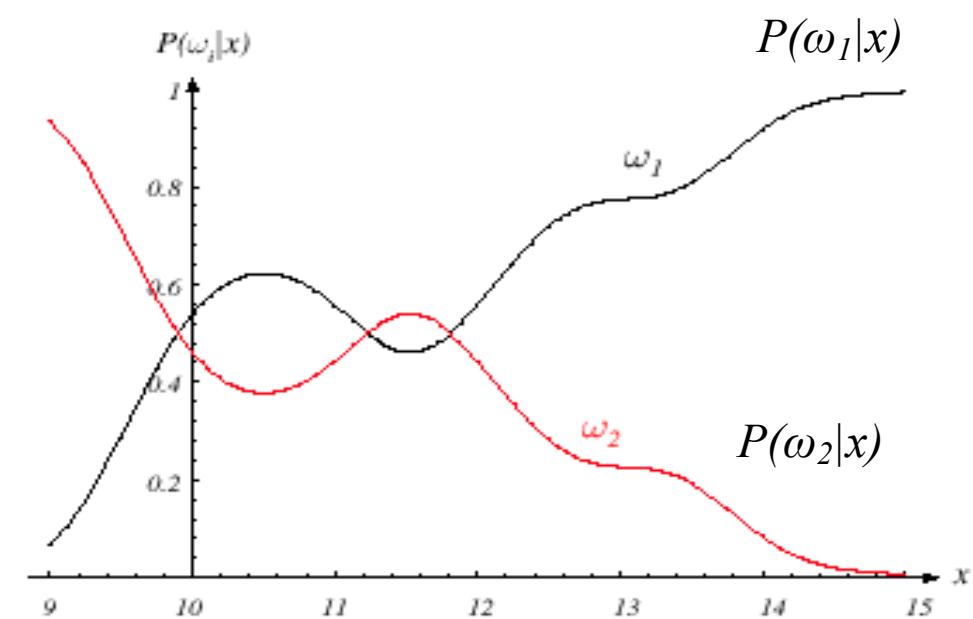
- If $P(\omega_1|x) > P(\omega_2|x)$, choose ω_1
- If $P(\omega_2|x) > P(\omega_1|x)$, choose ω_2



- This is called **Maximum A Posteriori (MAP)** inference
- More compactly:
$$\arg \max_w (P(\omega|x))$$
- To justify this, let's compute probability of error for such decisions
 - $P(error|x) = P(\omega_1|x)$ if we decide ω_2
 - $P(error|x) = P(\omega_2|x)$ if we decide ω_1

Maximum A Posteriori (MAP) Inference

- Again
 - $P(\text{error}|x) = P(\omega_1|x)$ if we decide ω_2
 - $P(\text{error}|x) = P(\omega_2|x)$ if we decide ω_1
- Will this rule minimize the average probability of error?
 - Average probability of error
$$P(\text{error}) = \int_{-\infty}^{\infty} P(\text{error}, x) dx = \int_{-\infty}^{\infty} P(\text{error}|x) P(x) dx$$
- By choosing ω as
 - When $P(\omega_1|x) > P(\omega_2|x)$, choose ω_1
 - When $P(\omega_2|x) > P(\omega_1|x)$, choose ω_2
- *error* is minimized everywhere in x . Thus, the average error is minimized by Bayes' decision rule.



Bayesian Decision in Multidimensions

- Let's consider more general cases
- Cases with more than one feature
 - Replace the scalar x with the feature vector \mathbf{x} where \mathbf{x} is in a d-dimensional Euclidean space \mathbf{R}^d , called feature space.
- A feature vector \mathbf{x} is a d-component vector-valued random variable
- Let $p(\mathbf{x}|\omega_j)$ be the probability density function for \mathbf{x} conditioned on ω_j being the true state of nature.
- $P(\omega_j)$ describes the prior probability that nature is in state ω_j .
- The posterior probability $P(\omega_j | \mathbf{x})$ can be computed from $p(\mathbf{x}|\omega_j)$ by Bayes' rule:

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

- where

$$p(\mathbf{x}) = \sum_{j=1}^c p(\mathbf{x}|\omega_j)P(\omega_j)$$

Multivariate Gaussian Distribution for Classification

- Let's model the likelihood functions using multivariate Gaussian distribution

$$p(\vec{x}|\omega_j) = \frac{1}{(2\pi)^{d/2} |\tilde{\Sigma}|^{1/2}} \exp \left[-\frac{1}{2} (\vec{x} - \vec{\mu})^T [\tilde{\Sigma}]^{-1} (\vec{x} - \vec{\mu}) \right]_{\omega_j}$$

$$\tilde{\vec{x}} = [\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N]$$

- The parameters can be estimated as
- d = dimensions of \vec{x}
- N = number of data points
- For a single dimension, this expression is reduced to

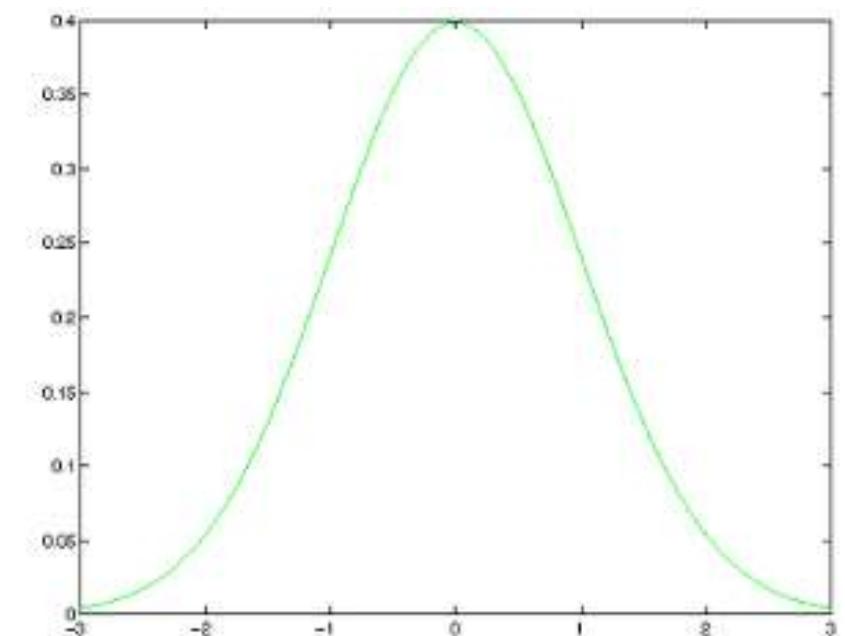
$$\vec{\mu} = \frac{1}{N} \sum_{n=1}^N \vec{x}_n$$

$$\tilde{\Sigma} = \frac{1}{N-1} \sum_{n=1}^N (\vec{x}_n - \vec{\mu})(\vec{x}_n - \vec{\mu})^T$$

$$p(x|\omega_j) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left[-\frac{1}{2} \left(\frac{x - \mu}{\sigma} \right)^2 \right]$$

$$\mu = \frac{1}{N} \sum_{n=1}^N x_n$$

$$\sigma^2 = \frac{1}{N-1} \sum_{n=1}^N (x_n - \mu)(x_n - \mu)^T$$



Multivariate Gaussian Distribution for Classification

- Mean vector

$$E[\vec{x}] = \hat{\mu} = [E[x_1], E[x_2], \dots, E[x_d]]^T = [\mu_1, \mu_2, \dots, \mu_d]^T$$

- Covariance matrix

$$\begin{aligned} COV[\vec{x}] &= \tilde{\Sigma} = E[(\vec{x} - \hat{\mu})(\vec{x} - \hat{\mu})^T] \\ &= \begin{bmatrix} E[(x_1 - \mu_1)^2] & \cdots & E[(x_1 - \mu_1)(x_d - \mu_d)] \\ \vdots & \ddots & \vdots \\ E[(x_d - \mu_d)(x_1 - \mu_1)] & \cdots & E[(x_d - \mu_d)^2] \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & \cdots & c_{1d} \\ \vdots & \ddots & \vdots \\ c_{d1} & \cdots & \sigma_d^2 \end{bmatrix} \end{aligned}$$

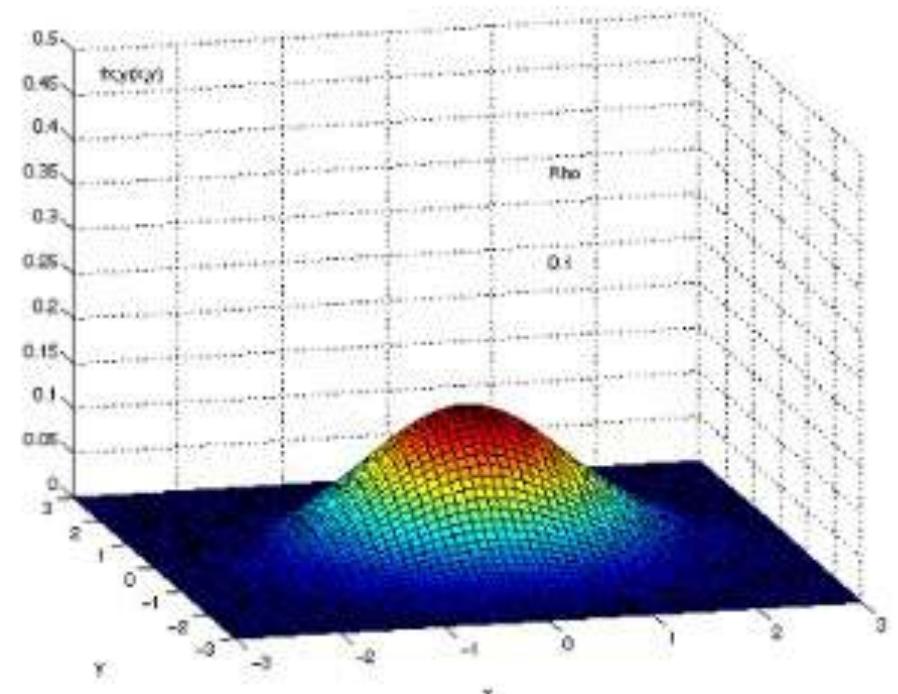
Multivariate Gaussian Distribution for Classification

- For 2-D Gaussian density function

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

$$p(x_1, x_2) = \frac{1}{2\pi\sigma_{x_1}\sigma_{x_2}\sqrt{1-\rho^2}} \exp\left\{ \frac{-1}{2(1-\rho^2)} \left[\frac{(x_1 - \bar{X}_1)^2}{\sigma_{x_1}^2} - \frac{2\rho(x_1 - \bar{X}_1)(x_2 - \bar{X}_2)}{\sigma_{x_1}\sigma_{x_2}} + \frac{(x_2 - \bar{X}_2)^2}{\sigma_{x_2}^2} \right] \right\}$$

- where $\bar{X}_1 = E[X_1]$
- $\bar{X}_2 = E[X_2]$
- $\sigma_{x_1}^2 = E[(X_1 - \bar{X}_1)^2]$
- $\sigma_{x_2}^2 = E[(X_2 - \bar{X}_2)^2]$
- $\rho = \frac{E[(X_1 - \bar{X}_1)(X_2 - \bar{X}_2)]}{\sigma_{x_1}\sigma_{x_2}}$

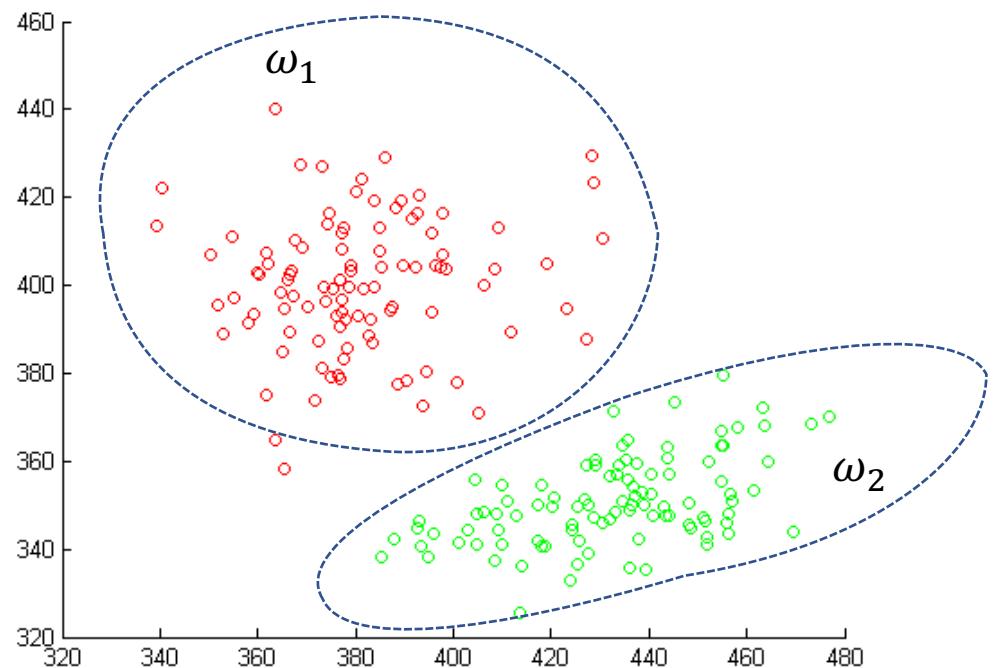


Maximum Likelihood (ML) Inference

- Ignoring $p(x)$ as stated previously, Bayes' decision rule can also be rewritten as:
- Decide ω_1 if $p(x|\omega_1)P(\omega_1) > p(x|\omega_2)P(\omega_2)$; otherwise decide ω_2 .
- If $P(\omega_1) = P(\omega_2)$ then,
 - Choose ω_1 if $p(x|\omega_1) > p(x|\omega_2)$; otherwise decide ω_2 .
 - This is called **Maximum Likelihood (ML) inference**
 - More compactly
$$\arg \max_w (p(x|\omega))$$

Implementation: ML Bayesian classifier

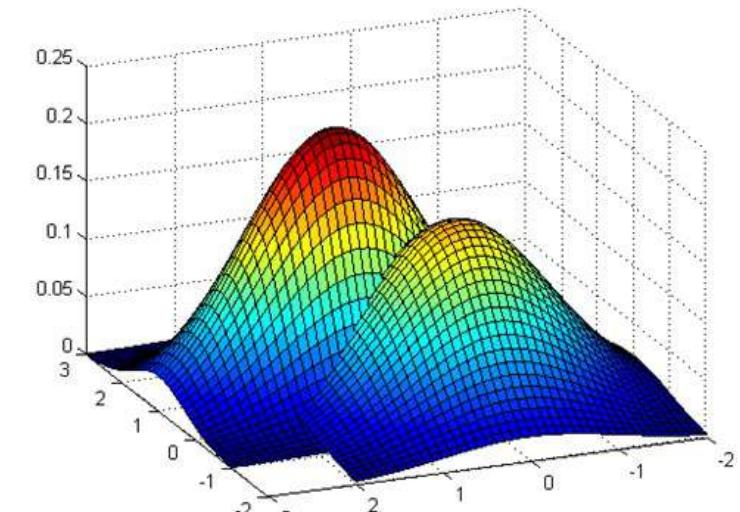
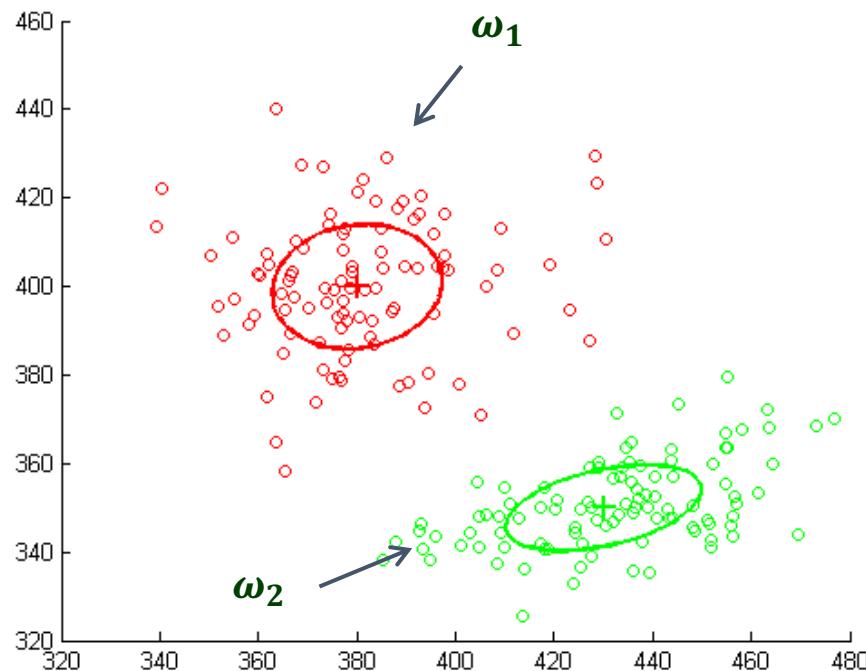
- Step1
 - Collect data ω_1 and ω_2 (= Sample observation)



Distribution of species ω_1 and ω_2 in a 2-D feature space

Implementation: ML Bayesian classifier

- Step2
- Compute likelihood $p(x|\omega_1)$ and $p(x|\omega_2)$ assuming multivariate Gaussian distribution from observation data.
- To obtain $p(x|\omega_1)$ and $p(x|\omega_2)$, compute mean and covariance from each data.(Assume Gaussian distribution)



$$p(x|\omega_i) = \frac{1}{(2\pi)^{n/2} |\tilde{\Sigma}|^{1/2}} \exp \left[\frac{1}{2} (x - \hat{\mu})^T [\tilde{\Sigma}]^{-1} (x - \hat{\mu}) \right]$$

$$\hat{\mu} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$$

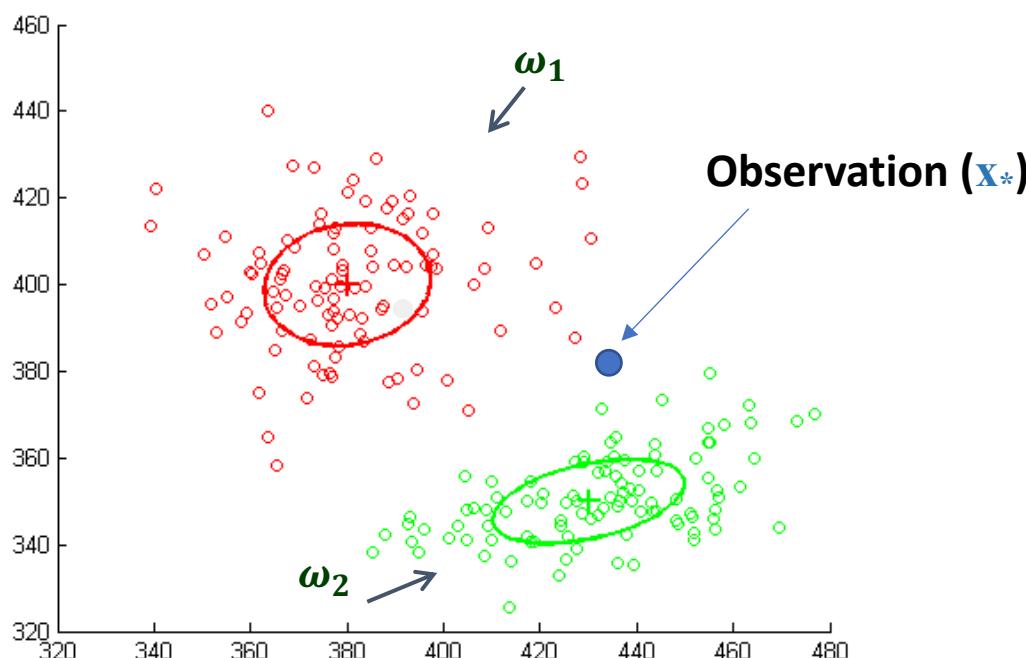
$$\tilde{\Sigma} = \frac{1}{N-1} \sum_{n=1}^N (\mathbf{x}_n - \hat{\mu})(\mathbf{x}_n - \hat{\mu})^T$$

Implementation: ML Bayesian classifier

- Step 3
 - For a given observation \mathbf{x}_*
 -

$$p(\mathbf{x}_* | \omega_1) = \frac{1}{(2\pi)^{n/2} |\tilde{\Sigma}_1|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x}_* - \hat{\mu}_1)^T [\tilde{\Sigma}_1]^{-1} (\mathbf{x}_* - \hat{\mu}_1) \right]$$

$$p(\mathbf{x}_* | \omega_2) = \frac{1}{(2\pi)^{n/2} |\tilde{\Sigma}_2|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x}_* - \hat{\mu}_2)^T [\tilde{\Sigma}_2]^{-1} (\mathbf{x}_* - \hat{\mu}_2) \right]$$



Compare $p(\mathbf{x}_* | \omega_1)$ and $p(\mathbf{x}_* | \omega_2)$

If $p(\mathbf{x}_* | \omega_1) > p(\mathbf{x}_* | \omega_2)$, fish = Sea bass (ω_1)

If $p(\mathbf{x}_* | \omega_1) < p(\mathbf{x}_* | \omega_2)$, fish = Salmon (ω_2)

Likelihood Ratio Test (LRT)

- Decision Rule for binary classification if the priors are the same
- Decide ω_1 if $p(x|\omega_1) P(\omega_1) > p(x|\omega_2) P(\omega_2)$; otherwise decide ω_2
- Rearrange the expression as

- Choose ω_1 if $\frac{p(x|\omega_1)}{p(x|\omega_2)} > \frac{P(\omega_2)}{P(\omega_1)}$

- And
$$\Lambda(x) = \frac{p(x|\omega_1)}{p(x|\omega_2)} = \textit{Likelihood Ratio}$$

- The decision rule based on the Likelihood Test (LRT)

Likelihood Ratio Test (LRT)

- Given a classification problem with the following class conditional densities, derive a decision rule based on the Likelihood Ratio Test (assume equal priors)

$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-4)^2} \quad p(x|\omega_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-10)^2}$$

Likelihood Ratio Test (LRT)

- Given a classification problem with the following class conditional densities, derive a decision rule based on the Likelihood Ratio Test (assume equal priors)

$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-4)^2} \quad p(x|\omega_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-10)^2}$$

- Solution

- Substituting the given likelihoods and priors into the LRT expression:

$$\Lambda(x) = \frac{p(x|\omega_1)}{p(x|\omega_2)} \stackrel{\omega_1}{>} P(\omega_2) \Leftrightarrow \Lambda(x) = \frac{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-4)^2}}{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-10)^2}} \stackrel{\omega_1}{>} 1$$

Likelihood Ratio Test (LRT)

- Given a classification problem with the following class conditional densities, derive a decision rule based on the Likelihood Ratio Test (assume equal priors)

$$p(x | \omega_1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-4)^2} \quad p(x | \omega_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-10)^2}$$

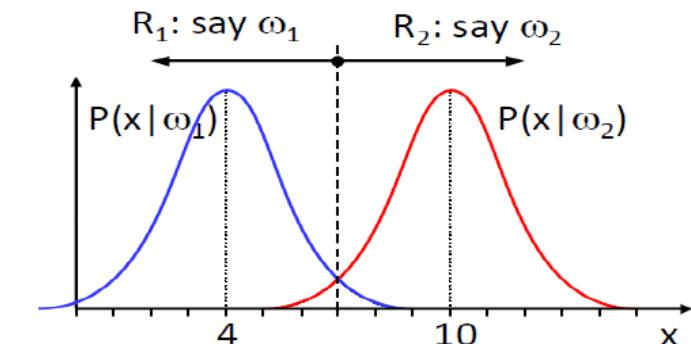
- Solution

- Substituting the given likelihoods and priors into the LRT expression:

$$\Lambda(x) = \frac{p(x | \omega_1)}{p(x | \omega_2)} \stackrel{\omega_1}{>} \frac{P(\omega_2)}{P(\omega_1)} \Leftrightarrow \Lambda(x) = \frac{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-4)^2}}{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-10)^2}} \stackrel{\omega_1}{>} 1$$

- Simplifying the LRT expression:

$$\Lambda(x) = e^{-\frac{1}{2}(x-4)^2 + \frac{1}{2}(x-10)^2} \stackrel{\omega_1}{>} 1 \\ \stackrel{\omega_2}{<}$$



Likelihood Ratio Test (LRT)

- Changing signs and taking logs

$$(x-4)^2 - (x-10)^2 \begin{matrix} > 0 \\[1ex] < \end{matrix}$$

Note

$$\begin{aligned} x^2 - 8x + 16 - x^2 + 20x - 100 \\ = 12x - 84 \\ = 12(x - 7) \end{aligned}$$

Likelihood Ratio Test (LRT)

- Changing signs and taking logs

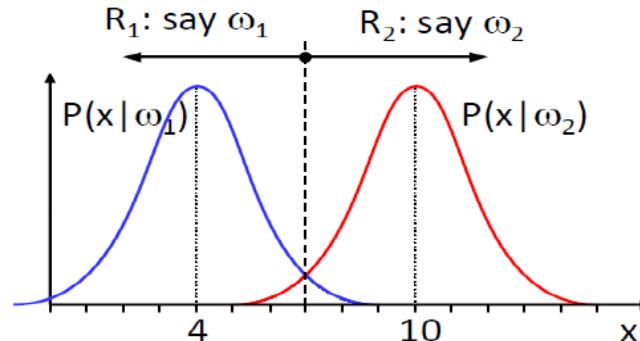
$$(x-4)^2 - (x-10)^2 > 0$$
$$\begin{array}{c} \omega_2 \\ > \\ < \\ \omega_1 \end{array}$$

- Which yields

$$x \begin{array}{c} \omega_2 \\ > \\ < \\ \omega_1 \end{array} 7$$

Note

$$\begin{aligned} x^2 - 8x + 16 - x^2 + 20x - 100 \\ = 12x - 84 \\ = 12(x - 7) \end{aligned}$$



Thus, the decision boundary is determined by the LRT at $x = 7$

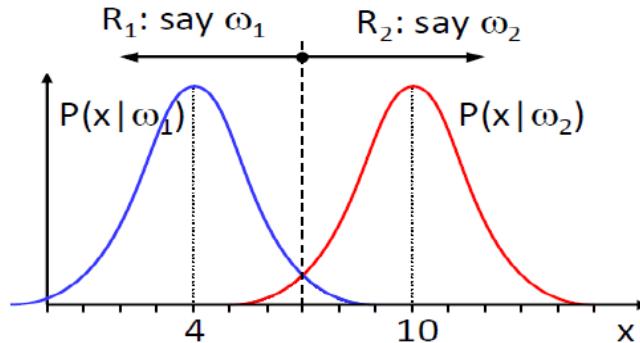
Likelihood Ratio Test (LRT)

- Changing signs and taking logs

$$(x-4)^2 - (x-10)^2 > 0$$
$$\begin{array}{c} \omega_2 \\ > \\ < \\ \omega_1 \end{array}$$

- Which yields

$$x \begin{array}{c} \omega_2 \\ > \\ < \\ \omega_1 \end{array} 7$$



Note

$$\begin{aligned} x^2 - 8x + 16 - x^2 + 20x - 100 \\ = 12x - 84 \\ = 12(x - 7) \end{aligned}$$

Thus, the decision boundary is determined by the LRT at $x = 7$

- How would the LRT decision rule change if, say, the priors were such that $P(\omega_1) = 2P(\omega_2)$?

Risk

- A teenager knows that she will be **grounded** if she chooses to **invite friends** over after school instead of doing her homework, but also knows that the likelihood of her parents finding out she did so is **slight**. If the teenager chooses to invite her friends over, she is taking a risk of getting in trouble with her parents.
- A gambler decides to take all of his winnings from the night and attempt a bet of "double or nothing." The gambler's choice is a **risk** in that he could lose all that he won in one bet.
- A woman gets into her car in the morning and notices that the **gas level is low**. She chooses to drive to work, regardless, without stopping at a gas station. By making this choice she is risking that she will **run out of gas** in her car on the way to work.

- A person taking a new Covid-19 vaccine. What risk?

Risk = Probability x Consequence (Loss)

	Consequence				
	Negligible	Minor	Moderate	Significant	Severe
Very Likely	Low	Moderate	High	High	High
Likely	Low	Moderate	Moderate	High	High
Possible	Low	Low	Moderate	Moderate	High
Unlikely	Low	Low	Moderate	Moderate	Moderate
Very Unlikely	Low	Low	Low	Moderate	Moderate

Bayesian Decision Theory (Bayesian Risk)

- Suppose we have an observation of \mathbf{x} and take action α_i .
- The loss function $\lambda(\alpha_i | \omega_j)$ describes the **loss incurred (consequence)** for taking **action α_i** when the state of nature is ω_j .
- Since $P(\omega_j | \mathbf{x})$ is the probability that the true state of nature is ω_j , the expected total loss associated with taking action α_i is
- $R(\alpha_i | \mathbf{x})$ is defined as conditional risk
- A point value function for \mathbf{x} and α_i
- Thus, with an observation \mathbf{x} , select an action that minimizes the conditional risk. → Bayes Decision Procedure → optimal performance on overall risk

Bayesian Decision Theory (Bayesian Risk)

- Suppose we have an observation of \mathbf{x} and take action α_i .
- The loss function $\lambda(\alpha_i|\omega_j)$ describes the **loss incurred (consequence)** for taking **action α_i** when the state of nature is ω_j .
- Since $P(\omega_j|\mathbf{x})$ is the **probability** that the **true state of nature** is ω_j , the expected **total loss** associated with taking action α_i is

$$R(\alpha_i|\mathbf{x}) = \sum_{j=1}^c \lambda(\alpha_i|\omega_j)P(\omega_j|\mathbf{x})$$

- $R(\alpha_i|\mathbf{x})$ is defined as conditional risk
- A point value function for \mathbf{x} and α_i
- Thus, with an observation \mathbf{x} , select an action that minimizes the conditional risk. → Bayes Decision Procedure → optimal performance on overall risk

Bayesian Decision Theory (Bayesian Risk)

- Suppose we have an observation of \mathbf{x} and take action α_i .
- The loss function $\lambda(\alpha_i|\omega_j)$ describes the **loss incurred (consequence)** for taking **action α_i** when the state of nature is ω_j .
- Since $P(\omega_j|\mathbf{x})$ is the **probability** that the **true state of nature** is ω_j , the expected **total loss** associated with taking action α_i is

$$R(\alpha_i|\mathbf{x}) = \sum_{j=1}^c \lambda(\alpha_i|\omega_j)P(\omega_j|\mathbf{x})$$

- $R(\alpha_i|\mathbf{x})$ is defined as **conditional risk**
- A point value function for **\mathbf{x} and α_i**
- Thus, with an observation \mathbf{x} , select an action that minimizes the conditional risk. → Bayes Decision Procedure → optimal performance on overall risk

Bayesian Decision Theory (Bayesian Risk)

- Our goal is to find a **decision rule** that **minimizes** the overall risk.
- The **decision function** $\alpha(\mathbf{x})$: assumes **one of the α values** $\alpha_1, \dots, \alpha_a$ for every \mathbf{x} .
- The **overall risk R** : expected loss associated with a given decision rule.
- Since $R(\alpha_i|\mathbf{x})$ is the **conditional risk** associated with **action α_i** , and since the decision rule specifies the action, the **overall risk** is given by

$$R = \int R(\alpha(\mathbf{x})|\mathbf{x})p(\mathbf{x})d\mathbf{x}$$

- where $d\mathbf{x}$ is a d-dimensional volume element and the **integral extends** over the **entire feature space**.

Bayesian Decision Theory (Bayesian Risk)

- Clearly, if $\alpha(x)$ is chosen so that $R(\alpha_i|x)$ is as small as possible for every x , then the overall risk will be minimized.
- To minimize the overall risk, compute the conditional risk

$$R(\alpha_i|x) = \sum_{j=1}^c \lambda(\alpha_i|\omega_j)P(\omega_j|x)$$

- for $i=1, \dots, a$ and select α_i for which $R(\alpha_i|x)$ is minimum.

Two Category Classification by Bayesian Risk

- For two-category classification problems.
 - Action α_1 corresponds to deciding that the true state of nature is ω_1
 - Action α_2 corresponds to deciding that it is ω_2 .
- Recall the loss function $\lambda(\alpha_i|\omega_j)$ describes the loss incurred for taking action α_i when the state of nature is ω_j .
- The conditional risk
 - $R(\alpha_1|x) = \lambda_{11}P(\omega_1|x) + \lambda_{12}P(\omega_2|x)$
 - $R(\alpha_2|x) = \lambda_{21}P(\omega_1|x) + \lambda_{22}P(\omega_2|x)$
- Decide ω_1
 - if $R(\alpha_1|x) < R(\alpha_2|x)$
- Decide ω_2 otherwise

Two Category Classification by Bayesian Risk

- $R(\alpha_1|\mathbf{x}) = \lambda_{11}P(\omega_1|\mathbf{x}) + \lambda_{12}P(\omega_2|\mathbf{x})$
- $R(\alpha_2|\mathbf{x}) = \lambda_{21}P(\omega_1|\mathbf{x}) + \lambda_{22}P(\omega_2|\mathbf{x})$
- Decide ω_1
- if $R(\alpha_1|\mathbf{x}) < R(\alpha_2|\mathbf{x})$
$$\lambda_{11}P(\omega_1|\mathbf{x}) + \lambda_{12}P(\omega_2|\mathbf{x}) < \lambda_{21}P(\omega_1|\mathbf{x}) + \lambda_{22}P(\omega_2|\mathbf{x})$$
- Collect like terms
 - $$\lambda_{12}P(\omega_2|\mathbf{x}) - \lambda_{22}P(\omega_2|\mathbf{x}) < \lambda_{21}P(\omega_1|\mathbf{x}) - \lambda_{11}P(\omega_1|\mathbf{x})$$
 - $$(\lambda_{12} - \lambda_{22})P(\omega_2|\mathbf{x}) < (\lambda_{21} - \lambda_{11}) P(\omega_1|\mathbf{x})$$
 - $$(\lambda_{21} - \lambda_{11}) P(\omega_1|\mathbf{x}) > (\lambda_{12} - \lambda_{22})P(\omega_2|\mathbf{x})$$

Two Category Classification by Bayesian Risk

- Recall λ_{21} = Loss incurred by choosing ω_2 when it was actually ω_1
- and λ_{11} = Loss incurred by choosing ω_1 when it was actually ω_1
- Loss incurred when choosing correctly should be less than the loss when incorrect choice was made.
- Thus $(\lambda_{21} - \lambda_{11}) > 0$ and $(\lambda_{12} - \lambda_{22}) > 0$
- Again, decide ω_1 if $(\lambda_{21} - \lambda_{11}) P(\omega_1|\mathbf{x}) > (\lambda_{12} - \lambda_{22}) P(\omega_2|\mathbf{x})$
- Substitute $P(\omega_1|\mathbf{x})$ by Bayes rule
$$(\lambda_{21} - \lambda_{11}) p(\mathbf{x}|\omega_1)P(\omega_1) > (\lambda_{12} - \lambda_{22}) p(\mathbf{x}|\omega_2)P(\omega_2)$$
- Alternatively, decide ω_1 if $\frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} > \frac{(\lambda_{12} - \lambda_{22})}{(\lambda_{21} - \lambda_{11})} \frac{P(\omega_2)}{P(\omega_1)}$
- Choose ω_2 otherwise

The Bayes Risk Example

- Consider a classification problem with two classes defined by the following likelihood functions

$$p(x | \omega_1) = \frac{1}{\sqrt{2\pi}\sqrt{3}} e^{-\frac{1}{2}\frac{x^2}{3}}$$

$$p(x | \omega_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-2)^2} .$$

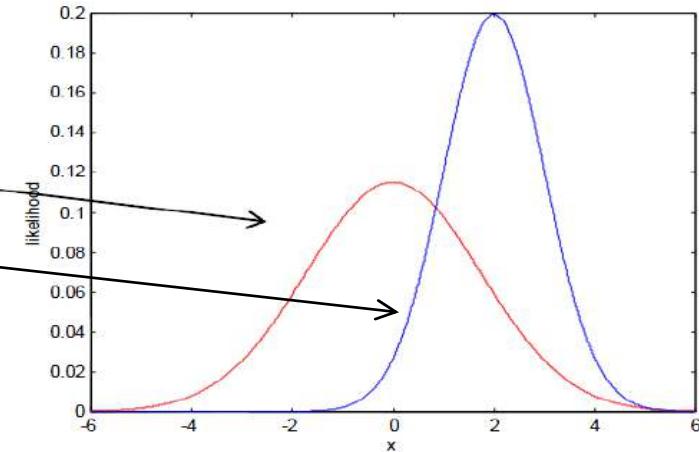
The Bayes Risk Example

- Consider a classification problem with two classes defined by the following likelihood functions

$$p(x | \omega_1) = \frac{1}{\sqrt{2\pi}\sqrt{3}} e^{-\frac{1}{2} \frac{x^2}{3}}$$

$$p(x | \omega_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-2)^2}$$

- Sketch the two densities
- What is the likelihood ratio?



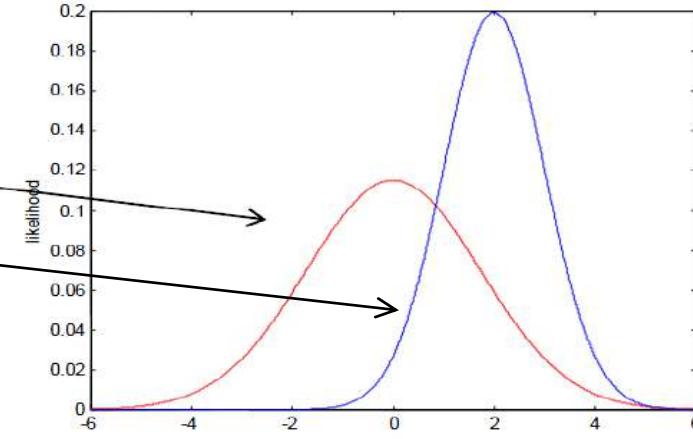
The Bayes Risk Example

- Consider a classification problem with two classes defined by the following likelihood functions

$$p(x | \omega_1) = \frac{1}{\sqrt{2\pi}\sqrt{3}} e^{-\frac{1}{2} \frac{x^2}{3}}$$

$$p(x | \omega_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-2)^2}$$

- Sketch the two densities
- What is the likelihood ratio?
- Assume $P(\omega_1) = P(\omega_2) = 0.5$, $C_{11} = C_{22} = 0$, $C_{12} = 1$ and $C_{21} = 3^{1/2}$ Determine a decision rule that minimizes the probability of error



The Bayes Risk Example

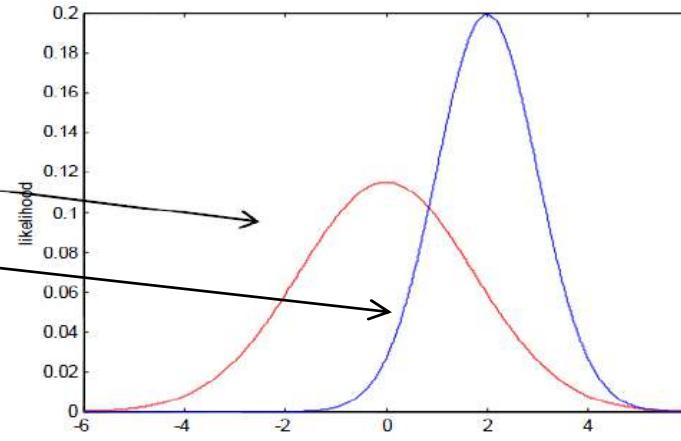
- Consider a classification problem with two classes defined by the following likelihood functions

$$p(x | \omega_1) = \frac{1}{\sqrt{2\pi}\sqrt{3}} e^{-\frac{1}{2}\frac{x^2}{3}}$$

$$p(x | \omega_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-2)^2}$$

- Sketch the two densities
- What is the likelihood ratio?
- Assume $P(\omega_1) = P(\omega_2) = 0.5$, $C_{11} = C_{22} = 0$, $C_{12} = 1$ and $C_{21} = 3^{1/2}$ Determine a decision rule that minimizes the probability of error

$$\begin{aligned} \frac{\frac{1}{\sqrt{2\pi}\sqrt{3}} e^{-\frac{1}{2}\frac{x^2}{3}}}{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-2)^2}} &> \frac{1}{\sqrt{3}} \\ \frac{e^{-\frac{1}{2}\frac{x^2}{3}}}{e^{-\frac{1}{2}(x-2)^2}} &< \frac{1}{\sqrt{3}} \end{aligned}$$



The Bayes Risk Example

- Consider a classification problem with two classes defined by the following likelihood functions

$$p(x | \omega_1) = \frac{1}{\sqrt{2\pi}\sqrt{3}} e^{-\frac{1}{2}\frac{x^2}{3}}$$

$$p(x | \omega_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-2)^2}$$

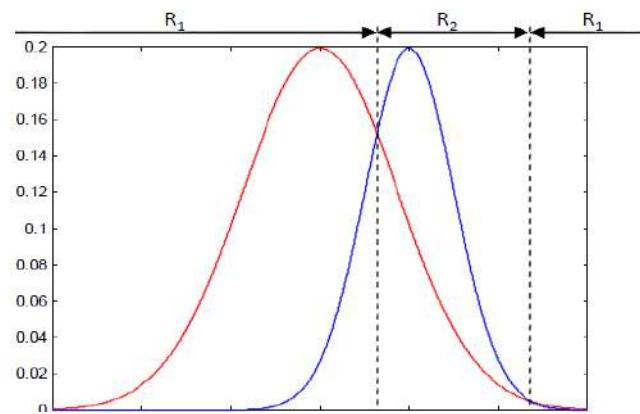
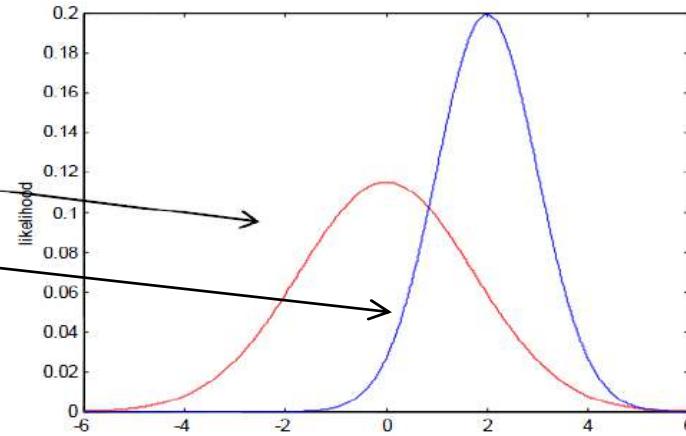
- Sketch the two densities
- What is the likelihood ratio?
- Assume $P(\omega_1) = P(\omega_2) = 0.5$, $C_{11} = C_{22} = 0$, $C_{12} = 1$ and $C_{21} = 3^{1/2}$ Determine a decision rule that minimizes the probability of error

$$\frac{\frac{1}{\sqrt{2\pi}\sqrt{3}} e^{-\frac{1}{2}\frac{x^2}{3}}}{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-2)^2}} > \frac{1}{\sqrt{3}}$$

$$= \frac{e^{-\frac{1}{2}\frac{x^2}{3}}}{e^{-\frac{1}{2}(x-2)^2}} > 1$$

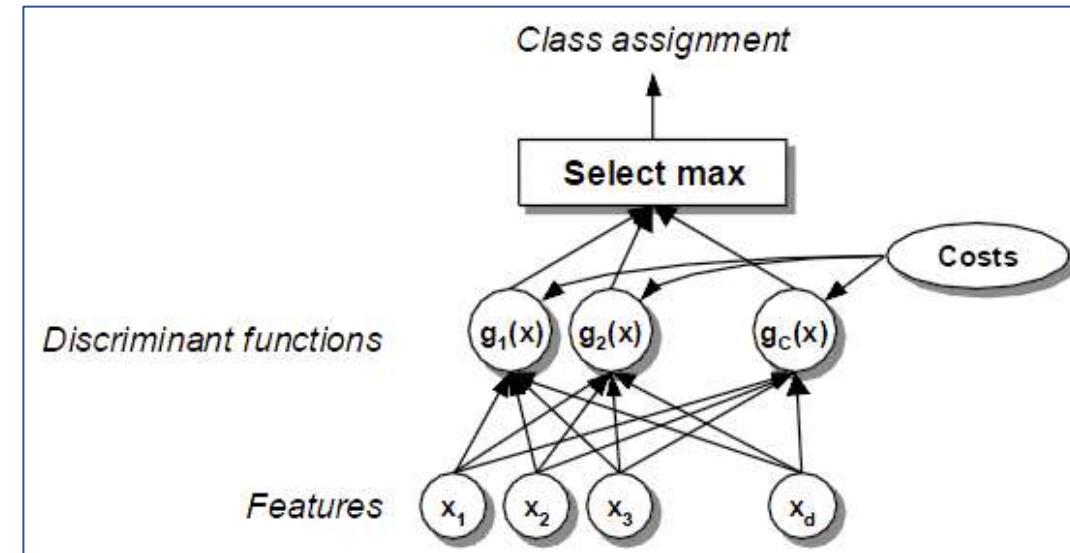
$$-\frac{1}{2}\frac{x^2}{3} + \frac{1}{2}(x-2)^2 > 0$$

$$2x^2 - 12x + 12 > 0 \Rightarrow x = \frac{\omega_1}{\omega_2} = 4.73, 1.27$$



Discriminant Function

- All the decision rules so far have the same structure
 - At each point \mathbf{x} in feature space choose class ω_i which maximizes (or minimizes) some measure $g_i(x)$
- This structure can be formalized with a set of discriminant functions $g_i(\mathbf{x})$, $i=1, \dots, C$, and the following decision rule:
 - "assign \mathbf{x} to class ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for every $j \neq i$ "
- Therefore, we can visualize the decision rule as a network or machine that computes C discriminant functions and selects the category corresponding to the largest discriminant. Such network is depicted as
- Summarize the three decision rules: Bayes Risk, MAP, and ML



Decision Rule	Discriminant Function
Bayes Risk	$g_i(x) = R(\omega_i \mathbf{x})$
MAP	$g_i(x) = P(\omega_i \mathbf{x})$
ML	$g_i(x) = p(x \omega_i)$

Bayesian Decision Theory

- In summary
- MAP
 - Maximum A Posteriori determined by $P(w_j|x) = \frac{p(x|w_j)P(w_j)}{p(x)}$
 - $P(\omega_j|x)$ directly if possible
- Or by using Bayes rule $\frac{p(x|\omega_j)P(\omega_j)}{p(x)}$
- Maximum Likelihood (ML)
 - Determined by the maximum likelihood: $P(x | \omega_j)$
- Bayes Risk Based choosing the minimum of $R(\alpha_i|x) = \sum_{j=1}^c \lambda(\alpha_i|\omega_j)P(\omega_j|x)$

Bayes classifiers for Gaussian distributed classes

□ General expression for Gaussian densities

- The multivariate Normal density function was defined as

$$f_{\mathbf{x}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right]$$

Bayes classifiers for Gaussian distributed classes

□ General expression for Gaussian densities

- The multivariate Normal density function was defined as

$$f_{\mathbf{x}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right]$$

- Using Bayes rule, the MAP discriminant function becomes

$$g_i(\mathbf{x}) = P(\omega_i | \mathbf{x}) = \frac{p(\mathbf{x} | \omega_i)P(\omega_i)}{p(\mathbf{x})} = \frac{1}{(2\pi)^{n/2} |\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right] P(\omega_i) \frac{1}{p(\mathbf{x})}$$

Bayes classifiers for Gaussian distributed classes

□ General expression for Gaussian densities

- The multivariate Normal density function was defined as

$$f_{\mathbf{x}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right]$$

- Using Bayes rule, the MAP discriminant function becomes

$$g_i(\mathbf{x}) = P(\omega_i | \mathbf{x}) = \frac{p(\mathbf{x} | \omega_i)P(\omega_i)}{p(\mathbf{x})} = \frac{1}{(2\pi)^{n/2} |\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right] P(\omega_i) \frac{1}{p(\mathbf{x})}$$

Sine the MAP picks the largest of g_i 's let's eliminate common terms

$$g_i(\mathbf{x}) = |\Sigma_i|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right] P(\omega_i)$$

Bayes classifiers for Gaussian distributed classes

□ General expression for Gaussian densities

- The multivariate Normal density function was defined as

$$f_{\mathbf{x}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right]$$

- Using Bayes rule, the MAP discriminant function becomes

$$g_i(\mathbf{x}) = P(\omega_i | \mathbf{x}) = \frac{p(\mathbf{x} | \omega_i)P(\omega_i)}{p(\mathbf{x})} = \frac{1}{(2\pi)^{n/2} |\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right] P(\omega_i) \frac{1}{p(\mathbf{x})}$$

Sine the MAP picks the largest of g_i s let's eliminate common terms

$$g_i(\mathbf{x}) = |\Sigma_i|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right] P(\omega_i)$$

Again, as the MAP picks the largest of g_i s let's take natural logs since logarithm is a monotonically increasing function

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\Sigma_i|) + \log(P(\omega_i))$$

- This expression is called a **quadratic discriminant function**

Case 1: $\Sigma_i = \sigma^2 I$

- This situation occurs when the features are statistically independent with the same variance for all classes*

- In this case, the quadratic discriminant function becomes

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i))$$

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T (\sigma^2 \mathbf{I})^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\sigma^2 \mathbf{I}|) + \log(P(\omega_i))$$

$$= -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_i)^T(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2}N \log(\sigma^2) + \log(P(\omega_i))$$

Case 1: $\Sigma_i = \sigma^2 I$

- This situation occurs when the features are statistically independent with the same variance for all classes*

- In this case, the quadratic discriminant function becomes

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i))$$

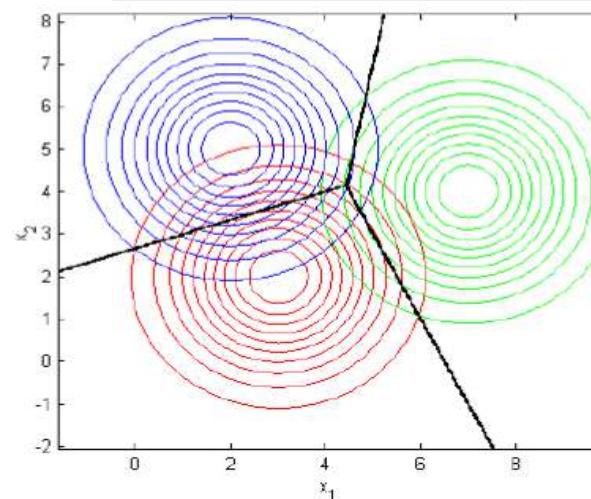
$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T (\sigma^2 \mathbf{I})^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\sigma^2 \mathbf{I}|) + \log(P(\omega_i))$$

$$= -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_i)^T(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} N \log(\sigma^2) + \log(P(\omega_i))$$

dropping the second term

$$= -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_i)^T(\mathbf{x} - \boldsymbol{\mu}_i) + \log(P(\omega_i))$$

$$\begin{aligned}\boldsymbol{\mu}_1 &= [3 \quad 2]^T & \boldsymbol{\mu}_2 &= [7 \quad 4]^T & \boldsymbol{\mu}_3 &= [2 \quad 5]^T \\ \boldsymbol{\Sigma}_1 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_2 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_3 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\end{aligned}$$



Case 1: $\Sigma_i = \sigma^2 I$

- This situation occurs when the features are statistically independent with the same variance for all classes*

- In this case, the quadratic discriminant function becomes

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i))$$

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T (\sigma^2 \mathbf{I})^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\sigma^2 \mathbf{I}|) + \log(P(\omega_i))$$

$$= -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_i)^T(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2}N \log(\sigma^2) + \log(P(\omega_i))$$

dropping the second term

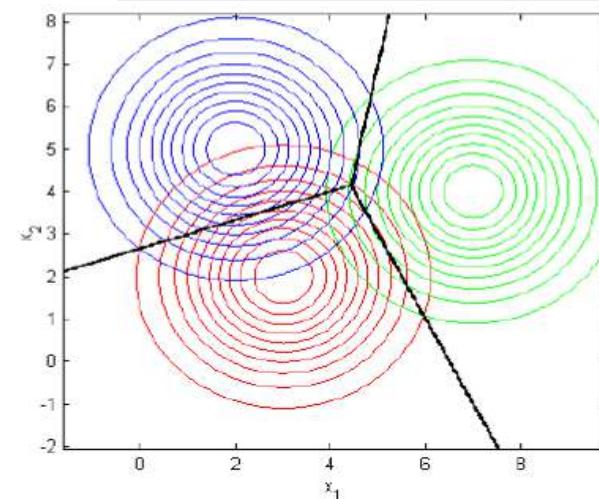
$$= -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_i)^T(\mathbf{x} - \boldsymbol{\mu}_i) + \log(P(\omega_i))$$

- Expanding this expression

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_i)^T(\mathbf{x} - \boldsymbol{\mu}_i) + \log(P(\omega_i))$$

$$= -\frac{1}{2\sigma^2}(\mathbf{x}^T \mathbf{x} - 2\boldsymbol{\mu}_i^T \mathbf{x} + \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i) + \log(P(\omega_i))$$

$$\begin{aligned}\boldsymbol{\mu}_1 &= [3 \quad 2]^T & \boldsymbol{\mu}_2 &= [7 \quad 4]^T & \boldsymbol{\mu}_3 &= [2 \quad 5]^T \\ \boldsymbol{\Sigma}_1 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_2 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_3 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\end{aligned}$$



Case 1: $\Sigma_i = \sigma^2 I$

- Eliminating the term $\mathbf{x}^\top \mathbf{x}$, which is constant for all classes

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(-2\boldsymbol{\mu}_i^\top \mathbf{x} + \boldsymbol{\mu}_i^\top \boldsymbol{\mu}_i) + \log(P(\omega_i)) = \boldsymbol{\omega}_i^\top \mathbf{x} + \omega_{i0}$$

where
$$\begin{cases} \boldsymbol{\omega}_i^\top = \frac{\boldsymbol{\mu}_i}{\sigma^2} \\ \omega_{i0} = -\frac{1}{2\sigma^2} \boldsymbol{\mu}_i^\top \boldsymbol{\mu}_i + \log(P(\omega_i)) \end{cases}$$

$\boldsymbol{\omega}_i^\top$ and ω_{i0} are constants

- Since the discriminant is linear, the decision boundaries $g_i(\mathbf{x}) = g_j(\mathbf{x})$, will be hyper-planes

From [schalkoff, 1992]

Case 1: $\Sigma_i = \sigma^2 I$

- Eliminating the term $\mathbf{x}^T \mathbf{x}$, which is constant for all classes

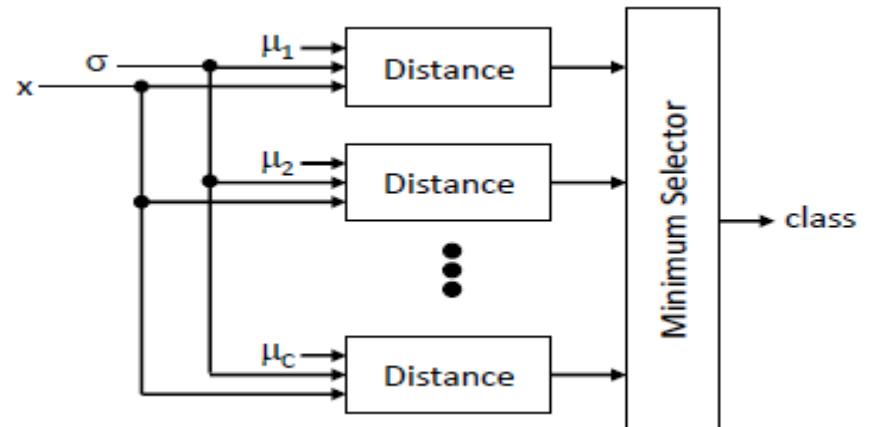
$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(-2\boldsymbol{\mu}_i^T \mathbf{x} + \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i) + \log(P(\omega_i)) = w_i^T \mathbf{x} + w_{i0}$$

where
$$\begin{cases} w_i^T = \frac{\boldsymbol{\mu}_i}{\sigma^2} \\ w_{i0} = -\frac{1}{2\sigma^2} \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + \log(P(\omega_i)) \end{cases}$$

- Since the discriminant is linear, the decision boundaries $g_i(\mathbf{x}) = g_j(\mathbf{x})$, will be hyper-planes
- If we assume equal priors

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_i)^T(\mathbf{x} - \boldsymbol{\mu}_i)$$

w_i^T and w_{i0} are constants



Depends on how far away \mathbf{x} is from the mean

- This is called a minimum-distance or nearest mean classifier
- The loci of constant probability for each class are hyper-spheres

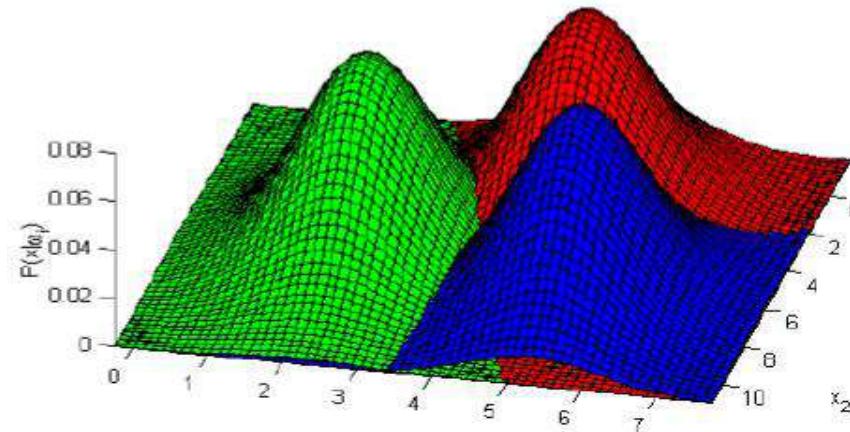
From [schalkoff, 1992]

Case 1: $\Sigma_i = \sigma^2 I$

- To illustrate the previous result, we will compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\boldsymbol{\mu}_1 = [3 \ 2]^T \quad \boldsymbol{\mu}_2 = [7 \ 4]^T \quad \boldsymbol{\mu}_3 = [2 \ 5]^T$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

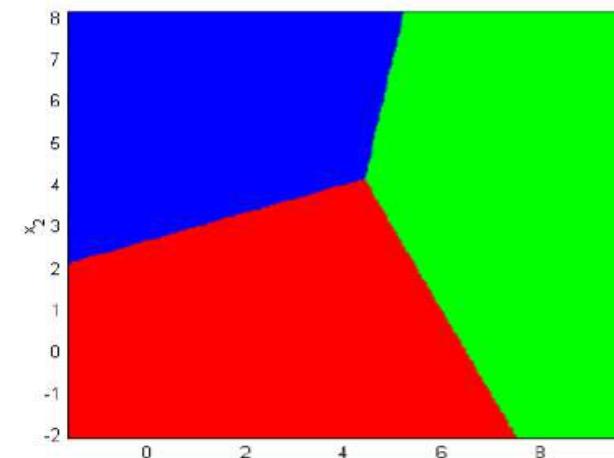
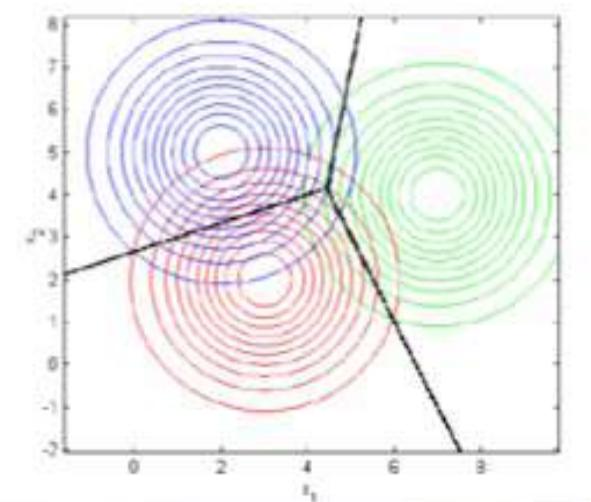
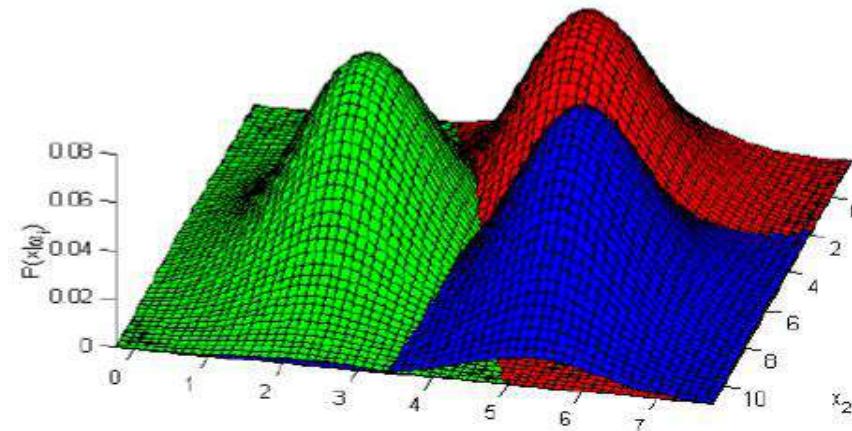


Case 1: $\Sigma_i = \sigma^2 I$

- To illustrate the previous result, we will compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\mu_1 = [3 \ 2]^T \quad \mu_2 = [7 \ 4]^T \quad \mu_3 = [2 \ 5]^T$$

$$\Sigma_1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad \Sigma_2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad \Sigma_3 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$



Case 2: $\Sigma_i = \Sigma$ (Σ diagonal)

- **The classes still have the same covariance matrix, but the features are allowed to have different variances**
 - In this case, the quadratic discriminant function becomes

$$\begin{aligned}g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2}\log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i)) \\&= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \begin{bmatrix} \sigma_1^{-2} & & \\ & \ddots & \\ & & \sigma_N^{-2} \end{bmatrix} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2}\log \begin{pmatrix} |\sigma_1|^2 & & \\ & \ddots & \\ & & |\sigma_N|^2 \end{pmatrix} + \log(P(\omega_i))\end{aligned}$$

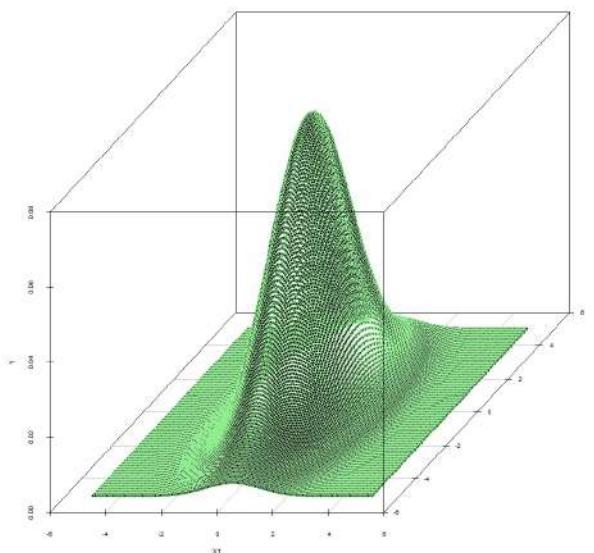
Case 2: $\Sigma_i = \Sigma$ (Σ diagonal)

- **The classes still have the same covariance matrix, but the features are allowed to have different variances**

- In this case, the quadratic discriminant function becomes

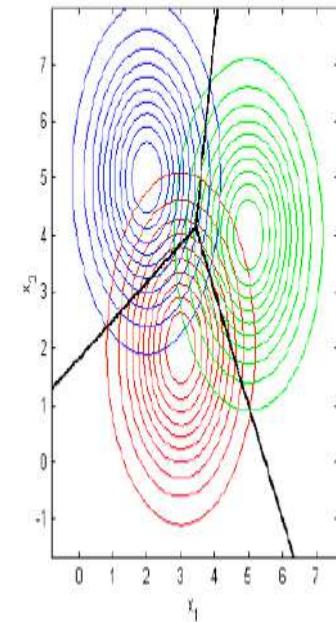
$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2}\log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i))$$

$$= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \begin{bmatrix} \sigma_1^{-2} & & \\ & \ddots & \\ & & \sigma_N^{-2} \end{bmatrix} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2}\log\left(\begin{vmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_N^2 \end{vmatrix}\right) + \log(P(\omega_i))$$



$$\begin{aligned} \boldsymbol{\mu}_1 &= [3 \quad 2]^T & \boldsymbol{\mu}_2 &= [5 \quad 4]^T & \boldsymbol{\mu}_3 &= [2 \quad 5]^T \\ \boldsymbol{\Sigma}_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_3 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$

From H. Ko, "Bayesian Learning, Naïve Bayes Classifier"



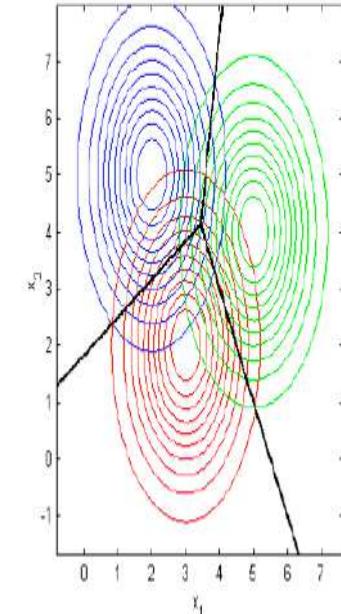
Case 2: $\Sigma_i = \Sigma$ (Σ diagonal)

- **The classes still have the same covariance matrix, but the features are allowed to have different variances**

- In this case, the quadratic discriminant function becomes

$$\begin{aligned}
 g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i)) \\
 &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \begin{bmatrix} \sigma_1^{-2} & & \\ & \ddots & \\ & & \sigma_N^{-2} \end{bmatrix} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log \begin{pmatrix} |\sigma_1|^2 & & \\ & \ddots & \\ & & |\sigma_N|^2 \end{pmatrix} + \log(P(\omega_i)) \\
 &= -\frac{1}{2} \sum_{k=1}^N \frac{(\mathbf{x}[k] - \boldsymbol{\mu}_i[k])^2}{\sigma_k^2} - \frac{1}{2} \log \prod_{k=1}^N \sigma_k^2 + \log(P(\omega_i)) \\
 &= -\frac{1}{2} \sum_{k=1}^N \frac{\mathbf{x}[k]^2 - 2\mathbf{x}[k]\boldsymbol{\mu}_i[k] + \boldsymbol{\mu}_i[k]^2}{\sigma_k^2} - \frac{1}{2} \log \prod_{k=1}^N \sigma_k^2 + \log(P(\omega_i))
 \end{aligned}$$

$$\begin{aligned}
 \boldsymbol{\mu}_1 &= [3 \quad 2]^T & \boldsymbol{\mu}_2 &= [5 \quad 4]^T & \boldsymbol{\mu}_3 &= [2 \quad 5]^T \\
 \boldsymbol{\Sigma}_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_3 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}
 \end{aligned}$$



Case 2: $\Sigma_i = \Sigma$ (Σ diagonal)

- Eliminating the term $x[k]^2$, which is constant for all classes

$$g_i(\mathbf{x}) = -\frac{1}{2} \sum_{k=1}^N \frac{-2x[k]\mu_i[k] + \mu_i[k]^2}{\sigma_k^2} - \frac{1}{2} \log \prod_{k=1}^N \sigma_k^2 + \log(P(\omega_i))$$

- This discriminant is linear, so the decision boundaries $g_i(x) = g_j(x)$, will also be hyper-planes

Case 2: $\Sigma_i = \Sigma$ (Σ diagonal)

- Eliminating the term $x[k]^2$, which is constant for all classes

$$g_i(\mathbf{x}) = -\frac{1}{2} \sum_{k=1}^N \frac{-2\mathbf{x}[k]\boldsymbol{\mu}_i[k] + \boldsymbol{\mu}_i[k]^2}{\sigma_k^2} - \frac{1}{2} \log \prod_{k=1}^N \sigma_k^2 + \log(P(\omega_i))$$

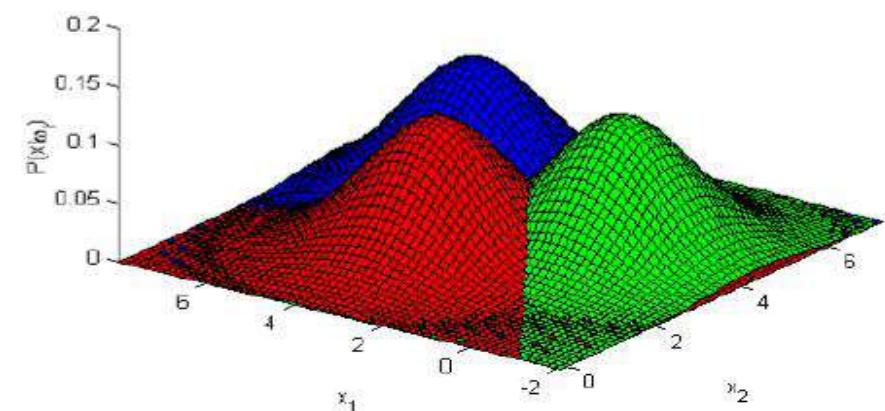
- This discriminant is linear, so the decision boundaries $g_i(x) = g_j(x)$, will also be hyper-planes
- The loci of constant probability are hyper-ellipses aligned with feature axes
- Note that only difference with the previous classifier is that the distance of each axis is normalized by the variance of the axis

Case 2: $\Sigma_i = \Sigma$ (Σ diagonal), Example

- To illustrate the previous result, we will compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\boldsymbol{\mu}_1 = [3 \ 2]^T \quad \boldsymbol{\mu}_2 = [5 \ 4]^T \quad \boldsymbol{\mu}_3 = [2 \ 5]^T$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

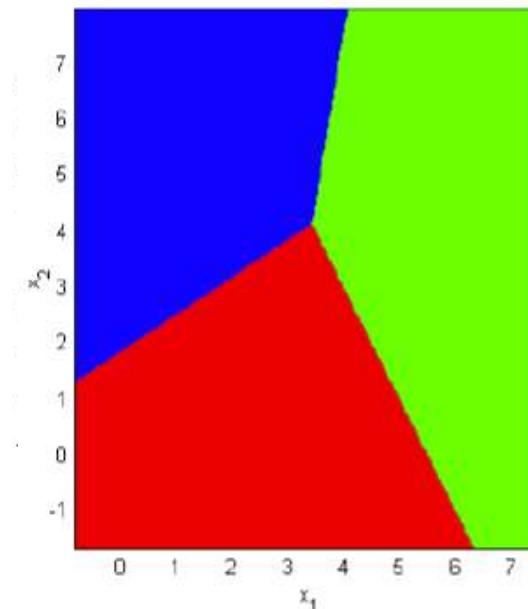
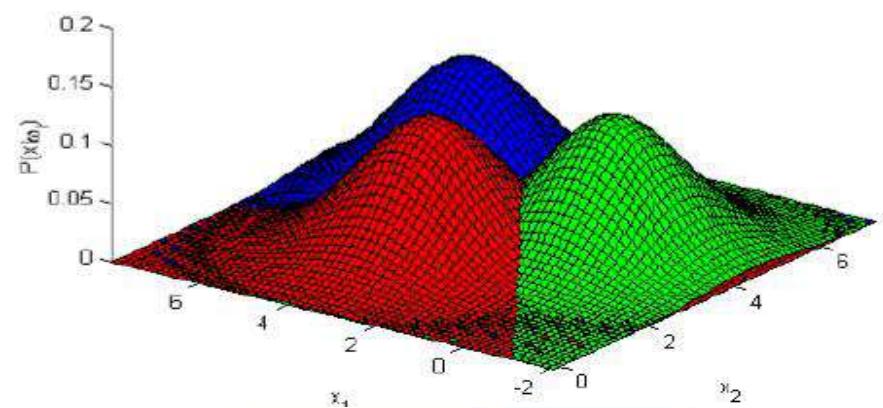
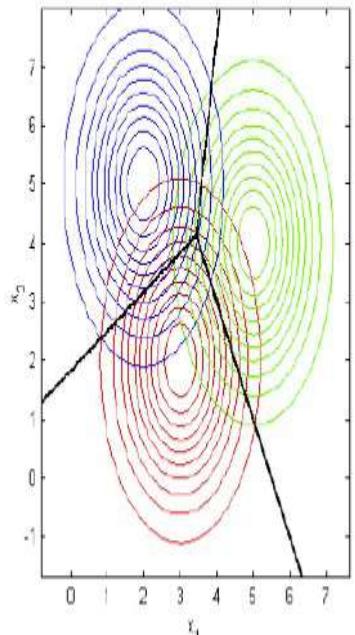


