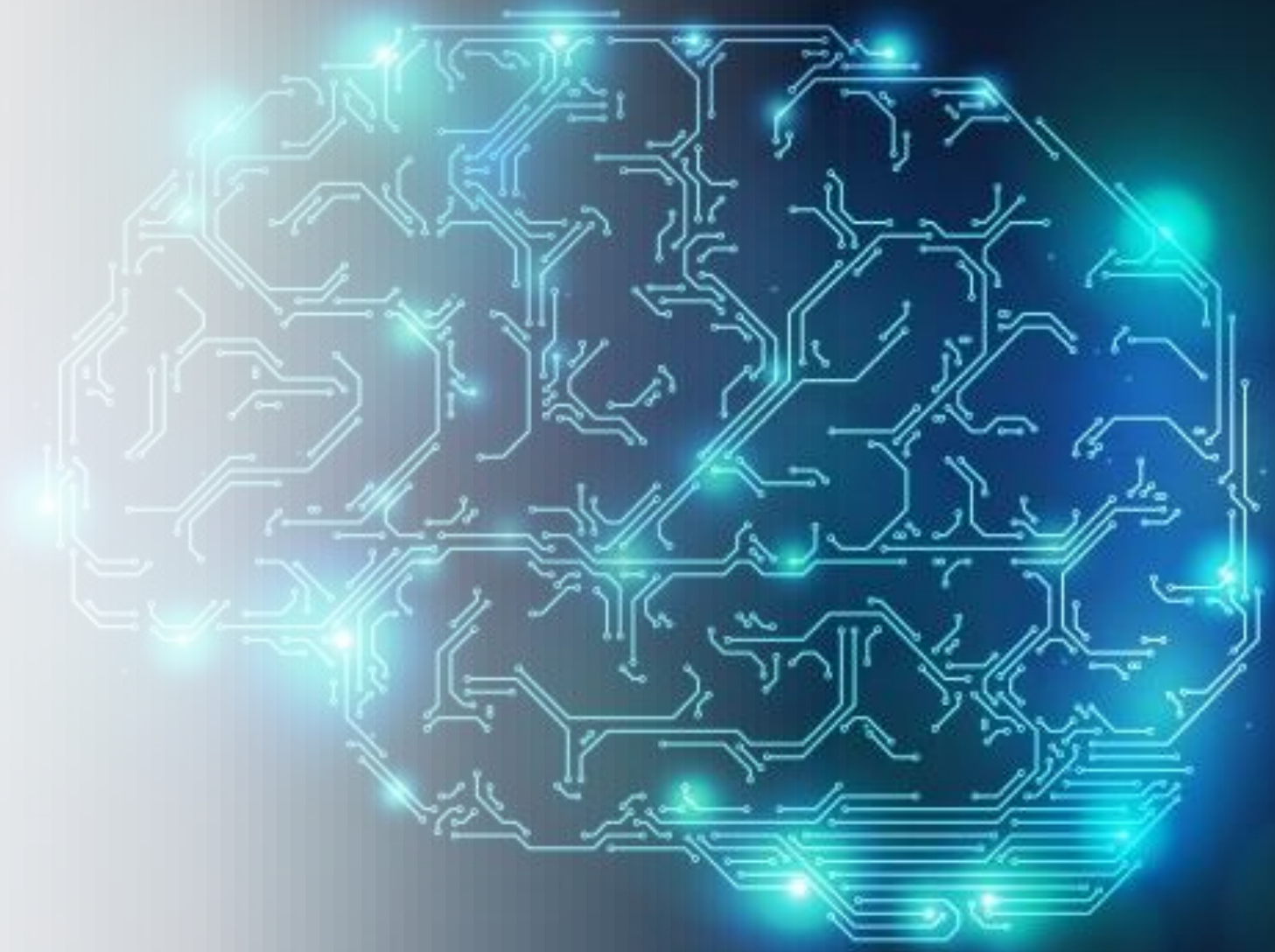
NCRM NATIONAL CENTRE FOR RESEARCH METHODS

UNIVERSITY OF EXETER
Q-STEP CENTRE

Computational Communication Methods Spring School

Week 2: Day 2



Deep Learning & AI

PIKAKSHI MANCHANDA, PHD

LECTURER IN COMPUTER SCIENCE

@itsPikakshi

p.manchanda@exeter.ac.uk

Overview of today's session

- Deep Learning – some applications
- What is AI, ML and DL?
- Deep Learning and its popularity
- Neural Networks
- NN Topologies
- Understanding the Math
- DL Frameworks at a glance: TensorFlow

https://github.com/Pikakshi/Introduction-to-Deep-Learning

Frequently Bought Together

Price for all three: $74.20

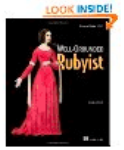
Add all three to Cart | Add all three to Wish List

Show availability and shipping details

- **This item:** Beginning Ruby: From Novice to Professional (Expert's Voice in Open Source) by Peter Cooper Paperback $27.78
- Learn to Program, Second Edition (The Facets of Ruby Series) by Chris Pine Paperback $16.94
- Ruby on Rails Tutorial: Learn Web Development with Rails (2nd Edition) (Addison-Wesley Professional Ruby … by Michael Hartl Paperback $29.48

Customers Who Bought This Item Also Bought

Learn to Program, Second Edition (The Facets of…
Chris Pine
★★★★½ 42
Paperback
$16.94 Prime



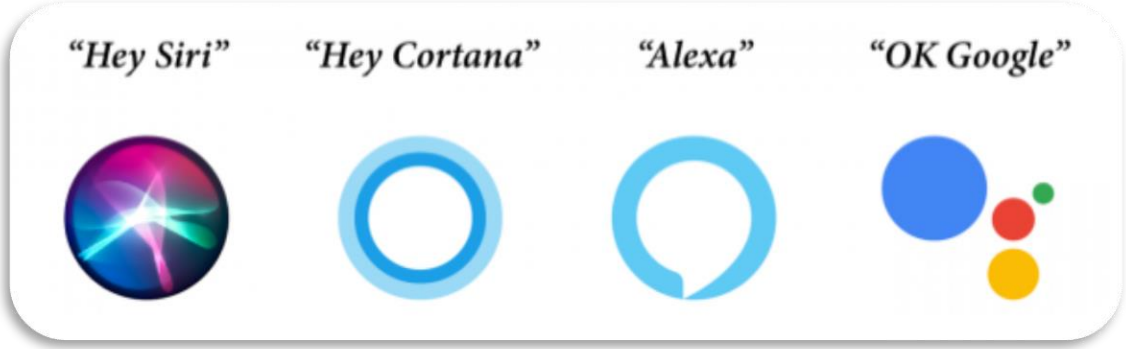
The Well-Grounded Rubyist
› David A. Black
★★★★★ 39
Paperback
$32.49 Prime

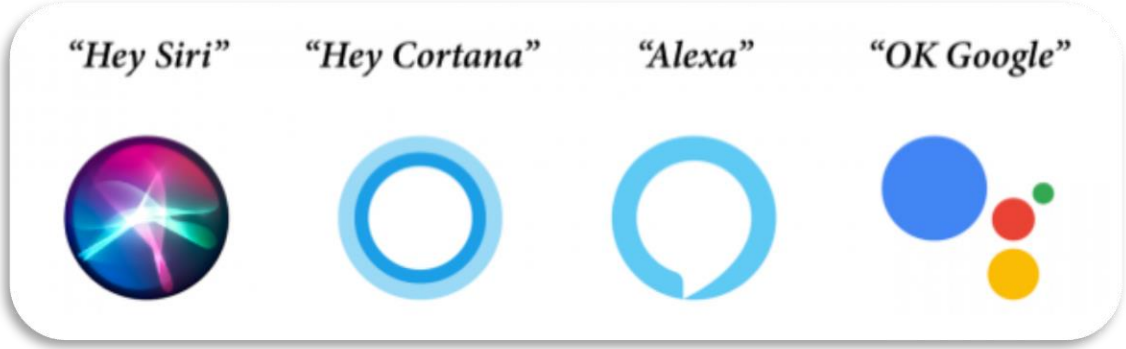
Ruby on Rails Tutorial: Learn Web Development…
› Michael Hartl
★★★★☆ 70
Paperback
$29.48 Prime

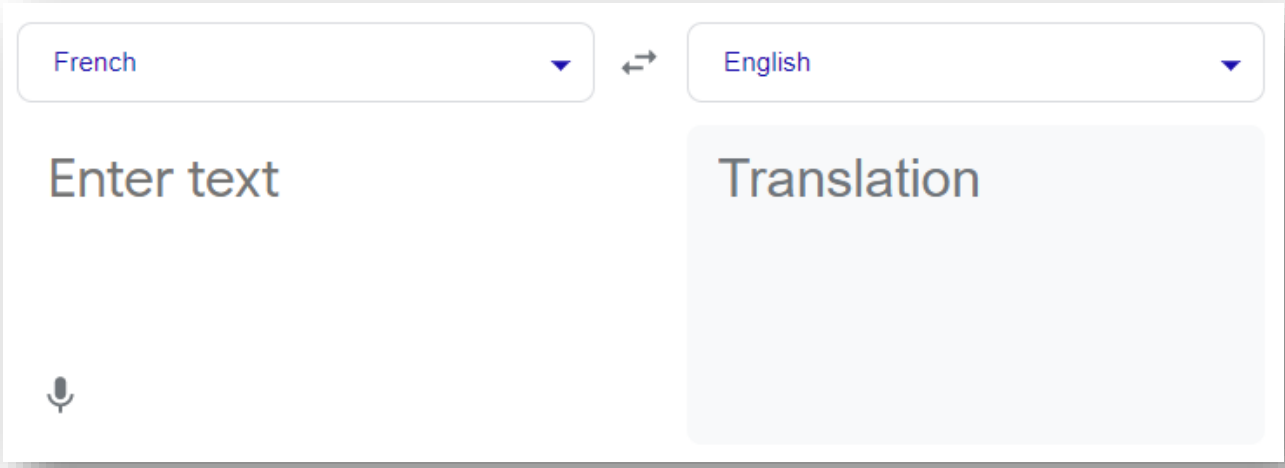
The Ruby Programming Language
› David Flanagan
★★★★½ 74
Paperback
$26.35 Prime

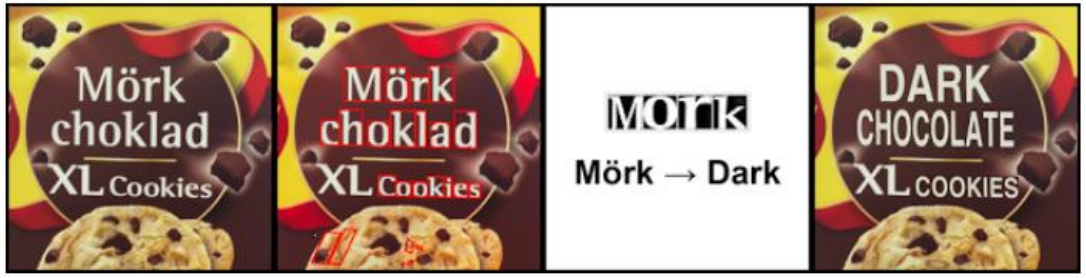
The Well-Grounded Rubyist
› David A. Black
★★★★½ 19
#1 Best Seller in Ruby Programming Computer
Paperback
$29.67 Prime

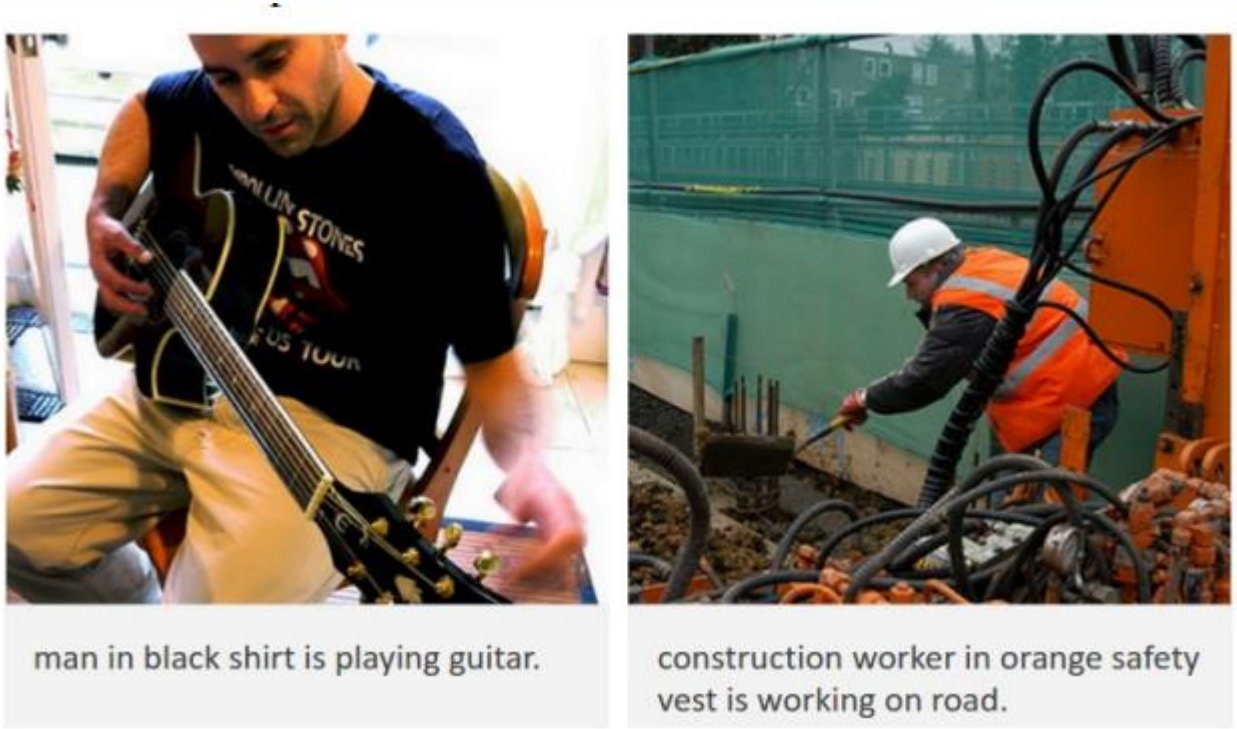

STOP
12


"Hey Siri"
"Hey Cortana"
"Alexa"
"OK Google"

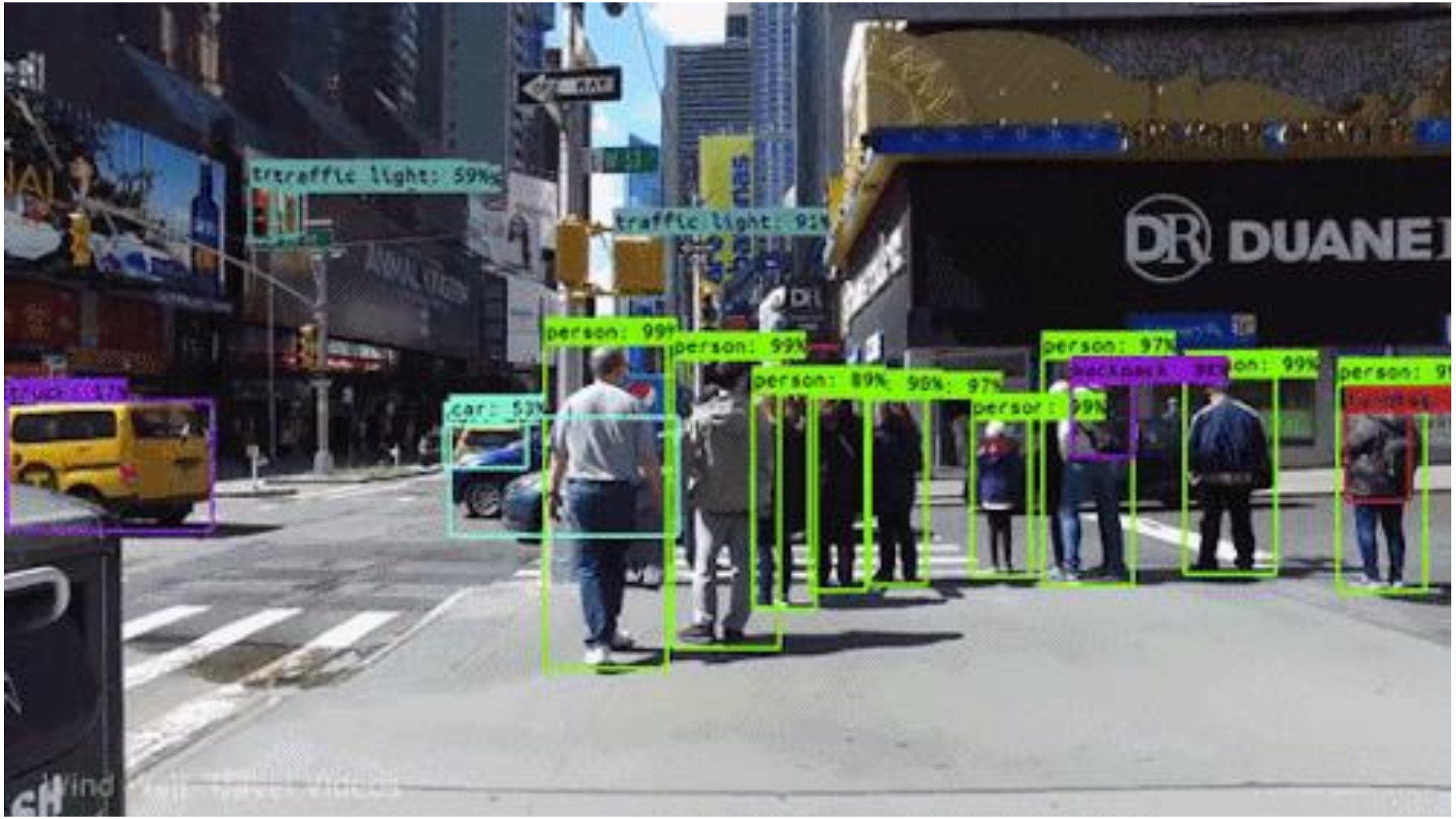






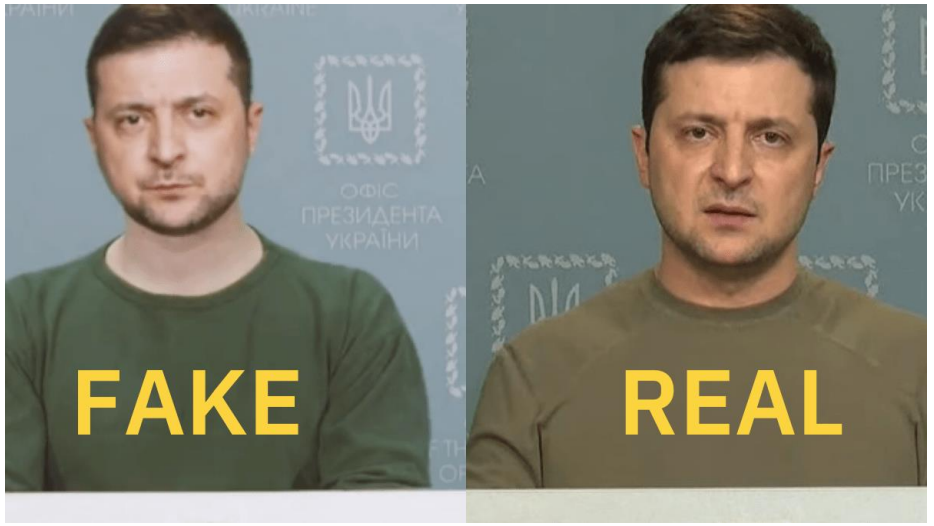


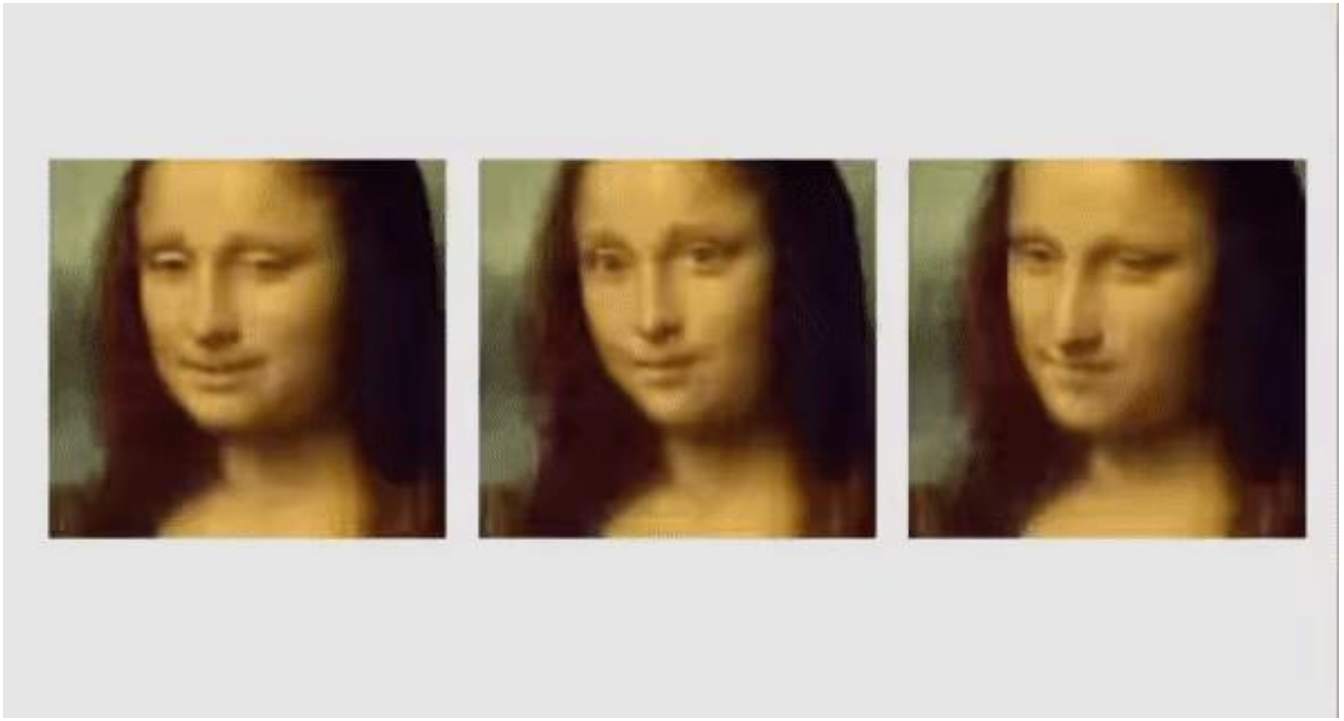
Source: Karpathy et al., 2015.



YOLO: Real-Time Object Detection

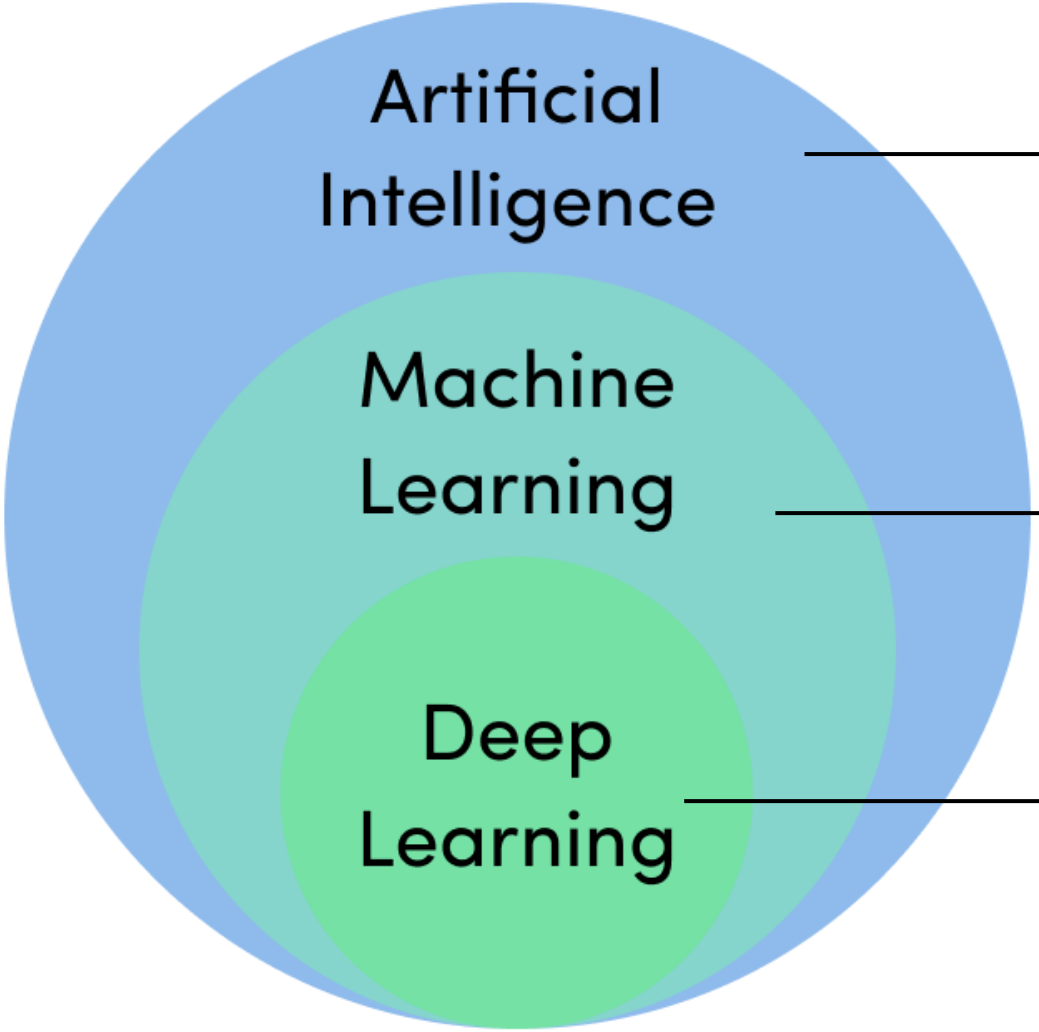
Source: Redmon et al., 2016


ОФІС
ПРЕЗИДЕНТА
УКРАЇНИ
FAKE
REAL



Buzzwords

- Artificial Intelligence
- Machine Learning
- Deep Learning
- Artificial Neural Networks
- TensorFlow
- Supervised vs Unsupervised Learning
- Convolutional Neural Networks
- LSTM
- Recurrent Models
- Feature Selection
- Gradient Descent
- Loss Optimization
- Backpropagation
- Explainable AI



Any technique that enables computers to mimic human behaviour.

Ability to learn without being explicitly programmed

Extract patterns from data using neural networks, with minimal human intervention

https://github.com/Pikakshi/Advanced_NLP_with_ML

Machine Learning

Supervised Learning

- Task-driven learning.
- Making predictions using data.
- Given a dataset with ‘right answers’, an algorithm learns to produce predictions on never-before-seen data.
- Types: Regression & Classification

Unsupervised Learning

- Data-driven learning.
- Given an unlabeled dataset x_1, x_2, …, x_n : an algorithm finds hidden structures in the data.
- Types: Clustering, Association Rule Mining

Reinforcement Learning

- Learning what to do and how to map situations to actions.

Deep Learning

Why do we care about Deep Learning?

- Traditional ML algo's need a lot of domain expertise.
- Human intervention is needed regularly.
- ML algorithms are capable of only doing what they have been designed for.

