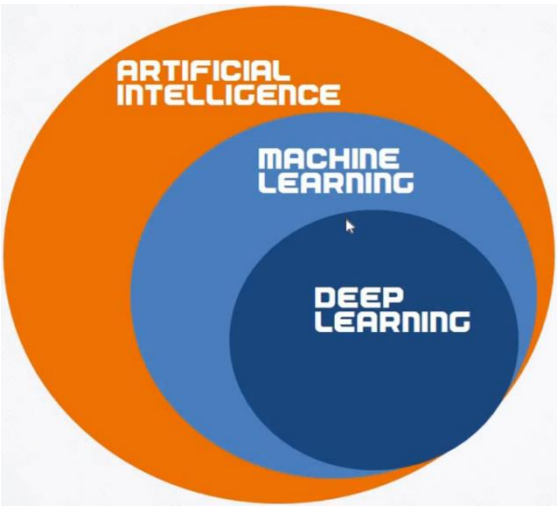
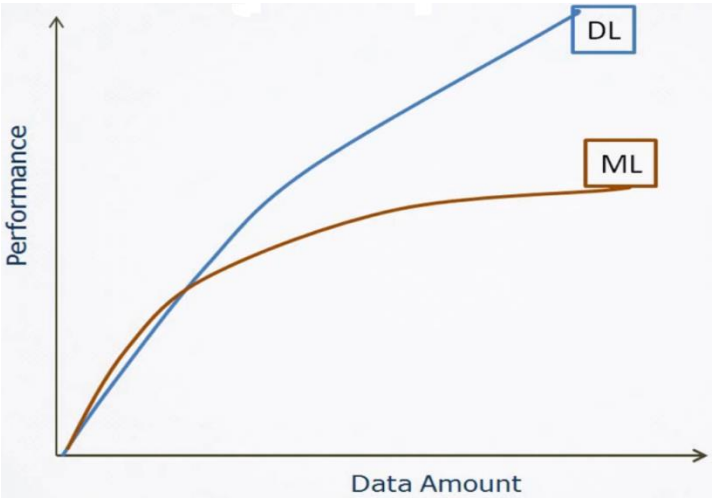
Deep Learning



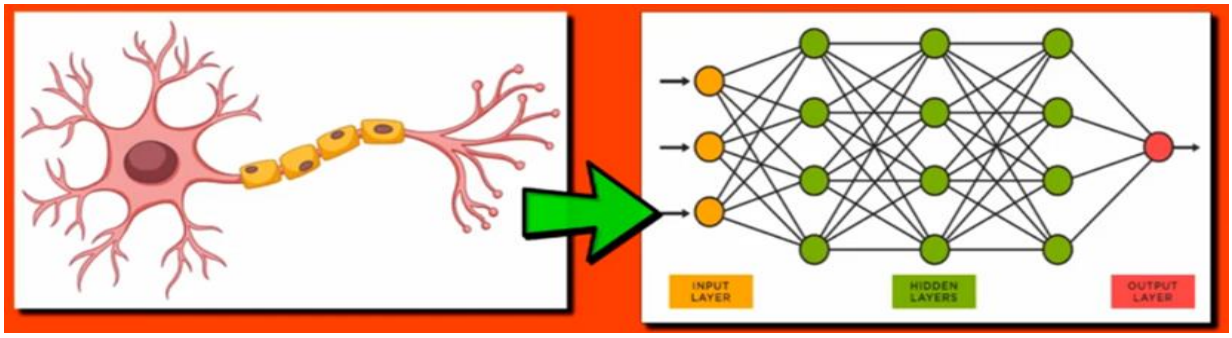


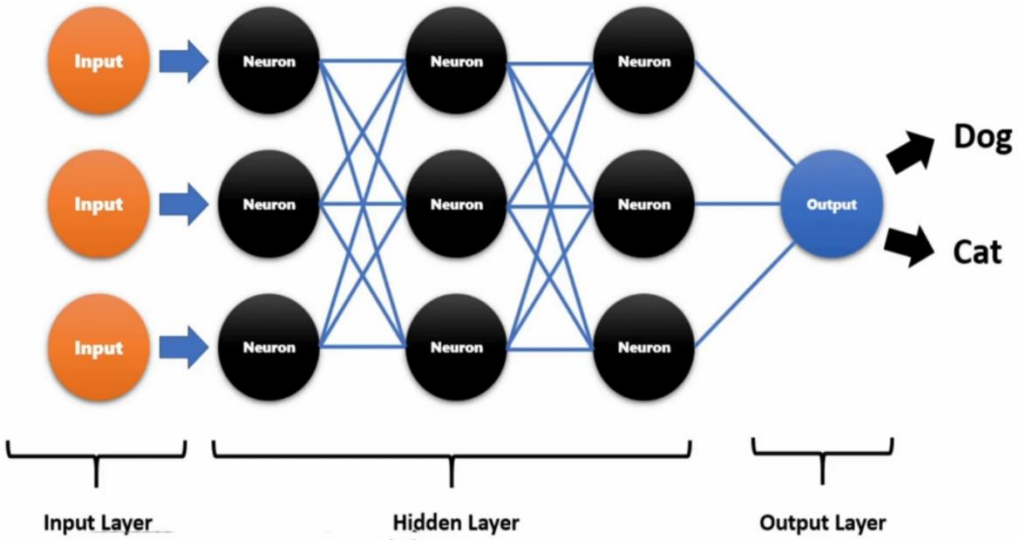
Deep learning is a computer software that **mimics the network of neurons in a brain**. It is a subset of machine learning and is called deep learning because it makes use of deep **neural networks.**

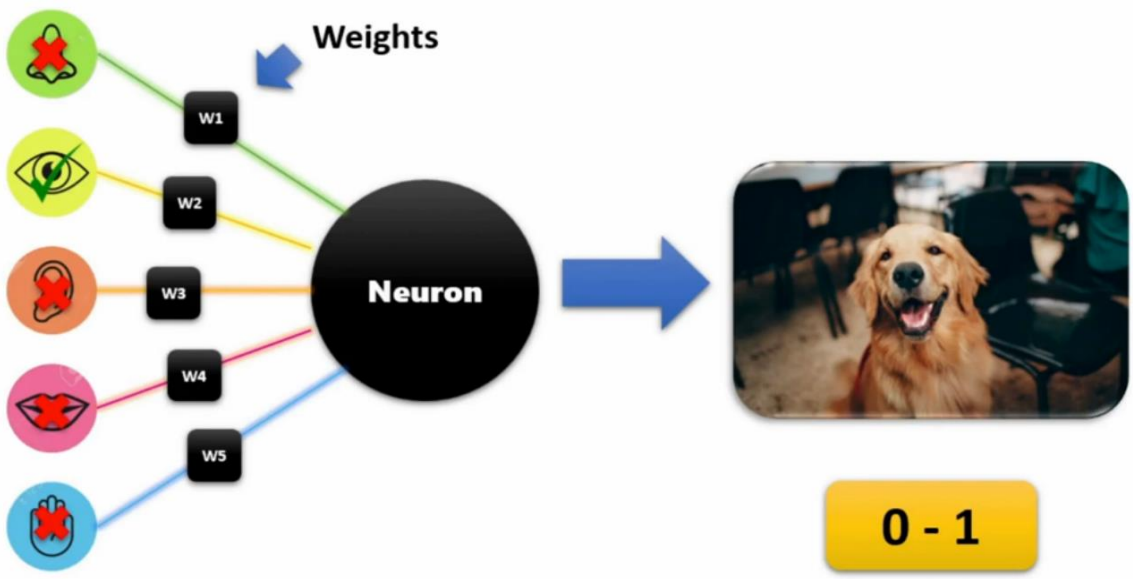
Deep Learning Model Process

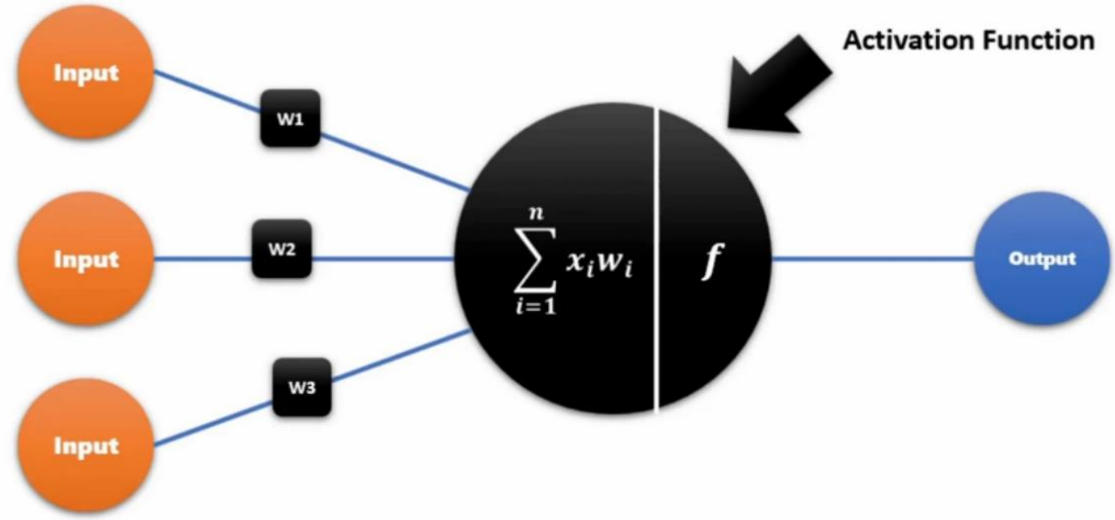
Deep learning algorithms are constructed with connected layers.

- ✓ The first layer - Input Layer
- ✓ The last layer - Output Layer
- ✓ All layers in between are called Hidden Layers.









Formula → $Activation\ Function(\sum_{i=1}^{n} x_i w_i + b)$

Types → **Linear and Non - Linear**

1. Load data
2. define keras models
3. compile
4. fit / train
5. evaluate

```
#(1)Load data
import numpy as np

dataset = np.loadtxt('/content/pima-indians-diabetes.data.csv',delimiter=',')
X = dataset[:,0:8]
y = dataset[:,8]
```

```
dataset
```

```
array([[  6.   , 148.   ,  72.   , ...,   0.627,  50.   ,   1.   ],
       [  1.   ,  85.   ,  66.   , ...,   0.351,  31.   ,   0.   ],
       [  8.   , 183.   ,  64.   , ...,   0.672,  32.   ,   1.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,   0.245,  30.   ,   0.   ],
       [  1.   , 126.   ,  60.   , ...,   0.349,  47.   ,   1.   ],
       [  1.   ,  93.   ,  70.   , ...,   0.315,  23.   ,   0.   ]])
```

```
X
```

```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
```

```
y
```

```
array([1., 0., 1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1.,
       1., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0.,
       1., 1., 0., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0., 0., 1.,
...
       0., 0., 0., 0., 1., 1., 0., 0., 0., 1., 0., 1., 1., 0., 0., 1., 0.,
        1., 1., 1., 0., 0., 1., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 0.,
        0., 1., 0.])
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
#(2)Define keras models
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(12,input_dim=8,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

```
#(3)Compile
model.compile(loss='binary_crossentropy')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12)                108

 dense_1 (Dense)             (None, 8)                 104

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 221
Trainable params: 221
Non-trainable params: 0
```

```
#(4)Fit/Train
model.fit(X,y,epochs=150,batch_size=10)
```

```
Epoch 1/150
77/77 [==============================] - 1s 2ms/step - loss: 4.5729
Epoch 2/150
77/77 [==============================] - 0s 2ms/step - loss: 1.1170
77/77 [==============================] - 0s 2ms/step - loss: 0.7077
Epoch 13/150
...
Epoch 149/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5202
Epoch 150/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5245
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
#(5)Evaluate
model.evaluate(X,y)
```

```
24/24 [==============================] - 0s 1ms/step - loss: 0.4989

0.49893298745155334
```

```
[2] import tensorflow as tf
    import numpy as np
    from tensorflow import keras
```

```
[3] print(tf.__version__)

    2.6.0
```

Data Set

```
xs = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)
```

Simple Neural Network

```
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
```

Compiling

```
[6] model.compile(optimizer='sgd', loss='mean_squared_error')
```

Training the Neural Network

```
model.fit(xs, ys, epochs=500)
```

```
Epoch 495/500
1/1 [==============================] - 0s 7ms/step - loss: 3.1498e-05
Epoch 496/500
1/1 [==============================] - 0s 6ms/step - loss: 3.0851e-05
Epoch 497/500
1/1 [==============================] - 0s 12ms/step - loss: 3.0217e-05
Epoch 498/500
1/1 [==============================] - 0s 8ms/step - loss: 2.9597e-05
Epoch 499/500
1/1 [==============================] - 0s 5ms/step - loss: 2.8990e-05
Epoch 500/500
1/1 [==============================] - 0s 5ms/step - loss: 2.8394e-05
<keras.callbacks.History at 0x7fda501f5890>
```

Prediction

```
[8] print(model.predict([5.0]))

