alitourani

Deep Learning from Scratch

Session #1: Introduction

by: Ali Tourani – Summer 2021

Agenda

- What is Deep Learning?
- Fundamental Concepts
- Neural Networks
- Training the Network
- Use cases of ANNs
- Assignment & Homework

What is Deep Learning?

How to enable computers to mimic advanced human capabilities?

Learning AI rules by analyzing a collection of known examples

A type of machine learning with need to be told about the important features

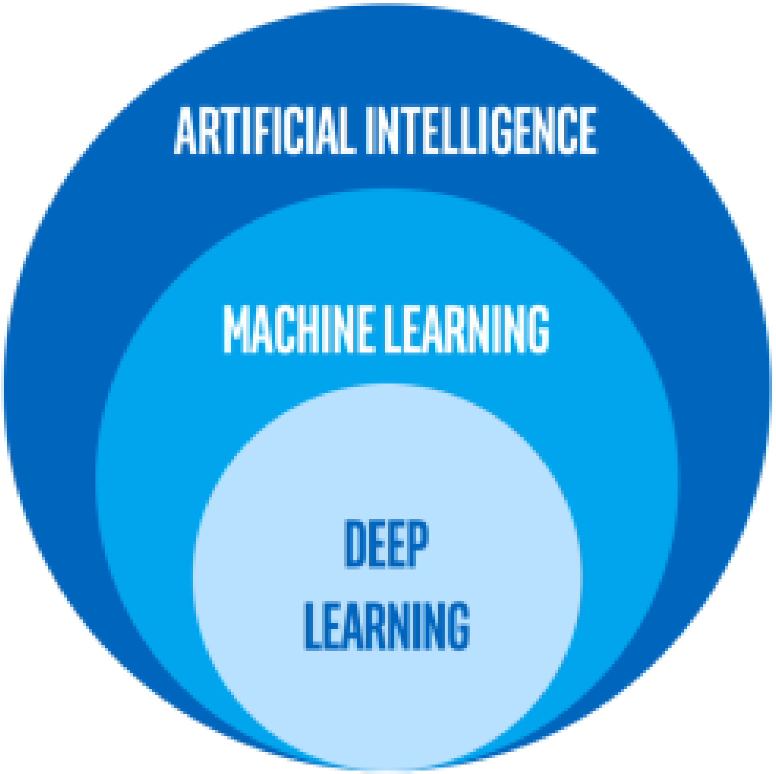


image downloaded from intel.la website

What is Deep Learning?

- Machine Learning (ML):
  - Using parameters from known and important features of data
  - Predict the outcomes on similar data
  - Final tool: a **Model**
- Deep Learning (DL):
  - Using Artificial Neural Networks
  - Learn tasks directly from law data
  - Needs better hardware and datasets

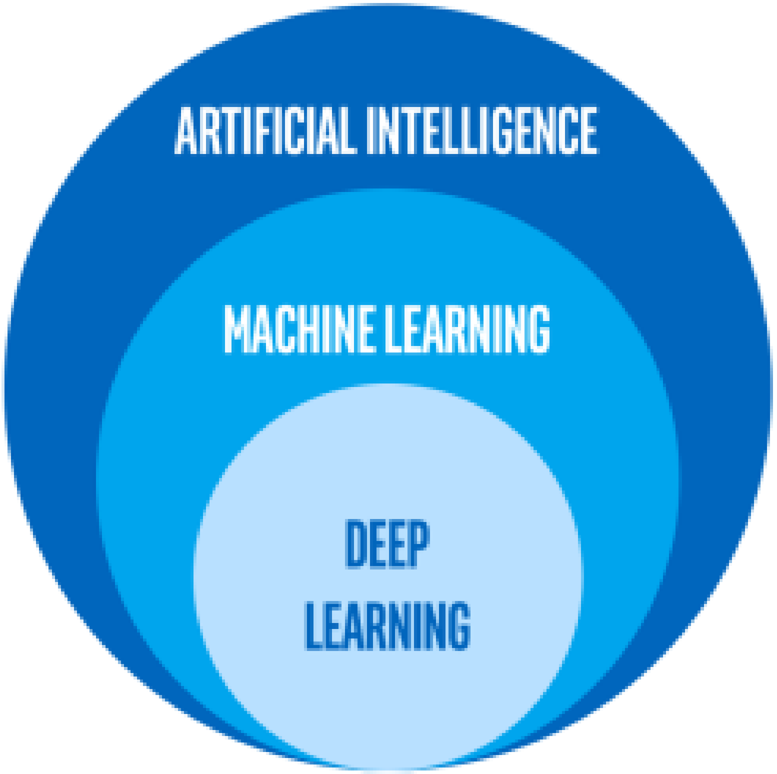
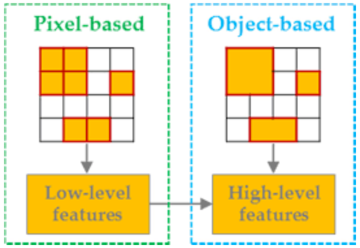


image downloaded from intel.la website

What is Deep Learning?

- How to learn data?
  - Using the features existing in data (like images, sounds, etc.)
  - Trying to find the rules and patterns
  - Decomposing existing features for this purpose



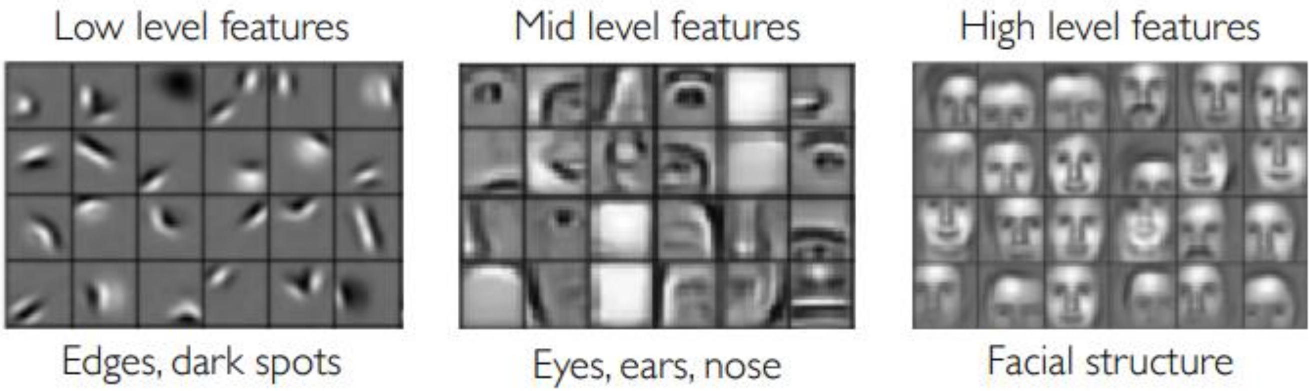


image downloaded from medium.com

What is Deep Learning?

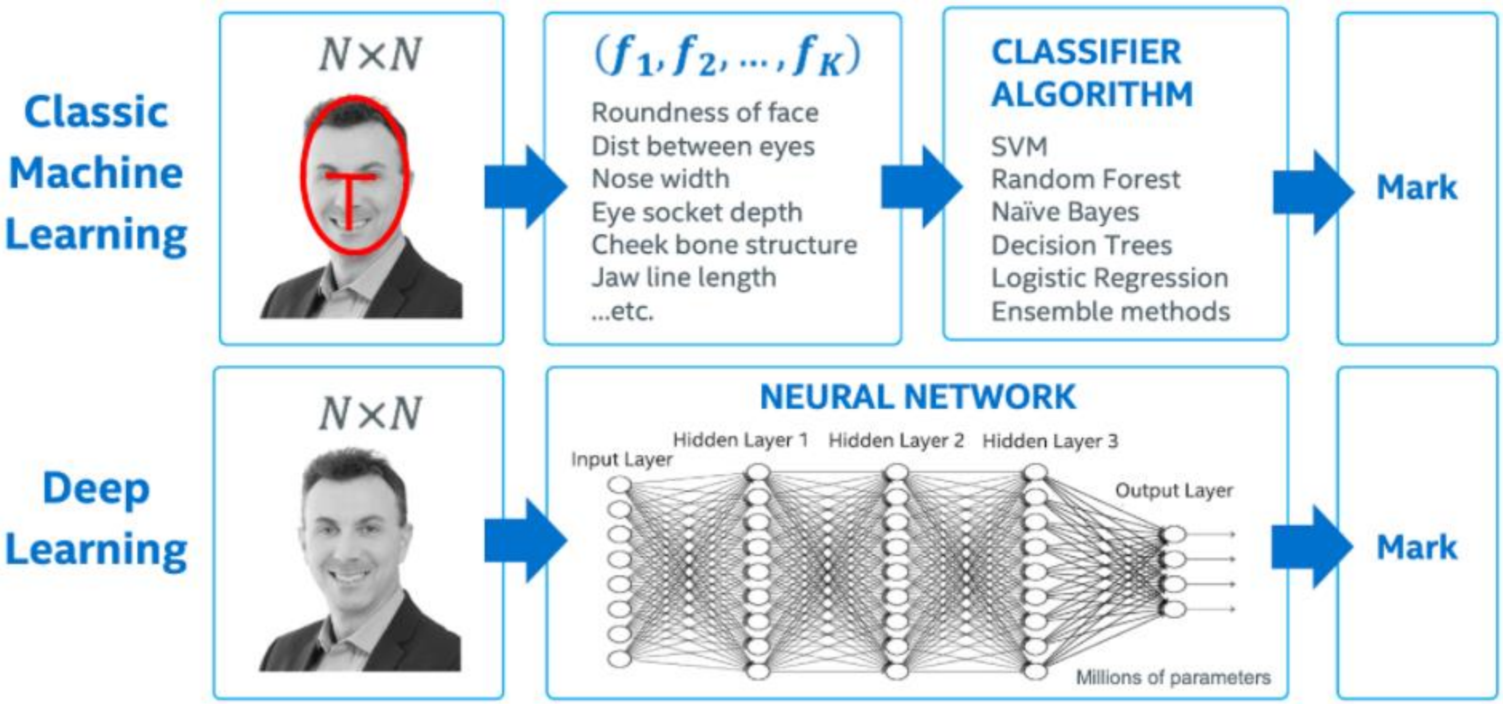
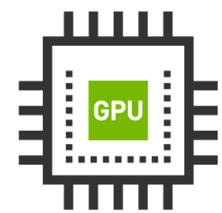


image downloaded from intel.la website

What is Deep Learning?

- What are the requirements to employ Deep Learning?
  - **Big Data**
    - To extract the features, we need to have all variations of data
    - Large datasets with thousands and millions of samples

  - **Powerful Hardware**
    - Highly parallelizable hardware are needed to process big data
    - Graphics Processing Units (GPUs) are the best options



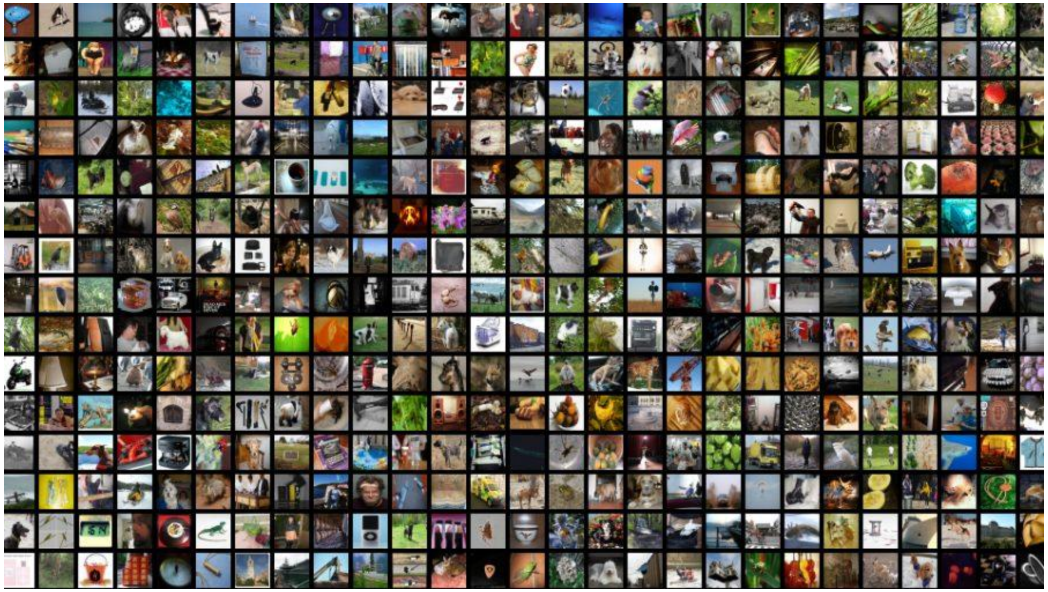
  - **Efficient Software**
    - Different toolboxes and platforms that enable us to process big data using GPUs

image downloaded from medium.com

What is Deep Learning?

- What are the requirements to employ Deep Learning?

Big Data + Powerful Hardware + Efficient Software



What is Deep Learning?

- What are the requirements to employ Deep Learning?

Big Data + Powerful Hardware + Efficient Software

What is Deep Learning?

- What are the requirements to employ Deep Learning?

Big Data + Powerful Hardware + Efficient Software

TensorFlow

PYTORCH

Keras

What is Deep Learning?