Source: The AI Index 2022 Annual Report, Zhang et al., 2022

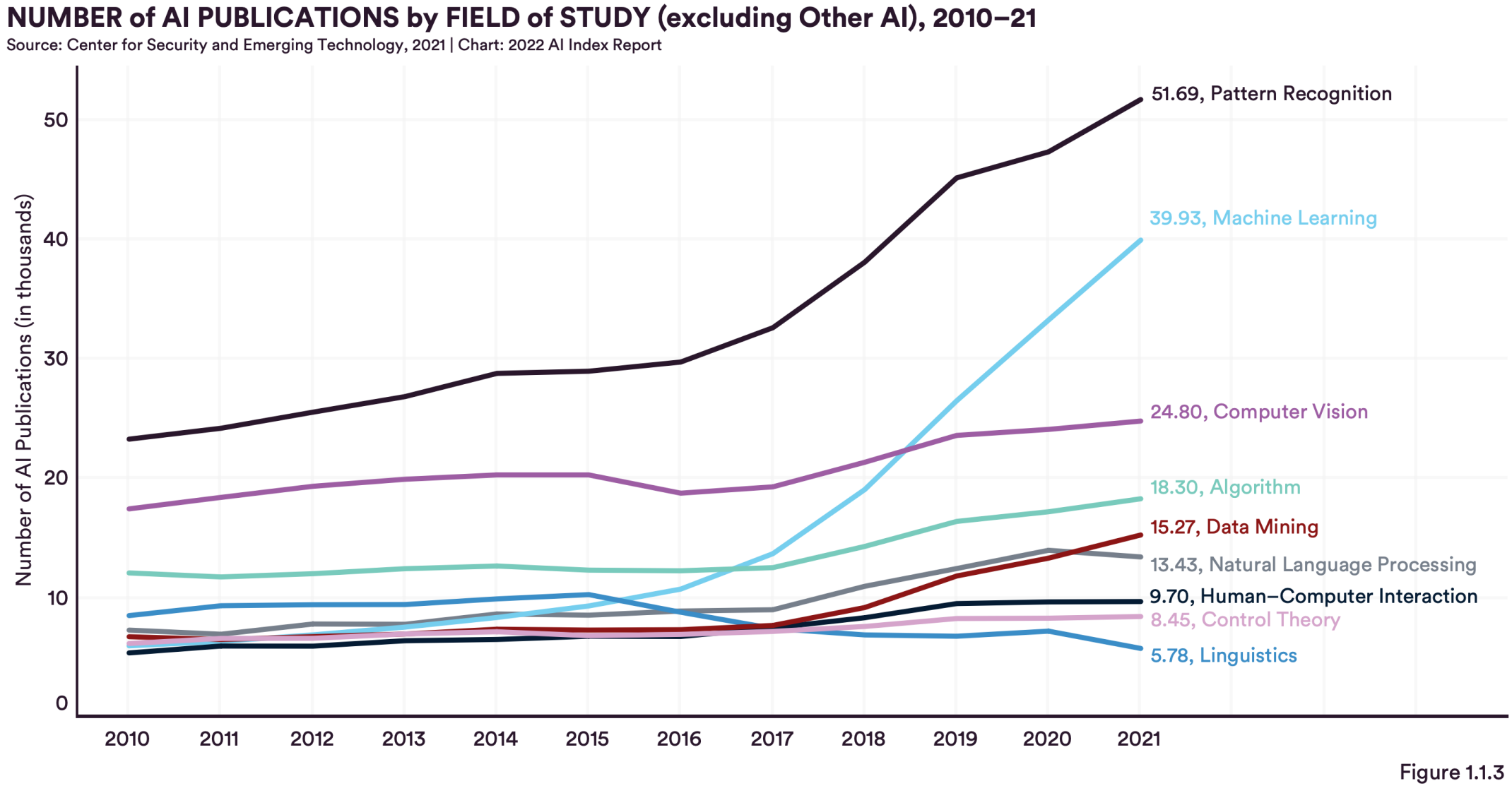


Figure 1.1.3

Academic Publications since 2011

Machine Learning:
~1.7M papers

Deep Learning:
~700,000 papers

Machine Learning:
49.2k questions

Deep Learning:
23.7k questions

AI JOB POSTINGS (% of ALL JOB POSTINGS) in the UNITED STATES by SKILL CLUSTER, 2010–21

Source: Emsi Burning Glass, 2021 | Chart: 2022 AI Index Report

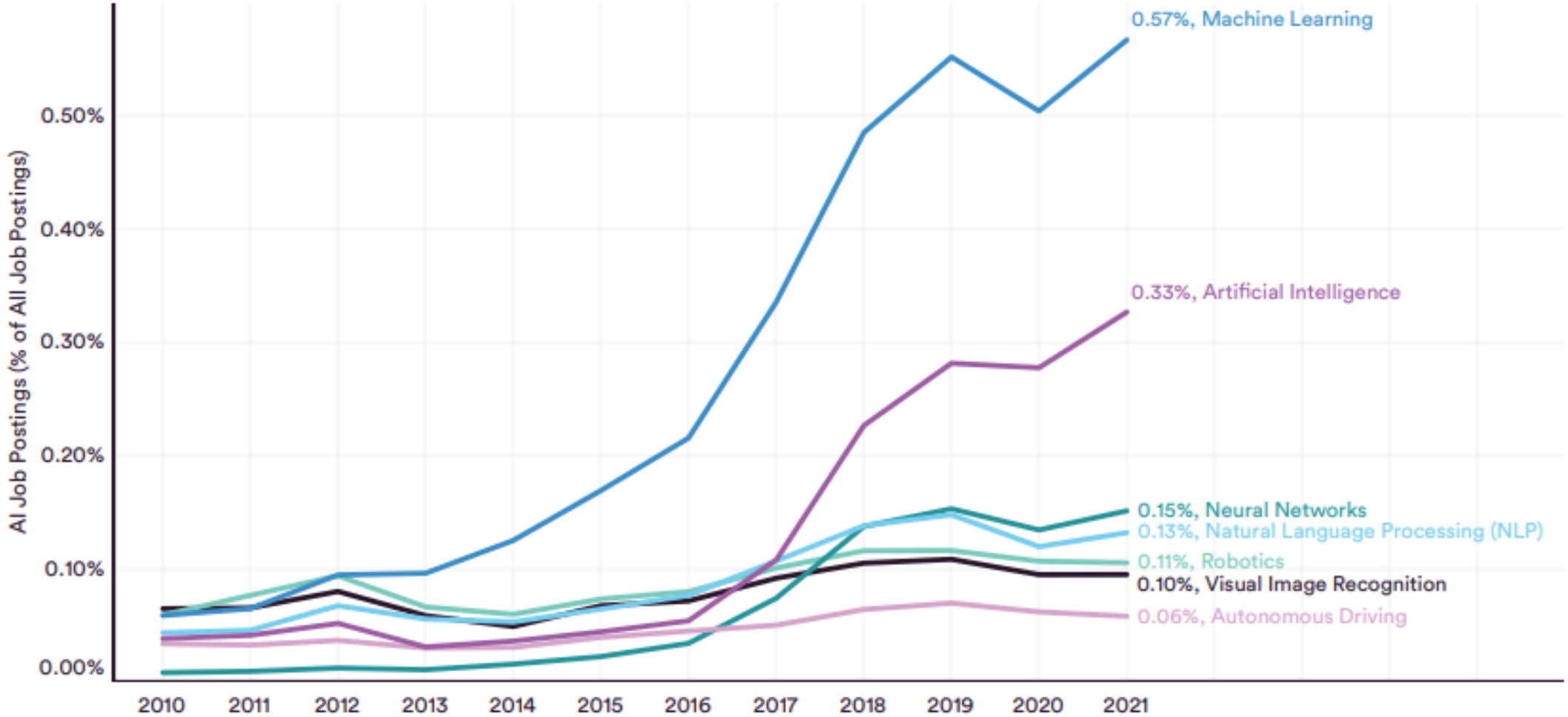


Figure 4.1.4

GLOBAL CORPORATE INVESTMENT in AI by INVESTMENT ACTIVITY, 2013–21

Source: NetBase Quid, 2021 | Chart: 2022 AI Index Report

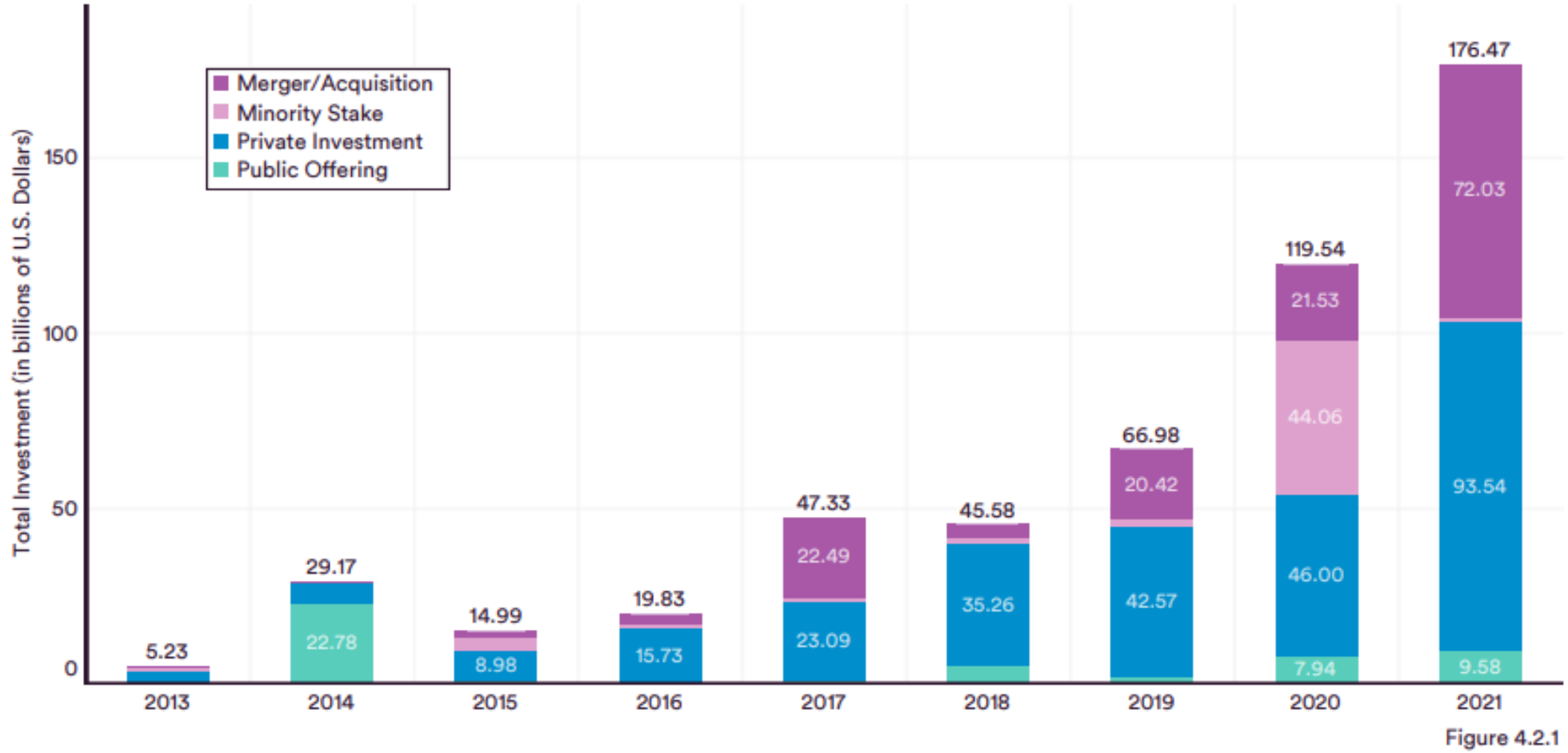
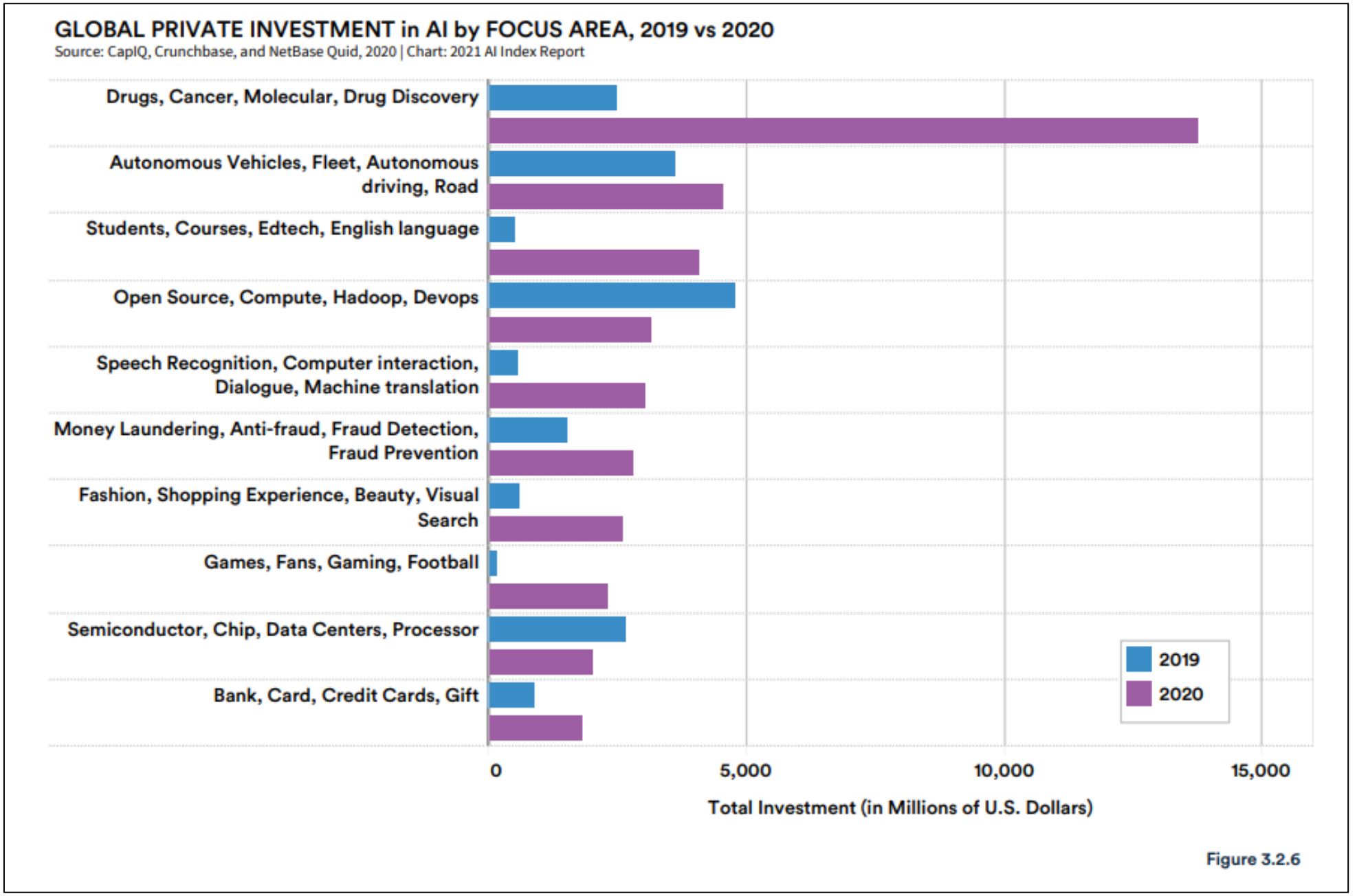
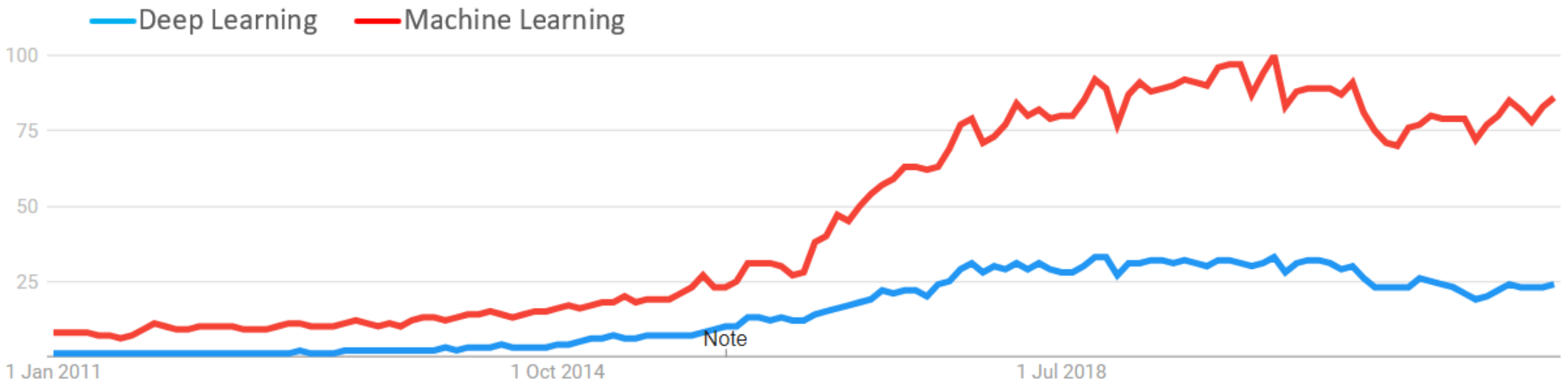
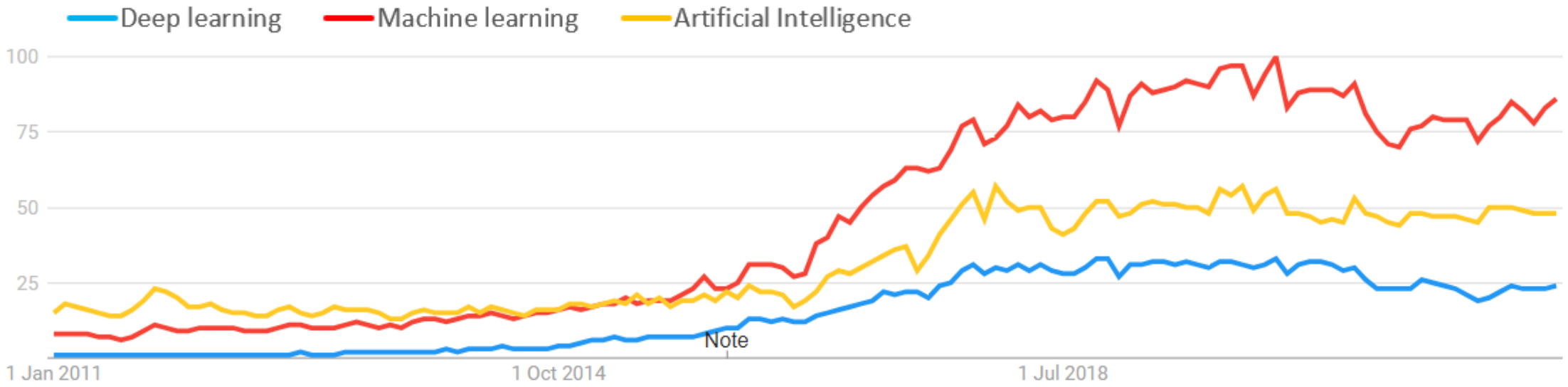


Figure 4.2.1


GLOBAL PRIVATE INVESTMENT in AI by FOCUS AREA, 2019 vs 2020
Source: CapIQ, Crunchbase, and NetBase Quid, 2020 | Chart: 2021 AI Index Report
Drugs, Cancer, Molecular, Drug Discovery
Autonomous Vehicles, Fleet, Autonomous driving, Road
Students, Courses, Edtech, English language
Open Source, Compute, Hadoop, Devops
Speech Recognition, Computer interaction, Dialogue, Machine translation
Money Laundering, Anti-fraud, Fraud Detection, Fraud Prevention
Fashion, Shopping Experience, Beauty, Visual Search
Games, Fans, Gaming, Football
Semiconductor, Chip, Data Centers, Processor
Bank, Card, Credit Cards, Gift
2019
2020
0
5,000
10,000
15,000
Total Investment (in Millions of U.S. Dollars)
Figure 3.2.6

Source: Google Trends



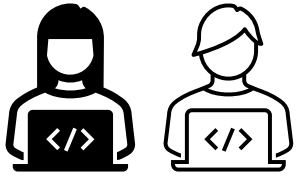


Deep Learning – Why now?

- 1943: *Walter* Pitts and *Warren McCulloch* published a paper titled “A Logical Calculus of the Ideas Immanent in Nervous Activity” which shows the mathematical model of biological neuron.

Deep Learning – Why now?

Abundance of Data



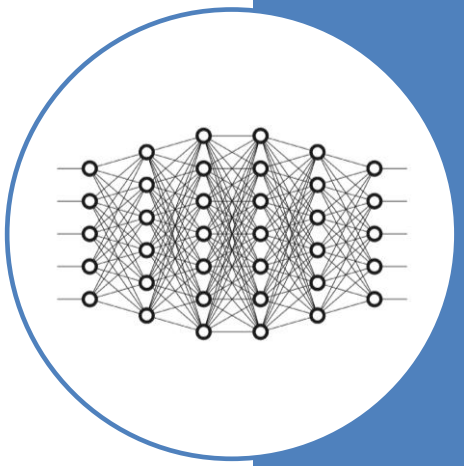
Availability of open source frameworks

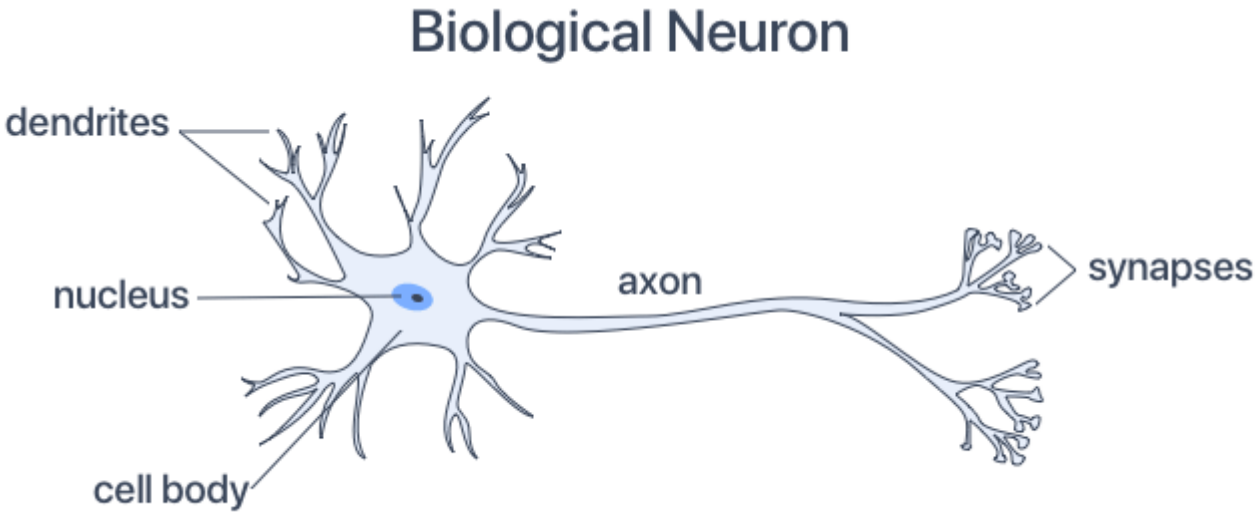
Keras PYTORCH TensorFlow

Availability of hardware & architecture

Artificial Neural Networks

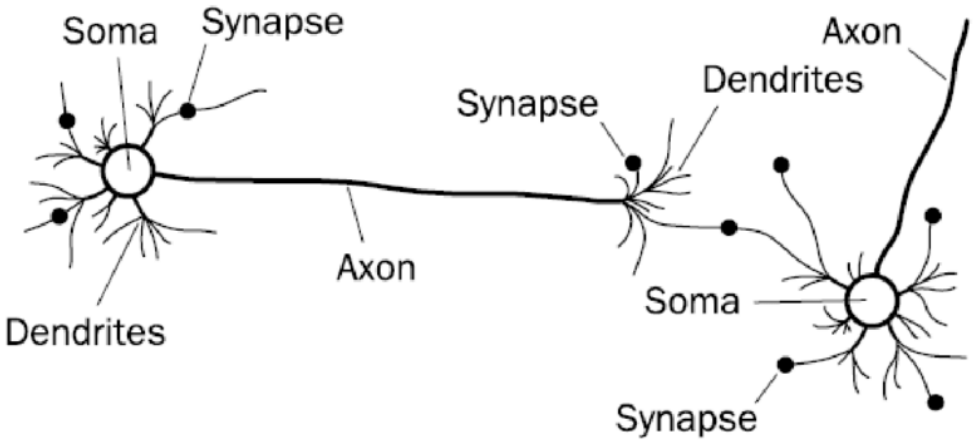
- Form the backbone of Deep Learning.
- Inspired by the structure of the human brain.
- How do they work?
  - ✓ Take data as *input* through input nodes.
  - ✓ *Train* themselves to understand patterns in the data.
  - ✓ Predict *outputs*.
  - ✓ Evaluate model.
  - ✓ Optimise model's performance.

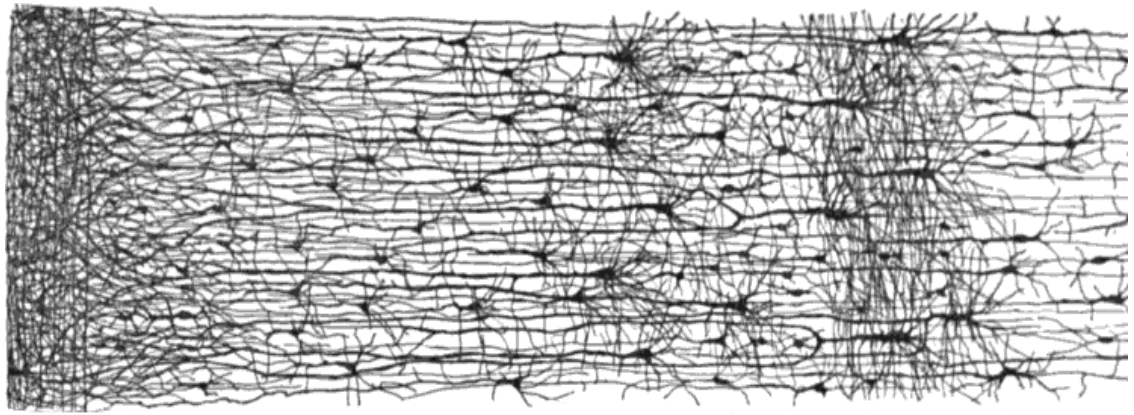




Biological Neural Network

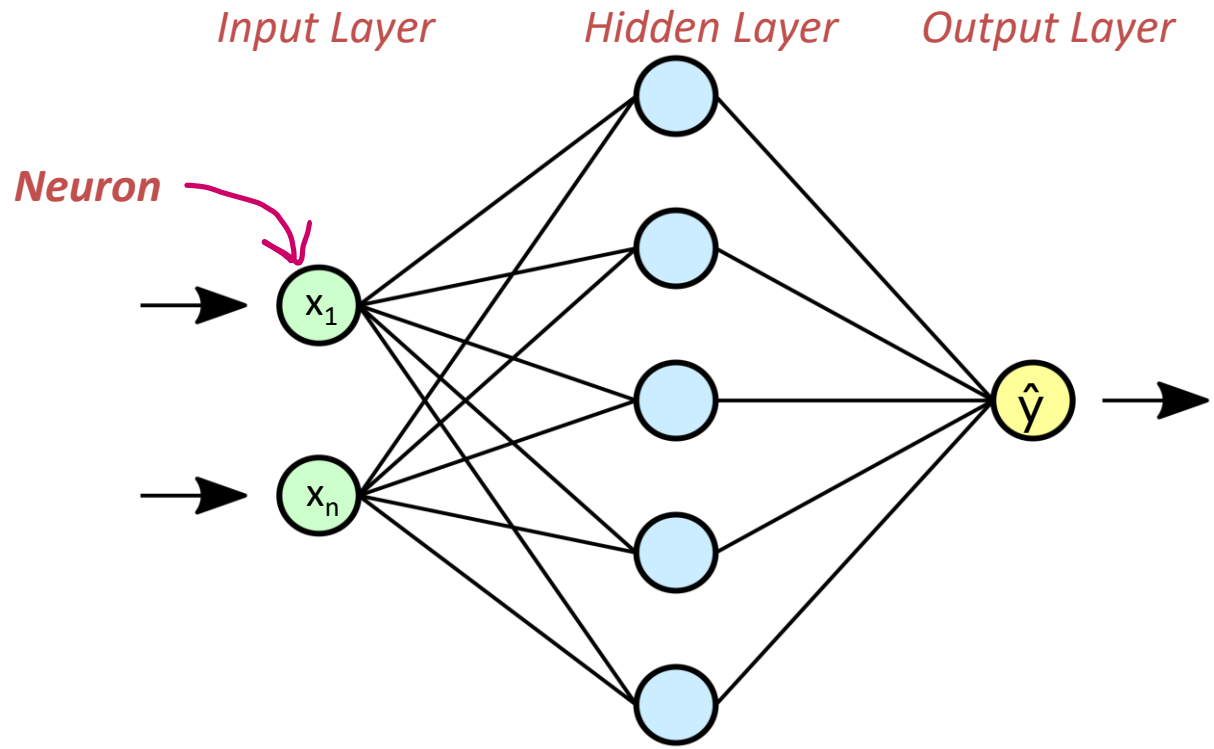
- Inputs: dendrites
- *Input nodes*: cell nucleus
- *Outputs*: axon





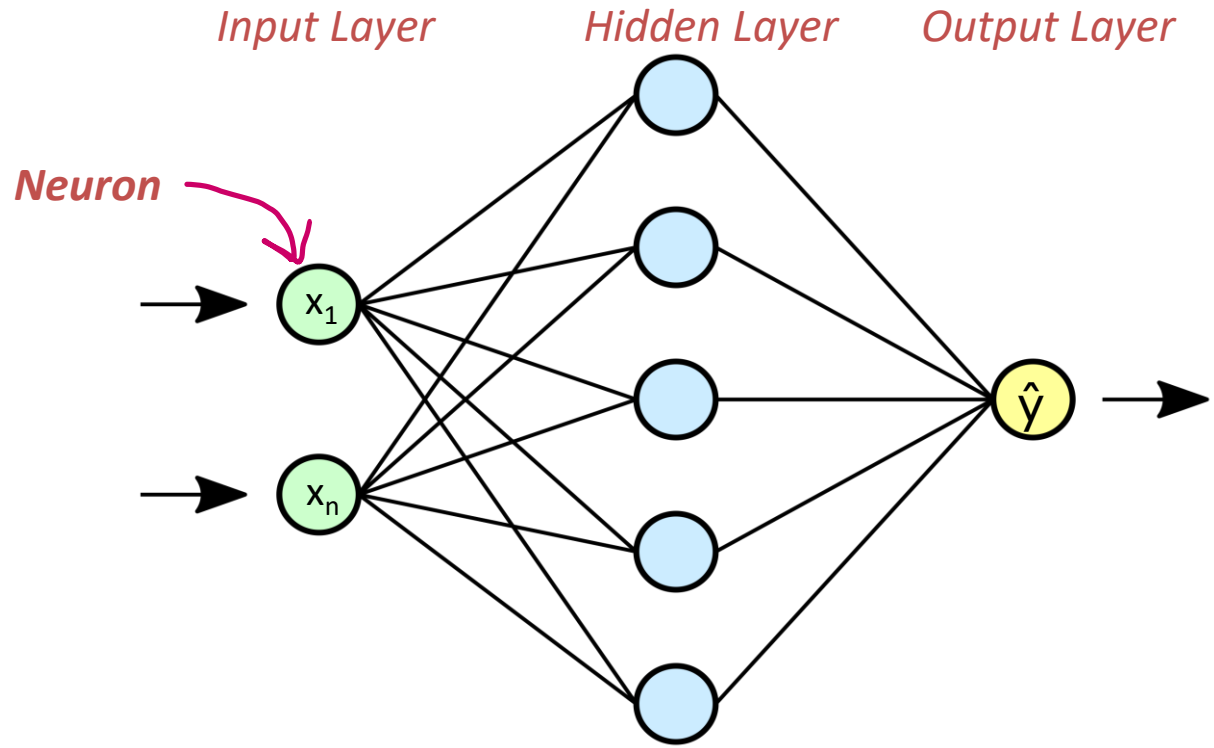
Artificial Neural Network

- Layers of interconnected nodes.
- Information propagates through three components:
  - Input layer
  - Hidden layer(s)
  - Output layer



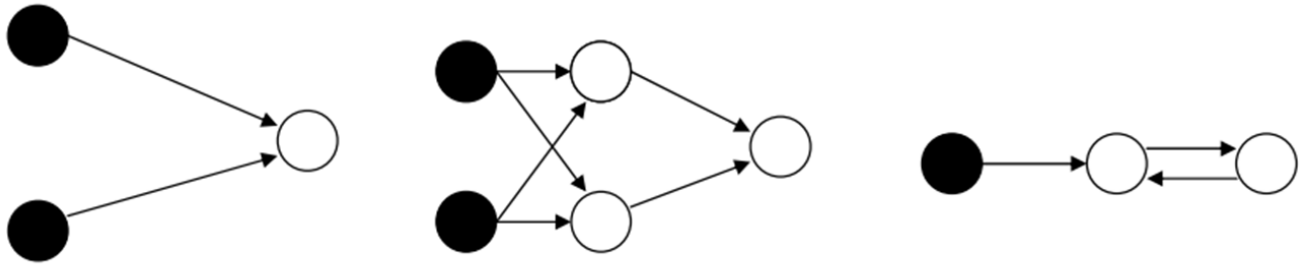
Artificial Neural Network

- Layers of interconnected nodes.
- Information propagates through three components:
  - Input layer
  - Hidden layer(s)
  - Output layer



ANN Architectures:

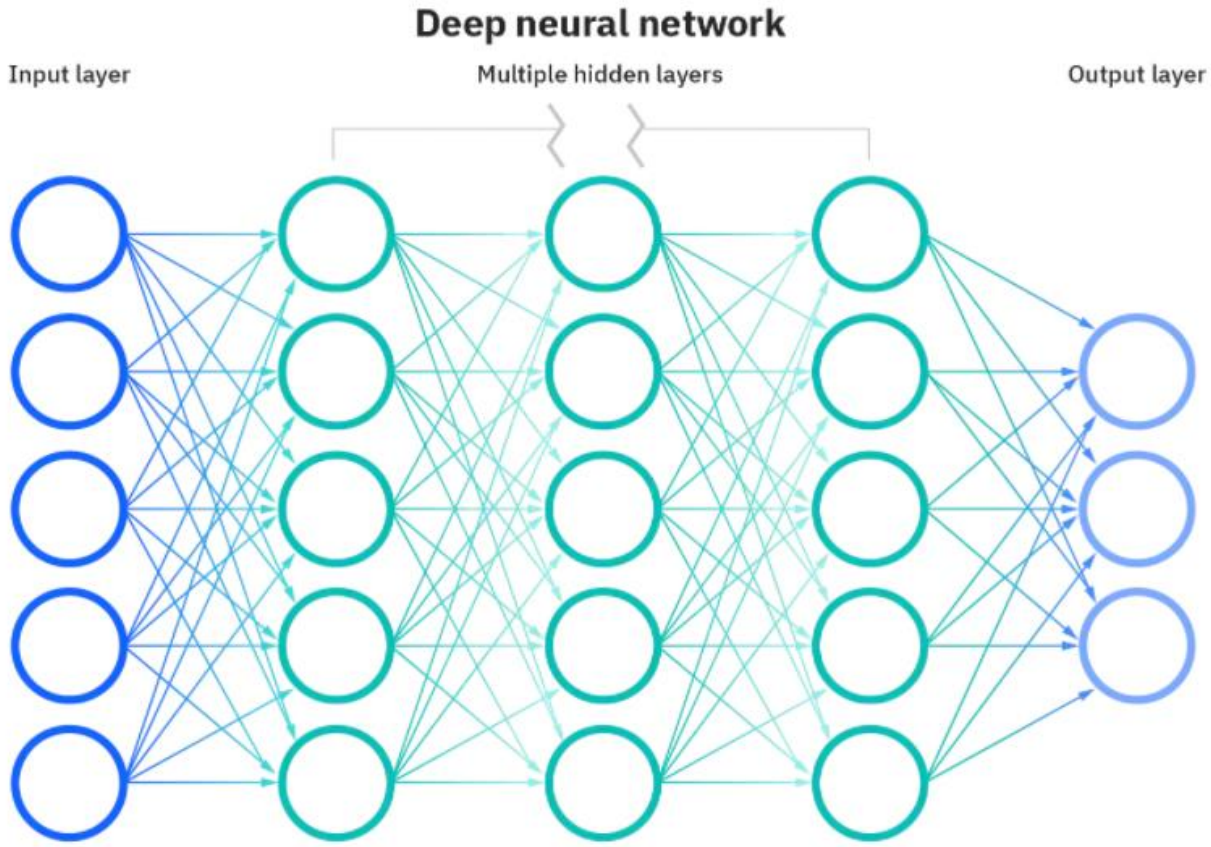
- Single-Layer Feed-Forward NN
- Multi-Layer Feed-Forward NN
- Recurrent NN



Single-Layer Feed-Forward NN

Multi-Layer Feed-Forward NN

Recurrent NN (RNN)

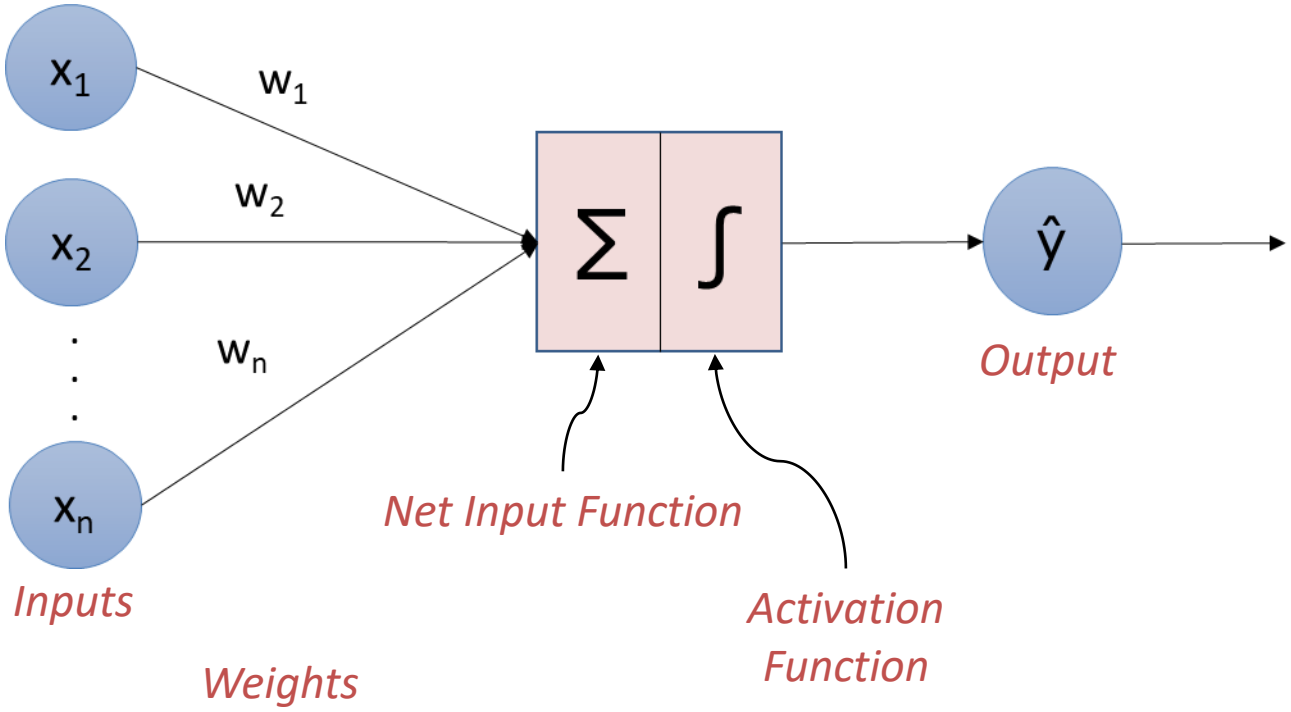

Deep neural network
Input layer
Multiple hidden layers
Output layer

Single Layer Perceptron

Perceptron

(Single layer perceptron)

- Frank Rosenblatt, 1957, 1958.
- Oldest type of neural network.
- Used for binary classification.
- Propagation of info from the input layer to the output layer.

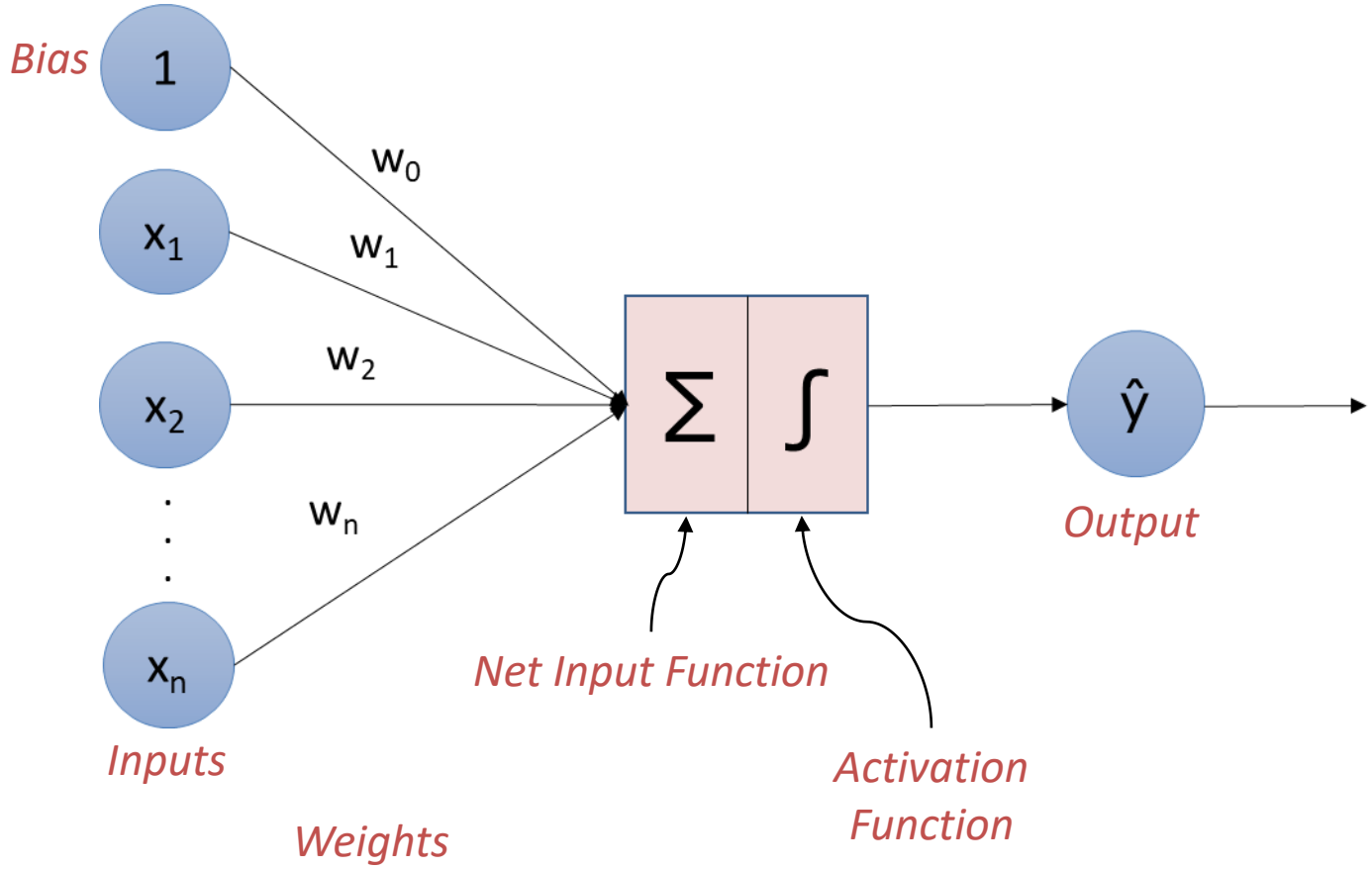


Weight: tells how significant a given neuron is. Higher the value, the more important the neuron is in the relationship.

Perceptron

(Single layer perceptron)

- Frank Rosenblatt, 1957, 1958.
- Oldest type of neural network.
- Used for binary classification.
- Propagation of info from the input layer to the output layer.



Weight: tells how significant a given neuron is. Higher the value, the more important the neuron is in the relationship.