Case 2: $\Sigma_i = \Sigma$ (Σ diagonal), Example

- To illustrate the previous result, we will compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\boldsymbol{\mu}_1 = [3 \ 2]^T \quad \boldsymbol{\mu}_2 = [5 \ 4]^T \quad \boldsymbol{\mu}_3 = [2 \ 5]^T$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$



From H. Ko, "Bayesian Learning, Naïve Bayes Classifier"

Case 3: $\Sigma_i = \Sigma$ (Σ non-diagonal)

- In this case, all the classes have the same covariance matrix, but this is no longer diagonal
- The quadratic discriminant becomes

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i))$$

Case 3: $\Sigma_i = \Sigma$ (Σ non-diagonal)

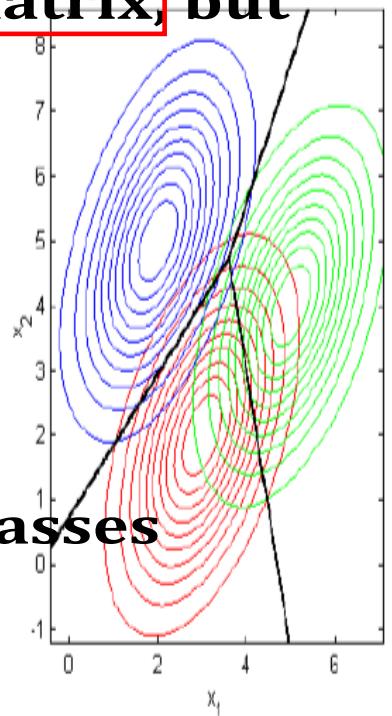
- In this case, all the classes have the same covariance matrix, but this is no longer diagonal
- The quadratic discriminant becomes

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\boldsymbol{\Sigma}|) + \log(P(\omega_i))$$

- Eliminating the term $\log|\boldsymbol{\Sigma}|$, which is constant for all classes

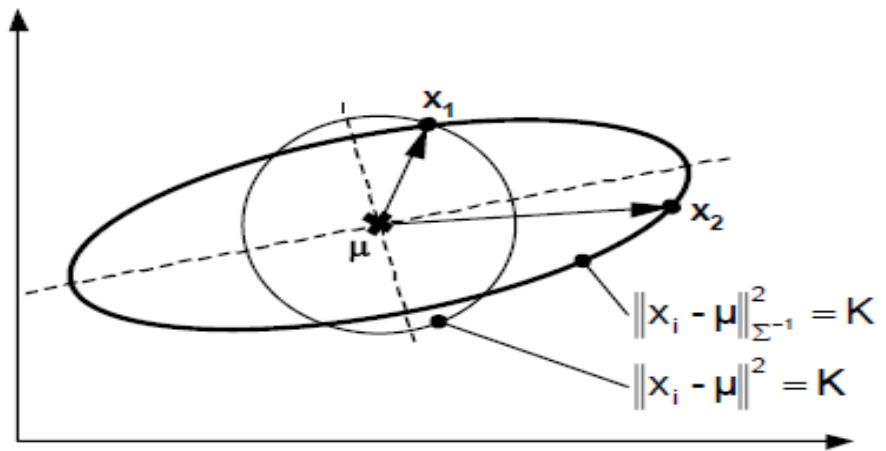
$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) + \log(P(\omega_i))$$

- The quadratic term is called the Mahalanobis distance, a very important distance in Statistical PR



Case 3: $\Sigma_i = \Sigma$ (Σ non-diagonal)

- **The Mahalanobis distance is a vector distance that uses a Σ^{-1} norm**
 - Σ^{-1} can be thought of as a stretching factor on the space
 - Note that for an identity covariance matrix ($\Sigma=I$), the Mahalanobis distance becomes the familiar Euclidean distance



Mahalanobis Distance

$$\|\mathbf{x} - \mathbf{y}\|_{\Sigma^{-1}}^2 = (\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})$$

Case 3: $\Sigma_i = \Sigma$ (Σ non-diagonal)

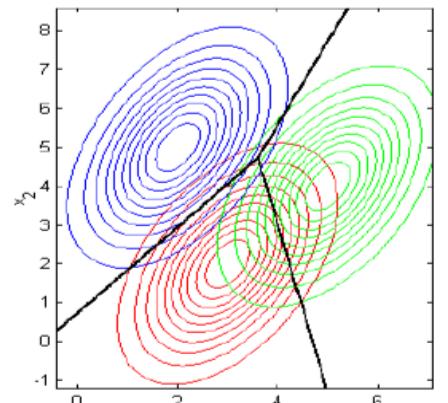
- Expansion of the quadratic term in the discriminant yields

$$\begin{aligned} g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) + \log(P(\omega_i)) \\ &= -\frac{1}{2}(\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x} - 2\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{x} + \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i) + \log(P(\omega_i)) \end{aligned}$$

- Removing the term $\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x}$, which is constant for all classes

$$g_i(\mathbf{x}) = -\frac{1}{2}(-2\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{x} + \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i) + \log(P(\omega_i))$$

$$\begin{aligned} \boldsymbol{\mu}_1 &= [3 \quad 2]^T & \boldsymbol{\mu}_2 &= [5 \quad 4]^T & \boldsymbol{\mu}_3 &= [2 \quad 5]^T \\ \boldsymbol{\Sigma}_1 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_2 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_3 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} \end{aligned}$$



Case 3: $\Sigma_i = \Sigma$ (Σ non-diagonal)

- Expansion of the quadratic term in the discriminant yields

$$\begin{aligned} g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) + \log(P(\omega_i)) \\ &= -\frac{1}{2}(\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x} - 2\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{x} + \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i) + \log(P(\omega_i)) \end{aligned}$$

- Removing the term $\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x}$, which is constant for all classes

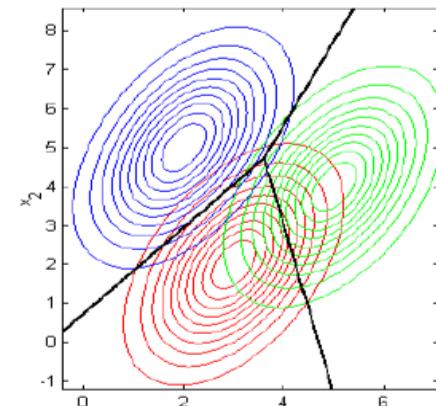
$$g_i(\mathbf{x}) = -\frac{1}{2}(-2\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{x} + \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i) + \log(P(\omega_i))$$

- Reorganizing terms we obtain

$$\mathbf{g}_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

where $\left\{ \begin{array}{l} \mathbf{w}_i^T = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i \\ w_{i0} = -\frac{1}{2} \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i + \log(P(\omega_i)) \end{array} \right.$

$$\begin{aligned} \boldsymbol{\mu}_1 &= [3 \quad 2]^T & \boldsymbol{\mu}_2 &= [5 \quad 4]^T & \boldsymbol{\mu}_3 &= [2 \quad 5]^T \\ \boldsymbol{\Sigma}_1 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_2 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_3 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} \end{aligned}$$

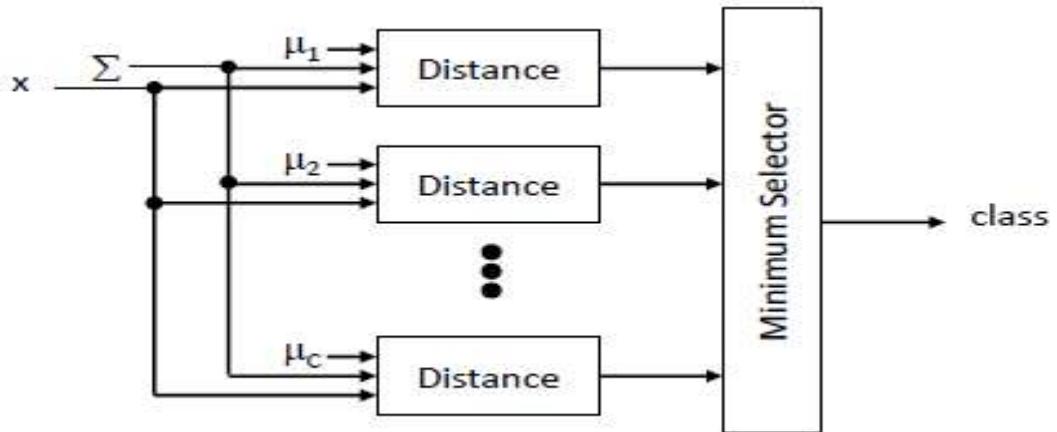


Case 3: $\Sigma_i = \Sigma$ (Σ non-diagonal)

- This discriminant is linear, so the decision boundaries will also be hyperplanes
- The constant probability loci are hyper-ellipses aligned with the eigenvectors of Σ

Case 3: $\Sigma_i = \Sigma$ (Σ non-diagonal)

- This discriminant is linear, so the decision boundaries will also be hyperplanes
 - The constant probability loci are hyper-ellipses aligned with the eigenvectors of Σ
 - If we can assume equal prior
- $$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)$$
- The classifier becomes a minimum (Mahalanobis) distance classifier

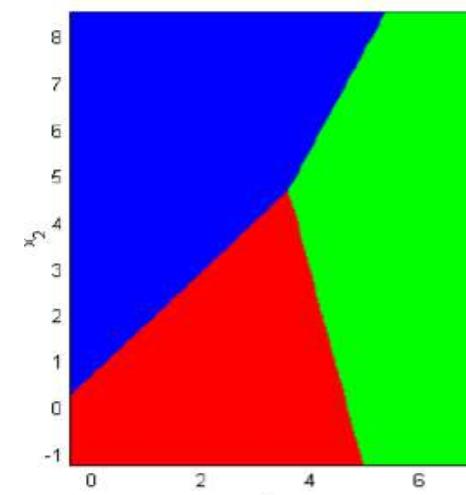
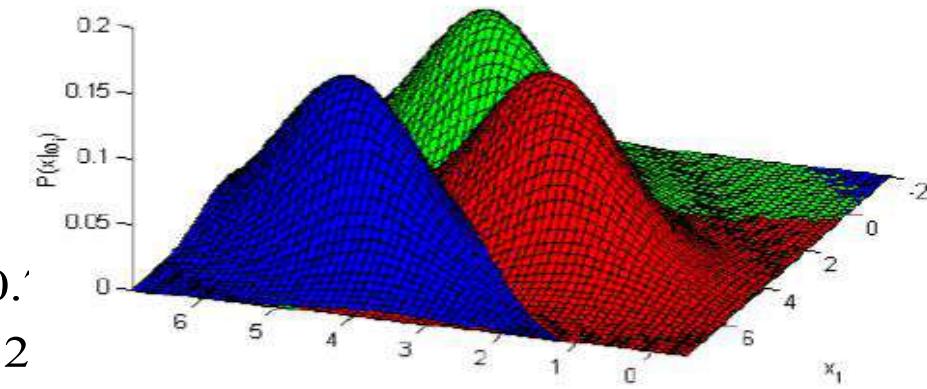
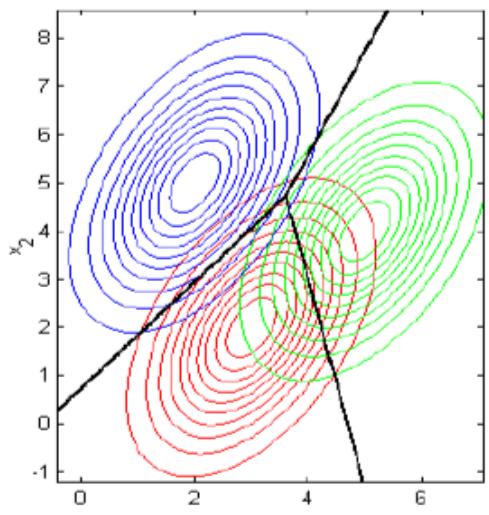


Case 3 : $\Sigma_i = \Sigma$ (Σ non-diagonal), example

- To illustrate the previous result, we will compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\boldsymbol{\mu}_1 = [3 \ 2]^T \quad \boldsymbol{\mu}_2 = [5 \ 4]^T \quad \boldsymbol{\mu}_3 = [2 \ 5]^T$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix}$$



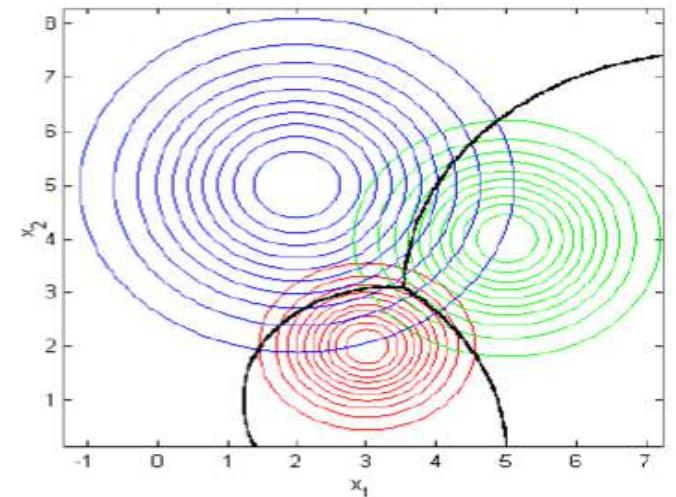
Case 4 : $\Sigma_i = \sigma_i^2 I$

- In this case, each class has a different covariance matrix, which is proportional to the identity matrix

- The quadratic discriminant becomes

$$\begin{aligned}g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i)) \\&= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\sigma}_i^{-2}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} N \log(\sigma_i^2) + \log(P(\omega_i))\end{aligned}$$

$$\begin{aligned}\boldsymbol{\mu}_1 &= [3 \quad 2]^T & \boldsymbol{\mu}_2 &= [5 \quad 4]^T & \boldsymbol{\mu}_3 &= [2 \quad 5]^T \\ \boldsymbol{\Sigma}_1 &= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} & \boldsymbol{\Sigma}_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \boldsymbol{\Sigma}_3 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\end{aligned}$$



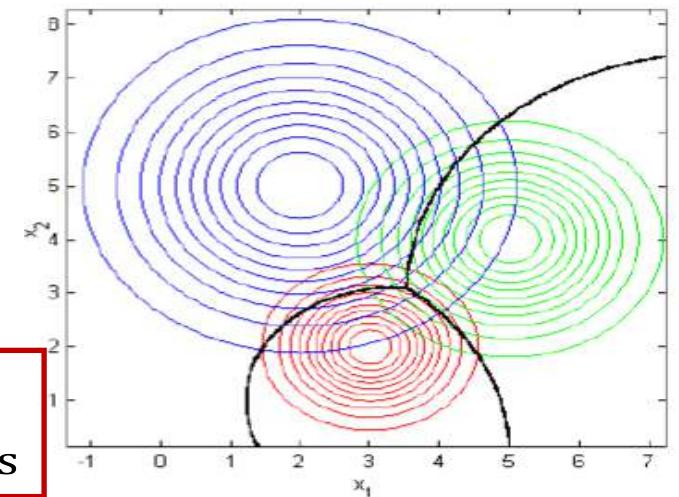
Case 4 : $\Sigma_i = \sigma_i^2 I$

- In this case, each class has a different covariance matrix, which is proportional to the identity matrix

- The quadratic discriminant becomes

$$\begin{aligned}g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i)) \\&= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\sigma}_i^{-2}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} N \log(\sigma_i^2) + \log(P(\omega_i))\end{aligned}$$

$$\begin{array}{lll}\boldsymbol{\mu}_1 = [3 \quad 2]^T & \boldsymbol{\mu}_2 = [5 \quad 4]^T & \boldsymbol{\mu}_3 = [2 \quad 5]^T \\ \boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} & \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \boldsymbol{\Sigma}_3 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\end{array}$$

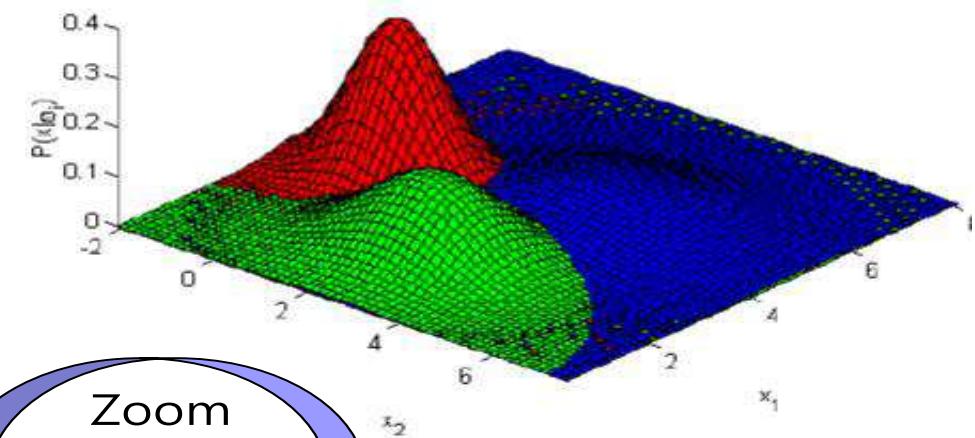
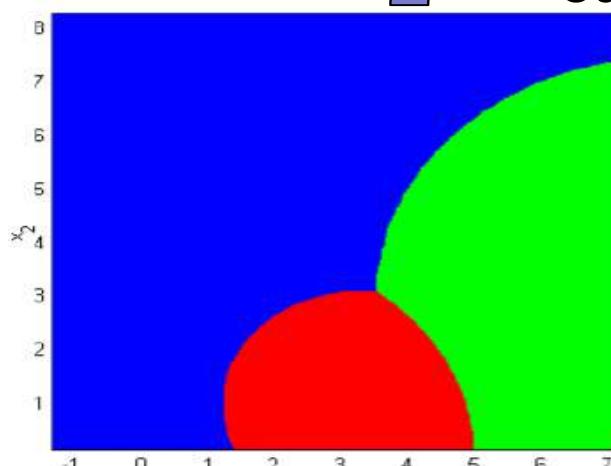
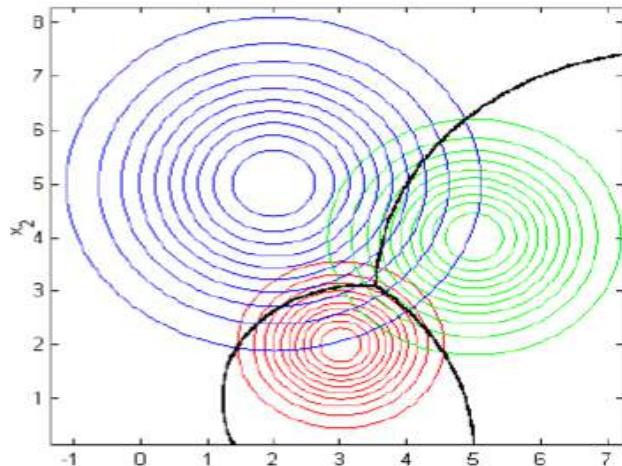


- This expression cannot be reduced further so
 - The decision boundaries are quadratic: hyper-ellipses
 - The loci of constant probability are hyper-spheres aligned with the feature axis

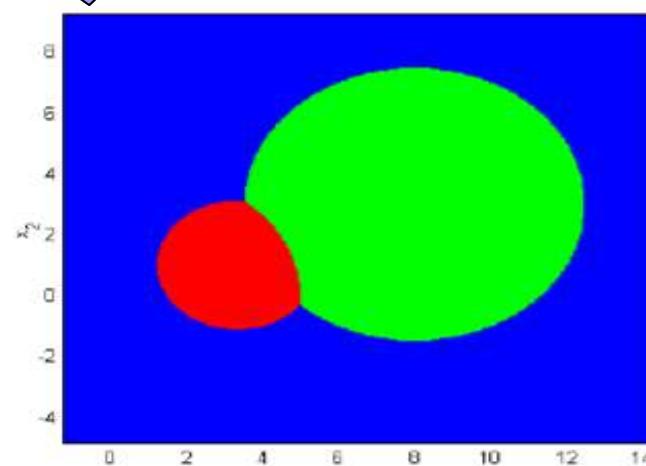
Case 4 : $\Sigma_i = \sigma_i^2 I$, Example

- To illustrate the previous result, we will compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\boldsymbol{\mu}_1 = [3 \ 2]^T \quad \boldsymbol{\mu}_2 = [5 \ 4]^T \quad \boldsymbol{\mu}_3 = [2 \ 5]^T$$
$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$



Zoom out



Case 5: $\Sigma_i \neq \Sigma_j$

- We already derived the expression for the general case at the beginning of this discussion

$$g_i(\mathbf{x}) = -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i))$$

Case 5: $\Sigma_i \neq \Sigma_j$

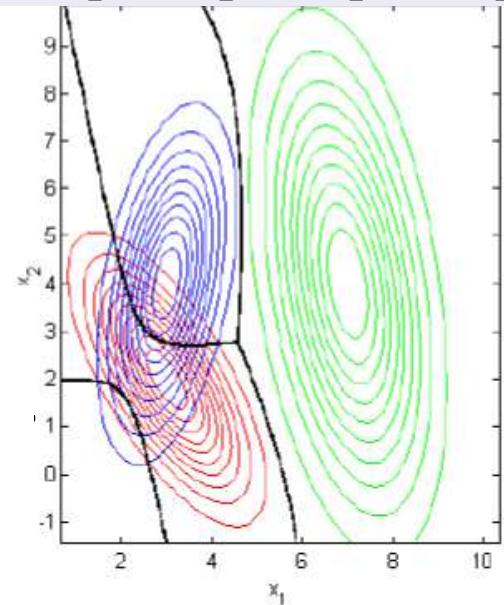
- Reorganizing terms in a quadratic form yields

$$g_i(\mathbf{x}) = \mathbf{x}^T \mathbf{w}_i \mathbf{x} + \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

$$\text{where } \begin{cases} \mathbf{W}_i = -\frac{1}{2} \boldsymbol{\Sigma}_i^{-1} \\ \mathbf{w}_i^T = \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i \\ w_{i0} = -\frac{1}{2} \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i - \frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i)) \end{cases}$$

$$\boldsymbol{\mu}_1 = [3 \quad 2]^T \quad \boldsymbol{\mu}_2 = [5 \quad 4]^T \quad \boldsymbol{\mu}_3 = [2 \quad 5]^T$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & -1 \\ -1 & 7 \end{bmatrix} \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 3 \end{bmatrix}$$



Case 5: $\Sigma_i \neq \Sigma_j$ General Case

- We already derived the expression for the general case at the beginning of this discussion

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2}\log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i))$$

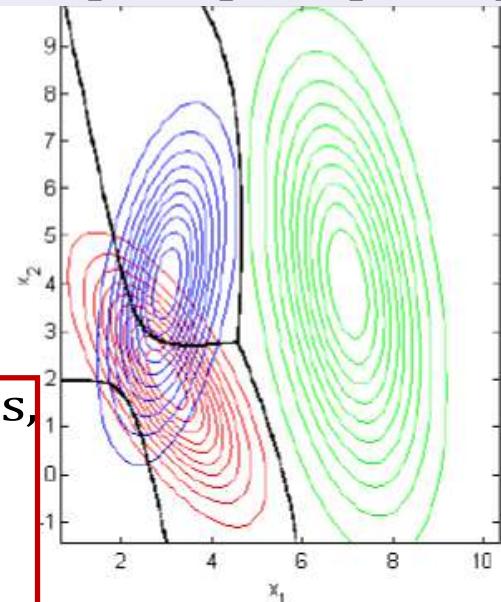
- Reorganizing terms in a quadratic form yields

$$g_i(\mathbf{x}) = \mathbf{x}^T \mathbf{w}_i \mathbf{x} + \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

where
$$\begin{cases} \mathbf{W}_i = -\frac{1}{2} \boldsymbol{\Sigma}_i^{-1} \\ \mathbf{w}_i^T = \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i \\ w_{i0} = -\frac{1}{2} \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i - \frac{1}{2}\log(|\boldsymbol{\Sigma}_i|) + \log(P(\omega_i)) \end{cases}$$

- The loci of constant probability for each class are hyper-ellipses, oriented with the eigenvectors of $\boldsymbol{\Sigma}_i$ for that class
- The decision boundaries are again quadratic : hyper-ellipses or hyper-paraboloids
- Notice that the quadratic expression in the discriminant is proportional to the Mahalanobis distance using the class-conditional covariance $\boldsymbol{\Sigma}_i$

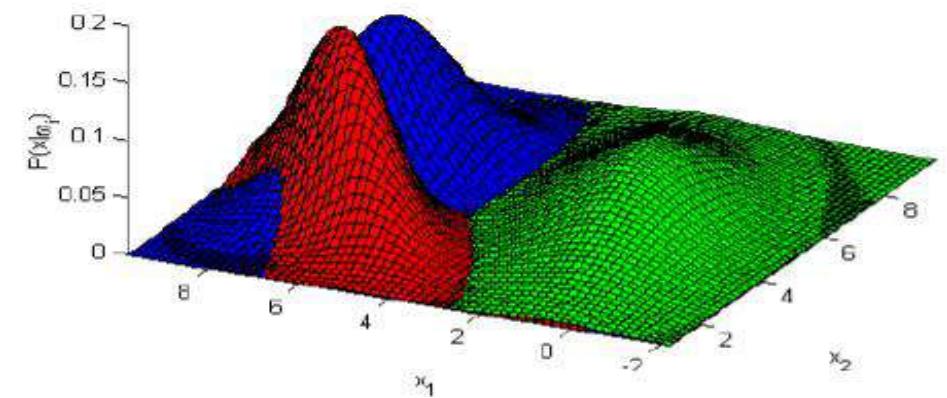
$$\begin{array}{lll} \boldsymbol{\mu}_1 = [3 \quad 2]^T & \boldsymbol{\mu}_2 = [5 \quad 4]^T & \boldsymbol{\mu}_3 = [2 \quad 5]^T \\ \boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} & \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & -1 \\ -1 & 7 \end{bmatrix} & \boldsymbol{\Sigma}_3 = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 3 \end{bmatrix} \end{array}$$



Case 5: $\Sigma_i \neq \Sigma_j$ General Case, Example

- To illustrate the previous result, we will compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\boldsymbol{\mu}_1 = [3 \quad 2]^T \quad \boldsymbol{\mu}_2 = [5 \quad 4]^T \quad \boldsymbol{\mu}_3 = [2 \quad 5]^T$$
$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & -1 \\ -1 & 7 \end{bmatrix} \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 3 \end{bmatrix}$$



Case 5: $\Sigma_i \neq \Sigma_j$ General Case, Example

- To illustrate the previous result, we will compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\boldsymbol{\mu}_1 = [3 \ 2]^T$$

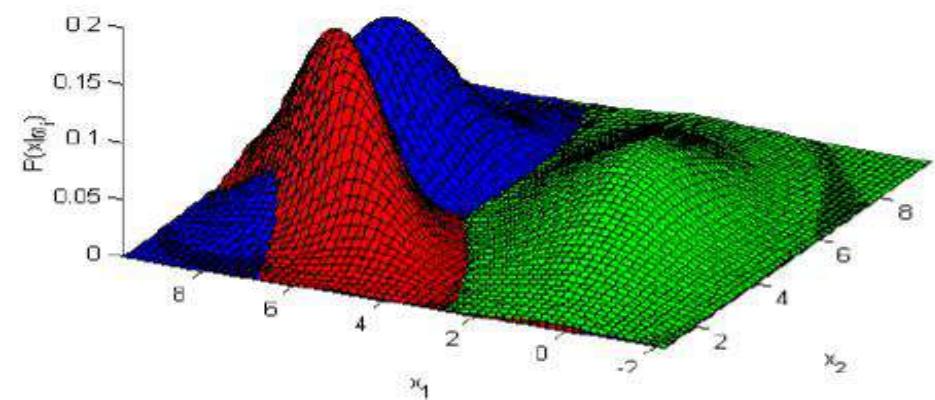
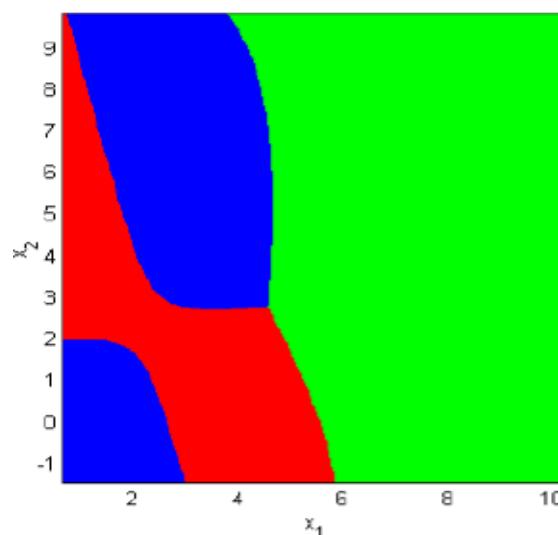
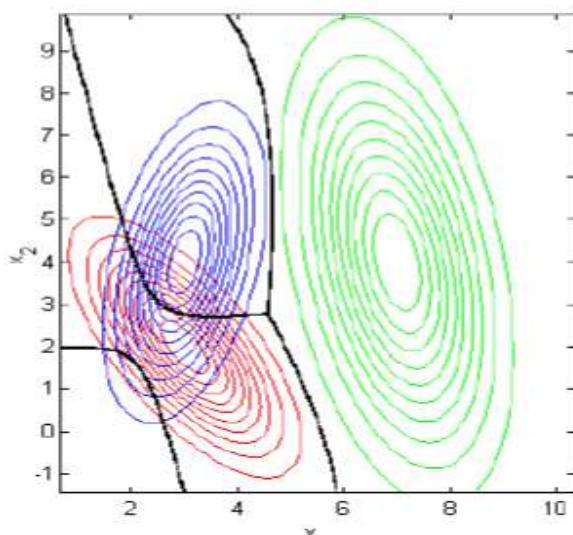
$$\boldsymbol{\mu}_2 = [5 \ 4]^T$$

$$\boldsymbol{\mu}_3 = [2 \ 5]^T$$

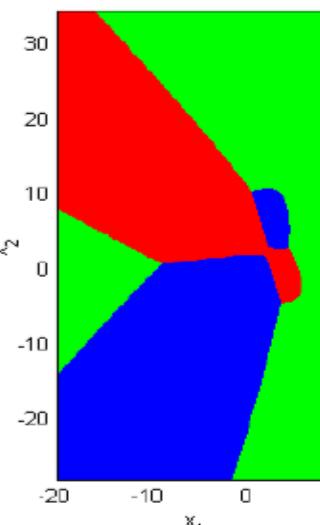
$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$$

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & -1 \\ -1 & 7 \end{bmatrix}$$

$$\boldsymbol{\Sigma}_3 = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 3 \end{bmatrix}$$



Zoom
out

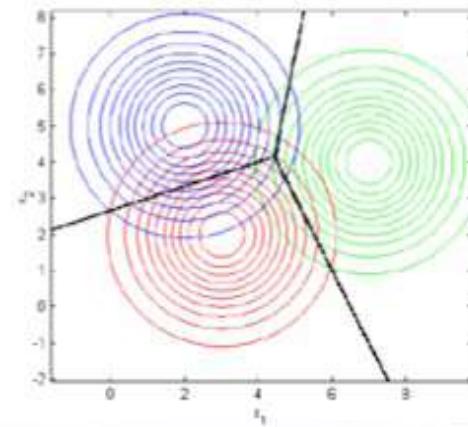


Numerical Example Case 1: $\Sigma_i = \sigma^2 I$

- Derive a linear discriminant function for a two class 3D classification problem defined by the following Gaussian Likelihoods

$$\mu_1 = [0 \ 0 \ 0]^T; \mu_2 = [1 \ 1 \ 1]^T; \Sigma_1 = \Sigma_2 = \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix}; P(\omega_2) = 2P(\omega_1)$$

Since $P(\omega_1) + P(\omega_2) = 1$, $P(\omega_1) + 2P(\omega_1) = 1$, thus $P(\omega_1) = \frac{1}{3}$, $P(\omega_2) = \frac{2}{3}$



- Solution
- $$g_i(x) = -\frac{1}{2\sigma^2}(x-\mu_i)^T(x-\mu_i) + \log(P(\omega_i)) = \frac{1}{2\left(\frac{1}{4}\right)} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \\ x_3 - \mu_3 \end{bmatrix}^T \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \\ x_3 - \mu_3 \end{bmatrix} + \log(P(\omega_i))$$

$$g_1(x) = -2 \begin{bmatrix} x_1 - 0 \\ x_2 - 0 \\ x_3 - 0 \end{bmatrix}^T \begin{bmatrix} x_1 - 0 \\ x_2 - 0 \\ x_3 - 0 \end{bmatrix} + \log\left(\frac{1}{3}\right)$$

$$g_2(x) = -2 \begin{bmatrix} x_1 - 1 \\ x_2 - 1 \\ x_3 - 1 \end{bmatrix}^T \begin{bmatrix} x_1 - 1 \\ x_2 - 1 \\ x_3 - 1 \end{bmatrix} + \log\left(\frac{2}{3}\right)$$

Numerical Example

$$g_1(x) = -2(x_1^2 + x_2^2 + x_3^2) + \log \frac{1}{3} \quad > \quad g_2(x) = -2((x_1 - 1)^2 + (x_2 - 1)^2 + (x_3 - 1)^2) + \log \frac{2}{3}$$

ω_1
 ω_2

$$x_1 + x_2 + x_3 \quad < \quad \frac{6 - \log 2}{4} = 1.32$$

ω_2
 ω_1

- Classify a test example: $x_u = [0.1 \ 0.7 \ 0.8]^\top$

$$0.1 + 0.7 + 0.8 = 1.6 \quad < \quad 1.32 \Rightarrow x_u \in \omega_2$$

ω_2
 ω_1

Can you predict if they will play?

- What is the probability of a couple playing tennis on a **sunny** and **cold** day given the past history of them playing at various conditions?

Outlook	Temperature	Play=Yes	Play=No
Sunny	High	0	2
Sunny	Mild	2	1
Sunny	Cold	2	0
Overcast	High	1	1
Overcast	Mild	2	0
Overcast	Cold	1	1
Rain	High	1	2
Rain	Mild	0	1
Rain	Cold	0	1
		9	9

- Note that there are two input features(x): Outlook and Temperature
- Outlook can have 3 values:
 - Sunny
 - Overcast
 - Rain
- Temperature can have 3 values:
 - High
 - Mild
 - Cold

Can you predict if they will play?

- Apply Bayesian classification

$$P(Play|Sunny, Cold) = \frac{p(Sunny, Cold | Play)P(Play)}{p(Sunny, Cold)}$$

Outlook	Temperature	Play=Yes	Play=No
Sunny	High	0	2
Sunny	Mild	2	1
Sunny	Cold	2	0
Overcast	High	1	1
Overcast	Mild	2	0
Overcast	Cold	1	1
Rain	High	1	2
Rain	Mild	0	1
Rain	Cold	0	1
		9	9

By MAP

Is $p(Sunny, Cold | Play)P(Play) > p(Sunny, Cold | No)P(Not Play)$?

$$(2/9) \times (1/2) > (0/9) \times ((1/2))$$

Thus they will play!

Note each of these are a unique combination of features

Can you predict if they will play?

More complicated past history of playing tennis

Play Tennis: Training examples					
Day	Outlook $x_1 = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} = \begin{bmatrix} \text{sunny} \\ \text{overcast} \\ \text{rain} \end{bmatrix}$	Temperature $x_2 = \begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} \text{Hot} \\ \text{Mild} \\ \text{Cool} \end{bmatrix}$	Humidity $x_3 = \begin{bmatrix} a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} \text{High} \\ \text{Normal} \end{bmatrix}$	Wind $x_4 = \begin{bmatrix} a_{41} \\ a_{42} \end{bmatrix} = \begin{bmatrix} \text{Strong} \\ \text{Weak} \end{bmatrix}$	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Cool	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

Can you predict if the person will play when:

- Outlook = Sunny
- Temperature = Cool
- Humidity = High
- Wind = Strong

Can you predict if they will play?

- There are now a total of 4 features
 - Outlook: 3 possible values
 - Temperature: 3 possible values
 - Humidity: 2 possible values
 - Wind: 2 possible values
- Total possible feature combinations
 - $3 \times 3 \times 2 \times 2 = 36$
- Number of data points: 14
- Insufficient data to fully cover the feature space!
- Apply Naïve Bayes

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	3/9	1/5
Mild	4/9	2/5
Cool	2/9	2/5

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

$$P(\text{Play}=\text{Yes}) = 9/14$$

Prior

$$P(\text{Play}=\text{No}) = 5/14$$

Naïve Bayes Classifier

- Naïve Bayes: a simple probabilistic classifier based on applying Bayes' rule with strong (naïve) independence assumptions between the features

$$P(C_k | x_1, x_2, \dots, x_d) = \frac{p(x_1, x_2, \dots, x_d | C_k) P(C_k)}{p(x_1, x_2, \dots, x_d)} \text{ for } 1 \leq k \leq K$$

- Applying independence assumption

$$p(x_1, x_2, \dots, x_d | C_k) P(C_k) = p(x_1 | C_k) p(x_2 | C_k) \dots p(x_d | C_k) P(C_k) \text{ for } 1 \leq k \leq K$$

- Then Bayes rule becomes

$$P(C_k | x_1, x_2, \dots, x_d) = \frac{p(x_1 | C_k) p(x_2 | C_k) \dots p(x_d | C_k) P(C_k)}{p(x_1, x_2, \dots, x_d)} \text{ for } 1 \leq k \leq K$$

Naïve Bayes Classifier Example

- Assuming priors are the same, let's apply ML

- Develop likelihoods for each feature

- $p(Outlook=Sunny|Play=Yes) = 2/9$
- $p(Temperature=Cool|Play=Yes) = 2/9$
- $p(Humidity=High|Play=Yes) = 3/9$
- $p(Wind=Strong|Play=Yes) = 3/9$

- $p(Outlook=Sunny|Play=No) = 3/5$
- $p(Temperature=Cool|Play=No) = 2/5$
- $p(Humidity=High|Play=No) = 4/5$
- $p(Wind=Strong|Play=No) = 3/5$

- Prior probabilities: $P(Play=Yes) = 9/14$, $P(Play=No) = 5/14$

- Assuming priors from the given data may not always be the best option if there are prior knowledges such as the couple has been observed on the average playing tennis in equal proportion

- Decision by the MAP rule with an assumption of equal prior

- $P(Yes|x) \sim p(Sunny|Yes) p(Cool|Yes) p(High|Yes) p(Strong|Yes) P(Yes) = (2/9)(2/9)(3/9)(3/9)(9/14) = 0.003527$
- $P(No|x) \sim p(Sunny|No) p(Cool|No) p(High|No) p(Strong|No) P(No) = (3/5)(2/5)(4/5)(3/5)(5/14) = 0.041142$

- Therefore, the prediction is **No**

Numerical Stability in Naïve Bayes Classifier

- Note the numbers calculated in the previous example
- $(2/9)(2/9)(3/9)(3/9)(9/14) = 0.003527$
- Naïve Bayes calculation involves multiplying numbers that are < 1 with other numbers < 1 many times over
- The example considered only involve 4 input features, but in general there are far more number of features included in the calculation
- Multiplying lots of probabilities → floating-point underflow.
- So how do you deal with calculating small numbers?
- In Naïve Bayes classification, we are only interested in comparing numbers rather than computing the exact values
- Recall: $\log(xy) = \log(x) + \log(y)$
- Take logs of probabilities
- Comparing log values of probabilities is still valid for classification.

Numerical Stability in Naïve Bayes Classifier

- Class with highest final un-normalized log probability score is still the most probable.
- Instead of comparing $P(\text{Yes} | X_1, \dots, X_n)$ with $P(\text{No} | X_1, \dots, X_n)$,
 - Compare their logarithms

$$\begin{aligned}\log(P(Y|X_1, \dots, X_n)) &= \log\left(\frac{P(X_1, \dots, X_n|Y) \cdot P(Y)}{P(X_1, \dots, X_n)}\right) \\ &= \text{constant} + \log\left(\prod_{i=1}^n P(X_i|Y)\right) + \log P(Y) \\ &= \text{constant} + \sum_{i=1}^n \log P(X_i|Y) + \log P(Y)\end{aligned}$$

Zero Frequency Problem in Naïve Bayes Classifier

- Another prediction:
 - Outlook = Overcast
 - Temperature = Cool
 - Humidity = High
 - Wind = Strong
- But there is no data on not playing when Outlook = Overcast!
- Solution:
 - Laplace Smoothing by adding a small value to the zero event category and add the same to each of the rest of categories

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	3/9	1/5
Mild	4/9	2/5
Cool	2/9	2/5

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

$$P(\text{Play}=\text{Yes}) = 9/14$$

$$P(\text{Play}=\text{No}) = 5/14$$

Zero Frequency Problem in Naïve Bayes Classifier

- Adding the values is limited to the data relevant to the Outlook feature
- Changes to the probabilities

$$P(\text{Outlook}=\text{Overcast} \mid \text{Play}=\text{Yes}) = 5/12$$

$$P(\text{Temperature}=\text{Cool} \mid \text{Play}=\text{Yes}) = 2/9$$

$$P(\text{Humidity}=\text{High} \mid \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Wind}=\text{Strong} \mid \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Play}=\text{Yes}) = 9/14$$

$$P(\text{Outlook}=\text{Overcast} \mid \text{Play}=\text{No}) = 1/8$$

$$P(\text{Temperature}=\text{Cool} \mid \text{Play}=\text{No}) = 2/5$$

$$P(\text{Humidity}=\text{High} \mid \text{Play}=\text{No}) = 4/5$$

$$P(\text{Wind}=\text{Strong} \mid \text{Play}=\text{No}) = 3/5$$

$$P(\text{Play}=\text{No}) = 5/14$$



Outlook	Play=Yes	Play=No	Temperature	Play=Yes	Play=No
Sunny	(2+1)/(9+3)	(3+1)/(5+3)	Hot	3/9	1/5
Overcast	(4+1)/(9+3)	(0+1)/(5+3)	Mild	4/9	2/5
Rain	(3+1)/(9+3)	(2+1)/(5+3)	Cool	2/9	2/5

Humidity	Play=Yes	Play=No	Wind	Play=Yes	Play=No
High	3/9	4/5	Strong	3/9	3/5
Normal	6/9	1/5	Weak	6/9	2/5

$$P(\text{Play}=\text{Yes}) = 9/14$$

$$P(\text{Play}=\text{No}) = 5/14$$

Zero Frequency Problem in Naïve Bayes Classifier

- Decision making with the MAP rule

$$P(\text{Yes} | \mathbf{x}') \approx [P(\text{Overcast} | \text{Yes})P(\text{Cool} | \text{Yes})P(\text{High} | \text{Yes})P(\text{Strong} | \text{Yes})]P(\text{Play} = \text{Yes}) = 0.0066$$

$$P(\text{No} | \mathbf{x}') \approx [P(\text{Overcast} | \text{No}) P(\text{Cool} | \text{No})P(\text{High} | \text{No})P(\text{Strong} | \text{No})]P(\text{Play} = \text{No}) = 0.0086$$

- Given that $P(\text{Yes} | \mathbf{x}') < P(\text{No} | \mathbf{x}')$, we label \mathbf{x}' to be “No”

Reviews of Linear Algebra

ECE 610, spring 2024
David Han
Drexel University

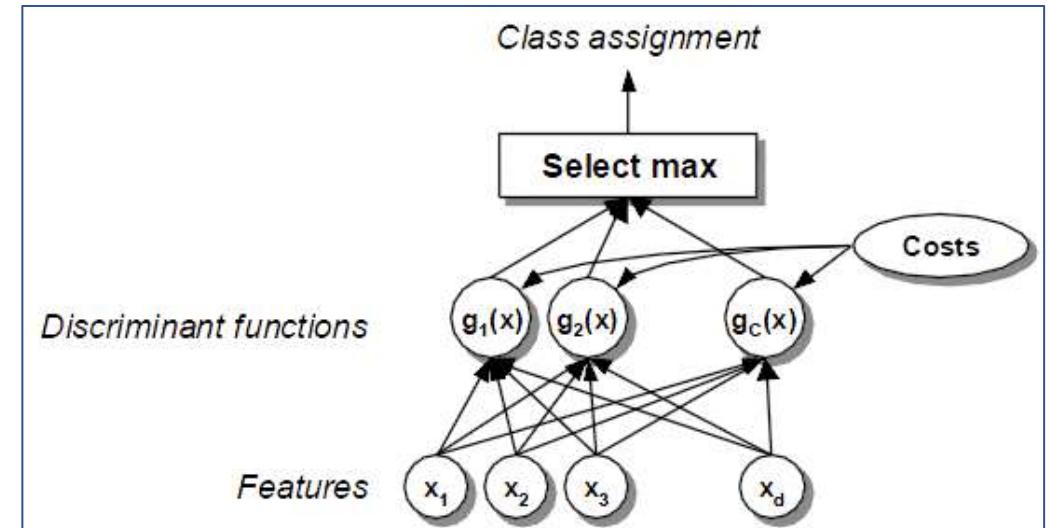
Outline

- Scalars, Vectors, Matrices, and Tensors
- Multiplying Matrices and Vectors
- Identity and Inverse Matrices
- Linear Dependence and Span
- Norms
- Special Kinds of Matrices and Vectors
- Eigendecomposition
- Principal Component Analysis

Linear Algebra Review

- Why Linear Algebra?
- Essential for understanding machine learning algorithms
- Input to ML algorithms, (x_1, x_2, \dots, x_n) are converted to outputs via a series of linear transformations
- Recall a linear discriminant function

$$g_i(x) = w_i^T x + w_{i0}$$



Linear Algebra Review

- Remember Naïve Bayes

$$P(C_k | x_1, x_2, \dots, x_d) = \frac{p(x_1 | C_k) p(x_2 | C_k) \dots p(x_d | C_k) P(C_k)}{p(x_1, x_2, \dots, x_d)} \quad \text{for } 1 \leq k \leq K$$

$$\begin{aligned} \log(P(Y|X_1, \dots, X_n)) &= \log \left(\frac{P(X_1, \dots, X_n|Y) \cdot P(Y)}{P(X_1, \dots, X_n)} \right) \\ &= \text{constant} + \log \left(\prod_{i=1}^n P(X_i|Y) \right) + \log P(Y) \\ &= \text{constant} + \sum_{i=1}^n \log P(X_i|Y) + \log P(Y) \end{aligned}$$

$$g_i(\mathbf{x}) = w_i^T \mathbf{x} + w_{i0}$$

Linear Algebra Review

- Scalars: a scalar is just a single number, used to define a magnitude
 - Ex. volume, density, speed, energy, mass, and time
- Vectors: \mathbf{x} vector is an array of numbers.
 - numbers are arranged in order.
 - First element of \mathbf{x} is x_1 , the second element is x_2 , etc.
 - d-dimensional vector: the vector has d elements, if real values → it resides in \mathbb{R}^d .

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \text{and} \quad \mathbf{x}^T = [x_1 \quad x_2 \quad \cdots \quad x_d]$$

- Vectors can be interpreted as identifying points in space, with each element giving the coordinate along a different axis.

Linear Algebra Review

- Matrices: a 2-D array of numbers
- For a real valued matrix A with a height of n and a width of d
 - $A \in \mathbb{R}^{n \times d}$
 - $A_{1,1}$ is the upper left entry of A and $A_{n,d}$ is the bottom right entry
 - An $n \times d$ (rectangular) matrix can be written as

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nd} \end{bmatrix}$$

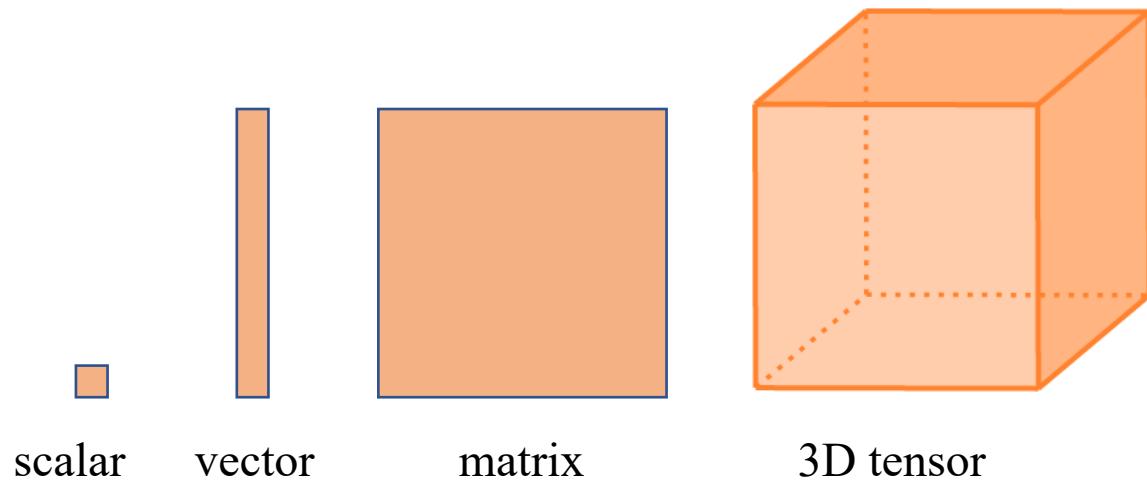
Linear Algebra Review

- Matrix-valued expressions can be defined by index.
- For example, $f(A)_{i,j}$ gives element (i, j) of the matrix computed by applying the function f to A .

Linear Algebra Review

- Tensor: An array of numbers arranged on a regular grid with variable number of axes.
- For a 3-D tensor A , the element of A at coordinates (i, j, k) can be written as $A_{i,j,k}$.

- Thus:
 - A scalar is a tensor with zero axis
 - A vector is a tensor with only one axis
 - A matrix is a tensor with 2 axes
 - A tensor can be in higher than 3D



- Matrix Transpose: mirror image of the matrix across its diagonal
 - Transpose of a matrix A written as A^T

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nd} \end{bmatrix} \quad \text{and} \quad A^T = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ a_{12} & \cdots & a_{n2} \\ \vdots & \ddots & \vdots \\ a_{1d} & \cdots & a_{nd} \end{bmatrix}$$

Linear Algebra Review

- Transpose of a vector is a matrix of only one row.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \text{ and } \mathbf{x}^T = [x_1 \quad x_2 \quad \cdots \quad x_d]$$

- Transpose of a scalar is the same: $a = a^T$
- Matrix addition: $\mathbf{C} = \mathbf{A} + \mathbf{B}$ where $C_{i,j} = A_{i,j} + B_{i,j}$
- Scalar addition and multiplication of a matrix
 - $\mathbf{D} = a \cdot \mathbf{B} + c$ where $D_{i,j} = a \cdot B_{i,j} + c$.
- Adding a vector to a matrix
 - $\mathbf{C} = \mathbf{A} + \mathbf{b}$, where $C_{i,j} = A_{i,j} + b_j$
 - called broadcasting

Linear Algebra Review

- Matrix multiplication

- $C = AB$, for this to work, number of columns in A must match number of rows in B
- If A is of shape $m \times d$ and B is of shape $d \times n$, then C is of shape $m \times n$.

$$AB = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{md} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ b_{21} & \cdots & b_{2n} \\ \vdots & \ddots & \vdots \\ b_{d1} & \cdots & b_{dn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}$$

- It can be also written out as

$$\text{where, } c_{ij} = \sum_{k=1}^d a_{ik} b_{kj}$$

Linear Algebra review

- Hadamard product: element-wise product of two matrices
 - Denoted as $A \odot B$
 - With $C = A \odot B$, $c_{i,j} = (a_{i,j}) (b_{i,j})$
- Matrix multiplication is distributive
 - $A(B+C) = AB + AC$
- Matrix multiplication is associative
 - $A(BC) = (AB)C$
- Matrix multiplication is Not commutative
 - $AB \neq BA$

Linear Algebra review

- Dot product between two vectors is commutative
 - $x^T y = y^T x$
- Additional identities

$$(A+B)^T = A^T + B^T$$

$$(AB)^T = B^T A^T$$

$$(AB)^{-1} = B^{-1}A^{-1} \text{ if individual inverses exist}$$

$$(A^{-1})^T = (A^T)^{-1}$$

Linear Algebra Review

- Identity and Inverse Matrices
- Identity matrix, I_n , preserves n -dimensional vectors as,
 - $I_n \in R^{n \times n}$ such that $\forall x \in R^n, I_n x = x$

$$I_n = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ . & 0 & \dots & 0 & . \\ . & 0 & \dots & 1 & . \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

- Matrix Inverse
 - $A^{-1}A = I$

Linear Algebra Review

- Consider a system of linear equations

$$A_{1,1}x_1 + A_{1,2}x_2 + \cdots + A_{1,n}x_n = b_1$$

$$A_{2,1}x_1 + A_{2,2}x_2 + \cdots + A_{2,n}x_n = b_2$$

...

$$A_{m,1}x_1 + A_{m,2}x_2 + \cdots + A_{m,n}x_n = b_m.$$

- This can be written in a matrix equation as $A \mathbf{x} = \mathbf{b}$
- where $A \in \mathbb{R}^{m \times n}$ is a known matrix, $\mathbf{b} \in \mathbb{R}^m$ is a known vector, and $\mathbf{x} \in \mathbb{R}^n$ is a vector of unknown variables to be solved
- If we know the inverse of A , we can solve the equation by premultiplying A^{-1} to the matrix equation as

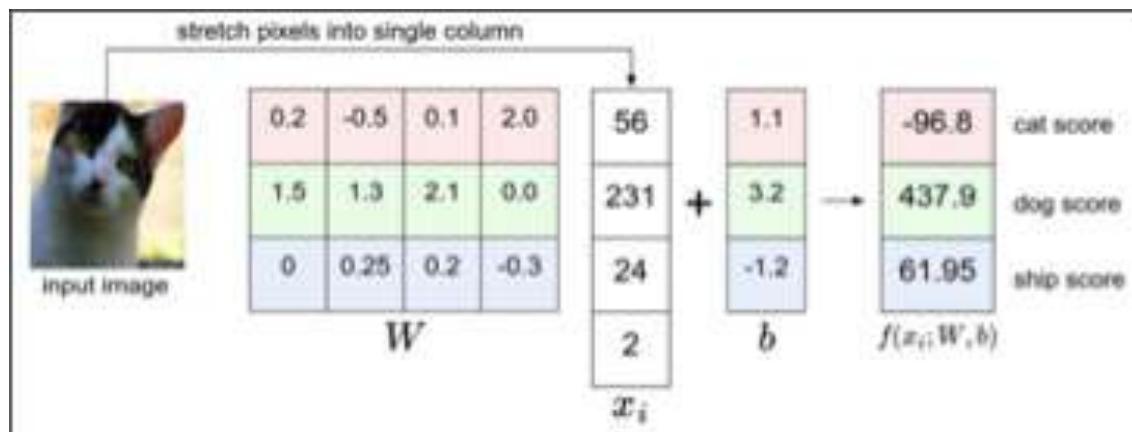
$$A^{-1}A \mathbf{x} = A^{-1}\mathbf{b} \rightarrow I \mathbf{x} = A^{-1}\mathbf{b} \rightarrow \mathbf{x} = A^{-1}\mathbf{b}$$

Linear Algebra Review

- The following are equivalent:
 - A is invertible, i. e. A^{-1} exists
 - $Ax = b$ has a unique solution
 - Columns of A are linearly independent
 - $\det(A) \neq 0$
 - $Ax = 0$ has a unique, trivial solution: $x = 0$

Machine learning application

- A linear Classifier
- An input image: picture
- Output y predicts the class category: cat, dog, ship.
- Input converted to vector x



- $y = Wx^T + b$

Linear Algebra Review

- Norm: used to measure magnitude of a vector
 - L^p norm defined as $\|x\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$
 - which measures the distance between vector x from the origin
- Norm, $f(x)$, satisfies the following properties:
 - $f(x) = 0$ if $x = \mathbf{0}$
 - $f(x+y) \leq f(x) + f(y)$ from triangular inequality
 - $\forall \alpha \in R, f(\alpha x) = |\alpha| f(x)$

Linear Algebra Review

- L^1 norm also known as Manhattan distance

$$\|x\|_1 = \sum_i |x_i|.$$

- L^2 norm also called Euclidean norm

- L^2 of $x = (x^T x)^{1/2}$

- Square norm of $L^2 = x^T x$

$$\|x\|_2 = 1.$$

- Unit vector is a vector with unit L^2 norm:
- L^2 inefficient when x near origin as it increases slowly
 - L^1 norm more effective in this case.
- L^∞ Norm: max norm defined as

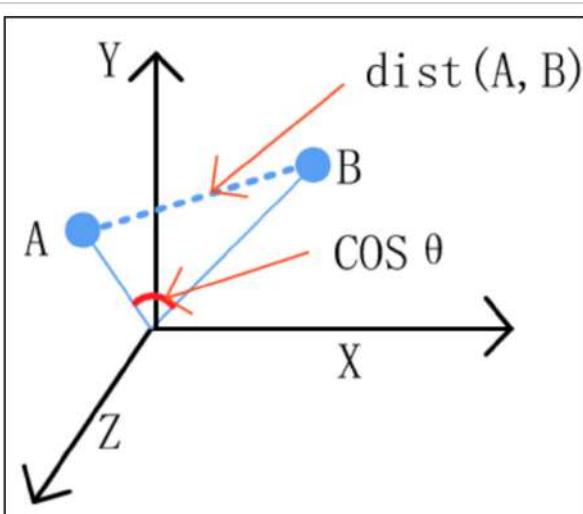
$$\|x\|_\infty = \max_i |x_i|$$

Linear Algebra Review

- Frobenius Norm: size of a matrix
- Similar to L^2 norm for vectors
- Defined as $\|A\|_F = \left(\sum_{i,j} |A_{i,j}|^2 \right)^{1/2}$
- Measure the size of matrix in terms of Euclidean distance
- Sum of magnitude of all columns (vectors) in the matrix
- May be used for regularization
 - Minimize Frobenius norm of weight matrices over layers

Linear Algebra Review

- Symmetric Matrix:
 - $A = A^T$
- Orthogonal Matrix
 - $A^T A = A^T A = I$ and $A^{-1} = A^T$
- Angle Between Vectors
 - Dot product of two vectors can be written in terms of L^2 norms and the angle θ between them
 - $a^T b = \|a\|_2 \|b\|_2 \cos(\theta)$

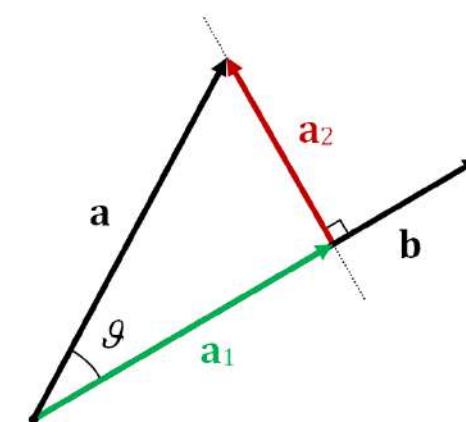


Linear Algebra Review

- Cosine Similarity
 - Cosine between two vectors is a measure of their similarity:
 - $\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$
- Orthogonal Vectors: two vectors \mathbf{a} and \mathbf{b} are orthogonal to each other if $\mathbf{a} \cdot \mathbf{b} = 0$.
- Vector Projection
 - Given two vectors \mathbf{a} and \mathbf{b} , let $\hat{\mathbf{b}} = \frac{\mathbf{b}}{\|\mathbf{b}\|}$ be the unit vector in the direction of \mathbf{b}

- Then $a_1 = \mathbf{a} \cdot \hat{\mathbf{b}}$ is the projection of \mathbf{a} onto a straight line parallel to \mathbf{b} , where

$$a_1 = \|\mathbf{a}\| \cos(\theta) = \mathbf{a} \cdot \hat{\mathbf{b}} = \mathbf{a} \cdot \frac{\mathbf{b}}{\|\mathbf{b}\|}$$



Linear Algebra Review

- Determinants
- Determinant of a square matrix is a mapping to a scalar
 - $\det(A)$ or $|A|$
- Measures how much multiplicaton by the matrix expands or contracts the space
- Determinant of product is the product of determinants:
 - $\det(AB) = \det(A)\det(B)$

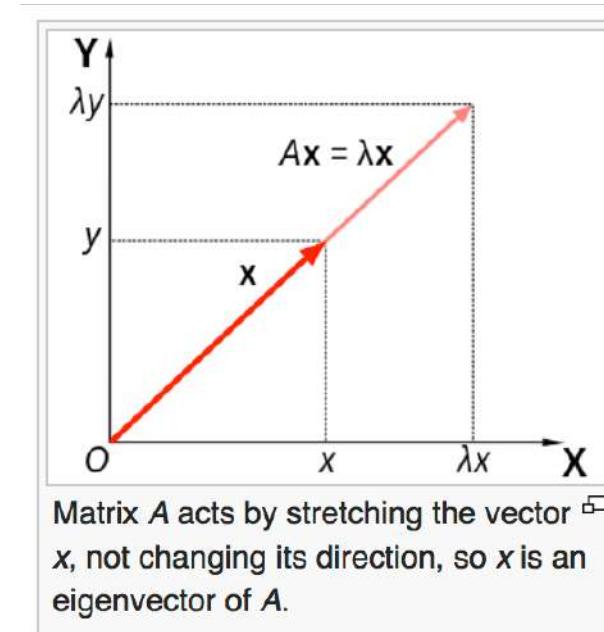
$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Linear Algebra Review

- Matrix Decomposition
- Matrices can be decomposed into factors, as in integers
 - Ex. $12 = 2 \times 2 \times 3$
- Decomposition of Matrix A as
 - $A = V \text{diag}(\lambda) V^{-1}$
 - Where V is composed of eigenvectors and λ are eigenvalues

Linear Algebra Review

- What are eigenvectors and eigenvalues?
- For a square matrix A , we can find a vector v such that the following is true.
 - $A v = \lambda v$ where λ is a scalar called eigenvalue



Linear Algebra Review

- If v is an eigenvalue of A , so is any rescaled vector sv . Additionally, sv still has the same eigenvalue.
- For more convenience, normalize eigenvector so that you have a unit eigenvector.

$$\hat{x} = \frac{x}{\|x\|_2}$$

- We can solve for eigenvectors and eigenvalues as follows:
 - $A v - \lambda v = 0$
 - $(A - \lambda)v = 0$
 - For the equation to have non-trivial solution,
 $\det(A - \lambda) = 0$

Linear Algebra Review

- Example eigenvector calculation

$$A = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

$$Av = \lambda v$$

$$(A - \lambda)v = 0$$

For non-trivial solution, $\det(A - \lambda) = 0$

$$\begin{vmatrix} 4-\lambda & 2 \\ 2 & 4-\lambda \end{vmatrix} = 0$$

$$(4-\lambda)^2 - 4 = 0$$

$$\lambda^2 - 8\lambda + 12 = 0, \quad \lambda=6 \text{ and } \lambda=2$$

for $\lambda=6$

$$\begin{bmatrix} 4-6 & 2 \\ 2 & 4-6 \end{bmatrix} \begin{bmatrix} v_1^{(1)} \\ v_2^{(1)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$-2v_1^{(1)} + 2v_2^{(1)} = 0, \quad v_1^{(1)} = v_2^{(1)}$$

$$v^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \text{ and similarly } v^{(2)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Linear Algebra Review

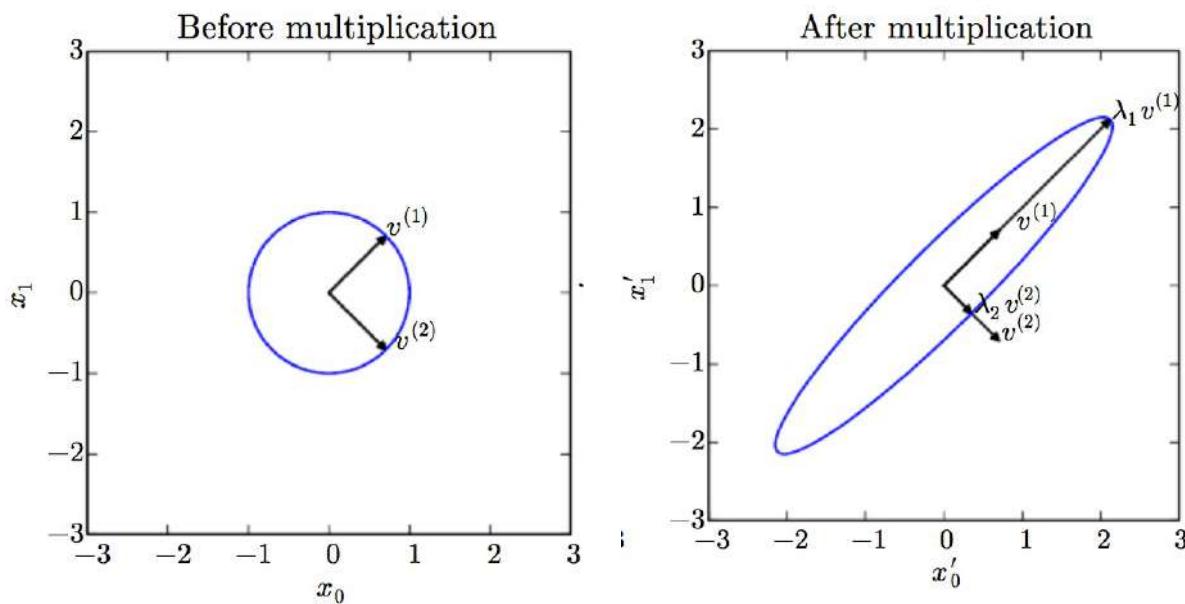
- Eigen Decomposition of Matrix
- Every **real symmetric** matrix A can be decomposed into real-valued eigenvectors and eigenvalues
- $A = V \Lambda V^T$
- where V is an orthogonal matrix composed of unit eigenvectors of A

$$v^{(1)} = \begin{bmatrix} v_1^{(1)} \\ v_2^{(1)} \\ v_3^{(1)} \end{bmatrix} \quad v^{(1)T} = [v_1^{(1)} \quad v_2^{(1)} \quad v_3^{(1)}]$$

- with Λ being a diagonal matrix of eigenvalues $\{\lambda_1, \dots, \lambda_n\}$
- Thus, $v^{(1)T} v^{(1)} = 1$ while $v^{(i)T} v^{(j)} = 0$ when $i \neq j$ as eigenvectors are orthogonal

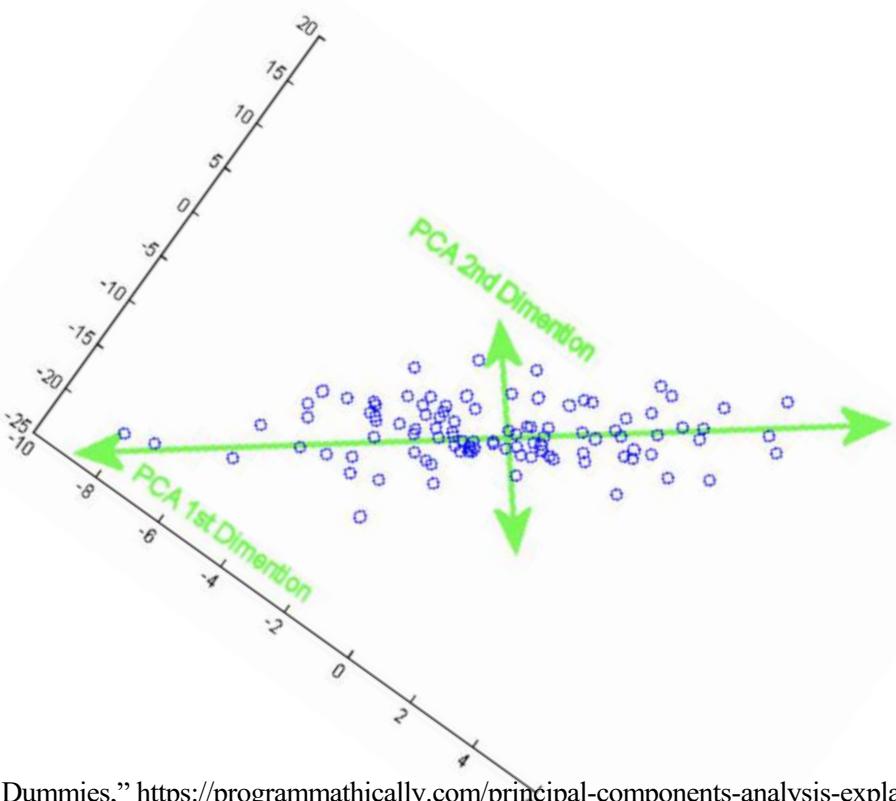
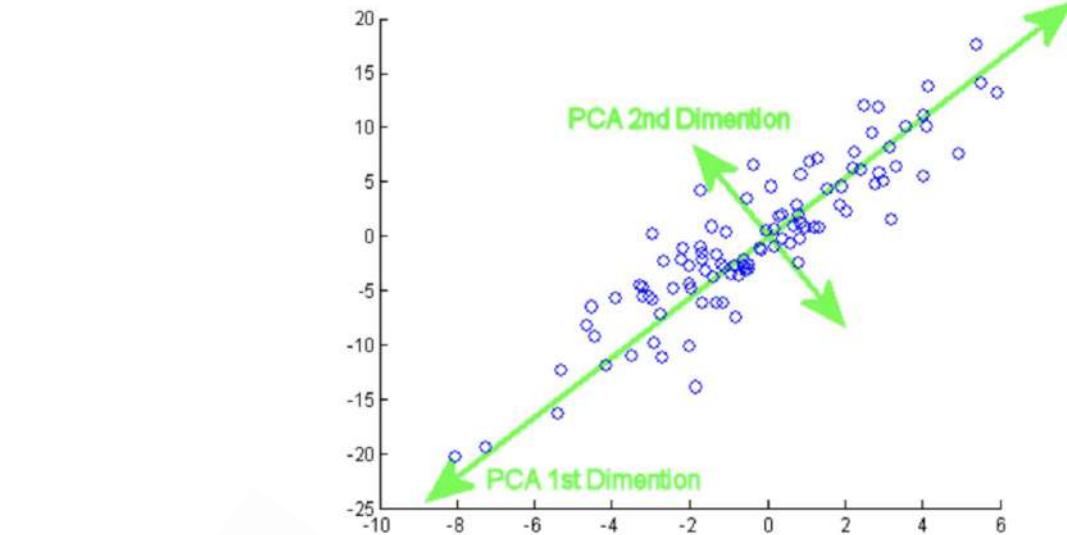
Linear Algebra Review

- Recall $A \mathbf{v} = \lambda \mathbf{v}$
- For an eigenvector, $\mathbf{v}^{(i)}$, matrix A scales the eigenvector by λ_i in the direction of $\mathbf{v}^{(i)}$
- An example of 2x2 matrix
- Two orthonormal eigenvectors $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$



Linear Algebra Review

- Principal Component Analysis (PCA)
- A method of reducing dimensionality of a dataset
- Process
 - Compute covariance matrix \mathbf{A} of the dataset
 - Compute eigenvalues of \mathbf{A}
 - Compute associated eigenvectors $v^{(i)}$
 - Sort eigenvalues and eigenvectors in order of sizes of eigenvalues
 - Determine the reduction of data dimensions
 - Form a matrix composed of eigenvectors according to the new dimensions
 - Transform the data into a new feature space with reduced dimensions



Linear Algebra Review

- Get the data

Student ID	Math	English	Arts
1	90	60	90
2	90	90	30
3	60	60	60
4	60	60	90
5	30	30	30

- Compute covariance matrix \mathbf{A} of the dataset

$$\begin{bmatrix} 504 & 360 & 180 \\ 360 & 360 & 0 \\ 180 & 0 & 720 \end{bmatrix}$$

- Compute eigenvalues of \mathbf{A}

$$\lambda \approx 44.81966 \dots \quad \lambda \approx 629.11039 \dots \quad \lambda \approx 910.06995 \dots$$

Linear Algebra Review

- Compute associated eigenvectors $v^{(i)}$

$$v = \begin{bmatrix} -3.75100 \dots \\ 4.28441 \dots \\ 1 \end{bmatrix} \quad v = \begin{bmatrix} -0.50494 \dots \\ -0.67548 \dots \\ 1 \end{bmatrix} \quad v = \begin{bmatrix} 1.05594 \dots \\ 0.69108 \dots \\ 1 \end{bmatrix}$$

- Sort eigenvalues and eigenvectors in descending order of eigenvalues

$$\begin{bmatrix} 910.06995 \dots \\ 629.11039 \dots \\ \textcolor{red}{-44.81966 \dots} \end{bmatrix}$$

- Determine the reduction of data dimensions: keep 2 D
- Form a matrix V composed of eigenvectors according to the new dimensions

$$V = \begin{bmatrix} 1.05594 \dots & -0.50494 \dots \\ 0.69108 \dots & -0.67548 \dots \\ 1 & 1 \end{bmatrix}$$

Linear Algebra Review

- Transform the data into a new feature space with reduced dimensions
- 1st data point

$$\bullet \quad x^{(1)} = \begin{bmatrix} 90 \\ 60 \\ 90 \end{bmatrix}$$

- Transform the data

$$y = V^T x^{(1)} = \begin{bmatrix} 1.05594 & -0.69108 & 1 \\ -0.50494 & -0.675481 & 1 \end{bmatrix} \begin{bmatrix} 90 \\ 60 \\ 90 \end{bmatrix}$$

Learning Loss and Gradient Descent

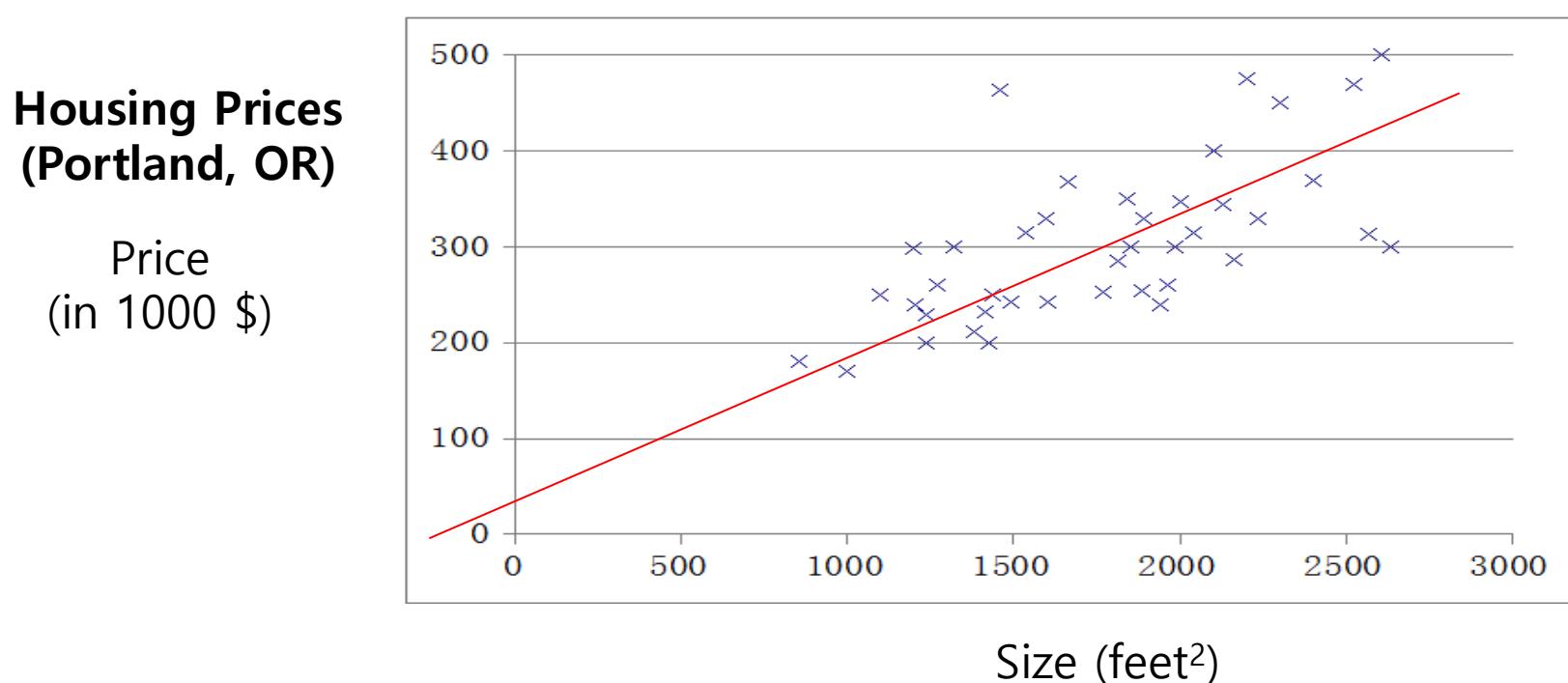
ECE 610

David Han

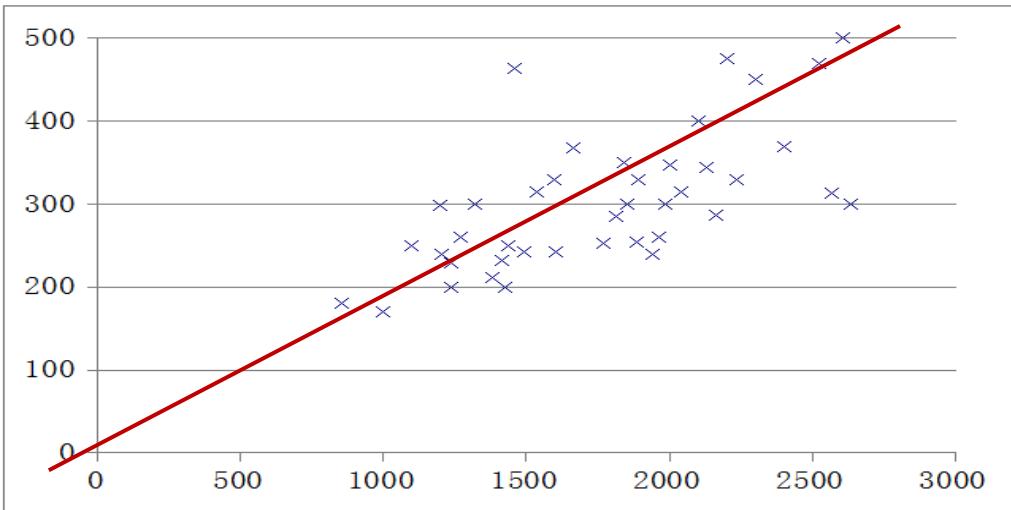
Drexel University

Linear Regression

- Simplest machine learning problem: Linear Regression
- Fit a **straight line** for a bunch of data points
 - Example: Housing Prices = *constant + slope × House Square footage*
- Housing Price is a linear function of the **input feature square footage**



Linear Regression: a machine learning problem



Training set of housing prices

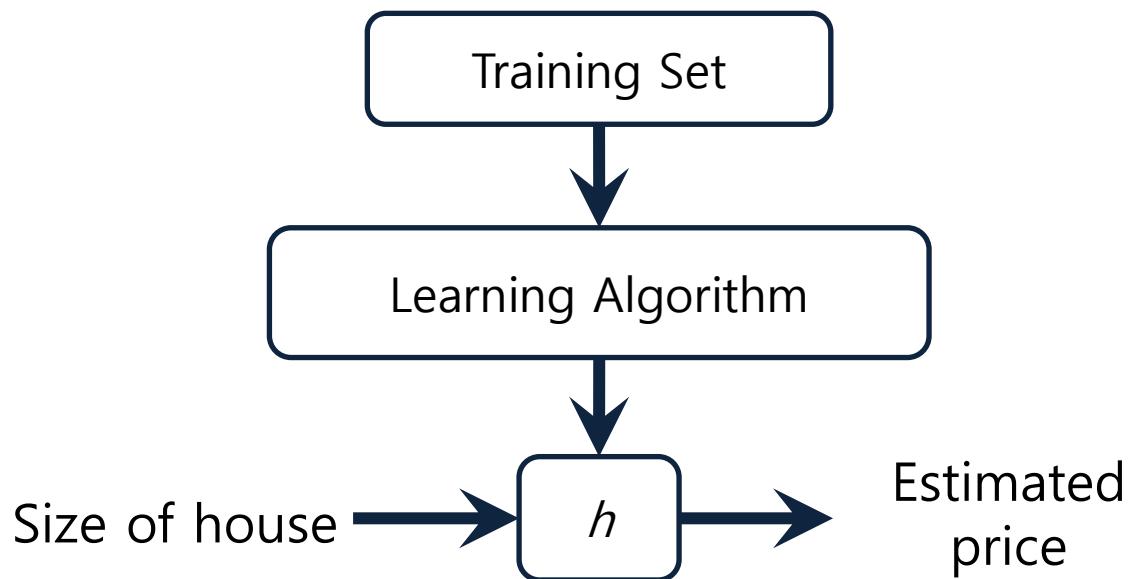
Size in feet ² (x)	Price (\$ in 1000's) (y)
2104	460
1416	232
1534	315
852	178
...	...