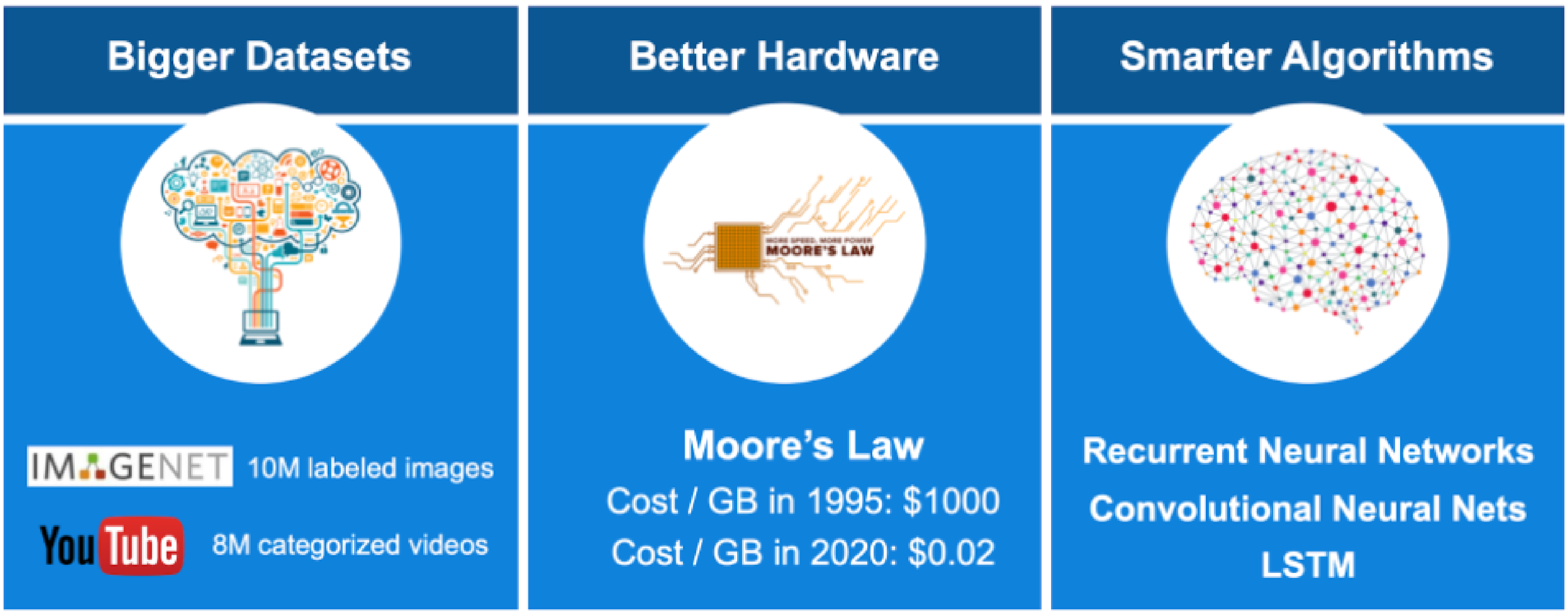
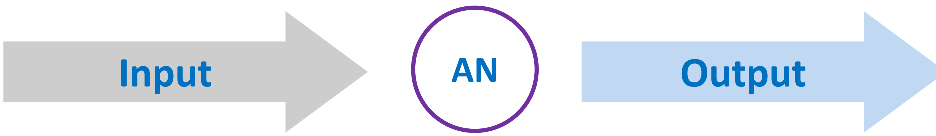


image downloaded from intel.la website

Fundamental Concepts

Artificial Neural Networks (ANNs, or simply, NNs)

- Inspired from neurons in human brain
- Contains at least one Artificial Neuron (AN)
  - Receives an input signal, process it, and generates and output signal



- Now, lets connect several Ans:
  - The output of neuron ***A*** will be the input of neuron ***B***
  - They will shape an ANN that are organized into different layers

Fundamental Concepts

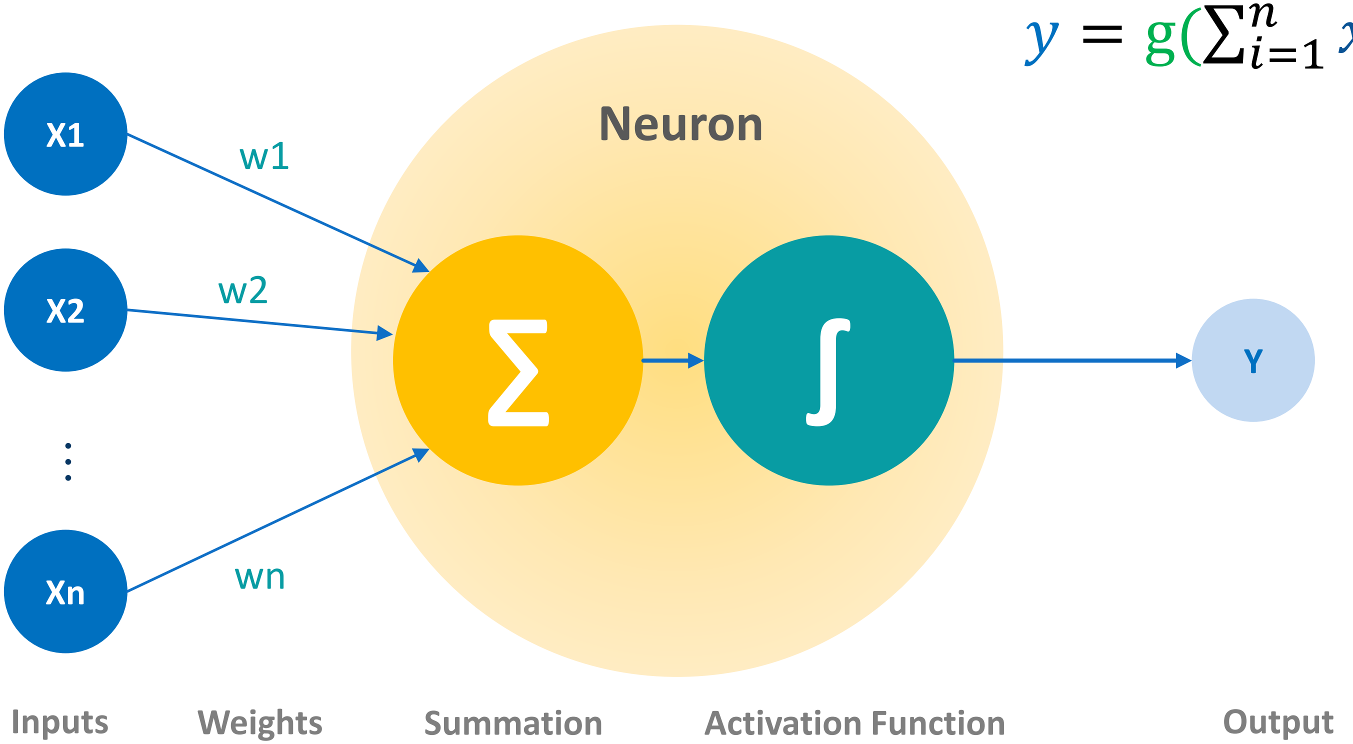
The Perceptron

- The simplest architecture of an ANN
  - At least **one input signal *X*** with an corresponding **weight *W***
    - The weight shows the importance/priority of the input signal
  - Calculating the summation of each input signal multiplied by its weight
    - The output will be a single number
  - Passing the sum value to a non-linear function, called Activation Function (AF)
  - Now, the **final output signal *Y*** is ready!

Fundamental Concepts

The Perceptron

$$y = g(\sum_{i=1}^{n} x_i w_i)$$



Fundamental Concepts

The Perceptron

- But how to make the AF independent from the input signals
  - **Solution:** adding a Bias node

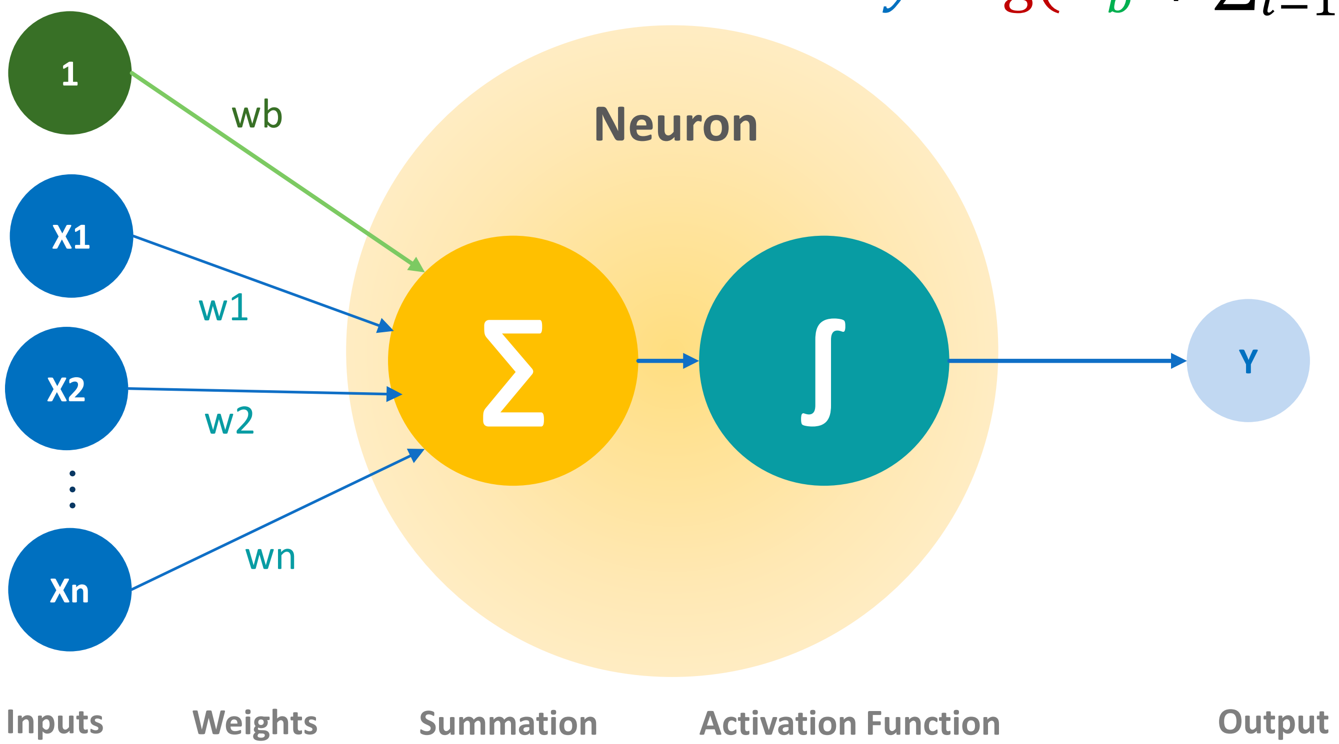
What is a Bias?

  - A weighted input signal with value of **1**
  - Using this, and by providing proper weight, we can shift our AFs to left and right

Fundamental Concepts

The Perceptron

$$y = g(w_b + \sum_{i=1}^{n} x_i w_i)$$



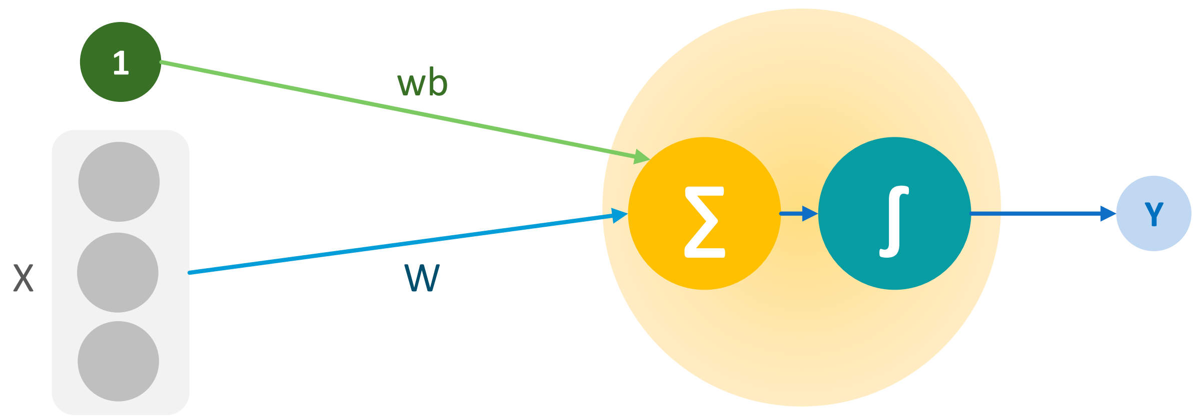
Fundamental Concepts

The Perceptron

- Now, we can mathematically define the output as

$$y = g(w_b + X^T W)$$

Where ***X*** and ***W*** are the vector of inputs and weights, respectively



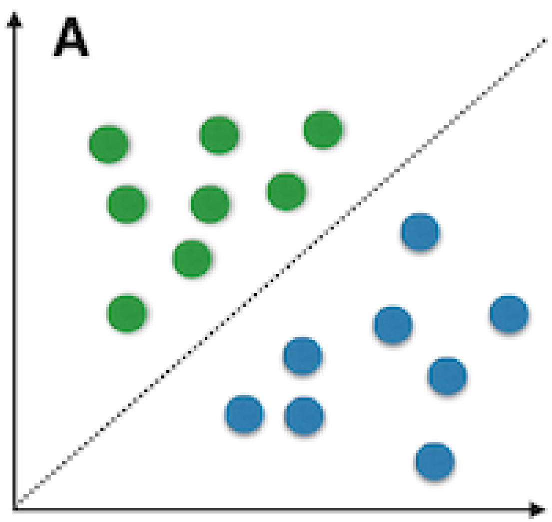
Fundamental Concepts

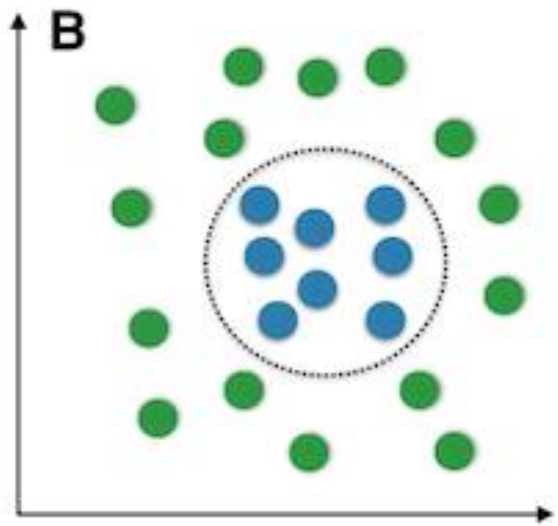
Activation Functions

$$y = g(w_b + X^T W)$$

- Now, why do we need AFs?
  - To provide **non linearity** and … **But Why?**

Linear data





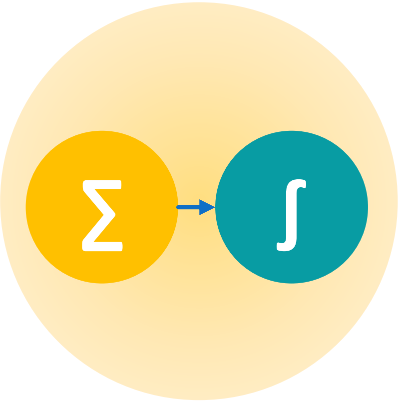
Non-linear data

Fundamental Concepts

Activation Functions

$$y = g(w_b + X^T W)$$

- Now, why do we need AFs?
  - To provide **non linearity** and use the ANNs in real-world scenarios
  - To define how the weighted sum is transformed into an output
  - To get access to much richer hypothesis space (especially, deep ones)
  - We need to choose them carefully in our ANNs!
    - There are many of them!
  - Technically, the same for all nodes in a layer of an ANN



