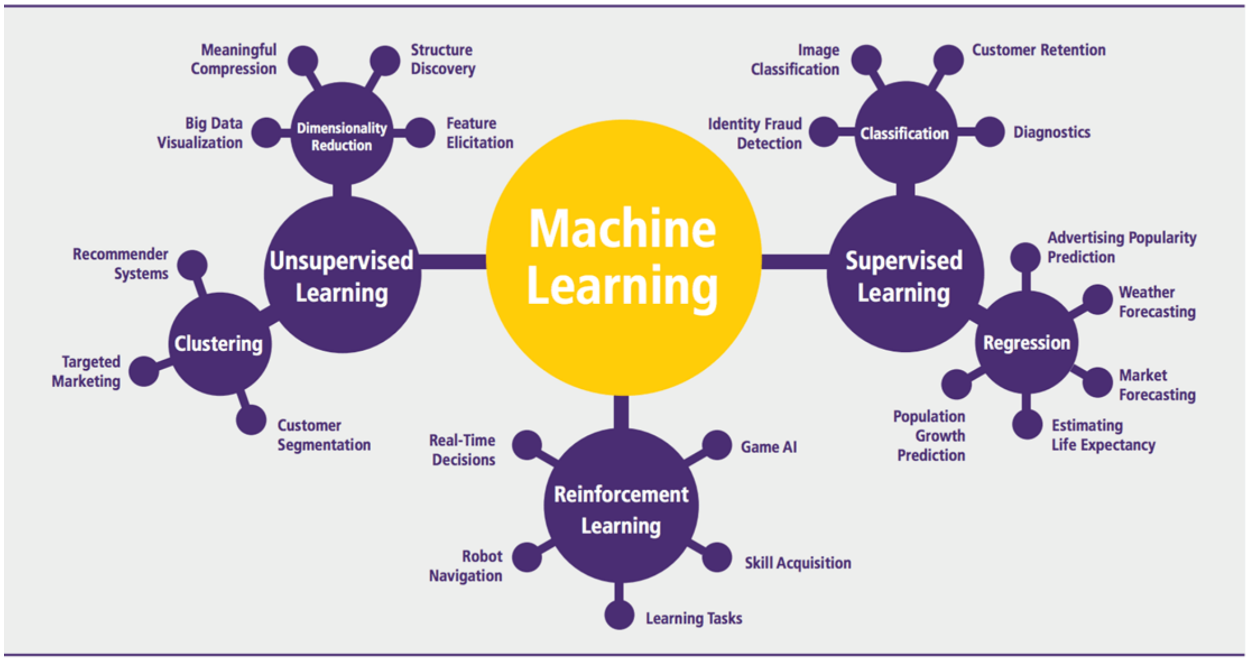
DEEP LEARNING REPORT 1

Nour Ahmed Aboelmagd
20176033

Deep Learning Report

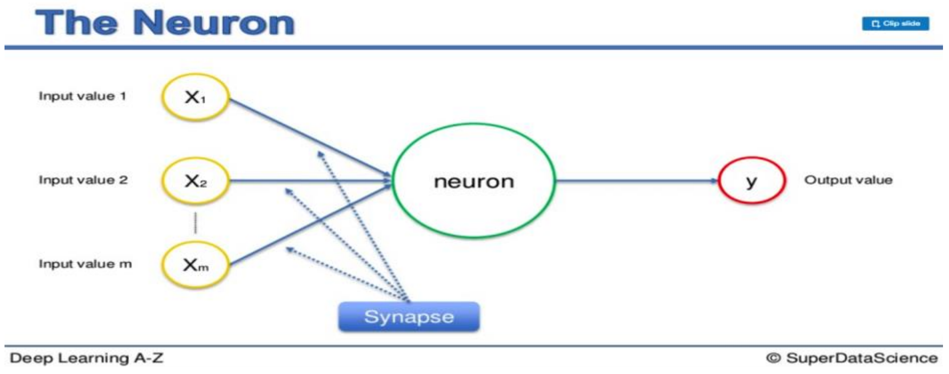


Week 1:

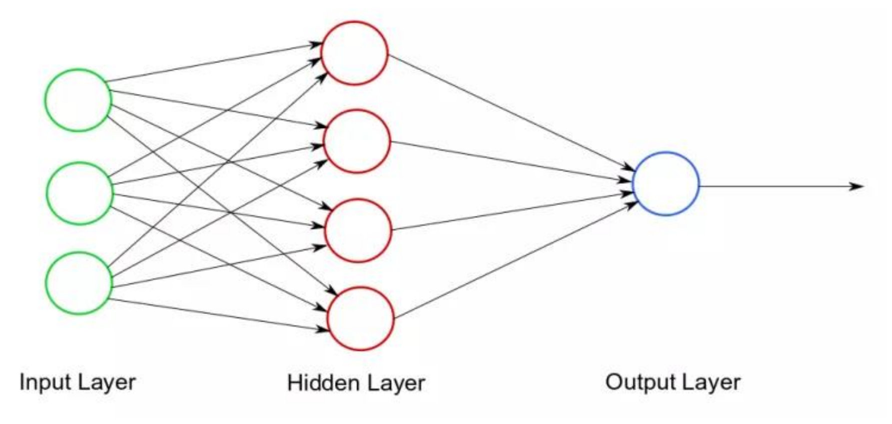
Neural networks:

It consists of a neuron or multiple neurons with inputs and outputs and the size of it depends on the number of neurons

simple neural network: consists of one neuron



complex neural network: consists of multiple neurons



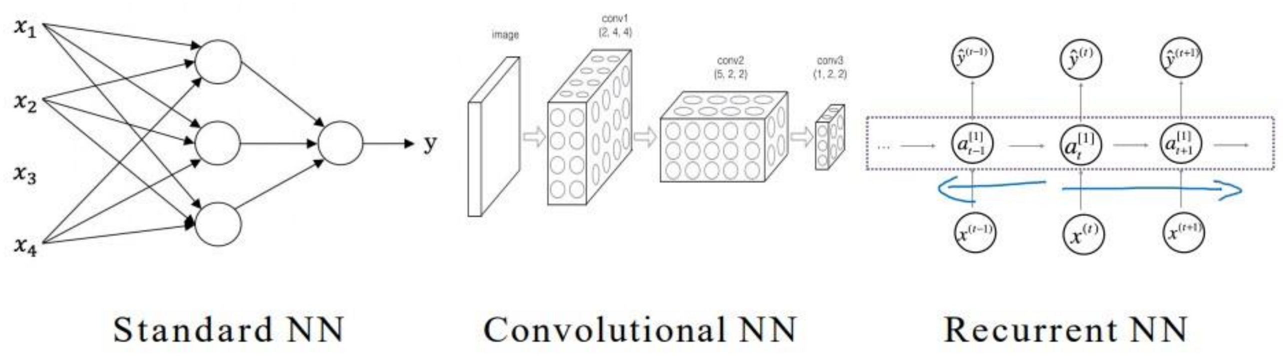
A neural network consists of:

- Input features
- Training sets
- Test sets
- Output

Supervised learning:

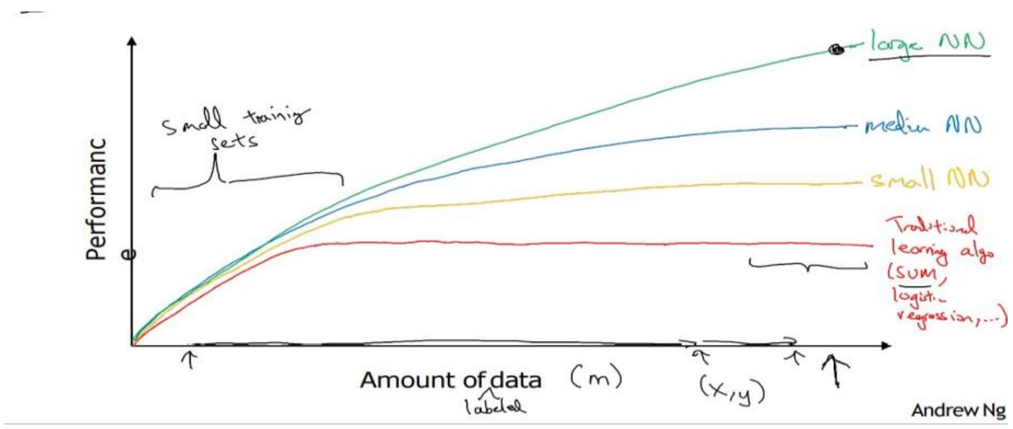
making the network learn and handle a new function by dealing with previous experiences

Types of neural networks:



Types of Data:

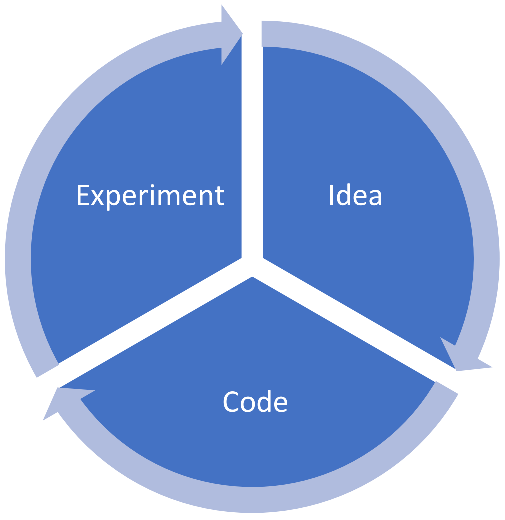
- Structured: databases
- Unstructured: photos – audio – text



With the continuous increase in data acquired, increase in neural network size and training sets is mandatory to adapt

Also, using more data leads to faster computation process

Cycle of deep learning:

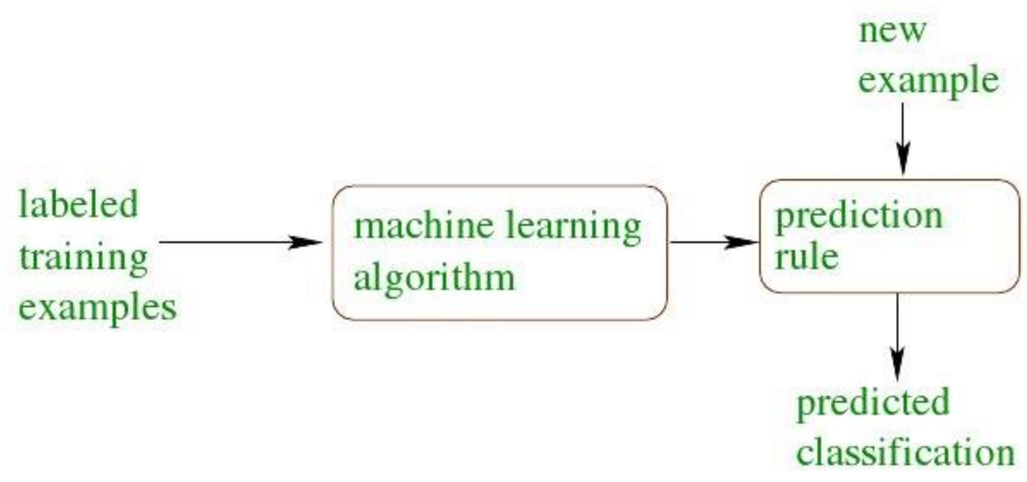


Week 2:

Supervised learning is divided into:

- Regression (one output either zero or one)
- Classification (multiple outputs divided into classes)