Notation:

m = Number of training examples

x 's = "input" variable / features

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



Question : How do we describe h ?

Linear Regression

- Housing Price is a linear function of the **input feature** *square footage*

$$\text{Housing Prices} = \theta_0 + \theta_1 \times \text{House Square footage}$$

- Model Parameters to be found: θ_0, θ_1 for 1-D case
- **More generally** for more than one input feature: x_n where $n > 1$

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

- where
 - \hat{y} is predicted value of the linear model
 - n is the number of features
 - x_i is the i^{th} feature value
 - θ_j is the j^{th} model parameter with θ_0 as the bias term and $\theta_1, \theta_2, \dots, \theta_n$ as the feature weights

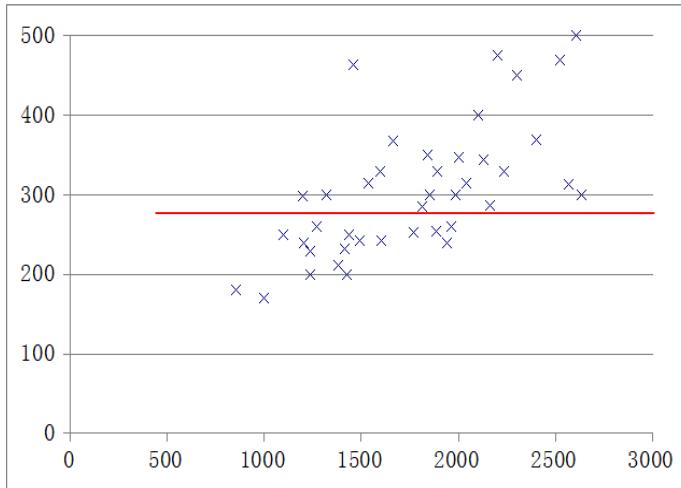
Linear Regression

- Or $\hat{y} = h_{\theta}(X) = \theta \cdot X$
- Where h_{θ} is the **hypothesis function** with model parameters θ
- With $\theta = [\theta_0, \theta_1, \theta_2, \dots, \theta_n]$ and $X = [x_0, x_1, x_2, \dots, x_n]$ and $x_0=1$.

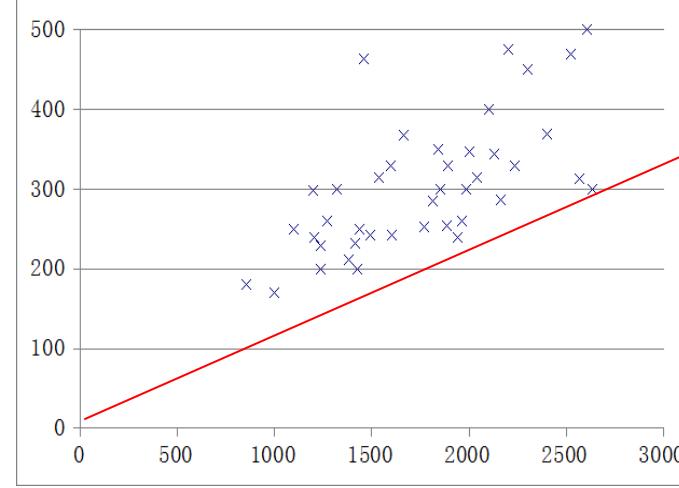
$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Linear Regression

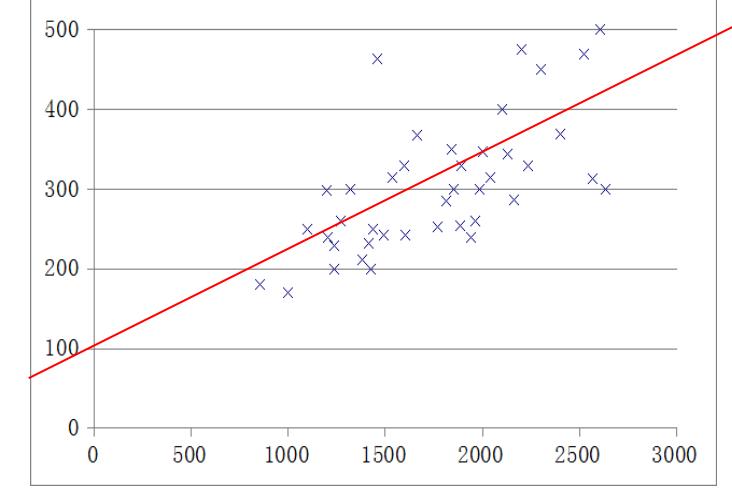
- For 1D feature case $h_{\theta}(x) = \theta_0 + \theta_1 x$
- How do you determine the model parameters: θ_0, θ_1 ?
- How do you make the machine find them for us?



$$\begin{aligned}\theta_0 &= 290 \\ \theta_1 &= 0\end{aligned}$$



$$\begin{aligned}\theta_0 &= 0 \\ \theta_1 &= 0.2\end{aligned}$$



$$\begin{aligned}\theta_0 &= 100 \\ \theta_1 &= 0.2\end{aligned}$$

Linear Regression

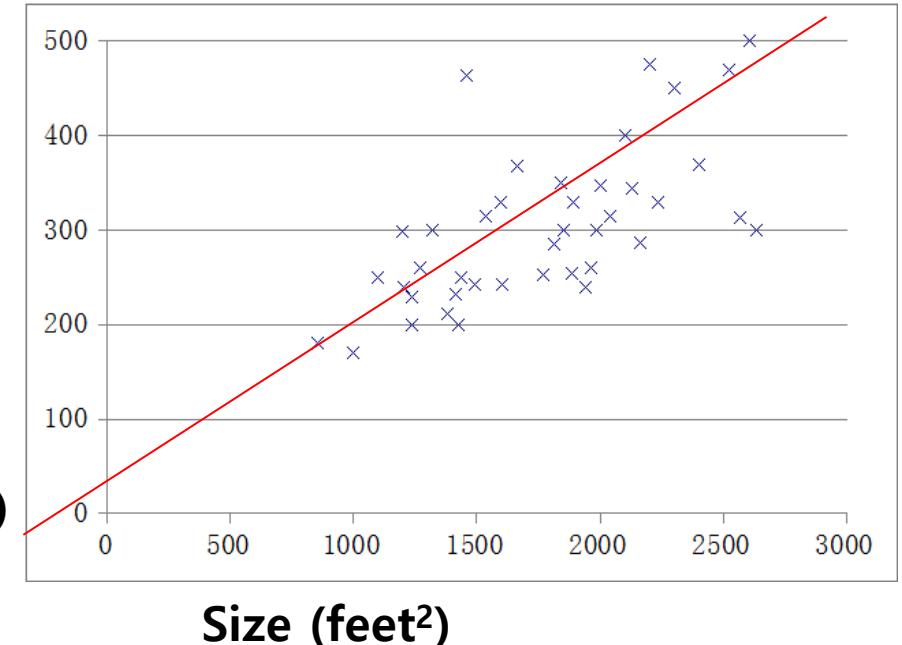
- Find the parameters that minimizes an error.
- (This is what most of ML is all about!)
- Define errors or loss function

For the i th data point $(\mathbf{X}^{(i)}, \mathbf{y}^{(i)})$

$$\text{Error} = h_{\theta}(\mathbf{X}^{(i)}) - \mathbf{y}^{(i)}$$

$$= \hat{\mathbf{y}}(\mathbf{X}^{(i)}) - \mathbf{y}^{(i)} = \theta \cdot \mathbf{X}^{(i)} - \mathbf{y}^{(i)}$$

Housing
Prices
(Portland, OR)



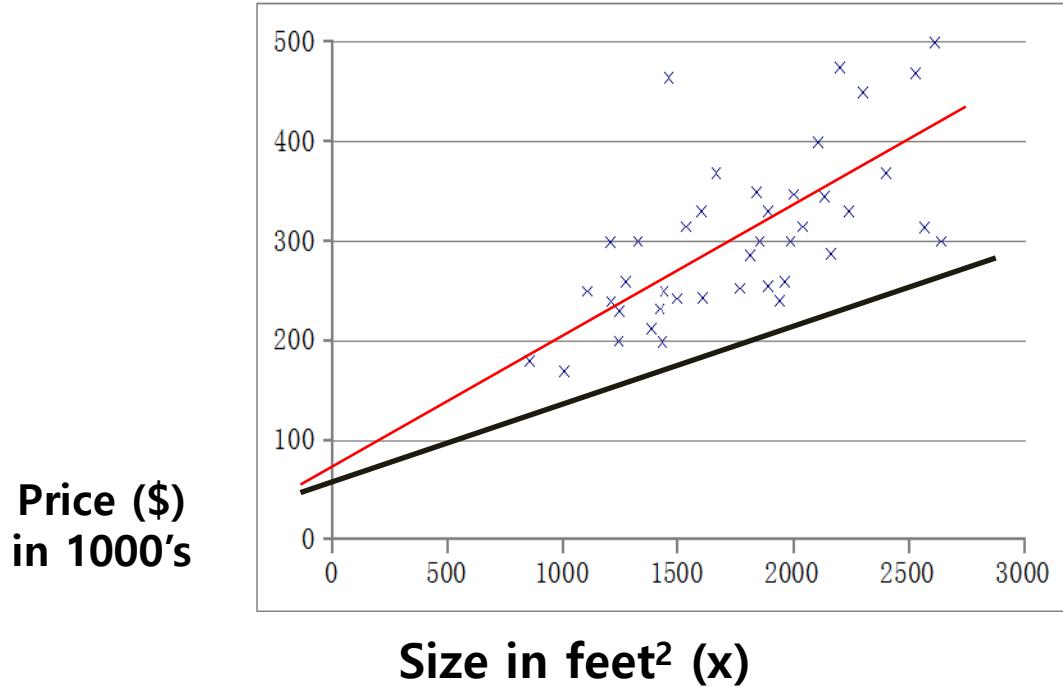
- Many different terms for the same meaning in ML:
 - **loss** function, **cost** function, **objective** function, etc.
- ML performance: how well the loss function is defined
- Loss function notation includes $(h_{\theta}(\mathbf{X}^{(i)}), \mathbf{y}^{(i)})$

Sum of square errors = $\sum_{i=1}^m (\theta \cdot X^{(i)} - y^{(i)})^2$, where m is the total number of data points

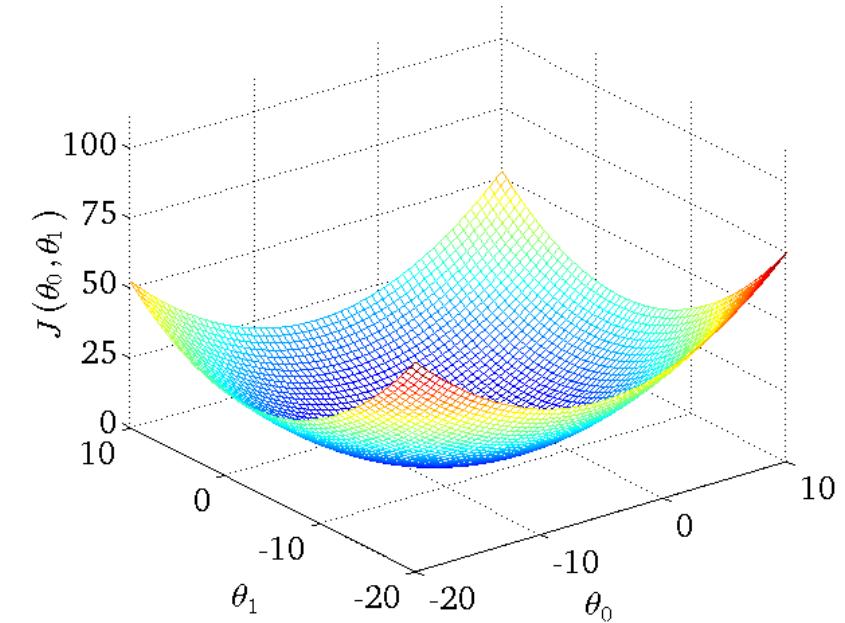
Some Distance Measures used as loss functions

- Root Mean Square Error (RMSE) = $\sqrt{\frac{1}{m} \sum_{i=1}^m (\theta \cdot X^{(i)} - y^{(i)})^2}$
- Manhattan Distance: L_1 norm = $\sum_{i=1}^m |\theta \cdot \mathbf{X}^{(i)} - \mathbf{y}^{(i)}|$
- L_n norm = $\left(\sum_{i=1}^m (\theta \cdot X^{(i)} - y^{(i)})^n \right)^{1/n}$

Linear Regression



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



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

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

- By minimizing the error, the best fitting line can be found

The Normal Equation

- How do you **find θ** that fits all the data points with the **least amount of total error or RMSE?**

$$\text{Minimum of } \text{MSE}(\theta) = \arg \min_{\theta} \frac{1}{m} \sum_{i=1}^m (\theta \cdot \mathbf{X}^{(i)} - \mathbf{y}^{(i)})^2$$

For m data points with n dimensional features, error between the prediction vector $\theta \cdot \mathbf{X}^{(i)}$ and data \mathbf{y} can be expressed in matrices as

$$\begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \dots & \vdots \\ x_0^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} = X\theta - Y$$

Note $i \rightarrow \text{data index}$

Note $m \neq n$

with $x_0 = 1$ to be multiplied to θ_0

The Normal Equation

- A **closed form** solution to the linear regression exists. Let $\text{MSE}(\boldsymbol{\theta}) = J(\boldsymbol{\theta})$
- sum of the square error = $J(\boldsymbol{\theta}) = (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{Y})^T (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{Y})$
Rewrite $J(\boldsymbol{\theta}) = ((\boldsymbol{X}\boldsymbol{\theta})^T - \boldsymbol{Y}^T)(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{Y}) = (\boldsymbol{X}\boldsymbol{\theta})^T \boldsymbol{X}\boldsymbol{\theta} - (\boldsymbol{X}\boldsymbol{\theta})^T \boldsymbol{Y} - \boldsymbol{Y}^T (\boldsymbol{X}\boldsymbol{\theta}) + \boldsymbol{Y}^T \boldsymbol{Y}$

basic formulae

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$$

$$(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$$

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$$

$$\text{if individual inverses exist } (\mathbf{AB})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$$

$$(\mathbf{A}^{-1})^T = (\mathbf{A}^T)^{-1}$$

The Normal Equation

- A **closed form** solution to the linear regression exists. Let $\text{MSE}(\boldsymbol{\theta}) = J(\boldsymbol{\theta})$
 - sum of the square error = $J(\boldsymbol{\theta}) = (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{Y})^T (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{Y})$
Rewrite $J(\boldsymbol{\theta}) = ((\boldsymbol{X}\boldsymbol{\theta})^T - \boldsymbol{Y}^T)(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{Y}) = (\boldsymbol{X}\boldsymbol{\theta})^T \boldsymbol{X}\boldsymbol{\theta} - (\boldsymbol{X}\boldsymbol{\theta})^T \boldsymbol{Y} - \boldsymbol{Y}^T (\boldsymbol{X}\boldsymbol{\theta}) + \boldsymbol{Y}^T \boldsymbol{Y}$
Note $(\boldsymbol{X}\boldsymbol{\theta})^T = \boldsymbol{\theta}^T \boldsymbol{X}^T$ and since $\boldsymbol{X}\boldsymbol{\theta}$ and \boldsymbol{Y} are vectors $(\boldsymbol{X}\boldsymbol{\theta})^T \boldsymbol{Y} = \boldsymbol{Y}^T (\boldsymbol{X}\boldsymbol{\theta})$
- $$J(\boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{X}^T \boldsymbol{X}\boldsymbol{\theta} - 2(\boldsymbol{X}\boldsymbol{\theta})^T \boldsymbol{Y} + \boldsymbol{Y}^T \boldsymbol{Y}$$

basic formulae

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$$

$$(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$$

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$$

$$\text{if individual inverses exist } (\mathbf{AB})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$$

$$(\mathbf{A}^{-1})^T = (\mathbf{A}^T)^{-1}$$

The Normal Equation

- A **closed form** solution to the linear regression exists. Let $\text{MSE}(\theta) = J(\theta)$

- sum of the square error = $J(\theta) = (X\theta - Y)^T (X\theta - Y)$

Rewrite $J(\theta) = ((X\theta)^T - Y^T)(X\theta - Y) = (X\theta)^T X\theta - (X\theta)^T Y - Y^T (X\theta) + Y^T Y$

Note $(X\theta)^T = \theta^T X^T$ and since $X\theta$ and Y are vectors $(X\theta)^T Y = Y^T (X\theta)$

$$J(\theta) = \theta^T X^T X\theta - 2(X\theta)^T Y + Y^T Y$$

Minimize the sum by finding its partial derivatives and set them to zero

$$\frac{\partial J}{\partial \theta} = \frac{\partial}{\partial \theta} (\theta^T X^T X\theta - 2(X\theta)^T Y + Y^T Y) = \boxed{2X^T X\theta - 2X^T Y} = 0$$

Derivative
or Gradient

The Normal Equation

- A **closed form** solution to the linear regression exists. Let $\text{MSE}(\theta) = J(\theta)$

- sum of the square error = $J(\theta) = (X\theta - Y)^T (X\theta - Y)$

Rewrite $J(\theta) = ((X\theta)^T - Y^T)(X\theta - Y) = (X\theta)^T X\theta - (X\theta)^T Y - Y^T (X\theta) + Y^T Y$

Note $(X\theta)^T = \theta^T X^T$ and since $X\theta$ and Y are vectors $(X\theta)^T Y = Y^T (X\theta)$

$$J(\theta) = \theta^T X^T X \theta - 2(X\theta)^T Y + Y^T Y$$

Minimize the sum by finding its partial derivatives and set them to zero

$$\frac{\partial J}{\partial \theta} = \frac{\partial}{\partial \theta} (\theta^T X^T X \theta - 2(X\theta)^T Y + Y^T Y) = \boxed{2X^T X \theta - 2X^T Y} = 0$$

Derivative
or Gradient

$$X^T X \hat{\theta} = X^T Y \quad \text{or} \quad \boxed{\hat{\theta} = (X^T X)^{-1} X^T Y \Rightarrow \text{the Normal Equation}}$$

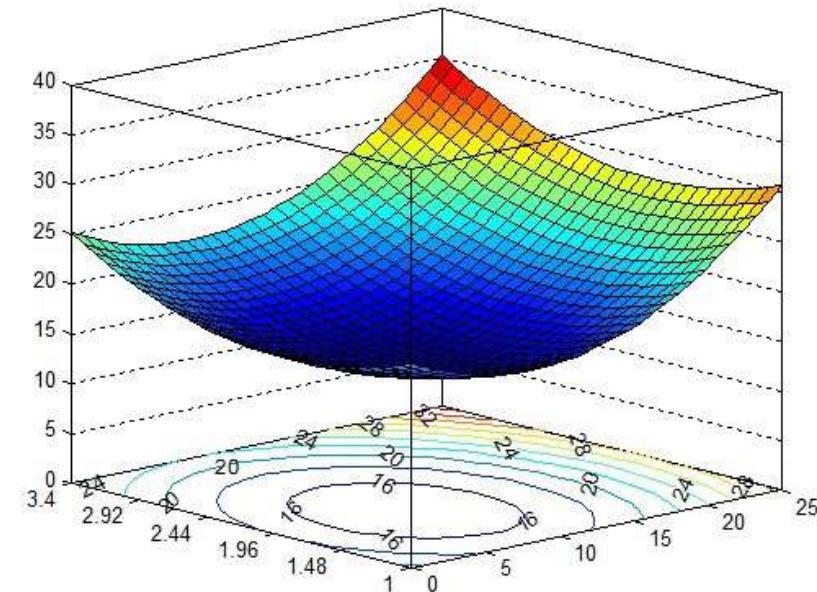
where $\hat{\theta}$ is the value of θ that minimizes the cost function

Gradient Descent

- In most cases, closed form solution doesn't exist.
- Recall

$$\text{Minimum of } \text{MSE}(\boldsymbol{\theta}) = \arg \min_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^m (\boldsymbol{\theta} \cdot \mathbf{x}^{(i)} - \mathbf{y}^{(i)})^2$$

- How do you find $\arg \min_{\boldsymbol{\theta}}$? What are θ_0 and θ_1 ?
- If you began anywhere on this surface, the strategy is to walk down hill
- $\theta_{next} = \theta_{current} + \Delta\theta_{step \text{ to lower } MSE}$
- Fast way to reach the minimum is with the steepest descent (gradient descent).
 - $\theta_{next} = \theta_{current} - \nabla_{\boldsymbol{\theta}} J$ where ∇ denotes gradient
- Unlike someone climbing down the slope, you don't see the landscape other than the parts immediately near (like being in the dark).

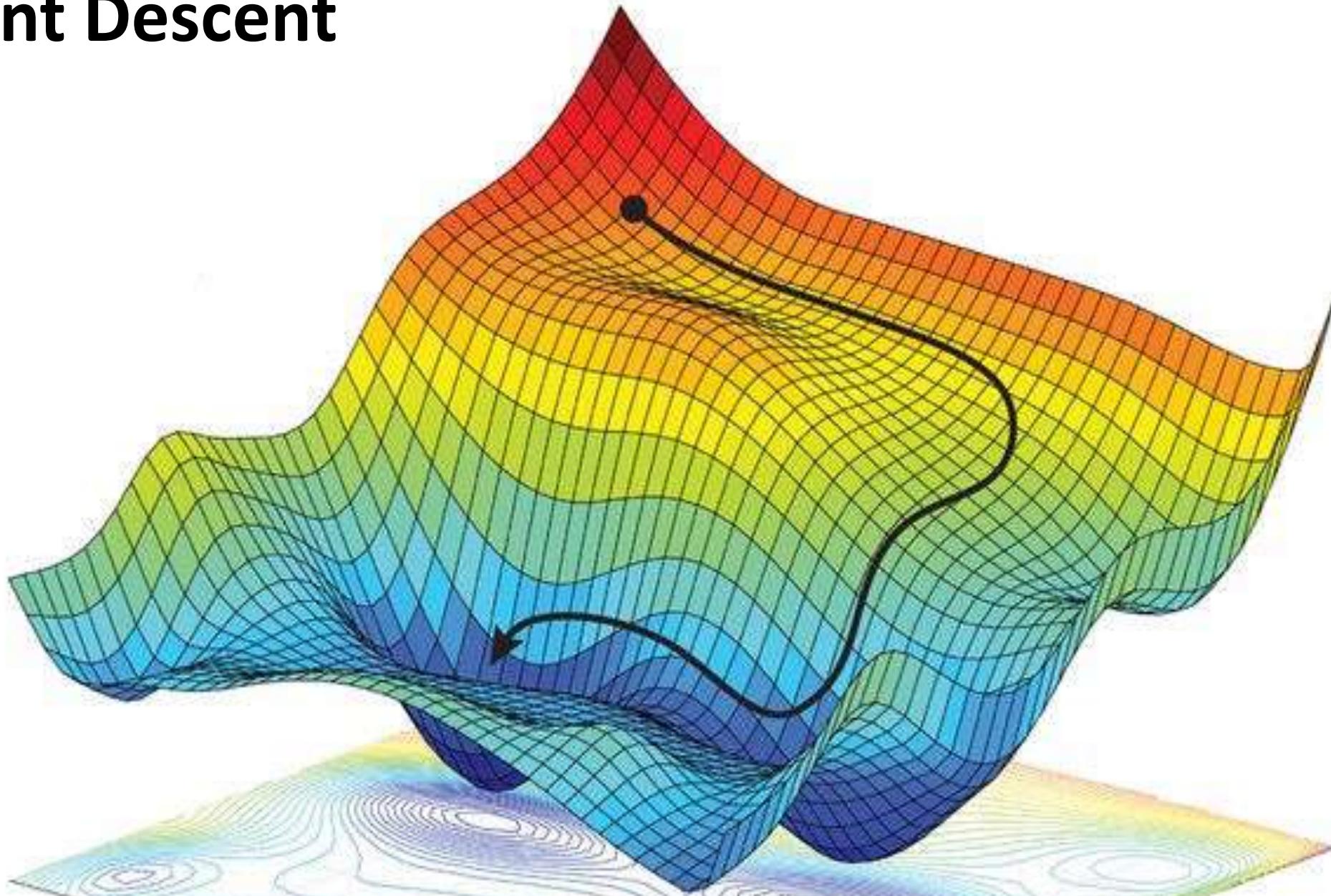


Gradient Descent Algorithm

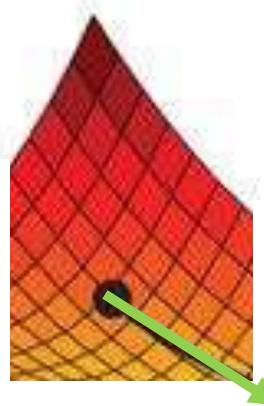
How to get to the bottom?

Which Way?

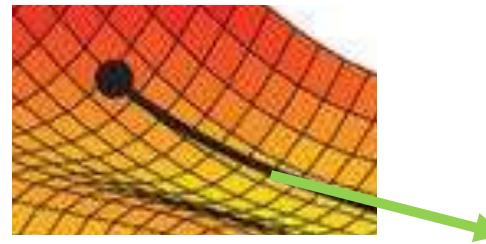
Gradient Descent



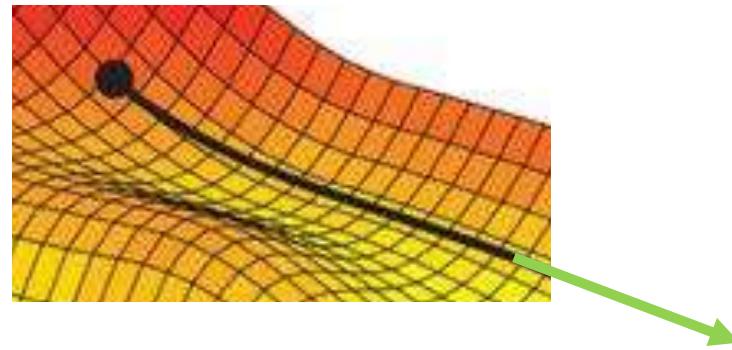
Gradient Descent



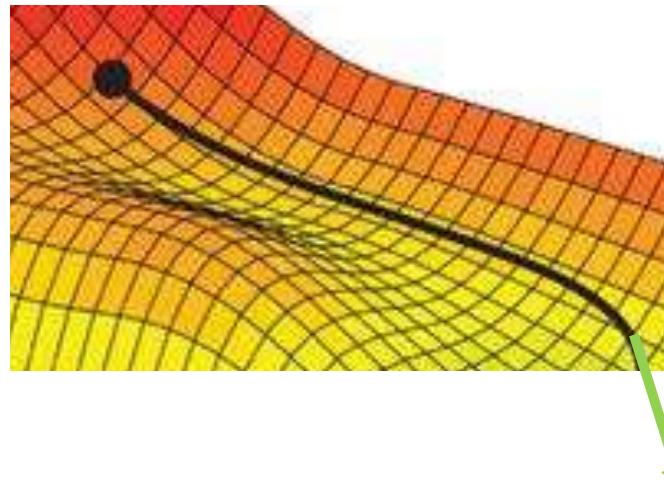
Gradient Descent



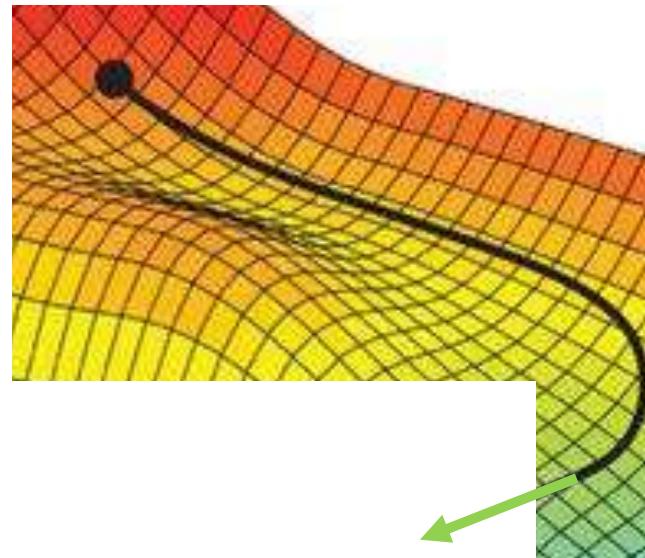
Gradient Descent



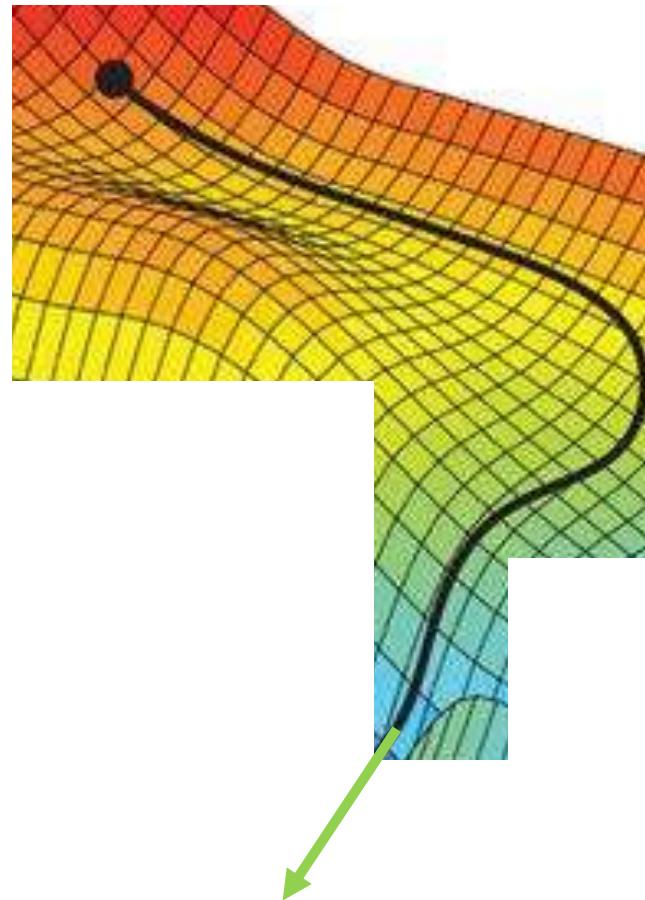
Gradient Descent



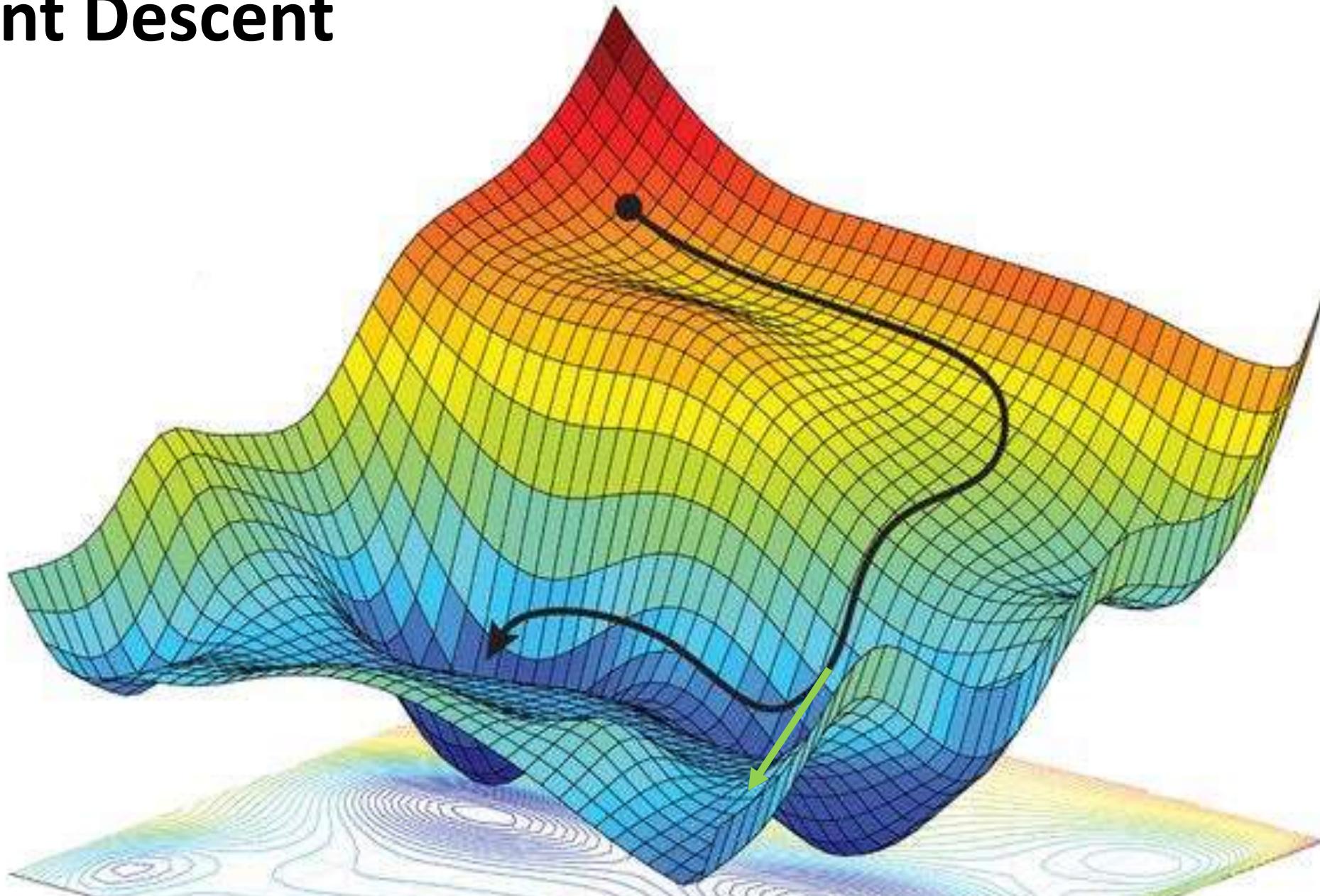
Gradient Descent



Gradient Descent

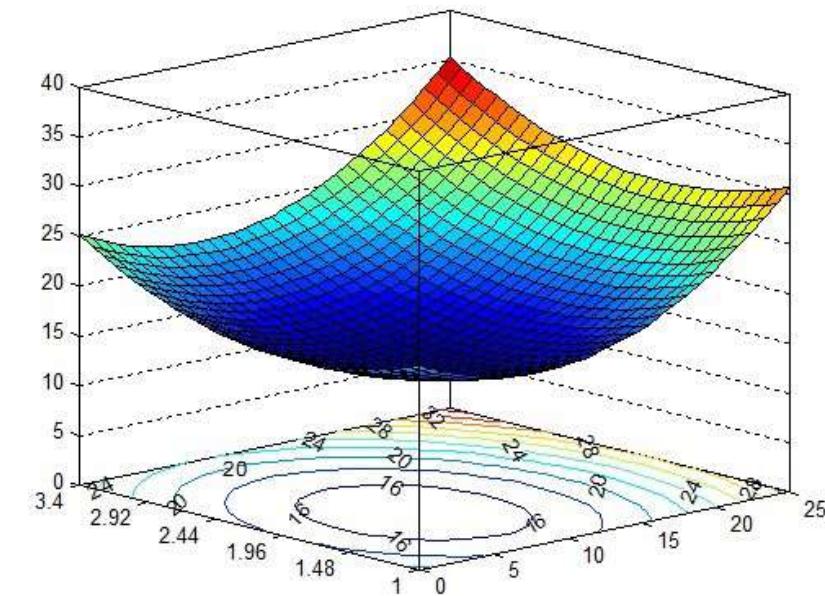


Gradient Descent



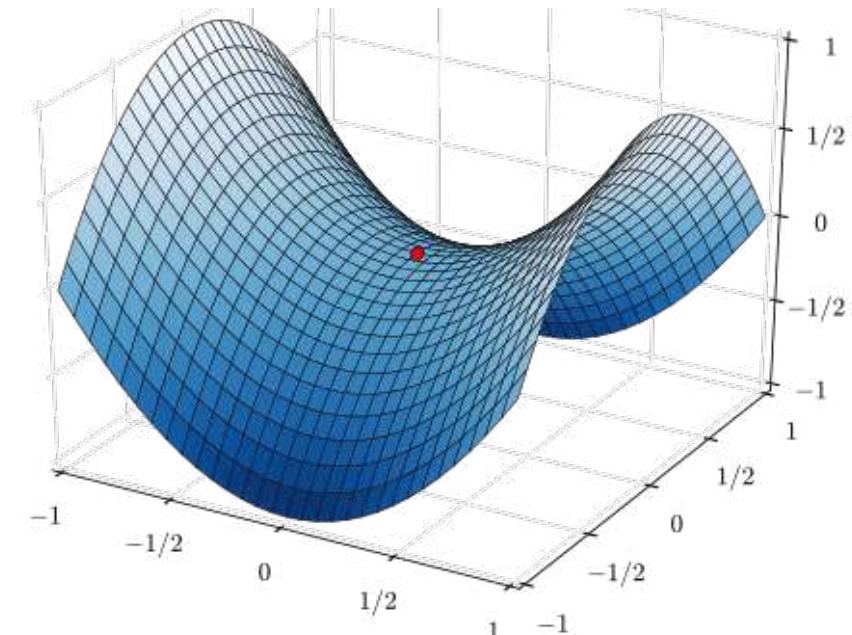
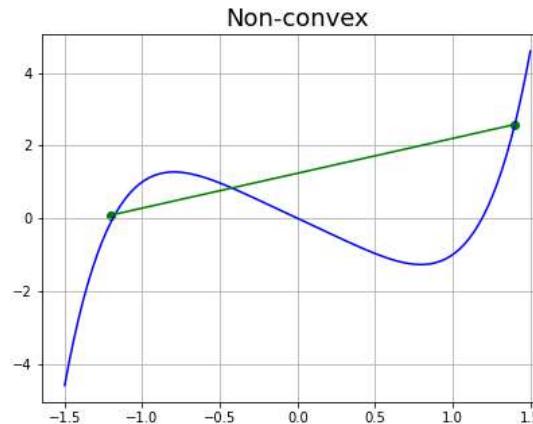
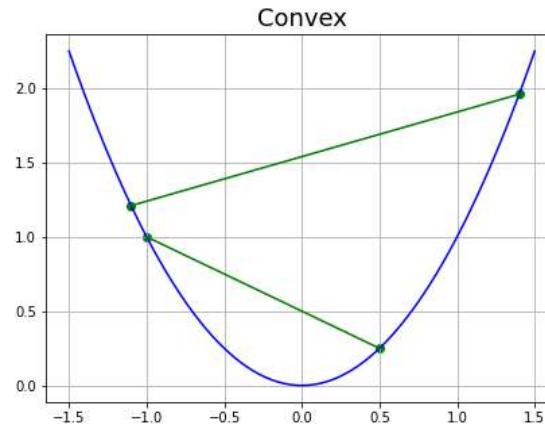
Gradient Descent

- You see the error going up and down depending on the step you take.
- Observe the slope and follow the lowest downhill.
- Being able to calculate the slope is crucial here! → calculate derivatives

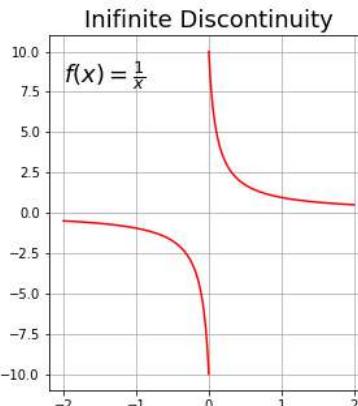
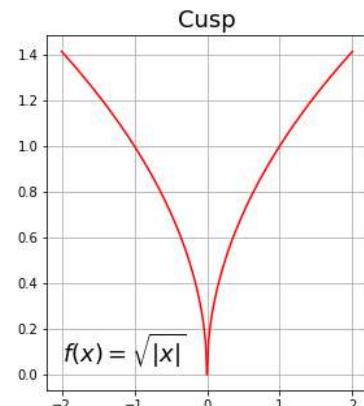
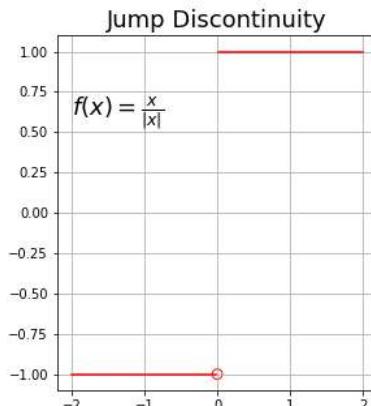


Gradient Descent

- Some conditions for this to work
 - Convex



- Differentiable



Batch Gradient Descent

- Computing partial derivatives

$$\frac{\partial}{\partial \theta_j} MSE(\theta) = \frac{2}{m} \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)}) x_j^{(i)}$$

- Calculation involves the full training set

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

- Gradient of MSE is computed by taking the average of the derivatives over the entire training batch. Thus, it is called batch gradient method.

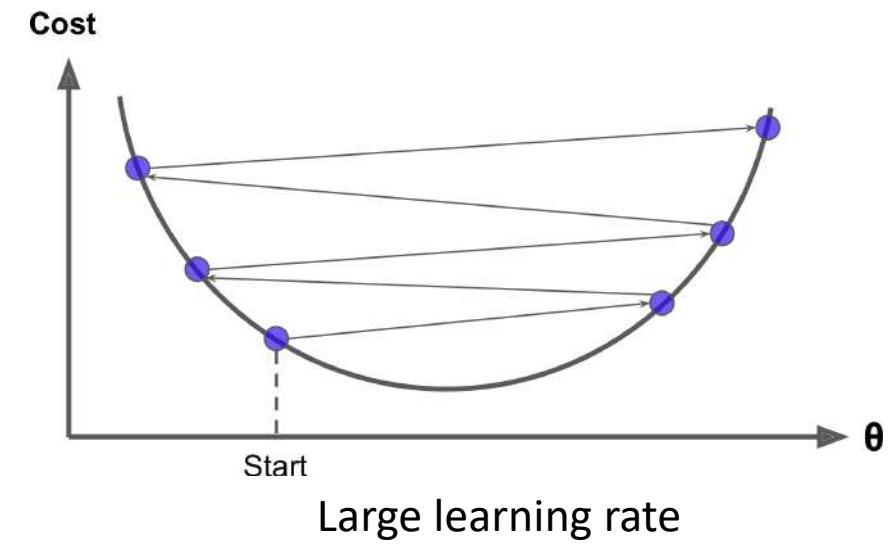
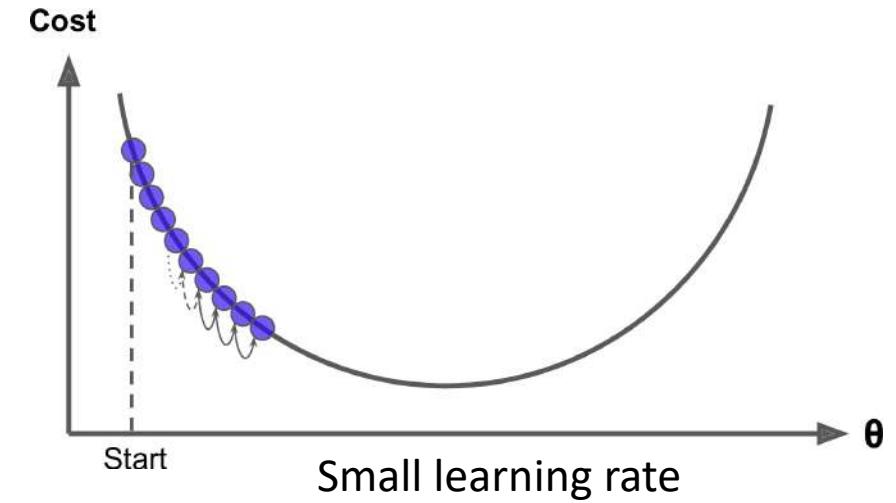
$$\theta^{(next\ step)} = \theta - \eta \nabla_{\theta} MSE(\theta)$$

where η is the learning rate

$$\nabla_{\theta} (MSE) = 2X^T X \theta - 2X^T Y = 2X^T (X \theta - Y)$$

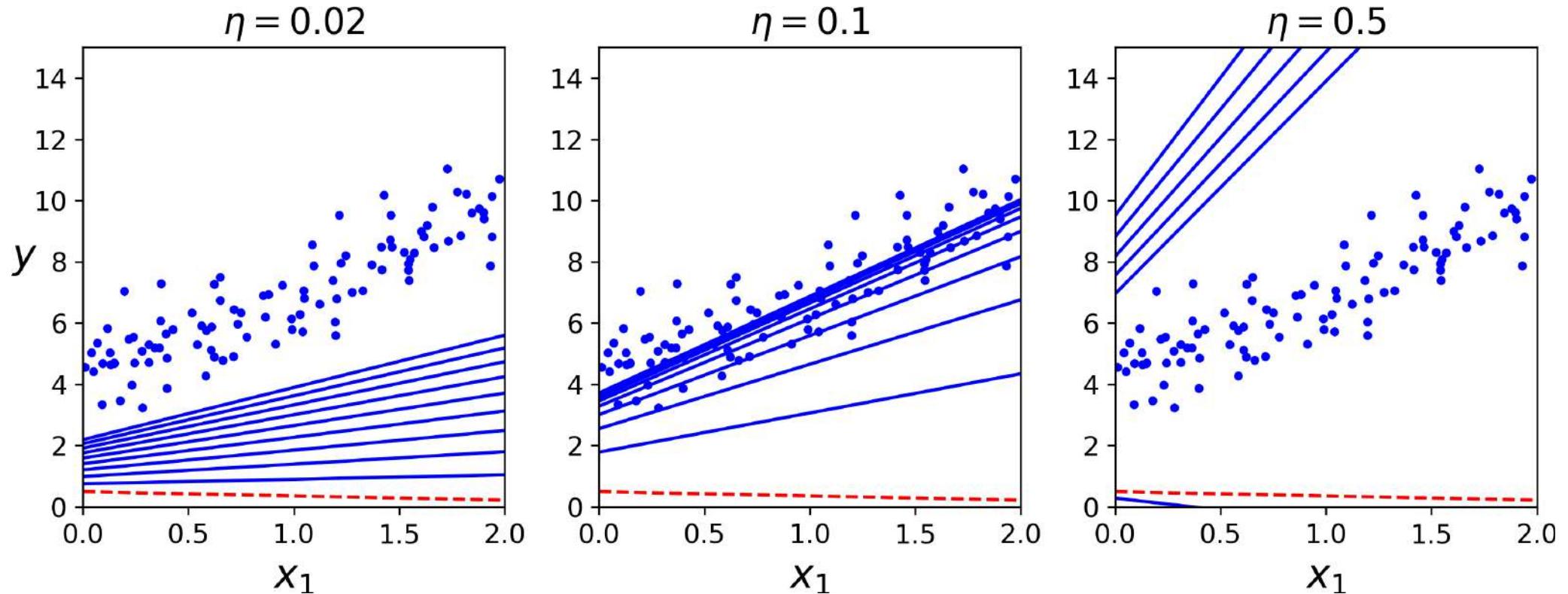
Gradient Descent

- In 1-D search
- How **fast to the minimum** is how large are the steps
- Gradient only tells you the direction and magnitude
- Step size control → learning rate η



Batch Gradient Descent

- Learning rate



- Setting a proper learning rate is crucial in machine learning. Often it is not clear what would be the most appropriate.
 - Consider grid search (chap. 2 of Aurelien Geron)

Gradient Descent Algorithm

- Repeat until convergence

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

Correct: Simultaneous update

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

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

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

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

Incorrect:

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

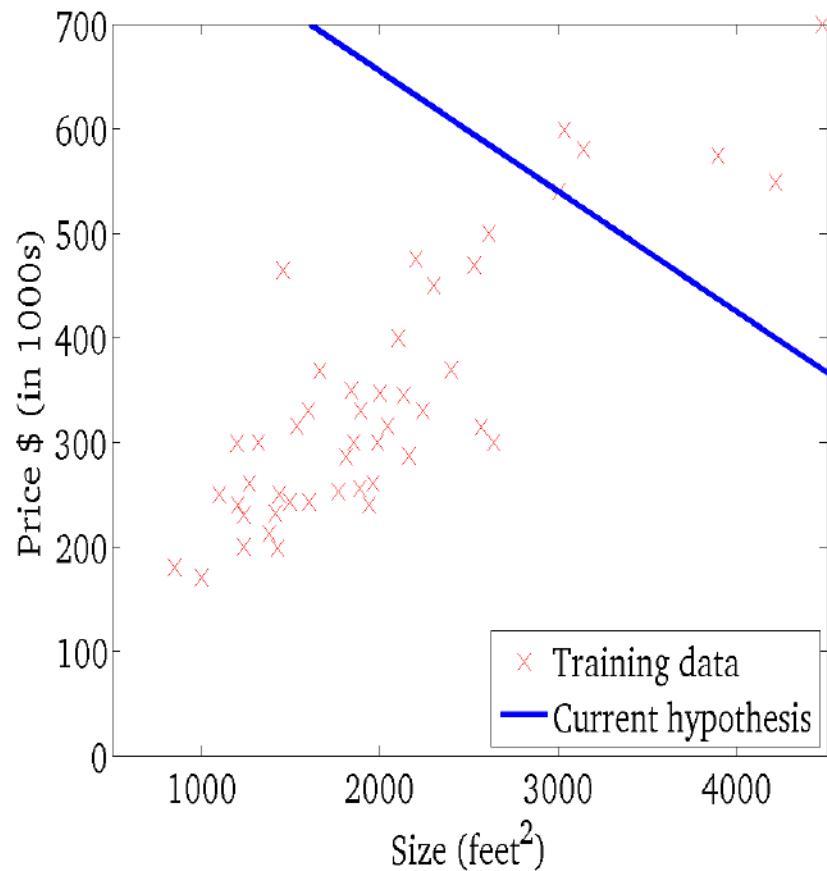
$$\theta_0 := \text{temp0}$$

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

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

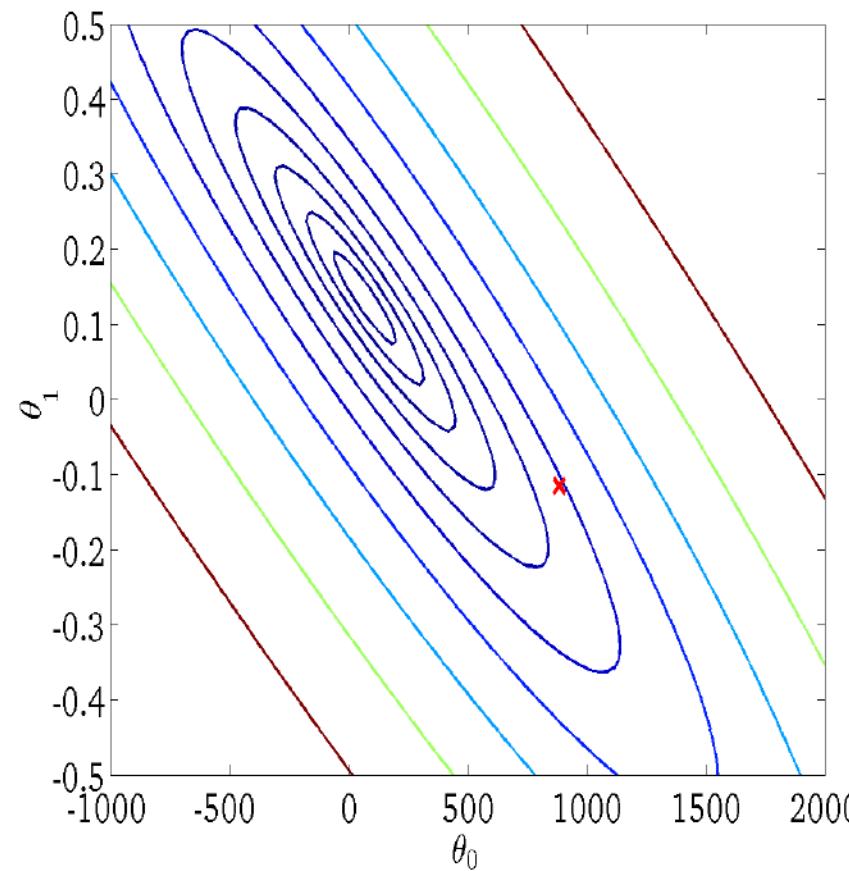
$$h_{\theta}(x)$$

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



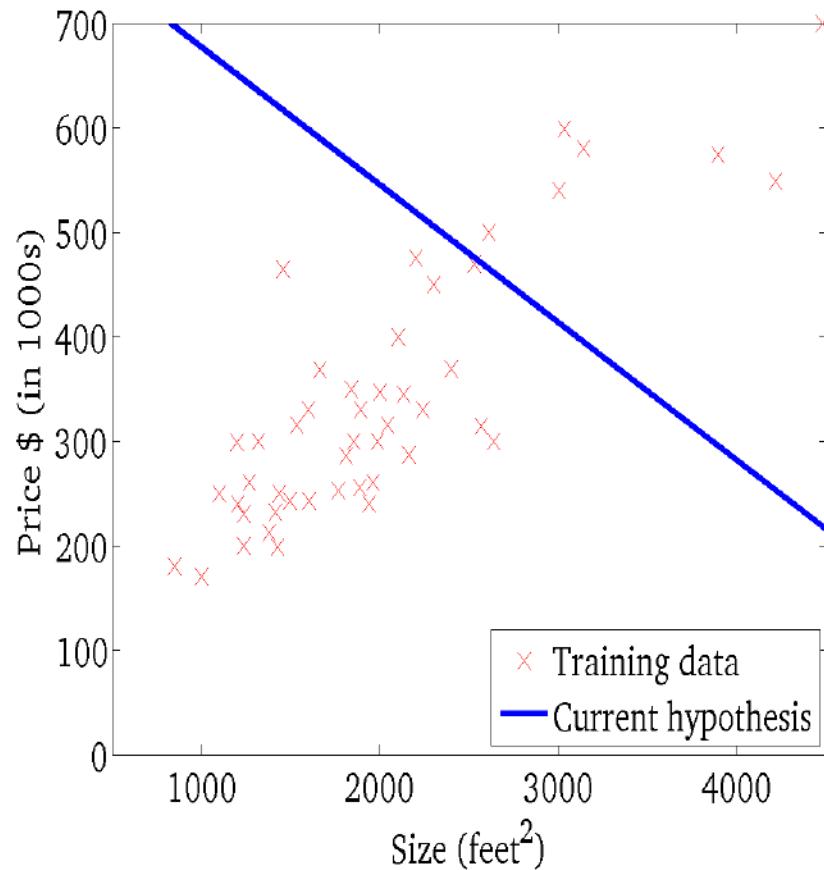
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



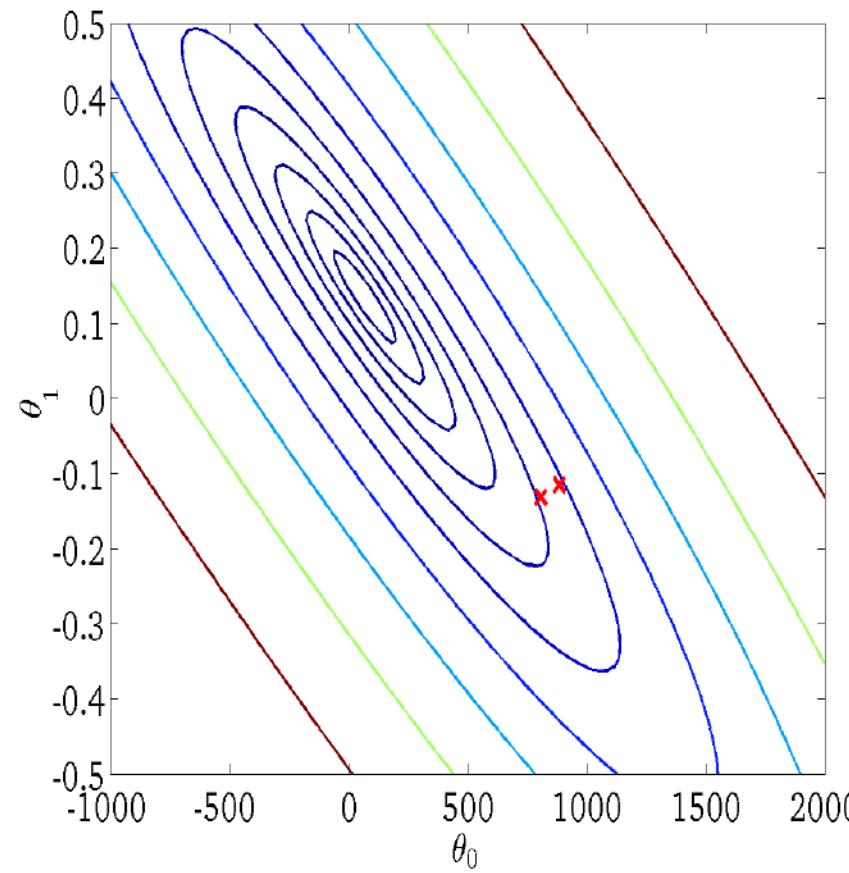
$$h_{\theta}(x)$$

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



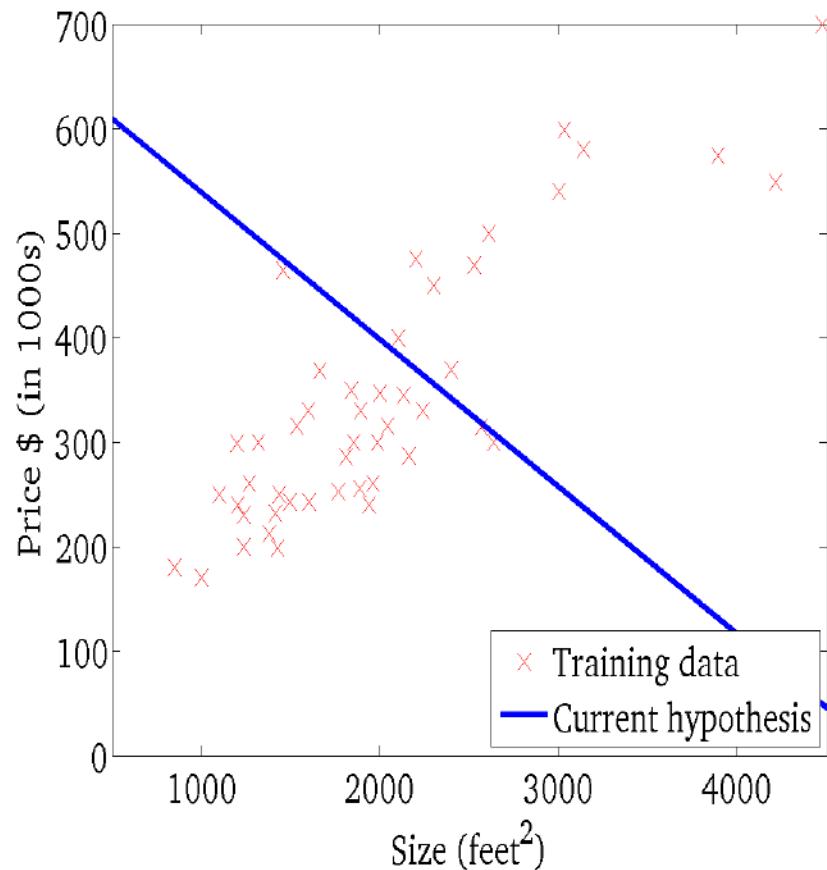
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