Fundamental Concepts

Activation Functions

- Some of the popular AFs
  - **Linear**
  - **Unit step**
  - **Sign (signum)**
  - **ReLU (Rectified Linear Activation)**
  - **Hyperbolic Tangent (Tanh)**
  - **Sigmoid/Logistic (maps real numbers to [0, 1])**
  - **Gaussian**

$$f(x) = x$$

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

$$f(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases}$$

$$f(x) = \max(0, z)$$

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

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

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\delta^2}}$$

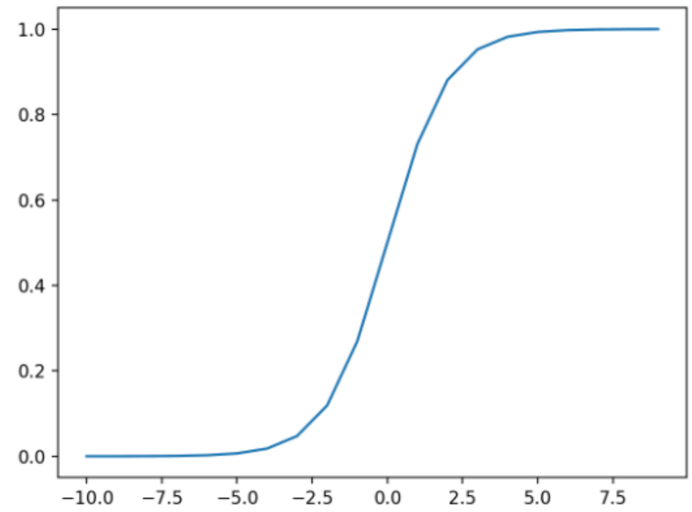
Fundamental Concepts

Activation Functions - Sigmoid

- Takes a real number as input
- Generates a number in the range 0 to 1
  - Larger input ➔ closer to 1
- A common use case: **probabilities**

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

```
# Applying the sigmoid function and
# storing the result in 'b'
b = tf.nn.sigmoid(a, name ='sigmoid')
```



Fundamental Concepts

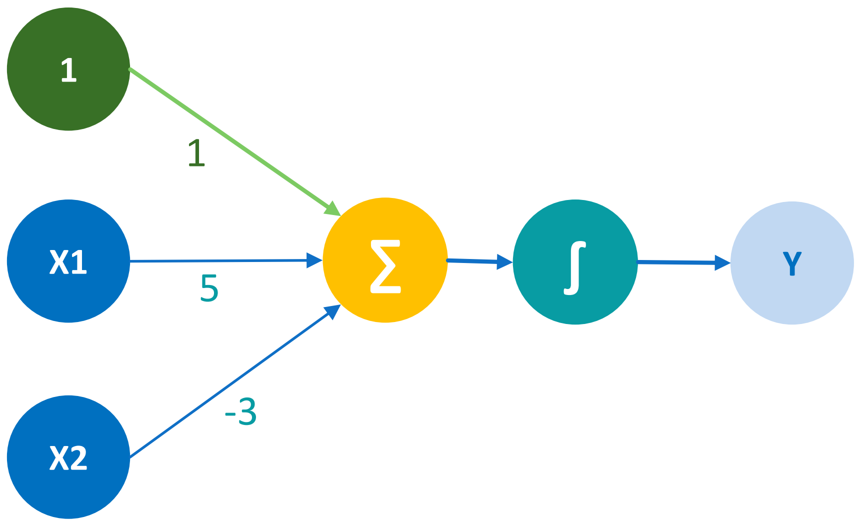
Let's see an example of Perceptron with Sigmoid AF

- Suppose we have this AN:
  - The vector of weights is:

$$W_0 = 1, \qquad W^T = [5 \quad -3]$$

  - The summation function provides:

$$y = g(w_b + X^T W)$$
$$= g(1 + 5x_1 - 3x_2)$$



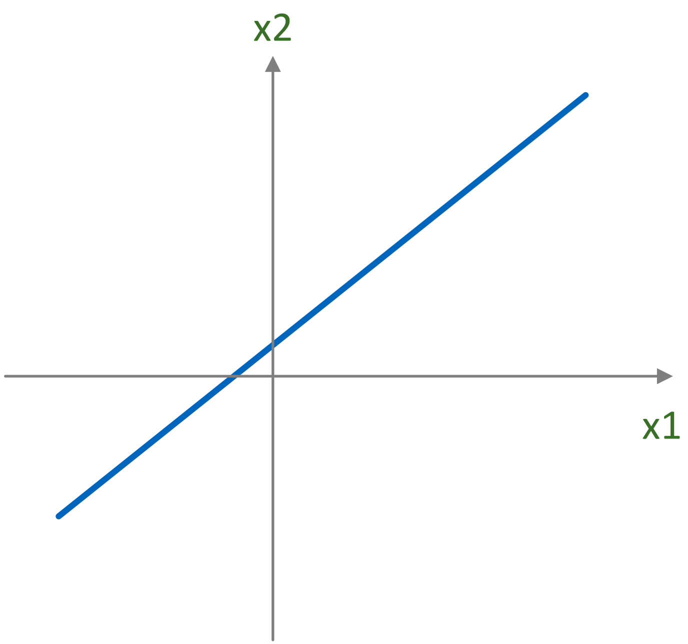
Fundamental Concepts

Let's see an example of Perceptron with Sigmoid AF

- Thus, we will have:

$$y = g(1 + 5x_1 - 3x_2)$$

- Some points:
  - $x = \begin{bmatrix} -2 \\ 2 \end{bmatrix} \Rightarrow y = g(-15) = 0.03$
  - $x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \Rightarrow y = g(0) = 0.5$
  - $x = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \Rightarrow y = g(5) = 0.99$

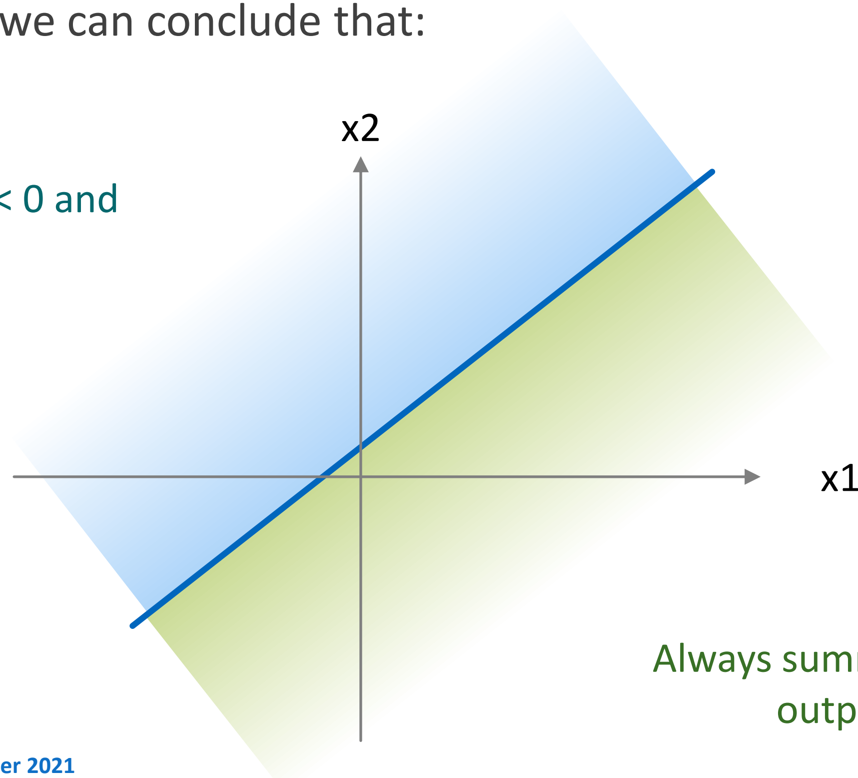


Fundamental Concepts

Let’s see an example of Perceptron with Sigmoid AF

- Thus, generally we can conclude that:

Always summation $z < 0$ and output $y < 0.5$



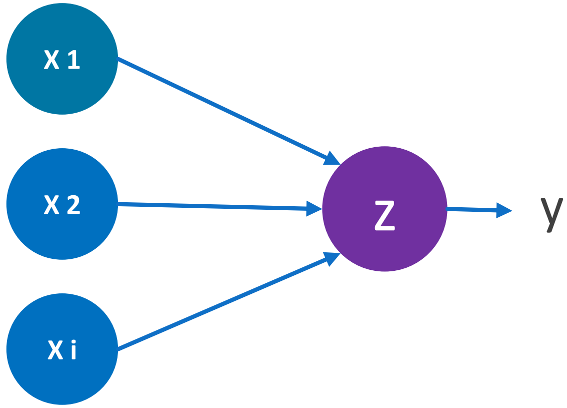
Always summation $z > 0$ and output $y > 0.5$

Neural Networks

- Now, let's simplify the diagram:

$$z = w_b + \sum_{i=1}^{m} x_i w_i$$

$$y = g(z)$$



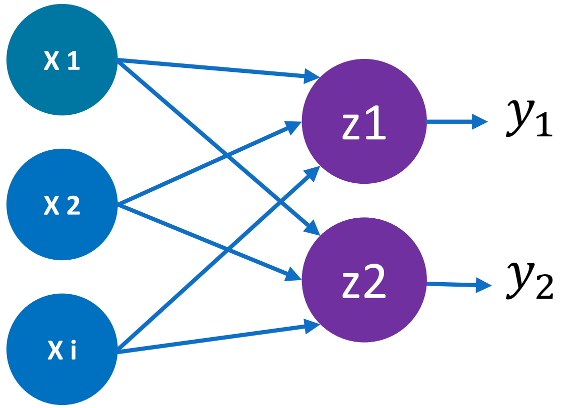
- What if our Perceptron provides multiple outputs?
  - Or, can we connect two Perceptron networks to provide two different outputs?

Neural Networks

- We can extend it to Multi Output Perceptron (MOP):

$$z_j = w_{b,j} + \sum_{i=1}^{m} x_i w_{i,j}$$

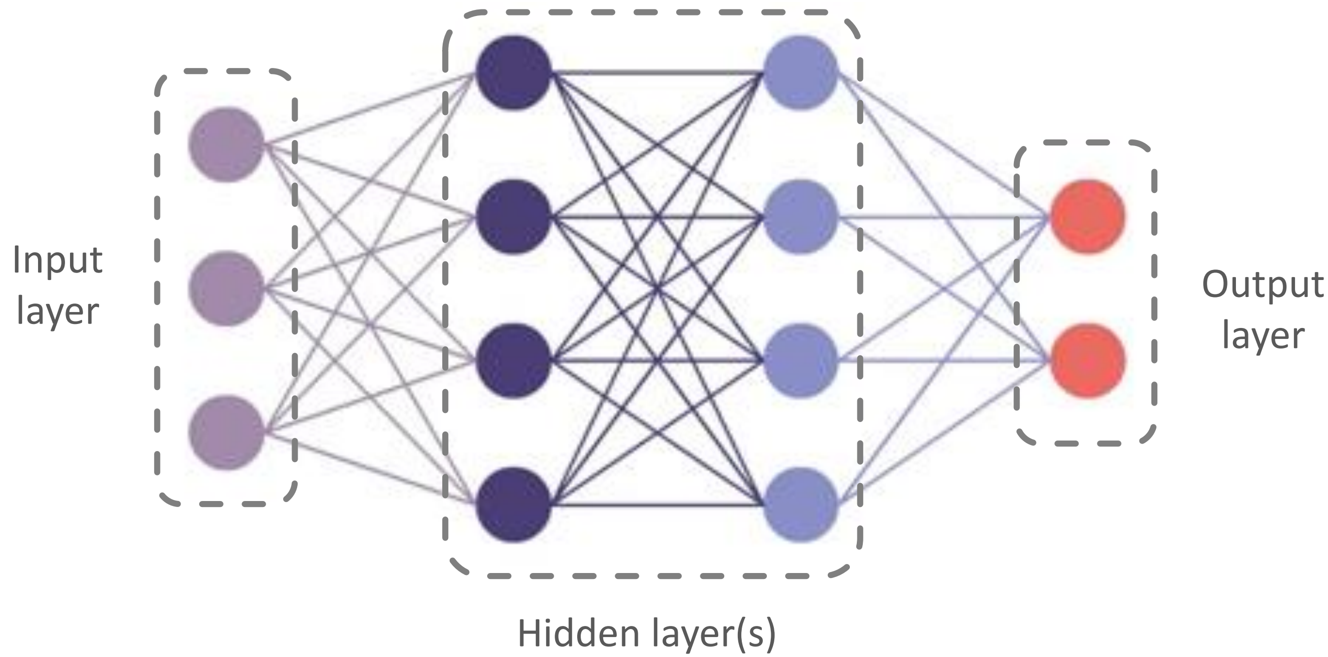
$$y_i = g(z_i)$$



- **Dense Layers**
  - All inputs are connected to all outputs

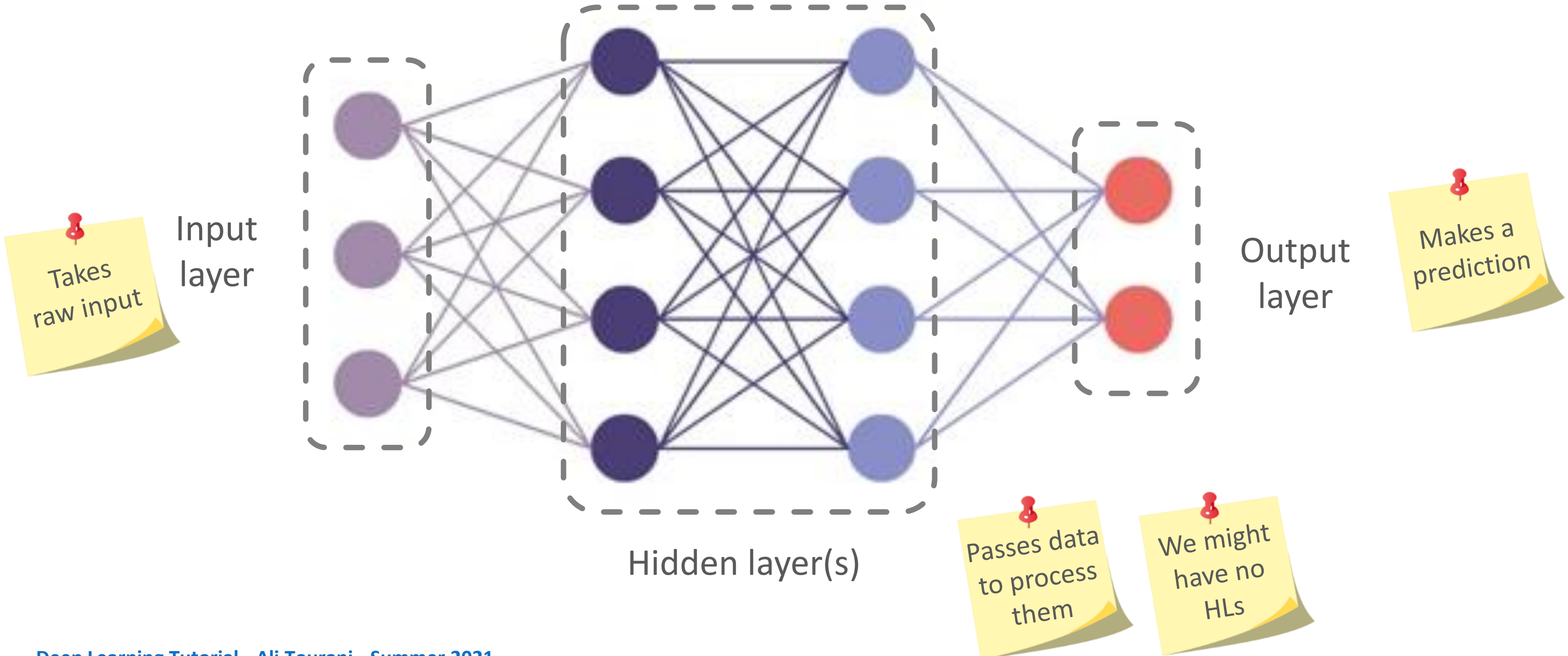
Neural Networks

- Let's connect several single nodes (Perceptron) together:



Neural Networks

- Let's connect several single nodes (Perceptron) together:

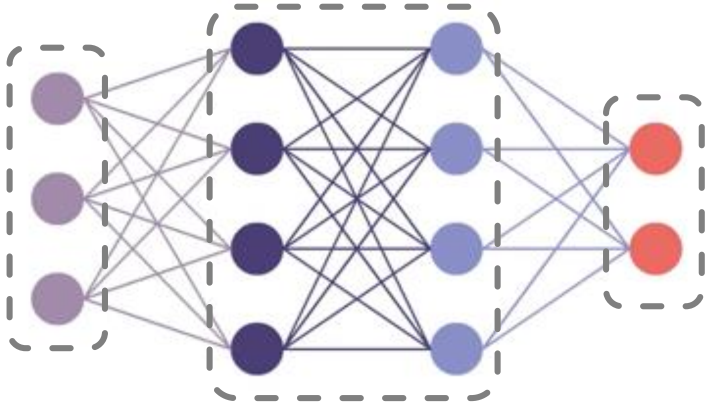


Neural Networks

Some Useful Notes

- If there is only one layer in HL,

we call it a **Single Layer NN**



- If the number of layers in HL is more

than five (usually), we call it a **Deep NN**

- We can always find sequential dense

layers in NNs, like the image above with 3x4, 4x4, and 4x2 dense layers

Neural Networks

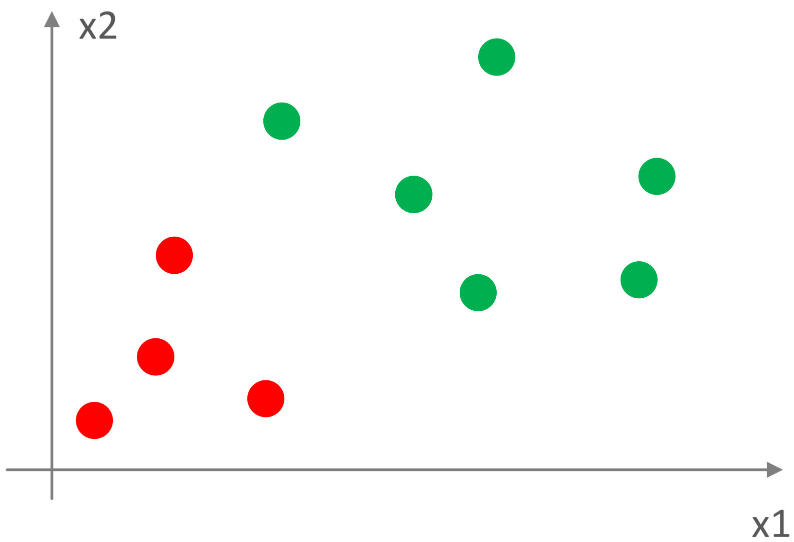
Sample#1: passing a driving test

- Assume we need to pass both theory and practical tests

$$x_1 = \{topics\ achieved\ in\ theory\ classes\}$$

$$x_2 = \{hours\ spent\ on\ practical\ tests\}$$

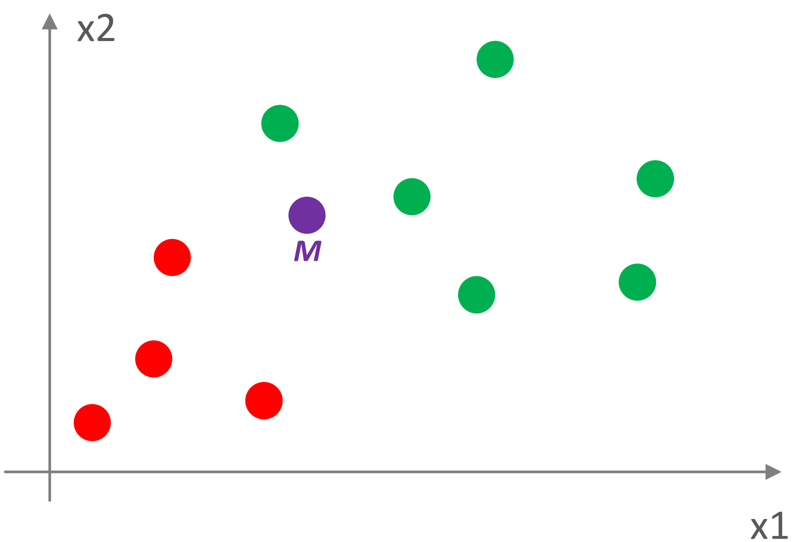
- So, we will have:
  - Distribution of people who have passed or failed the test
  - The chart is 2D, as we have two features



Neural Networks

Sample#1: passing a driving test

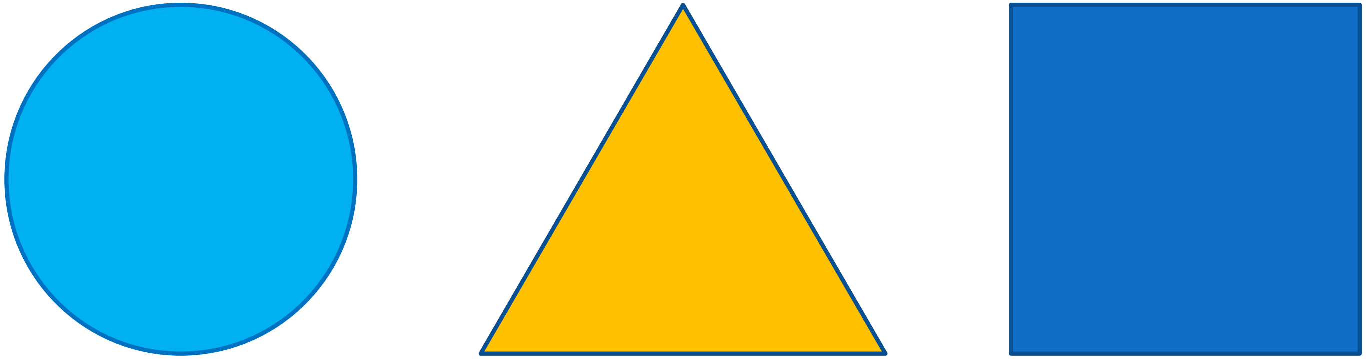
- Now, the question is will a given node *M* pass the test?
- Assume M=[10, 12], meaning that
  - 10: number of achieved topics
  - 12: hours spent training
- Answer: we need to **train** the ANN first
  - Training = adjusting the weights



Neural Networks

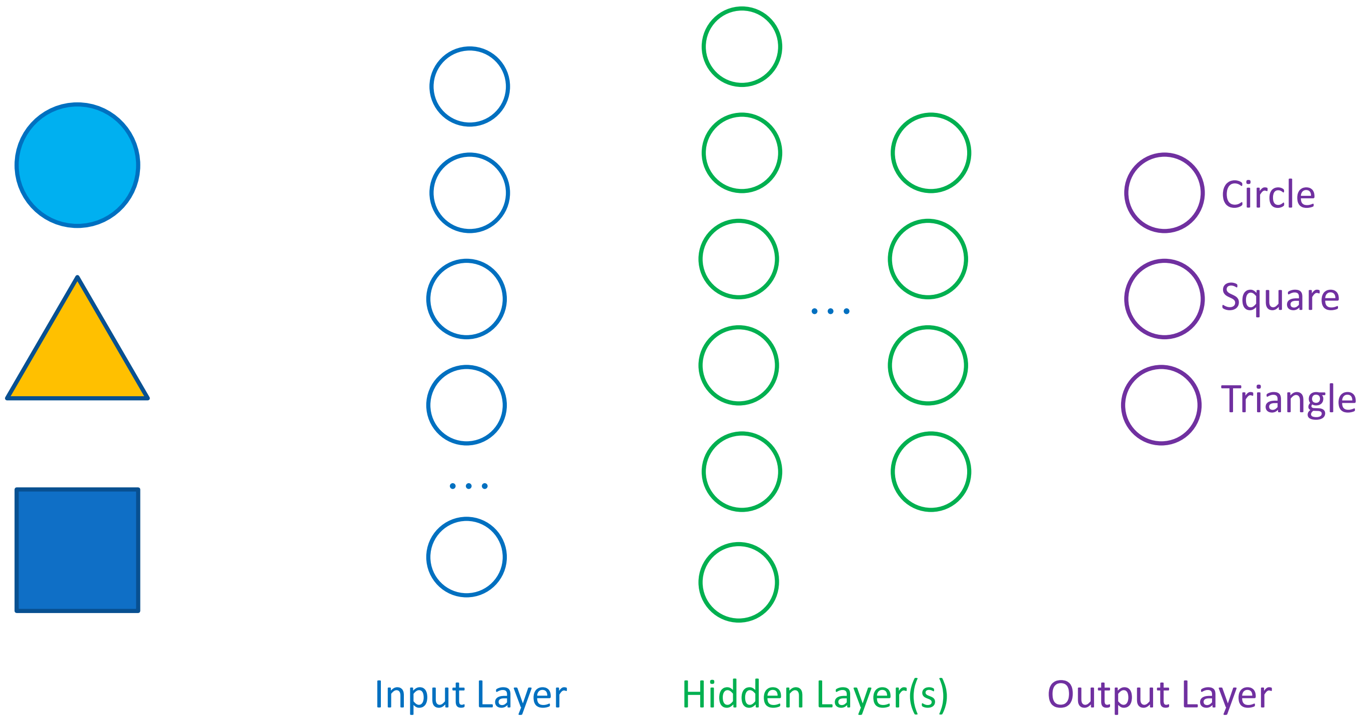
Sample#2: Classification of objects into pre-defined classes

