Week2 Assignments – SATISH RAMACHANDRAN

PROBLEM #2

Sales prediction based on media advertisement

```
# Assignment 2
# Sales prediction

# We'll use Tensor flow 1.x
import tensorflow.compat.v1 as tf
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

# Disable 2.0 behavior
tf.disable_v2_behavior()

# Random seed initialization
RANDOM_SEED = 55
tf.set_random_seed(RANDOM_SEED)

# Input file data
sales_file_data = pd.read_csv('Advertising.csv')
#print(sales_file_data)

# Split features and result
ad_media = sales_file_data[['TV','Radio','Newspaper']]
sales = sales_file_data[['Sales']]
#print(ad_media)
#print(sales)

# Scale the features
# NOTE: We don't need to scale the sales.
scaled_ad_media = preprocessing.minmax_scale(ad_media)
print(scaled_ad_media)
```

```
# Need to convert sales from dataFrame to numpy array
# NOTE: This is very important.
sales=sales.to_numpy()
print(sales)

print("Splitting trian and test set")
# Split the input data into training and testing partitions
train_media, test_media, train_sales, test_sales = train_test_split(scaled_ad_media, sales,
test_size=0.30, random_state=RANDOM_SEED)

ad_media_shape = train_media.shape[1]
sales_shape = train_sales.shape[1]

learning_rate = 0.008
# Number of iterations
epochs = 3000

# =====================================================
# Neural network model parameters
# Inputs are TV, Radio and Newspaper
n_input = ad_media_shape
n_hidden = 8
# The output is a single value
n_output = sales_shape

print("model dimenstions: input: {inp}, hidden: {hidden}, output: {out}".format(inp=n_input,
hidden=n_hidden, out=n_output))
inputs = tf.placeholder("float", shape=[None, n_input])
output = tf.placeholder("float", shape=[None, n_output])
# =====================================================

# Weights from the input layer to the hidden layer
W1 = tf.Variable(tf.random_uniform([n_input, n_hidden], -1.0, 1.0))
# weight_initer = tf.truncated_normal_initializer(mean=0.0, stddev=0.01)
# W1 = tf.get_variable(name="Weight1", dtype=tf.float32, shape=[n_input, n_hidden],
initializer=weight_initer)
# W1 = tf.get_variable(name="W1", shape=[n_input, n_hidden],
initializer=tf.contrib.layers.xavier_initializer())

# Weights from the hidden layer to the output layer
W2 = tf.Variable(tf.random_uniform([n_hidden, n_output], -1.0, 1.0))
# W2 = tf.get_variable(name="Weight2", dtype=tf.float32, shape=[n_hidden, n_output],
initializer=weight_initer)
```

```
# W2 = tf.get_variable(name="W1", shape=[n_hidden, n_output],
initializer=tf.contrib.layers.xavier_initializer())

# Bias values for nodes in hidden layer
b1 = tf.Variable(tf.zeros([n_hidden]), name='Bias1')
# Bias value for the node in the output layer
b2 = tf.Variable(tf.zeros([n_output]), name='Bias2')

# Use RELU for the activation function
# Output of the hidden layer
L2 = tf.nn.relu(tf.matmul(inputs,W1) + b1)
# Final model output
compOutput = tf.math.add(tf.matmul(L2,W2), b2)

# Linear regression model cost function
cost = tf.reduce_mean(tf.math.square(tf.math.subtract(compOutput, output)))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# root mean squared error (RMSE)
modelOutput = tf.placeholder("float", shape=[None, n_output])
error = tf.math.sqrt(tf.math.reduce_mean(tf.math.square(tf.math.subtract(modelOutput,
output))))
init = tf.global_variables_initializer()

# Print helpers
print_W1 = tf.print(W1)
print_W2 = tf.print(W2)
print_b1 = tf.print(b1)
print_b2 = tf.print(b2)

with tf.Session() as session:
   session.run(init)
   print("****** Model training begin ******")
   print("Length of train media:" + str(len(train_media)))
   print("Length of train sales:" + str(len(train_sales)))
   print(train_media[0:1])
   print(train_sales[0:1])
   for step in range(epochs):
      # Train with each example
      # Train the model with the training set
      for i in range(len(train_media)):
         session.run(optimizer, feed_dict={inputs: train_media[i: i + 1], output: train_sales[i: i +
1]})
```

```
print("****** Model training complete ******")
# Print what we have
print('Weights between input layer and hidden layer')
print('------------------------------------------')
session.run(print_W1)

print('Weights between hidden layer and output layer')
print('-------------------------------------------')
session.run(print_W2)

print('Bias values for the hidden layer')
print('--------------------------------')
session.run(print_b1)

print('Bias value for the output layer')
print('-------------------------------')
session.run(print_b2)

print("Based on the model, this is what the sales will be for the test media input")
test_output = session.run(compOutput, feed_dict={inputs: test_media})
print(test_output)

print("Computing the accuracy")
test_accuracy = session.run(error, feed_dict={modelOutput: test_output, output: test_sales})

print(test_media)
print(test_sales)
print(test_output)

print("Test Accuracy:")
print("=============")
print(test_accuracy)
```

