Amity School of Engineering and Technology

B.Tech. Computer Science and Engineering

Semester VI

Generative Artificial Intelligence (Generative AI)

[AIML 303]

Faculty: Dr Rinki Gupta

Generative Artificial Intelligence (Generative AI) [AIML303]

3L+1P=4Credits

- **Module I: Introduction to Generative AI and Fundamental Study** 20%

Basic introduction to Generative AI, Applications of Generative AI in various fields, Perceptron and Multilayer Perceptron (MLP), Gradient Descent, Backpropagation algorithm, Loss functions, Understanding Bias and Variance

- **Module II: Model Parameters Optimization and Convolution Neural Networks** 20%

Methods to deal with overfitting issues, Optimizers (Momentum, Nesterov Accelerated Gradient, Adagrad, RMSprop, Adam), Convolution neural networks, Understanding the architectural characteristics of deep CNNs, Transfer Learning and Fine-tuning

- **Module III: Generative Adversarial Networks** 20%

Brief overview of Generative Models, Introduction to Autoencoders and Variational Autoencoders, Generative Adversarial Networks (GANs) for Realistic Data Synthesis, Conditional Generative Models for Controlled Data Generation, Dealing with common issues like mode collapse and instability, Use cases of GANs in image synthesis and data augmentation

Generative Artificial Intelligence (Generative AI) [AIML303]

3L+1P=4Credits

- **Module IV: Use of Recurrent Neural Networks in Generative AI** 20%

Recurrent Neural Networks (RNNs), Sequential models for solving Long-term dependency issues., Applications of RNNs in Generative AI: Text generation with RNNs, Music generation using RNNs, Speech synthesis and recognition.

- **Module V: Attention Mechanism and Transformers** 20%

Introduction to attention mechanism, Use of attention in convolution neural network, Self-Attention, Spatial and Channel Attention, Transformer architecture, Vision Transformer, Transfer Learning with vision transformer, Applications of Vision Transformers (Image classification, Object detection)

Assessment/ Examination Scheme:

Theory	Lab	Total
75%	25%	100%

L	P	Credits
3	1	4

Theory Assessment

Internal Assessment				End Term Exam
Mid-term	Quiz	Home Assignment	Attendance	
15%	10%	10%	5%	60%

Practical Assessment

Internal Assessment				External Practical Exam
Lab Record	Viva	Performance	Attendance	
15%	10%	10%	5%	60%

Books

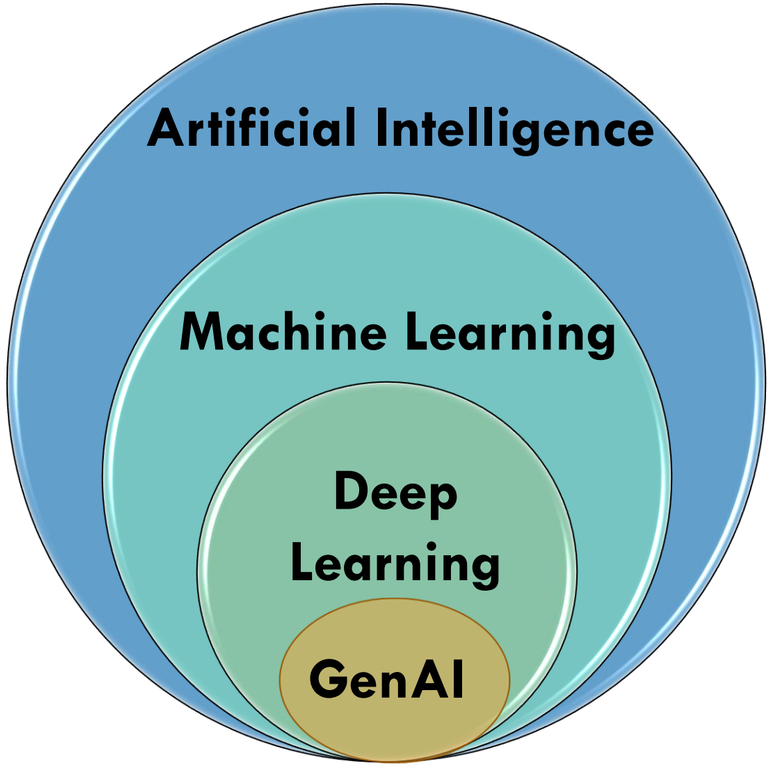
Reference Books:

- "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville, published by MIT Press in 2016. http://www.deeplearningbook.org
- "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron, published by O'Reilly Media in 2019.
- Deep Learning with Python" by François Chollet, published by Manning Publications in 2018.

Other sources:

- Deep Neural Networks by Prof. P. K. Biswas, IIT Kharagpur 2019, https://www.youtube.com/playlist?list=PLbRMhDVUMngc7NM-gDwcBzIYZNFSK2N1a

AI, ML, DL, GenAI



AI is a field

AI is the theory and development of Computer Systems to enable performing tasks normally requiring human intelligence

What is Machine Learning?

One of the Earliest definition of Machine Learning:

Arthur Samuel (1959):

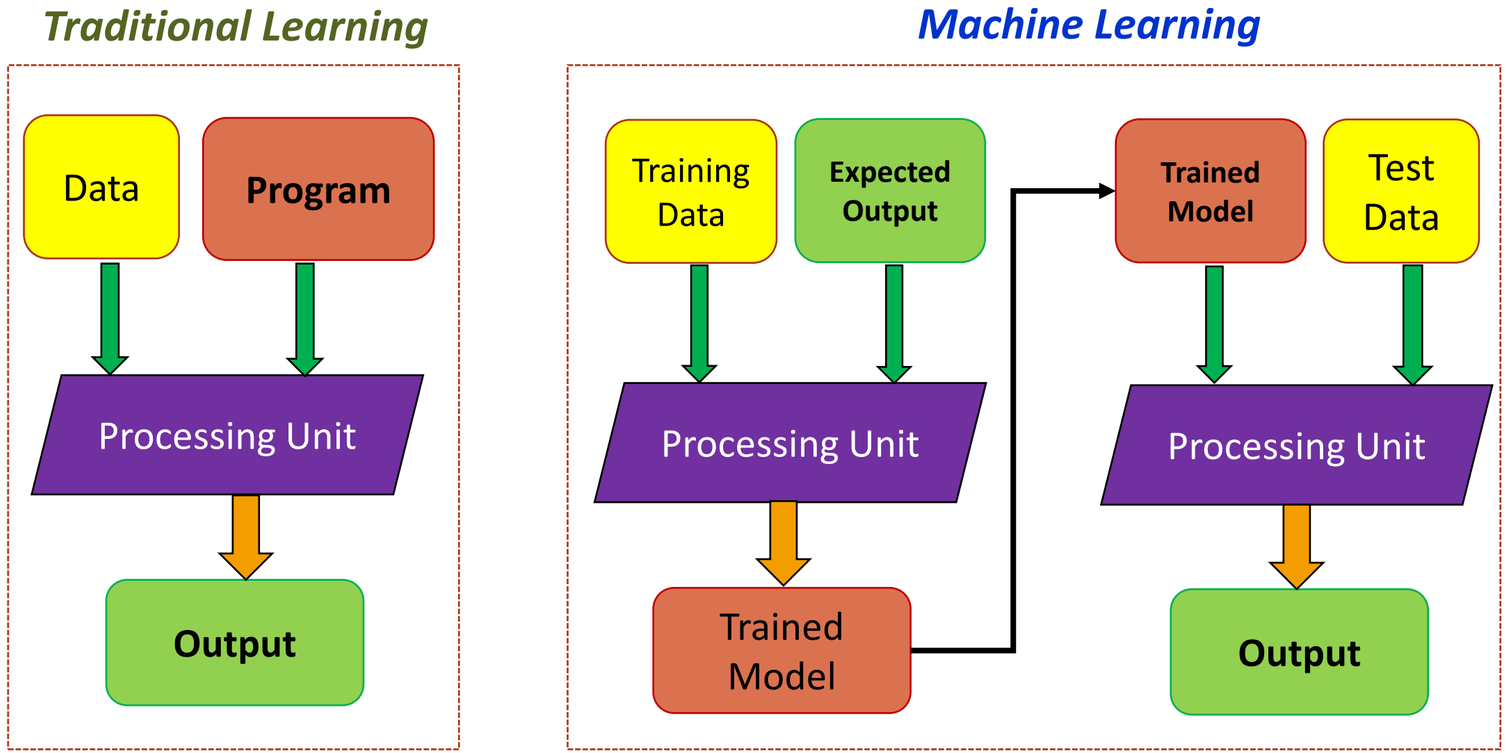
"Field of study that gives computers the ability to learn without being explicitly programmed."

The Samuel Checkers-playing Program was among the world's first successful self-learning programs, and as such a very early demonstration of the fundamental concept of artificial intelligence.

***On February 24, 1956**, Arthur Samuel's Checkers program was developed for play on the IBM 701, In 1962, Self-proclaimed checkers master Robert Nealey played the game on an IBM 7094 computer. The computer won. It is still considered a milestone for artificial intelligence, as an example of the capabilities of an electronic computer.*

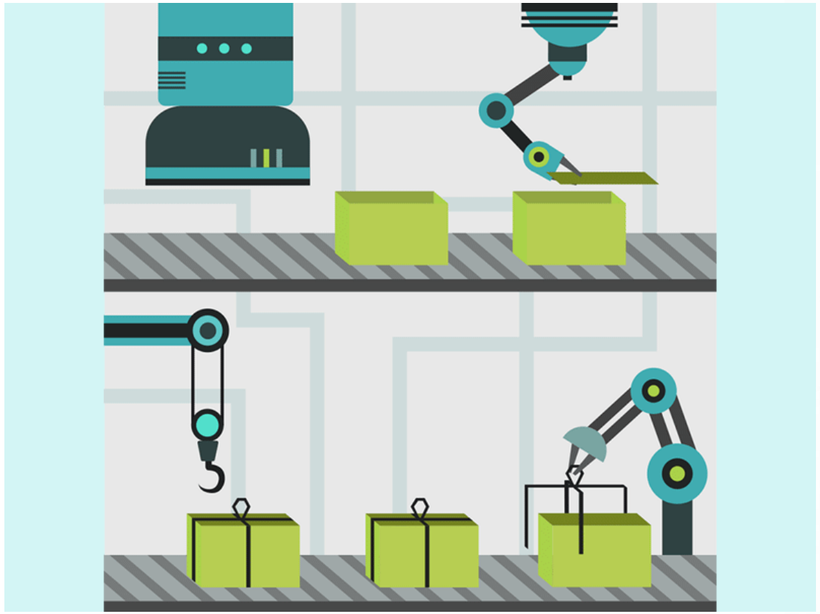
Courtesy: ibm.com

Traditional Learning vs. Machine Learning



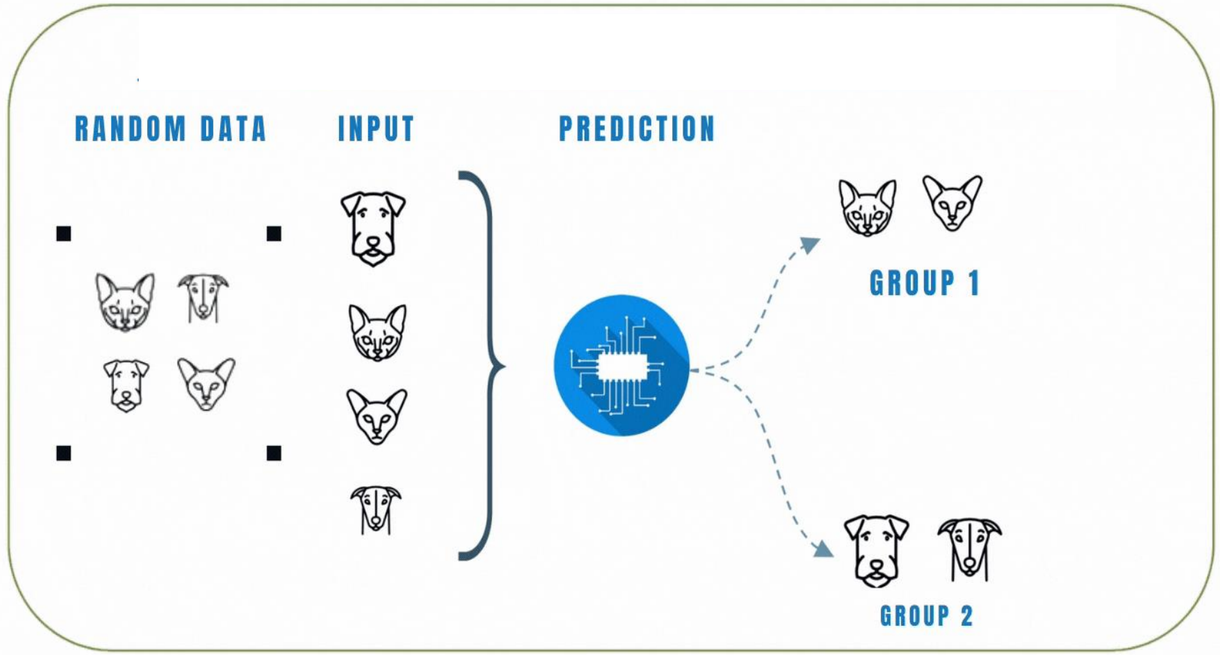
Machine Learning vs Process Automation

Process Automation



Follow orders, Pre-programmed Rules to run a process, Monotonous, Repetitive process

Machine Learning



Mimic human-ability to think and do, Machine seeks patterns, adapts with experience

Example of Machine Learning: Email Classification

Experience/Data

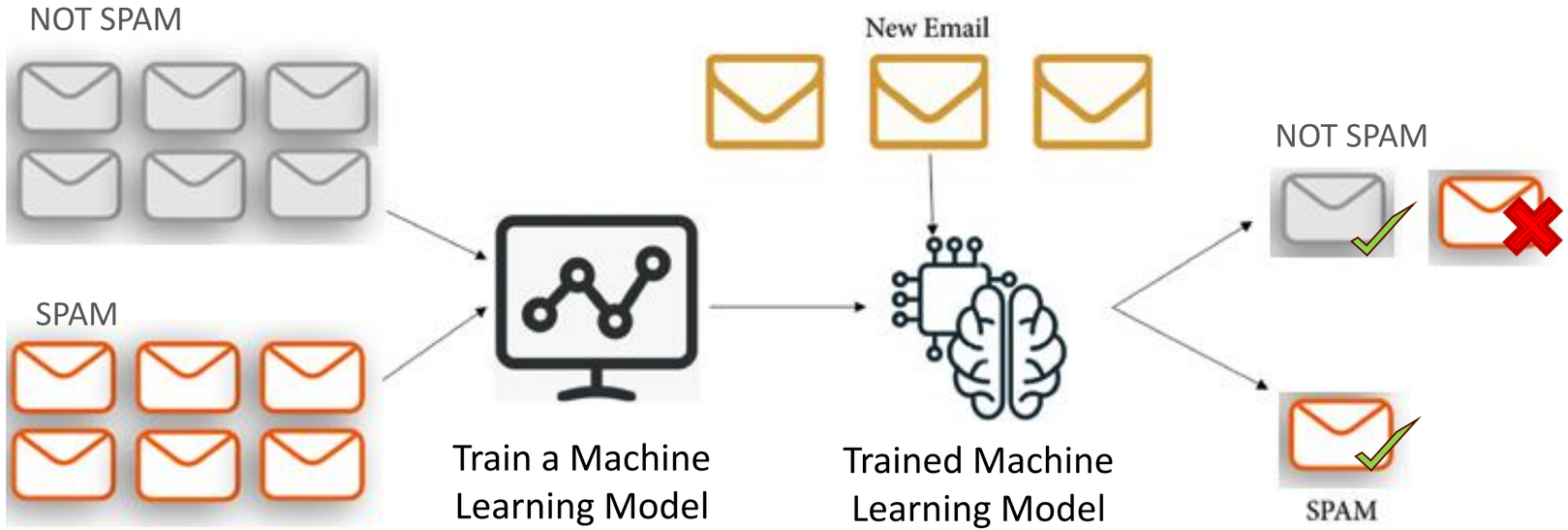
Lots of
labelled e-mails

Task

Classify as
Spam/Not Spam

Performance

$$\frac{\text{Correct Prediction}}{\text{Total Number of Prediction}} = \frac{2}{3}$$



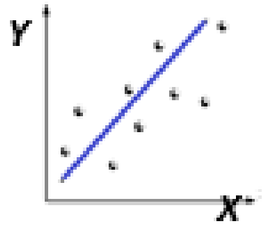
Example of Machine Learning: Score Prediction

Students in Class

Experience/Data

Labelled Data: Lectures attended vs. Score

	Lectures Attended	Score In Chemistry
Student 1	3	35
Student 2	7	45
Student 3	16	75
Student 4	14	60



X Y

Train ML Model

Task

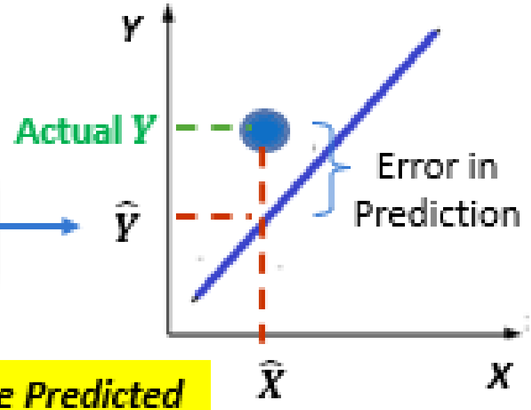
Predict Score in Chemistry

New Value of X

Trained ML Model

Performance

Error in Prediction



$\hat{Y}$ = Score Predicted by Model

Formal Definition of Machine Learning

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

Tom M. Mitchel (1998)

T

Task

- What is to be done? Prediction/ Clustering

E

Experience

- **Data– Images, time-series, predictors**

P

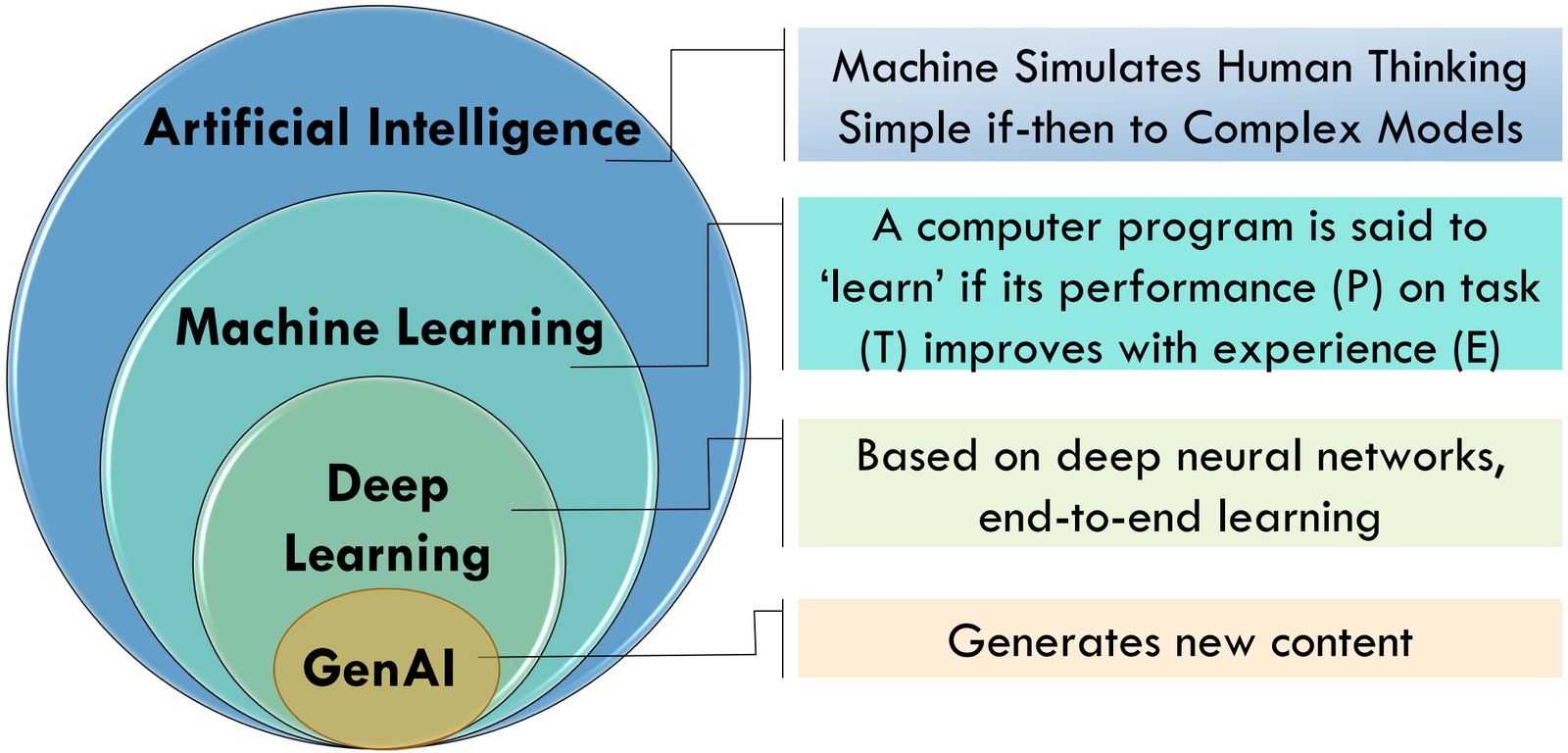
Performance

- **Quantitative measure of performance.**

AI, ML, DL, GenAI

Generative Models

generates new data instances based on a learned probability distribution of existing data



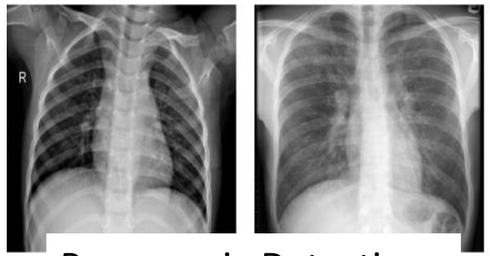
Useful Terminology

Useful Terminology

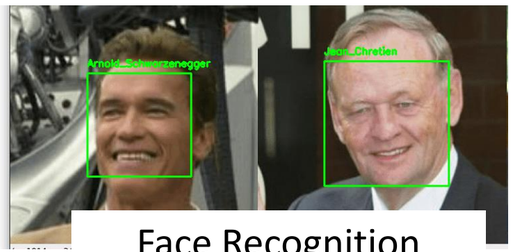
- **Data**: collection of examples/observations/instances/samples

Examples of Data

Images



Pneumonia Detection



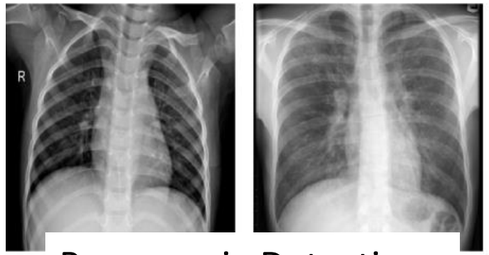
Face Recognition

Video

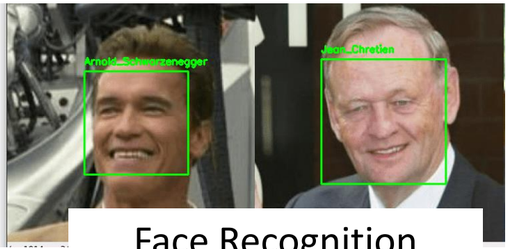
Activity Detection

Examples of Data

Images



Pneumonia Detection

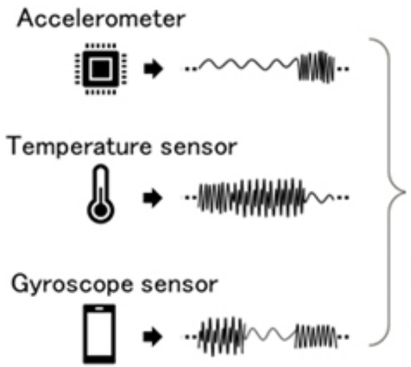


Face Recognition

Video

Activity Detection

Sensor data

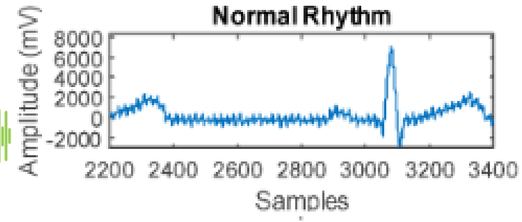


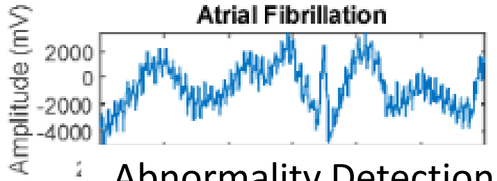
Activity Detection

Speech

Speech Recognition

Biomedical (ECG)

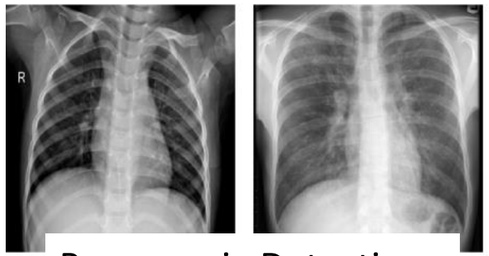




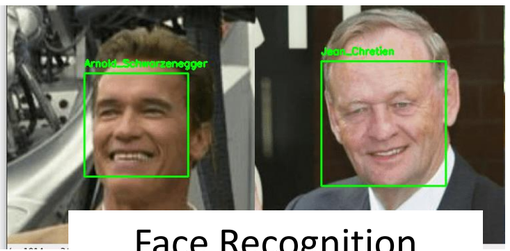
Abnormality Detection

Examples of Data

Images



Pneumonia Detection

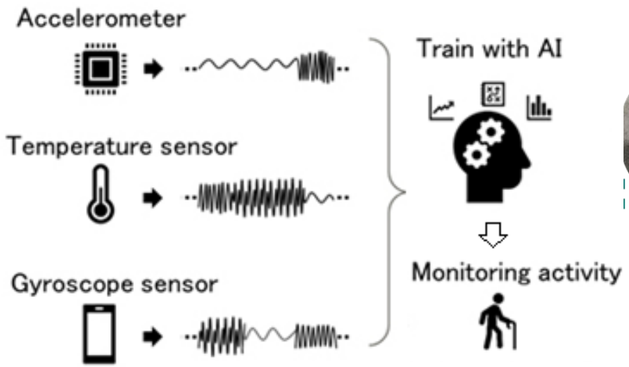


Face Recognition

Video

Activity Detection

Sensor data

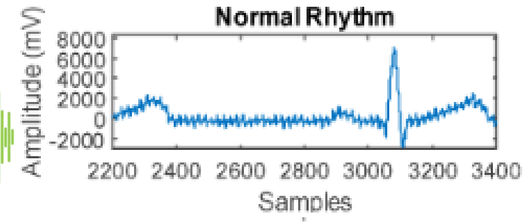


Activity Detection

Speech

Speech Recognition

Biomedical (ECG)



Abnormality Detection

Demographic

Population Analysis

Other Tabular Data

Score Prediction	Lectures Attended	Previous CGPA	Final Exam Score
Student 1	3	15	35
Student 2	7	25	45
Student 3	16	55	75

Useful Terminology

- **Data**: collection of examples/observations/instances/samples

Unstructured data

- Easier to collect
- Loosely organized
- Qualitative
- Difficult to search

The university has 5600 students. John's ID is number 1, he is 18 years old and already holds a B.Sc. degree. David's ID is number 2, he is 31 years old and holds a Ph.D. degree. Robert's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.

Structured data

ID	Name	Age	Degree
1	John	18	B.Sc.
2	David	31	Ph.D.
3	Robert	51	Ph.D.
4	Rick	26	M.Sc.
5	Michael	19	B.Sc.

- Difficult to collect
- Organized
- Quantitative
- Easy to search

Useful Terminology

- **Data**: collection of examples/observations/instances/samples

- **Features**: ?

Useful Terminology

- **Data**: collection of examples/observations/instances/samples
- **Features**: Descriptor of the object. Feature is derived from the data and it represents the data. Features are properties/attributes/parameters that describe the data
- **Feature Vector ?**

Useful Terminology

- **Data**: collection of examples/observations/instances/samples
- **Features**: Descriptor of the object. Feature is derived from the data and it represents the data. Features are properties/attributes/parameters that describe the data
- **Feature Vector**, **x**= $[x_1,x_2,\ldots,x_M]$

1xM vector, M=dimensionality of data

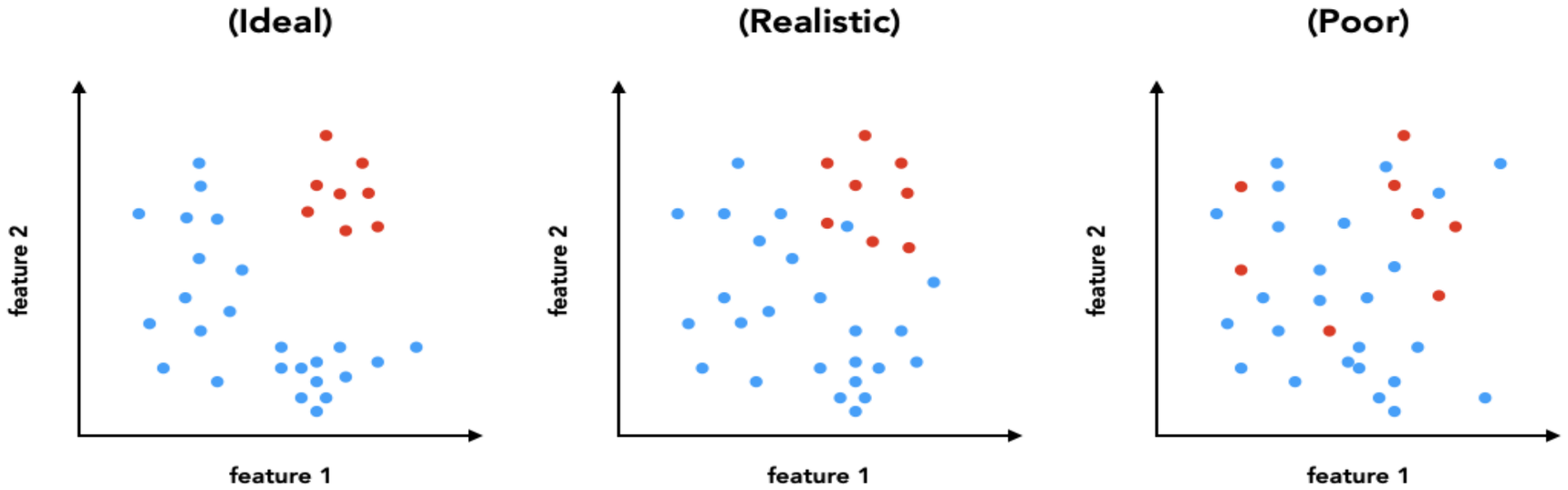
Features				Labels
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	Iris setosa

- Feature Matrix $X = \begin{bmatrix} x^1 \\ x^2 \\ \\ x^N \end{bmatrix}$,Target Class $Y = \begin{bmatrix} y^1 \\ y^2 \\ \\ y^N \end{bmatrix}$ (Label)

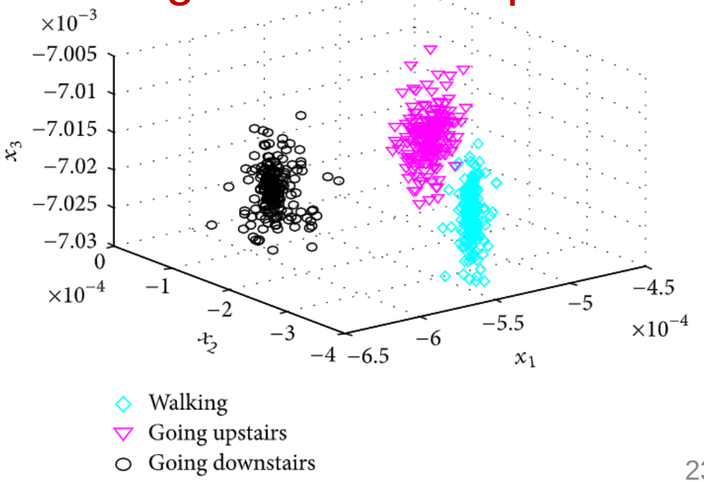
Features				Labels
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	Iris setosa
4.9	3.0	1.4	0.2	Iris setosa
7.0	3.2	4.7	1.4	Iris versicolor
6.4	3.2	4.5	1.5	Iris versicolor
6.3	3.3	6.0	2.5	Iris virginica
5.8	3.3	6.0	2.5	Iris virginica

- Observation is a point in M-dimensional feature space

Eg: 2D feature space

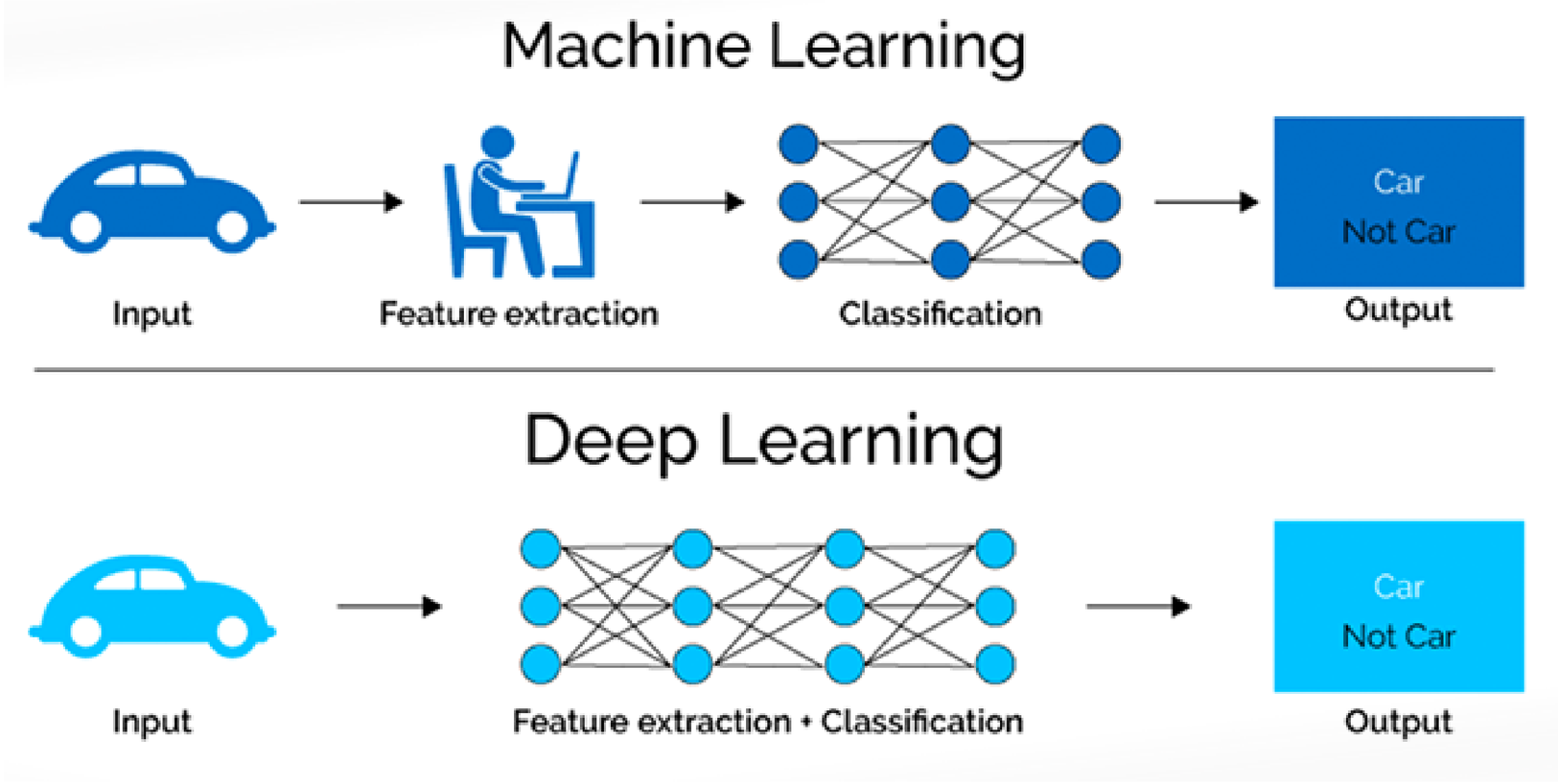


Eg: 3D feature space

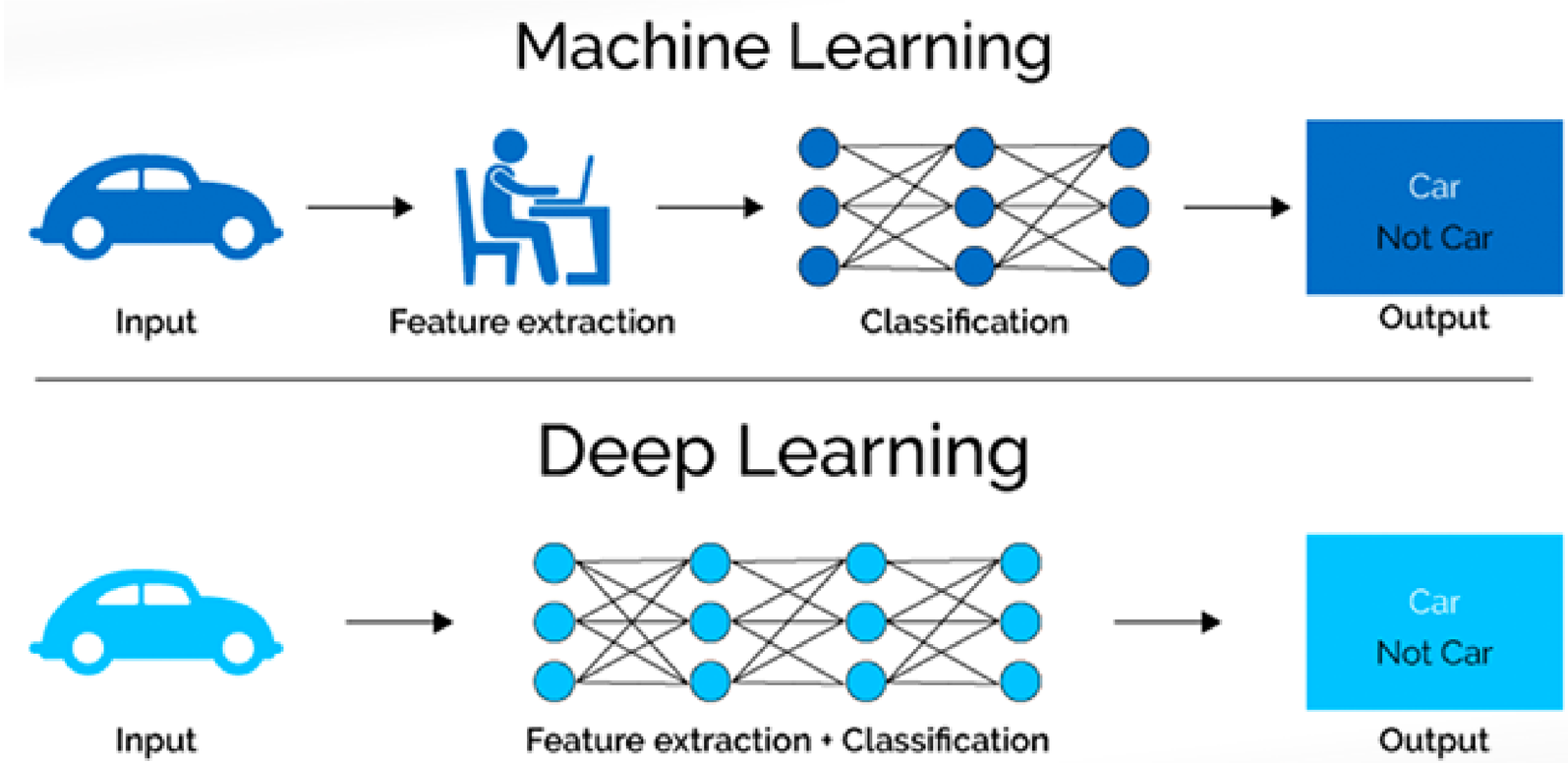


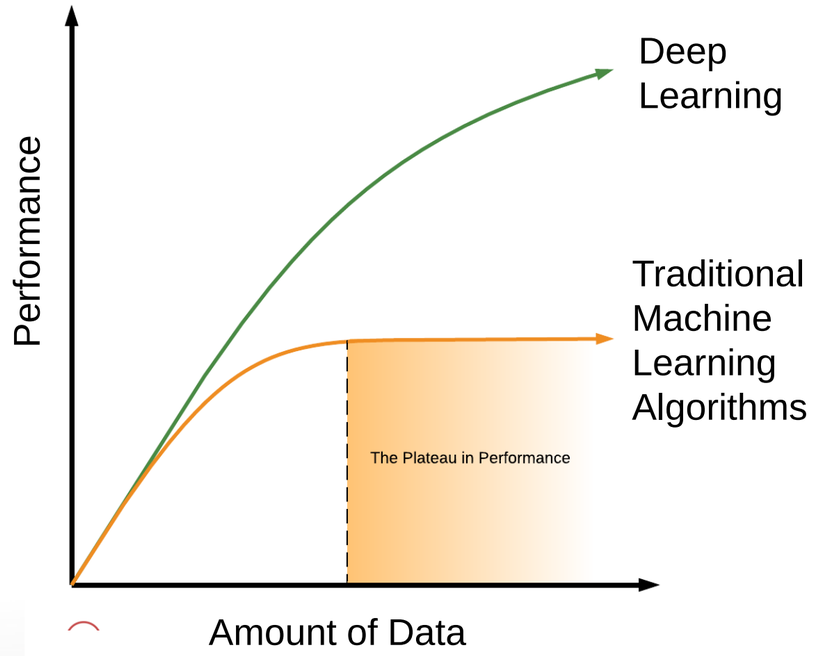
Machine Learning vs. Deep Learning

Machine Learning vs. Deep Learning



Machine Learning vs. Deep Learning





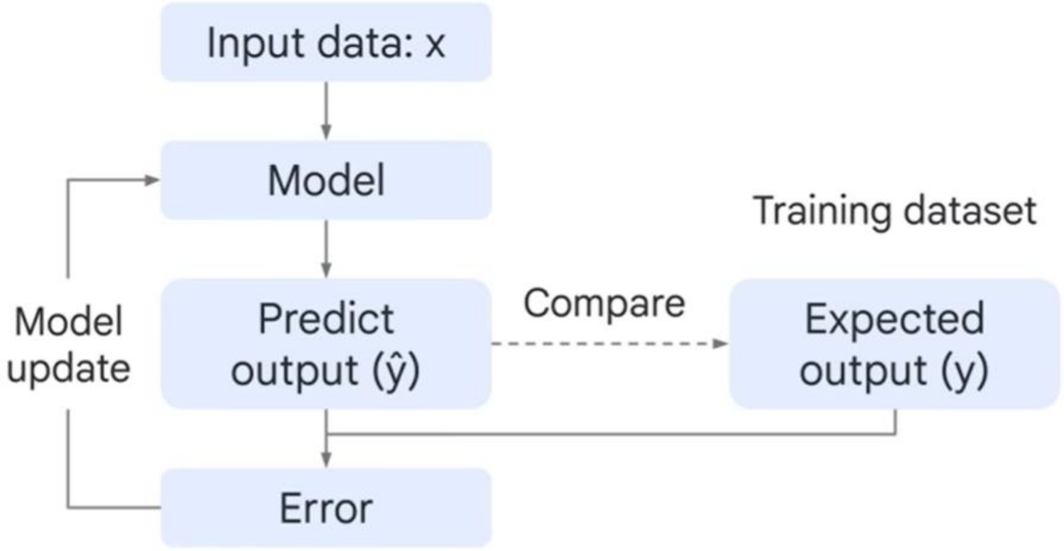
Types of Machine Learning

Machine Learning

- **Supervised Learning**
  - Task Driven
  - Predict Future Values
  - → Classification (Categorical Labels)
    - NOT SPAM / SPAM
  - → Regression (Continuous Labels)
    - Predict Score in Chemistry
- **Unsupervised Learning**
  - Data Driven
  - Clustering
  - → (No defined Labels)
    - Cluster based on Number of Legs
- **Reinforcement Learning**
  - Learning from the mistakes
  - agent, environment, actions, rewards, observations
  - Environment, Reward, Action, Agent

