Assignment 4

> Siva Prasad Nidamanuri

Multi Layer Perceptron

- MLP is essentially a combination of layers of perceptrons weaved together. It uses the outputs of the first layer as inputs of the next layer until finally after a particular number of layers, it reaches the output layer.

Step Function

- A step function is a function like that used by the original Perceptron.
- The values used by the Perceptron were node1 = 10 and node2 = -10.

In [13]:
```
def step_function(_node):
    if _node>0:
        return 1
    else:
        return 0
```

Implementation

In [24]:
```
_node1 = 10
print(step_function(_node1))

_node2 = -10
print(step_function(_node2))
```

```
1
0
```

In [19]:
```
def step_function(_node):
    return 1 if _node>0 else 0
```

In [22]:
```
_node3 = -5
print(step_function(_node3))
_node4 = 5
print(step_function(_node4))
```

```
0
1
```

Implementing with numpy

In [25]:
```
import numpy as np
```

In [26]:
```python
def step_function(_node):
    _out = _node>0
    return int(_out)
```

In [27]:
```python
_node1 = 1
print(step_function(_node1))

_node2 = -1
print(step_function(_node2))
```

```
1
0
```

In [30]:
```python
_nArry = np.array([-4,9,-10,1,13,-90])
```

In [32]:
```python
_output = _nArry>0
_output
```

Out[32]:
```
array([False,  True, False,  True,  True, False], dtype=bool)
```

In [36]:
```python
_output.astype(np.int)
```

Out[36]:
```
array([0, 1, 0, 1, 1, 0])
```

In [37]:
```python
def step_function(_node):
    _out = _node>0
    return _out.astype(int)
```

In [40]:
```python
print(_output)
step_function(_output)
```

```
[False  True False  True  True False]
```

Out[40]:
```
array([0, 1, 0, 1, 1, 0])
```

In [41]:
```python
def step_function(_node):
    return np.array(_node>0, dtype=int)
```

In [69]:
```python
_xAxis = np.arange(-5,5,0.1)
print(_xAxis)
```

```
[ -5.00000000e+00  -4.90000000e+00  -4.80000000e+00  -4.70000000e+00
  -4.60000000e+00  -4.50000000e+00  -4.40000000e+00  -4.30000000e+00
  -4.20000000e+00  -4.10000000e+00  -4.00000000e+00  -3.90000000e+00
  -3.80000000e+00  -3.70000000e+00  -3.60000000e+00  -3.50000000e+00
  -3.40000000e+00  -3.30000000e+00  -3.20000000e+00  -3.10000000e+00
  -3.00000000e+00  -2.90000000e+00  -2.80000000e+00  -2.70000000e+00
  -2.60000000e+00  -2.50000000e+00  -2.40000000e+00  -2.30000000e+00
  -2.20000000e+00  -2.10000000e+00  -2.00000000e+00  -1.90000000e+00
  -1.80000000e+00  -1.70000000e+00  -1.60000000e+00  -1.50000000e+00
  -1.40000000e+00  -1.30000000e+00  -1.20000000e+00  -1.10000000e+00
  -1.00000000e+00  -9.00000000e-01  -8.00000000e-01  -7.00000000e-01
  -6.00000000e-01  -5.00000000e-01  -4.00000000e-01  -3.00000000e-01
  -2.00000000e-01  -1.00000000e-01  -1.77635684e-14   1.00000000e-01
   2.00000000e-01   3.00000000e-01   4.00000000e-01   5.00000000e-01
   6.00000000e-01   7.00000000e-01   8.00000000e-01   9.00000000e-01
   1.00000000e+00   1.10000000e+00   1.20000000e+00   1.30000000e+00
   1.40000000e+00   1.50000000e+00   1.60000000e+00   1.70000000e+00
   1.80000000e+00   1.90000000e+00   2.00000000e+00   2.10000000e+00
   2.20000000e+00   2.30000000e+00   2.40000000e+00   2.50000000e+00
   2.60000000e+00   2.70000000e+00   2.80000000e+00   2.90000000e+00
   3.00000000e+00   3.10000000e+00   3.20000000e+00   3.30000000e+00
   3.40000000e+00   3.50000000e+00   3.60000000e+00   3.70000000e+00
   3.80000000e+00   3.90000000e+00   4.00000000e+00   4.10000000e+00
   4.20000000e+00   4.30000000e+00   4.40000000e+00   4.50000000e+00
   4.60000000e+00   4.70000000e+00   4.80000000e+00   4.90000000e+00]
```