    [[8.995719]]
```

Dot Product

Matrix A -> m × n matrix
Matrix B -> n × p matrix

Dot product of matrix A and matrix B = ?

(m × n).(n × p) = **m × p** matrix

where the n entries across a row of A are multiplied with the
n entries down a column of B and summed to generate an entry of resulting matrix.

Why we do Dot Product

The dot product greatly reduces the computation time especially
when we have a large number of independent equations to solve.

Note : number of columns in the first matrix should always be equal to the number of rows in the second matrix.

$$\begin{bmatrix} p1 & q1 & r1 \\ p2 & q2 & r2 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \\ X3 \end{bmatrix}$$

Dot Product Illustration

$$\begin{bmatrix} p1 & q1 & r1 \\ p2 & q2 & r2 \end{bmatrix} \bullet \begin{bmatrix} X1 \\ X2 \\ X3 \end{bmatrix} = \begin{bmatrix} p1xX1 + q1xX2 + r1xX3 \\ p2xX1 + q2xX2 + r2xX3 \end{bmatrix} = \begin{bmatrix} y1 \\ y2 \end{bmatrix}$$

Shape = (2x3).(3x1) = (2X1)

Broadcasting

Matrix A -> m × n matrix

Matrix B -> shape should be shape (1 x n) or (m X 1) or just a scalar number.

Row vector [x1 x2 x3 .. Xn]

Column vector $\begin{bmatrix} x1 \\ x2 \\ . \\ . \\ xn \end{bmatrix}$

➢If B shape is (1 x n) matrix (row vector),
then it gets replicated itself column-wise to become (m x n) matrix.

➢If B shape is (m x 1) matrix (column vector),
it gets replicated row-wise to become (m x n) matrix.

➢If it is a scalar number, then
it gets converted to (m x n) matrix where each of the element is equal to the scalar number

➢Broadcasting also works in case of function

Broadcasting Illustration

$$\begin{bmatrix} p1 & q1 & r1 \\ p2 & q2 & r2 \end{bmatrix} + [X1\ \ X2\ \ X3] = \begin{bmatrix} P1+X1 & q1+X2 & r1+X3 \\ p2+X1 & q2+X2 & r2+X3 \end{bmatrix} = \begin{bmatrix} Y1 & y2 & y3 \\ Y4 & y5 & y6 \end{bmatrix}$$

2. Multiplication of a matrix and a column vector

$$\begin{bmatrix} p1 & q1 & r1 \\ p2 & q2 & r2 \end{bmatrix} X [X1\ \ X2\ \ X3\ ] = \begin{bmatrix} P1 \times X1 & q1xX2 & r1 \times X3 \\ p2 \times X1 & q2xX2 & r2 \times X3 \end{bmatrix} = \begin{bmatrix} Y7 & y8 & y9 \\ Y10 & y11 & y12 \end{bmatrix}$$

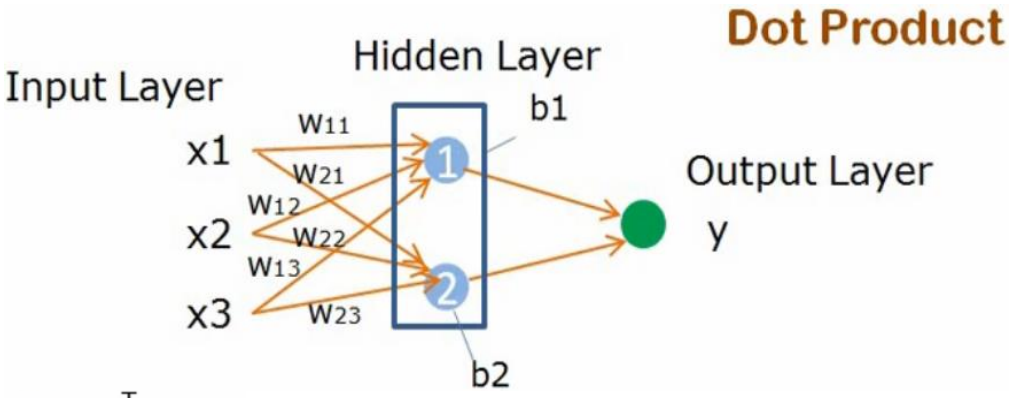
2. Addition of a matrix and a scalar

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + 2 = \begin{bmatrix} 1+2 & 2+2 & 3+2 \\ 4+2 & 5+2 & 6+2 \end{bmatrix} = \begin{bmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

3. Element-wise function call

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \text{Fun } **2 = \begin{bmatrix} 1**2 & 2**2 & 3**2 \\ 4**2 & 5**2 & 6**2 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 9 \\ 16 & 25 & 36 \end{bmatrix}$$

Dot Product



$$Z = W^T X + B$$

$$\hat{Y} = Z$$

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

$$\begin{bmatrix} w11 & W12 & w13 \\ w21 & w22 & w23 \end{bmatrix} \cdot \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \end{bmatrix} = \begin{bmatrix} w11 \times X1 + w12xX2 + w13xX3 + b1 \\ w21x\,X1 + w22xX2 + w23xX3 + b2 \end{bmatrix}$$

$$\begin{bmatrix} w11 & W12 & w13 \end{bmatrix} \cdot \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} + b1 = w11 \times X1 + w12xX2 + w13xX3 + b1 = z1$$

$$a1 = \hat{y}1 = \sigma(z1)$$

$$\begin{bmatrix} w21 & w22 & w23 \end{bmatrix} \cdot \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} + b2 = w21x\,X1 + w22xX2 + w23xX3 + b2 = z2$$

$$a2 = \hat{y}2 = \sigma(z2)$$

$$\begin{bmatrix} w01 & Wo2 \end{bmatrix} \cdot \begin{bmatrix} a1 \\ a2 \end{bmatrix} + b0 = w01 \times a1 + w02 \times a2 + b0 = z$$

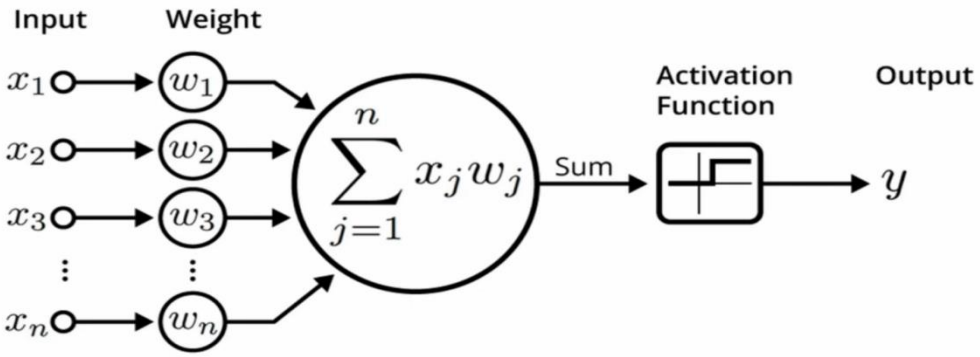
$$y = \sigma(z)$$

What is Activation Functions?

Activation functions are decision making units of neural networks. They calculates net output of a neural node

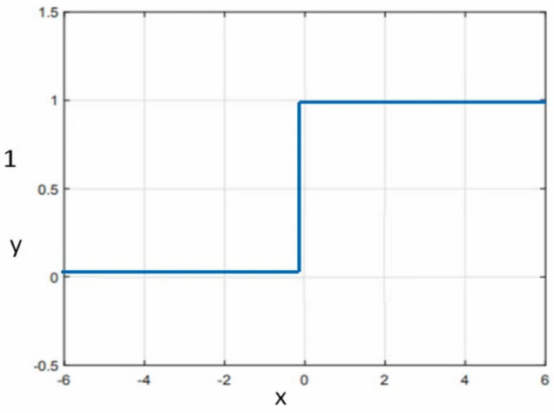
Why do we need Activation Functions?

Without activation function neural network is just a linear regression model



Heaviside or Binary Step Function

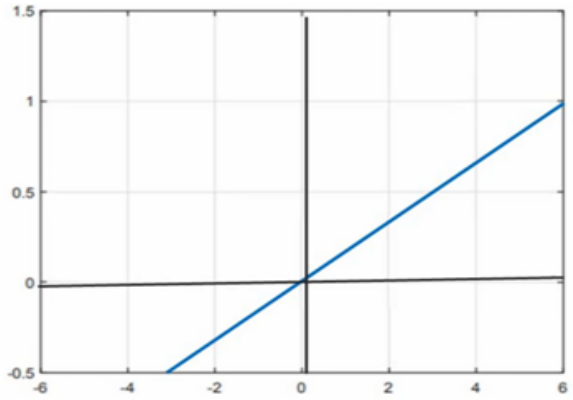
Value Range : 0 to 1



f(x) = 1 if x > 0 else 0 if x < 0

Linear Function

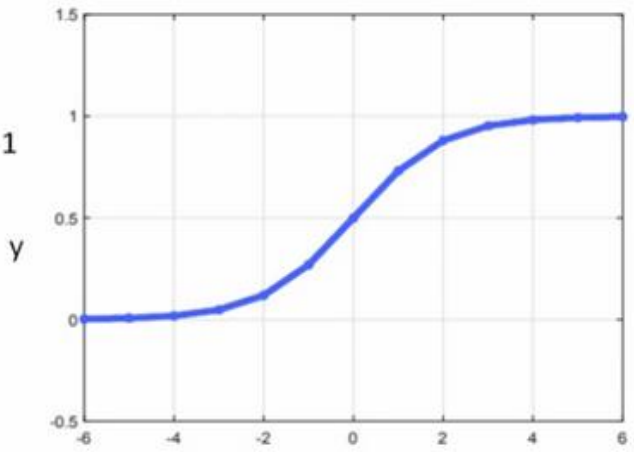
Value Range : ∞ to - ∞



f(x) = x

Sigmoid Function

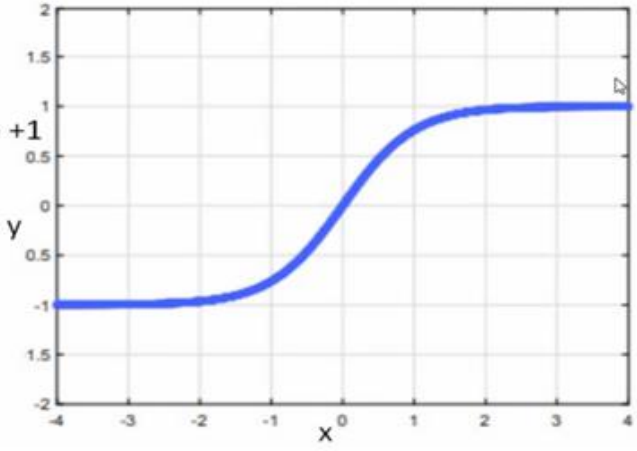
Value Range : 0 to 1



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

Tanh Function

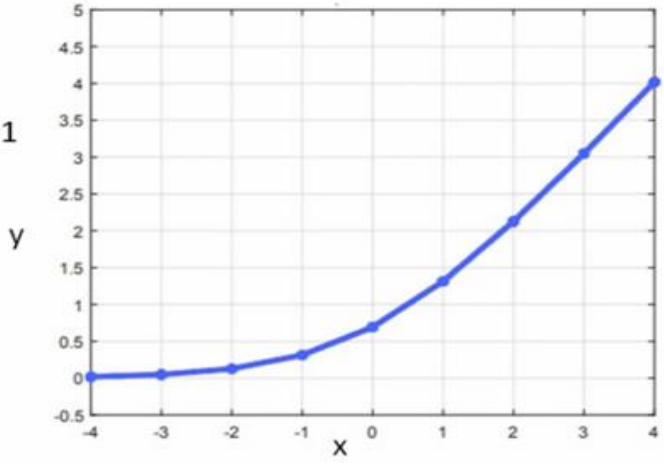
Value Range : **-1** to +1



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

Softplus Function

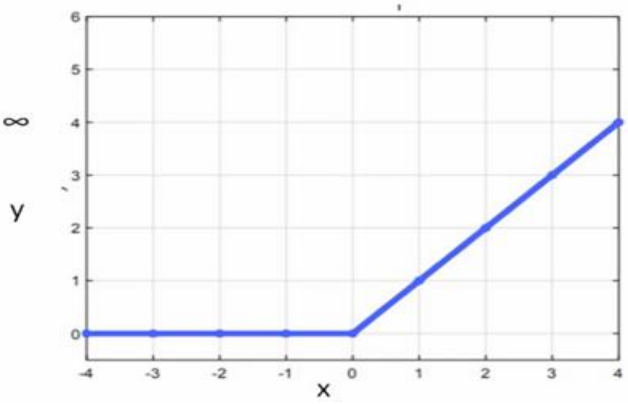
Value Range : 0 to 1



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

ReLu (Rectified linear unit) Function

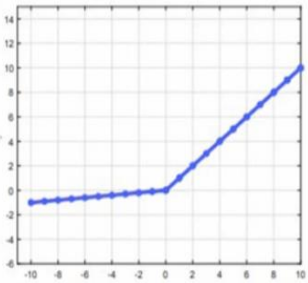
Value Range : 0 to ∞



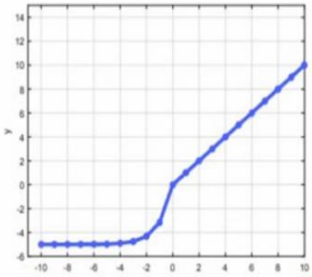
$$f(x) = \max(0, x)$$

Variations in ReLu Function

$$f(x) = \begin{cases} x & (x > 0) \\ a \cdot x & (otherwise) \end{cases}$$



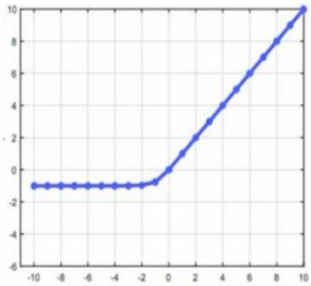
Leaky ReLu Function



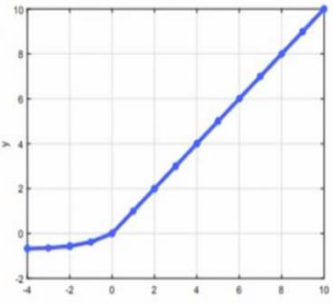
Elu Function

$$f(x) = \begin{cases} x & (x \geqslant 0) \\ a \cdot (e^x - 1) & (otherwise) \end{cases}$$

$$f(x) = \begin{cases} \frac{1-e^{-2x}}{1+e^{-2x}} & (x < 0) \\ \max(0, x) & (otherwise) \end{cases}$$



Tanh ReLu Function

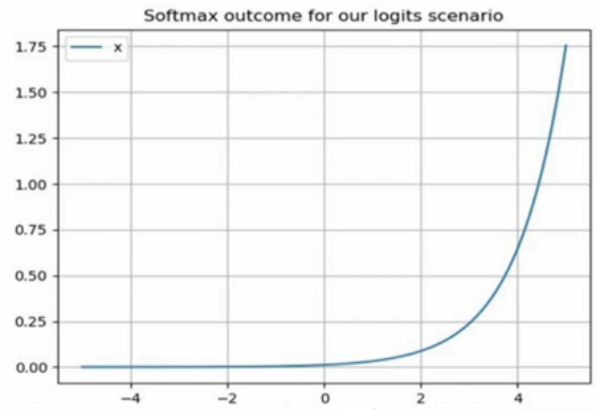


Softplus (SmoothReLU)

