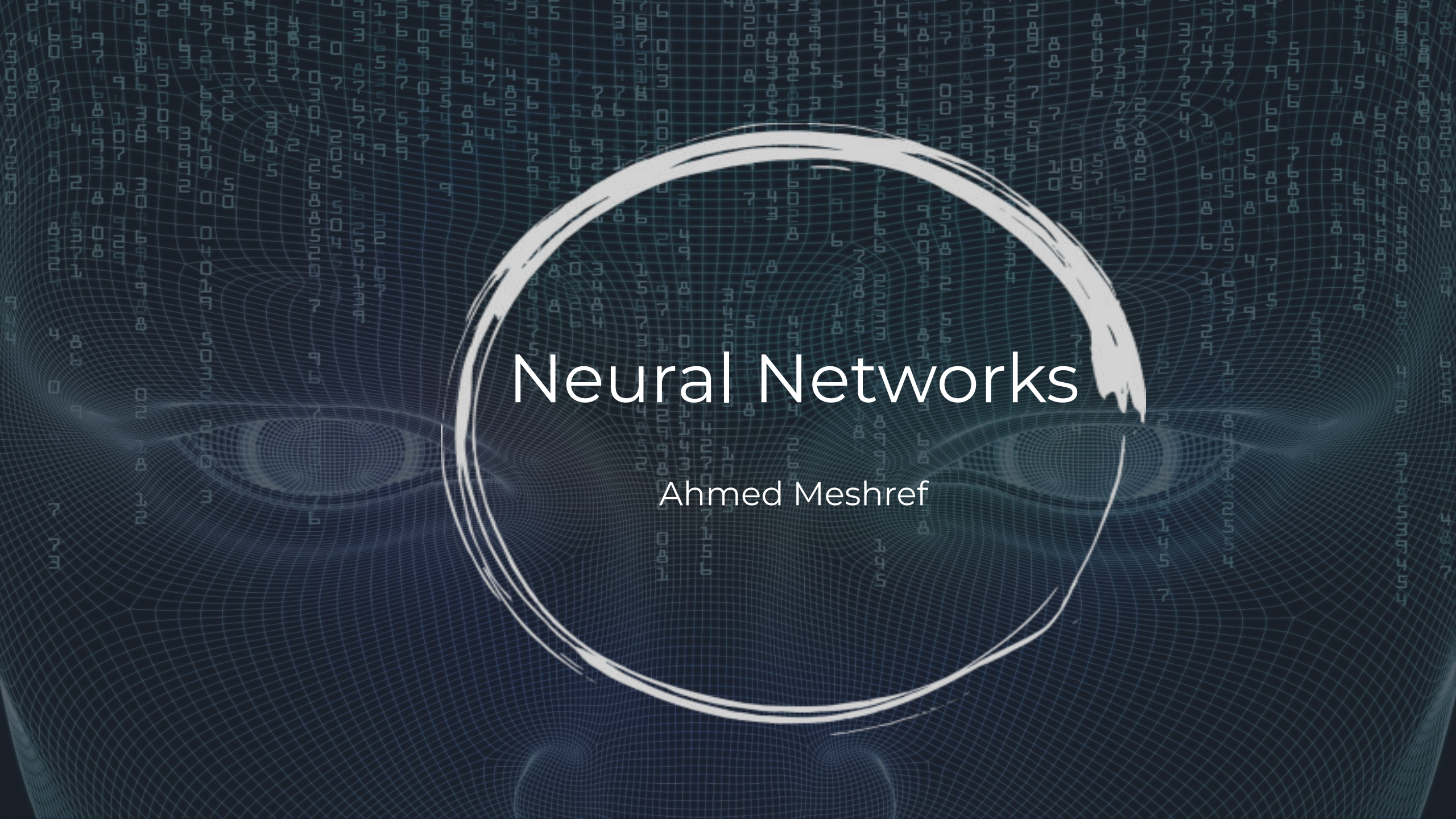
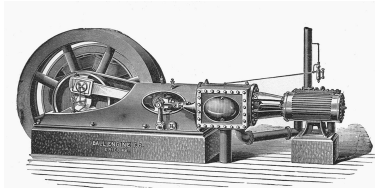

Neural Networks
Ahmed Meshref

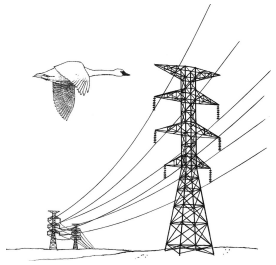
THE 4TH INDUSTRIAL REVOLUTION IS HERE

Machines



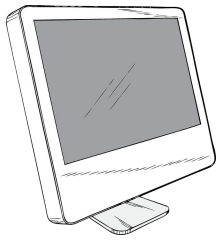
1760

Electricity



1900

Digital



1960

AI
+$16T

NOW

WHAT IS AI

Take guessing out of decision-making

Decisions based on past evidence

A lot faster than humans can

AI TRANSFORMING THE WORLD

PROBLEMS WE COULDN'T TACKLE BEFORE

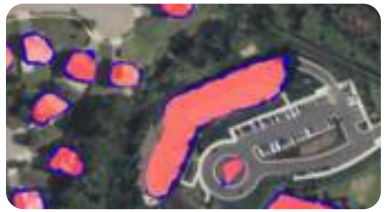
- Discovery and cure of new diseases like amyotrophic lateral sclerosis
- Safer driving. According to Bloomberg 25% of all cars on the road will be fully autonomous by 2036
- Smarter use of energy
- Preserving the environment and reversing the impact of climate change

AI IS REVOLUTIONIZING INDUSTRIES – CASES

Ask what AI can't do

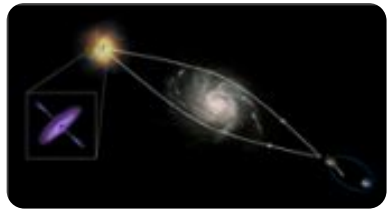
AUTOMOTIVE
Autonomous Driving
Object detection
Mobility as service and connected vehicle

FINANCIAL SERVICES
Claim fraud
Customer service chatbots/routing
Risk evaluation



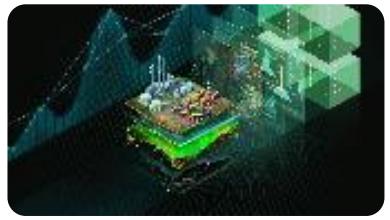
GOV'T & DEFENSE
Identifying threats
Satellite imagery & Optimize logistics management
Support emergency response teams

HEALTHCARE
Improve Clinical Care
Drive Operational Efficiency
Speed Up Drug Discovery



HIGHER EDUCATION/RESEARCH
Climate & Melting of Sea Ice Prediction
Design more effective drugs and new treatments for diseases
Predictive analytics in the sports industry

INDUSTRIAL
Remaining Useful Life Estimation
Failure Prediction
Demand Forecasting



OIL & GAS
Sensor Data Tag Mapping
Anomaly Detection
Robust Fault Prediction

SMART CITIES
intelligent video analytics
Optimize city resources from parking management to law enforcement and city services

RETAIL
Supply Chain & Inventory Management
Price Management / Markdown Optimization

TELCO
Detect Network/Security Anomalies
Forecasting Network Performance
Network Resource Optimization (SON)

AI VS ML VS DL

ARTIFICIAL INTELLIGENCE

Any application using past data evidence to help computers answer future questions.

As computers, they can perform many more questions much faster than humans

Answer many more questions much faster and better

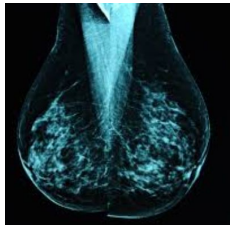
MACHINE LEARNING

A lot of data + simple layer functions like linear regressions = teach computer to answer simple questions

How will ice-cream sales vary by the day of the week?