In [70]:

```python
_yAxis = step_function(_xAxis)
print(_yAxis)
```

```
[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 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]
```

Implementing with matplot for visualizing the scenarion's

In [71]:

```python
from matplotlib import pyplot as plot
```

In [72]:

```python
plot.figure(figsize=(5,5))
```

Out[72]:

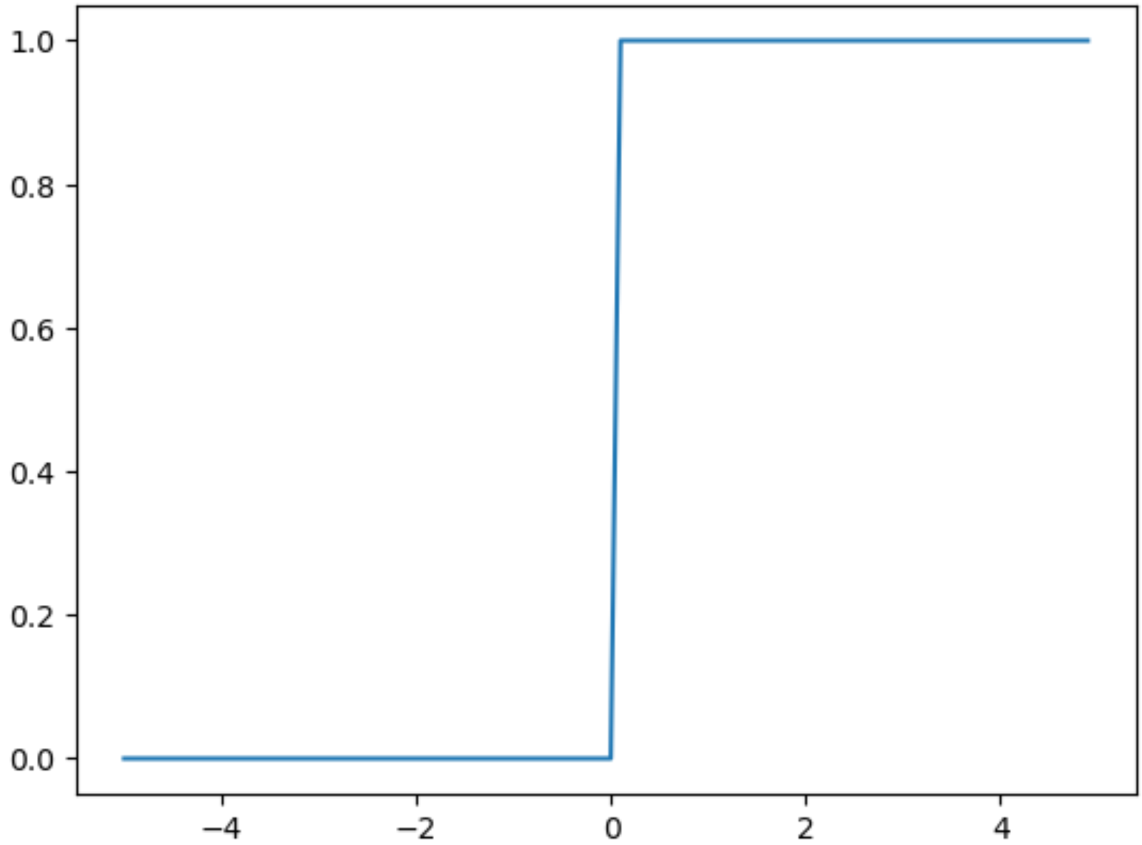
```
<Figure size 500x500 with 0 Axes>

<Figure size 500x500 with 0 Axes>
```

Ploting step function

In [73]:

```python
plot.plot(_xAxis,_yAxis)
plot.show()
```



In [52]:
```
def sigmoid(_node):
    return 1/(1+np.exp(-_node))
```

In [56]:
```
_node1 = 5
print(sigmoid(_node1))
_node2 = -5
print(sigmoid(_node2))
```

```
0.993307149076
0.00669285092428
```