$$f(x) = \begin{cases} \ln(1 + e^x) - \ln 2 & (x < 0) \\ \max(0, x) & (otherwise) \end{cases}$$

ReLu Function

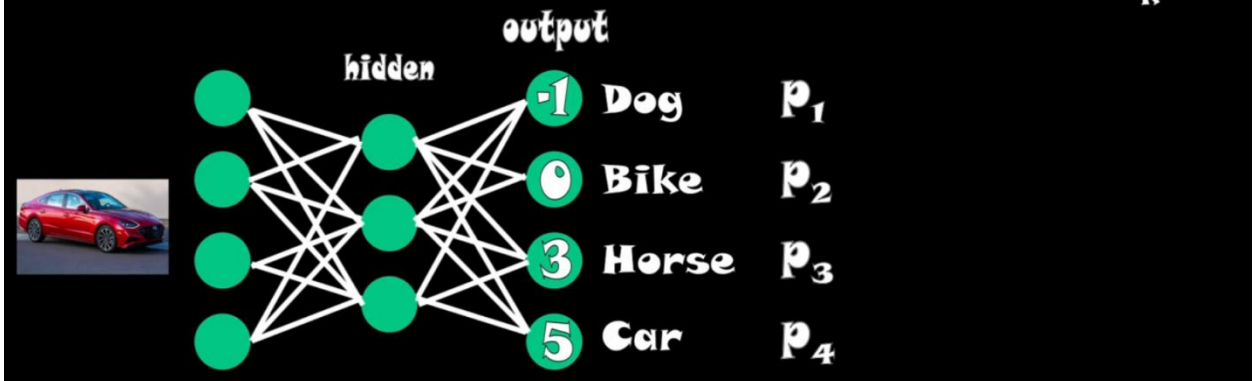
Softmax Function



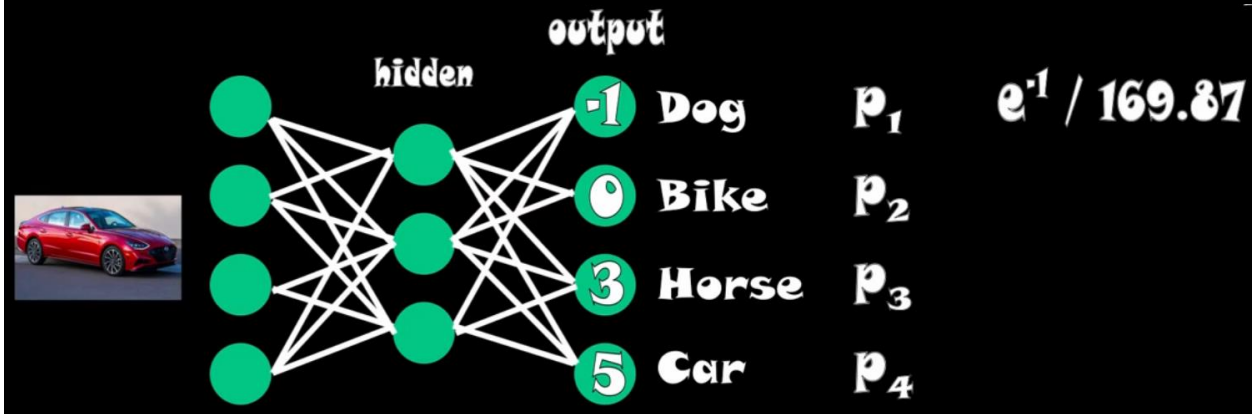
$$f_i(\vec{a}) = \frac{e^{a_i}}{\sum_k e^{a_k}}$$

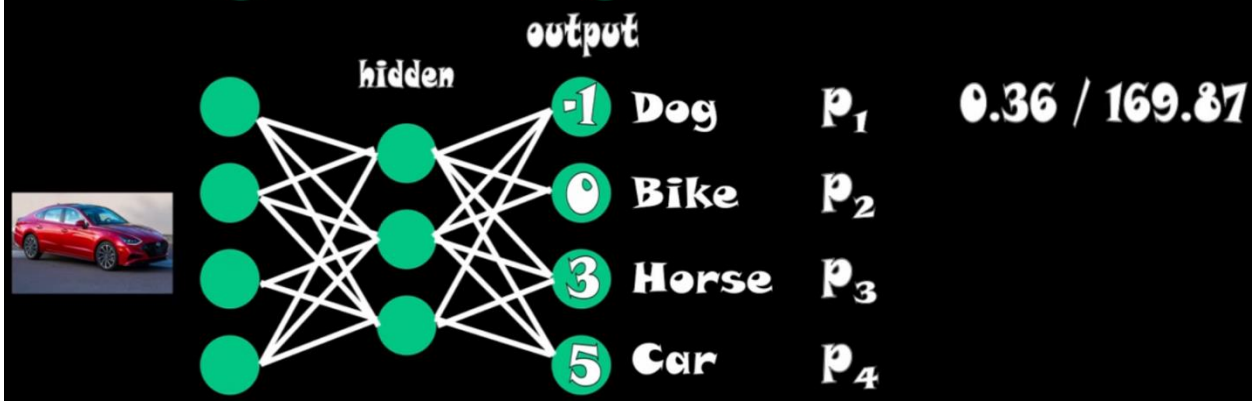
Softmax

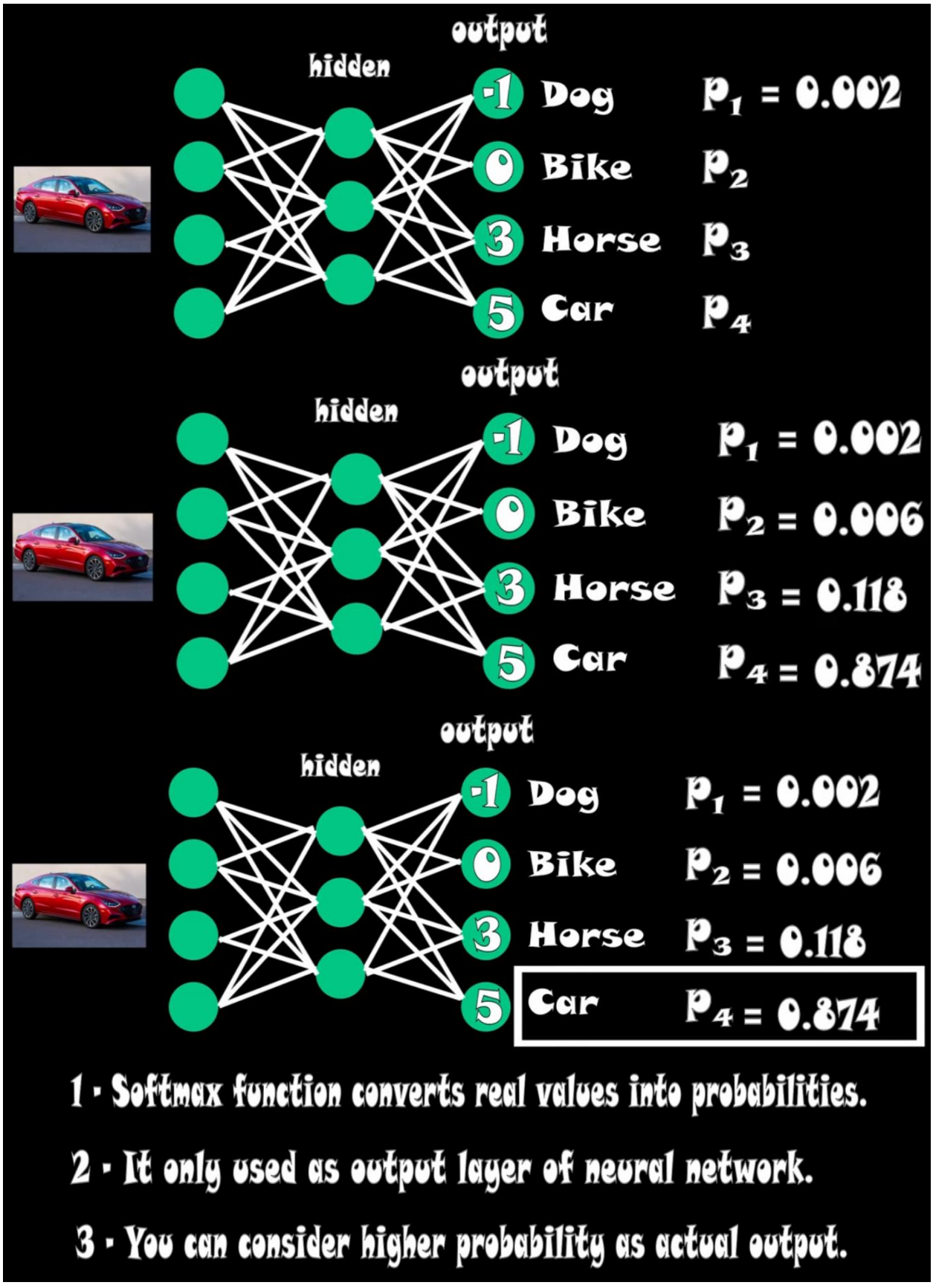
$$f(y_i) = \frac{e^{y_i}}{\sum_k e^{y_k}}$$



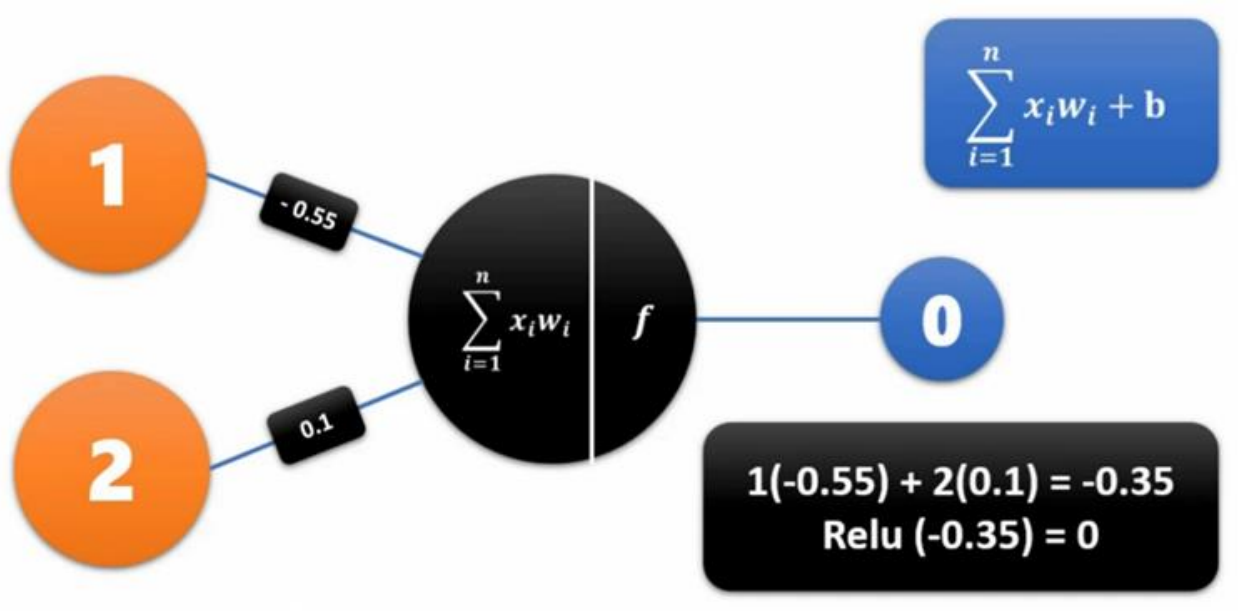
$$\sum_k e^{y_k} = e^{-1} + e^{0} + e^{3} + e^{5} = 169.87$$

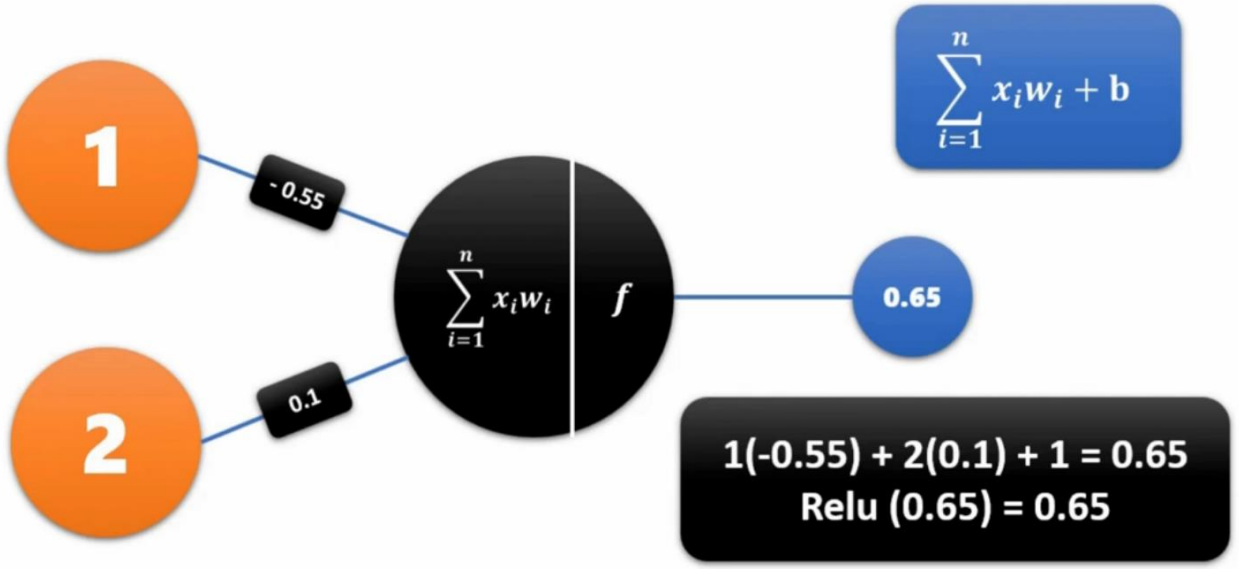





output
hidden
-1 Dog
0 Bike
3 Horse
5 Car
p_1 = 0.002
p_2
p_3
p_4
output
hidden
-1 Dog
0 Bike
3 Horse
5 Car
p_1 = 0.002
p_2 = 0.006
p_3 = 0.118
p_4 = 0.874
output
hidden
-1 Dog
0 Bike
3 Horse
5 Car
p_1 = 0.002
p_2 = 0.006
p_3 = 0.118
p_4 = 0.874
1 - Softmax function converts real values into probabilities.
2 - It only used as output layer of neural network.
3 - You can consider higher probability as actual output.

What is the Role of Bias in ANN?





Vanishing and Exploding Gradient

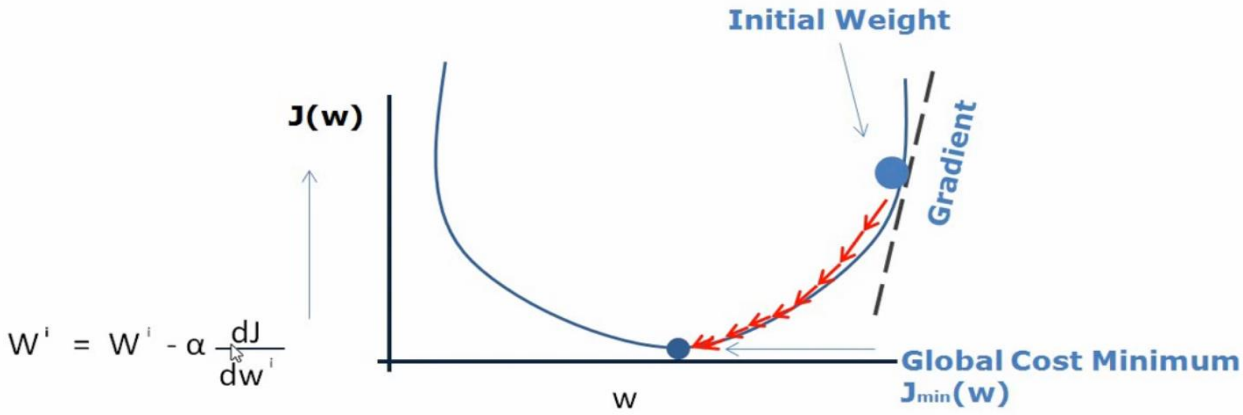
- Vanishing Gradient and Exploding Gradient Problem are difficulties found in training certain Artificial Neural Networks with gradient based methods like Back Propagation.
- In particular, this problem makes it really hard to learn and tune the parameters of the earlier layers in the network.

To calculate derivative of error w.r.t first weight, back-propagate via chain rule

$$\frac{dJ}{dw1} = \frac{dJ}{d\hat{y}} * \frac{d\hat{y}}{da2} * \frac{da2}{da1} * \frac{da1}{dw1}$$

$$W\ 1 = W1 - \alpha \frac{dJ}{dw1}$$

How does Gradient Descent work?

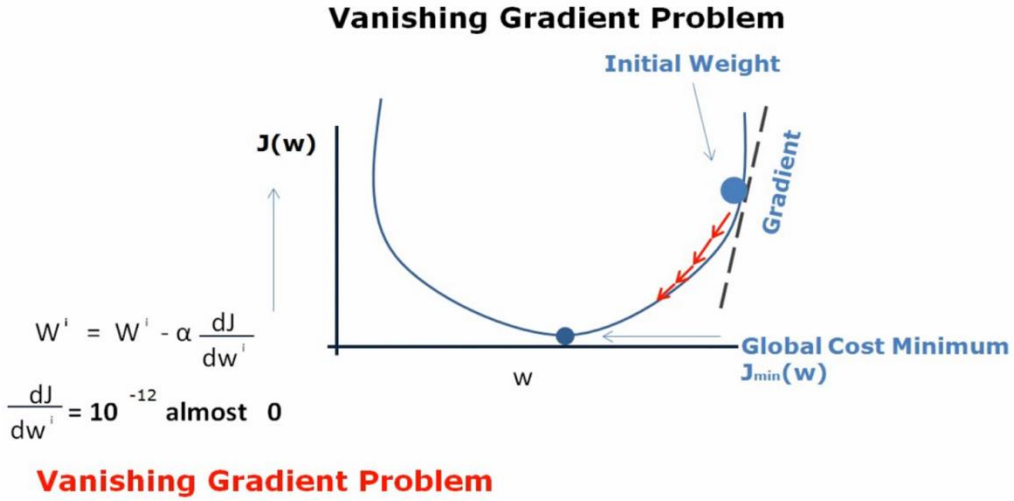


$$W^{i} = W^{i} - \alpha \frac{dJ}{dw^{i}}$$

α = learning rate

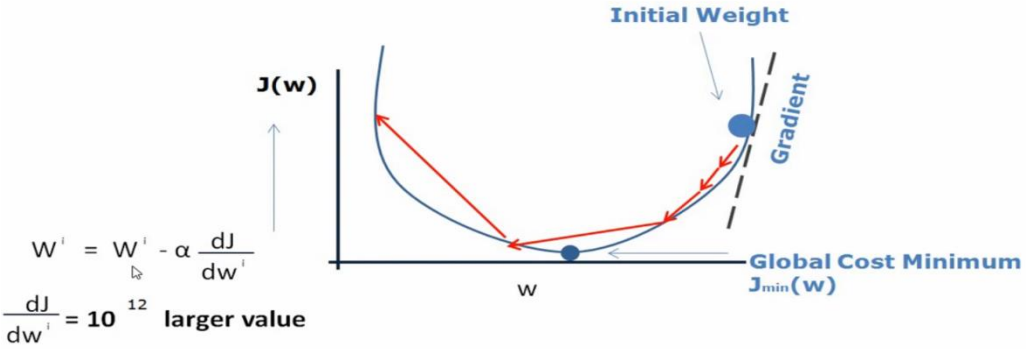
In Maths **Derivative** is a way to show rate of change at a given point

Vanishing Gradient Problem



Vanishing Gradient Problem

Exploding Gradient Problem



Exploding Gradient Problem

W ~ small - > Vanishing
W ~ large - > Exploding

Solution for Vanishing Gradient Problem

Vanishing Gradient Problem was because of Sigmoid function. Range 0-.25
Even if I use tanh same problem I will face Vanishing Gradient Problem tanh 0-1.

-The simplest solution is to use
ReLU activation function instead of sigmoid or tanh activation
which doesn't cause a small derivative.

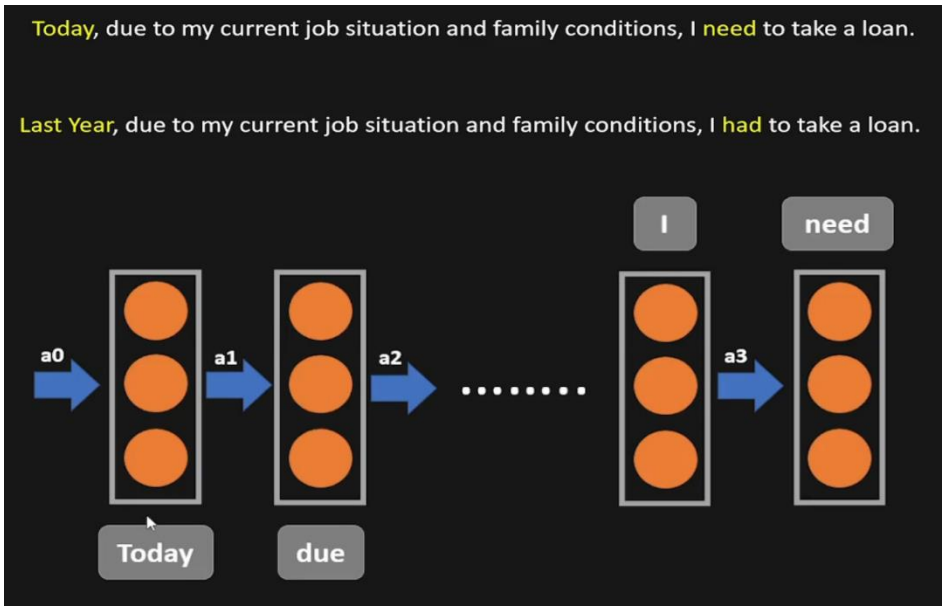
-Use weight initializers.

-Use some other network

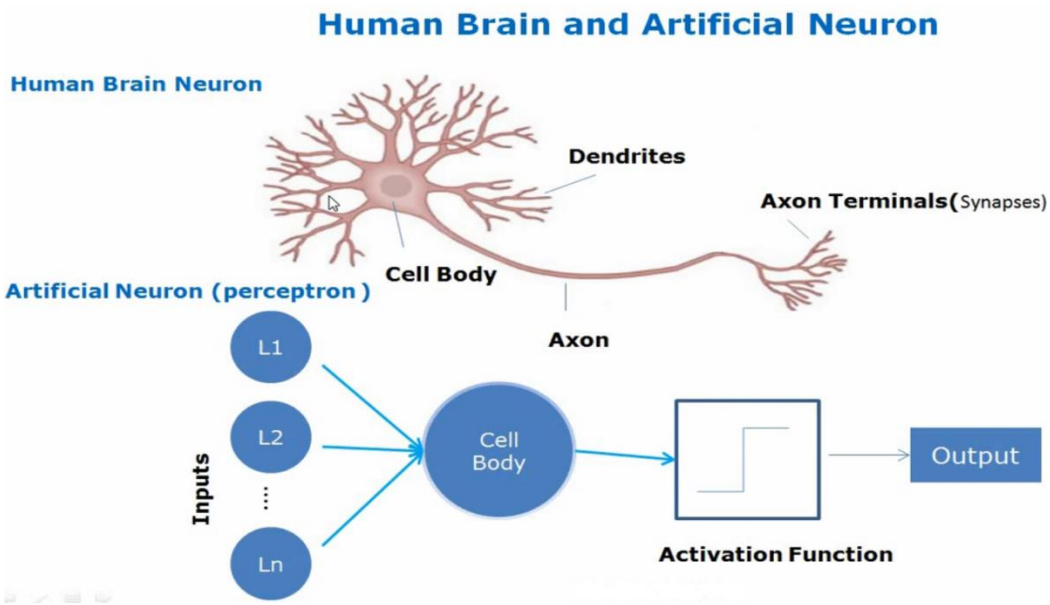
Solution for Exploding Gradient Problem

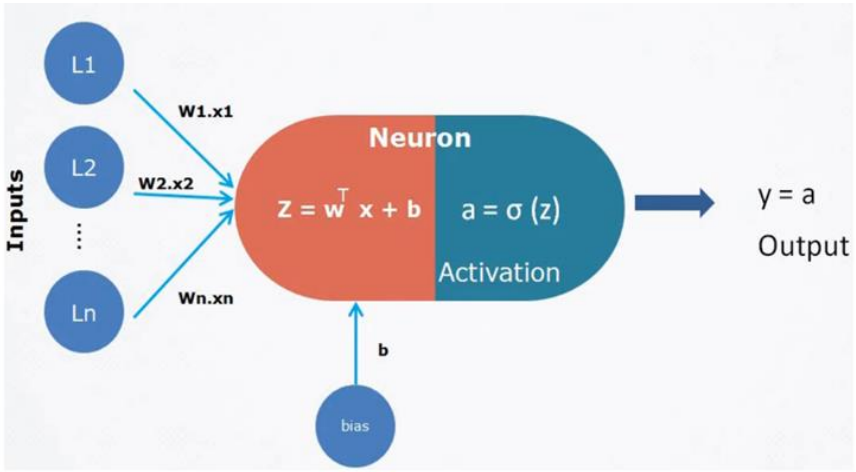
**A solution to fix this is to apply gradient clipping.
which places a predefined threshold on the gradients to prevent it
from getting too large, and by doing this it doesn't change the direction
of the gradients it only change its length.**

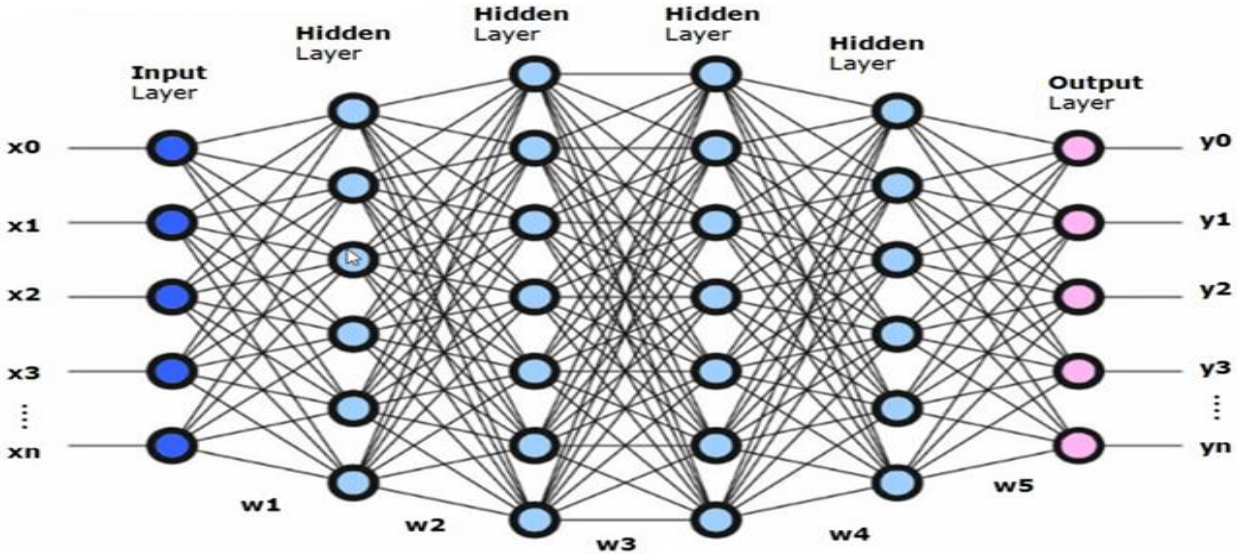

VANISHING GRADIENT
Subscribe to Digital Daru
OK
4 Dino Ke Baad
Mene tumhe kuch yaad rakhne ko kha tha
Digital Daru ko subscribe krna hai
4 Months Ke Baad
Mene tumhe kuch yaad rakhne ko kha tha
Mujhe to Kl ka khaya ni yaad rehta to ye kase rahta


Today, due to my current job situation and family conditions, I need to take a loan.
Last Year, due to my current job situation and family conditions, I had to take a loan.
I
need
a0
a1
a2
a3
Today
due


Solution ?
GRU
LSTM


Human Brain and Artificial Neuron
Human Brain Neuron
Dendrites
Axon Terminals(Synapses)
Cell Body
Axon
Artificial Neuron (perceptron)
L1
L2
Ln
Inputs
Cell Body
Output
Activation Function

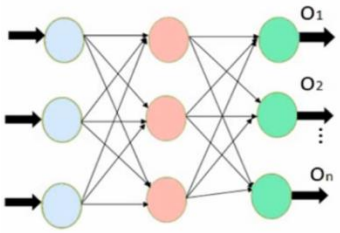

L1
L2
Ln
Inputs
W1.x1
W2.x2
Wn.xn
Neuron
z = w^T x + b
a = σ (z)
Activation
y = a
Output
b
bias


Input Layer
Hidden Layer
Hidden Layer
Hidden Layer
Hidden Layer
Output Layer
x0
x1
x2
x3
xn
y0
y1
y2
y3
yn
w1
w2
w3
w4
w5

Feed Forward Neural Network

- ✓ FFNN is one of the simplest forms of ANN
- ✓ A feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle
- ✓ because inputs are processed only in the forward direction
- ✓ FFNN could be a Single layer perceptron or Multi layer perceptron
- ✓ We don't use it where order of sequence does matter. Like sentence as it predict only for current node , does not depend on previous.

Mean squared error(MSE) in FFNN



$$E_{total} = \frac{1}{\text{Total output}} \sum (\text{Actual output} - \text{predicted output})^2$$

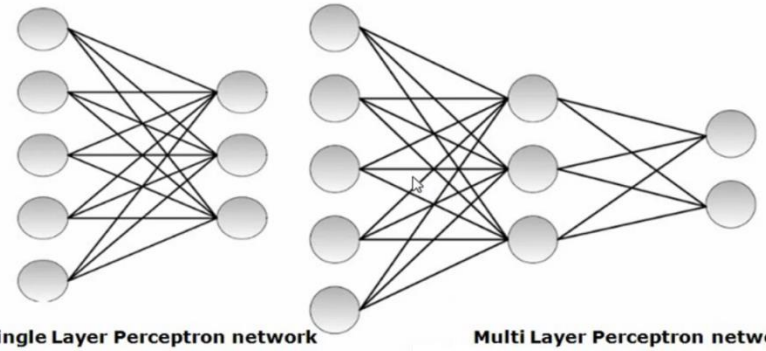
$$E_{total} = \frac{1}{n} \sum_{m=1}^{n} (Om - \hat{Om})^2$$

Try to minimize the error for better prediction

Perceptron

Single Layer Perceptron (SLP) –
This is the simplest feedforward neural network and does not contain any hidden layer.

Multi Layer Perceptron (MLP)
A Multi Layer Perceptron has one or more hidden layers.

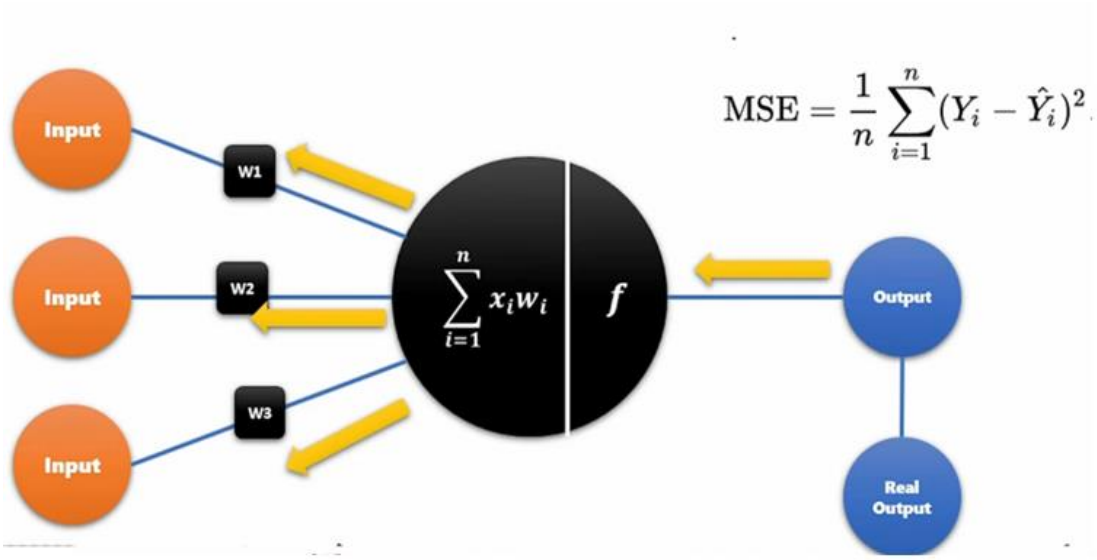


Single Layer Perceptron network **Multi Layer Perceptron network**

Applications

- ✓Data Compression.
- ✓Pattern Recognition.
- ✓Computer Vision.
- ✓Sonar Target Recognition.
- ✓Speech Recognition.
- ✓Handwritten Characters Recognition.

Back Propagation Neural Network



Try to minimize the error for better prediction model

Error Calculation – difference of model output from the actual output.

Minimum Error – Check whether the error is minimized or not.

Update the parameters – If the error is huge then, update the parameters weights and biases and again check the error. Repeat the process until the error becomes minimum.