DEEP LEARNING

A ton of data + complex layer functions = teach company to answer more complex questions, closer to humans

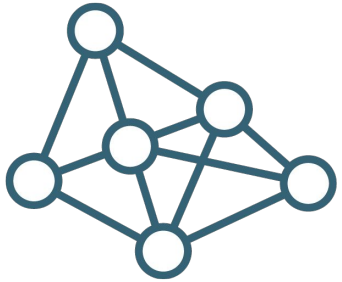


Look at this MRI scan, does this patient have cancer?

DEEP LEARNING TOPICS

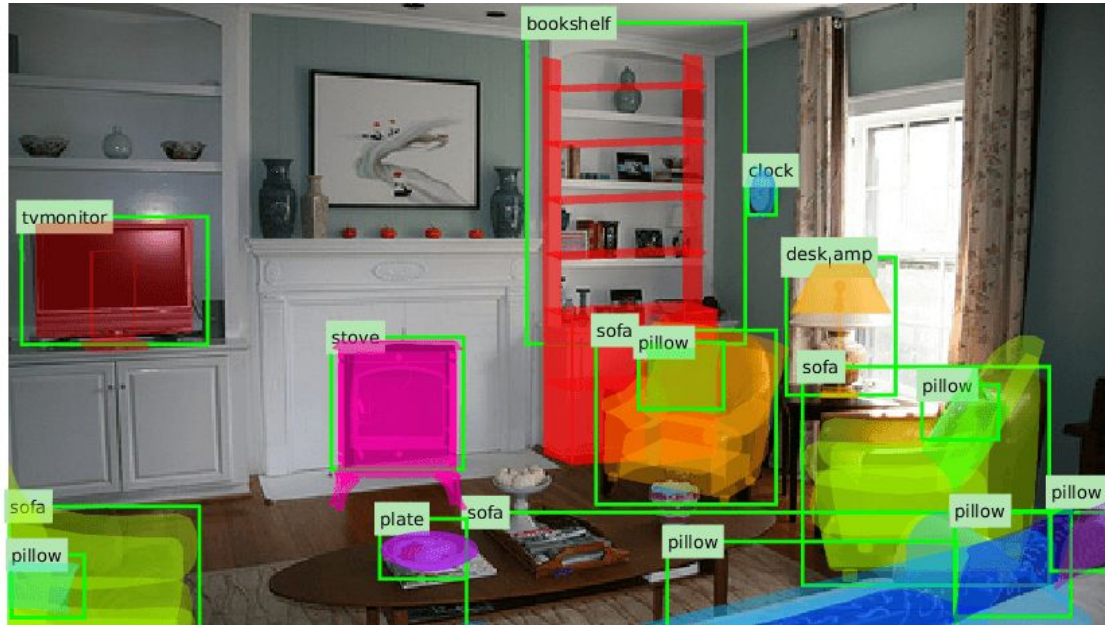
SPEECH/AUDIO PROCESSING

COMPUTER VISION



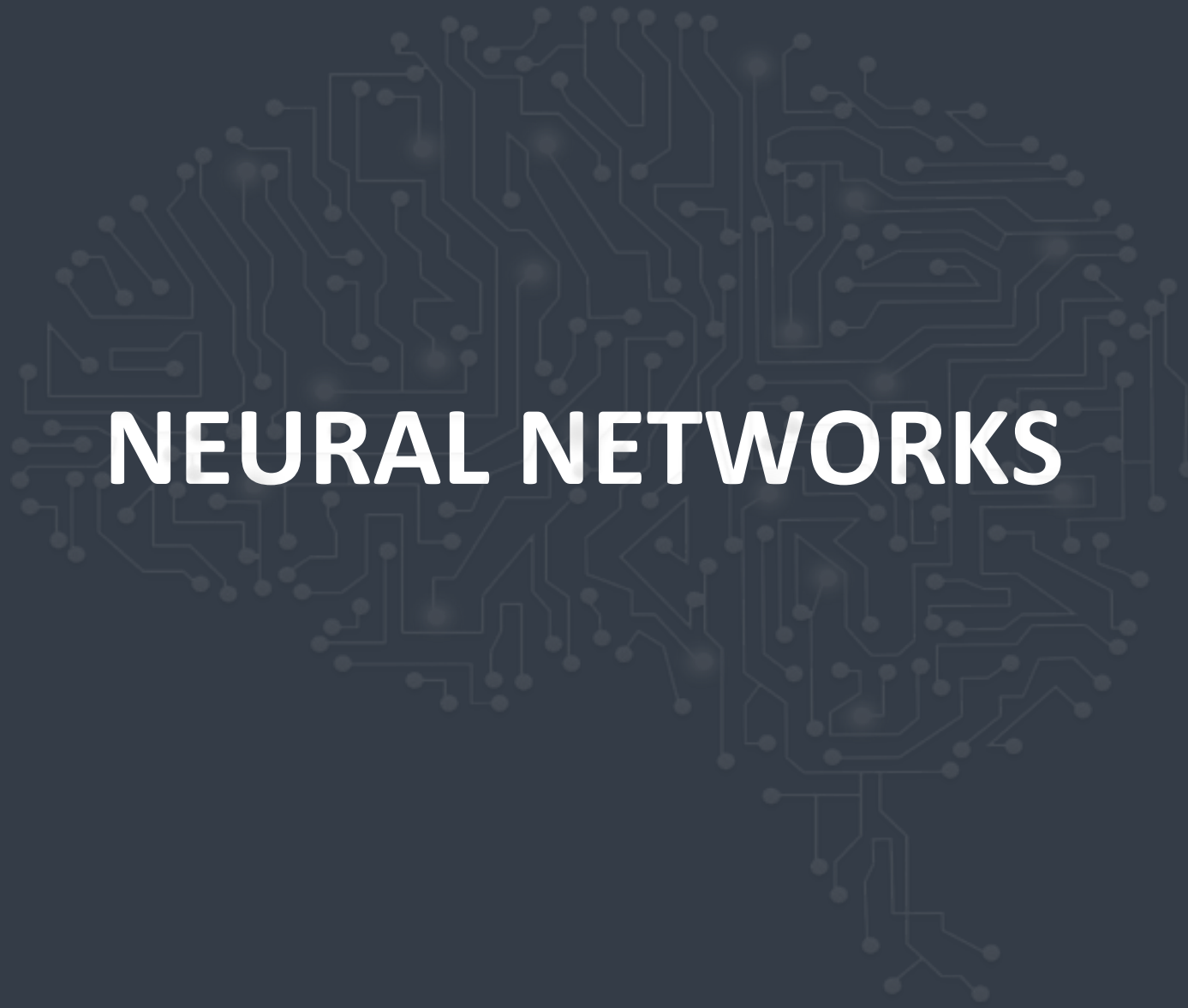
NATURAL LANGUAGE PROCESSING

DEEP LEARNING APPLICATIONS



Object Detection

Speech Recognition

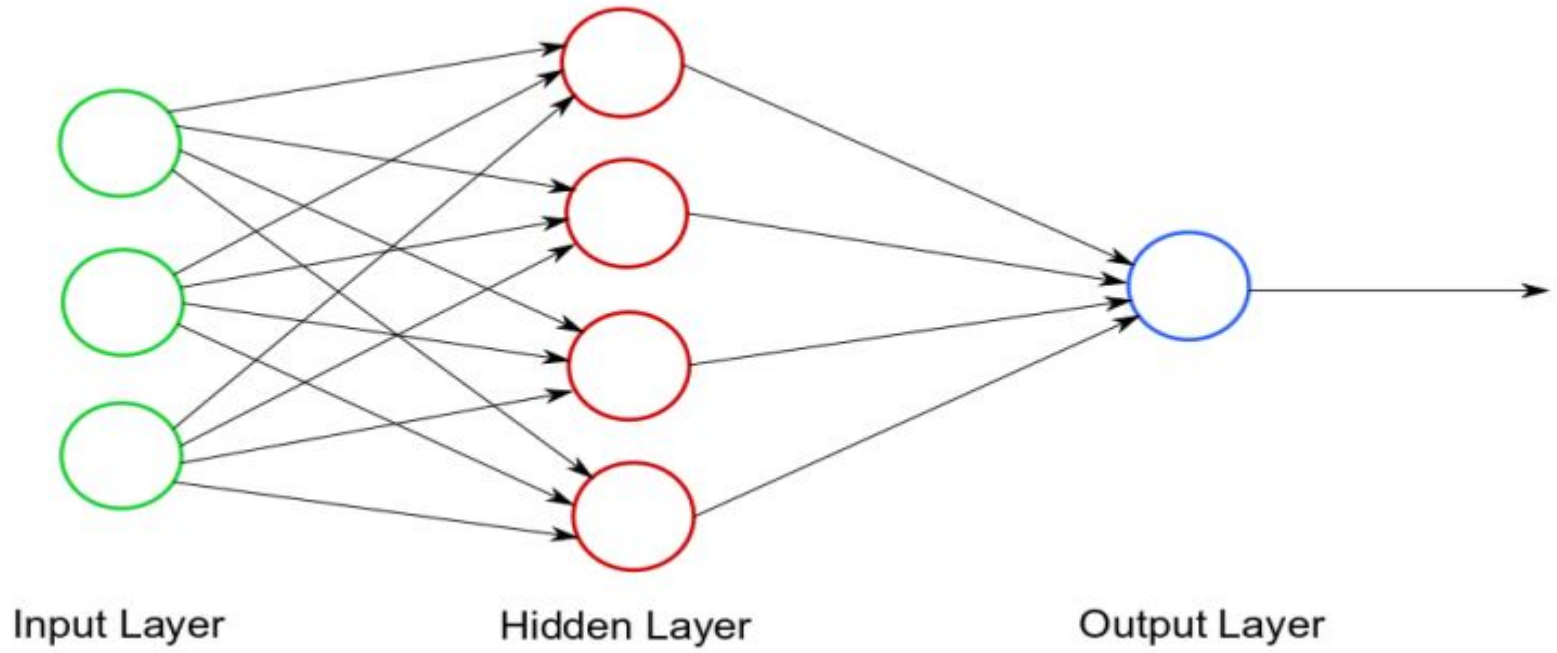


NEURAL NETWORKS

ARTIFICIAL NEURAL NETWORKS