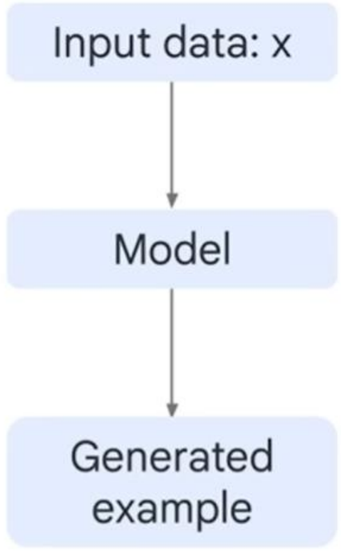
Supervised Learning

- Training Data is Labelled
- Task driven
- Classification/Regression



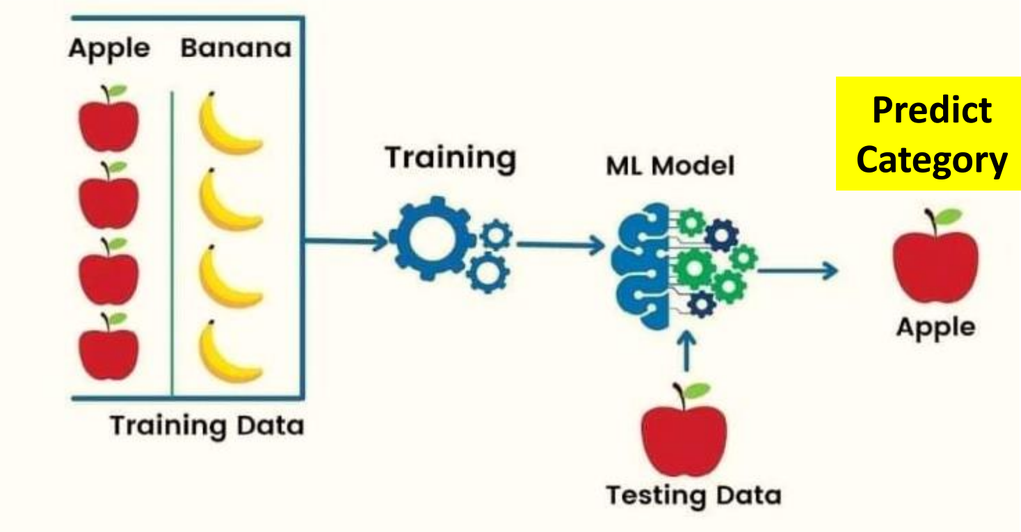
Unsupervised Learning

- Training Data is Not Labelled
- Data driven
- Clustering, Dimensionality Reduction



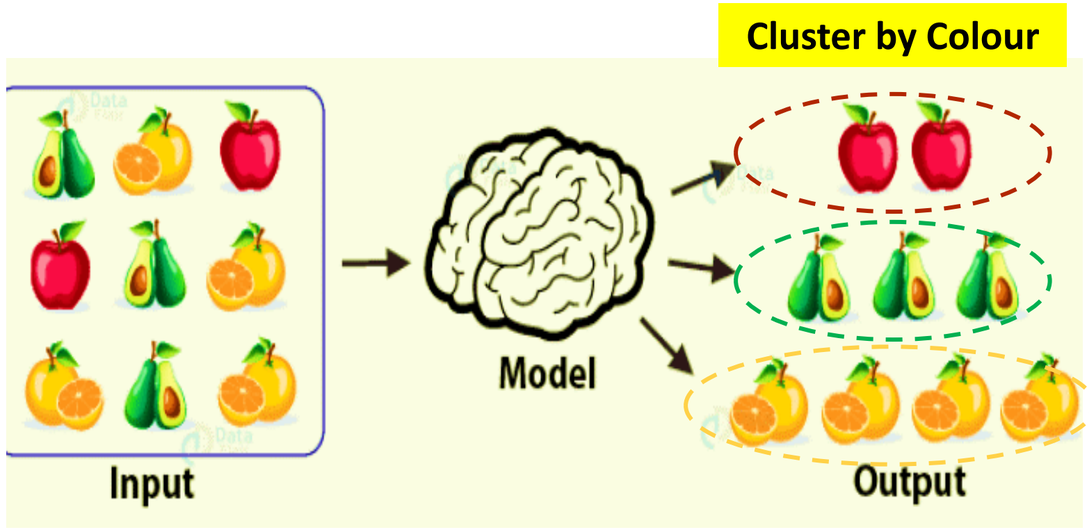
Supervised Learning

- Training Data is Labelled
- Task driven
- Classification/Regression



Unsupervised Learning

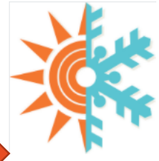
- Training Data is Not Labelled
- Data driven
- Clustering, Dimensionality Reduction



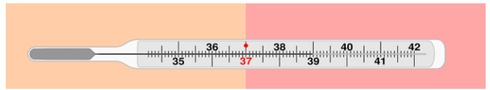
Example of Classification vs. Regression (Supervised Learning)

Parameters	Classification	Regression
Output type	Discrete	Continuous
Trying to find	A boundary	Best Fit Line
Evaluation	Accuracy	Sum of squared errors

Examples

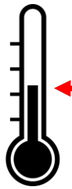


Will it be cold or Hot tomorrow?



Cold Hot

What is the temperature going to be tomorrow?



$42^{o}C$
(Prediction)

Example of Classification vs. Regression (Supervised Learning)

Students in Class

Classification → **Pass or Fail?**

Student Prediction

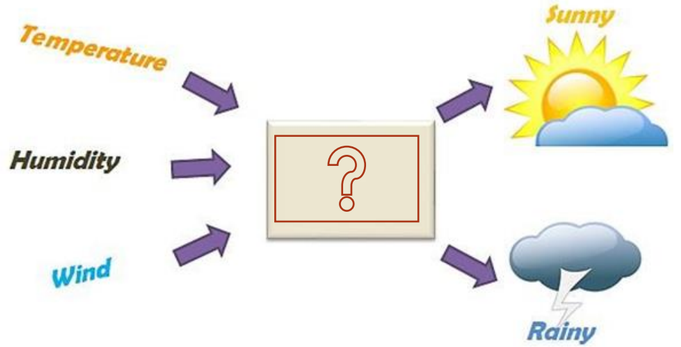
Regression → **Percentage?**

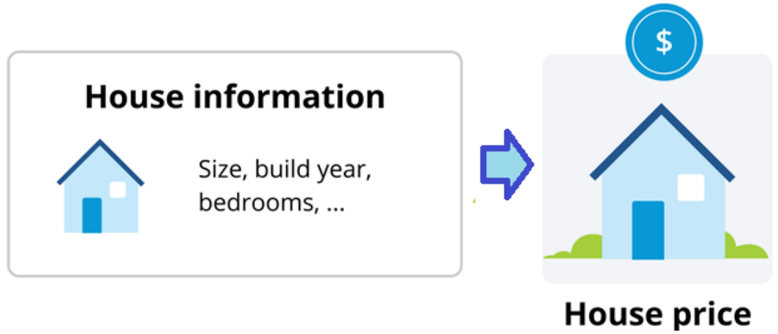
Student's marks Prediction

Q : Which ML technique is applicable?

1. Determine whether it will be rainy or sunny depending on temperature, humidity and wind observed on previous rainy and sunny days
2. Predict house price based on information related to the house, such as Size, build year, number of bedrooms.
3. Segment an image according to intensity

   (gray scale value)

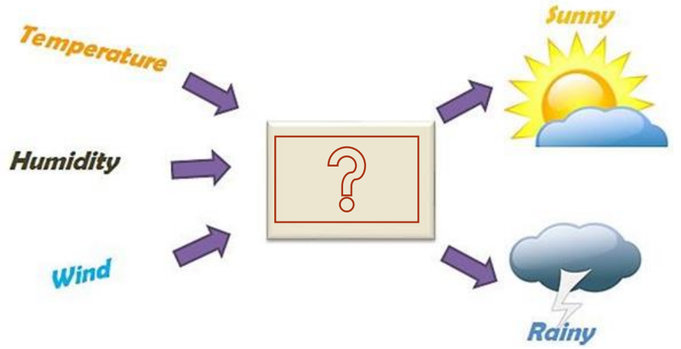




Q : Which ML technique is applicable?

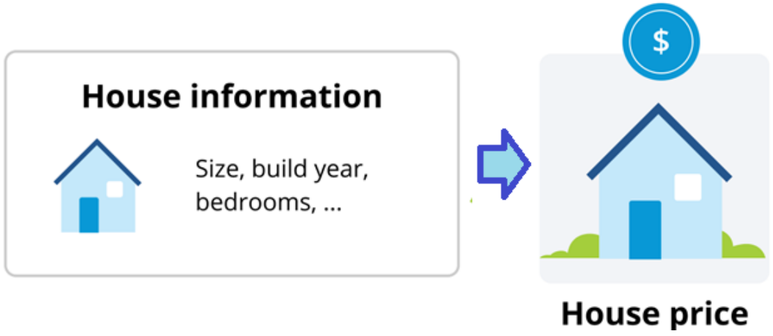
1. Determine whether it will be rainy or sunny depending on temperature, humidity and wind observed on previous rainy and sunny days

Classification



2. Predict house price based on information related to the house, such as Size, build year, number of bedrooms.

Regression



3. Segment an image according to intensity

(gray scale value)

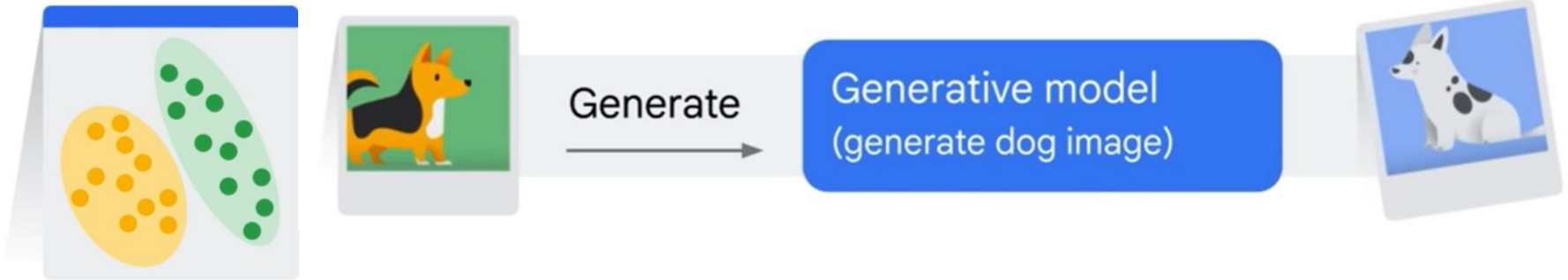
Clustering

Types of Deep Learning Models

Discriminative:

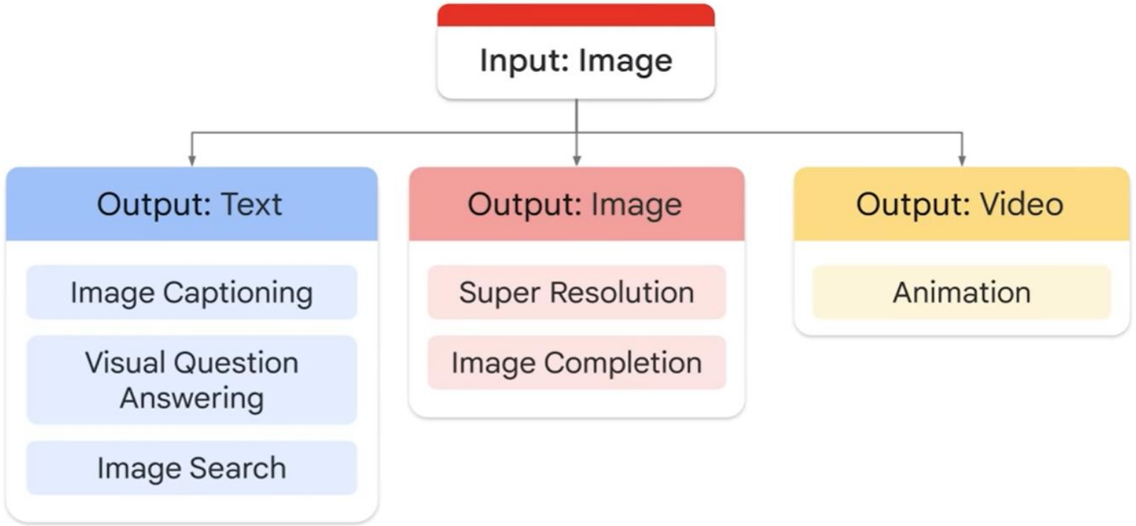
- Used to classify or predict
- Typically trained on labelled data
- Learn relationship between features of data points and labels

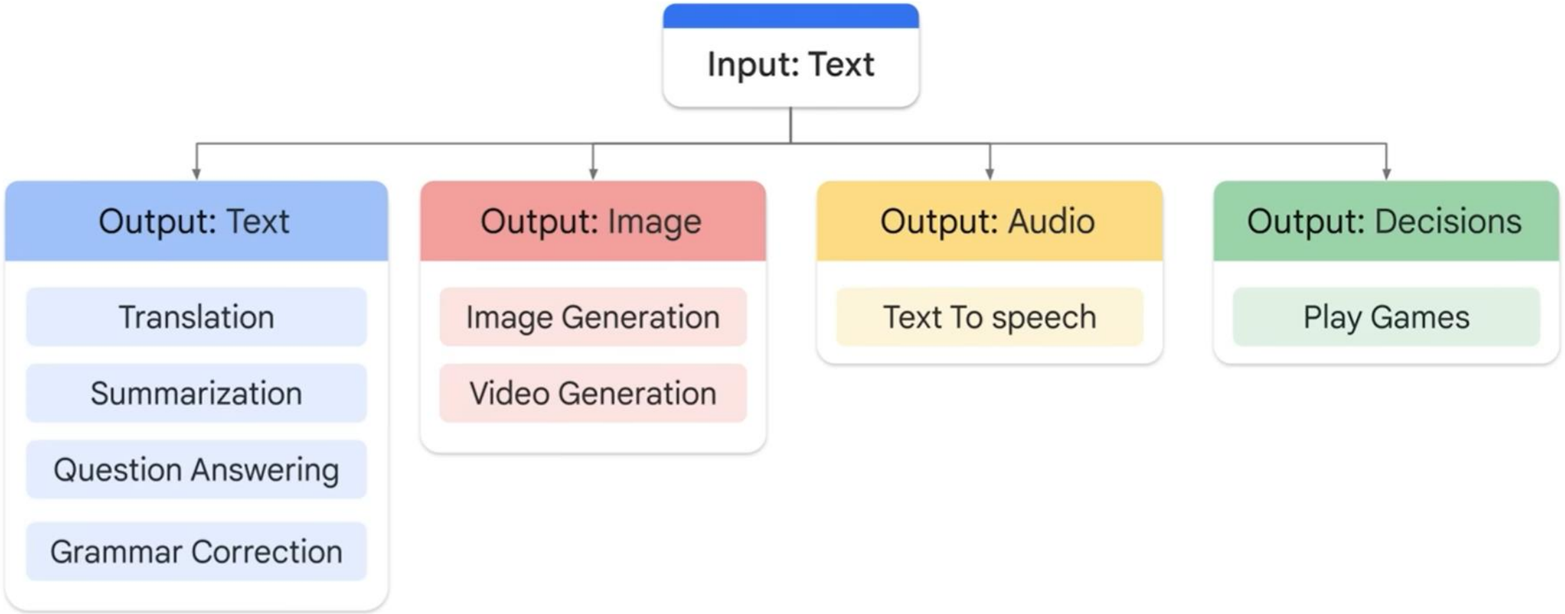
Generative:



- Generate new data that is similar to data it was trained on
- Understand distribution of data and how likely a given example is

Applications of Generative AI





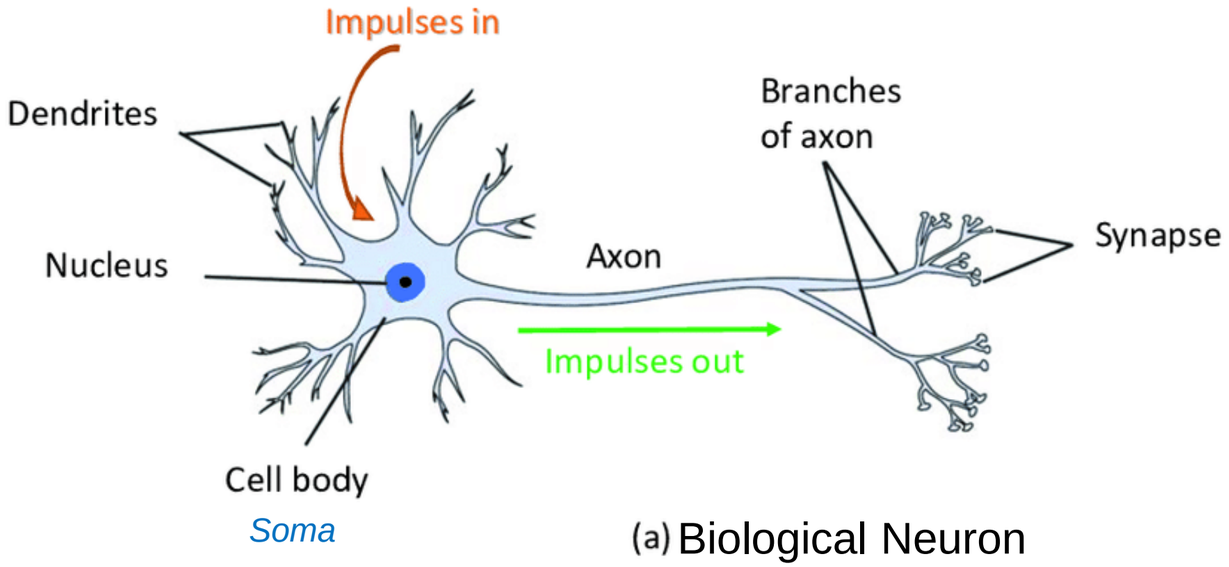
Module I: Introduction to Generative AI and Fundamental Study

Module I (20%)

Basic introduction to Generative AI, Applications of Generative AI in various fields, Perceptron and Multilayer Perceptron (MLP), Gradient Descent, Backpropagation algorithm, Loss functions, Understanding Bias and Variance

Neuron

- *Neurons*: cells forming the human nervous system
- Neuron collects signals from *dendrites*
- Sends out spikes of electrical activity through an *axon*, which splits into thousands of branches
- At end of each branch, a *synapses* converts activity into either exciting or inhibiting activity of a dendrite at another neuron



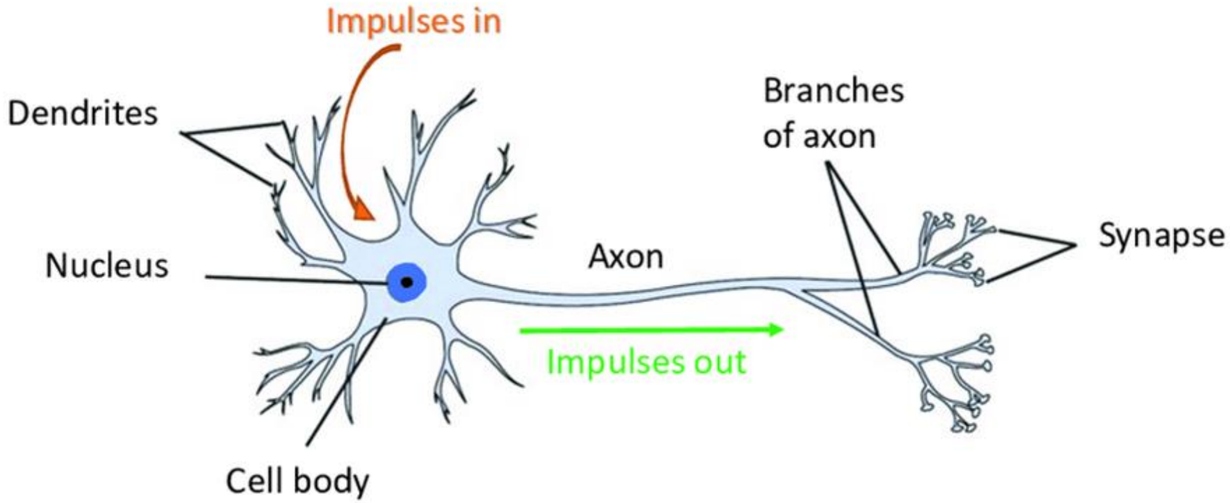
(a) Biological Neuron

Neuron in ANN

- ANN has Nodes or Units
- Each unit
  - computes a weighted sum of its inputs
  - then, applies an activation function g to derive the output activation

$$y = g\left(\sum_{i=0}^{n} w_i x_i\right), \qquad x_0 = 1$$

- Activation Functions g is mostly a Non-linear function



(a) Biological Neuron

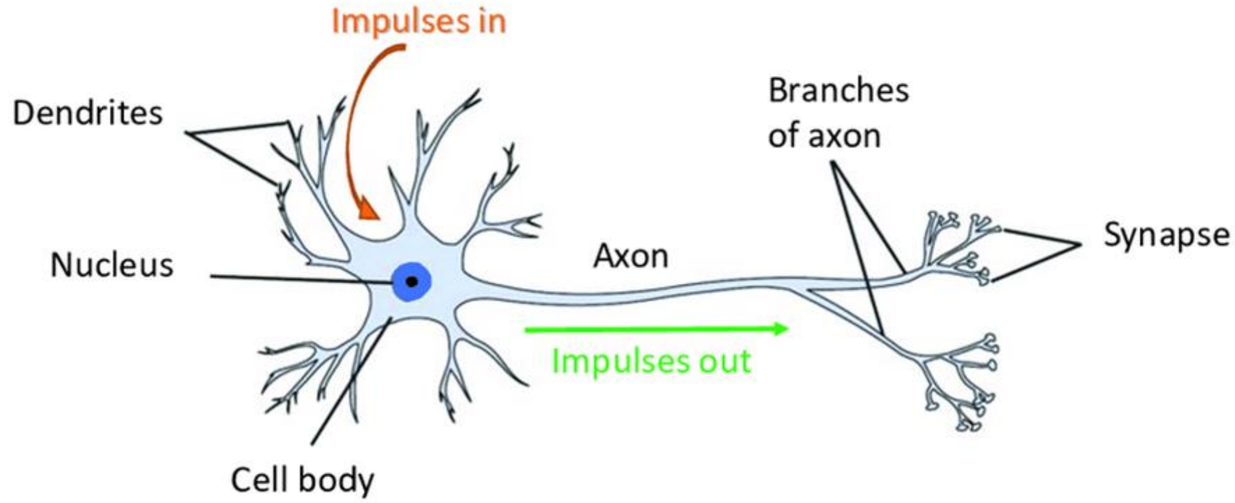
(b) Artificial Neuron

Neuron in ANN

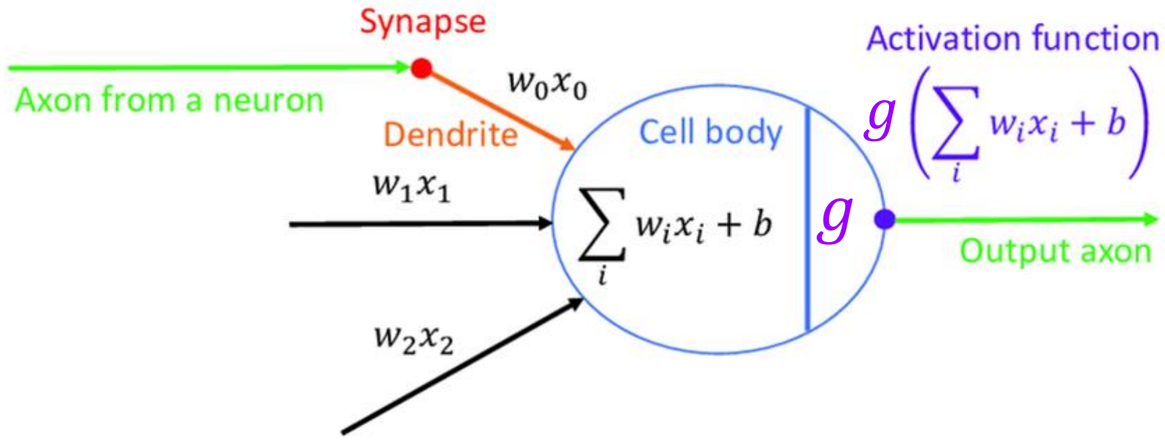
- ANN has Nodes or Units
- Each unit
  - computes a weighted sum of its inputs
  - then, applies an activation function g to derive the output activation

$$y = g\left(\sum_{i=0}^{n} w_i x_i\right), \qquad x_0 = 1$$

- Activation Functions g is mostly a Non-linear function



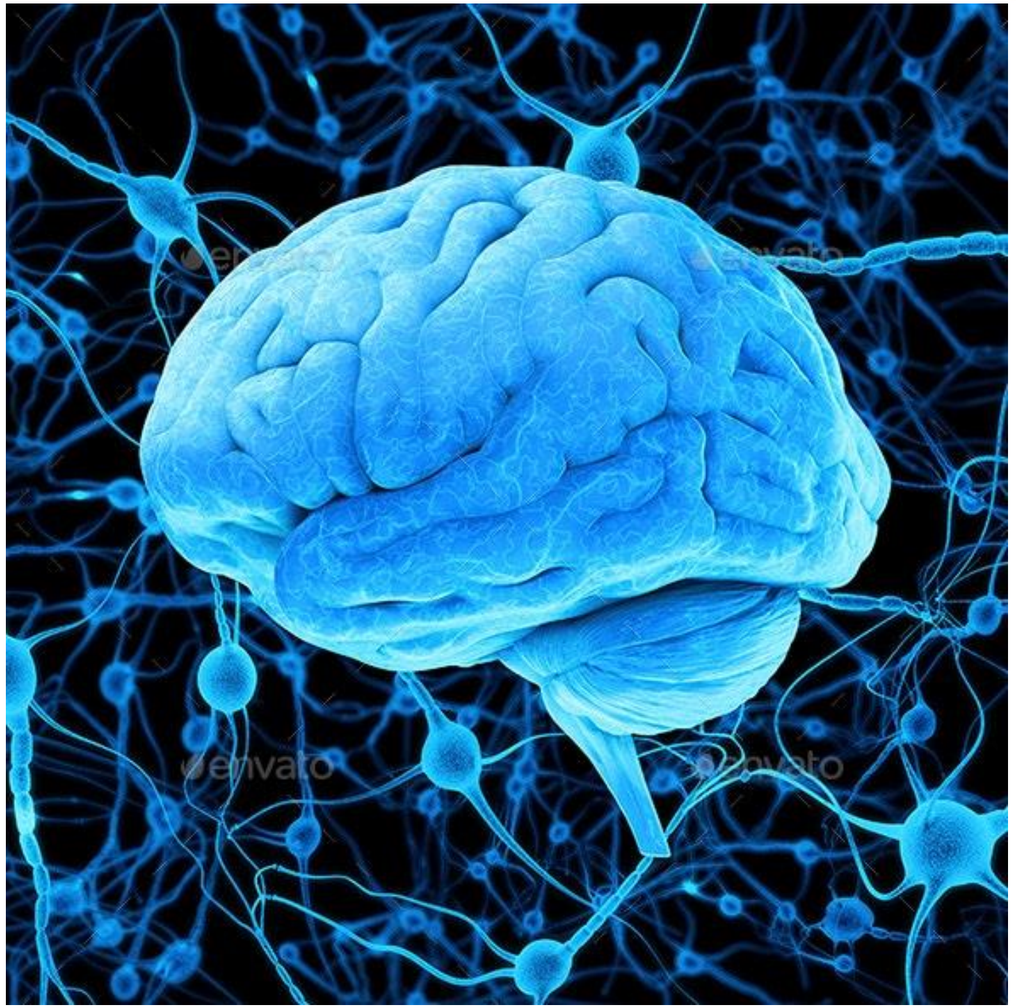
(a) Biological Neuron



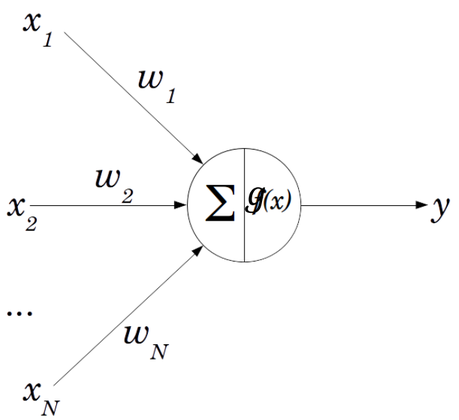
(b) Artificial Neuron

“Brain has a Network of Neurons”

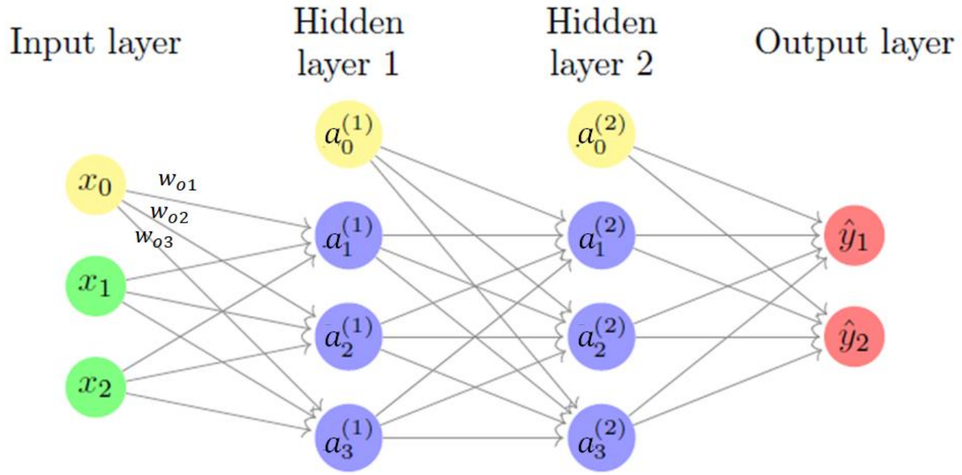
An average Human Brain has more contains about 86 billion neurons and more than a 100 trillion synapses (connections)!



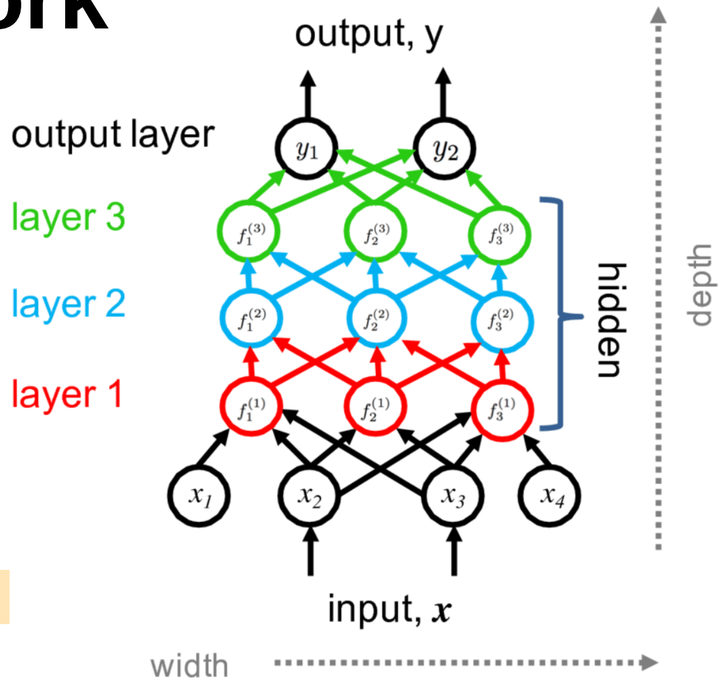
Artificial Neural Network



Single Perceptron



Multi Layer Perceptron (MLP)



- ANN has Nodes or Units connected by directed links
- The computational units are connected to one another through weights

- Weights ≡ strengths of synaptic connections in biological organisms
- Learning occurs by changing the weights connecting the neurons

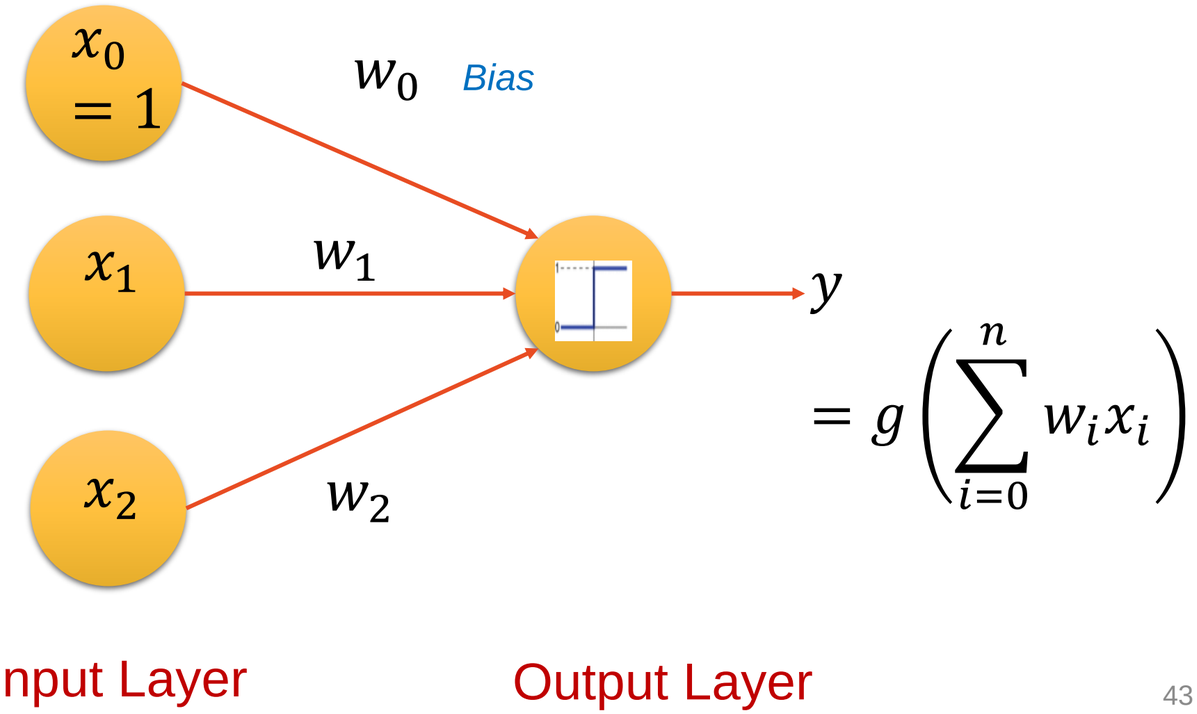
The Perceptron

The Perceptron

- NN comprising of a Single Neuron
- Single Layer Neural Network
- An artificial neuron using the Heaviside step function as the activation function

Thresholding function

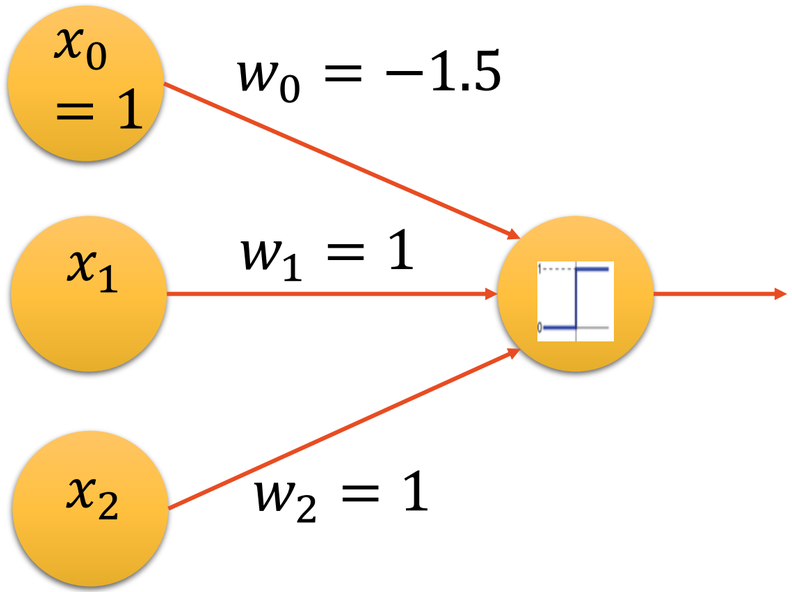
$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$



Single Layer NN: Learning simple logic

- AND Logic

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



$$y = g\left(\sum_{i=0}^{n} w_i x_i\right)$$

Thresholding function

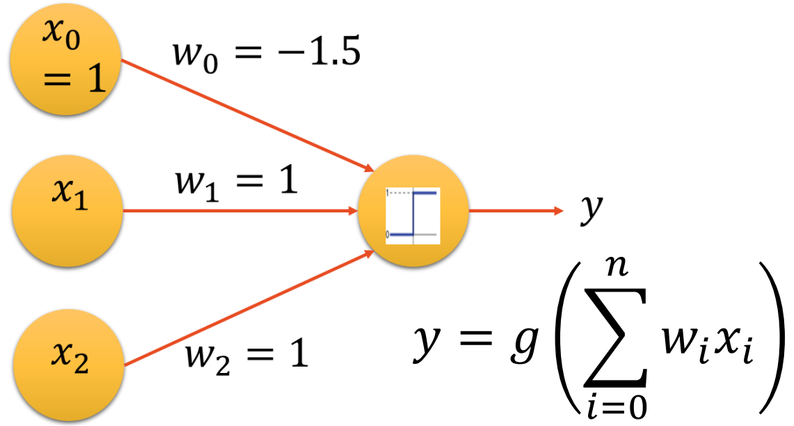
$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

Single Layer NN: Learning simple logic

- AND Logic: Equation Form

x_1	x_2	$y = g(w_0x_0 + w_1x_1 + w_2x_2) = g\left(\sum_0^2 w_ix_i\right)$
0	0	$g(-1.5 + 1\times 0 + 1\times 0) = g(-1.5) = 0$
0	1	$g(-1.5 + 1\times 0 + 1\times 1) = g(-0.5) = 0$
1	0	$g(-1.5 + 1\times 1 + 1\times 0) = g(-0.5) = 0$
1	1	$g(-1.5 + 1\times 1 + 1\times 1) = g(0.5) = 1$

Bias



Non-linearity:

Thresholding function

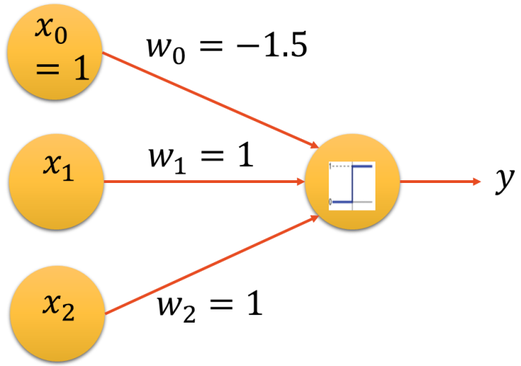
$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

Learning simple logic

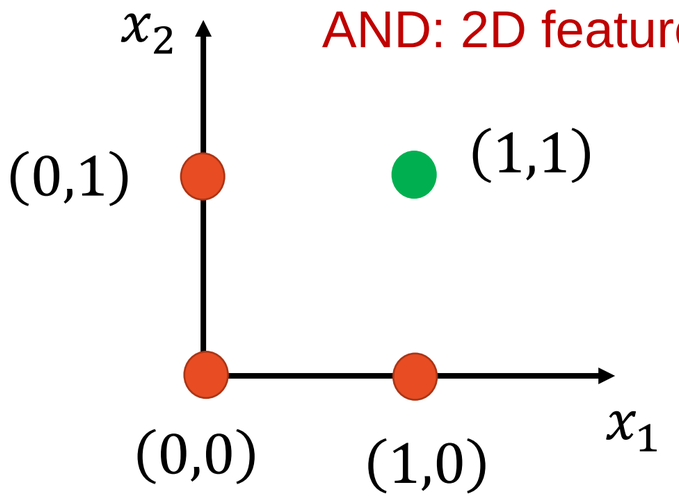
- AND Logic

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

- Shown data points are linearly separable



$$y = g(w_0x_0 + w_1x_1 + w_2x_2)$$



AND: 2D feature space

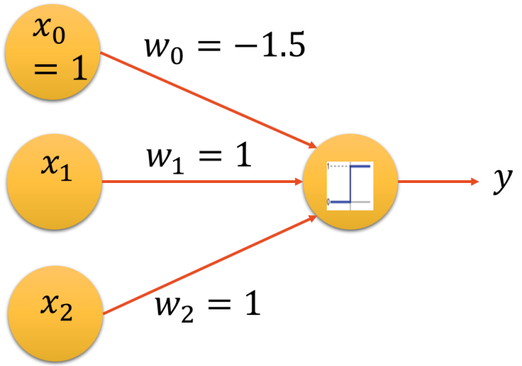
Output :0, :1

Learning simple logic

- AND Logic

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

- Shown Perceptron is a Linearly Binary Classifier



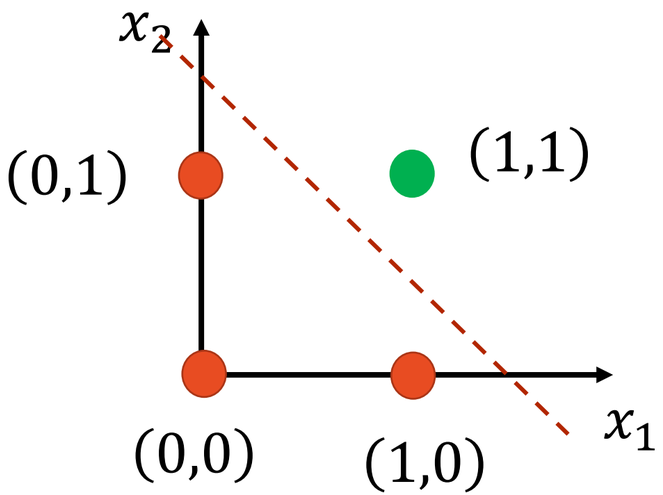
$$y = g(w_0x_0 + w_1x_1 + w_2x_2)$$

Linear separator:

$$w_0 + w_1x_1 + w_2x_2 = 0$$

Eg.