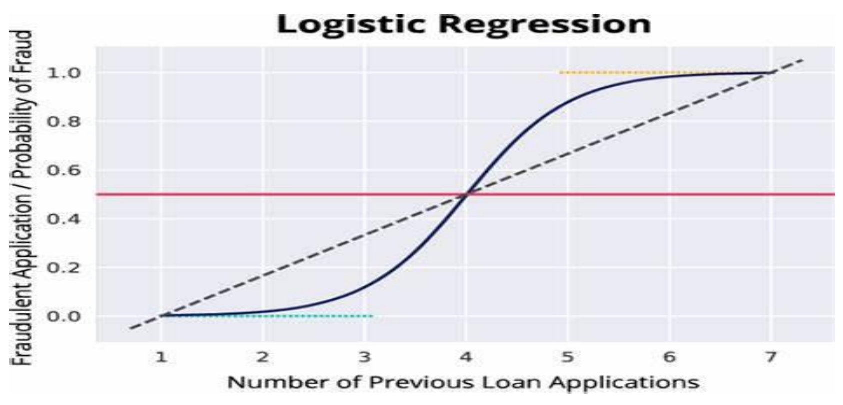
Binary classification:



RGB matrcies are unrowed into vector (x) which are the dimensions of the input features, normally it is equal to (64*64*3=12288)

Logistic Regression:

Where the output is either Zero or one and we estimate the result by using the activation function (Sigmoid)



Loss function:

Computes the error in one training example

LogLoss for binary classification

- $L = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i))$

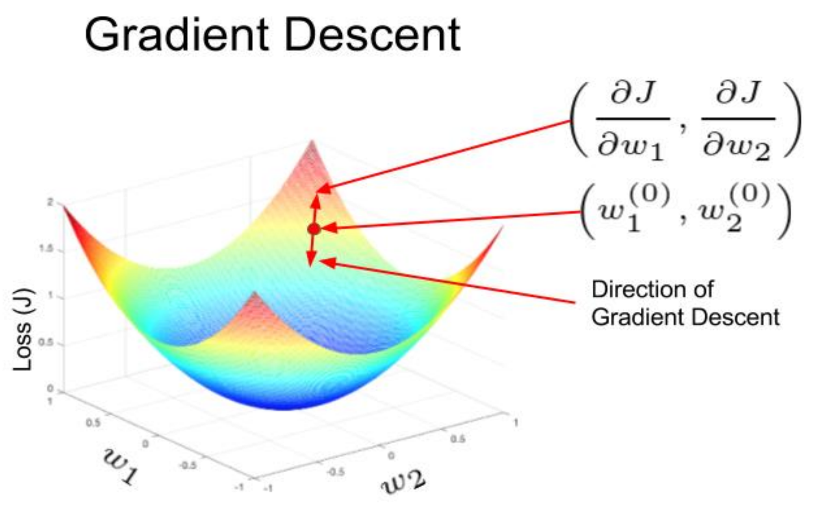
Cost function:

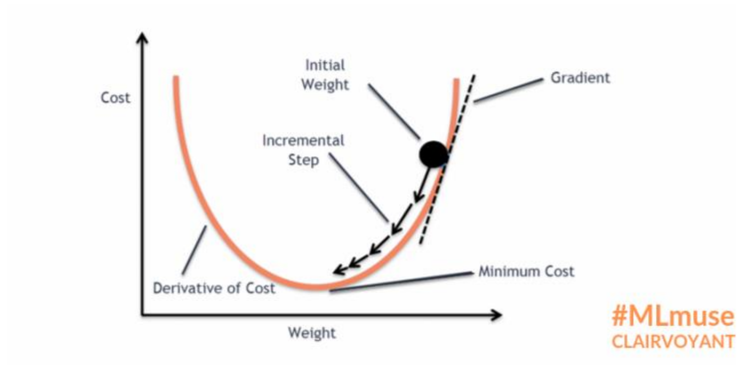
Computes the average of error in the entire training set

$$\underline{J(w,b)} = \frac{1}{m}\sum_{i=1}^{m}\underline{\mathcal{L}(\boxed{\hat{y}^{(i)}},\boxed{y^{(i)}})} = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log\hat{y}^{(i)} + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})$$