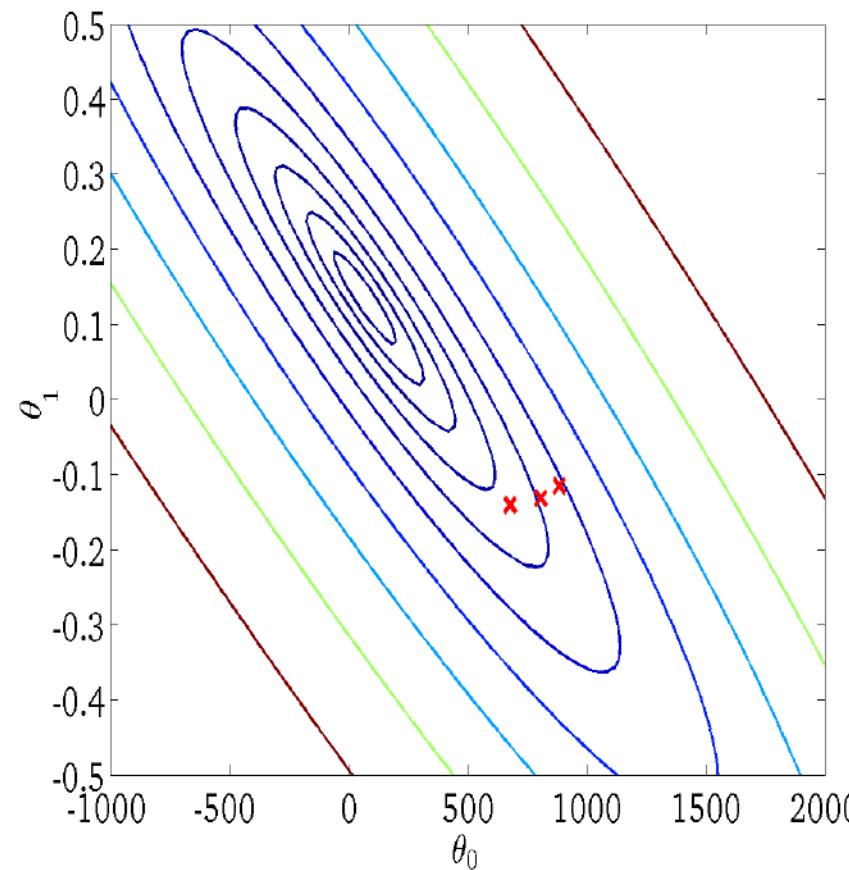
$$h_{\theta}(x)$$

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



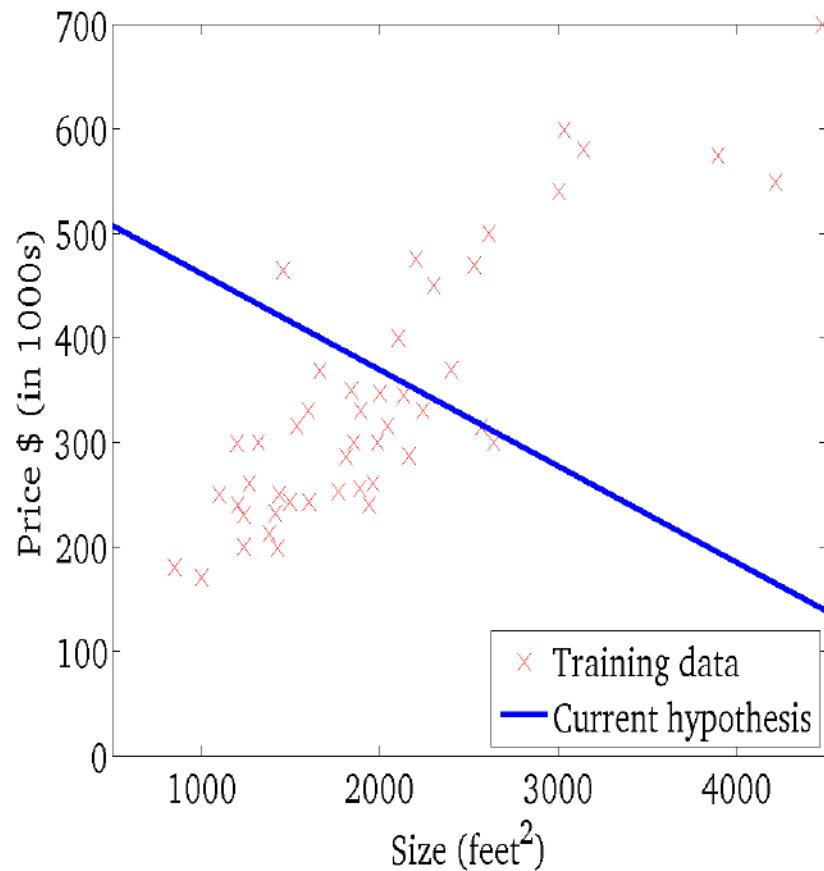
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



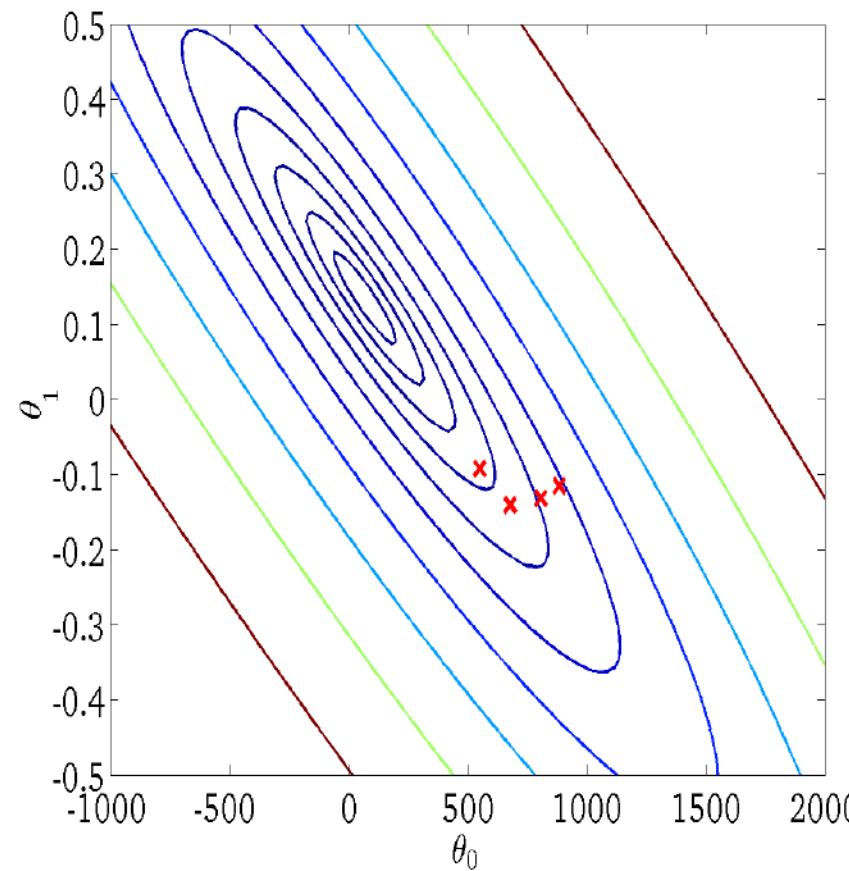
$$h_{\theta}(x)$$

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



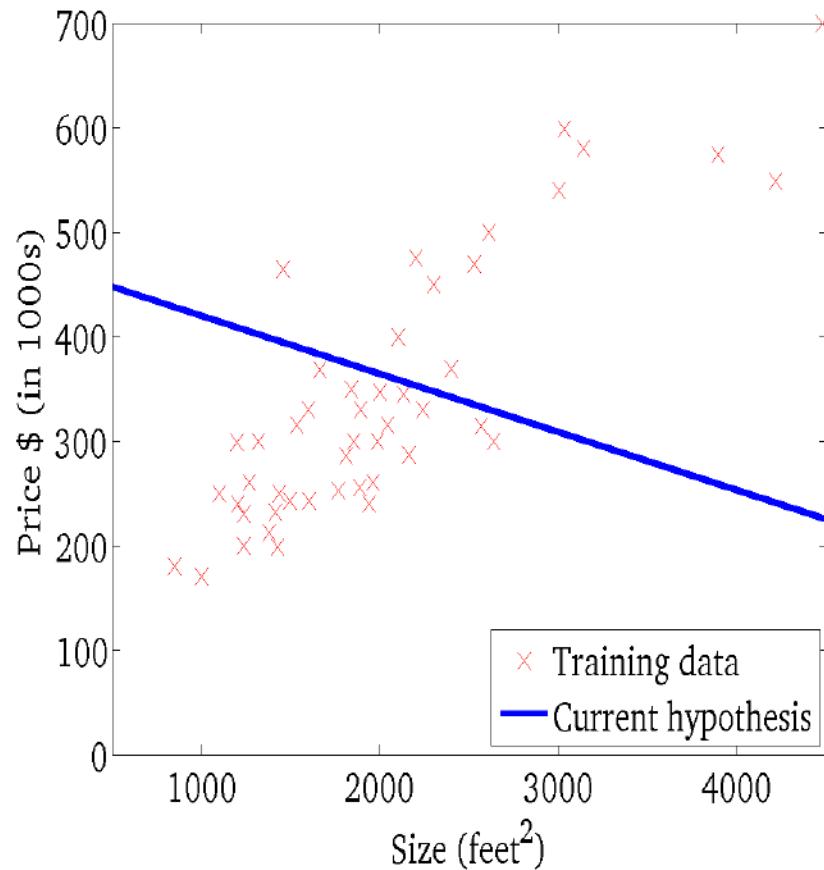
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



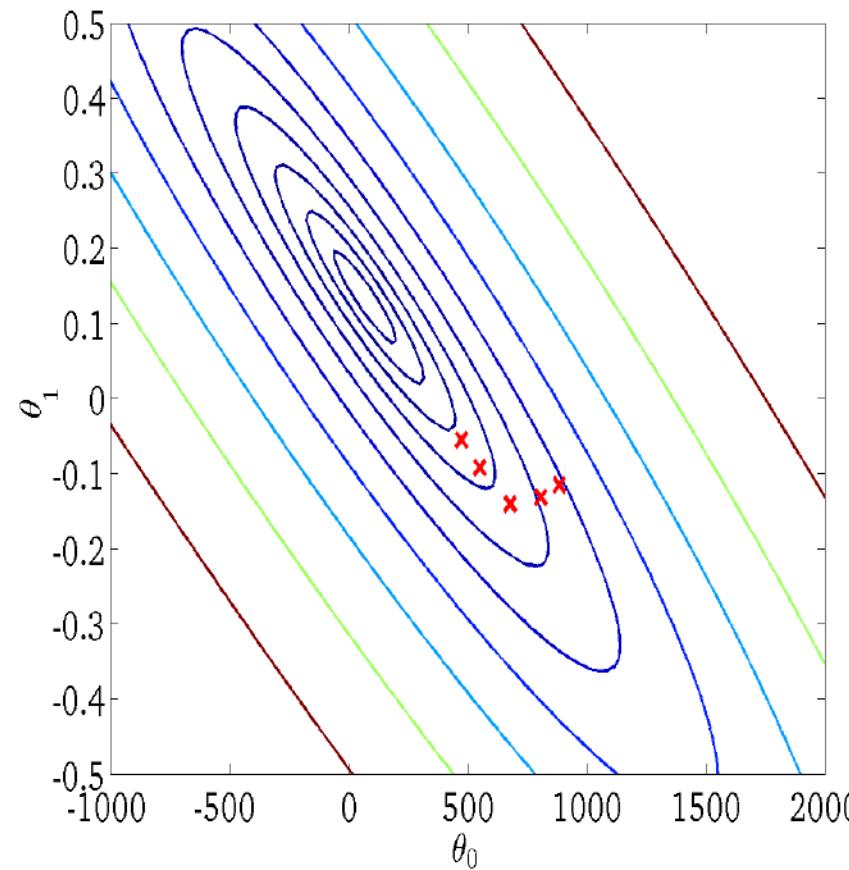
$$h_{\theta}(x)$$

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



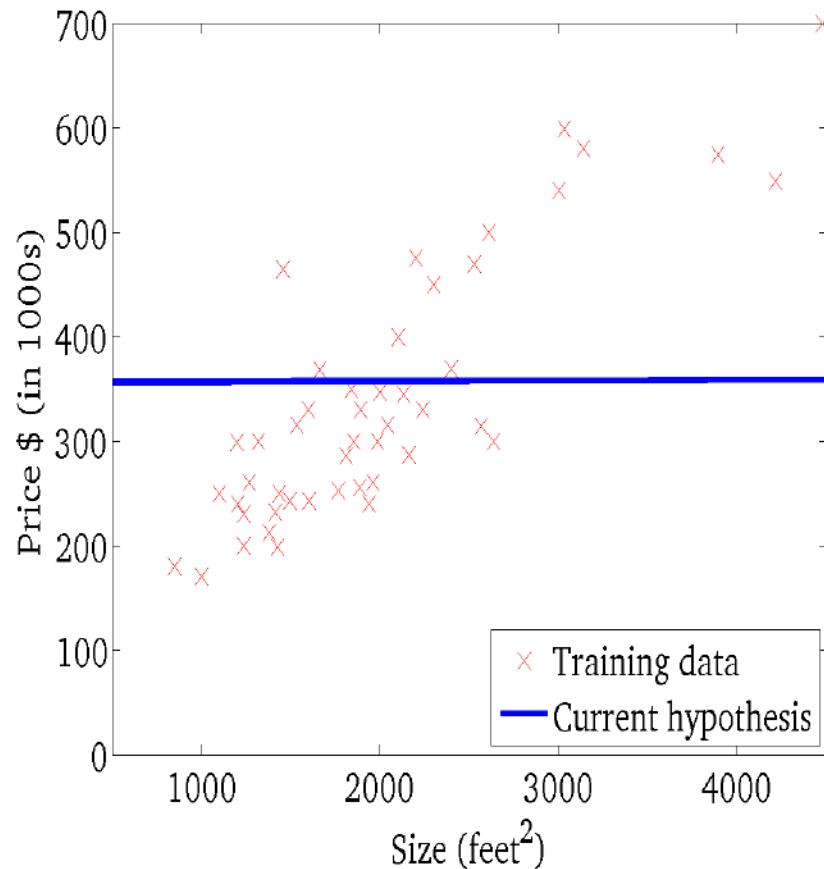
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



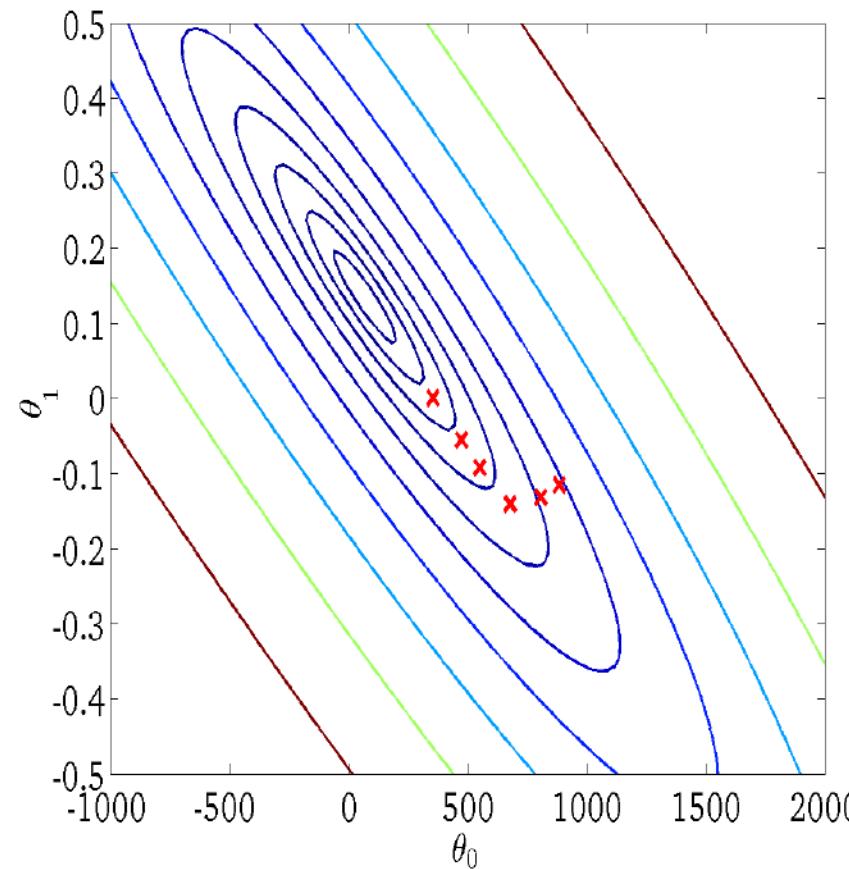
$$h_{\theta}(x)$$

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



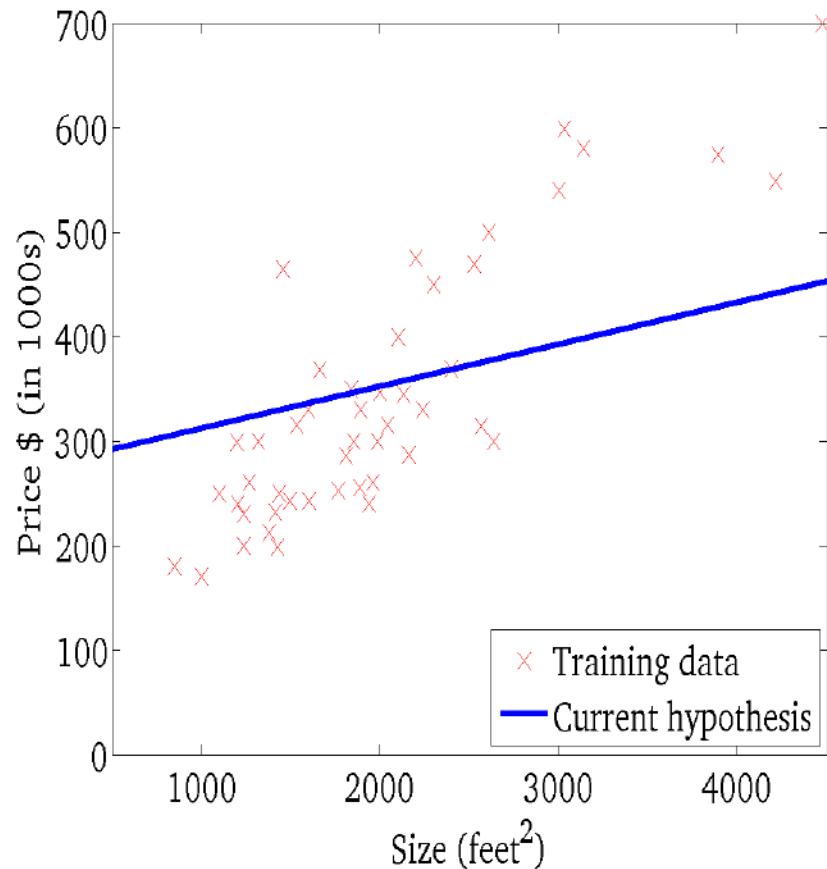
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



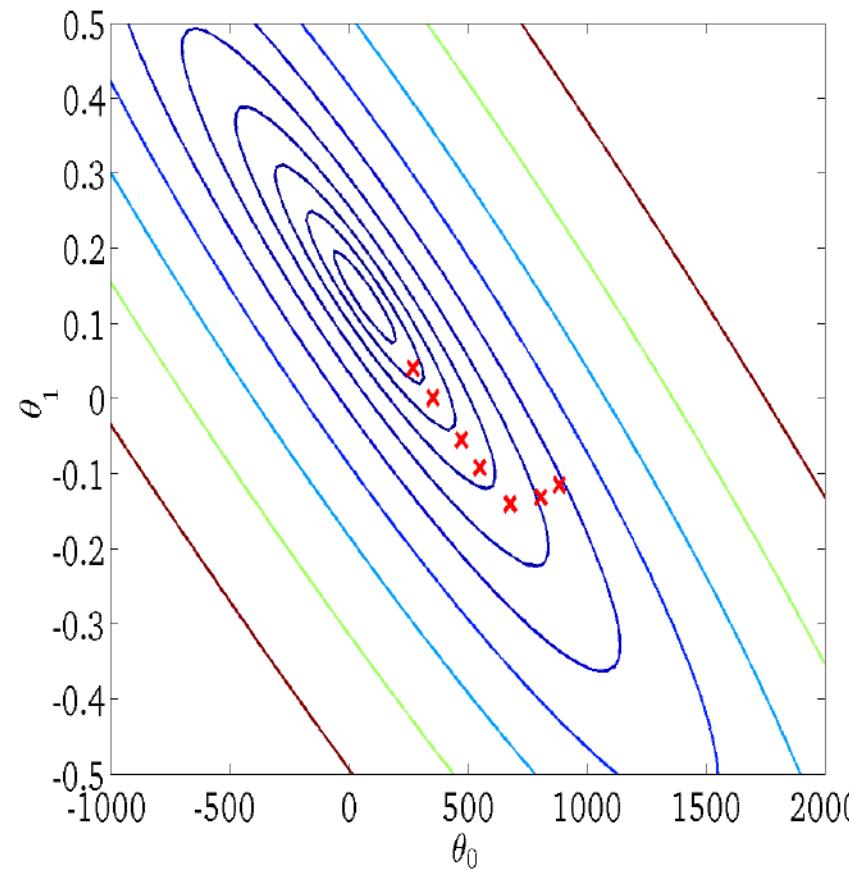
$$h_{\theta}(x)$$

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



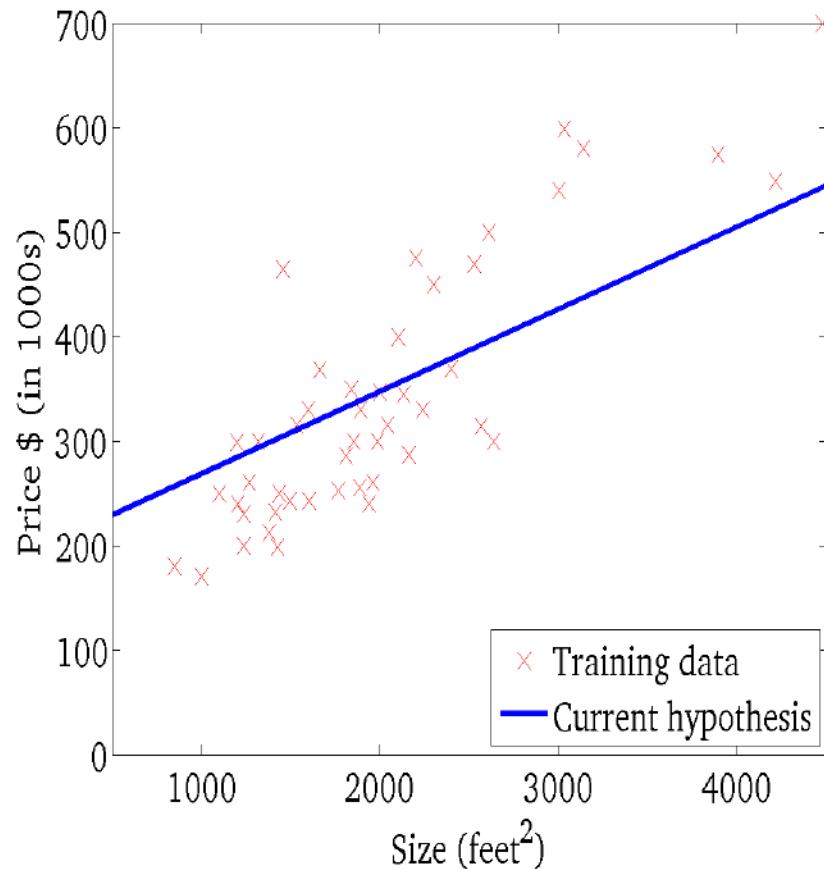
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



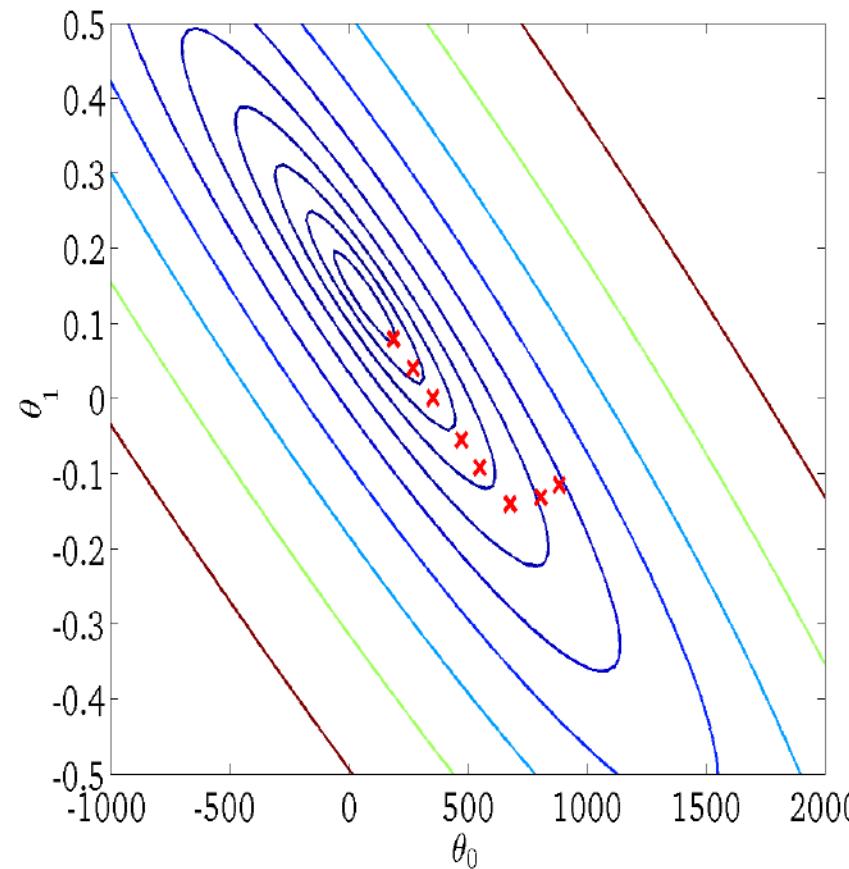
$$h_{\theta}(x)$$

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



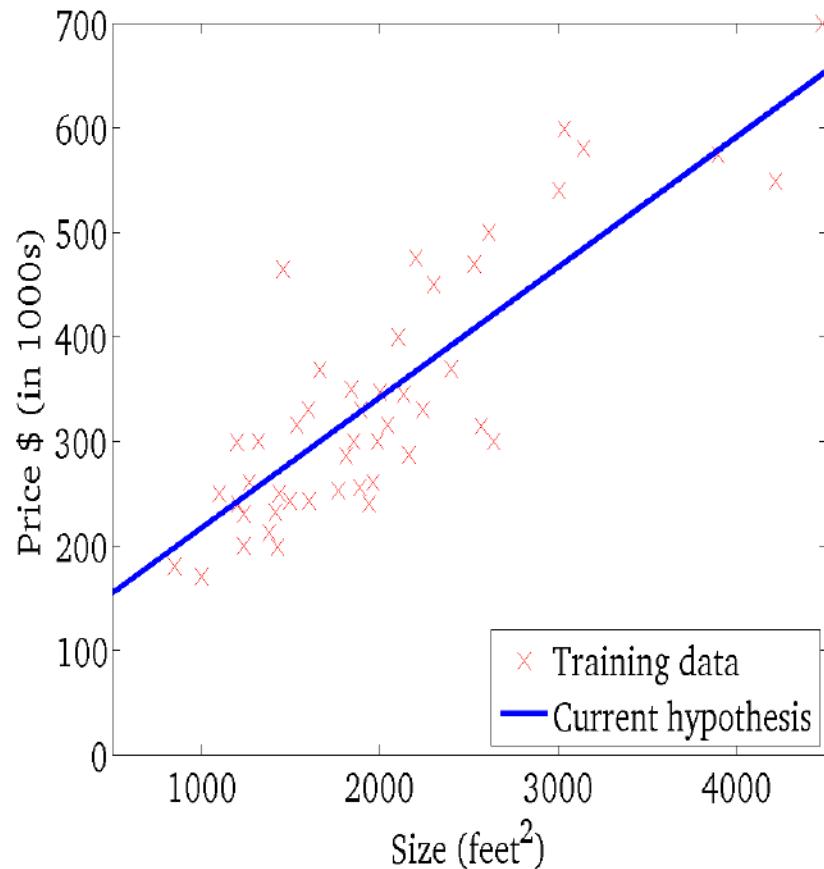
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



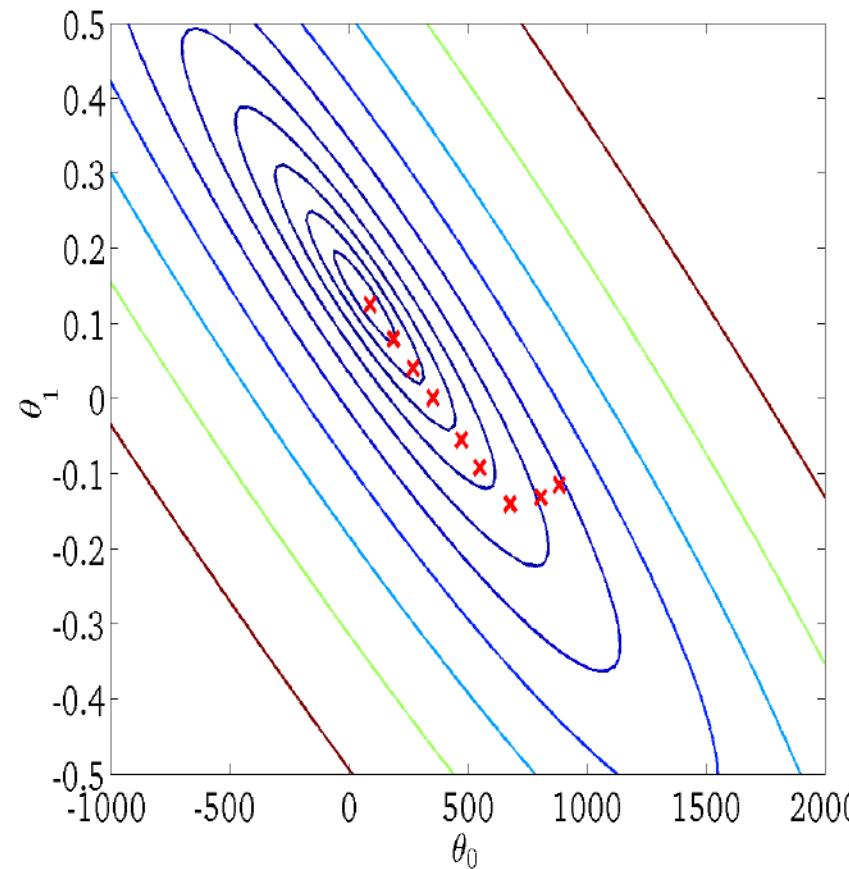
$$h_{\theta}(x)$$

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



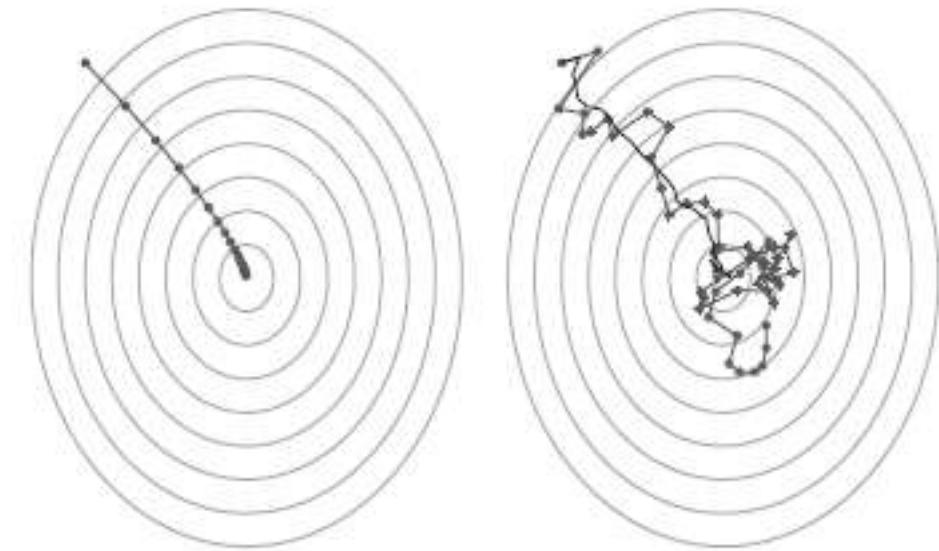
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



Stochastic Gradient Descent

- Computing gradient by taking a random **single instance** (single data point) and update the solution value immediately.
- Takes **less time** to compute gradient
- **Approach** to the solution becomes **stochastic** as a random sample from the data is taken at each step.
- More **efficient** in approaching to optimal solution when the **data set is large**
- Even when the algorithm approaches near an **optimal solution**, loss function value may **fluctuate**, and may not reach the optimal values although the solution will be very close to it.
 - Gradual **reduction of learning rate** may help.
- May work **better** than batch gradient when the cost function is **not convex**.

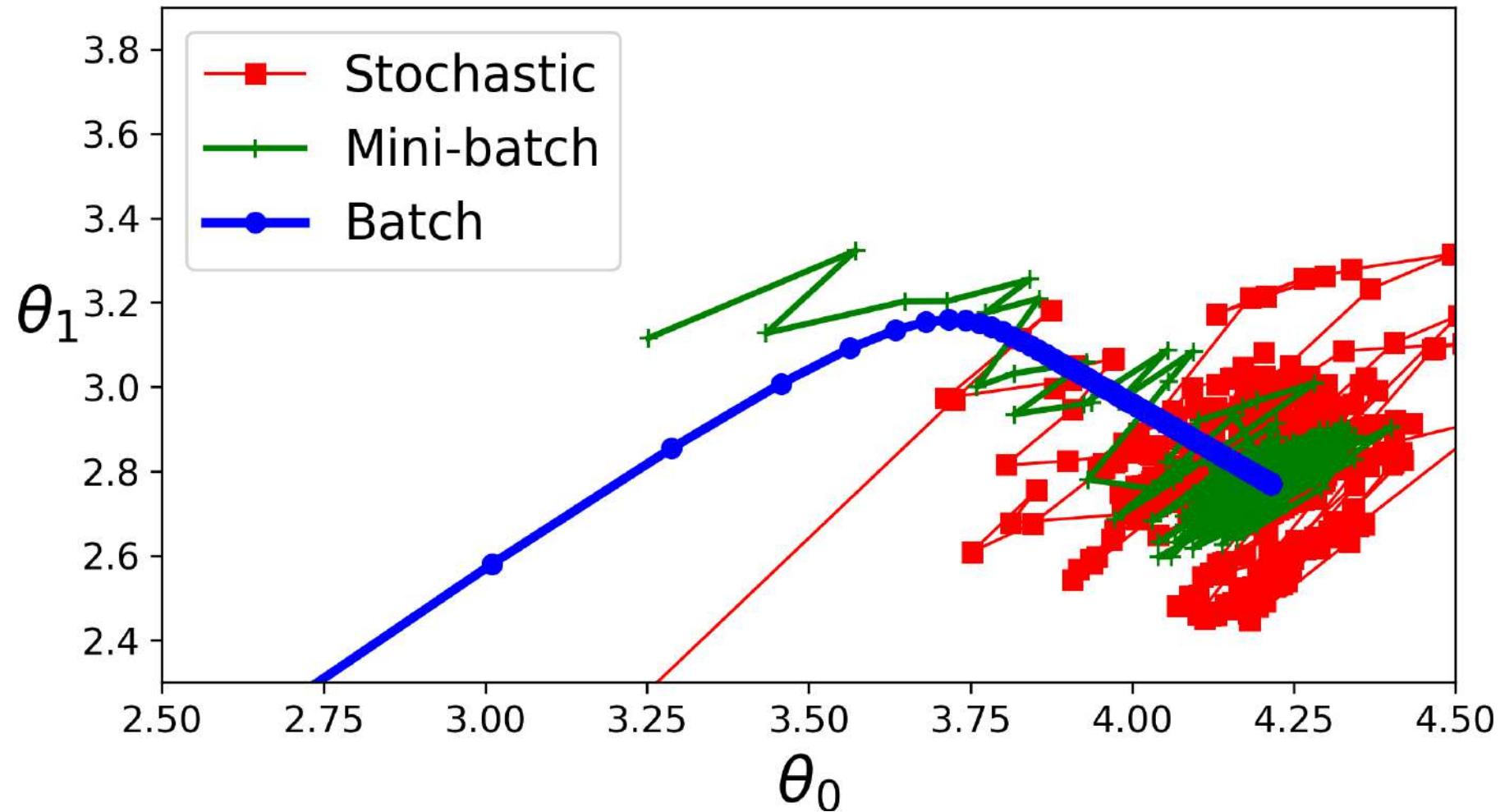


Batch vs Stochastic Gradient Descent

$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\theta} \text{MSE}(\theta)$$

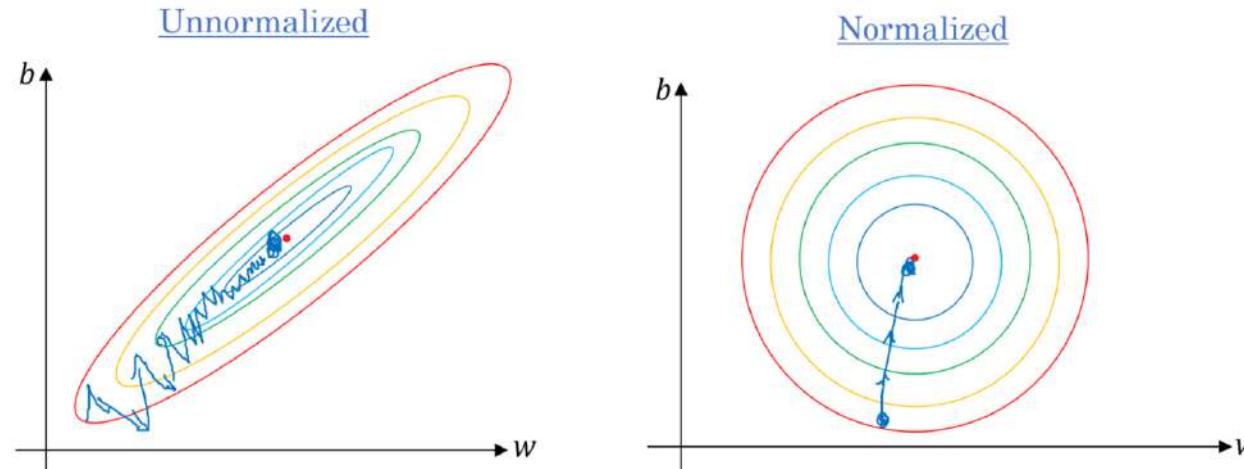
where η is the learning rate

Batch, Stochastic, & Mini-Batch



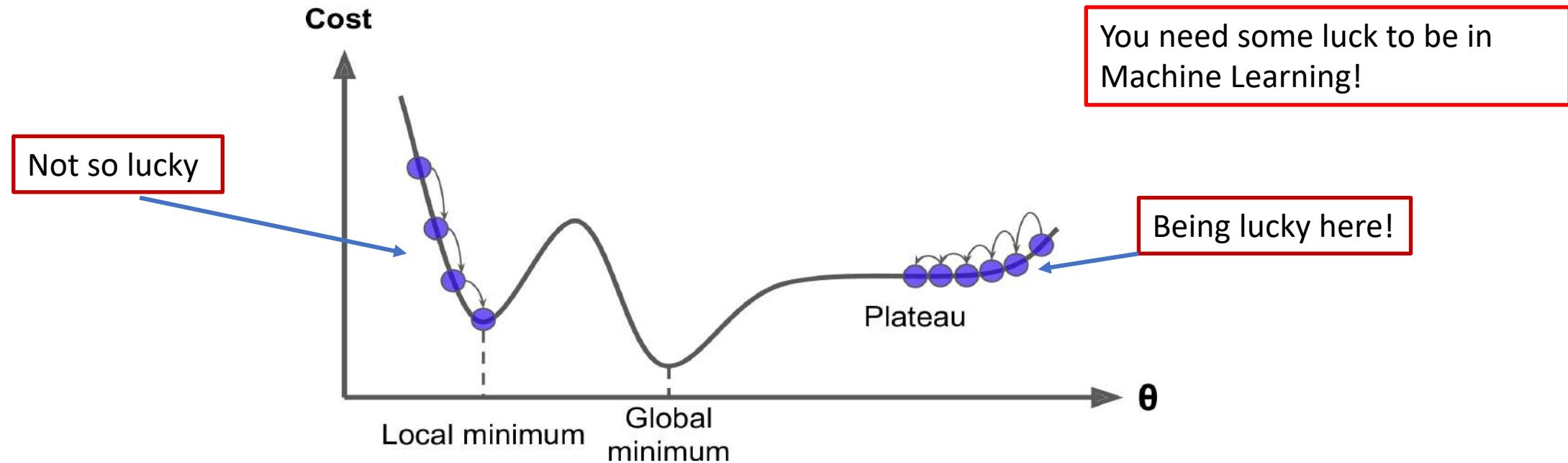
Gradient Descent

- **Convex function**: no local minimum and only one global minimum
- For convex functions, gradient descent is **guaranteed** to work.
- **Feature scaling** and gradient descent



- Scaled features → More rapid descent

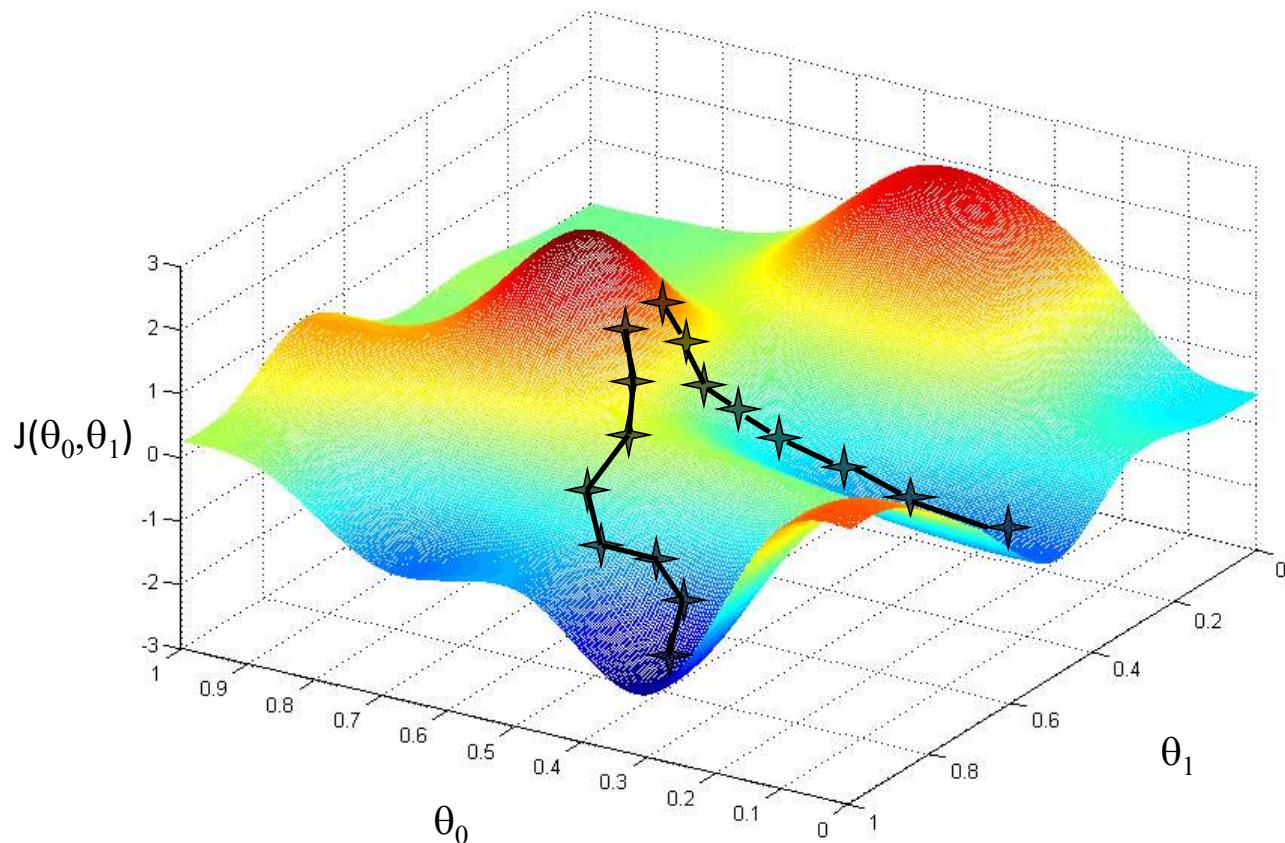
Gradient Descent



- Some cost functions (in fact most of the cost functions for interesting problems) are **not shaped like bowls (not convex)**.
- Thus, there may be local minima on the path to the global minimum.
- Nonlinear problems such as neural network often results such local minima.
- Gradient descent may not work well in such cases.

Gradient Descent

- Therefore, the path to the minimum can be quite an adventure!



Logistic Regression

ECE 610

David Han

Drexel University

Probabilistic Interpretation of Linear Regression

- Let's consider the linear regression problem in terms of probability and see why the cost function J is a reasonable choice.
- Let's assume that the target variables and the inputs are related via the equation
 - $y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$
 - where $\varepsilon^{(i)}$ is an error term that captures either unmodeled effects or random noise.
- Let's also assume that $\varepsilon^{(i)}$ s are distributed IID (Independently and Identically Distributed) according to a Gaussian distribution with zero mean and variance of σ^2
- Therefore, $p(\varepsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\varepsilon^{(i)})^2}{2\sigma^2}\right)$
- Also by substitution $p(y^{(i)} | \vec{x}^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T \vec{x}^{(i)})^2}{2\sigma^2}\right)$

Probabilistic Interpretation of Linear Regression

- “ $p(y^{(i)} | \mathbf{x}^{(i)} ; \theta)$ ” indicates that this is the probability distribution of $y^{(i)}$ given $\mathbf{x}^{(i)}$ and parameterized by θ .
- In other words, it describes the distribution of the $y^{(i)}$ ’s, given \mathbf{X} (the design matrix, which contains all the $\mathbf{x}^{(i)}$ ’s) and θ .
- We can also call it the likelihood function:
 - $L(\theta) = L(\theta; \mathbf{X}, \mathbf{y}) = p(\mathbf{y}|\mathbf{X}; \theta)$
 - as the probability will change as a function of the choice of θ
- Now, by assuming that each $\epsilon^{(i)}$ is independent, consider the entire dataset pair and the likelihood of the dataset occurring

$$L(\theta) = \prod_{i=1}^m p(y^{(i)} | \vec{\mathbf{x}}^{(i)}; \theta) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T \vec{\mathbf{x}}^{(i)})^2}{2\sigma^2}\right)$$

Probabilistic Interpretation of Linear Regression

- Now, given this probabilistic model relating the $y^{(i)}$'s and the $\mathbf{x}^{(i)}$'s, what is a reasonable way of choosing our best guess of the parameters θ ?
- Maximum likelihood: choose θ to make the data as high probability as possible.
 - Choose θ to maximize $L(\theta)$
- Maximizing $L(\theta)$ may be a bit messy. So let's maximize its log or max the log likelihood:

$$\begin{aligned}\ell(\theta) &= \log L(\theta) = \log \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T \vec{\mathbf{x}}^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T \vec{\mathbf{x}}^{(i)})^2}{2\sigma^2}\right) \\ &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T \vec{\mathbf{x}}^{(i)})^2\end{aligned}$$

Probabilistic Interpretation of Linear Regression

- Discarding the constants, maximizing $\ell(\theta)$ then becomes maximizing

$$\arg \max_{\theta} \left[-\frac{1}{2} \sum_{i=1}^m \left(y^{(i)} - \theta^T \vec{x}^{(i)} \right)^2 \right]$$

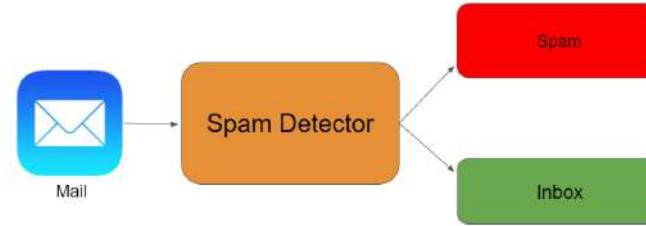
- which is the same as minimizing the negative of the summation term. It is the same term to be minimized as in the linear regression!

$$\arg \min_{\theta} \left[\sum_{i=1}^m \left(y^{(i)} - \theta^T \vec{x}^{(i)} \right)^2 \right]$$

- Under the probabilistic assumptions on the data, least-squares regression corresponds to finding the maximum likelihood estimate of θ .
- Thus, the least-squares regression can be justified as a method for maximum likelihood estimation.

Logistic Regression

- Binary Classification Problems



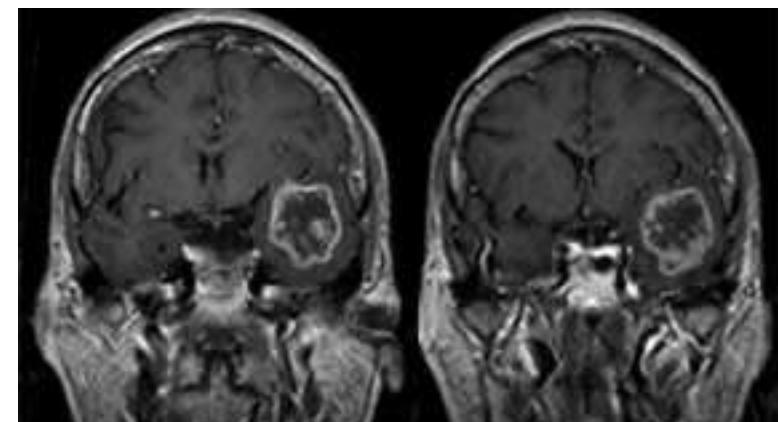
- Email: Spam or Ham?



- ATM machine currency check: Fraudulent (yes or no)?

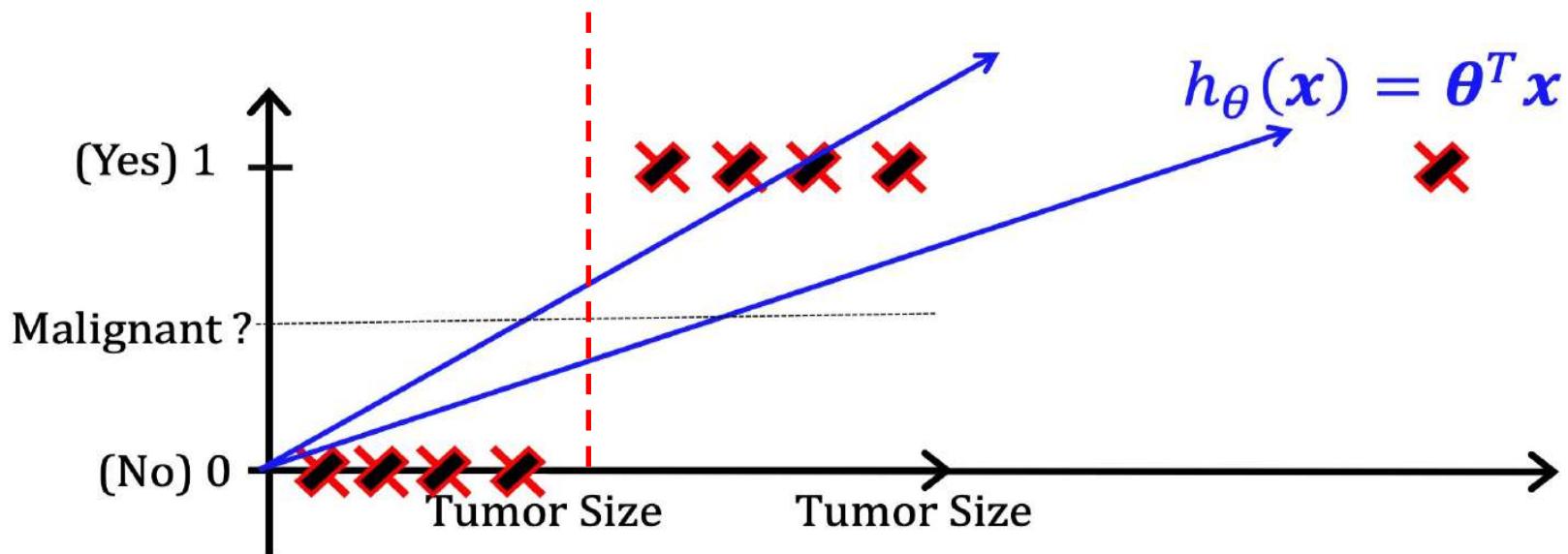
- Tumor: Malignant or Benign?

- $y \in \{0, 1\}$
- 0: Negative Class (e.g. benign tumor)
- 1: Positive Class (e.g. malignant tumor)



Logistic Regression

- Consider a hypothesis function for 1D feature case of predicting cancer: $h_{\theta}(x) = \theta^T x = w_1 x + b$



- A threshold based 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”

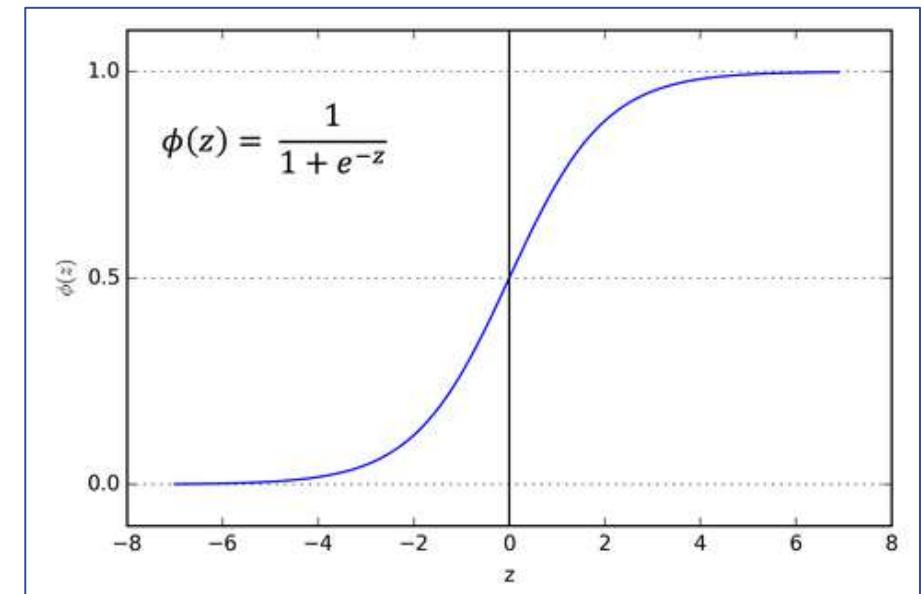
Logistic Regression

- Linear regression model is not the most suitable one for classification
- Hypothesis function now needs to predict values $y \in \{0, 1\}$.
- Logistic regression: binary prediction of $\{0, 1\}$.
- Thus, it is the right choice for a binary classification.

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

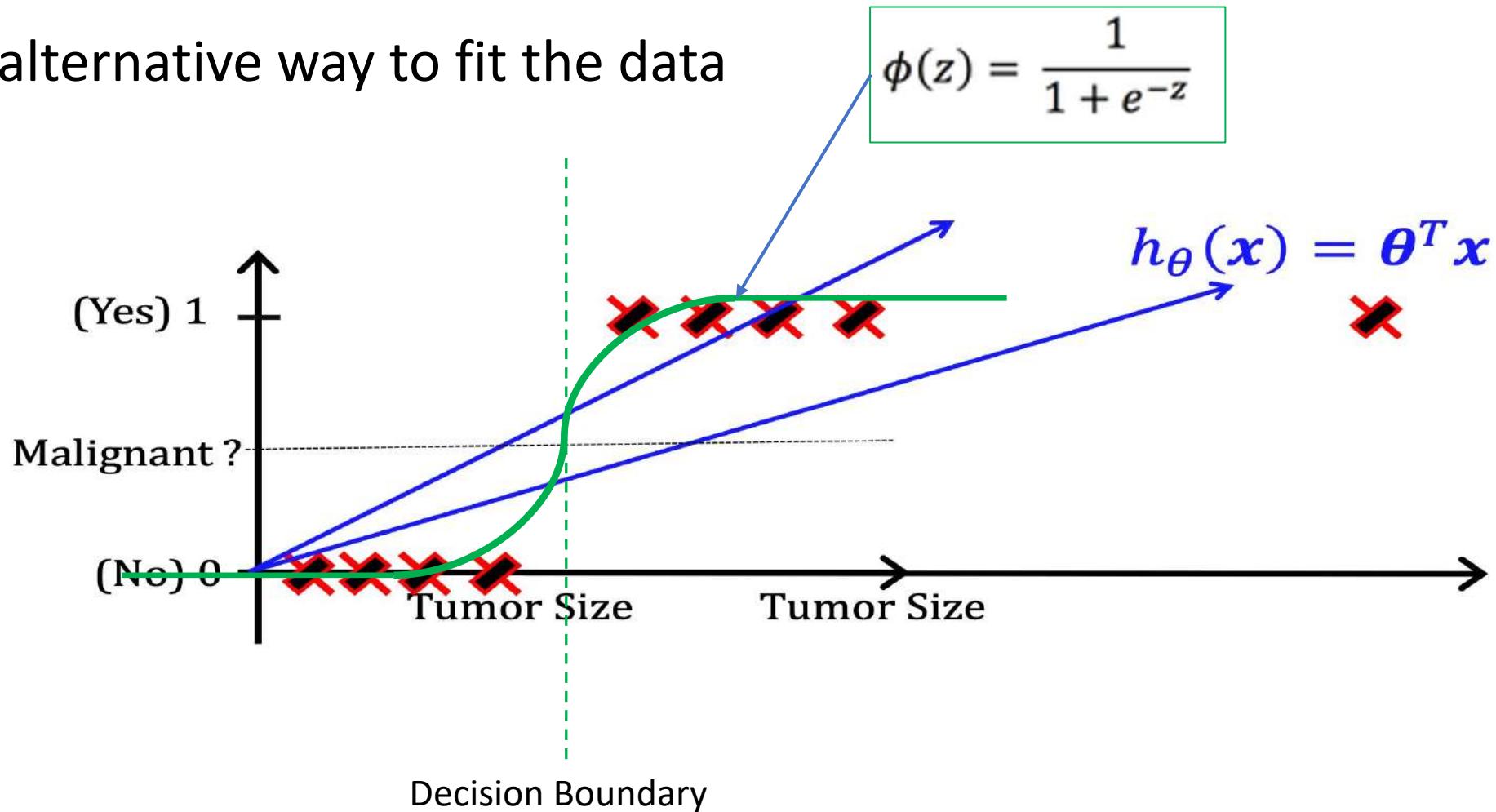
$$\text{where } g(z) = \frac{1}{1 + e^{-z}}$$

- is called logistic function or the sigmoid function



Logistic Regression

- An alternative way to fit the data



Logistic Regression

- Notice that $g(z)$ tends towards 1 as $z \rightarrow \infty$, and $g(z)$ tends towards 0 as $z \rightarrow -\infty$.
- Moreover, $g(z)$, and hence also $h(x)$, is always bounded between 0 and 1. As before, we are keeping the convention of letting $x_0 = 1$, so that

$$\theta^T \vec{x} = \theta_0 + \sum_{j=1}^n \theta_j x_j$$

- Some useful property of the derivative of the sigmoid function written as g'

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1+e^{-z}} = \frac{1}{(1+e^{-z})^2} (e^{-z}) \\ &= \frac{1}{(1+e^{-z})} \left(1 - \frac{1}{(1+e^{-z})} \right) = g(z)(1-g(z)) \end{aligned}$$

Logistic Regression

- Recall for N-dimensional features (\mathbf{x})
 - $\theta^T \mathbf{x} = [\theta_0, \theta_1, \theta_2, \dots, \theta_N]^T [x_0, x_1, x_2, \dots, x_N]$
 - where $x_0 = 1$
- Let's make a slight notational adjustment:
- $\theta^T \mathbf{x} = [b, w_1, w_2, \dots, w_N]^T [x_0, x_1, x_2, \dots, x_N]$
- $h_\theta(\mathbf{x}) = h(\theta^T \mathbf{x}) = h(b + w_1 x_1 + w_2 x_2 + \dots + w_N x_N)$
- Thus, for 1-D feature
- $h_\theta(\mathbf{x}) = h(w_1 x_1 + b)$

Logistic Regression

- How do we set θ to best fit the data? This is now logistic regression, not linear regression.
- Using the same probabilistic assumption on the data, fit the parameters via Maximum Likelihood.
- Let's assume
 - $P(y = 1 \mid \mathbf{x}; w, b) = h_{w,b}(\mathbf{x})$
 - $P(y = 0 \mid \mathbf{x}; w, b) = 1 - h_{w,b}(\mathbf{x})$
- More compactly
 - $P(y \mid \mathbf{x}; w, b) = (h_{w,b}(\mathbf{x}))^y (1 - h_{w,b}(\mathbf{x}))^{1-y}$

Logistic Regression

- With m training examples assumed to be generated independently, we can then write down the likelihood of the parameters as

$$\begin{aligned}L(w, b) &= p(y|X; w, b) = \prod_{i=1}^m p(y^{(i)}|\vec{x}^{(i)}; w, b) \\&= \prod_{i=1}^m \left(h_{w,b}\left(\vec{x}^{(i)}\right)\right)^{y^{(i)}} \left(1 - h_{w,b}\left(\vec{x}^{(i)}\right)\right)^{1-y^{(i)}}\end{aligned}$$

The objective here is to maximize the likelihood by w and b to match the occurrence of data composed of (x, y) pairs.

- As before, it is easier to maximize the log likelihood

$$l(w, b) = \log L(w, b) = \sum_{i=1}^m \left(y^{(i)} \log h_{w,b}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log (1 - h_{w,b}(\mathbf{x}^{(i)})) \right)$$

Logistic Regression

- How do we maximize the likelihood?
- As in linear regression, use gradient ascent.
- Written in vectorial notation, our updates will be given by

$$w^{(\text{next step})} = w + \eta \nabla_w \ell(w, b)$$

$$b^{(\text{next step})} = b + \eta \nabla_b \ell(w, b)$$

where η is the learning rate

- Note the positive rather than negative sign in the update formula, since we're maximizing, rather than minimizing, a function now.

Logistic Regression

- For **batch gradient ascent**, we need to sum up for all the data points as

$$l(w, b) = \log L(w, b) = \sum_{i=1}^m \left(y^{(i)} \log h_{w,b}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log (1 - h_{w,b}(\mathbf{x}^{(i)})) \right)$$

- Alternatively, taking one training example (x, y) and immediately take derivatives to update w and b is called **stochastic gradient ascent** rule:
- For stochastic gradient ascent, the **summation goes away** as we only have to deal with one data point

$$l(w, b) = y^{(i)} \log h_{w,b}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log (1 - h_{w,b}(\mathbf{x}^{(i)}))$$

- Let's substitute **sigmoid** function

$$l(w, b) = y^{(i)} \log \frac{1}{1 + e^{-(w^T \mathbf{x} + b)}} + (1 - y^{(i)}) \log \left(1 - \frac{1}{1 + e^{-(w^T \mathbf{x} + b)}} \right)$$

Logistic Regression

- For compactness $\ell(w, b) = y^{(i)} \log[g(w^T x + b)] + (1 - y^{(i)}) \log[1 - g(w^T x + b)]$
- where $g(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$
- Computing gradient of the log likelihood w.r.t. w_j $\frac{\partial}{\partial w_j} \ell(w, b) = \left(y \frac{1}{g(w^T x + b)} - (1 - y) \frac{1}{1 - g(w^T x + b)} \right) \frac{\partial}{\partial \theta_j} g(w^T x + b)$
- But recall the derivative of g $g'(z) = g(z)(1 - g(z))$
- Thus $\frac{\partial}{\partial w_j} \ell(w, b) = \left(y \frac{1}{g(w^T x + b)} - (1 - y) \frac{1}{1 - g(w^T x + b)} \right) g(w^T x + b)(1 - g(w^T x + b)) \frac{\partial}{\partial w_j}(w^T x + b)$

Logistic Regression

- Continue calculating the gradient

$$\begin{aligned}\frac{\partial}{\partial w_j} \ell(w, b) &= \left(y \left(1 - g(w^T x + b) \right) - (1 - y) g(w^T x + b) \right) x_j \\ &= (y - h_{w,b}(x)) x_j\end{aligned}$$

- Similarly for b

$$\begin{aligned}\frac{\partial}{\partial b} \ell(w, b) &= \left(y \frac{1}{g(w^T x + b)} - (1 - y) \frac{1}{1 - g(w^T x + b)} \right) \frac{\partial}{\partial \theta_j} g(w^T x + b) \\ &= \left(y \frac{1}{g(w^T x + b)} - (1 - y) \frac{1}{1 - g(w^T x + b)} \right) g(w^T x + b) (1 - g(w^T x + b)) \\ &= (y (1 - g(w^T x + b)) - (1 - y) g(w^T x + b)) \\ &= (y - h_{w,b}(x))\end{aligned}$$

Logistic Regression

- Therefore

$$\frac{\partial}{\partial w_j} \ell(w, b) = (y - h_{w,b}(x)) x_j$$

- And

$$\frac{\partial}{\partial b} \ell(w, b) = (y - h_{w,b}(x))$$

- Thus the update rules according to stochastic gradient ascent for w_j and b are

$$w_j = w_j + \eta (y - h_{w,b}(x)) x_j$$

$$b = b + \eta (y - h_{w,b}(x))$$

Logistic Regression

- For 1D case

$$w_1 = w_1 + \eta \left(y - h_{w,b} (w_1 x + b) \right) x = w_1 + \eta \left(y - \frac{1}{1 + e^{-(w_1 x + b)}} \right) x$$

$$b = b + \eta \left(y - h_{w,b} (x) \right) = b + \eta \left(y - \frac{1}{1 + e^{-(w_1 x + b)}} \right)$$

- Applying stochastic gradient ascent with updating the gradient at the i^{th} data

$$w_1 = w_1 + \eta \left(y^{(i)} - \frac{1}{1 + e^{-(w_1 x^{(i)} + b)}} \right) x^{(i)}$$

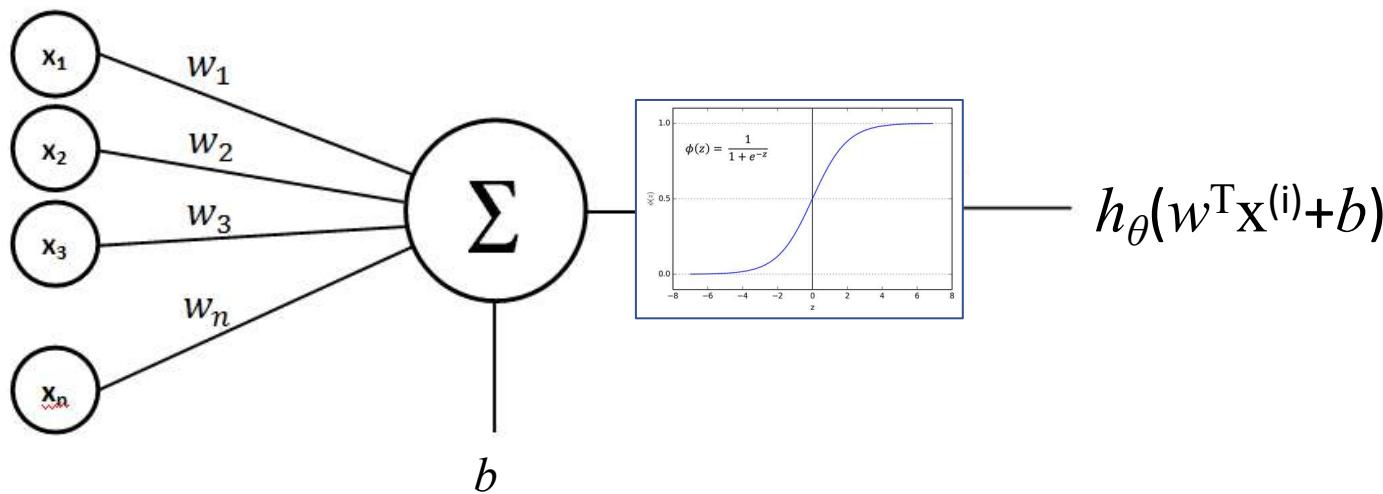
$$b = b + \eta \left(y^{(i)} - \frac{1}{1 + e^{-(w_1 x^{(i)} + b)}} \right)$$

Logistic Regression

- Compared to the update rule of the linear regression, it looks identical. However this is not the same since $h_{\theta}(\mathbf{x}^{(i)})$ is a non-linear function of $\theta^T \mathbf{x}^{(i)}$
- Now consider a logistic regression problem of multidimensional input features

$$h_{w,b}(\vec{\mathbf{x}}) = g(b + w_1 x_1 + w_2 x_2 + \dots + w_N x_N)$$

- The multidimensional logistic regression in pictorial representation

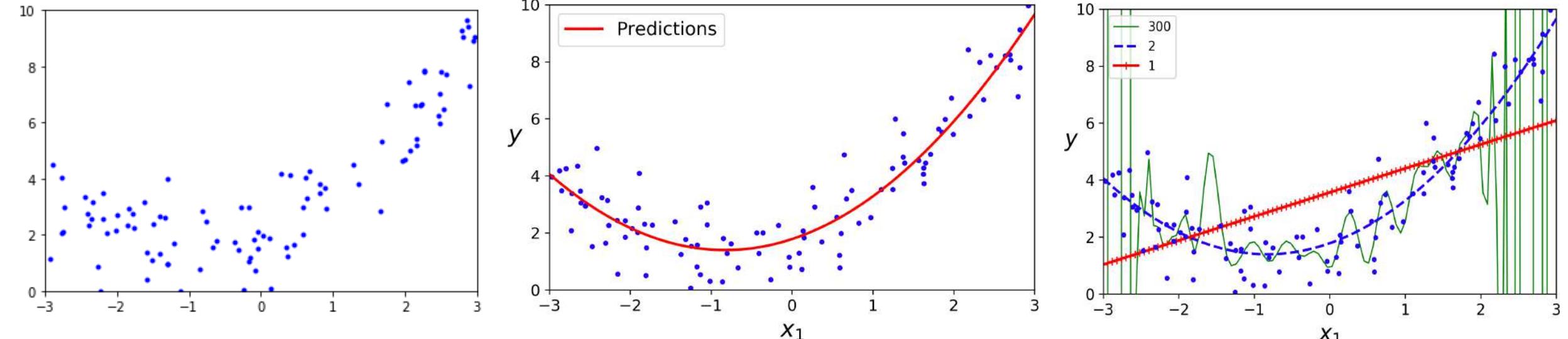


Now, this looks like
a neural network!