$$-1.5 + x_1 + x_2 = 0$$

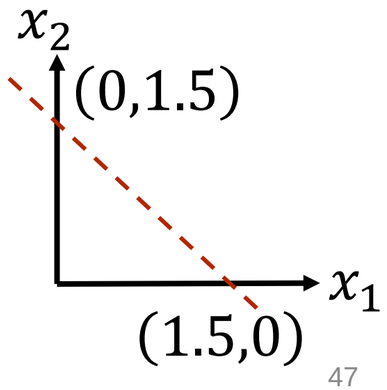


$$\frac{(x_2 - 0)}{(1.5 - 0)} = \frac{(x_1 - 1.5)}{(0 - 1.5)}$$

$$-1.5x_2 = 1.5x_1 - 1.5 * 1.5$$

$$-x_2 = x_1 - 1.5$$

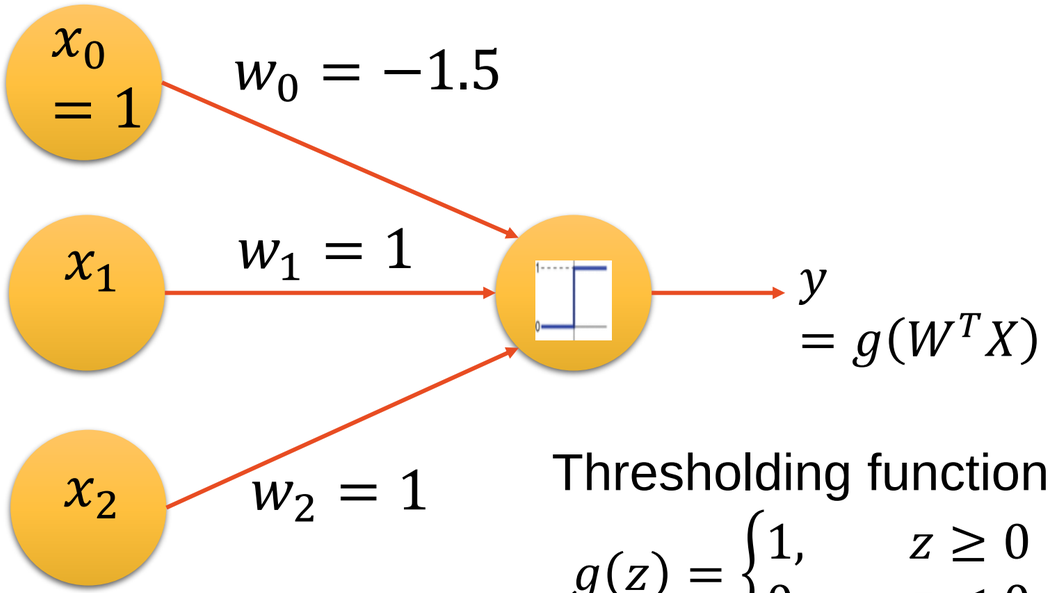
$$-1.5 + x_1 + x_2 = 0$$



Single Layer NN: Learning simple logic

- AND Logic

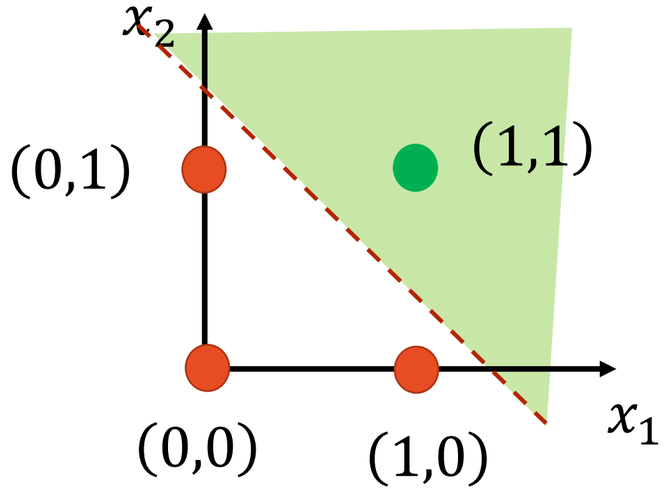
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



Thresholding function

$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

$-1.5 + x_1 + x_2 > 0 \Rightarrow$ Class 1



AND: 2D feature space

Output ● :0, ● :1

Single Layer NN: Learning simple logic

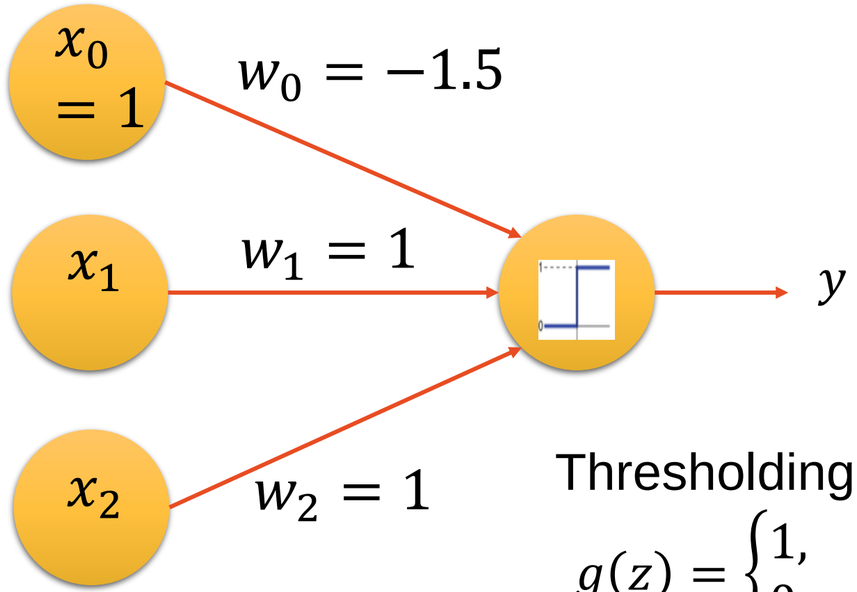
- AND Logic: Matrix Notation
- $W = \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}, X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$
- $W^T X = \underset{1 \times 3}{\begin{bmatrix} -1.5 & 1 & 1 \end{bmatrix}} \underset{3 \times 4}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}}$

$$= [-1.5 \quad -0.5 \quad -0.5 \quad 0.5]$$

- Output activation $y = g(W^T X)$

$$= [0 \quad 0 \quad 0 \quad 1]$$

$$y = g(w_0 x_0 + w_1 x_1 + w_2 x_2) = g(W^T X)$$

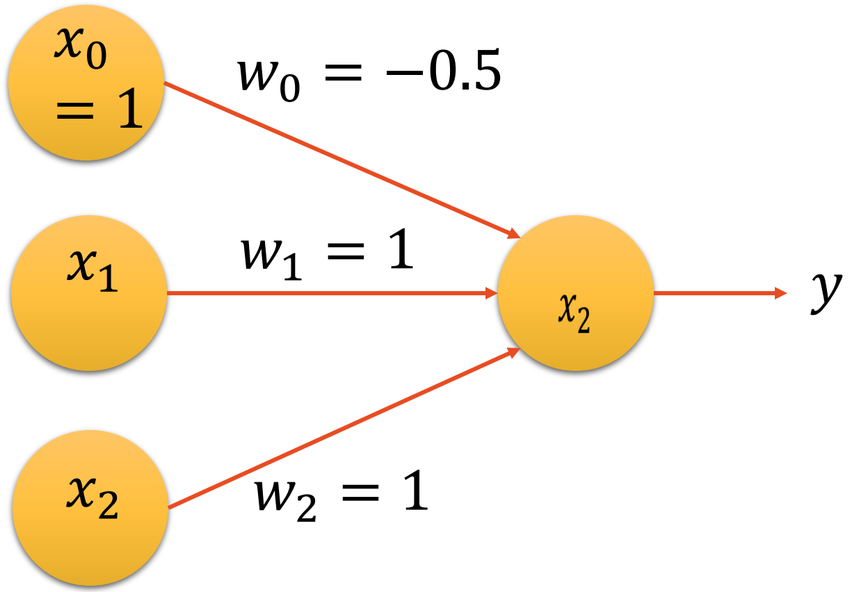


Thresholding function

$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

Q. Which logic does this perceptron learn?

a) AND
b) OR
c) NAND
d) NOR

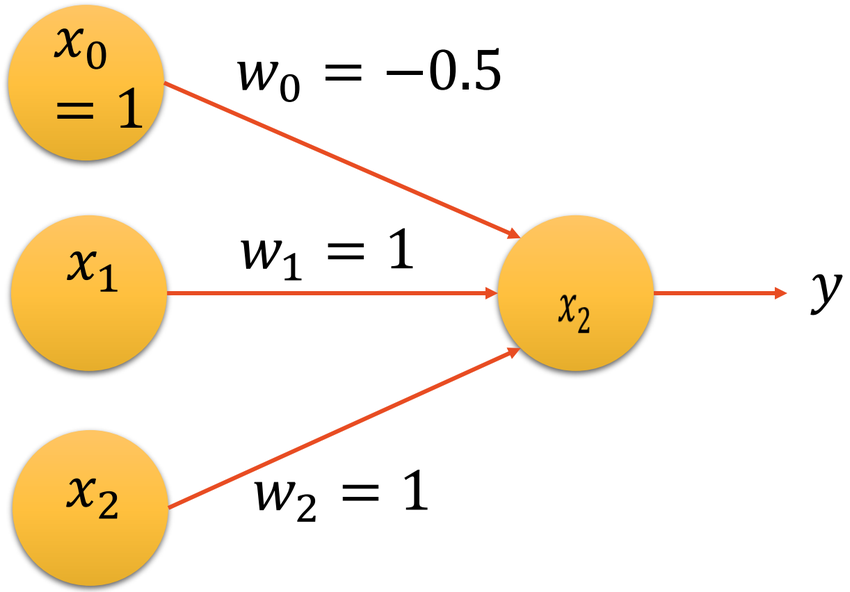


x_1, x_2 are either 0 or 1

Q. Which logic does this perceptron learn?

Given:

- $W = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}, X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$
- $W^T X = \underset{1 \times 3}{\begin{bmatrix} -0.5 & 1 & 1 \end{bmatrix}} \underset{3 \times 4}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}} = ?$
- Output activation $y = g(W^T X)$



x_1, x_2 are either 0 or 1

OR Logic

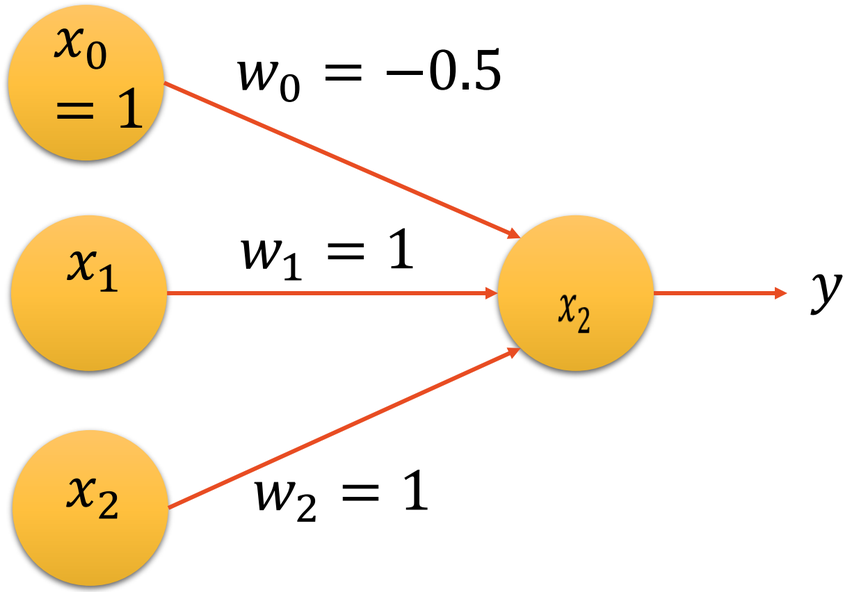
Given:

- $W = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}, X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$
- $W^T X = \underset{1 \times 3}{\begin{bmatrix} -0.5 & 1 & 1 \end{bmatrix}} \underset{3 \times 4}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}}$

$$= [-0.5 \quad 0.5 \quad 0.5 \quad 1.5]$$

- Output activation $y = g(W^T X)$

$$= [0 \quad 1 \quad 1 \quad 1]$$



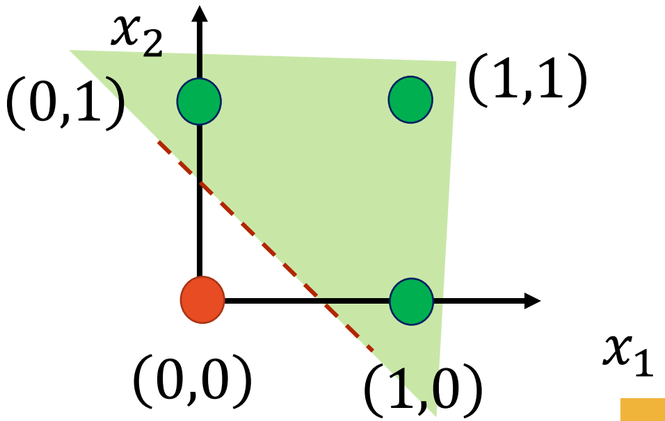
x_1, x_2 are either 0 or 1

Weights Enable Learning !

Given:

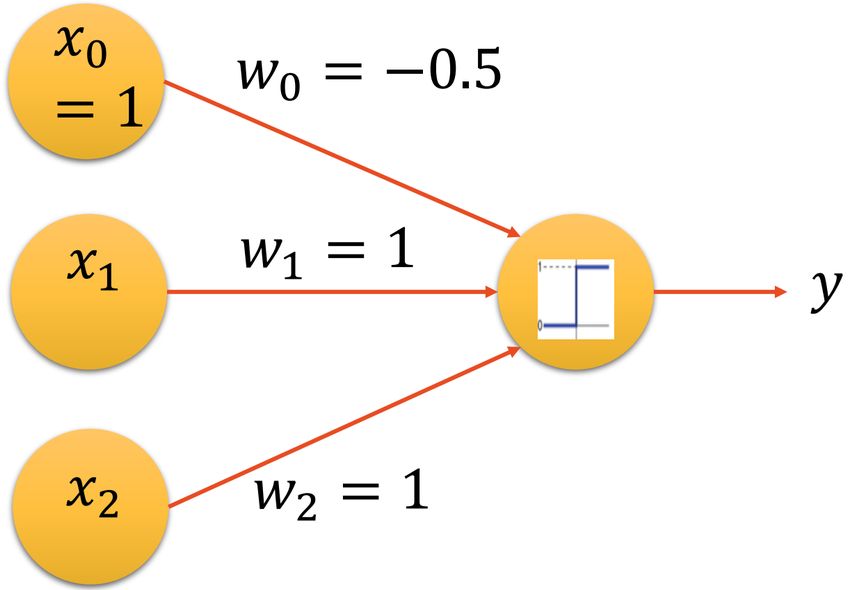
- $W = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}, X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$
- Output activation $y = g(W^T X)$

$$= [0 \quad 1 \quad 1 \quad 1]$$



x_1	x_2	$y = g(W^T X)$
0	0	0
0	1	1
1	0	1
1	1	1

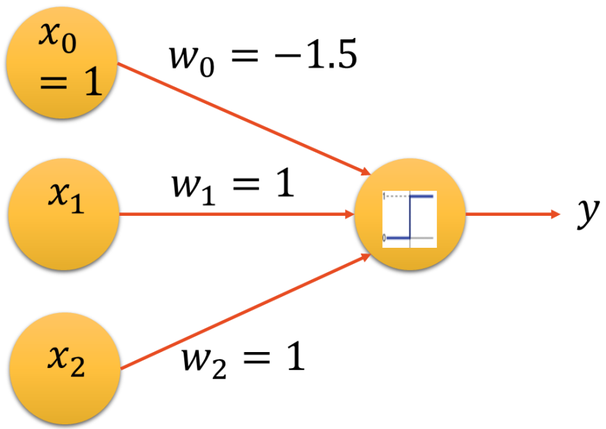
OR Logic

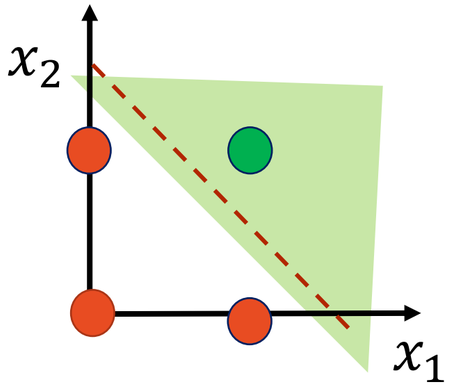


Thresholding function

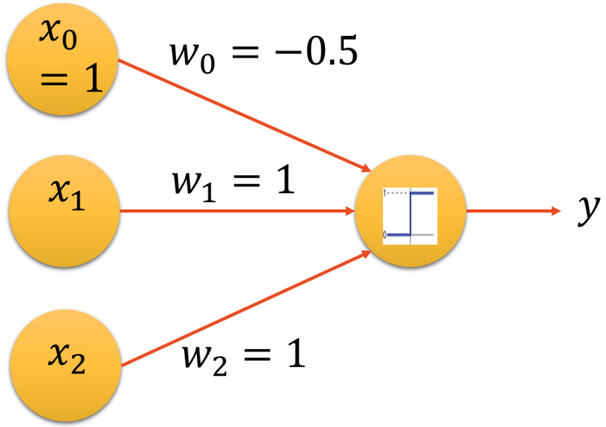
$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

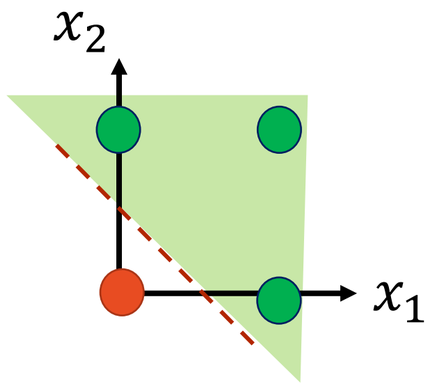
AND



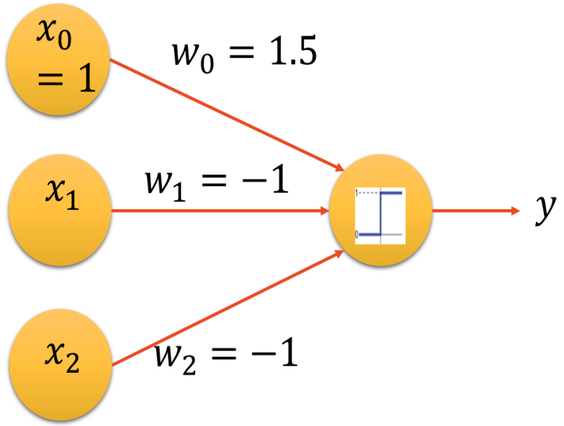


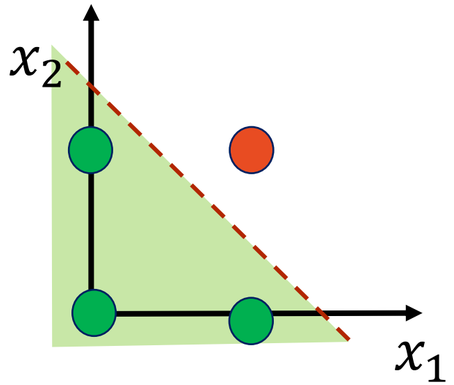
OR



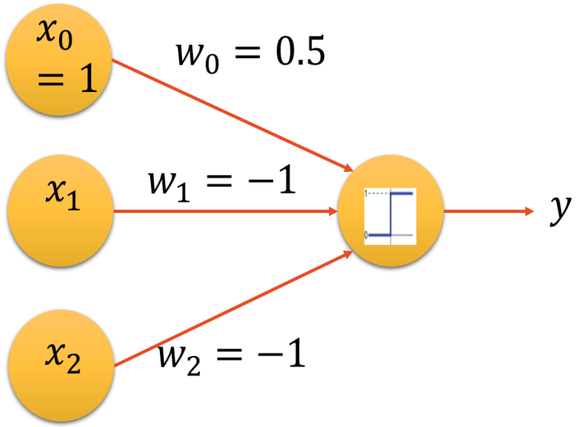


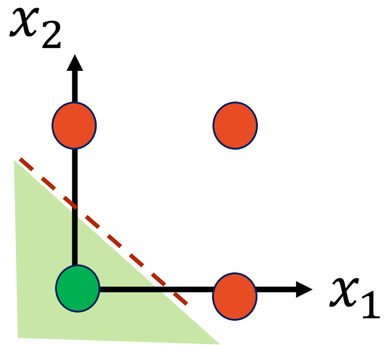
NAND





NOR

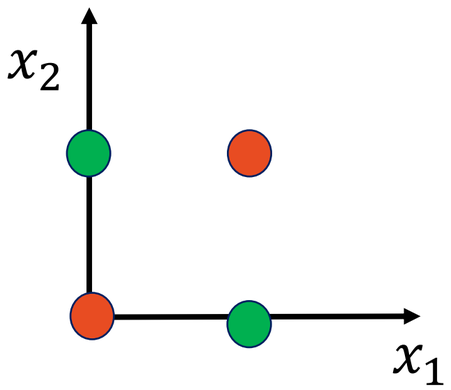




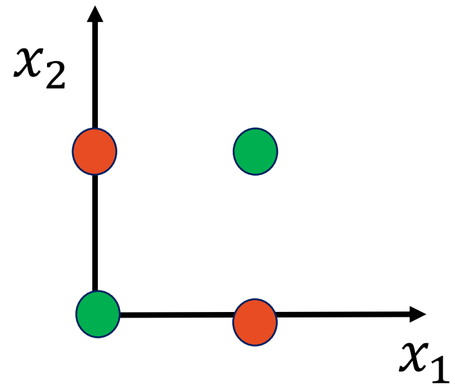
Can these be classified using a Linear Classifier ?

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



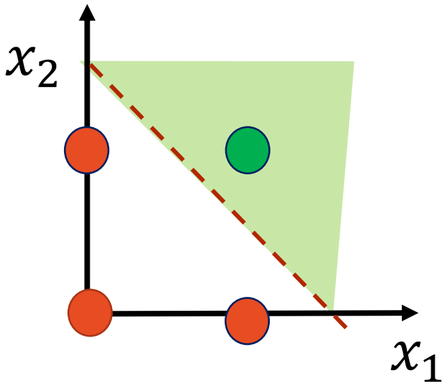
XNOR



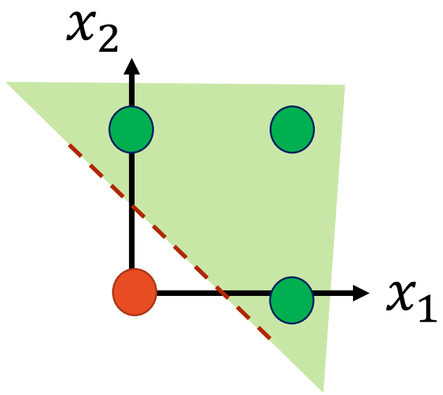
x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

Linearly Separable

AND

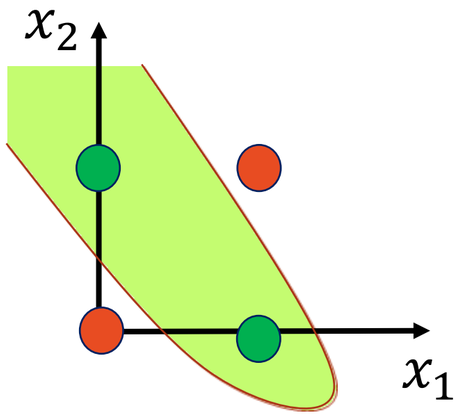


OR

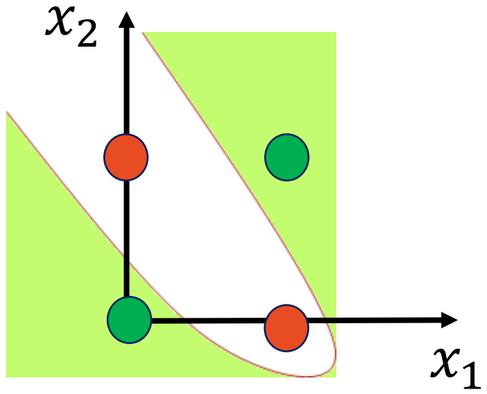


Not Linearly Separable

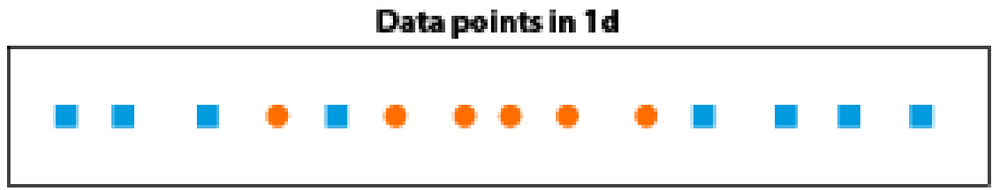
XOR



XNOR



Datapoint (1D) Not linearly Separable

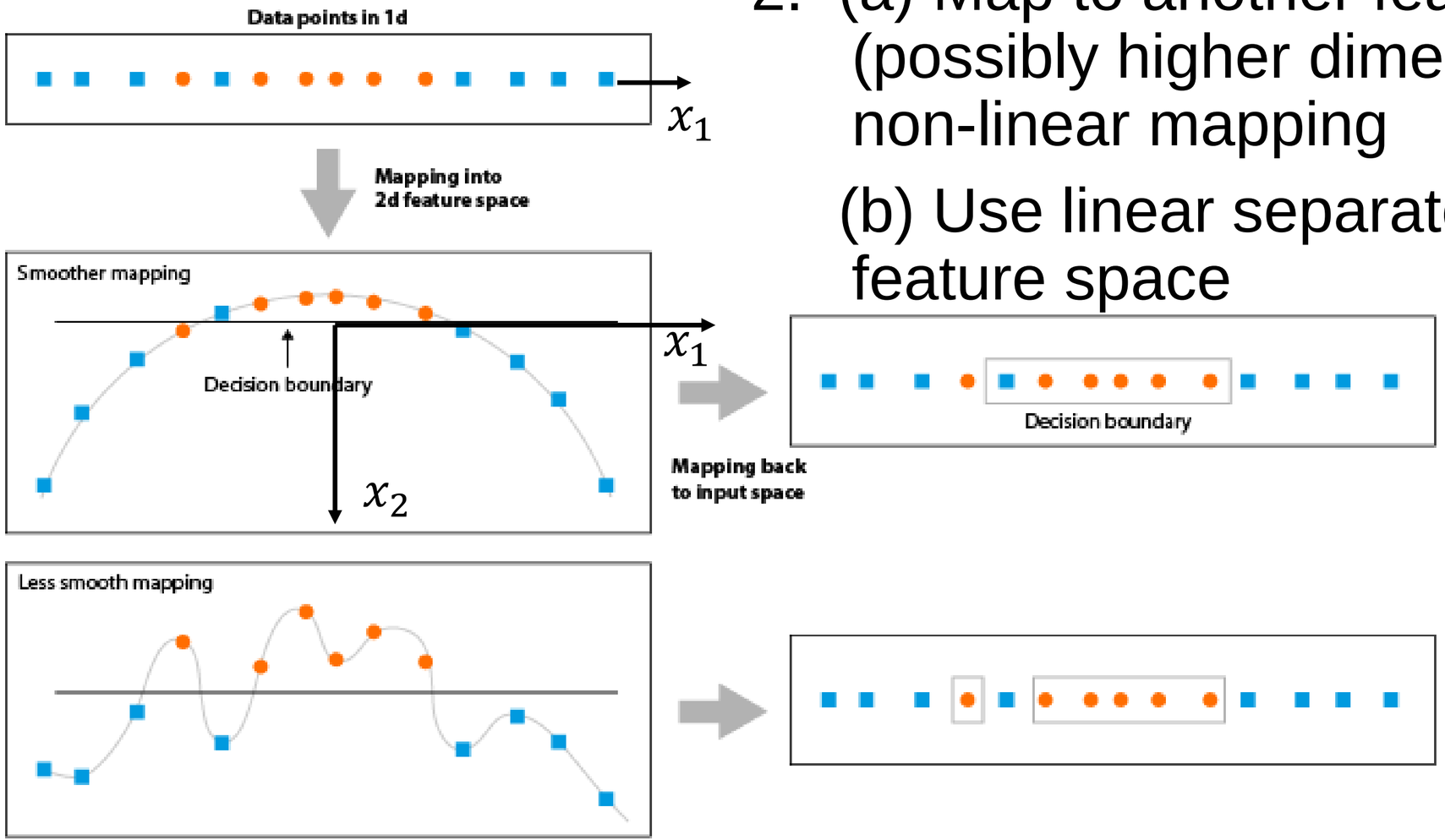


1. Non-linear separator

Not linearly Separable

2. (a) Map to another feature space (possibly higher dimension) using non-linear mapping

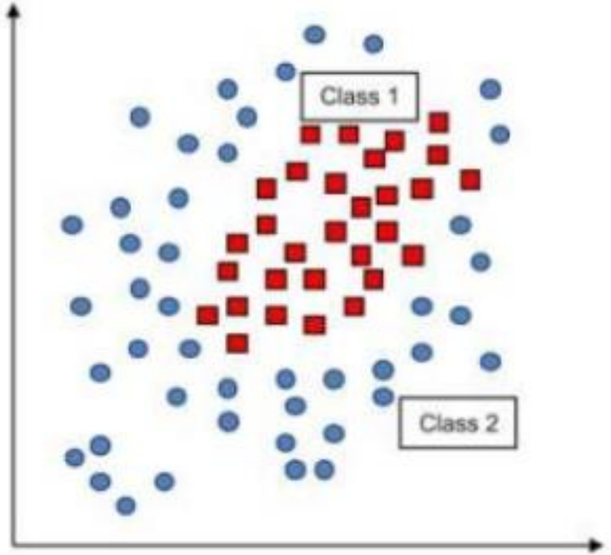
   (b) Use linear separator in new feature space

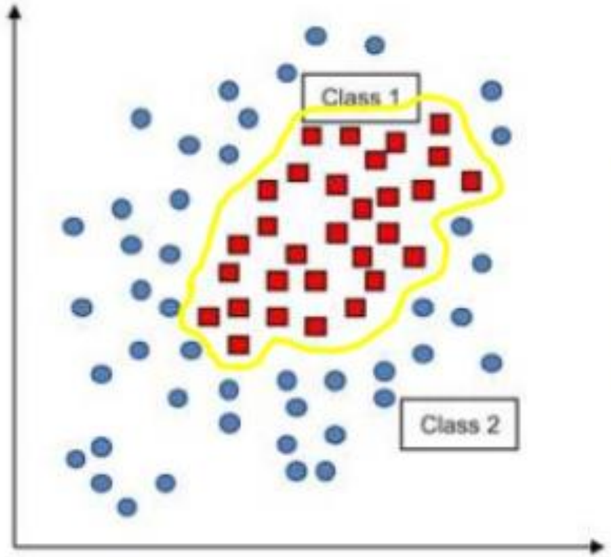


AMITY UNIVERSITY

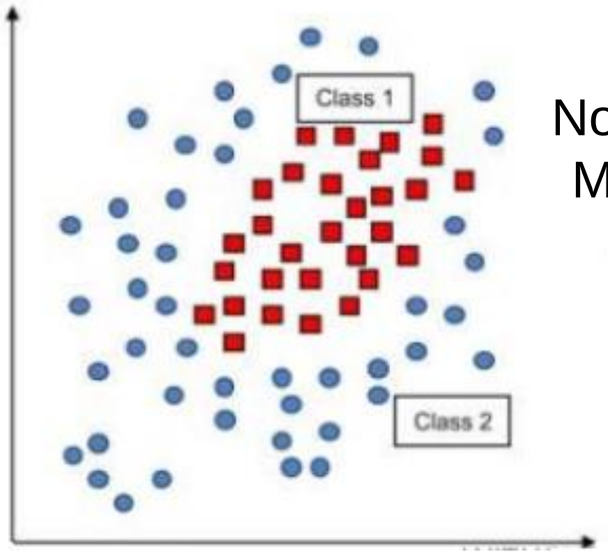
Datapoints in 2D feature space-

Not Linearly Separable

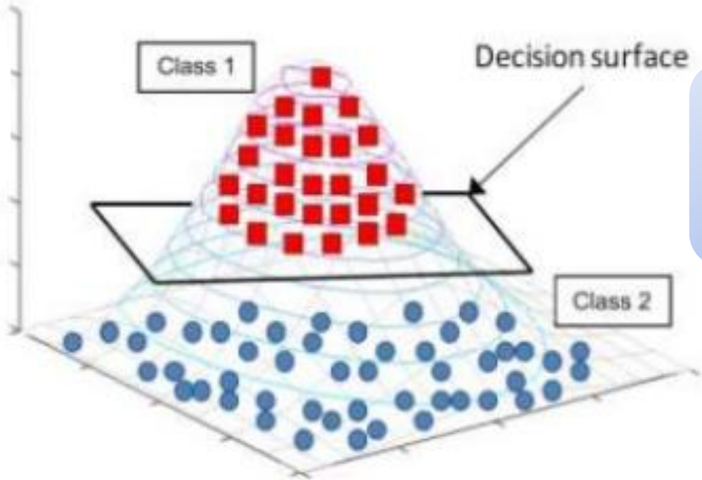




Non-linear Decision Boundary

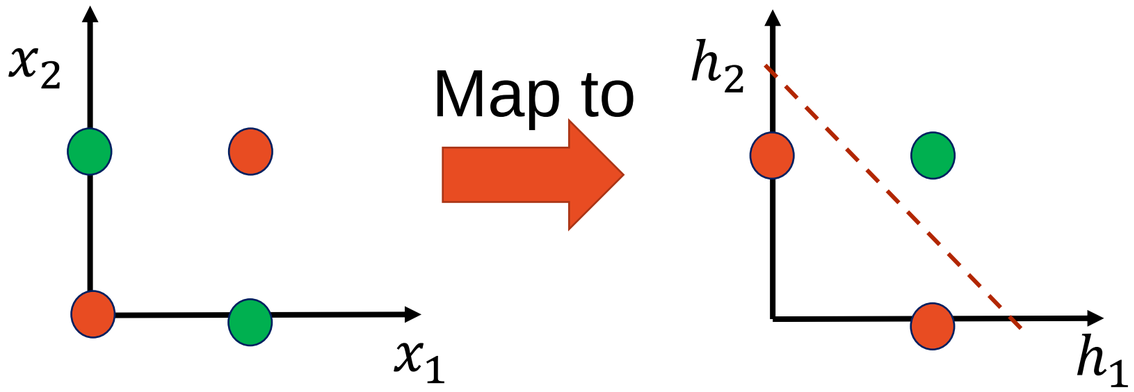


Non-linear Mapping



Linearly Separable

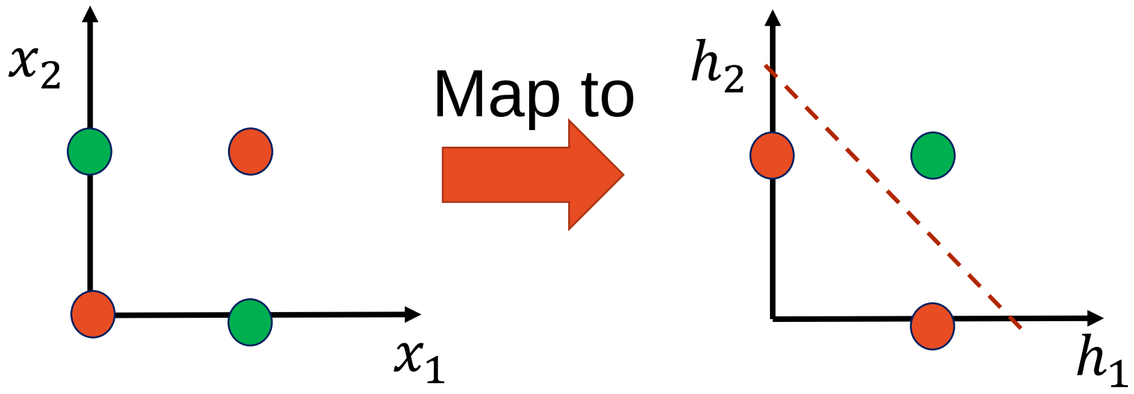
Learning XOR Logic: Multi-layer Perceptron



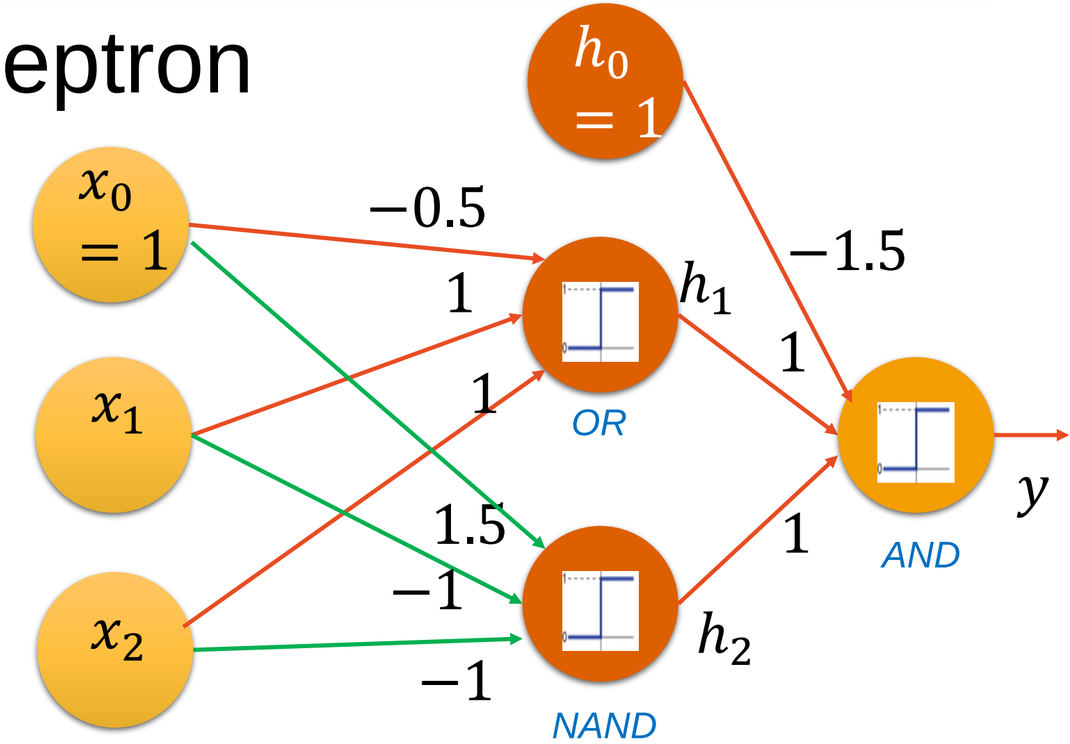
x_1 XOR x_2 =
(x_1 OR x_2) AND (x_1 NAND x_2)

x_1	x_2	$h_1 = x_1$ **OR** x_2	$h_2 = x_1$ **NAND** x_2	$y = h_1$ **AND** h_2
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Learning XOR Logic: Multi-layer Perceptron

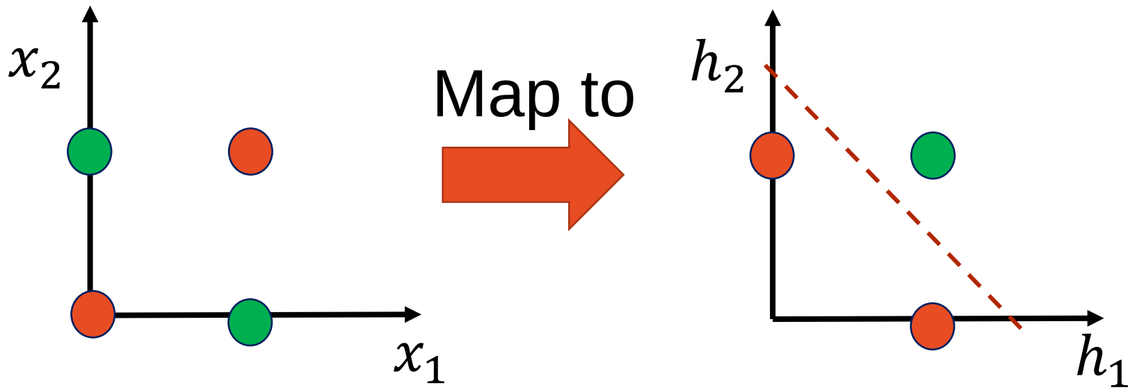


x_1	x_2	$h_1 = x_1$ **OR** x_2	$h_2 = x_1$ **NAND** x_2	$y = h_1$ **AND** h_2
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