Gradient Descent:

In order to compute the parameters (w,b) , gradient descent with the goal in mind is to minimize the value of the cost function as possible



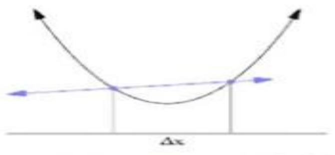


-As b is assumed to be zero, the graph turns into a 2D one

-First w is initialized as a certain value, then it is updated until the global optimum is found

-The update of (w) is performed by the learning rate changing

$$\theta := \theta - \alpha \frac{d}{d\theta} J(\theta)$$



θ: parameters

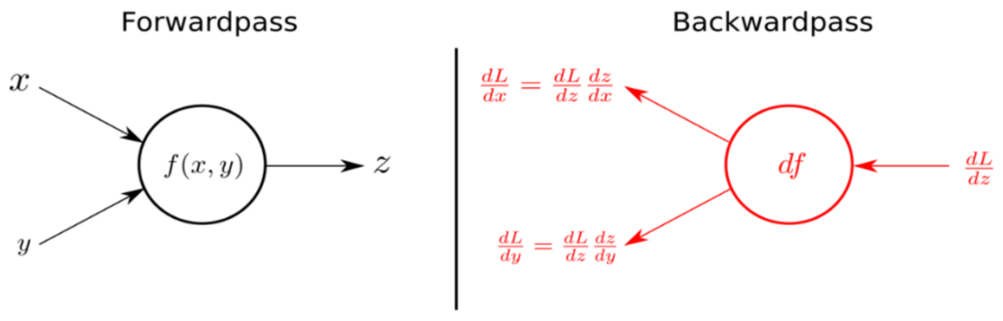
α: learning rate

$\frac{d}{d\theta} J(\theta)$: derivative of the cost function $J(\theta)$

(direction to update)

Backward propagation in computation graph:

In order to update the parameters (w, b), we have to deduce the derivative by backward propagation


Forwardpass
x
y
$f(x,y)$
z
Backwardpass
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$
df
$\frac{dL}{dz}$

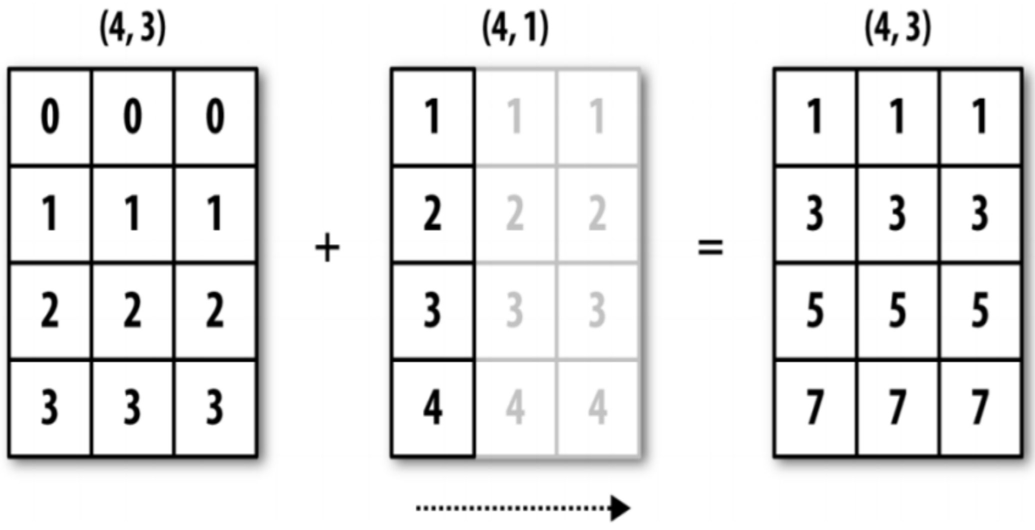
Vectorization:

Vectorization is a method in deep learning implemented to nearly eliminate the use of the for loop for faster computation by replacing it with one line of code

For example: the training examples (x1,x2,…, xm) are all accumulated into one vector training set X

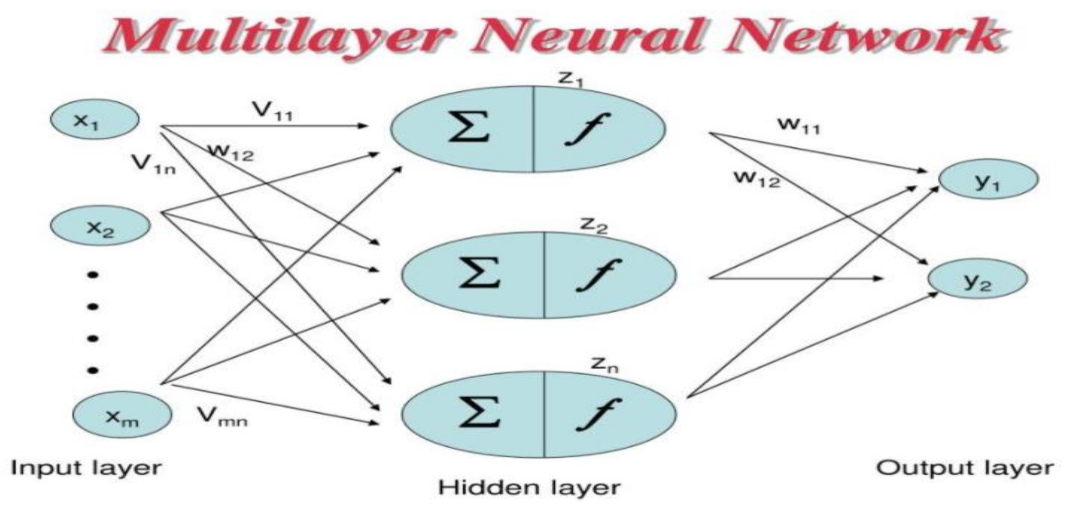
Broadcasting:

When matricies are multiplied it was supposed that the number of columns in first matrix is equal to the number of columns in the second matrix but python solves this problem with the condition that either the number of rows or columns is the same in both matricies



Week 3:

Neural Network presentation:



Neural network is divided into multiple layers which are:

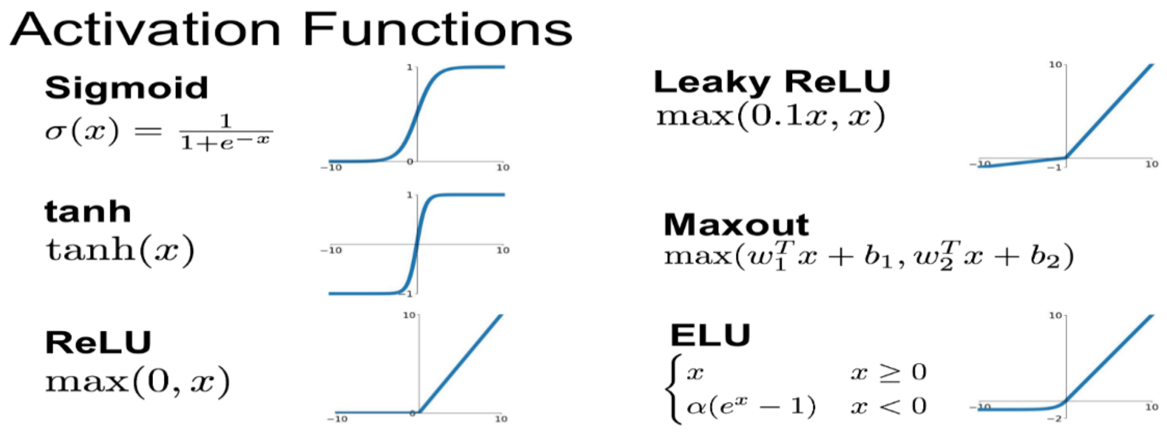
1. Input layer
2. Hidden layers
3. Output layer