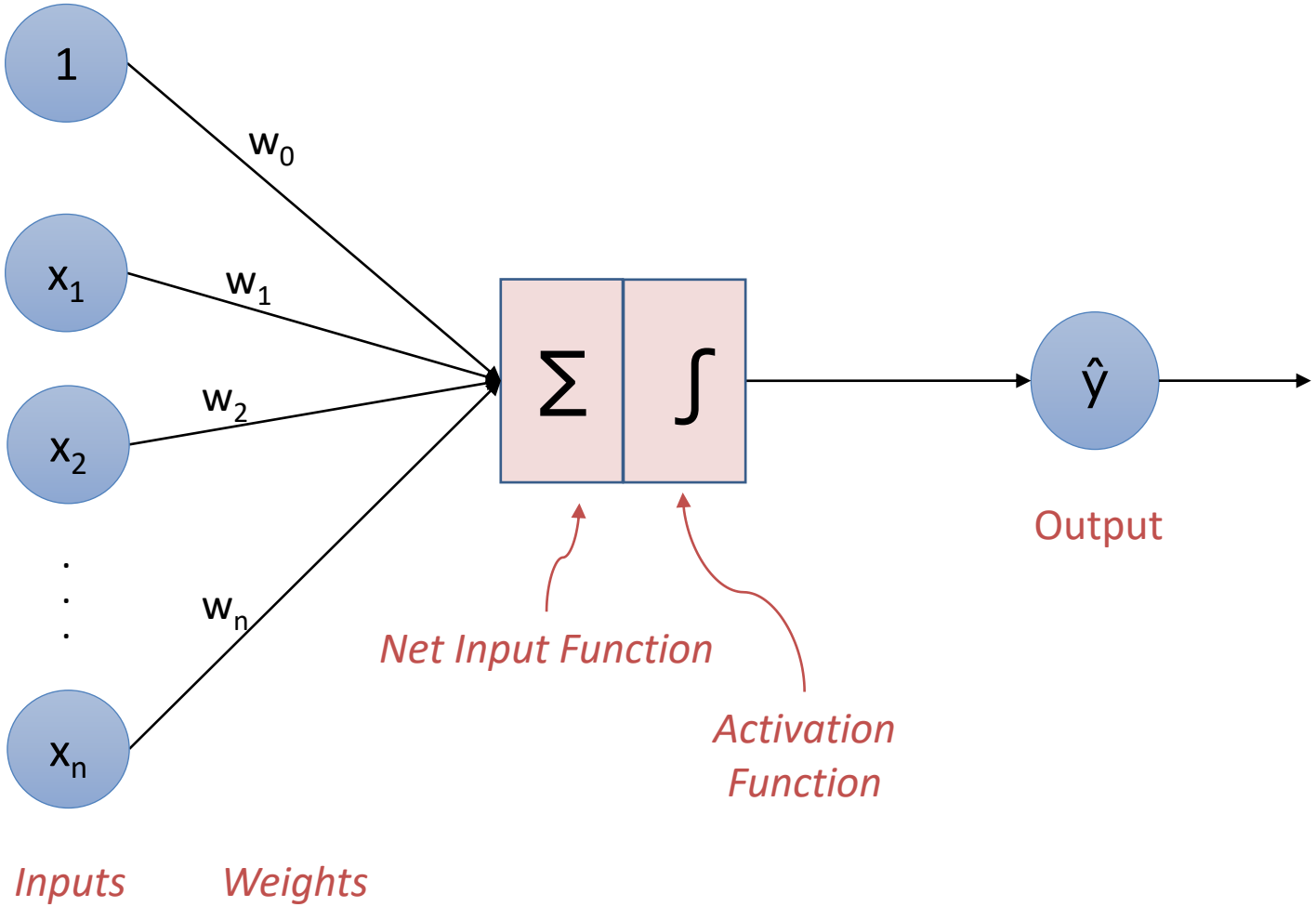
Perceptron

(Single layer perceptron)

Step 1: Inputs are multiplied with weights, a summation is performed.

Step 2: Add a bias.

$$z = \sum_{i=1}^{n} w_i x_i + 1 * w_0$$



Perceptron

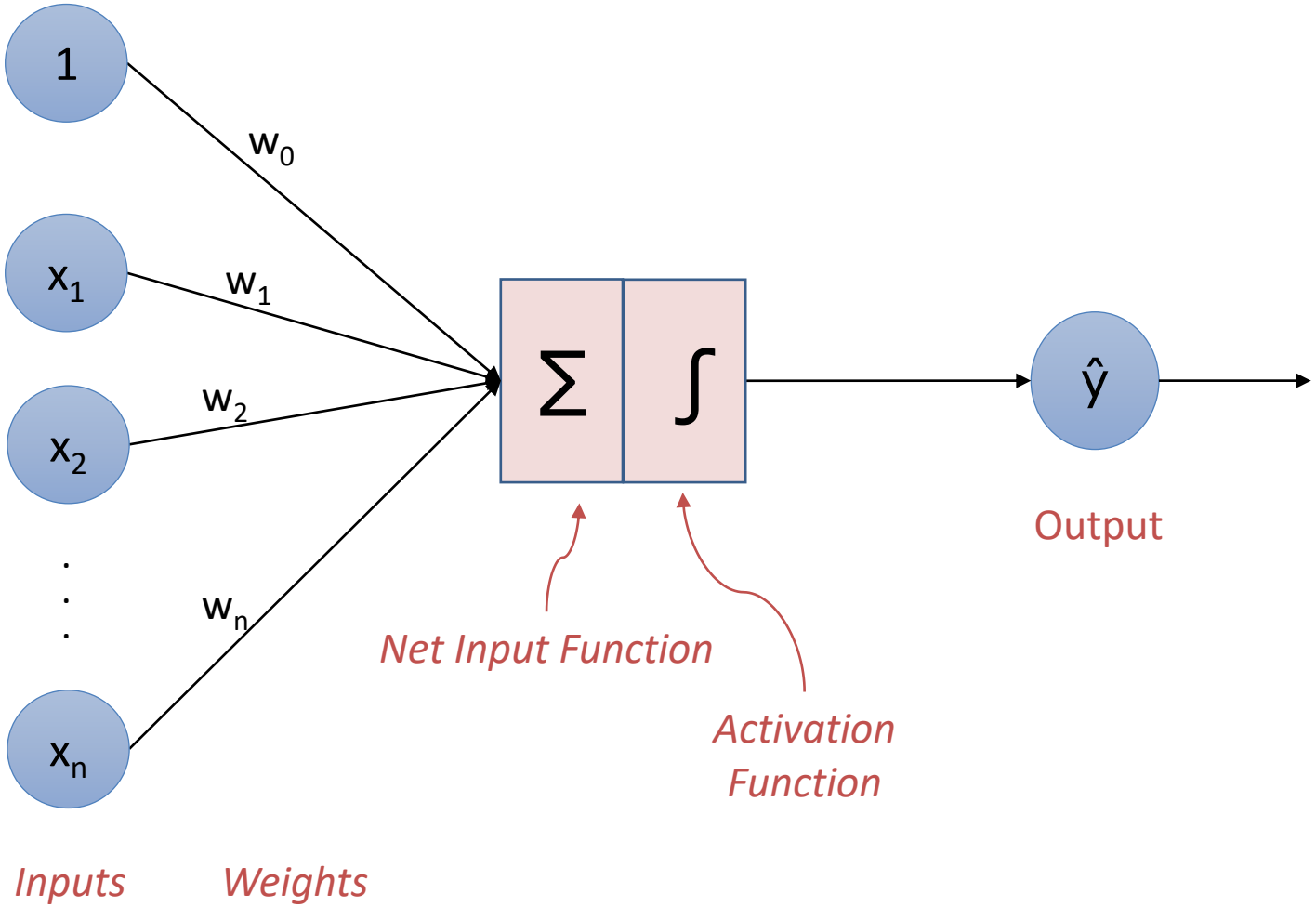
(Single layer perceptron)

Step 1: Inputs are multiplied with weights, a summation is performed.

Step 2: Add a bias.

$$z = \sum_{i=1}^{n} w_i x_i + 1 * w_0$$

Step 3: Weighted sum of inputs is passed through an activation function to determine if a neuron will be fired (activated) or not.



Activation function → serves to introduce non-linearity into the network.

$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

Perceptron

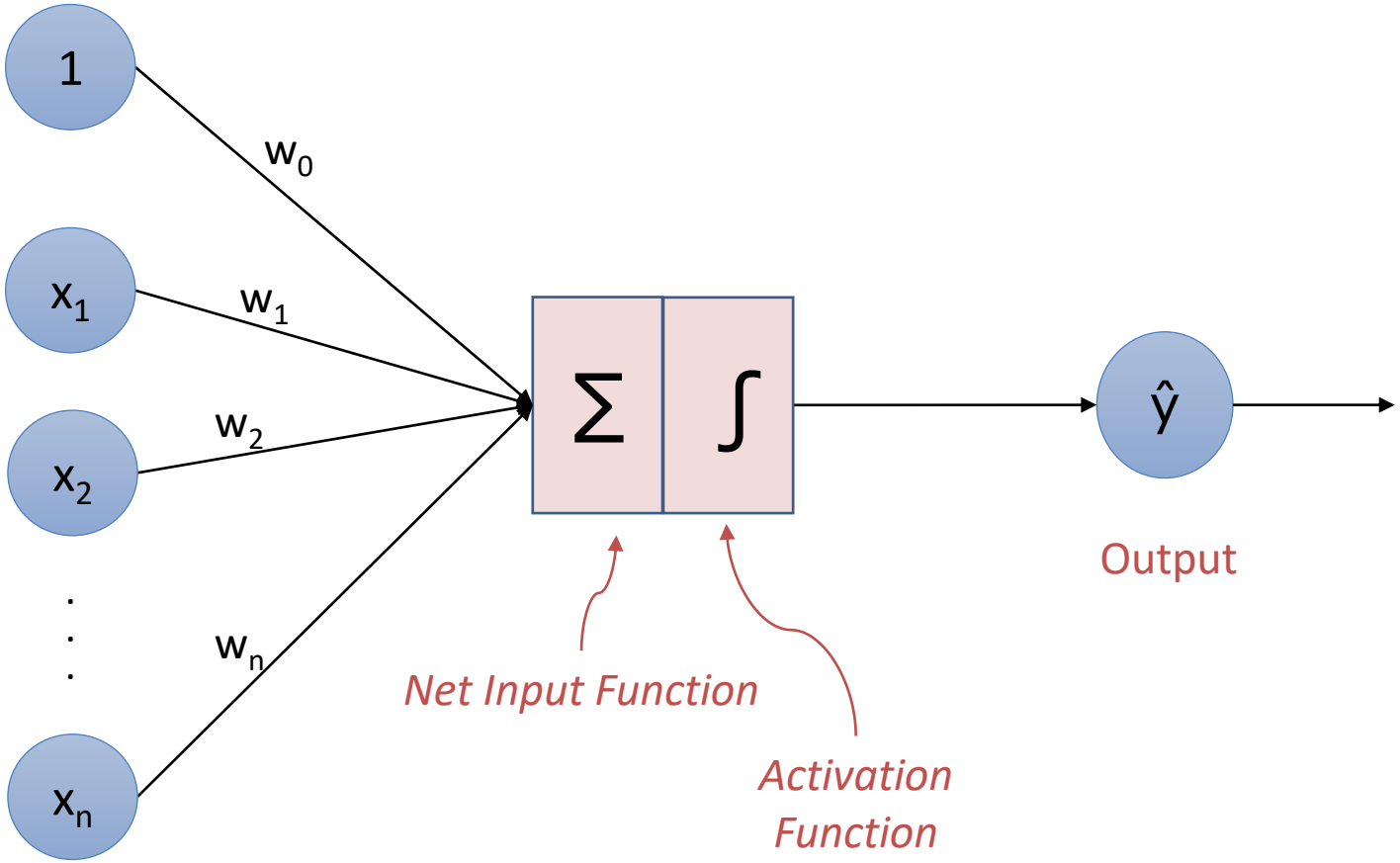
(Single layer perceptron)

Step 1: Inputs are multiplied with weights, a summation is performed.

Step 2: Add a bias.

$$z = \sum_{i=1}^{n} w_i x_i + 1 * w_0$$

Step 3: Weighted sum of inputs is passed through an activation function to determine if a neuron will be fired (activated) or not.



Activation function → serves to introduce non-linearity into the network.

$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

$$\hat{y} = g(W^T X + w_0) \quad \text{, where: } W = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_n \end{bmatrix} \text{ and } X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix}$$

Perceptron

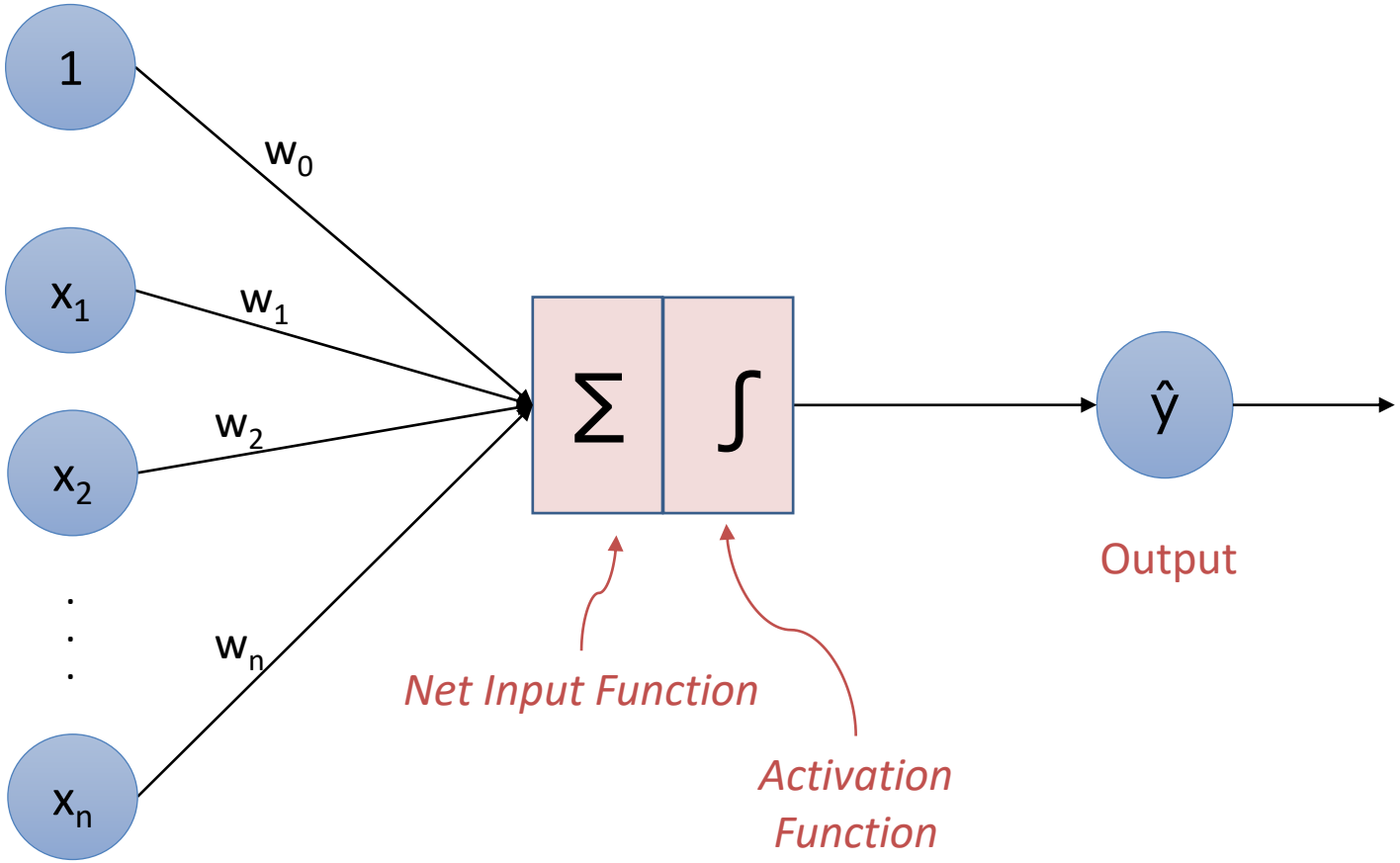
(Single layer perceptron)

Step 1: Inputs are multiplied with weights, a summation is performed.

Step 2: Add a bias.

$$z = \sum_{i=1}^{n} w_i x_i + 1 * w_0$$

Step 3: Weighted sum of inputs is passed through an activation function to determine if a neuron will be fired (activated) or not.



Inputs *Weights*

Activation function → serves to introduce non-linearity into the network.

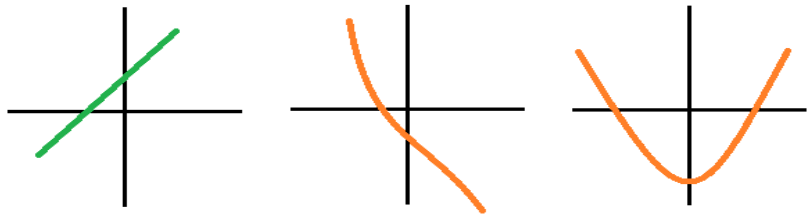
$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

→ $\hat{y} = g(z)$

$$\hat{y} = g(W^T X + w_0) \quad \text{, where: } W = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_n \end{bmatrix} \text{ and } X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix}$$

Activation/Transfer Function

$\hat{y} = g(z)$

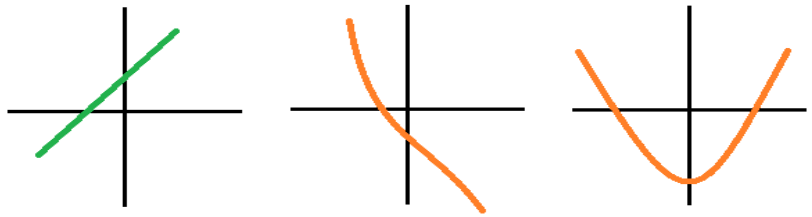


Linear function Non-linear function

- Serves to introduce non-linearity into the network.
- Linear functions: polynomials of degree 1 (y = mx+c) → straight line.
  - ✓ Derivative of a linear function is a constant, i.e., y'(x) = m.
- Non-linear functions: polynomials of degree > 1 ($y = 3x^2 + 5x + 6$) → not a straight line.
  - ✓ Differentiable, i.e., first-order derivative can be calculated, i.e., y'(x) = 6x + 5.

Activation/Transfer Function

$\hat{y} = g(z)$

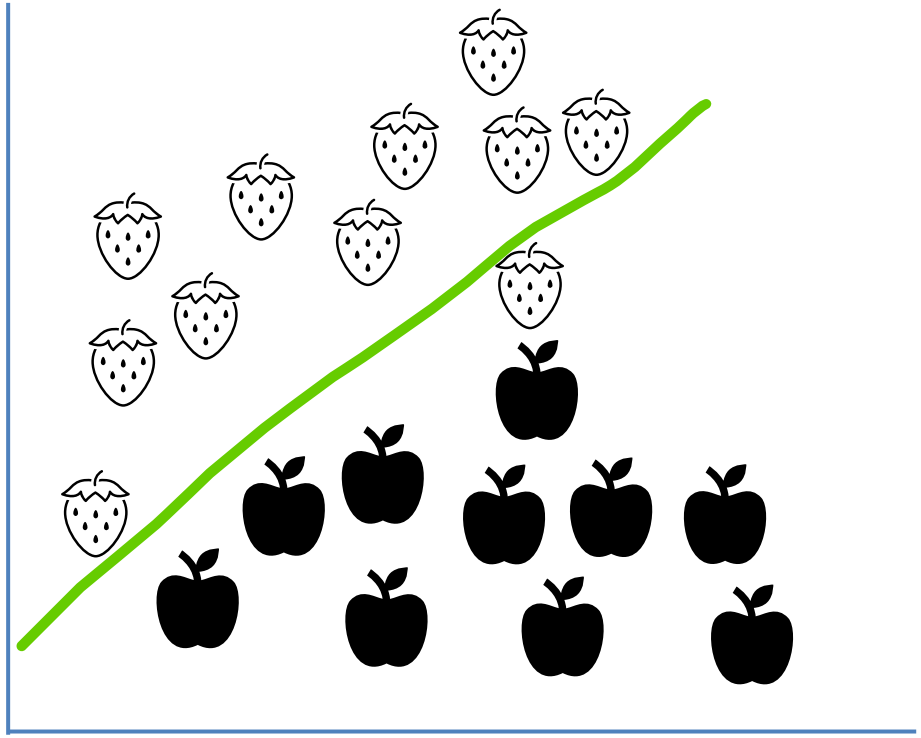


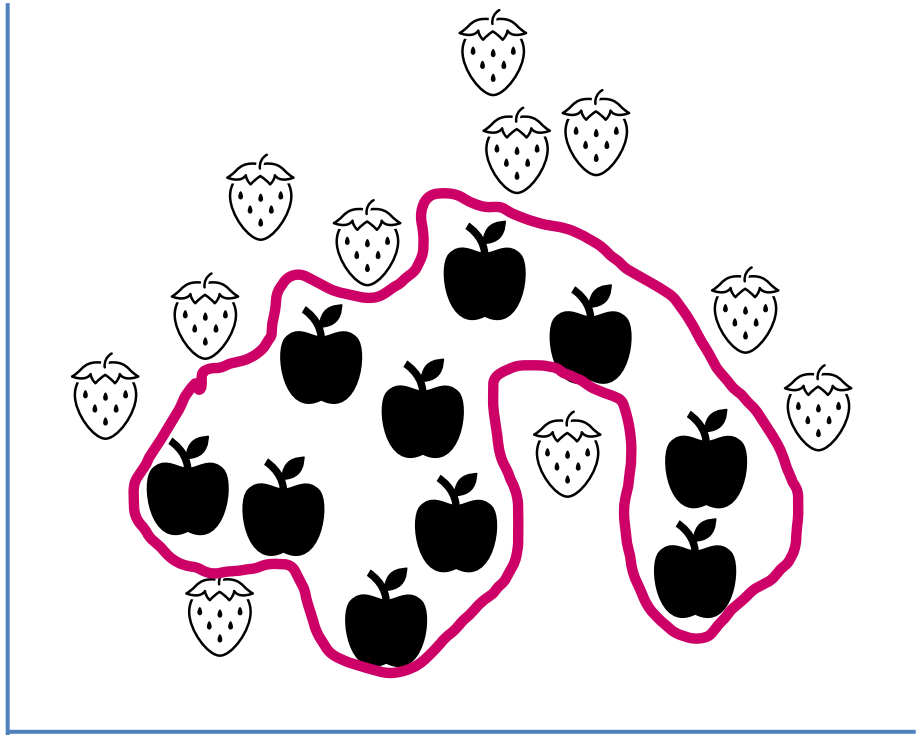
Linear function Non-linear function

- Serves to introduce non-linearity into the network.
- Linear functions: polynomials of degree 1 (y = mx+c) → straight line.
  - ✓ Derivative of a linear function is a constant, i.e., y'(x) = m.
- Non-linear functions: polynomials of degree > 1 ($y = 3x^2 + 5x + 6$) → not a straight line.
  - ✓ Differentiable, i.e., first-order derivative can be calculated, i.e., y'(x) = 6x + 5.

- If a linear activation function is used, then no matter how many hidden layers are used in the NN, the NN will always become equivalent to having a single layer N/W.

→ If f(x) and g(x) are linear functions, then g(f(x)) and f(g(x)) will also be linear.

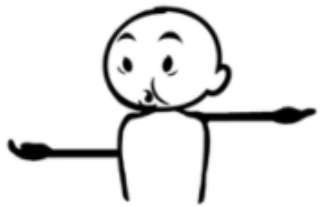


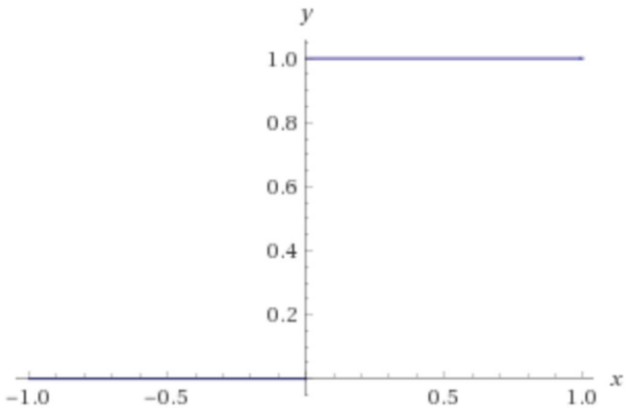


For a DNN that can model any type of data and approximate complex functions, a non-linear activation function is thus needed.

Types of Activation Functions

Step Function



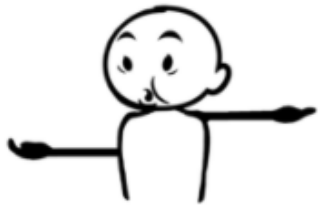


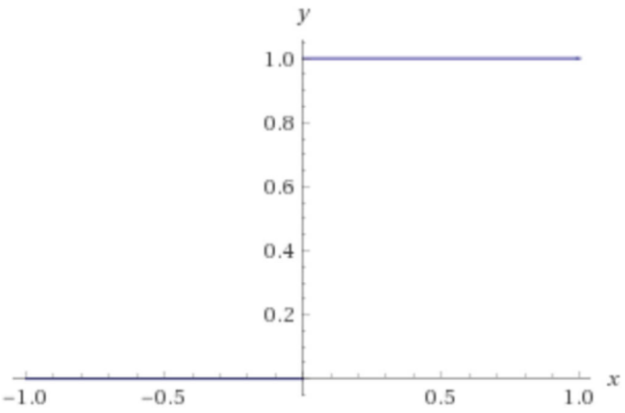
$$g(z) = \begin{cases} 1, z > \ominus \\ 0, z \leq \ominus \end{cases}$$

Types of Activation Functions

Non-linear functions

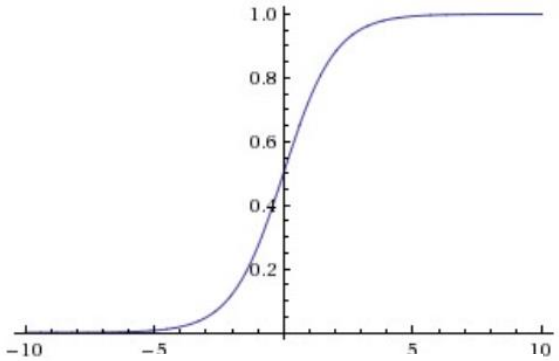
Step Function





$$g(z) = \begin{cases} 1, z > \ominus \\ 0, z \leq \ominus \end{cases}$$

Sigmoid



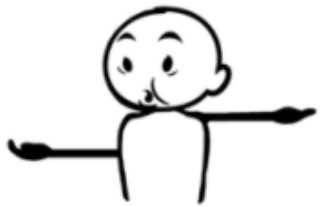
$$g(z) = \frac{1}{1 + e^{-z}}$$

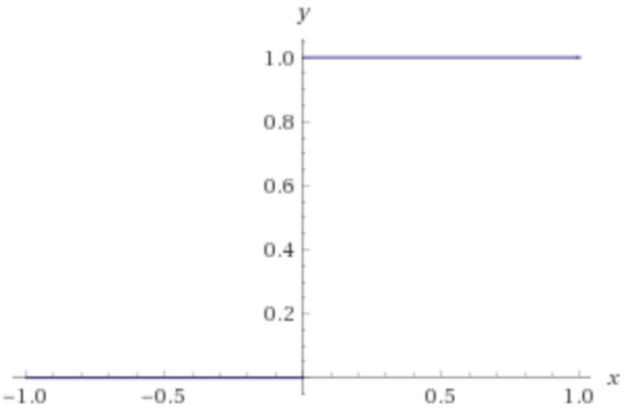
$g(z) \in (0,1)$

Types of Activation Functions

Non-linear functions

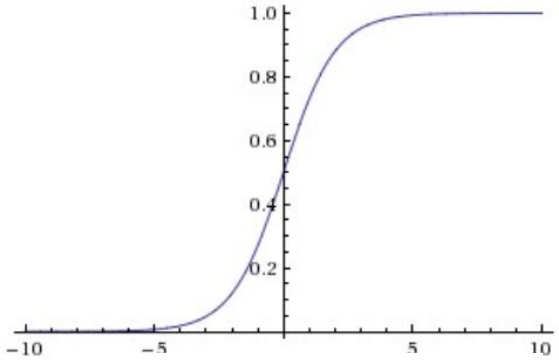
Step Function





$$g(z) = \begin{cases} 1, z > \Theta \\ 0, z \leq \Theta \end{cases}$$

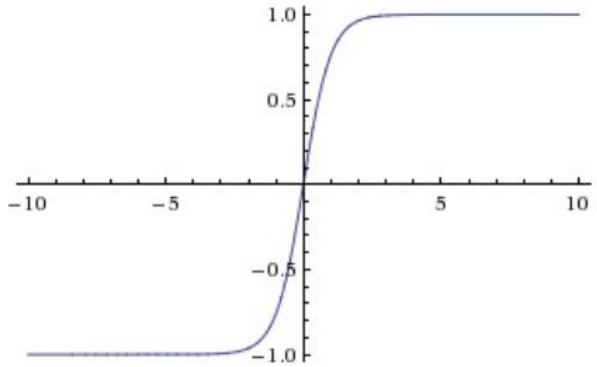
Sigmoid



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

$g(z) \in (0,1)$

Tanh



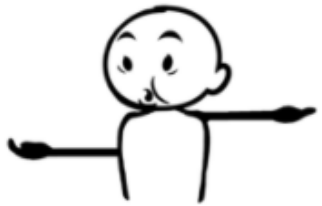
$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

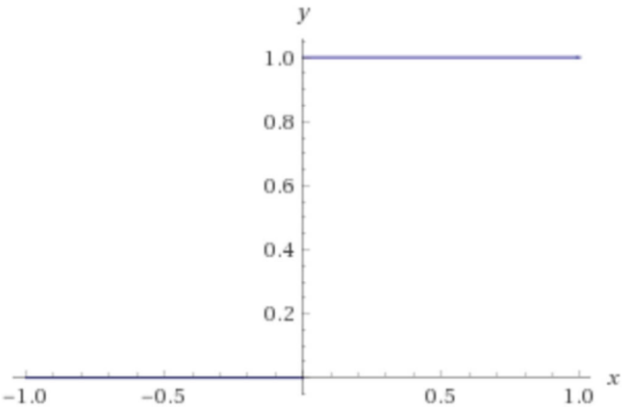
$g(z) \in (-1,1)$

Types of Activation Functions

Non-linear functions

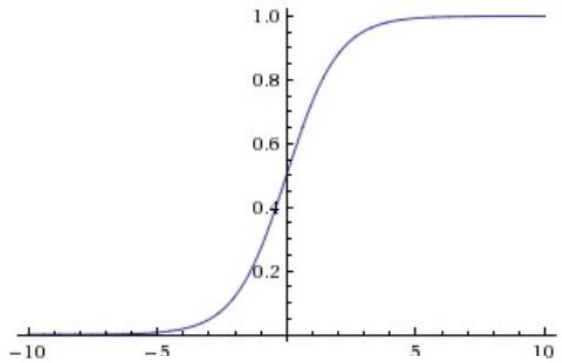
Step Function





$$g(z) = \begin{cases} 1, z > \Theta \\ 0, z \leq \Theta \end{cases}$$

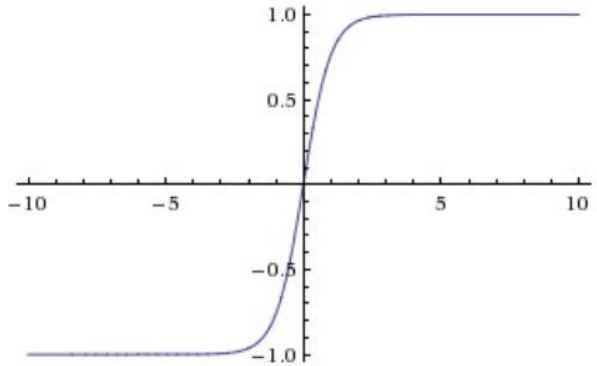
Sigmoid



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

$g(z) \in (0,1)$

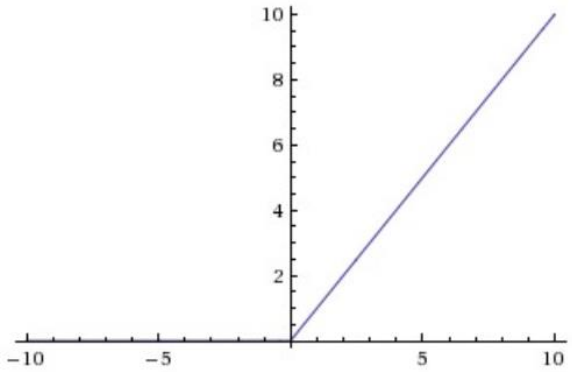
Tanh



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

$g(z) \in (-1,1)$

ReLU



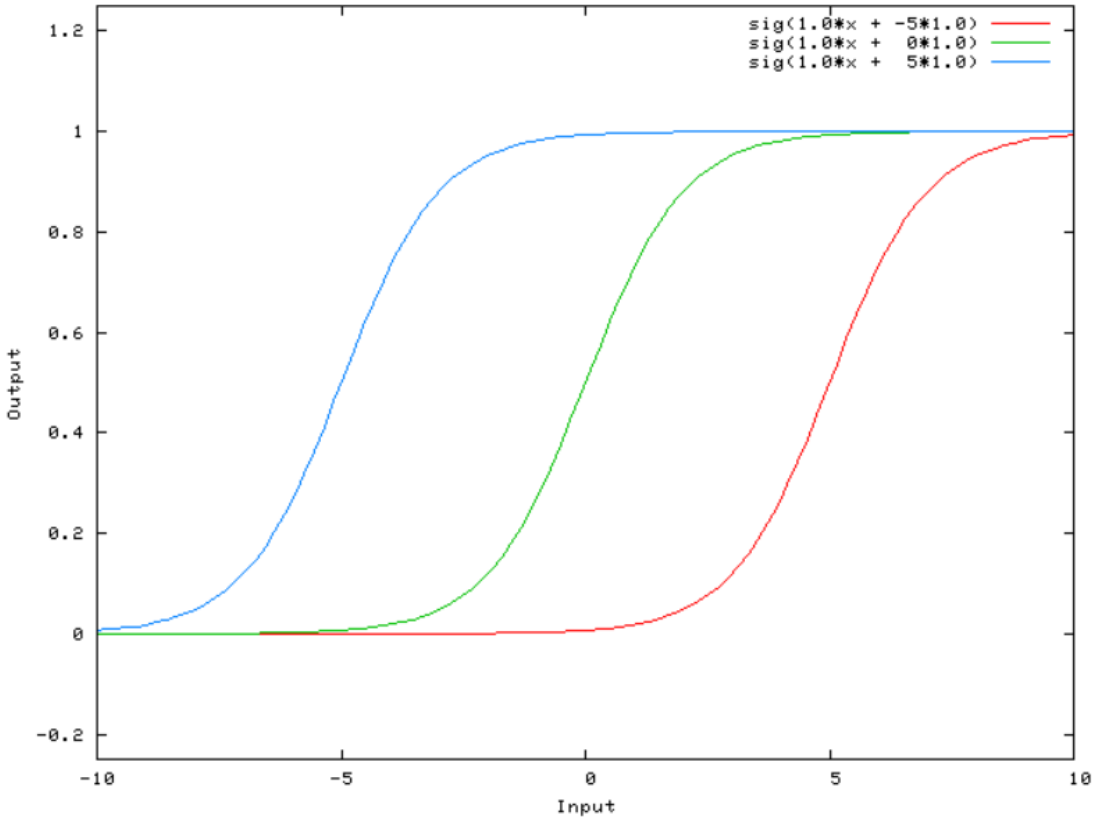
$$g(z) = \max(0, z)$$

$g(z) \in [0, \infty)$

Bias

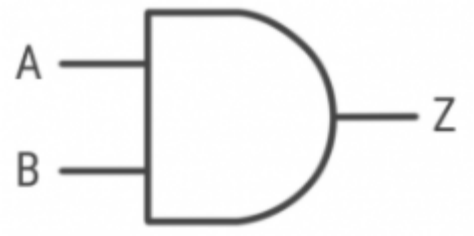
$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

- Analogous to the role of a constant in a linear function.
- Serves to shift the activation function to the left or right by adding a scalar value to the function.
- Helps the model to better fit the data.



Single Perceptron: Example

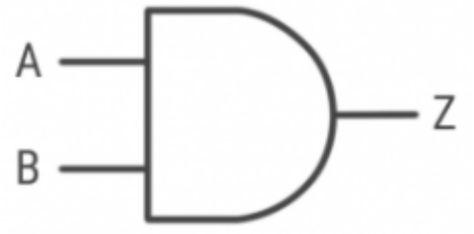
- Implementing an **AND gate** with a **threshold** activation function.