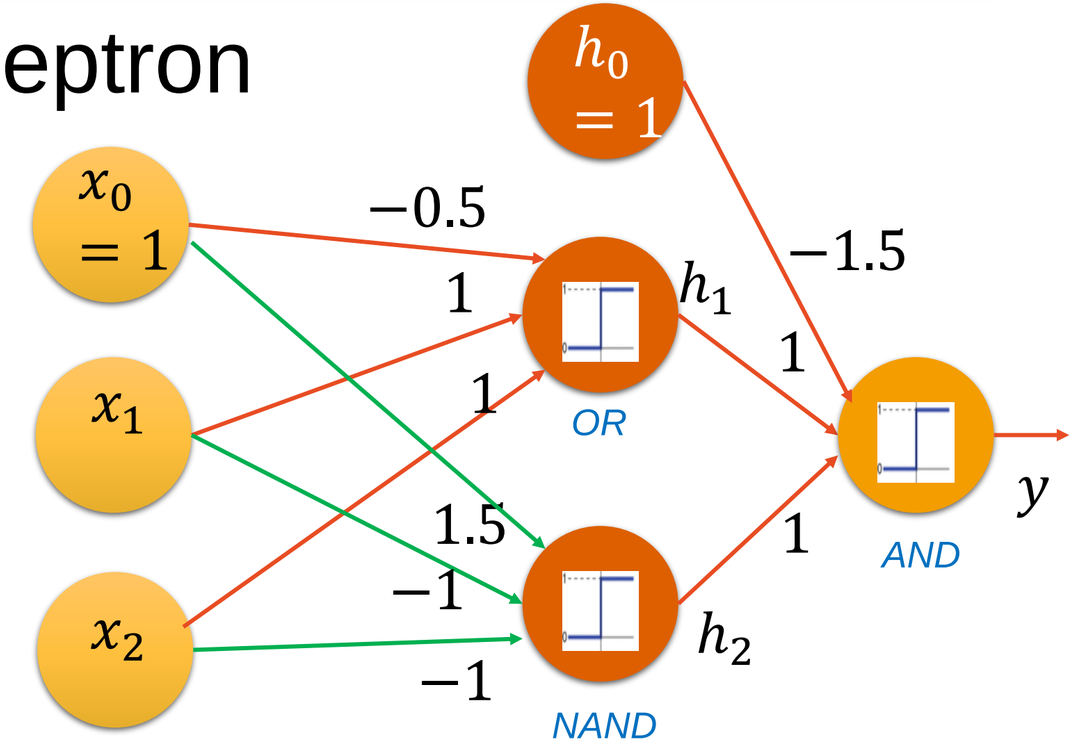


x_1 XOR x_2 =
(x_1 OR x_2) AND (x_1 NAND x_2)

Learning XOR Logic: Multi-layer Perceptron



x_1	x_2	$h_1 = x_1$ **OR** x_2	$h_2 = x_1$ **NAND** x_2	$y = h_1$ **AND** h_2
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0



2 layers NN

Learning XOR Logic: Multi-layer Perceptron

Layer 1 (Hidden Layer)

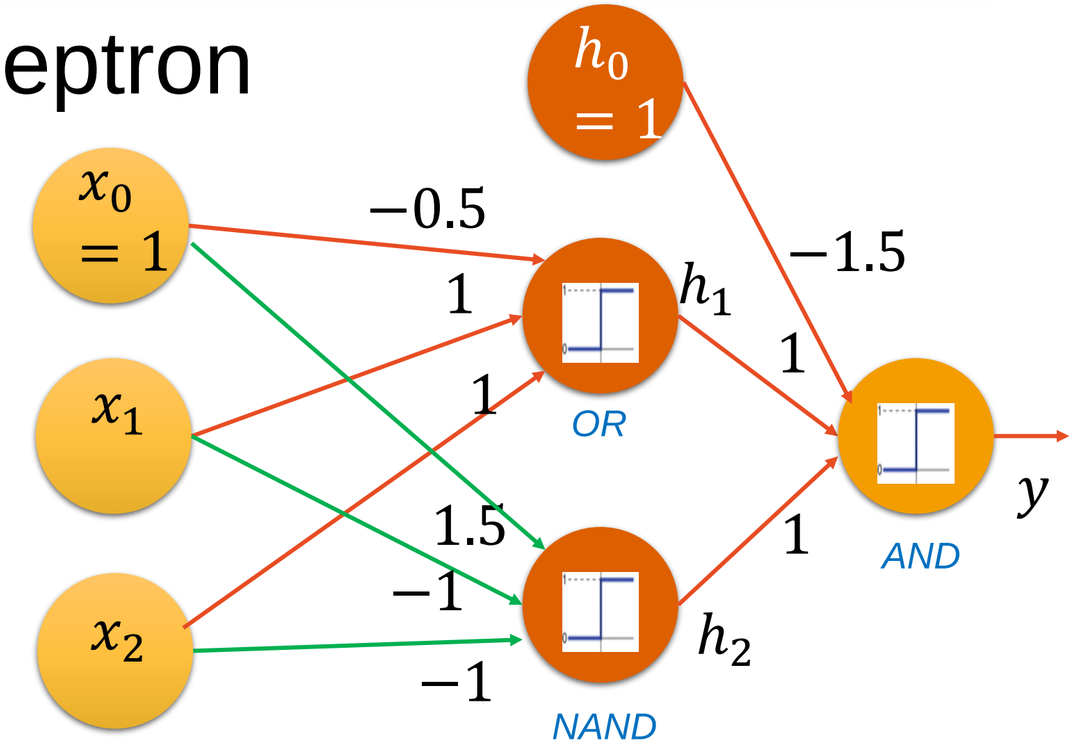
- $W_1{}^T X = \begin{bmatrix} -0.5 & 1 & 1 \\ 1.5 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

$$= \begin{bmatrix} -0.5 & 0.5 & 0.5 & 1.5 \\ 1.5 & 0.5 & 0.5 & -0.5 \end{bmatrix}$$

$$h = g\left(W_1{}^T X\right) \Rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Layer 2 (Output Layer)

$$y = g\left(W_2{}^T h\right) =$$



Learning XOR Logic: Multi-layer Perceptron

Layer 1 (Hidden Layer)

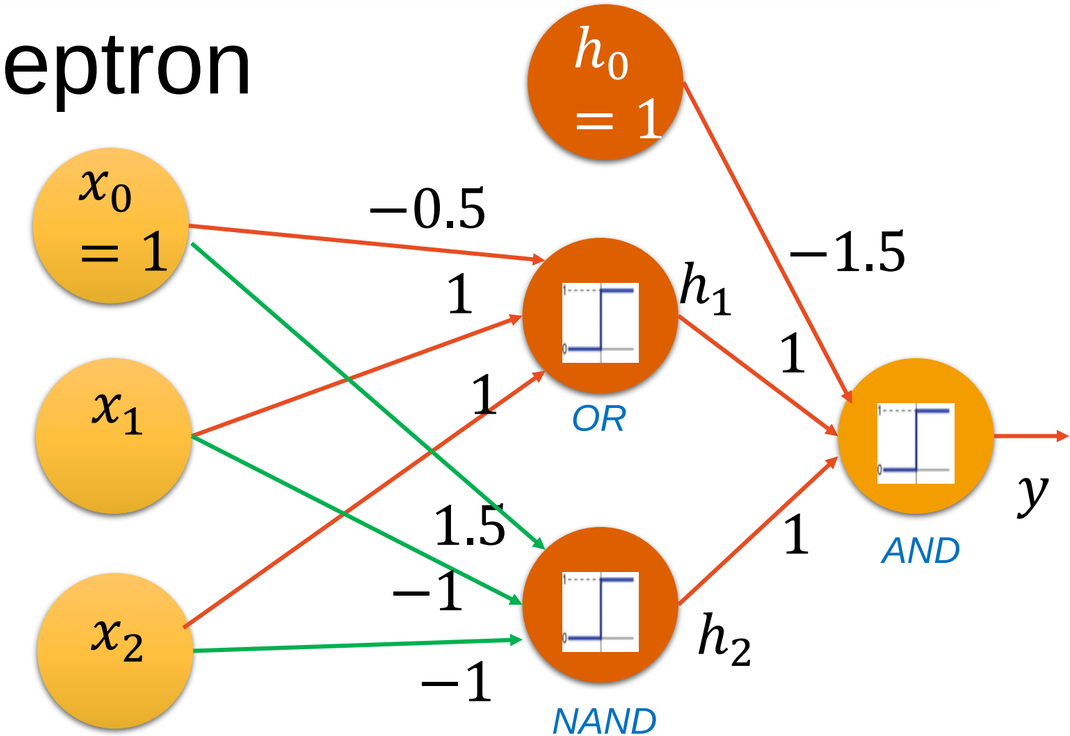
- $W_1{}^T X = \begin{bmatrix} -0.5 & 1 & 1 \\ 1.5 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

$$= \begin{bmatrix} -0.5 & 0.5 & 0.5 & 1.5 \\ 1.5 & 0.5 & 0.5 & -0.5 \end{bmatrix}$$

$$h = g\left(W_1{}^T X\right) \Rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Layer 2 (Output Layer)

$$y = g\left(W_2{}^T h\right) = [0 \quad 1 \quad 1 \quad 0]$$

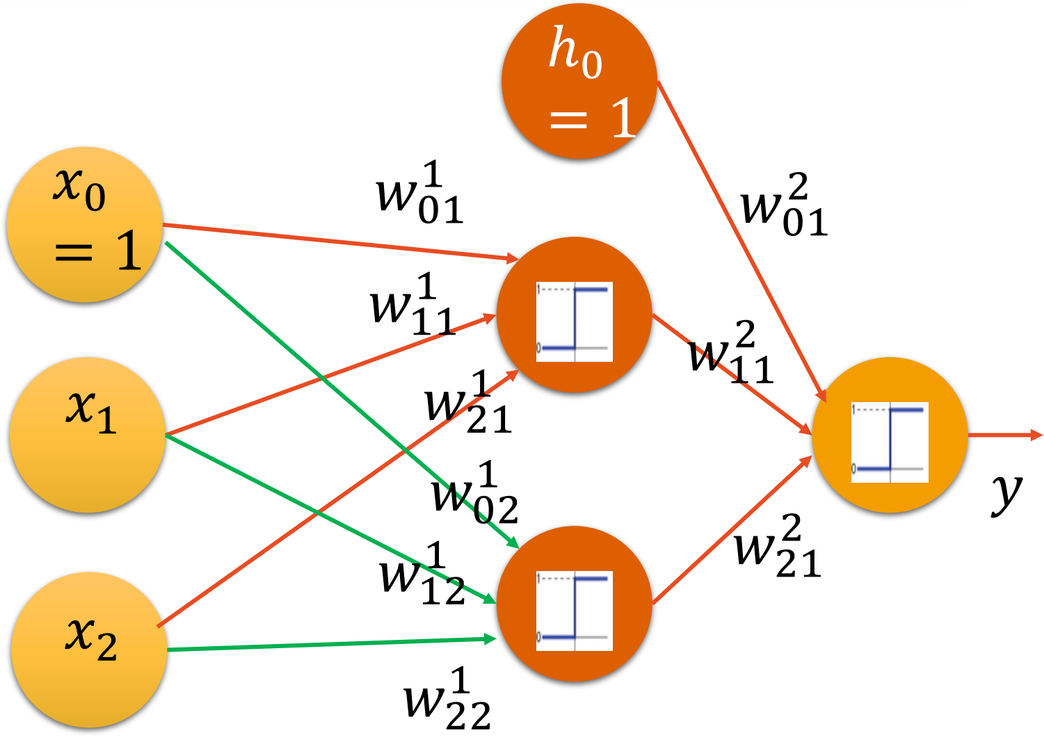


$$W_2{}^T h = [-1.5 \quad 1 \quad 1] \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$= [-0.5 \quad 0.5 \quad 0.5 \quad -0.5]$$

Try it Yourself: XNOR Logic

x_1	x_2	$h_1 = x_1$ **NOR** x_2	$h_2 = x_1$ **AND** x_2	$y = h_1$ **OR** h_2
0	0	1	0	1
0	1	0	0	0
1	0	0	0	0
1	1	0	1	1



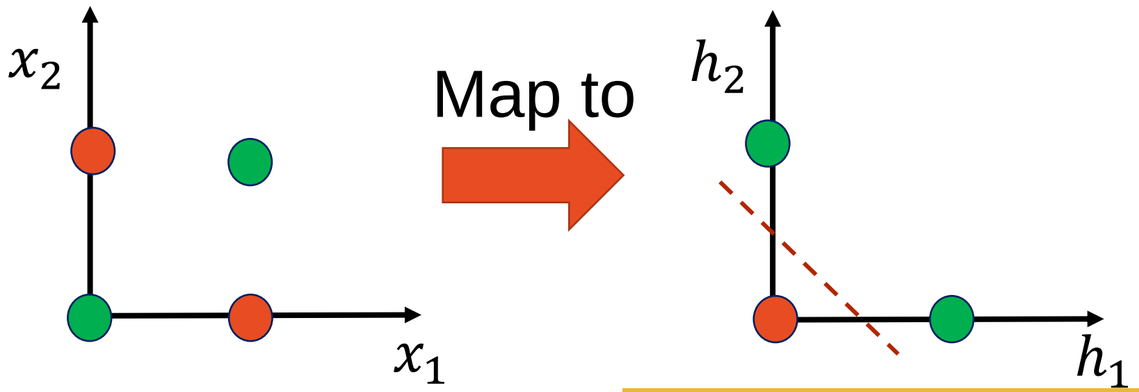
Determine the Weights.

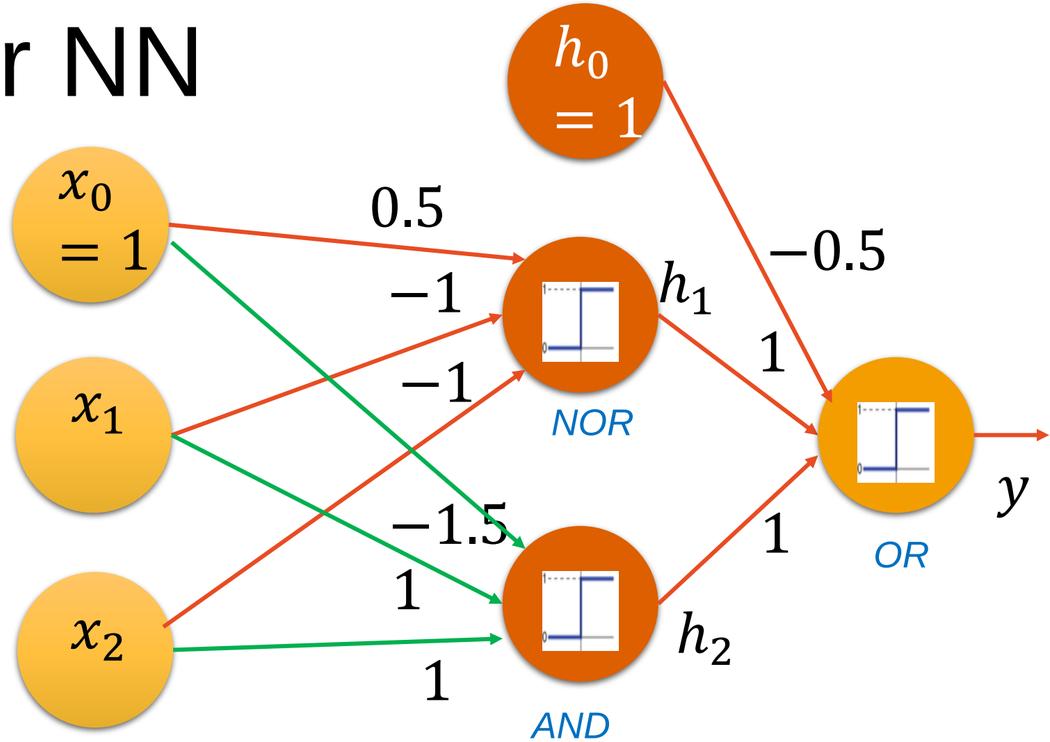
$w_{ij}^{(k+1)}$ = Weight of connection between i^{th} node in k^{th} layer to j^{th} node in $(k+1)^{th}$ layer

x_1 XNOR x_2 =
(x_1 NOR x_2) OR (x_1 AND x_2)

Learning XNOR Logic: Multi-layer NN

x_1	x_2	$h_1 = x_1$ **NOR** x_2	$h_2 = x_1$ **AND** x_2	$y = h_1$ **OR** h_2
0	0	1	0	1
0	1	0	0	0
1	0	0	0	0
1	1	0	1	1





Input Layer → **Hidden Layer** → **Output Layer**

2 layers NN

Learning XNOR Logic: Multi-layer NN

Layer 1 (Hidden Layer)

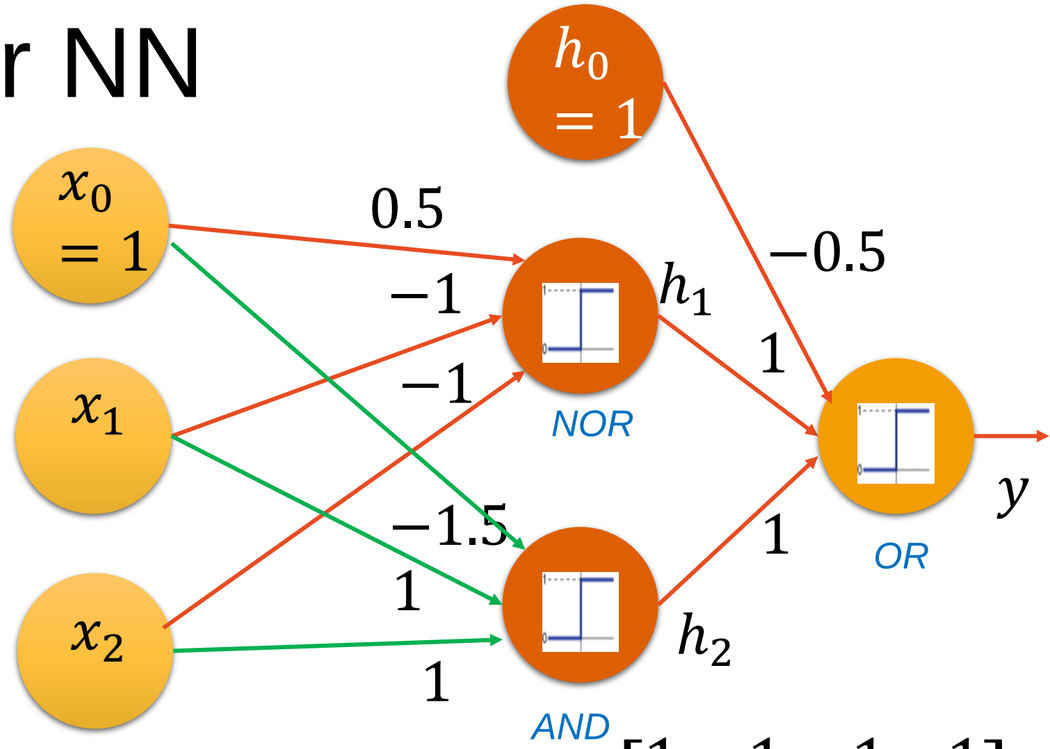
- $W_1{}^T X = \begin{bmatrix} 0.5 & -1 & -1 \\ -1.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

$$= \begin{bmatrix} 0.5 & -0.5 & -0.5 & -1.5 \\ -1.5 & -0.5 & -0.5 & 0.5 \end{bmatrix}$$

$$h = g(W_1{}^T X) \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Layer 2 (Output Layer)

$$y = g(W_2{}^T h) = [1 \quad 0 \quad 0 \quad 1]$$

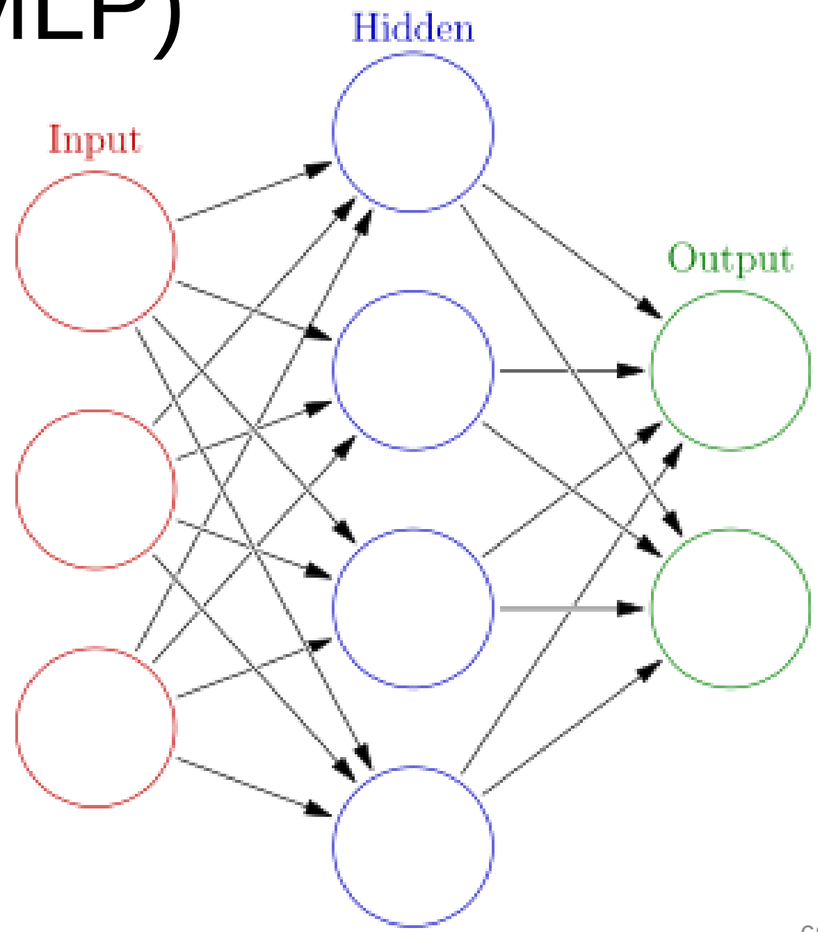


$$W_2{}^T h = [-0.5 \quad 1 \quad 1] \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= [0.5 \quad -0.5 \quad -0.5 \quad 0.5]$$

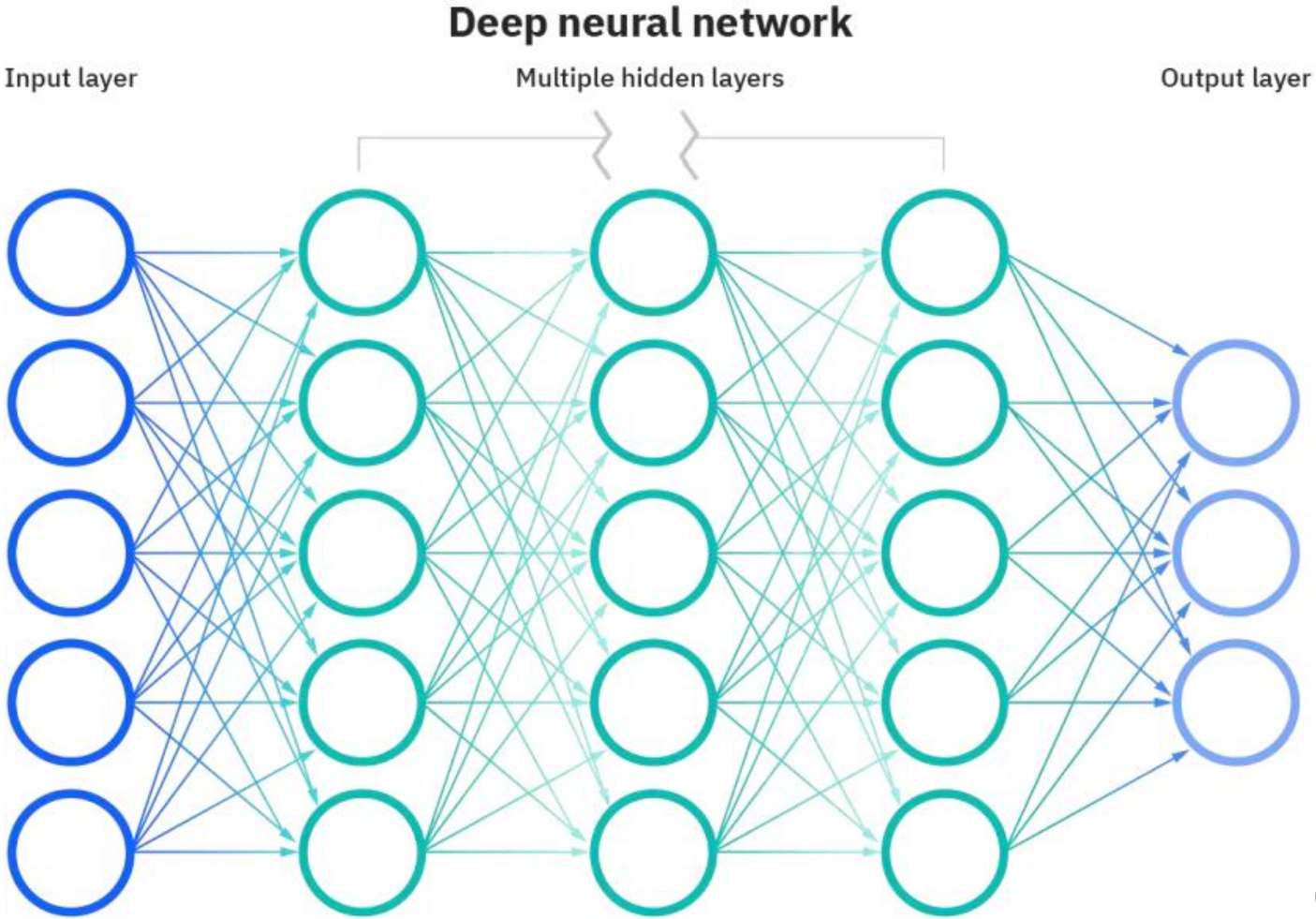
Multilayer Perceptron (MLP)

- MLP is a feedforward artificial neural network
  - Data flows in forward direction
- MLP must have at least three layers: the input layer, a hidden layer and the output layer
- They are fully connected; each node in one layer connects with a weight to every node in the next layer
- Each node computes a non-linear function



ANN can learn more complex boundaries by Increasing

- the Neurons in Hidden Layers
- the Number of Hidden Layers

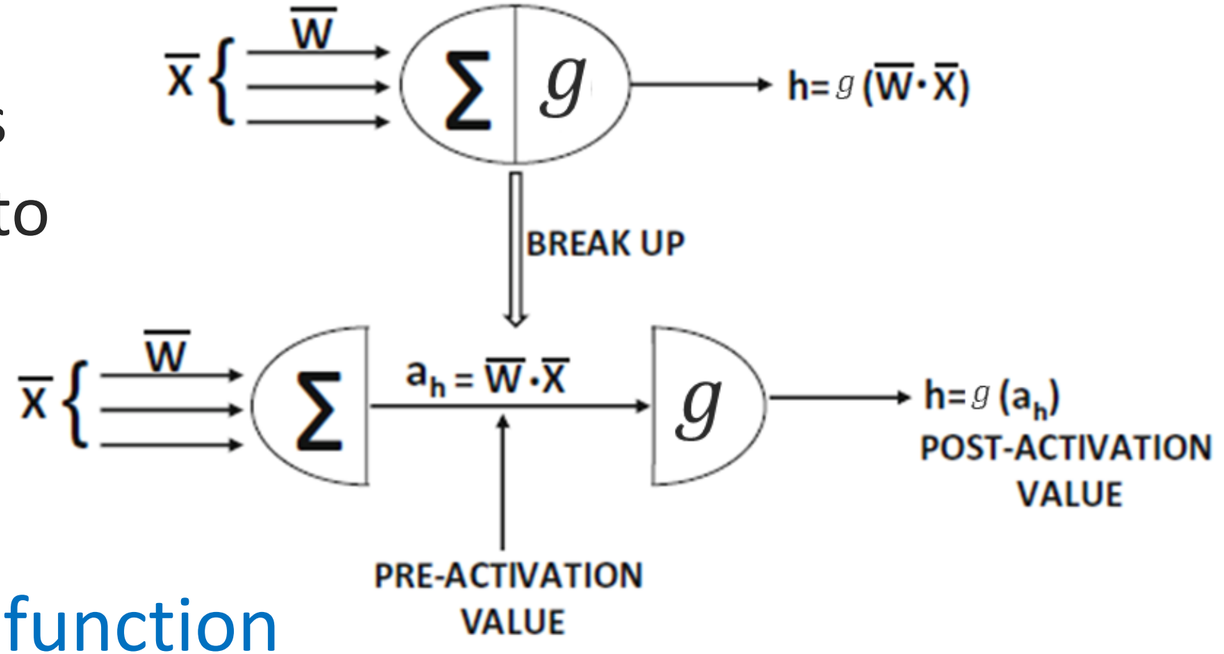


Activation Functions

Structure of a Node/Unit in ANN

- Each unit in ANN
  - computes a weighted sum of its inputs
  - then, applies an activation function g to derive the output activation

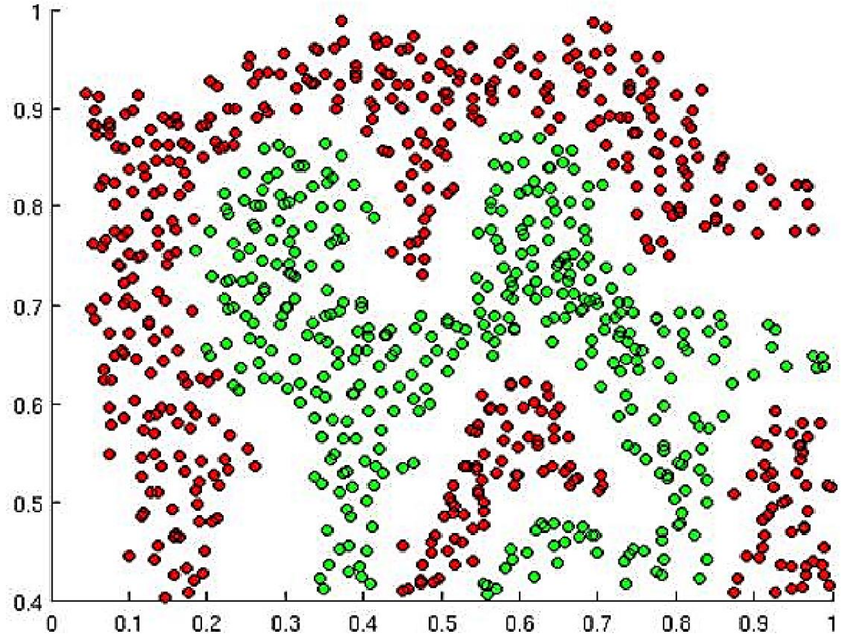
$$h = g\left(\sum_{i=0}^{n} w_{ij} x_i\right), \qquad x_0 = 1$$



- An activation function is a non-linear function applied by a neuron to introduce non-linear properties in the network

Why Activation Function?

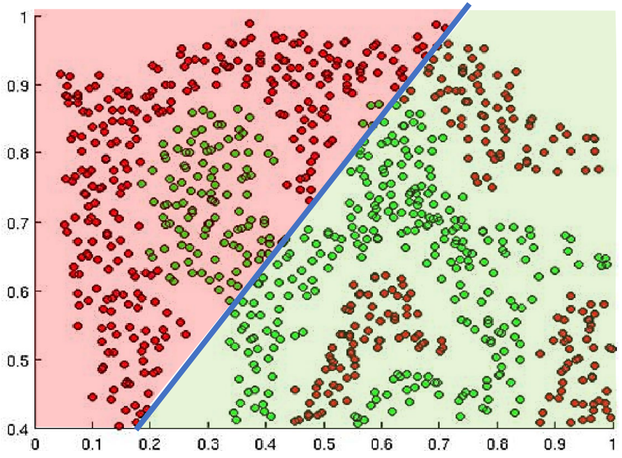
- Application of the activation function tells us that which neurons in each layer will be triggered. Only the neurons with some relevant information are activated in every layer.
- The activation takes place depending on some rule or threshold
- The purpose of the activation function is to **introduce non-linearity** into the network.
- As most of the data in real life is non linear.



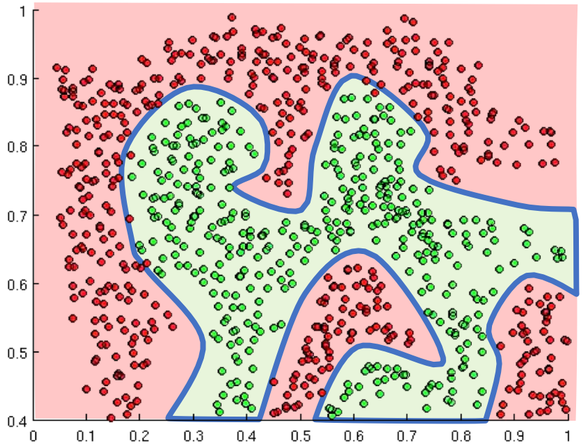
Example : **Separating green points from red points in the graph.**

Importance of Activation Functions

- Separating green points from red points using linear function results into under fitting problem.
- No matter how deep and how large is the network if using linear activation function it just composing lines on top of lines to get another line.
- Whereas using non linear function generates non linear boundaries in the network which is extremely powerful for classification task.



Linear Activation functions produce linear decisions no matter the network size

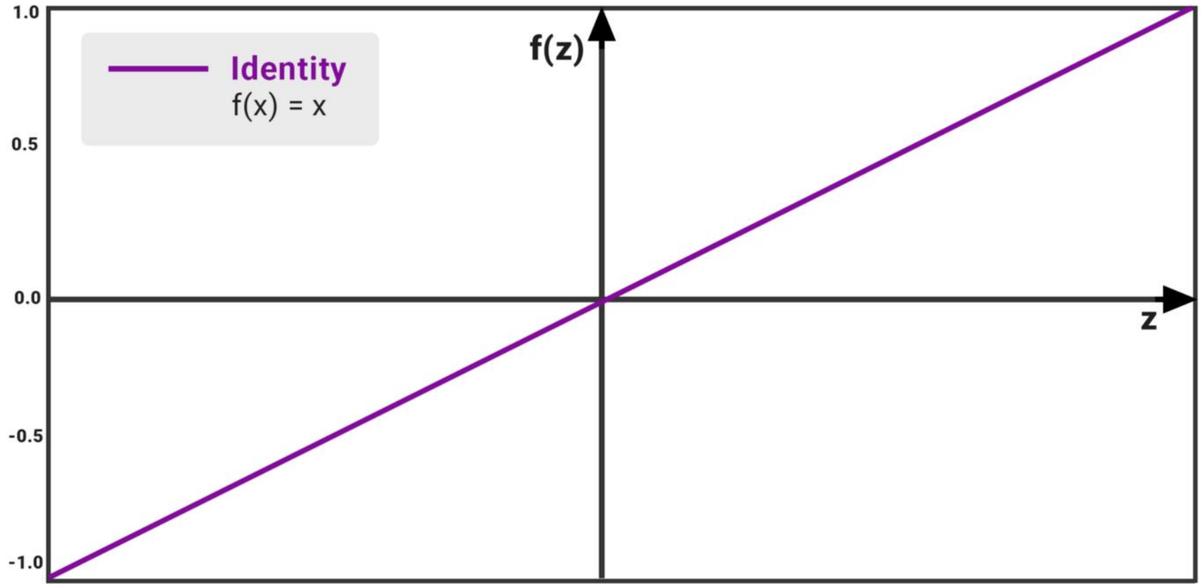


Non-linearities allow us to approximate arbitrarily complex functions

Activation Functions

- **Identity Function or Linear Function**: returns simply the input value unchanged,

$$I(z) = z$$



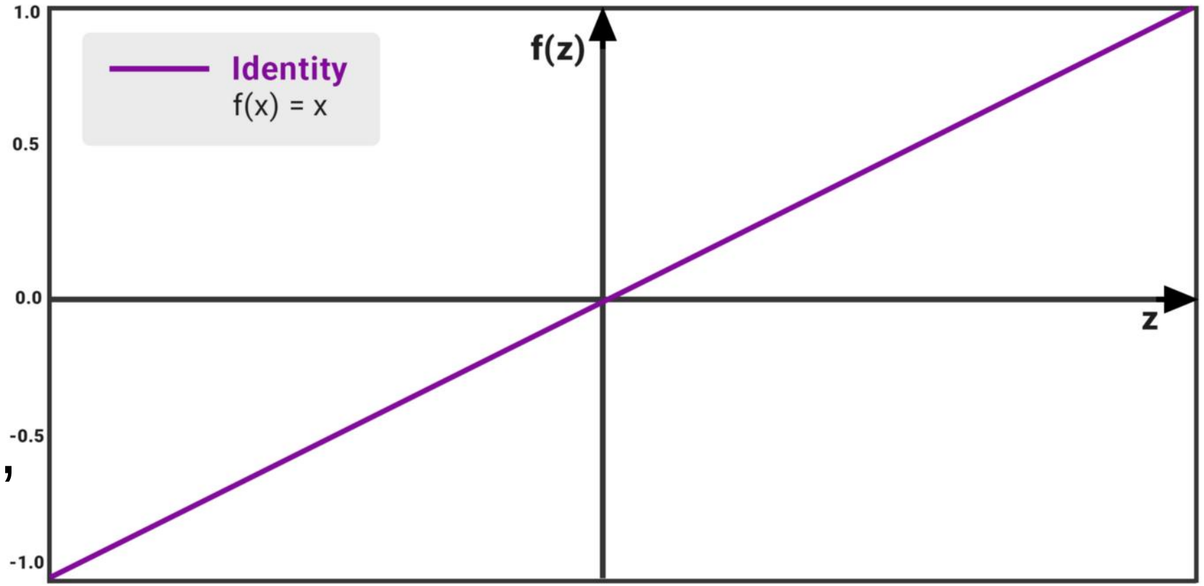
- Can learn simple Linear Regression task
- Used at output layer in Regression task

Activation Functions

- **Identity Function or Linear Function**: returns simply the input value unchanged,

$$I(z) = z$$

$$\text{Gradient, } I'(z) = 1, \quad \forall z$$



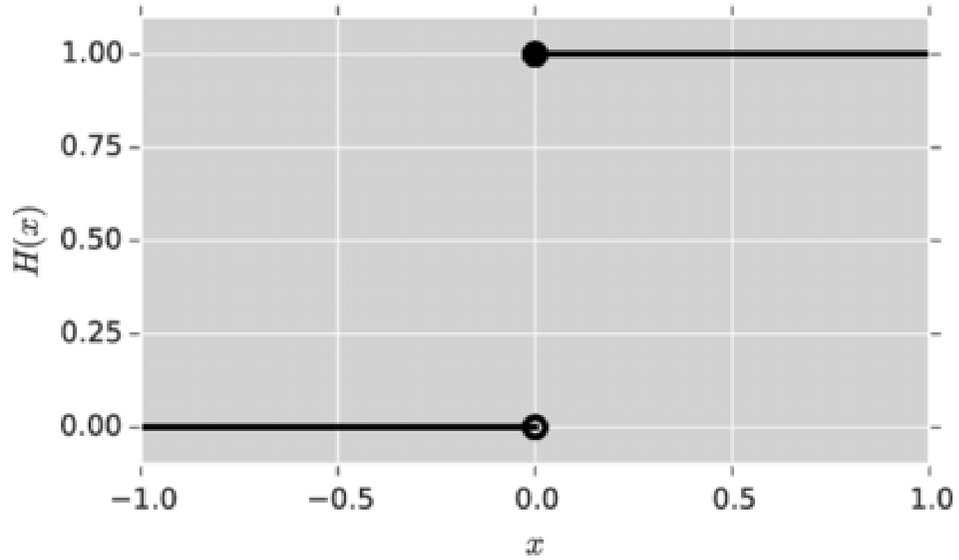
- Gradient based algorithms cannot be used, since gradient=1 everywhere
- Thus Not used in Hidden Layers

(The gradient/derivative of the function is the slope)

Activation Functions

- **Heaviside Step or Sign Activation**:

$$H(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

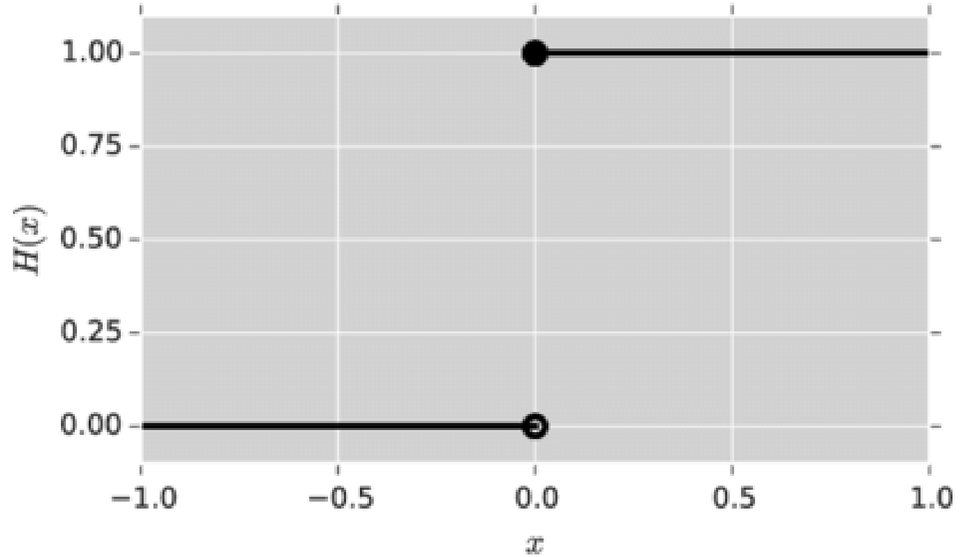


Activation Functions

- **Heaviside Step or Sign Activation**:

$$H(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

Gradient, $H'(z) = 0$, except at $z = 0$ where it is discontinuous and non-differentiable



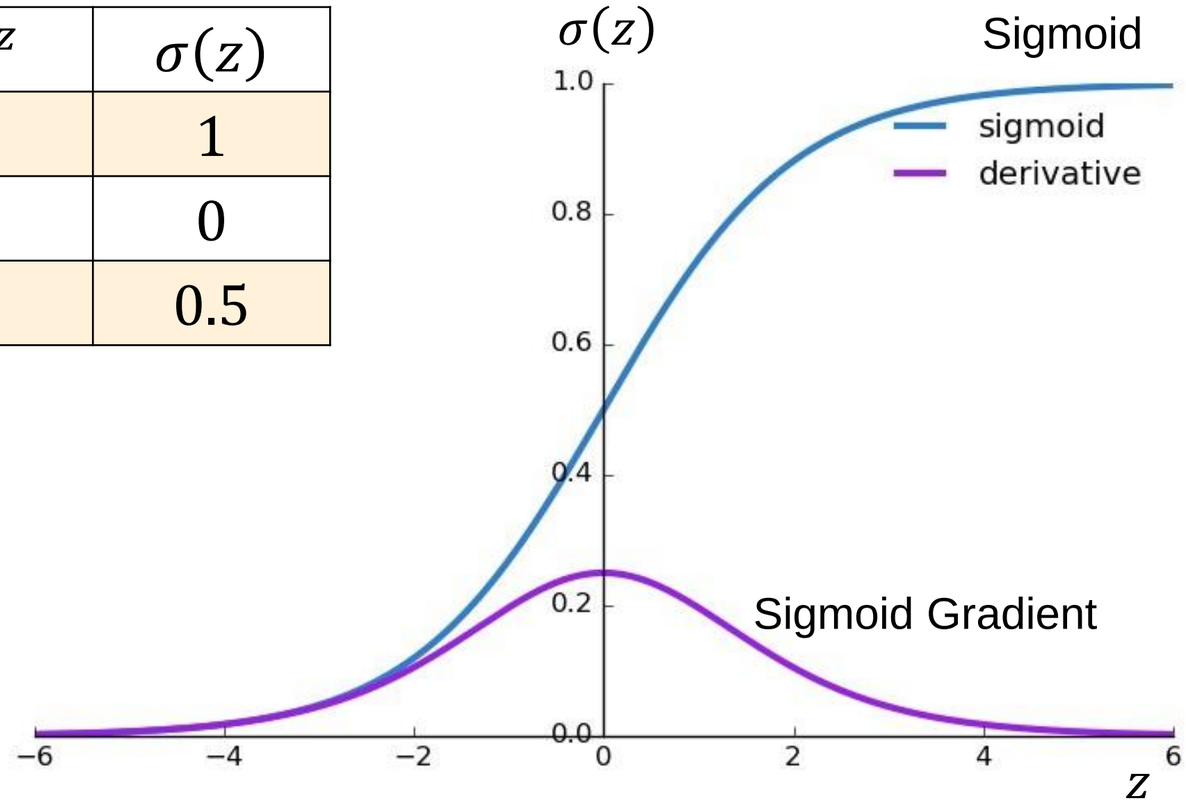
Because of the zero gradient and non-differentiability of this activation function, it is rarely used in the loss function even when it is used for prediction at testing time.