- Input layers has the input features each one is connected to every neuron in the hidden layer next
- In the hidden layers, each neuron computes a linear function followed by the activation function
- Output layer computes the estimation of the output

vectorization:

each layer has its content vectorized like the weights of first hidden layer (W1)

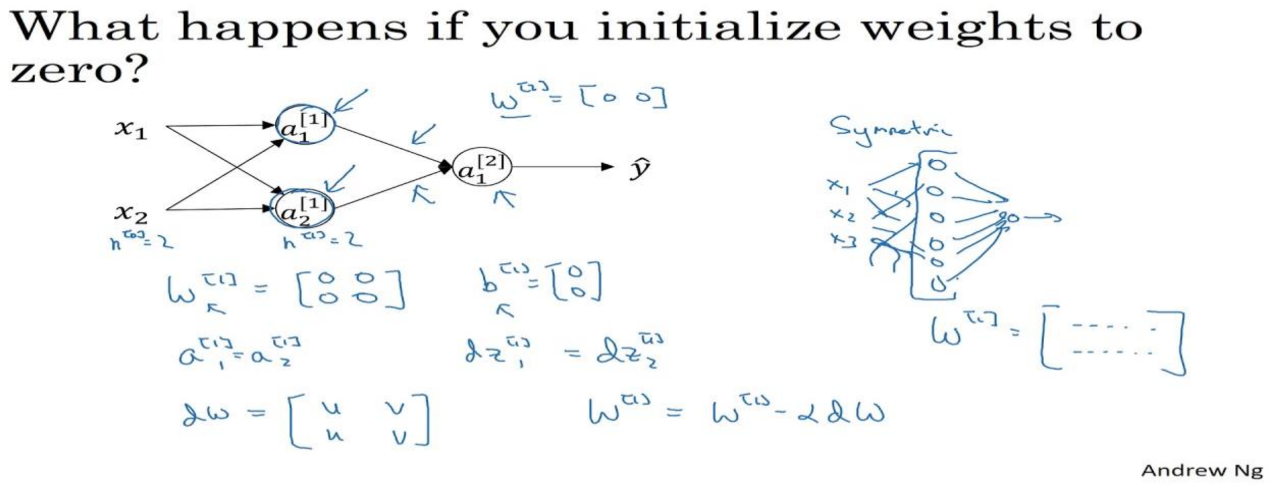
Types of activation functions:



Non linear activation functions such as tanh has a great advantage in that the functions vary from (-1,1) by making zero the mean which enables easier learning experience where if linear function is used, the output will be the linear function of input which is basically nothing

But in output layer the sigmoid can be used because there is only one output which makes the learning part already limited

Random initialization:


What happens if you initialize weights to zero?
x_1
x_2
$a_1^{[1]}$
$a_2^{[1]}$
$a_1^{[2]}$
$\hat{y}$
$n^{[0]} = 2$
$n^{[1]} = 2$
$W^{[2]} = [0 \; 0]$
$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
$b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
$a_1^{[1]} = a_2^{[1]}$
$dz_1^{[1]} = dz_2^{[1]}$
$dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$
$W^{[1]} = W^{[1]} - \alpha dW$
Symmetric
x_1
x_2
x_3
$W^{[1]}$
Andrew Ng

The initialization of bias (b) is fine to be zeroes but in weights its different

If weights are initialized by zero, each neuron in the hidden layer will compute the same and after iterations of gradient descent and updating it will still be the same

This problem is called (break symmetry) and it stalls the learning process without an end

So, weights must be initialized with random variables

But must be small values for the gradient descent to be small to continue learning in the area where the slopes vary for better learning and faster optimization