The heart of deep learning

Deep Learning uses different **architecture of Neural Networks** to implement **Machine Learning**

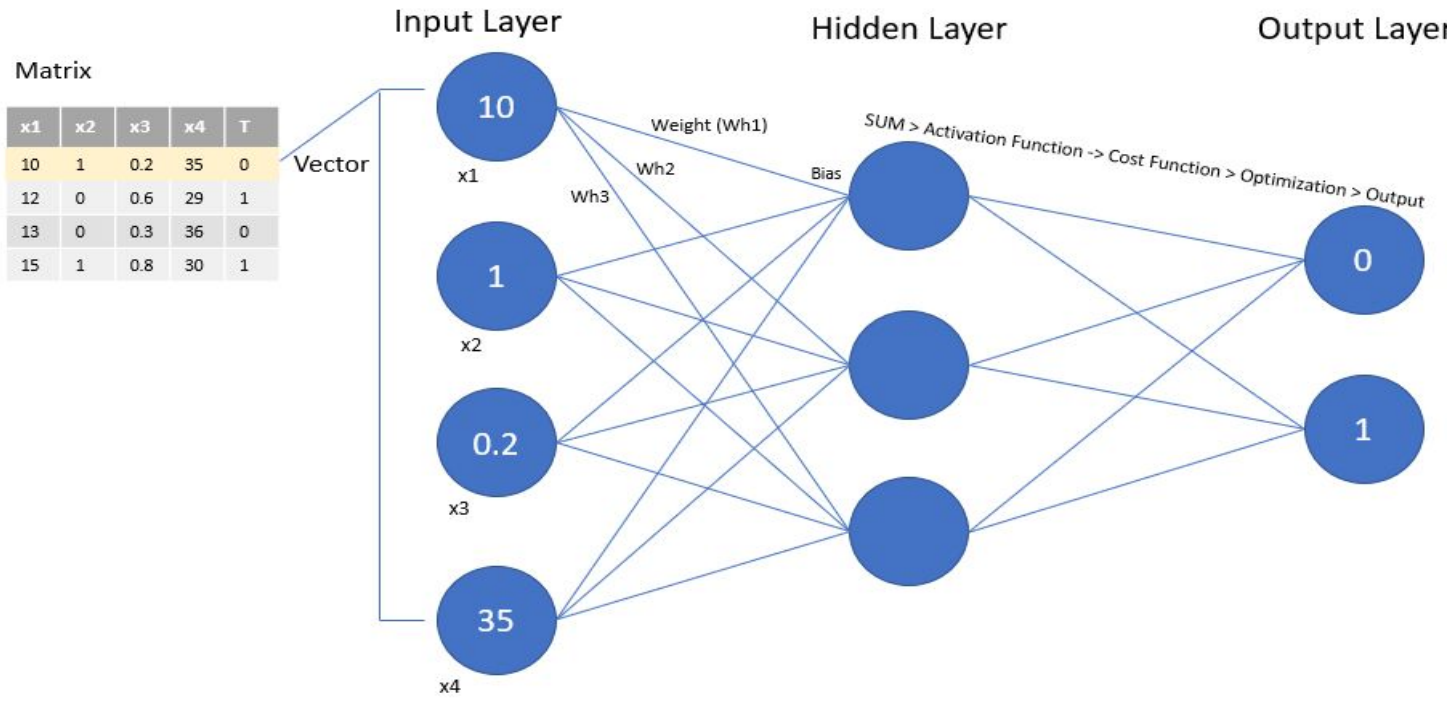


ARTIFICIAL NEURAL NETWORKS

The heart of deep learning

Deep Learning uses different **architecture of Neural Networks** to implement **Machine Learning**

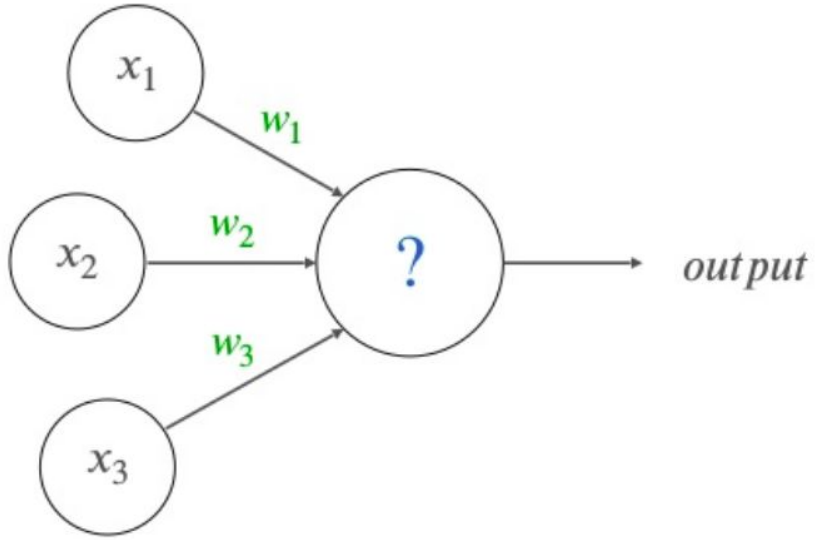
Plain Vanilla architecture



Each layer is connect to the other with different weight matrix represented on each arrow that indicates the importance of each neuron input to the output.