- **Sigmoid Activation Function or logistic function**

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

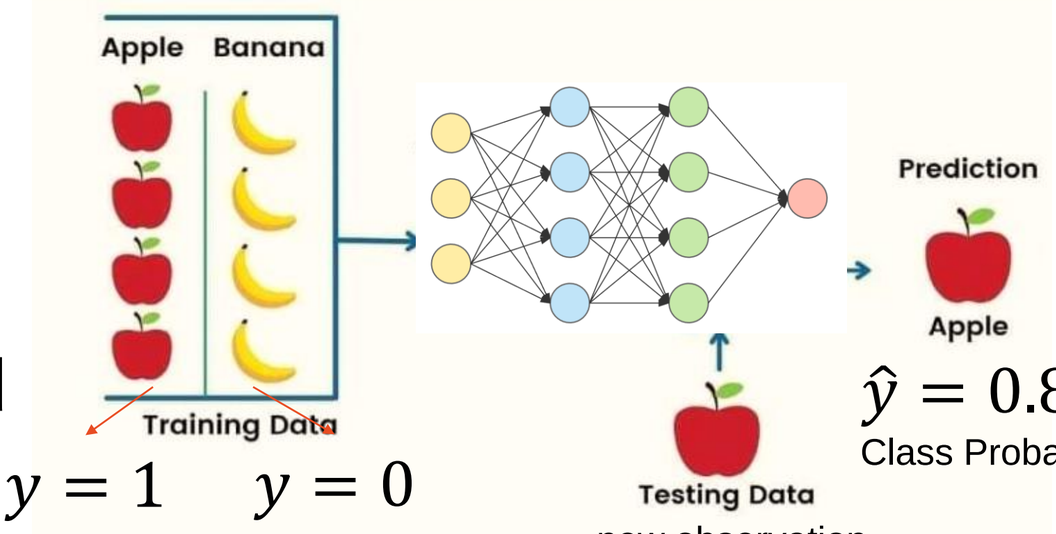
z	e^{-z}	$\sigma(z)$
$z \gg 0$	0	1
$z \ll 0$	∞	0
$z \approx 0$	1	0.5

- $\sigma(z)$ lies in range $[0,1]$



- **Sigmoid Activation Function or logistic function**

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



- Good when output is to be interpreted as probability
- Used at output layer of Binary Classification task
- May be used at hidden layers in shallow networks

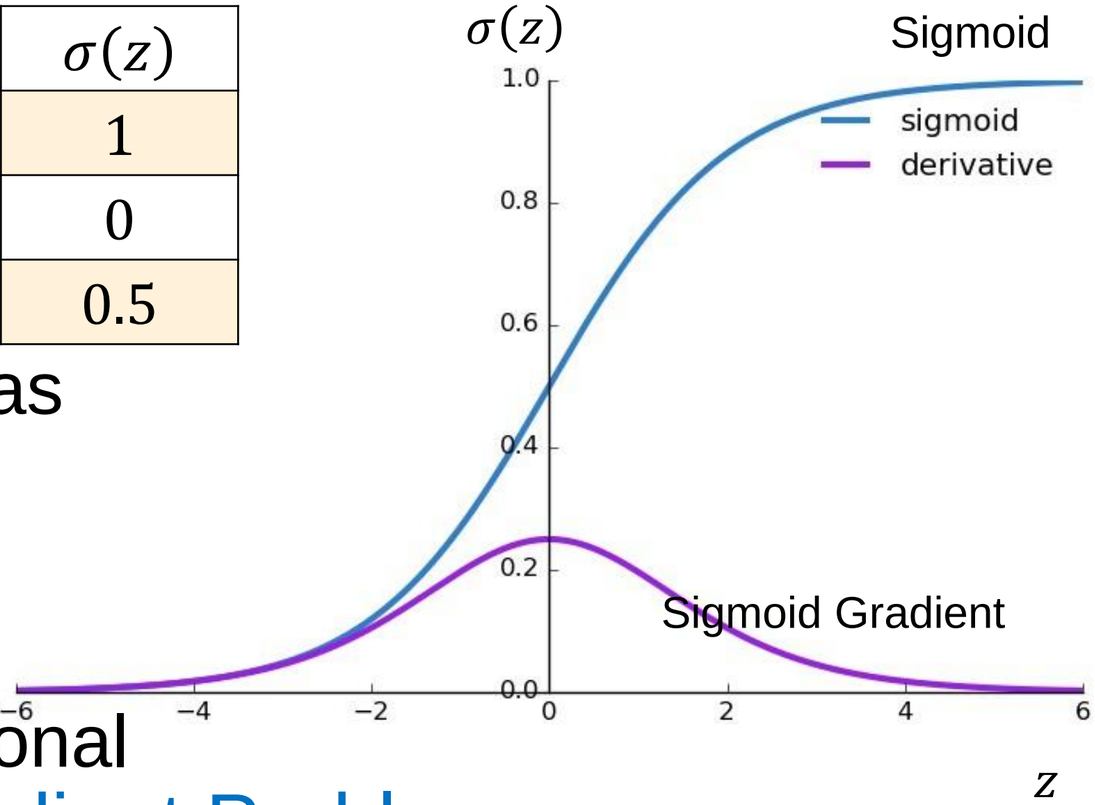
$\sigma(z) \Rightarrow P(y = 1|X)$ =>Class Probability

Class Probability indicates higher likelihood of new observation belonging to class 1

- **Sigmoid Activation Function or logistic function**

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

z	e^{-z}	$\sigma(z)$
$z \gg 0$	0	1
$z \ll 0$	∞	0
$z \approx 0$	1	0.5



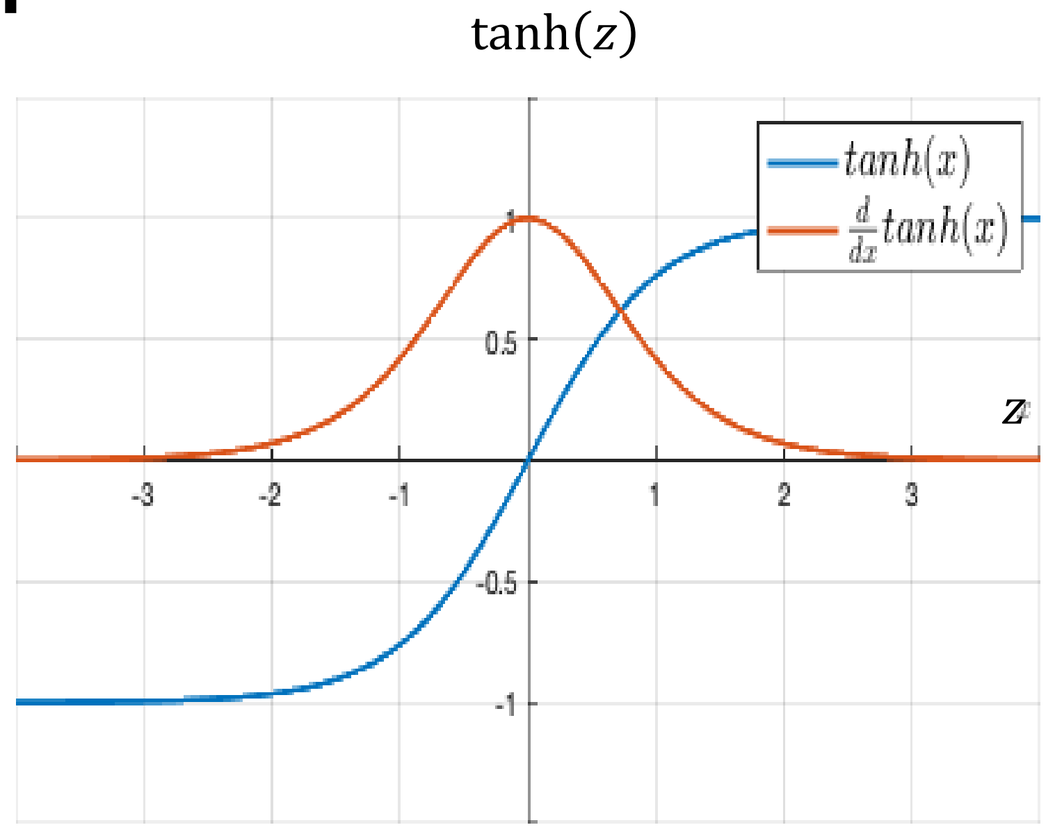
- Good when output is to be interpreted as probability
- Gradient: $\boldsymbol{\sigma'(z)} = \sigma(z)\big(1 - \sigma(z)\big)$
- When z<<0 or z>>0, weights may not update well as weights update proportional to $\boldsymbol{\sigma'(z)}$, and $\sigma(z) \to 0$: Vanishing Gradient Problem

- **Tanh (Hyperbolic Tangent) Activation**

$$f(z) = \tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$$

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

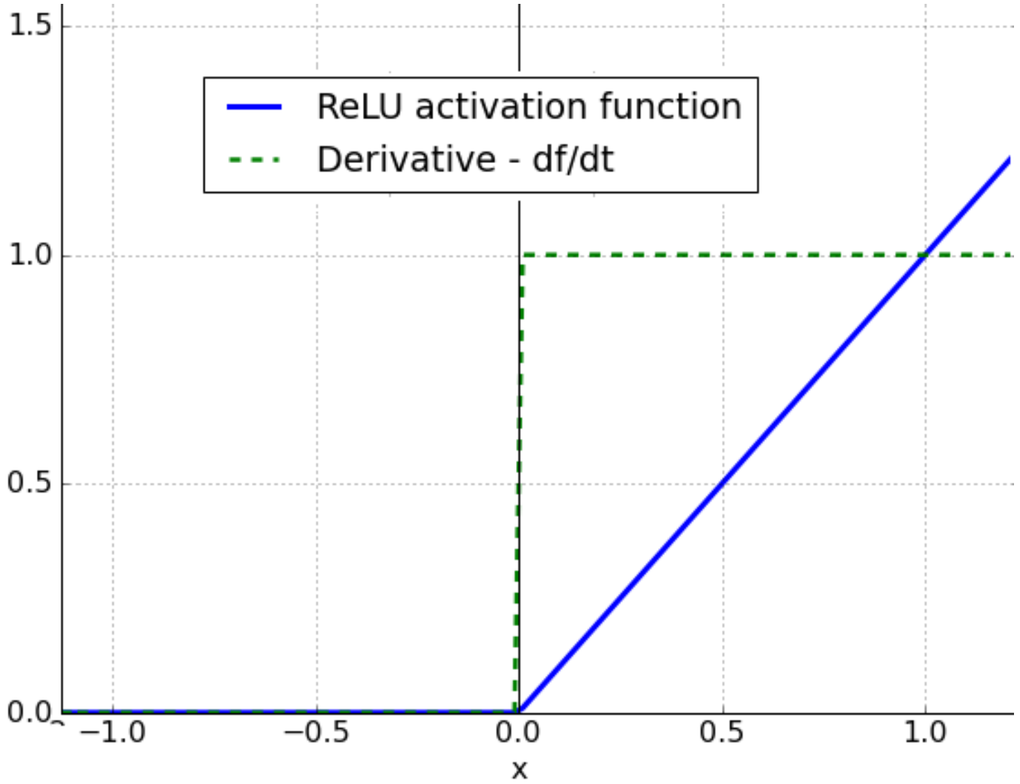
z	e^{2z}	$\tanh(z)$
$z \gg 0$	∞	1
$z \ll 0$	0	-1
$z \approx 0$	1	0



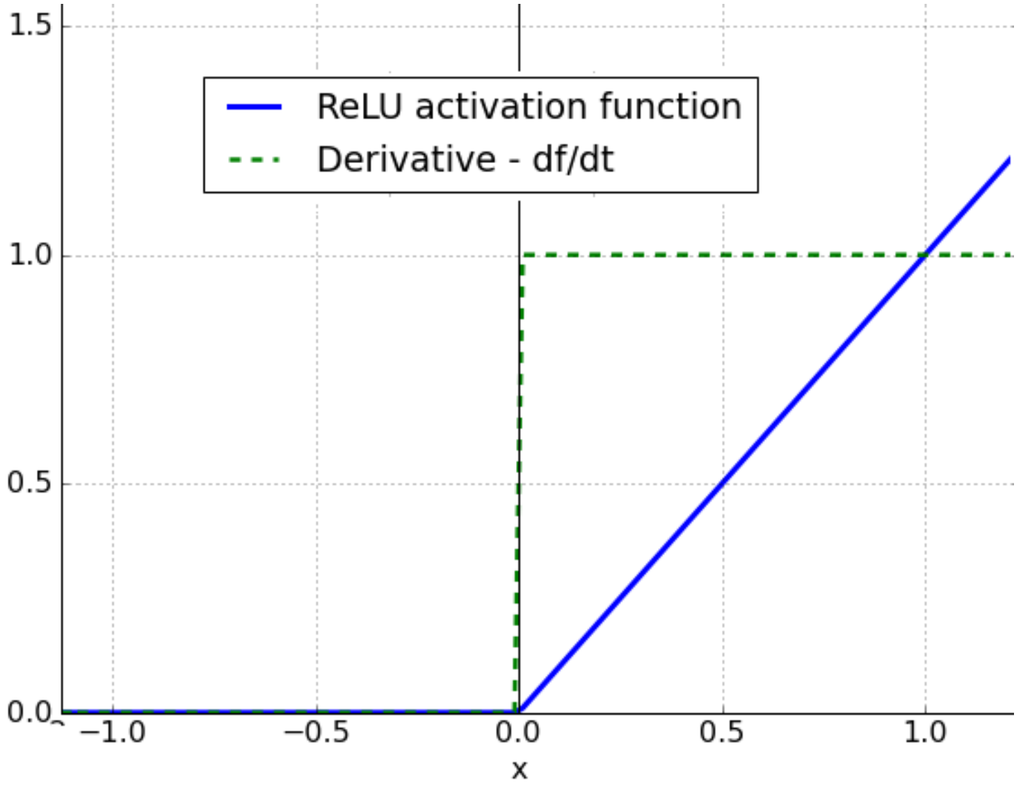
- Gradient: $f'(z) = 1 - f(z)^2$
- Similar issues as Sigmoid Activation

- **ReLU (Rectified Linear Unit) Activation Function**
- $f(z) = \max(0, z) = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases}$
  - piecewise linear function
  - sigmoid and tanh activation functions cannot be used in networks with many layers due to the vanishing gradient problem even when z>>0.

    ReLU overcomes this problem

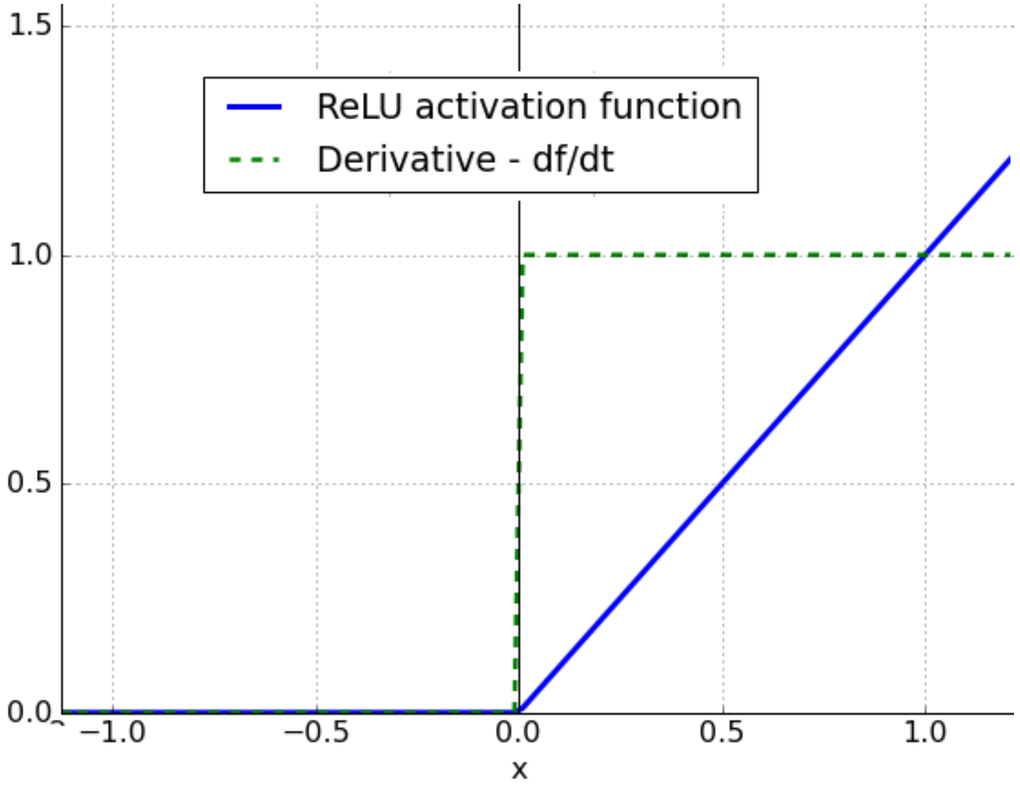


- **ReLU (Rectified Linear Unit) Activation Function**
- $f(z) = \max(0, z) = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases}$
  - Another Advantage: creates sparse representation
- As compared to other activation function it does not activate all the neurons at same time
- The neurons will only be deactivated if the output of linear transformation is < 0.
- Since only a certain no. of neurons are activated, the ReLU is far more computationally efficient compared to sigmoid & tanh function



- **ReLU (Rectified Linear Unit) Activation Function**

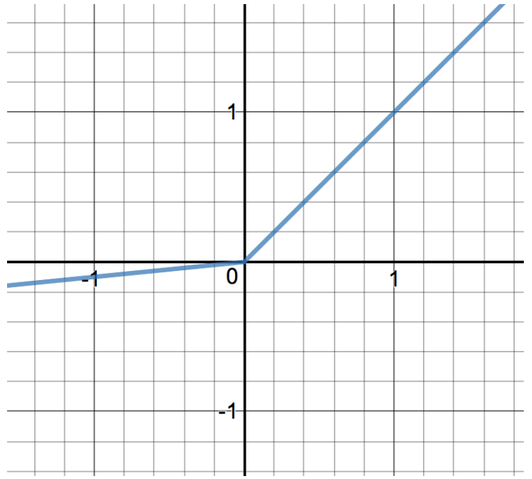
- $f(z) = \max(0, z) = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases}$
  - Gradient: $f'(z) = \begin{cases} 1 & z > 0 \\ 0, & z < 0 \end{cases}$ (dying ReLu problem)



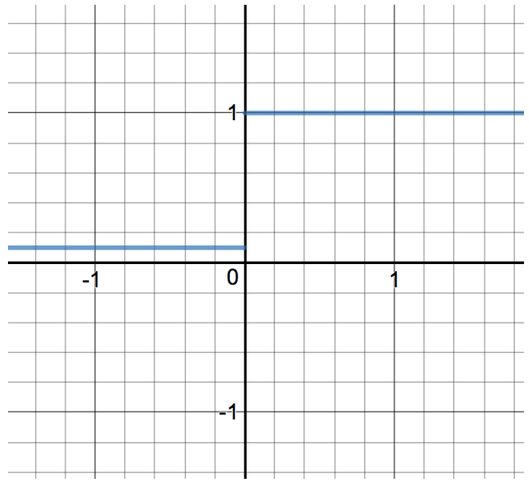
- At negative side of graph, the gradient value is 0. Thus during the backpropogation process, the weights and biases for some neurons are not updated.
- This may create dead neurons which never get activated.
- This is taken care of by the *'Leaky' ReLU function*.

- **LeakyReLU**
- $f(z) = \begin{cases} z, & z > 0 \\ \alpha z, & z \leq 0 \end{cases}$
  - Normally, $\alpha = 0.01$
  - attempts to fix the “dying ReLU” problem, effectiveness unclear
  - Gradient: $f'(z) = \begin{cases} 1 & z > 0 \\ \alpha, & z < 0 \end{cases}$

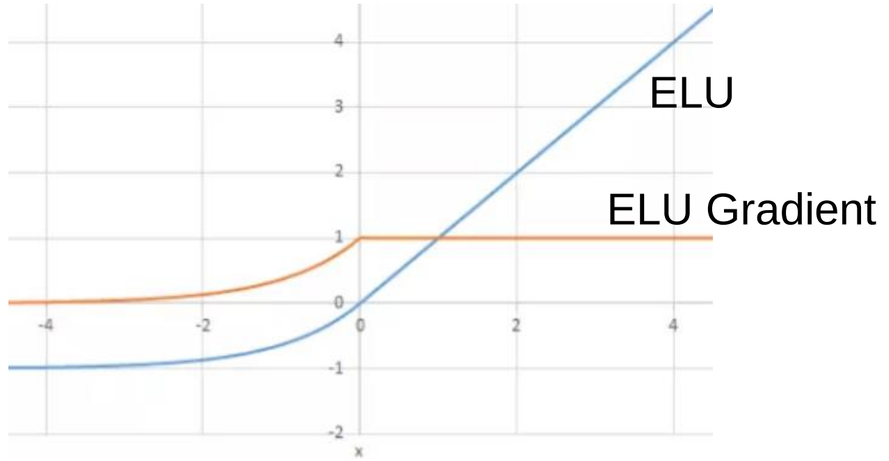
$f(z)$



$f'(z)$



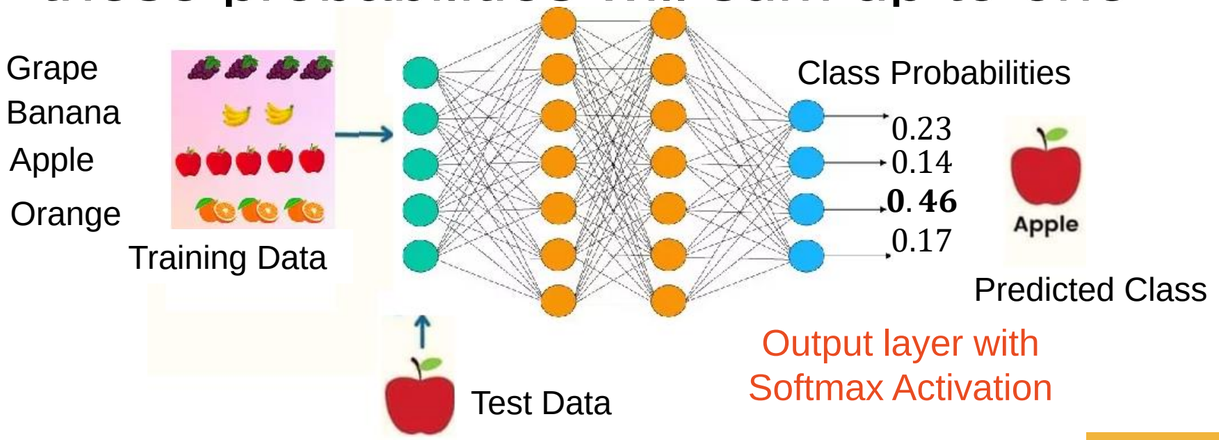
- **Exponential Linear unit (ELU)**
- $g(z) = \begin{cases} z, & z > 0 \\ \alpha(e^z - 1), & z \leq 0 \end{cases}$



- **Softmax Activation Function**

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}}, j = 1,2, \dots K$$

  - Used as the activation function in the output layer of NN (multi-class classification)
  - outputs the probability for each class and these probabilities will sum up to one



$$\begin{bmatrix} P(Grape) \\ P(Banana) \\ P(Apple) \\ P(Orange) \end{bmatrix} = \sigma\left(\begin{bmatrix} 0.8 \\ 0.3 \\ 1.5 \\ 0.5 \end{bmatrix}\right)$$

Output scores

$$= \begin{bmatrix} \frac{e^{0.8}}{e^{0.8}+e^{0.3}+e^{1.5}+e^{0.5}} \\ \frac{e^{0.3}}{e^{0.8}+e^{0.3}+e^{1.5}+e^{0.5}} \\ \frac{e^{1.5}}{e^{0.8}+e^{0.3}+e^{1.5}+e^{0.5}} \\ \frac{e^{0.5}}{e^{0.8}+e^{0.3}+e^{1.5}+e^{0.5}} \end{bmatrix} = \begin{bmatrix} 0.23 \\ 0.14 \\ \mathbf{0.46} \\ 0.17 \end{bmatrix}$$

Class Probabilities

New observation is likely to be an Apple

Where which activation function?

- Hidden units of DNNs : ReLU, ELU, tanh
- Output unit of Classifiers:
  - Sigmoid for Binary Classification
  - Softmax for Multi-class Classification
- Output unit of Regression NN: Linear Activation function

Module I: Introduction to Generative AI and Fundamental Study

Module I (20%)

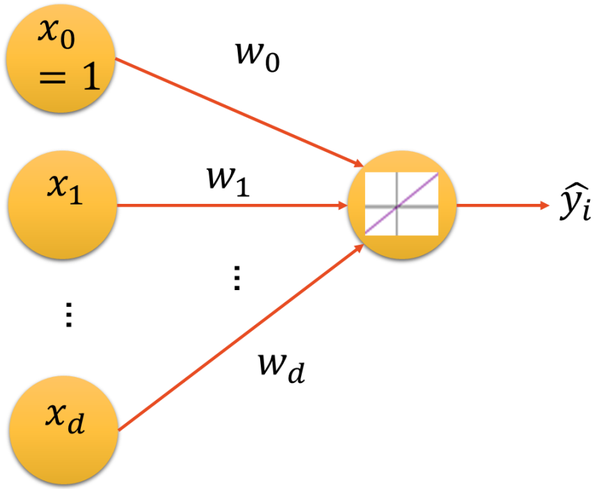
Basic introduction to Generative AI, Applications of Generative AI in various fields, Perceptron and Multilayer Perceptron (MLP), Gradient Descent, Backpropagation algorithm, Loss functions, Understanding Bias and Variance

Weight Update Rule

$$W \leftarrow W - \eta \nabla_W \mathrm{E}$$

New Previous

Rate of Convergence Factor/ Learning Rate



- Weights should be such that they MINIMIZE Error in prediction

$$\mathrm{E} = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Mean Squared Error or Quadratic Loss Function

N : number of observations in data

$\hat{y}_i$: predicted output

y_i : true output

Weight Update Rule

$$W \leftarrow W - \eta \nabla_W \mathrm{E}$$

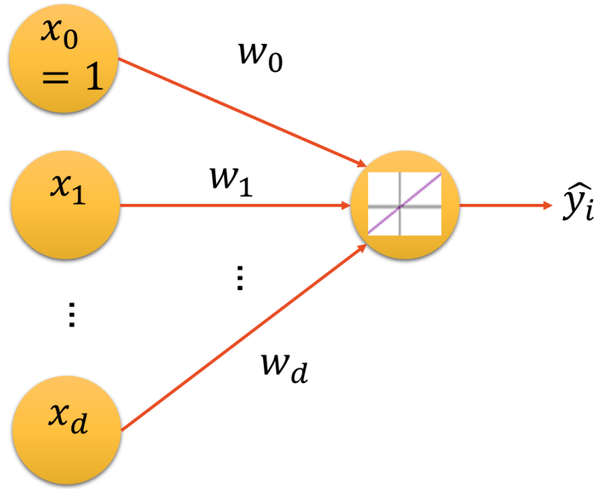
New Previous

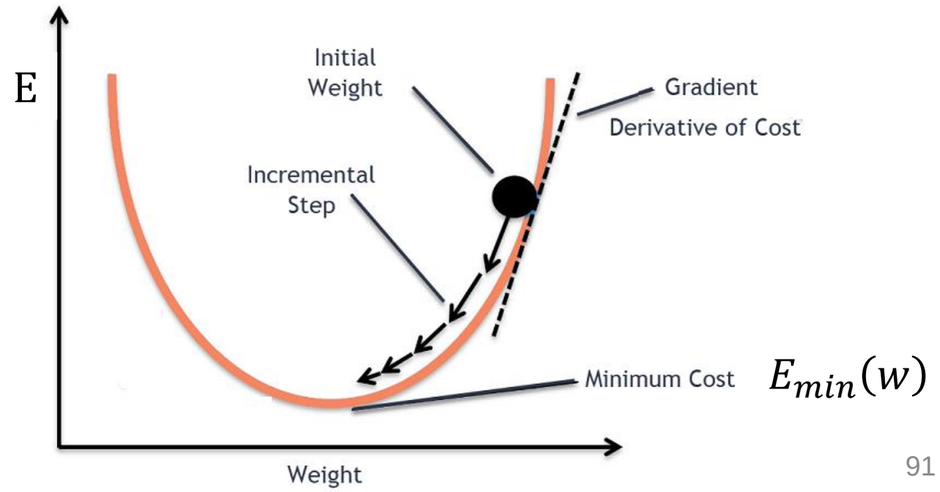
Rate of Convergence Factor/ Learning Rate

- Weights should be such that they MINIMIZE Error in prediction

$$\mathrm{E} = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

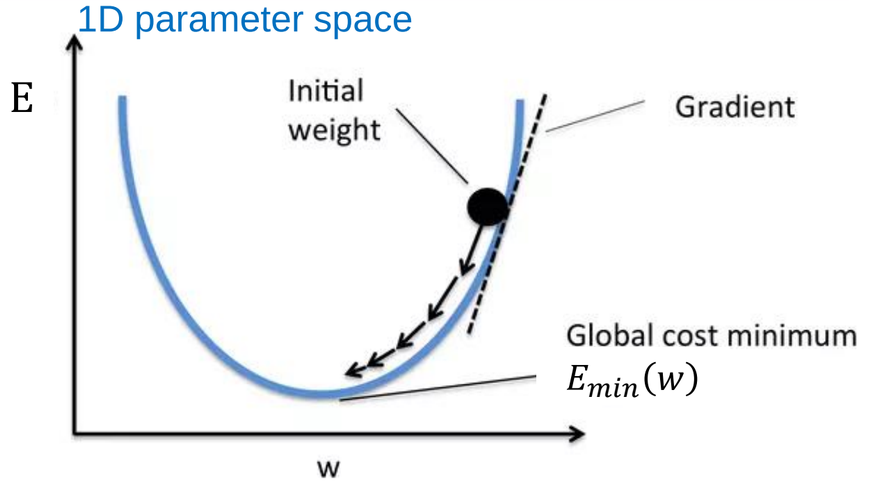
- Gradient of Error w.r.t. weight $(\nabla_W \mathrm{E})$ decreases towards minima of the function

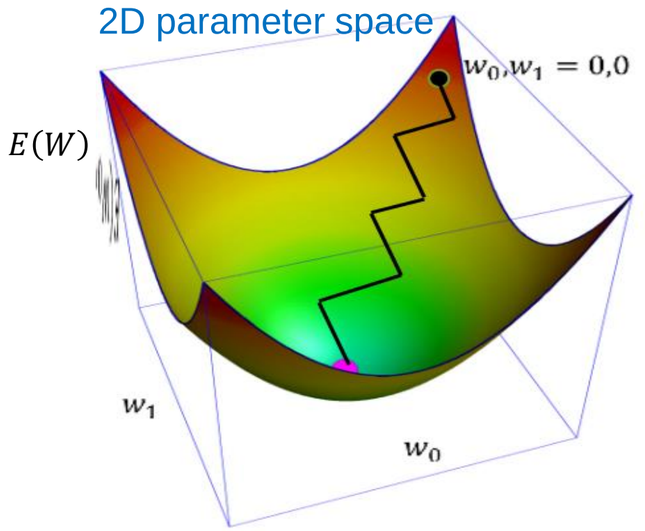




Gradient Descent

1. Randomly Initialize Weights, W
2. Evaluate Gradient of Error w.r.t. weight $\nabla_W \mathrm{E}$
3. Update the weights:
   $W \leftarrow W - \eta \nabla_W \mathrm{E}$
4. Repeat 2 and 3 until convergence





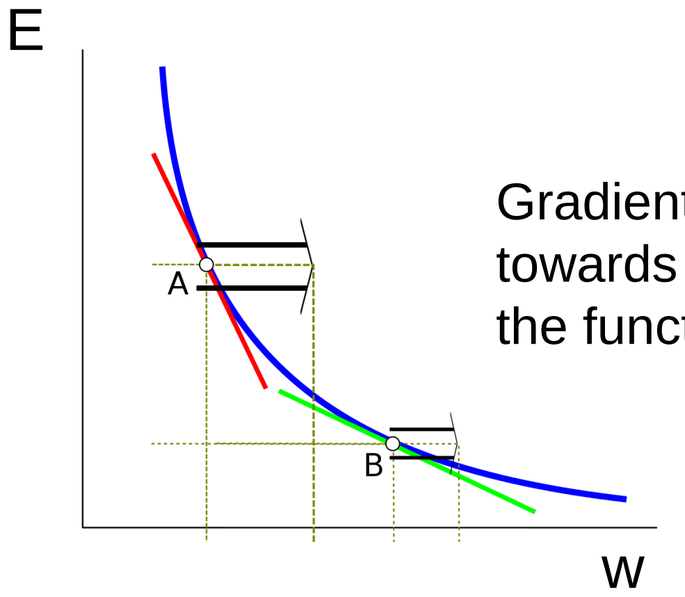
$$W \leftarrow W - \eta \nabla_W \mathrm{E}$$

- Say, $\nabla_W \mathrm{E} > 0$: (B -> A)
  - W is updated as

  $W \leftarrow W - \eta |\nabla_W \mathrm{E}|$ (decrease)

- Say, $\nabla_W \mathrm{E} < 0$: (A -> B)
  - W is updated as

  $W \leftarrow W + \eta |\nabla_W \mathrm{E}|$(increased)



Gradient decreases towards minima of the function

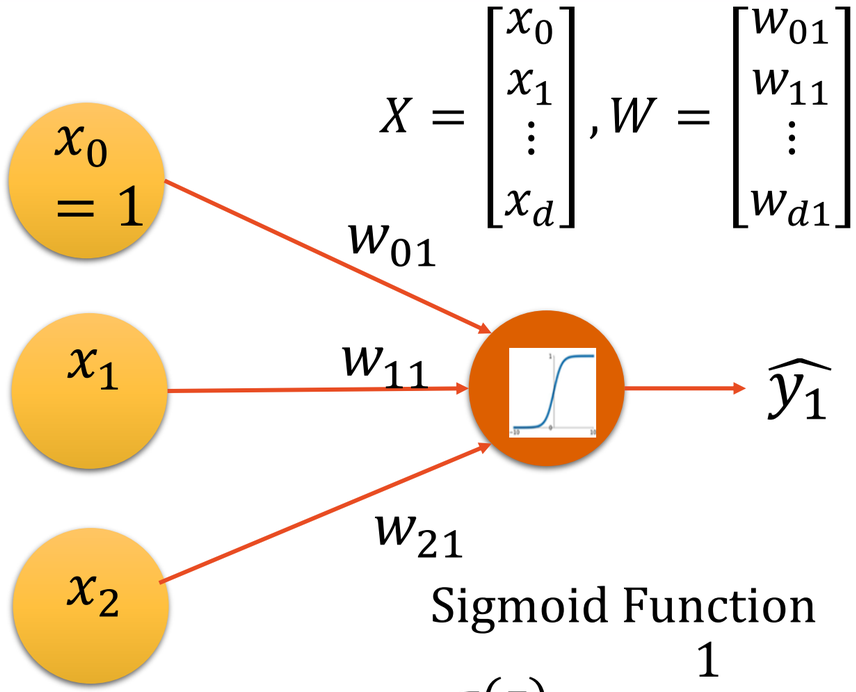
$W \leftarrow W - \eta \nabla_W \mathrm{E}$ updates weights in the direction of decreasing gradient

Single layer, Single output NN

- Heaviside Step activation is non-differentiable
- Consider Single layer, Single output NN, with sigmoid activation function

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1$ for single o/p
- Error in prediction (Mean Square Error Loss)

$\mathrm{E} = \frac{1}{2}(\hat{y}_j - y_j)^2$ for one observation $(X \quad , y_j)$



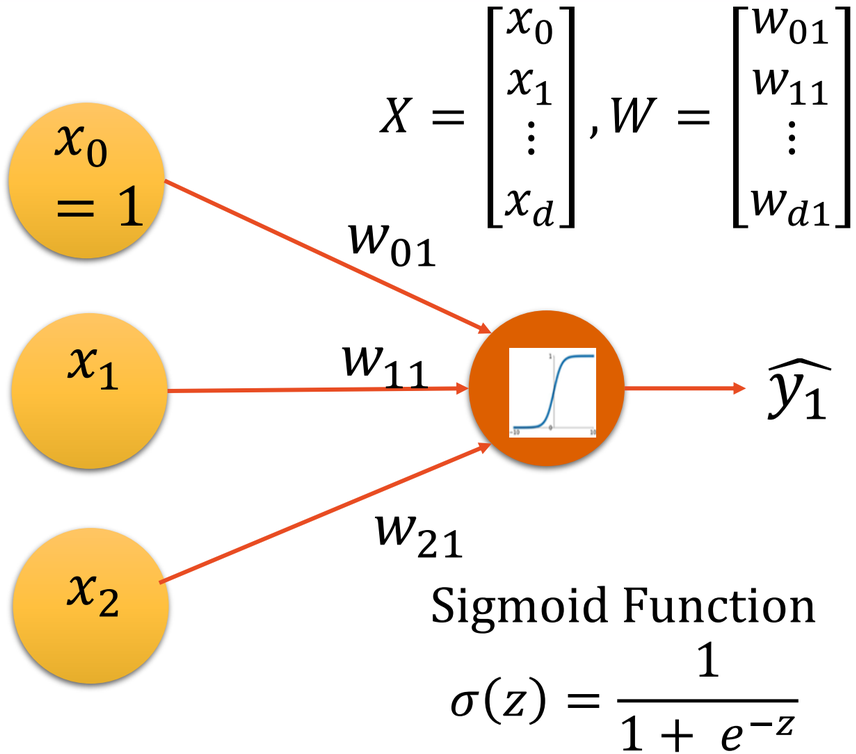
w_{ij}^k = Weight of connection b/w i^{th} node in $(k-1)^{th}$ layer to j^{th} node in k^{th} layer

Single layer, Single output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1$
- Error in prediction

$$\mathrm{E} = \frac{1}{2}\left(\hat{y}_j - y_j\right)^2$$

- Gradient of Error/Loss function
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial \theta_j} \frac{\partial \theta_j}{\partial w_{ij}}$ (Chain Rule)
- $\frac{\partial E}{\partial w_{ij}} =$

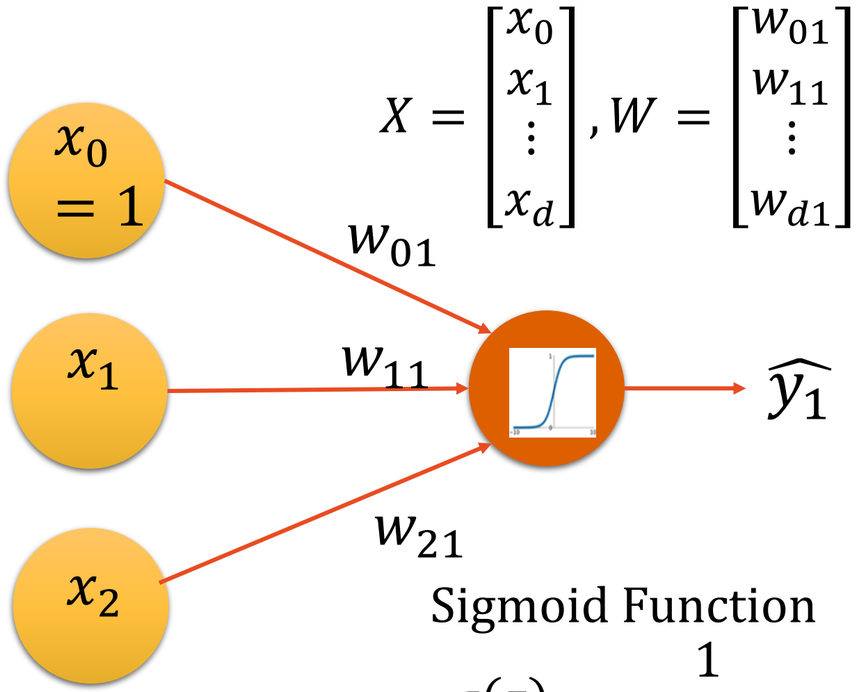


Single layer, Single output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1$
- Error in prediction

$$\mathrm{E} = \frac{1}{2}\left(\hat{y}_j - y_j\right)^2$$

- Gradient of Error/Loss function
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$ (Chain Rule)
- $\frac{\partial E}{\partial w_{ij}} = \left(\hat{y}_j - y_j\right)\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$



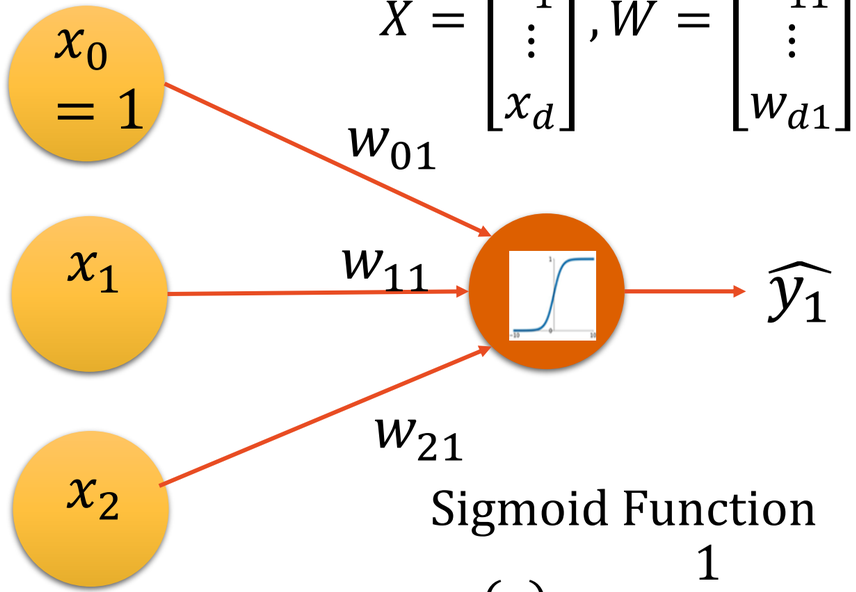
Single layer, Single output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1$
- Error in prediction

$$\mathrm{E} = \frac{1}{2}\left(\hat{y}_j - y_j\right)^2$$

- Gradient of Error/Loss function
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$ (Chain Rule)
- $\frac{\partial E}{\partial w_{ij}} = \left(\hat{y}_j - y_j\right)\hat{y}_j\left(1 - \hat{y}_j\right)\frac{\partial \theta_j}{\partial w_{ij}}$

