Name: Season Shrestha Student ID: 1929904

Workshop 8: Linear Algebra basics, Matrix Manupulation

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import math
```

In [2]:

```
#Create matrices using numpy array
#implement addition,subtraction, multiplication
#Explain the rule regarding matrix multiplication
x = np.array([ [2,4],[3,6] ])
y = np.array([ [4,-1],[3,3] ])
z= x+y
z
```

Out[2]:

```
array([[6, 3],
       [6, 9]])
```

In [3]:

```
##Subtraction
z = x-y
z
```

Out[3]:

```
array([[-2,  5],
       [ 0,  3]])
```

In [4]:

```
##Multiplication
z = np.matmul(x,y)
z
```

Out[4]:

```
array([[20, 10],
       [30, 15]])
```

In [5]:

```
x= np.array( [ [2,3,5] , [2,3,-1]])
y = np.array( [ [2,1], [3,5], [5,6]])
z = np.matmul(x,y)
z
```

Out[5]:

```
array([[38, 47],
       [ 8, 11]])
```

Implementation of ACTIVATION FUNCTIONS

In [7]:

```
#Step Function
def step (x):
    if (x<=0):
```

```
        return 0
    else:
        return 1
v_step = np.vectorize(step)
```

In [8]:

```
x = np.linspace(-6,6,100)
x
```

Out[8]:

```
array([-6.        , -5.87878788, -5.75757576, -5.63636364, -5.51515152,
       -5.39393939, -5.27272727, -5.15151515, -5.03030303, -4.90909091,
       -4.78787879, -4.66666667, -4.54545455, -4.42424242, -4.3030303 ,
       -4.18181818, -4.06060606, -3.93939394, -3.81818182, -3.6969697 ,
       -3.57575758, -3.45454545, -3.33333333, -3.21212121, -3.09090909,
       -2.96969697, -2.84848485, -2.72727273, -2.60606061, -2.48484848,
       -2.36363636, -2.24242424, -2.12121212, -2.        , -1.87878788,
       -1.75757576, -1.63636364, -1.51515152, -1.39393939, -1.27272727,
       -1.15151515, -1.03030303, -0.90909091, -0.78787879, -0.66666667,
       -0.54545455, -0.42424242, -0.3030303 , -0.18181818, -0.06060606,
        0.06060606,  0.18181818,  0.3030303 ,  0.42424242,  0.54545455,
        0.66666667,  0.78787879,  0.90909091,  1.03030303,  1.15151515,
        1.27272727,  1.39393939,  1.51515152,  1.63636364,  1.75757576,
        1.87878788,  2.        ,  2.12121212,  2.24242424,  2.36363636,
        2.48484848,  2.60606061,  2.72727273,  2.84848485,  2.96969697,
        3.09090909,  3.21212121,  3.33333333,  3.45454545,  3.57575758,
        3.6969697 ,  3.81818182,  3.93939394,  4.06060606,  4.18181818,
        4.3030303 ,  4.42424242,  4.54545455,  4.66666667,  4.78787879,
        4.90909091,  5.03030303,  5.15151515,  5.27272727,  5.39393939,
        5.51515152,  5.63636364,  5.75757576,  5.87878788,  6.        ])
```

In [9]:

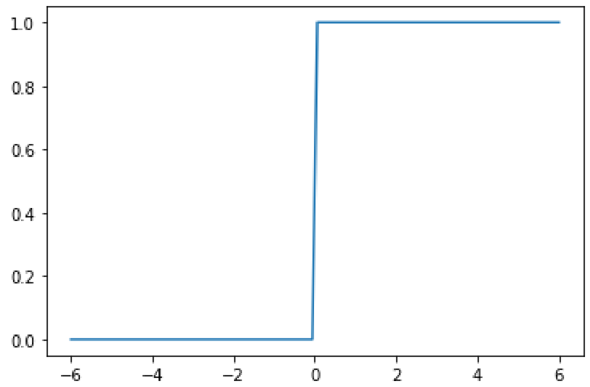
```
y = v_step(x)
y
```

Out[9]:

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

In [10]:

```
plt.plot(x,y)
plt.show()
```



In [11]:

```
#Sigmoid Function
def sigmoid(X):
    return (1/(1+math.exp(-X)))
```

```
v_sigmoid = np.vectorize(sigmoid)
```

In [13]:

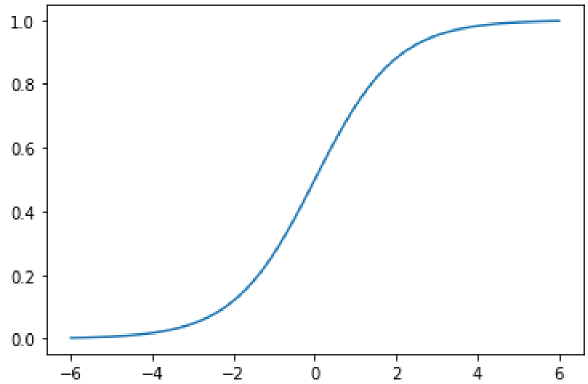
```
x = np.linspace(-6,6,100)
y = v_sigmoid(x)
y
```

Out[13]:

```
array([0.00247262, 0.00279037, 0.00314881, 0.00355314, 0.00400918,
       0.00452348, 0.00510342, 0.00575729, 0.00649438, 0.00732514,
       0.00826129, 0.00931596, 0.01050384, 0.01184139, 0.01334695,
       0.01504103, 0.01694644, 0.01908854, 0.0214955 , 0.02419847,
       0.02723188, 0.03063359, 0.0344452 , 0.03871212, 0.04348381,
       0.04881379, 0.05475969, 0.06138311, 0.06874939, 0.07692721,
       0.08598797, 0.09600494, 0.10705215, 0.11920292, 0.13252816,
       0.14709422, 0.16296047, 0.18017659, 0.1987796 , 0.21879075,
       0.24021244, 0.26302536, 0.2871859 , 0.31262432, 0.33924363,
       0.36691963, 0.39550202, 0.42481687, 0.45467026, 0.48485312,
       0.51514688, 0.54532974, 0.57518313, 0.60449798, 0.63308037,
       0.66075637, 0.68737568, 0.7128141 , 0.73697464, 0.75978756,
       0.78120925, 0.8012204 , 0.81982341, 0.83703953, 0.85290578,
       0.86747184, 0.88079708, 0.89294785, 0.90399506, 0.91401203,
       0.92307279, 0.93125061, 0.93861689, 0.94524031, 0.95118621,
       0.95651619, 0.96128788, 0.9655548 , 0.96936641, 0.97276812,
       0.97580153, 0.9785045 , 0.98091146, 0.98305356, 0.98495897,
       0.98665305, 0.98815861, 0.98949616, 0.99068404, 0.99173871,
       0.99267486, 0.99350562, 0.99424271, 0.99489658, 0.99547652,
       0.99599082, 0.99644686, 0.99685119, 0.99720963, 0.99752738])
```

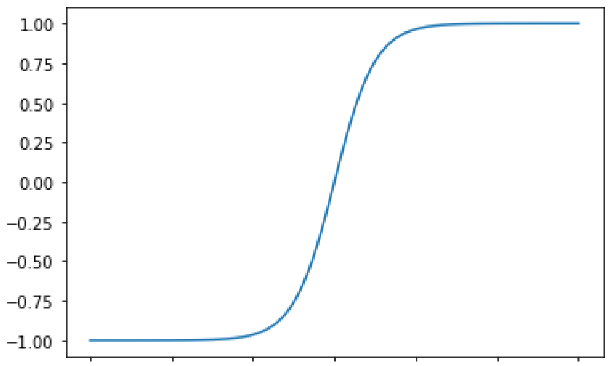
In [14]:

```
plt.plot(x,y)
plt.show()
```



In [15]:

```
#Hyperbolic Tangent
x = np.linspace(-6,6,100)
y = np.tanh(x)
plt.plot(x,y)
plt.show()
```



In [16]:

```
y
```

Out[16]:

```
array([-0.99998771, -0.99998434, -0.99998004, -0.99997457, -0.99996759,
       -0.9999587 , -0.99994738, -0.99993294, -0.99991454, -0.9998911 ,
       -0.99986123, -0.99982316, -0.99977465, -0.99971284, -0.99963408,
       -0.99953372, -0.99940584, -0.9992429 , -0.99903531, -0.99877082,
       -0.99843388, -0.99800466, -0.99745797, -0.99676173, -0.99587519,
       -0.99474658, -0.99331021, -0.99148279, -0.98915888, -0.9862053 ,
       -0.98245414, -0.97769438, -0.97166188, -0.96402758, -0.95438418,
       -0.94223163, -0.92696251, -0.90784899, -0.88403458, -0.85453511,
       -0.81825539, -0.77402985, -0.72069563, -0.6572057 , -0.58278295,
       -0.49710574, -0.40049842, -0.2940833 , -0.17984082, -0.06053197,
        0.06053197,  0.17984082,  0.2940833 ,  0.40049842,  0.49710574,
        0.58278295,  0.6572057 ,  0.72069563,  0.77402985,  0.81825539,
        0.85453511,  0.88403458,  0.90784899,  0.92696251,  0.94223163,
        0.95438418,  0.96402758,  0.97166188,  0.97769438,  0.98245414,
        0.9862053 ,  0.98915888,  0.99148279,  0.99331021,  0.99474658,
        0.99587519,  0.99676173,  0.99745797,  0.99800466,  0.99843388,
        0.99877082,  0.99903531,  0.9992429 ,  0.99940584,  0.99953372,
        0.99963408,  0.99971284,  0.99977465,  0.99982316,  0.99986123,
        0.9998911 ,  0.99991454,  0.99993294,  0.99994738,  0.9999587 ,
        0.99996759,  0.99997457,  0.99998004,  0.99998434,  0.99998771])
```

In [17]:

```
#Rectified Linear Unit (ReLU)
def relu(x):
    return np.maximum(0,x)
v_relu = np.vectorize(relu)
```

In [19]:

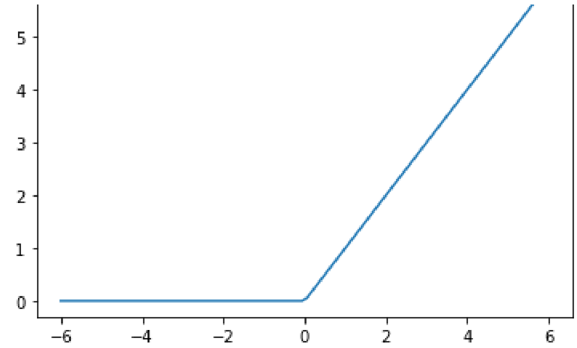
```
x = np.linspace(-6,6,100)
y = v_relu(x)
x
```

Out[19]:

```
array([-6.        , -5.87878788, -5.75757576, -5.63636364, -5.51515152,
       -5.39393939, -5.27272727, -5.15151515, -5.03030303, -4.90909091,
       -4.78787879, -4.66666667, -4.54545455, -4.42424242, -4.3030303 ,
       -4.18181818, -4.06060606, -3.93939394, -3.81818182, -3.6969697 ,
       -3.57575758, -3.45454545, -3.33333333, -3.21212121, -3.09090909,
       -2.96969697, -2.84848485, -2.72727273, -2.60606061, -2.48484848,
       -2.36363636, -2.24242424, -2.12121212, -2.        , -1.87878788,
       -1.75757576, -1.63636364, -1.51515152, -1.39393939, -1.27272727,
       -1.15151515, -1.03030303, -0.90909091, -0.78787879, -0.66666667,
       -0.54545455, -0.42424242, -0.3030303 , -0.18181818, -0.06060606,
        0.06060606,  0.18181818,  0.3030303 ,  0.42424242,  0.54545455,
        0.66666667,  0.78787879,  0.90909091,  1.03030303,  1.15151515,
        1.27272727,  1.39393939,  1.51515152,  1.63636364,  1.75757576,
        1.87878788,  2.        ,  2.12121212,  2.24242424,  2.36363636,
        2.48484848,  2.60606061,  2.72727273,  2.84848485,  2.96969697,
        3.09090909,  3.21212121,  3.33333333,  3.45454545,  3.57575758,
        3.6969697 ,  3.81818182,  3.93939394,  4.06060606,  4.18181818,
        4.3030303 ,  4.42424242,  4.54545455,  4.66666667,  4.78787879,
        4.90909091,  5.03030303,  5.15151515,  5.27272727,  5.39393939,
        5.51515152,  5.63636364,  5.75757576,  5.87878788,  6.        ])
```

In [20]:

```
plt.plot(x,y)
plt.show()
```



In [21]:

```
y
```

Out[21]:

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.06060606, 0.18181818, 0.3030303 , 0.42424242, 0.54545455,
       0.66666667, 0.78787879, 0.90909091, 1.03030303, 1.15151515,
       1.27272727, 1.39393939, 1.51515152, 1.63636364, 1.75757576,
       1.87878788, 2.        , 2.12121212, 2.24242424, 2.36363636,
       2.48484848, 2.60606061, 2.72727273, 2.84848485, 2.96969697,
       3.09090909, 3.21212121, 3.33333333, 3.45454545, 3.57575758,
       3.6969697 , 3.81818182, 3.93939394, 4.06060606, 4.18181818,
       4.3030303 , 4.42424242, 4.54545455, 4.66666667, 4.78787879,
       4.90909091, 5.03030303, 5.15151515, 5.27272727, 5.39393939,
       5.51515152, 5.63636364, 5.75757576, 5.87878788, 6.        ])
```

Describe about all Activation Functions

Activation Function: Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Linear Function :-

Equation : Linear function has the equation similar to as of a straight line i.e. y = ax No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer. Range : -inf to +inf Uses : Linear activation function is used at just one place i.e. output layer. Issues : If we will differentiate linear function to bring non-linearity, result will no more depend on input “x” and function will become constant, it won’t introduce any ground-breaking behavior to our algorithm.

Sigmoid Function :-

It is a function which is plotted as ‘S’ shaped graph. Equation : A = 1/(1 + e-x) Nature : Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y. Value Range : 0 to 1 Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.

Tanh Function :- The activation that works almost always better than sigmoid function is Tanh function also knows as Tangent Hyperbolic function. It’s actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.

Equation :- f(x) = tanh(x) = 2/(1 + e-2x) - 1 OR tanh(x) = 2 * sigmoid(2x) - 1 Value Range :- -1 to +1 Nature :- non-linear Uses :- Usually used in hidden layers of a neural network as it’s values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.

RELU :- Stands for Rectified linear unit. It is the most widely used activation function. Chiefly implemented in hidden layers of Neural network.

Equation :- A(x) = max(0,x). It gives an output x if x is positive and 0 otherwise. Value Range :- [0, inf) Nature :- non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function. Uses :- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.