HIDDEN LAYERS

Let’s see how the hidden layers work



$$output = \begin{cases} 0, & \sum_{j=0}^{n} w_j x_j \leq threshold \\ 1, & \sum_{j=0}^{n} w_j x_j > threshold \end{cases}$$

Say, x1, x2, x3 = 1, 1, 0. With weights as follows w1, w2, w3 = 1, 1, 0. Assume that our threshold = 5 performing dot product we will get:

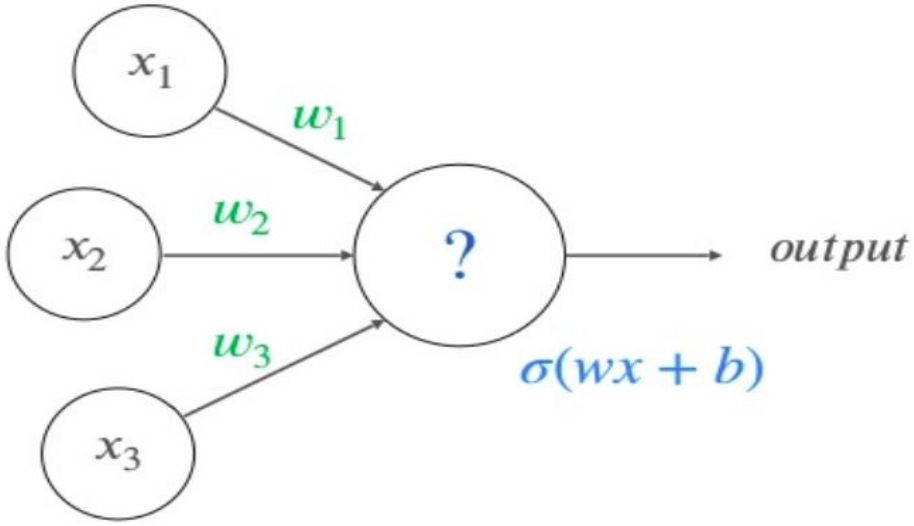
1- w.x = 2

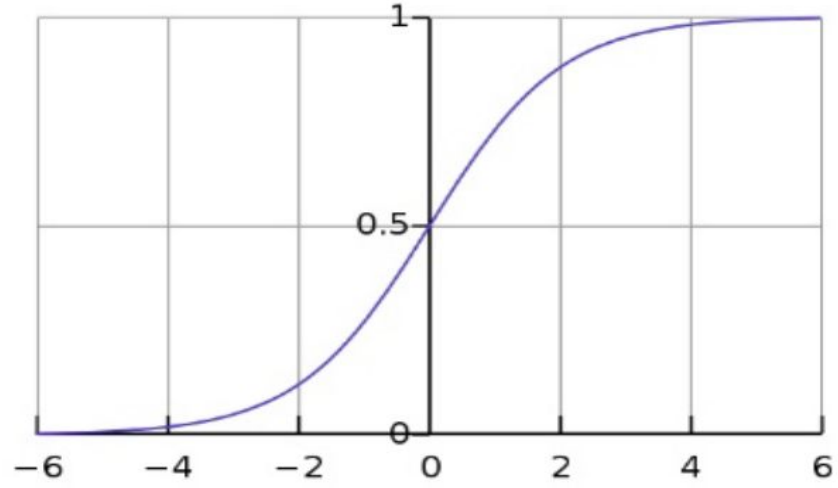
2- 2 > 5

3= Output = 0

ACTIVATION FUNCTION

The more common artificial neuron



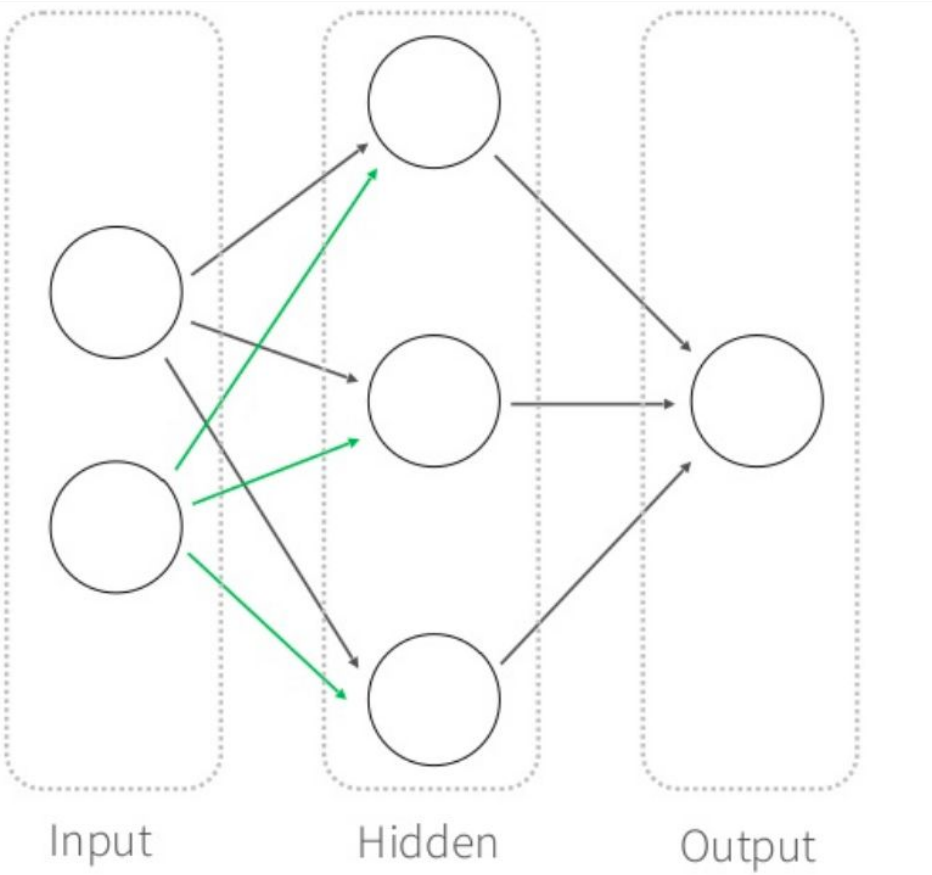


Instead of [0, 1], now (0…1)

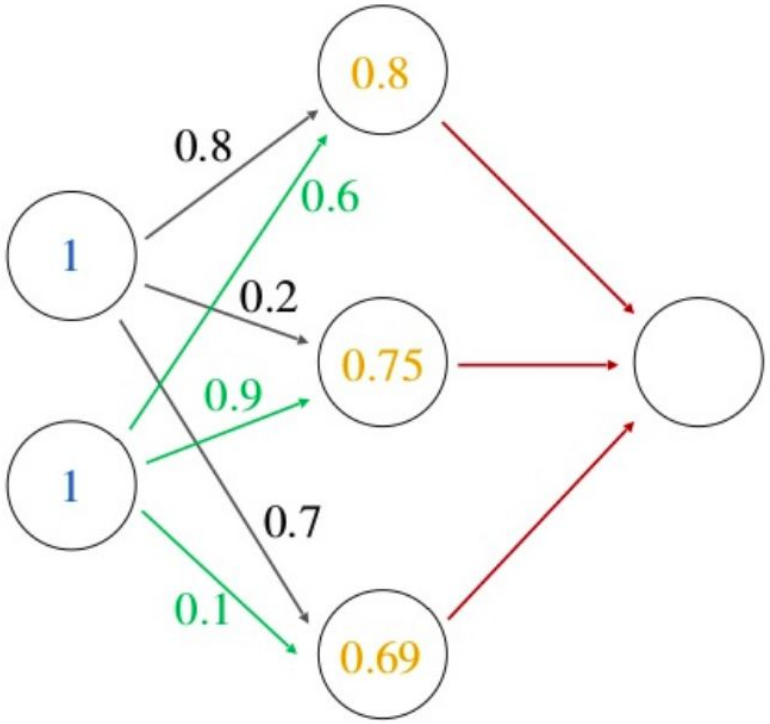
Where output is defined by $\sigma(wx + b)$

Adding the bias (b) is important in controlling the neuron. Also, its **value allows us to shift the activation function to the left or right**, which may be critical for successful learning.