Logistic Regression

$$h_{w,b}(\vec{x}) = g(b + w_1x_1 + w_2x_2 + \dots + w_Nx_N)$$

- Not all the input feature vector may contribute equally to the target value
- Forcing the algorithm to fit the data to N-dimensional model may result in overfitting



Regularization

- Regularization: to **constrain overfitting** of a model
- **Ridge Regression** (Tikhonov regularization)
 - A regularization term $\alpha \sum_{i=1}^n w_i^2$ subtracted from the likelihood function.
 - Note that the bias term b is not regularized. It can be added but its influence is typically small, so it is not considered in regularization
 - The terms are subtracted **during training only**. The **hyperparameter** α controls how much to regularize.
 - For $\alpha=0$ the model becomes **logistic regression**

$$J(\theta) = \ell(\theta) - \alpha \frac{1}{2} \sum_{i=1}^n w_i^2$$

Regularization

- Thus the regularized log likelihood is

$$\ell(w, b) = \sum_{i=1}^m y^{(i)} \log h_{w,b}(\vec{x}^{(i)}) + (1 - y^{(i)}) \log (1 - h_{w,b}(\vec{x}^{(i)})) - \frac{\alpha}{2} \|w\|^2$$

- with the regularization term, update rule for the gradient ascent is

$$w_j = w_j + \eta (y^{(i)} - h_{w,b}(x^{(i)})) x_j^{(i)} - \alpha w_j$$
$$b = b + \eta (y^{(i)} - h_{w,b}(x^{(i)}))$$

- Turning into Batch Gradient **Ascent** of $J(w^T \mathbf{x} + b)$

- Uses entire training set to take an iteration

$$w_j = w_j - \eta \left[\sum_{i=1}^m (h_{w,b}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \alpha w_j \right]$$

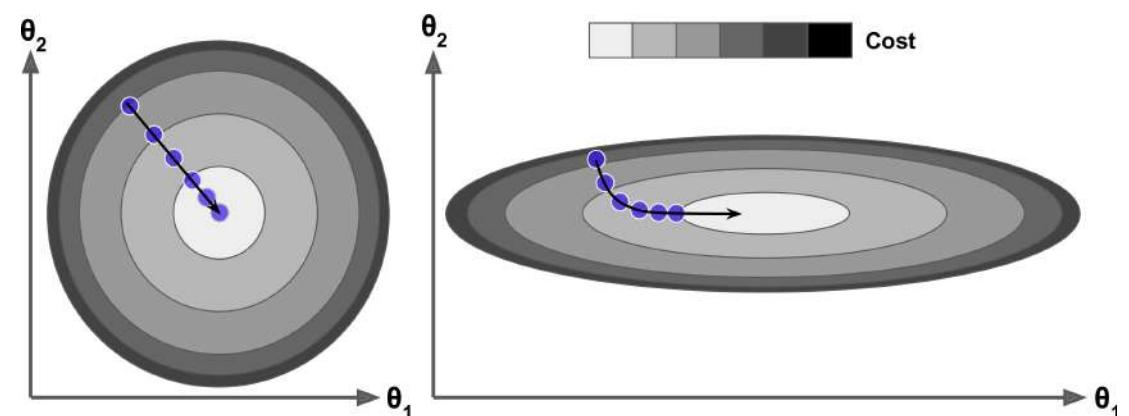
$$b = b - \eta \left[\sum_{i=1}^m (h_{w,b}(x^{(i)}) - y^{(i)}) \right]$$

Standard Pre-processing

- Most regularized models are **sensitive** to input feature **scales**. Thus, it is important to **scale the data** (e.g., using a StandardScaler) before performing Ridge Regression.
- Rescale each feature \mathbf{x} to have zero mean and unit variance

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \text{and} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m [x_j^{(i)} - \mu_i]^2$$

- Rescaling and zero mean
$$X_j^{(i)} = \frac{x_j^{(i)} - \mu_i}{\sigma_j}$$
- May omit preprocessing if the features are on comparable scales.
 - e.g. $x_1 \in [-5, 5]$, $x_2 \in [-3, 3]$
- However, if $x_1 \in [0, 1000]$, $x_2 \in [0, 1]$
- Gradient descent will be very slow



Introduction to Artificial Neural Networks (ANN)

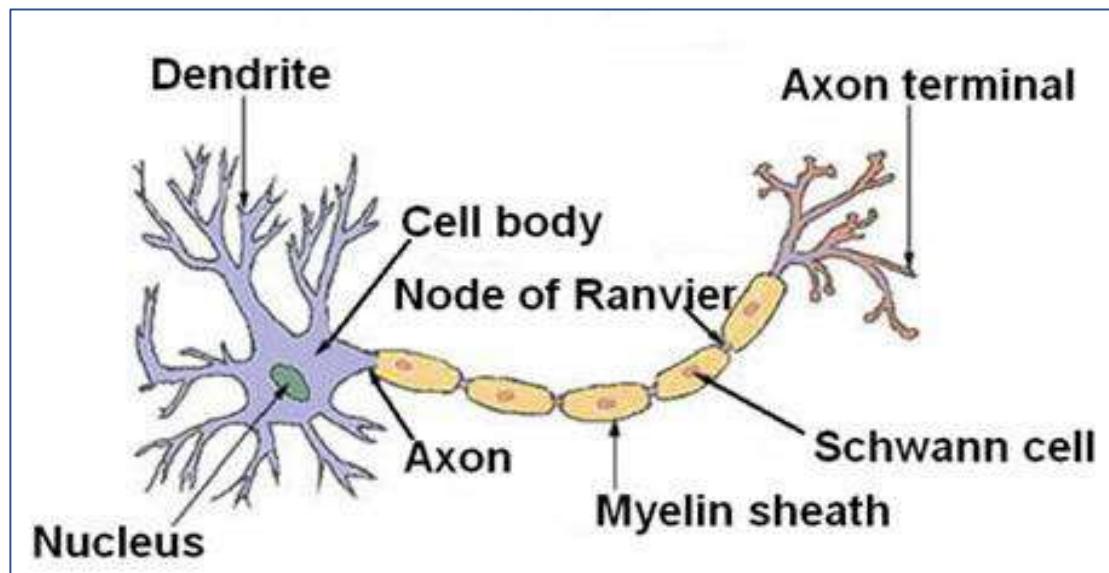
ECE 610

David Han

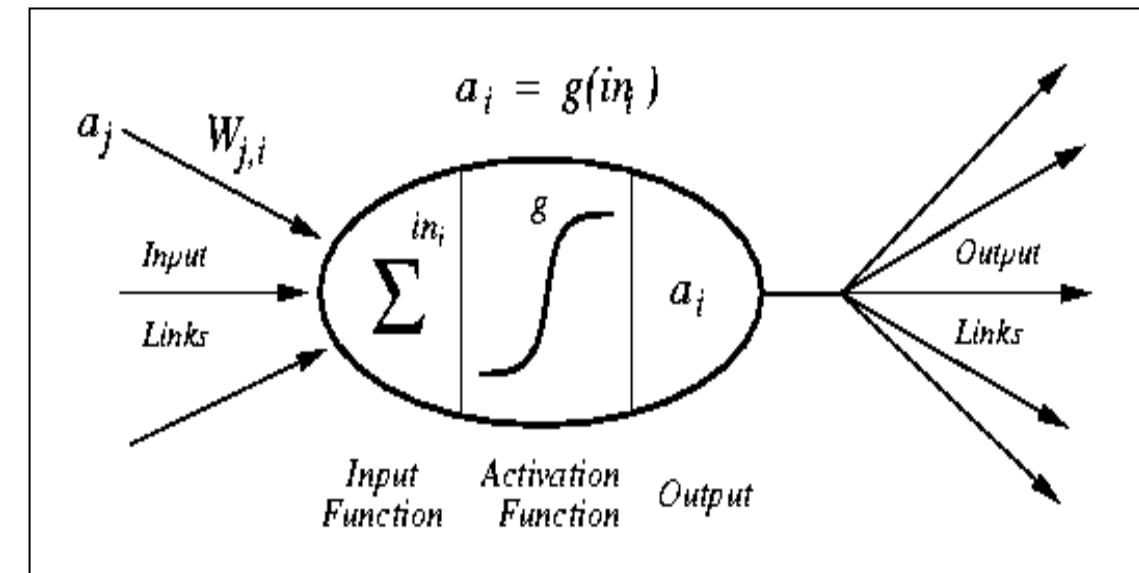
Drexel University

Artificial Neural Networks (ANN)

- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Huge Comeback in 2010s: State-of-the-art technique for many applications



Biological Neuron

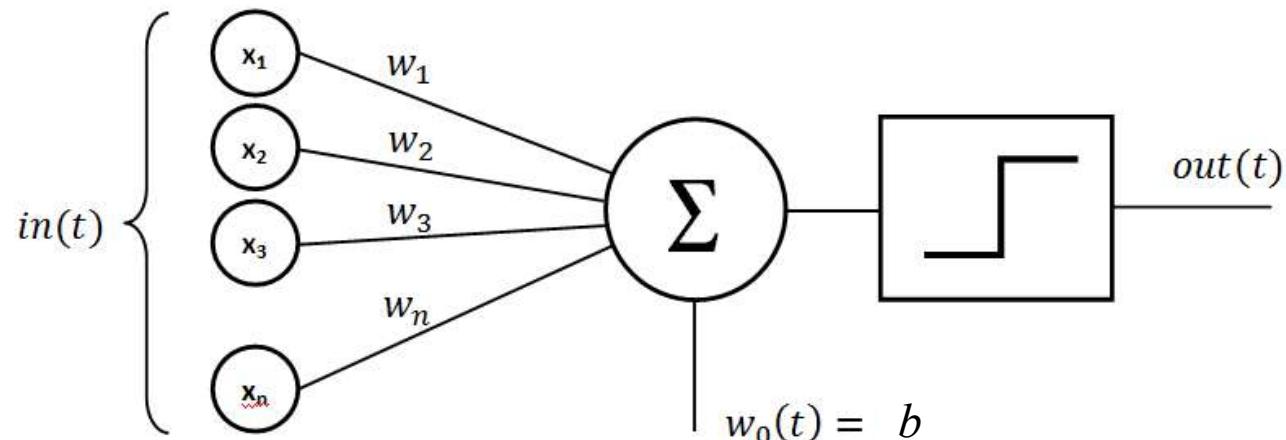


Artificial Neuron

Perceptron

- The **simplest** of Artificial Neural Network (ANN) architectures
- Invented in 1957 by **Frank Rosenblatt**
- Also called **Threshold Logic Unit (TLU)**
- TLU takes **weighted inputs** and **sums** them
- Produces a binary output as a **Heaviside step function (threshold function)**:

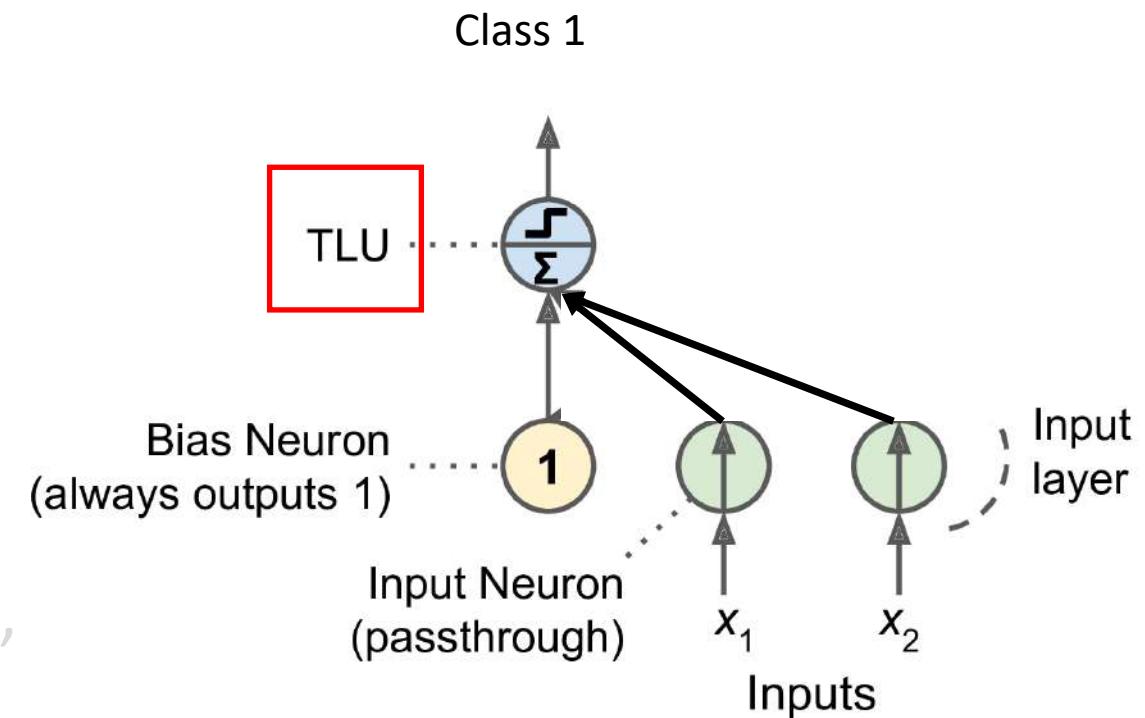
$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b = \mathbf{x}^T \mathbf{w} + b$$



$$O = \begin{cases} 1 : \left(\sum_i w_i x_i \right) + b > 0 \\ 0 : \text{otherwise} \end{cases}$$

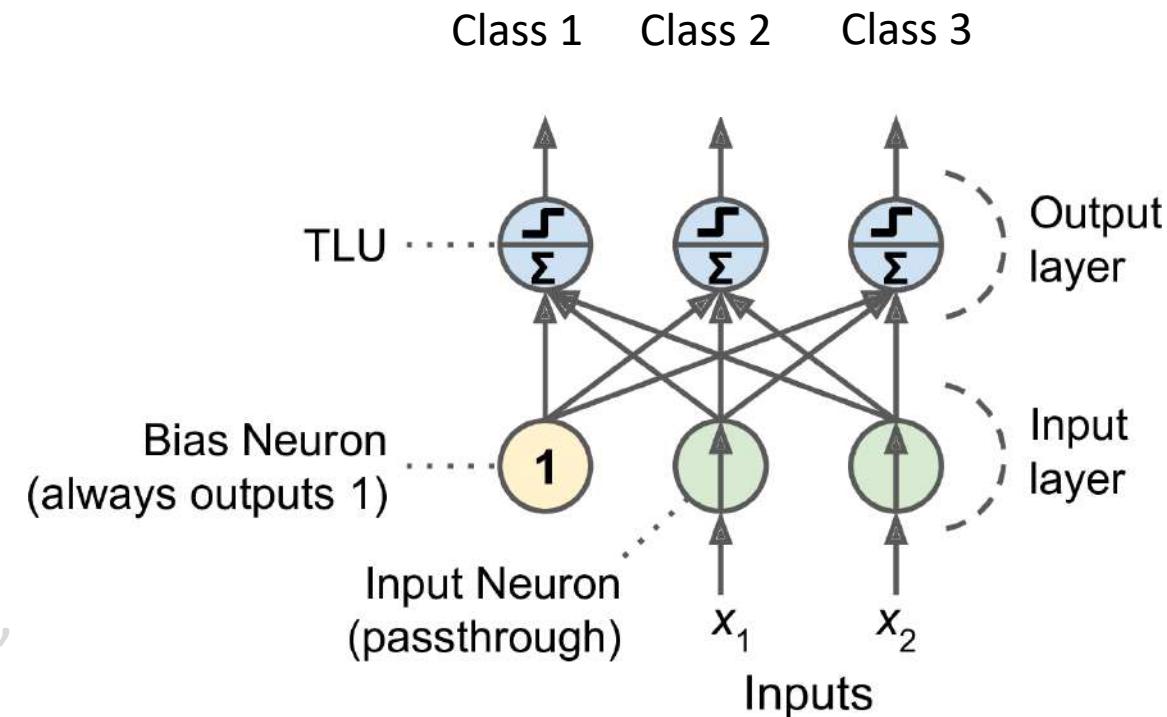
Perceptron

- Can be used for a simple **linear binary classification**
- Composed of a **single layer** of TLUs
- Multiclass classification possible by input connected to multiple TLUs
- Each TLU connected to all the inputs
- “Fully connected layer” or a “dense layer” means when all the neurons in a layer are connected to every neuron in the previous layer.



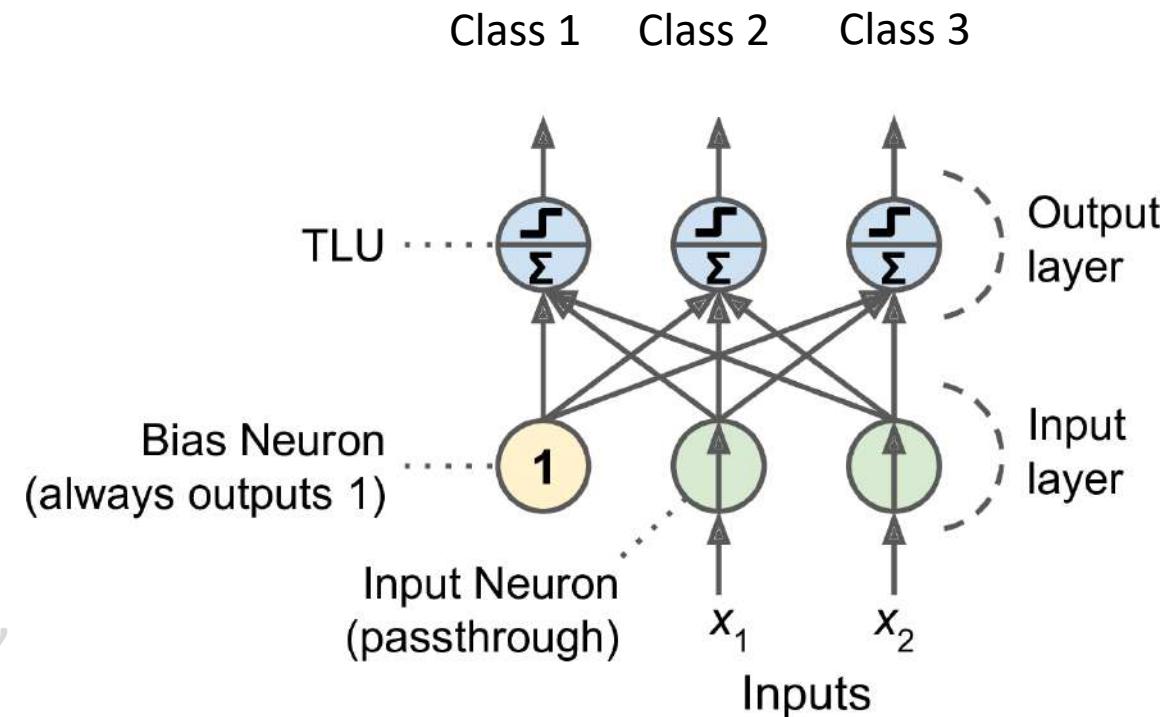
Perceptron

- Can be used for a simple **linear binary classification**
- Composed of a **single layer** of TLUs
- **Multiclass** classification possible by input connected to **multiple TLUs**
- Each TLU connected to all the inputs
- “Fully connected layer” or a “dense layer” means when all the neurons in a layer are connected to every neuron in the previous layer.



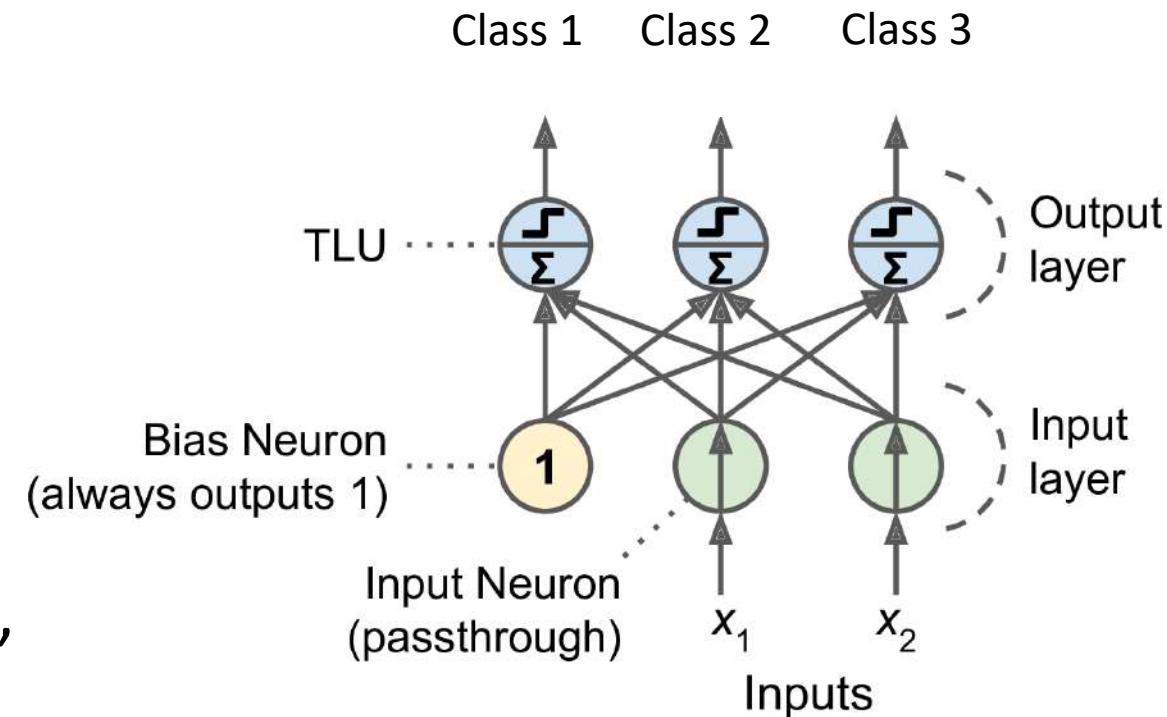
Perceptron

- Can be used for a simple **linear binary classification**
- Composed of a **single layer** of TLUs
- **Multiclass** classification possible by input connected to **multiple TLUs**
- Each TLU connected to **all the inputs**
- “Fully connected layer” or a “dense layer” means when all the neurons in a layer are connected to every neuron in the previous layer.

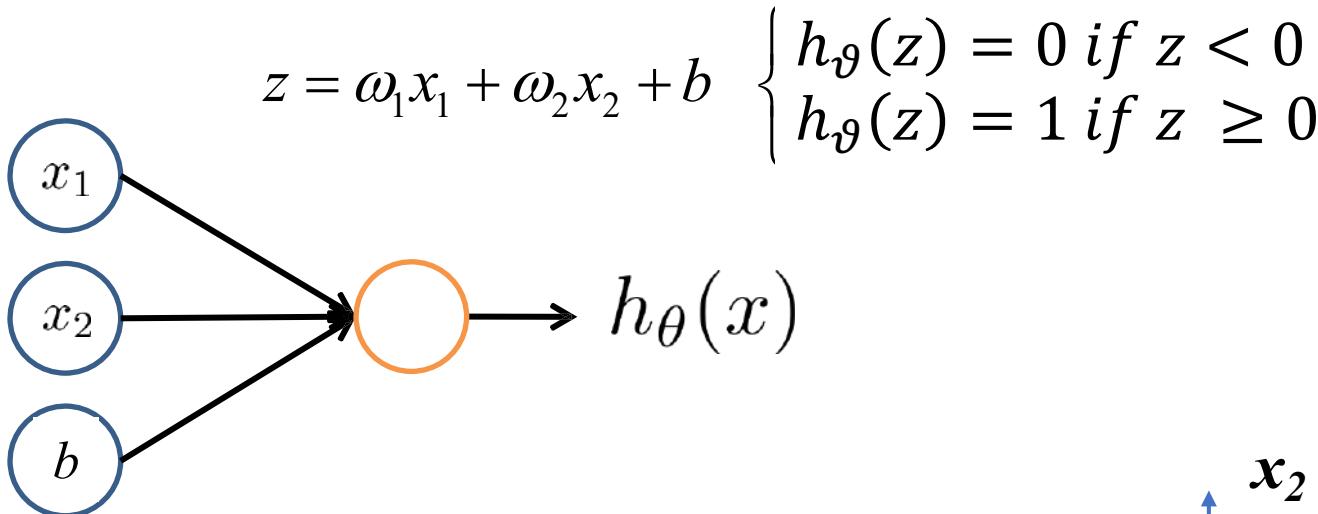


Perceptron

- Can be used for a simple **linear binary classification**
- Composed of a **single layer** of TLUs
- **Multiclass** classification possible by input connected to **multiple TLUs**
- Each TLU connected to **all the inputs**
- “**Fully connected layer**” or a “**dense layer**” means when all the neurons in a layer are **connected to every neuron** in the previous layer.



Perceptron



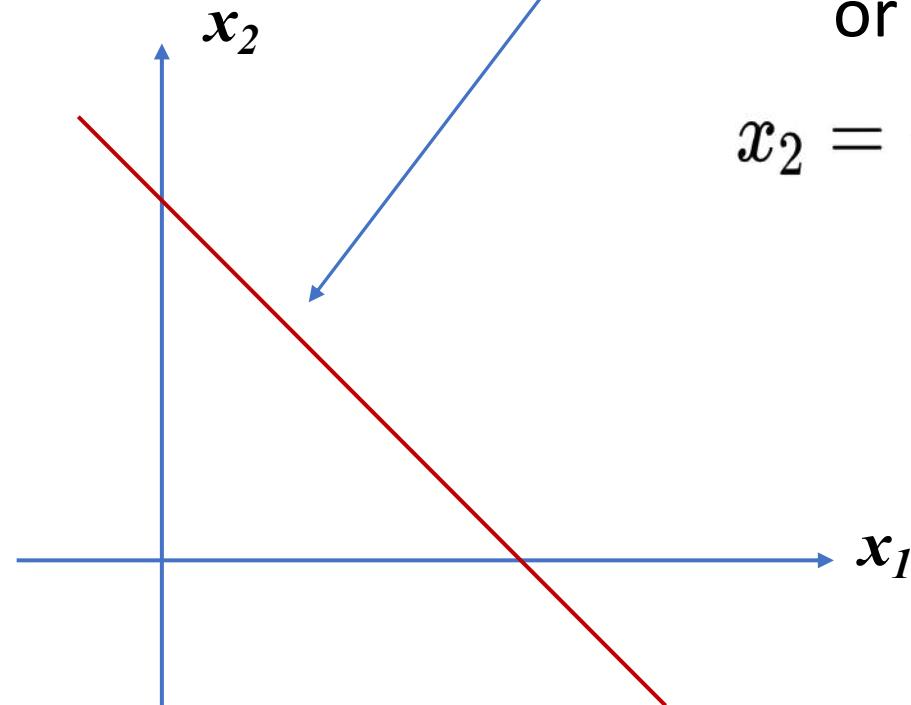
- Recall an **equation of line** in 2D space from **linear algebra**.

- Thus, $z=0$ forms a decision boundary.

$$w_1 x_1 + w_2 x_2 + b = 0$$

or

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$



Perceptron

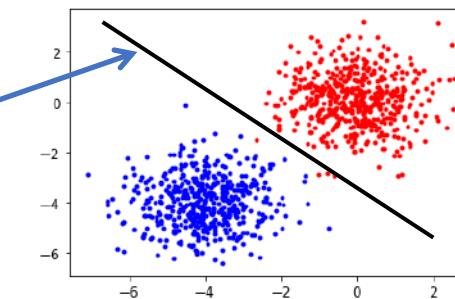
- Consider a linearly separable dataset
- A properly drawn line of equation would serve as the classification **decision boundary**.
- The classifier has to provide a **binary output** while taking the input value of z .
- Thus an **activation function** h needs to behave as.

$$h(z) = \begin{cases} 0 & \text{if } w_1x_1 + w_2x_2 + b < 0 \\ 1 & \text{if } w_1x_1 + w_2x_2 + b \geq 0 \end{cases}$$

- **Heaviside step function** was chosen

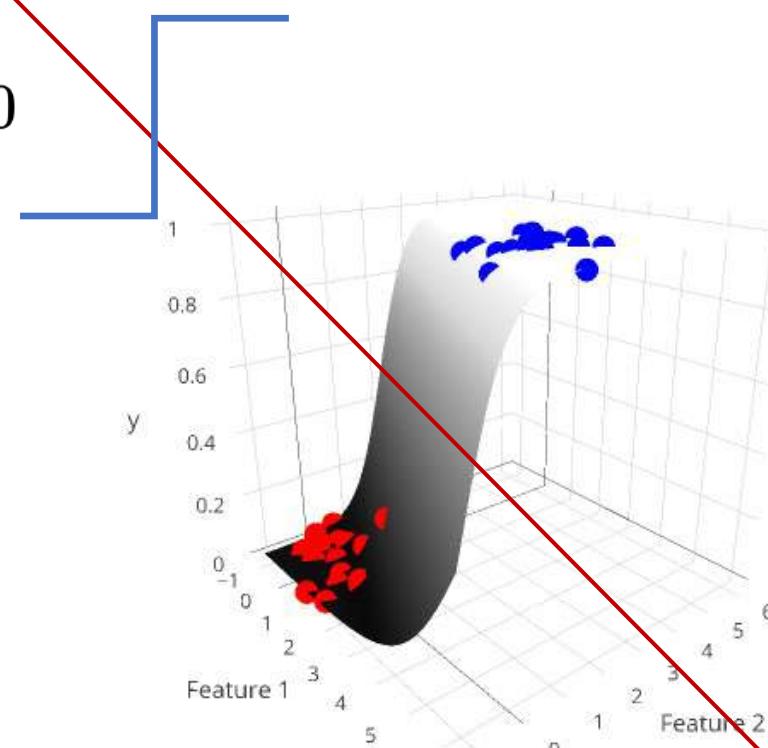
$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

$$w_1x_1 + w_2x_2 + b = 0$$



$$w_1x_1 + w_2x_2 + b \geq 0$$

$$w_1x_1 + w_2x_2 + b < 0$$



Decision Boundary

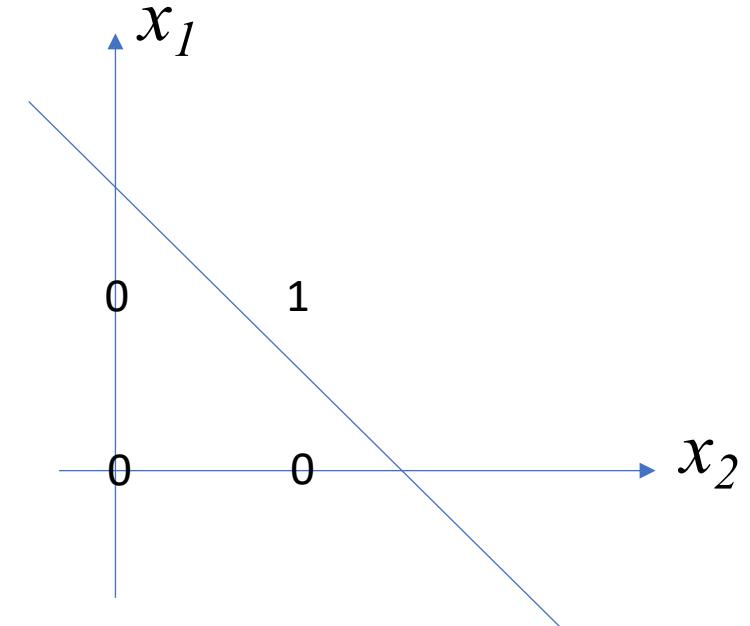
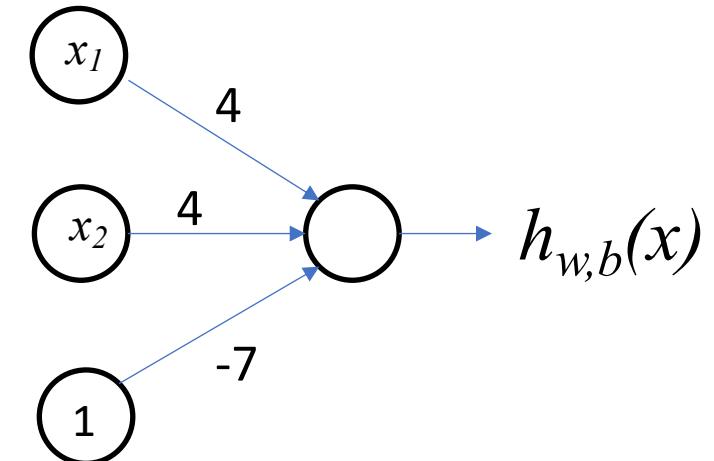
Perceptron

- Perceptron can also do some logical computations
- Let x_1 and x_2 be booleans
- Consider a perceptron with the following weights

$$z = 4x_1 + 4x_2 - 7$$

x_1	x_2	z	$h(z)$
0	0	-7	0
0	1	-3	0
1	0	-3	0
1	1	1	1

This is an “AND” operation



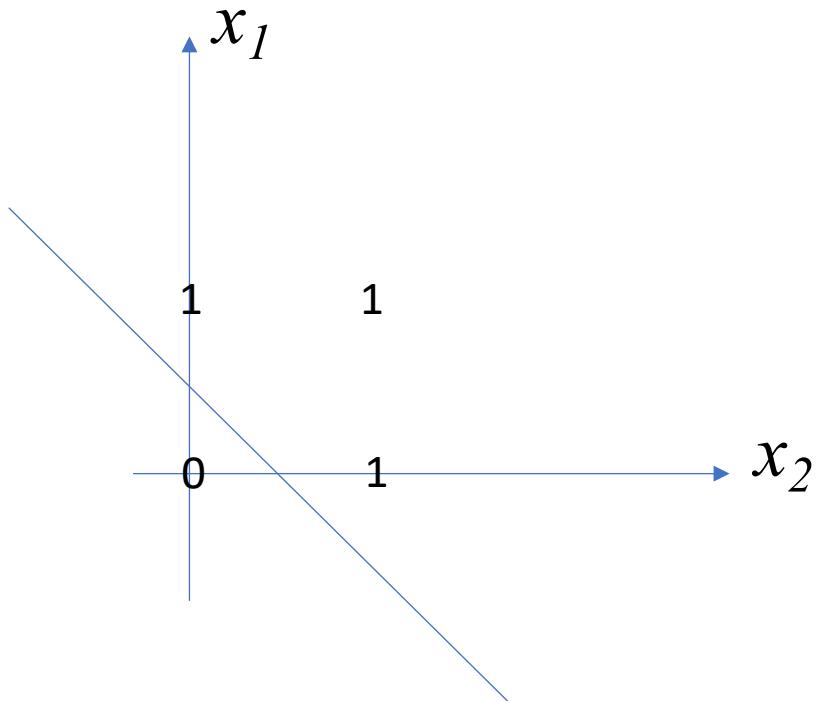
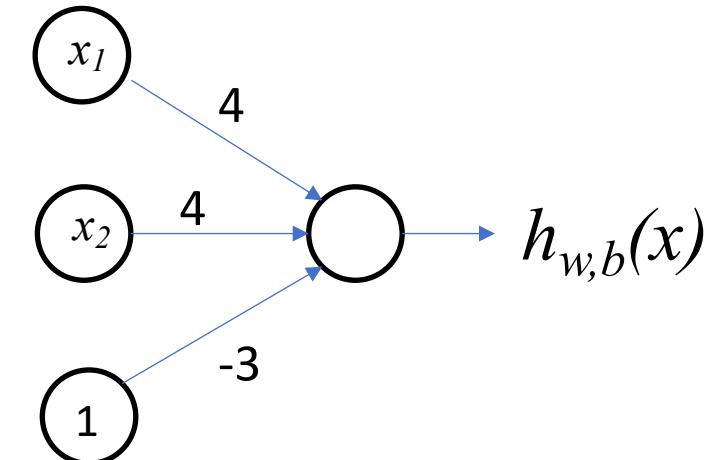
Perceptron

- More logical computations by perceptron

$$z = 4x_1 + 4x_2 - 3$$

x_1	x_2	z	$h(z)$
0	0	-3	0
0	1	1	1
1	0	1	1
1	1	5	1

An “OR” operation



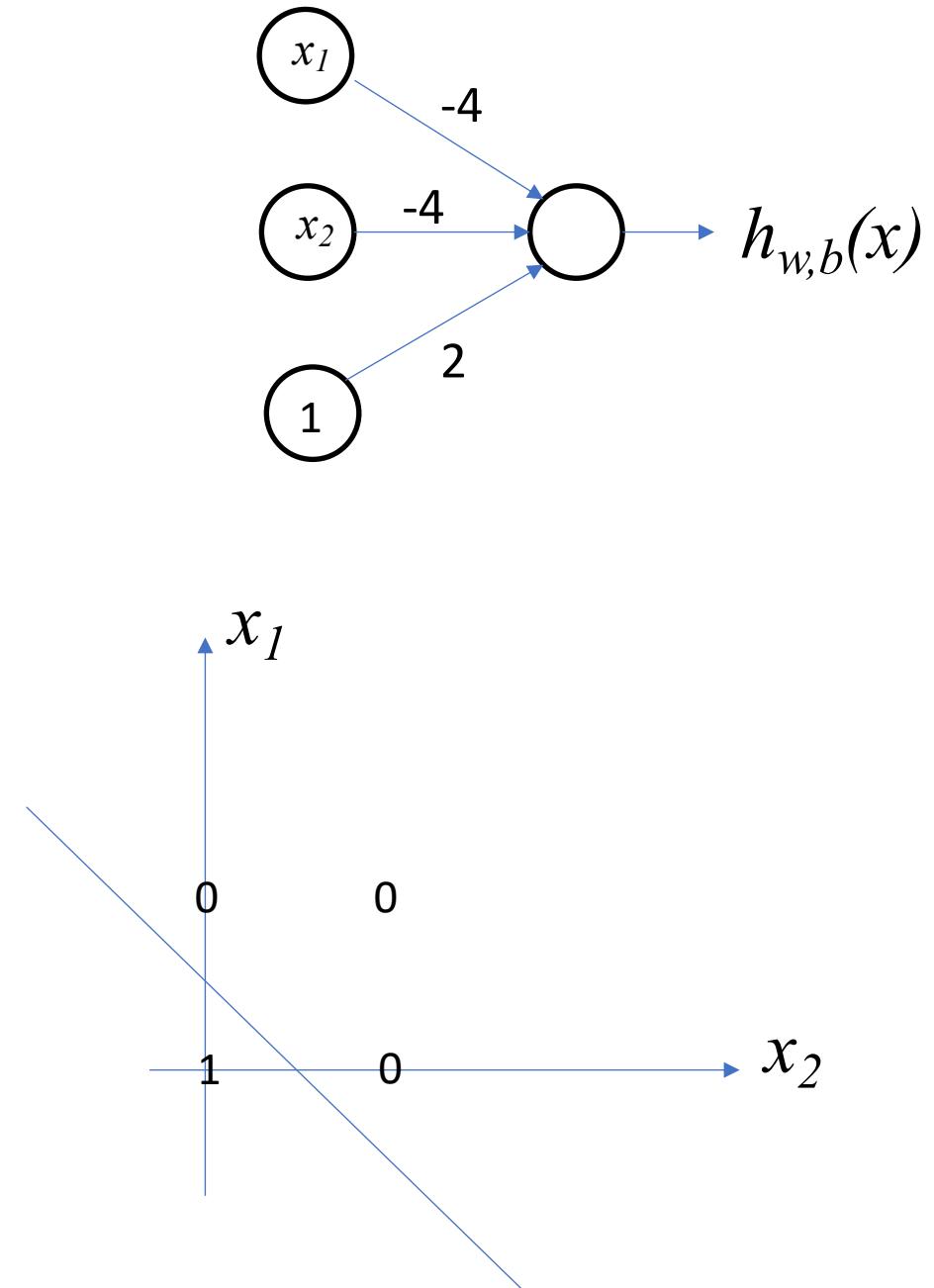
Perceptron

- More logical computations by perceptron

$$z = -4x_1 - 4x_2 + 2$$

x_1	x_2	z	$h(z)$
0	0	2	1
0	1	-2	0
1	0	-2	0
1	1	-6	0

A “NOR” operation



Perceptron Training

- But, how do we find the right w and b ? How do we train perceptron?
- Based on Hebbian learning
 - Originated from Hebb's rule in *The Organization of Behavior*:

$$w_i^{(nextstep)} = w_i^{(current)} + \eta(y - \hat{y})x_i$$

- The learning rule reinforces connections between connected neurons that help reduce the error.

where

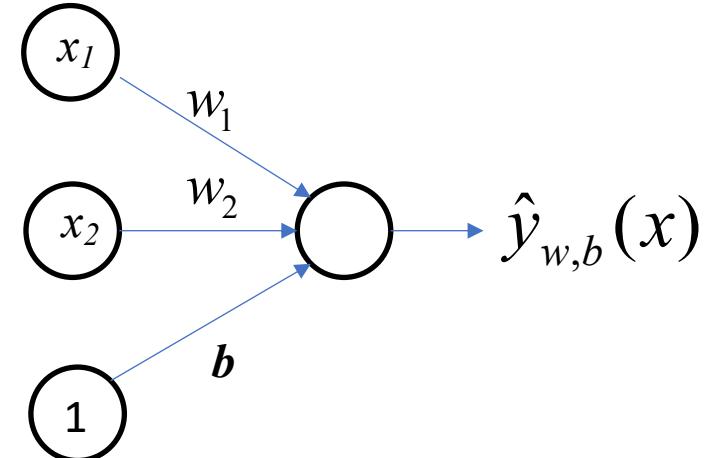
w_i is the connection weight between the i^{th} input neuron and the output neuron.

x_i is the i^{th} input value of the current training instance.

\hat{y} is the output of the output neuron for the current training instance.

y is the target output of the output neuron for the current training instance.

η is the learning rate.



Logistic Regression Update Rule

$$w_j = w_j + \eta(y - h_{w,b}(x))x_j$$

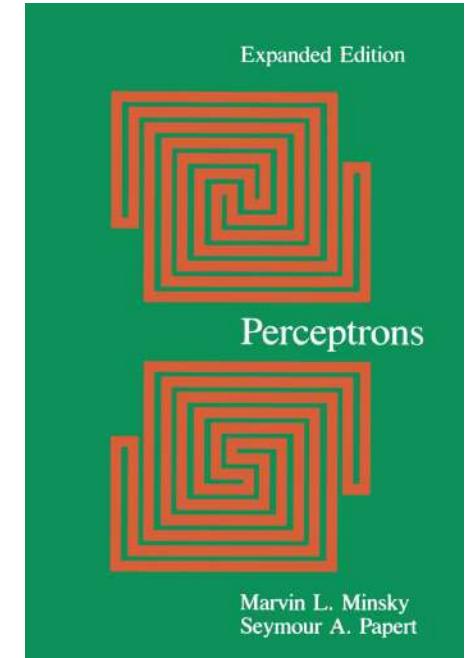
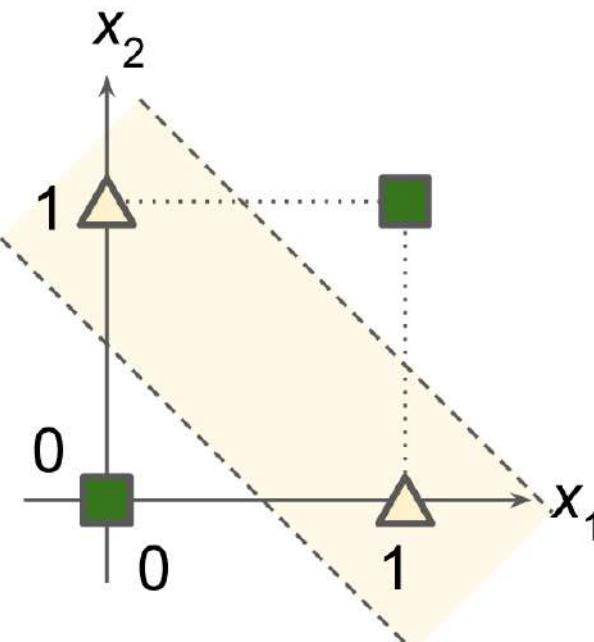
$$b_j = b_j + \eta(y - h_{w,b}(x))$$

Perceptron

- The decision boundary is linear, so perceptron **can't** be applied to **complex** problems
- If the training instances **are linearly separable**, Rosenblatt demonstrated that this algorithm would **converge** to a solution. This is called the **Perceptron convergence theorem**.
- Though it looks similar, perceptron is **different** compared to logistic regression and least square linear regression.
- It is hard to relate its predictions with meaningful probabilistic interpretations
 - Can't show it as a maximum likelihood estimation algorithm.

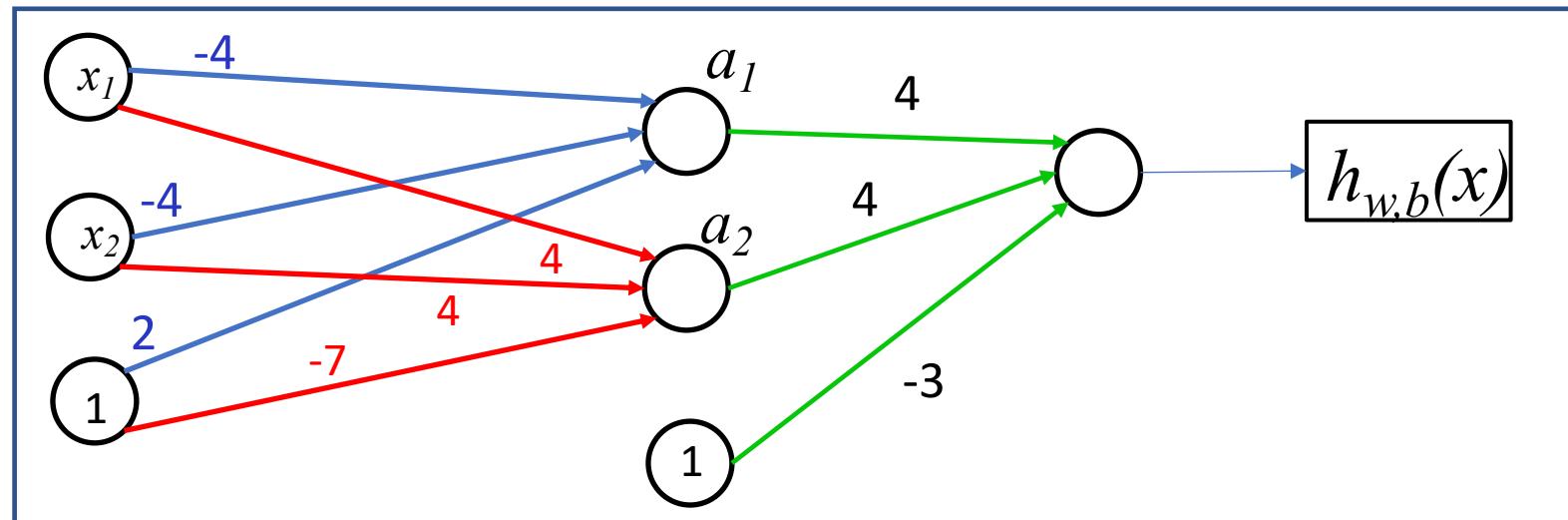
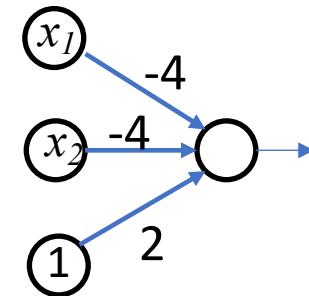
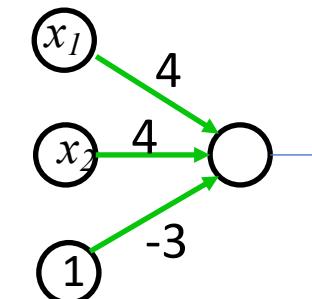
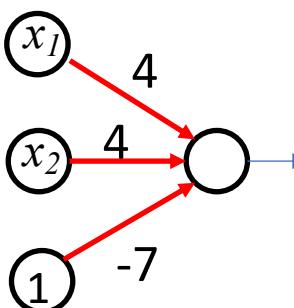
Perceptron

- Death of perceptron:
“**Perceptron**” by Marvin Minsky and Seymour Papert
- Highlighted major weaknesses
 - Incapable of XOR (Exclusive OR)
- Minsky and Papert’s work froze any further advancement of neural network for decades!



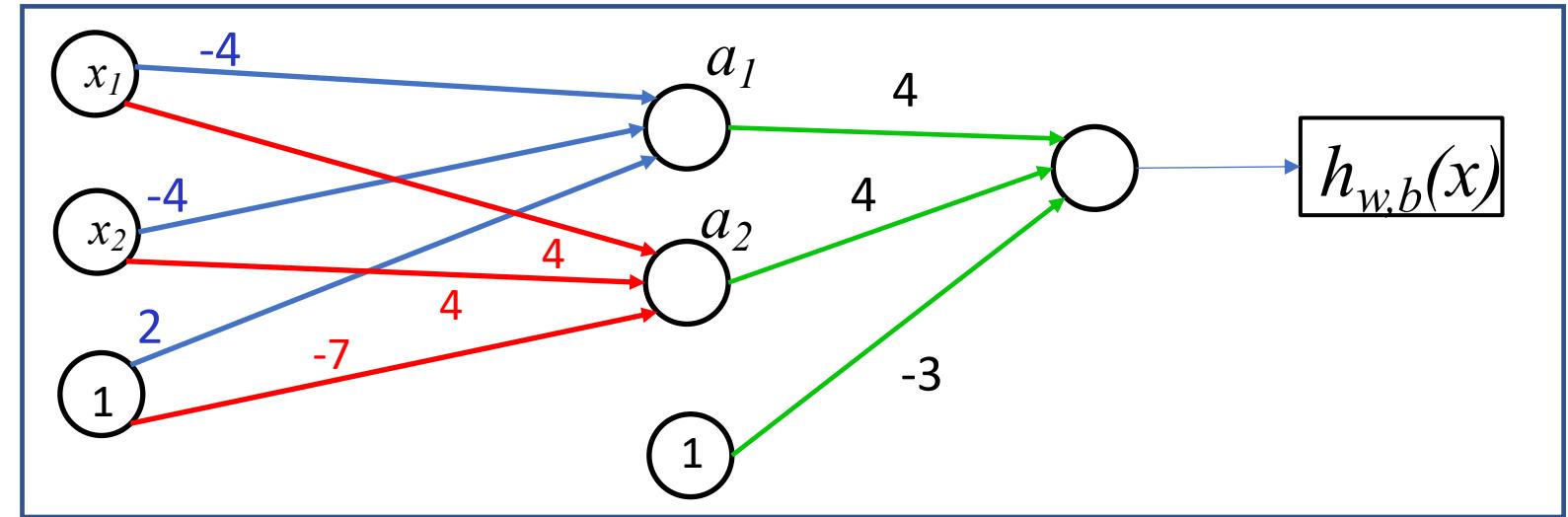
Perceptron

- It turned out a simple addition to perceptron would solve XOR problem.
- Let's combine three of the perceptrons used for logical computations previously

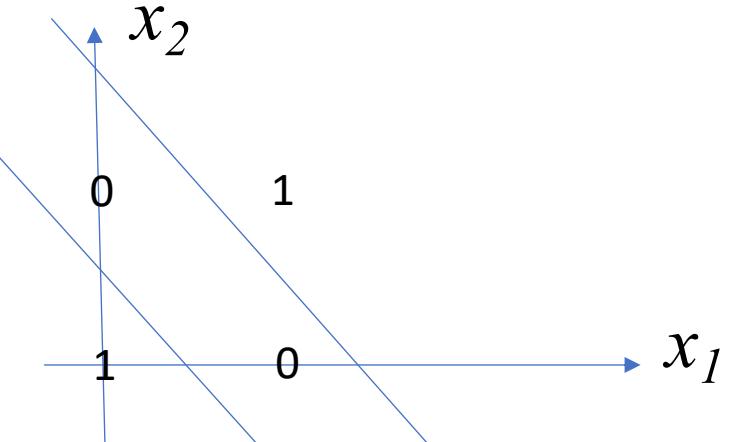


Perceptron

- XOR can be done by a stacked 2 layer perceptron!

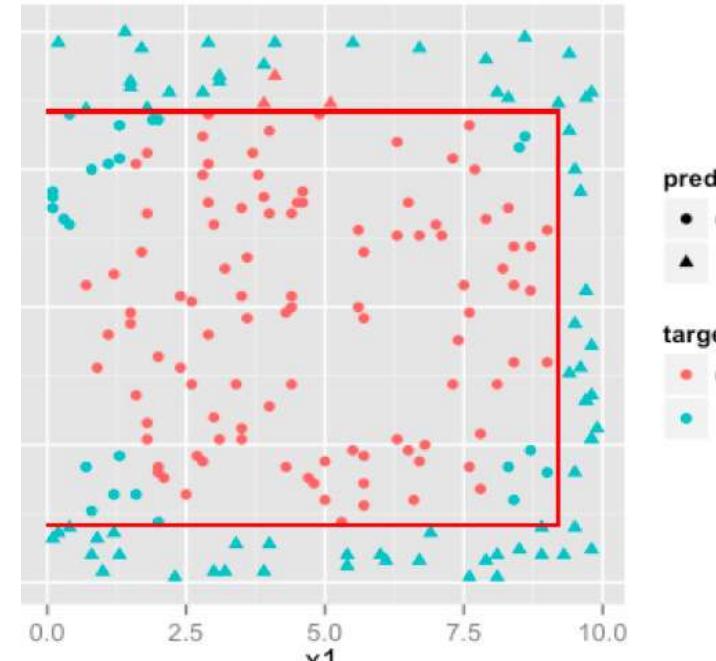
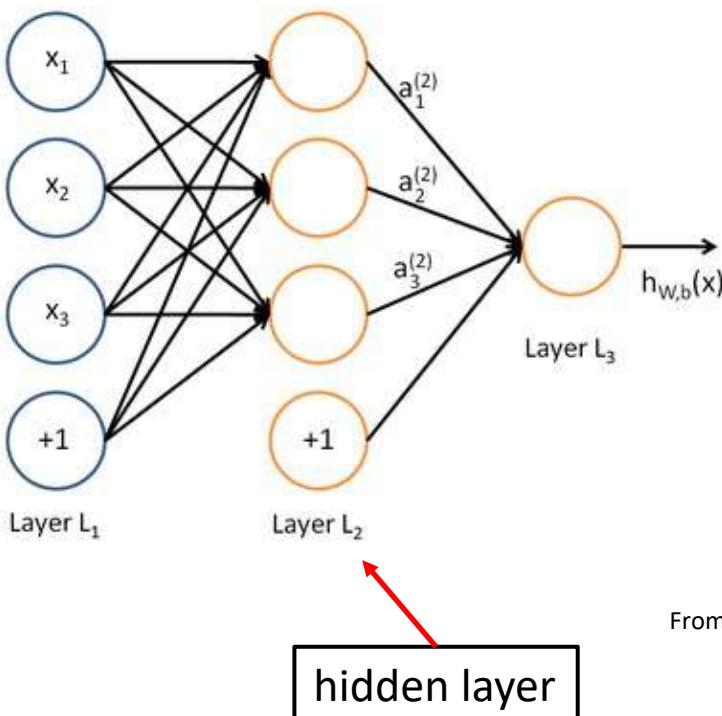


x_1	x_2	a_1	a_2	$h_{w,b}(x)$
0	0	1	0	1
0	1	0	0	0
1	0	0	0	0
1	1	0	1	1

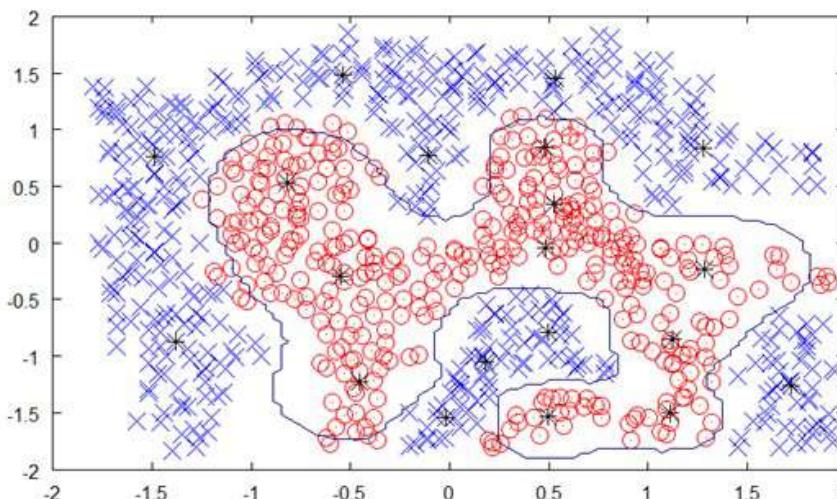


Multi-Layer Perceptron

- Thus, by adding more layers, it is possible to draw more complex decision boundaries.
 - Adding a hidden layer
- Multi-Layer Perceptron (MLP)

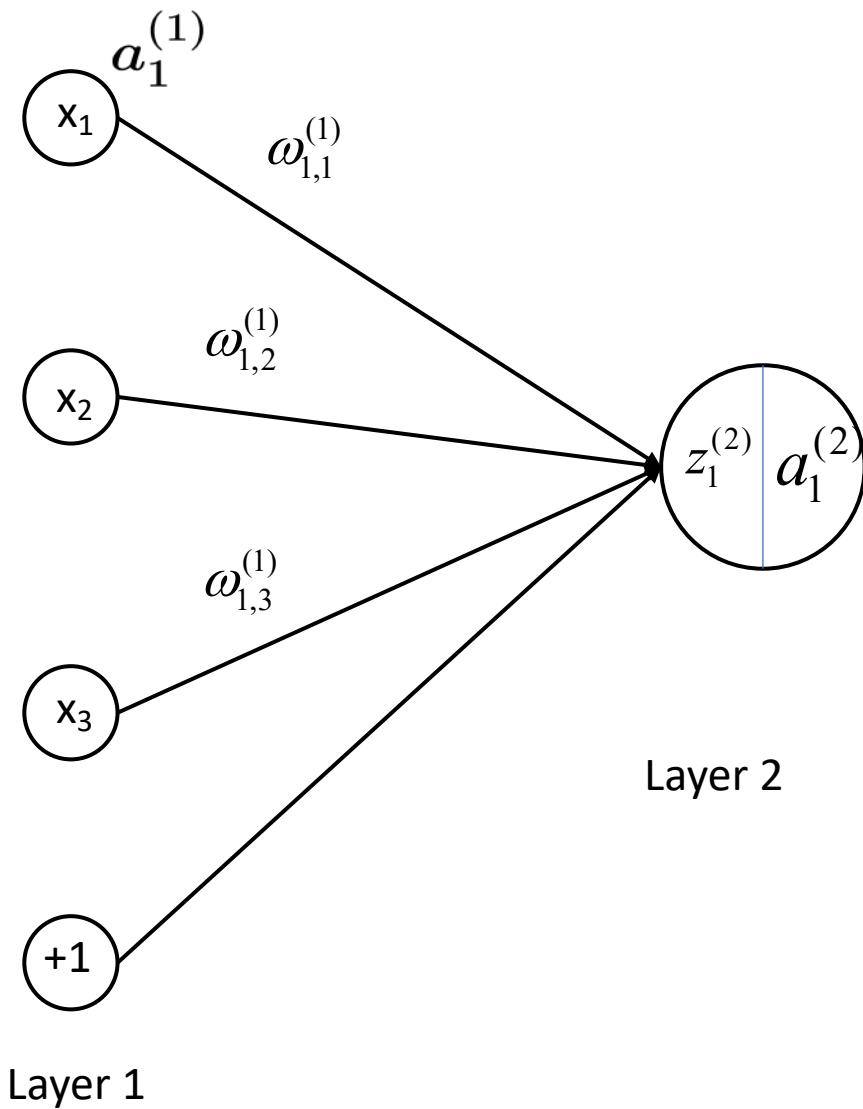


Decision Boundary of Deep Architecture



From: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788295758/6/ch06lvl1sec59/introduction-to-deep-learning

Perceptron Computation



A two step process

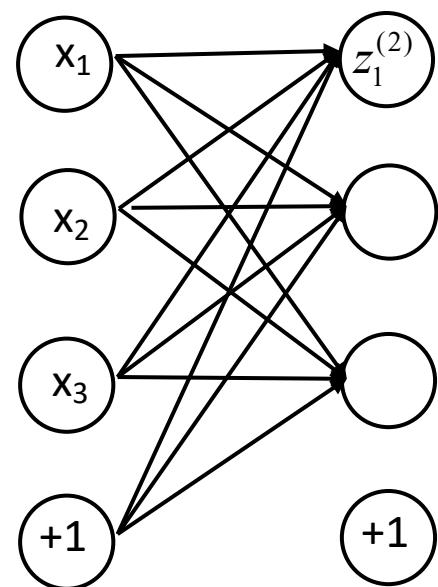
$$z_1^{(2)} = \omega_{1,1}^{(1)}x_1 + \omega_{1,2}^{(1)}x_2 + \omega_{1,3}^{(1)}x_3 + b_1^{(1)}$$

$$a_1^{(2)} = f(z_1^{(2)})$$

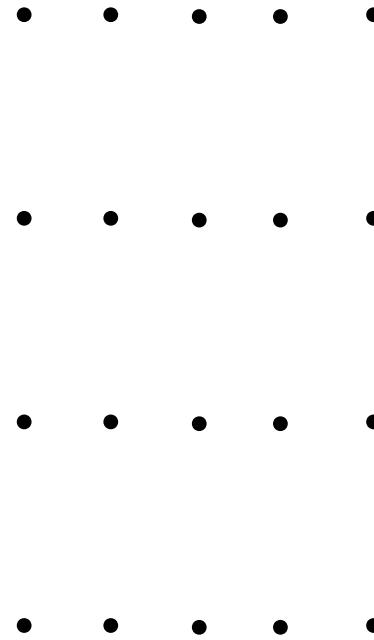
Where f is an activation function

Neural Network Notations

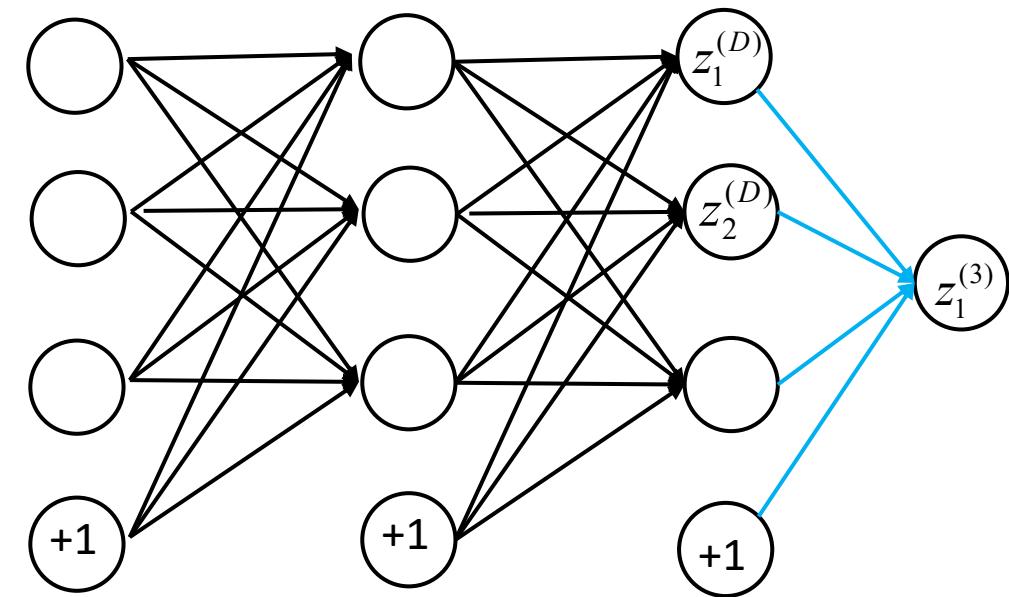
- Multilayer Perceptrons



Layer 1



Layer 2

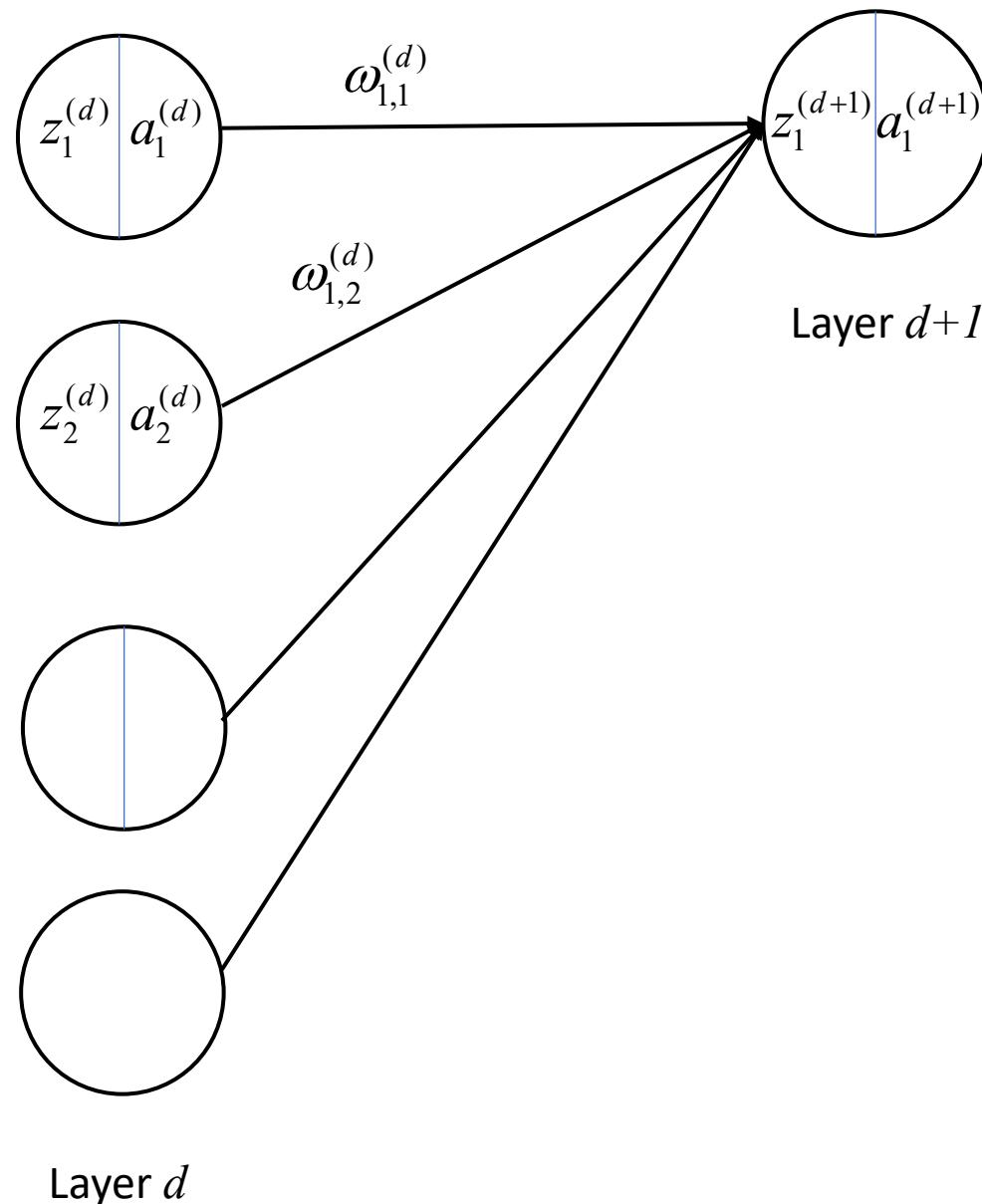


Layer D-2

Layer D-1

Layer D

Neural Network Notations



The term $\omega_{i,j}^{(l)}$ denotes the weight applied to an activation $a_j^{(l)}$ and added to $z_i^{(l+1)}$.

