Thomas Whitcomb
thomaswhitcomb@yahoo.com
Feb 17th, 2019

Lesson 4

Code

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn import datasets
from sklearn import preprocessing
import pandas as pd
import sys

class Graph:
    def __init__(self):
        tf.reset_default_graph()
        tf.set_random_seed(self.random_seed())

    def cost_calc(self):
        print("Need to override calc_cost")

    def activation_calc(self):
        print("Need activaation calc")

    def variable_initializer(self):
        print("Missing variable_initializer")

    def calc_accuracy(self,sess,features,labels):
        print("Missing calc_accuracy")

    def random_seed(self):
        return 42

    def initialize(self,features,labels,hidden):
        self.features = features
        self.labels = labels
```

```
        self.hidden = hidden

        input_shape = self.features.shape[1]
        out_shape = self.labels.shape[1]
        self.inputs = tf.placeholder(
                tf.float32,
                shape=[None, input_shape])
        self.output = tf.placeholder(
                tf.float32,
                shape=[None, out_shape])
        self.test_outputss = tf.placeholder(
                tf.float32,
                shape=[None, out_shape])
        self.learning_rate = tf.placeholder(tf.float32)

        self.w1 = tf.get_variable(name="w1",
                shape=[input_shape, hidden],
                initializer=self.variable_initializer()())
        self.b1 = tf.get_variable(name="b1",
                shape=[hidden],
                initializer=tf.constant_initializer(0.0))
        self.h1 = self.activation_calc()(
                tf.matmul(self.inputs, self.w1) + self.b1)
        self.w2 = tf.get_variable(name="w2",
                shape=[hidden, out_shape],
                initializer=self.variable_initializer()())
        self.b2 = tf.get_variable(name="b2",
                shape=[out_shape],
                initializer=tf.constant_initializer(0.0))

        self.o1 = self.activation_calc()(
                tf.matmul(self.h1, self.w2) + self.b2)

        correct_prediction = tf.equal(
                tf.argmax(self.o1,axis=1),
                tf.argmax(self.test_outputss,axis=1))
        self.accuracy = tf.reduce_mean(
                tf.cast(correct_prediction, tf.float32))

        self.cost = self.cost_calc()(self.output,self.o1)
        self.updates = tf.train.GradientDescentOptimizer(
                self.learning_rate).minimize(self.cost)

    def train(self,tsize,epochs,lr):
        train_features,test_features,train_labels,test_labels =
train_test_split(
             self.features,
             self.labels,
```

```
            test_size=tsize ,
            random_state=self.random_seed())

        with tf.Session() as sess:
            init = tf.global_variables_initializer()
            sess.run(init)
            for epoch in range(epochs+1):
                # Train with each example
                for i in range(len(train_features)):
                    op,cst = sess.run([self.updates,self.cost],
                            feed_dict={
                                self.inputs: train_features[i: i + 1],
                                self.output: train_labels[i: i + 1],
                                self.learning_rate:lr})
                if (epoch % (epochs/20)) == 0:
                    test_accuracy = self.calc_accuracy(
                            sess,test_features,
                            test_labels)
                    print("Epoch: %d, accuracy: %.5f, cost: %.5f" %
                            (epoch, test_accuracy, cst))

            print("Weights Level 1:\n",sess.run(self.w1))
            print("Bias Level 1:\n",sess.run(self.b1))
            print("Weights Level 2:\n",sess.run(self.w2))
            print("Bias Level 2:\n",sess.run(self.b2))

            return test_accuracy,cst

class Graph1(Graph):
    def cost_calc(self):
        def calculator(target,compValue):
            return tf.reduce_mean(-target*tf.log(compValue) -
                    (1-target)*tf.log(1-compValue))
        return calculator
    def activation_calc(self):
        return tf.sigmoid
    def variable_initializer(self):
        return tf.contrib.layers.xavier_initializer
    def calc_accuracy(self,sess,features,labels):
        return sess.run(
                self.accuracy,
                feed_dict={
                    self.inputs:features,
                    self.test_outputss:labels})

class Graph2(Graph):
    def cost_calc(self):
        def calculator(target,compValue):
```

```
            return tf.reduce_mean(
                    tf.pow(target - compValue,2))
        return calculator

    def activation_calc(self):
        return tf.nn.relu

    def variable_initializer(self):
        return tf.initializers.random_uniform

    def calc_accuracy(self,sess,features,labels):
        xx1 = np.stack(labels,axis=0)
        xx2 = np.stack(
                sess.run(self.o1,
                    feed_dict={
                        self.inputs:features,
                        self.test_outputss:labels}),axis=0)
        return sess.run(
                tf.sqrt(
                    tf.reduce_mean(
                        tf.square(
                            tf.subtract(xx1,xx2)))))

def scale(t):
    tMin = t.min(axis=0)
    tMax = t.max(axis=0)
    return (t-tMin)/(tMax-tMin)

def problem1(epochs,learning_rate,hidden):
    f = open("Admissions.csv")
    f.readline()
    dataset = np.genfromtxt(fname = f, delimiter = ',')
    features = dataset[:,1:] # antecedents
    features_scaled = scale(features)
    labels = dataset[:,0:1]  # consequent
    one_hot = np.zeros(shape=(len(labels),2))
    for i in range(0,len(labels)):
        one_hot[i,int(labels[i])] = 1

    g = Graph1()
    g.initialize(features_scaled,one_hot,hidden)
    return g.train(0.30,epochs,learning_rate)

def problem2(epochs,learning_rate,hidden):
    f = open("Advertising.csv")
    f.readline()
    dataset = np.genfromtxt(fname = f, delimiter = ',')
    features = dataset[:,1:4] # antecedents
```