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}, W = \begin{bmatrix} w_{01} \\ w_{11} \\ \vdots \\ w_{d1} \end{bmatrix}$$



Sigmoid Function

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

$$\sigma'(z) = \sigma(z)\left(1 - \sigma(z)\right)$$

$$\frac{\partial \hat{y}_j}{\partial \theta_j} = \sigma(\theta_j)\left(1 - \sigma(\theta_j)\right)$$

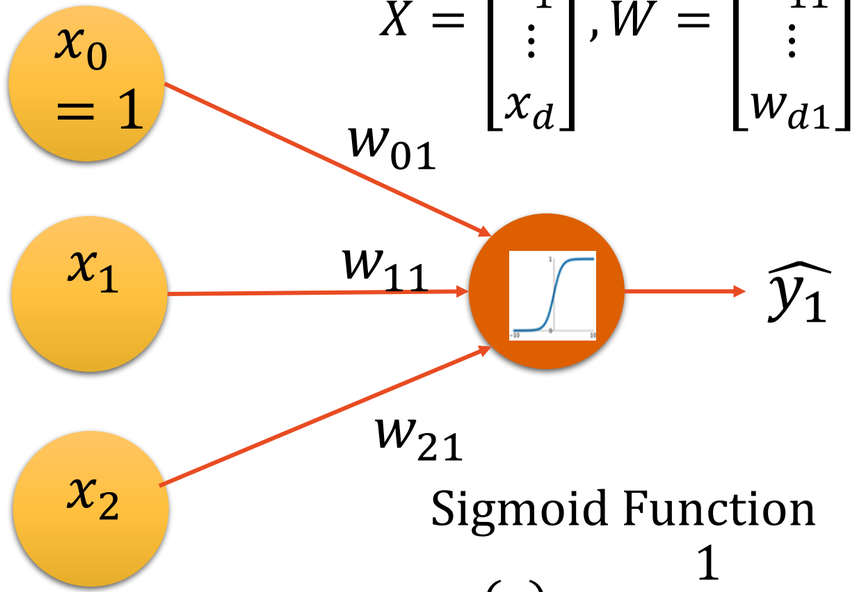
Single layer, Single output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1$
- Error in prediction

$$\mathrm{E} = \frac{1}{2}\left(\hat{y}_j - y_j\right)^2$$

- Gradient of Error/Loss function
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$ (Chain Rule)
- $\frac{\partial E}{\partial w_{ij}} = \left(\hat{y}_j - y_j\right)\hat{y}_j\left(1 - \hat{y}_j\right)x_i$

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}, W = \begin{bmatrix} w_{01} \\ w_{11} \\ \vdots \\ w_{d1} \end{bmatrix}$$



Sigmoid Function

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

$$\theta_j = w_{0j}x_0 + \cdots + w_{ij}x_i + \cdots w_{dj}x_d$$

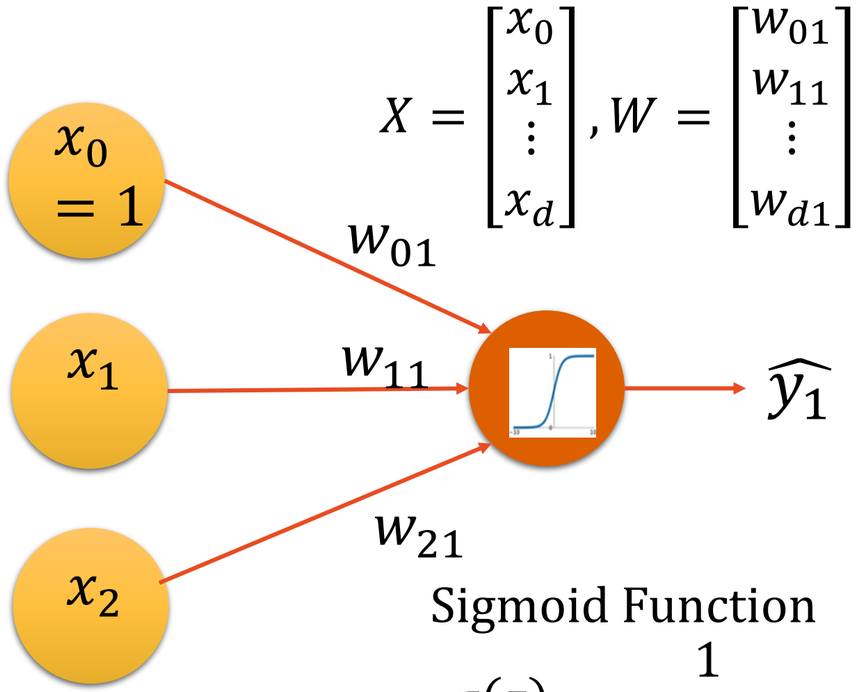
$$\frac{\partial \theta_j}{\partial w_{ij}} = x_i$$

Single layer, Single output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1$
- Error in prediction

$$\mathrm{E} = \frac{1}{2}\left(\hat{y}_j - y_j\right)^2$$

- Gradient of Error/Loss function
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$ (Chain Rule)
- $\frac{\partial E}{\partial w_{ij}} = \left(\hat{y}_j - y_j\right)\hat{y}_j\left(1 - \hat{y}_j\right)x_i$
- Weight Updation: $w_{ij} \leftarrow w_{ij} - \eta\left(\hat{y}_j - y_j\right)\hat{y}_j\left(1 - \hat{y}_j\right)x_i$



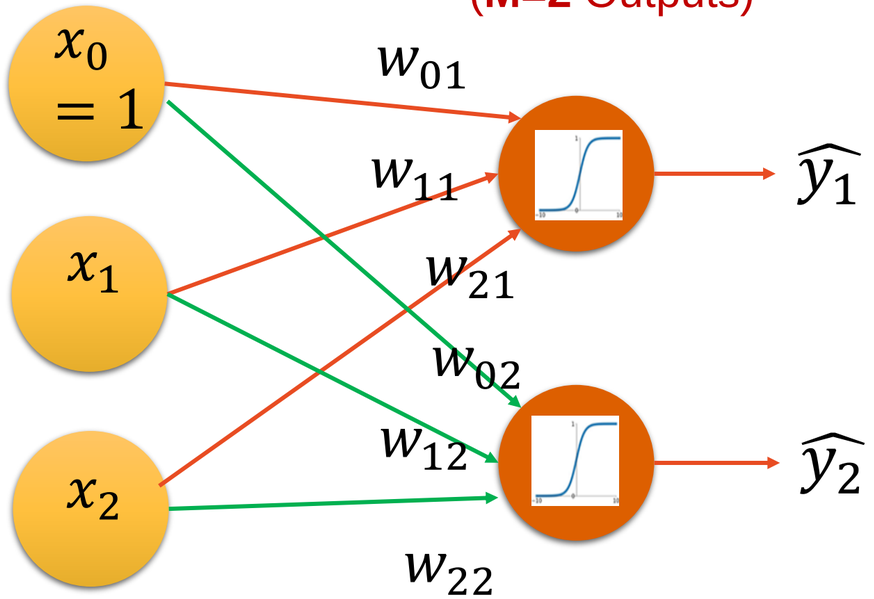
Weight updated Simultaneously

Single layer, Multiple output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y_j} = \sigma(\theta_j),\ j = 1, \ldots, M$
- Error in prediction (sum over all classes)

$$\mathrm{E} = \frac{1}{2}\sum_{j=1}^{M}(\hat{y_j} - y_j)^2$$

Single Layer
(**M=2** Outputs)



w_{ij}^{k} = Weight of connection b/w i^{th} node in $(k-1)^{th}$ layer to j^{th} node in k^{th} layer

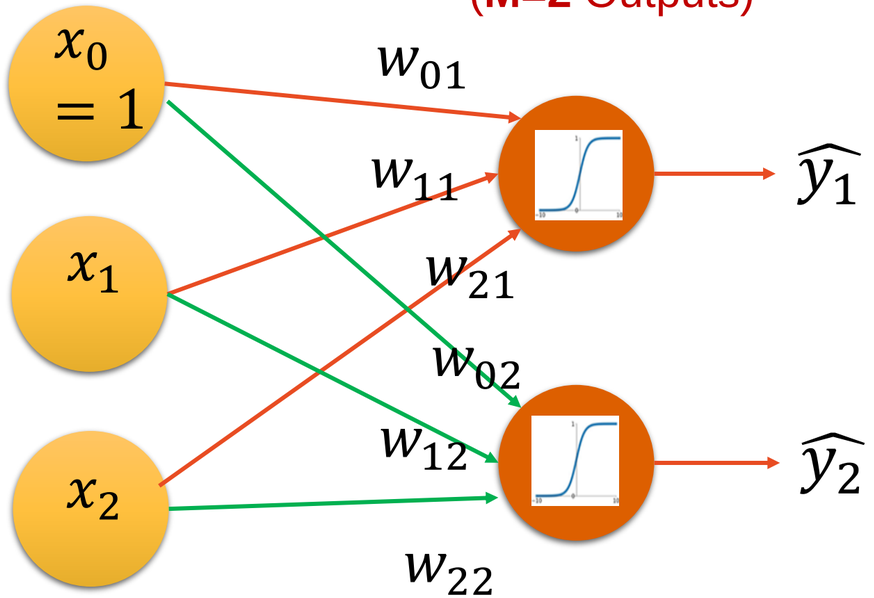
Single layer, Multiple output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1, \ldots, M$
- Error in prediction (sum over all classes)

$$\mathrm{E} = \frac{1}{2}\sum_{j=1}^{M}\left(\hat{y}_j - y_j\right)^2$$

- Gradient of Error/Loss function
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$ (Chain Rule)
- $\frac{\partial E}{\partial w_{ij}} =$

Single Layer
(**M=2** Outputs)



w_{ij} = Weight of connection b/w i^{th} node in $previous$ layer to j^{th} node in $next$ layer

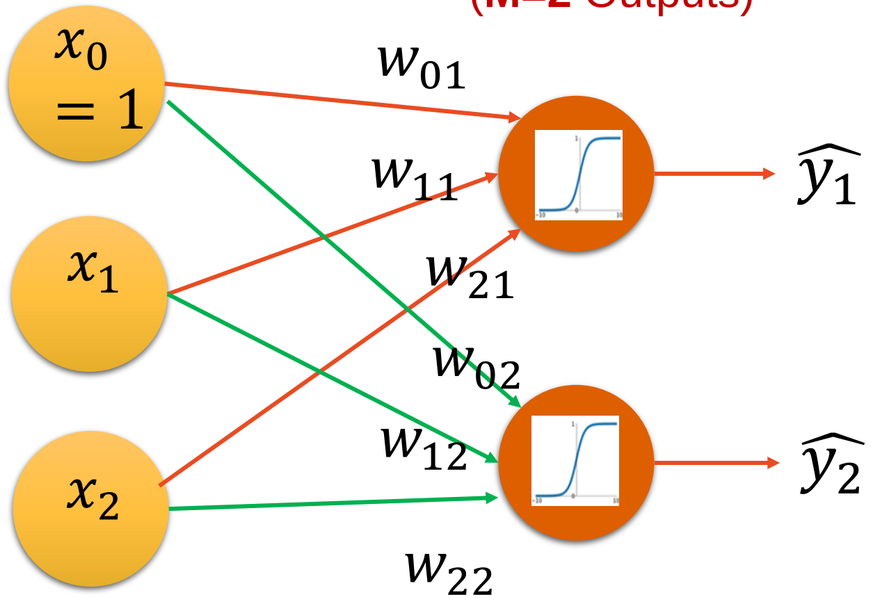
Single layer, Multiple output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1, \ldots, M$
- Error in prediction (sum over all classes)

$$\mathrm{E} = \frac{1}{2}\sum_{j=1}^{M}\left(\hat{y}_j - y_j\right)^2$$

- Gradient of Error/Loss function
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$ (Chain Rule)
- $\frac{\partial E}{\partial w_{ij}} = \left(\hat{y}_j - y_j\right)\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$

Single Layer
(**M=2** Outputs)



For, Sigmoid Function, the Gradient is

$$\sigma'(z) = \sigma(z)\left(1 - \sigma(z)\right)$$

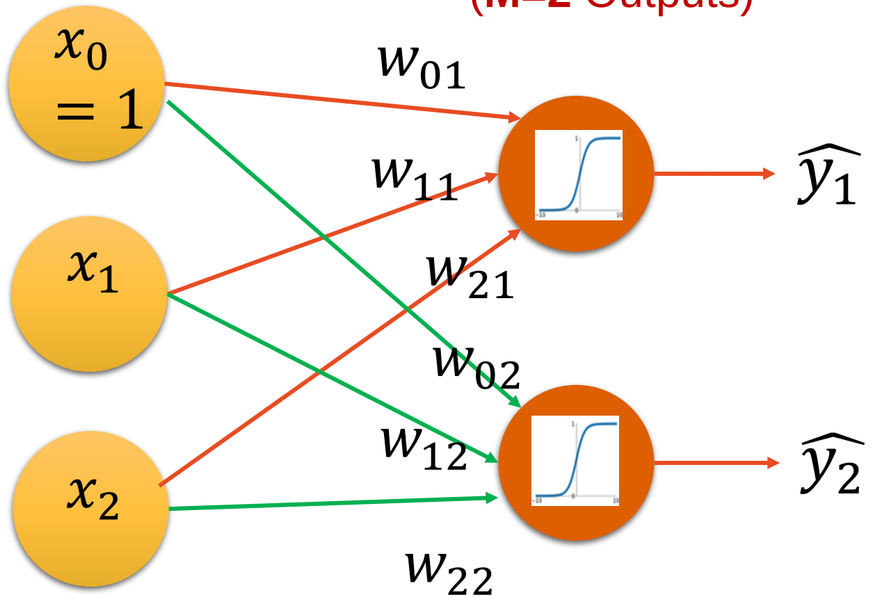
Single layer, Multiple output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1, \ldots, M$
- Error in prediction (sum over all classes)

$$\mathrm{E} = \frac{1}{2}\sum_{j=1}^{M}(\hat{y}_j - y_j)^2$$

- Gradient of Error/Loss function
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$ (Chain Rule)
- $\frac{\partial E}{\partial w_{ij}} = (\hat{y}_j - y_j)\hat{y}_j(1 - \hat{y}_j)\frac{\partial \theta_j}{\partial w_{ij}}$

Single Layer
(**M=2** Outputs)



$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

$$\frac{\partial \hat{y}_j}{\partial \theta_j} = \sigma(\theta_j)\left(1 - \sigma(\theta_j)\right)$$

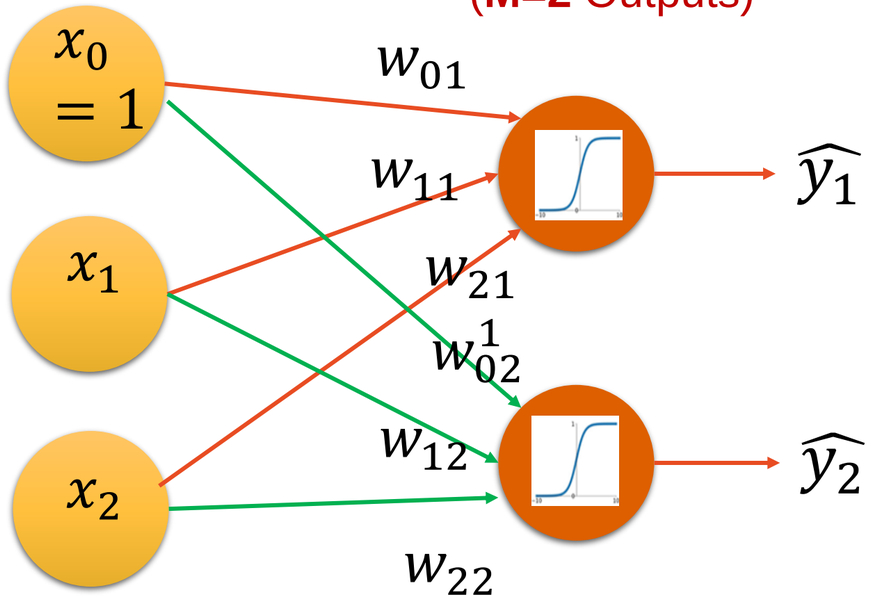
Single layer, Multiple output NN

- Pre-activation, $\theta_j = \sum_{i=0}^{d} w_{ij} x_i$
- Output activations, $\hat{y}_j = \sigma(\theta_j),\ j = 1, \ldots, M$
- Error in prediction (sum over all classes)

$$\mathrm{E} = \frac{1}{2}\sum_{j=1}^{M}(\hat{y}_j - y_j)^2$$

- Gradient of Error/Loss function
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \theta_j}\frac{\partial \theta_j}{\partial w_{ij}}$ (Chain Rule)
- $\frac{\partial E}{\partial w_{ij}} = (\hat{y}_j - y_j)\hat{y}_j(1 - \hat{y}_j)x_i$
- Weight Updation: $w_{ij} \leftarrow w_{ij} - \eta\,(\hat{y}_j - y_j)\hat{y}_j(1 - \hat{y}_j)\,x_i$

Single Layer
(**M=2** Outputs)



Weight updated Simultaneously

Types of Gradient Descent

- Batch Gradient Descent (BGD)
- Stochastic Gradient Descent (SGD)
- Mini Batch Gradient Descent

Types of Gradient Descent

- Batch Gradient Descent (BGD): $W \leftarrow W - \eta \sum_{i=1}^{N} (\hat{y}_i - y_i)\, \hat{y}_j \left(1 - \hat{y}_j\right) X_i$

It uses the *entire dataset* at every step, making it slow for large datasets. However, it is computationally efficient, since it produces a *stable* error gradient and a stable convergence

- Stochastic Gradient Descent (SGD)

- Mini Batch Gradient Descent

Types of Gradient Descent

- Batch Gradient Descent (BGD): $W \leftarrow W - \eta \sum_{i=1}^{N} (\hat{y}_i - y_i)\, \hat{y}_j \left(1 - \hat{y}_j\right) X_i$

It uses the *entire dataset* at every step, making it slow for large datasets. However, it is computationally efficient, since it produces a *stable* error gradient and a stable convergence

- Stochastic Gradient Descent (SGD): $W \leftarrow W - \eta (\hat{y}_i - y_i) \hat{y}_j \left(1 - \hat{y}_j\right) X_i$

It is on the other extreme of the idea, using a *single example* (batch of 1) for each learning step. Much faster, may return *noisy gradients* which can cause the error rate to jump around

- Mini Batch Gradient Descent

Types of Gradient Descent

- Batch Gradient Descent (BGD): $W \leftarrow W - \eta \sum_{i=1}^{N} (\hat{y}_i - y_i)\, \hat{y}_j \left(1 - \hat{y}_j\right) X_i$

It uses the *entire dataset* at every step, making it slow for large datasets. However, it is computationally efficient, since it produces a *stable* error gradient and a stable convergence

- Stochastic Gradient Descent (SGD): $W \leftarrow W - \eta (\hat{y}_i - y_i) \hat{y}_j \left(1 - \hat{y}_j\right) X_i$

It is on the other extreme of the idea, using a *single example* (batch of 1) per each learning step. Much faster, may return *noisy gradients* which can cause the error rate to jump around

- Mini Batch Gradient Descent: $W \leftarrow W - \eta \sum_{X_i \in Minibatch} (\hat{y}_i - y_i)\, \hat{y}_j \left(1 - \hat{y}_j\right) X_i$
  Computes the gradients on small random sets of instances called *mini batches*. Reduce noise from SGD and still more efficient than BGD

One epoch for SGD:

1. Take an example
2. Feed it to Neural Network and determine the loss
3. Calculate it's gradient
4. Use the gradient calculated in step 3 to update the weights
5. Repeat steps 1–4 for all the examples in training dataset

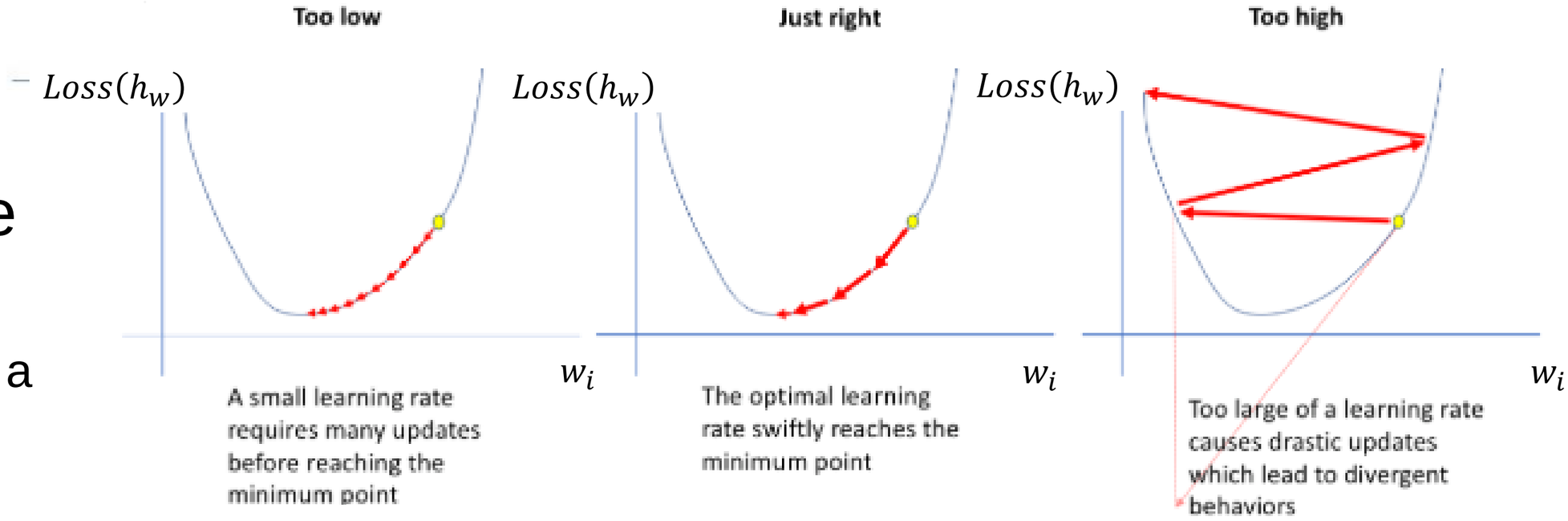
One epoch for Mini Batch GD **:**

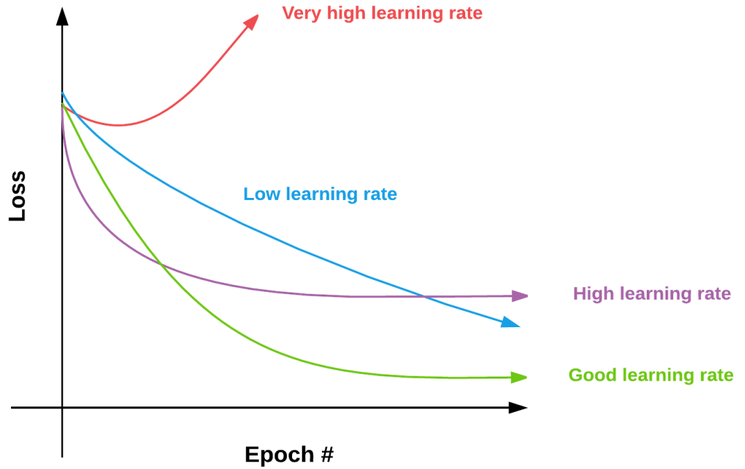
1. Pick a mini-batch
2. Feed it to Neural Network and determine the loss
3. Calculate mean gradient of the mini-batch
4. Use the mean gradient calculated in step 3 to update the weights
5. Repeat steps 1–4 for all the mini-batches

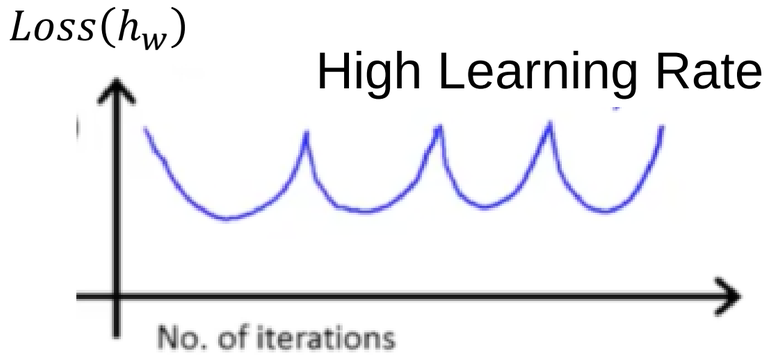
For BGD, consider entire data instead of mini-batches

Learning Rate

- η= Learning Rate is a Hyperparameter (selected/tuned, not learned during training)







https://srdas.github.io/DLBook/GradientDescentTechniques.html#issues-with-gradient-descent

Single Layer NN

- What is meant by “Training”?
- How do we Train a NN?
- What is the weight updation rule?

Single Layer NN

- What is meant by “Training”?
  - Learning trainable parameters (weights w) given the training data
- How do we Train a NN?

- What is the weight updation rule?

Single Layer NN

- What is meant by “Training”?
  - Learning trainable parameters (weights w) given the training data
- How do we Train a NN?
  - Minimize Error in Prediction: $\hat{y}_j - y_j$

    by optimizing weights using techniques such as Gradient descent
- What is the weight updation rule?

Single Layer NN

- What is meant by “Training”?
  - Learning trainable parameters (weights w) given the training data
- How do we Train a NN?
  - Minimize Error in Prediction: $\hat{y}_j - y_j$

    by optimizing weights using techniques such as Gradient descent
- What is the weight updation rule?
  - $W \leftarrow W - \eta \, \nabla_W \mathrm{E}$

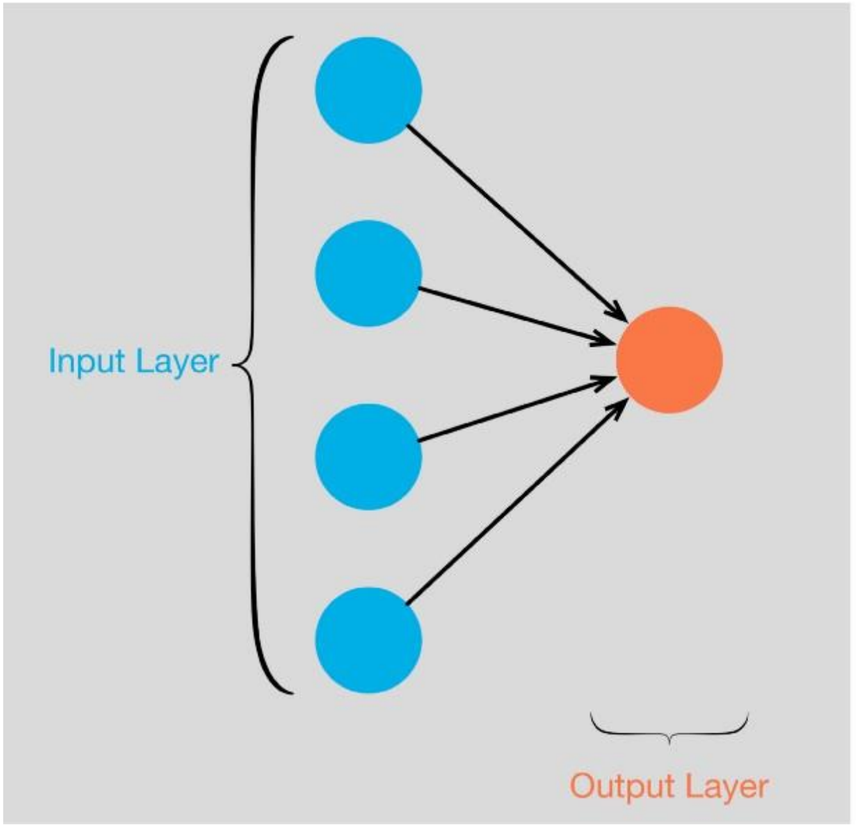
For Perceptron with Sigmoid Activation

$$\nabla_W \mathrm{E} = (\hat{y}_i - y_i)\, \hat{y}_i (1 - \hat{y}_i)\, X_i$$

Training neural networks-

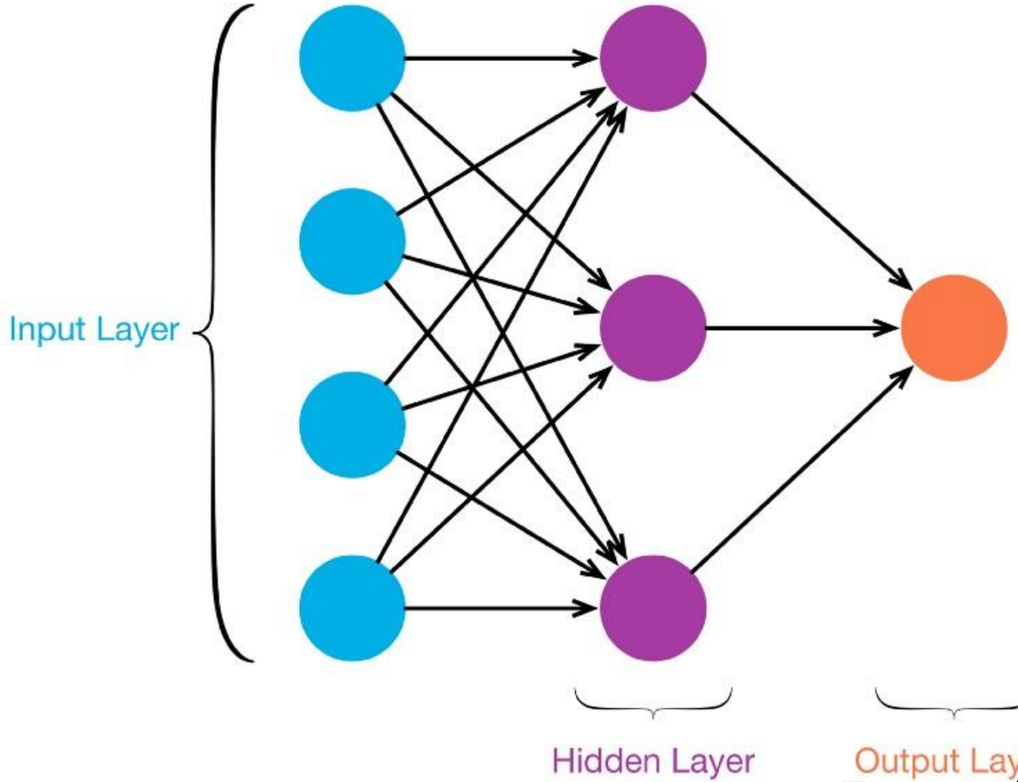
Backpropagation algorithm

Single Layer Single Output NN



Expected Output
is Known

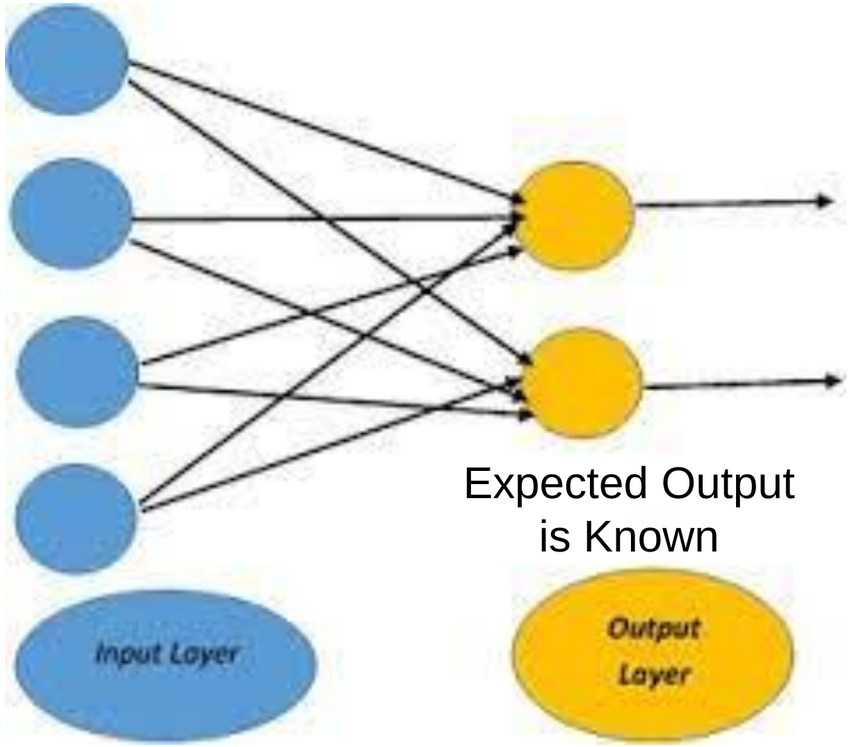
Multi Layer Single Output NN



Expected Output
Not Known

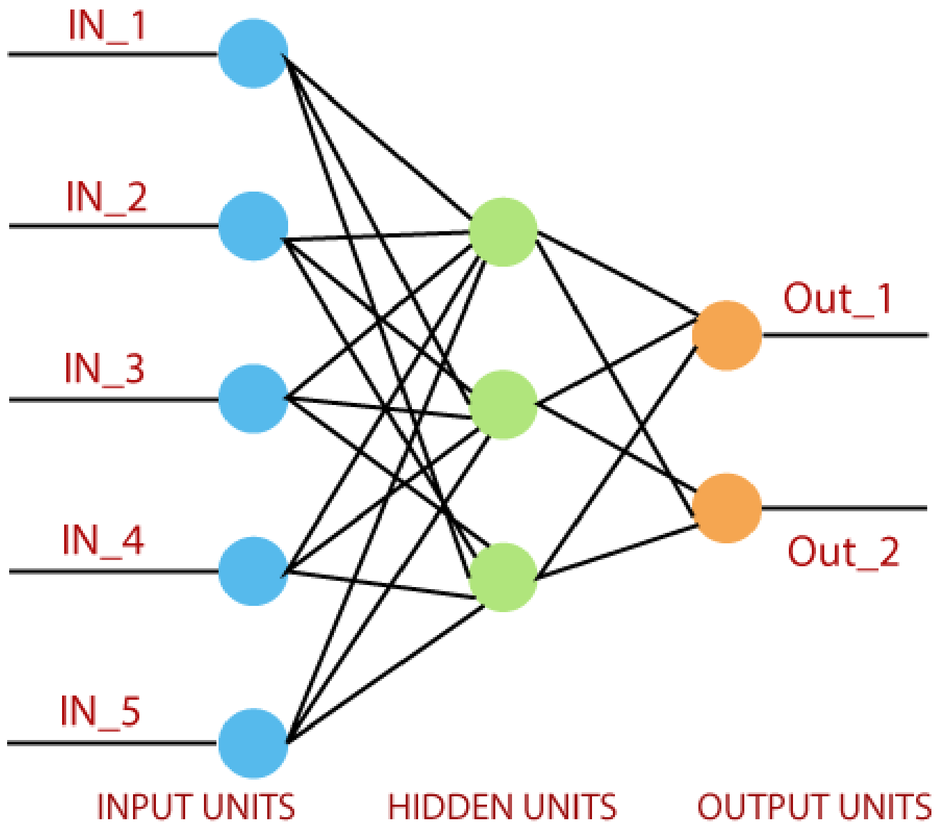
Expected Output
is Known

Single Layer Multiple Output NN



Expected Output
is Known

Multi Layer Multiple Output NN

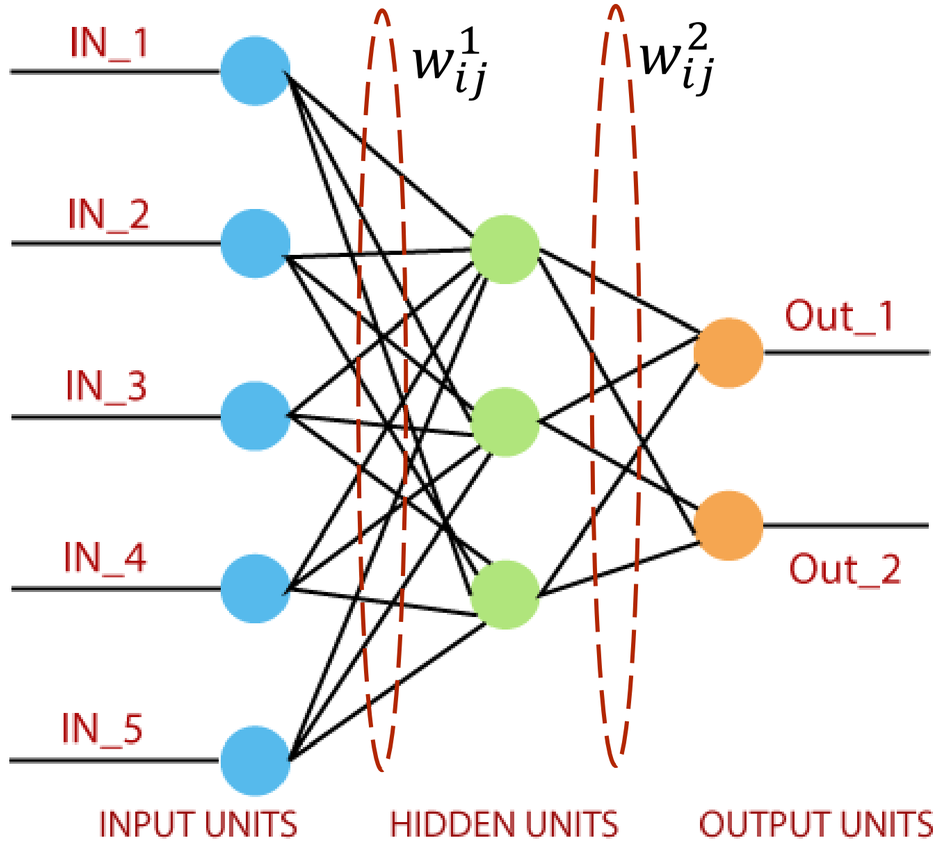


Expected Output
Not Known

Expected Output
is Known

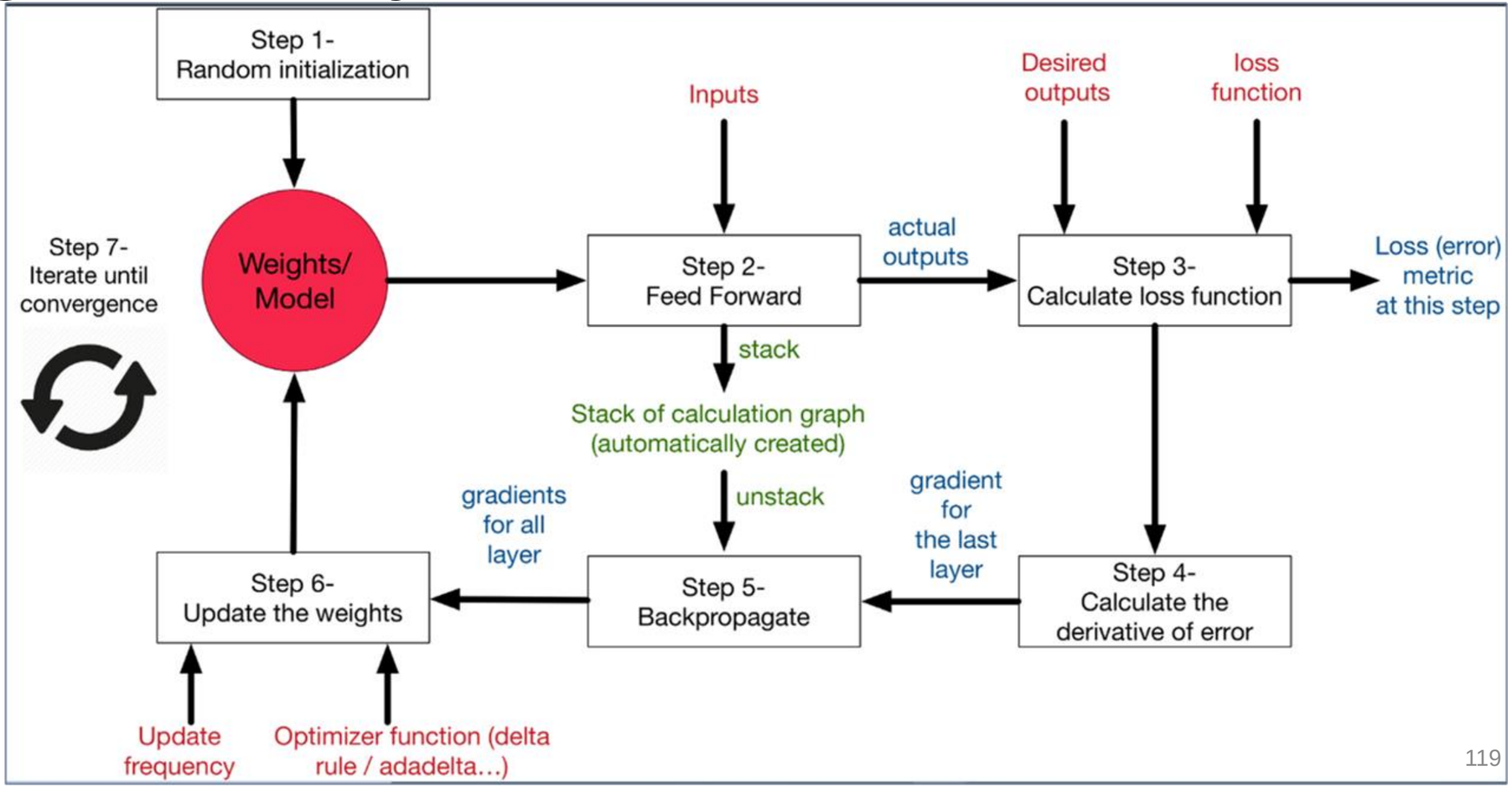
Multi-layer Perceptron

- Output of hidden layers is not known
- Error cannot be calculated for output of hidden layers
- Training of weights for hidden layers is done by assessing the effect of weights of hidden layer on the Error of output layer (Backpropagation)