Input A	Input B	Output Z
0	0	0
0	1	0
1	0	0
1	1	1

Single Perceptron: Example

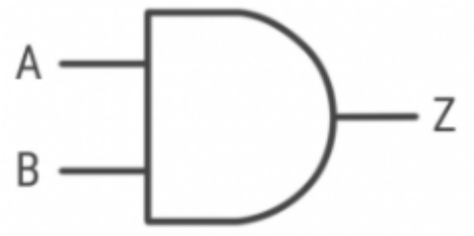
- Implementing an **AND gate** with a **threshold** activation function.



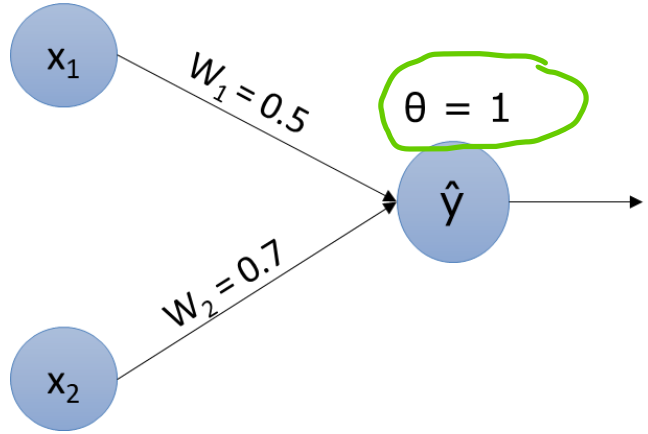
x_1 Input A	x_2 Input B	Output Z
0	0	0
0	1	0
1	0	0
1	1	1

Single Perceptron: Example

- Implementing an **AND gate** with a **threshold** activation function.



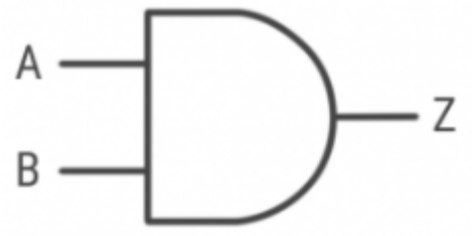
Input A	Input B	Output Z
0	0	0
0	1	0
1	0	0
1	1	1



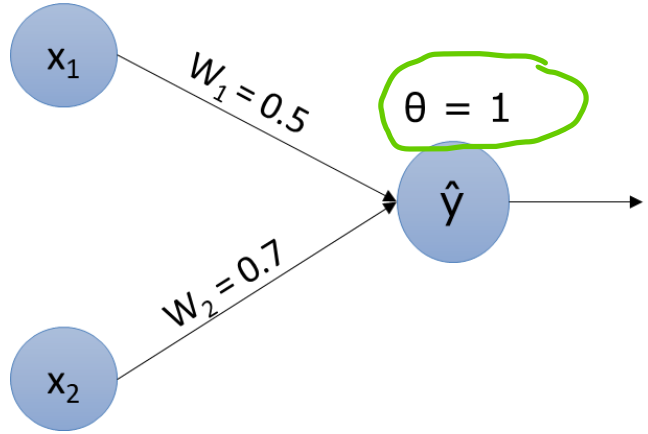
$$g(z) = \begin{cases} 1, z > \Theta \\ 0, z \leq \Theta \end{cases}$$

Single Perceptron: Example

- Implementing an **AND gate** with a **threshold** activation function.



Input A	Input B	Output Z
0	0	0
0	1	0
1	0	0
1	1	1



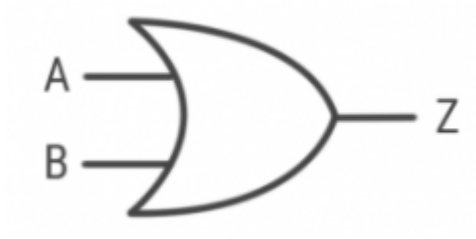
w_1x_1	w_2x_2	O/P = $w_1x_1 + w_2x_2$
0.5*0	0.7*0	0.0
0.5*0	0.7*1	0.7
0.5*1	0.7*0	0.5
0.5*1	0.7*1	1.2

$$g(z) = \begin{cases} 1, z > \Theta \\ 0, z \leq \Theta \end{cases}$$

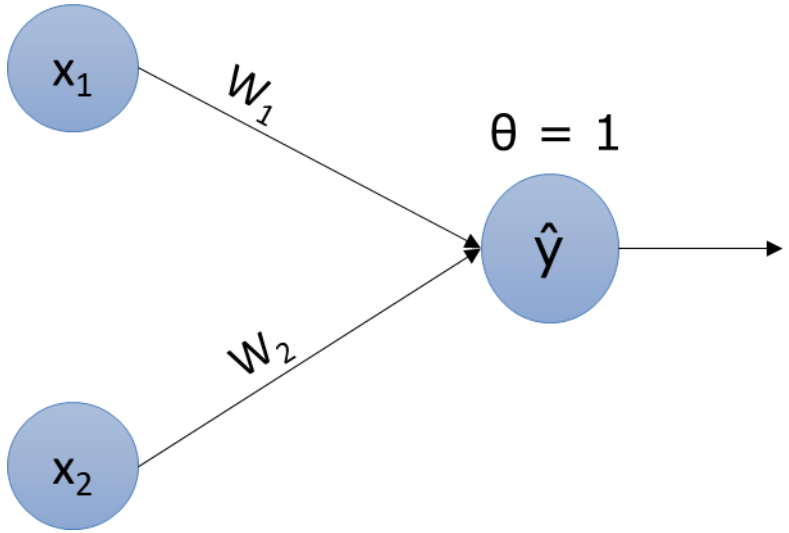
➔ Output = 1

Single Perceptron: Exercise

- Implementing an OR gate with a threshold activation function.

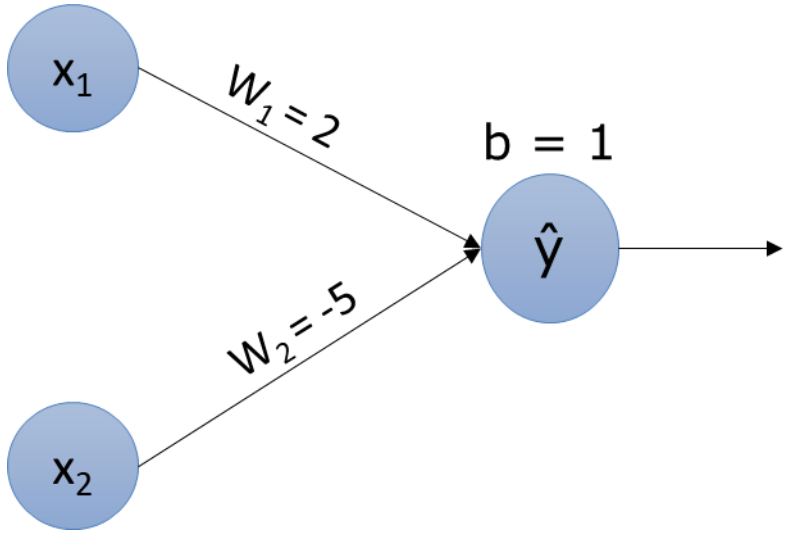


Input A	Input B	Output Z
0	0	0
0	1	1
1	0	1
1	1	1



Single Perceptron: Exercise

- Use of a non-linear activation (sigmoid) function.

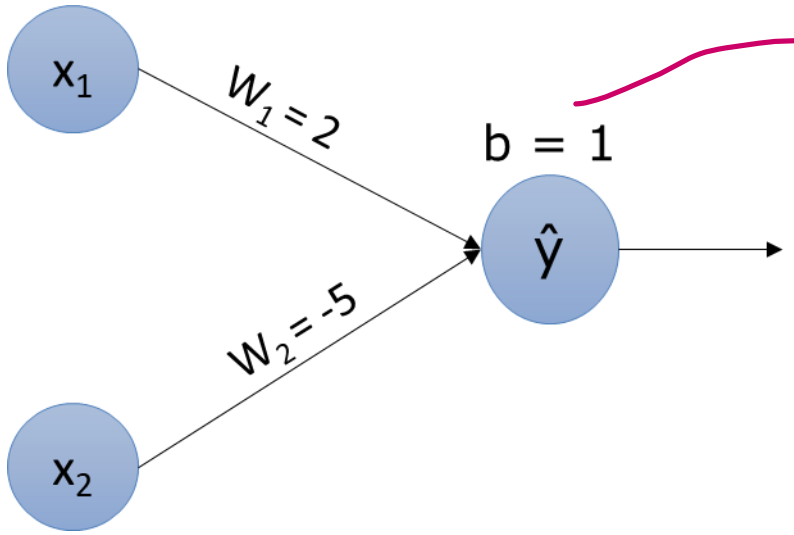


$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

Consider the data point $X = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Single Perceptron: Exercise

- Use of a non-linear activation (sigmoid) function.



Bias (b) = 1, $w_0 = 1$

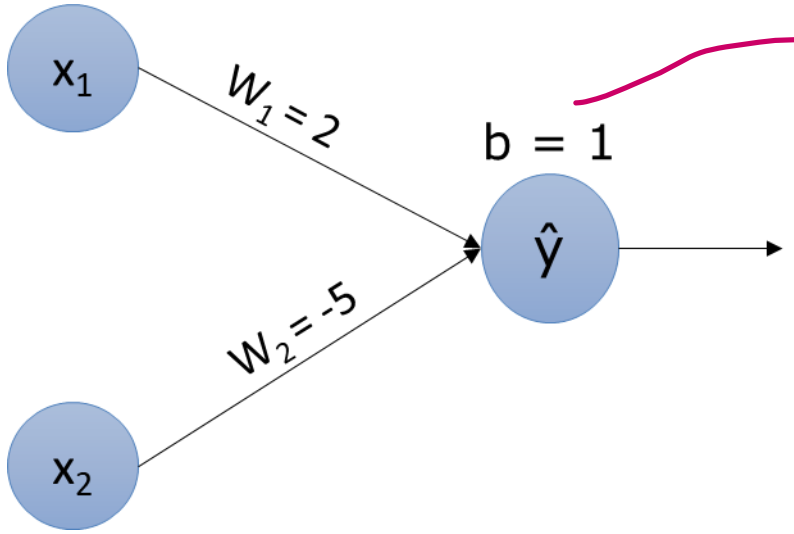
Bias (b) = 0, $w_0 = 1$

$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

Consider the data point $X = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Single Perceptron: Exercise

- Use of a non-linear activation (sigmoid) function.



Bias (b) = 1, w_0 = 1

$\hat{y} = g(2x_1 - 5x_2 + 1)$

➔ $\hat{y} = g(2 * 1 - 5 * 1 + 1)$

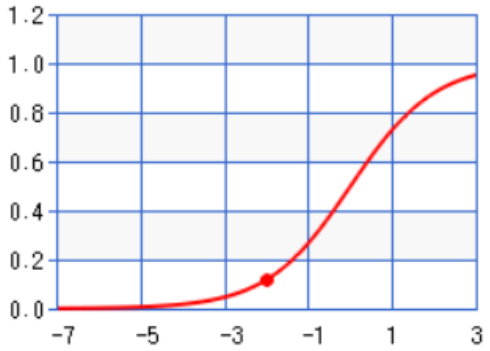
➔ $\hat{y} = g(-2)$

➔ $\hat{y} = 0.1192$

Bias (b) = 0, w_0 = 1

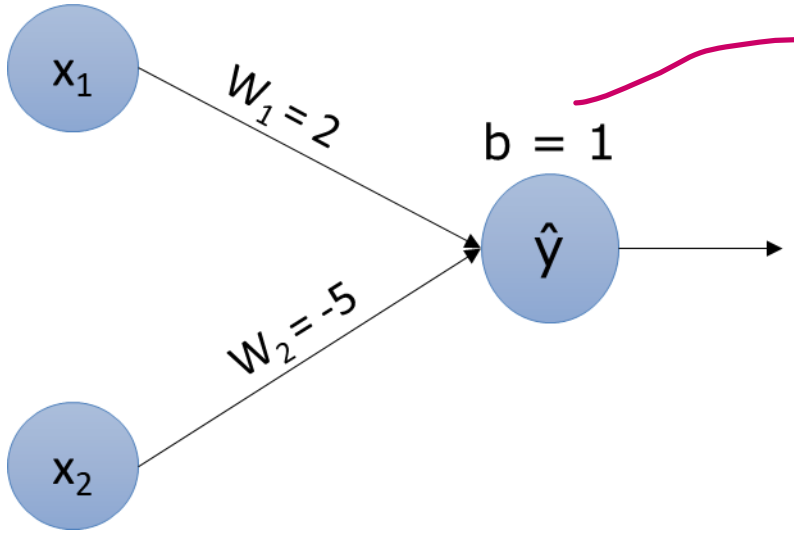
$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

Consider the data point $X = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$



Single Perceptron: Exercise

- Use of a non-linear activation (sigmoid) function.



$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

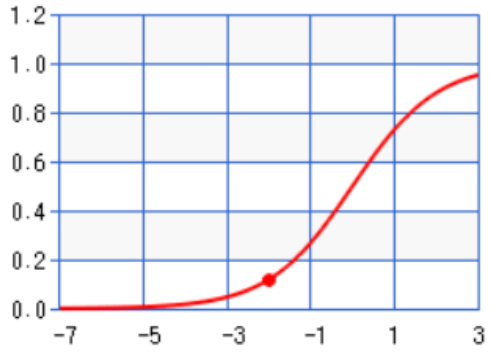
Consider the data point $X = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Bias (b) = 1, w_0 = 1

$\hat{y} = g(2x_1 - 5x_2 + 1)$

➔ $\hat{y} = g(2 * 1 - 5 * 1 + 1)$
➔ $\hat{y} = g(-2)$

➔ $\hat{y} = 0.1192$

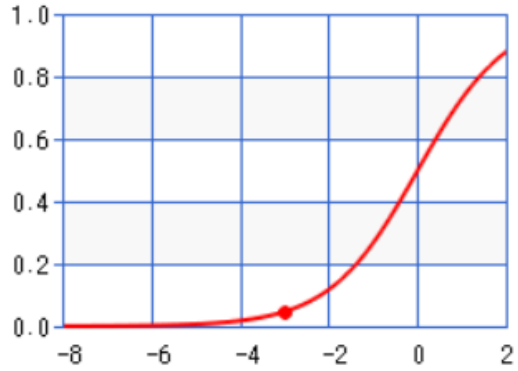


Bias (b) = 0, w_0 = 1

$\hat{y} = g(2x_1 - 5x_2)$

➔ $\hat{y} = g(2 * 1 - 5 * 1)$
➔ $\hat{y} = g(-3)$

➔ $\hat{y} = 0.0474$

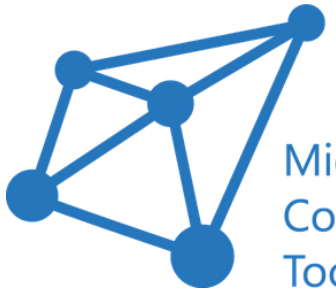


Deep Learning Frameworks

TensorFlow

Caffe

theano



Microsoft Cognitive Toolkit

Popular DL Frameworks

mxnet

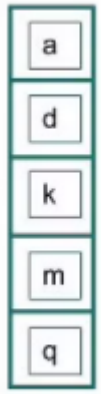
PYTORCH

DL4J

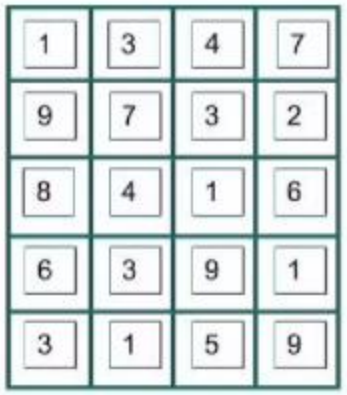
Keras

- TensorFlow is a DL tool to define and run computations involving **tensors**.
- Accepts multi-dimensional arrays as input known as **tensors**.
  - ✓ Useful in handling vast quantities of data.
- **Tensor**: generalisation of vectors and matrices to potentially high dimensions.
- **Rank**: number of dimensions used to represent the data

TensorFlow



Tensor of dimension 5
Rank: 1



Tensor of dimension [5,4]
Rank: 2

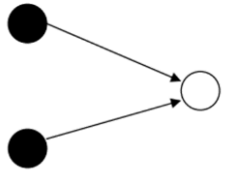
Tensor of dimension [3,3,3]
Rank: 3

TensorFlow programs

```
import tensorflow as tf
```

Building a computational graph

- Computations that need to be carried out happen in the form of a graph.
- These data flow graphs consist of tensors connected together.
- A tensor comprises of a node and an edge.
- Writing code for preparing the graph.



Executing the computational graph

- Graphs are executed within a session.
- Can be executed on CPU's or GPU's or in a distributed manner.

Launching a session: `sess = tf.Session()`

Running a session to evaluate a tensor: `sess.run()`

Working with data in TensorFlow

- Constants: values that do not change.

```
a = tf.constant(5.0,tf.float32)
b = tf.constant(3.0)
```

- Placeholders: allow to parametrize a graph to accept external inputs.

```
node_1 = tf.placeholder(tf.int32)
node_2 = tf.placeholder(tf.int32)
```

- Variables: allow to add new **trainable** parameters to the graph.

```
node_3 = tf.Variable([0.3],tf.float32)
node_4 = tf.Variable([0.7],tf.float32)
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

explicitly initialising variables within a session

TensorFlow operations

- tf.add(a, b)
- tf.subtract(a, b)
- tf.multiply(a, b)
- tf.matmul(a, b)
- tf.div(a, b)
- tf.pow(a, b)
- tf.exp(a)
- tf.sqrt(a)

TensorFlow Datatypes

- Floating point: tf.float32, tf.float64
- Signed Integer: tf.int8, tf.int16, tf.int32, tf.int64
- Unsigned Integer: tf.uint8, tf.unit16
- String: tf.string
- Boolean: tf.bool

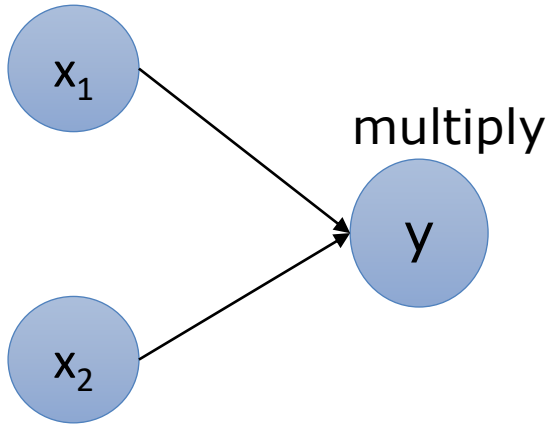
https://www.tensorflow.org/

TensorFlow Functions for Activation Functions

- tf.math.sigmoid(z)
- tf.math.tanh(z)
- tf.nn.relu(z)
- tf.nn.softmax(z)

Steps followed

1. Defining data values
2. Defining the computation
3. Execute the operation
   a) Create a session
   b) Run the session

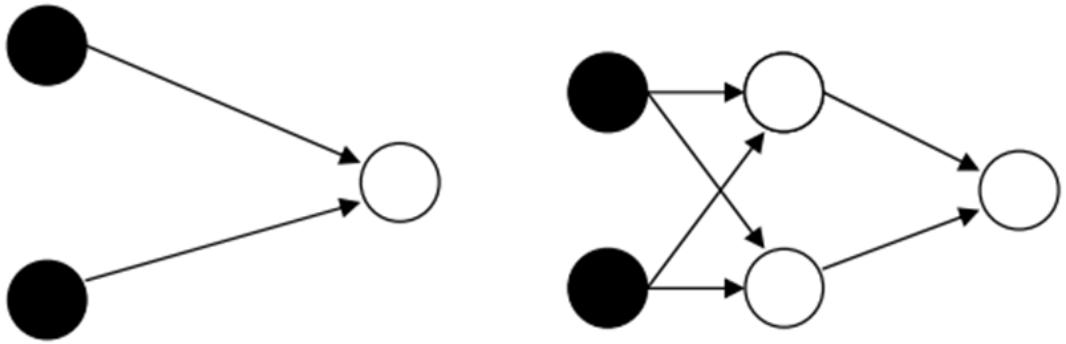



```
x1 = tf.placeholder(tf.int32)
x2 = tf.placeholder(tf.int32)
sess = tf.Session()
print(sess.run(tf.multiply(x1,x2),{x1:[3,4],x2:[5,1]}))
```

https://github.com/Pikakshi/Introduction-to-Deep-Learning

Multi-Layer Perceptron

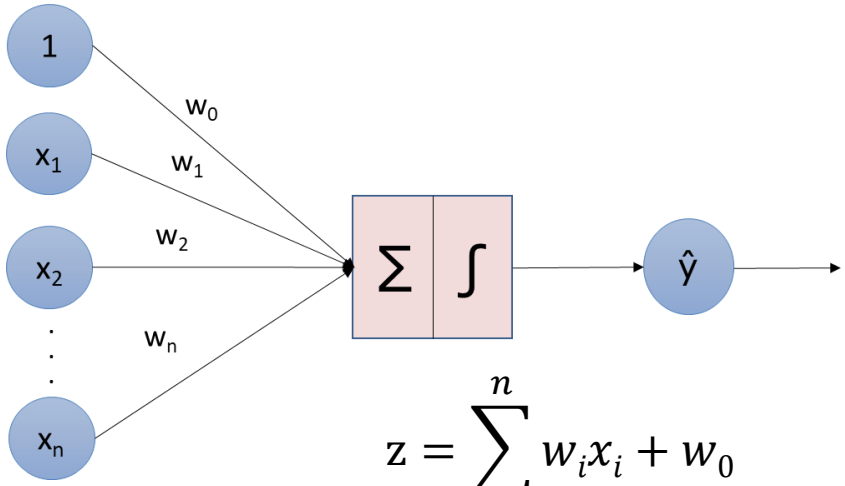
Multi-layer FFNN/Multi-layer Perceptron (Fully Connected)



Single-Layer Feed-Forward NN

Multi-Layer Feed-Forward NN

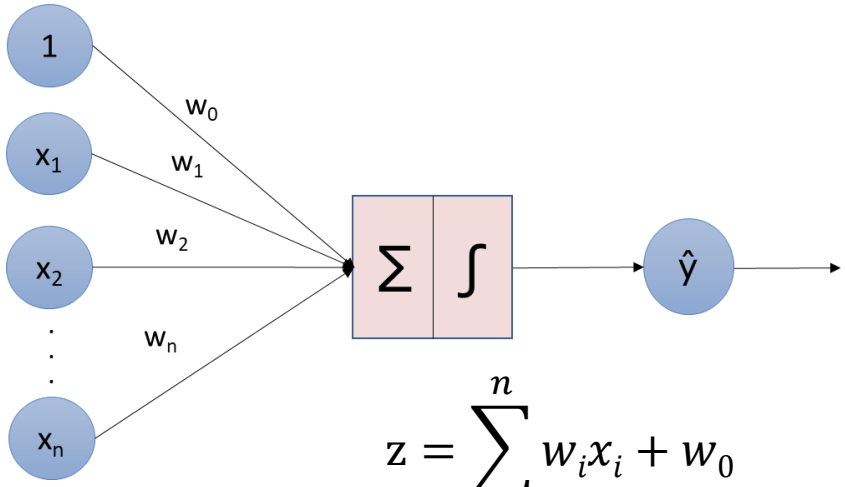
- Each neuron in the preceding layer is connected to every neuron in the subsequent layer.
- One or more hidden layers.
- No backward connection/feedback loops.
- Minsky et al., 1969 reported that a single-layer perceptron can't be applied to non-linear data.
- Applications: suitable for multi-class classification problems, simple regression predictions.
- Not ideal for processing sequential data, use of RNN's in such cases.



$$z = \sum_{i=1}^{n} w_i x_i + w_0$$

$$\hat{y} = g(z)$$

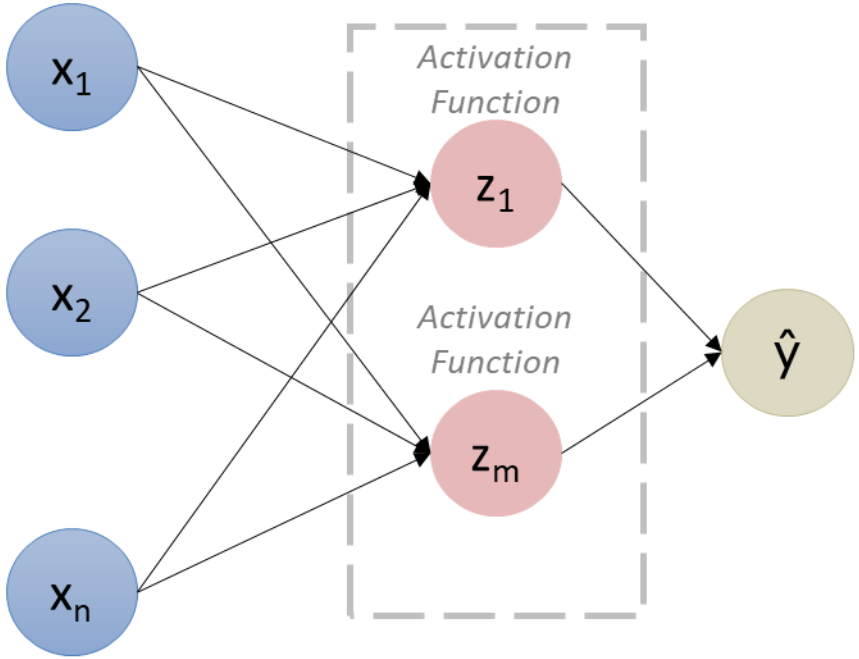
$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$



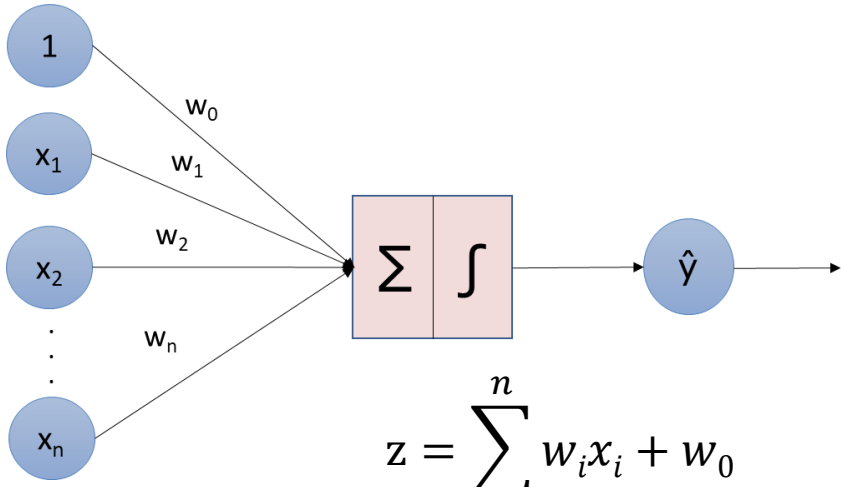
$$z = \sum_{i=1}^{n} w_i x_i + w_0$$

$$\hat{y} = g(z)$$

$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$



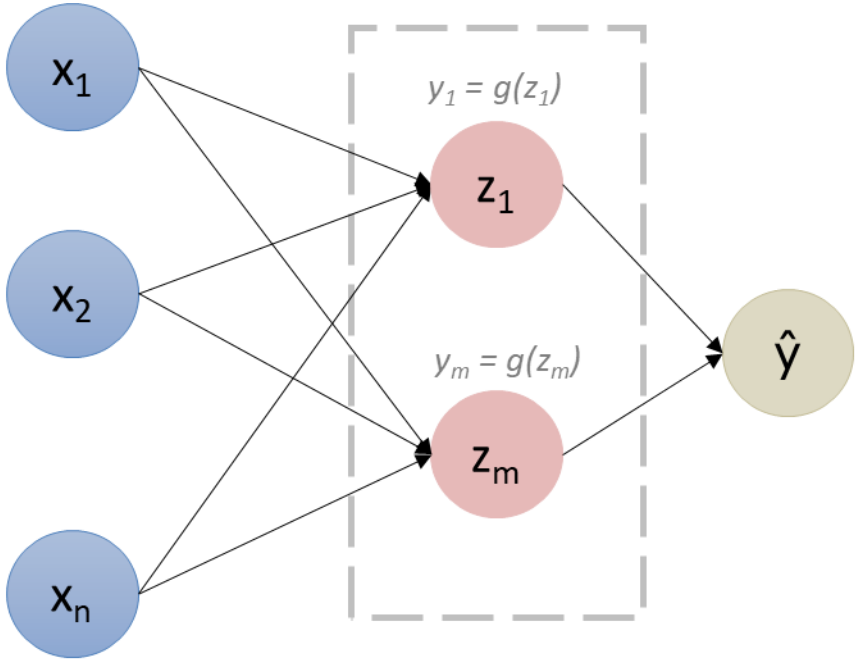
Input Layer *Hidden Layer* *Output Layer*



$$z = \sum_{i=1}^{n} w_i x_i + w_0$$

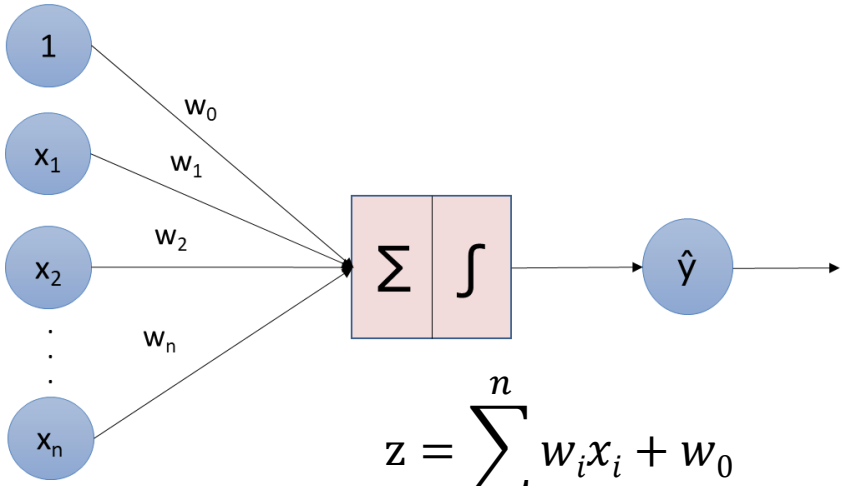
$$\hat{y} = g(z)$$

$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$



Input Layer *Hidden Layer* *Output Layer*

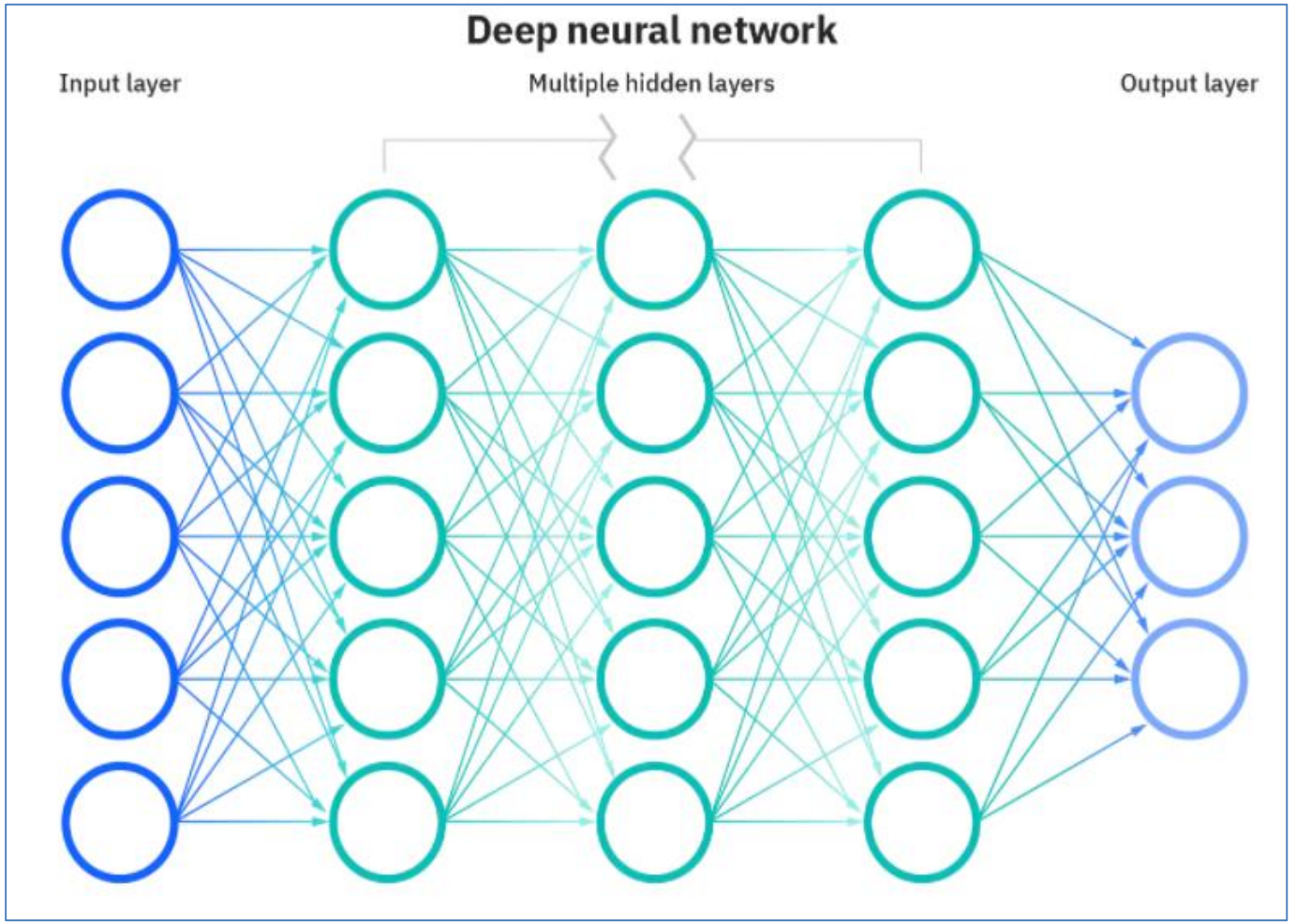
Forward Propagation

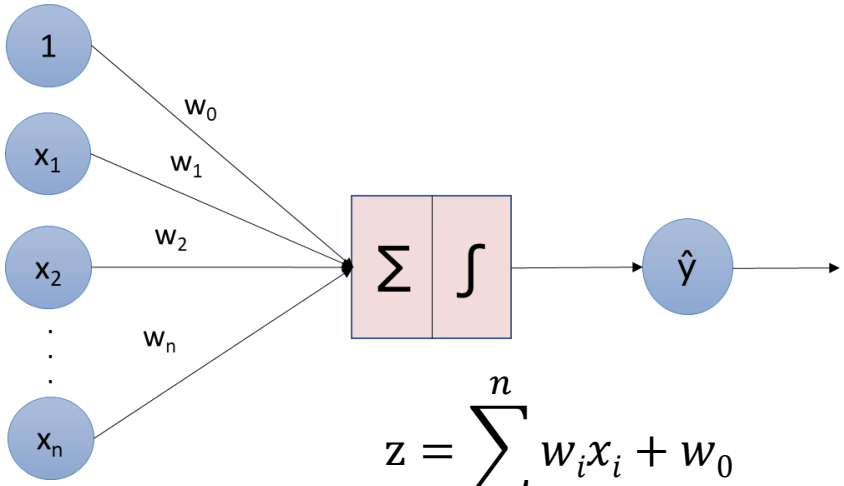


$$z = \sum_{i=1}^{n} w_i x_i + w_0$$

$$\hat{y} = g(z)$$

$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

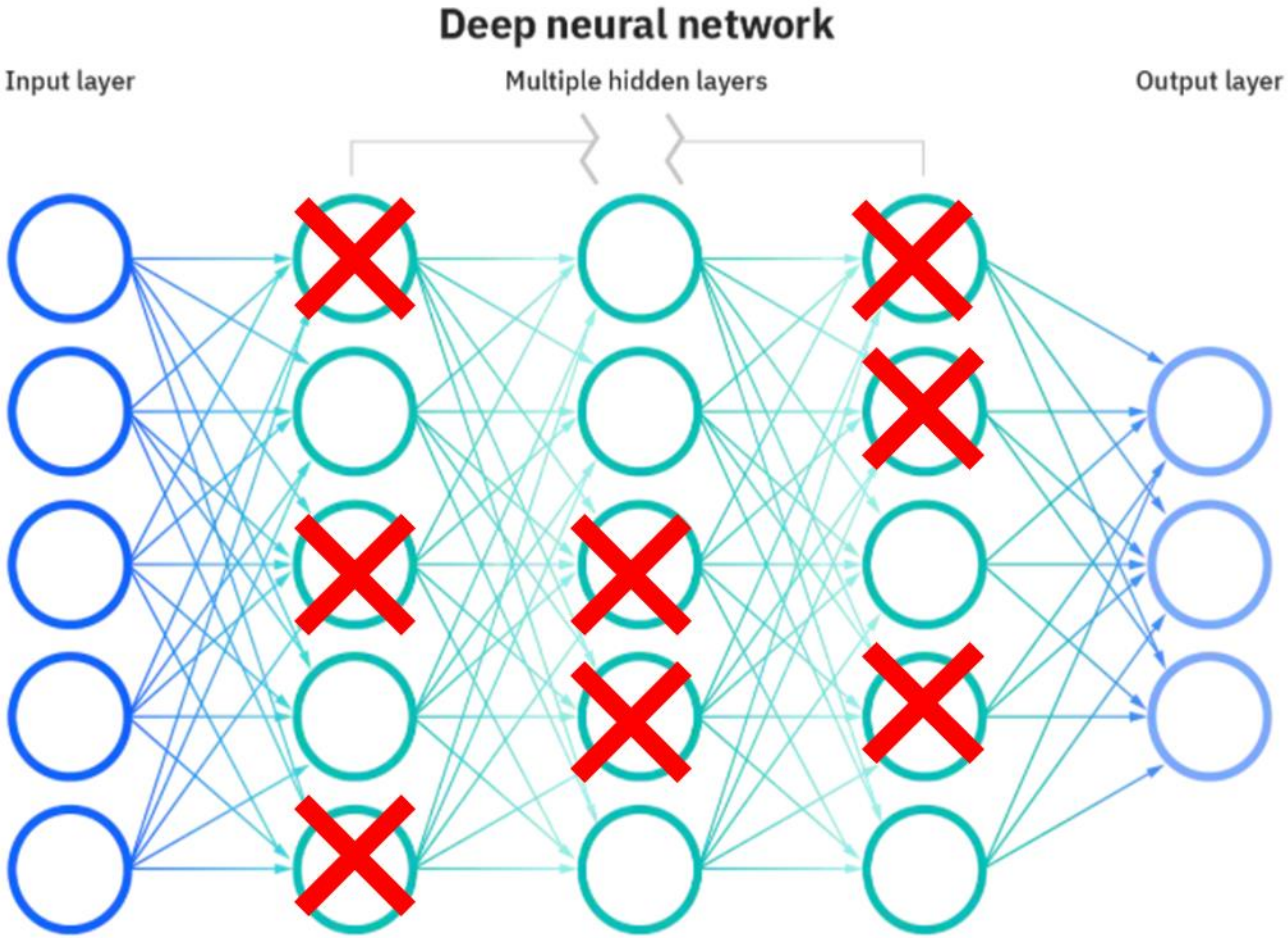




$$z = \sum_{i=1}^{n} w_i x_i + w_0$$

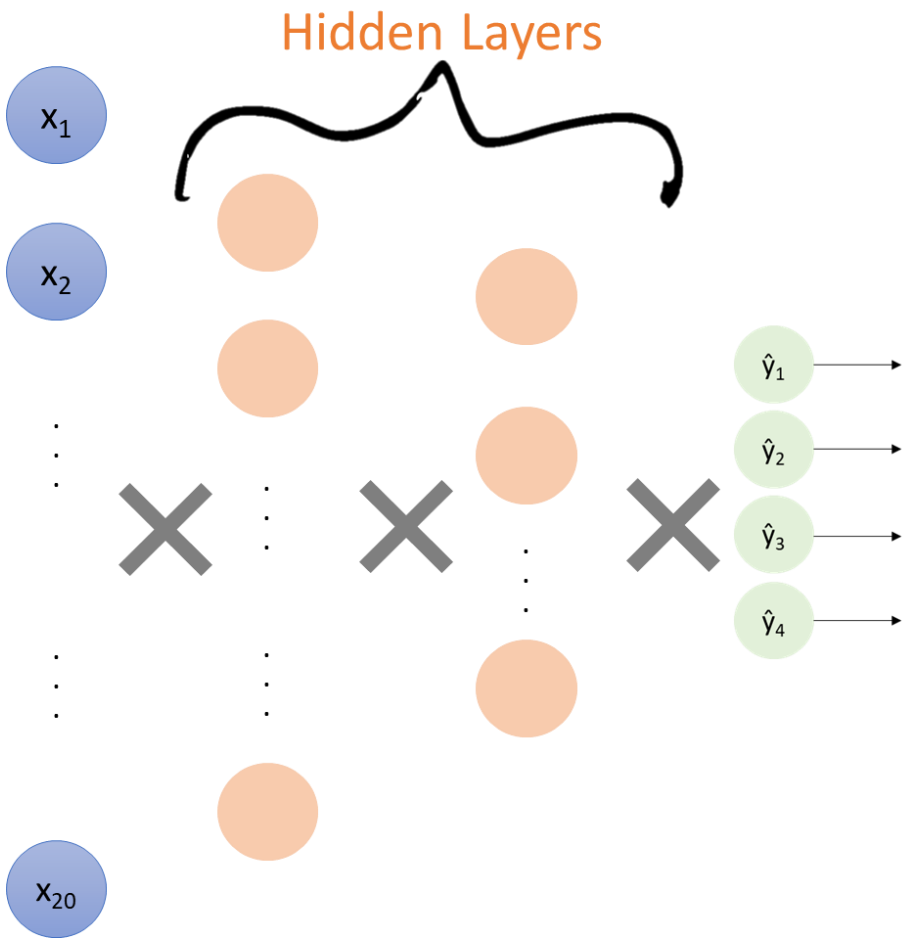
$$\hat{y} = g(z)$$

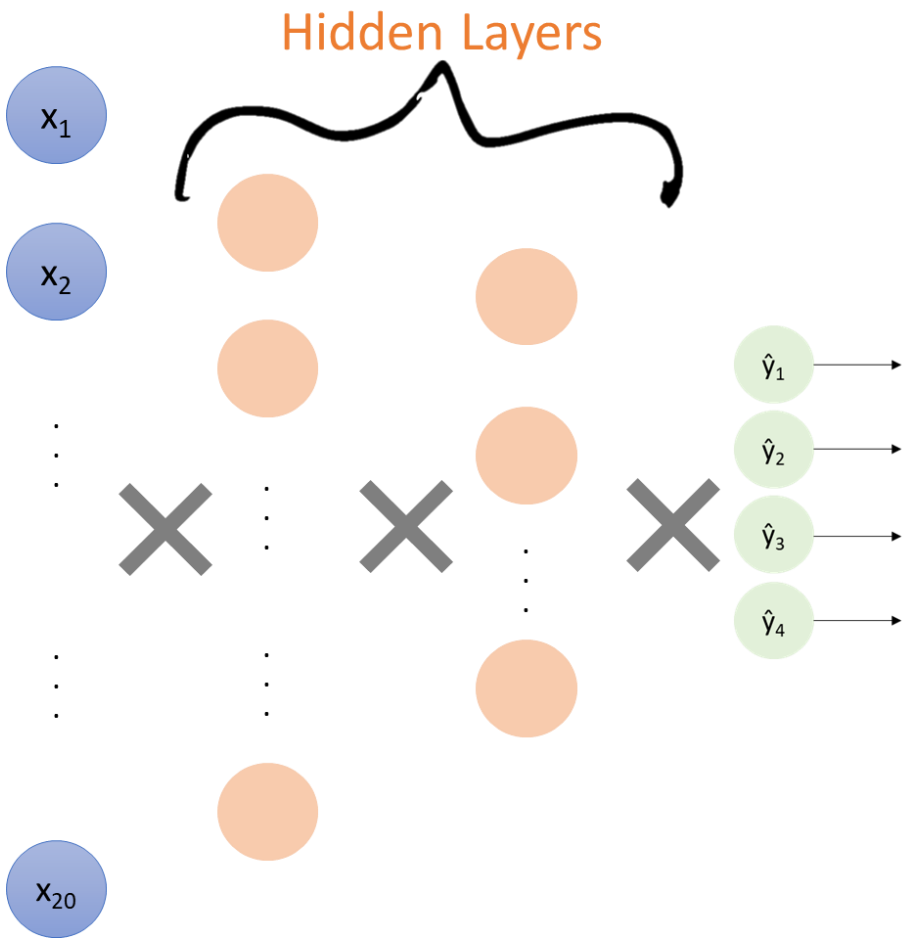
$$\hat{y} = g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$



Dropout → to avoid overfitting

tf.keras.layers.Dropout(rate)


Hidden Layers
x_1
x_2
x_{20}
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$



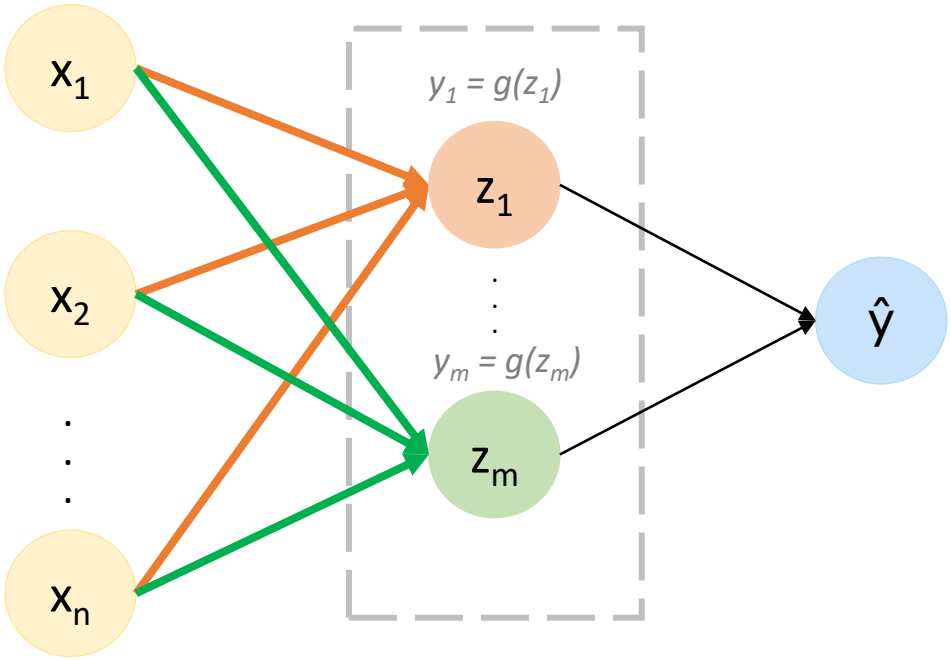
Creating a Sequential model since models in keras are grouped as a linear stack of layers.

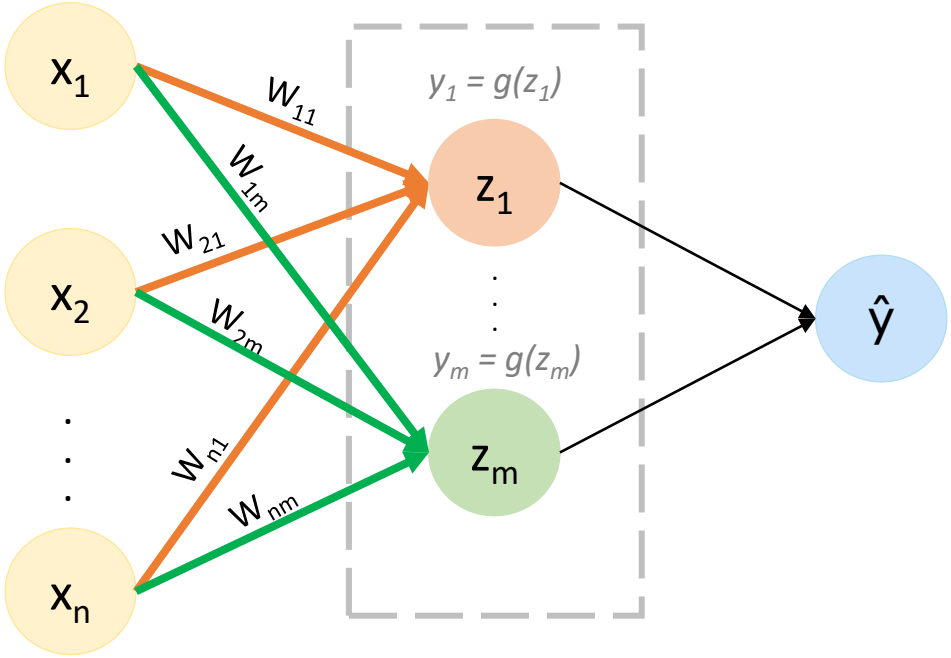
2 hidden layers

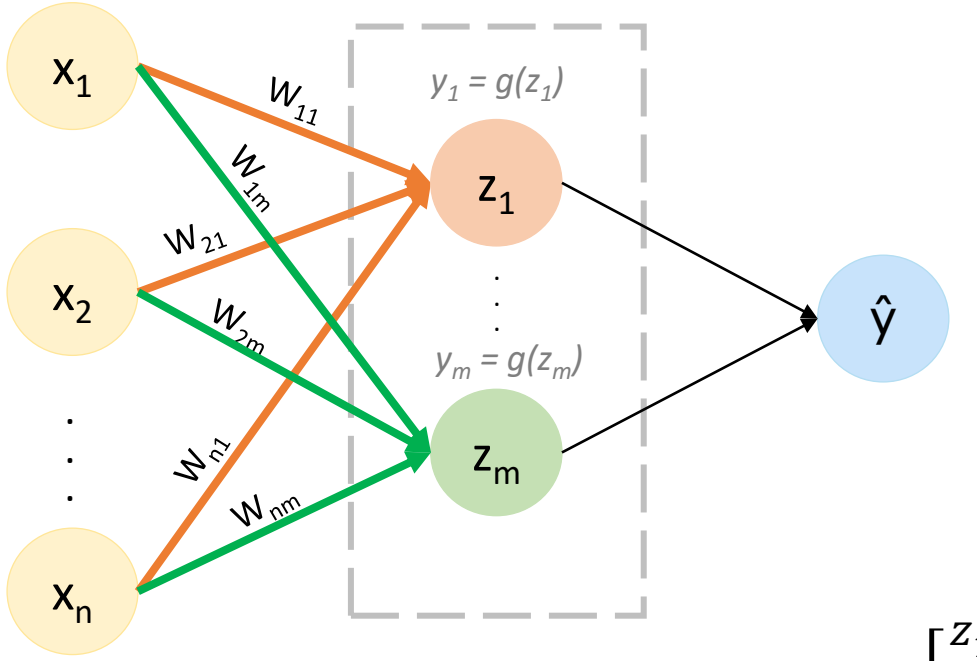
```
#Dependencies
import keras
from keras.models import Sequential
from keras.layers import Dense