These are the different weights and bias values, as printed by the code:

(Output taken with 8 hidden layers and a learning rate of 0.0.8)

Weights between input layer and hidden layer

[[-0.258197784 3.5517993 4.14433479 0.446795374 0.128205389 7.43452168 -0.162838459 6.1842227]
 [-0.869605064 4.29196167 0.462701529 0.0571018718 0.236857086 -4.11374426 -0.640768051 0.726293445]

[-0.425169 -0.968646705 0.29915148 0.0323934332 -0.559727 0.318706155 -0.188223124 0.476360887]]

Weights between hidden layer and output layer

[[0.810407877]
 [1.17881262]
 [1.40577877]
 [0.152001724]
 [-0.763012648]
 [-1.80858612]
 [-0.767709494]
 [2.09448767]]

Bias values for the hidden layer

[0 -3.2457931 -0.160463646 -0.0207778309 -0.410278052 -1.21030152 0 -0.209612012]

Bias value for the output layer

[5.56935549]

The RSME value for this was 0.58935064. That is, the predicted value was off by ~0.58 on either side. This is the output of actual vs predicted for the TEST set.

TEST DATA (MINMAX ADJUSTED)	TEST DATA OUTPUT	MODEL COMPUTED OUTPUT
[[0.38992222 0.15524194 0.2005277]	[[11.]	[[10.859688]
[0.46838011 0.28830645 0.22251539]	[12.2]	[12.582172]
[0.08420697 0.78629032 0.07915567]	[9.5]	[8.729829]
[0.19918837 0.24193548 0.3764292]	[9.7]	[9.718873]
[0.77375719 0.65120968 0.64995602]	[19.7]	[20.050718]
[0.13121407 0.82862903 0.04837291]	[10.8]	[10.110434]
[0.4721001 0.03830645 0.07651715]	[10.3]	[10.073532]
[0.04227257 0.32056452 0.43359719]	[5.6]	[7.014326]
[0.96482922 0.28024194 0.02990325]	[15.9]	[16.61986]
[0.61515049 0.93145161 0.51363237]	[21.2]	[22.13255]
[0.65268854 0.71370968 0.66226913]	[19.2]	[19.80381]
[0.73655732 0.10887097 0.23834653]	[12.2]	[12.317692]
[0.41393304 0.69758065 0.1064204]	[15.2]	[15.690046]
[0.16672303 0.23387097 0.15919085]	[8.4]	[8.7802515]
[0.05816706 0.75806452 0.18733509]	[8.]	[7.956123]
[0.78254988 0.1733871 0.07387863]	[13.4]	[13.290646]
[0.06323977 0.32258065 0.19349164]	[6.6]	[7.0721674]
[0.4582347 0.38709677 0.14335972]	[13.2]	[13.410755]
[0.74095367 0.67540323 0.39401935]	[19.6]	[20.166885]
[0.4808928 0.59072581 0.10817942]	[15.]	[16.45926]
[1. 0.73185484 0.88478452]	[23.8]	[23.334421]
[0.65843761 0.96169355 0.46262093]	[22.4]	[23.3342]
[0.12614136 0.8125 0.10202287]	[10.9]	[9.891915]
[0.65099763 0.37096774 0.57519789]	[15.2]	[14.7818985]
[0.35136963 0.11491935 0.29991205]	[10.4]	[10.347713]
[0.92323301 0.58266129 0.52242744]	[20.8]	[20.51519]
[0.26208996 0.94354839 0.30079156]	[14.6]	[14.027985]
[0.5539398 0.42137097 0.41424802]	[14.5]	[14.638744]
[0.80960433 0.33669355 0.19876869]	[15.9]	[15.910494]
[0.05478526 0.88104839 0.78364116]	[8.7]	[8.970381]
[0.55765979 0.2016129 0.15215479]	[12.6]	[12.17074]
[0.03719986 0.74395161 0.39489886]	[7.3]	[7.7867785]
[0.44098749 0.86290323 0.25153914]	[18.]	[17.550934]
[0.02603991 0.5483871 0.01583113]	[5.7]	[6.6105633]
[0.59621238 0.1875 0.05364996]	[12.8]	[12.159769]
[0.94690565 0.28024194 0.32277924]	[16.1]	[16.3616]
[0.95671288 0.84677419 0.57959543]	[25.5]	[24.696447]
[0.32127156 0.02822581 0.06244503]	[9.5]	[9.15005]
[0.80047345 0.55443548 0.0941073]	[18.9]	[19.050247]
[0.70645925 0.41532258 0.09146878]	[15.9]	[16.107496]
[0.94521474 0.20362903 0.18557608]	[14.8]	[15.261208]
[0.52722354 0.05241935 0.0703606]	[10.5]	[10.501797]
[0.25160636 0.70564516 0.46086192]	[12.6]	[12.10918]
[0.74974636 0.08669355 0.4353562]	[11.7]	[12.342392]
[0.72370646 0.48387097 0.03254178]	[17.4]	[17.296524]
[0.97024011 0.86693548 0.62884785]	[26.2]	[25.107574]
[0.66587758 0.07056452 0.04925242]	[11.7]	[11.40732]
[0.72607372 0.47580645 0.50395778]	[17.1]	[17.061735]
[0.83902604 0.54637097 0.19876869]	[18.9]	[19.270607]
[0.71017924 0.59475806 0.07915567]	[18.4]	[18.780878]
[0.67331755 0.05241935 0.18381706]	[10.6]	[11.386885]
[0.4687183 0.29233871 0.08707124]	[13.4]	[12.508098]
[0.82008793 0.98790323 0.38698329]	[25.4]	[25.515118]
[0.02671627 0.04233871 0.00615655]	[4.8]	[5.5693555]
[0.22691917 0.73790323 1.]	[12.5]	[12.210353]
[0.12242137 0.77822581 0.57431838]	[10.8]	[9.723003]
[0.74332093 0.66935484 0.33069481]	[20.1]	[20.119476]
[0.71964829 0.86693548 0.294635]	[21.7]	[22.803602]
[0.89448766 0.05846774 0.37554969]	[12.7]	[12.801914]
[0.5732161 0.15725806 0.30694811]]	[11.7]]	[11.959122]]