In [59]:
```
#_nArry = np.array([-4,9,-10,1,13,-90])
_nArry = np.array([-4,9,-10,1,13,-90])
```

In [60]:
```
sigmoid(_nArry)
```

Out[60]:
```
array([  1.79862100e-02,   9.99876605e-01,   4.53978687e-05,
         7.31058579e-01,   9.99997740e-01,   8.19401262e-40])
```

In [64]:
```
_xAxis = np.arange(-5,5,0.1)
print(_xAxis)
```

```
[ -5.00000000e+00  -4.90000000e+00  -4.80000000e+00  -4.70000000e+00
  -4.60000000e+00  -4.50000000e+00  -4.40000000e+00  -4.30000000e+00
  -4.20000000e+00  -4.10000000e+00  -4.00000000e+00  -3.90000000e+00
  -3.80000000e+00  -3.70000000e+00  -3.60000000e+00  -3.50000000e+00
  -3.40000000e+00  -3.30000000e+00  -3.20000000e+00  -3.10000000e+00
  -3.00000000e+00  -2.90000000e+00  -2.80000000e+00  -2.70000000e+00
  -2.60000000e+00  -2.50000000e+00  -2.40000000e+00  -2.30000000e+00
  -2.20000000e+00  -2.10000000e+00  -2.00000000e+00  -1.90000000e+00
  -1.80000000e+00  -1.70000000e+00  -1.60000000e+00  -1.50000000e+00
  -1.40000000e+00  -1.30000000e+00  -1.20000000e+00  -1.10000000e+00
  -1.00000000e+00  -9.00000000e-01  -8.00000000e-01  -7.00000000e-01
  -6.00000000e-01  -5.00000000e-01  -4.00000000e-01  -3.00000000e-01
  -2.00000000e-01  -1.00000000e-01  -1.77635684e-14   1.00000000e-01
   2.00000000e-01   3.00000000e-01   4.00000000e-01   5.00000000e-01
   6.00000000e-01   7.00000000e-01   8.00000000e-01   9.00000000e-01
   1.00000000e+00   1.10000000e+00   1.20000000e+00   1.30000000e+00
   1.40000000e+00   1.50000000e+00   1.60000000e+00   1.70000000e+00
   1.80000000e+00   1.90000000e+00   2.00000000e+00   2.10000000e+00
   2.20000000e+00   2.30000000e+00   2.40000000e+00   2.50000000e+00
   2.60000000e+00   2.70000000e+00   2.80000000e+00   2.90000000e+00
   3.00000000e+00   3.10000000e+00   3.20000000e+00   3.30000000e+00
   3.40000000e+00   3.50000000e+00   3.60000000e+00   3.70000000e+00
   3.80000000e+00   3.90000000e+00   4.00000000e+00   4.10000000e+00
   4.20000000e+00   4.30000000e+00   4.40000000e+00   4.50000000e+00
   4.60000000e+00   4.70000000e+00   4.80000000e+00   4.90000000e+00]
```

In [65]:

```python
_yAxis = sigmoid(_xAxis)
print(_yAxis)
```

```
[ 0.00669285  0.00739154  0.00816257  0.0090133   0.0099518   0.01098694
  0.01212843  0.01338692  0.01477403  0.0163025   0.01798621  0.01984031
  0.02188127  0.02412702  0.02659699  0.02931223  0.03229546  0.03557119
  0.03916572  0.04310725  0.04742587  0.05215356  0.05732418  0.06297336
  0.06913842  0.07585818  0.0831727   0.09112296  0.09975049  0.10909682
  0.11920292  0.13010847  0.14185106  0.15446527  0.16798161  0.18242552
  0.19781611  0.21416502  0.23147522  0.24973989  0.26894142  0.2890505
  0.31002552  0.33181223  0.35434369  0.37754067  0.40131234  0.42555748
  0.450166    0.47502081  0.5         0.52497919  0.549834    0.57444252
  0.59868766  0.62245933  0.64565631  0.66818777  0.68997448  0.7109495
  0.73105858  0.75026011  0.76852478  0.78583498  0.80218389  0.81757448
  0.83201839  0.84553473  0.85814894  0.86989153  0.88079708  0.89090318
  0.90024951  0.90887704  0.9168273   0.92414182  0.93086158  0.93702664
  0.94267582  0.94784644  0.95257413  0.95689275  0.96083428  0.96442881
  0.96770454  0.97068777  0.97340301  0.97587298  0.97811873  0.98015969
  0.98201379  0.9836975   0.98522597  0.98661308  0.98787157  0.98901306
  0.9900482   0.9909867   0.99183743  0.99260846]
```

