In [1]:

*# python notebook for Make Your Own Neural Network*

*# (c) Tariq Rashid, 2016*

*# license is GPLv2*

In [2]:

**import** **numpy**

*# scipy.special for the sigmoid function expit()*

**import** **scipy.special**

In [3]:

*# neural network class definition*

**class** **neuralNetwork**:

*# initialise the neural network*

**def** \_\_init\_\_(self, inputnodes, hiddennodes, outputnodes, learningrate):

*# set number of nodes in each input, hidden, output layer*

self.inodes = inputnodes

self.hnodes = hiddennodes

self.onodes = outputnodes

*# link weight matrices, wih and who*

*# weights inside the arrays are w\_i\_j, where link is from node i to node j in the next layer*

*# w11 w21*

*# w12 w22 etc*

self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5), (self.hnodes, self.inodes))

self.who = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.onodes, self.hnodes))

*# learning rate*

self.lr = learningrate

*# activation function is the sigmoid function*

self.activation\_function = **lambda** x: scipy.special.expit(x)

**pass**

*# train the neural network*

**def** train(self, inputs\_list, targets\_list):

*# convert inputs list to 2d array*

inputs = numpy.array(inputs\_list, ndmin=2).T

targets = numpy.array(targets\_list, ndmin=2).T

*# calculate signals into hidden layer*

hidden\_inputs = numpy.dot(self.wih, inputs)

*# calculate the signals emerging from hidden layer*

hidden\_outputs = self.activation\_function(hidden\_inputs)

*# calculate signals into final output layer*

final\_inputs = numpy.dot(self.who, hidden\_outputs)

*# calculate the signals emerging from final output layer*

final\_outputs = self.activation\_function(final\_inputs)

*# output layer error is the (target - actual)*

output\_errors = targets - final\_outputs

*# hidden layer error is the output\_errors, split by weights, recombined at hidden nodes*

hidden\_errors = numpy.dot(self.who.T, output\_errors)

*# update the weights for the links between the hidden and output layers*

self.who += self.lr \* numpy.dot((output\_errors \* final\_outputs \* (1.0 - final\_outputs)), numpy.transpose(hidden\_outputs))

*# update the weights for the links between the input and hidden layers*

self.wih += self.lr \* numpy.dot((hidden\_errors \* hidden\_outputs \* (1.0 - hidden\_outputs)), numpy.transpose(inputs))

**pass**

*# query the neural network*

**def** query(self, inputs\_list):

*# convert inputs list to 2d array*

inputs = numpy.array(inputs\_list, ndmin=2).T

*# calculate signals into hidden layer*

hidden\_inputs = numpy.dot(self.wih, inputs)

*# calculate the signals emerging from hidden layer*

hidden\_outputs = self.activation\_function(hidden\_inputs)

*# calculate signals into final output layer*

final\_inputs = numpy.dot(self.who, hidden\_outputs)

*# calculate the signals emerging from final output layer*

final\_outputs = self.activation\_function(final\_inputs)

**return** final\_outputs

In [4]:

*# number of input, hidden and output nodes*

input\_nodes = 3

hidden\_nodes = 3

output\_nodes = 3

*# learning rate is 0.3*

learning\_rate = 0.3

*# create instance of neural network*

n = neuralNetwork(input\_nodes,hidden\_nodes,output\_nodes, learning\_rate)

In [5]:

*# test query (doesn't mean anything useful yet)*

n.query([1.0, 0.5, -1.5])

Out[5]:

array([[ 0.43461026],

[ 0.40331273],

[ 0.56675401]])