# Neural network
model = Sequential()
model.add(Dense(16, input_dim=20, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

Output Layer


BRACE YOURSELVES
MATH IS COMING
memegenerator.net


x_1
x_2
x_n
$y_1 = g(z_1)$
z_1
$y_m = g(z_m)$
z_m
$\hat{y}$

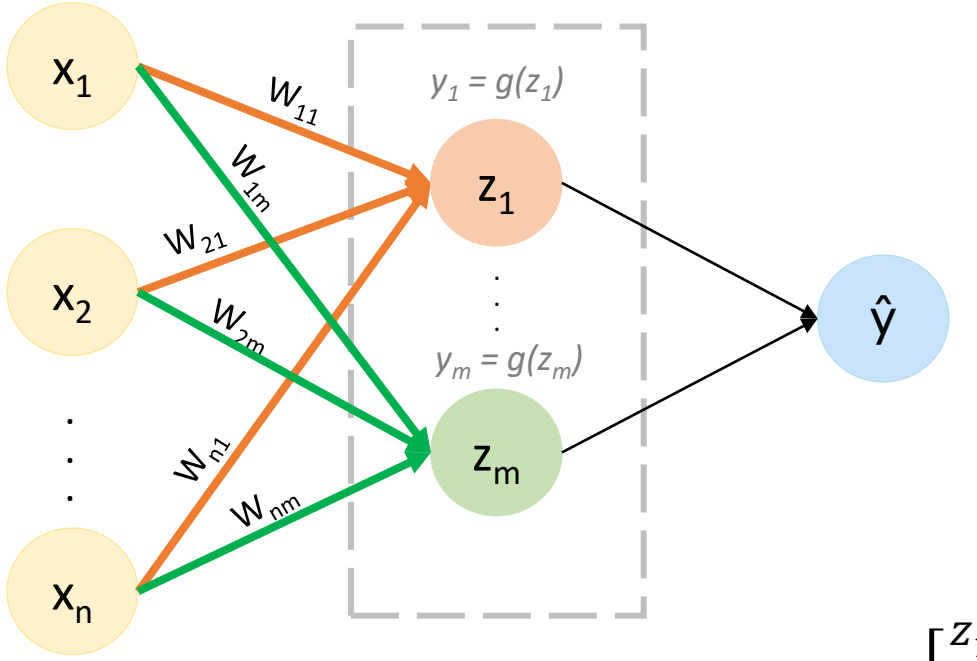

x1
x2
xn
w11
w1m
w21
w2m
wn1
wnm
y1 = g(z1)
ym = g(zm)
z1
zm
ŷ



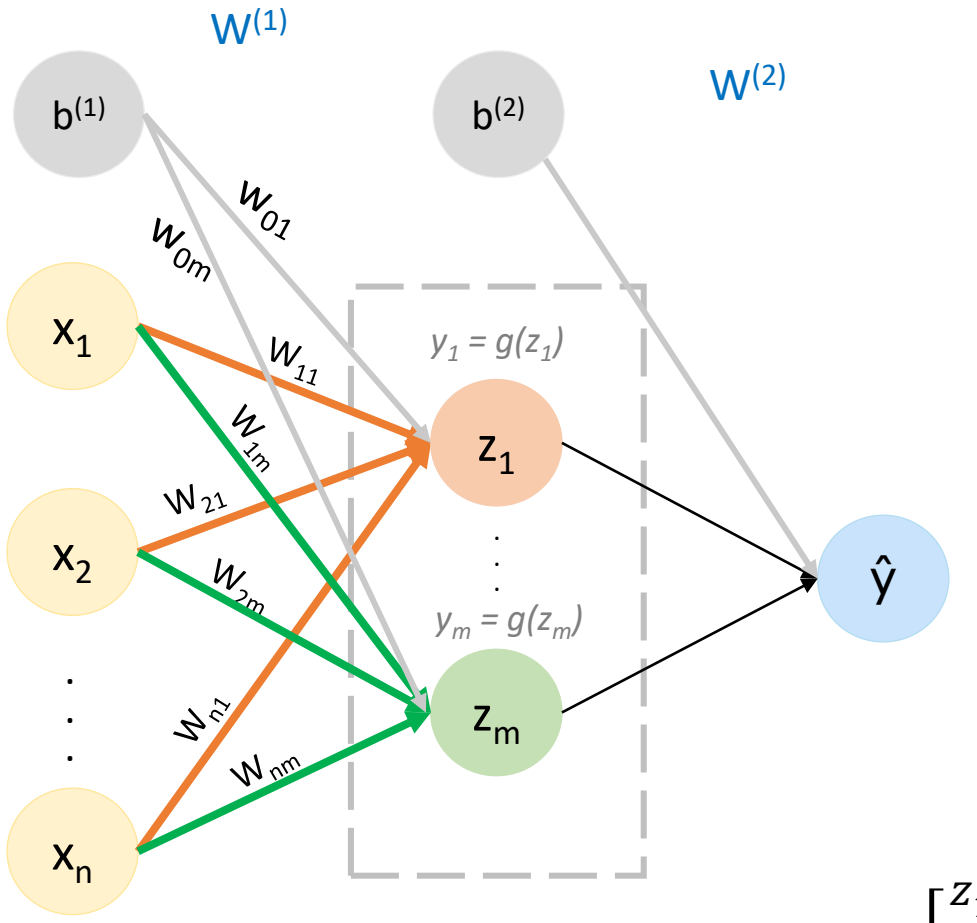
$$X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad W^{(1)} = \begin{bmatrix} w_{11}\, w_{21} & \dots w_{n1} \\ w_{21}\, w_{22} & \dots w_{n2} \\ & . \\ & . \\ & . \\ w_{m1}\, w_{m2} & \dots w_{mn} \end{bmatrix} \quad Z = \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ . \\ z_m \end{bmatrix}$$

$W^{(1)}$

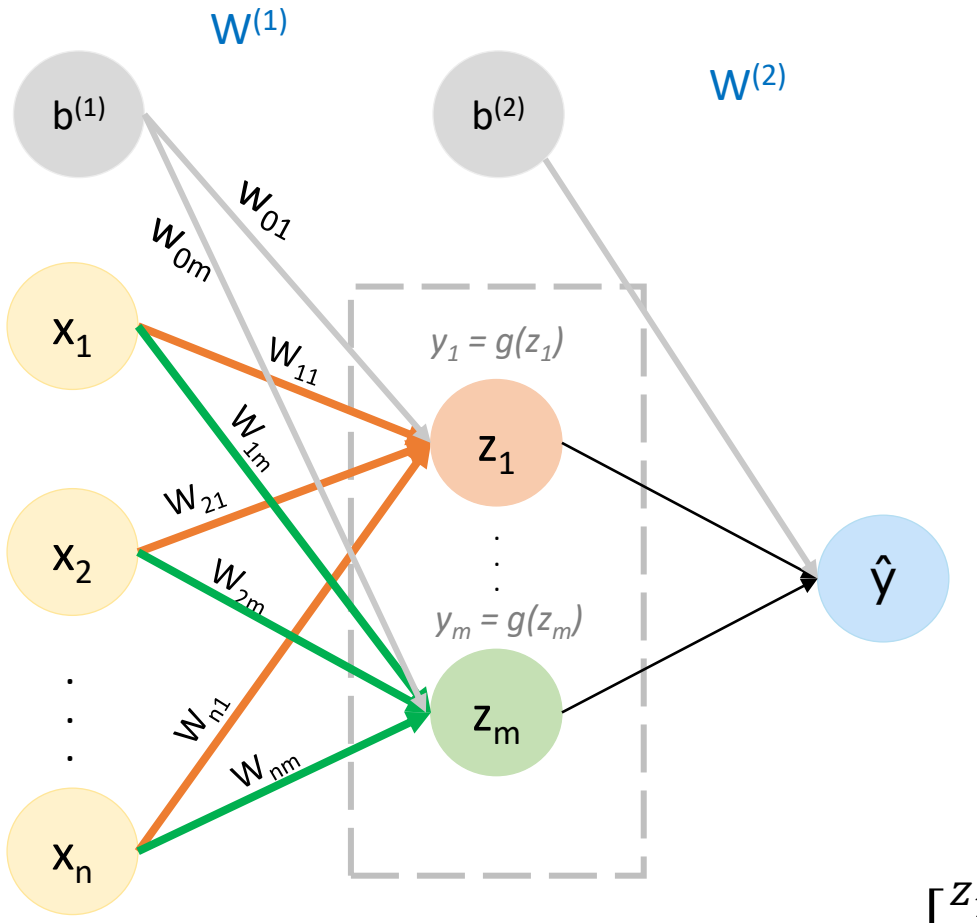
$W^{(2)}$



$$X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad W^{(1)} = \begin{bmatrix} w_{11}\, w_{21} & \dots w_{n1} \\ w_{21}\, w_{22} & \dots w_{n2} \\ & . \\ & . \\ & . \\ w_{m1}\, w_{m2} & \dots w_{mn} \end{bmatrix} \quad Z = \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ . \\ z_m \end{bmatrix}$$

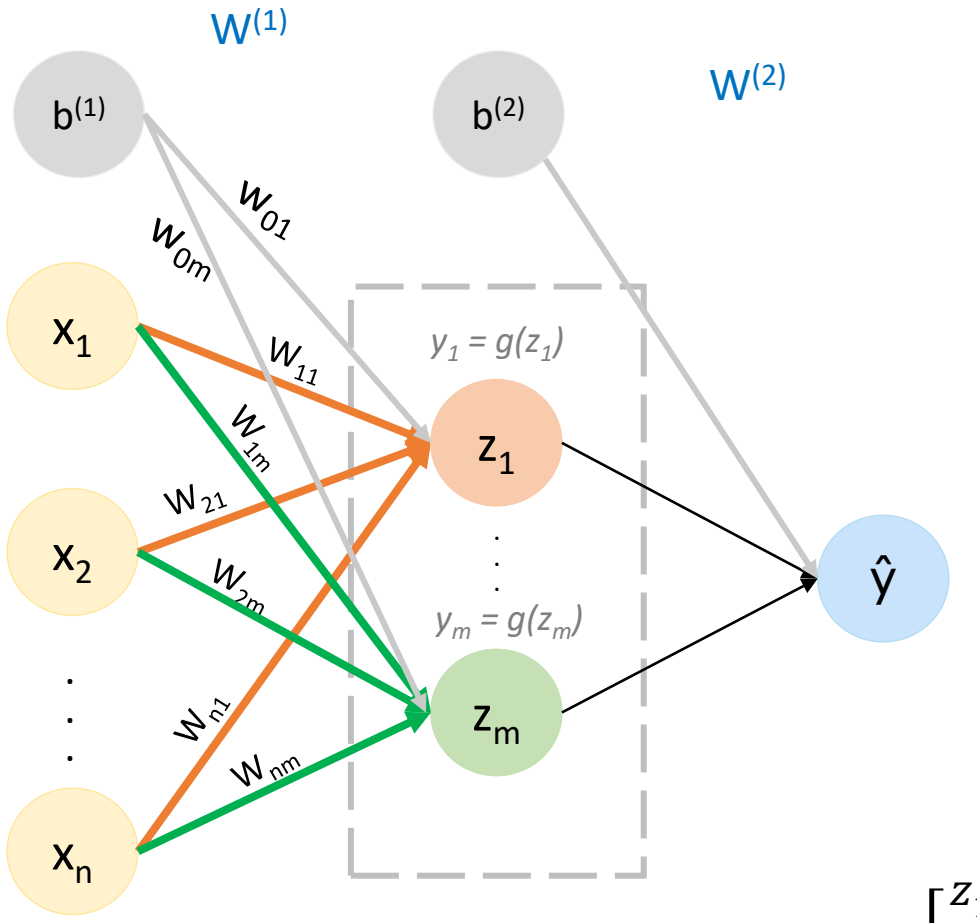


$$X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad W^{(1)} = \begin{bmatrix} w_{11}\, w_{21} & \dots w_{n1} \\ w_{21}\, w_{22} & \dots w_{n2} \\ & . \\ & . \\ & . \\ w_{m1}\, w_{m2} & \dots w_{mn} \end{bmatrix} \quad Z = \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ . \\ z_m \end{bmatrix}$$



$$z_j = w_{0j}^{(1)} + \sum_{i=1}^{n} x_i w_{ij}^{(1)}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad W^{(1)} = \begin{bmatrix} w_{11}\, w_{21} & \dots w_{n1} \\ w_{21}\, w_{22} & \dots w_{n2} \\ & . \\ & . \\ & . \\ w_{m1}\, w_{m2} & \dots w_{mn} \end{bmatrix} \quad Z = \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ . \\ z_m \end{bmatrix}$$

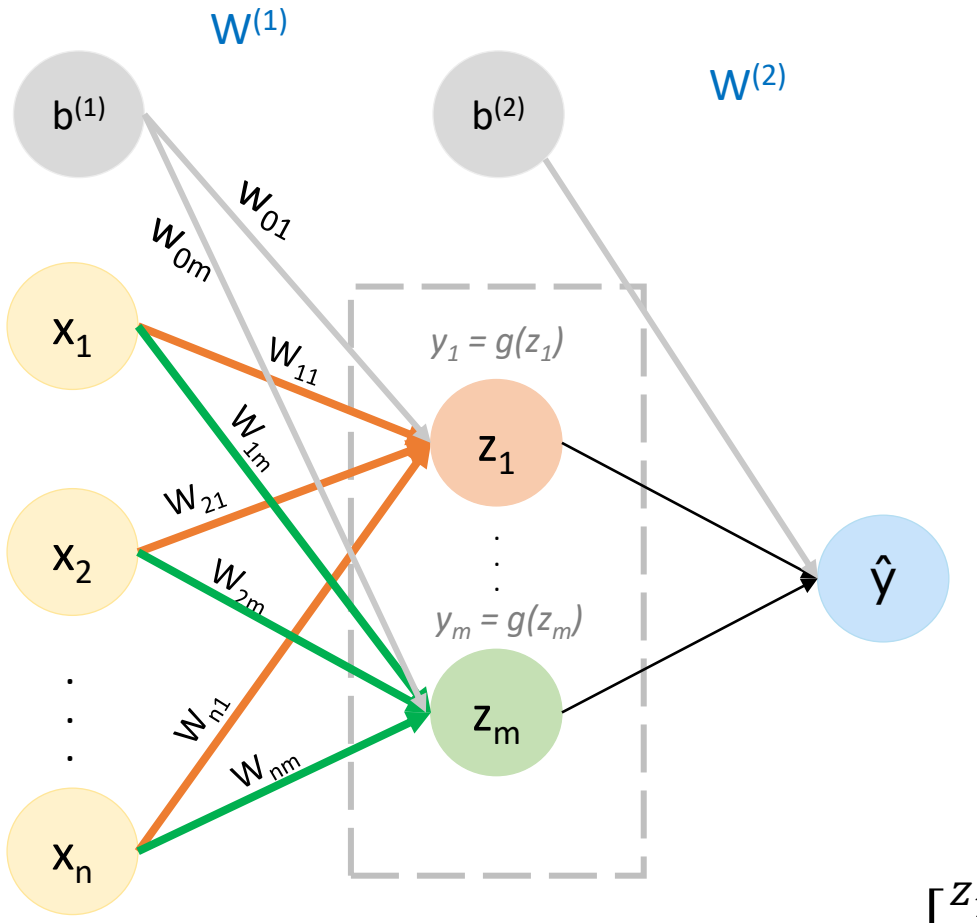


$$z_j = w_{0j}^{(1)} + \sum_{i=1}^{n} x_i w_{ij}^{(1)}$$

$$z_1 = w_{01}^{(1)} + \sum_{i=1}^{n} x_i w_{i1}^{(1)}$$

$$z_1 = w_{01}^{(1)} + x_1 w_{11}^{(1)} + x_2 w_{21}^{(1)} + x_n w_{n1}^{(1)}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad W^{(1)} = \begin{bmatrix} w_{11}\, w_{21} & \dots w_{n1} \\ w_{21}\, w_{22} & \dots w_{n2} \\ & . \\ & . \\ & . \\ w_{m1}\, w_{m2} & \dots w_{mn} \end{bmatrix} \quad Z = \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ . \\ z_m \end{bmatrix}$$



$$z_j = w_{0j}^{(1)} + \sum_{i=1}^{n} x_i w_{ij}^{(1)}$$

$$z_1 = w_{01}^{(1)} + \sum_{i=1}^{n} x_i w_{i1}^{(1)}$$

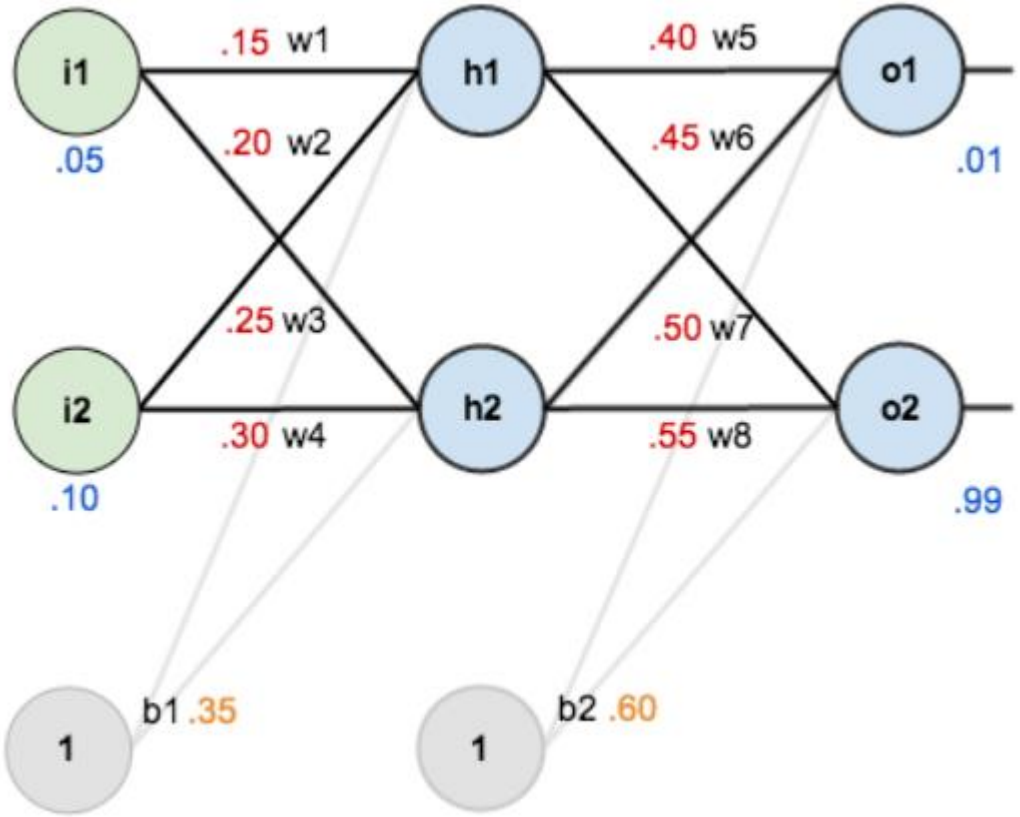
$$z_1 = w_{01}^{(1)} + x_1 w_{11}^{(1)} + x_2 w_{21}^{(1)} + x_n w_{n1}^{(1)}$$

Final output:

$$\hat{y}_k = g(w_{0k}^{(2)} + \sum_{j=1}^{m} g(z_j) w_{jk}^{(2)})$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad W^{(1)} = \begin{bmatrix} w_{11}\, w_{21} & \dots w_{n1} \\ w_{21}\, w_{22} & \dots w_{n2} \\ & . \\ & . \\ & . \\ w_{m1}\, w_{m2} & \dots w_{mn} \end{bmatrix} \quad Z = \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ . \\ z_m \end{bmatrix}$$

