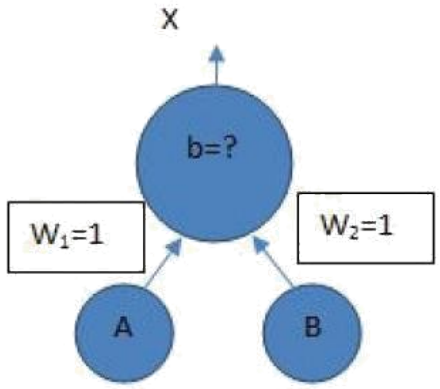
Assignment No. 5

Titile: Implement single layer perceptron

Problem Statement:

Write a program to find the Boolean function to implement following single layer perceptron. Assume all activation functions to be the threshold function which is 1 for all input values greater than zero and 0, otherwise.



Objectives:

1. To learn single layer perceptron.
2. To learn Boolean logic implementation using perceptron.

Software Requirements:

- Ubuntu 16.04

Hardware Requirements:

- Pentium IV system with latest configuration

Theory:

The most common Boolean functions are AND, OR, NOT. The Boolean logic AND only returns 1 if both inputs are 1 else 0, Boolean logic OR returns 1 for all inputs with 1, and will only return 0 if both input is 0 and lastly logic NOT returns the invert of the input, if the input is 0 it returns 1, if the input is 1 it returns 0. To make it clear the image below shows the truth table for the basic Boolean Function.

Truth table of AND gate:

x	y	x . y
0	0	0
0	1	0
1	0	0
1	1	1

Truth table of OR gate:

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

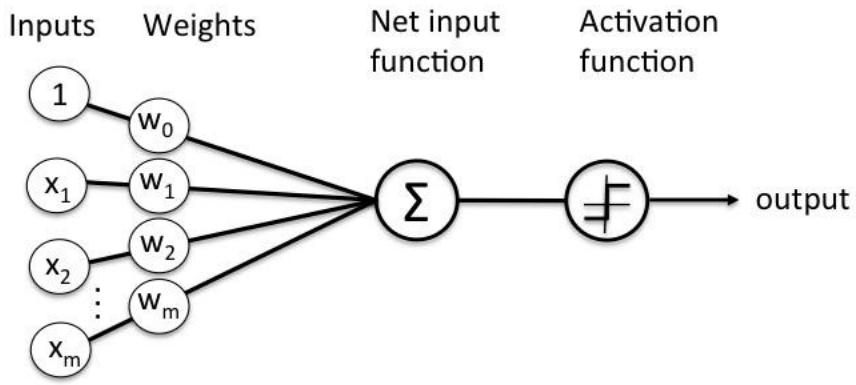
Truth table of NOT gate:

x	$x^{'}$
0	1
1	0

The columns X and Y are the inputs and last is the output. So, for the inputs X = 0, Y = 0 the output is 0.

Perceptron

A perceptron is the basic part of a neural network. A perceptron represents a single neuron on a human's brain, it is composed of the dataset (Xm) , the weights (Wm) and an activation function, that will then produce an output and a bias.



The datasets (inputs) are converted into an ndarray which is then matrix multiplied to another ndarray that holds the weights. Summing up all matrix multiply and adding a bias will create the net input function, the output would then passed into an activation function that would determine if the neuron needs to fire an output or not.

Most common activation function used for classification used is a sigmoid function, which is a great function for classification (Although sigmoid is not the leading activation function for middle

layers of neural networks, it still is widely used for final classifications). The perceptron is simply separating the input into 2 categories, those that cause a fire, and those that don't. It does this by looking at (in the 2-dimensional case):

w1I1 + w2I2 < t

If the LHS is < t, it doesn't fire, otherwise it fires. That is, it is drawing the line:

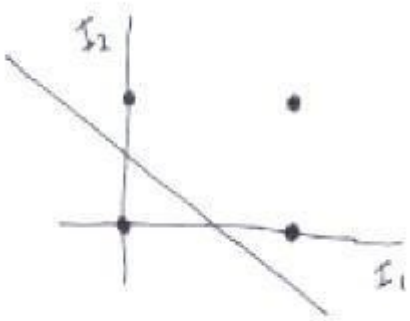
w1I1 + w2I2 = t

and looking at where the input point lies. Points on one side of the line fall into 1 category, points on the other side fall into the other category. And because the weights and thresholds can be anything, this is just *any line* across the 2 dimensional input space.

So what the perceptron is doing is simply drawing a line across the 2-d input space. Inputs to one side of the line are classified into one category, inputs on the other side are classified into another. e.g. the OR perceptron, w1=1, w2=1, t=0.5, draws the line:

I1+I2=0.5

across the input space, thus separating the points (0,1),(1,0),(1,1) from the point (0,0):



As you might imagine, not every set of points can be divided by a line like this. Those that can be, are called *linearly separable*.