PROBLEM #1

XOR tenorflow solution

This is the code that I used for the XOR solution. When I set the EPOCHs to 100, it did not match the output expected. But, when I set the EPOCHs to 10000, the model trained very close, and the error was mimimal.

```
# Assignment 2
# XOR implementation in Tensorflow

# We'll use Tensor flow 1.x
import tensorflow.compat.v1 as tf
import numpy as np

# Disable 2.0 behavior
tf.disable_v2_behavior()

# Input data
x_data = np.array([ [0,0],[1,0],[0,1],[1,1] ])
# Expected Output
y_data = np.array([ [0],[1],[1],[0] ])

learning_rate = 0.1
# Number of iterations
epochs = 10000

# Neural network model parameters
n_input = 2
n_hidden = 3
n_output = 1
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
modelOutput = tf.placeholder(tf.float32)

# Weights from the input layer to the hidden layer
W1 = tf.Variable(tf.random_uniform([n_input, n_hidden], -1.0, 1.0))
# Weights from the hidden layer to the output layer
W2 = tf.Variable(tf.random_uniform([n_hidden, n_output], -1.0, 1.0))

# Bias values for nodes in hidden layer
b1 = tf.Variable(tf.zeros([n_hidden]), name='Bias1')
# Bias value for the node in the output layer
b2 = tf.Variable(tf.zeros([n_output]), name='Bias2')
```

```
# Output of the hidden layer
L2 = tf.sigmoid(tf.matmul(X,W1) + b1)
# Final model output
compOutput = tf.sigmoid(tf.matmul(L2,W2) + b2)

cost = tf.reduce_mean(-Y*tf.log(compOutput) - (1-Y)*tf.log(1-compOutput))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# mean squared error
error = tf.math.square(tf.math.subtract(modelOutput, Y))
init = tf.global_variables_initializer()

# Print helpers
print_W1 = tf.print(W1, summarize=-1)
print_W2 = tf.print(W2, summarize=-1)
print_b1 = tf.print(b1, summarize=-1)
print_b2 = tf.print(b2, summarize=-1)

with tf.Session() as session:
   session.run(init)
   for step in range(epochs):
      session.run(optimizer, feed_dict={X: x_data, Y: y_data})

   # Print what we have
   print('Weights between input layer and hidden layer')
   print('-------------------------------------------')
   session.run(print_W1)

   print('Weights between hidden layer and output layer')
   print('--------------------------------------------')
   session.run(print_W2)

   print('Bias values for the hidden layer')
   print('--------------------------------')
   session.run(print_b1)

   print('Bias value for the output layer')
   print('-------------------------------')
   session.run(print_b2)

   # Compute the value for the four inputs
   print('==========================')
   print('Outputs after the training')
   print('==========================')
```

```
    output = session.run(compOutput, feed_dict={X: x_data})
    print(output)
    #print(session.run(compOutput, feed_dict={X: x_data}))

    print('========================================')
    print('Squared output error after the training')
    print('========================================')
    print(session.run(error, feed_dict={modelOutput: output, Y: y_data}))
```

These are the trained weights:

```
Weights between input layer and hidden layer
---------------------------------------------
[[6.00337744 -0.154329389 -6.04099703]
 [-6.18553638 0.748931348 5.82831907]]
Weights between hidden layer and output layer
---------------------------------------------
[[9.6076889]
 [-0.902243376]
 [9.79126644]]
Bias values for the hidden layer
--------------------------------
[-3.32150602 0.0453489386 -3.19583607]
Bias value for the output layer
-------------------------------
[-4.26451874]
```

===========================

Outputs after the training

===========================

[[0.01787861]
[0.98664176]
[0.9859263]
[0.01391172]]

==

Squared output error after the training

==

[[0.00031964]
[0.00017844]
[0.00019807]
[0.00019354]]