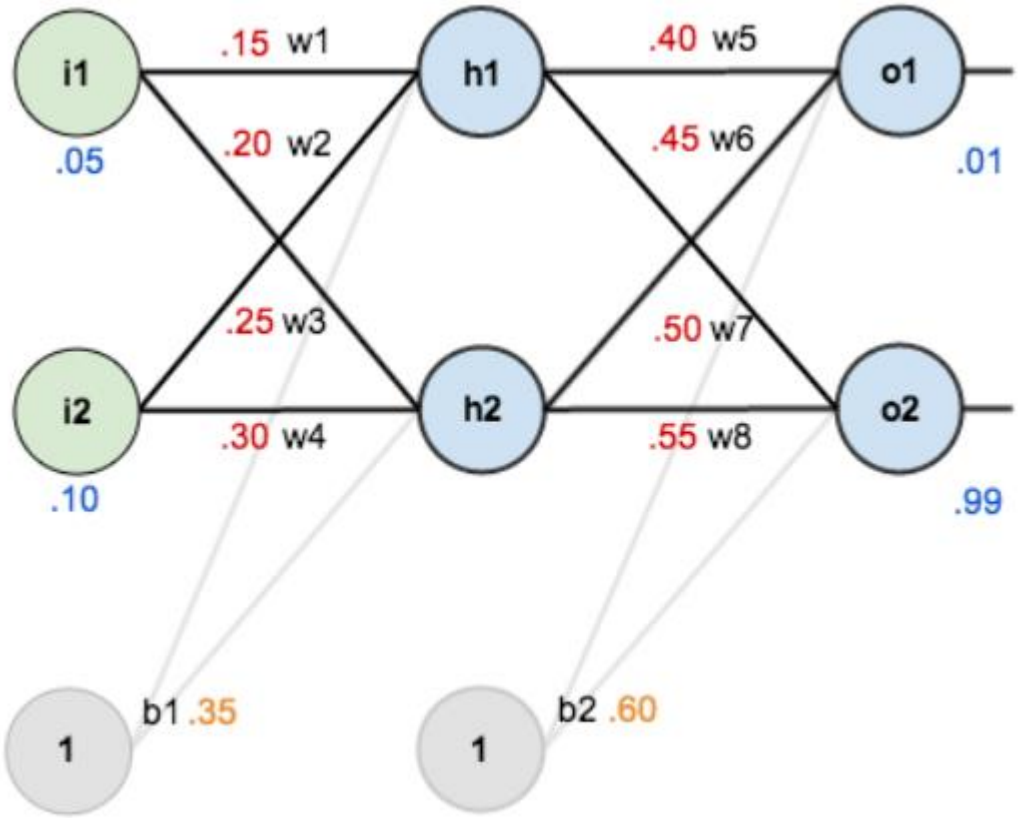
Let's consider an example



Source

Use of sigmoid function as an activation function

Computing the outputs – Forward Pass



$net_{h1} = w1*i1 + w2*i2 + 1*b1$

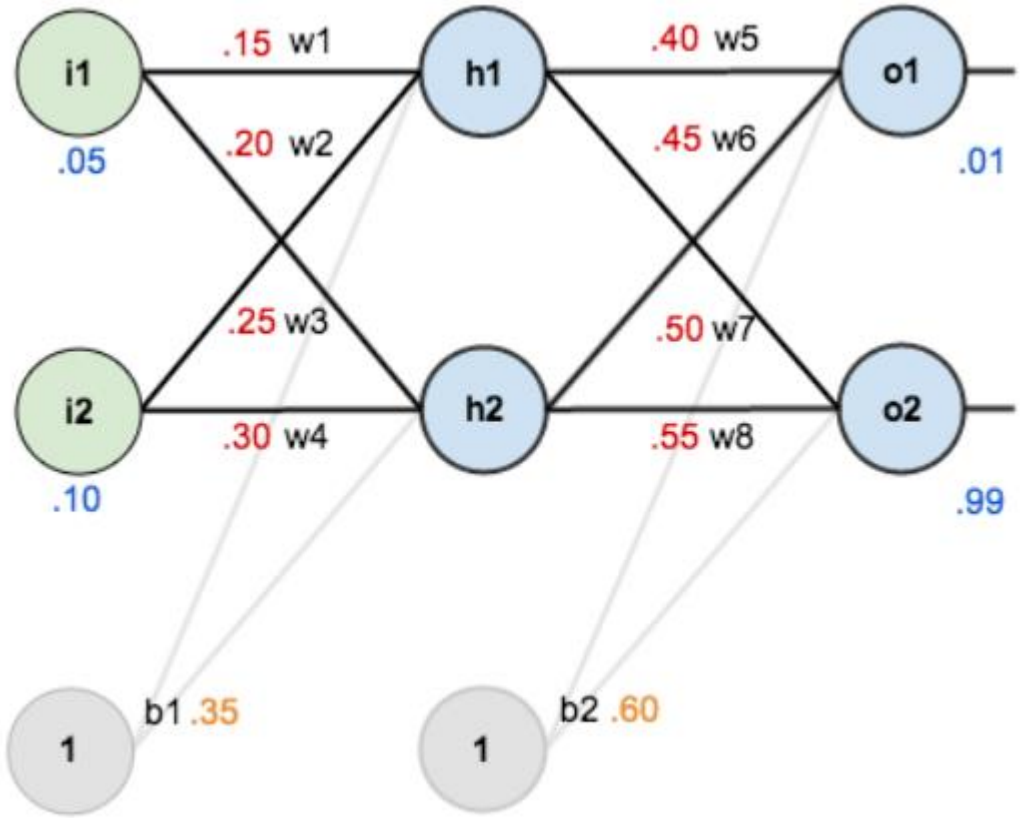
$net_{h1} = 0.15*0.05 + 0.20*0.10 + 0.35 = 0.3775$

$out_{h1} = sig(net_{h1}) = 0.593269992$

Source

Use of sigmoid function as an activation function

Computing the outputs – Forward Pass



$net_{h1} = w1*i1 + w2*i2 + 1*b1$

$net_{h1} = 0.15*0.05 + 0.20*0.10 + 0.35 = 0.3775$

$out_{h1} = sig(net_{h1}) = 0.593269992$

$net_{h2} = w3*i1 + w4*12 + 1*b1$

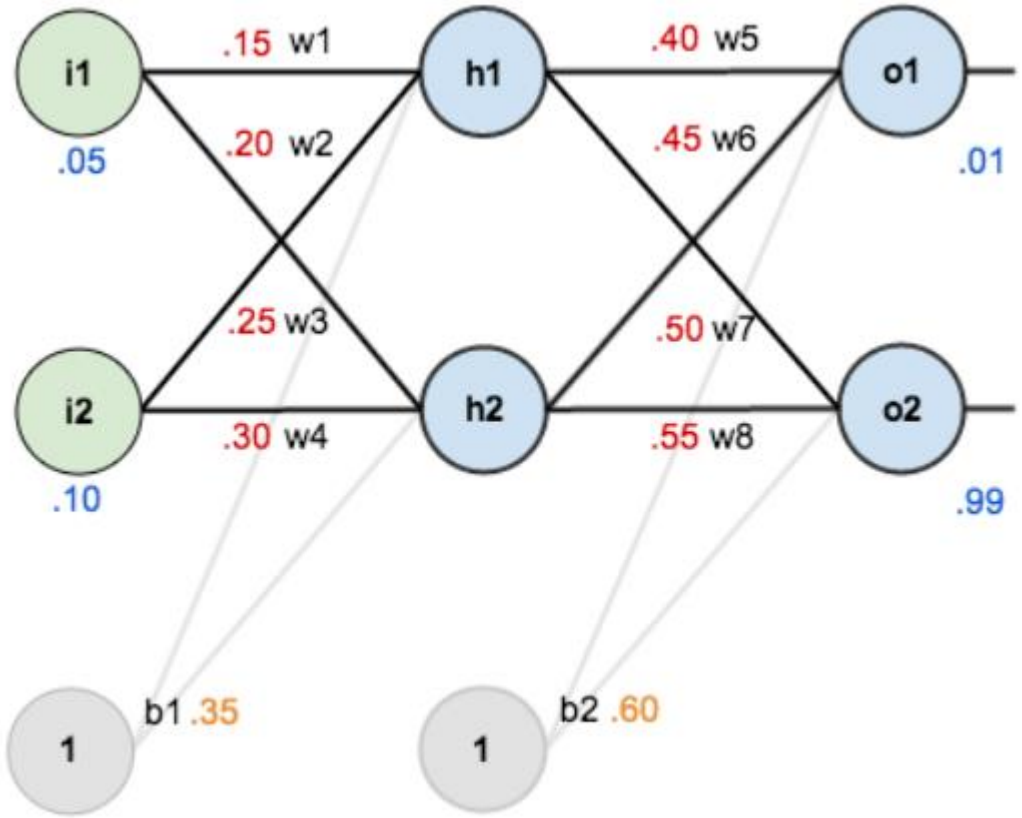
$net_{h2} = 0.25*0.05 + 0.30*0.10 + 0.35 = 0.3925$

$out_{h2} = sig(net_{h2}) = 0.596884378$

Source

Use of sigmoid function as an activation function

Computing the outputs – Forward Pass



Source

Use of sigmoid function as an activation function

$net_{h1} = w1*i1 + w2*i2 + 1*b1$

$net_{h1} = 0.15*0.05 + 0.20*0.10 + 0.35 = 0.3775$

$out_{h1} = sig(net_{h1}) = 0.593269992$

$net_{h2} = w3*i1 + w4*12 + 1*b1$

$net_{h2} = 0.25*0.05 + 0.30*0.10 + 0.35 = 0.3925$

$out_{h2} = sig(net_{h2}) = 0.596884378$

$net_{o1} = w5*out_{h1} + w6*out_{h2} + 1*b2$

$net_{o1} = 0.40*0.593269992 + 0.45*0.596884378 + 0.60 = 1.105905967$

$out_{o1} = sig(net_{o1}) = 0.75136507$

Similarly, $out_{o2} = 0.772928465$

Did the model learn?
How well the model performed?

- ✓ Take data as *input* through input nodes.
- ✓ *Train* themselves to understand patterns in the data.
- ✓ Predict *outputs*.
- ✓ Evaluate model.
- ✓ Optimise model's performance.

Did the model learn?
How well the model performed?

- ✓ Take data as *input* through input nodes.
- ✓ *Train* themselves to understand patterns in the data.
- ✓ Predict *outputs*.
- ✓ Evaluate model.
- ✓ Optimise model's performance.

- *Loss*: quantification of the deviation of the predicted O/P by the NN from the expected O/P.

  ✓ *Loss function/Cost function* : A function used to calculate the loss.

$$= y_i - \hat{y}_i$$

$$= y_i - f(x_i, w)$$

  ✓ The higher the loss, the worse the model performs.
  ✓ *Goal*: Minimise the loss and optimize the model's performance.

"The function we want to minimise or maximise is called the **objective function or criterion**. When we are minimising it, we may also call it the cost function, loss function, or error function".

~ *Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.*

Loss functions

a.) Regression Loss

- Mean Squared Error (L2 Loss)
  - ✓ Penalizes the model for making large errors by squaring them.

$$MSE = J(W) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

tf.keras.losses.MSE(y_true,y_pred)

- Root Mean Squared Error

$$RMSE = J(W) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- tf.sqrt(tf.reduce_mean(tf.squared_difference(y_true,y_pred)))
- tf.sqrt(tf.reduce_sum(tf.squared_difference(y_true,y_pred)))

Loss functions

a.) Regression Loss

- Mean Squared Error (L2 Loss)
  - ✓ Penalizes the model for making large errors by squaring them.

$$MSE = J(W) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

tf.keras.losses.MSE(y_true,y_pred)

- Root Mean Squared Error

$$RMSE = J(W) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- tf.sqrt(tf.reduce_mean(tf.squared_difference(y_true,y_pred)))
- tf.sqrt(tf.reduce_sum(tf.squared_difference(y_true,y_pred)))

- Mean Absolute Error (L1 Loss)
  - ✓ More robust towards outliers than MSE.

$$MAE = J(W) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

tf.keras.losses.MAE(y_true,y_pred)

- Mean Bias Error

$$MBE = J(W) = \frac{1}{n}\sum_{i=1}^{n} y_i - \hat{y}_i$$

Loss functions

b.) Classification Loss

i. Binary Classification

- Binary Cross-Entropy Loss (aka Logarithmic Loss)
  - ✓ Use of sigmoid or softmax activation function

$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y_true,y_pred))

- Hinge Loss
  - ✓ Intended for where the target values are in the set {-1, 1}.
  - ✓ Initially developed to use with the SVM models.

$$J(W) = \max(0, 1 - y_i * \hat{y}_i)$$

tf.keras.losses.Hinge(y_true,y_pred)

Loss functions

b.) Classification Loss

ii. Multi-Class Classification

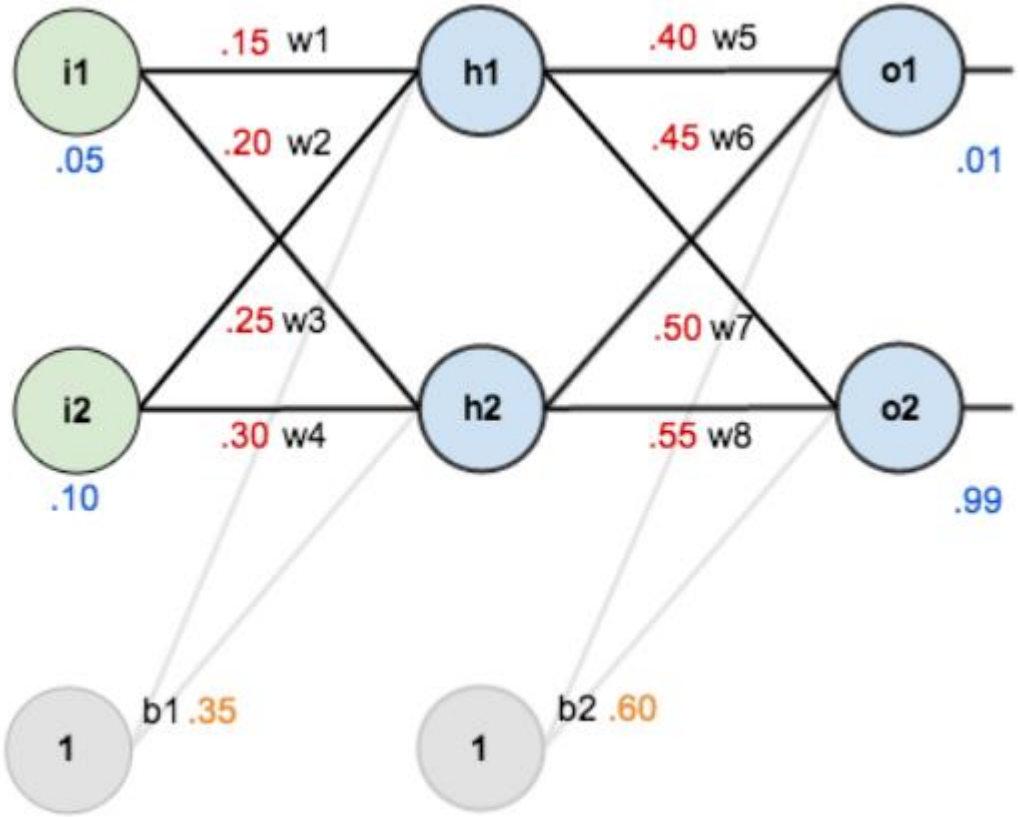
- Categorical Cross-Entropy Loss (aka Logarithmic Loss)
  - ✓A generalisation of the binary cross-entropy loss


tf.keras.losses.CategoricalCrossentropy(y_true,y_pred)

- Kullback Leibler Divergence Loss
  - ✓A measure of how a distribution varies from a reference (or a baseline) distribution.
  - ✓A KL Divergence Loss of zero means that both the probability distributions are identical.


tf.keras.losses.KLDivergence(y_true,y_pred)

Continuing with the example



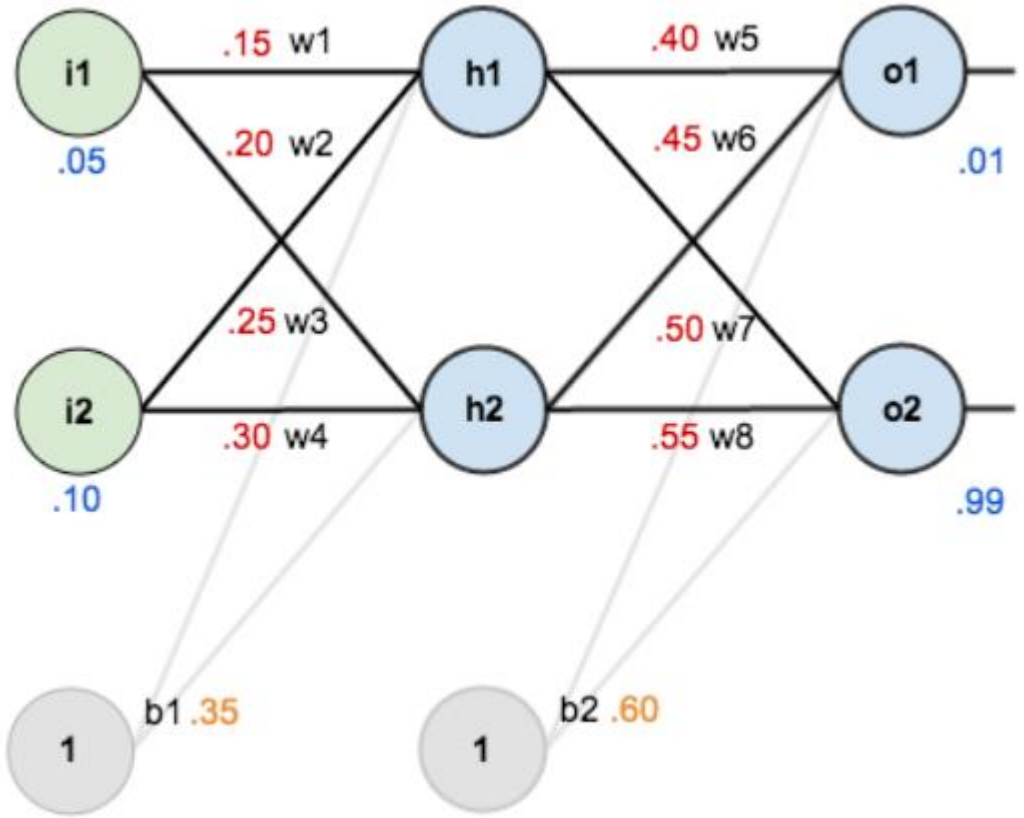
[Source]

Use of sigmoid function as an activation function

Computing the loss

$out_{o1} = sig(net_{o1}) = 0.75136507$

Similarly, $out_{o2} = 0.772928465$



[Source]

Use of sigmoid function as an activation function

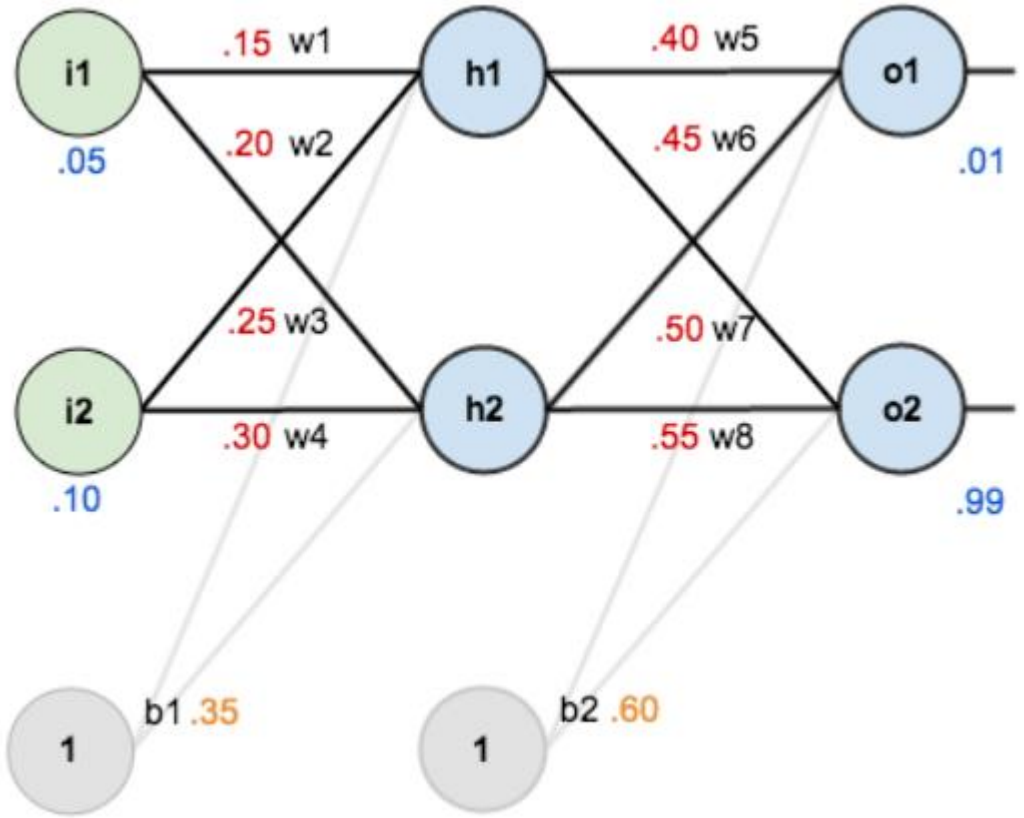
Computing the loss

$out_{o1} = sig(net_{o1}) = 0.75136507$

Similarly, $out_{o2} = 0.772928465$

$$J(W) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$J(W) = \frac{1}{2}\sum(y_i - \hat{y}_i)^2$$



[Source]

Use of sigmoid function as an activation function

Computing the loss

$out_{o1} = sig(net_{o1}) = 0.75136507$

Similarly, $out_{o2} = 0.772928465$

$$J(W) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$J(W) = \frac{1}{2}\sum(y_i - \hat{y}_i)^2$$

$$J(out_{o1}) = \frac{1}{2}(actual_{o1} - out_{o1})^2$$

$$J(out_{o1}) = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Similarly, $J(out_{02}) = 0.023560026$

$J(W)_{total} = J(out_{01}) + J(out_{o2}) = 0.298371109$

Minimising the Loss

Formalising the Optimization Problem

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}_i, y_i)$$

$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i, w), y_i)$$

Optimization: minimising the loss function by updating the network weights to achieve the ***lowest loss***.

Optimizers

- Gradient Descent
- Stochastic Gradient Descent
- Adagrad (Adaptive Gradient)
- Adadelta
- RMSprop (Root Mean Square Propagation)
- Adam (Adaptive Moment Estimation)


tf.keras.optimizers.SGD(learning_rate=0.01)


tf.keras.optimizers.Adagrad(learning_rate=0.01)


tf.keras.optimizers.Adadelta(learning_rate=0.01)


tf.keras.optimizers.RMSprop(learning_rate=0.01)


tf.keras.optimizers.Adam(learning_rate=0.01)

Gradient Descent

- An iterative algorithm that helps in computing the optimal values of the weights that minimize the loss function.
- Based on the idea of following the local slope of the loss landscape.
- Gradient: Derivative (slope) of a function.

Gradient Descent

$$\hat{y} = g(w_0 + w_1 x)$$

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

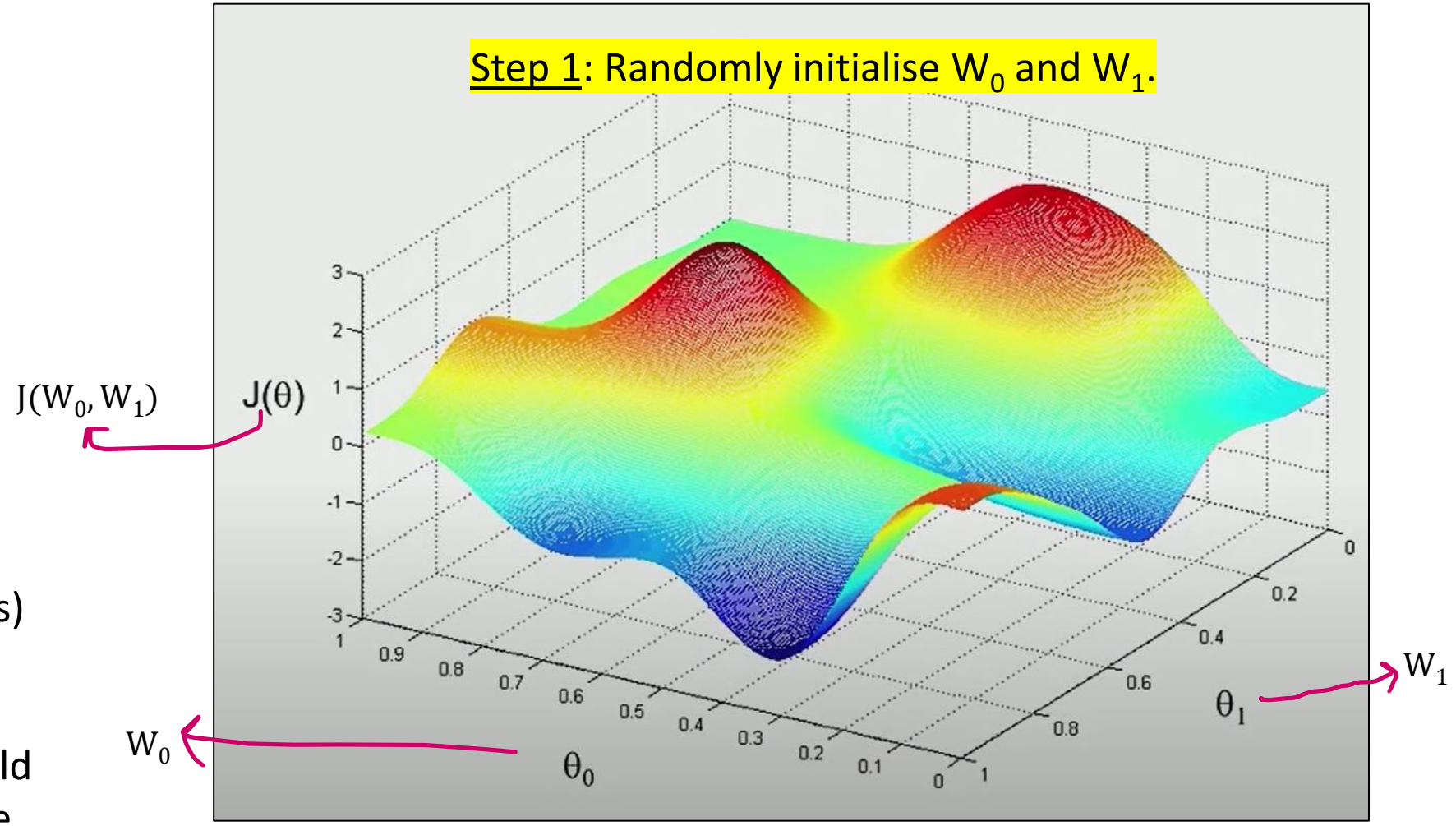
- Interpreting loss as a descending a high-dimensional landscape.

Gradient Descent

$$\hat{y} = g(w_0 + w_1 x)$$

$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$

- Interpreting loss as a descending a high-dimensional landscape.
- For a 2D plot, the height (z axis) of the landscape would be the value of the loss function and the horizontal (x & y) axis would be the value of the weights (i.e., w_0 and w_1)



The Loss Landscape

Image Credits: Andrew Ng

- Goal: find the **minima** for the loss function, i.e., reach the lowest point of the landscape by iteratively exploring the space, thus, reaching a minimal loss value and optimal values of w_0 and w_1.

Gradient Descent

- An iterative algorithm that helps in computing the optimal values of the weights that minimize the loss function.
- Based on the idea of following the local slope of the loss landscape.
- Gradient: Derivative (slope) of a function.
- Take the direction of the steepest decline → direction opposite to the direction of the gradient.

- Compute the gradient. $\frac{\partial J(W)}{\partial W}$

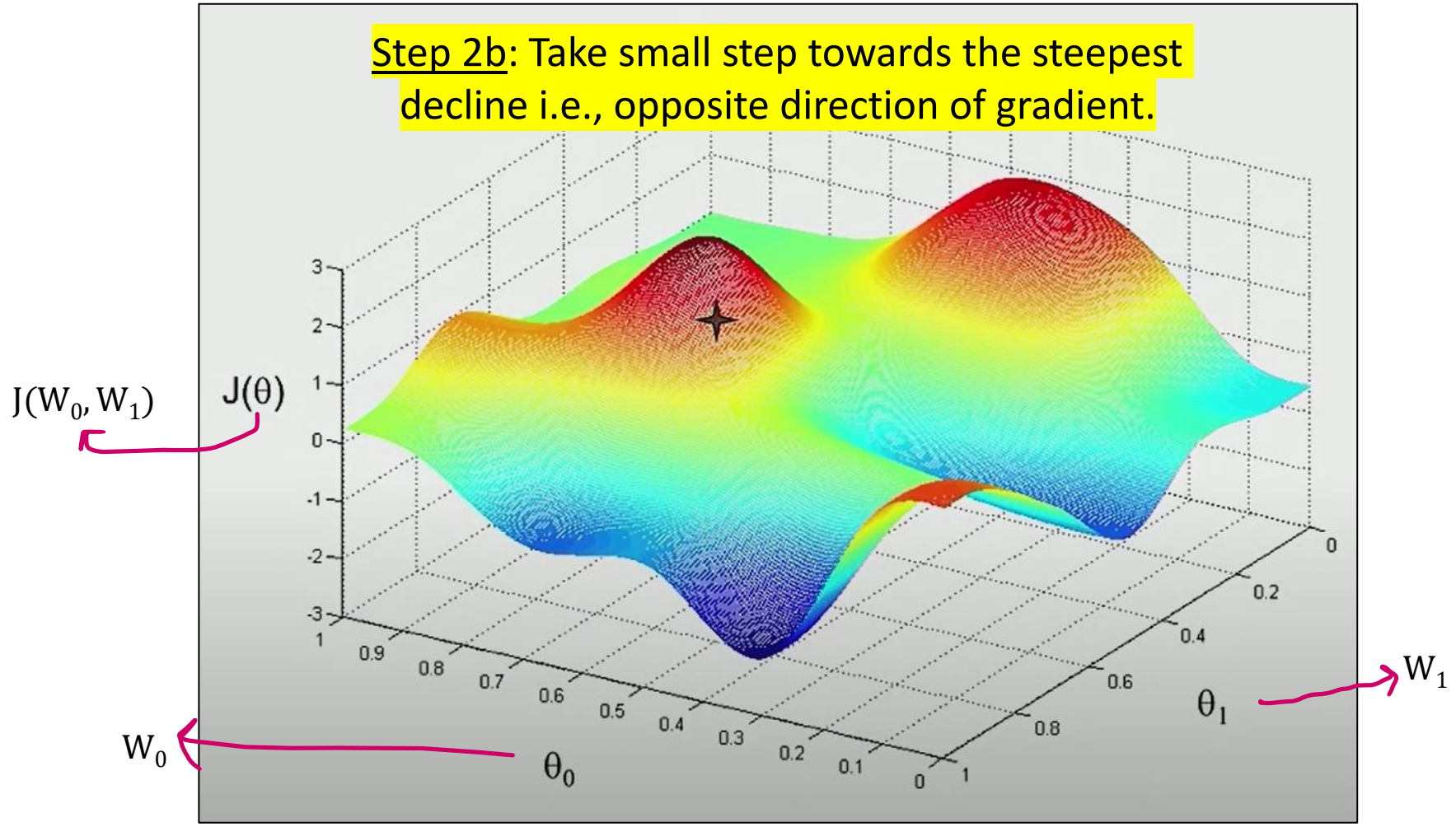
Gradient Descent

$$\hat{y} = g(w_0 + w_1 x)$$

$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$

Step 2a: Compute gradient

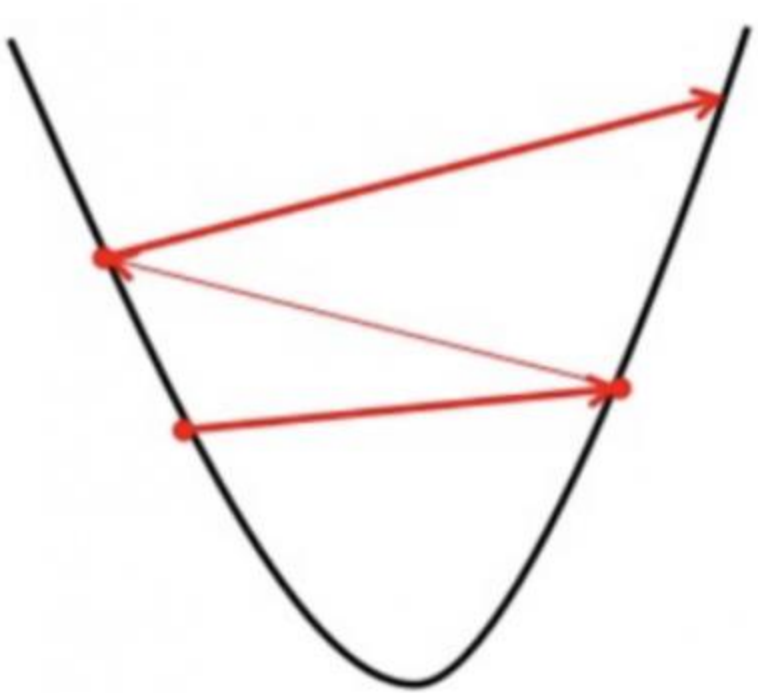
$$\frac{\partial J(W)}{\partial W}$$



The Loss Landscape

Image Credits: Andrew Ng

Big Learning Rate



Small Learning Rate

Learning Rate

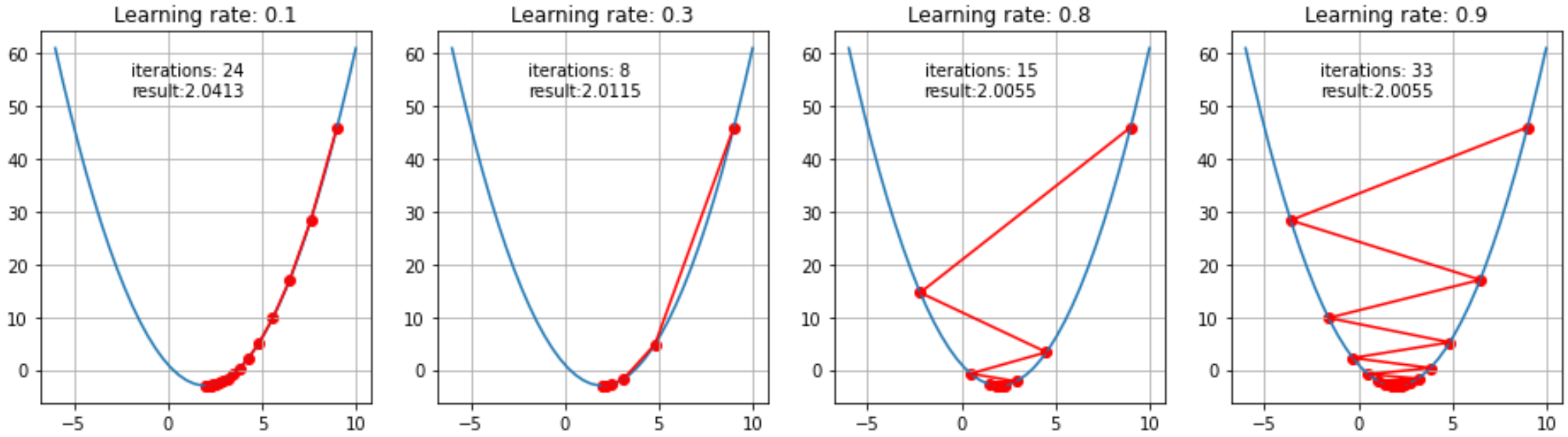


Image Source

Gradient Descent

$$\hat{y} = g(w_0 + w_1 x)$$

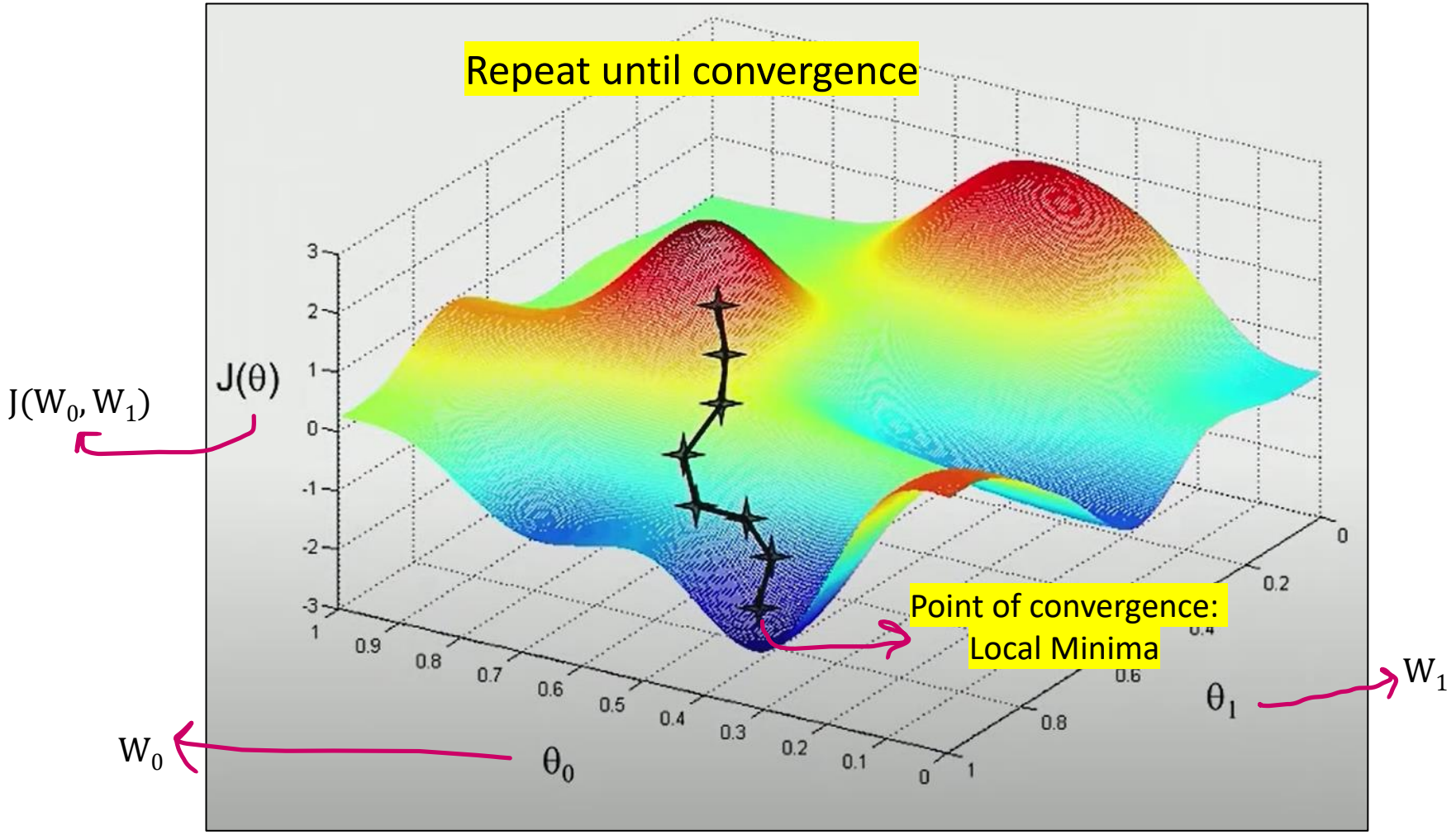
$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