- **Inputs:** Rectangle, Circle, Triangle
- **Goal:** Correct classification



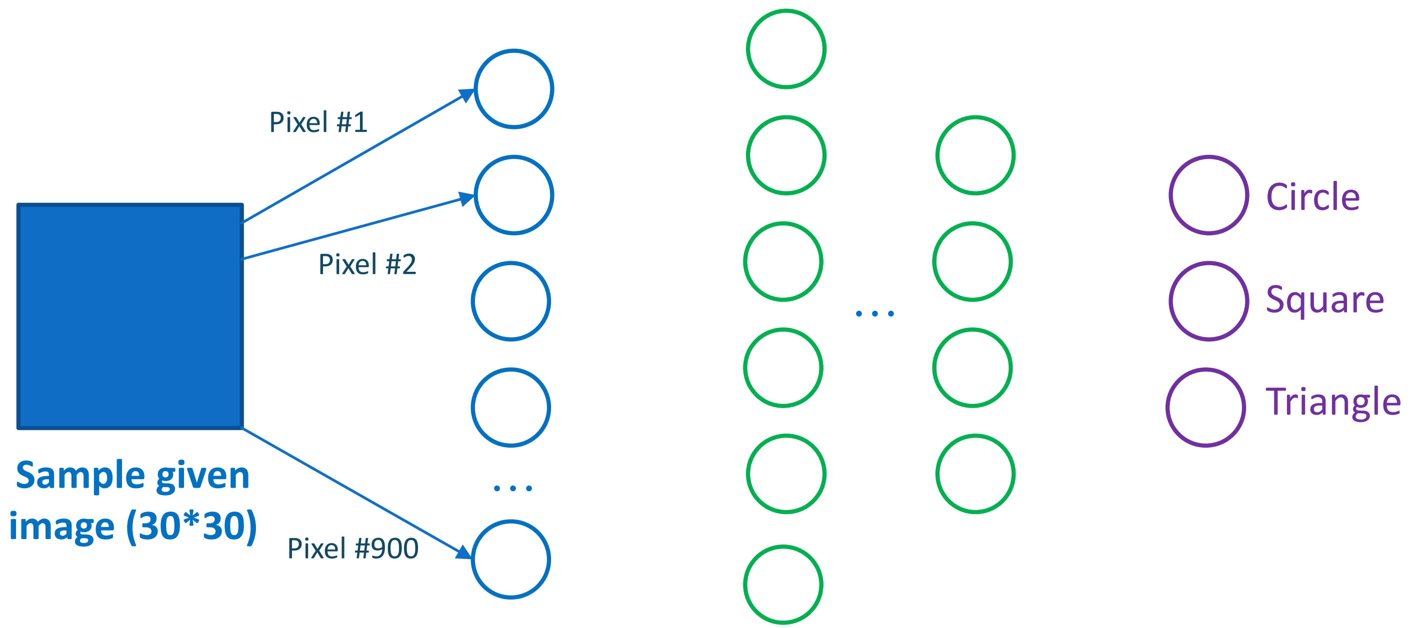
Neural Networks

Sample#2: Classification of objects into pre-defined classes



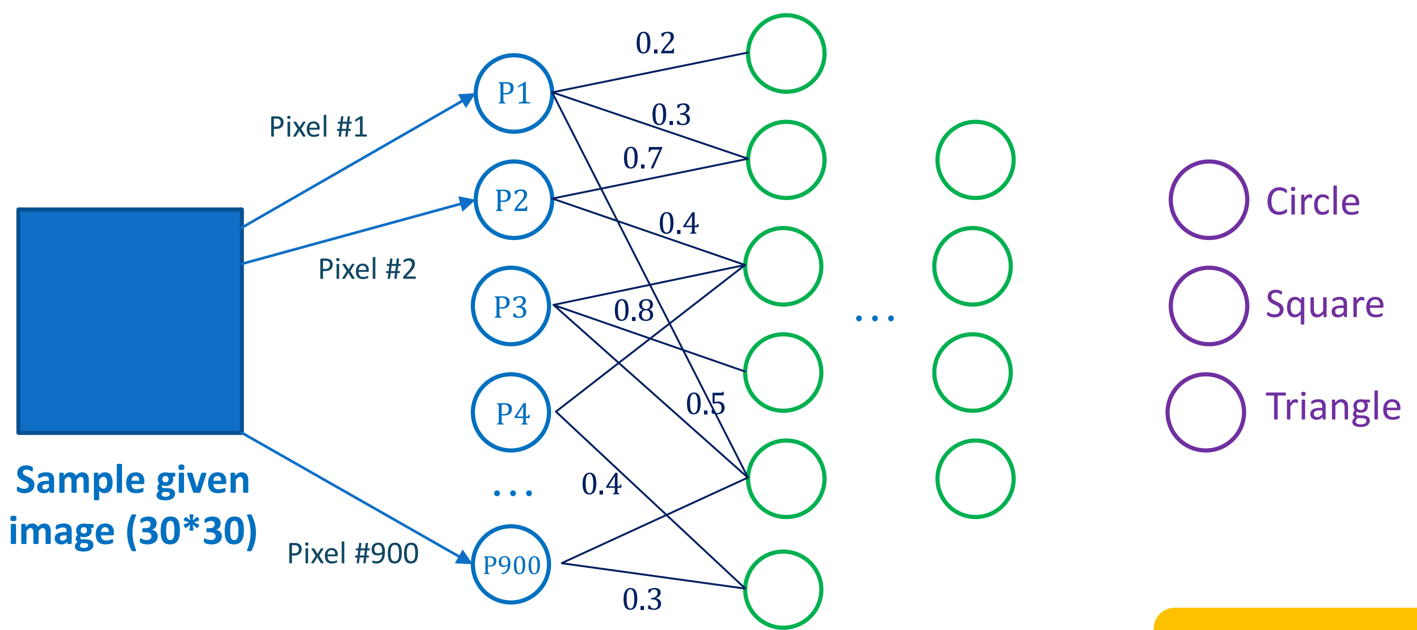
Neural Networks

Each pixel should be mapped to a **single neuron** from the input layer



Neural Networks

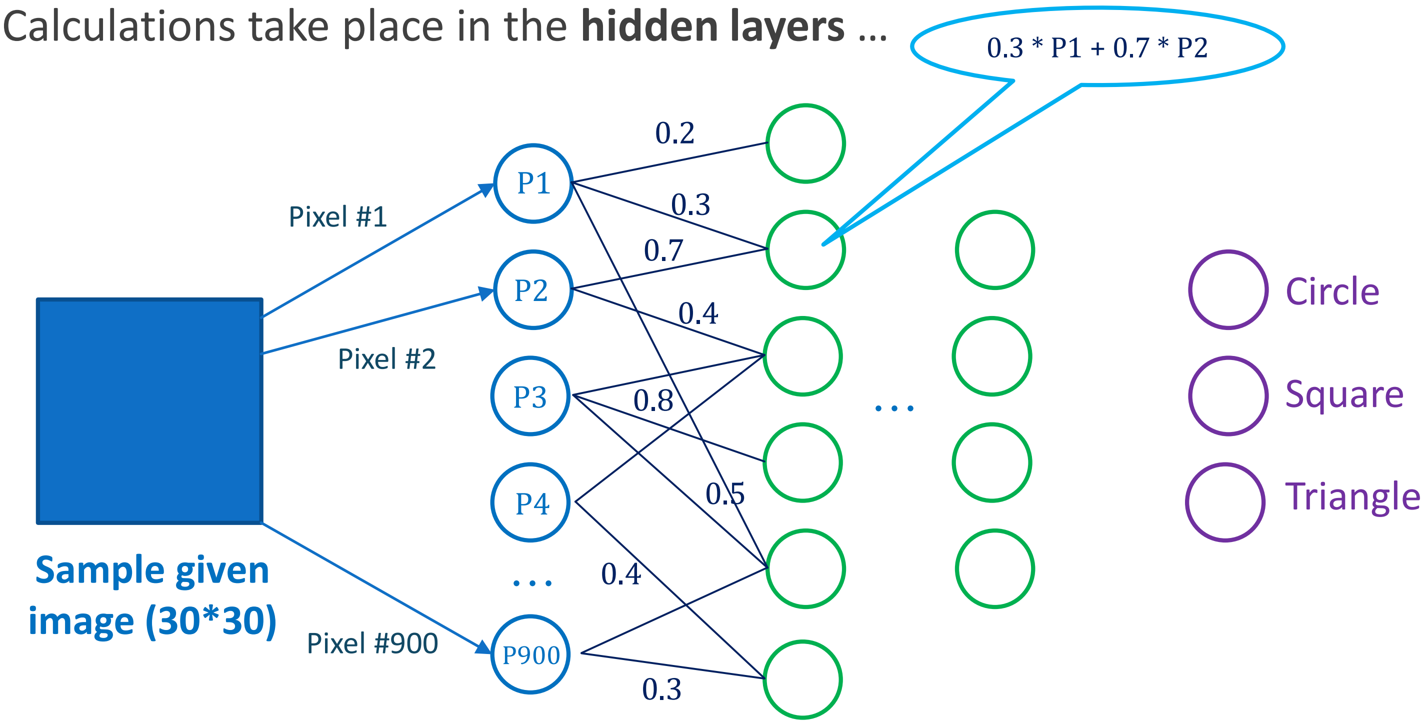
Calculations take place in the **hidden layers** …



Initialization: Random weights

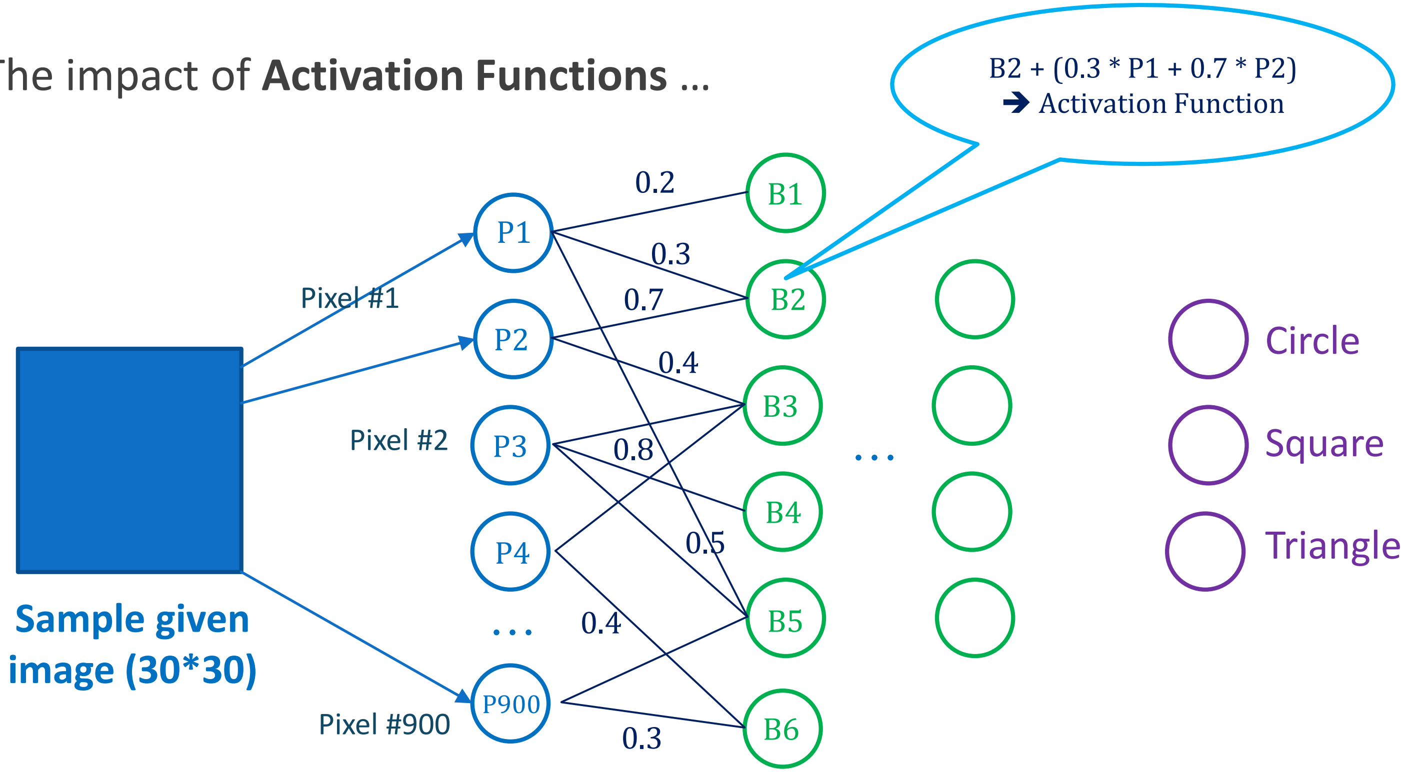
Neural Networks

Calculations take place in the **hidden layers** …



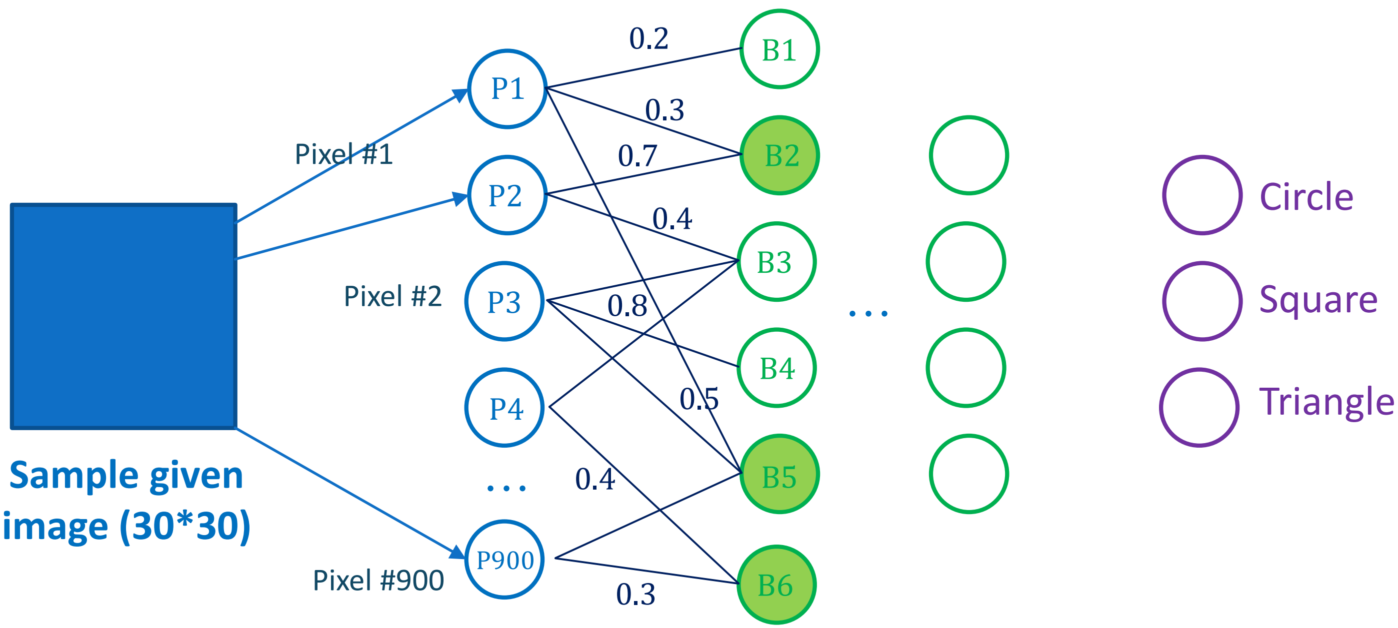
Neural Networks

The impact of **Activation Functions** …



Neural Networks

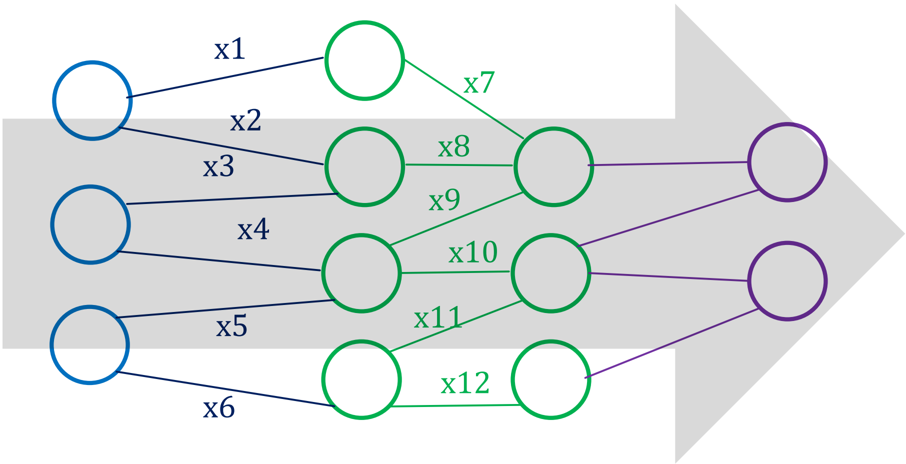
The impact of **Activation Functions** (Active nodes with green backgrounds) …