Model is ready to make a prediction – Now model will produce the output more accurate.

Applications of Backpropagation

i. **Speech Recognition**
ii. **Face Recognition**
iii. **Signature Recognition**

Convolutional Neural Network (CNN)

CNN is a special type of Feed Forward Neural network

- ✓Images recognition
- ✓Images classifications
- ✓Objects detections
- ✓Faces Recognition

Convolutional Neural Network (CNN)

Filters/Kernals

-1	1	-1
-1	1	-1
1	1	1

1	1	1
-1	1	-1
-1	1	-1

-1	1	-1
1	1	1
-1	1	-1

Training Data

-1	-1	1	-1	-1
-1	-1	1	-1	-1
1	1	1	1	1
-1	-1	1	-1	-1
-1	-1	1	-1	-1

Testing Data

-1	-1	1	-1	-1
-1	-1	1	-1	-1
1	1	1	1	1
-1	-1	1	-1	-1
-1	-1	1	-1	-1

KotaCoaching.com

Activate Windows

Filters

-1	1	-1
-1	1	-1
1	1	1

1	1	1
-1	1	-1
-1	1	-1

-1	1	-1
1	1	1
-1	1	-1

(1 + 1 + 1 + 1 + 1 + 1 +1 +1 +1)/9 = 1

-1	-1	1	-1	-1
-1	1	1	-1	-1
1	1	1	1	1
-1	-1	1	-1	-1
-1	-1	1	-1	-1

1	1	1
1	1	1
1	1	1

Feature Map

.11	1	

Stride is the number of pixels shifts over the input matrix.
Here Stride = 1

KotaCoaching.com

Activate Windows

Filters

-1	1	-1
-1	1	-1
1	1	1

1	1	1
-1	1	-1
-1	1	-1

-1	1	-1
1	1	1
-1	1	-1

-1	-1	1	-1	-1
-1	-1	1	-1	-1
1	1	1	1	1
-1	-1	1	-1	-1
-1	-1	1	-1	-1

to make the final filter more informative
we use padding in image matrix

Featured Images

.11	1	.33
-.33	.11	-.33
-.33	.11	-.11

-33	.11	-.33
-33	.11	-33
.11	1	-11

-.11	.11	-33
-33	1	.33
-33	.11	-11

KotaCoaching.com

ReLu Layer

Filters

-1	1	-1
-1	1	-1
1	1	1

1	1	1
-1	1	-1
-1	1	-1

-1	1	-1
1	1	1
-1	1	-1

-1	-1	1	-1	-1
-1	-1	1	-1	-1
1	1	1	1	1
-1	-1	1	-1	-1
-1	-1	1	-1	-1

$$f(x) = \begin{cases} 0 \text{ if } x < 0 \\ X \text{ if } x >= 0 \end{cases}$$

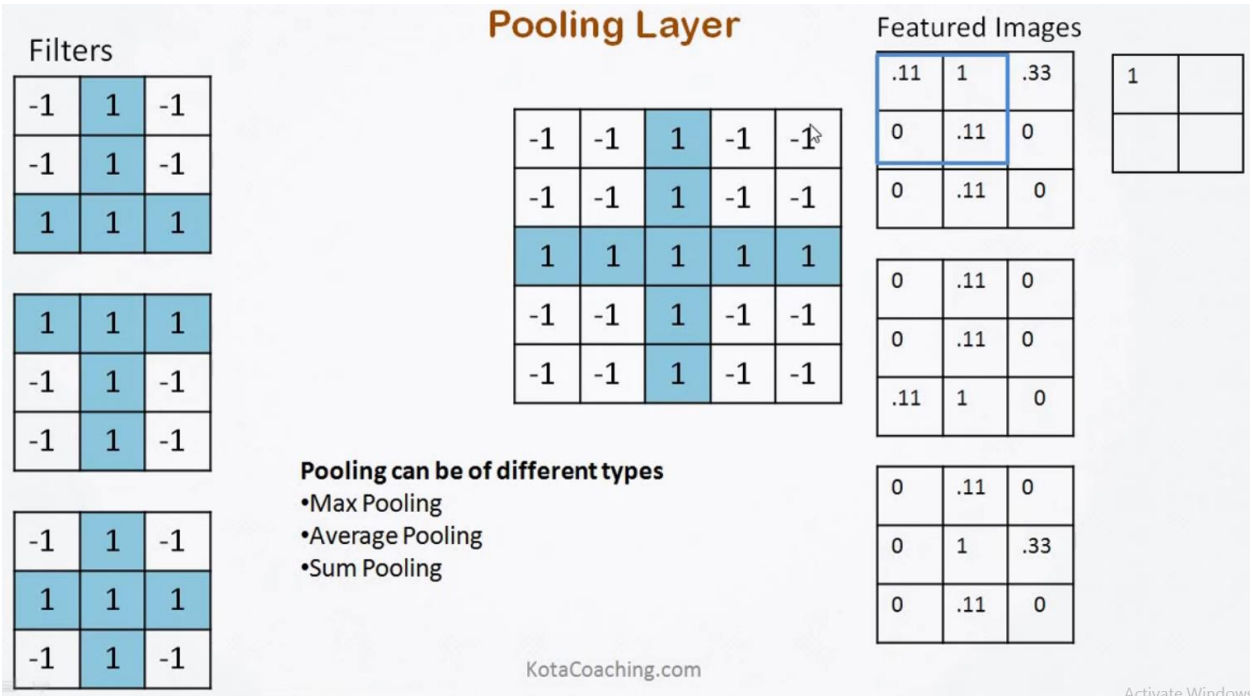
Featured Images

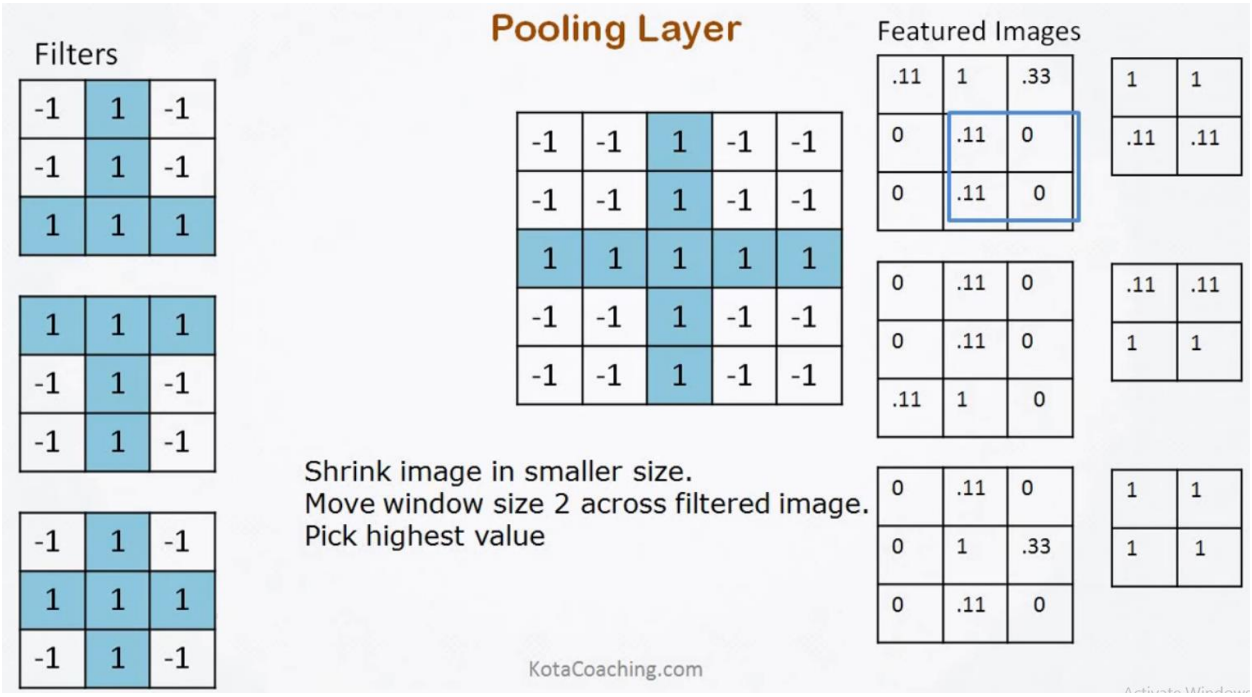
.11	1	.33
0	.11	0
0	.11	0

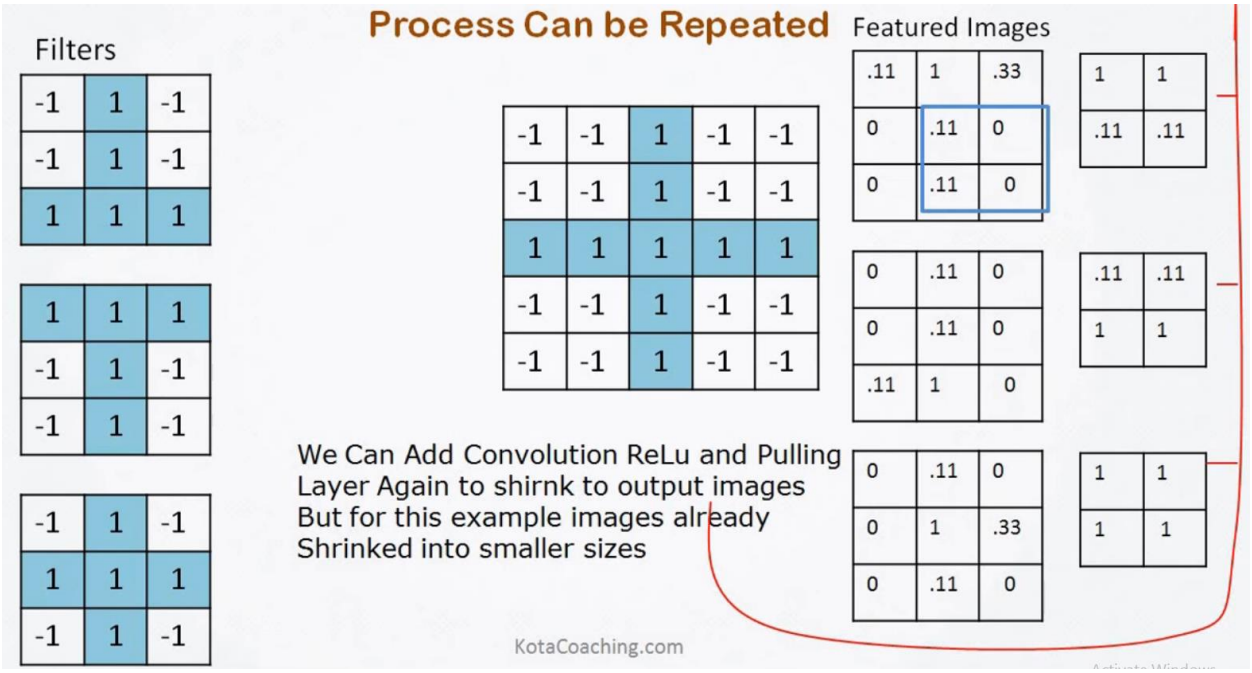
0	.11	0
0	.11	0
.11	1	0

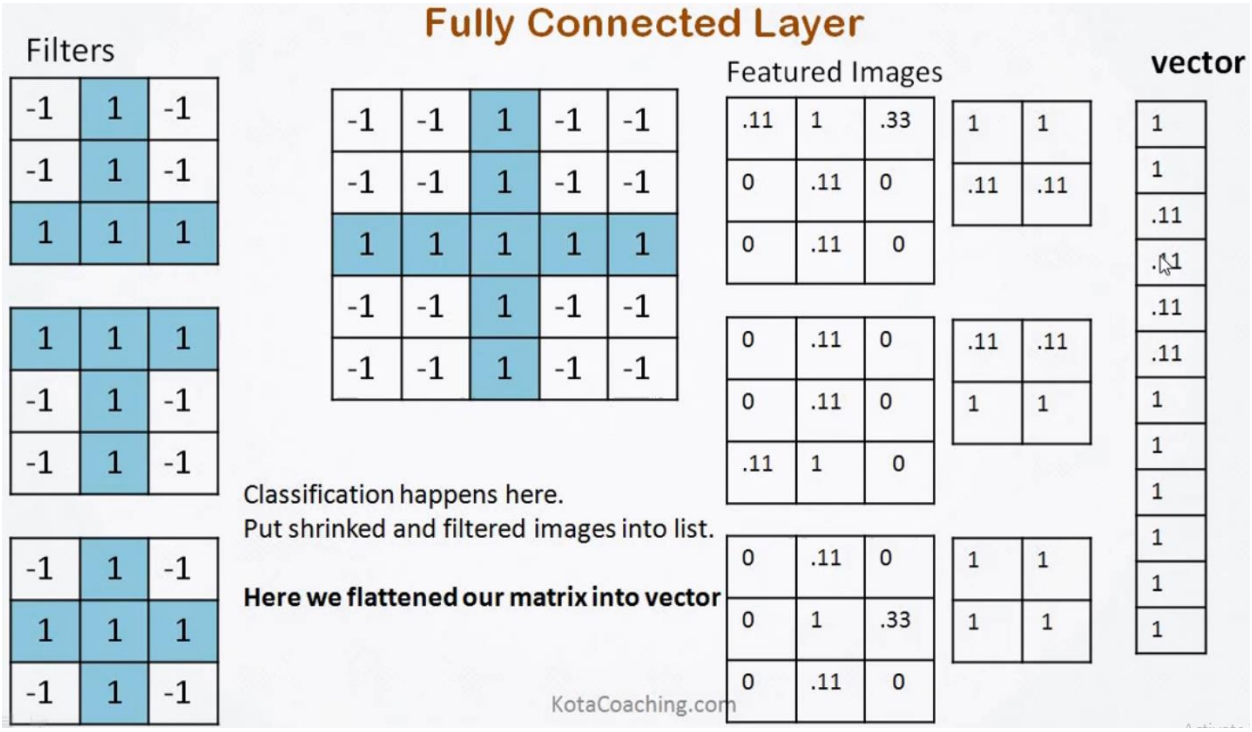
0	.11	0
0	1	.33
0	.11	0

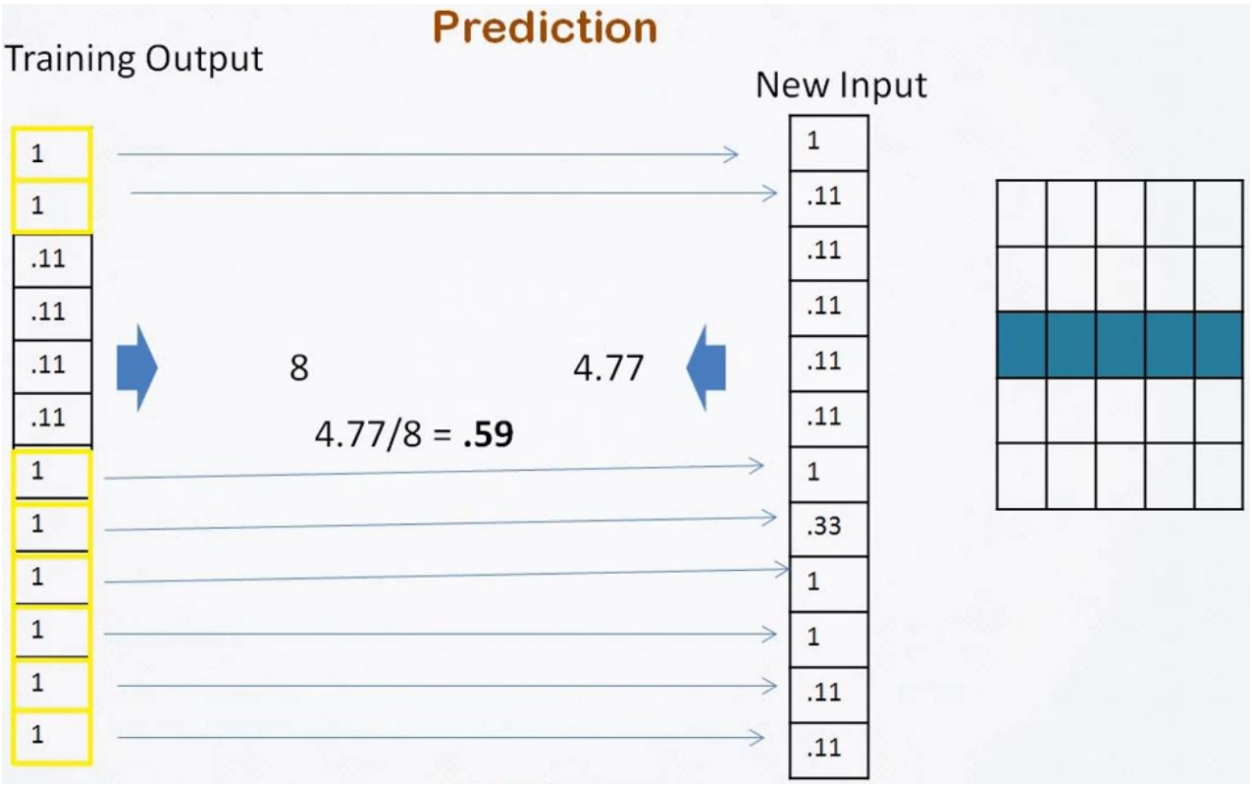
KotaCoaching.com

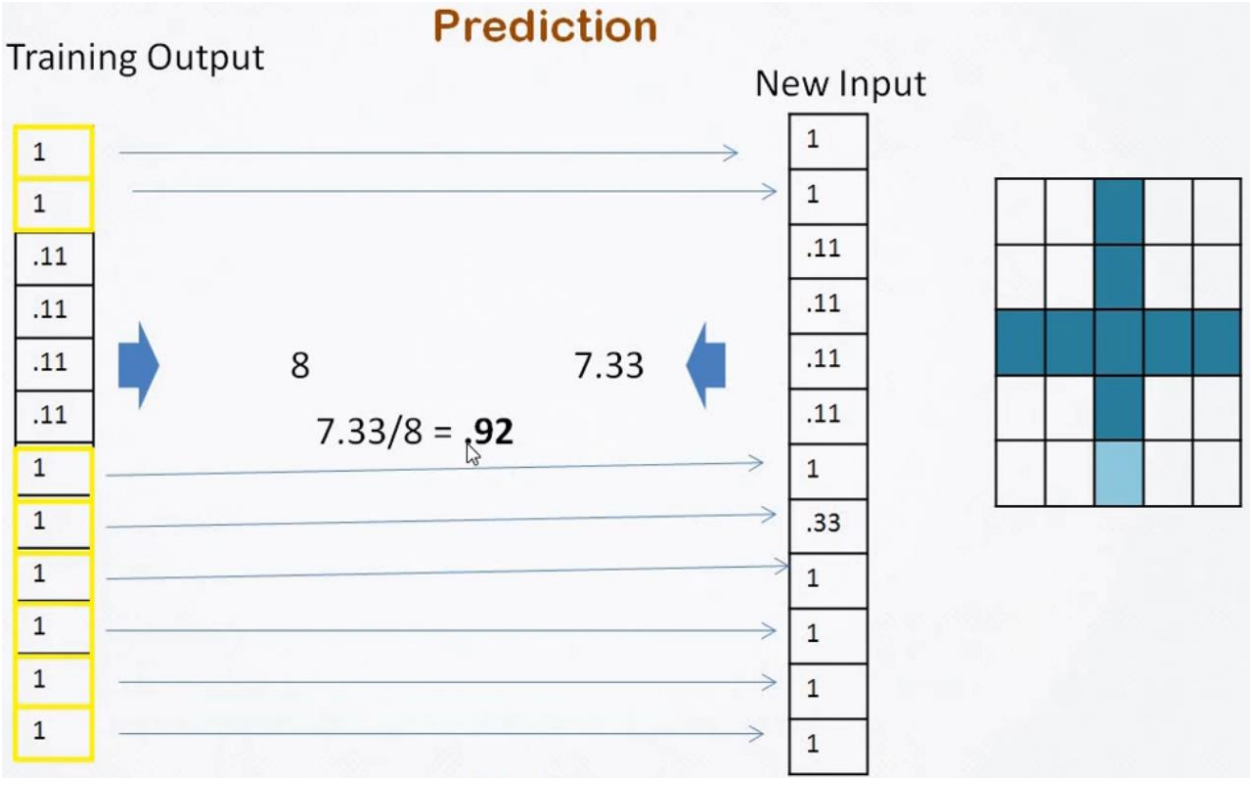

Pooling Layer
Filters
-1 1 -1
-1 1 -1
1 1 1
1 1 1
-1 1 -1
-1 1 -1
-1 1 -1
1 1 1
-1 1 -1
-1 -1 1 -1 -1
-1 -1 1 -1 -1
1 1 1 1 1
-1 -1 1 -1 -1
-1 -1 1 -1 -1
Featured Images
.11 1 .33
0 .11 0
0 .11 0
1
0 .11 0
0 .11 0
.11 1 0
0 .11 0
0 1 .33
0 .11 0
Pooling can be of different types
•Max Pooling
•Average Pooling
•Sum Pooling
KotaCoaching.com


Pooling Layer
Filters
-1 1 -1
-1 1 -1
1 1 1
1 1 1
-1 1 -1
-1 1 -1
-1 1 -1
1 1 1
-1 1 -1
-1 -1 1 -1 -1
-1 -1 1 -1 -1
1 1 1 1 1
-1 -1 1 -1 -1
-1 -1 1 -1 -1
Featured Images
.11 1 .33
0 .11 0
0 .11 0
1 1
.11 .11
0 .11 0
0 .11 0
.11 1 0
.11 .11
1 1
0 .11 0
0 1 .33
0 .11 0
1 1
1 1
Shrink image in smaller size.
Move window size 2 across filtered image.
Pick highest value
KotaCoaching.com


Process Can be Repeated
Filters
-1 1 -1
-1 1 -1
1 1 1
1 1 1
-1 1 -1
-1 1 -1
-1 1 -1
1 1 1
-1 1 -1
-1 -1 1 -1 -1
-1 -1 1 -1 -1
1 1 1 1 1
-1 -1 1 -1 -1
-1 -1 1 -1 -1
Featured Images
.11 1 .33
0 .11 0
0 .11 0
1 1
.11 .11
0 .11 0
0 .11 0
.11 1 0
.11 .11
1 1
0 .11 0
0 1 .33
0 .11 0
1 1
1 1
We Can Add Convolution ReLu and Pulling Layer Again to shirnk to output images But for this example images already Shrinked into smaller sizes
KotaCoaching.com


Fully Connected Layer
Filters
-1 1 -1
-1 1 -1
1 1 1
1 1 1
-1 1 -1
-1 1 -1
-1 1 -1
1 1 1
-1 1 -1
-1 -1 1 -1 -1
-1 -1 1 -1 -1
1 1 1 1 1
-1 -1 1 -1 -1
-1 -1 1 -1 -1
Featured Images
.11 1 .33
0 .11 0
0 .11 0
1 1
.11 .11
0 .11 0
0 .11 0
.11 .11
1 1
.11 1 0
0 .11 0
0 1 .33
0 .11 0
1 1
1 1
vector
1
1
.11
.11
.11
.11
1
1
1
1
1
1
Classification happens here.
Put shrinked and filtered images into list.
Here we flattened our matrix into vector
KotaCoaching.com


Prediction
Training Output
New Input
1
1
.11
.11
.11
.11
1
1
1
1
1
1
1
.11
.11
.11
.11
.11
1
.33
1
1
.11
.11
8
4.77
4.77/8 = .59


Prediction
Training Output
New Input
1
1
.11
.11
.11
.11
1
1
1
1
1
1
1
.11
.11
.11
.11
.11
1
.33
1
1
.11
.11
8
7.33
7.33/8 = .92

0	0	0	0	0	0	0
0	-1	-1	1	-1	-1	0
0	-1	-1	1	-1	-1	0
0	1	1	1	1	1	0
0	-1	-1	1	-1	-1	0
0	-1	-1	1	-1	-1	0
0	0	0	0	0	0	0

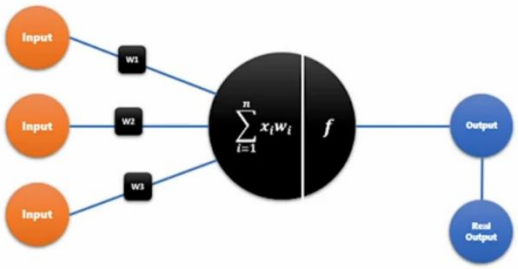
Padding : it refers to the amount of pixels added to an image when it is being processed by the kernel of a CNN

Padding is simply a process of adding layers of zeros to our input image

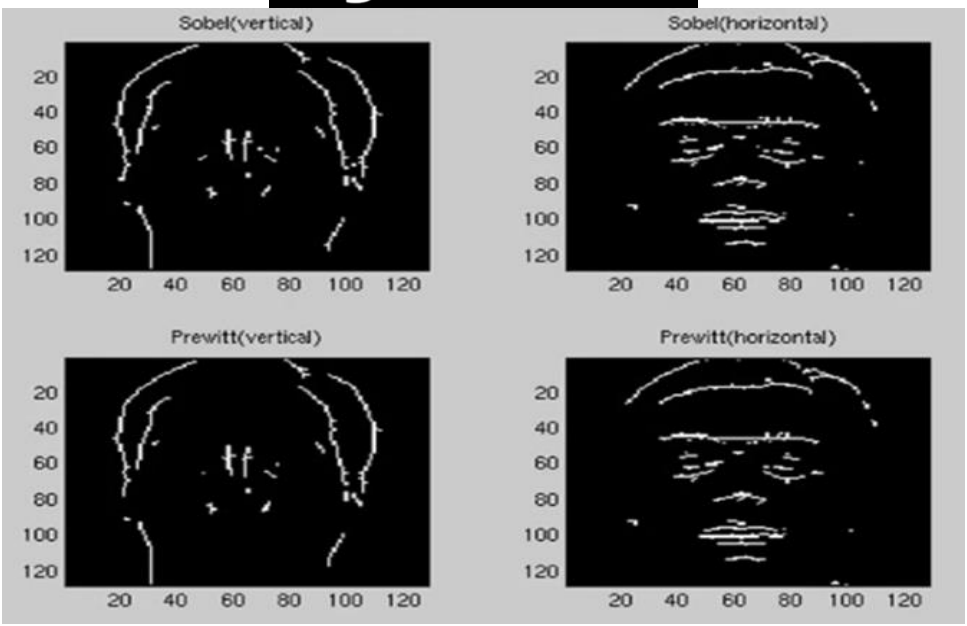
Computer Vision

1920 x 1080 x 3

6,220,800 px



Edge Detection



Vertical Edge Detection

6 x 6

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

Filter

3 x 3

1	0	-1
1	0	-1
1	0	-1

=

4 x 4

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

(3x1)+(1x1)+(2x1)+(0x0)+(5x0)+(7x0)+(1x-1)+(8x-1)+(2x-1) = -5

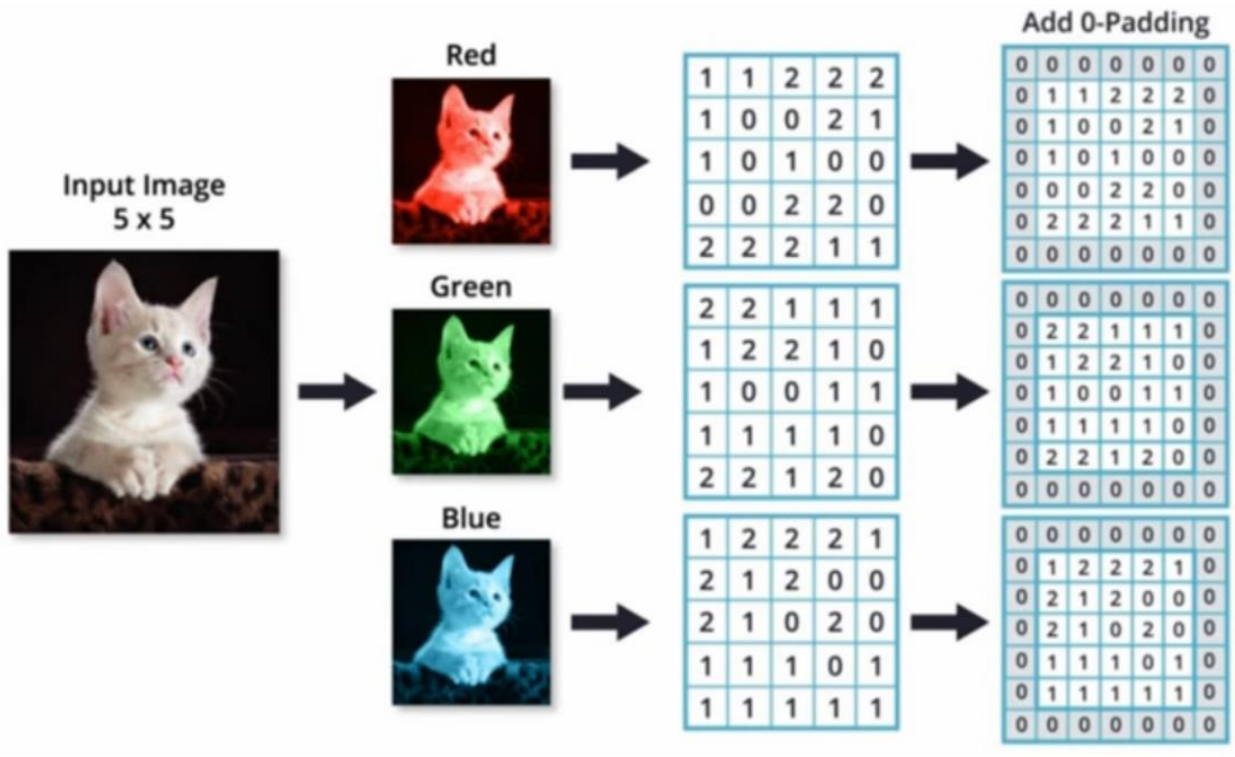
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

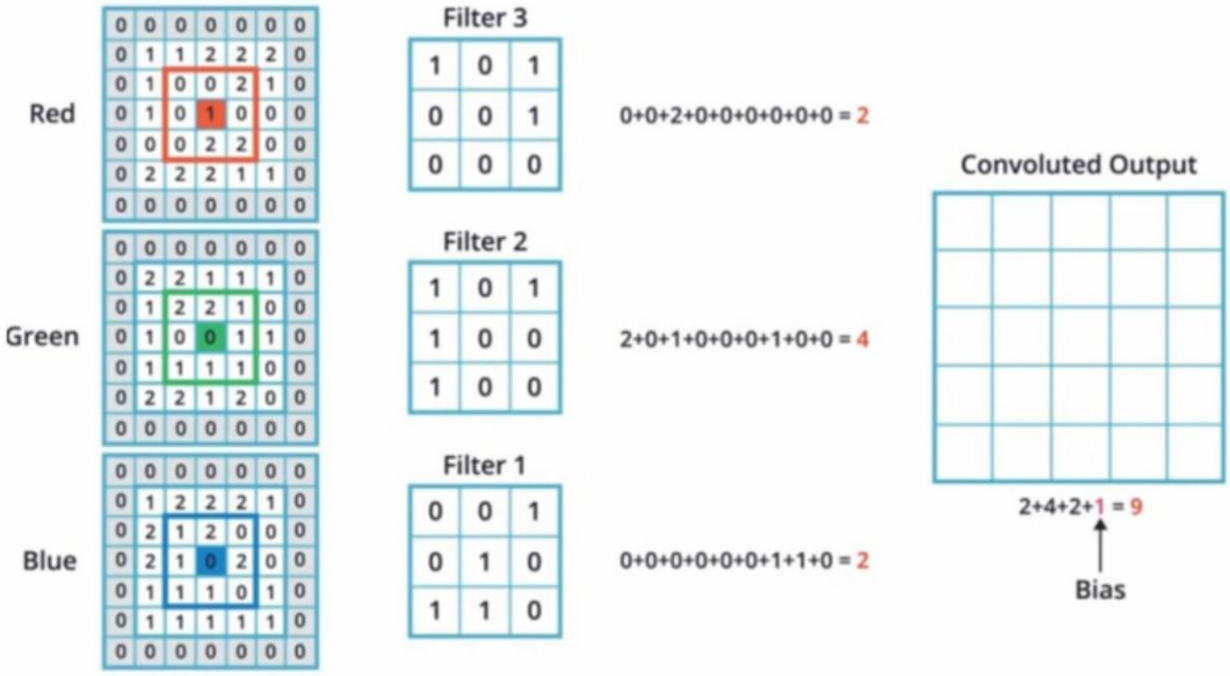
*

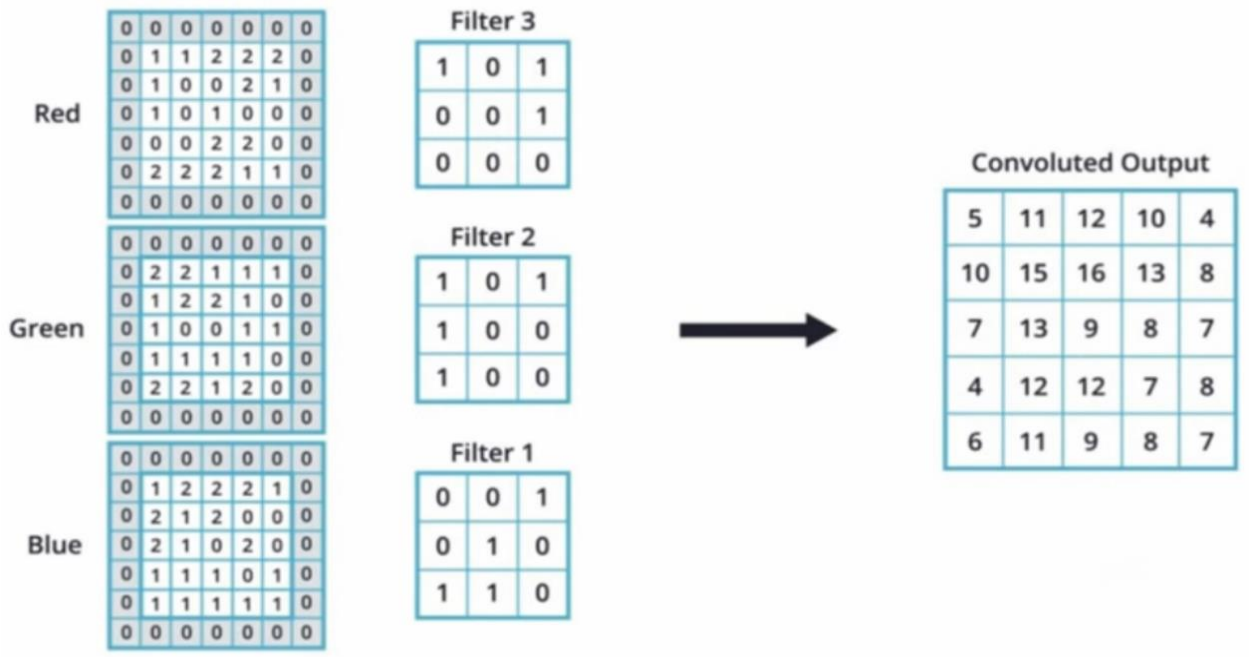
1	0	-1
1	0	-1
1	0	-1

=

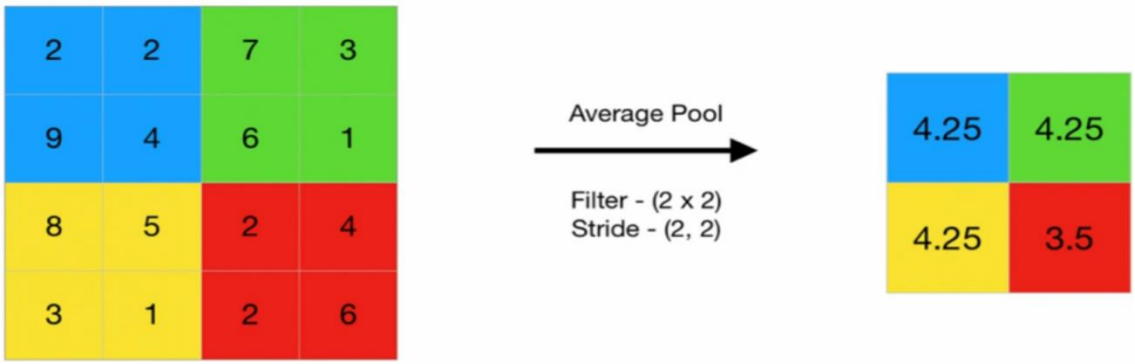
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0


Input Image
5 x 5
Red
Green
Blue
Add 0-Padding

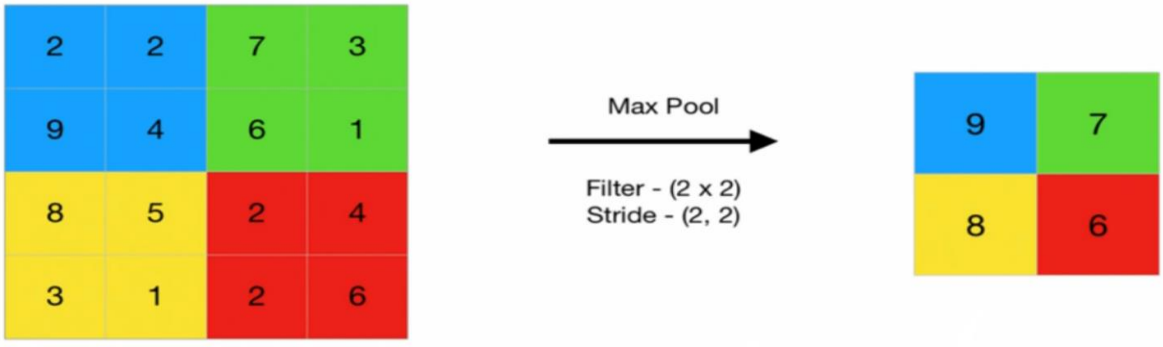

Red
Green
Blue
Filter 3
Filter 2
Filter 1
0+0+2+0+0+0+0+0+0 = 2
2+0+1+0+0+0+1+0+0 = 4
0+0+0+0+0+0+1+1+0 = 2
Convoluted Output
2+4+2+1 = 9
Bias



Average Pooling



Max Pooling



Importing Libraries and Image