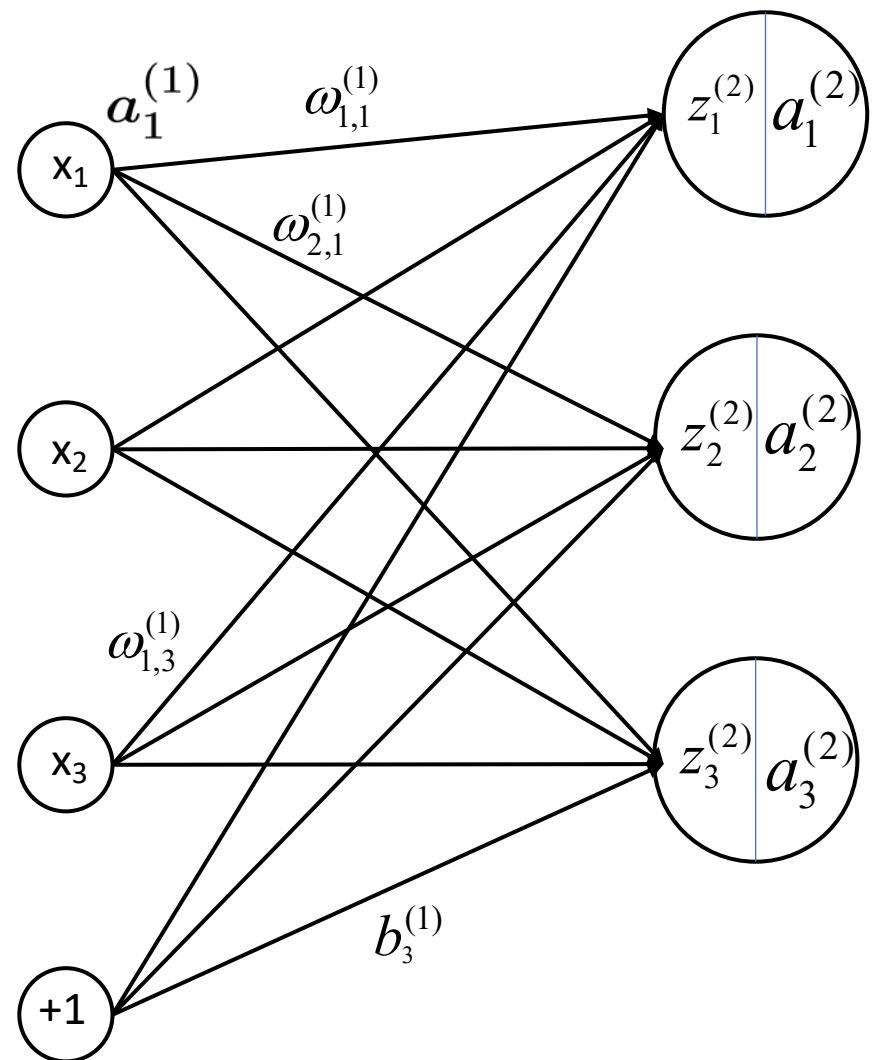
$a_i^{(l)}$ denotes activation of unit i in layer l
for $l = 1$, we use $a_i^{(1)} = x_i$
 $f(..)$ is the activation function such as a sigmoid

$$z_1^{(d+1)} = \omega_{11}^{(d)} a_1^{(d)} + \omega_{12}^{(d)} a_2^{(d)} + \omega_{13}^{(d)} a_3^{(d)} + b_1^{(d)}$$

$$a_1^{(d+1)} = f(z_1^{(d+1)})$$

Layer d

Neural Network Details



Layer 1

Layer 2

For Layers 1 and 2

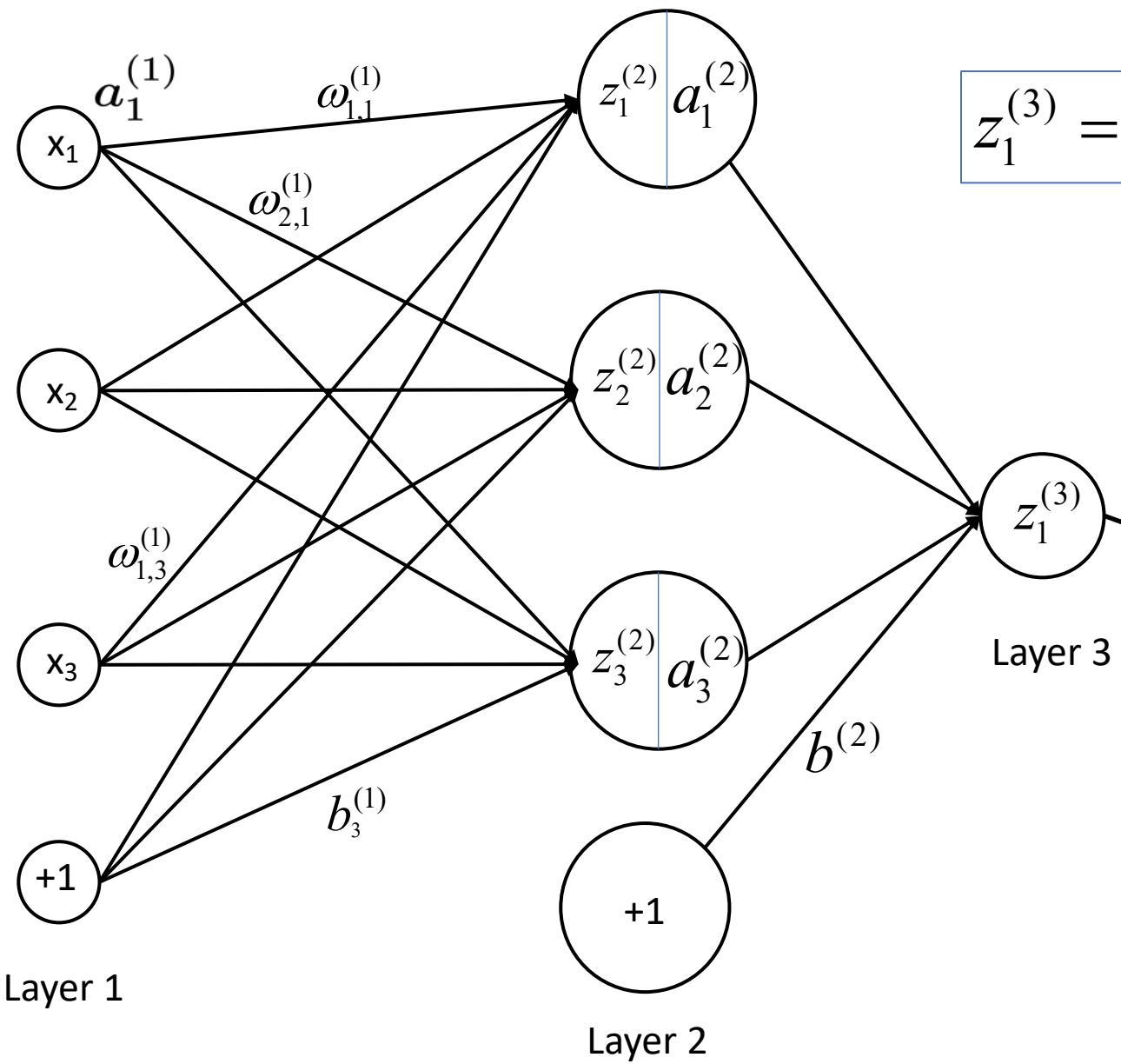
$$z_2^{(2)} = \omega_{21}^{(1)}x_1 + \omega_{22}^{(1)}x_2 + \omega_{23}^{(1)}x_3 + b_2^{(1)}$$

$$a_2^{(2)} = f(z_2^{(2)})$$

$$z_3^{(2)} = \omega_{31}^{(1)}x_1 + \omega_{32}^{(1)}x_2 + \omega_{33}^{(1)}x_3 + b_3^{(1)}$$

$$a_3^{(2)} = f(z_3^{(2)})$$

Neural Network Details



$$z_1^{(3)} = \omega_{11}^{(2)} a_1^{(2)} + \omega_{12}^{(2)} a_2^{(2)} + \omega_{13}^{(2)} a_3^{(2)} + b_1^{(2)}$$

$$h_{W,b} = a_1^{(3)} = f(z_1^{(3)})$$

Layer 3

$$h_{W,b}(x)$$

Layer 1

Layer 2

Multi-Layer Perceptron Training

- Deeper layers seem to allow perceptrons to deal with more complex problems, but how do you train the deep network?
- Recall the perceptron learning rule?

$$w_i^{(nextstep)} = w_i^{(current)} + \eta(y - \hat{y})x_i$$

- What do you do for hidden layers? No y values are available for training on hidden layers!
- In 1986, David Rumelhart, **Geoffrey Hinton** and Ronald Williams introduced the backpropagation training algorithm based on finding weights to minimize the cost function using **Gradient Descent**.
- This requires using a different activation function than the step function, since its **gradient** is mostly zero (flat).

$$\theta^{(next\ step)} = \theta - \eta \nabla_{\theta} J(\theta)$$

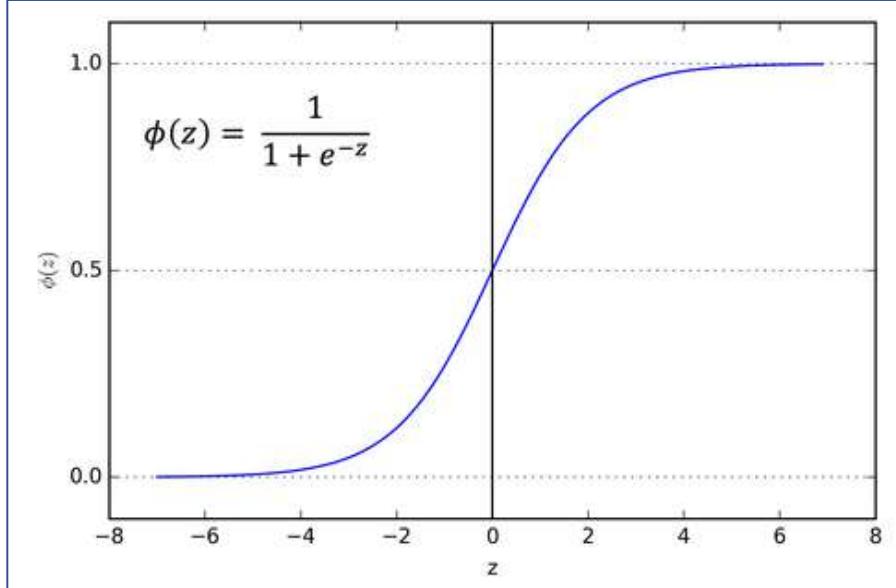
where η is the learning rate

Activation Functions

- Step function has zero derivative everywhere except at $z=0$
- This would make it difficult to perform gradient descent in back propagation
- Rumelhart, et. al changed the **activation function** from step function to **sigmoid function** (logistic function)

- Recall

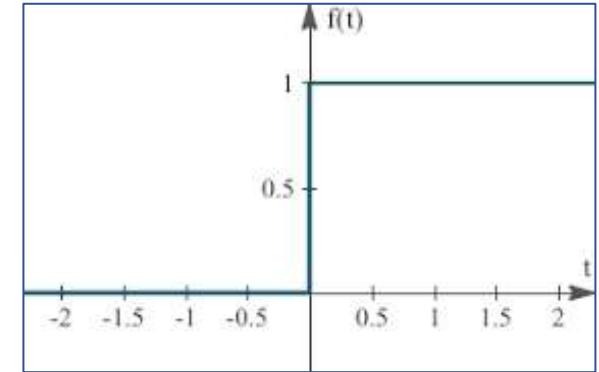
$$\sigma(z) = \frac{1}{(1+\exp(-z))}$$



as $z \rightarrow \infty$, $\sigma(z) \rightarrow 1$
as $z \rightarrow -\infty$, $\sigma(z) \rightarrow 0$
 $\sigma(0) = \frac{1}{2}$

- Sigmoid Function Properties:
- Its derivative exists everywhere!

$$\begin{aligned}g'(z) &= \frac{d}{dz} \frac{1}{1+e^{-z}} = \frac{1}{(1+e^{-z})^2} (e^{-z}) \\&= \frac{1}{(1+e^{-z})} \left(1 - \frac{1}{(1+e^{-z})} \right) = g(z)(1-g(z)) \\g'(\infty) &= 0 \\g'(-\infty) &= 0 \\g'(0) &= \frac{1}{4}\end{aligned}$$

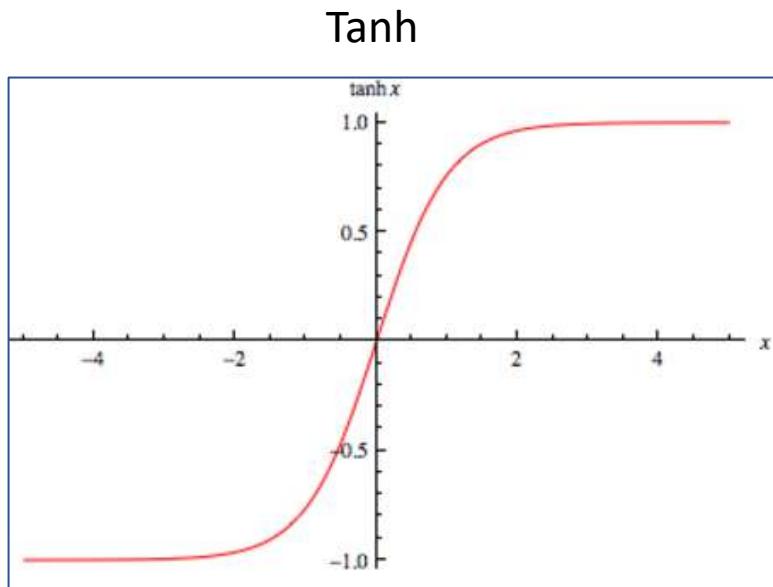


Activation Functions

- Hyperbolic tangent function (Tanh)
 - Continuous, differentiable
 - Range -1 to +1

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

- Hyperbolic tangent Properties:
- The derivative exists everywhere



as $z \rightarrow \infty$, $g(z) \rightarrow 1$
as $z \rightarrow -\infty$, $g(z) \rightarrow -1$
 $g(0) = 0$

$$\begin{aligned} g'(z) &= \frac{d}{dz} \left(\frac{e^z - e^{-z}}{e^z + e^{-z}} \right) = 1 - (\tanh(z))^2 \\ &= 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}} \right)^2 \\ g'(\infty) &= 0 \\ g'(-\infty) &= 0 \\ g'(0) &= 1 \end{aligned}$$

Activation Functions

- **Rectified Linear Unit (ReLU)**

- Continuous, differentiable except at $z=0$
- Fast to compute
- Derivative constant at $z >> 0 \rightarrow$ alleviates vanishing gradient problem in deep network

$$a(z) = \max(0, z)$$

- ReLU Properties:
- Its gradient exists everywhere except at $z=0$

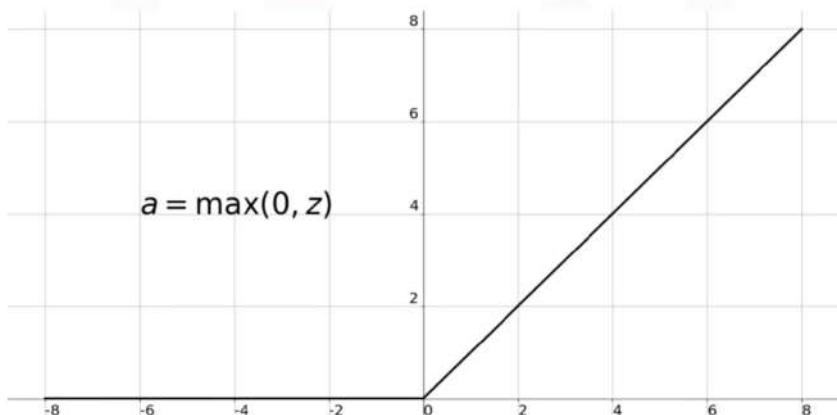
for $z \leq 0$, $a(z) = z$

for $z < 0$, $a(z) = 0$

for $z > 0$, $a'(z) = z$

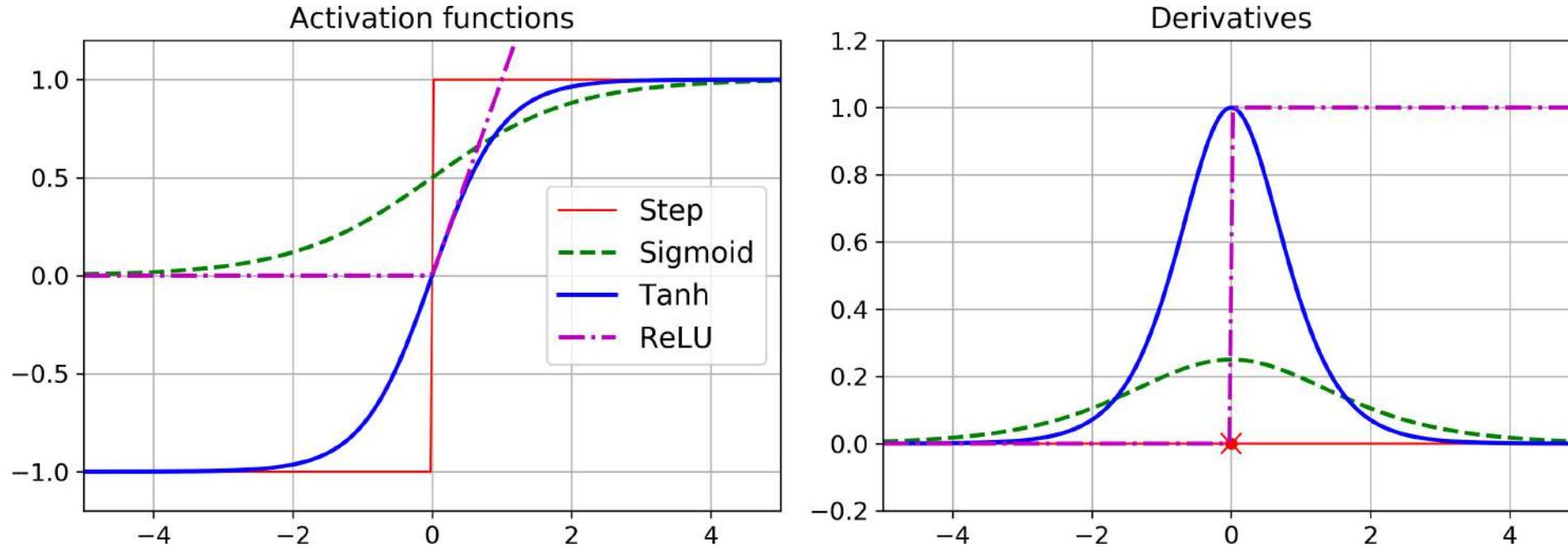
for $z < 0$, $a'(z) = 0$

ReLU Function



Activation Functions

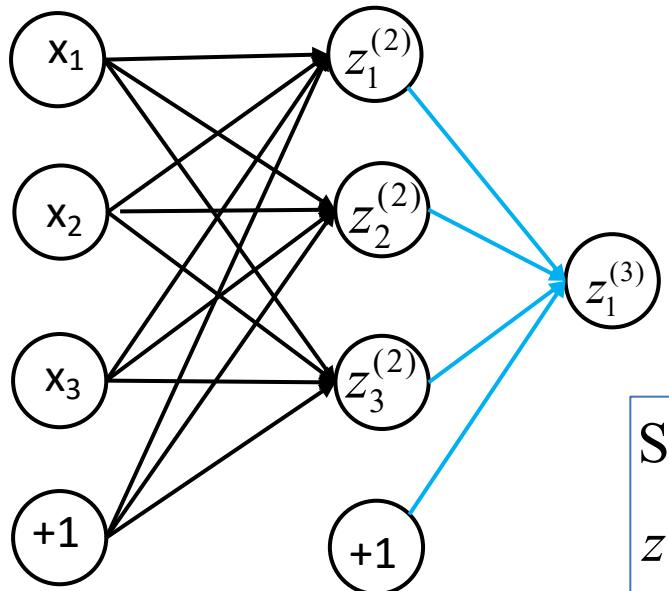
- Activation functions compared: all **nonlinear**



- Why** do we need nonlinear functions?
- Multilayered resulted nonlinear decision boundaries
 - With linear activation, multilayer can be **collapsed into a single layer**

Why Nonlinear Activation Functions?

- Consider linear activation function



$$z^{(2)} = W^{(1)}x + b^{(1)}$$

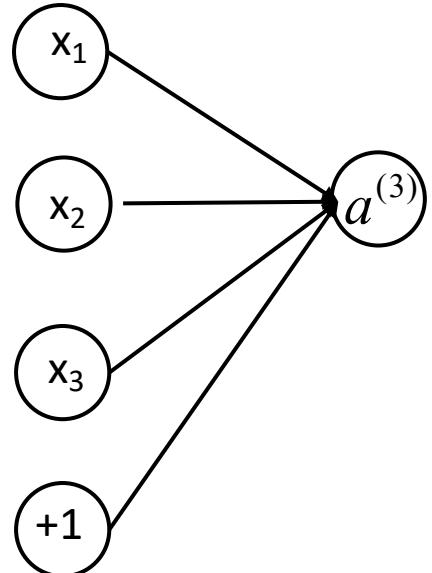
$$a^{(2)} = g^{(2)}(z^{(2)})$$

For linear activation $g^{(2)}(z^{(2)}) = z^{(2)}$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = g^{(3)}(z^{(3)}) = z^{(3)}$$

Equivalent



Since $a^{(2)} = z^{(2)} = W^{(1)}x + b^{(1)}$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} = W^{(2)}(W^{(1)}x + b^{(1)}) + b^{(2)}$$

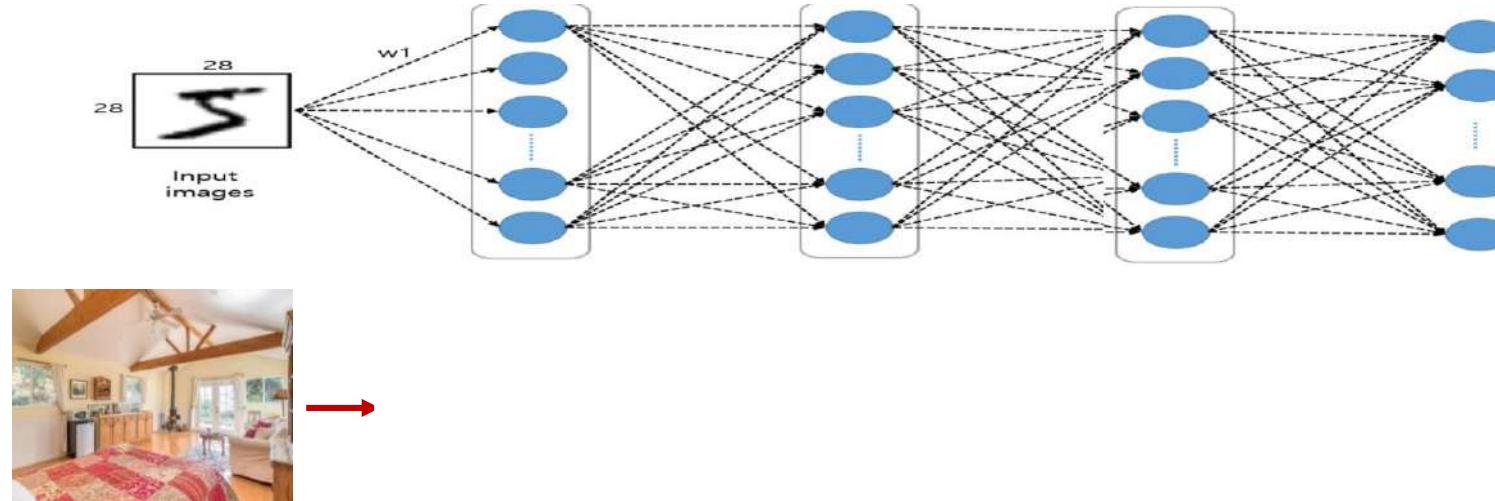
$$a^{(3)} = z^{(3)} = W^{(2)}(W^{(1)}x + b^{(1)}) + b^{(2)}$$

$$= W^{(2)}W^{(1)}x + W^{(2)}b^{(1)} + b^{(2)}$$

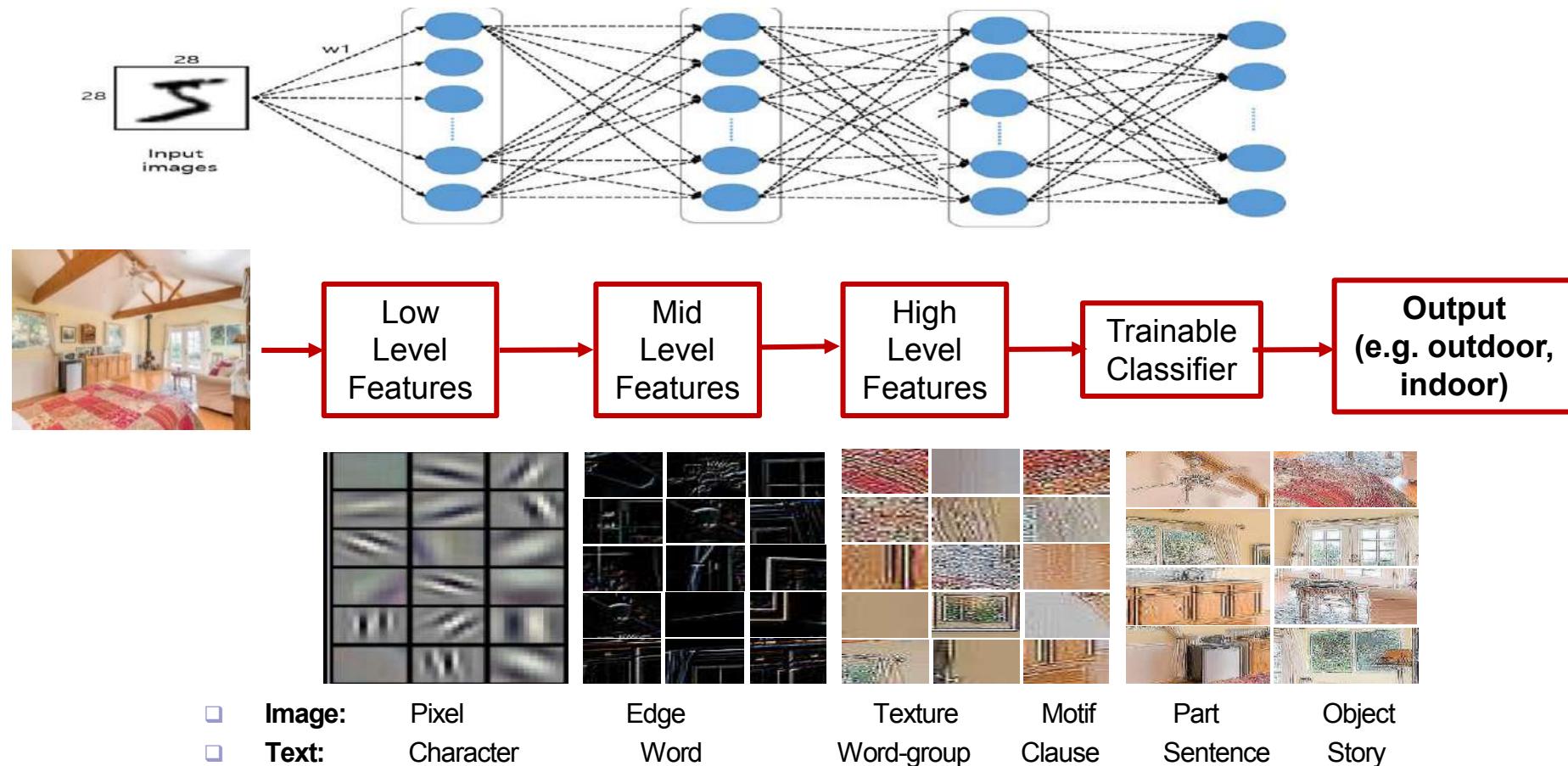
$$= Wx + b$$

A deep network with linear activation is equivalent to a single layer network!

Intuition about deep representation



Intuition about deep representation

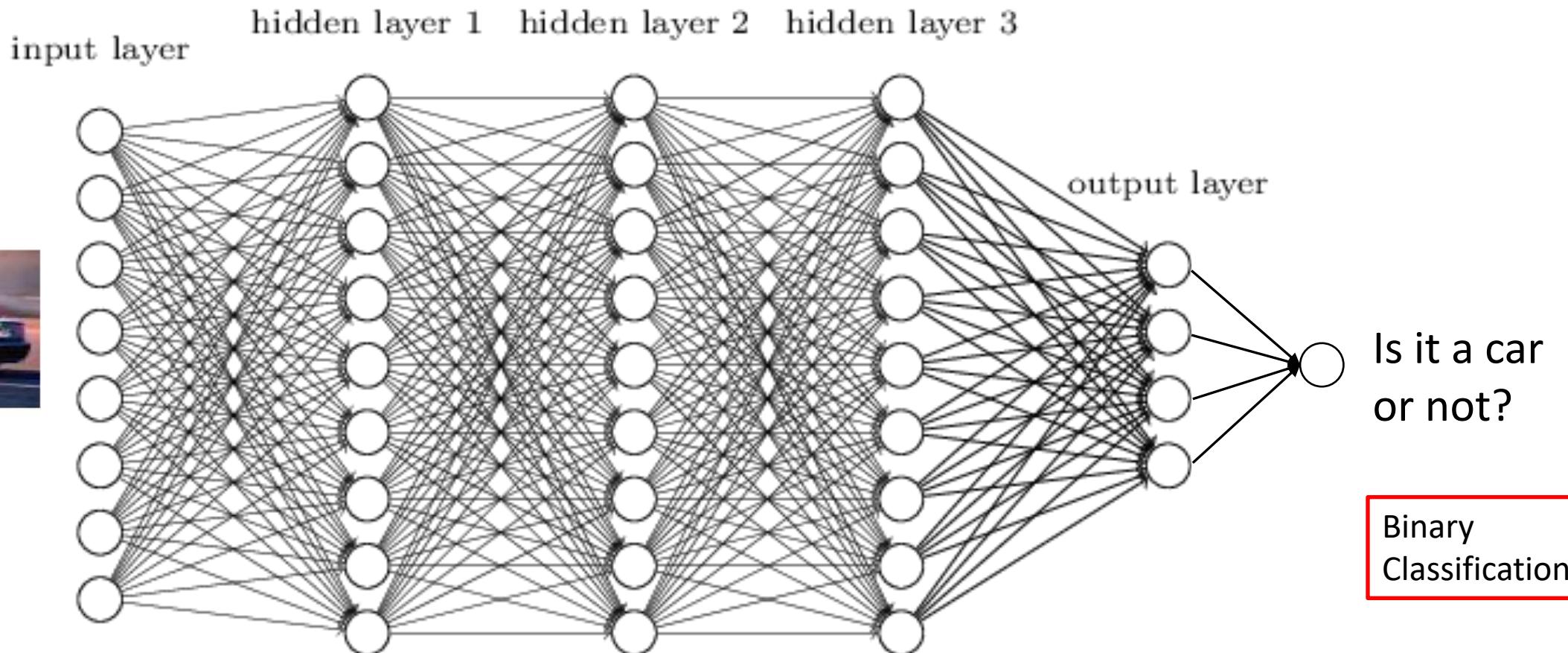


Multi-Layer Perceptron

- Deeper and deeper layers for more complex problems and for higher performance



Car

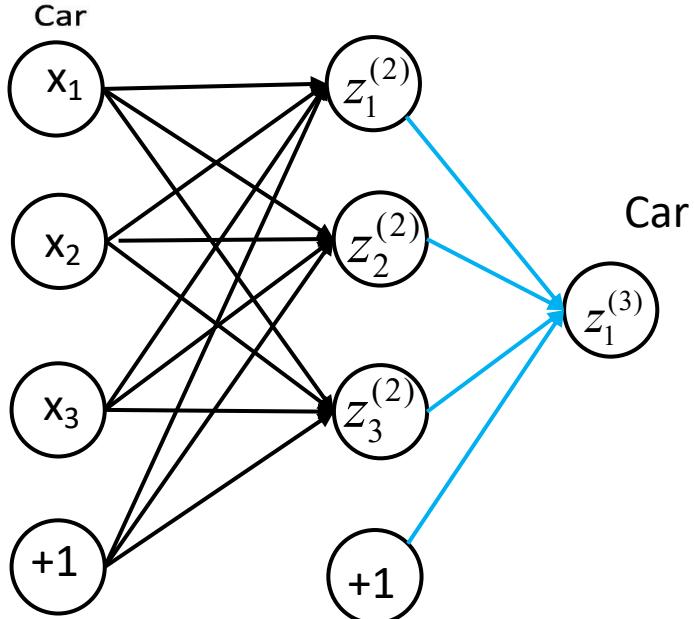


Is it a car
or not?

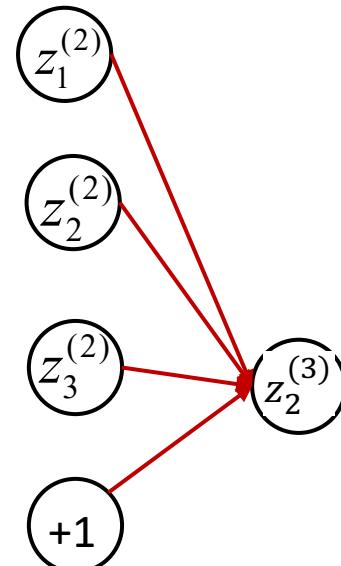
Binary
Classification

Multi-Layer Perceptron

- Binary Classification by MLP



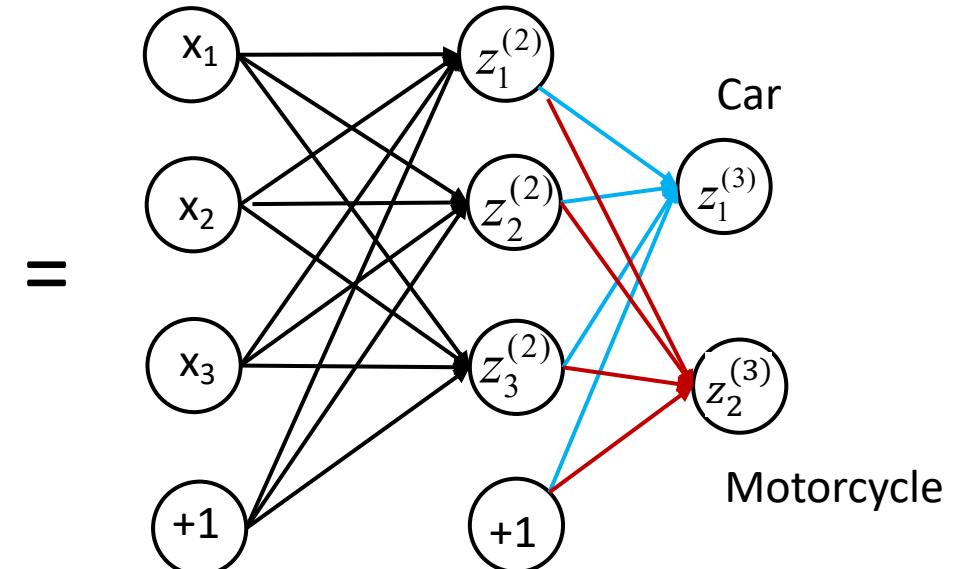
+



- Multiclass Classification by MLP by adding another output node at the last layer



Car Motorcycle



Multi-Layer Perceptron

- Multiclass Classification by MLP



Pedestrian



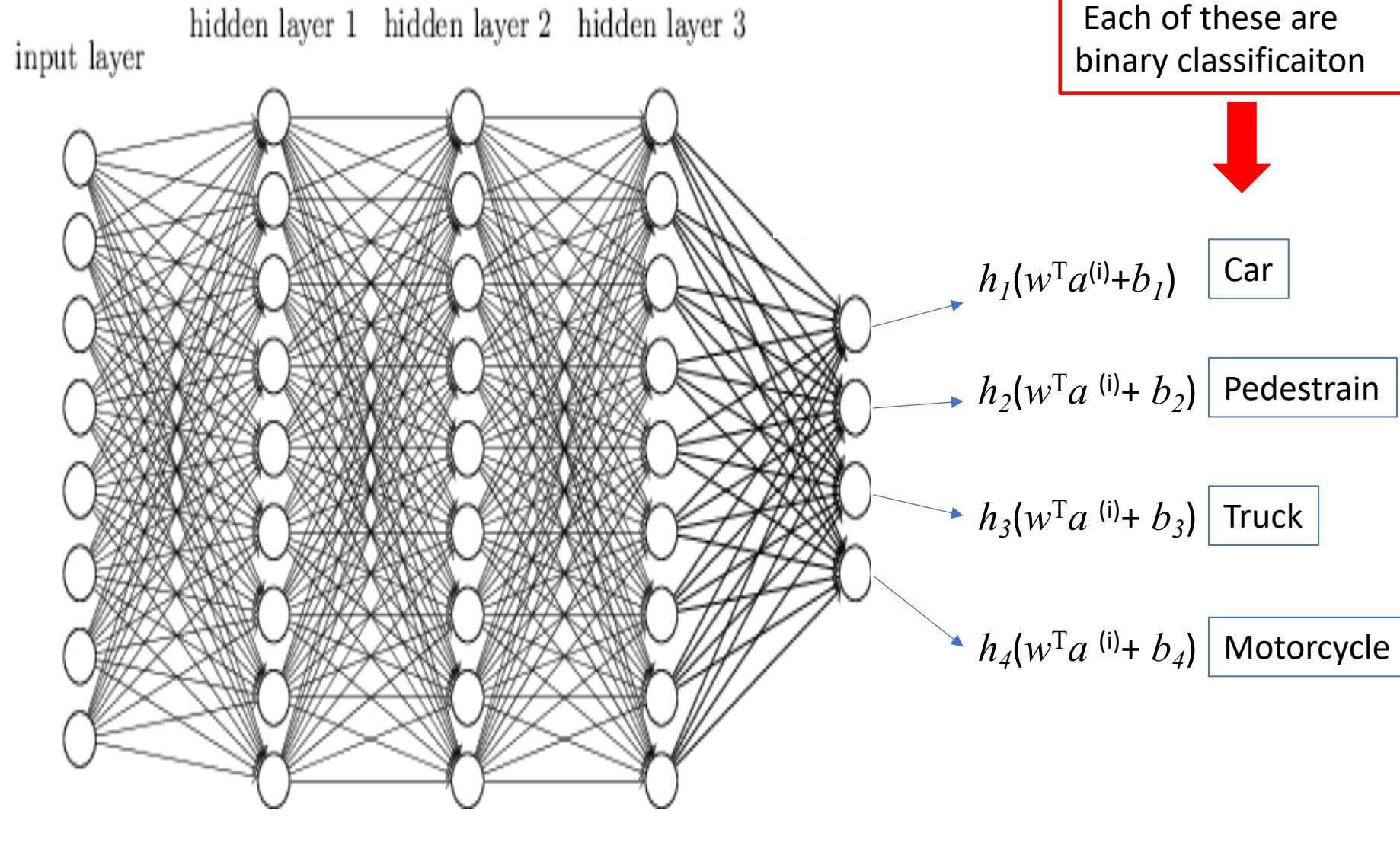
Car



Motorcycle

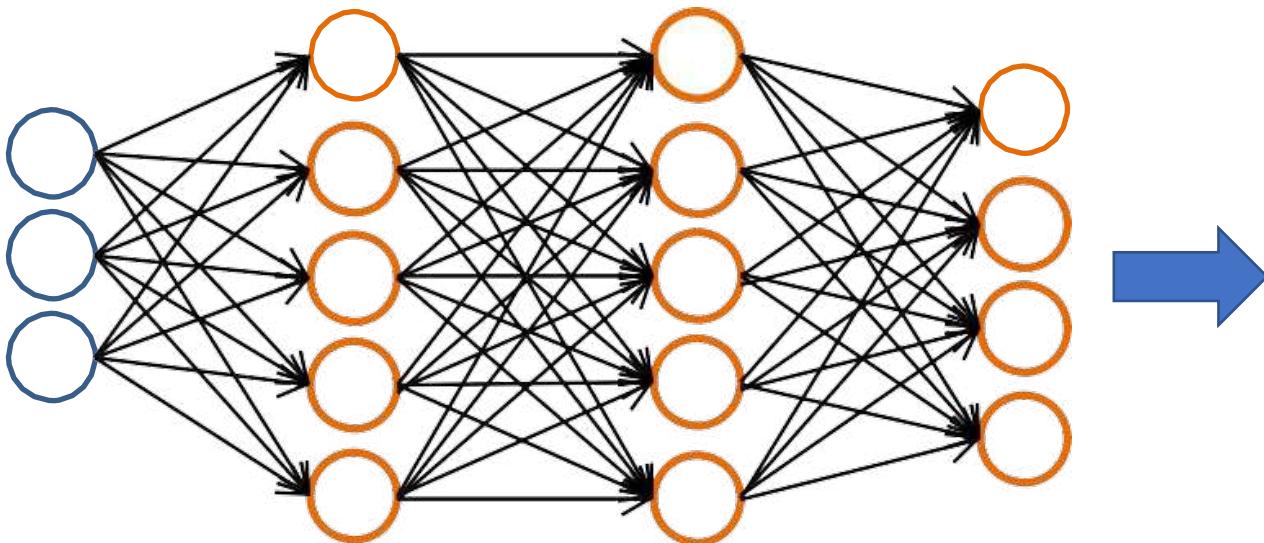


Truck



Multiclass Classification by MLP

- For a multiclass classification, the output is a vector with the dimension matching to the number of the classification categories



$$\begin{bmatrix} 0.86 \\ 0.34 \\ 0.56 \\ 0.02 \end{bmatrix} \begin{bmatrix} 0.06 \\ 0.79 \\ 0.14 \\ 0.57 \end{bmatrix} \begin{bmatrix} 0.35 \\ 0.29 \\ 0.92 \\ 0.45 \end{bmatrix} \begin{bmatrix} 0.15 \\ 0.36 \\ 0.28 \\ 0.65 \end{bmatrix}$$



Pedestrian



Car



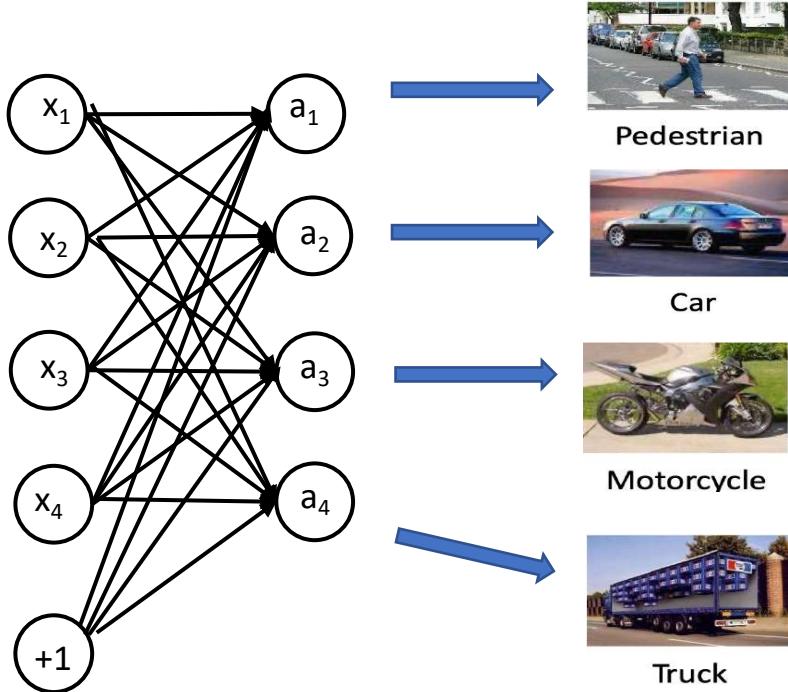
Motorcycle



Truck

Single layer MLP and Naïve Bayes

- Consider a single layer for multiclass classification



$$z = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

Naïve Bayes Classifier

- Consider a naïve bayes classifier for classifying cars with 4-dimensional input features (x_1, x_2, x_3, x_4)

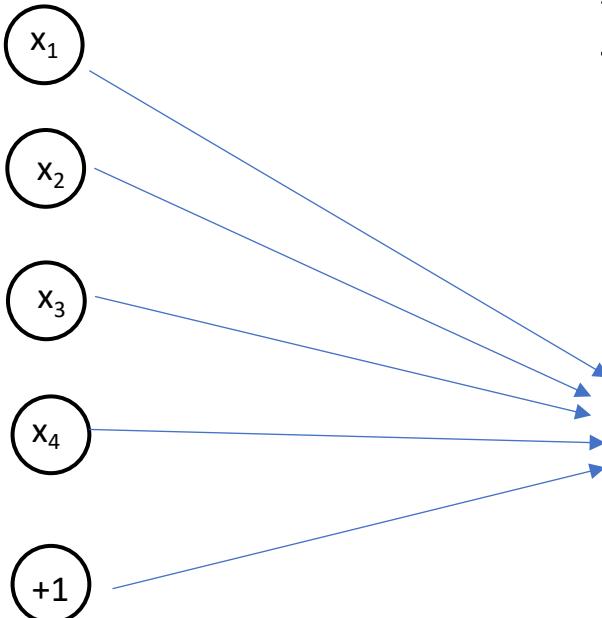
$$P(\text{car} | x_1, x_2, x_3, x_4) = \frac{p(x_1 | \text{car}) p(x_2 | \text{car}) p(x_3 | \text{car}) p(x_4 | \text{car}) P(\text{car})}{p(x_1, x_2, x_3, x_4)}$$

Assuming equal priors, removing evidence term, and taking log likelihood

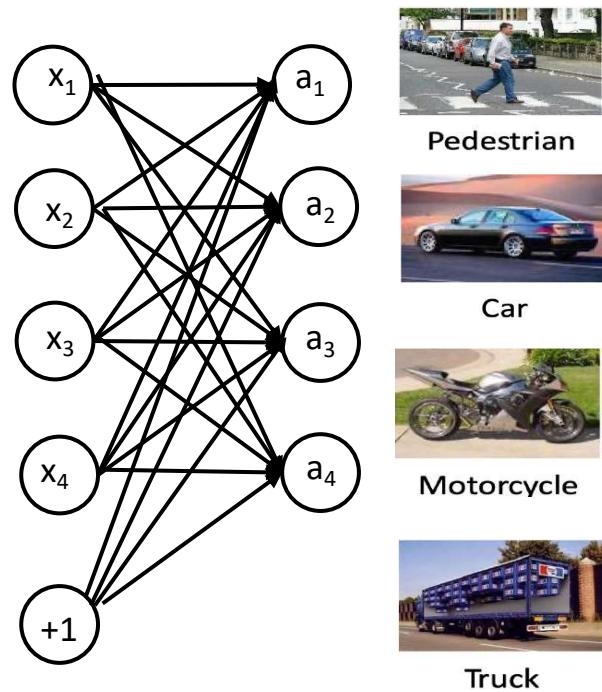
$$\log(P(\text{car} | x_1, x_2, x_3, x_4))$$

$$= \text{constant} + \log P(x_1 | \text{car}) + \log P(x_2 | \text{car}) + \log P(x_3 | \text{car}) + \log P(x_4 | \text{car}) + \log P(\text{car})$$

$$= \text{constant} + \log P(x_1 | \text{car}) + \log P(x_2 | \text{car}) + \log P(x_3 | \text{car}) + \log P(x_4 | \text{car})$$



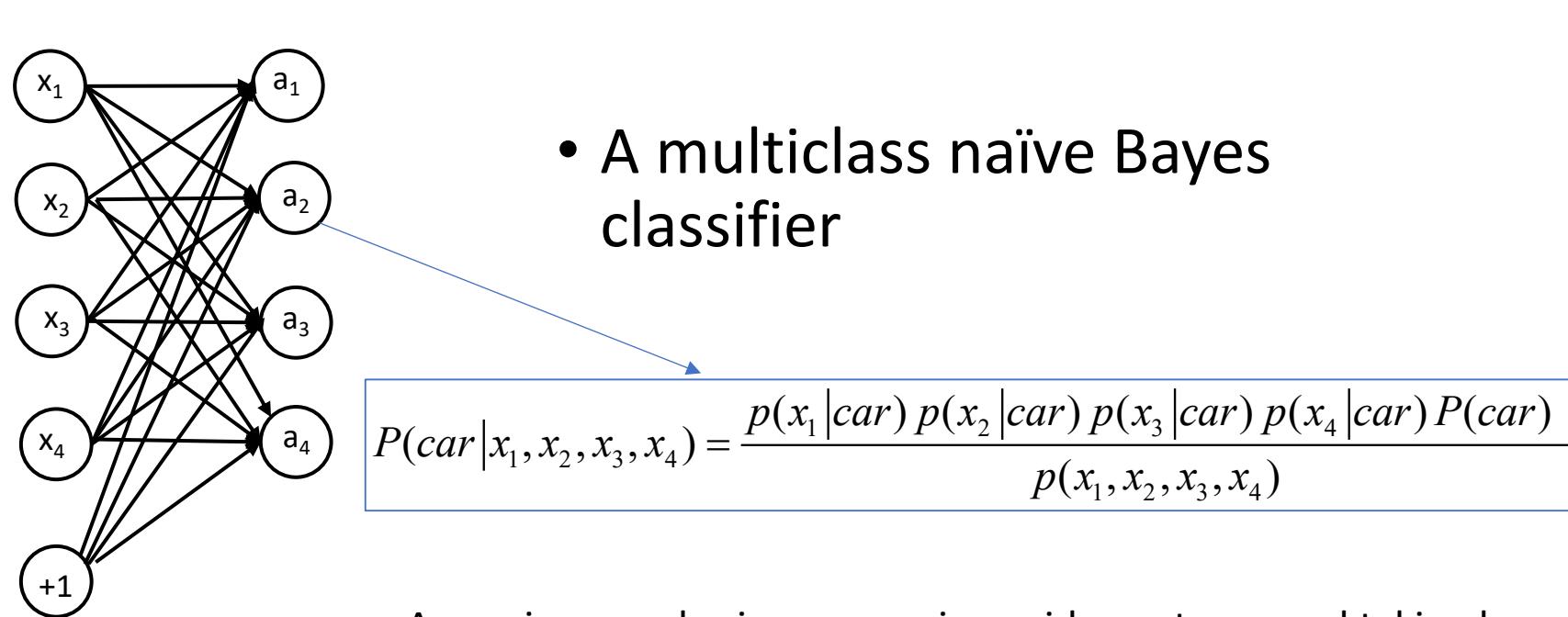
Single layer Neural Net and Naïve Bayes



- A single layer multiclass perceptron

$$z = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

$$h_1(w^T x^{(i)} + b)$$

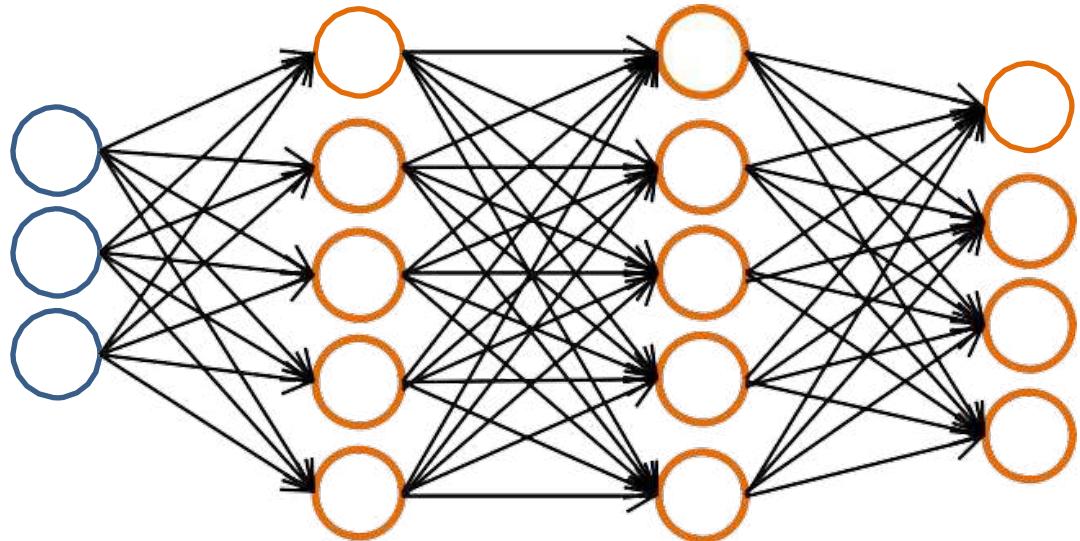


- A multiclass naïve Bayes classifier

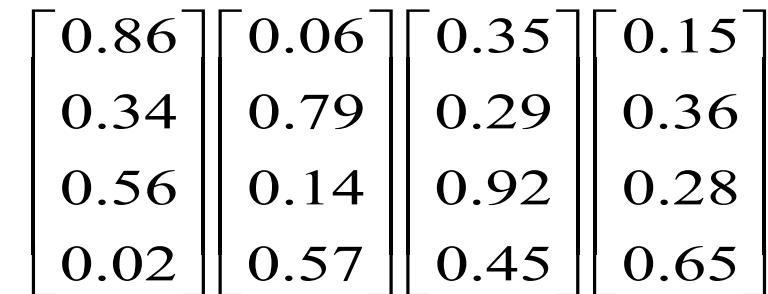
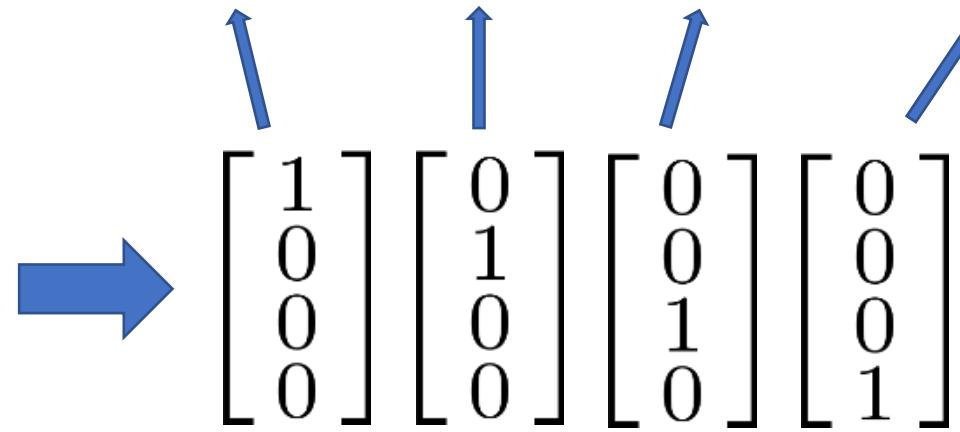
Assuming equal priors, removing evidence term, and taking log likelihood

$$\begin{aligned} & \log(P(\text{car} | x_1, x_2, x_3, x_4)) \\ &= \text{constant} + \log P(x_1 | \text{car}) + \log P(x_2 | \text{car}) + \log P(x_3 | \text{car}) + \log P(x_4 | \text{car}) + \log P(\text{car}) \\ &= \text{constant} + \log P(x_1 | \text{car}) + \log P(x_2 | \text{car}) + \log P(x_3 | \text{car}) + \log P(x_4 | \text{car}) \end{aligned}$$

Multiclass Classification by MLP



Pedestrian Car Motorcycle Truck



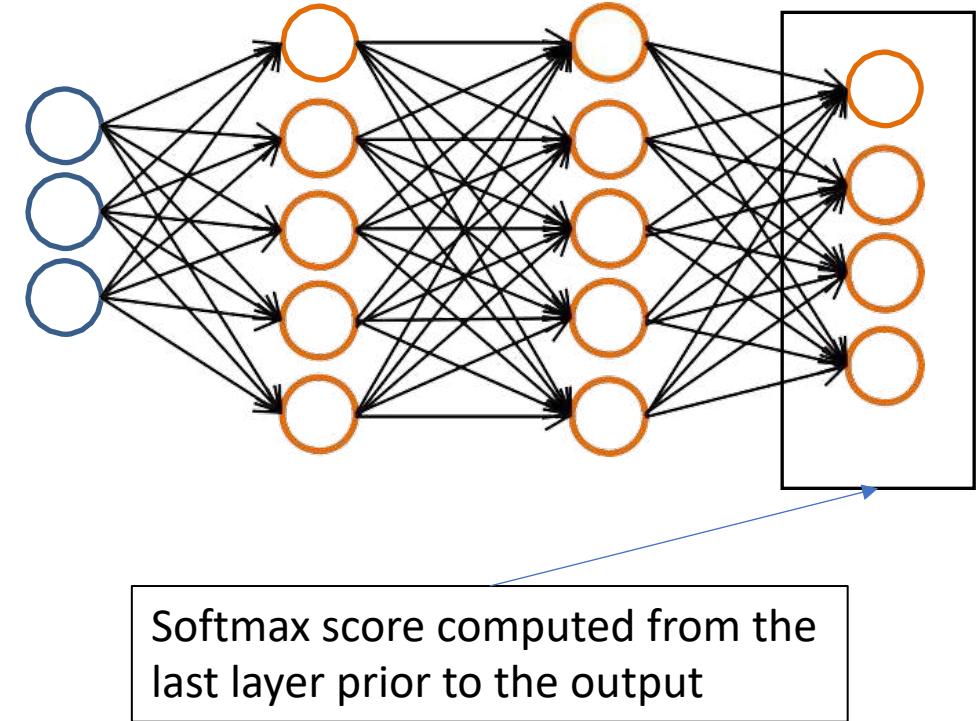
Ideal

Actual

- Can we turn the output to probabilities?

Softmax Regression

- Generalizing logistic regression for multiple classes: **Multinomial Logistic Regression**
- For each class k , the softmax regression model computes a score $z_k^{(d)}(x)$ for each class k .
 - Thus, for k classes, you need k classifiers: $z_k^{(d)}$
 - Softmax score for class k :
- Softmax function then estimates the probability of each class from the scores.

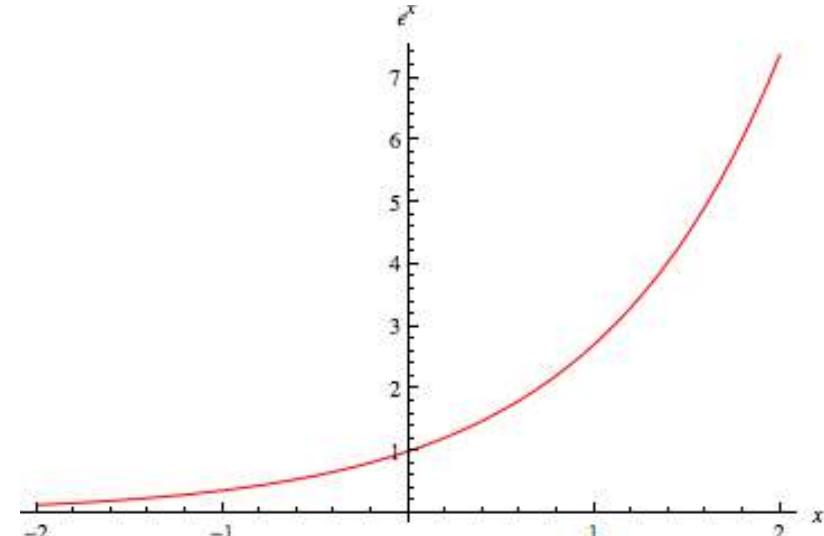


$$z_k^{(d)} = \omega_{k,1}^{(d-1)} a_1^{(d-1)} + \omega_{k,2}^{(d-1)} a_2^{(d-1)} + \omega_{k,3}^{(d-1)} a_3^{(d-1)} + b_k^{(d-1)}$$

Softmax Regression

- Score for class k is computed by a softmax function as

$$\hat{p}_k = \sigma(z(x))_k = \frac{\exp(z_k^{(d)}(x))}{\sum_{j=1}^K \exp(z_j^{(d)}(x))}$$



Exp function ensures that all the values are positive to conform to probability

where K is the number of classes

$z(x)$ is a vector containing the scores of each class for the instance x

$\sigma(z(x))_k$ is the estimated probability that x belongs to class k given the scores of each class for that instance

Softmax Regression

- Consider an example output as follows:

$$\begin{bmatrix} 0.86 \\ 0.34 \\ 0.56 \\ 0.02 \end{bmatrix}$$

- Thus, : $z_1^{(d)}(x)=0.86$, $z_2^{(d)}(x)=0.34$, $z_3^{(d)}(x)=0.56$, $z_4^{(d)}(x)=0.02$

$$\hat{p}_k = \sigma(z(x))_k = \frac{\exp(z_k^{(d)}(x))}{\sum_{j=1}^K \exp(z_j^{(d)}(x))}$$

$$\hat{p}_1 = \frac{\exp(0.86)}{\exp(0.86) + \exp(0.34) + \exp(0.56) + \exp(0.02)} = \frac{2.36}{2.36 + 1.40 + 1.75 + 1.02} = \frac{2.36}{6.53} = 0.36$$

$$\hat{p}_2 = 0.21 \quad \hat{p}_3 = 0.27 \quad \hat{p}_4 = 0.16$$

- Thus, $\hat{p}_1 + \hat{p}_2 + \hat{p}_3 + \hat{p}_4 = 1$

Softmax scores are turned into probabilities

Softmax Regression

- Prediction by Softmax

$$\hat{y} = \operatorname{argmax}_k \sigma(z(x))_k = \operatorname{argmax}_k z_k(x) = \operatorname{argmax}_k \left((\theta^{(k)})^T x \right)$$

- Note that the prediction returns the value of k that maximizes the estimated probability $\sigma(s(x))_k$
- Softmax classifier predicts only one class at a time: use only for mutually exclusive classes

Softmax Regression

- How do you compute a cost function for softmax classifier?
- Use **Cross entropy loss** : measures how well the predicted class probabilities match with the target classes
- Cross entropy between two probability distributions p and q :

$$H(p, q) = -\sum_x p(x) \log q(x)$$

- Since we are computing a cost function, minimizing it is the objective.
- Consider a single example case of binary cross entropy loss
 - Note that H is always > 0 since $\log[q(x)] < 0$ as $q(x) < 1$
 - For $p(x) = 1$ and $q(x) = 1$, $H(p, q) = 0$
 - For $p(x) = 0$ and $q(x) = 0$, $H(p, q) = 0$
 - For $p(x) = 1$ and $q(x) = 0$, $H(p, q) \rightarrow \infty$ as $\log(0) \rightarrow -\infty$
 - Thus only when $p(x) = q(x)$, cross entropy loss is minimized

Softmax Regression

- Cross Entropy Loss for multiclass classifier

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(\hat{p}_k^{(i)})$$

where $y_k^{(i)}$ is the target probability that the i^{th} instance belongs to class k .

$\mathbf{y_1}$	$\mathbf{y_2}$	$\mathbf{y_3}$	$\mathbf{p_1}$	$\mathbf{p_2}$	$\mathbf{p_3}$
1	0	0	0.76	0.22	0.02
0	0	1	0.02	0.11	0.87
1	0	0	0.95	0.01	0.04
0	1	0	0.02	0.53	0.45

- For binary classification, the above equation degenerates to the logistic regression cost function shown earlier. Thus $K=2$.
- Consider loss for one instance: $m=1$

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(\hat{p}_k^{(i)}) = -\sum_{k=1}^K y_k^{(i)} \log(\hat{p}_k^{(i)})$$

Softmax Regression

$$J(\Theta) = -y_1 \log(\hat{p}_1) - y_2 \log(\hat{p}_2)$$

but in binary, $\hat{p}_2 = 1 - \hat{p}_1$ and $y_2 = 1 - y_1$

let $y_1 = y$ and $\hat{p}_1 = \hat{p}$

$$\text{then } J(\Theta) = -y \log \hat{p} - (1 - y) \log(1 - \hat{p})$$

- Recall **maximum** log likelihood of logistic regression

$$\log L(w, b) = \sum_{i=1}^m y^{(i)} \log h_{w,b}(\vec{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{w,b}(\vec{x}^{(i)}))$$

- For a single instance with one data point: $m = 1$
- $l(w, b) = y^{(i)} \log h_{w,b}(\vec{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{w,b}(\vec{x}^{(i)}))$

Softmax Regression

- Continue calculating the gradient if sigmoid

$$\frac{\partial}{\partial w_j} \ell(w, b) = (y - h_{w,b}(x)) x_j$$

- Similarly for b

$$\frac{\partial}{\partial b} \ell(w, b) = (y - h_{w,b}(x))$$

- For multiclass, the gradient vector for the cost function becomes

$$\nabla_{\theta^{(k)}} J(\Theta) = (\hat{p}_k^{(i)} - y_k^{(i)}) \mathbf{x}^{(i)}$$

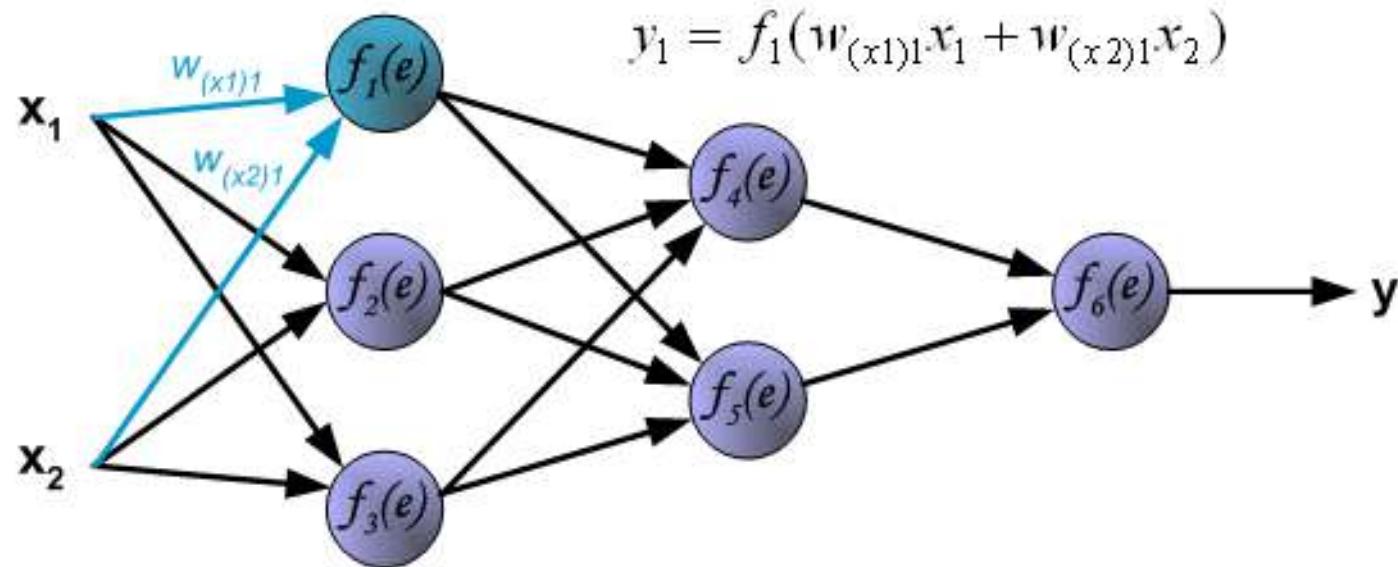
Back Propagation

What caused?

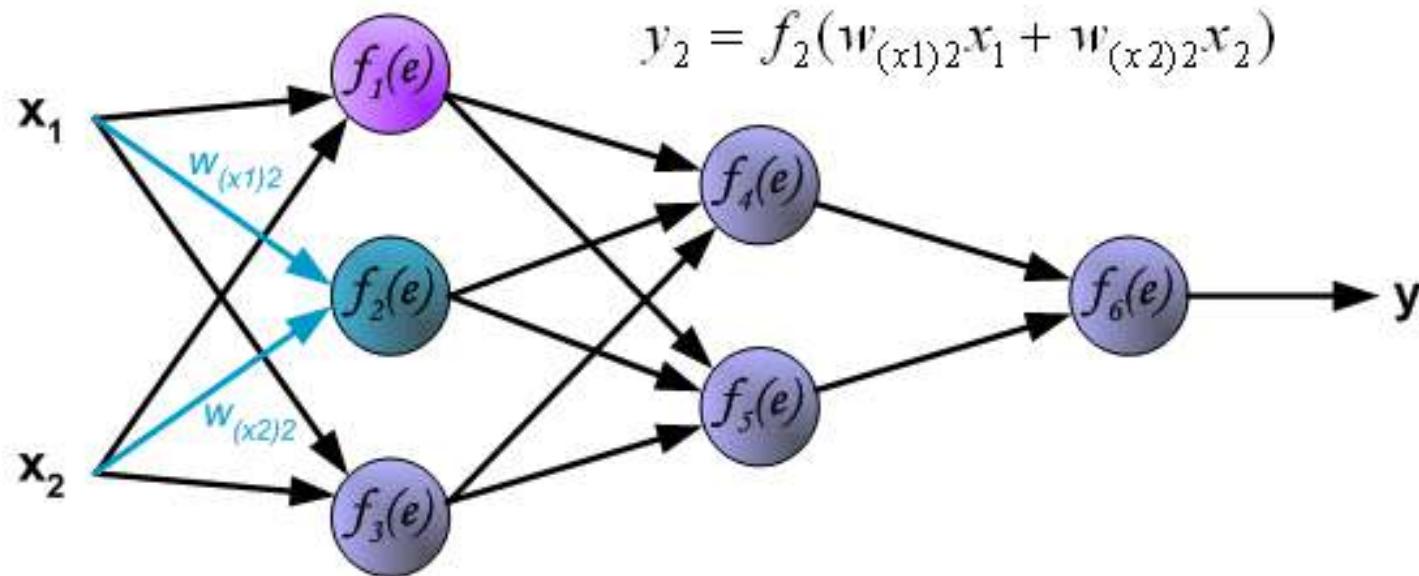


Learning Algorithm: Backpropagation

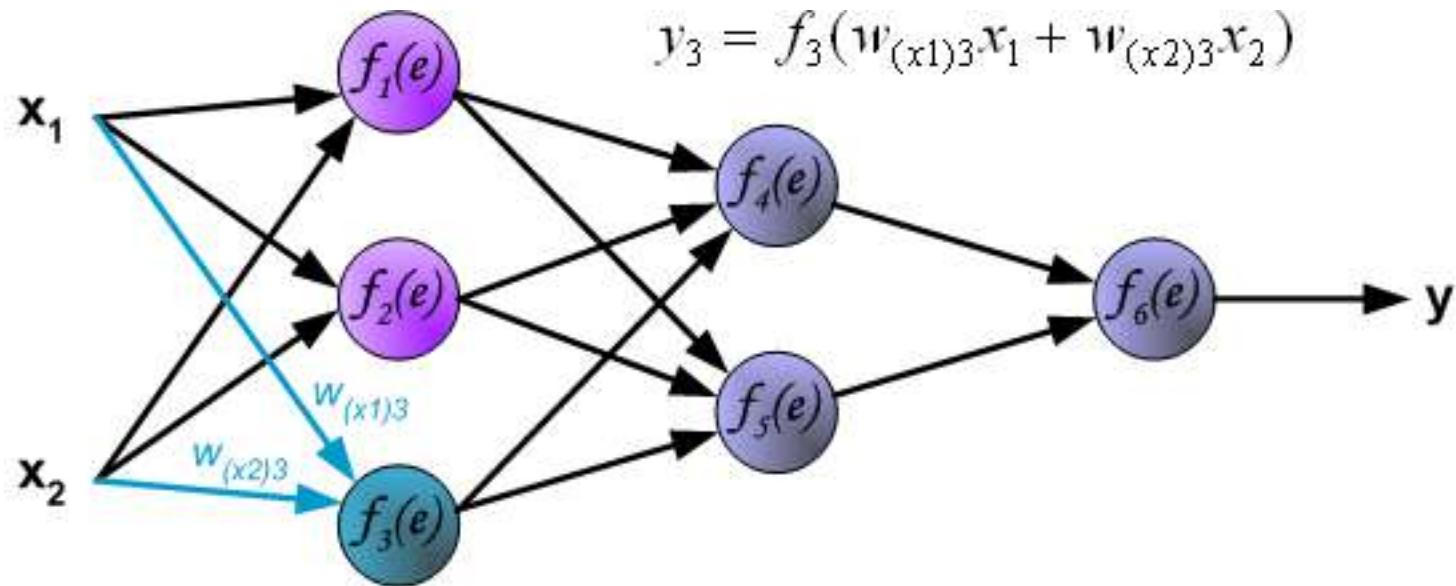
Pictures below illustrate how signal is propagating through the network,
Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and
neuron n in input layer. Symbols y_n represents output signal of neuron n .



Learning Algorithm: Backpropagation

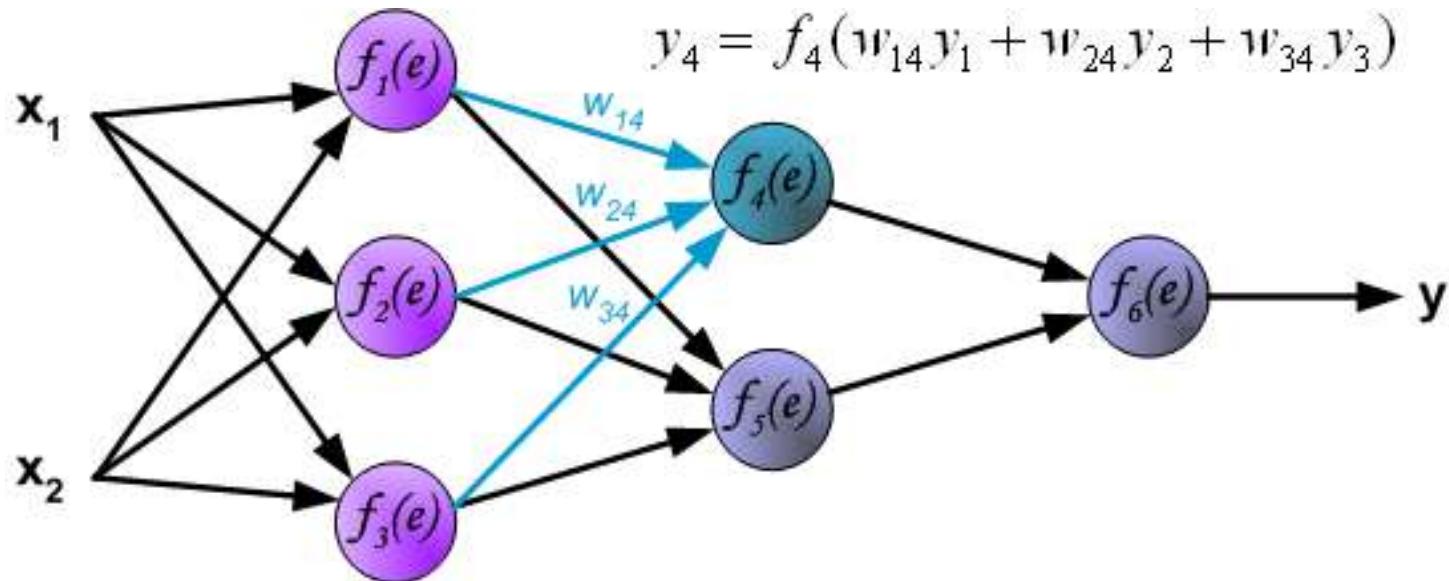


Learning Algorithm: Backpropagation

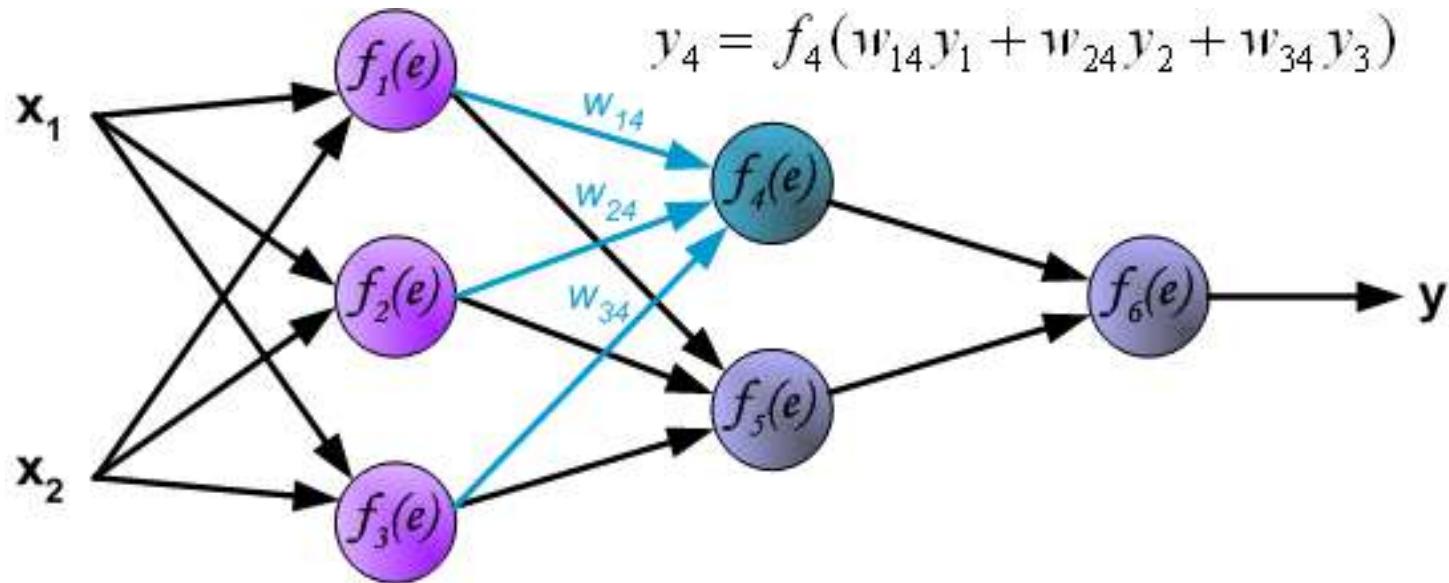


Learning Algorithm: Backpropagation

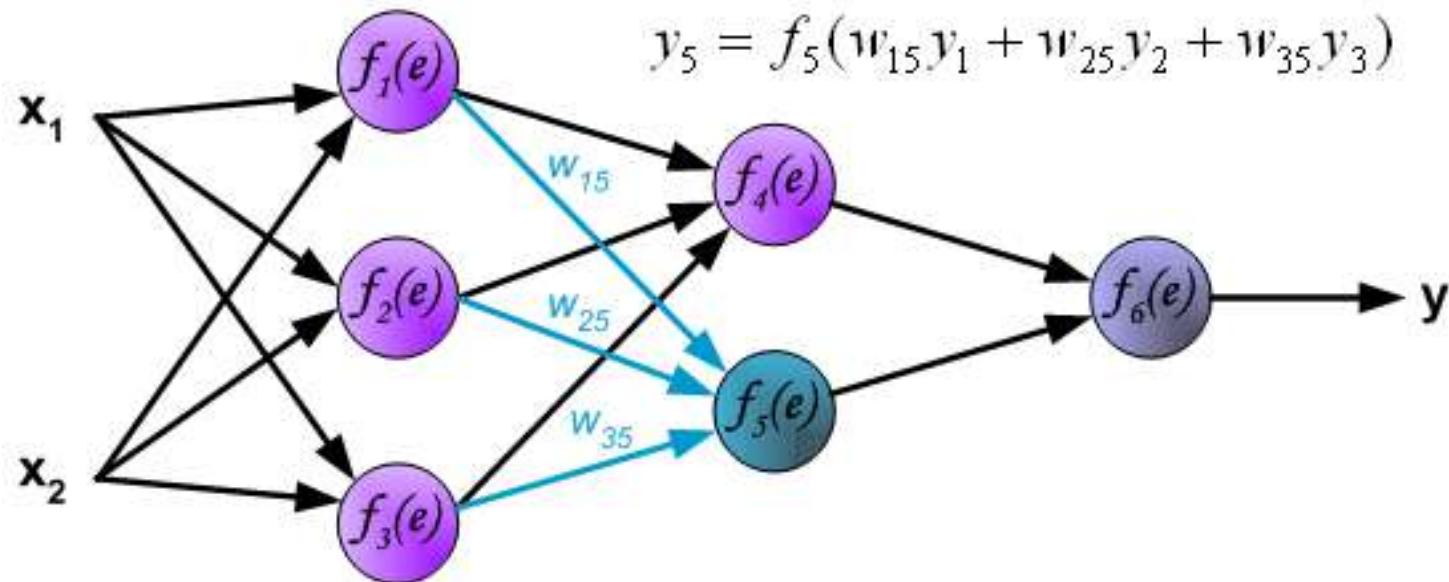
Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron m and input of neuron n in the next layer.