Neural Networks

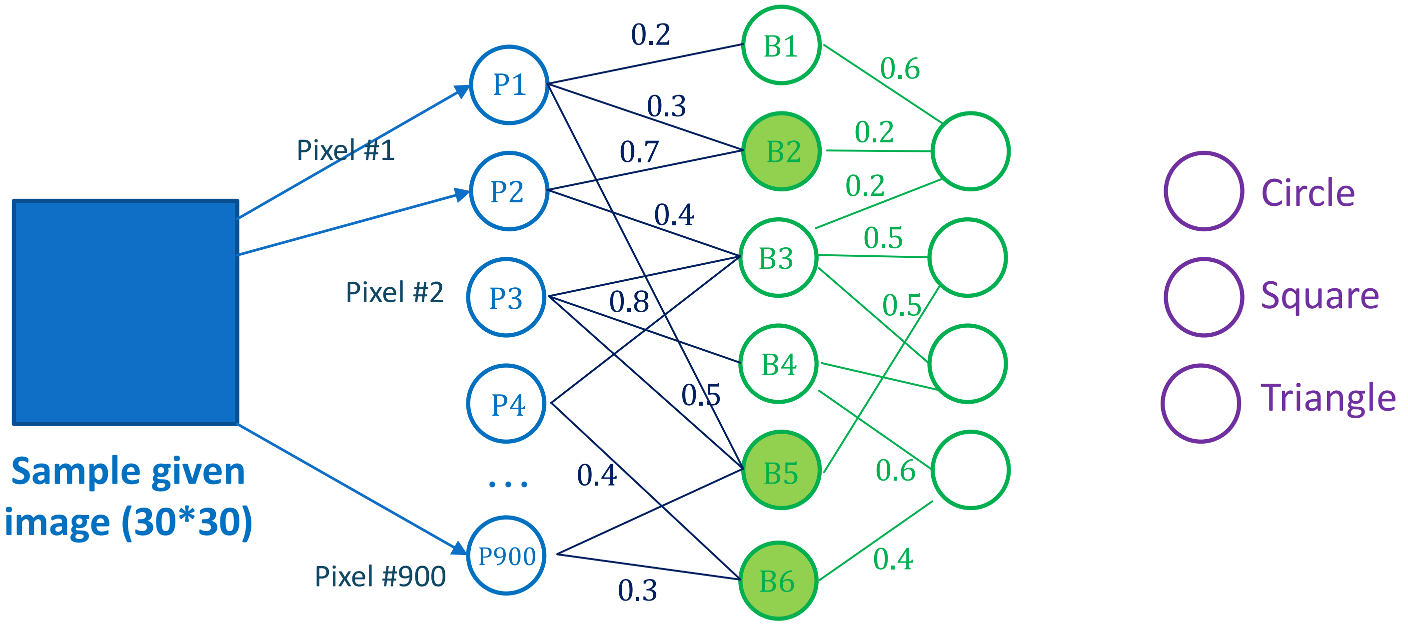
Forward pass

- the main flow of calculations
  - Take the inputs from the previous layer
  - Process them (Summation + AF)
  - Generate outputs
  - Pass them to the next layer
  - Do while generating the F.O



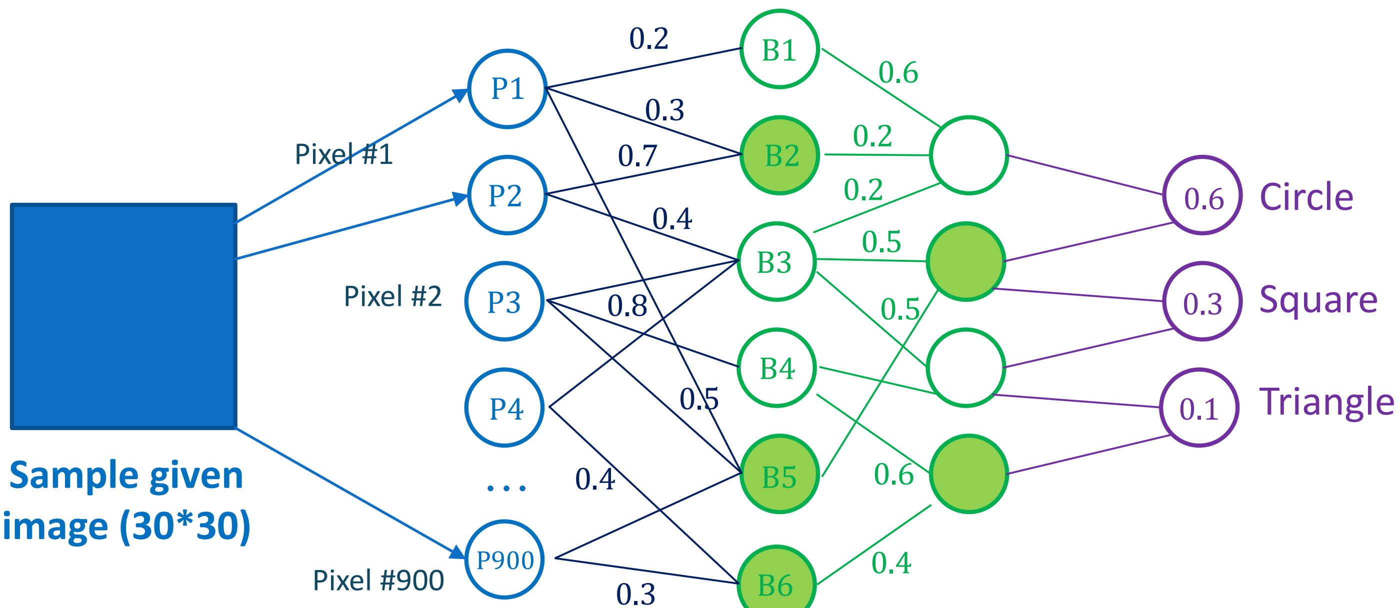
Neural Networks

Forward pass



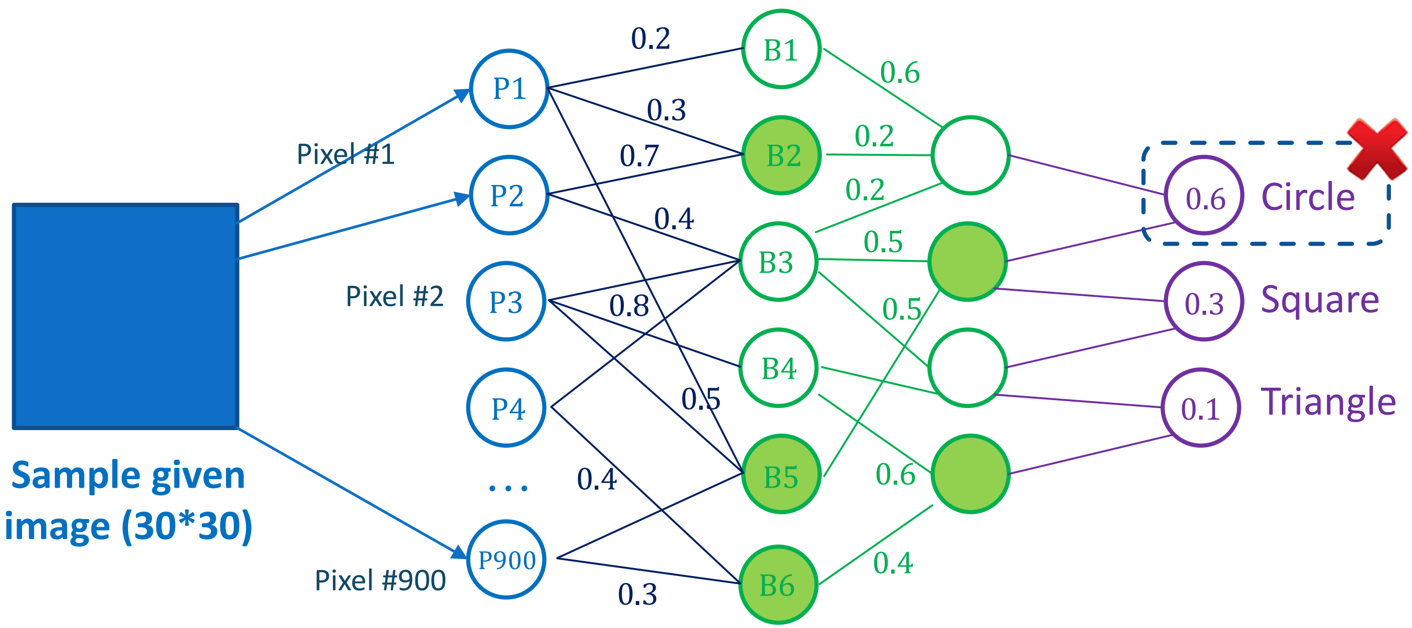
Neural Networks

Forward pass



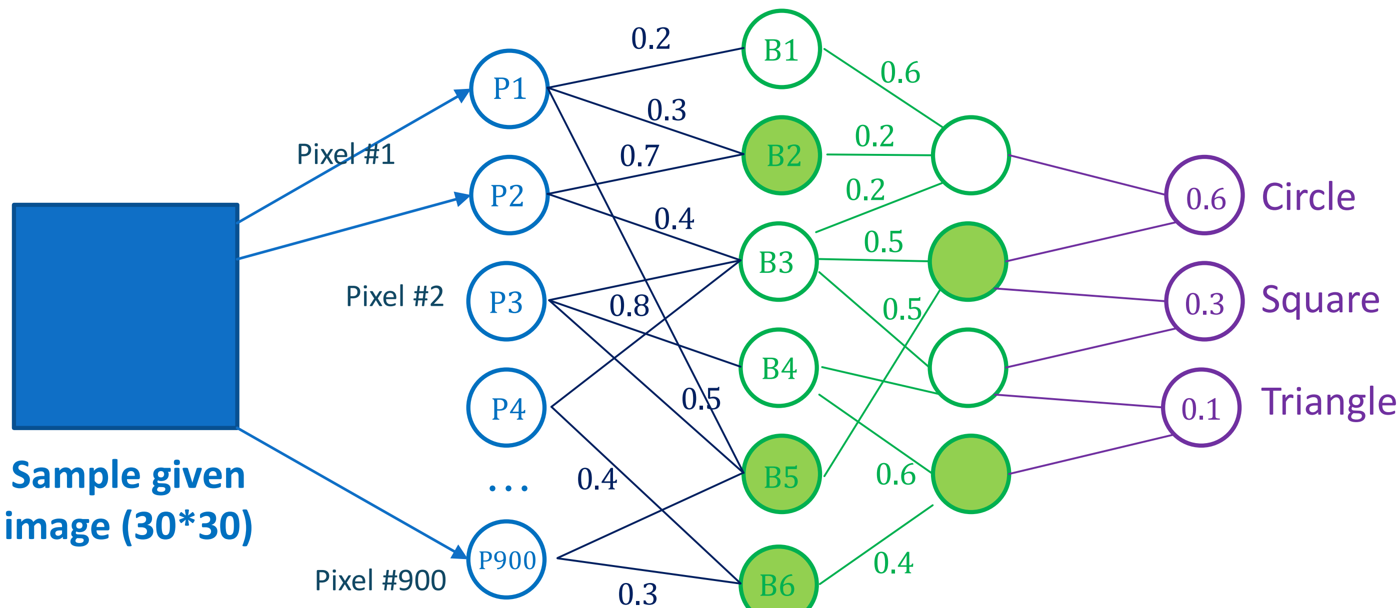
Neural Networks

Wrong prediction! (The highest probability is circle with 0.6)



Neural Networks

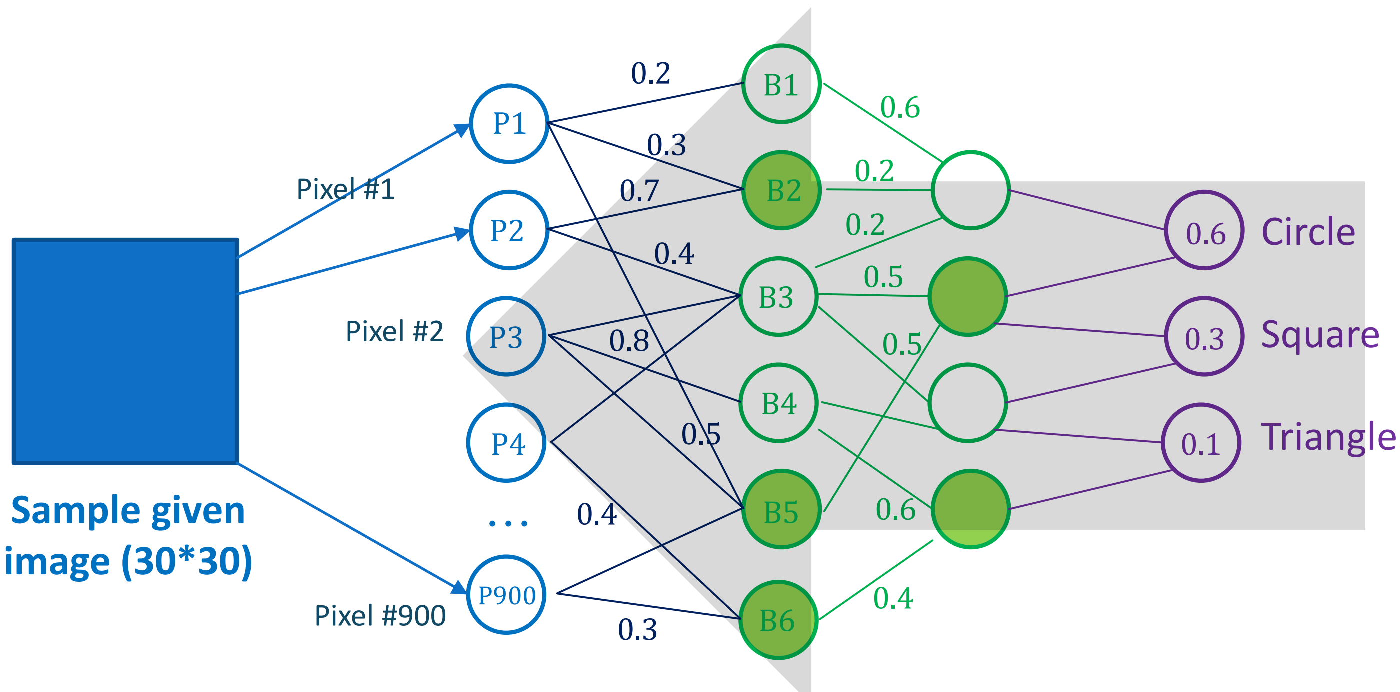
How to know? By comparing to the **Actual (expected) Outputs**



Actual	Error
0	-0.6
1	0.7
0	-0.1

Neural Networks

How to resolve? **Backpropagation**



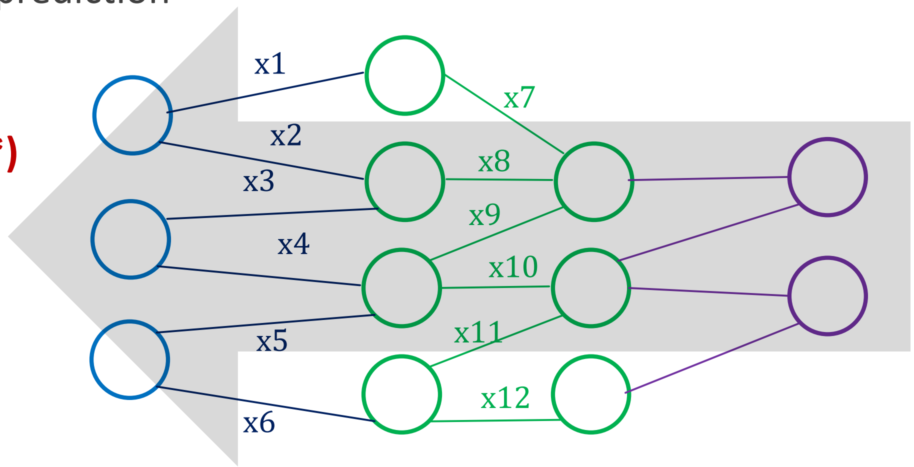
Actual	Error
0	-0.6
1	0.7
0	-0.1

Neural Networks

Backpropagation (the backward pass)

