DSC-550: Project 3

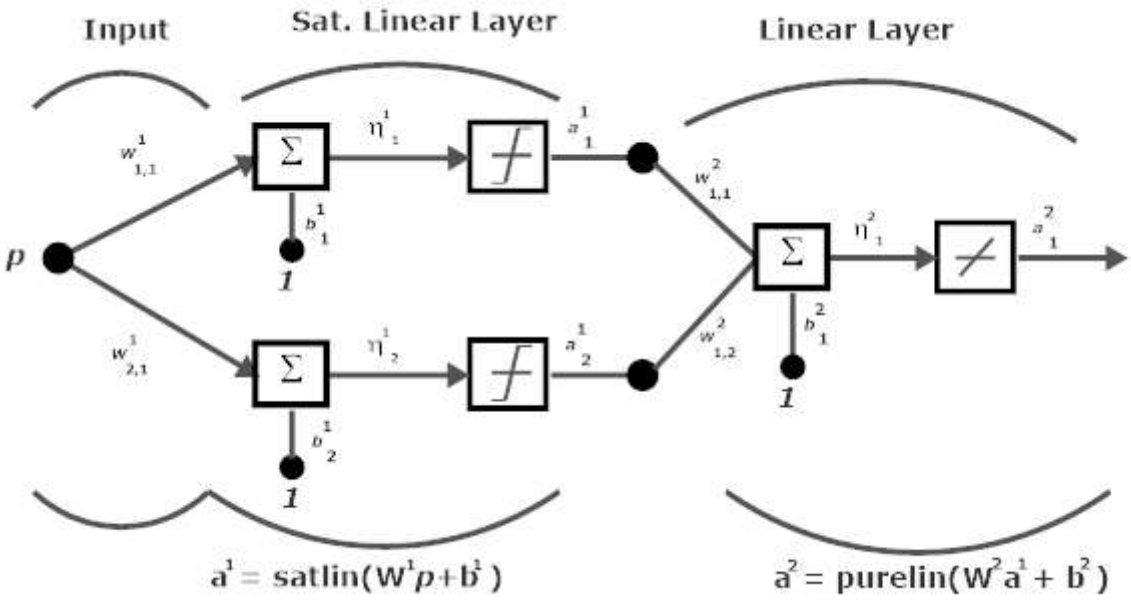
By Wilson Peguero Rosario

Starting with a single neuron and with single layer of neurons, generate the transfer function of a multiple layer network. Examine the inside of the artificial neural network, devise a transfer function for each layer, briefly research how a topology can be obtained, analyze recurrent neural networks, and investigate forward propagation, cost function, and backward propagation.

Using Python and the correspondent libraries, sketch the following responses (plot the indicated variable versus p for (-3 < p < 3))

Solution

A neural network is a series of perceptrons that together function to achieve complex tasks by independently completing simple tasks. The topology of a neural network can be obtained by considering first how many inputs and outputs the neural network has as well as the number of nodes and the amount of hidden layers. From the image below, one can observe that the neural network below is composed of four total layers (an input layer, a hidden layer with two nodes, a one node hidden layer and an output layer).



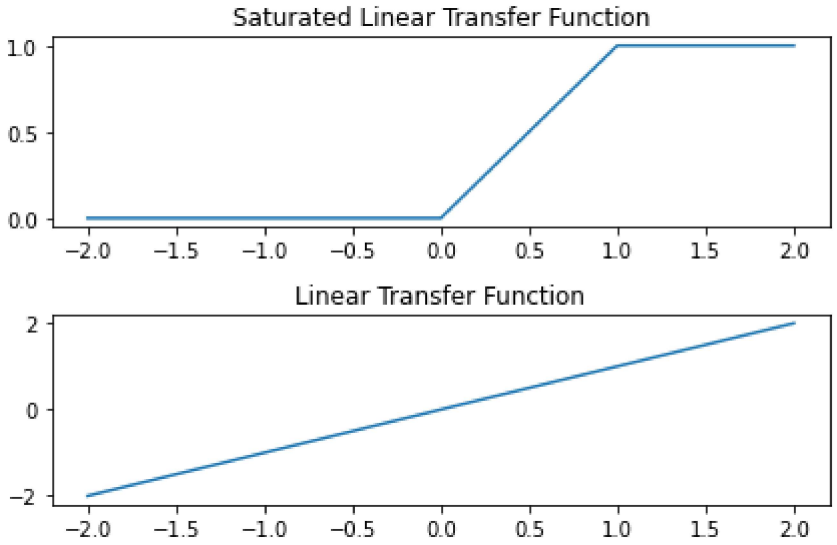
First start by importing the relevant libraries.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
```

Now that the relevant libraries have been imported, one should be able to graphically show the saturating linear and the linear response.

In [ ]:
```
p = np.arange(-2,3)

y_sat = [0]
for i in p:
    if i < 0:
```

```
        y_sat.append(0)
    elif i >= 0 and i <=1:
        y_sat.append(i*1)
    elif i > 1:
        y_sat.append(1)

y_linear = p

figure, axis = plt.subplots(2,1)
axis[0].plot(p,y_sat)
axis[0].set_title("Saturated Linear Transfer Function")

axis[1].plot(p,y_linear)
axis[1].set_title("Linear Transfer Function")
plt.tight_layout()
```



Now that we have the transfer functions, one should be able to obtain the following responses as well as plot them:

1. n_1^1
2. n_2^1
3. a_1^1
4. a_2^1
5. n_1^2
6. a_1^2

In [ ]:

```
weights1 = [2, 1]
weights2 = [1, -1]
bias1 = [2, -1]
bias2 = [0]
node1 = 0
node2 = 0
for i in p:
    node1 += i * weights1[0]
    node2 += i * weights1[1]

node1 += bias1[0]
node2 += bias1[1]

print("The output of the net function for the first node in the saturated linear layer is {}.".for
print("The output of the net function for the second node in the saturated linear layer is {}.".fo
```

```
The output of the net function for the first node in the saturated linear layer is 2.
The output of the net function for the second node in the saturated linear layer is -1.
```

Now that the net function results are out, one can use it as the input for the transfer function which yields the following result:

1. $n_1^1 = 2$
2. $n_2^1 = -1$
3. $a_1^1 = 1$
4. $a_2^1 = 0$

Now that the output from the transfer functions have been obtained, one can obtain the out put of the last net function.

In [ ]:

```
new_inputs = [1, 0]

last_net = bias2[0]
last_net += new_inputs[0] * weights2[0]
last_net += new_inputs[1] * weights2[1]

print("The output of the net function for the first node in the pure linear layer is {}.".format(l
```

```
The output of the net function for the first node in the pure linear layer is 1.
```

The solution for the last outputs are below:

1. $n_1^2 = 1$
2. $a_1^2 = 1$