EXAMPLE

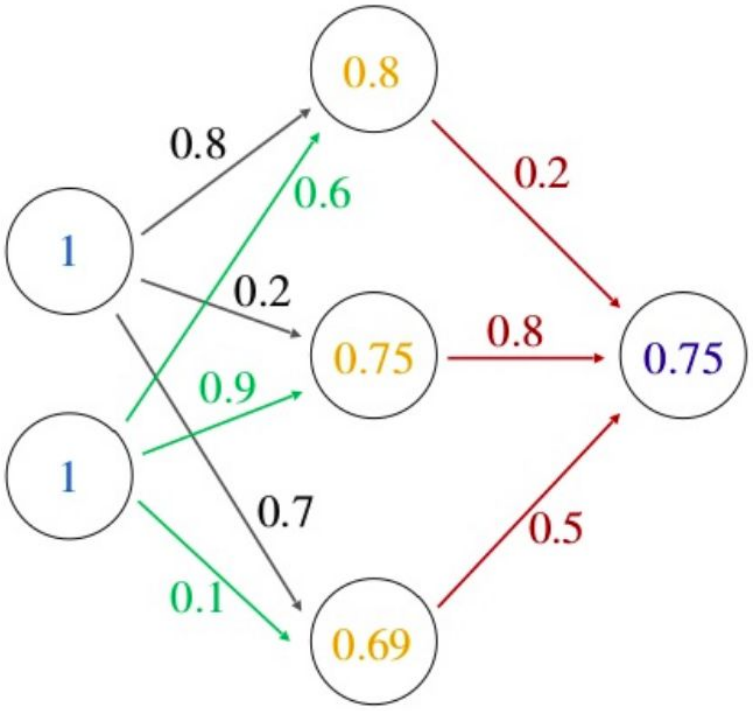


EXAMPLE



$$h_1 = \sigma(1x0.8 + 1x0.6) = 0.80$$
$$h_2 = \sigma(1x0.2 + 1x0.9) = 0.75$$
$$h_3 = \sigma(1x0.7 + 1x0.1) = 0.69$$

EXAMPLE



$$out = \sigma(0.2x0.8 + 0.8x0.75 + 0.5x0.69)$$

$$= \sigma(1.105)$$

$$= 0.75$$

COMPUTER OPERATION TO PERFORM FFNN

$$X = \begin{bmatrix} x1 \\ x2 \end{bmatrix} \text{ vector (input layer)}$$

$$W = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \text{ matrix (the weights for hidden layer 1)}$$

the output is given by

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \cdot \begin{bmatrix} x1 \\ x2 \end{bmatrix} \text{ (the product of vector and matrices)}$$

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} = \begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} + bias$$

finally,

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} = f\left(\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix}\right) \text{ (activation step)}$$

CODE EXAMPLE

In [8]:

```
import numpy as np
def sigmoid(x):
    #applying the sigmoid function
    return 1 / (1 + np.exp(-x))

input_vector = np.array([1, 2])
w = np.random.randn(3, 2)
b = 1
h = w.dot(input_vector) + b
h = sigmoid(h)
w2 = np.random.randn(3, 1)
y = np.dot(w2.transpose(), h)
y = sigmoid(y)
print(y)
[1 if y > .5 else 0]
```

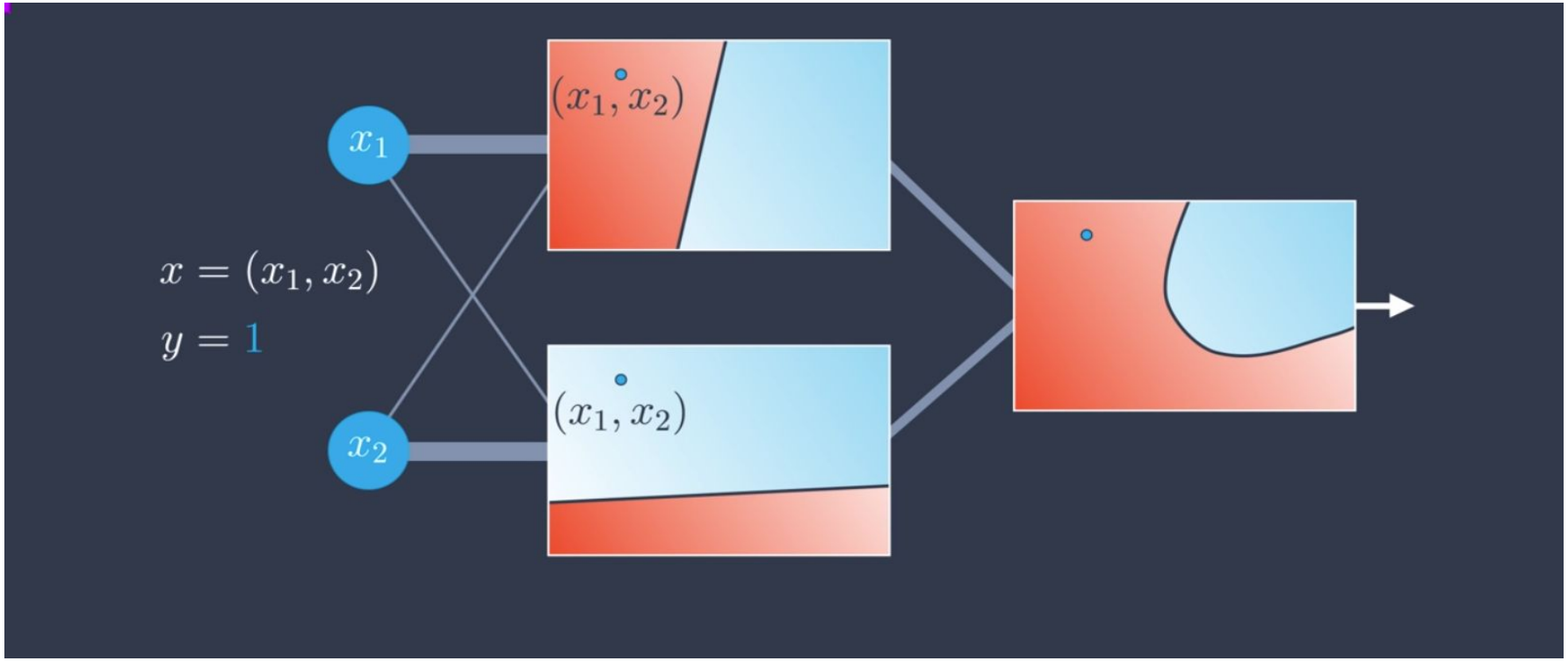
```
[0.7964323]
```

OPTIMIZATION

ERRORS

All of the previous process is called **Feedforward** which indicates the process of feeding our Neural Network with the data and plugin that into the sigmoid function to get a probability for the output between 0 and 1

Q: What if we misclassified our points?



FORMULATE A PROBLEM

The network needs to make predictions as close as possible to the real values. To measure this, we use a metric of how wrong the predictions are, the **error** to improve the neural network **weights**

Mean Square Error Function:

$$E = \frac{1}{2}\sum_{\mu}\sum_{j}\left[y_j^{\mu} - \hat{y}_j^{\mu}\right]^2$$

Where
1- *y_hat* represents the prediction and *y* is the true value.
2- The variable ***j*** represents the output units of the network. So this inside sum is saying for each output unit, find the difference between the true value *y* and the predicted value from the network then square the difference, then sum up all those squares.
3- The variable ***μ*** indicates loop over all of the data point that we feed our neural network with.

To reach our gool, we need to minimize the error and we will user a method called backpropagation to do that.