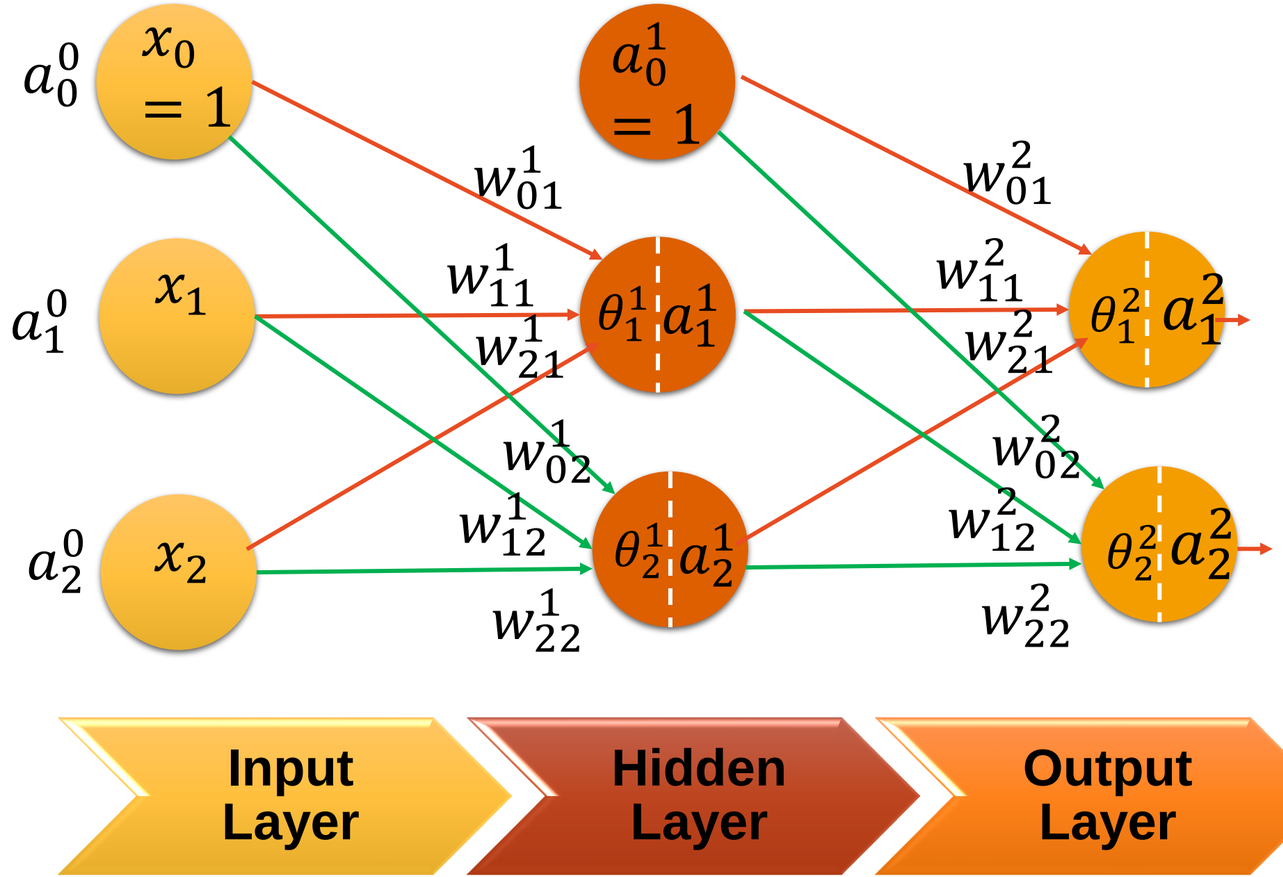
Training of Multilayer Neural Networks

- Forward Propagation:
  - Weights are fixed
  - Loss is calculated for each input
- Backward Propagation:
  - Loss is considered as a function of weights
  - Weights are varied to reduce loss



Example: MLP training with Backpropagation

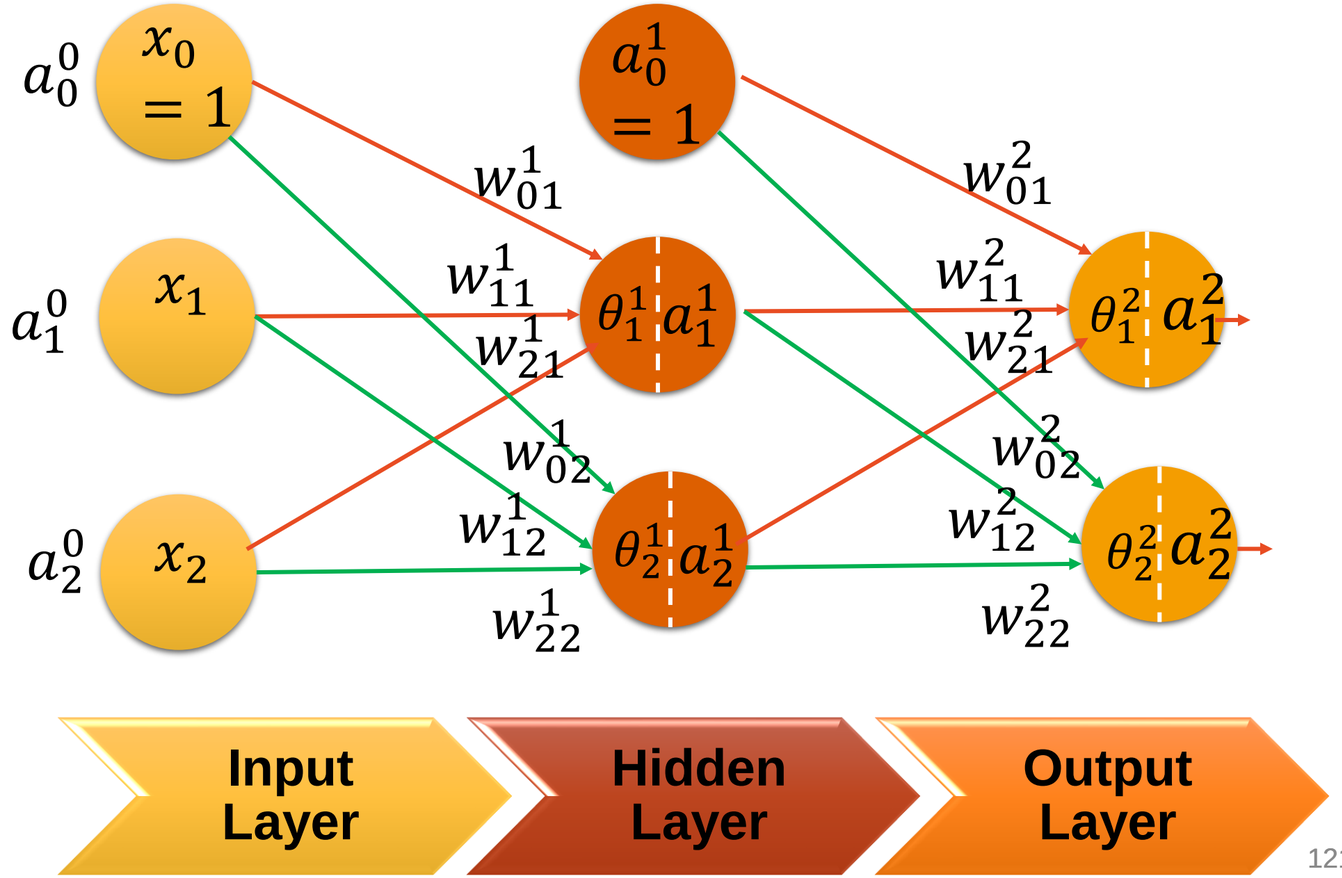
- Given:
  - Inputs, $X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.05 \\ 0.10 \end{bmatrix}$
  - Outputs, $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.99 \end{bmatrix}$
- Aim: ?



Example: MLP training with Backpropagation

- Given:
  - Inputs, $X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.05 \\ 0.10 \end{bmatrix}$
  - Outputs, $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.99 \end{bmatrix}$
- Aim: Determine Weights

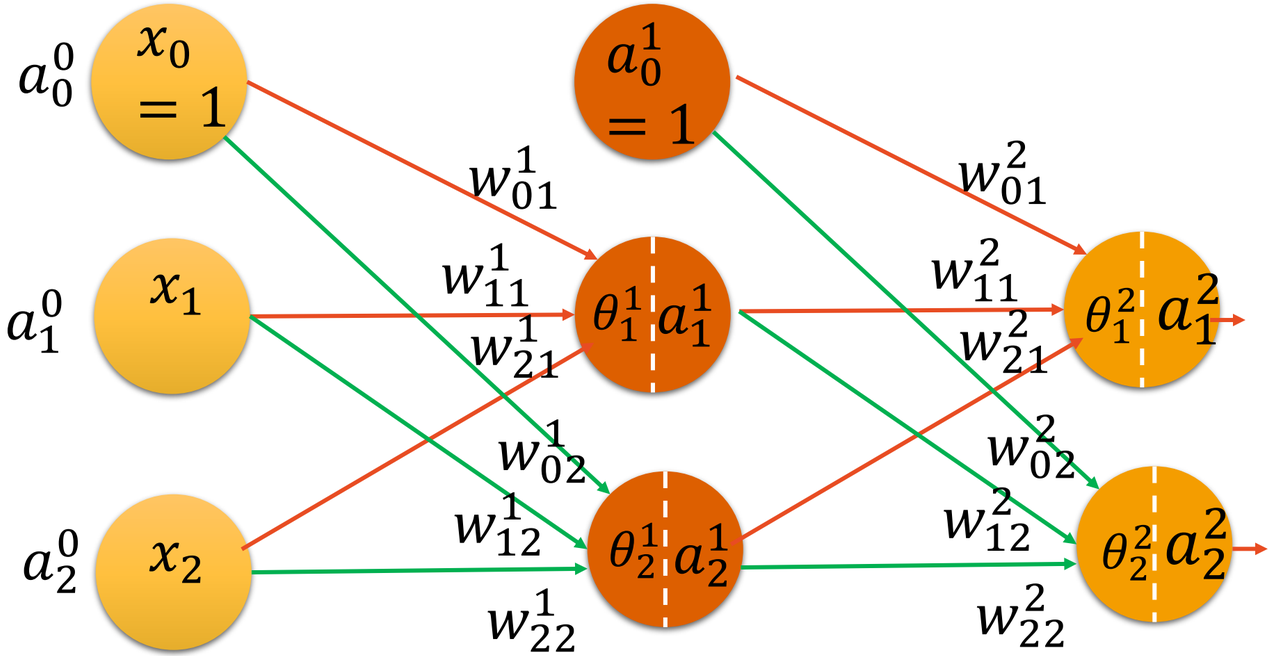
$$W_1 = \begin{bmatrix} w_{01}^1 & w_{02}^1 \\ w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix}, \; W_2 = \begin{bmatrix} w_{01}^2 & w_{02}^2 \\ w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \end{bmatrix}$$



Example: MLP training with Backpropagation

1. Initialize Weights

- $W_1 = \begin{bmatrix} w_{01}^1 & w_{02}^1 \\ w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix} = \begin{bmatrix} 0.35 & 0.35 \\ 0.15 & 0.25 \\ 0.20 & 0.30 \end{bmatrix}$
- $W_2 = \begin{bmatrix} w_{01}^2 & w_{02}^2 \\ w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \end{bmatrix} = \begin{bmatrix} 0.60 & 0.60 \\ 0.40 & 0.50 \\ 0.45 & 0.55 \end{bmatrix}$

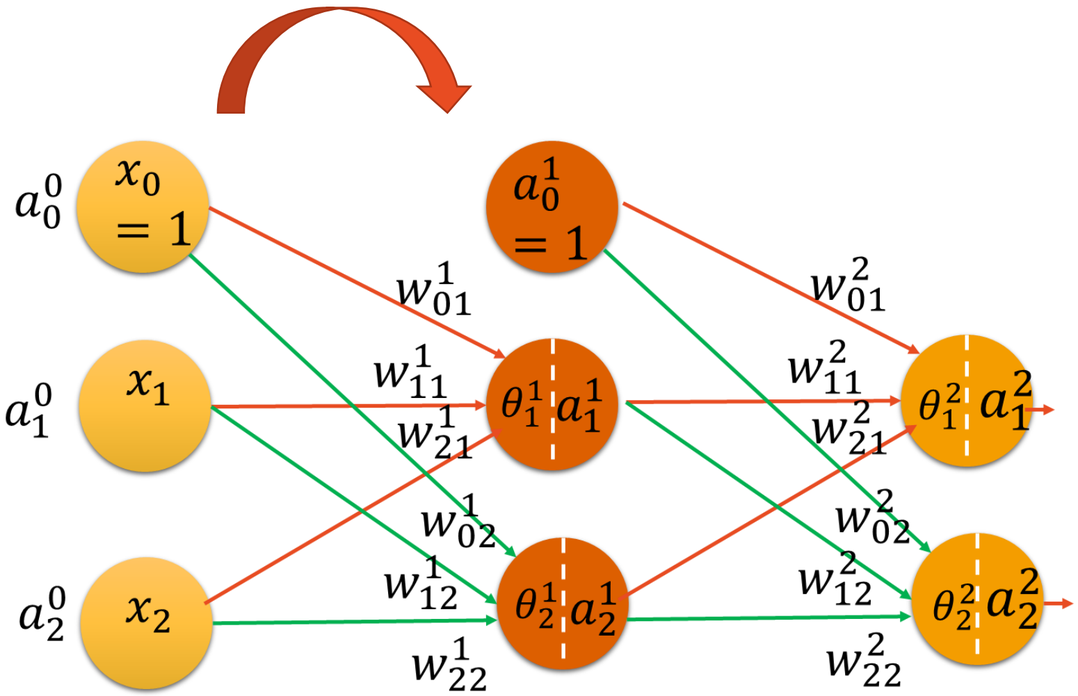


2. Determine Total Error (Forward Pass)

- **Determine Outputs of Layer 1**

- $\begin{bmatrix}\theta_1^1\\ \theta_2^1\end{bmatrix} = {W_1}^T X = \begin{bmatrix}0.35 & 0.15 & 0.20\\ 0.35 & 0.25 & 0.30\end{bmatrix}\begin{bmatrix}1\\ 0.05\\ 0.10\end{bmatrix} = \begin{bmatrix}0.3775\\ 0.3925\end{bmatrix}$

- $\begin{bmatrix}a_1^1\\ a_2^1\end{bmatrix} = \sigma\left({W_1}^T X\right) = \begin{bmatrix}\frac{1}{1+e^{-0.3775}}\\ \frac{1}{1+e^{-0.3925}}\end{bmatrix} = \begin{bmatrix}0.5933\\ 0.5969\end{bmatrix}$



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

2. Determine Total Error (Forward Pass)

- Determine Outputs of Layer 1

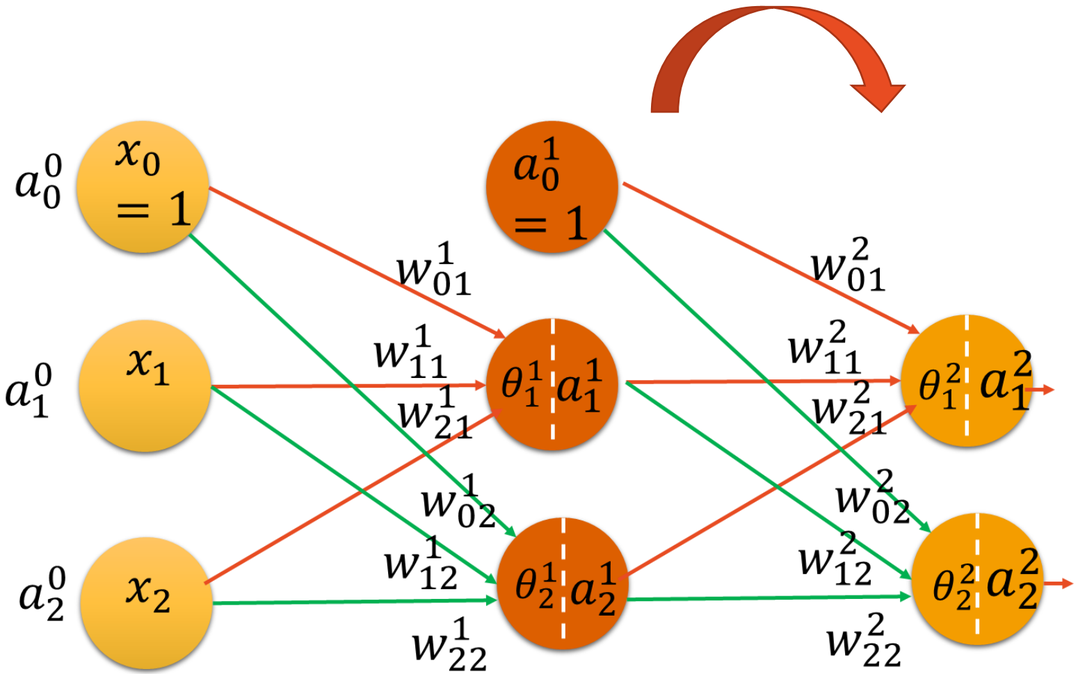
- $\begin{bmatrix}\theta_1^1\\ \theta_2^1\end{bmatrix} = {W_1}^T X = \begin{bmatrix}0.35 & 0.15 & 0.20\\ 0.35 & 0.25 & 0.30\end{bmatrix}\begin{bmatrix}1\\ 0.05\\ 0.10\end{bmatrix} = \begin{bmatrix}0.3775\\ 0.3925\end{bmatrix}$

- $\begin{bmatrix}a_1^1\\ a_2^1\end{bmatrix} = \sigma\left({W_1}^T X\right) = \begin{bmatrix}\frac{1}{1+e^{-0.3775}}\\ \frac{1}{1+e^{-0.3925}}\end{bmatrix} = \begin{bmatrix}0.5933\\ 0.5969\end{bmatrix}$

- **Determine Outputs of Layer 2**

- $\begin{bmatrix}\theta_1^2\\ \theta_2^2\end{bmatrix} = {W_2}^T A_1 = \begin{bmatrix}0.60 & 0.40 & 0.45\\ 0.60 & 0.50 & 0.55\end{bmatrix}\begin{bmatrix}1\\ 0.5933\\ 0.5969\end{bmatrix} = \begin{bmatrix}1.1059\\ 1.2249\end{bmatrix}$

- $\begin{bmatrix}a_1^2\\ a_2^2\end{bmatrix} = \sigma\left({W_2}^T A_1\right) = \begin{bmatrix}\frac{1}{1+e^{-1.1059}}\\ \frac{1}{1+e^{-1.2249}}\end{bmatrix} = \begin{bmatrix}0.7514\\ 0.7729\end{bmatrix}$



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

2. Determine Total Error (Forward Pass)

- Determine Outputs of Layer 1

- $\begin{bmatrix}\theta_1^1\\ \theta_2^1\end{bmatrix} = {W_1}^T X = \begin{bmatrix}0.35 & 0.15 & 0.20\\ 0.35 & 0.25 & 0.30\end{bmatrix}\begin{bmatrix}1\\ 0.05\\ 0.10\end{bmatrix} = \begin{bmatrix}0.3775\\ 0.3925\end{bmatrix}$

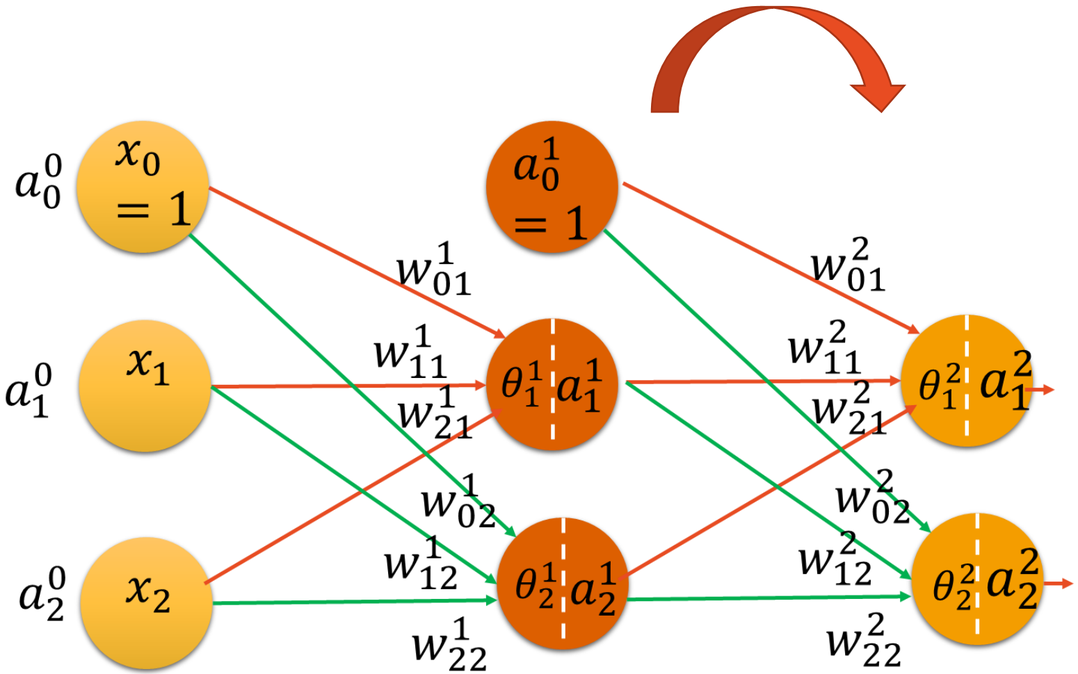
- $\begin{bmatrix}a_1^1\\ a_2^1\end{bmatrix} = \sigma\left({W_1}^T X\right) = \begin{bmatrix}\frac{1}{1+e^{-0.3775}}\\ \frac{1}{1+e^{-0.3925}}\end{bmatrix} = \begin{bmatrix}0.5933\\ 0.5969\end{bmatrix}$

- **Determine Outputs of Layer 2**

- $\begin{bmatrix}\theta_1^2\\ \theta_2^2\end{bmatrix} = {W_2}^T A_1 = \begin{bmatrix}0.60 & 0.40 & 0.45\\ 0.60 & 0.50 & 0.55\end{bmatrix}\begin{bmatrix}1\\ 0.5933\\ 0.5969\end{bmatrix} = \begin{bmatrix}1.1059\\ 1.2249\end{bmatrix}$

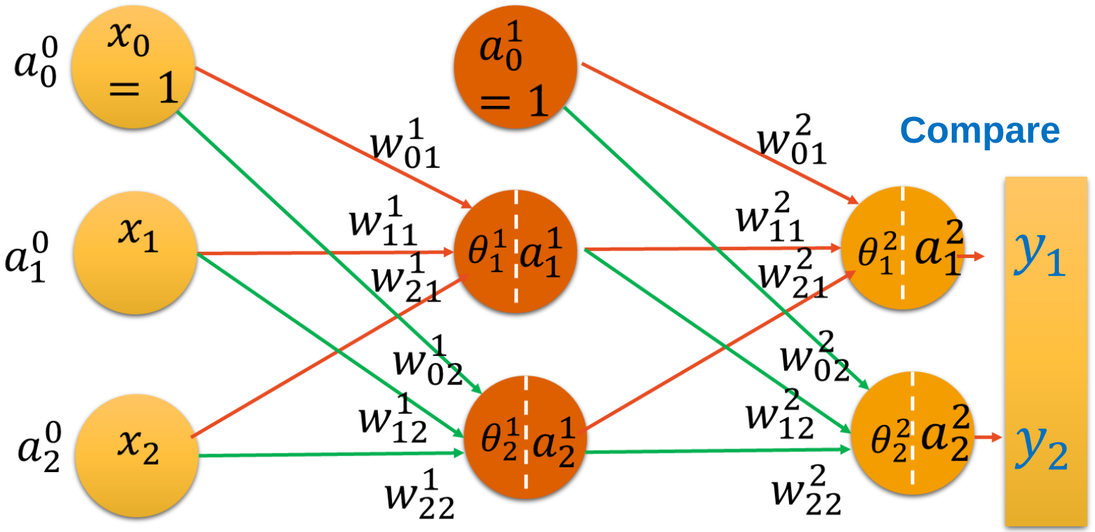
- $\begin{bmatrix}a_1^2\\ a_2^2\end{bmatrix} = \sigma\left({W_2}^T A_1\right) = \begin{bmatrix}\frac{1}{1+e^{-1.1059}}\\ \frac{1}{1+e^{-1.2249}}\end{bmatrix} = \begin{bmatrix}0.7514\\ 0.7729\end{bmatrix}$

Prediction made by the network



2. Determine Total Error (Forward Pass)

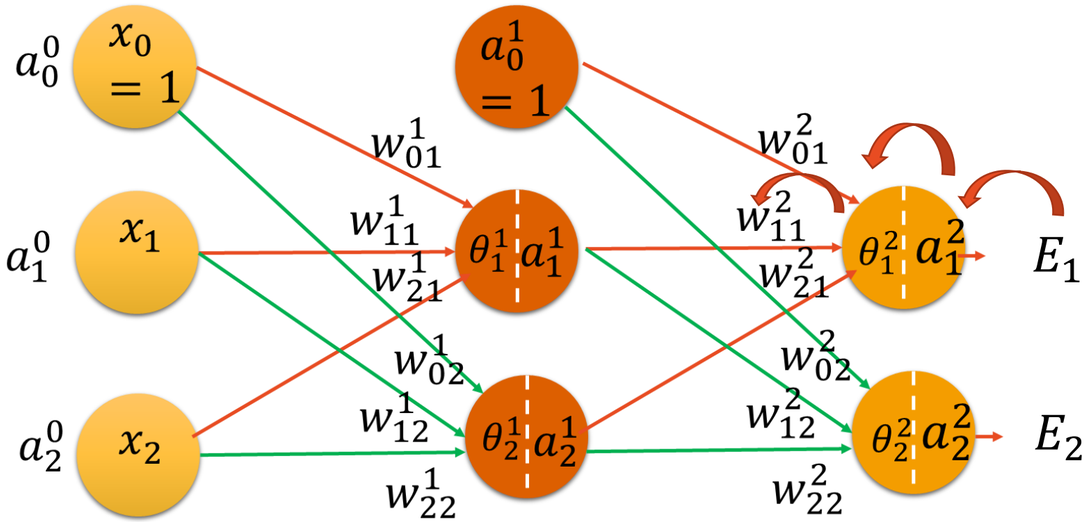
- Error at each output unit
- $\begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} a_1^2 - y_1 \\ a_2^2 - y_2 \end{bmatrix} = \begin{bmatrix} 0.7514 - 0.01 \\ 0.7729 - 0.99 \end{bmatrix} = \begin{bmatrix} 0.7414 \\ -0.2171 \end{bmatrix}$
- **Total Error**
- $\mathrm{E} = \frac{1}{2}\sum_{j=1}^{M_k}\left(a_j^k - y_j\right)^2 = \frac{1}{2}(0.7414^2 + (-0.2171)^2) = 0.2984$



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

3. Update the weights **(Backward Pass)**

- Update the weights (Layer 2)
- $\frac{\partial E}{\partial w_{11}^2} = \frac{\partial E}{\partial a_1^2}\frac{\partial a_1^2}{\partial \theta_1^2}\frac{\partial \theta_1^2}{\partial w_{11}^2}$
- $\frac{\partial E}{\partial w_{11}^2} = (a_1^2 - y_1)a_1^2(1-a_1^2)a_1^1 = \delta_1^2 a_1^1$
- $\delta^2 = \begin{bmatrix}\delta_1^2 \\ \delta_2^2\end{bmatrix} = \begin{bmatrix}E_1 \\ E_2\end{bmatrix} .* \begin{bmatrix}a_1^2 \\ a_2^2\end{bmatrix} .* \begin{bmatrix}1-a_1^2 \\ 1-a_2^2\end{bmatrix} = \begin{bmatrix}0.1385 \\ -0.0381\end{bmatrix}$
- $\nabla_{W_2}\mathrm{E} = \begin{bmatrix}1 \\ a_1^1 \\ a_2^1\end{bmatrix}(\delta^2)' = \begin{bmatrix}0.1385 & -0.0381 \\ 0.0822 & -0.0226 \\ 0.0827 & -0.0227\end{bmatrix}$
- Thus, $W_2^{new} = W_2 - \eta\nabla_{W_2}\mathrm{E}$



$$\mathrm{E} = E_1^2 + E_2^2$$

- Pre-activation,
  $\theta_1^2 = w_{01}^2\, a_0^1 + w_{11}^2\, a_1^1 + w_{21}^2\, a_2^1,$
- Post-activation
  $a_1^2 = \sigma(\theta_1^2)$

3. Update the weights **(Backward Pass)**

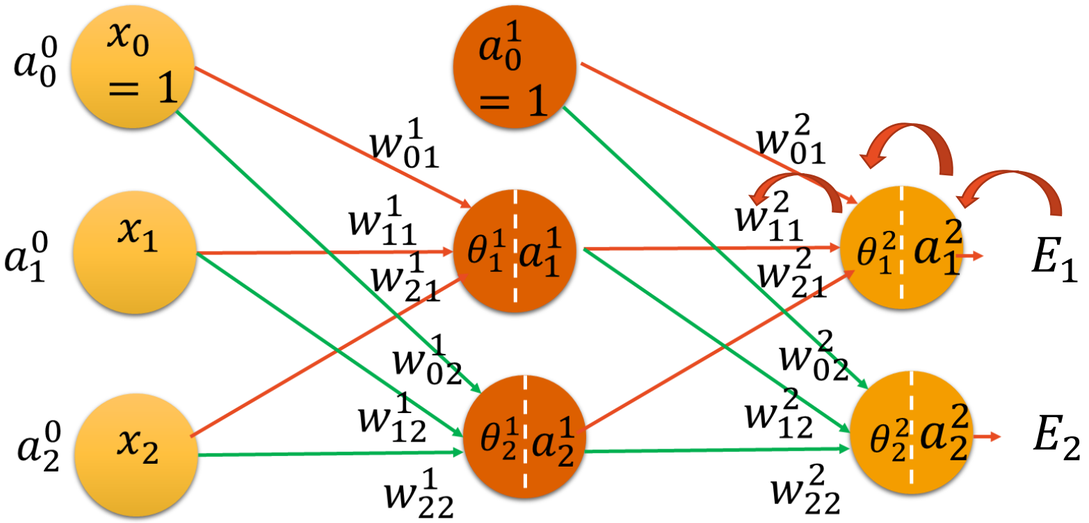
- Determine Error Gradient (Layer 2)

- $\nabla_{W_2} \mathrm{E} = \begin{bmatrix} 1 \\ a_1^1 \\ a_2^1 \end{bmatrix} \left(\delta^2\right)' = \begin{bmatrix} 0.1385 & -0.0381 \\ 0.0822 & -0.0226 \\ 0.0827 & -0.0227 \end{bmatrix}$

- $W_2^{new} = W_2 - \eta \nabla_{W_2} \mathrm{E}$

$$= \begin{bmatrix} 0.60 & 0.60 \\ 0.40 & 0.50 \\ 0.45 & 0.55 \end{bmatrix} - 0.5 \begin{bmatrix} 0.1385 & -0.0381 \\ 0.0822 & -0.0226 \\ 0.0827 & -0.0227 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5308 & 0.6190 \\ 0.3589 & 0.5113 \\ 0.4087 & 0.5614 \end{bmatrix}$$

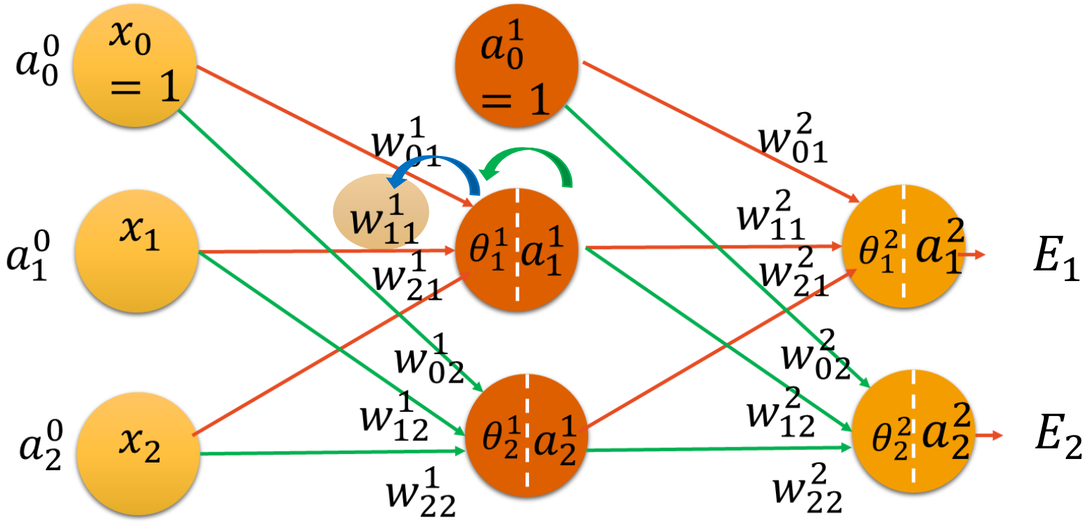


$$\mathrm{E} = E_1^2 + E_2^2$$

- Pre-activation,
  $\theta_1^2 = w_{01}^2\, a_0^1 + w_{11}^2\, a_1^1 + w_{21}^2\, a_2^1,$
- Post-activation
  $a_1^2 = \sigma(\theta_1^2)$

3. Update the weights **(Backward Pass)**

- Determine Error Gradient (Layer 1)
- $\frac{\partial E}{\partial w_{11}^1} = \frac{\partial E}{\partial a_1^1}\frac{\partial a_1^1}{\partial \theta_1^1}\frac{\partial \theta_1^1}{\partial w_{11}^1} = \frac{\partial E}{\partial a_1^1} a_1^1(1-a_1^1)a_1^0$



- Pre-activation,
  $\theta_1^1 = w_{01}^1\, a_0^0 + w_{11}^1\, a_1^0 + w_{21}^1\, a_2^0,$
- Post-activation
  $a_1^1 = \sigma(\theta_1^1)$

$$\mathrm{E} = E_1^2 + E_2^2$$

3. Update the weights (Backward Pass)

- Determine Error Gradient (Layer 1)

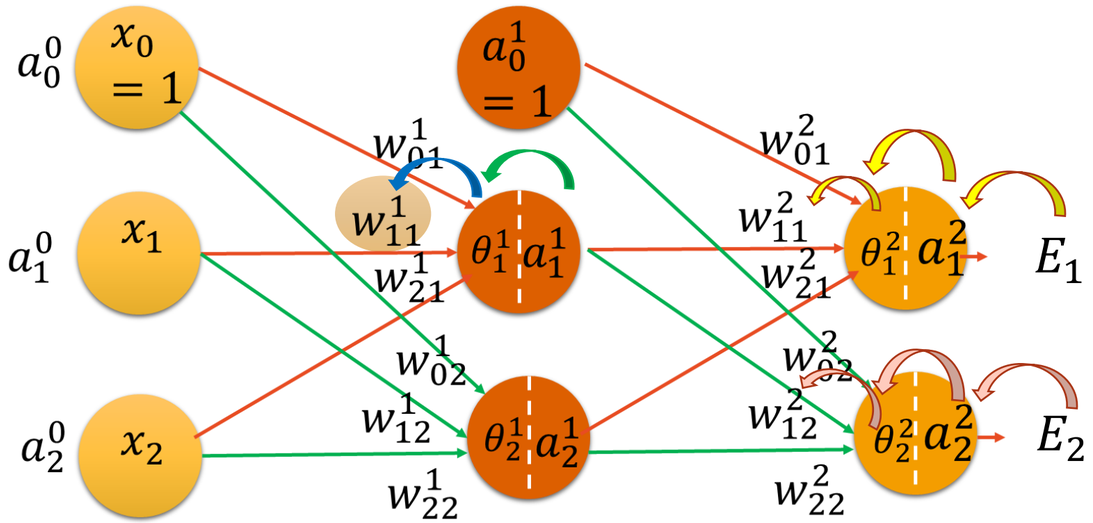
- $\frac{\partial E}{\partial w_{11}^1} = \frac{\partial E}{\partial a_1^1}\frac{\partial a_1^1}{\partial \theta_1^1}\frac{\partial \theta_1^1}{\partial w_{11}^1} = \frac{\partial E}{\partial a_1^1} a_1^1(1-a_1^1)a_1^0$ ($a_1^0 = x_1$)

- $\frac{\partial E}{\partial a_1^1} = \frac{\partial E}{\partial a_1^2}\frac{\partial a_1^2}{\partial \theta_1^2}\frac{\partial \theta_1^2}{\partial a_1^1} + \frac{\partial E}{\partial a_2^2}\frac{\partial a_2^2}{\partial \theta_2^2}\frac{\partial \theta_2^2}{\partial a_1^1}$

- $= (a_1^2 - y_1)a_1^2(1-a_1^2)w_{11}^2 + (a_2^2 - y_2)a_2^2(1-a_2^2)w_{12}^2$

- $= \delta_1^2 w_{11}^2 + \delta_2^2 w_{12}^2$

- Then $w_{11}^1 \leftarrow w_{11}^1 - \eta\frac{\partial E}{\partial w_{11}^1}$



$$E = E_1^2 + E_2^2$$

- Pre-activation,
  $\theta_1^2 = w_{01}^2\, a_0^1 + w_{11}^2\, a_1^1 + w_{21}^2\, a_2^1,$
- Post-activation
  $a_1^2 = \sigma(\theta_1^2)$

Hidden Layer ($(k-1)^{\text{th}}$ layer)

- $w_{pi}^{k-1} \leftarrow w_{pi}^{k-1} - \eta\frac{\partial E}{\partial w_{pi}^{k-1}} = w_{pi}^{k-1} - \eta\delta_i^{k-1}a_p^{k-2}$

- $\delta_i^{k-1} = \sum_{j=1}^{M_k}\left(a_j^k - y_j\right)a_j^k\left(1-a_j^k\right)w_{ij}^k\, a_i^{k-1}\left(1-a_i^{k-1}\right)$
  $= \sum_{j=1}^{M_k}\delta_j^k\, w_{ij}^k\, a_i^{k-1}\left(1-a_i^{k-1}\right)$

3. Update the weights (Backward Pass)

- Determine Error Gradient (Layer 1)

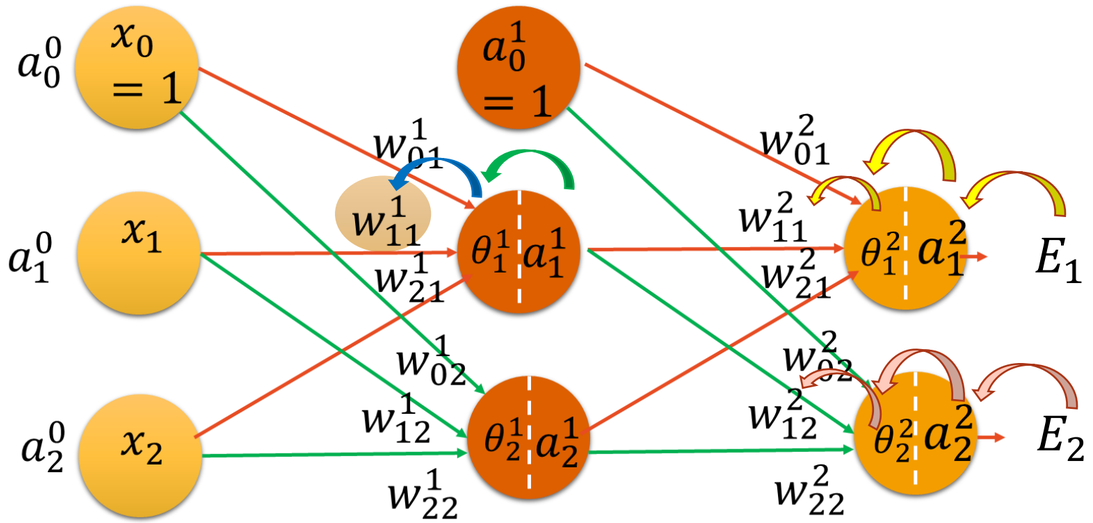
- $W_{1(new)} = W_1 - \eta \, \mathrm{X} \, (\delta^1)^T = W_1 - \eta \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} (\delta^1)^T$

- $\delta^1 = \begin{bmatrix} \delta_1^1 \\ \delta_2^1 \end{bmatrix} = \begin{bmatrix} \delta_1^2 w_{11}^2 a_1^1 (1 - a_1^1) + \delta_2^2 w_{12}^2 a_1^1 (1 - a_1^1) \\ \delta_1^2 w_{21}^2 a_2^1 (1 - a_2^1) + \delta_2^2 w_{22}^2 a_2^1 (1 - a_2^1) \end{bmatrix}$

- $\delta^1 = \begin{bmatrix} 0.00877 \\ 0.00995 \end{bmatrix}$

- $W_{1(new)} = \begin{bmatrix} 0.35 & 0.35 \\ 0.15 & 0.25 \\ 0.20 & 0.30 \end{bmatrix} - 0.5 \begin{bmatrix} 1 \\ 0.05 \\ 0.10 \end{bmatrix} (\delta^1)^T$

$$= \begin{bmatrix} 0.3456 & 0.3450 \\ 0.1498 & 0.2498 \\ 0.1996 & 0.2995 \end{bmatrix}$$



$$\mathrm{E} = E_1^2 + E_2^2$$

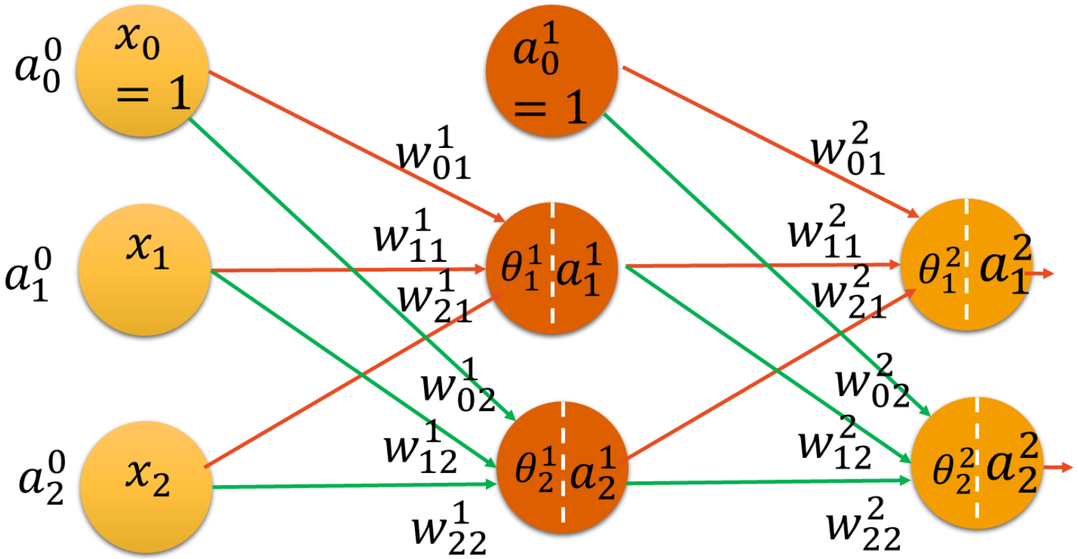
Hidden Layer ($(k-1)^{\text{th}}$ layer)

- $w_{pi}^{k-1} \leftarrow w_{pi}^{k-1} - \eta \frac{\partial E}{\partial w_{pi}^{k-1}} = w_{pi}^{k-1} - \eta \delta_i^{k-1} a_p^{k-2}$

- $\delta_i^{k-1} = \sum_{j=1}^{M_k} \underbrace{(a_j^k - y_j) \, a_j^k (1 - a_j^k)}_{\boldsymbol{\delta_j^k}} w_{ij}^k \, a_i^{k-1} (1 - a_i^{k-1})$

(Forward Pass: epoch2)

- Determine Outputs of Layer 1
- $\begin{bmatrix}\theta_1^1\\ \theta_2^1\end{bmatrix} = {W_1}^T X = \begin{bmatrix}0.3456 & 0.3450\\ 0.1498 & 0.2498\\ 0.1996 & 0.2995\end{bmatrix}^T \begin{bmatrix}1\\ 0.05\\ 0.10\end{bmatrix} = \begin{bmatrix}0.3731\\ 0.3875\end{bmatrix}$
- $\begin{bmatrix}a_1^1\\ a_2^1\end{bmatrix} = \sigma\left({W_1}^T X\right) = \begin{bmatrix}\frac{1}{1+e^{-0.3731}}\\ \frac{1}{1+e^{-0.3875}}\end{bmatrix} = \begin{bmatrix}0.5922\\ 0.5957\end{bmatrix}$
- Determine Outputs of Layer 2
- $\begin{bmatrix}\theta_1^2\\ \theta_2^2\end{bmatrix} = {W_2}^T A_1 = \begin{bmatrix}0.5308 & 0.6190\\ 0.3589 & 0.5113\\ 0.4087 & 0.5614\end{bmatrix}^T \begin{bmatrix}1\\ 0.5922\\ 0.5957\end{bmatrix} = \begin{bmatrix}0.9867\\ 1.2562\end{bmatrix}$
- $\begin{bmatrix}a_1^2\\ a_2^2\end{bmatrix} = \sigma\left({W_2}^T A_1\right) = \begin{bmatrix}\frac{1}{1+e^{-0.9867}}\\ \frac{1}{1+e^{-1.2562}}\end{bmatrix} = \begin{bmatrix}0.7284\\ 0.7784\end{bmatrix}$



- **Total Error**
- $\mathrm{E} = \frac{1}{2}\sum_{j=1}^{M_k}\left(a_j^k - y_j\right)^2$

$= \frac{1}{2}(0.7184^2 + (-0.2116)^2)$

$= 0.2805$ (reduced)

$E = 0.2984$ (before epoch1)

Backpropagation Numerical Example

- A Step by Step Backpropagation Example
  https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Module I: Introduction to Generative AI and Fundamental Study

Module I (20%)

Basic introduction to Generative AI, Applications of Generative AI in various fields, Perceptron and Multilayer Perceptron (MLP), Gradient Descent, Backpropagation algorithm, Loss functions, Understanding Bias and Variance

Loss Function

Loss function is a method of evaluating
how well your algorithm is modelling your dataset.

