**Neural Network**

**Tabish, CSE-09/16**

**Department of Computer Science & Engineering, NIT Srinagar**

**Introduction to the Algorithm:**

**Neural networks** are artificial systems that were inspired by biological neural networks. These systems learn to perform tasks by being exposed to various datasets and examples without any task-specific rules. The idea is that the system generates identifying characteristics from the data they have been passed without being programmed with a pre-programmed understanding of these datasets. Neural networks are based on computational models for threshold logic. Threshold logic is a combination of algorithms and mathematics. Neural networks are based either on the study of the brain or on the application of neural networks to artificial intelligence. The work has led to improvements in finite automata theory.

Neural networks learn via supervised learning: Supervised machine learning involves an input variable x and output variable y. The algorithm learns from a training dataset. With each correct answer, algorithms iteratively make predictions on the data. The learning stops when the algorithm reaches an acceptable level of performance.  
Unsupervised machine learning has input data X and no corresponding output variables. The goal is to model the underlying structure of the data for understanding more about the data. The keywords for supervised machine learning are classification and regression. For unsupervised machine learning, the keywords are clustering and association.

A single neuron transforms given input into some output. Depending on the given input and weights assigned to each input, decide whether the neuron fired or not. Let’s assume the neuron has 3 input connections and one output

We will be using tanh activation function in given example.

The end goal is to find the optimal set of weights for this neuron which produces correct results. Do this by training the neuron with several different training examples. At each step calculate the error in the output of neuron, and back propagate the gradients. The step of calculating the output of neuron is called *forward propagation* while calculation of gradients is called *back propagation*.

**Algorithm:**

for each particle i = 1, ..., S do

Initialize the particle's position with a uniformly distributed random vector: xi ~ U(blo, bup)

Initialize the particle's best known position to its initial position: pi ← xi

if f(pi) < f(g) then

update the swarm's best known position: g ← pi

Initialize the particle's velocity: vi ~ U(-|bup-blo|, |bup-blo|)

while a termination criterion is not met do:

for each particle i = 1, ..., S do

for each dimension d = 1, ..., n do

Pick random numbers: rp, rg ~ U(0,1)

Update the particle's velocity: vi,d ← ω vi,d + φp rp (pi,d-xi,d) + φg rg (gd-xi,d)

Update the particle's position: xi ← xi + vi

if f(xi) < f(pi) then

Update the particle's best-known position: pi ← xi

if f(pi) < f(g) then

Update the swarm's best-known position: g ← pi

**Code:**

from numpy import exp, array, random, dot, tanh

class NeuralNetwork():

def \_\_init\_\_(self):

# Using seed to make sure it'll generate same weights in every run

random.seed(1)

# 3x1 Weight matrix

self.weight\_matrix = 2 \* random.random((3, 1)) - 1

# tanh as activation fucntion

def tanh(self, x):

return tanh(x)

# derivative of tanh function. Needed to calculate the gradients.

def tanh\_derivative(self, x):

return 1.0 - tanh(x) \*\* 2

# forward propagation

def forward\_propagation(self, inputs):

return self.tanh(dot(inputs, self.weight\_matrix))

# training the neural network.

def train(self, train\_inputs, train\_outputs,

num\_train\_iterations):

# Number of iterations we want to perform for this set of input.

for iteration in range(num\_train\_iterations):

output = self.forward\_propagation(train\_inputs)

error = train\_outputs - output

# multiply the error by input and then

# by gradient of tanh funtion to calculate

# the adjustment needs to be made in weights

adjustment = dot(train\_inputs.T, error \*

self.tanh\_derivative(output))

# Adjust the weight matrix

self.weight\_matrix += adjustment

if \_\_name\_\_ == "\_\_main\_\_":

neural\_network = NeuralNetwork()

print ('Random weights at the start of training')

print (neural\_network.weight\_matrix)

train\_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])

train\_outputs = array([[0, 1, 1, 0]]).T

neural\_network.train(train\_inputs, train\_outputs, 10000)

print ('New weights after training')

print (neural\_network.weight\_matrix)

print ("Testing network on new examples ->")

print (neural\_network.forward\_propagation(array([1, 0, 0])))

**Output Observed for different Inputs:**

* **Input passed:**

Random weights at the start of training

[[-0.16595599]

[ 0.44064899]

[-0.99977125]]

* **Output Obtained:**

Random weights at the start of training

[[-0.16595599]

[ 0.44064899]

[-0.99977125]]

New weights after training

[[5.39428067]

[0.19482422]

[0.34317086]]

Testing network on new examples ->

[0.99995873]