Step 2a: Compute gradient

$$\frac{\partial J(W)}{\partial W}$$

Step 3: Update the weights

$$W \leftarrow W - \alpha \frac{\partial J(W)}{\partial W}$$



The Loss Landscape

Image Credits: Andrew Ng

Gradient Descent

$$\hat{y} = g(w_0 + w_1 x)$$

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

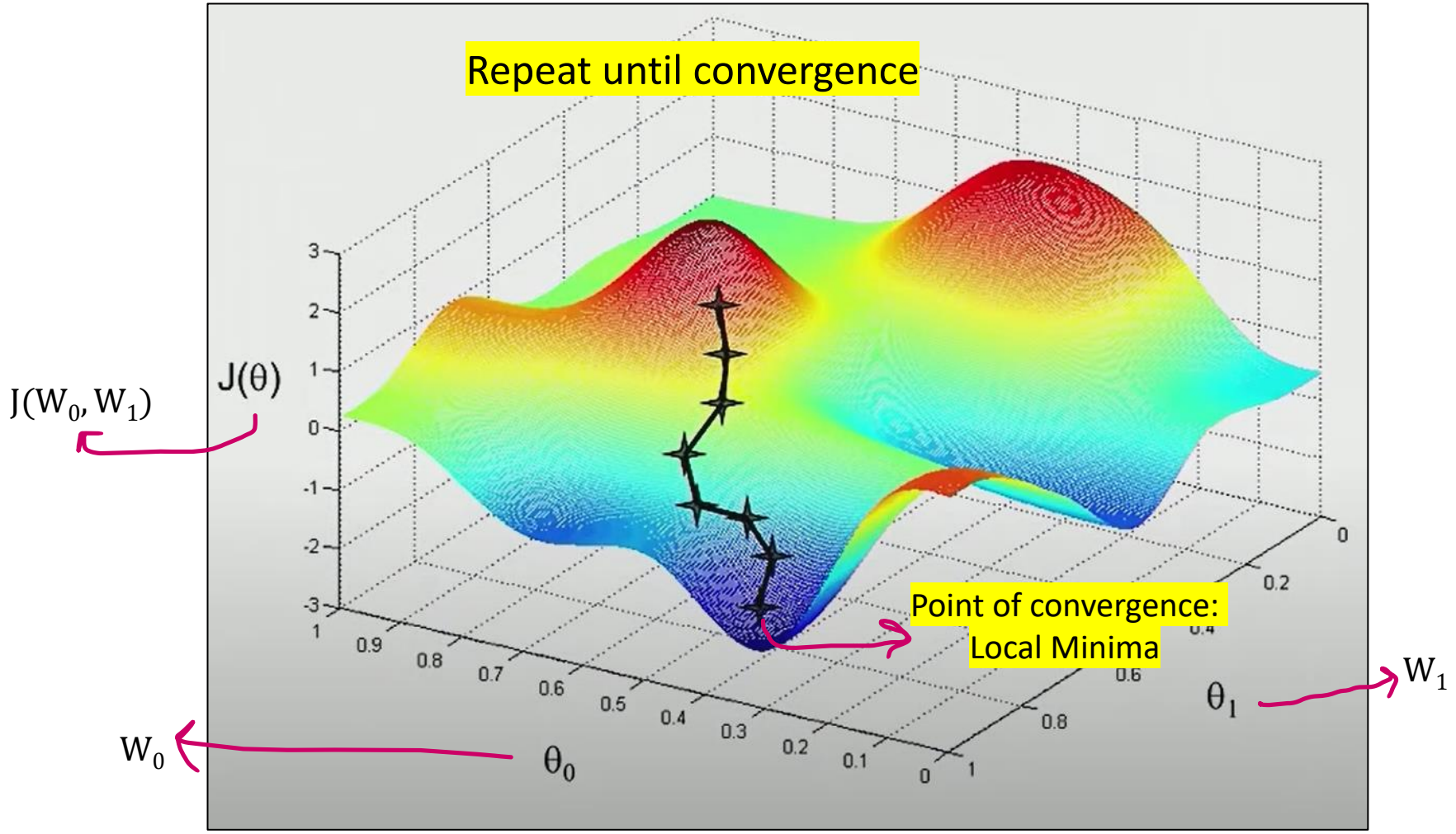
Step 2a: Compute gradient

$$\frac{\partial J(W)}{\partial W}$$

Step 3: Update the weights

$$W \leftarrow W - \alpha \frac{\partial J(W)}{\partial W}$$

Learning rate



The Loss Landscape

Image Credits: Andrew Ng

Gradient Descent

$$\hat{y} = g(w_0 + w_1 x)$$

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

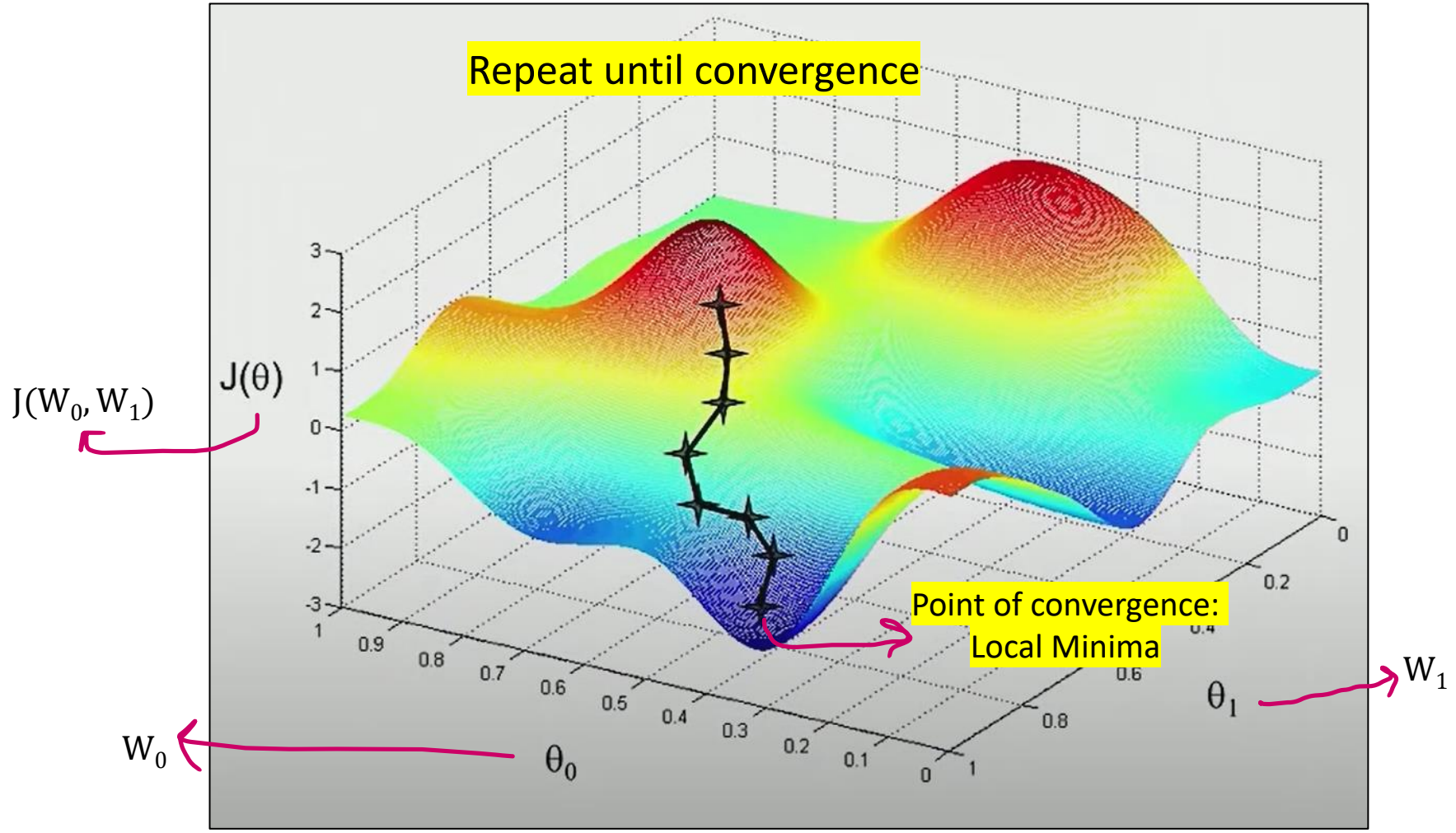
Step 2a: Compute gradient

$$\frac{\partial J(W)}{\partial W}$$

Step 3: Update the weights

$$W \leftarrow W - \alpha \frac{\partial J(W)}{\partial W}$$

Learning rate



The Loss Landscape

Image Credits: Andrew Ng

Finding an optimal value of learning rate:

- Hit and trial.
- Train a network starting from a low learning rate (say 0.01) and increase the learning rate exponentially for every batch, Smith 2017.

Gradient Descent

$$\hat{y} = g(w_0 + w_1 x)$$

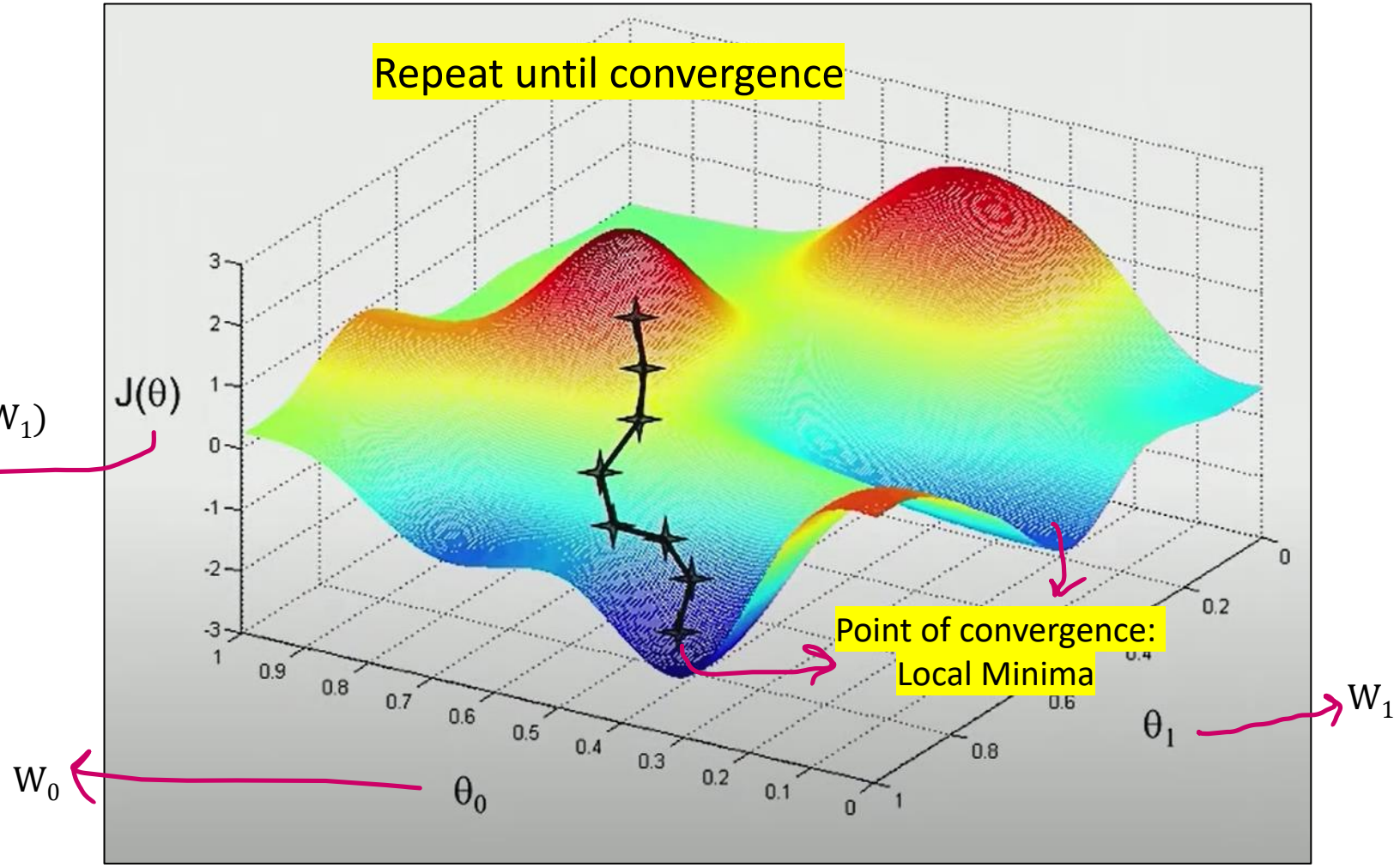
$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