```
import numpy as np
from scipy import misc
i = misc.face(gray=True)
```

```
import matplotlib.pyplot as plt
plt.grid(False)
plt.gray()
plt.axis('off')
plt.imshow(i)
plt.show()
```

```
i_transformed = np.copy(i)
size_x = i_transformed.shape[0]
size_y = i_transformed.shape[1]
```

Apply Filter On image

```
#Experiment with different values for fun effects.
#filter = [ [0, 1, 0], [1, -4, 1], [0, 1, 0]]

# A couple more filters to try for fun!
#filter = [ [-1, -2, -1], [0, 0, 0], [1, 2, 1]]
#filter = [ [-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]

filter = [[1,0,-1],
          [1,0,-1],
          [1,0,-1]]
```

6 x 6

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

Filter

3 x 3

1	0	-1
1	0	-1
1	0	-1

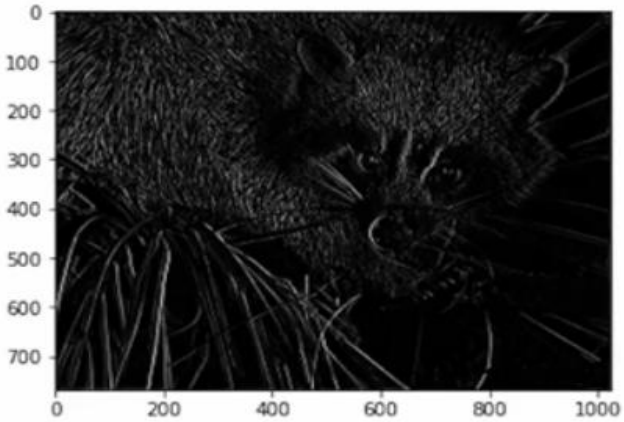
=

4 x 4

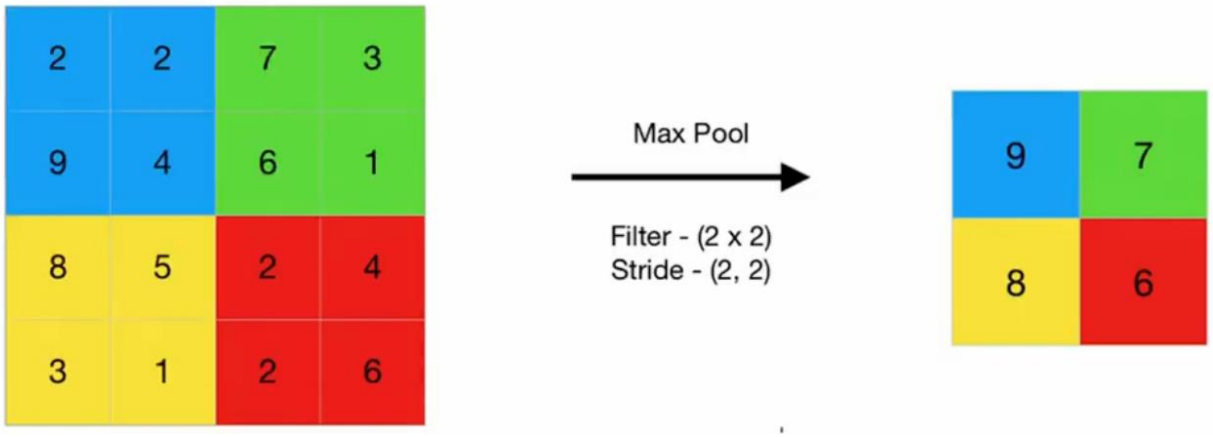
-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

```
for x in range(1,size_x-1):
  for y in range(1,size_y-1):
      convolution = 0.0
      convolution = convolution + (i[x-1, y-1] * filter[0][0])
      convolution = convolution + (i[x-1, y] * filter[0][1])
      convolution = convolution + (i[x-1, y+1] * filter[0][2])
      convolution = convolution + (i[x, y-1] * filter[1][0])
      convolution = convolution + (i[x, y] * filter[1][1])
      convolution = convolution + (i[x, y+1] * filter[1][2])
      convolution = convolution + (i[x+1, y-1] * filter[2][0])
      convolution = convolution + (i[x+1, y] * filter[2][1])
      convolution = convolution + (i[x+1, y+1] * filter[2][2])
      if(convolution<0):
        convolution=0
      if(convolution>255):
        convolution=255
      i_transformed[x, y] = convolution
```

```
[8] # Plot the image. Note the size of the axes -- they are 512 by 512
    plt.gray()
    plt.grid(False)
    plt.imshow(i_transformed)
    #plt.axis('off')
    plt.show()
```

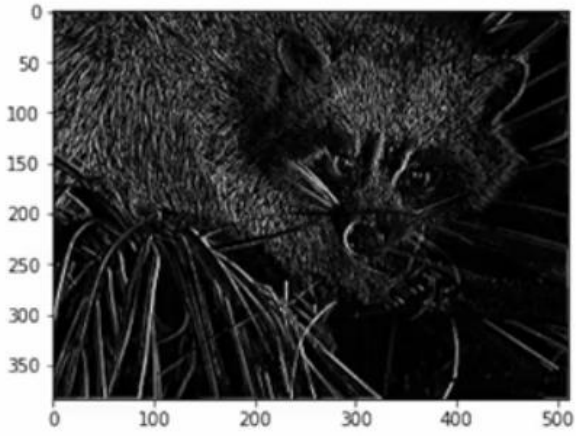


Max Pooling




```
new_x = int(size_x/2)
new_y = int(size_y/2)
newImage = np.zeros((new_x, new_y))
for x in range(0, size_x, 2):
  for y in range(0, size_y, 2):
    pixels = []
    pixels.append(i_transformed[x, y])
    pixels.append(i_transformed[x+1, y])
    pixels.append(i_transformed[x, y+1])
    pixels.append(i_transformed[x+1, y+1])
    newImage[int(x/2),int(y/2)] = max(pixels)