In 2 input dimensions, we draw a 1 dimensional line. In n dimensions, we are drawing the (n-1)

dimensional *hyperplane*:

w1I1 + .. + wnIn = t

Conclusion

Thus we learned implementation of single layer perceptron for AND, OR and NOT Boolean function.

```
%tensorflow_version 1.x
```

```
TensorFlow 1.x selected.
```

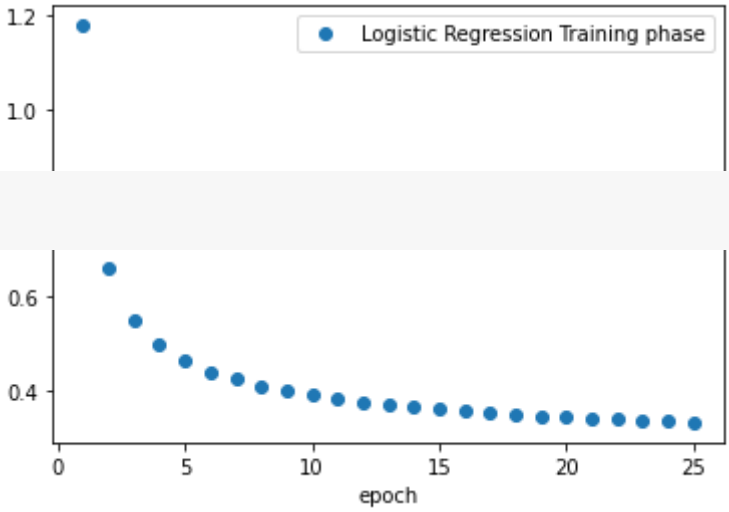
```
!pip install -q git+https://github.com/tensorflow/examples.git
```

```
  Building wheel for tensorflow-examples (setup.py) ... done
```

```
# Import MINST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot = True)

import tensorflow as tf
import matplotlib.pyplot as plt

# Parameters
learning_rate = 0.01
training_epochs = 25
batch_size = 100
display_step = 1

# tf Graph Input
x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28 = 784
y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

# Create model
# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# Construct model
activation = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cross_entropy = y*tf.log(activation)
cost = tf.reduce_mean(-tf.reduce_sum(cross_entropy,reduction_indices = 1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

#Plot settings
avg_set = []
epoch_set = []

# Initializing the variables
init = tf.initialize_all_variables()
# Launch the graph
with tf.Session() as sess:
   sess.run(init)

   # Training cycle
   for epoch in range(training_epochs):
      avg_cost = 0.
      total_batch = int(mnist.train.num_examples/batch_size)
```

```
    total_batch = int(mnist.train.num_examples/batch_size)

    # Loop over all batches
    for i in range(total_batch):
       batch_xs, batch_ys = mnist.train.next_batch(batch_size)
       # Fit training using batch data
       sess.run(optimizer,feed_dict = {x: batch_xs, y: batch_ys})
       # Compute average loss
       avg_cost += sess.run(cost,feed_dict = {x: batch_xs,y: batch_ys})/total_bat
    # Display logs per epoch step
    if epoch % display_step == 0:
       print ("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))
       avg_set.append(avg_cost)
       epoch_set.append(epoch+1)
print ("Training phase finished")

plt.plot(epoch_set,avg_set, 'o', label = 'Logistic Regression Training phase')
plt.ylabel('cost')
plt.xlabel('epoch')
plt.legend()
plt.show()

# Test model
correct_prediction = tf.equal(tf.argmax(activation, 1), tf.argmax(y, 1))

# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print ("Model accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.lab
```

```
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting /tmp/data/train-labels-idx1-ubyte.gz
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/contrib/
Instructions for updating:
Please use tf.one_hot on tensors.
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/contrib/
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/mode
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/u
Instructions for updating:
Use `tf.global_variables_initializer` instead.
Epoch: 0001 cost= 1.176937575
Epoch: 0002 cost= 0.662759472
Epoch: 0003 cost= 0.550749971
Epoch: 0004 cost= 0.496915375
Epoch: 0005 cost= 0.463788255
Epoch: 0006 cost= 0.440970741
Epoch: 0007 cost= 0.424019307
Epoch: 0008 cost= 0.410684467
Epoch: 0009 cost= 0.399959141
Epoch: 0010 cost= 0.390958790
Epoch: 0011 cost= 0.383321118
Epoch: 0012 cost= 0.376798455
Epoch: 0013 cost= 0.371062163
Epoch: 0014 cost= 0.365964811
Epoch: 0015 cost= 0.361388665
Epoch: 0016 cost= 0.357294495
Epoch: 0017 cost= 0.353549648
Epoch: 0018 cost= 0.350138700
Epoch: 0019 cost= 0.346981994
Epoch: 0020 cost= 0.344132194
Epoch: 0021 cost= 0.341434061
Epoch: 0022 cost= 0.339009315
Epoch: 0023 cost= 0.336673638
Epoch: 0024 cost= 0.334476622
Epoch: 0025 cost= 0.332499700
Training phase finished
```




```
Model accuracy: 0.914
```