```
    features_scaled = scale(features)
    labels = dataset[:,4:]  # consequent
    labels_scaled = scale(labels)
    g = Graph2()
    g.initialize(features_scaled,labels_scaled,hidden)
    return g.train(0.30,epochs,learning_rate)

def main():
    print("Problem 1")
    print("=========")
    problem1(375,0.1,9)
    print("Problem 2")
    print("=========")
    print("Accuracy is via RMSE")
    problem2(500,0.01,5)

if __name__ == "__main__":
    main()
```

Results

Run > python lesson4.py

```
Problem 1
=========
Epoch: 0, accuracy: 0.67500, cost: 0.36755
Epoch: 75, accuracy: 0.73333, cost: 0.11847
Epoch: 150, accuracy: 0.75000, cost: 0.11610
Epoch: 225, accuracy: 0.75000, cost: 0.11407
Epoch: 300, accuracy: 0.75000, cost: 0.11176
Epoch: 375, accuracy: 0.75000, cost: 0.11009

Weights Level 1:
 [[-2.3026052  -0.26621556  0.34624323 -0.20901091  0.62681246 -1.0863187
  -0.44966298 -3.2398062  -1.6787095 ]
 [-2.2877874  -0.10465505 -0.60933286  1.7770392   2.6080217   1.8389196
   0.84129864 -2.1970472  -1.8251772 ]
 [ 1.5223188  -1.5600401  -1.6974384  -1.916973   -3.2841446  -1.8059342
  -1.7172989   1.4502791   1.2959756 ]]

Bias Level 1:
 [-0.2882644  -1.414716   -1.4287888  -2.0827897  -1.8387002  -1.741329
 -1.6675537   0.19879904 -0.83083284]

Weights Level 2:
 [[ 1.6850691  -1.2131442 ]
```

```
 [ 0.2716324   0.78746194]
 [-0.47689864  0.21940975]
 [-0.55776393  0.91958785]
 [-1.1342436   0.9533675 ]
 [-0.61192983  0.9399046 ]
 [-0.9596164   0.05750658]
 [ 1.711513   -2.077902  ]
 [ 1.031563   -1.2603813 ]]

Bias Level 2:
 [ 0.9256766  -0.93577576]

Problem 2
=========
Accuracy is via RMSE

Epoch: 0, accuracy: 0.27416, cost: 0.04632
Epoch: 25, accuracy: 0.07419, cost: 0.00622
Epoch: 50, accuracy: 0.06980, cost: 0.00545
Epoch: 75, accuracy: 0.06594, cost: 0.00475
Epoch: 100, accuracy: 0.06244, cost: 0.00422
Epoch: 125, accuracy: 0.05942, cost: 0.00382
Epoch: 150, accuracy: 0.05682, cost: 0.00328
Epoch: 175, accuracy: 0.05418, cost: 0.00243
Epoch: 200, accuracy: 0.05206, cost: 0.00169
Epoch: 225, accuracy: 0.04999, cost: 0.00121
Epoch: 250, accuracy: 0.04825, cost: 0.00098
Epoch: 275, accuracy: 0.04659, cost: 0.00080
Epoch: 300, accuracy: 0.04518, cost: 0.00064
Epoch: 325, accuracy: 0.04409, cost: 0.00055
Epoch: 350, accuracy: 0.04278, cost: 0.00047
Epoch: 375, accuracy: 0.04168, cost: 0.00043
Epoch: 400, accuracy: 0.04064, cost: 0.00039
Epoch: 425, accuracy: 0.03966, cost: 0.00036
Epoch: 450, accuracy: 0.03885, cost: 0.00031
Epoch: 475, accuracy: 0.03819, cost: 0.00031
Epoch: 500, accuracy: 0.03744, cost: 0.00032

Weights Level 1:
 [[ 0.48358107  0.37768707  0.96805984  0.8175911   0.86515313]
 [-0.117202    0.4416568  -0.09218649  0.5522364  -0.38549593]
 [ 0.04471642 -0.13990739  0.2977764   0.1392493   0.2520448 ]]

Bias Level 1:
 [-0.46927968 -0.3457328   0.04907916 -0.08450499 -0.24562386]

Weights Level 2:
 [[ 0.7695015 ]
```

```
 [ 0.57346714]
 [ 0.4843294 ]
 [ 0.28136104]
 [-0.5615065 ]]

Bias Level 2:
 [0.07882915]
```