Neural Network Modeling with Neural Net Libraries

- A number of libraries have been developed to model Perceptrons as well as multi-layered feed-forward network.
  - These models have many traits in common
  - The purpose of this lecture is to introduce a library for python **neurolab**, which has many of the features that allow for the setting up of and neural net training.

Python - Neurolab

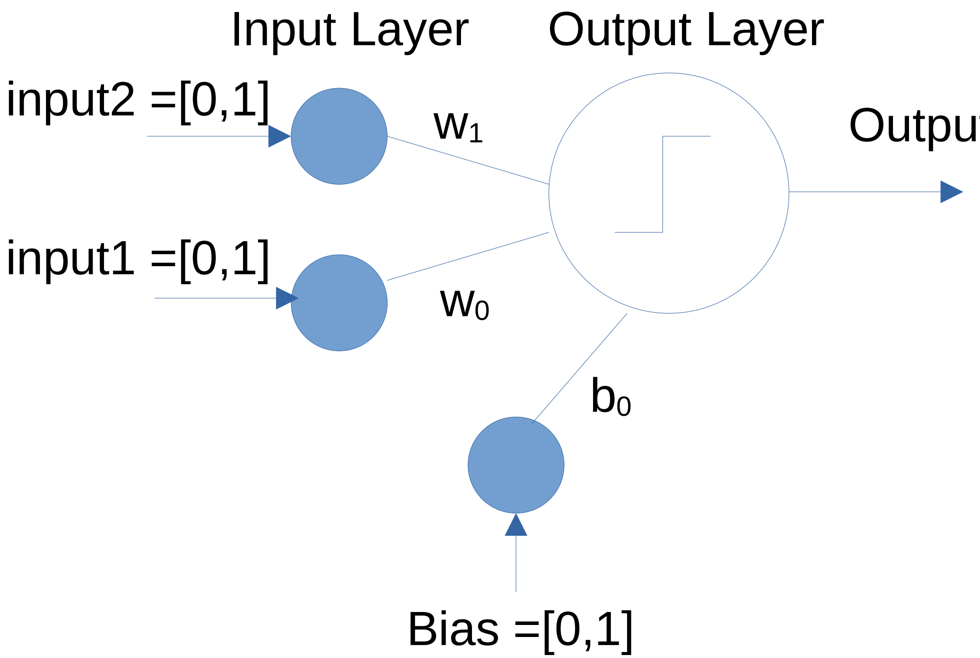
- Neurolab for Python has a number of network models such as a simple perceptron (newp), and multi-layered feed-forward models (newff)
- Other models include Convolutional Nets, Hopfield Nets, etc., that have been developed for neural nets.

Python - Neurolab

- Neurolab can be installed in Python as a package with PIP
- Neurolab can be installed in Thonny by using its package manager.
- It is used in conjunction with Numpy
- At the beginning of a python program, you must import the neurolab library as well as numpy and matplotlib:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
```

Neurolab Model for simple perceptron - **newp**



Input = [[0,1], [0,1]]
Bias = [[0,1]]

While there are 2 layers (Input Layer and Output Layer), neurolab encodes the net as a 1 layer perceptron.

Setting up the Perceptron Neural net in Neurolab

```
#single_layer_perceptron (newp) training in neurolab
#training is done with delta rule
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

#setting up input and target arrays
input = [[0,0], [0,1], [1,0], [1,1]]
target = [[0], [0], [1], [1]]

x, y = [0,0,1,1], [0, 1, 0, 1]

#Create a net with two inputs and 1 output neuron
#train the network with output 1 iff the input is [1, 0] [1, 1] and 0 for
#input [0, 0] and [0,1]

Inputnum=2
outputnum=1
#create single unit perceptron
net = nl.net.newp([[0,1]]*inputnum, outputnum)
```

Accessing weight and bias parameters

```
#number of network inputs can be accessed by net.ci
print("numberof network inputs:",net.ci)

#number of network outputs can be accessed by  net.co
print ("number of network outputs:", net.co)

#number of network layers
print ("number of network layers:", 1+len(net.layers))

#Bias output layer:
net.layers[-1].np['b'][:]=np.array([[0]])
print ("initial bias:", net.layers[-1].np['b'])
```

Setting Layer Weights and Simulation

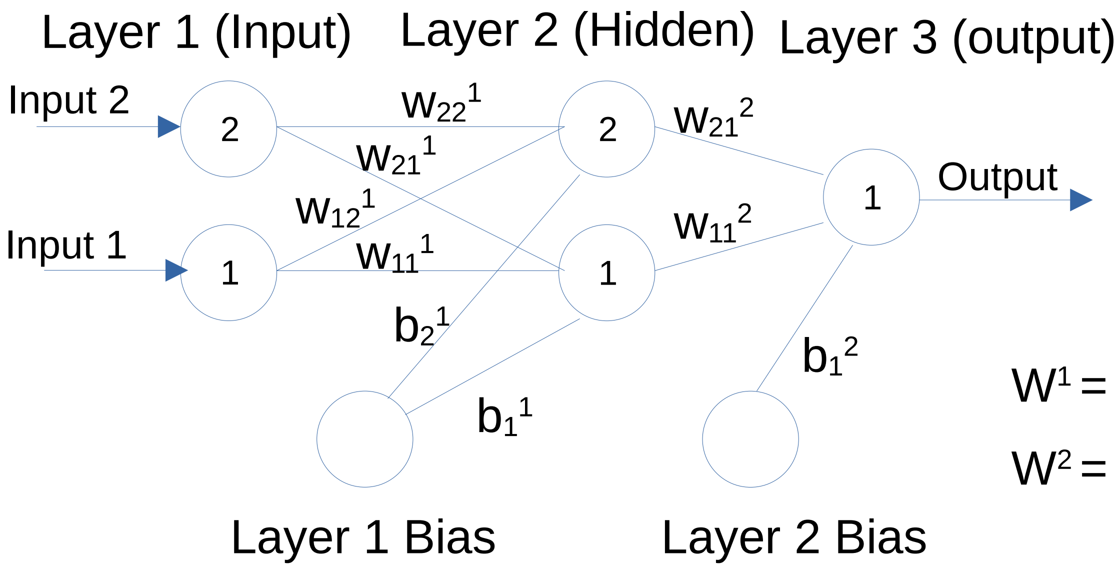
```
net.layers[0].np['w'][:]=np.array([[0,0]])
print("initial weights:",net.layers[0].np['w'])