Compute gradient

$$\frac{\partial J(W)}{\partial W}$$

Update the weights

$$W \leftarrow W - \alpha \frac{\partial J(W)}{\partial W}$$



The Loss Landscape

Image Credits: Andrew Ng

Gradient Descent: Algorithm

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate = 0.5)
train = optimizer.minimize(loss)
```

1. Initialise weights randomly.
2. Repeat until convergence:
   - Compute gradient $\frac{\partial J(W)}{\partial W}$
   - Update weights

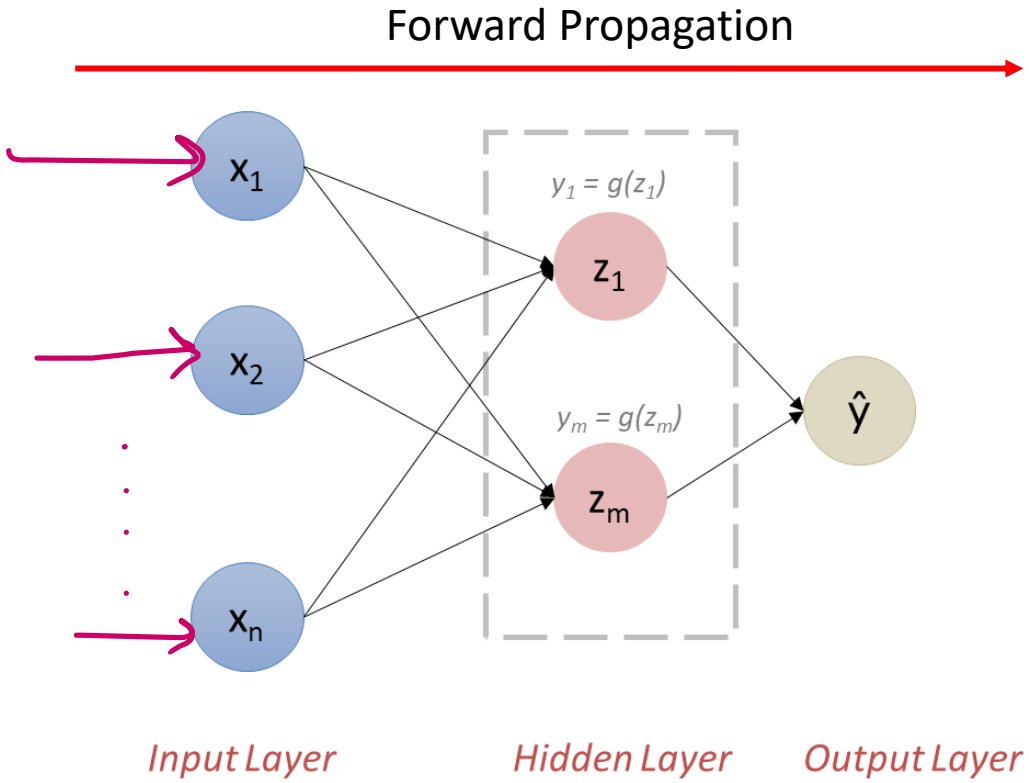
$$W_j \leftarrow W_j - \alpha \frac{\partial J(W_0, W_1)}{\partial W_j}$$

$$for\ j = 0\ and\ j = 1$$

3. Return weights.

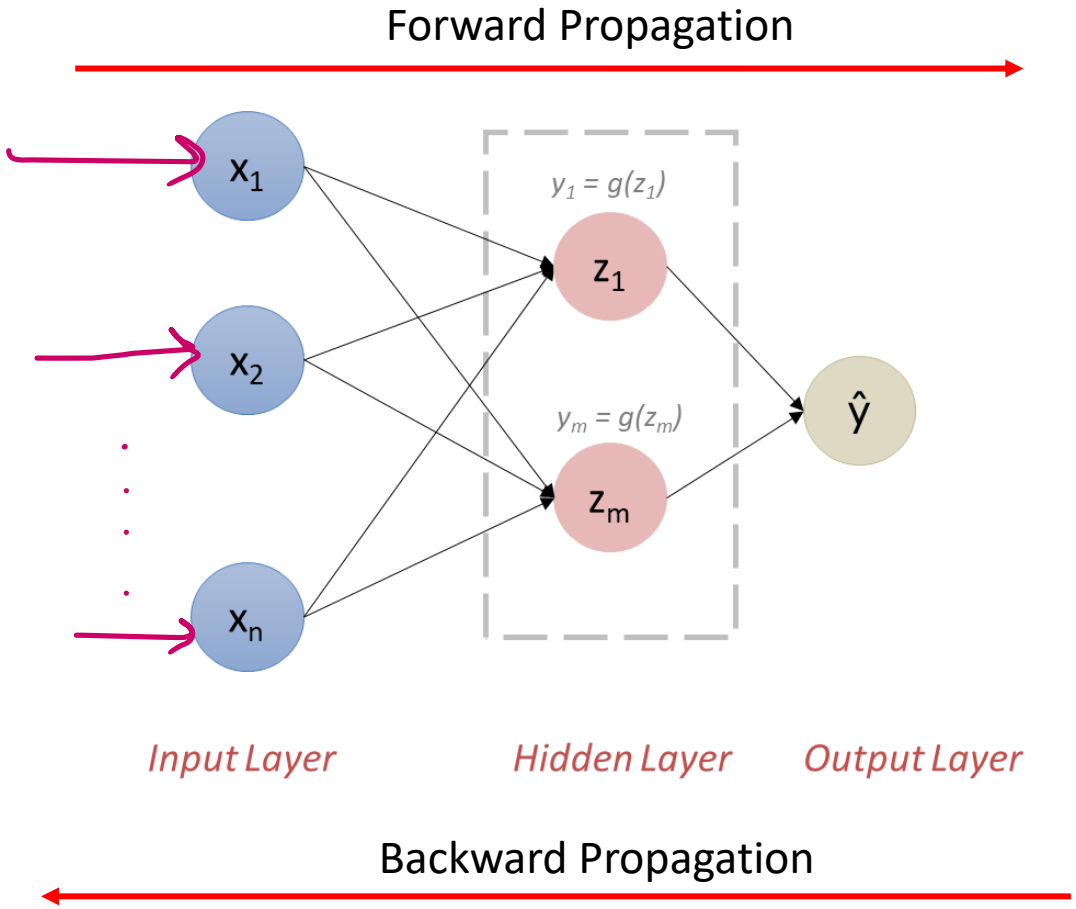
MLP/ML-FFNN: Forward Propagation

- Input the data.
- Randomly initialised the weights and bias.
- Calculated the activation functions for all neurons in the hidden layer(s).
- Predicted an output (ŷ).
- Calculated loss function by comparing expected output with predicted output.



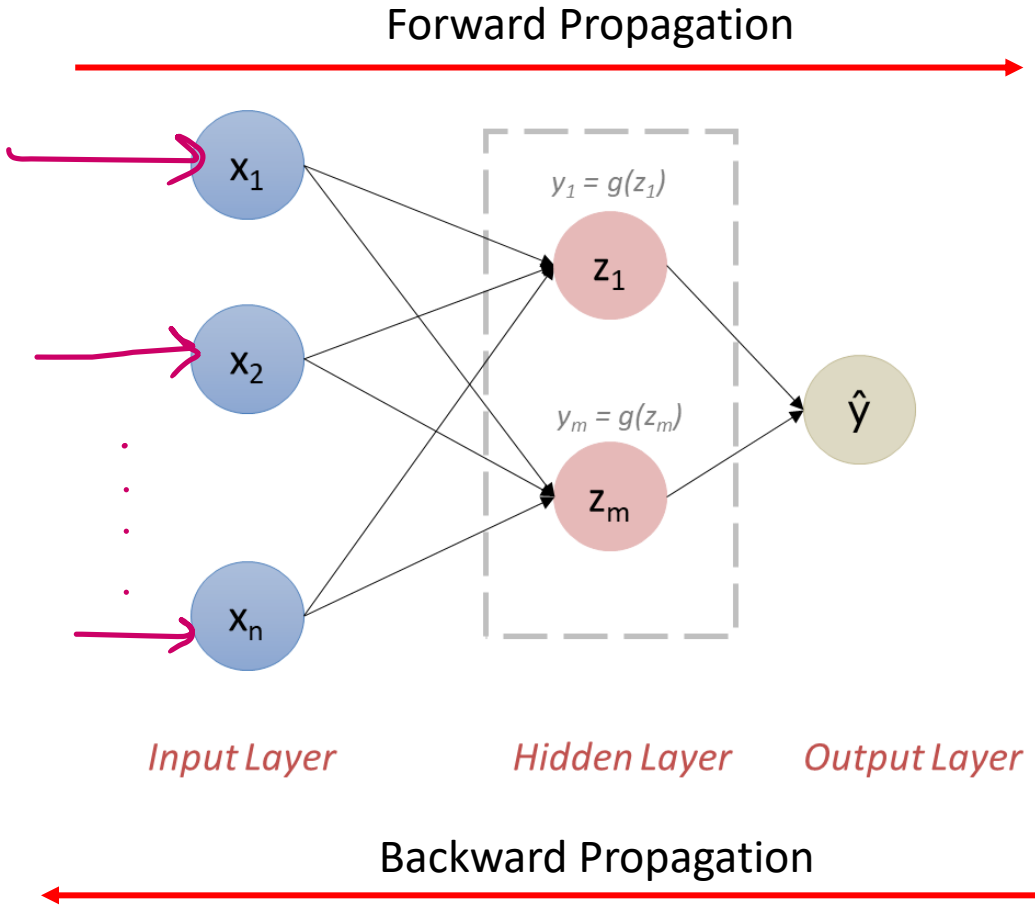
MLP/ML-FFNN: Forward Propagation

- Input the data.
- Randomly initialised the weights and bias.
- Calculated the activation functions for all neurons in the hidden layer(s).
- Predicted an output (ŷ).
- Calculated loss function by comparing expected output with predicted output.



MLP/ML-FFNN: Forward Propagation

- Input the data.
- Randomly initialised the weights and bias.
- Calculated the activation functions for all neurons in the hidden layer(s).
- Predicted an output (ŷ).
- Calculated loss function by comparing expected output with predicted output.



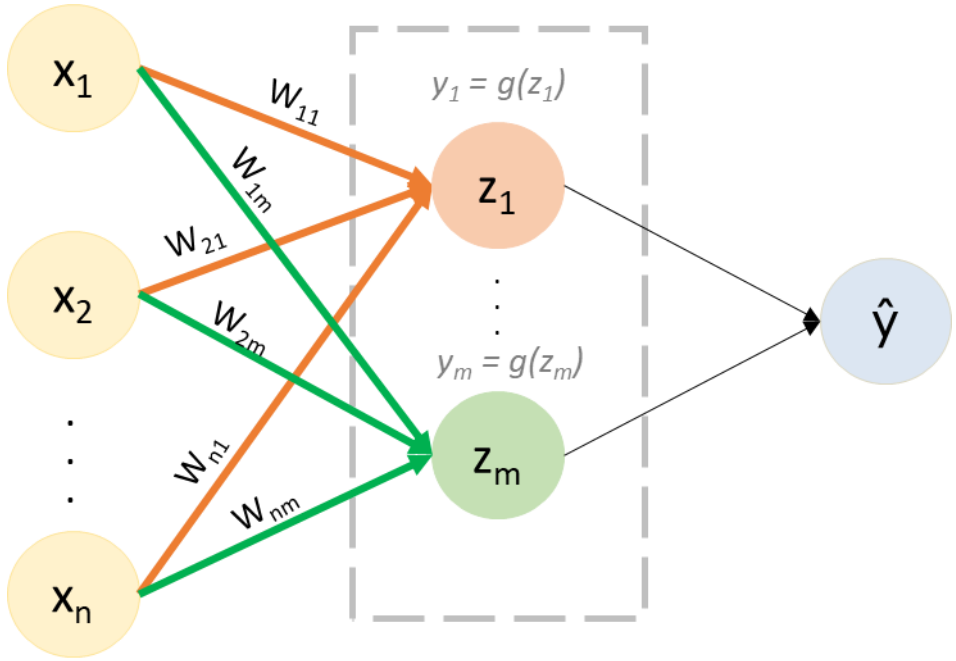
Backpropagation

- Adjusting the weights to minimize the loss function by making use of an optimizer (e.g., gradient descent).
- Use of chain rule of calculus.

Backpropagation

- The partial derivatives of the loss function w.r.t various weights and biases are backpropagated through the MLP.

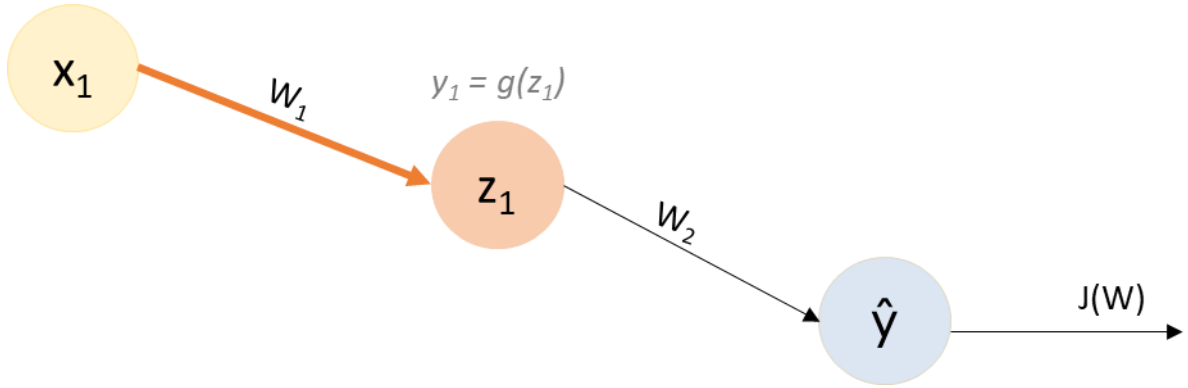
$$W_j \leftarrow W_j - \alpha \frac{\partial J(W_0, W_1)}{\partial W_j}$$



Backpropagation

- The partial derivatives of the loss function w.r.t various weights and biases are backpropagated through the MLP.

$$W_j \leftarrow W_j - \alpha \frac{\partial J(W_0, W_1)}{\partial W_j}$$

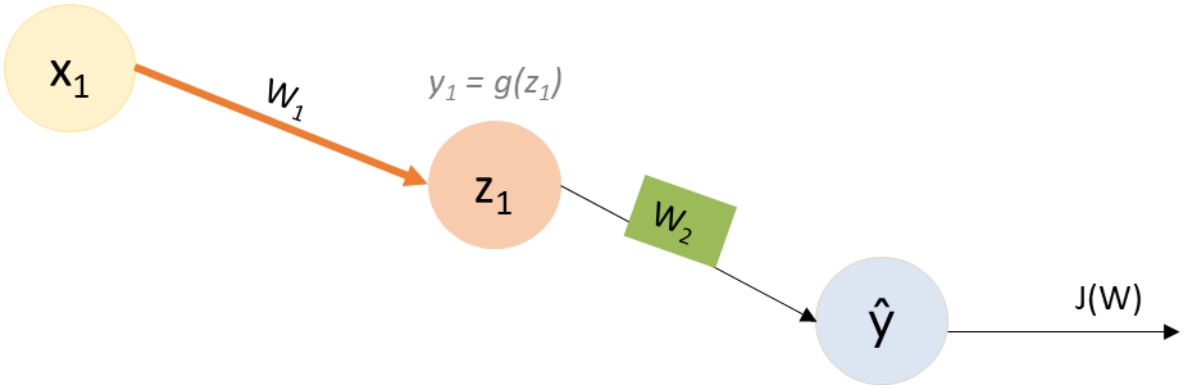


Backpropagation

- The partial derivatives of the loss function w.r.t various weights and biases are backpropagated through the MLP.

$$W_j \leftarrow W_j - \alpha \frac{\partial J(W_0, W_1)}{\partial W_j}$$

$$\frac{\partial J(W)}{\partial W_2} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial W_2}$$



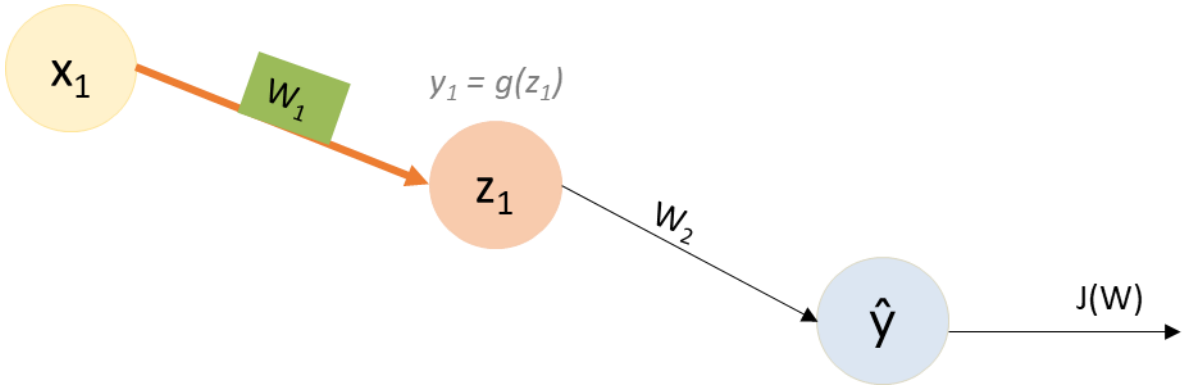
Backpropagation

- The partial derivatives of the loss function w.r.t various weights and biases are backpropagated through the MLP.

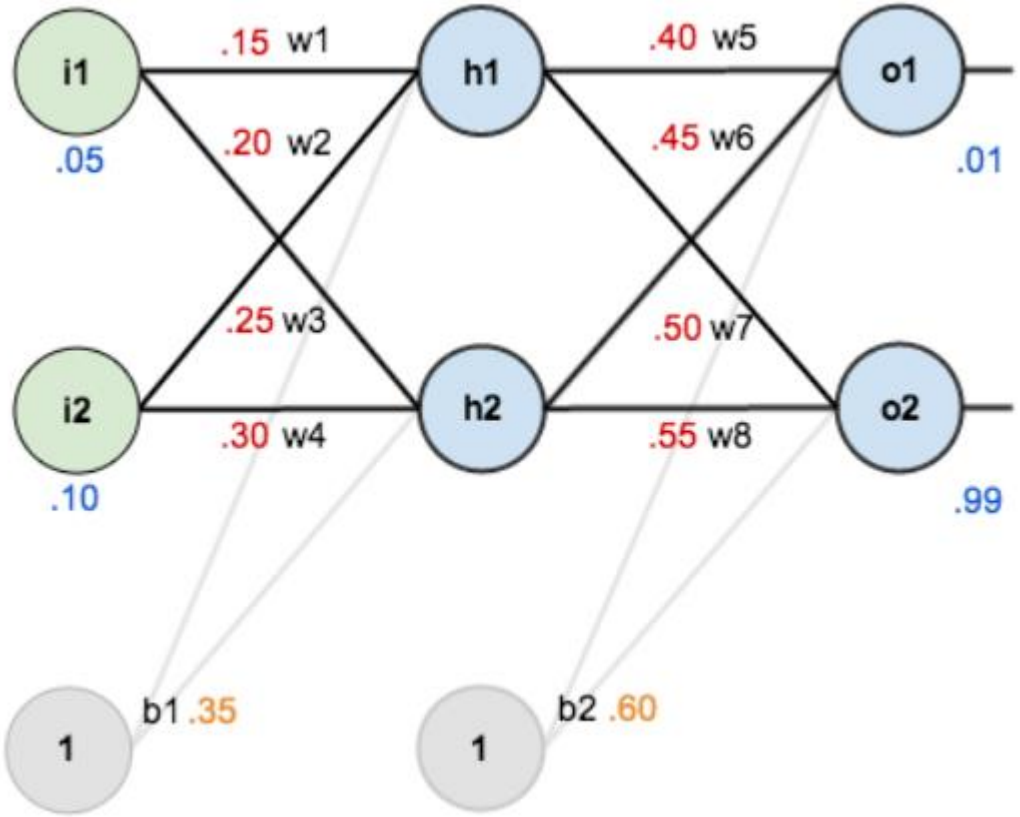
$$W_j \leftarrow W_j - \alpha \frac{\partial J(W_0, W_1)}{\partial W_j}$$

$$\frac{\partial J(W)}{\partial W_2} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial W_2}$$

$$\frac{\partial J(W)}{\partial W_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial W_1}$$



Continuing with the example



Source

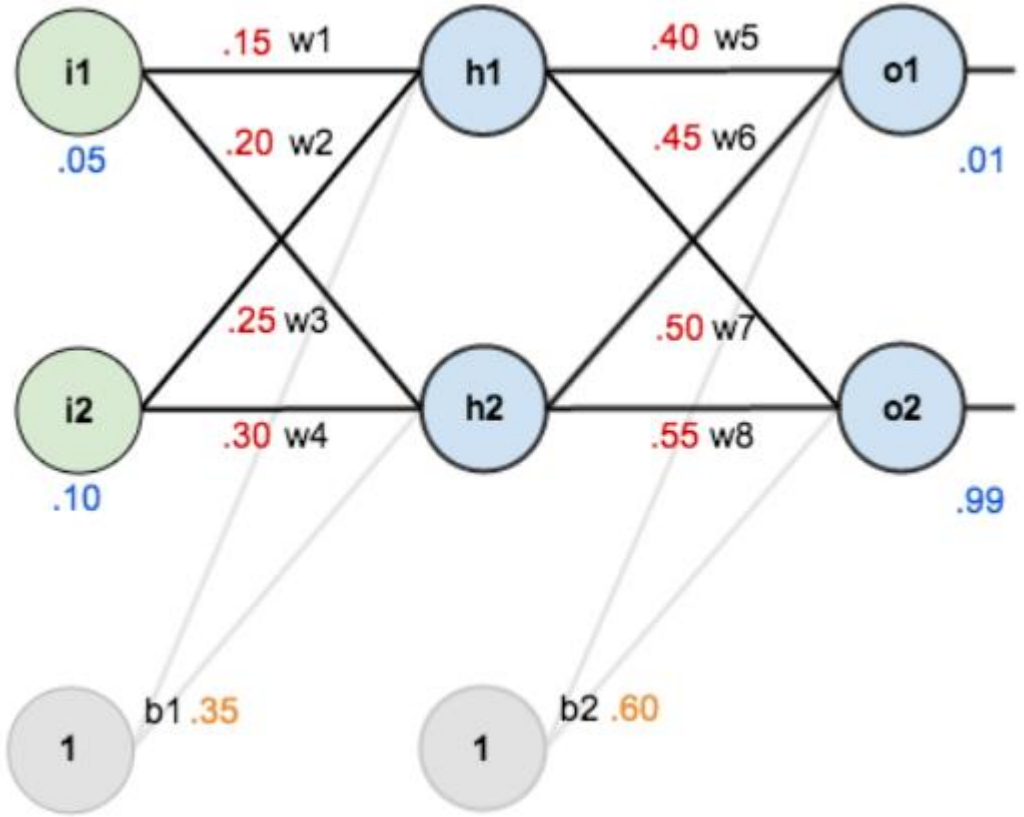
Use of sigmoid function as an activation function

Backward Pass

$$J(out_{o1}) = 0.274811083$$

Similarly, $J(out_{02})$ = 0.023560026

$J(W)_{total} = J(out_{01}) + J(out_{o2}) = 0.298371109$



Source

Use of sigmoid function as an activation function

Backward Pass

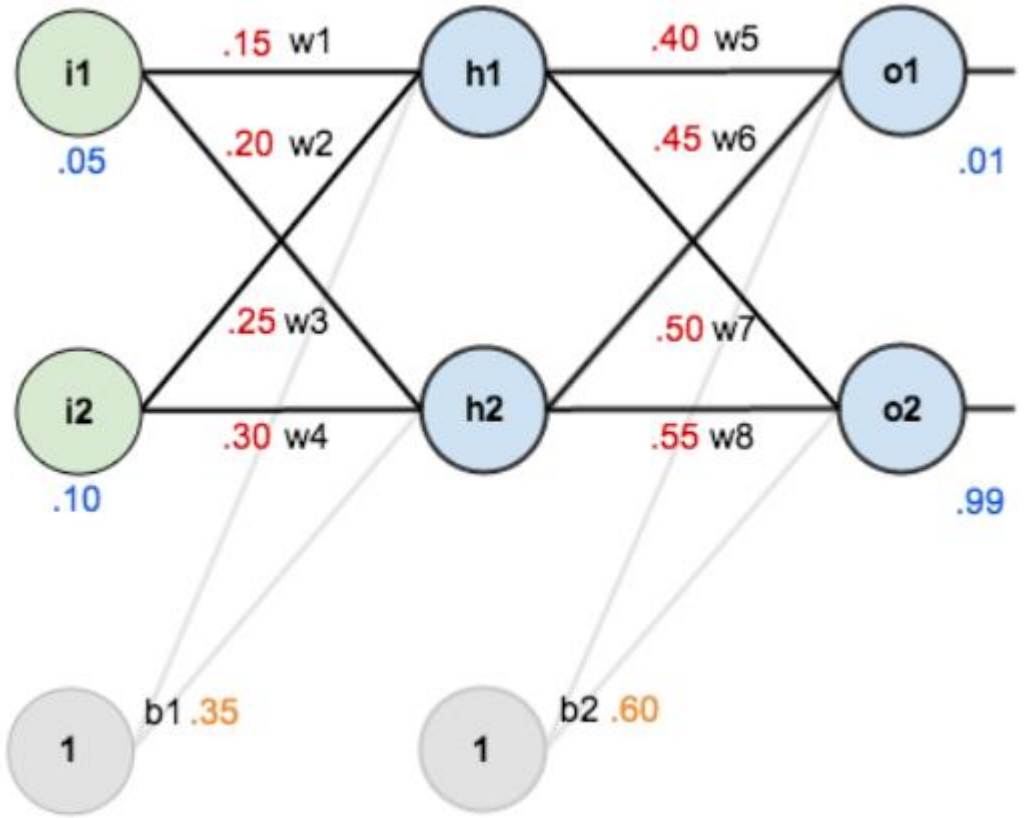
$$J(out_{o1}) = 0.274811083$$

Similarly, $J(out_{02}) = 0.023560026$

$J(W)_{total} = J(out_{01}) + J(out_{o2}) = 0.298371109$

$$\frac{\partial J(W)_{total}}{\partial W_5} = \frac{\partial J(W)_{total}}{\partial (out_{o1})} * \frac{\partial (out_{o1})}{\partial (net_{o1})} * \frac{\partial (net_{o1})}{\partial (W_5)}$$

Backward Pass



$$J(out_{o1}) = 0.274811083$$

Similarly, $J(out_{02}) = 0.023560026$

$J(W)_{total} = J(out_{01}) + J(out_{o2}) = 0.298371109$

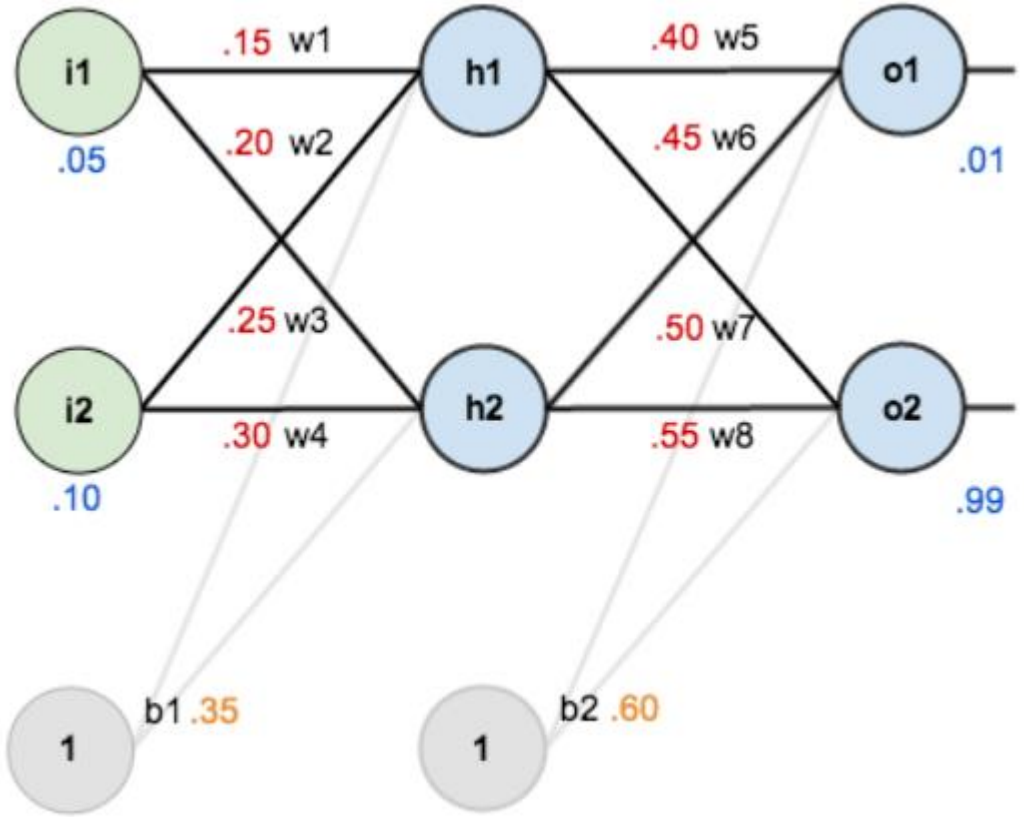
$$\frac{\partial J(W)_{total}}{\partial W_5} = \frac{\partial J(W)_{total}}{\partial (out_{o1})} * \frac{\partial (out_{o1})}{\partial (net_{o1})} * \frac{\partial (net_{o1})}{\partial (W_5)}$$

$$J(W)_{total} = \frac{1}{2}(actual_{o1} - out_{o1})^2 + \frac{1}{2}(actual_{o2} - out_{o2})^2$$

$$\frac{\partial J(W)_{total}}{\partial (out_{o1})} = 2 * \frac{1}{2} * (actual_{o1} - out_{o1}) * -1 + 0 = 0.74136507$$

[Source]

Use of sigmoid function as an activation function



Backward Pass

$$J(out_{o1}) = 0.274811083$$

Similarly, $J(out_{02}) = 0.023560026$

$J(W)_{total} = J(out_{01}) + J(out_{o2}) = 0.298371109$

$$\frac{\partial J(W)_{total}}{\partial W_5} = \frac{\partial J(W)_{total}}{\partial (out_{o1})} * \frac{\partial (out_{o1})}{\partial (net_{o1})} * \frac{\partial (net_{o1})}{\partial (W_5)}$$

$$J(W)_{total} = \frac{1}{2}(actual_{o1} - out_{o1})^2 + \frac{1}{2}(actual_{o2} - out_{o2})^2$$

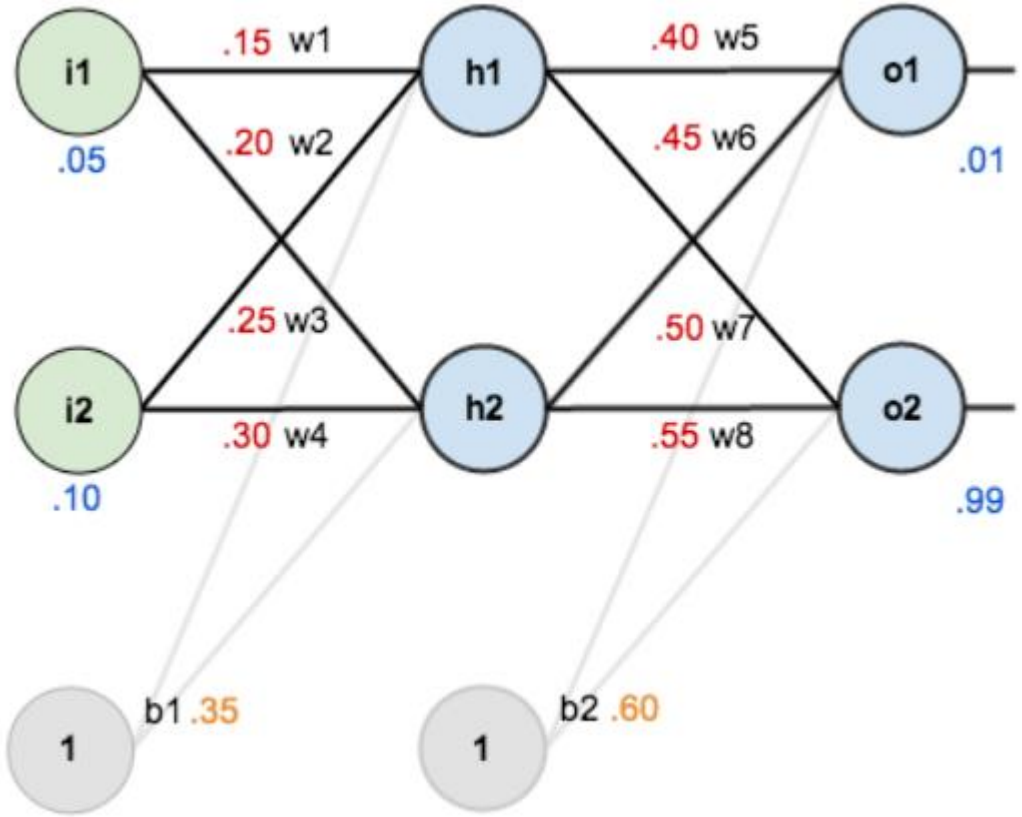
$$\frac{\partial J(W)_{total}}{\partial (out_{o1})} = 2 * \frac{1}{2} * (actual_{o1} - out_{o1}) * -1 + 0 = 0.74136507$$

$out_{o1} = sig(net_{01}) = \frac{1}{1+e^{-net_{o1}}}$

[Source]

Use of sigmoid function as an activation function

$$\frac{\partial (out_{o1})}{\partial (net_{o1})} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$



[Source]

Use of sigmoid function as an activation function

Backward Pass

$$J(out_{o1}) = 0.274811083$$

Similarly, $J(out_{02}) = 0.023560026$

$J(W)_{total} = J(out_{01}) + J(out_{o2}) = 0.298371109$

$$\frac{\partial J(W)_{total}}{\partial W_5} = \frac{\partial J(W)_{total}}{\partial (out_{o1})} * \frac{\partial (out_{o1})}{\partial (net_{o1})} * \frac{\partial (net_{o1})}{\partial (W_5)}$$

$$J(W)_{total} = \frac{1}{2}(actual_{o1} - out_{o1})^2 + \frac{1}{2}(actual_{o2} - out_{o2})^2$$

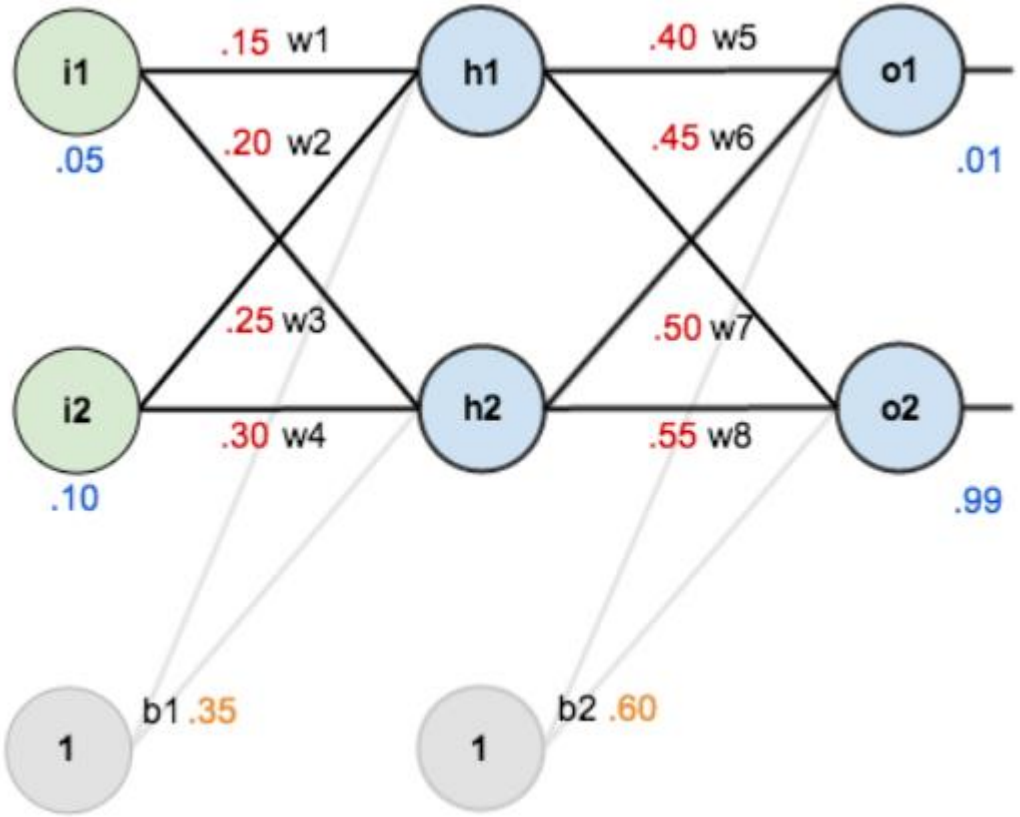
$$\frac{\partial J(W)_{total}}{\partial (out_{o1})} = 2 * \frac{1}{2} * (actual_{o1} - out_{o1}) * -1 + 0 = 0.74136507$$

$out_{o1} = sig(net_{01}) = \frac{1}{1+e^{-net_{o1}}}$

$$\frac{\partial (out_{o1})}{\partial (net_{o1})} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$net_{o1} = w5*out_{h1} + w6*out_{h2} + 1*b2$

$$\frac{\partial (net_{o1})}{\partial W_5} = 1 * (out_{h1}) + 0 + 0 = 0.593269992$$



[Source]

Use of sigmoid function as an activation function

Backward Pass

$$J(out_{o1}) = 0.274811083$$

Similarly, $J(out_{02}) = 0.023560026$

$J(W)_{total} = J(out_{01}) + J(out_{o2}) = 0.298371109$

$$\frac{\partial J(W)_{total}}{\partial W_5} = \frac{\partial J(W)_{total}}{\partial (out_{o1})} * \frac{\partial (out_{o1})}{\partial (net_{o1})} * \frac{\partial (net_{o1})}{\partial (W_5)}$$

$$J(W)_{total} = \frac{1}{2}(actual_{o1} - out_{o1})^2 + \frac{1}{2}(actual_{o2} - out_{o2})^2$$

$$\frac{\partial J(W)_{total}}{\partial (out_{o1})} = 2 * \frac{1}{2} * (actual_{o1} - out_{o1}) * -1 + 0 = 0.74136507$$

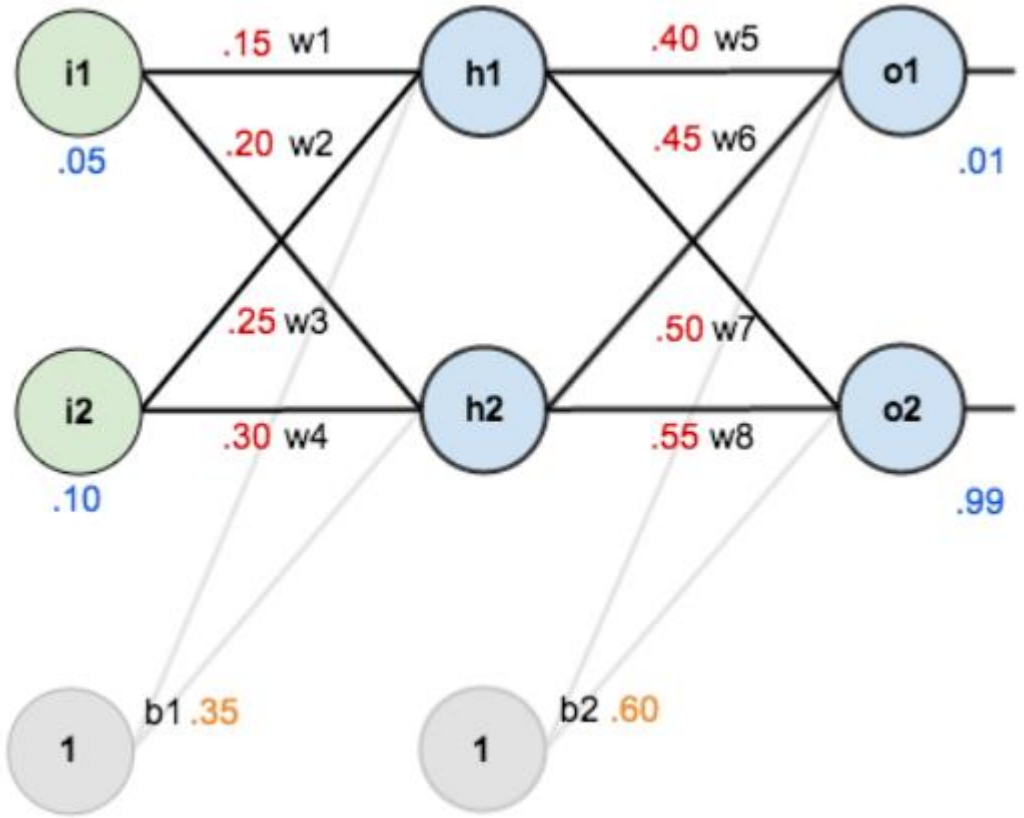
$out_{o1} = sig(net_{01}) = \frac{1}{1+e^{-net_{o1}}}$

$out_{o1} = sig(net_{01}) = \frac{1}{1+e^{-net_{o1}}}$

$$\frac{\partial (out_{o1})}{\partial (net_{o1})} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$net_{o1} = w5*out_{h1} + w6*out_{h2} + 1*b2$

$$\frac{\partial (net_{o1})}{\partial W_5} = 1 * (out_{h1}) + 0 + 0 = 0.593269992$$



Source

Use of sigmoid function as an activation function

Backward Pass

$$J(out_{o1}) = 0.274811083$$

Similarly, $J(out_{02}) = 0.023560026$

$J(W)_{total} = J(out_{01}) + J(out_{o2}) = 0.298371109$

$$\frac{\partial J(W)_{total}}{\partial W_5} = \frac{\partial J(W)_{total}}{\partial (out_{o1})} * \frac{\partial (out_{o1})}{\partial (net_{o1})} * \frac{\partial (net_{o1})}{\partial (W_5)}$$

$$J(W)_{total} = \frac{1}{2}(actual_{o1} - out_{o1})^2 + \frac{1}{2}(actual_{o2} - out_{o2})^2$$

$$\frac{\partial J(W)_{total}}{\partial (out_{o1})} = 2 * \frac{1}{2} * (actual_{o1} - out_{o1}) * -1 + 0 = 0.74136507$$

$out_{o1} = sig(net_{01}) = \frac{1}{1+e^{-net_{o1}}}$

$out_{o1} = sig(net_{01}) = \frac{1}{1+e^{-net_{o1}}}$

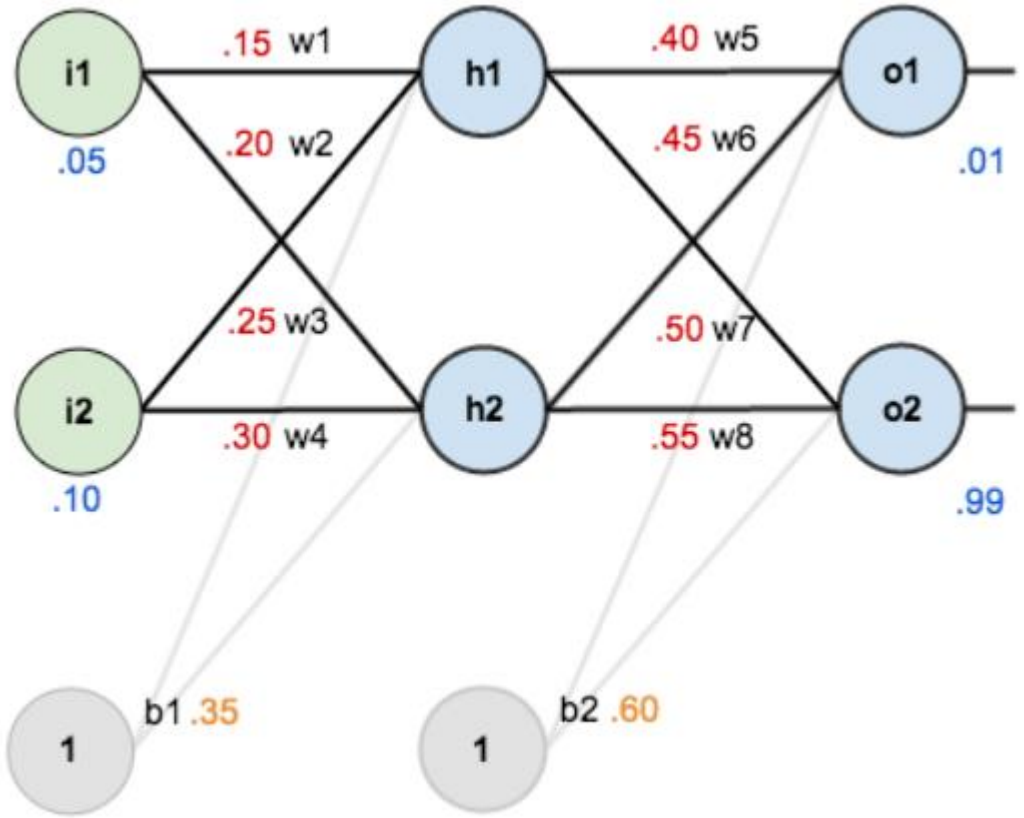
$$\frac{\partial (out_{o1})}{\partial (net_{o1})} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$net_{o1} = w5*out_{h1} + w6*out_{h2} + 1*b2$

$$\frac{\partial (net_{o1})}{\partial W_5} = 1 * (out_{h1}) + 0 + 0 = 0.593269992$$

$$\frac{\partial J(W)_{total}}{\partial W_5} = 0.74136507 * 0.186815602 * 0.593269992$$

$$= 0.082167041$$



[Source]

Use of sigmoid function as an activation function

$$\frac{\partial J(W)_{total}}{\partial W_5} = 0.74136507 * 0.186815602 * 0.593269992$$

$$\frac{\partial J(W)_{total}}{\partial W_5} = 0.082167041$$

$$W_j \leftarrow W_j - \alpha \frac{\partial J(W_0, W_1)}{\partial W_j}$$

Lets assume the learning rate (α) is 0.5.

$$W^*_5 \leftarrow W_5 - 0.5 \frac{\partial J(W_{total})}{\partial W_5}$$

$$W^*_5 = 0.4 - 0.5 * 0.082167041$$

$$\boxed{W^*_5 = 0.35891648}$$

Similarly, the remaining weights can be updated.

Types of Gradient Descent Algorithms

Differ in the amount of data being used

Batch GD

- The entire training set is used.
- The parameters will be updated only once per epoch, i.e., take the entire training set, perform forward propagation, calculate loss function, then update the weights to minimize loss.
- Computationally expensive.

➔ Loss function reduces smoothly.

Types of Gradient Descent Algorithms

Differ in the amount of data being used

Batch GD

- The entire training set is used.
- The parameters will be updated only once per epoch, i.e., take the entire training set, perform forward propagation, calculate loss function, then update the weights to minimize loss.
- Computationally expensive.

➔ Loss function reduces smoothly.

Stochastic GD

- Loss function is calculated for every single observation in the training set and parameters are updated.

$$\frac{\partial J_i(W)}{\partial W}$$

- Parameters will be updated after every iteration of a single observation in the dataset.
- Gradient can be too noisy.

➔ Lots of variations in loss function.

Types of Gradient Descent Algorithms

Differ in the amount of data being used

Batch GD

- The entire training set is used.
- The parameters will be updated only once per epoch, i.e., take the entire training set, perform forward propagation, calculate loss function, then update the weights to minimize loss.
- Computationally expensive.

➔ Loss function reduces smoothly.

Stochastic GD

- Loss function is calculated for every single observation in the training set and parameters are updated.

$$\frac{\partial J_i(W)}{\partial W}$$

- Parameters will be updated after every iteration of a single observation in the dataset.
- Gradient can be too noisy.

➔ Lots of variations in loss function.

Mini-batch GD

- Parameters are updated after a subset of data (or a mini-batch B) has been processed through the model.

$$\frac{\partial J(W)}{\partial W} = \frac{1}{B}\sum_{i=1}^{B}\frac{\partial J_i(W)}{\partial W}$$

- Computation time lesser than SGD.
- Computation cost lesser than Batch GD.

➔ Smoother loss function as compared to SGD.

Terminology used in the previous slide…

- *Batch = No. of training examples in a single 'chunk' of data*
- *Iterations = No. of batches needed to complete one epoch*
- *Epoch = one epoch is when an entire dataset is passed forward and backwards through the network once.*

For eg: Training dataset of 50,000 examples. Divided into 500 chunks.

- *Batch size = 500*
- *Iterations = no. of batches = 100*
- *1 Epoch = 100 iterations*

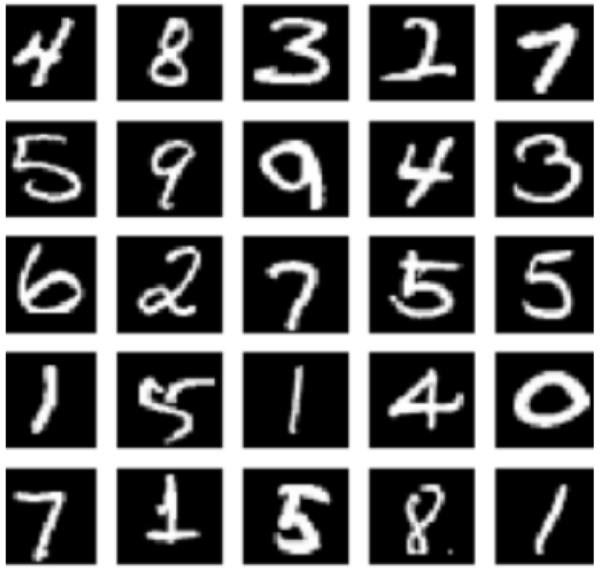
Bringing it all together: Multi-layer Perceptron (Forward and Backward propagation)

The MNIST Use-Case: Handwritten Digit Recognition

Case Study: Handwritten Digit Recognition

- Use of the MNIST (Modified National Institute of Standards and Technology) dataset, Lecun, Y. 1998.
  - ✓ Consists of 70k images of digits from 0 to 9.
  - ✓ Training set: 60k images, test set: 10k images.
  - ✓ Each image is represented as a 28x28 pixel matrix where each cell represents a grayscale pixel value.

- Goal: Identify numbers based on handwritten digits.
  - ✓ Multi-class classification problem (0 -9 digits ➔ 10 classes)



Subset of Images From the MNIST Dataset

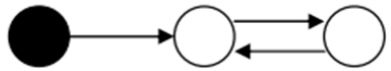
https://github.com/aayushmnit/Deep_learning_explorations/blob/master/2_MLP_tensorflow/my1stNN.ipynb

Steps Involved

1. Import the required modules
2. Load the MNIST dataset and split into training and test sets
3. Flattening the data
   - Since each image is represented by a 2D matrix, it needs to be converted into a 1D format for feeding into the model i.e., the 28x28 image is converted into a 1D array of 784 (i.e., 28x28).
4. Building the Neural Network: Multi-layer Perceptron with two hidden layers.
   a. Initialise the network parameters, weights and biases, declare placeholders for input data
   b. Compute output of every neuron by multiplying weights with input, adding the bias and passing through an activation function.
   c. Addressing overfitting by use of Dropout.
   d. Defining the loss function.
   e. Define an optimizer to minimize the loss and adjusting the weights.
   f. Define model evaluation metrics.
   g. Train the network for n number of epochs.

Recurrent Neural Networks

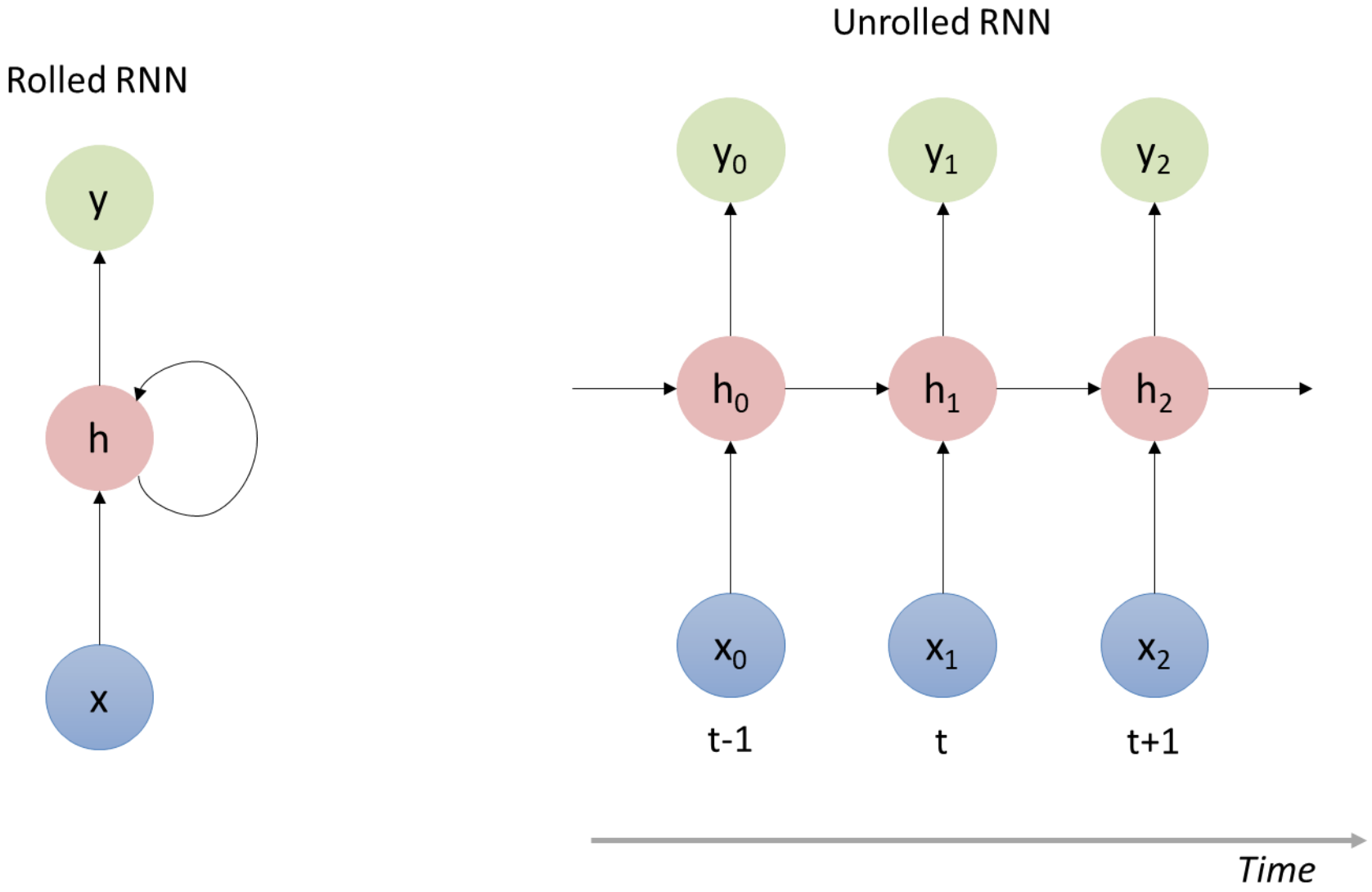
Recurrent Neural Networks



Recurrent NN
(RNN)

- Use *feedback loop* in the hidden layers.
- Make use of a “memory”.
- Most commonly used for time series data or sequential data.
- Example applications: *Voice search, machine translation, image captioning, speech recognition, etc.*
- E.g., For predicting next word/letter in a sequence, the model needs to remember the sequence of the words/letters that have appeared in the preceding timestamps.
- Variants of RNN’s:
  - Gated Recurrent Unit (GRU)
  - Long-Short Term Memory Units (LSTM)

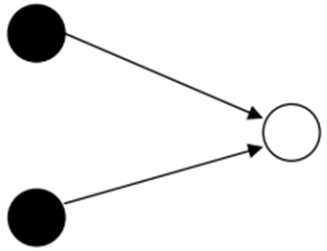
Recurrent Neural Networks



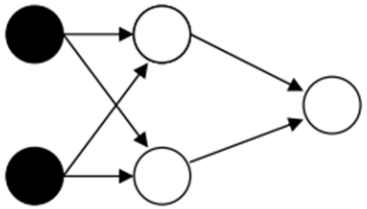
Conclusion

Conclusion

- Deep Learning
- Neural Network Architectures
  - Activation Functions and Bias
  - Loss Functions
  - Optimizers: Gradient Descent
  - Back Propagation
- TensorFlow Basics
- Implementation of ANN topologies using TensorFlow & Keras
- Play around with your own NN's here.



Single-Layer
Feed-Forward NN



Multi-Layer
Feed-Forward NN

Recurrent NN
(RNN)

References

1. A Brief History of Deep Learning. (2017). Retrieved from Dataversity: https://www.dataversity.net/brief-history-deep-learning/
2. Karpathy, A., & Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3128-3137).
3. LeCun, Y. (1998). The MNIST database of handwritten digits. http://yann. lecun. com/exdb/mnist/.
4. McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.
5. Minsky, M., & Papert, S. (1969). Perceptrons.
6. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
7. Rosenblatt, F. (1957). The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory.
8. Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.
9. Smith, L. N. (2017, March). Cyclical learning rates for training neural networks. In 2017 IEEE winter conference on applications of computer vision (WACV) (pp. 464-472). IEEE.
10. Zhang D., et al. “The AI Index 2022 Annual Report,” AI Index Steering Committee, Stanford Institute for Human-Centered AI, Stanford University, March 2022.

Reading Material

- Books:
  - ✓*Deep learning* by Ian Goodfellow, Yoshua Bengio, & Aaron Courville.
  - ✓*Neural Networks and Deep Learning* by Michael Nielsen.
  - ✓*Neural networks: a systematic introduction* by R.Rojas.
- Online Lecture Series:
  - ✓Andrew Ng's Machine Learning course on Coursera.
  - ✓Practical Deep Learning for Coders by Sylvain Gugger and Jeremy Howard (Fast.ai, also available on Youtube).
  - ✓Deep Learning Specialization by Ng et. al (DeepLearning.AI)

Thank You for your attention!