# Plot the image. Note the size of the axes -- now 256 pixels instead of 512
plt.gray()
plt.grid(False)
plt.imshow(newImage)
#plt.axis('off')
plt.show()
```



Recurrent Neural Networks (RNN)

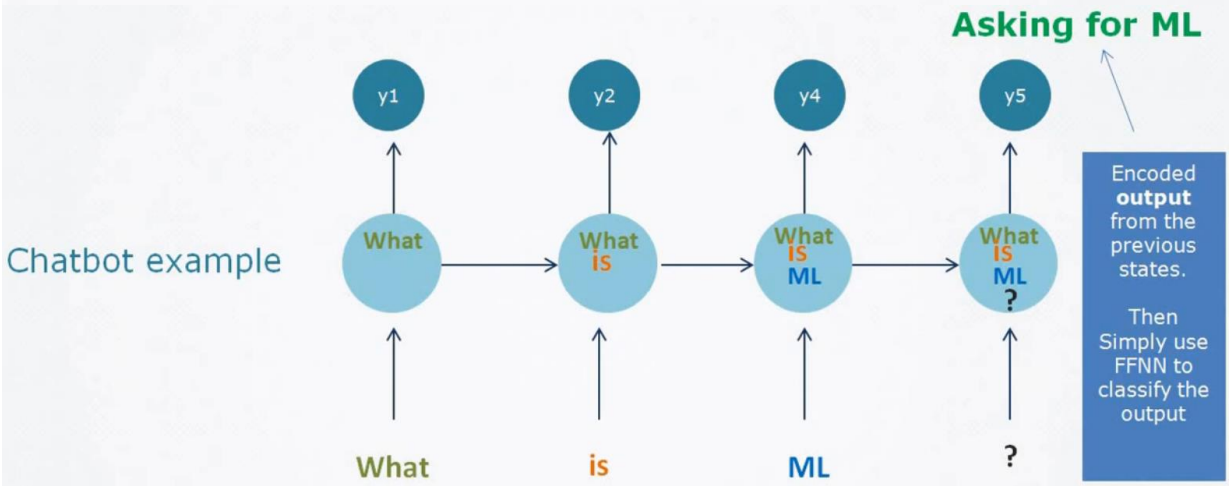
- Recurrent Neural Networks(**RNN**) are a type of Neural Network where the output from the previous step is fed as input to the current step

- *RNNs is that they have a "memory" which captures information about what has been calculated so far.*

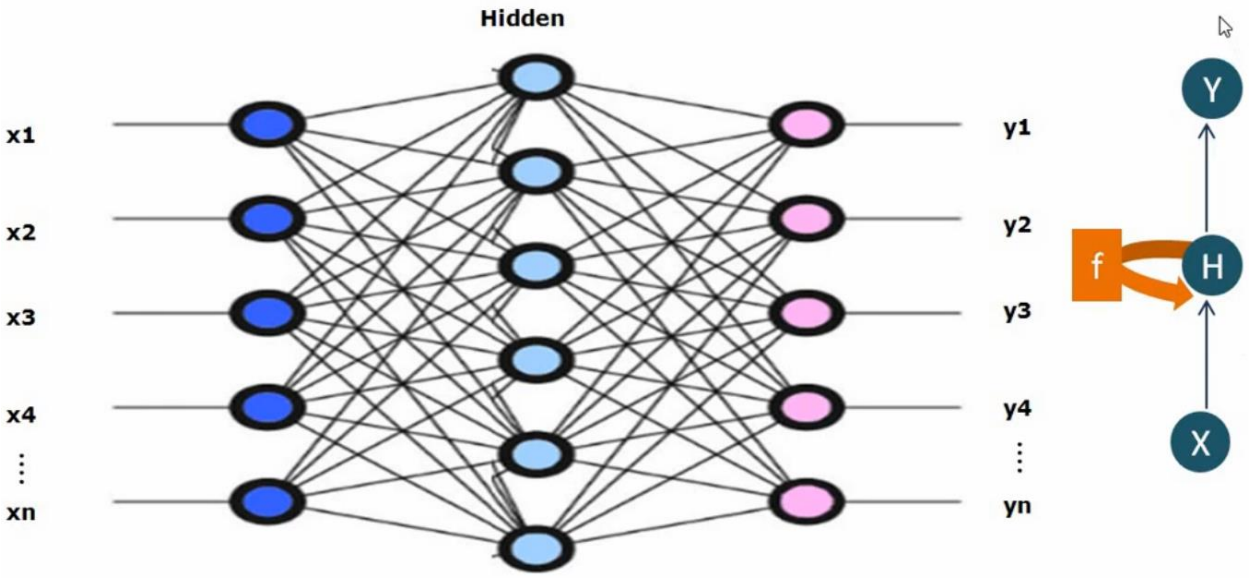
- **RNN's** are mainly used for, Sequence Classification — Sentiment Classification

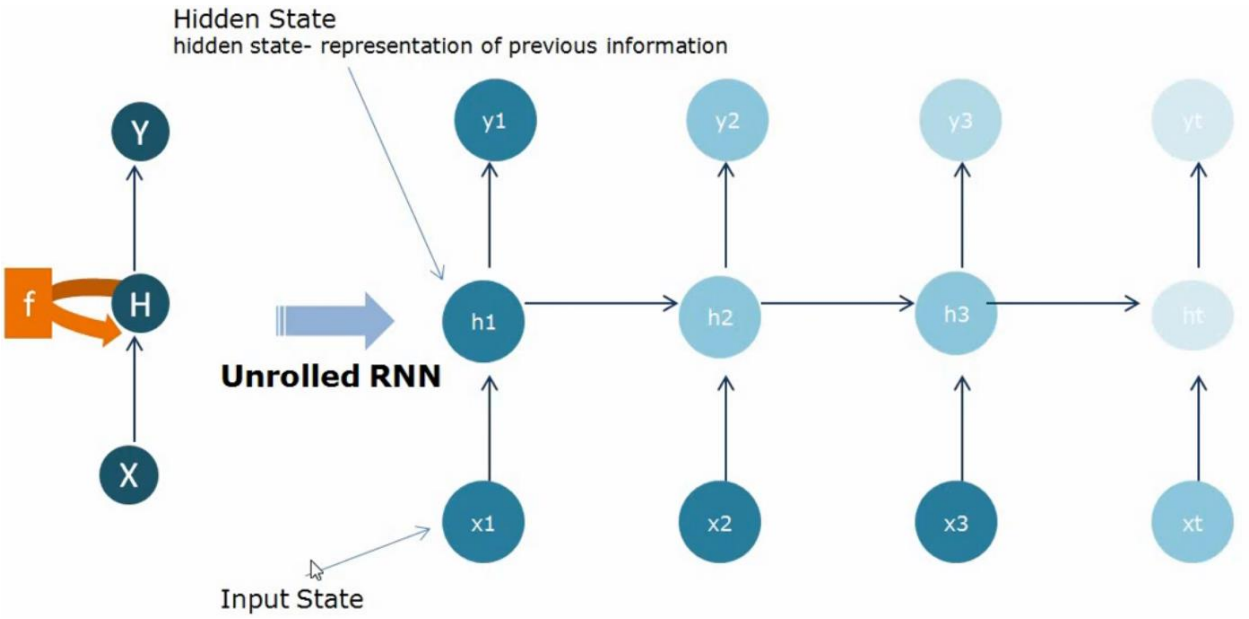
Sequential Data

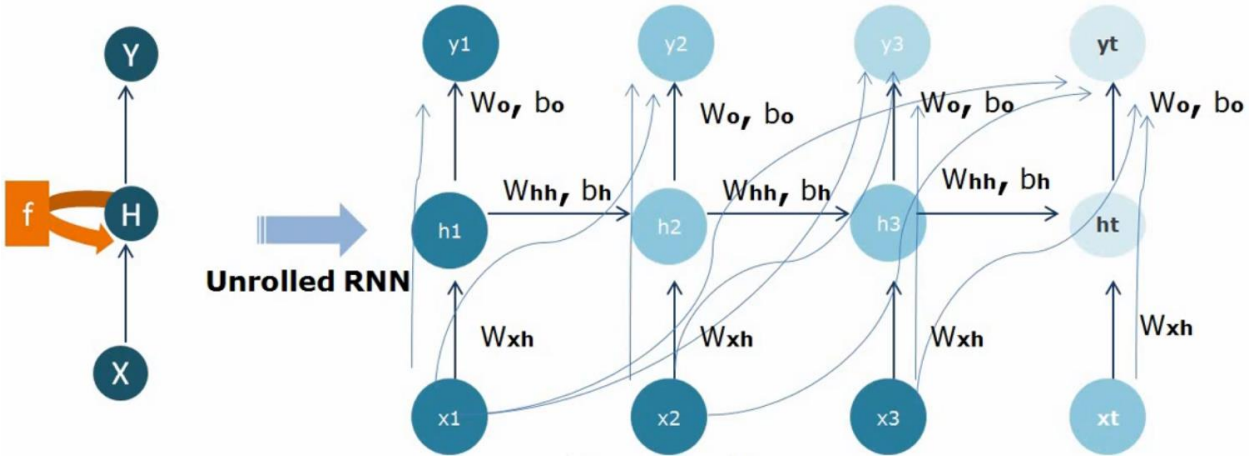
- Audio
- Video
- Text
- Financial Data



Make the hidden state depend on the previous hidden state







tanh activation function

$$h_t = \sigma(W_{xh}^T x_1 + W_{hh}^T h_{t-1} + b_h)$$

$$\hat{y}_t = \sigma(W_0^T h_t + b_0)$$

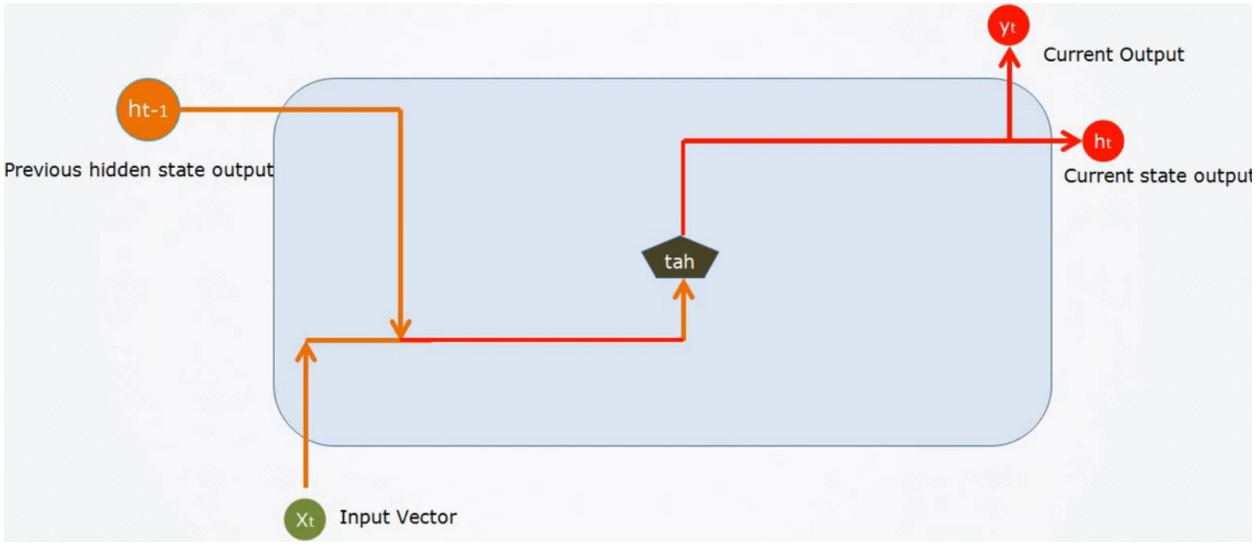
$$h_1 = \sigma(W_{xh}^T x_1 + W_{hh}^T h_0 + b_h)$$

$$\hat{y}_1 = \sigma(W_0^T h_1 + b_0)$$

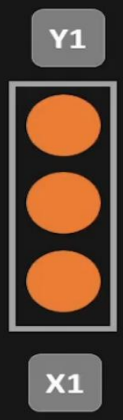
you have to initialize an h0 at the first hidden state.

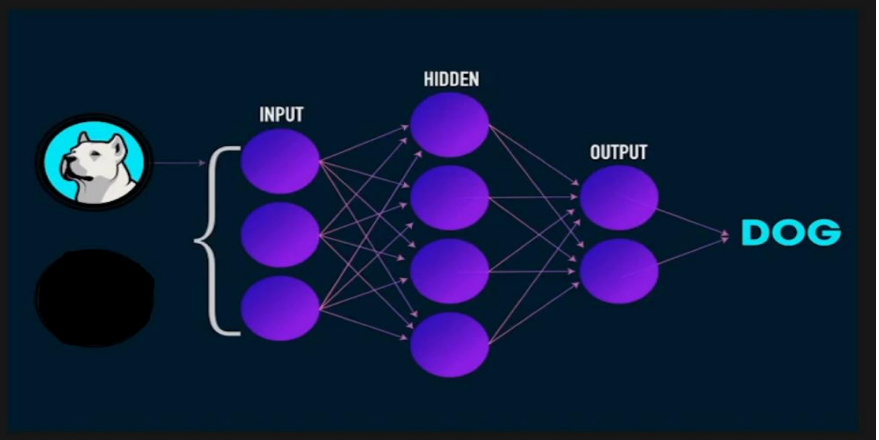
$$h_2 = \sigma(W_{xh}^T x_2 + W_{hh}^T h_1 + b_h)$$

$$\hat{y}_2 = \sigma(W_0^T h_2 + b_0)$$

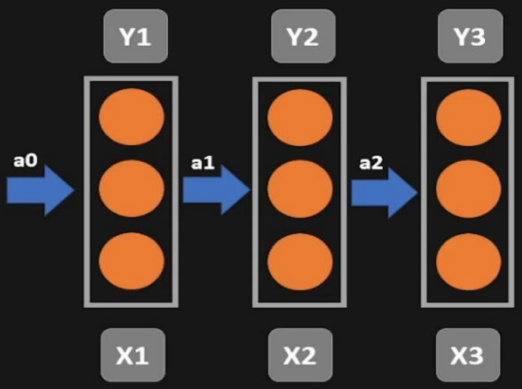


One to One





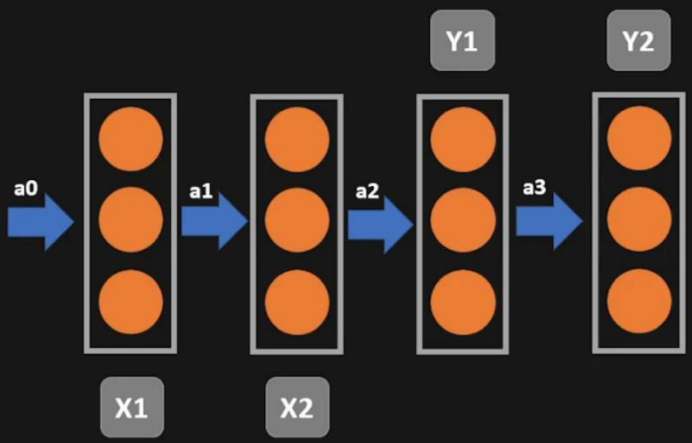
Many to Many



DNA SEQUENCE ANALYSIS

AGCCCCTGTGAG

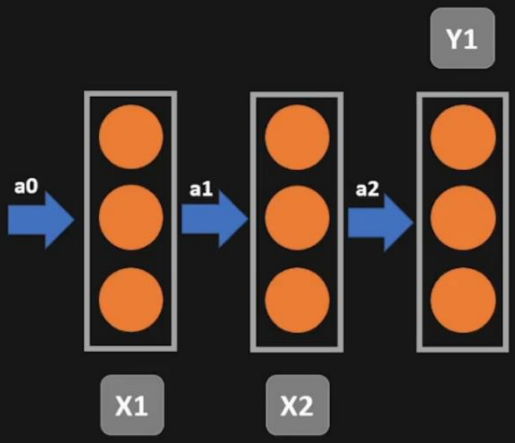
AGCCCCTGTGAG



मैं तुमसे प्यार करता हूँ

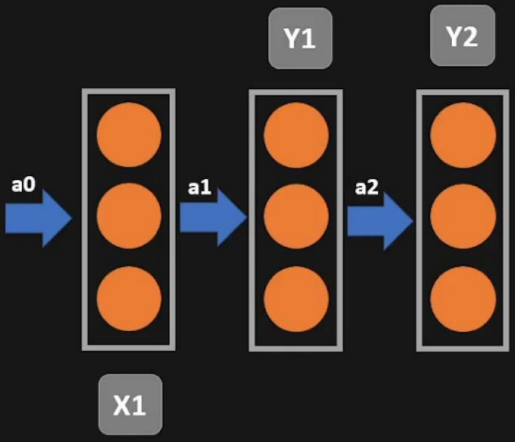
"I LOVE YOU"

Many to One



"I DON'T LIKE IT"

One to Many



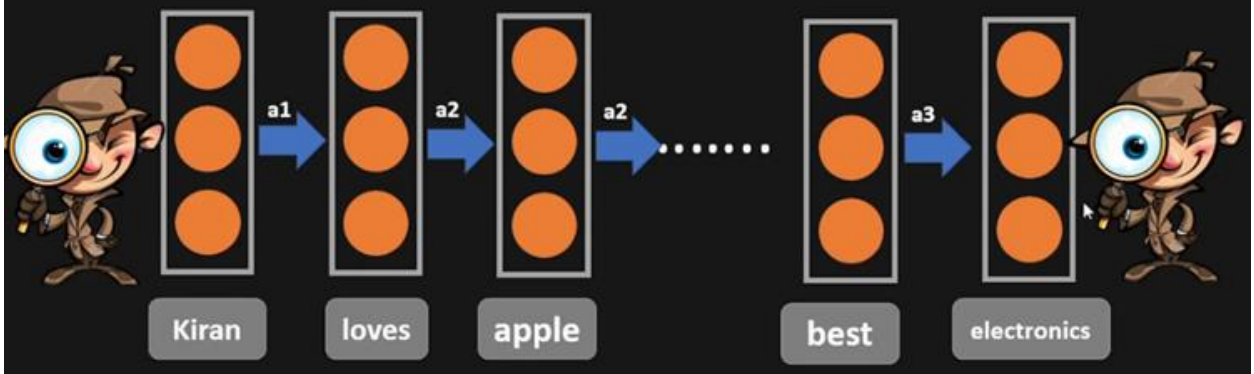
Music Generation

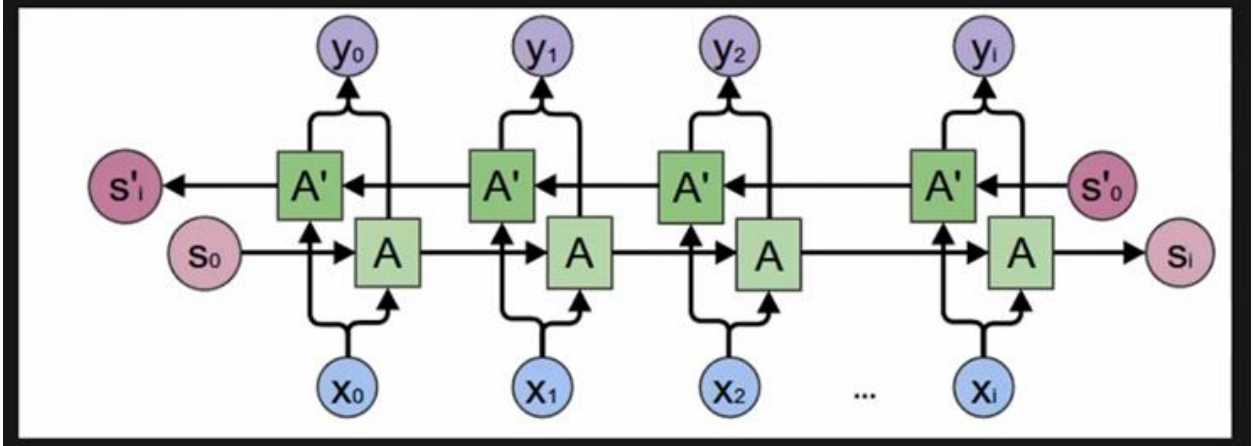
Bidirectional RNN

Name Entity Recognition

Kiran loves **apple**, it keeps him **healthy**

Kiran loves **apple**, the company produces best **electronics**





Long Short-Term Memory (LSTM)

Recurrent Neural Networks work just well with short-term dependencies.
That is when applied to problems like
The colour of parrot is ---

Prediction :
The colour of parrot is **Green**

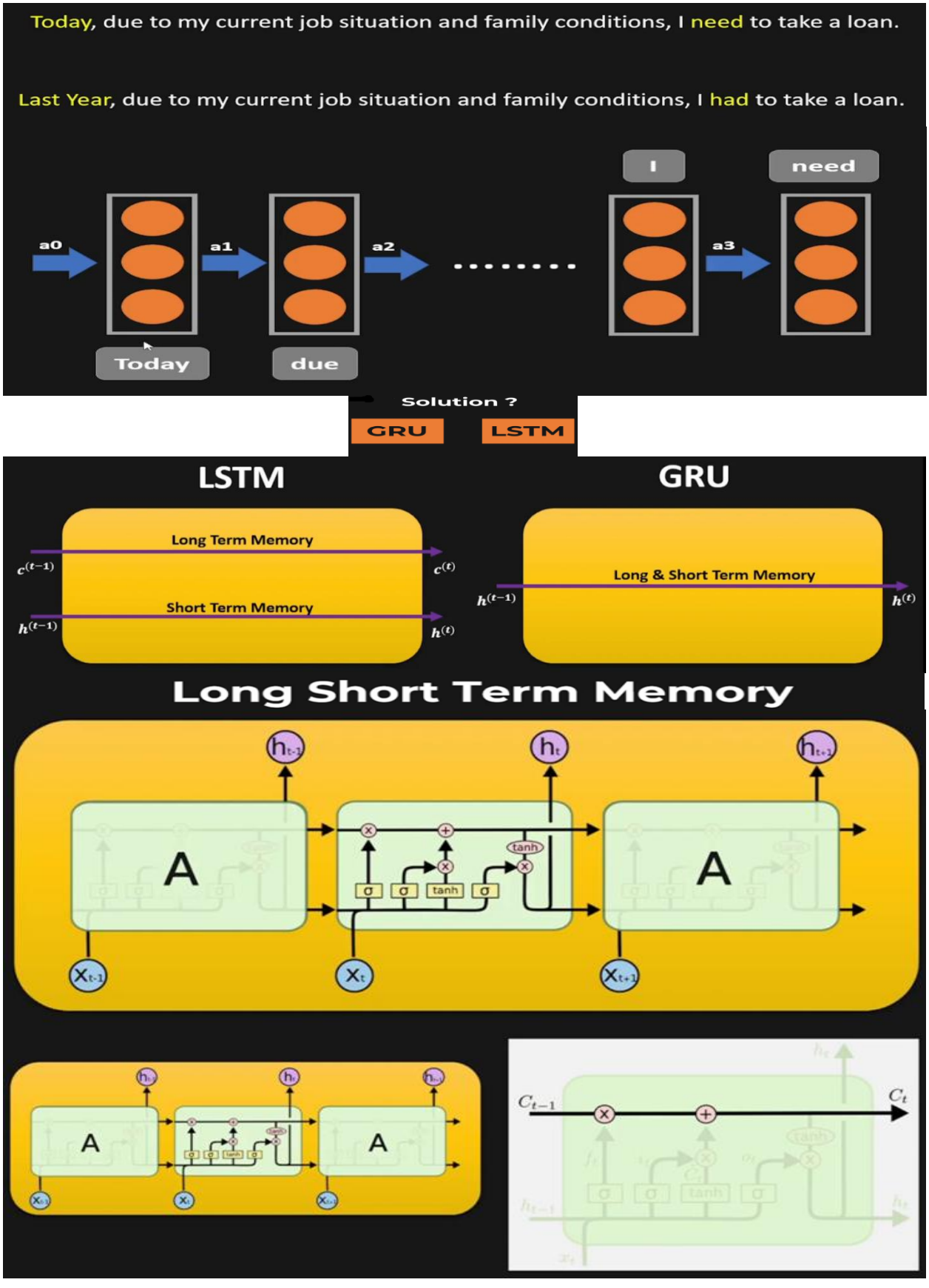
Something that was said long before, cannot be recalled when making predictions in the present.
Example
I have 10 years experience in most reputed IT organizations. Last year I switched the company.
.....
.....
I have strong knowledge in ..... technologies.

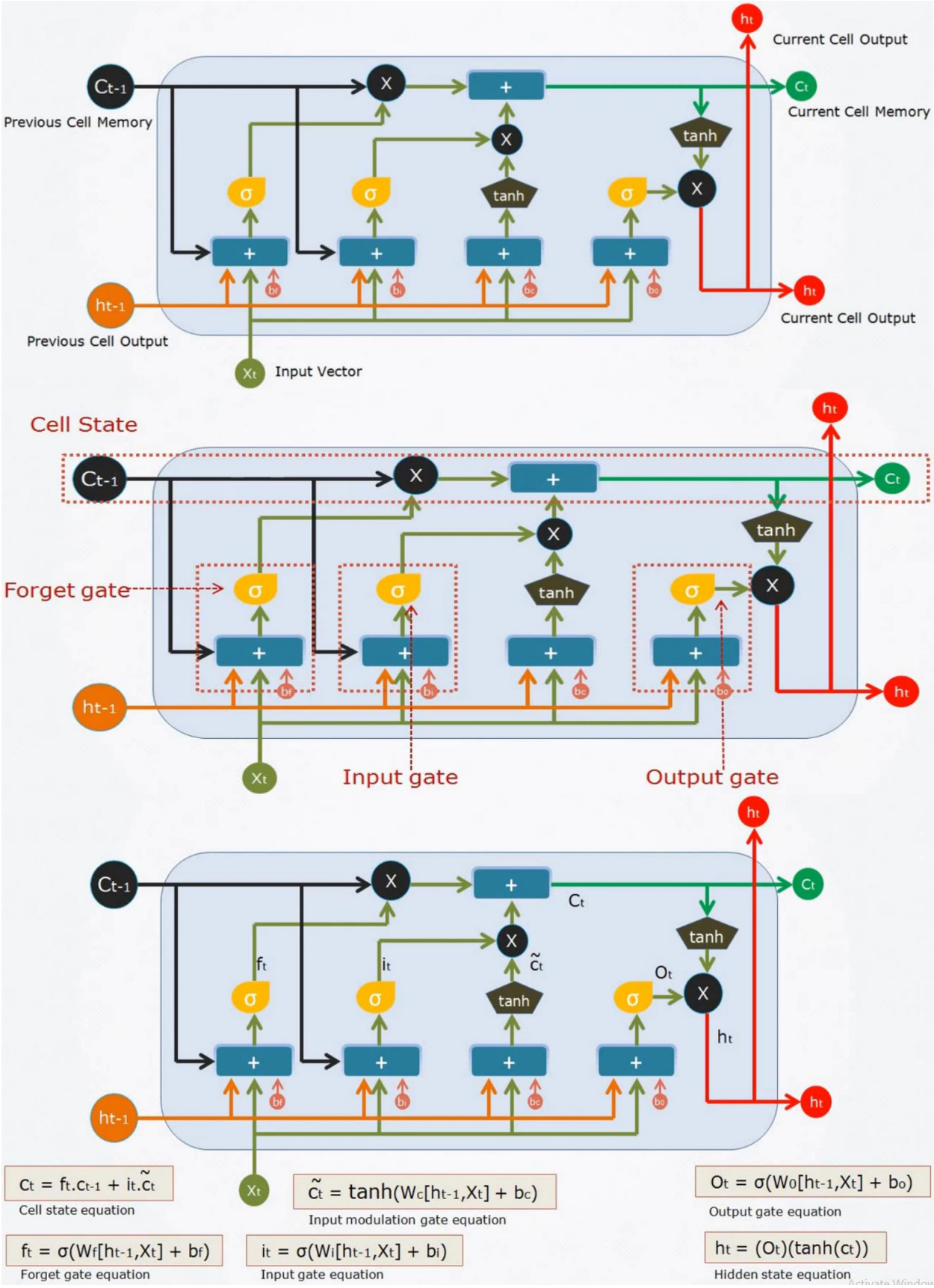
