[Week2 Assignments – SATISH RAMACHANDRAN 1](#_Toc37616689)

[PROBLEM #2 1](#_Toc37616690)

[PROBLEM #1 7](#_Toc37616691)

# 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.**

****

## 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]]**