print ("initial simulation with initialized weights”)
simulate=net.sim ([[0,0]])
print("response to (0,0)",simulate)

simulate=net.sim ([[0,1]])
print("response to (0,1)",simulate)

simulate=net.sim ([[1,0]])
print("response to (1,0)",simulate)

simulate=net.sim ([[1,1]])
print("response to (1,1)",simulate)
```

error = net.train (input, target, epochs=100, show=10, lr=0.1)

Plotting the Simulation Results

```
#The training is done by a delta rule, where the epochs are
#the number of training cycles. Show is the frequency of the output
# and lr is the learning rate parameter.

plt.scatter (x, y)

print ("simulation after training")

simulate=net.sim ([[0,0]])
print("response to (0,0)",simulate)

simulate=net.sim ([[0,1]])
print("response to (0,1)",simulate)

simulate=net.sim ([[1,0]])
print("response to (1,0)",simulate)

simulate=net.sim ([[1,1]])
print("response to (1,1)",simulate)
```

3-Layer Network showing weights and bias components



$W^1 = [[W_{11}^1], [w_{12}^1], [w_{21}^1], w_{22}^1]]$

$W^2 = [[W_{11}^2], [w_{21}^2]]$

While there are 3 Layers, Neurolab encodes the weights as 2 Layers.

Three Layer Network Implementation Using Neurolab (newff)

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

#create feed forward multilayer perceptron

#2 units in the input layer (layer 1)
#input range for each input is [0,1]
layer1=2
#2 units in hidden layer (layer2)
layer2 =2
#1 unit in output layer
layer3=1

#1  output
out=1

#Create network. use tarining with gradient descent
net =nl.net.newff([[0,1]]*layer1, [layer2,layer3])
net.trainf=nl.train.train_gd
```

Accessing the parameters

```
#create learning samples

input = [[0,0], [0,1], [1,0], [1,1]]
x, y = [0,0,1,1], [0,1,0,1]
target = [[0], [1], [1],[0]]

#number of network inputs is parameter net.ci
print("numberof network inputs:",net.ci)

#number of network outputs is parameter net.co
print ("number of network outputs:", net.co)

#number of network layers
print ("number of network layers:",1+len(net.layers))
```

```
#initializing layer1 weights from input layer to hidden layer
#has 2 neurons, each connects to 2 hidden layer neurons.
#This constitutes an 2-array of 2 lists.
#All these weights can be initialized to 0.

#layer1_w = np.array ([[0,0], [0,0]])

#initializing layer 2 weights from hidden layer, which has 2 units
#to output layer, which has 1 unit
#This constitutes an 2-array of single lists
#All these weights can be initialized to 0.
#layer2_w = np.array ([[0],[0]])

#We now print the initialized weights in the network.

print("layer 1-2 initial weights:",net.layers[0].np['w'])
print("layer 2-3 initial weights:",net.layers[1].np['w'])
```

Simulate with Initial Weights and Bias

```
print ("initial simulation with untrained weights and bias values")

simulate=net.sim ([[0,0]])
print("response to (0,0)",simulate)

simulate=net.sim ([[0,1]])
print("response to (0,1)",simulate)

simulate=net.sim ([[1,0]])
print("response to (1,0)",simulate)

simulate=net.sim ([[1,1]])
print("response to (1,1)",simulate)
```

Training the Network

```
#Train network
error = net.train (input, target, epochs=500, show=10, lr=0.1)

#The training is done by a delta rule, where the epochs are
#the number of training cycles. Show is the frequency of the output
# and lr is the learning rate parameter.

print ("the error decline is:",error)
print("final converged weights:", net.layers[0].np['w'], net.layers[1].np['w'])
print ("final bias:", net.layers[0].np['b'], net.layers[1].np['b'])
```

Simulate and Plot Results

```
#Plot results
plt.scatter (x, y)

print ("simulation after training")

simulate=net.sim ([[0,0]])
print("response to (0,0)",simulate)

simulate=net.sim ([[0,1]])
print("response to (0,1)",simulate)

simulate=net.sim ([[1,0]])
print("response to (1,0)",simulate)

simulate=net.sim ([[1,1]])

print("response to (1,1)",simulate)
plt.show()
```