Learning Algorithm: Backpropagation

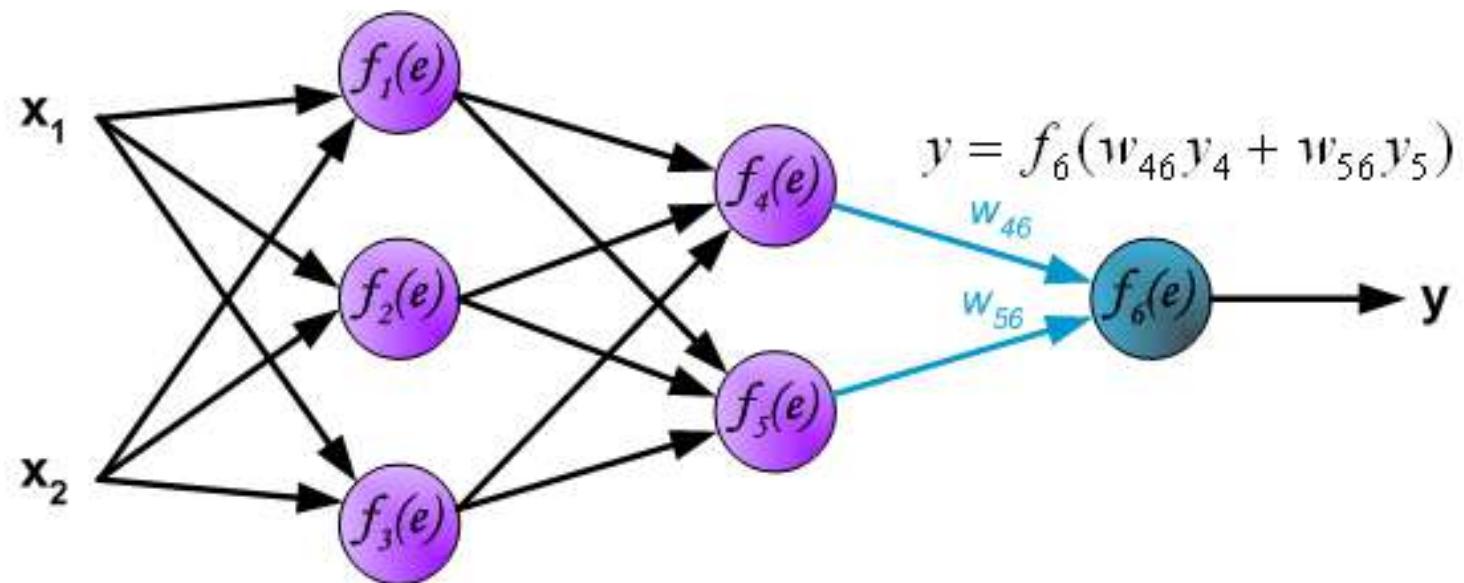


Learning Algorithm: Backpropagation

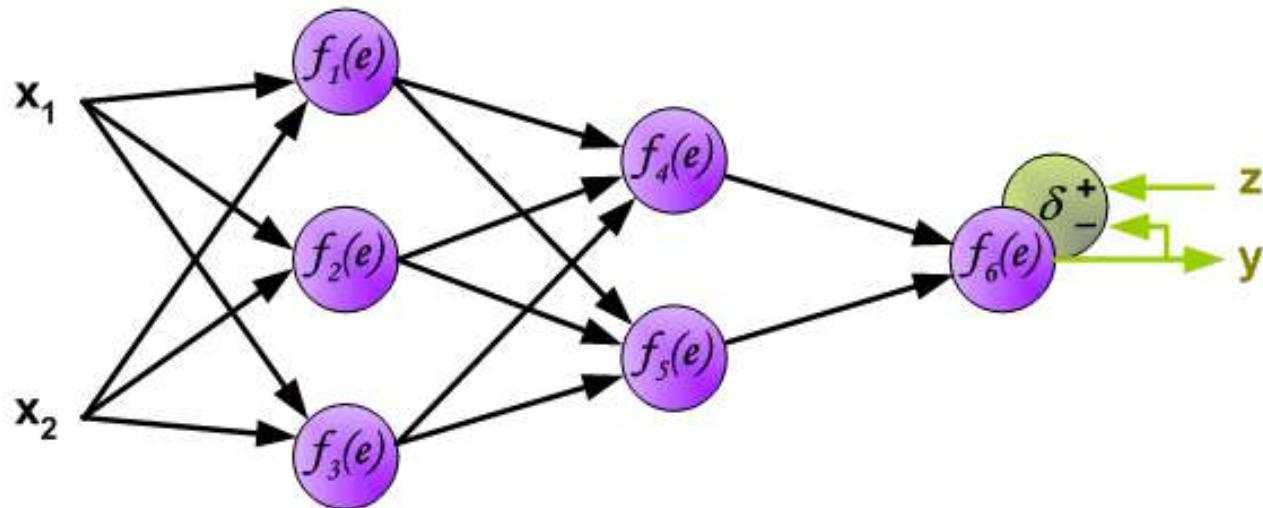


Learning Algorithm: Backpropagation

Propagation of signals through the output layer.

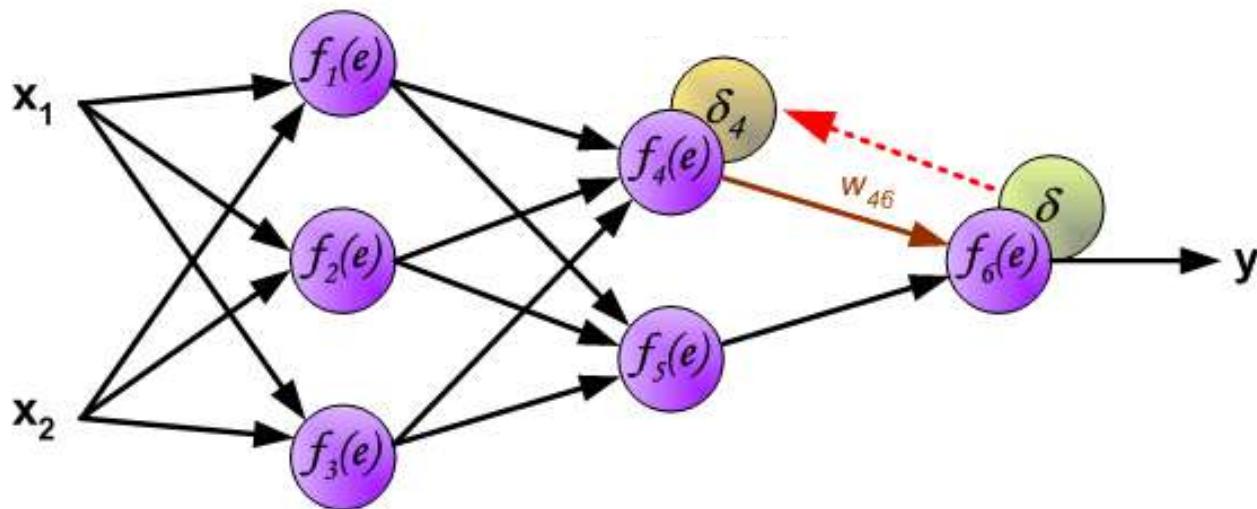


Learning Algorithm: Backpropagation



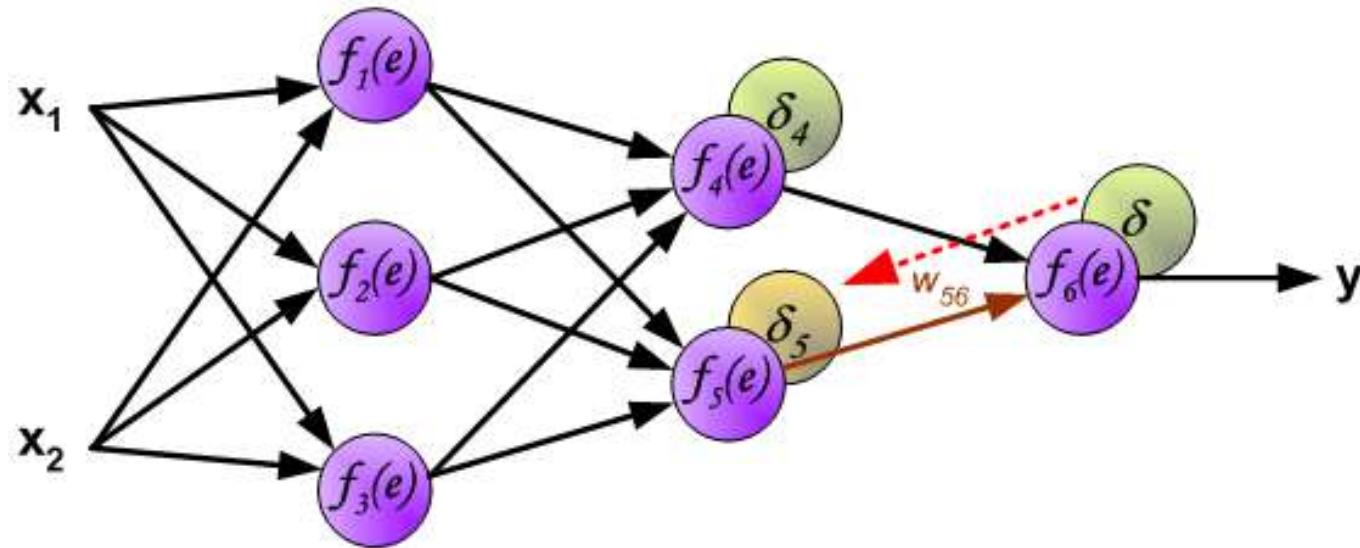
Learning Algorithm: Backpropagation

The idea is to propagate error signal d (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.

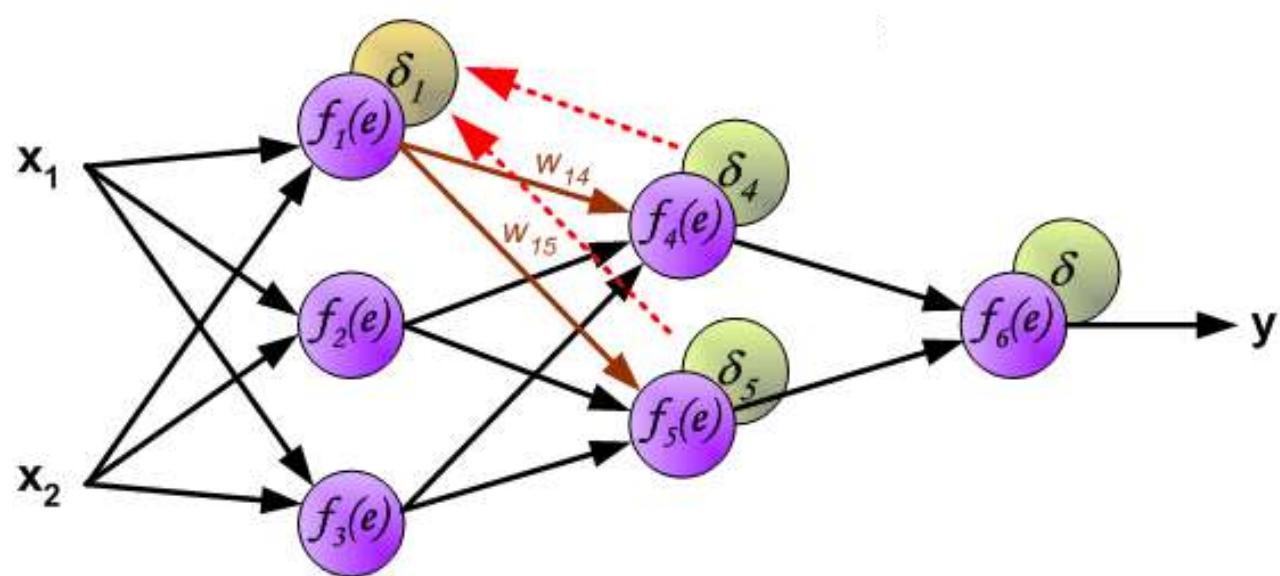


Learning Algorithm: Backpropagation

The idea is to propagate error signal d (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.

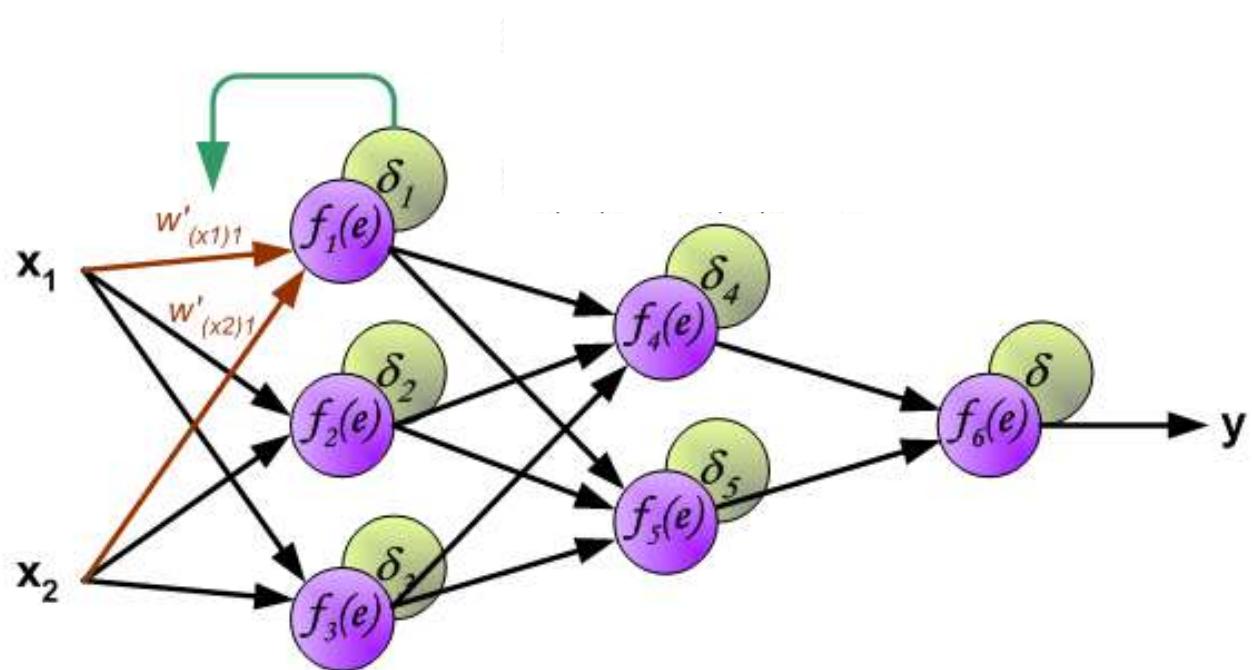


Learning Algorithm: Backpropagation

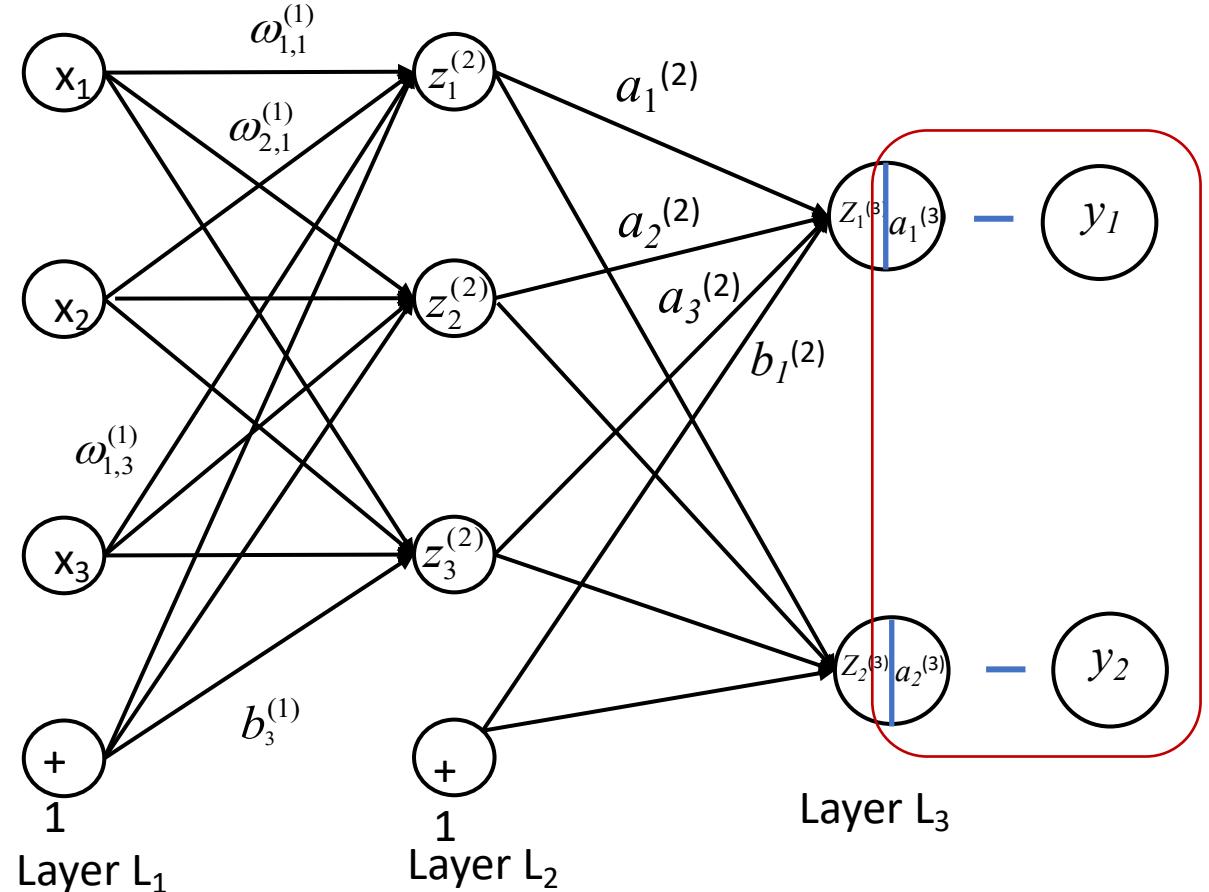


Learning Algorithm: Backpropagation

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified.



Multi-Layer Perceptron and Back-propagation



- Consider a neural network with two outputs

$$z_1^{(3)} = \omega_{1,1}^{(2)} a_1^{(2)} + \omega_{1,2}^{(2)} a_2^{(2)} + \omega_{1,3}^{(2)} a_3^{(2)} + b_1^{(2)}$$

$$a_1^{(3)} = h_1 = f(z_1^{(3)}) = f(\omega_{1,1}^{(2)} a_1^{(2)} + \omega_{1,2}^{(2)} a_2^{(2)} + \omega_{1,3}^{(2)} a_3^{(2)} + b_1^{(2)})$$

$$\longrightarrow J(\omega, b; x, y) = \frac{1}{2} \|h_{\omega, b}(x) - y\|^2$$

- Cost function $J(w_{i,j}, b_i)$ used to optimize **weights $w_{i,j}$ and b_i** by taking partial derivatives

$$\omega_{i,j}^{(l)} := \omega_{i,j}^{(l)} - \alpha \frac{\partial}{\partial \omega_{i,j}^{(l)}} J(\omega, b)$$

$$b_i^l := b_i^l - \alpha \frac{\partial}{\partial b_i^{(l)}} J(\omega, b)$$

Multi-Layer Perceptron and Back-propagation

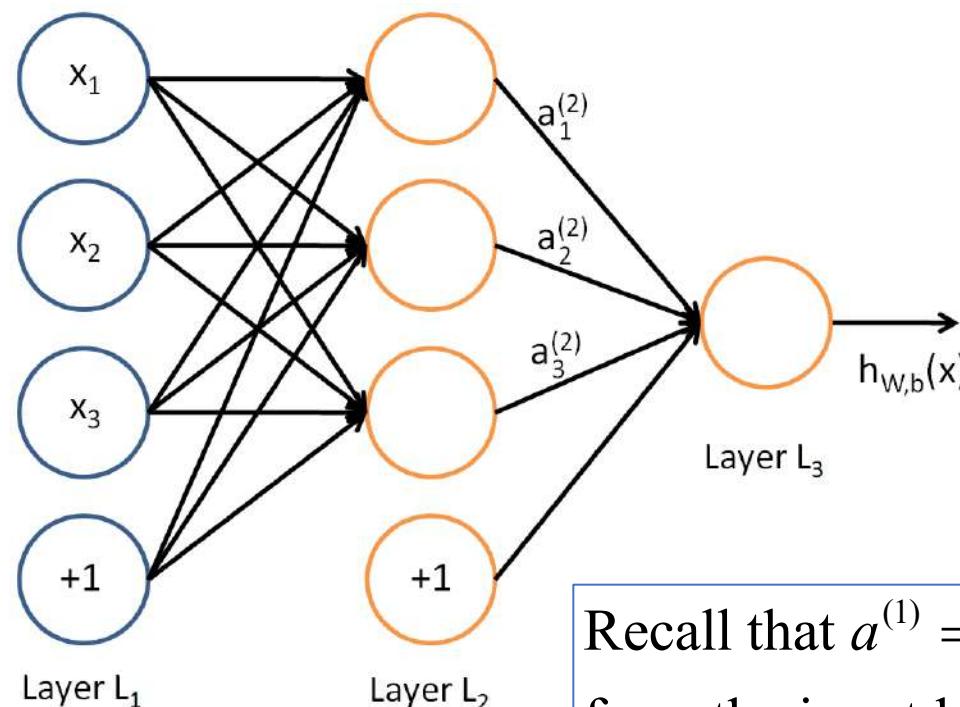
- The goal is to **minimize** $J(w,b)$ as a function of w and b .
- $J(w,b)$ is **not convex**, thus the gradient descent process might get stuck on local minima. Usually, however, gradient descent **works**.
- **Initialization** of w and b important as in any optimization problem.
- **Initialize** the parameters **randomly, non-zeros**.
- **Updating** w and b by gradient descent
- **Important** to be **able to calculate** gradient of activation function

$$\omega_{i,j}^{(l)} := \omega_{i,j}^{(l)} - \alpha \frac{\partial}{\partial \omega_{i,j}^{(l)}} J(\omega, b)$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(\omega, b)$$

where α is the learning rate

Notational Convention in Back-propagation



$z_i^{(l)}$ denotes the total weighted sum of inputs to unit i in layer l including the bias term

$$\text{Thus, } z_i^{(2)} = \sum_{j=1}^n \omega_{ij}^{(1)} x_j + b_i^{(1)}$$

$$\text{so that } a_i^{(l)} = f(z_i^{(l)})$$

Recall that $a^{(1)} = x$ which denotes the values from the input layer. Without the subscript index, it is implied both $a^{(1)}$ and x are vectors.

Similarly, we can further write compactly as

$$z^{(l+1)} = \omega^{(l)} a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f(z^{(l+1)})$$

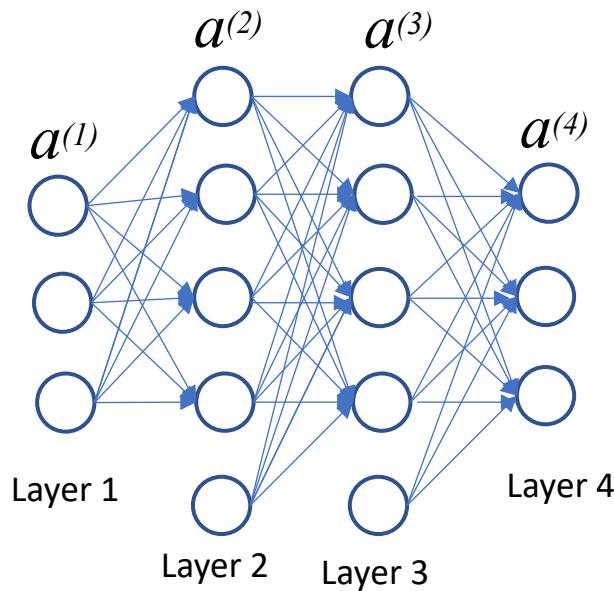
$$z^{(2)} = \omega^{(1)} x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = \omega^{(2)} a^{(2)} + b^{(2)}$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

Partial Derivatives and Chain Rule



Let $\theta^{(l)} = \text{matrix of } w_{i,j}^{(l)} \text{ and } b_i^{(l)}$

$$\begin{aligned}a^{(4)} &= h_{\omega,b}(x) = f(z^{(4)}) = f(\theta^{(3)}a^{(3)}) \\&= f(\theta^{(3)}g(z^{(3)})) = f(\theta^{(3)}g(\theta^{(2)}a^{(2)})) \\&= f(\theta^{(3)}g(\theta^{(2)}p(z^{(2)}))) = f(\theta^{(3)}g(\theta^{(2)}p(\theta^{(1)}a^{(1)}))) \\&= f(\theta^{(3)}g(\theta^{(2)}p(\theta^{(1)}x)))\end{aligned}$$


$$J(\omega, b; x, y) = \frac{1}{2} \|h_{\omega,b}(x) - y\|^2$$

- Taking partial derivatives of a complex function
 - Using the chain rule from calculus

$$\frac{d}{dx} [f(g(x))] = f'(g(x))g'(x)$$



Taken from: Why are Russian Dolls so Full of Themselves?
by Sofia Boulamrach | Medium

- Using the chain rule is fine and good, but it can be very messy and time consuming.
- Let's consider a scheme based on the chain rule but more approachable in getting the partial derivatives
- Back Propagation

Multi-Layer Perceptron and Back-propagation

- For a fixed set of training data $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

- Define a loss function for a single sample

$$J(\omega, b; x, y) = \frac{1}{2} \|h_{\omega, b}(x) - y\|^2$$

- For m examples

$$\begin{aligned} J(\omega, b) &= \left[\frac{1}{2} \sum_{i=1}^m J(\omega, b; x^{(i)}, y^{(i)}) \right] \\ &= \left[\frac{1}{m} \sum_{i=1}^m \left(\frac{1}{2} \|h_{\omega, b}(x^{(i)}) - y^{(i)}\|^2 \right) \right] \end{aligned}$$

- Let sigmoid be used for activation
 - For classification, $y=0$ or 1 data labels.

Multi-Layer Perceptron and Back-propagation

1. **Forward pass:** Starting from the **input layer**, the algorithm computes the output of **all the neurons in each layer** for every instance in the batch until the output of the **last layer** (prediction value) is computed.
2. All **intermediate results** are preserved for the **backward pass**.
3. The network's output error (loss function) computed.
4. Chain rule (calculus) applied to analytically compute how much each output connection contributed to the error.
5. The algorithm then measures how much of these error contributions came from each connection in the layer below, again using the chain rule—and so on until the algorithm reaches the input layer.
6. The reverse pass efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward through the network (hence the name of the algorithm).
7. Finally, the algorithm performs a Gradient Descent step to tweak all the connection weights in the network, using the error gradients it just computed.

Multi-Layer Perceptron and Back-propagation

1. **Forward pass:** Starting from the **input layer**, the algorithm computes the output of **all the neurons in each layer** for every instance in the batch until the output of the **last layer** (prediction value) is computed.
2. All **intermediate results** are preserved for the **backward pass**.
3. The network's **output error** (loss function) computed.
4. Chain rule (calculus) applied to analytically compute how much each output connection contributed to the error.
5. The algorithm then measures how much of these error contributions came from each connection in the layer below, again using the chain rule—and so on until the algorithm reaches the input layer.
6. The reverse pass efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward through the network (hence the name of the algorithm).
7. Finally, the algorithm performs a Gradient Descent step to tweak all the connection weights in the network, using the error gradients it just computed.

Multi-Layer Perceptron and Back-propagation

1. **Forward pass:** Starting from the **input layer**, the algorithm computes the output of **all the neurons in each layer** for every instance in the batch until the output of the **last layer** (prediction value) is computed.
2. All **intermediate results** are preserved for the **backward pass**.
3. The network's **output error** (loss function) computed.
4. **Chain rule** (calculus) applied to analytically compute how much **each output** connection **contributed** to the error.
5. The algorithm then measures how much of these error contributions came from each connection in the layer below, again using the chain rule—and so on until the algorithm reaches the input layer.
6. The reverse pass efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward through the network (hence the name of the algorithm).
7. Finally, the algorithm performs a Gradient Descent step to tweak all the connection weights in the network, using the error gradients it just computed.

Multi-Layer Perceptron and Back-propagation

1. **Forward pass:** Starting from the **input layer**, the algorithm computes the output of **all the neurons in each layer** for every instance in the batch until the output of the **last layer** (prediction value) is computed.
2. All **intermediate results** are preserved for the **backward pass**.
3. The network's **output error** (loss function) computed.
4. **Chain rule** (calculus) applied to analytically compute how much **each output** connection **contributed** to the error.
5. The algorithm then measures **how much** of these **error contributions** came from each connection in the **layer below**, again using the **chain rule**—and so on until the algorithm reaches the input layer.
6. The reverse pass efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward through the network (hence the name of the algorithm).
7. Finally, the algorithm performs a Gradient Descent step to tweak all the connection weights in the network, using the error gradients it just computed.

Multi-Layer Perceptron and Back-propagation

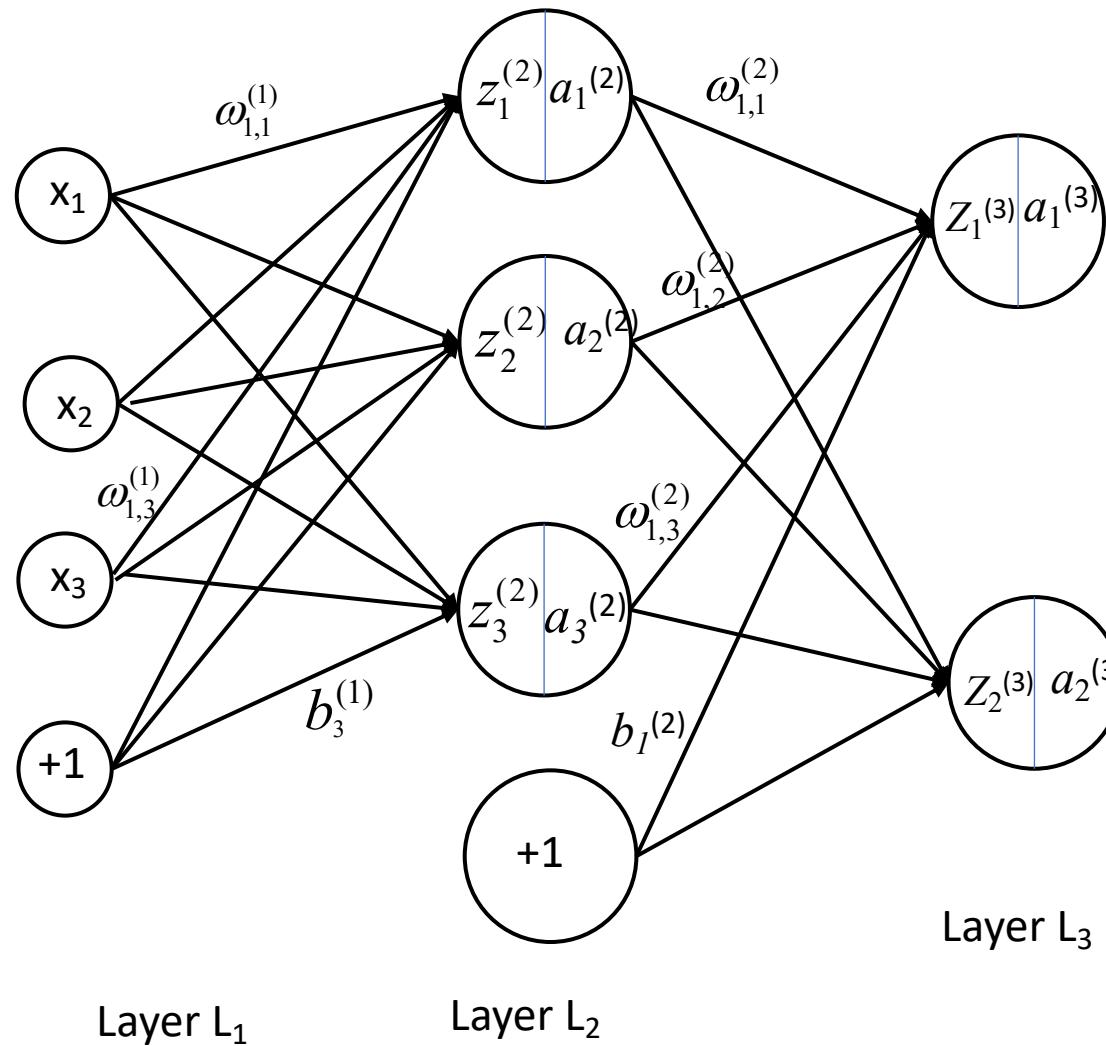
1. **Forward pass:** Starting from the **input layer**, the algorithm computes the output of **all the neurons in each layer** for every instance in the batch until the output of the **last layer** (prediction value) is computed.
2. All **intermediate results** are preserved for the **backward pass**.
3. The network's **output error** (loss function) computed.
4. **Chain rule** (calculus) applied to analytically compute how much **each output** connection **contributed** to the error.
5. The algorithm then measures **how much** of these **error contributions** came from each connection in the **layer below**, again using the **chain rule**—and so on until the algorithm reaches the input layer.
6. The reverse pass **efficiently measures** the **error gradient** across all the connection weights in the network by **propagating the error gradient backward** through the network (hence the name of the algorithm).
7. Finally, the algorithm performs a Gradient Descent step to tweak all the connection weights in the network, using the error gradients it just computed.

Multi-Layer Perceptron and Back-propagation

1. **Forward pass:** Starting from the **input layer**, the algorithm computes the output of **all the neurons in each layer** for every instance in the batch until the output of the **last layer** (prediction value) is computed.
2. All **intermediate results** are preserved for the **backward pass**.
3. The network's **output error** (loss function) computed.
4. **Chain rule** (calculus) applied to analytically compute how much **each output** connection **contributed** to the error.
5. The algorithm then measures **how much** of these **error contributions** came from each connection in the **layer below**, again using the **chain rule**—and so on until the algorithm reaches the input layer.
6. The reverse pass **efficiently measures** the **error gradient** across all the connection weights in the network by **propagating the error gradient backward** through the network (hence the name of the algorithm).
7. Finally, the algorithm performs a **Gradient Descent step** to tweak all the connection weights in the network, using the error gradients it just computed.

Multi-Layer Perceptron and Back-propagation

- Let's compute the forward pass of a network with one hidden layer.



$$z_1^{(2)} = \omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2 + \omega_{13}^{(1)} x_3 + b_1^{(1)}$$

$$z_2^{(2)} = \omega_{21}^{(1)} x_1 + \omega_{22}^{(1)} x_2 + \omega_{23}^{(1)} x_3 + b_2^{(1)}$$

$$z_3^{(2)} = \omega_{31}^{(1)} x_1 + \omega_{32}^{(1)} x_2 + \omega_{33}^{(1)} x_3 + b_3^{(1)}$$

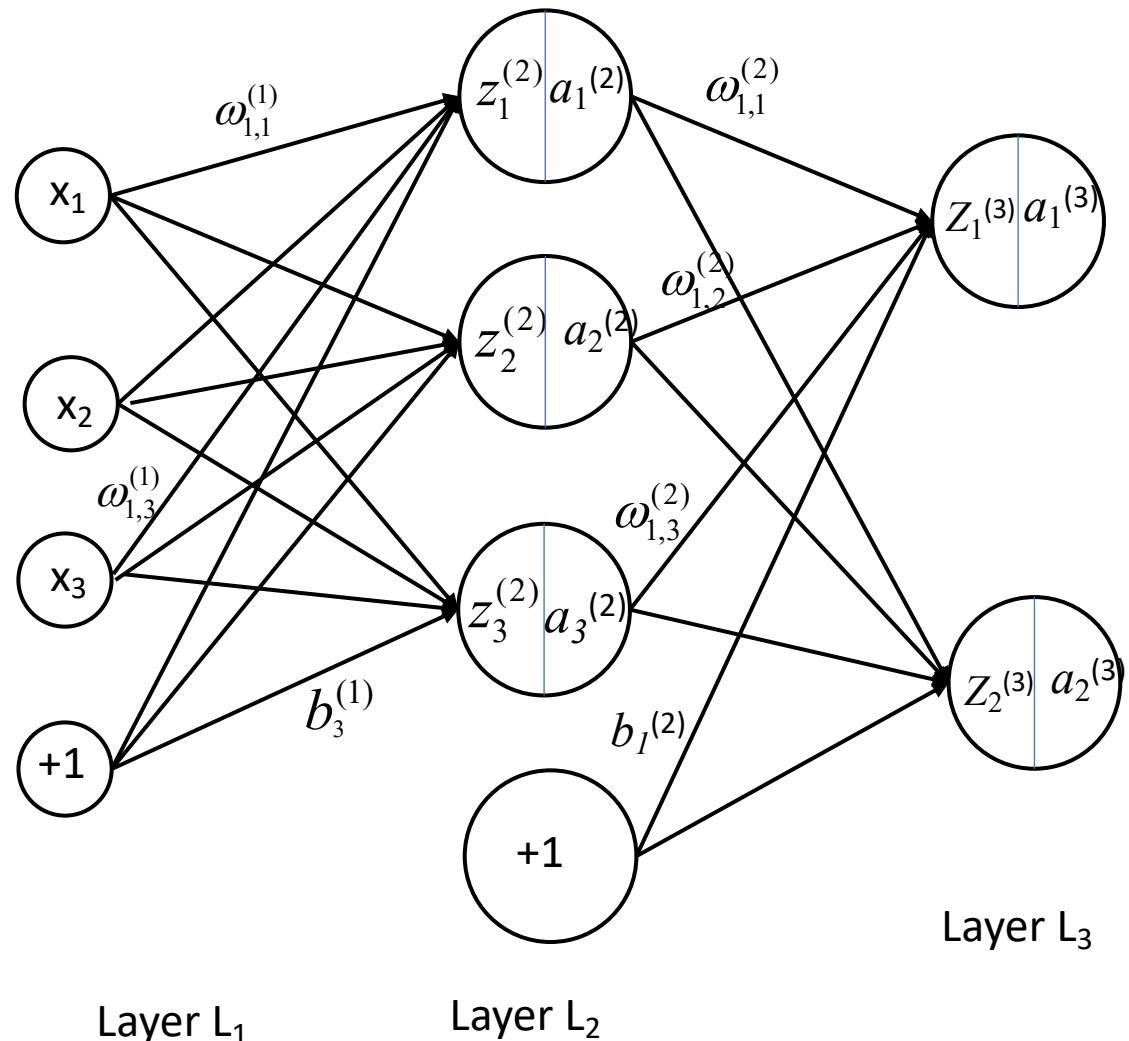
$$\vec{x} = \vec{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix}$$

$$\vec{z}^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \tilde{\omega}^{(1)} = \begin{bmatrix} \omega_{11}^{(1)} & \omega_{12}^{(1)} & \omega_{13}^{(1)} \\ \omega_{21}^{(1)} & \omega_{22}^{(1)} & \omega_{23}^{(1)} \\ \omega_{31}^{(1)} & \omega_{32}^{(1)} & \omega_{33}^{(1)} \end{bmatrix}$$

$$\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \begin{bmatrix} \omega_{11}^{(1)} & \omega_{12}^{(1)} & \omega_{13}^{(1)} \\ \omega_{21}^{(1)} & \omega_{22}^{(1)} & \omega_{23}^{(1)} \\ \omega_{31}^{(1)} & \omega_{32}^{(1)} & \omega_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}$$

Multi-Layer Perceptron and Back-propagation

- Continue the forward pass



$$\vec{a}^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{(2)}) \\ \sigma(z_2^{(2)}) \\ \sigma(z_3^{(2)}) \end{bmatrix}$$

Continue to the final layer

$$z_1^{(3)} = \omega_{11}^{(2)} a_1^{(2)} + \omega_{12}^{(2)} a_2^{(2)} + \omega_{13}^{(2)} a_3^{(2)} + b_1^{(2)}$$

$$z_2^{(3)} = \omega_{21}^{(2)} a_1^{(2)} + \omega_{22}^{(2)} a_2^{(2)} + \omega_{23}^{(2)} a_3^{(2)} + b_2^{(2)}$$

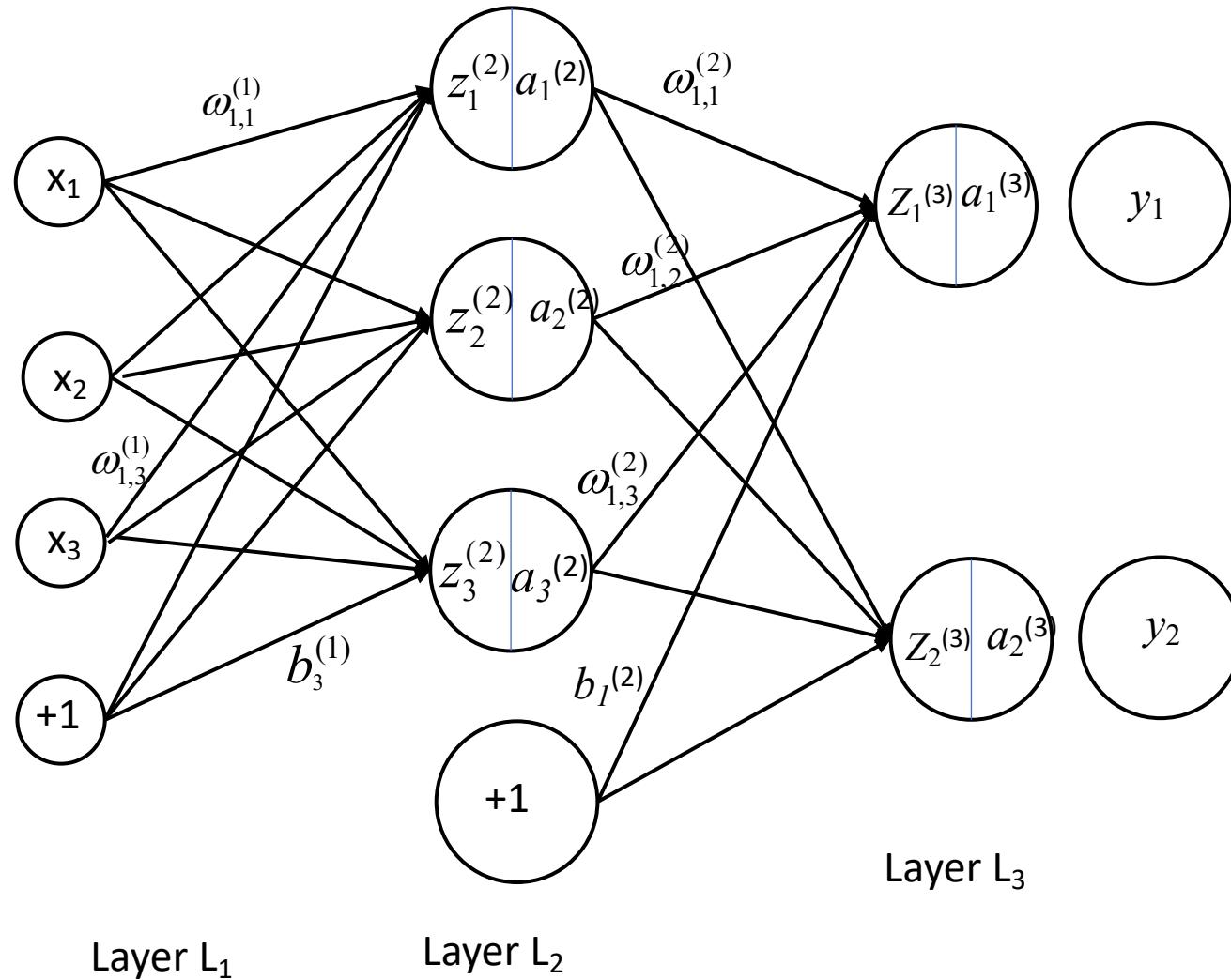
$$\vec{z}^{(3)} = \tilde{\omega}^{(2)} \vec{a}^{(2)}$$

$$\text{where } \vec{z}^{(3)} = \begin{bmatrix} z_1^{(3)} \\ z_2^{(3)} \end{bmatrix} \quad \tilde{\omega}^{(2)} = \begin{bmatrix} \omega_{11}^{(2)} & \omega_{12}^{(2)} & \omega_{13}^{(2)} \\ \omega_{21}^{(2)} & \omega_{22}^{(2)} & \omega_{23}^{(2)} \end{bmatrix}$$

$$\begin{bmatrix} z_1^{(3)} \\ z_2^{(3)} \end{bmatrix} = \begin{bmatrix} \omega_{11}^{(2)} & \omega_{12}^{(2)} & \omega_{13}^{(2)} \\ \omega_{21}^{(2)} & \omega_{22}^{(2)} & \omega_{23}^{(2)} \end{bmatrix} \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \end{bmatrix}$$

Multi-Layer Perceptron and Back-propagation

- Loss function J



$$\begin{aligned} J &= \frac{1}{2} \sum_{i=1}^C (a_i^{(3)} - y_i)^2 \\ &= \frac{1}{2} \left[(a_1^{(3)} - y_1)^2 + (a_2^{(3)} - y_2)^2 \right] \end{aligned}$$

Multi-Layer Perceptron and Back-propagation

- Loss Function

$$\begin{aligned} J &= \frac{1}{2} \left[(\sigma(z_1^{(3)}) - y_1)^2 + (\sigma(z_2^{(3)}) - y_2)^2 \right] \\ &= \frac{1}{2} \left[\sigma(\omega_{11}^{(2)} a_1^{(2)} + \omega_{12}^{(2)} a_2^{(2)} + \omega_{13}^{(2)} a_3^{(2)} + b_1^{(2)}) - y_1 \right]^2 \\ &\quad + \frac{1}{2} \left[\sigma(\omega_{21}^{(2)} a_1^{(2)} + \omega_{22}^{(2)} a_2^{(2)} + \omega_{23}^{(2)} a_3^{(2)} + b_2^{(2)}) - y_2 \right]^2 \\ &= \frac{1}{2} \left[\sigma(\omega_{11}^{(2)} \sigma(z_1^{(2)}) + \omega_{12}^{(2)} \sigma(z_2^{(2)}) + \omega_{13}^{(2)} \sigma(z_3^{(2)}) + b_1^{(2)}) - y_1 \right]^2 \\ &\quad + \frac{1}{2} \left[\sigma(\omega_{21}^{(2)} \sigma(z_1^{(2)}) + \omega_{22}^{(2)} \sigma(z_2^{(2)}) + \omega_{23}^{(2)} \sigma(z_3^{(2)}) + b_2^{(2)}) - y_2 \right]^2 \end{aligned}$$

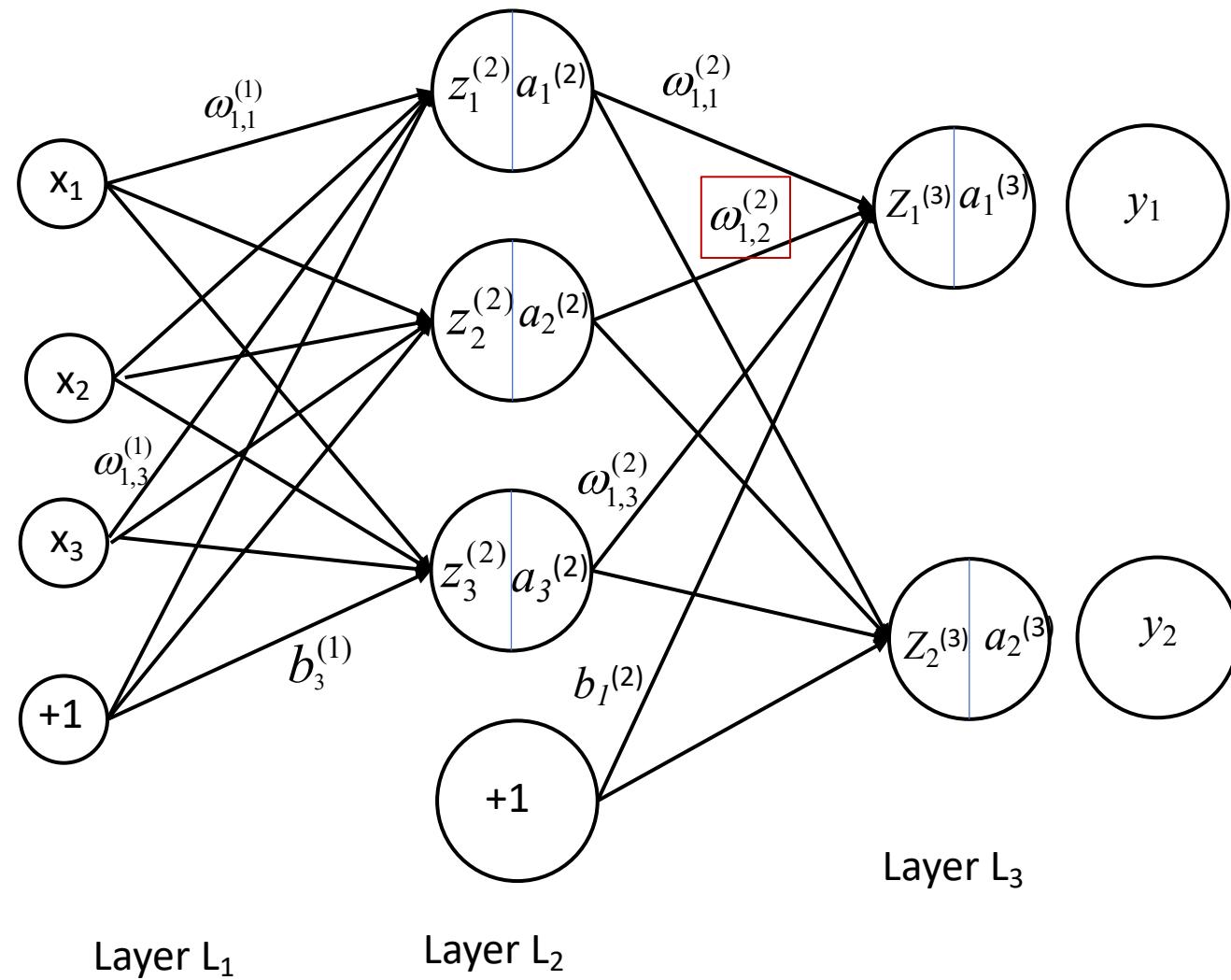
Multi-Layer Perceptron and Back-propagation

$$J = \frac{1}{2} \left[\sigma(\omega_{11}^{(2)} \sigma(\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2 + \omega_{13}^{(1)} x_3 + b_1^{(1)})) + \omega_{12}^{(2)} \sigma(\omega_{21}^{(1)} x_1 + \omega_{22}^{(1)} x_2 + \omega_{23}^{(1)} x_3 + b_2^{(1)}) + \omega_{13}^{(2)} \sigma(\omega_{31}^{(1)} x_1 + \omega_{32}^{(1)} x_2 + \omega_{33}^{(1)} x_3 + b_3^{(1)}) + b_1^{(2)} - y_1 \right]^2 \\ + \frac{1}{2} \left[\sigma(\omega_{21}^{(2)} \sigma(\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2 + \omega_{13}^{(1)} x_3 + b_1^{(1)})) + \omega_{22}^{(2)} \sigma(\omega_{21}^{(1)} x_1 + \omega_{22}^{(1)} x_2 + \omega_{23}^{(1)} x_3 + b_2^{(1)}) + \omega_{23}^{(2)} \sigma(\omega_{31}^{(1)} x_1 + \omega_{32}^{(1)} x_2 + \omega_{33}^{(1)} x_3 + b_3^{(1)}) + b_2^{(2)} - y_2 \right]^2$$



Multi-Layer Perceptron and Back-propagation

- Partial derivative $\frac{\partial J}{\partial \omega_{12}^{(2)}}$ of loss function J



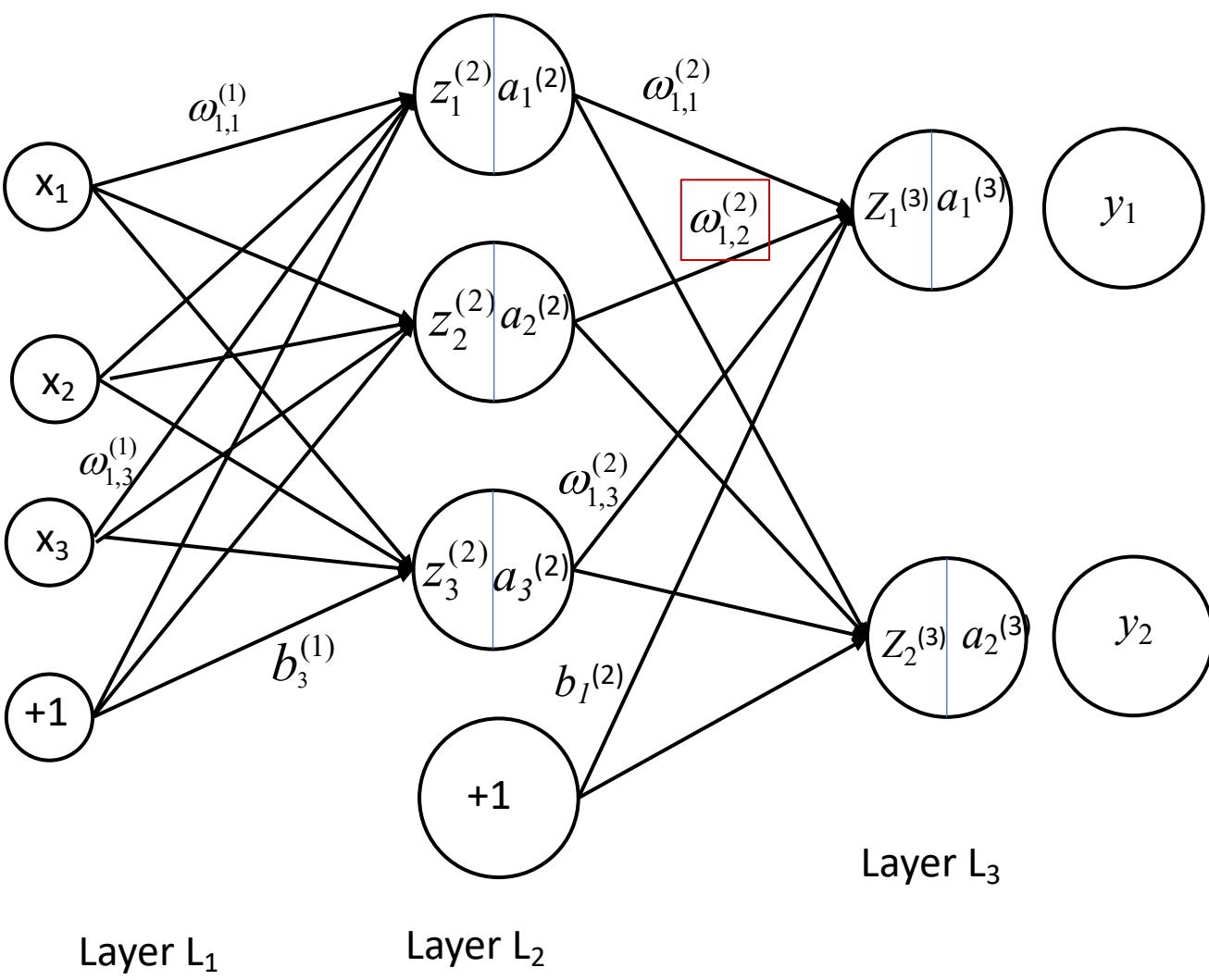
$$\begin{aligned} J &= \frac{1}{2} \sum_{i=1}^C (a_i^{(3)} - y_i)^2 \\ &= \frac{1}{2} \left[(a_1^{(3)} - y_1)^2 + (a_2^{(3)} - y_2)^2 \right] \end{aligned}$$

$$\frac{\partial J}{\partial \omega_{12}^{(2)}} = \frac{\partial}{\partial \omega_{12}^{(2)}} \left\{ \frac{1}{2} \left[(a_1^{(3)} - y_1)^2 + (a_2^{(3)} - y_2)^2 \right] \right\}$$

But

$$\frac{\partial}{\partial \omega_{12}^{(2)}} \left\{ \frac{1}{2} \left[(a_2^{(3)} - y_2)^2 \right] \right\} = 0$$

Multi-Layer Perceptron and Back-propagation



Remaining term

$$\begin{aligned}\frac{\partial J}{\partial \omega_{12}^{(2)}} &= \frac{\partial}{\partial \omega_{12}^{(2)}} \left\{ \frac{1}{2} \left[(a_1^{(3)} - y_1)^2 \right] \right\} \\ &= (a_1^{(3)} - y_1) \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \omega_{12}^{(2)}}\end{aligned}$$

Consider the second term

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = \frac{\partial \sigma(z_1^{(3)})}{\partial z_1^{(3)}}$$

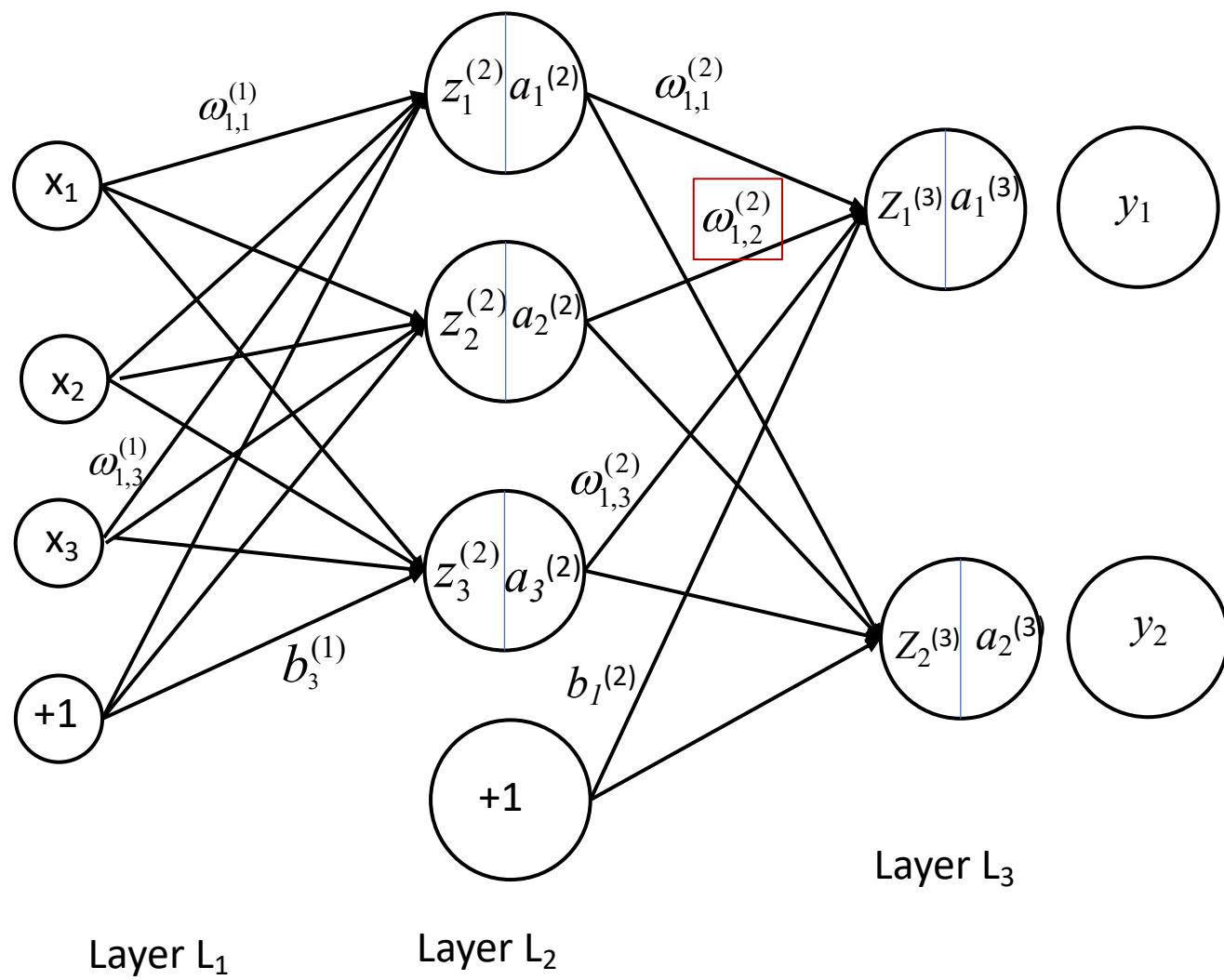
Recall derivative of sigmoid

$$g' = g(1-g) \text{ and } g(z_1^{(3)}) = a_1^{(3)}$$

Thus

$$\frac{\partial \sigma(z_1^{(3)})}{\partial z_1^{(3)}} = f'(z_1^{(3)}) = a_1^{(3)}(1 - a_1^{(3)})$$

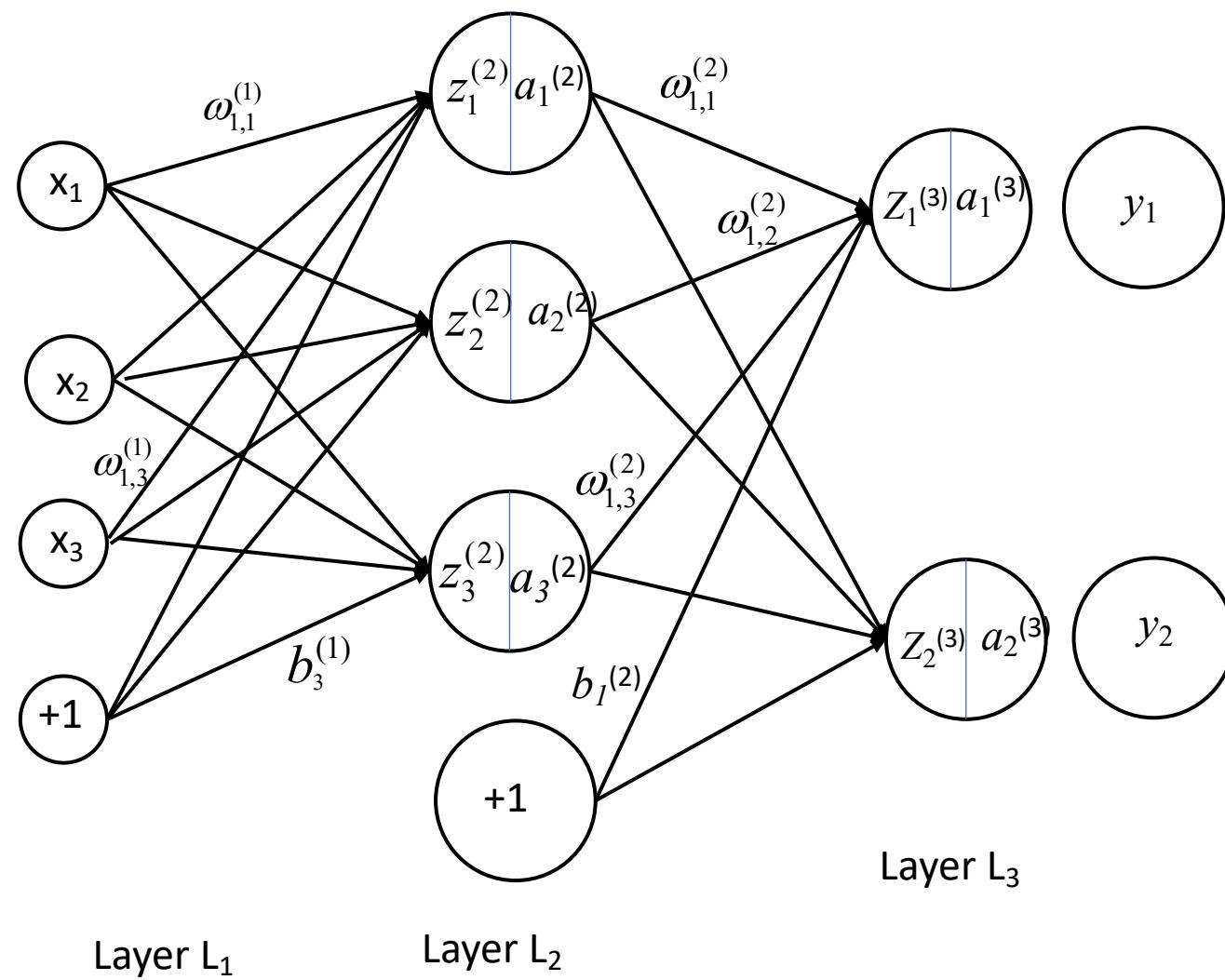
Multi-Layer Perceptron and Back-propagation



Remaining term

$$\begin{aligned}\frac{\partial J}{\partial \omega_{12}^{(2)}} &= \frac{\partial}{\partial \omega_{12}^{(2)}} \left\{ \frac{1}{2} \left[(a_1^{(3)} - y_1)^2 \right] \right\} \\ &= (a_1^{(3)} - y_1) \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \omega_{12}^{(2)}} \\ &= (a_1^{(3)} - y_1) a_1^{(3)} (1 - a_1^{(3)}) a_2^{(2)}\end{aligned}$$

Multi-Layer Perceptron and Back-propagation

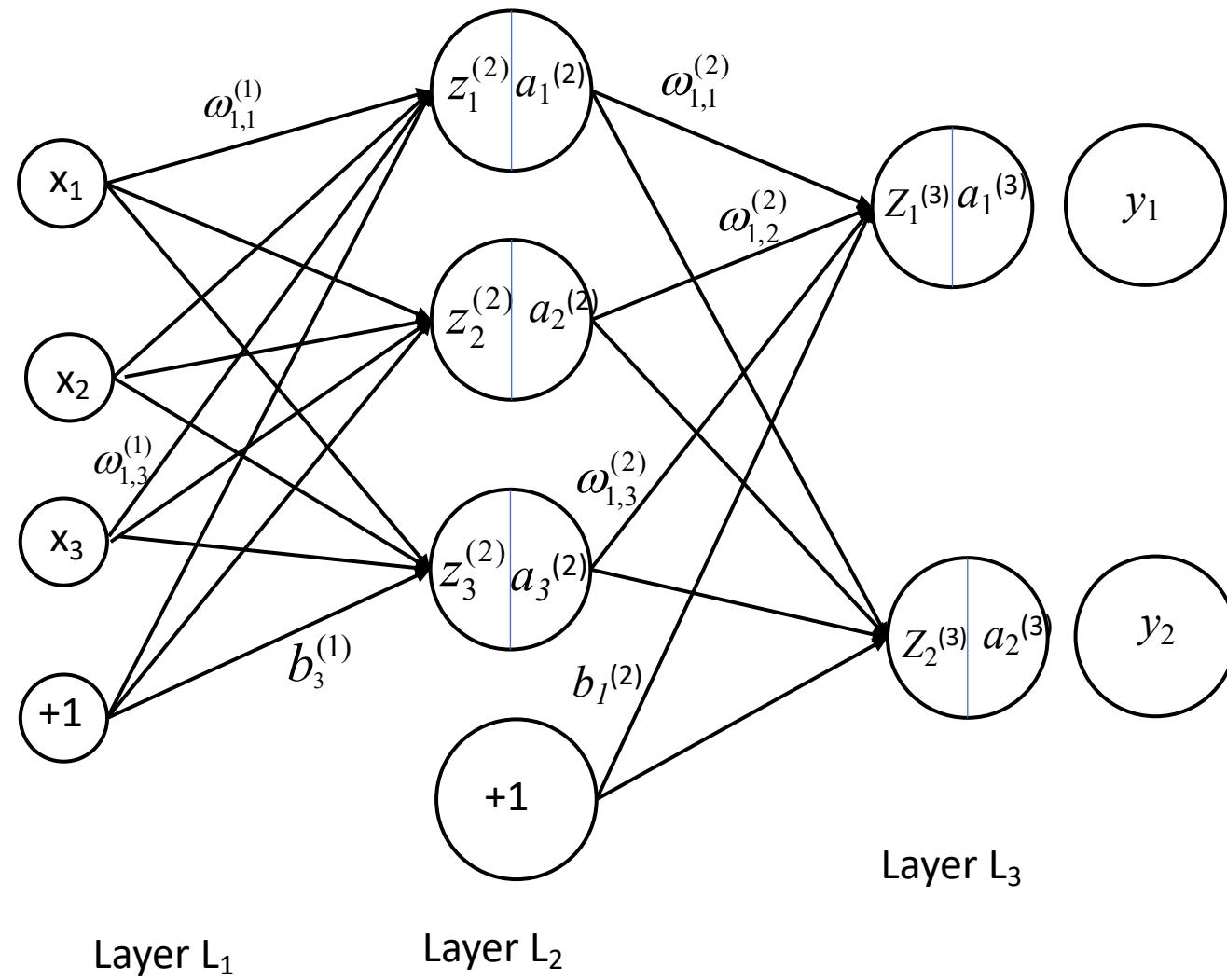


$$\begin{aligned}\frac{\partial J}{\partial \omega_{12}^{(2)}} &= \frac{\partial J}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \omega_{12}^{(2)}} \\ &= \frac{\partial J}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \omega_{12}^{(2)}} = \delta_1^{(3)} \frac{\partial z_1^{(3)}}{\partial \omega_{12}^{(2)}}\end{aligned}$$

where

$$\begin{aligned}\delta_1^{(3)} &\equiv \frac{\partial J}{\partial z_1^{(3)}} \\ &= \frac{\partial J}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = (a_1^{(3)} - y_1) f'(z_1^{(3)})\end{aligned}$$

Multi-Layer Perceptron and Back-propagation



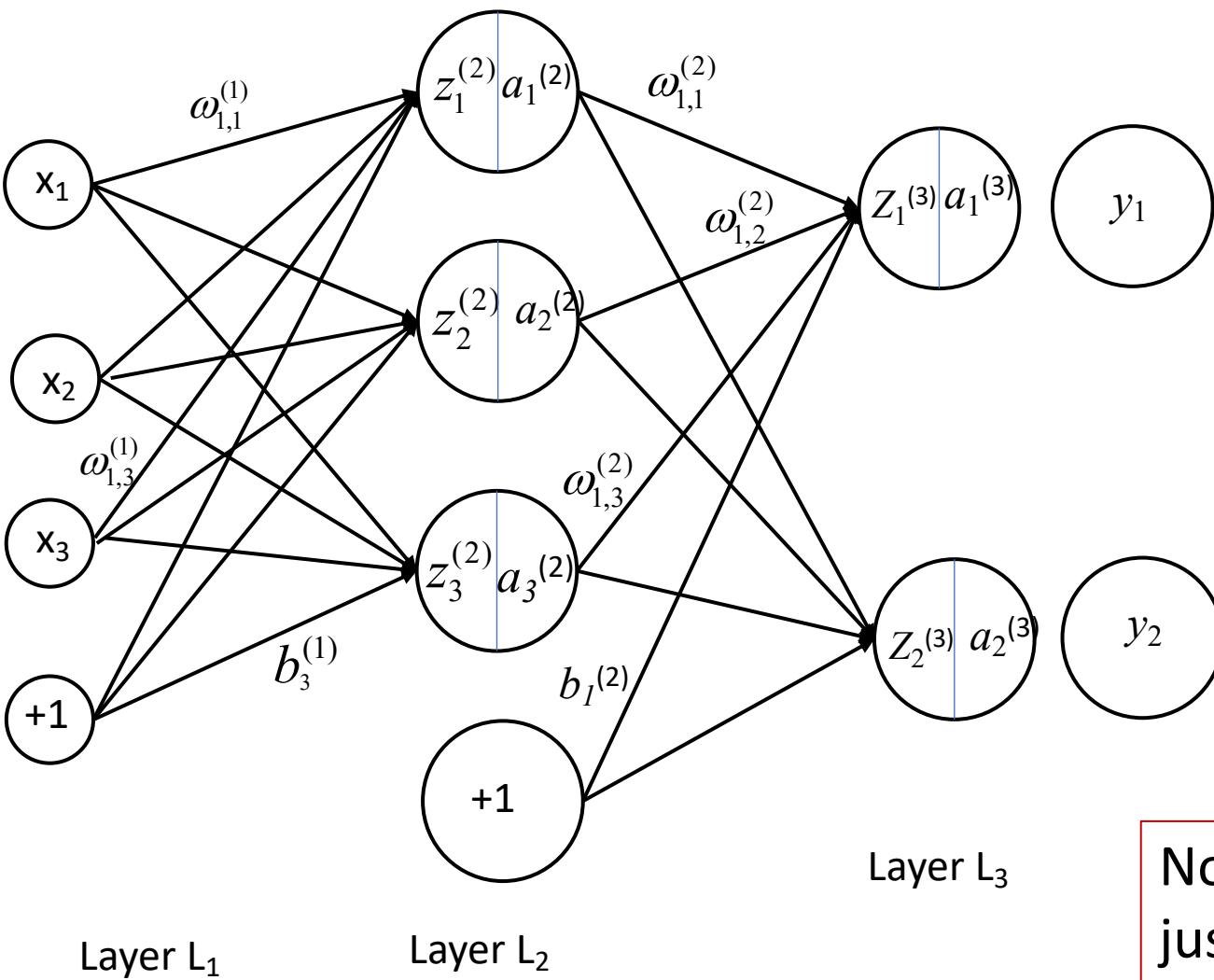
The partial derivative

$$\frac{\partial J}{\partial \omega_{12}^{(2)}} = \delta_1^{(3)} \frac{\partial z_1^{(3)}}{\partial \omega_{12}^{(2)}} = \delta_1^{(3)} a_2^{(2)}$$

Similarly, define

$$\begin{aligned}\delta_2^{(3)} &\equiv \frac{\partial J}{\partial z_2^{(3)}} \\ &= \frac{\partial J}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} = (a_2^{(3)} - y_2) f'(z_2^{(3)})\end{aligned}$$

Multi-Layer Perceptron and Back-propagation



When the outermost layer is N

$$\frac{\partial J}{\partial z_i^{(N)}} = \frac{\partial J}{\partial a_i^{(N)}} \frac{\partial a_i^{(N)}}{\partial z_i^{(N)}} = (a_i^{(N)} - y_i)a_i^{(N)}(1 - a_i^{(N)})$$

and

$$\delta_i^{(N)} \equiv \frac{\partial J}{\partial z_i^{(N)}}$$

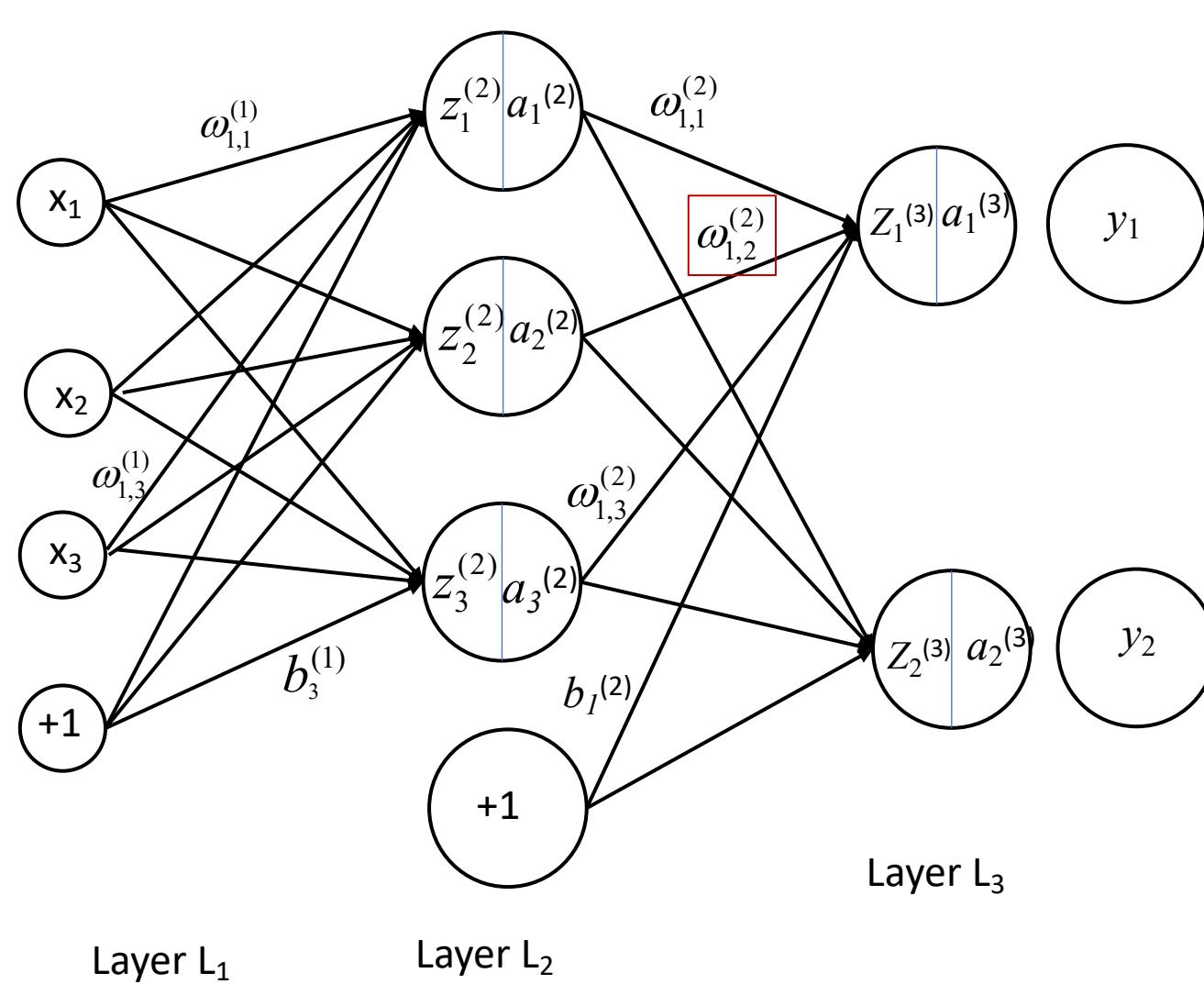
Additionally

$$\begin{aligned}\frac{\partial J}{\partial \omega_{i,j}^{(N-1)}} &= (a_i^{(N)} - y_i)a_i^{(N)}(1 - a_i^{(N)})a_j^{(N-1)} = \frac{\partial J}{\partial z_i^{(N)}}a_j^{(N-1)} \\ &= \delta_i^{(N)}a_j^{(N-1)}\end{aligned}$$

Note that these are only applicable at a layer just under the outermost layer N !