- Why do we need this?
  - To discover the optimal weights for neurons
  - To minimize the **Loss Function (*)**
  - To generate the best possible prediction
- How?
  - **Gradient descent algorithm (*)**

*** We'll talk about them soon!**



Training the Network

- In training a neural network, we feed some data and the network learns how to adjust its weights based on it
- The process is **Supervised**
  - We already know the **Correct Answers (Actual)**
- By comparing Actual and Predicted results, we tell the network:
  - The result was correct, no need to change the weights!
  - The result was incorrect, change the weights and get closer to the correct answer!
- Simply we can define **Loss** as:
  - The cost of incurred from incorrect predictions

Training the Network

Empirical Loss (Empirical Risk Minimization)

- Actually, the average of all individual losses
  - Comparing the predicted and real values
- Total loss over all dataset samples

$$J(w) = 1/n \sum_{i=1}^{n} Loss(h(x_i), y_i)$$

Binary Cross Entropy Loss (Log Loss for Binary Classification)

- Where the predicted value is a probability between 0 and 1
- The goal is to solve a binary problem (pass or fail)
- Compare the real and estimated distributions

$$J(w) = 1/n \sum_{i=1}^{n} (y_i . \log(h(x_i)) + (1 - y_i) . \log(1 - h(x_i))$$

Training the Network

Mean Squared Error Loss

- Used in regression models that provide continuous real numbers
  - E.g. what will be someone's final grade? (non-binary)
- Subtracting the actual and predicted values
- Then, calculating the average of its squared errors

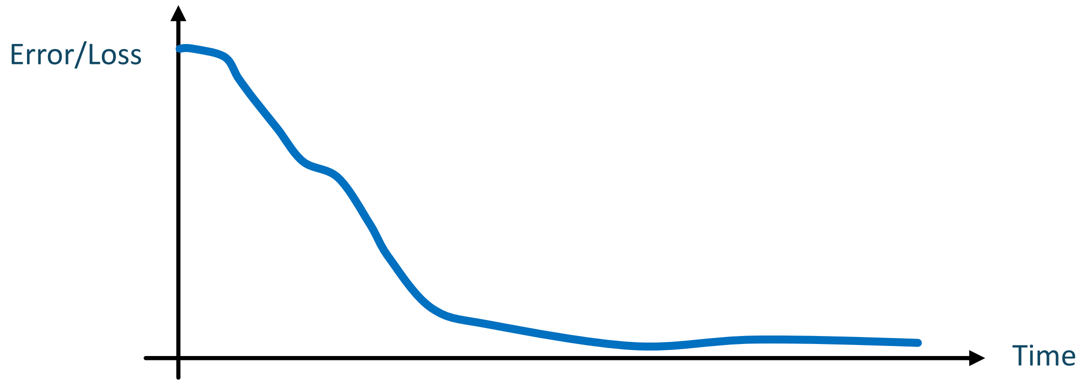
$$J(w) = 1/n \sum_{i=1}^{n} (h(x_i) - y_i)^2$$

So, we can use different types of **Loss functions** for different problems.
Using these quantified errors, we can train our ANN to **find optimal weights**.

Training the Network

How to deal with loss/error functions?

- They are very useful, as they show how far the Actual (expected) Output is from the current (predicted) outputs of ANN
- Our goals?
  - To minimize the loss function (as much as possible!)
  - To bring outputs as close as possible to the actual values



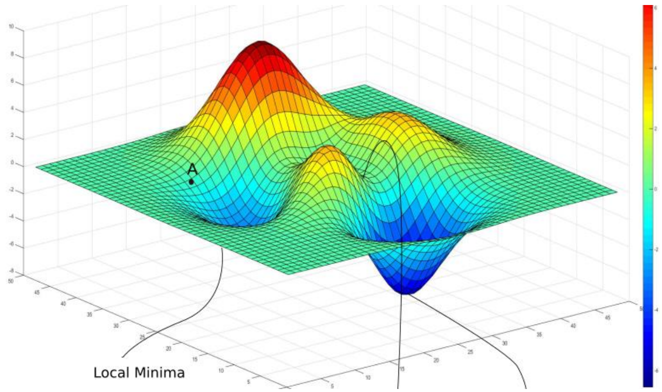
Training the Network

Loss Optimization

- Now, we want to minimize the loss to achieve the optimal weights
  - Actually, the goal is to achieve the lowest loss
- If W^* is optimal values for weights vector W:

$$W^* = argmin\, J(W)$$

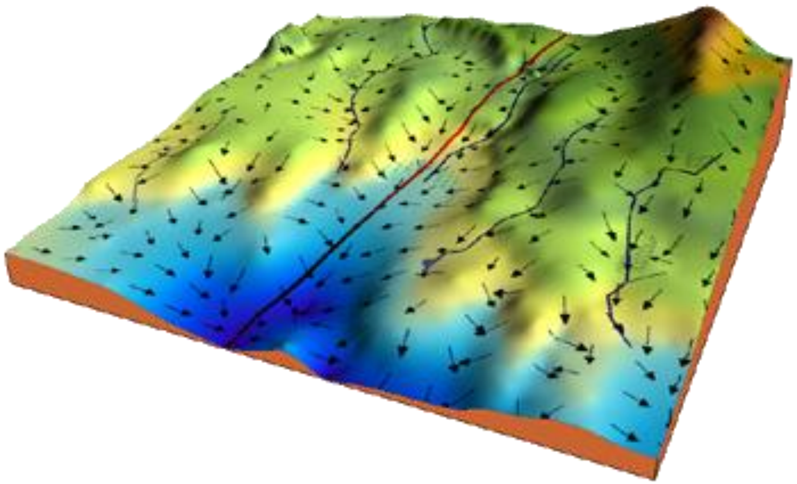
- Gets more complicated when the number of weights increases
- **Goal:** Find the minimum loss
  - Local Minima and Global Minima
- How? **Gradient Descent Algorithm**



Training the Network

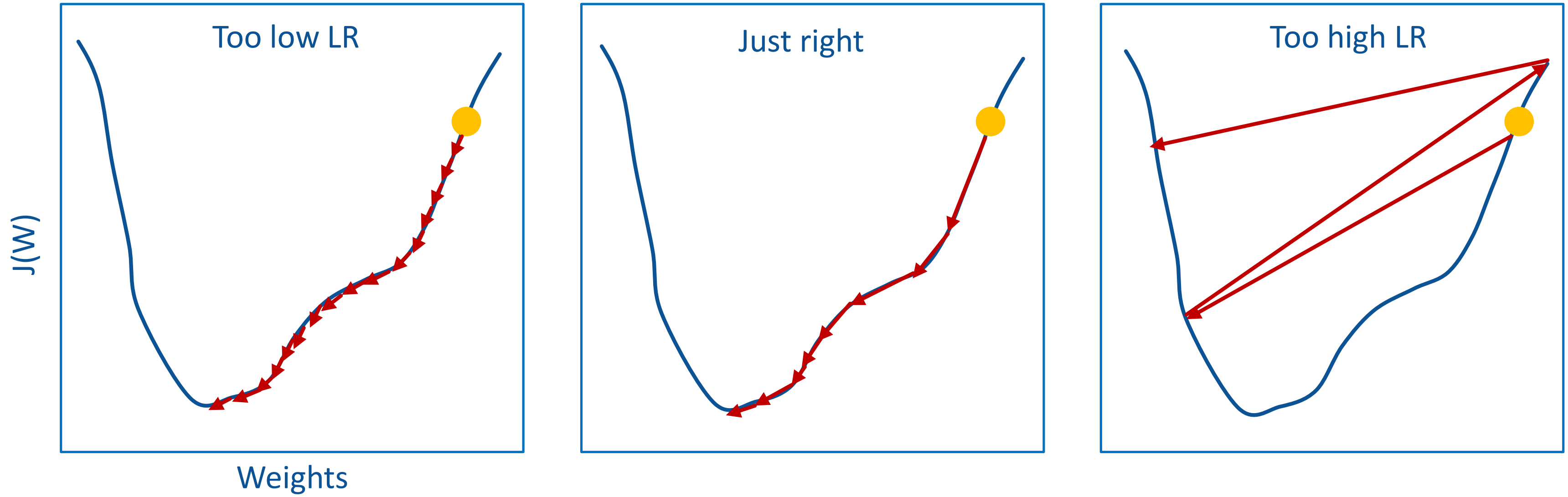
Gradient Descent Algorithm

- An iterative optimization algorithm for finding a local minimum
  - By iteratively moving in the direction of the steepest descent
- Just like moving from a mountain towards the sea
  - An step-to-step downhill in the direction with **negative gradient**
- **Learning Rate:** the size of steps
  - High LR ➔ We may miss the optimal point
  - Low LR ➔ We may be too slow



Training the Network

Gradient Descent Algorithm

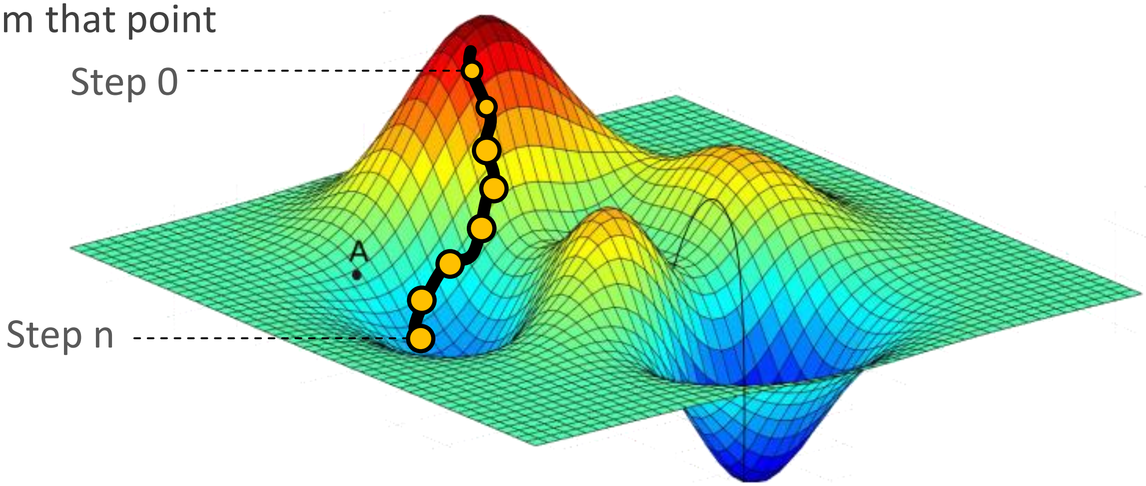


How to find proper LR? **1) trail and error**, **2) Adaptive LR**

Training the Network

Gradient Descent Algorithm

- Pick any random point in the landscape
- Compute gradient using $\frac{\partial J(W)}{\partial W}$
- Take a small step towards the lower stages (steep)
- Repeat this process until reaching a **Local Minimum**
- Get the weights from that point

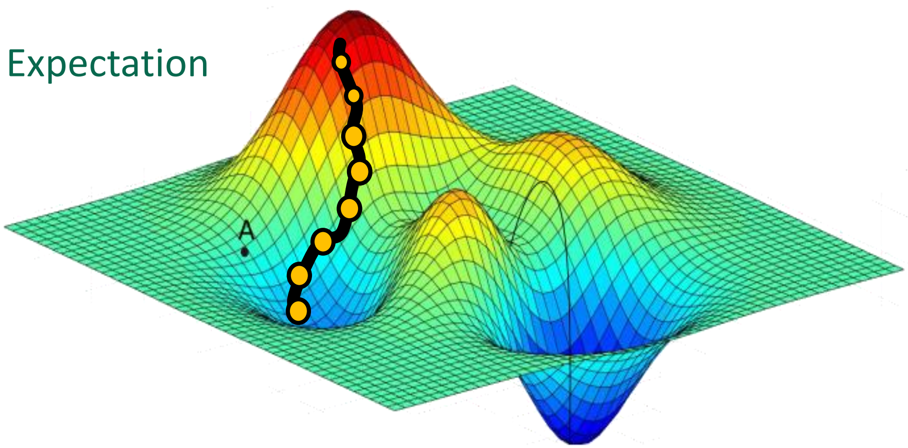


Training the Network

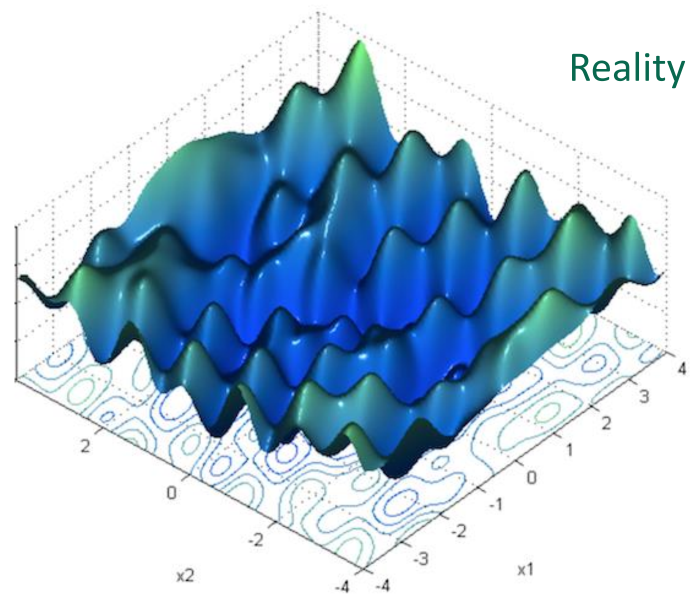
Gradient Descent Algorithm

- It is not always that simple!
  - In real-world scenarios, the algorithm may stuck