Prediction :
I have strong knowledge in **Microsoft** technologies.

✓RNN suffers from two problems: vanishing gradient and exploding gradient, which make it unusable.

✓LSTM (long short term memory) was invented to solve this issue by explicitly introducing a memory unit, called the cell into the network.

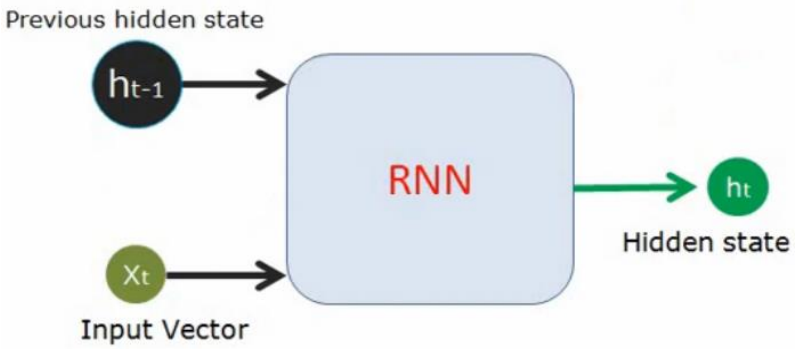
✓This single unit makes decision by considering the current input, previous output and previous memory. And it generates a new output and alters its memory

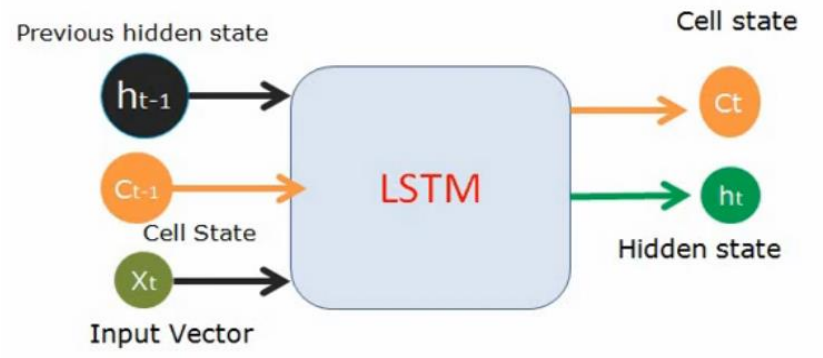

Today, due to my current job situation and family conditions, I need to take a loan.
Last Year, due to my current job situation and family conditions, I had to take a loan.
a0
a1
a2
a3
Today
due
I
need
Solution ?
GRU
LSTM
LSTM
Long Term Memory
Short Term Memory
GRU
Long & Short Term Memory
Long Short Term Memory
A
A
tanh
σ
σ
tanh
σ

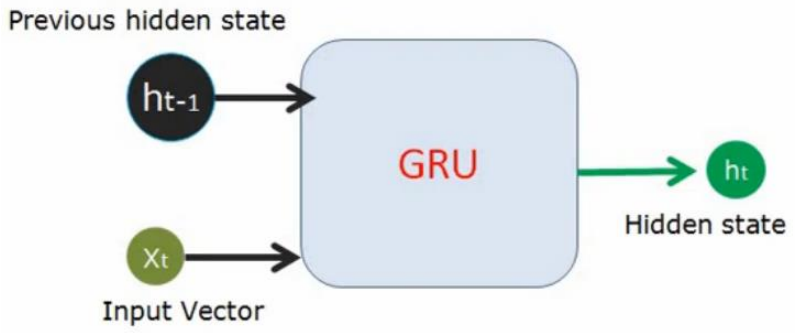

Current Cell Output
Previous Cell Memory
Current Cell Memory
Current Cell Output
Previous Cell Output
Input Vector
Cell State
Forget gate
Input gate
Output gate
Cell state equation
Input modulation gate equation
Output gate equation
Forget gate equation
Input gate equation
Hidden state equation

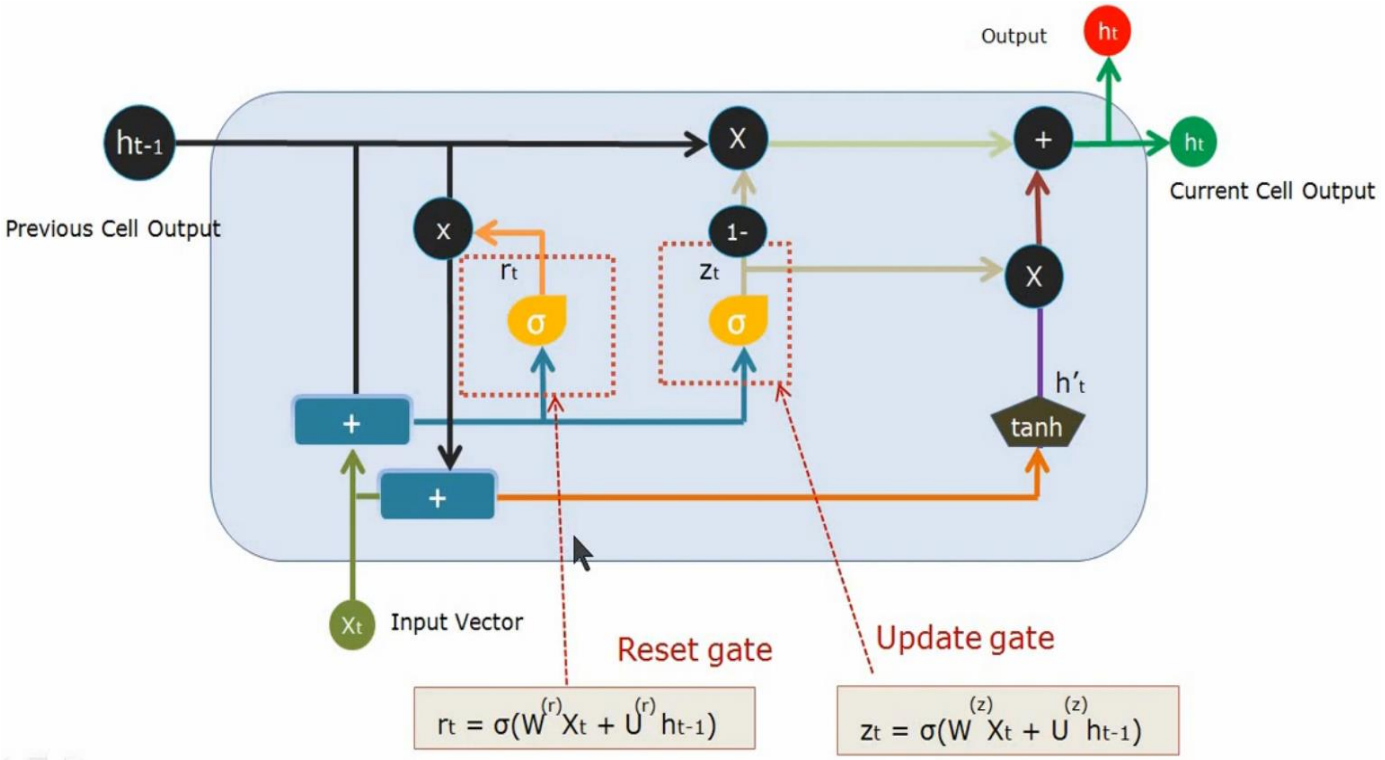
GRU

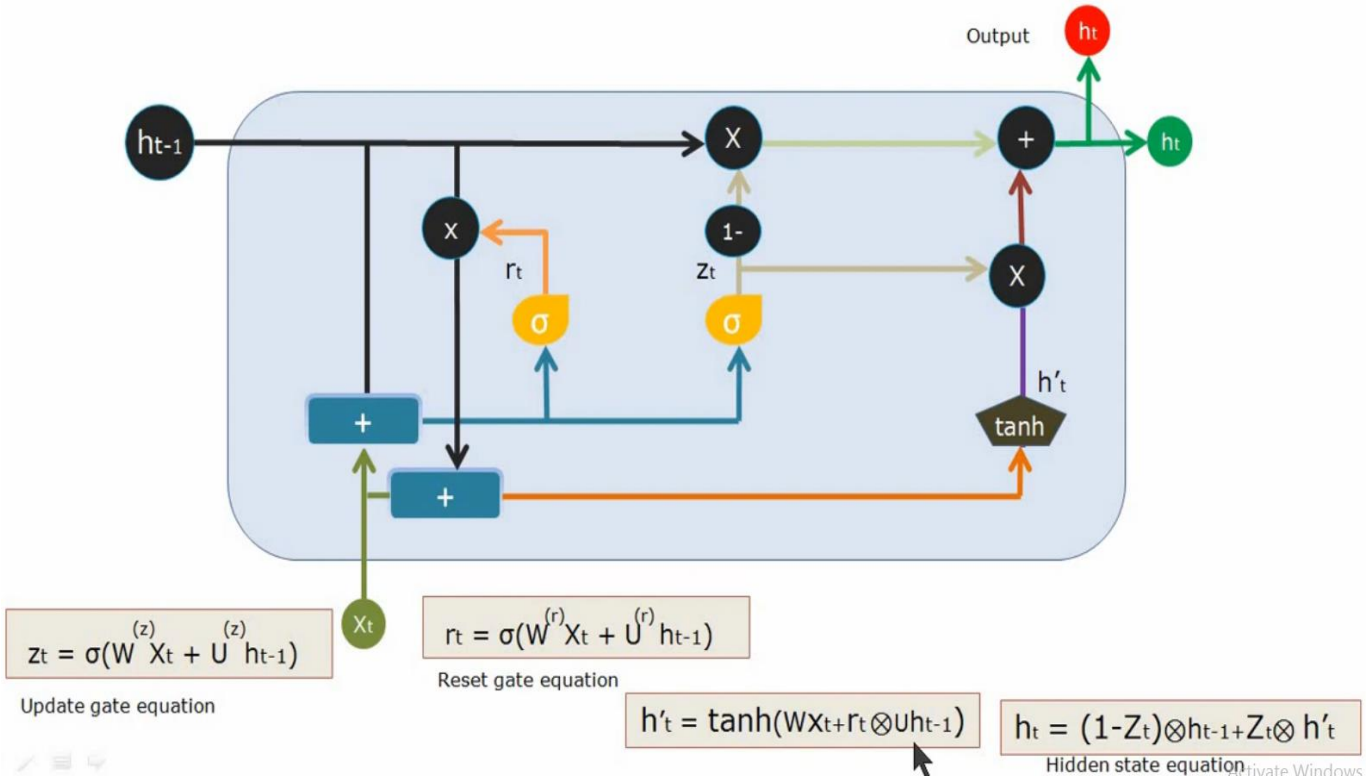
- ✓ Very similar to LSTM
- ✓ It merges the cell state and hidden state.
- ✓ It combines the forget and input gates into a single "update gate".
- ✓ Computationally more efficient.
  - less parameters, less complex structure





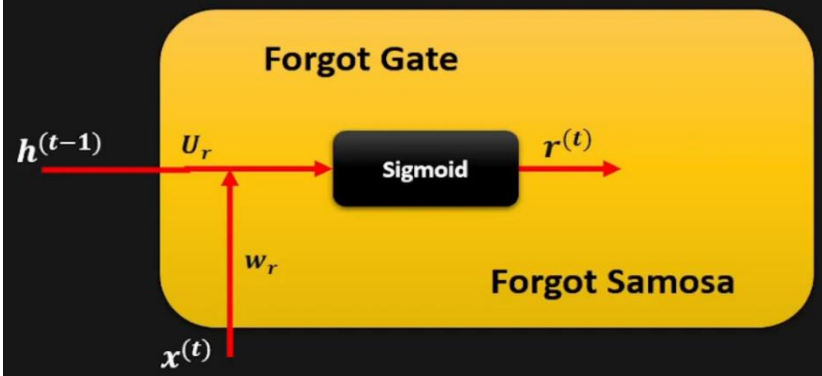



Output
ht
ht-1
Previous Cell Output
X
+
ht
Current Cell Output
x
1-
rt
zt
σ
σ
X
h't
tanh
+
+
Xt
Input Vector
Reset gate
Update gate
rt = σ(W(r) Xt + U(r) ht-1)
zt = σ(W(z) Xt + U(z) ht-1)

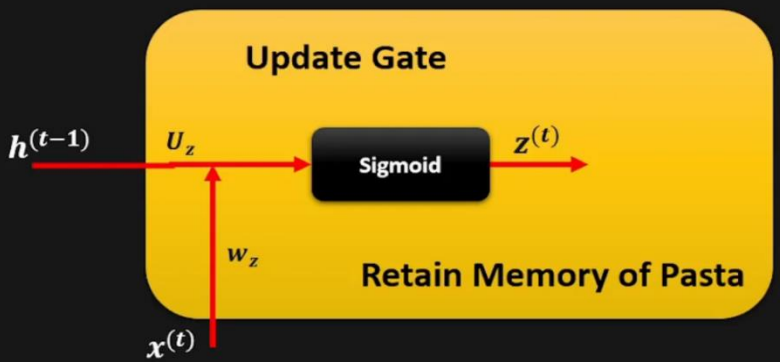

Output
ht
ht-1
X
+
ht
x
1-
rt
zt
σ
σ
X
h't
tanh
+
+
Xt
zt = σ(W(z) Xt + U(z) ht-1)
Update gate equation
rt = σ(W(r) Xt + U(r) ht-1)
Reset gate equation
h't = tanh(WXt+rt⊗Uht-1)
ht = (1-Zt)⊗ht-1+Zt⊗ h't
Hidden state equation

Gated Recurrent Units

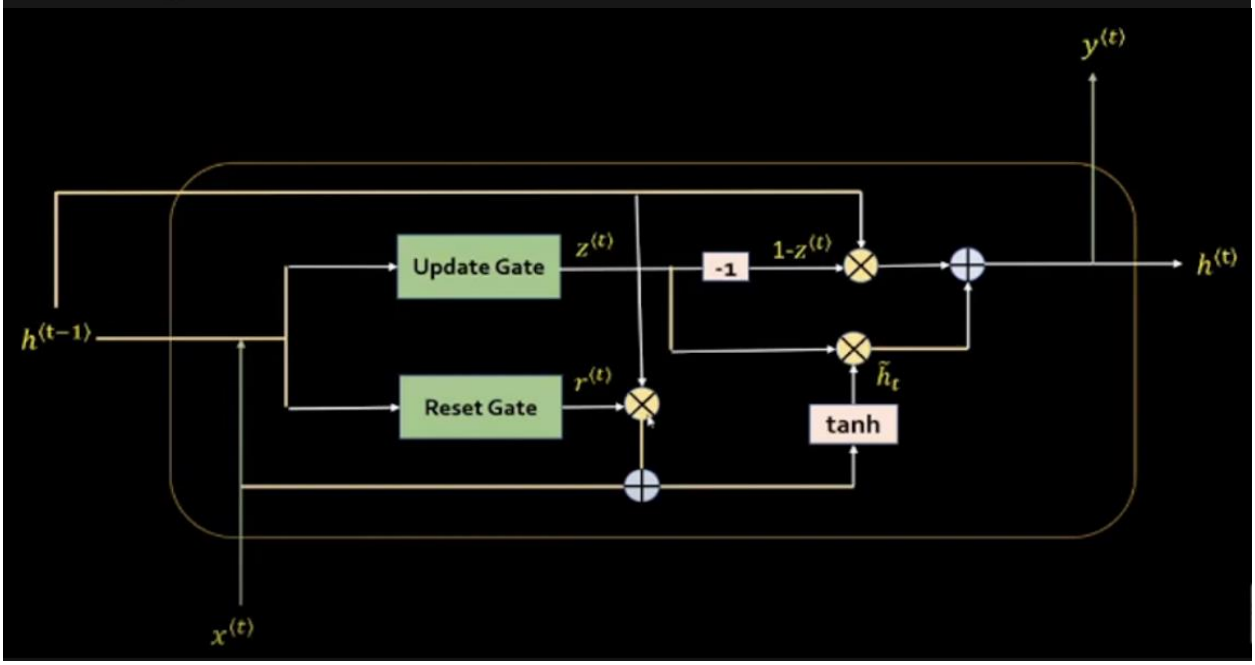
Kiran eats samosa almost everyday, it shouldn't be hard to guess that his favorite cuisine Is Indian. His brother Bhavin however is a lover of pasta and cheese that means Bhavin's favorite cuisine is Italian.



$$r^{(t)} = \sigma\left(w_r x^t + U_r h^{(t-1)}\right)$$



$$z^{(t)} = \sigma\left(w_z x^t + U_z h^{(t-1)}\right)$$