- If your predictions are totally off, your loss function will output a higher number.

  If they're pretty good, it'll output a lower number.
- As you tune your algorithm to try and improve your model, your loss function will tell you if you're improving or not.
- 'Loss' helps us to understand how much the predicted value differ from actual value

Loss Function value		Algorithm Performance
High	→	Poor
Low	→	Great

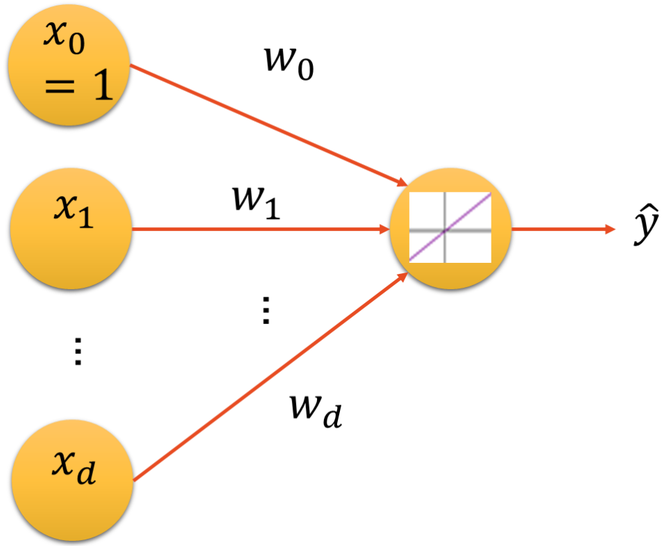
Loss Functions for Regression Task

- Output Layer Configuration:

  One node with a linear activation unit

  Eg. Predict a real-value quantity (Regression)

- Loss Functions:



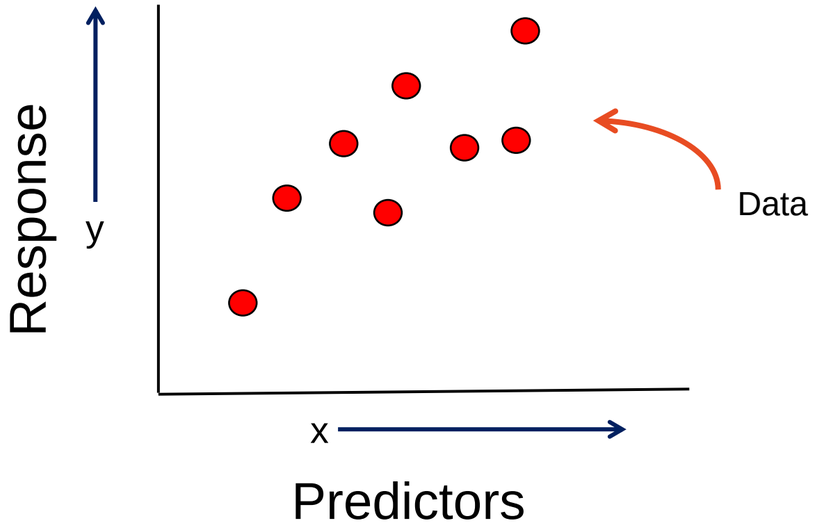
Mean Squared Error, $\mathrm{MSE} = \frac{1}{N}\sum_{k=1}^{N}|e_k|^2$

Root Mean Squared Error, $\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}|e_k|^2}$

Mean Absolute Error, $\mathrm{MAE} = \frac{1}{N}\sum_{k=1}^{N}|e_k|$

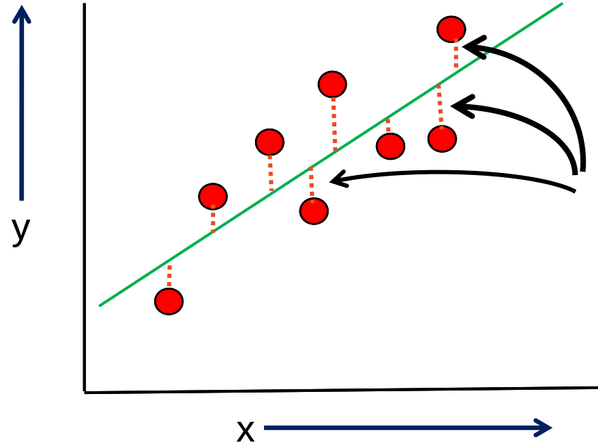
Eg: Loss Functions for Regression Task

Fitting of line in Linear Regression



Root Mean Square Error (RMSE) Calculation

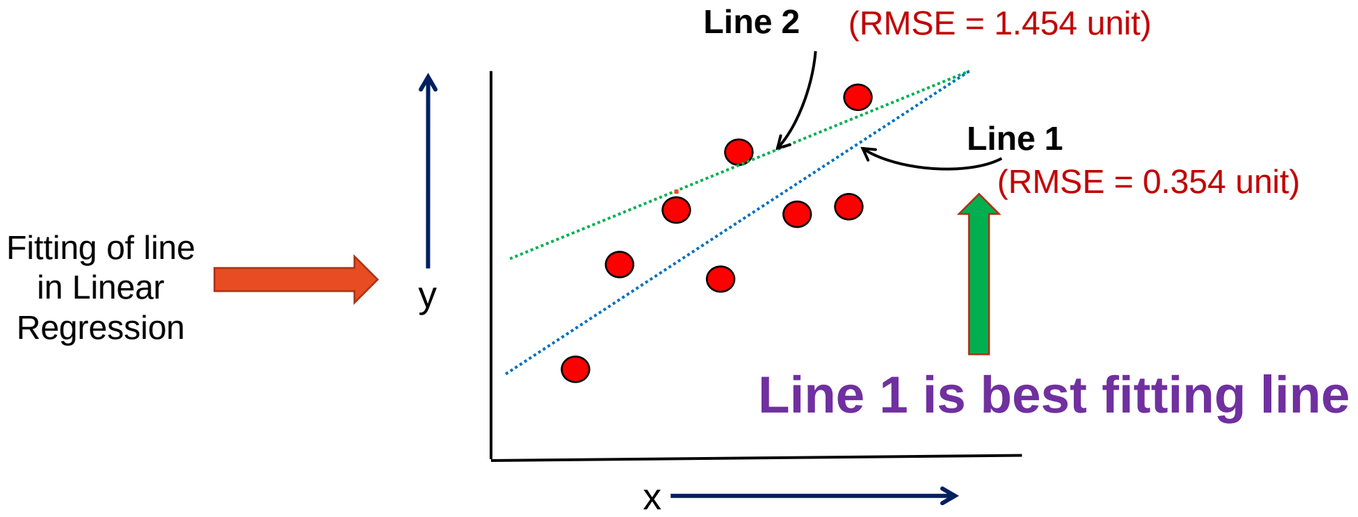
$$\sqrt{(d_1{}^2 + d_2{}^2 + d_3{}^2 + \ldots\ldots + d_n{}^2)} = 0.354$$ (assumption for line 1)



Calculation of distance of data point from a reference line to compute root mean square error

Eg: Loss Functions for Regression Task

After rotating a line for multiple angles, RMSE is calculated for each line



Cost Function and Loss Function

Single Sample

Input			Output	Model Prediction
Temperature (°C)	Vegetation Index	Elevation (m)	Actual Annual Rainfall (mm)	Predicted Annual Rainfall (mm)
30	0.8	500	1200	1100
25	0.7	600	800	750
35	0.6	700	1500	1600
28	0.9	450	1000	950

Loss Function/Error Function

$\hat{Y}$-Y	$(\hat{Y}-Y)^2$
100	10000
50	2500
100	10000
50	2500

Entire Data

Cost Function = $\frac{1}{4}((\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + (\hat{y}_3 - y_3)^2 + (\hat{y}_4 - y_4)^2)$

$= \frac{1}{4}(10000 + 2500 + 10000 + 2500) = 6250$

Loss Function

- Output Layer Configuration:

  One node with a linear activation unit

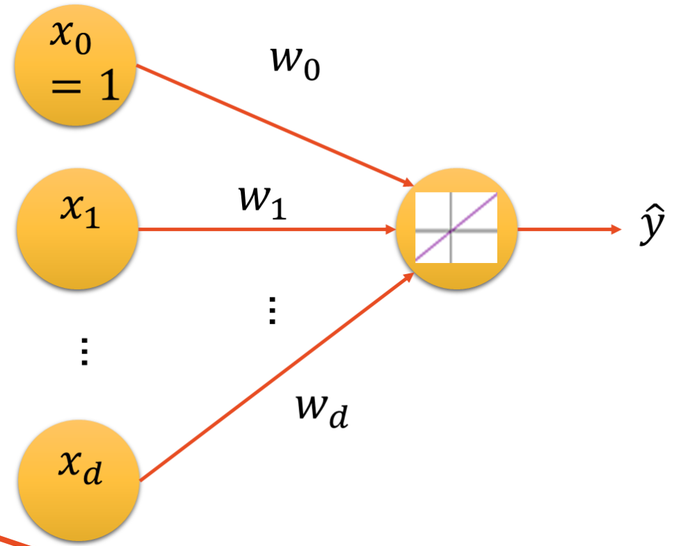
  Eg. Predict a real-value quantity (Regression)

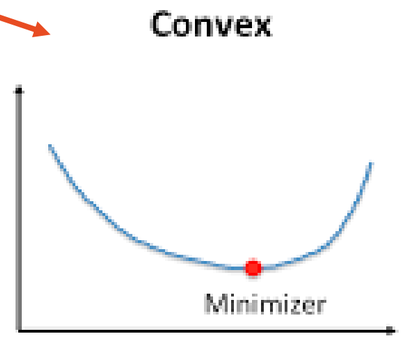
- For linear activation at output unit, Loss Function Mean Squared Error (MSE) is convex

- In Classification, Output Layer uses

Sigmoid activation (Binary Classification) or
Softmax activation (Multi-class Classification)

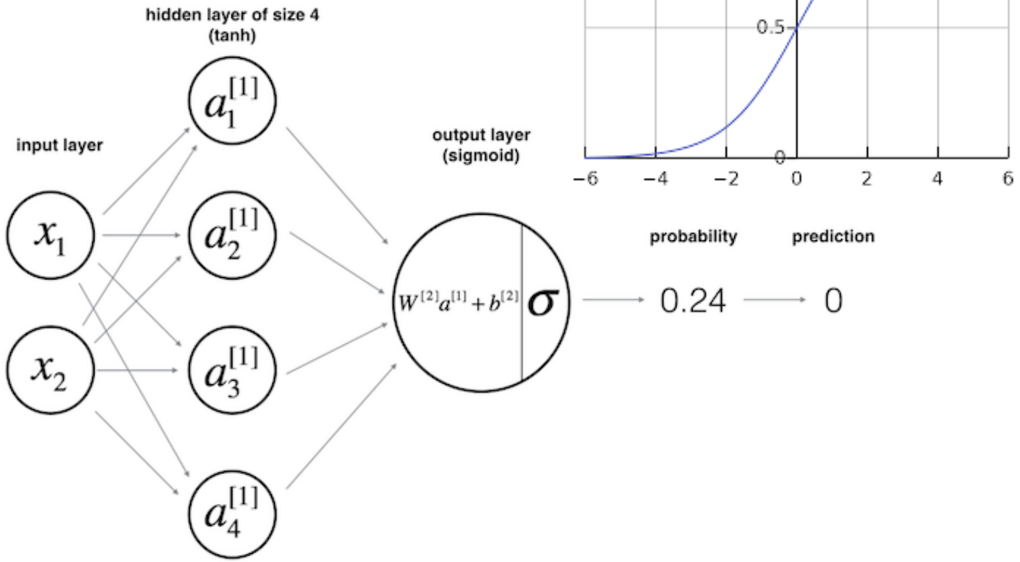
Limit the output between [0,1]
Then it is interpreted as class probability





Sigmoid Activation

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

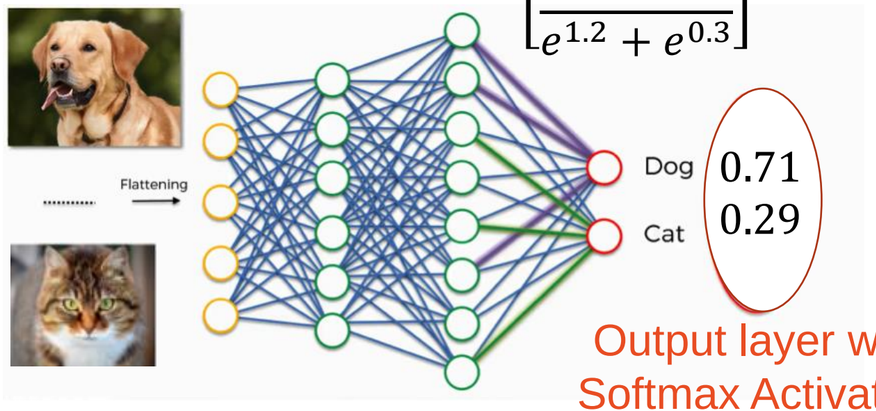


- Good when output is to be interpreted as probability

Softmax Activation

- $\sigma(z)_j = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}}, j = 1,2,\ldots K$
- outputs the probability for each class and these probabilities will sum up to one

$$\sigma\left(\begin{bmatrix}1.2\\0.3\end{bmatrix}\right) = \begin{bmatrix}\frac{e^{1.2}}{e^{1.2}+e^{0.3}}\\ \frac{e^{0.3}}{e^{1.2}+e^{0.3}}\end{bmatrix} = \begin{bmatrix}0.71\\0.29\end{bmatrix} = \begin{bmatrix}P(dog)\\P(cat)\end{bmatrix}$$

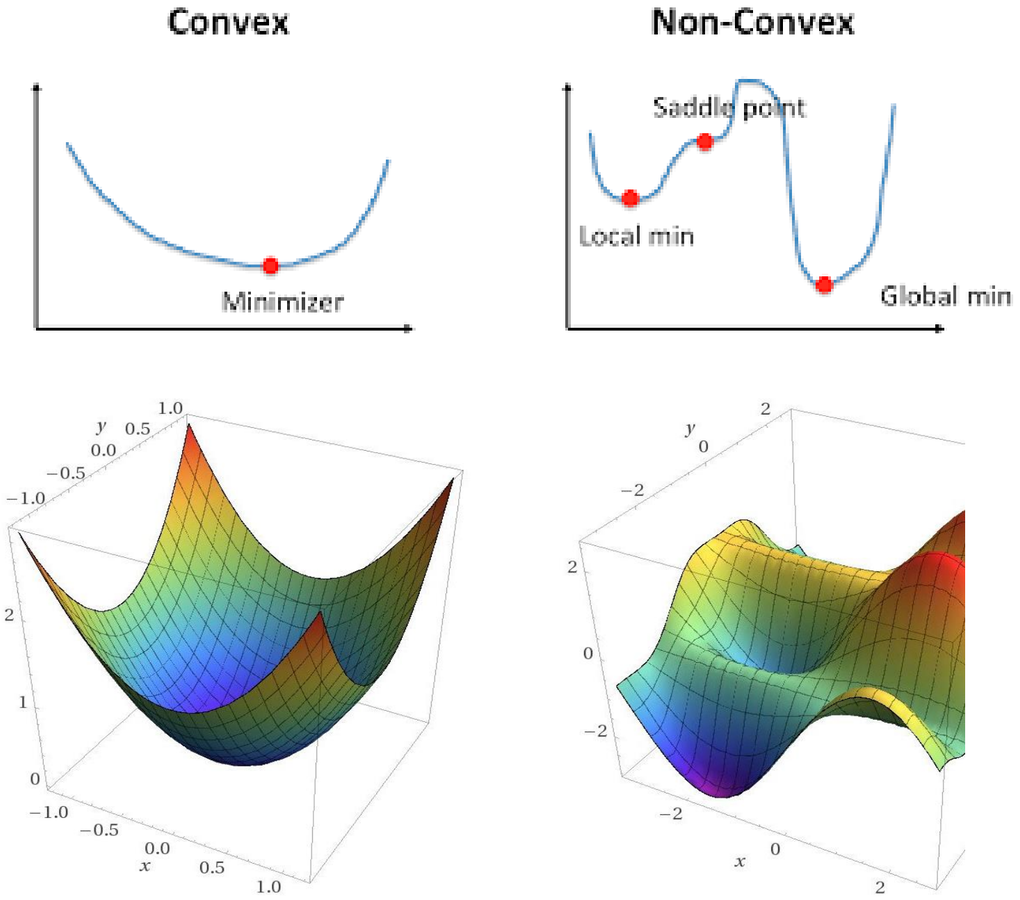


Output layer with Softmax Activation

Loss Function

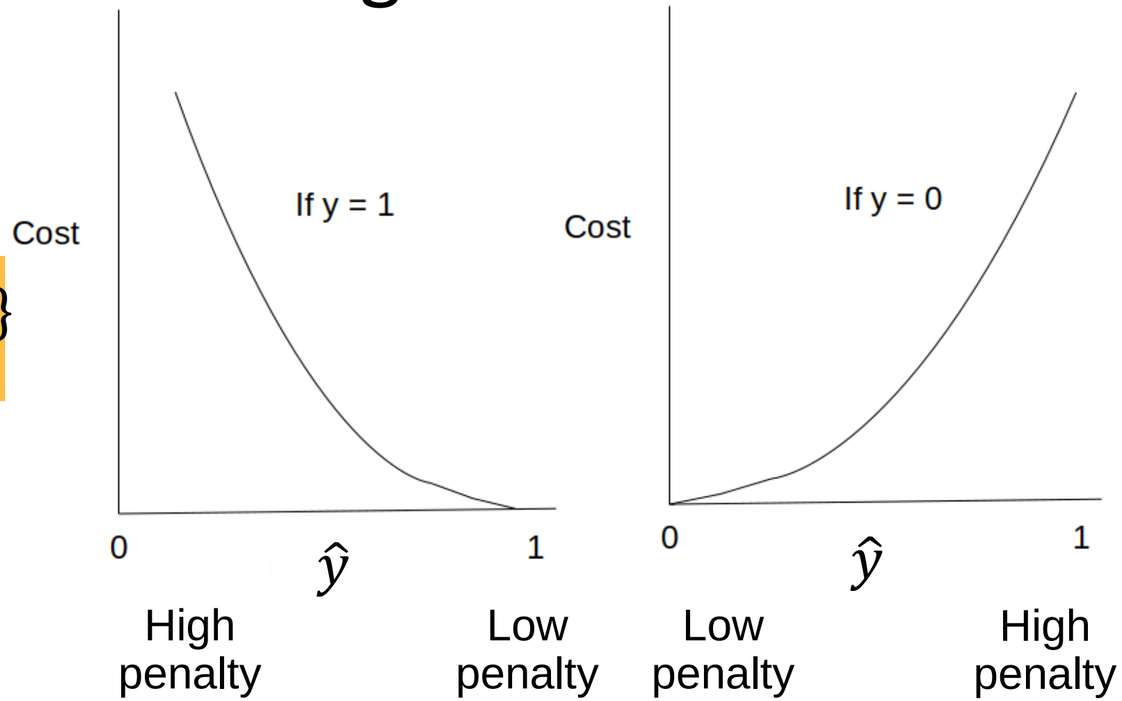
- In Classification, Output Layer uses Sigmoid activation (Binary Classification) or Softmax activation (Multi-class Classification)
- MSE cost function with sigmoid activation function, is non-convex (many local minima)

=> Use Cross-Entropy Loss (or Log Loss) to determine model weights



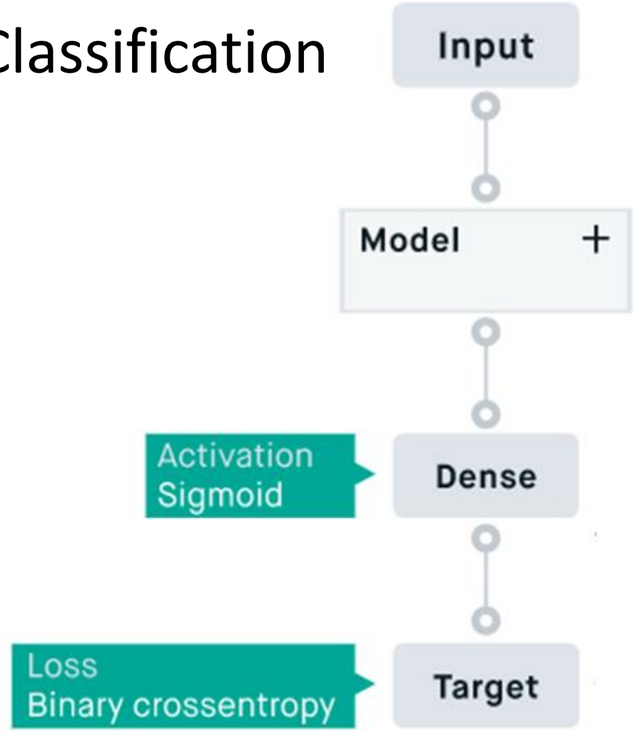
Cross-entropy loss function or "log loss"

- $Loss(\underline{w}) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1-\hat{y}) & if\ y = 0 \end{cases}$
- $Loss(\underline{w}) = \frac{1}{N}\sum_{i=1}^{N}\{-y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)\}$
- $\underline{w}^* = \underset{\underline{w}}{\operatorname{argmin}}\, Loss(\underline{w})$
- $Loss(\underline{w})$ is a value between 0 and 1
- Cross-entropy loss increases as the predicted probability diverges from the actual label



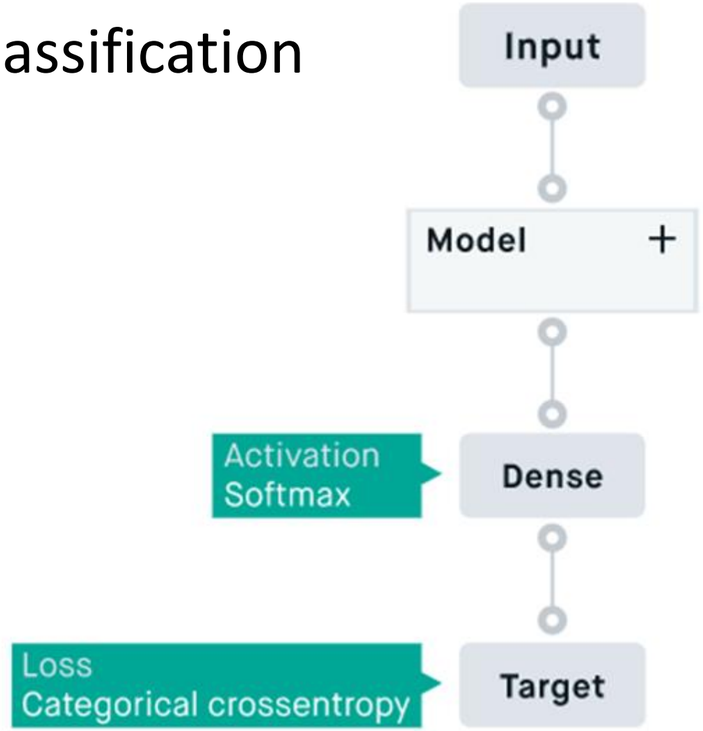
Binary Cross-Entropy Loss Function

- Binary Classification



Categorical Cross-Entropy Loss Function

- Multi-class Classification



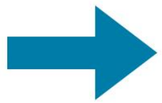
Cross-Entropy Loss Function

Used with softmax activation for multi-class classification

- Categorical Cross Entropy Loss Function:
  - Requires one-hot encoding
- Sparse Categorical Cross Entropy Loss Function:
  - Used when number of classes is too large (eg 1000)
  - Avoids one-hot encoding, which requires large memory

One-hot Encoding

Color
Red
Red
Yellow
Green
Yellow



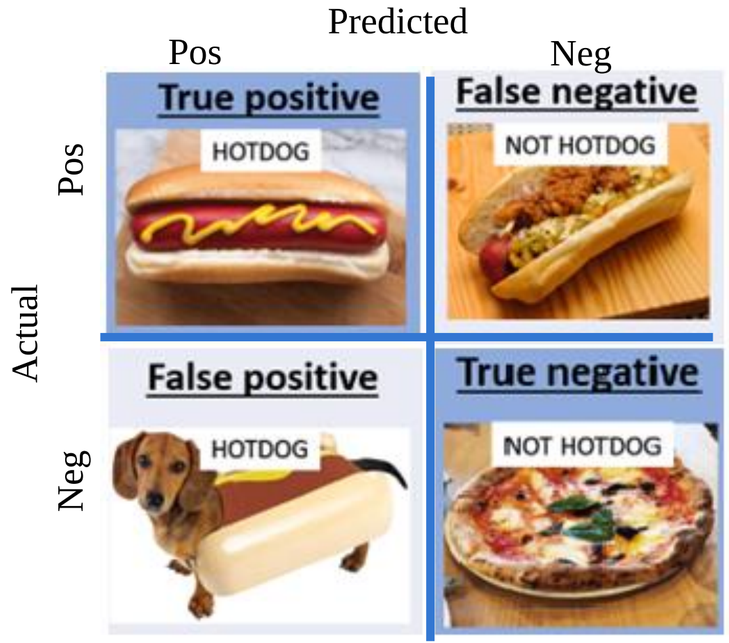
Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

Classification: Model Performance

- Confusion Matrix (Good Model => diagonal elements are large, and others are small)

	Predicted Positive	Predicted Negative
Actual as Positive	True Positive	False Negative
Actual as Negative	False Positive	True Negative

- **True Positive**: Actual value was Positive, the Model predicts Positive.
- **True Negative**: Actual value was Negative, the Model predicts Negative.
- **False Negative**: Actual value was Positive, the Model predicts Negative.
- **False Positive**: Actual value was Negative, the Model predicts Positive.



Classification: Model Performance

- Confusion Matrix (Good Model => diagonal elements are large, and others are small)

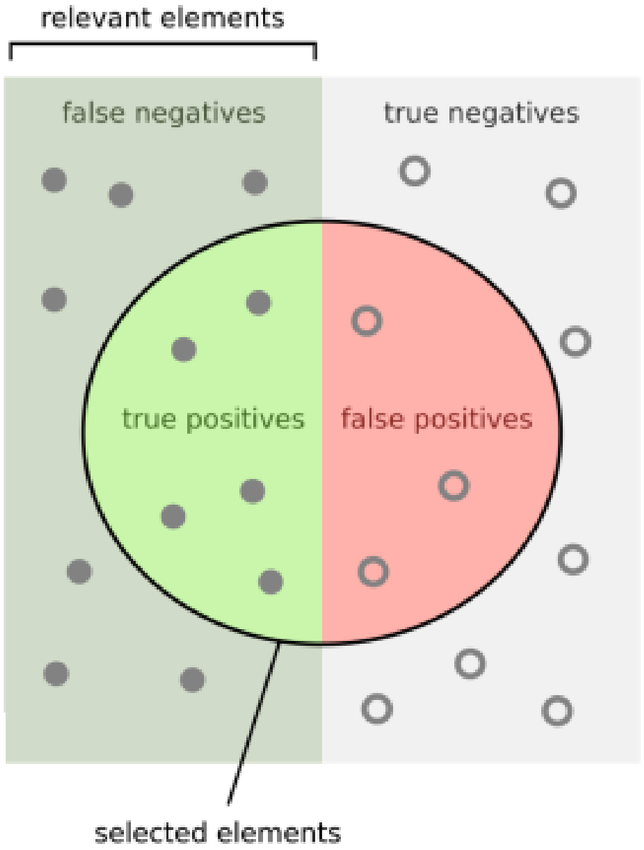
$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative} = \frac{True\ Predictions}{All\ Predictions}$$

$$Sensitivity = Recall = True\ Positive\ Rate = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{Positive\ Class}$$

$$Specificity = \frac{True\ Negative}{True\ Negative + False\ Positive} = \frac{True\ Negative}{Negative\ Class}$$

$$Precision = Positive\ Predictive\ Value = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{True\ Positive}{Predicted\ as\ Positive}$$

$$False\ Positive\ Rate = 1 - Specificity = \frac{False\ Positive}{Faslse\ Positive + True\ Negative} = \frac{False\ Positive}{Negative\ Class}$$



Classification: Model Performance

- Confusion Matrix (Good Model => diagonal elements are large, and others are small)

	Predicted Positive	Predicted Negative
Actual as Positive	True Positive	False Negative
Actual as Negative	False Positive	True Negative

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative} = \frac{True\ Predictions}{All\ Predictions}$$

$$Sensitivity = Recall = True\ Positive\ Rate = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{Positive\ Class}$$

$$Specificity = \frac{True\ Negative}{True\ Negative + False\ Positive} = \frac{True\ Negative}{Negative\ Class}$$

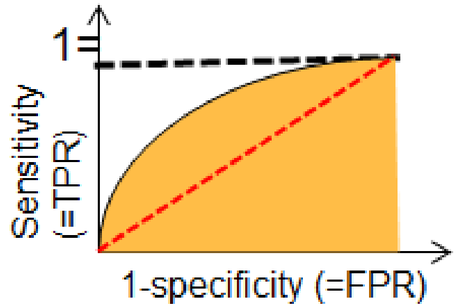
$$Precision = Positive\ Predictive\ Value = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{True\ Positive}{Predicted\ as\ Positive}$$

$$False\ Positive\ Rate = 1 - Specificity = \frac{False\ Positive}{Faslse\ Positive + True\ Negative} = \frac{False\ Positive}{Negative\ Class}$$

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Area Under Curve (AUC) of ROC

A measure of how fast ROC rises and how soon a good balance point between specificity and sensitivity is achieved



Example: Given 30 human photographs, a computer predicts 19 to be male, 11 to be female. Among the 19 male predictions, 3 predictions are not correct. Among the 11 female predictions, 1 prediction is not correct

Calculate

- Accuracy
- Precision
- Recall
- F1 Score

Example: Given 30 human photographs, a computer predicts 19 to be male, 11 to be female. Among the 19 male predictions, 3 predictions are not correct. Among the 11 female predictions, 1 prediction is not correct

Calculate

- Accuracy
- Precision
- Recall
- F1 Score

Confusion Matrix

		Predicted Class	
		Male	Female
Actual Class	Male	a = TP = 16	b = FN = 1
	Female	c = FP = 3	d = TN = 10

		Predicted Class	
Actual Class		Male	Female
	Male	a = TP = 16	b = FN = 1
	Female	c = FP = 3	d = TN = 10

Calculate

- Accuracy = (16+10)/(16+3+1+10) = 0.867
- Precision = 16 / (16+3) = 0.842
- Recall = 16 / (16+1) = 0.941
- F1 Score = 2(0.842*0.941)/(0.842+0.941) = 0.889

Module I: Introduction to Generative AI and Fundamental Study

Module I (20%)

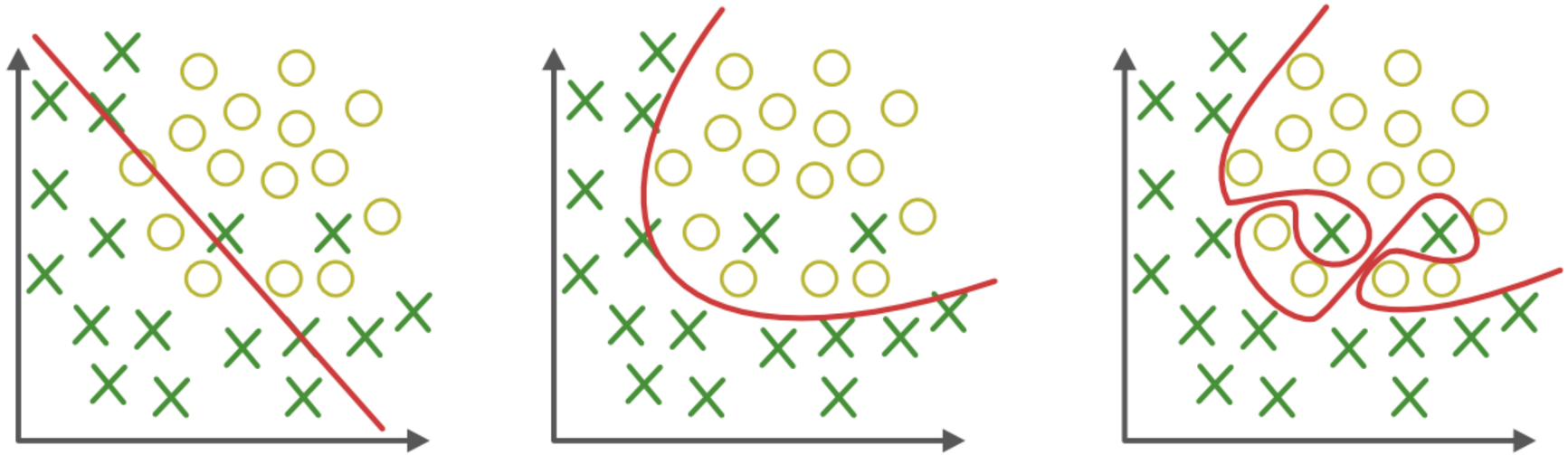
Basic introduction to Generative AI, Applications of Generative AI in various fields, Perceptron and Multilayer Perceptron (MLP), Gradient Descent, Backpropagation algorithm, Loss functions, Understanding Bias and Variance

Which Model is Better?

Learning is an ill posed problem: Multiple possible hypothesis

Model selected after evaluating several candidate models

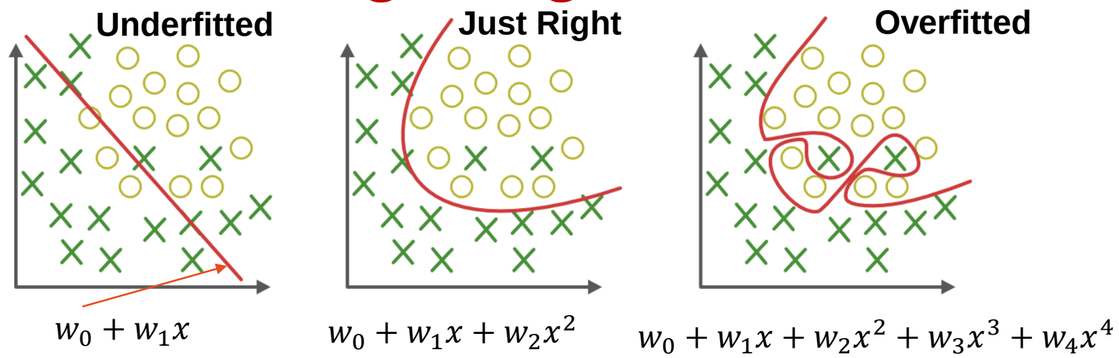
Example of Binary Classification

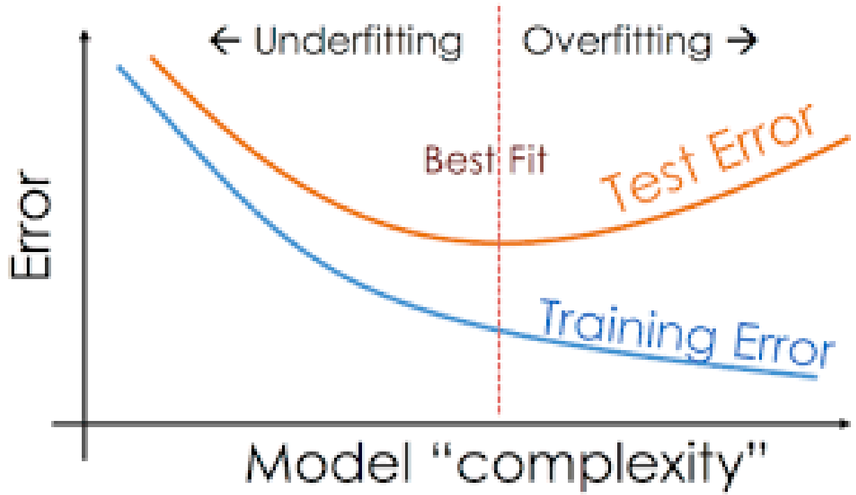


Generalization

- The real aim of supervised learning is to do well on test data that is not known during learning
- Choosing the values for the parameters that minimize the loss function on the training data is not necessarily the best policy
- Generalization refers to How well the model trained on the training data predicts the correct output for new instances
  - We want the learning machine to model the true regularities in the data and to ignore the noise in the data.
  - But the learning machine does not know which regularities are real and which are accidental quirks of the particular set of training examples we happen to pick
- So how can we be sure that the machine will generalize correctly to new data?

Trading off goodness of fit against complexity of the model

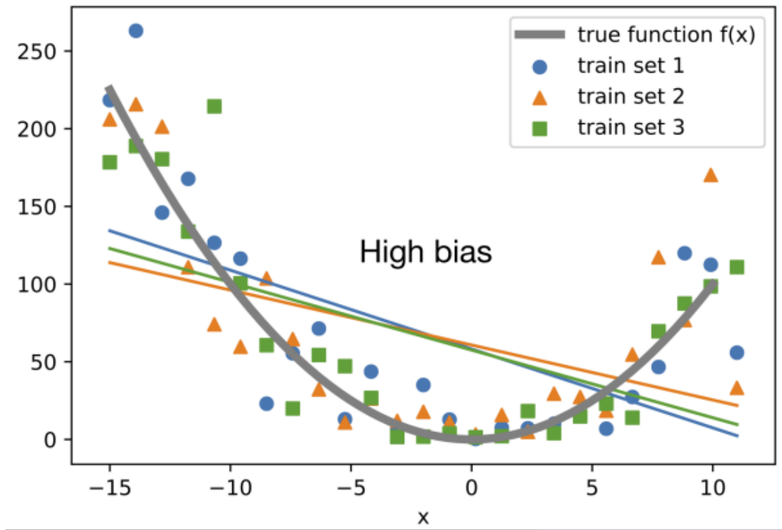




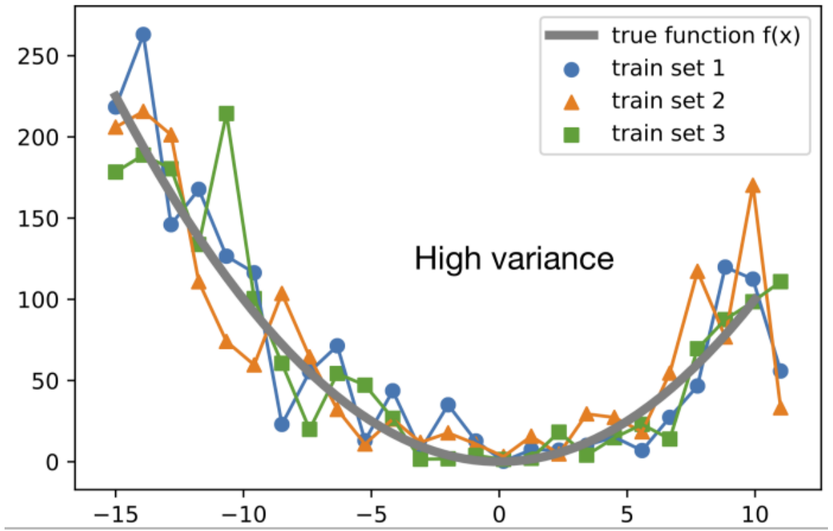
Simple model has less parameters to be learned (Low complexity, low capacity)	**Complex model** has more parameters to be learned (High complexity, High capacity)
Model may **Underfit**, it may not capture underlying trend of the data	Model may **Overfit**, it may start learning from noise and inaccurate data entries
Higher error for training data, may give high error for validation data also	Lower error for training data, may give higher error for validation data
High Bias, Low Variance	Low Bias, High Variance

Bias vs Variance

- **Bias** of a Model: Underlying assumptions to make learning possible. Simpler model=>More assumption=> High Bias



- **Variance** of a Model: Variability of model for given data points, Model with high variance pays a lot of attention to training data, may end up memorizing data rather than learning from it



Bias vs Variance

- **Bias** of a Model: Underlying assumptions to make learning possible. Simpler model=>More assumption=> High Bias
- **Variance** of a Model: Variability of model for given data points, Model with high variance pays a lot of attention to training data, may end up memorizing data rather than learning from it