Expectation



Reality



Training the Network

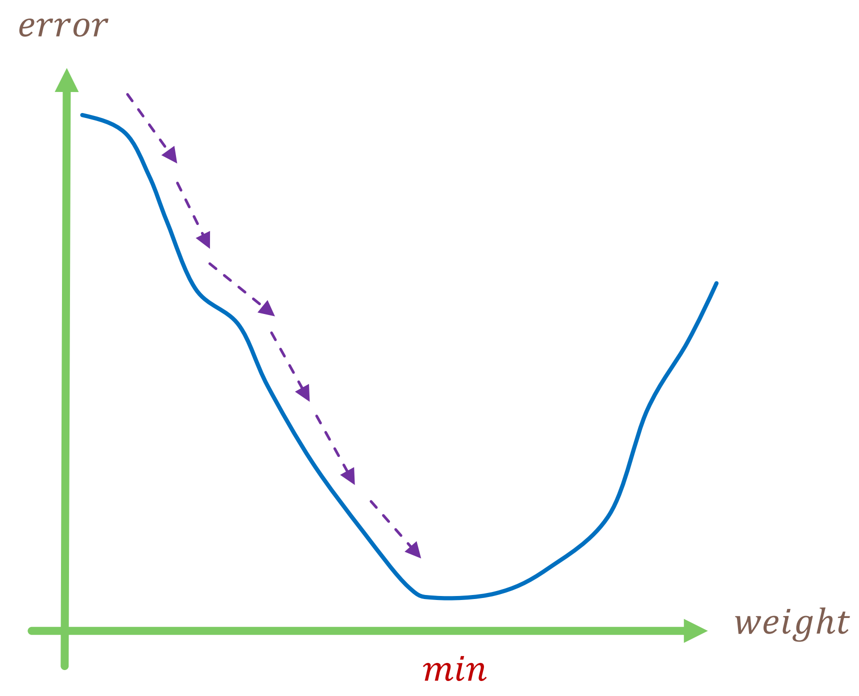
Gradient Descent Algorithm

- How to update weights?

$$w_i(t) = w_i(t-1) + \Delta w_i(t)$$

$$\Delta w_i(t) = \mu\left(-\frac{\delta J(W)}{\delta w}\right)$$

Learning rate



Training the Network

Gradient Descent Algorithm

- How do weight changes spread in the network?
  - **Remember**: the main goal is to decrease the final loss
  - **Remember**: we should use Backpropagation (Slide#48)
  - According to GDA:

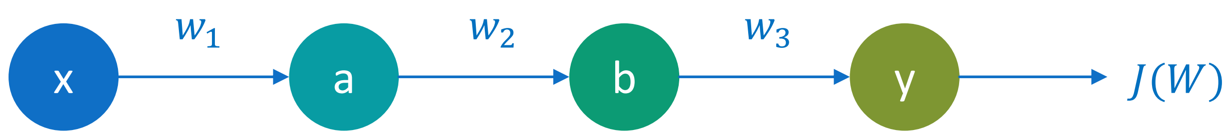
$$x \xrightarrow{w_1} a \xrightarrow{w_2} b \xrightarrow{w_3} y \longrightarrow J(W)$$

$$\frac{\delta J(W)}{\delta w_1} = \frac{\delta J(W)}{\delta y} * \frac{\delta y}{\delta b} * \frac{\delta b}{\delta a} * \frac{\delta a}{\delta w_1}$$

Training the Network

Gradient Descent Algorithm

- How do weight changes spread in the network?



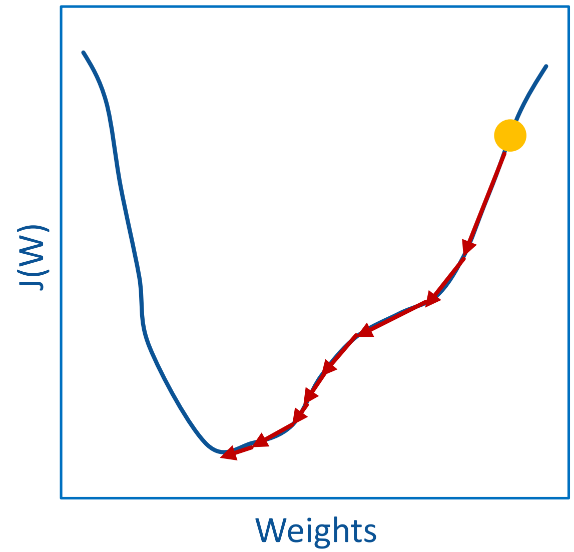
$$\frac{\delta J(W)}{\delta w_1} = \frac{\delta J(W)}{\delta y} * \frac{\delta y}{\delta b} * \frac{\delta b}{\delta a} * \frac{\delta a}{\delta w_1}$$

- This way, we can calculate all the weights from output to input
- **Good news:** we do not need to do this practically in our applications!
  - Backpropagation is implemented in different DL frameworks

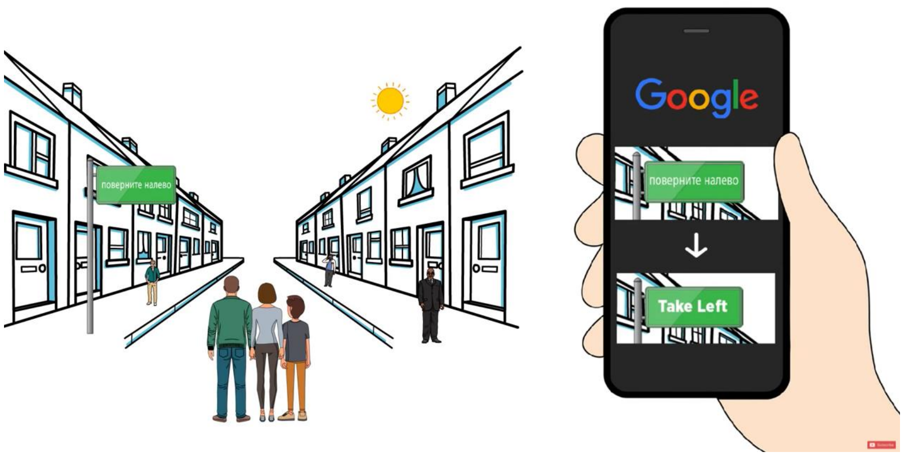
Training the Network

Adaptive Learning Rates

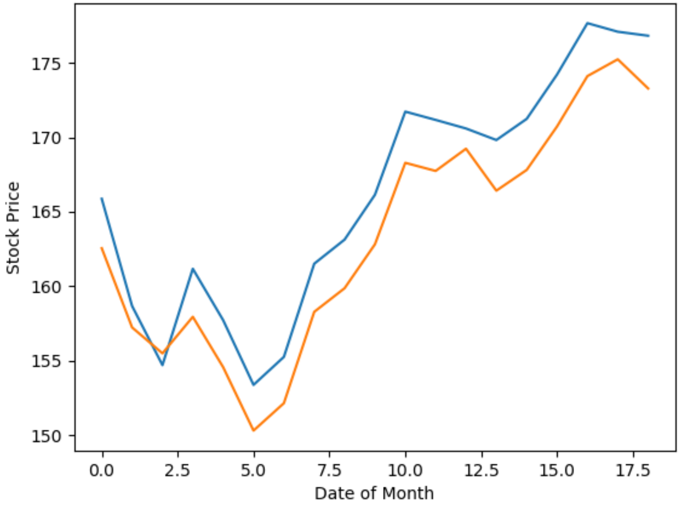
- In contrast with fixed LR, the value of LR may vary over time
  - It may become larger or smaller
- Factors affecting the value of LR
  - Learning speed (algorithm)
  - Size of weights
  - The value of gradient
- **Algorithms (optimization)**
  - Adam, AdaDelta, etc.



Use cases of ANNs

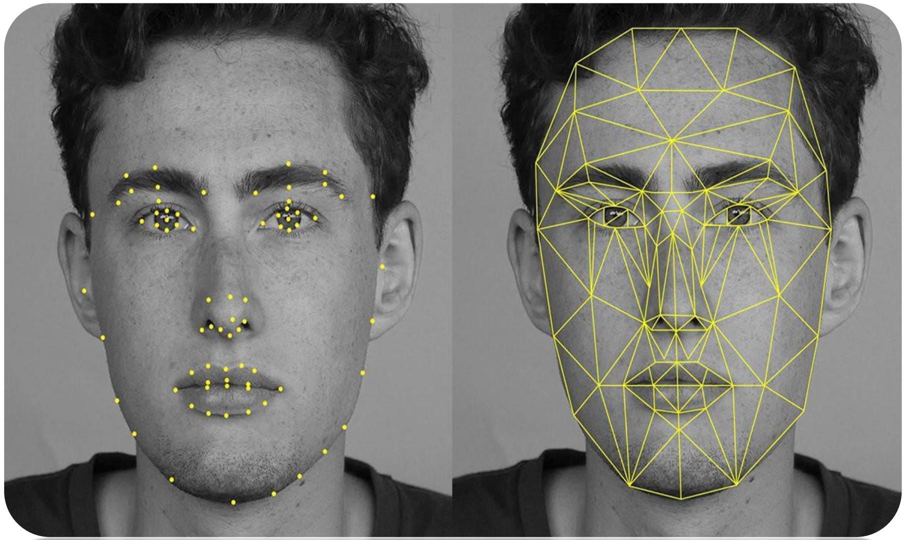


Google online translation tool

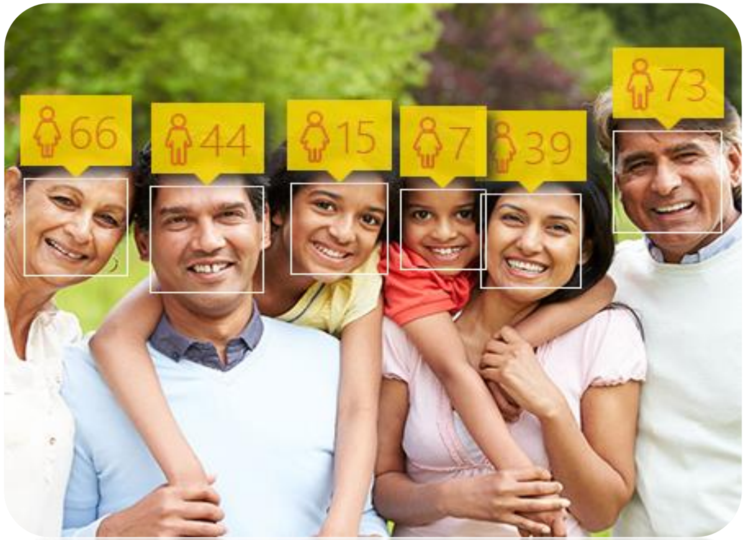


Stock market prediction

Use cases of ANNs



Face recognition



Age estimation

Use cases of ANNs

Image completion

***Reference**: https://github.com/bamos/dcgan-completion.tensorflow*

Use cases of ANNs

Suggested Projects – General ideas

Beginner **Image Classification**

- Identify the class of an input image

Beginner **Visual Tracking**

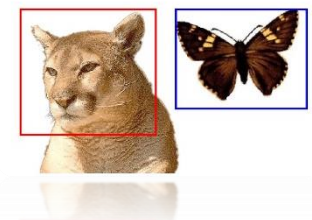
- Locate moving objects for surveillance

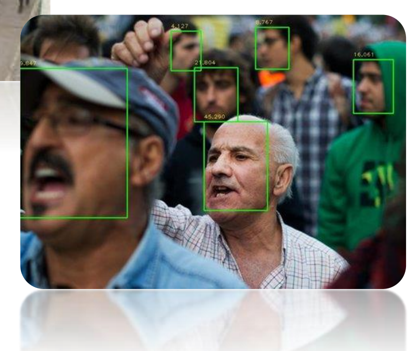
Beginner **Face Detection**

- Track and visualize human faces

Intermediate **Image Caption Generation**

- Analyze the context of an image





Use cases of ANNs

Suggested Projects – General ideas

Intermediate Text-to-Image

- Generate image from a given text

Intermediate Gender/Age Detection

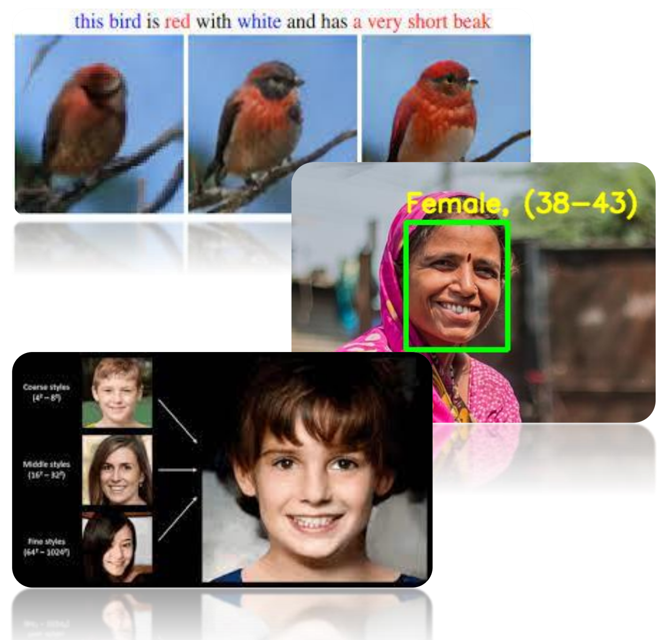
- Estimate human age from facial features

Expert Human Face Generation

- Track and visualize human faces

Expert Colorize Black & White Images

- Make B&W images colorful using a deep model



Use cases of ANNs

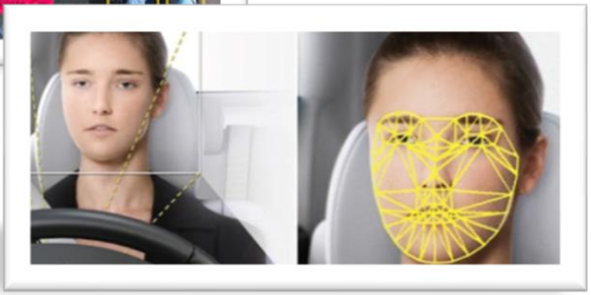
Suggested Projects – Intelligent Transportation Systems Lab

Vehicles

- Vehicle detection and tracking
- Vehicle type classification
- Vehicle pose estimation

Drivers

- Face pose detection
- Driver actions detection
- Driver drowsiness detection



Assignment & Homework

Find your Field of Interest

- Google your desired field of interest
- Find issues that can be resolved using Deep Learning
- Find some academic papers about it
- Send me your titles to discuss

References

Websites

- http://www.IntroToDeepLearning.com
- https://www.towardsdatascience.com
- https://data-flair.training/blogs/deep-learning-project-ideas/
- https://www.deeplearning.ai/
- https://blog.paperspace.com/a-practical-guide-to-deep-learning-in-6-months/
- https://wiki.pathmind.com/neural-network

Questions?