```
# First, we get the data
dataset = pd.read_csv('IBM_2006-01-01_to_2018-01-01.csv', index_col='Date', parse_dates=['Date'])
dataset.head()
```

Date	Open	High	Low	Close	Volume	Name
2006-01-03	82.45	82.55	80.81	82.06	11715200	IBM
2006-01-04	82.20	82.50	81.33	81.95	9840600	IBM
2006-01-05	81.40	82.90	81.00	82.50	7213500	IBM
2006-01-06	83.95	85.03	83.41	84.95	8197400	IBM
2006-01-09	84.10	84.25	83.38	83.73	6858200	IBM

```
# Checking for missing values
training_set = dataset['2016'].iloc[:,1:2].values
test_set = dataset['2017':].iloc[:,1:2].values
```

```
# We have chosen 'High' attribute for prices. Let's see what it looks like
dataset["High"][:'2016'].plot(figsize=(16,4),legend=True)
dataset["High"]['2017':].plot(figsize=(16,4),legend=True)
plt.legend(['Training set (Before 2017)','Test set (2017 and beyond)'])
plt.title('IBM stock price')
plt.show()
```


```
# Scaling the training set
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
```

```
# Since GRU store long term memory state, we create a data structure with 60 timesteps and 1 output
# So for each element of training set, we have 60 previous training set elements
X_train = []
y_train = []
for i in range(60,2769):
    X_train.append(training_set_scaled[i-60:i,0])
    y_train.append(training_set_scaled[i,0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

```
# Reshaping X_train for efficient modelling
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
```

```
# The GRU architecture
modelGRU = Sequential()
# First GRU layer with Dropout regularisation
modelGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
modelGRU.add(Dropout(0.2))
# Second GRU layer
modelGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
modelGRU.add(Dropout(0.2))
# Third GRU layer
modelGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
modelGRU.add(Dropout(0.2))
# Fourth GRU layer
modelGRU.add(GRU(units=50, activation='tanh'))
modelGRU.add(Dropout(0.2))
# The output layer
modelGRU.add(Dense(units=1))
# Compiling the RNN
modelGRU.compile(optimizer=SGD(lr=0.01, decay=1e-7, momentum=0.9, nesterov=False),loss='mean_squared_error')
# Fitting to the training set
modelGRU.fit(X_train,y_train,epochs=50,batch_size=150)
```

```
Epoch 1/50
19/19 [==============================] - 10s 223ms/step - loss: 0.1769
Epoch 2/50
19/19 [==============================] - 4s 227ms/step - loss: 0.0647
Epoch 3/50
19/19 [==============================] - 4s 230ms/step - loss: 0.0276
Epoch 48/50
19/19 [==============================] - 5s 239ms/step - loss: 0.0022
Epoch 49/50
19/19 [==============================] - 5s 239ms/step - loss: 0.0021
Epoch 50/50
19/19 [==============================] - 5s 244ms/step - loss: 0.0021
<tensorflow.python.keras.callbacks.History at 0x7f444e914898>
```

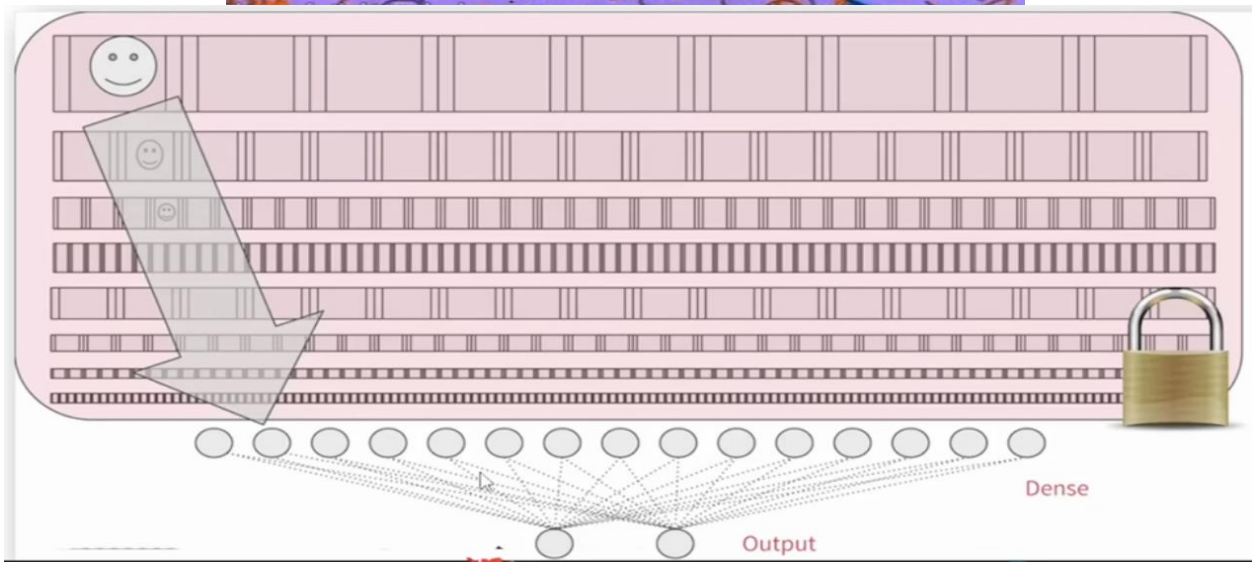
```
# Preparing X_test and predicting the prices
X_test = []
for i in range(60,311):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
GRU_predicted_stock_price = modelGRU.predict(X_test)
GRU_predicted_stock_price = sc.inverse_transform(GRU_predicted_stock_price)
```

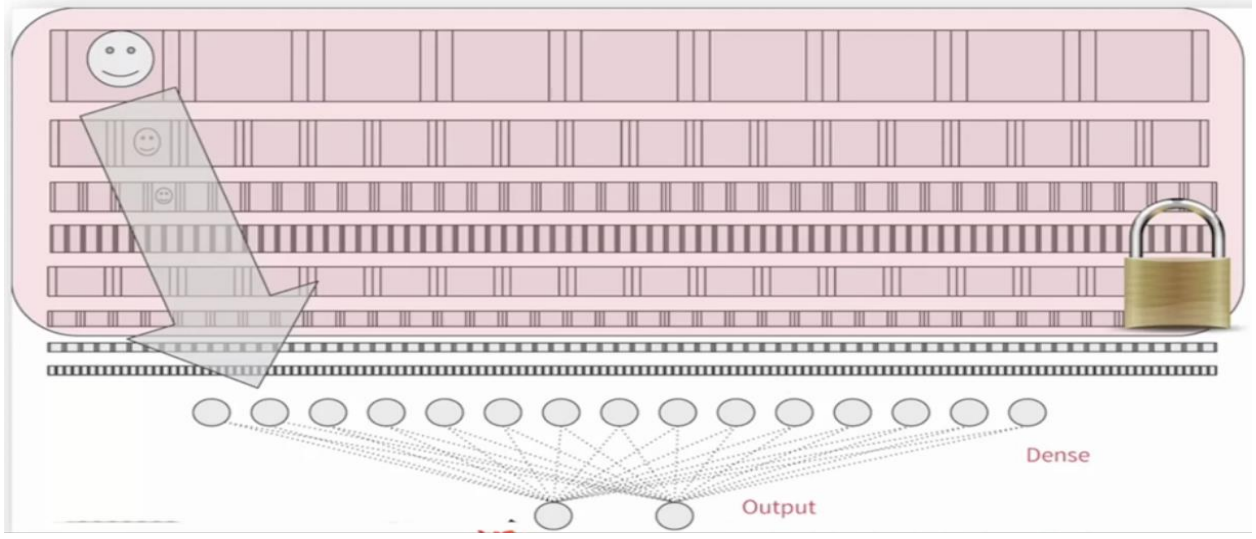
```
# Visualizing the results for LSTM
plot_predictions(test_set,predicted_stock_price)
```


```
# Evaluating our model
return_rmse(test_set,predicted_stock_price)
```

```
The root mean squared error is 2.592514562101024.
```

Transfer Learning





TensorFlow Version

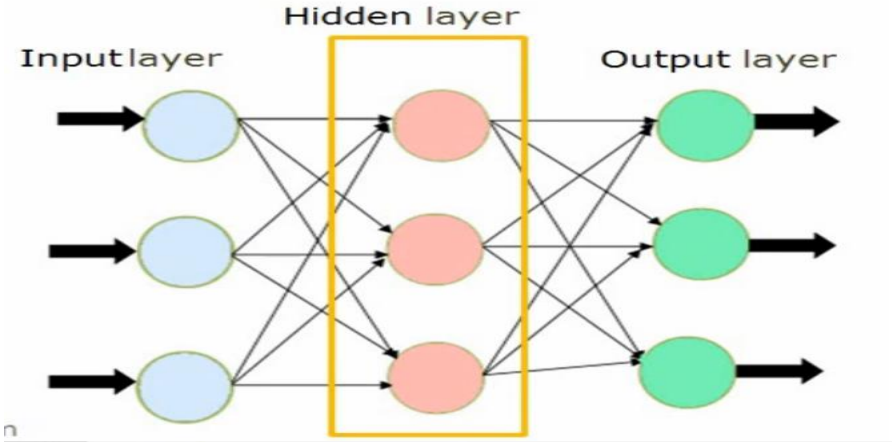
```
import tensorflow as tf

tf.__version__
```

```
'2.3.0'
```

Keras layers API

Layers are the basic building blocks of neural networks in Keras




```
import tensorflow as tf
from tensorflow.keras import layers

#creating layer
layer = layers.Dense(4)

inputs = tf.random.uniform(shape=(5, 10))
outputs = layer(inputs)

outputs
```

```
<tf.Tensor: shape=(5, 4), dtype=float32, numpy=
array([[ 0.23317587, -0.5723395 , -0.5922195 , -0.15031755],
       [ 0.42952067, -1.0333114 ,  0.36681855,  0.4331702 ],
       [ 0.02772742, -0.38887852, -0.36778417, -0.27738553],
       [ 0.14641379, -0.73764247,  0.25093132,  0.03845632],
       [ 0.8514145 , -1.3955642 , -0.01848543,  0.46734422]],
      dtype=float32)>
```

Dense Layer

Dense implements the operation:

model.add(layers.Dense ())

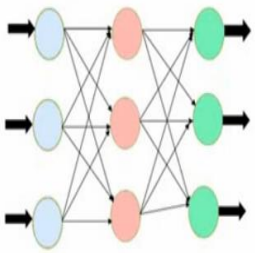
output = activation(dot(input, kernel) + bias)

Y = σ(W.X + B)

activation - activation function
kernel is a weights matrix created by the layer
bias is a bias vector created by the layer (only applicable if use_biasis True).

Dense Layer

```
tf.keras.layers.Dense
(
 units,
 activation=None,
 use_bias=True,
 kernel_initializer="glorot_uniform",
 bias_initializer="zeros",
 kernel_regularizer=None,
 bias_regularizer=None,
 activity_regularizer=None,
 kernel_constraint=None,
 bias_constraint=None,
 **kwargs
)
```



units: Positive integer, dimensionality of the output space. (Number of Neurons)
activation: Activation function to use. If you don't specify anything, no activation is applied
(ie. "linear" activation: a(x) = x).
use_bias: Boolean, whether the layer uses a bias vector.
kernel_initializer: Initializer for the kernel weights matrix.
bias_initializer: Initializer for the bias vector.
kernel_regularizer: Regularizer function applied to the kernel weights matrix.
bias_regularizer: Regularizer function applied to the bias vector.
activity_regularizer: Regularizer function applied to the output of the layer (its "activation").
kernel_constraint: Constraint function applied to the kernel weights matrix.
bias_constraint: Constraint function applied to the bias vector.

Flatten Layer

Flattens the input. Does not affect the batch size.

```
model.add(layers.Flatten())
```

Flatten layer converts the pooled vector to a single column
If inputs are shaped (batch,) without a channel dimension, then flattening adds an extra channel dimension and output shapes are (batch, 1).

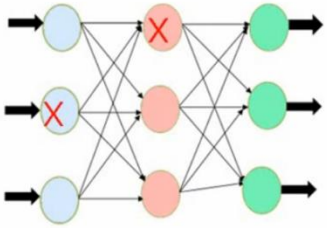
$$\begin{bmatrix} 1 & 2 & 4 \\ 4 & 6 & 7 \\ 5 & 5 & 6 \end{bmatrix} \xrightarrow{\text{Flatten()}} \begin{bmatrix} 1 \\ 2 \\ 4 \\ 4 \\ 6 \\ 7 \\ 5 \\ 5 \\ 6 \end{bmatrix}$$

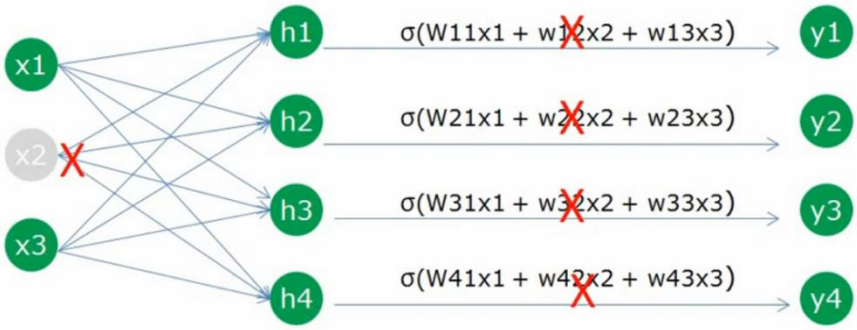
Dropout Layer

```
model.add(layers.Dropout (rate, noise,_shape=None))
```

rate - Float between 0 and 1. Fraction of the input units to drop.
Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly.
Use this layer to avoid model overfitting.
Dropout is a technique to regularize in neural networks. When we drop certain nodes out, these units are not considered during a particular forward or backward pass in a network

$$\begin{bmatrix} 1 & 2 & 4 \\ 4 & 6 & 7 \\ 5 & 5 & 6 \end{bmatrix} \xrightarrow{\text{Dropout(0.3)}} \begin{bmatrix} 1 & 0 & 4 \\ 0 & 6 & 7 \\ 5 & 5 & 6 \end{bmatrix}$$





Multiple layers

```
# Create 3 layers
layer1 = layers.Dense(2, activation="relu", name="layer1")
layer2 = layers.Dense(3, activation="relu", name="layer2")
layer3 = layers.Dense(4, name="layer3") # Call layers on a
test input x = tf.ones((3, 3))
y = layer3(layer2(layer1(x)))
```

```
# Define Sequential model with 3 layers
model = keras.Sequential
(
  [
     layers.Dense(2, activation="relu", name="layer1"),
     layers.Dense(3, activation="relu", name="layer2"),
     layers.Dense(4, name="layer3"),
  ]
)

x = tf.ones((3, 3))
y = model(x)
```

```
# Define Sequential model with 3 layers
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
x = tf.ones((3, 3))
y = model(x)
```

Specifying the input shape in advance

```
# Define Sequential model with 1 layer
model = keras.Sequential()
model.add(keras.Input(shape=(4,)))
model.add(layers.Dense(2, activation="relu"))
model.summary()
```

```
# Define Sequential model with 1 layer
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu", input_shape=(4,)))
model.summary()
```

Passing an Input object to your model, so that it knows its input shape from the start while creating model.

Note that the Input object is not displayed as part of model.layers, since it isn't a layer:

TensorFlow weights visualization

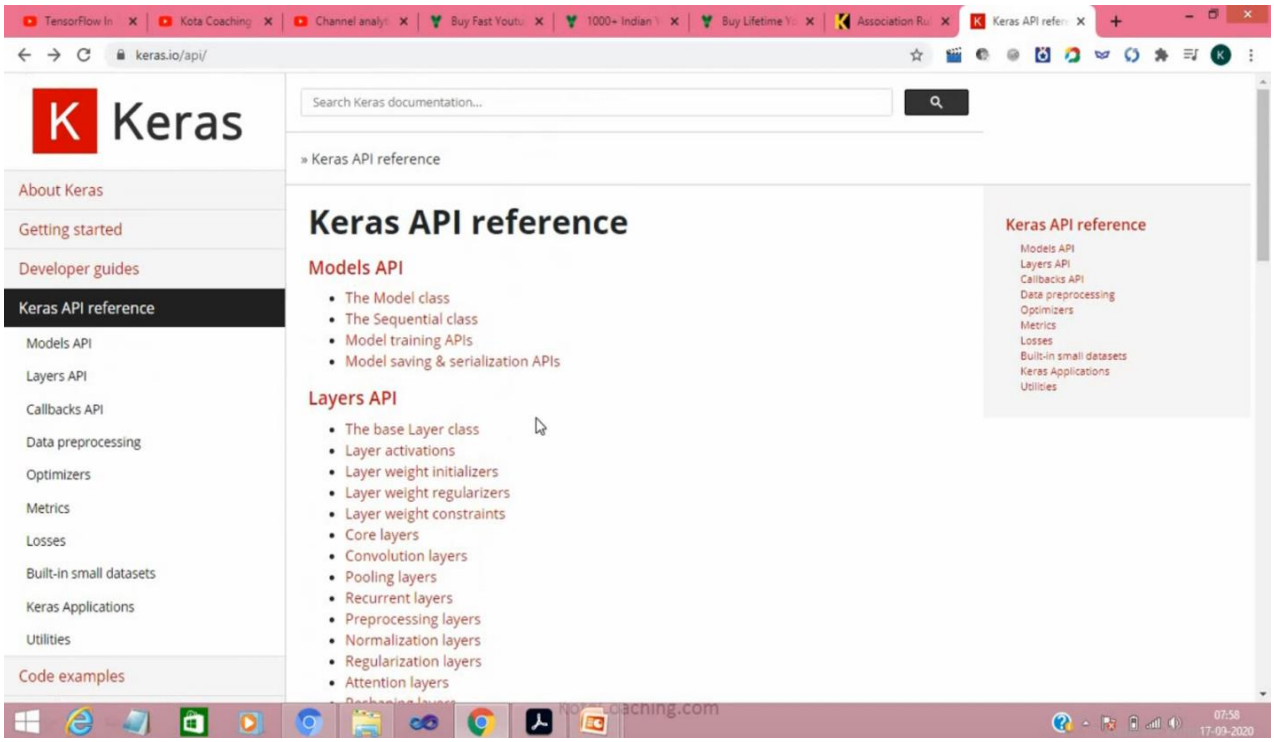
```
from tensorflow.keras import layers
import tensorflow as tf
model =  tf.keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(4))
x = tf.ones((3,3))
y = model(x)
model.weights
```

```
[<tf.Variable 'dense_30/kernel:0' shape=(3, 2) dtype=float32, numpy=
 array([[ 0.11687744, -0.7544714 ],
        [ 0.51128435, -0.55170286],
        [ 0.4792596 , -0.47417873]], dtype=float32)>,
 <tf.Variable 'dense_30/bias:0' shape=(2,) dtype=float32, numpy=array([0., 0.], dtype=float32)>,
 <tf.Variable 'dense_31/kernel:0' shape=(2, 4) dtype=float32, numpy=
 array([[ 0.8916414 ,  0.47133827,  0.43887568,  0.56327534],
        [ 0.89368916, -0.7732425 , -0.7410698 ,  0.32620478]],
       dtype=float32)>,
 <tf.Variable 'dense_31/bias:0' shape=(4,) dtype=float32, numpy=array([0., 0., 0., 0.], dtype=float32)>]
```

TensorFlow model summary

```
from tensorflow.keras import layers
import tensorflow as tf
model =  tf.keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(4))
x = tf.ones((3,3))
y = model(x)
model.summary()
```

```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_49 (Dense)             (3, 2)                    8
_________________________________________________________________
dense_50 (Dense)             (3, 4)                    12
=================================================================
Total params: 20
Trainable params: 20
Non-trainable params: 0
_________________________________________________________________
```


keras.io/api/
Keras
Search Keras documentation...
About Keras
Getting started
Developer guides
Keras API reference
Models API
Layers API
Callbacks API
Data preprocessing
Optimizers
Metrics
Losses
Built-in small datasets
Keras Applications
Utilities
Code examples
» Keras API reference
Keras API reference
Models API
The Model class
The Sequential class
Model training APIs
Model saving & serialization APIs
Layers API
The base Layer class
Layer activations
Layer weight initializers
Layer weight regularizers
Layer weight constraints
Core layers
Convolution layers
Pooling layers
Recurrent layers
Preprocessing layers
Normalization layers
Regularization layers
Attention layers
Keras API reference
Models API
Layers API
Callbacks API
Data preprocessing
Optimizers
Metrics
Losses
Built-in small datasets
Keras Applications
Utilities
07:58
17-09-2020

Why do we need to save model?

```
from pandas import read_csv
# load the dataset
df = read_csv(= 'https://rawexample.com/output.csv', header=None)

# split into input and output columns
X, y = df.values[:, :-1], df.values[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

# determine the number of input features
n_features = X_train.shape[1]

# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal',
input_shape=(n_features,)))
model.add(Dense(1, activation='sigmoid'))

# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# fit the model
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=0)

# make a prediction
PreedictedValue = model.predict(= [1,0,0.99539,0.18641,-0.45300])
print('Predicted: %.3f' % PreedictedValue)
```

Scikit-Learn Machine Learning Models

Save The Model

```
from sklearn.linear_model import LogisticRegression
import pickle
model = LogisticRegression()
model.fit(xtrain, ytrain)

# save the model to disk
pickle.dump(model, open("modelName.pkl", 'wb'))
```

Load The Model

```
model = pickle.load(open("modelName.pkl", 'rb'))

Result = model.predict(InputValue)
```

Use **Pickle** to serialise and save the models

Save The Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.externals import joblib
model = LogisticRegression()
model.fit(xtrain, ytrain)

# save the model to disk
joblib.dump(model, "modelName.pkl")
```

Load The Model

```
model = joblib.load ("modelName.pkl")

Result = model.predict(InputValue)
```

Use **joblib** to serialise and save the models

Saving model architecture by two ways

Save Model to JSON.

Save Model to YAML.

Save The Model

```
from keras.models import Sequential
from keras.layers import Dense# create model

model = Sequential()
# Fit the model
model.fit(xtrain, ytrain)

# serialize to JSON
json_file = model.to_json()
with open ("modelName.json", "w") as file:
file.write(json_file)
# serialize weights to HDF5
model.save_weights("modelName.h5")
```

Load The Model

```
from keras.models import model_from_json

# load json and create model
file = open(("modelName.json", 'r')
loaded_model_json = file.read()
file.close()

model = model_from_json(loaded_model_json)

# load weights
model.load_weights ("modelName.h5")

Result = model.predict(InputValue)
```

Save The Model

```
from keras.models import Sequential
from keras.layers import Dense# create model

model = Sequential()
# Fit the model
model.fit(xtrain, ytrain)

# serialize to JSON
json_file = model.to_yaml ()
with open("modelName.yaml", "w") as file:
file.write(json_file)

# serialize weights to HDF5
model.save_weights("modelName.h5")
```

Load The Model

```
from keras.models import model_from_yaml

# load yaml and create model
file = open("modelName.yaml", 'r')
loaded_model_yaml = file.read()
file.close()

model = model_from_yaml(loaded_model_yaml)

# load weights
model.load_weights("modelName.h5")

Result = model.predict(InputValue)
```

Save The Model

```
from keras.models import Sequential
from keras.layers import Dense# create model

model = Sequential()

# Fit the model
model.fit(xtrain, ytrain)

# serialize weights to HDF5
model.save ("modelName.h5")
```

Load The Model

```
from keras.models import load_model

# load model
model = load_model("modelName.h5")

Result = model.predict(InputValue)
```