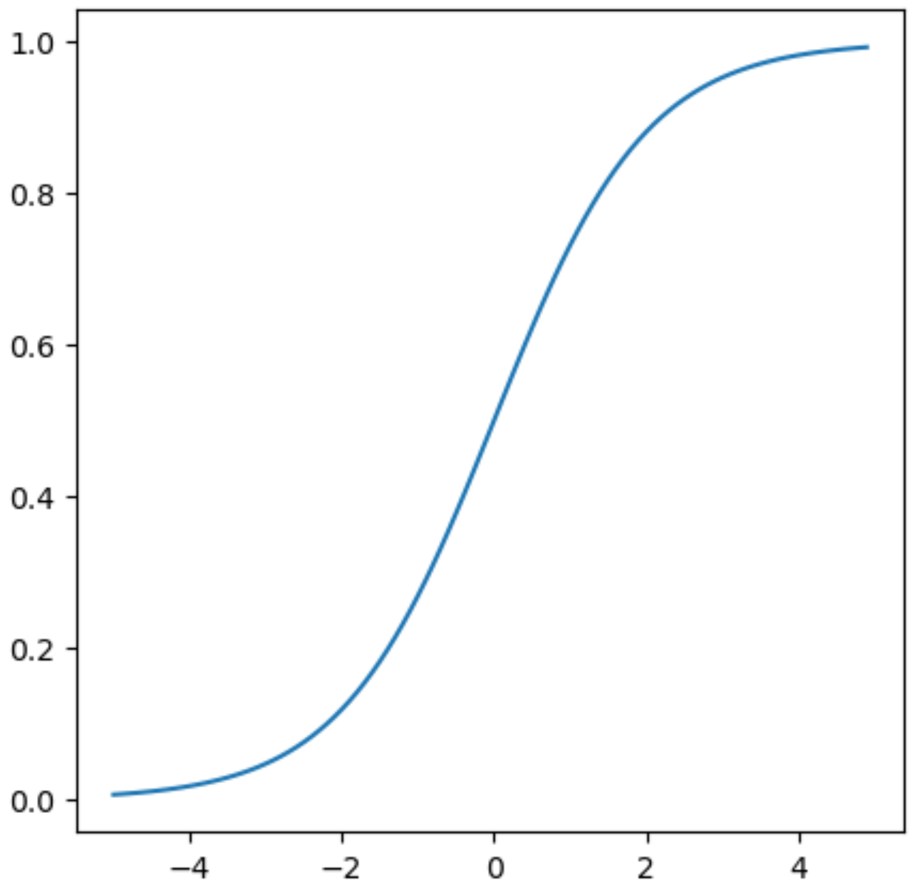
plotting Sigmoid Function

A log-sigmoid function, also known as a logistic function, is given by the relationship:

> sigma(t) =((1)/(1+e^(-beta*t))))

Where β is a slope parameter. This is called the log-sigmoid because a sigmoid can also be constructed using the hyperbolic tangent function instead of this relation, in which case it would be called a tan-sigmoid. Here, we will refer to the log-sigmoid as simply "sigmoid". The sigmoid has the property of being similar to the step function, but with the addition of a region of uncertainty. Sigmoid functions in this respect are very similar to the input-output relationships of biological neurons, although not exactly the same. Below is the graph of a sigmoid function.