BACKPROPAGATION

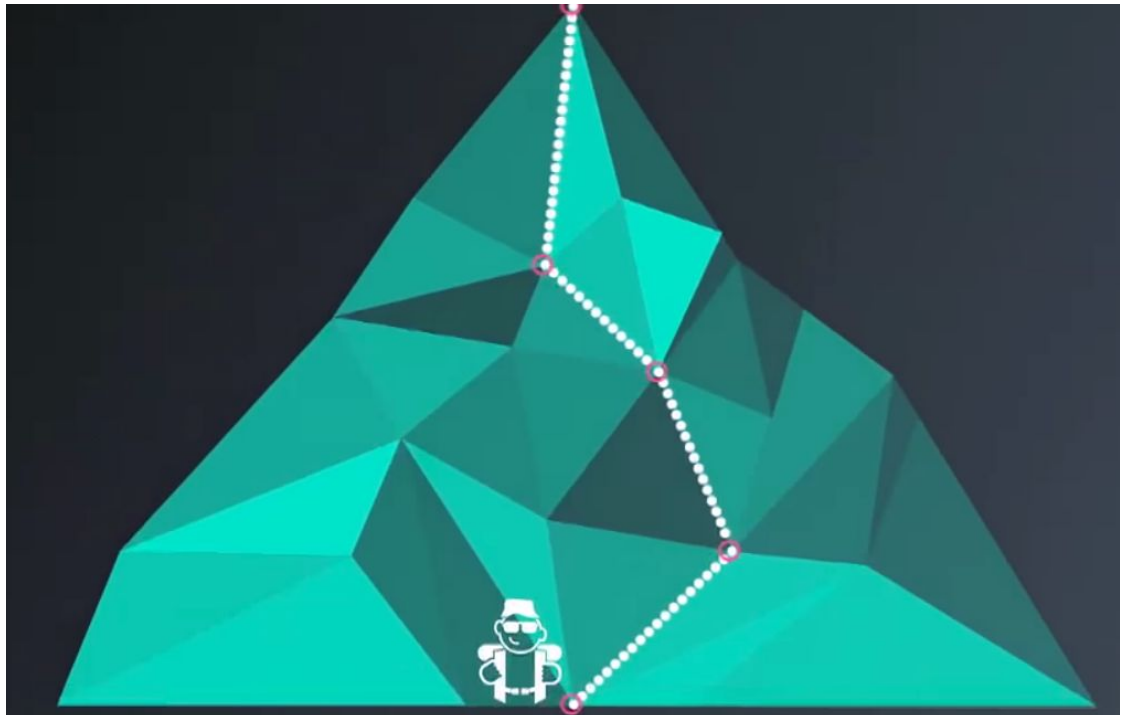
Big picture steps:

- Do a feedforward operation.
- Comparing the output of the model with the desired output.
- Calculating the error.
- Running the feedforward operation backwards (backpropagation) to spread the error to each of the weights.
- Use this to update the weights, and get a better model.
- Continue this until we have a model that is good.

BACKPROPAGATION

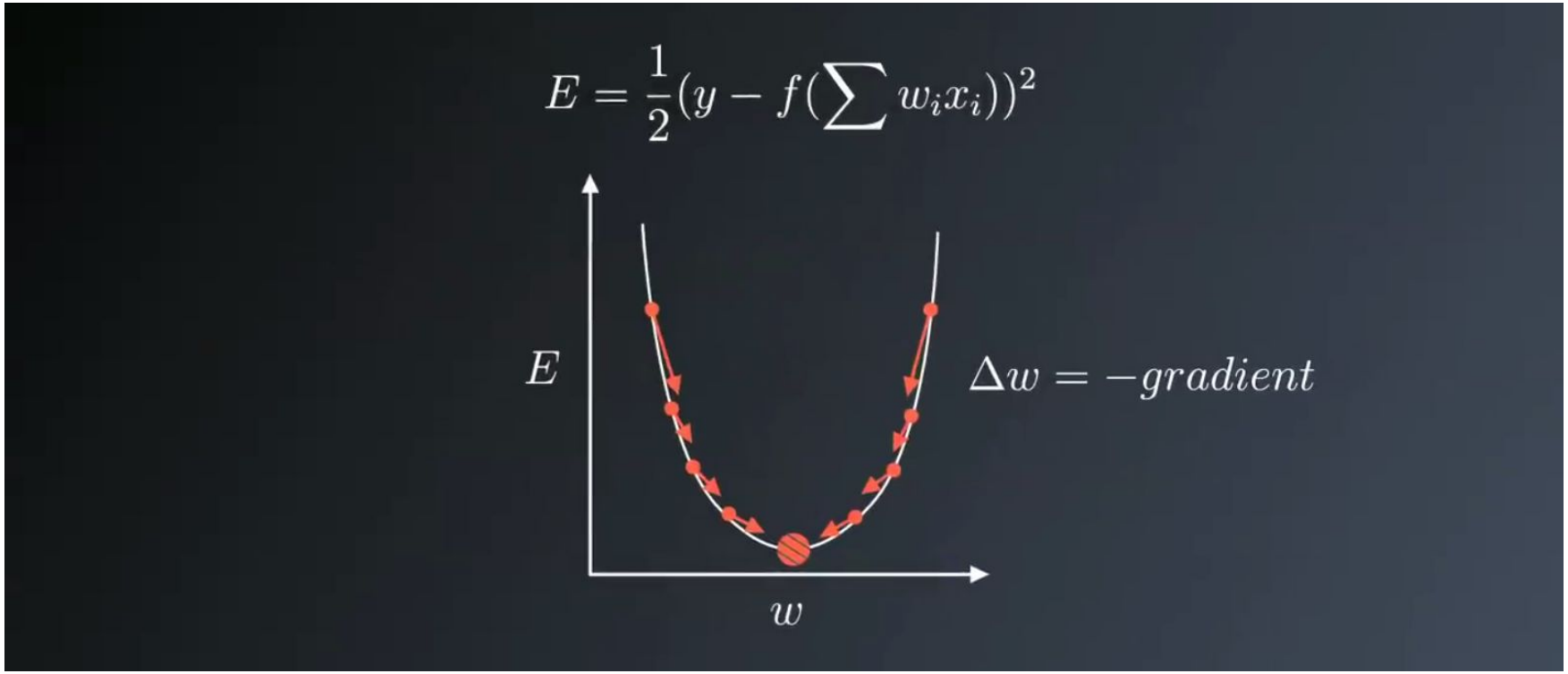
Backpropagation is the optimization of the neural network and it is simply optimizing the error function to get its minimum.

Thinking of that mountain as out Error function plotted, then from the top of it, reaching the bottom of the mountain (the steeps point of the mountinue) is our goal and the way to do that is by looking around us each time and finding the next steeps point (min error). So, we can find the best direction by calculating the *gradient* of the squared error function



BACKPROPAGATION WITH GRADIENT DESCENT

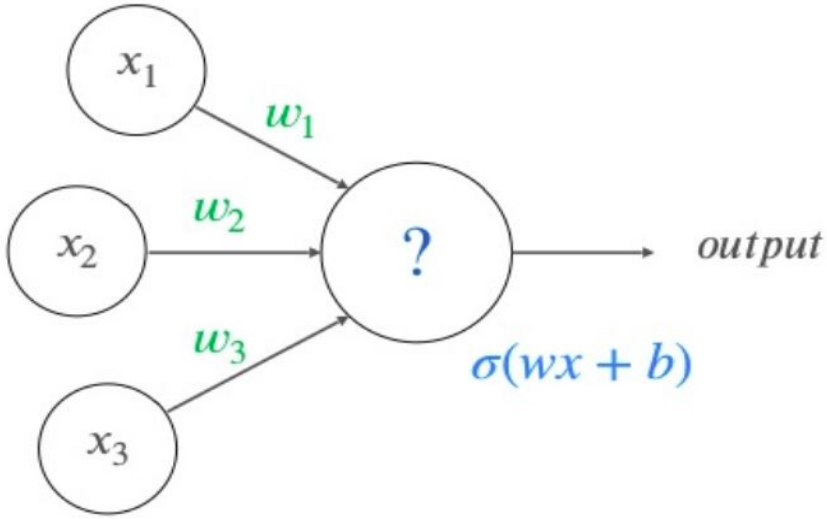
Gradient descent is an optimization algorithm that minimizes the error of each weight in our network to improve the results.