Multi-Layer Perceptron and Back-propagation

- Similarly for $b_i^{(3)}$



$$\begin{aligned}
 \frac{\partial J}{\partial b_i^{(2)}} &= \frac{\partial}{\partial b_i^{(2)}} \left\{ \frac{1}{2} \left[(a_1^{(3)} - y_1)^2 + (a_2^{(3)} - y_2)^2 \right] \right\} \\
 &= \frac{\partial}{\partial b_i^{(2)}} \left\{ \frac{1}{2} \left[(a_i^{(3)} - y_i)^2 \right] \right\} \\
 &= (a_i^{(3)} - y_i) \frac{\partial a_i^{(3)}}{\partial b_i^{(2)}} = (a_i^{(3)} - y_i) \frac{\partial a_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial b_i^{(2)}} \\
 &= (a_i^{(3)} - y_i) f'(z_i^{(3)}) = \delta_i^{(3)}
 \end{aligned}$$

For a general case of outerlayer N

$$= \frac{\partial J}{\partial b_i^{(N-1)}} = (a_i^{(N)} - y_i) f'(z_i^{(N)}) = \delta_i^{(N)}$$

Multi-Layer Perceptron and Back-propagation

- For an output node, the difference between the network's activation and the true target value ($a_i^{(N)} - y_i$) can be directly measured.
- $\delta_i^{(N)}$ can be used now to compute the partial derivative for weights on layer N-1

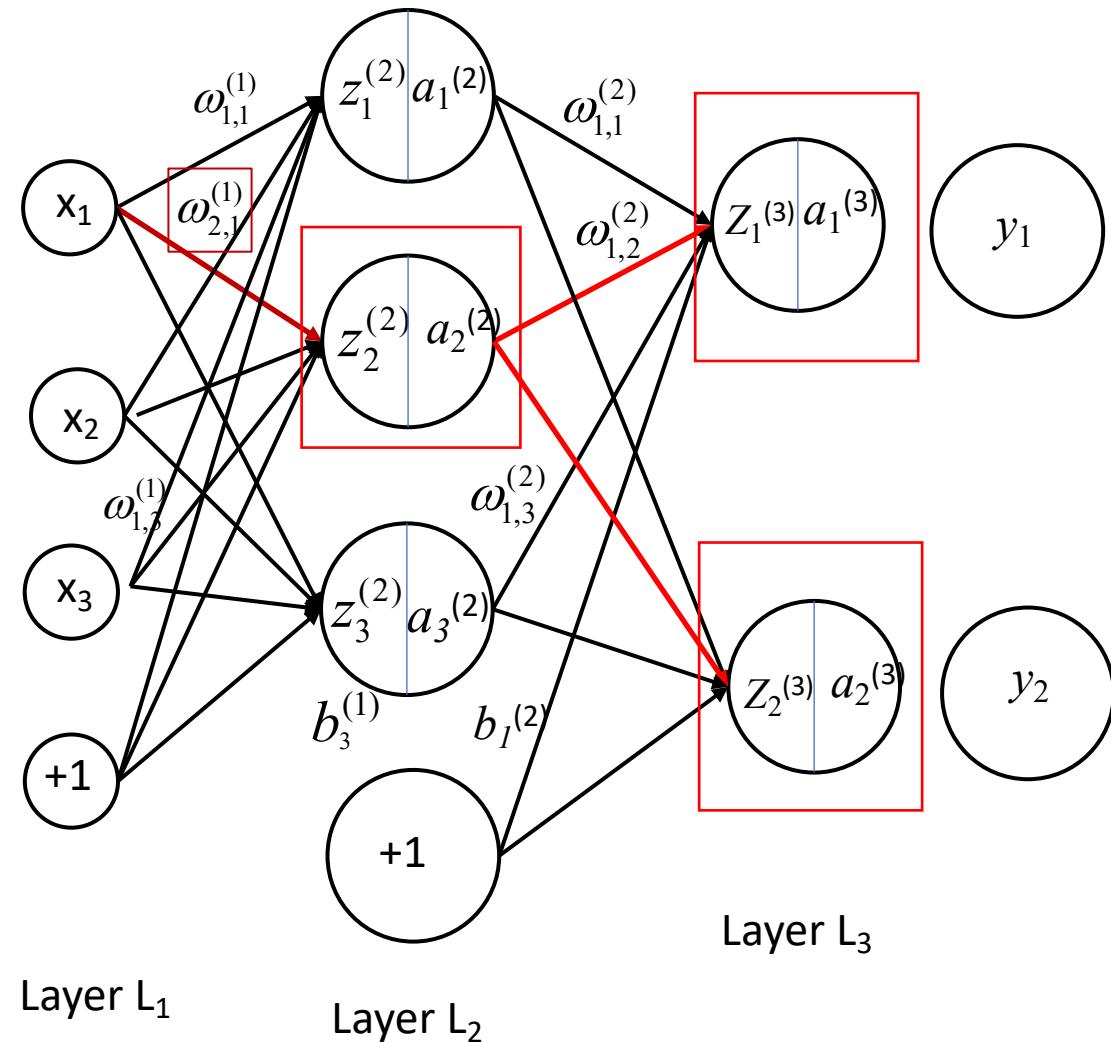
Since $\delta_i^{(N)} = \frac{\partial J}{\partial z_i^{(N)}} = (a_i^{(N)} - y_i) a_i^{(N)} (1 - a_i^{(N)})$

$$\frac{\partial J}{\partial \omega_{i,j}^{(N-1)}} = \frac{\partial J}{\partial z_i^{(N)}} \frac{\partial z_i^{(N)}}{\partial \omega_{i,j}^{(N-1)}} = \delta_i^{(N)} a_i^{(N-1)}$$
$$\frac{\partial J}{\partial b_i^{(N-1)}} = \delta_i^{(N)}$$

Multi-Layer Perceptron and Back-propagation

- Consider a weight $\omega_{2,1}^{(1)}$ in layer 1

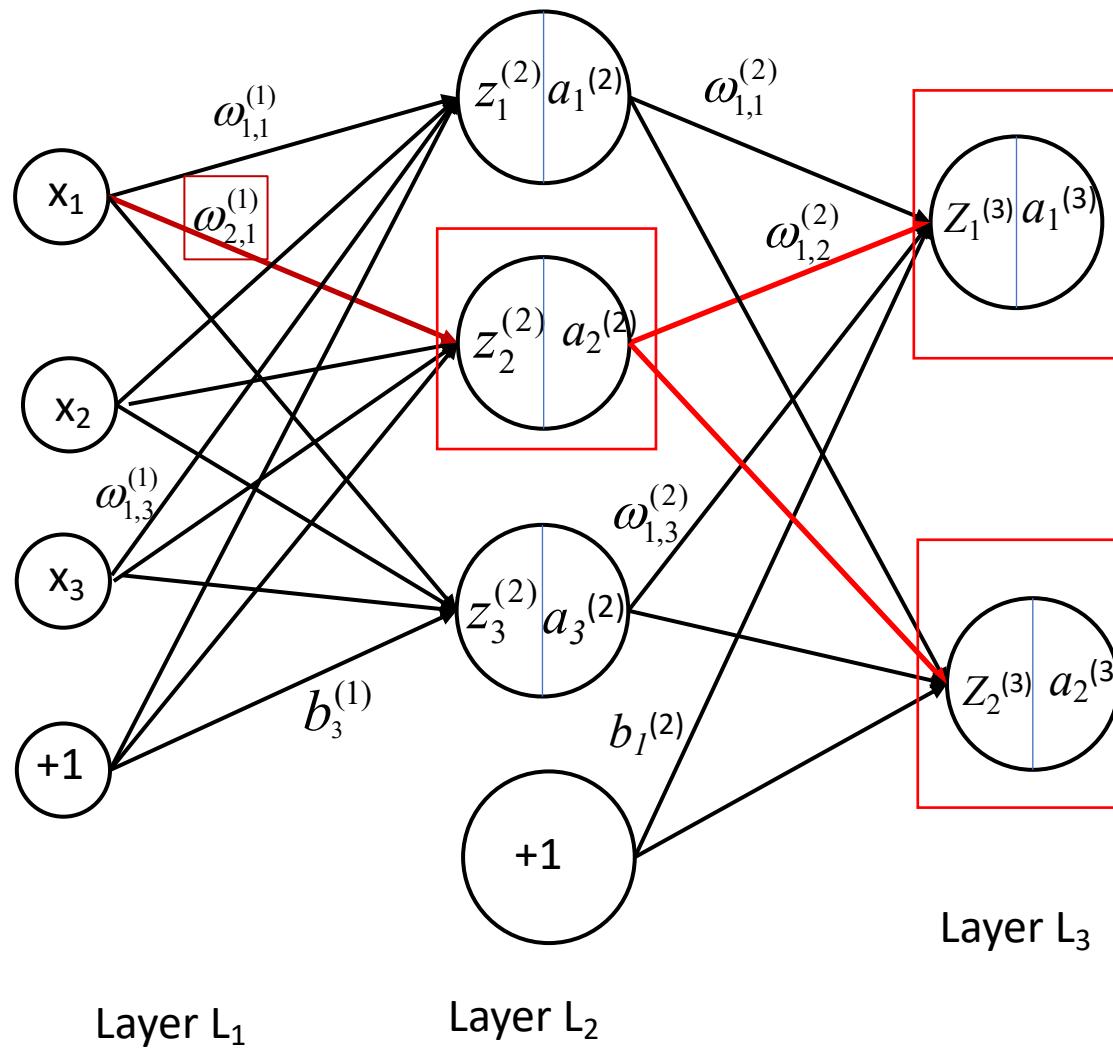
- Note the influence of $\omega_{2,1}^{(1)}$ to the loss function



$$J = \frac{1}{2} \left[(a_1^{(3)} - y_1)^2 + (a_2^{(3)} - y_2)^2 \right]$$

$$\begin{aligned}\frac{\partial J}{\partial \omega_{21}^{(1)}} &= (a_1^{(3)} - y_1) \frac{\partial a_1^{(3)}}{\partial \omega_{21}^{(1)}} + (a_2^{(3)} - y_2) \frac{\partial a_2^{(3)}}{\partial \omega_{21}^{(1)}} \\ &= (a_1^{(3)} - y_1) \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \omega_{21}^{(1)}} + (a_2^{(3)} - y_2) \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \frac{\partial z_2^{(3)}}{\partial \omega_{21}^{(1)}} \\ &= (a_1^{(3)} - y_1) f'(z_1^{(3)}) \frac{\partial z_1^{(3)}}{\partial \omega_{21}^{(1)}} + (a_2^{(3)} - y_2) f'(z_2^{(3)}) \frac{\partial z_2^{(3)}}{\partial \omega_{21}^{(1)}}\end{aligned}$$

Multi-Layer Perceptron and Back-propagation

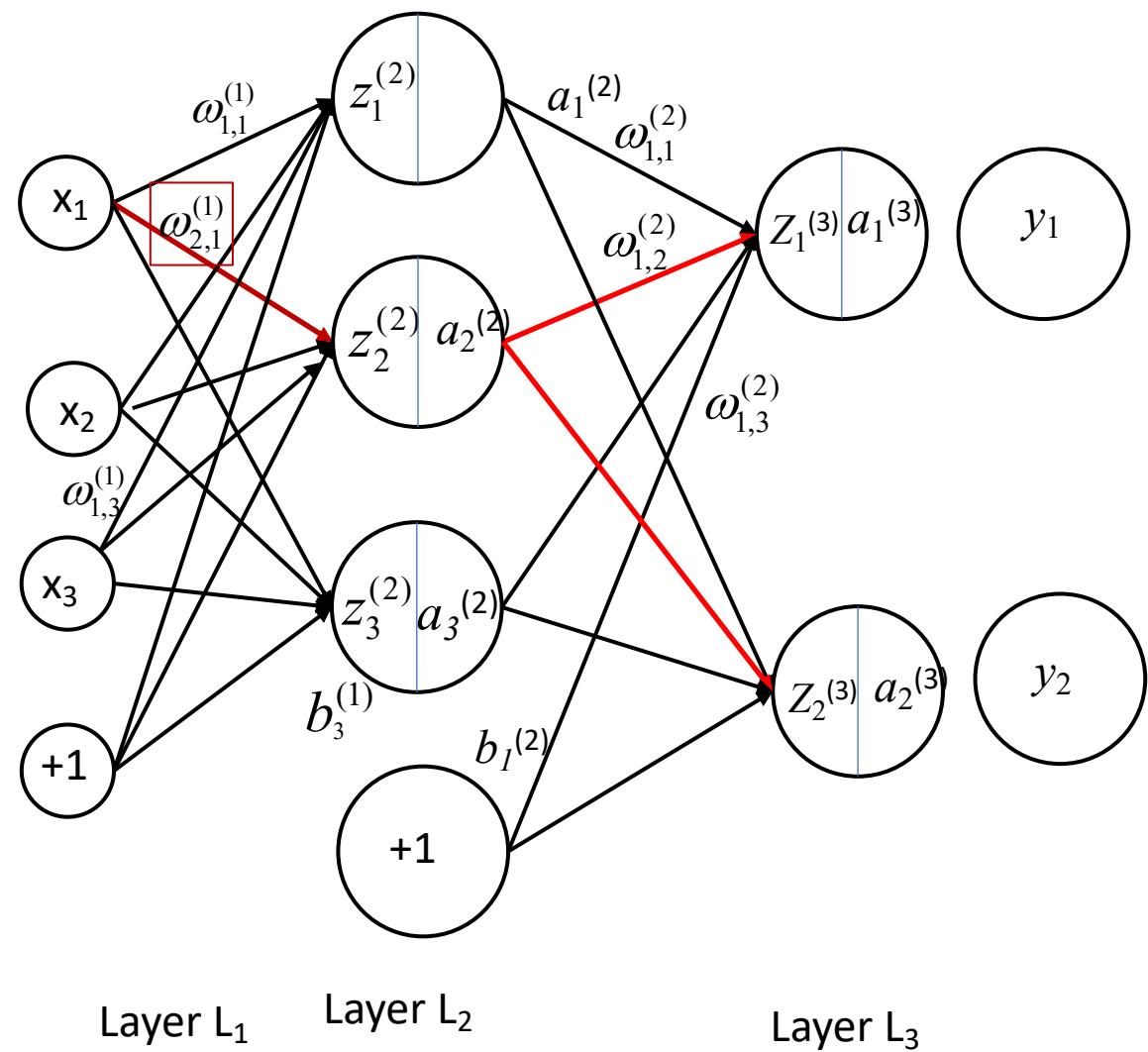


$$\begin{aligned}
 \frac{\partial z_1^{(3)}}{\partial \omega_{21}^{(1)}} &= \frac{\partial}{\partial \omega_{21}^{(1)}} \left[\omega_{11}^{(2)} a_1^{(2)} + \omega_{12}^{(2)} a_2^{(2)} + \omega_{13}^{(2)} a_3^{(2)} + b_1^{(2)} \right] \\
 &= \frac{\partial}{\partial \omega_{21}^{(1)}} \left[\omega_{12}^{(2)} a_2^{(2)} \right] = \omega_{12}^{(2)} \frac{\partial a_2^{(2)}}{\partial \omega_{21}^{(1)}} = \omega_{12}^{(2)} \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \frac{\partial z_2^{(2)}}{\partial \omega_{21}^{(1)}} \\
 &= \omega_{12}^{(2)} f'(z_2^{(2)}) x_1
 \end{aligned}$$

Similarly

$$\begin{aligned}
 \frac{\partial z_2^{(3)}}{\partial \omega_{21}^{(1)}} &= \frac{\partial}{\partial \omega_{21}^{(1)}} \left[\omega_{21}^{(2)} a_1^{(2)} + \omega_{22}^{(2)} a_2^{(2)} + \omega_{23}^{(2)} a_3^{(2)} + b_2^{(2)} \right] \\
 &= \frac{\partial}{\partial \omega_{21}^{(1)}} \left[\omega_{22}^{(2)} a_2^{(2)} \right] = \omega_{22}^{(2)} \frac{\partial a_2^{(2)}}{\partial \omega_{21}^{(1)}} = \omega_{22}^{(2)} \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \frac{\partial z_2^{(2)}}{\partial \omega_{21}^{(1)}} \\
 &= \omega_{22}^{(2)} f'(z_2^{(2)}) x_1
 \end{aligned}$$

Multi-Layer Perceptron and Back-propagation



Putting them together

$$\begin{aligned}\frac{\partial J}{\partial \omega_{21}^{(1)}} &= (a_1^{(3)} - y_1) f'(z_1^{(3)}) \omega_{12}^{(2)} f'(z_2^{(2)}) x_1 \\ &\quad + (a_2^{(3)} - y_2) f'(z_2^{(3)}) \omega_{22}^{(2)} f'(z_2^{(2)}) x_1 \\ &= \delta_1^{(3)} \omega_{12}^{(2)} f'(z_2^{(2)}) x_1 + \delta_2^{(3)} \omega_{22}^{(2)} f'(z_2^{(2)}) x_1 \\ &= (\delta_1^{(3)} \omega_{12}^{(2)} + \delta_2^{(3)} \omega_{22}^{(2)}) f'(z_2^{(2)}) x_1\end{aligned}$$

Multi-Layer Perceptron and Back-propagation

From the previous derivation

$$\frac{\partial J}{\partial \omega_{21}^{(1)}} = (\delta_1^{(3)} \omega_{12}^{(2)} + \delta_2^{(3)} \omega_{22}^{(2)}) f'(z_2^{(2)}) x_1$$

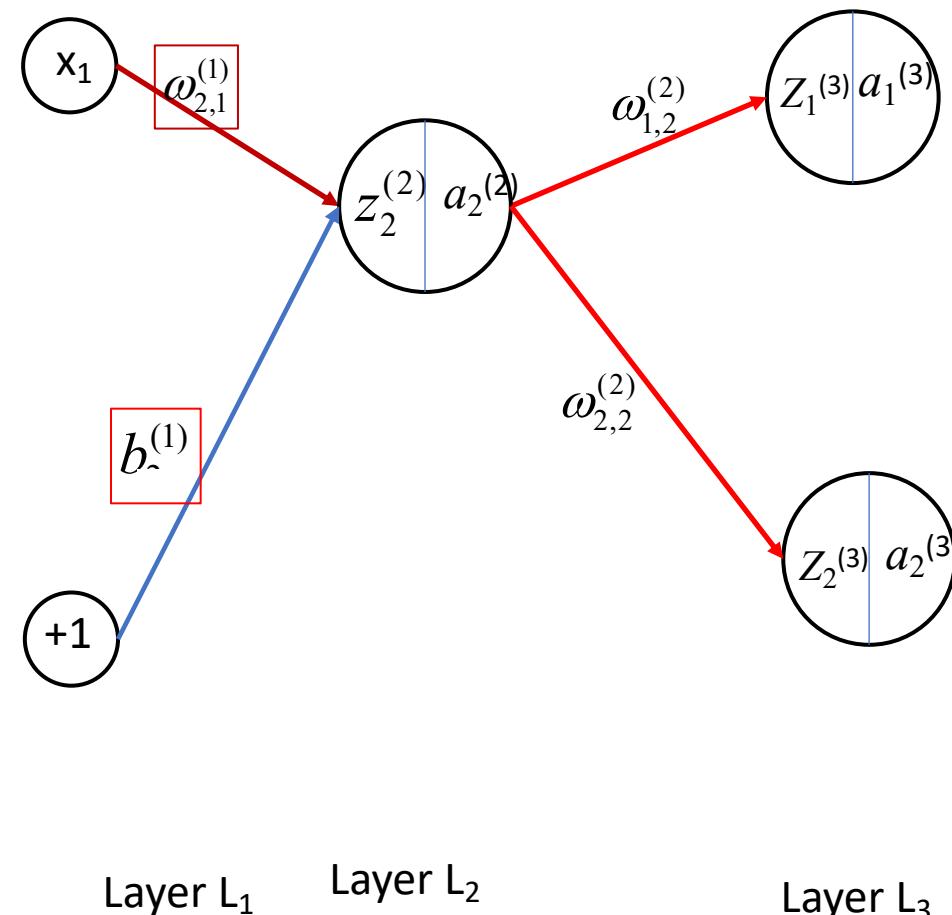
Can be shown similarly

$$\frac{\partial J}{\partial b_2^{(1)}} = (\delta_1^{(3)} \omega_{12}^{(2)} + \delta_2^{(3)} \omega_{22}^{(2)}) f'(z_2^{(2)})$$

Notice the flow of information from upper layer to the lower layer

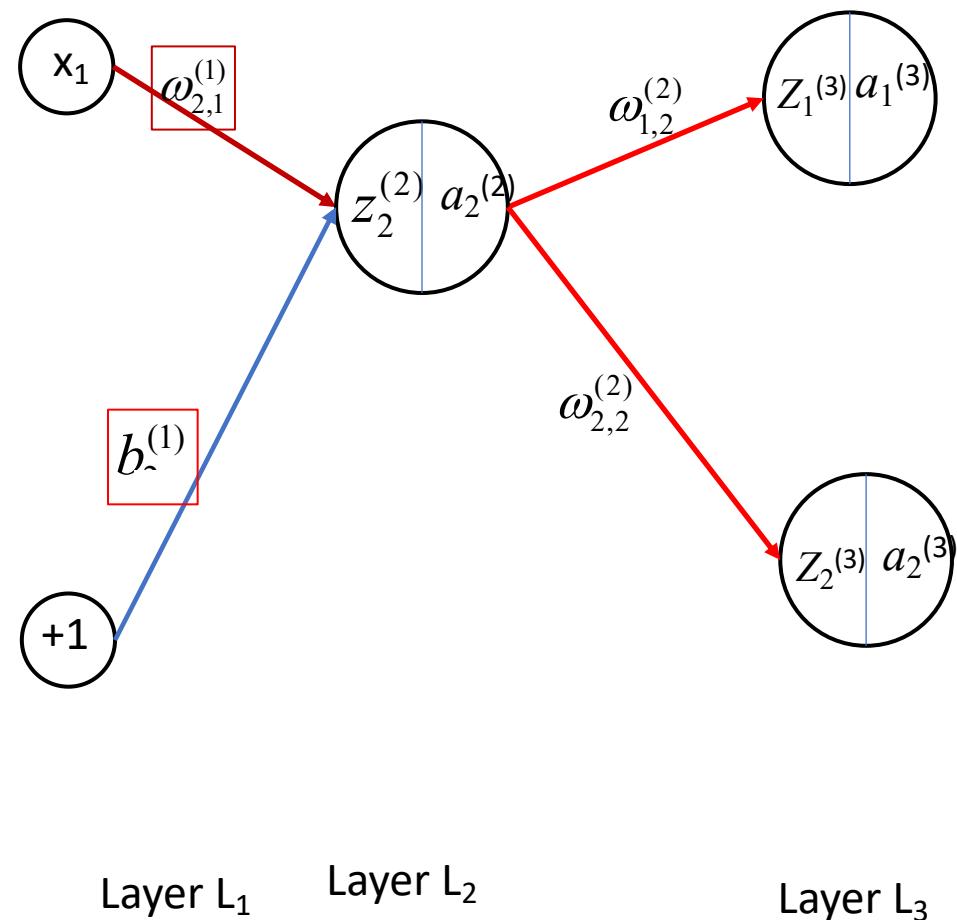
$$\begin{aligned} \text{Let } \delta_2^{(2)} &= (\delta_1^{(3)} \omega_{12}^{(2)} + \delta_2^{(3)} \omega_{22}^{(2)}) f'(z_2^{(2)}) \\ &= \sum_{j=1}^2 \delta_j^{(3)} \omega_{j2}^{(2)} f'(z_2^{(2)}) \end{aligned}$$

$$\text{Thus } \frac{\partial J}{\partial \omega_{21}^{(1)}} = \delta_2^{(2)} x_1, \quad \frac{\partial J}{\partial b_2^{(1)}} = \delta_2^{(2)}$$



Multi-Layer Perceptron and Back-propagation

- The gradient descent update rule can be applied now



$$\begin{aligned}\omega_{21}^{(1)} &:= \omega_{21}^{(1)} - \alpha \frac{\partial J}{\partial \omega_{21}^{(1)}} = \omega_{21}^{(1)} - \alpha \delta_2^{(2)} x_1 \\ b_2^{(1)} &:= b_2^{(1)} - \alpha \delta_2^{(2)}\end{aligned}$$

$$\delta_2^{(2)} = \sum_{j=1}^2 \delta_j^{(3)} \omega_{j2}^{(2)} f'(z_2^{(2)})$$

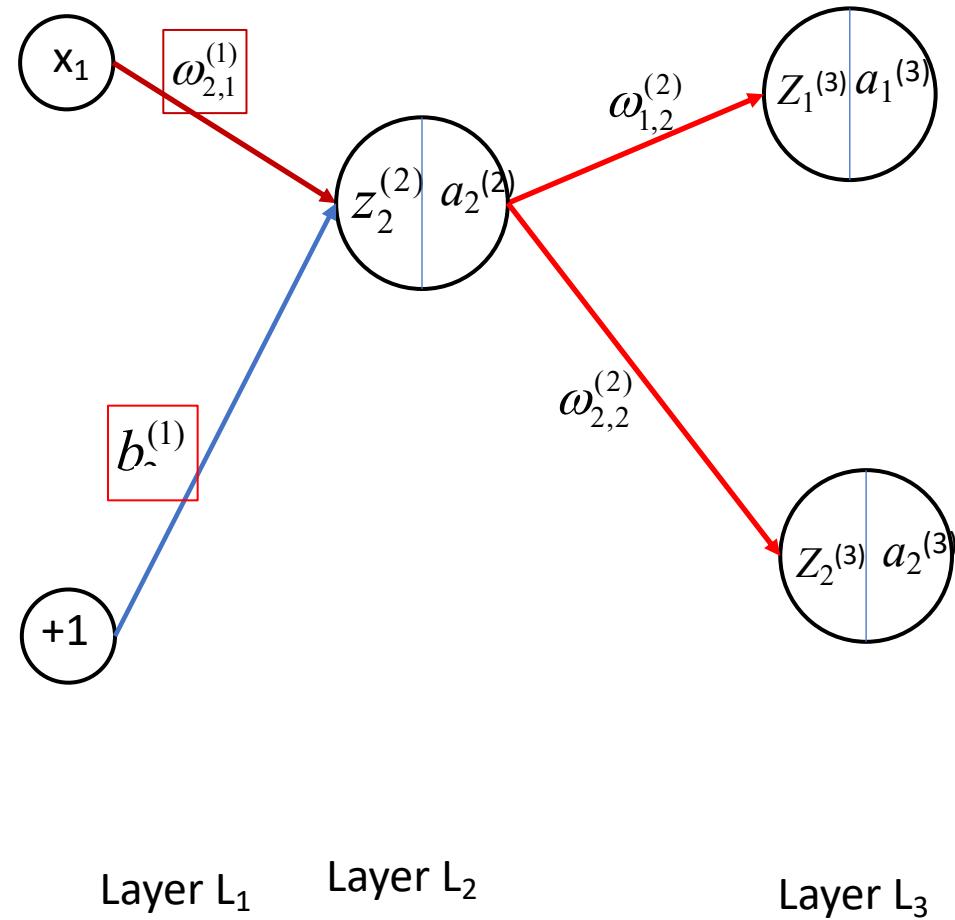
More generally

$$\delta_i^{(l)} = \left(\sum_{j=1}^{S_{l+1}} \delta_j^{(l+1)} \omega_{ji}^{(l)} \right) f'(z_i^{(l)})$$

Where S_{l+1} = Number of nodes on layer $l + 1$

Multi-Layer Perceptron and Back-propagation

More generally



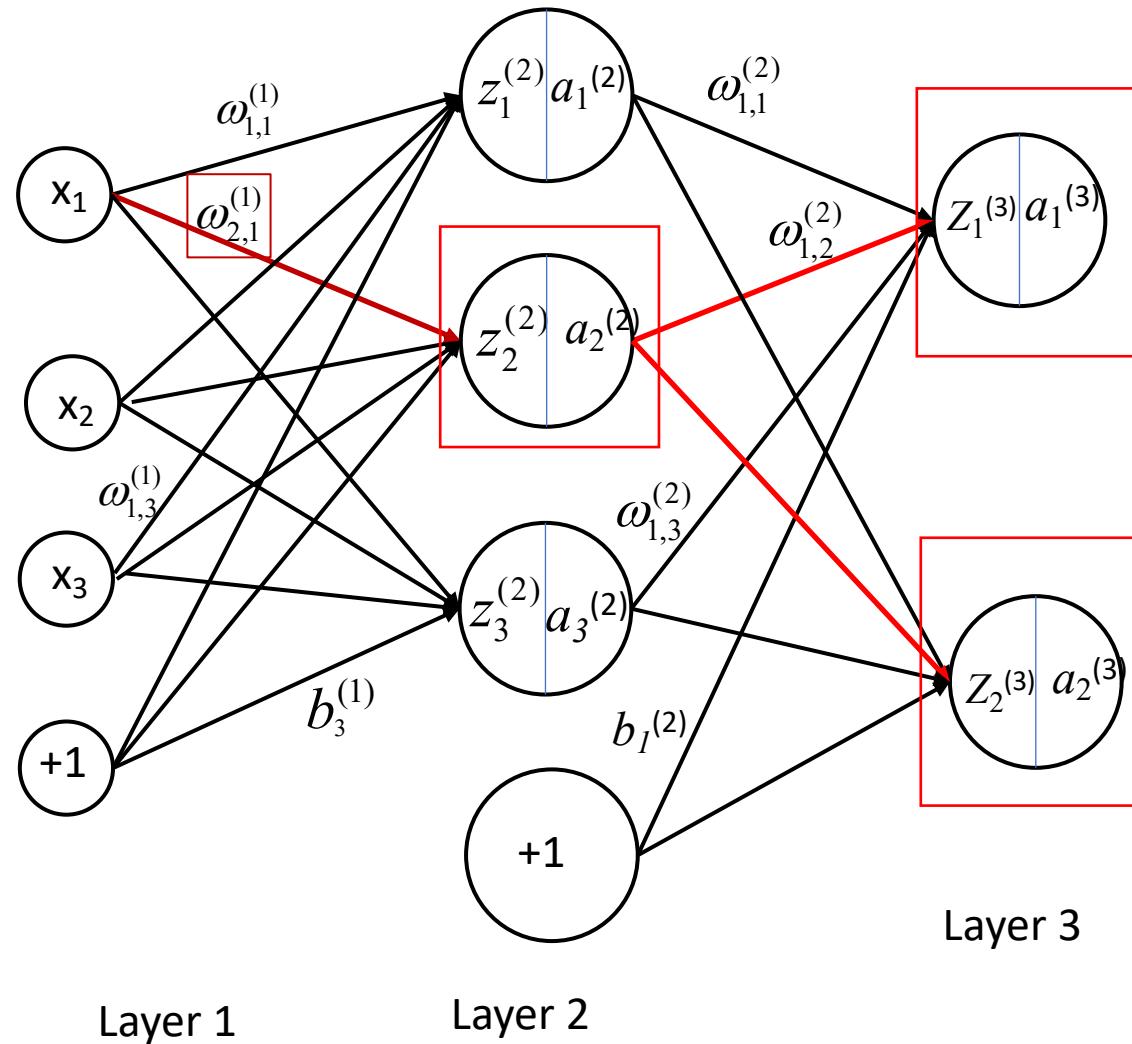
$$\frac{\partial J}{\partial \omega_{ij}^{(l-1)}} = \delta_i^{(l)} a_j^{(l-1)}, \quad \frac{\partial J}{\partial b_i^{(l-1)}} = \delta_i^{(l)}$$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{S_{l+1}} \delta_j^{(l+1)} \omega_{ji}^{(l)} \right) f'(z_i^{(l)})$$

Where S_{l+1} = Number of nodes on layer $l + 1$

Multi-Layer Perceptron and Back-propagation

- Let's train starting the outer layer at $N=3$



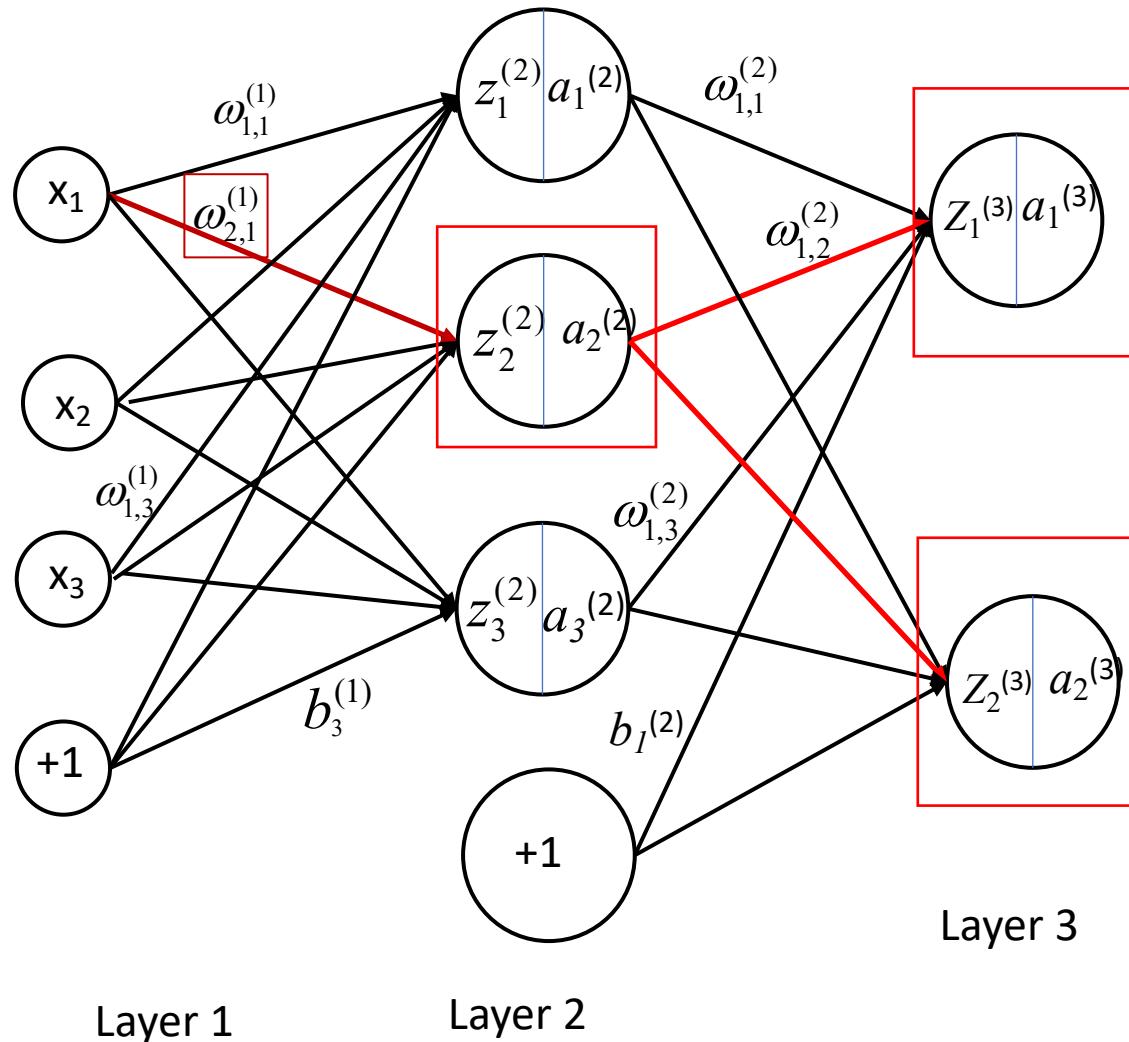
$$\frac{\partial J}{\partial z_i^{(3)}} = \delta_i^{(3)} = (a_i^{(3)} - y_i)a_i^{(3)}(1 - a_i^{(3)})$$

$$\frac{\partial J}{\partial \omega_{i,j}^{(2)}} = \delta_i^{(3)}a_j^{(2)}, \quad \frac{\partial J}{\partial b_i} = \delta_i^{(3)}$$

$$\begin{aligned}\frac{\partial J}{\partial \omega_{i,j}^{(2)}} &= \delta_i^{(3)}a_j^{(2)} = (a_i^{(3)} - y_i)a_i^{(3)}(1 - a_i^{(3)})a_j^{(2)} \\ \frac{\partial J}{\partial b_i} &= (a_i^{(3)} - y_i)a_i^{(3)}(1 - a_i^{(3)})\end{aligned}$$

Multi-Layer Perceptron and Back-propagation

- Training the outer layer at $N=3$



$$\frac{\partial J}{\partial \omega_{11}^{(2)}} = \delta_1^{(3)} a_1^{(2)} = (a_1^{(3)} - y_1) a_1^{(3)} (1 - a_1^{(3)}) a_1^{(2)}$$

$$\frac{\partial J}{\partial \omega_{12}^{(2)}} = \delta_1^{(3)} a_2^{(2)} = (a_1^{(3)} - y_1) a_1^{(3)} (1 - a_1^{(3)}) a_2^{(2)}$$

$$\frac{\partial J}{\partial \omega_{13}^{(2)}} = \delta_1^{(3)} a_3^{(2)} = (a_1^{(3)} - y_1) a_1^{(3)} (1 - a_1^{(3)}) a_3^{(2)}$$

$$\frac{\partial J}{\partial \omega_{21}^{(2)}} = \delta_2^{(3)} a_1^{(2)} = (a_2^{(3)} - y_2) a_2^{(3)} (1 - a_2^{(3)}) a_1^{(2)}$$

$$\frac{\partial J}{\partial \omega_{22}^{(2)}} = \delta_2^{(3)} a_2^{(2)} = (a_2^{(3)} - y_2) a_2^{(3)} (1 - a_2^{(3)}) a_2^{(2)}$$

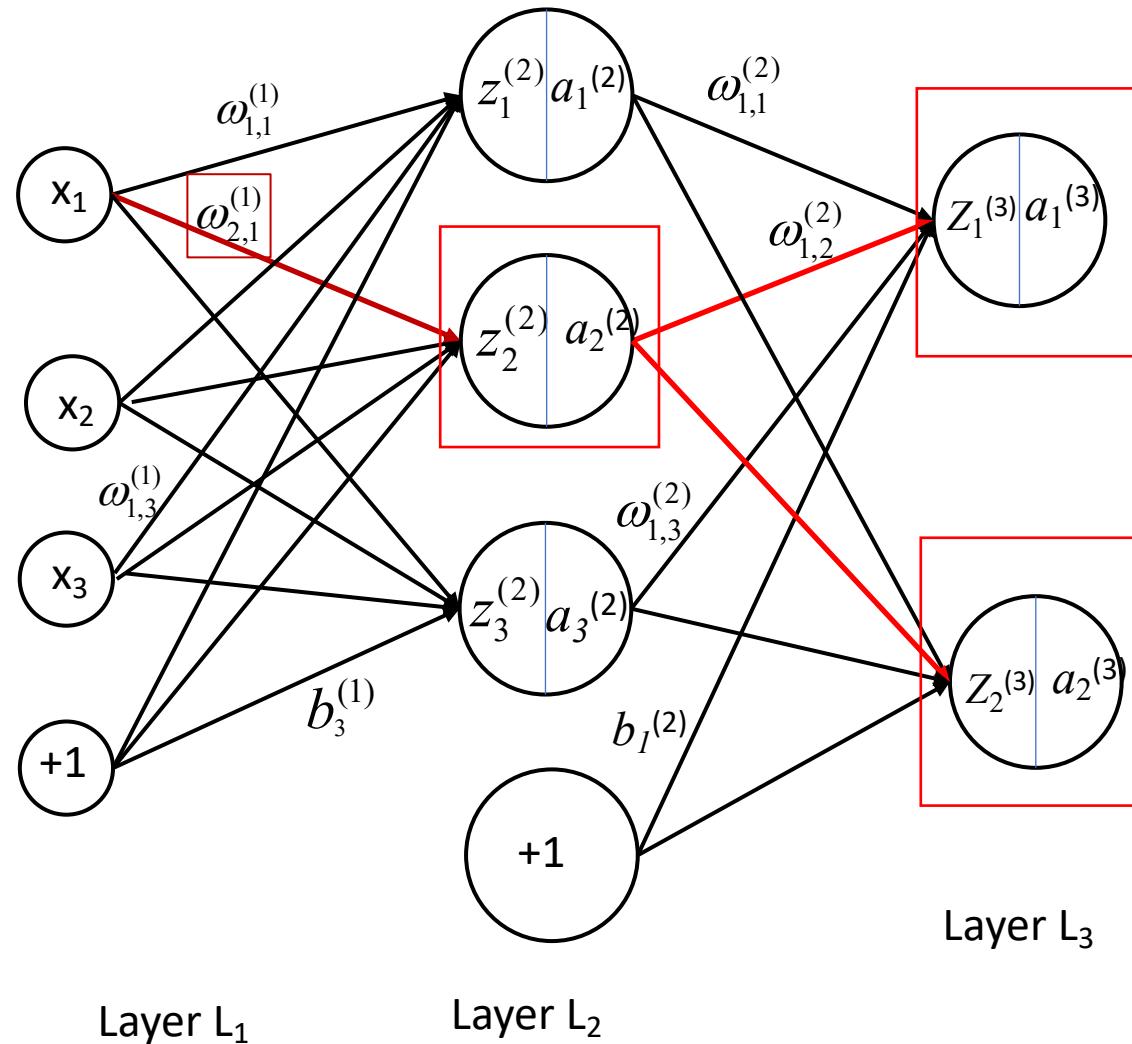
$$\frac{\partial J}{\partial \omega_{23}^{(2)}} = \delta_2^{(3)} a_3^{(2)} = (a_2^{(3)} - y_2) a_2^{(3)} (1 - a_2^{(3)}) a_3^{(2)}$$

$$\frac{\partial J}{\partial b_1^{(2)}} = (a_1^{(3)} - y_1) a_1^{(3)} (1 - a_1^{(3)})$$

$$\frac{\partial J}{\partial b_2^{(2)}} = (a_2^{(3)} - y_2) a_2^{(3)} (1 - a_2^{(3)})$$

Multi-Layer Perceptron and Back-propagation

- Training the inner layer at 2



$$\frac{\partial J}{\partial \omega_{ij}^{(l-1)}} = \delta_i^{(l)} a_j^{(l-1)}, \quad \frac{\partial J}{\partial b_i} = \delta_i^{(l)}$$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{S_{l+1}} \delta_j^{(l+1)} \omega_{ji}^{(l)} \right) f'(z_i^{(l)})$$

Where S_{l+1} = Number of nodes on layer $l+1$

$l = 2, S_{l+1}$ = Nodes at layer ($l+1 = 3$): 2

$$\delta_i^{(2)} = \left(\sum_{j=1}^2 \delta_j^{(3)} \omega_{ji}^{(2)} \right) f'(z_i^{(2)})$$

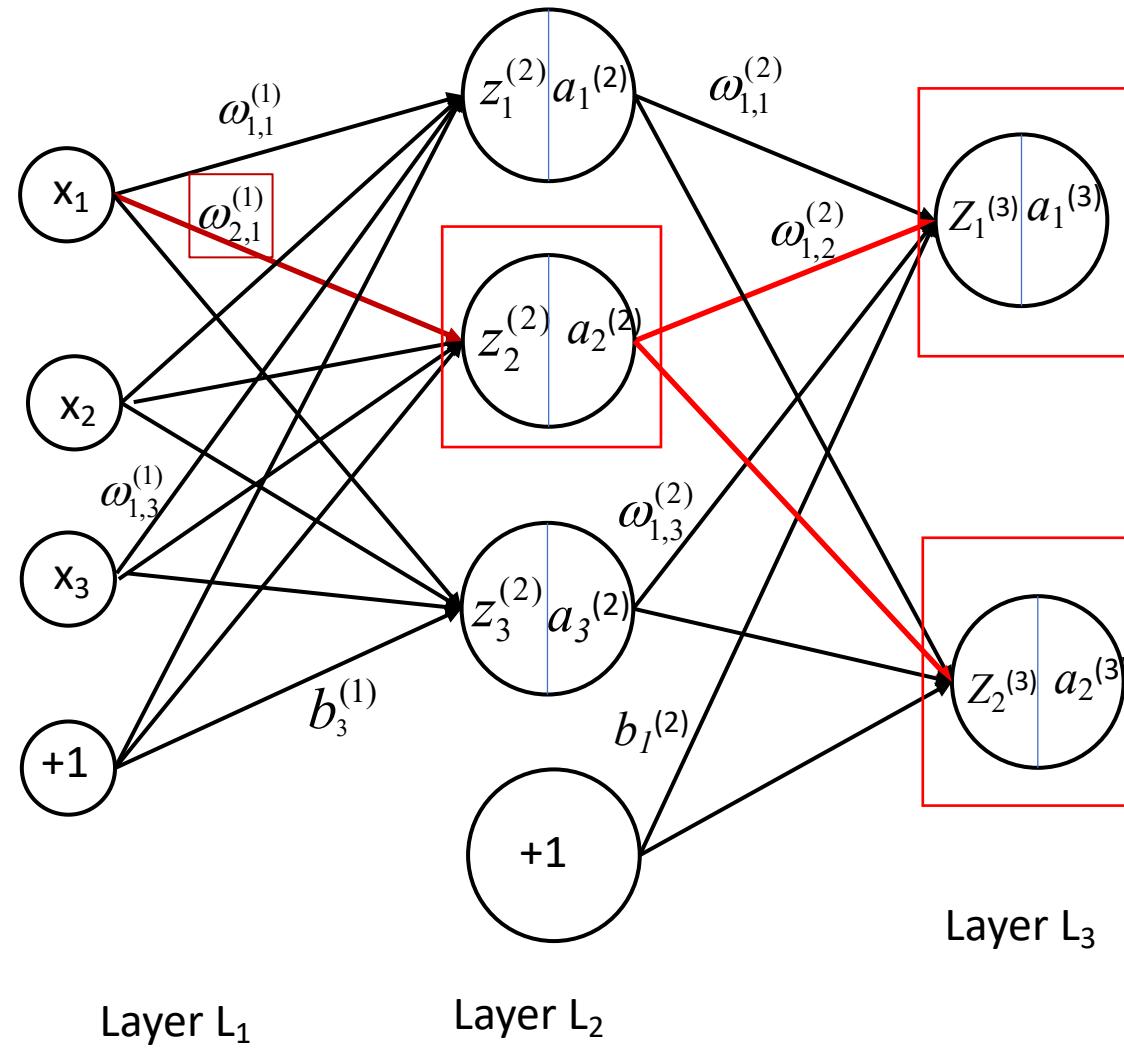
$$\delta_1^{(2)} = \delta_1^{(3)} \omega_{11}^{(2)} f'(z_1^{(2)}) + \delta_2^{(3)} \omega_{21}^{(2)} f'(z_1^{(2)})$$

$$\delta_2^{(2)} = \delta_1^{(3)} \omega_{12}^{(2)} f'(z_2^{(2)}) + \delta_2^{(3)} \omega_{22}^{(2)} f'(z_2^{(2)})$$

$$\delta_3^{(2)} = \delta_1^{(3)} \omega_{13}^{(2)} f'(z_3^{(2)}) + \delta_2^{(3)} \omega_{23}^{(2)} f'(z_3^{(2)})$$

Multi-Layer Perceptron and Back-propagation

- Training the inner layer at 1



$$\frac{\partial J}{\partial \omega_{ij}^{(1)}} = \delta_i^{(2)} a_j^{(1)}, \quad \frac{\partial J}{\partial b_i^{(1)}} = \delta_i^{(2)}$$

$$\frac{\partial J}{\partial \omega_{11}^{(1)}} = \delta_1^{(2)} a_1^{(1)}, \quad \frac{\partial J}{\partial \omega_{21}^{(1)}} = \delta_2^{(2)} a_1^{(1)}, \quad \frac{\partial J}{\partial \omega_{31}^{(1)}} = \delta_3^{(2)} a_1^{(1)}$$

$$\frac{\partial J}{\partial \omega_{12}^{(1)}} = \delta_1^{(2)} a_2^{(1)}, \quad \frac{\partial J}{\partial \omega_{22}^{(1)}} = \delta_2^{(2)} a_2^{(1)}, \quad \frac{\partial J}{\partial \omega_{32}^{(1)}} = \delta_3^{(2)} a_2^{(1)}$$

$$\frac{\partial J}{\partial \omega_{13}^{(1)}} = \delta_1^{(2)} a_3^{(1)}, \quad \frac{\partial J}{\partial \omega_{23}^{(1)}} = \delta_2^{(2)} a_3^{(1)}, \quad \frac{\partial J}{\partial \omega_{33}^{(1)}} = \delta_3^{(2)} a_3^{(1)}$$

$$\frac{\partial J}{\partial b_1^{(1)}} = \delta_1^{(2)}, \quad \frac{\partial J}{\partial b_2^{(1)}} = \delta_2^{(2)}, \quad \frac{\partial J}{\partial b_3^{(1)}} = \delta_3^{(2)}$$

Multi-Layer Perceptron and Back-propagation

- Update weights from backpropagation

$$\omega_{i,j}^{(l)} := \omega_{i,j}^{(l)} - \alpha \frac{\partial}{\partial \omega_{i,j}^{(l)}} J(\omega, b)$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(\omega, b)$$

$$\omega_{i,j}^{(l)} := \omega_{i,j}^{(l)} - \alpha \left(a_j^{(l)} \delta_i^{(l+1)} \right)$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \delta_i^{(l+1)}$$

where α = learning rate

$$\delta_i^{(l)} = \left(\sum_{j=1}^{S_{l+1}} \delta_j^{(l+1)} \omega_{ji}^{(l)} \right) f'(z_i^{(l)})$$

Where S_{l+1} = Number of nodes on layer $l + 1$

$$\text{Since } \delta_i^{(l+1)} = \frac{\partial J}{\partial z_i^{(l+1)}}$$

$$\frac{\partial J}{\partial \omega_{i,j}^{(l)}} = \frac{\partial J}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial \omega_{i,j}^{(l)}} = \delta_i^{(l+1)} \frac{\partial z_i^{(l+1)}}{\partial \omega_{i,j}^{(l)}}$$

Multi-Layer Perceptron and Back-propagation

- Partial derivatives of the loss function $J(\omega, b)$ for a single instance (x, y) :

$$\frac{\partial}{\partial \omega_{i,j}^{(l)}} J(\omega, b; x^{(i)}, y^{(i)}) \text{ and } \frac{\partial}{\partial b_i^{(l)}} J(\omega, b; x^{(i)}, y^{(i)})$$

- As such, we can compute batch gradients as

$$\frac{\partial}{\partial \omega_{i,j}^{(l)}} J(\omega, b) = \frac{1}{m} \sum_{k=1}^m \frac{\partial}{\partial \omega_{i,j}^{(l)}} J(\omega, b; x^{(k)}, y^{(k)})$$

$$\frac{\partial}{\partial b_i^{(l)}} J(\omega, b) = \frac{1}{m} \sum_{k=1}^m \frac{\partial}{\partial b_i^{(l)}} J(\omega, b; x^{(k)}, y^{(k)})$$

Developing an MLP with one hidden layer for a classification task

- As in the previous derivation, $N=3$, l_1 is the input layer and l_2 is the hidden layer.
- Input x has P dimensions
- Hidden layer l_2 has Q nodes.
- The output layer N has R nodes.
- Perform a feedforward pass, computing the activations for layers l_2 and N .

$$\begin{aligned}
 z_1^{(2)} &= \omega_{11}^{(1)}x_1 + \omega_{12}^{(1)}x_2 + \omega_{13}^{(1)}x_3 + \dots + \omega_{1P}^{(1)}x_P + b_1^{(1)} \\
 z_2^{(2)} &= \omega_{21}^{(1)}x_1 + \omega_{22}^{(1)}x_2 + \omega_{23}^{(1)}x_3 + \dots + \omega_{2P}^{(1)}x_P + b_2^{(1)} \\
 z_3^{(2)} &= \omega_{31}^{(1)}x_1 + \omega_{32}^{(1)}x_2 + \omega_{33}^{(1)}x_3 + \dots + \omega_{3P}^{(1)}x_P + b_3^{(1)} \\
 &\dots \\
 z_Q^{(2)} &= \omega_{M1}^{(1)}x_1 + \omega_{M2}^{(1)}x_2 + \omega_{M3}^{(1)}x_3 + \dots + \omega_{QP}^{(1)}x_P + b_Q^{(1)}
 \end{aligned}$$

$$\vec{z}^{(2)} = \tilde{\omega}^{(1)}\vec{x} + \vec{b}^{(1)}$$

$$\vec{z}^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ \vdots \\ z_Q^{(2)} \end{bmatrix}, \quad \tilde{\omega}^{(1)} = \begin{bmatrix} \omega_{11}^{(1)} & \omega_{12}^{(1)} & \cdot & \cdot & \omega_{1P}^{(1)} \\ \omega_{21}^{(1)} & \omega_{22}^{(1)} & \cdot & \cdot & \omega_{2P}^{(1)} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \omega_{Q1}^{(1)} & \cdot & \cdot & \cdot & \omega_{QP}^{(1)} \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_P \end{bmatrix}, \quad \vec{a}^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ \vdots \\ a_Q^{(2)} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{(2)}) \\ \sigma(z_2^{(2)}) \\ \vdots \\ \sigma(z_Q^{(2)}) \end{bmatrix}$$

Developing an MLP with one hidden layer for a classification task

- Continue the forward pass calculation to the output layer
- Keep all the activation values for later backpropagation calculations

$$\begin{aligned}z_1^{(3)} &= \omega_{11}^{(2)} a_1^{(2)} + \omega_{12}^{(2)} a_2^{(2)} + \dots + \omega_{1Q}^{(2)} a_Q^{(2)} + b_1^{(2)} \\z_2^{(3)} &= \omega_{21}^{(2)} a_1^{(2)} + \omega_{22}^{(2)} a_2^{(2)} + \dots + \omega_{2Q}^{(2)} a_Q^{(2)} + b_2^{(2)} \\&\vdots \\&\vdots \\z_R^{(3)} &= \omega_{R1}^{(2)} a_1^{(2)} + \omega_{R2}^{(2)} a_2^{(2)} + \dots + \omega_{RQ}^{(2)} a_Q^{(2)} + b_R^{(2)}\end{aligned}$$

$$\vec{z}^{(3)} = \tilde{\omega}^{(2)} \vec{a}^{(2)} + \vec{b}^{(2)}$$

$$\vec{z}^{(3)} = \begin{bmatrix} z_1^{(3)} \\ z_2^{(3)} \\ \vdots \\ z_R^{(3)} \end{bmatrix} \quad \tilde{\omega}^{(2)} = \begin{bmatrix} \omega_{11}^{(2)} & \omega_{12}^{(2)} & \cdot & \cdot & \omega_{1Q}^{(2)} \\ \omega_{21}^{(2)} & \omega_{22}^{(2)} & \cdot & \cdot & \omega_{2Q}^{(2)} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \omega_{R1}^{(2)} & \omega_{R2}^{(2)} & \cdot & \cdot & \omega_{RQ}^{(2)} \end{bmatrix} \quad \vec{a}^{(3)} = \begin{bmatrix} a_1^{(3)} \\ a_2^{(3)} \\ \vdots \\ a_R^{(3)} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{(3)}) \\ \sigma(z_2^{(3)}) \\ \vdots \\ \cdot \\ \sigma(z_R^{(3)}) \end{bmatrix}$$

Developing an MLP with one hidden layer for a classification task

- For each output unit i in layer L_3 (the output layer), set

$$\delta_i^{(N)} = (a_i^{(N)} - y_i) a_i^{(N)} (1 - a_i^{(N)})$$

Since $N=3$

$$\delta_i^{(3)} = (a_i^{(3)} - y_i) a_i^{(3)} (1 - a_i^{(3)})$$

- For the hidden layer l_2 , set

Note S_3 is the number of nodes at layer $N (=3)$

$$\delta_1^{(2)} = \sum_{j=1}^{S_3} \delta_j^{(3)} \omega_{j1}^{(2)} f'(z_1^{(2)}) = \sum_{j=1}^{S_3} \delta_j^{(3)} \omega_{j1}^{(2)} a_1^{(2)} (1 - a_1^{(2)})$$

$$\delta_2^{(2)} = \sum_{j=1}^{S_3} \delta_j^{(3)} \omega_{j2}^{(2)} a_2^{(2)} (1 - a_2^{(2)})$$

As noted earlier, you need the activation values for backpropagation

Developing an MLP with one hidden layer for a classification task

- Update the weights
- For layer l_2

$$\frac{\partial J}{\partial \omega_{i,j}^{(2)}} = \delta_i^{(3)} a_j^{(2)} = (a_i^{(3)} - y_i) a_i^{(3)} (1 - a_i^{(3)}) a_j^{(2)}$$

$$\frac{\partial J}{\partial b_i^{(2)}} = \delta_i^{(3)} = (a_i^{(3)} - y_i) a_i^{(3)} (1 - a_i^{(3)})$$

- For layer l_1

$$\delta_i^{(2)} = \sum_{j=1}^{S_3} \delta_j^{(3)} \omega_{ji}^{(2)} a_i^{(2)} (1 - a_i^{(2)})$$

$$\frac{\partial J}{\partial \omega_{ij}^{(1)}} = \delta_i^{(2)} a_j^{(1)} , \quad \frac{\partial J}{\partial b_i^{(1)}} = \delta_i^{(2)}$$

Developing an MLP with one hiddern layer for a classification task

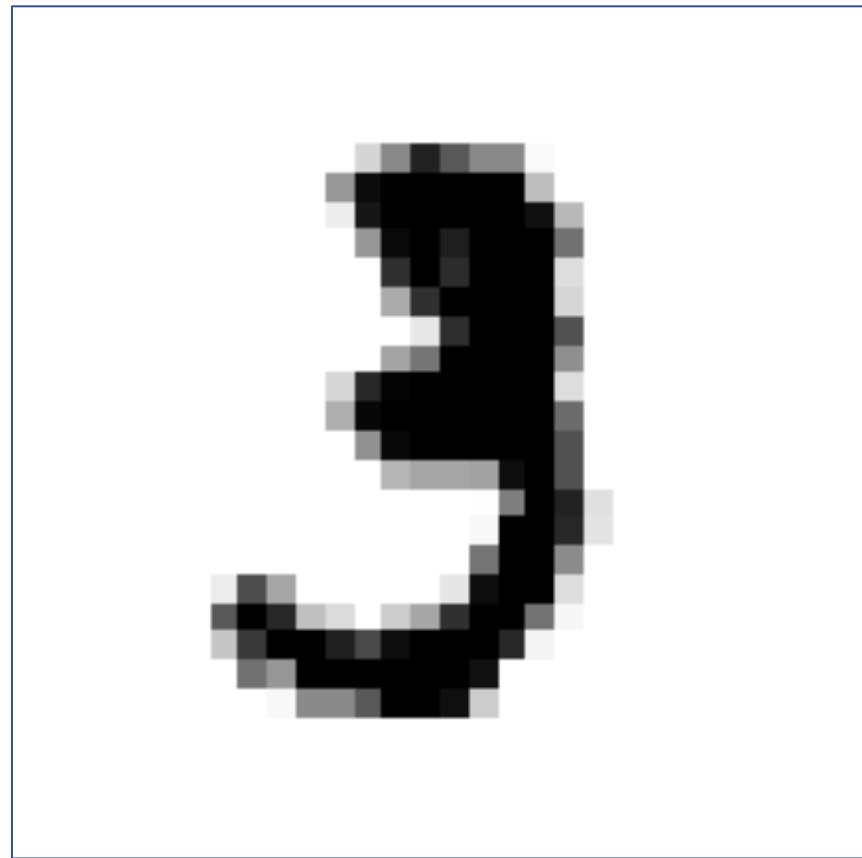
- Project 1
- MNIST digit handwritten dataset

```
from sklearn.datasets import fetch_openml  
mnist = fetch_openml('mnist_784', version=1)  
mnist.keys()  
#%%  
import numpy as np  
X, y = mnist["data"], mnist["target"]  
y = y.astype(np.uint8)  
  
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```



Developing an MLP with one hidden layer for a classification task

- 70,000 images of 28×28 pixels
- Each image labeled as 0 - 9
- Develop an ANN classifier for MNIST dataset with one hidden layer
- Input dimension = $28 \times 28 = 784$
- Output dimension = 10
- Thus, P=784, Q = ?, R = 10
- Thus the output has to be sent to a softmax activation to classify 10 different digit classes.
- Recommend using stochastic gradient descent instead of batch gradient



Project 1 Specifics

- Use the MNIST data samples for training and testing. The python code posted on the blackboard will let you download MNIST. Develop the classifier code to run on Google Colab (<https://colab.research.google.com/notebooks/intro.ipynb#recent=true>) so that you don't have to worry about setting up your machine correctly.
- Develop a code Python using the code provided as the starting point to design a neural network to perform 10 digit classification.
- Experiment with 3 different hidden units and see the performance differences.
- Summarize the results and report them. Include the code you ran in the report.

Project Specifics

- Submit (upload your files to the Blackboard) your code by uploading the original python file (*.py) after changing the extension to “*.py.txt”
- Submit the project report via the Blackboard
- Use L_2 norm as the loss function. Also use 10 dimensional vector as the output for the digit classification.
- Key Elements to be included in the report (I will be grading the report based on these elements).
 - Describe your network design and hyperparameters
 - Number of layers
 - Number of nodes in each layer
 - Activation functions used
 - Learning rate
 - Describe the forward pass of your code
 - Describe the backpropagation of your code
 - Include learning curves: both training and validation losses
 - Describe the validation process used
 - Describe the criteria you used to terminate the training (when and why).
 - Predict the performance of the code on an unseen dataset.
 - Source code to be also included as a part of the report.

Project Specifics

- To get the full credit of the project, I should be able to run your code on Google Colab. So, please verify that it runs.
- Fully describe what you have done in the code. Include comments on the code so that I can understand what you did.
- Do NOT use any of Keras, TensorFlow library classifiers. I would like you to build a neural network classifier from the grounds up. You may use other functions within python or ScikitLearn, but your core code should be done by python including forward pass and backpropagation.