In [67]:
```
plot.figure(figsize=(5,5))
plot.plot(_xAxis,_yAxis)
plot.show()
```



In [79]:
```
_matrix1 = np.array([[3,4,5],[8,9,10]])
print(_matrix1)
```

```
[[ 3  4  5]
 [ 8  9 10]]
```

In [80]:
```
_matrix2 = np.array([[10,20]])
```

In [84]:
```
print("matrix 1",np.shape(_matrix1))
print("matrix 2", np.shape(_matrix2))
```

```
matrix 1 (2, 3)
matrix 2 (1, 2)
```

In [88]:
```
np.dot(_matrix2,_matrix1)
print(np.dot(_matrix2,_matrix1))
```

```
[[190 220 250]]
```

In [89]:
```
_nodes = np.array([1,1.5])
_weights = np.array([[0.1,0.2,0.15],[0.4,0.2,0.8]])
_bias = np.array([0.1,0.2,0.05])
```

In [91]:
```
_output = np.dot(_nodes,_weights)+_bias
print(_output)
```

```
[ 0.8  0.7  1.4]
```

In [98]:
```
_sig = sigmoid(_output)
print(_sig)
```

```
[ 0.68997448  0.66818777  0.80218389]
```

In [101…
```
_weights1 = np.array([[0.1,0.2],[0.3,0.4],[0.5,0.6]])
_bias1 = np.array([0.2,0.3])
_output1 = np.dot(_sig,_weights1) +_bias1
print(_output1)
```

```
[ 0.87054572  1.18658034]
```

In [103…
```
_sig1 = sigmoid(_output1)
print(_sig1)
```

```
[ 0.70485924  0.7661289 ]
```

In [106…
```
_weights2 = np.array([[0.7,0.8],[0.9,1]])
_bias2 = np.array([0.2,0.3])
_output2 = np.dot(_sig1,_weights2) +_bias2
print(_output2)
```

```
[ 1.38291748  1.63001629]
```

In [108…
```
_sig2 = sigmoid(_output2)
print(_sig2)
```

```
[ 0.79945915  0.83617187]
```

Feedforward

- The first part of creating a MLP is developing the feedforward algorithm. Feedforward is essentially the process used to turn the input into an output. However, it is not as simple as in the perceptron, but now needs to iterated over the various number of layers. Using matrix operations, this is done with relative ease in python:

In [109...

```
class MultilayerPerceptron:
    def __init__(self,_weight1,_bias1,_weight2,_bias2,_weight3,_bias3):
        self.net={}
        self.net['_weight1']=_weight1
        self.net['_bias1']=_bias1
        self.net['_weight2']=_weight2
        self.net['_bias2']=_bias2
        self.net['_weight3']=_weight3
        self.net['_bias3']=_bias3

    def sigmoid(self, _node):
        return 1/(1+np.exp(-_node))

    def forward(self, _node):
        _weight1,_weight2,_weight3=self.net['_weight1'],self.net['_weight
        _bias1,_bias2,_bias3=self.net['_bias1'],self.net['_bias2'],self.n
        _node1=np.dot(_node,_weight1)+_bias1
        _sig1=self.sigmoid(_node1)
        _node2=np.dot(_sig1,_weight2)+_bias2
        _sig2=self.sigmoid(_node2)
        _node3=np.dot(_sig2,_weight3)+_bias3
        _sig3=self.sigmoid(_node3)
        return _sig3
```

In [116...

```
_weight1=np.array([[.1,.2,.3],[.4,.5,.6]])
_bias1=np.array([.1,.2,.1])
_weight2=np.array([[.1,.3],[.4,.6],[.2,.4]])
_bias2=np.array([.1,.1])
_weight3=np.array([[.2,.3],[.5,.6]])
_bias3=np.array([.1,.2])
mlp=MultilayerPerceptron(_weight1, _bias1, _weight2, _bias2, _weight3, _b
x=np.array([2,3])
y=mlp.forward(x)
print(x,y)
```

```
[2 3] [ 0.65095855  0.70439219]
```

In [117...

```
_weight1=np.random.rand(2,3)
_bias1=np.random.rand(3,)
_weight2=np.random.rand(3,2)
_bias2=np.random.rand(2,)
_weight3=np.random.rand(2,2)
_bias3=np.random.rand(2,)
mlp=MultilayerPerceptron(_weight1, _bias1, _weight2, _bias2, _weight3, _b
x=np.array([2,3])
y=mlp.forward(x)
print(x,y)
```

```
[2 3] [ 0.82618913  0.85833169]
```

In [ ]:

```
```