As shown, the error function is a function of each weight, so we need to get the partial derivative of the error function corresponding to each weight to reach the minimum of the error at that weight.

CHAIN RULE

Update the value of w1



Our error function is a function of y_output, and our y_output is a function of h, and our h is a function of the weight. So, to get the partial derivative of the error function corresponding to the weight, we need to use the chain rule.

CHAIN RULE APPLIED

$$\frac{\partial E(y_h)}{\partial W1} = \frac{\partial E(y_h)}{\partial y_h(h)} * \frac{\partial y_h(h)}{\partial h(w)} * \frac{\partial h(w)}{\partial W}$$

$$1 - E = \frac{1}{2}(y - y_h)^2$$

$$2 - \frac{\partial E(y_h)}{\partial y_h(h)} = -(y - y_h)$$

$$3 - y_h = \sigma(h(w))$$

$$4 - \frac{\partial y_h}{\partial h} = \sigma(h(w)) * (1 - \sigma(h(w)))$$

$$5 - h(w) = w_1x_1 + w_2x_2 + w_3x_3$$

$$6 - \frac{\partial h(w)}{\partial w_1} = x_1$$

Note

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

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

$$\sigma(x)' = -1(1 + e^{-x})^{-2}(-e^{-x})$$

$$\sigma(x)' = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})} \cdot \frac{e^{-x}}{(1+e^{-x})}$$

$$\sigma(x)' = \frac{1}{(1+e^{-x})} \cdot \frac{(1+e^{-x})-1}{(1+e^{-x})}$$

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

$$\frac{\partial E(y_h)}{\partial W1} = -x_1(y - y_h)\sigma(w_1x_1 + w_2x_2 + w_3x_3)(1 - \sigma(w_1x_1 + w_2x_2 + w_3x_3))$$

UPDATING THE WEIGHTS

$$w_i = w_i + \Delta w_i$$

$$\Delta w_i \propto -\frac{\partial E}{\partial w_i} \longrightarrow \text{THE GRADIENT}$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

LEARNING RATE

So, we update each weight with the negative value of the gradient which points towards the minimum multiplied by the learning rate (a length) that determines the step we take each time we update to reach the minimum.

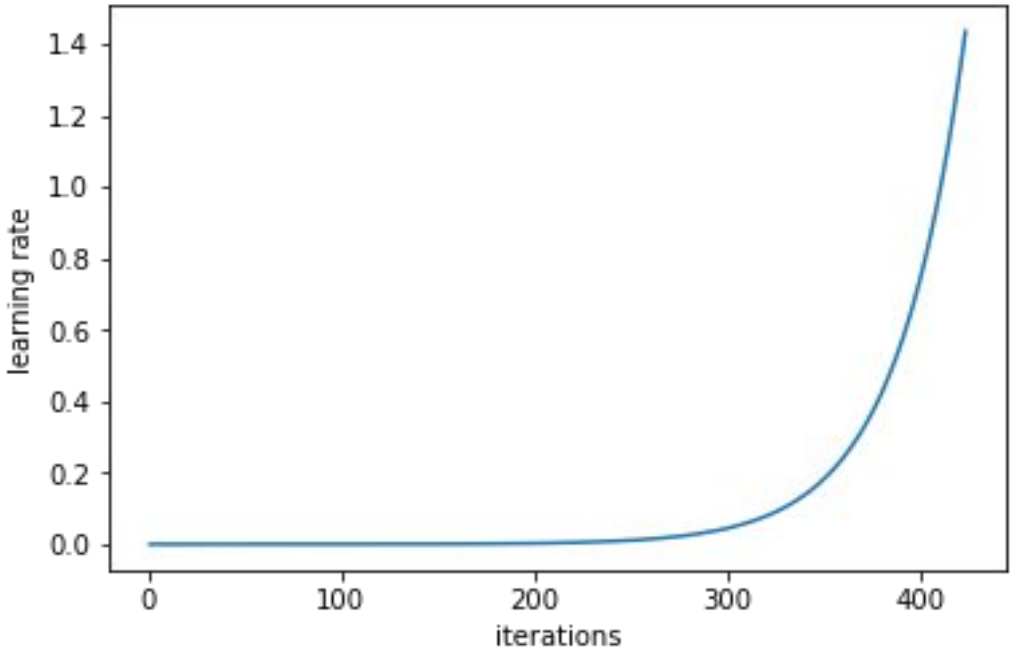
The value of **learning rate** shouldn't be: 1- too big because we won't saturate in the global or local minimum

2- too small because it will make the process of reaching the minimum (converge) slow.

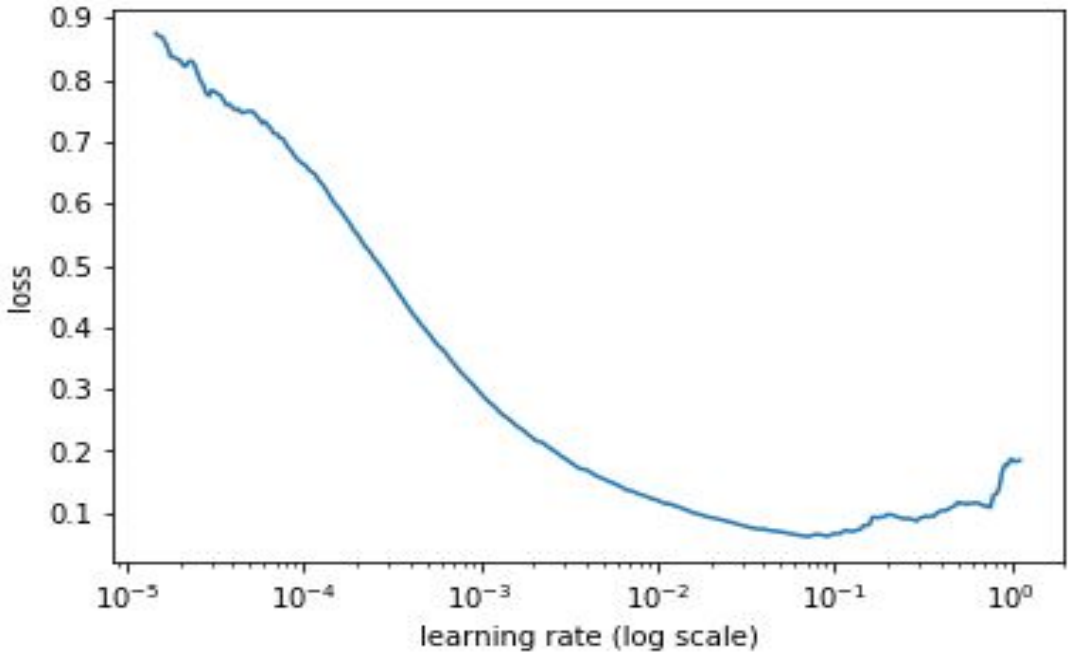
GOOD WAY TO CALC LEARNING RATE

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient.

A way of calculating a good learning rate is to choose very low learning rate and increasing it (either linearly or exponentially) at each iteration of the traning. Then record the learning rate at each iteration and plot the learning rate (log scale) against the loss function and we can see that point where our loss stopped to decrease and started to increase. This point indicates optimal learning rate (in the example below the good learning rate is between 0.001 to 0.01)

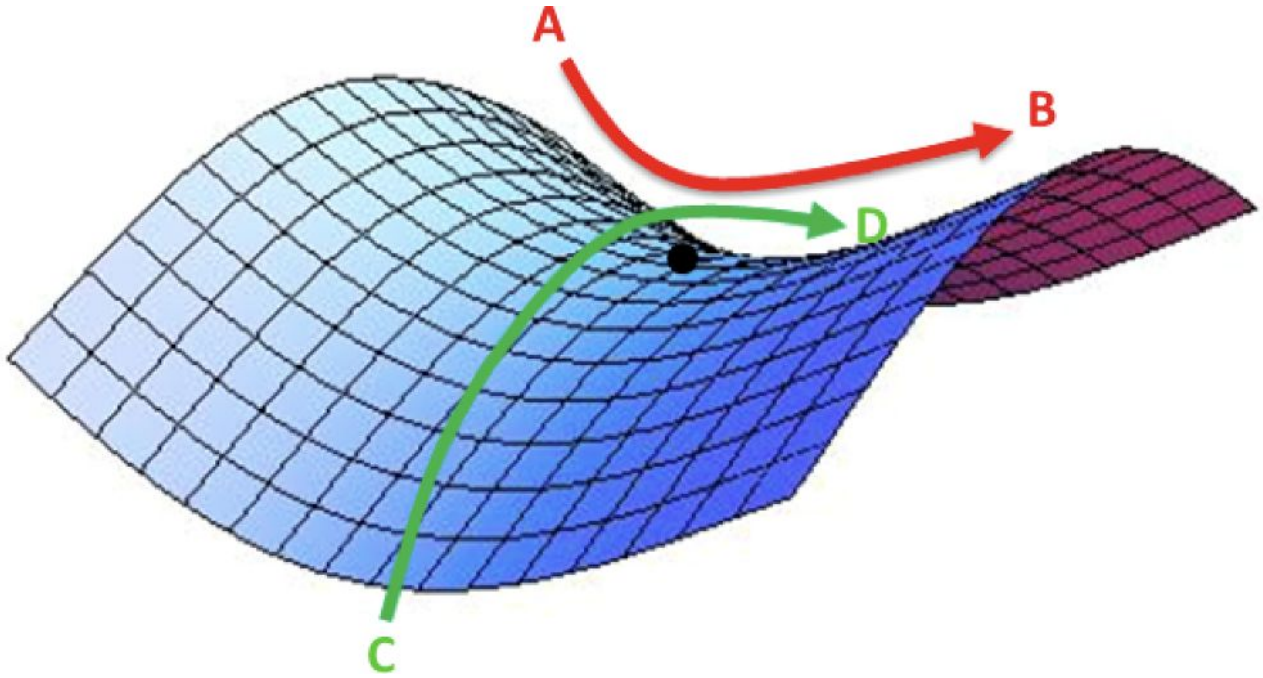


Learning rate increases after each mini-batch



LIMITATION OF CALCULUS

Sometimes, minimizing the loss becomes hard if we reach a saddle point like the one indicated in the graph because the gradient at this point is 0 which make is the effect of learning rate disappears and the weights values will be fixed.



For that, we can use another technique called **stochastic gradient descent**. The SGD simply trains cycles, during each cycle, it chooses different weights and restarts the learning rate and does the same process as GD. By that, we can plot the error at different weights which helps to choose the actual global or local minimum in a shorter period of time (optimization).

LIMITATION OF CALCULUS

The main **objective** of training a neural network is have a final model that performs well both on the data that we used to train it and the new data on which the model will be used to make predictions (**good fit model**). So, we split our data to train/split data and calculate the error on both data.

Another scenario that might happen is over-optimizing the weights which is called **overfitting** the training date. Overfitting means that the error on the training set is driven to a very small value, but when new data is presented to the network the error is large. Also, the model can **underfit** which means it can't learn.

Root of the problem:

1- The model has too little capacity so it won't learn the problem (**underfitting**).

2- The model has too much capacity so it can learn the problem too well and overfit the training dataset (**overfitting**).

Both cases will result in a model that does not generalize well on the test examples.

Note: Capacity refers to the ability of a model to fit a variety of functions for mapping the input to output..

SOLUTION FOR UNDER AND OVER FITTING

We can address **underfitting** by increasing the capacity of the model. THis can be done via changing the structure of the model, such as adding more layers and/or more nodes to layers.

For **overfitting,** we can:

- Increase the number of training data.
- changing the capacity of the network by changing the number of weights and hidden layers or changing the values of the weights (**SGD**).
- **Regorluzation** and one of its forms is keeping the weights values small to not have any bias via penalize the model during training based on the magnitude of the weights (**weight decay**).

CODE

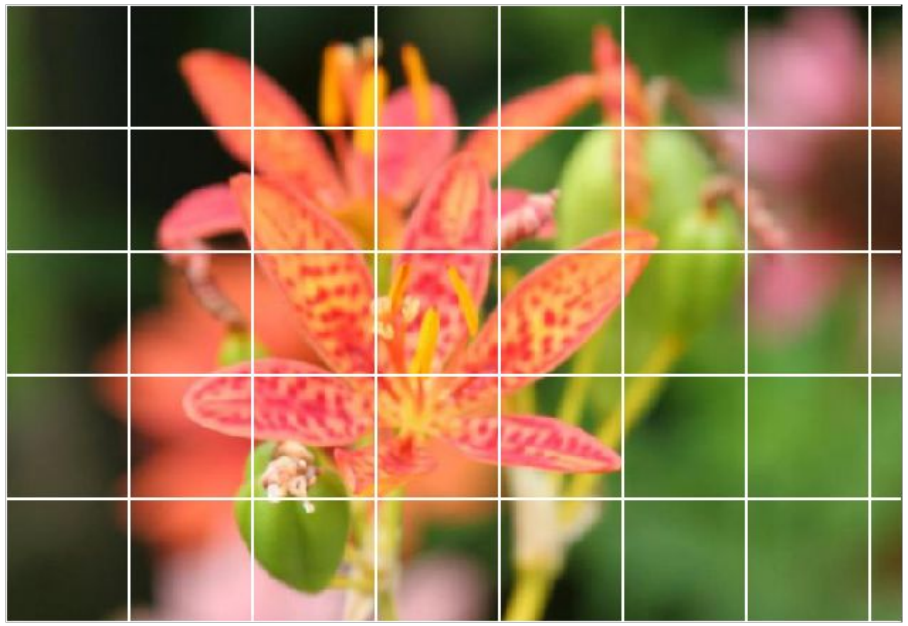
Click on github icon below for the code

APPLICATION

Image CLASSIFIER

Let's train deep learning model on images training set and use it to classify new images.

Problem: given a flower picture like the one down, identify its type among 4 different types of flowers.



CODE

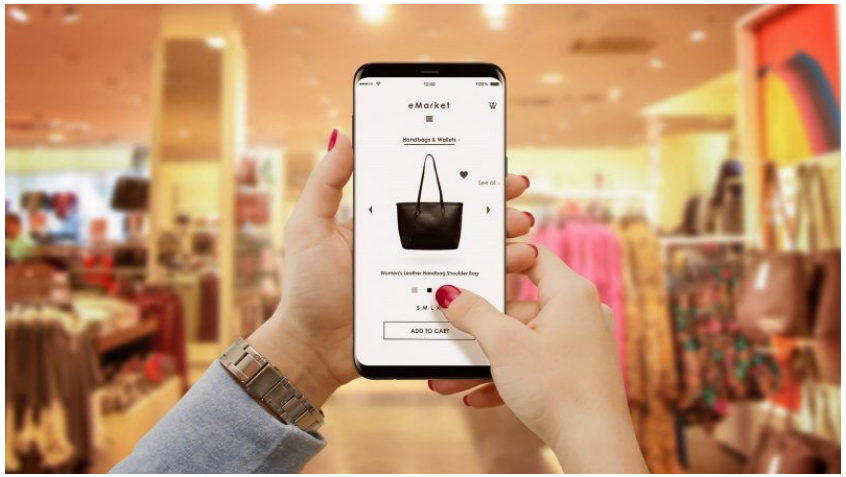
The project with its explanation are in the following github account.

Click here

IMAGE CLASSIFIER APPLICATIONS

Automated Image grouping

This is a Deep Learning application that classifies similar images together in the cloud.



Visual Search for Improved Product Discoverability

Built at the top of image grouping and used in the online stores to show similar products the ones you like.

THANK YOU!!!

ANY QUESTIONS?