# -\*- coding: utf-8 -\*-

"""

Created on Mon Aug 7 09:56:11 2017

@author: Common

"""

#if file load problem, try saving in same folder as py file

#!/usr/bin/env python

import tensorflow as tf

import numpy as np

import pdb

import data\_handler as dh

import os

qwe=pdb.set\_trace

class NeuralNetwork():

def \_\_init\_\_(self,

lr=0.02,#lr=0.001,

dc=1e-10,

sizes=[5],#sizes=[200,100,50],

L2=0.001,

L1=0

):

self.lr=lr

self.dc=dc

self.sizes=sizes

self.L2=L2

self.L1=L1

self.saverName="ann"

for layer in range(len(sizes)):

self.saverName+=('\_'+str(sizes[layer]))

self.saverDir=os.getcwd()+'\\saved\\'

if(os.path.exists(self.saverDir)!=True):

os.makedirs(self.saverDir)

def init\_weights(self):

self.weights,sizes=[],self.sizes

for layer in range(len(sizes)):

prevSize=self.input\_size

if layer>0:

prevSize=sizes[layer-1]

self.weights.append(tf.Variable(tf.random\_normal([prevSize,sizes[layer]], stddev=0.01),name=('w'+str(layer))))

self.weights.append(tf.Variable(tf.random\_normal([sizes[layer],self.classes\_no], stddev=0.01),name=('w'+str(layer+1))))#add last layer for classification

return

def init\_bias(self):

self.biases,sizes=[],self.sizes

for layer in range(len(sizes)):

self.biases.append(tf.Variable(tf.random\_normal([sizes[layer]], stddev=0.01),name='b'+str(layer)))

self.biases.append(tf.Variable(tf.random\_normal([self.classes\_no], stddev=0.01),name='b'+str(layer+1)))

return

**def train(self,trainset,Y\_labels,epoch,output=True,save=True,load=False,printInterval=1000):**

**#start of tf initializer!!**

**# sess.run('w0:0')**

**#print(sess.run(self.weights[0]))**

**sess=tf.Session()**

**if(load &(os.path.exists(self.saverDir+self.saverName+'.meta'))):**

**ns = tf.train.import\_meta\_graph(self.saverDir+self.saverName+'.meta')**

**ns.restore(sess, tf.train.latest\_checkpoint(self.saverDir)) #restore Variables**

**else:**

**if(load):print('Load file not found.')**

**self.input\_size=len(trainset[0])**

**self.classes\_no=len(Y\_labels[0])**

**''' weight & biases'''**

**self.init\_weights()**

**self.init\_bias()**

**'''input'''**

**self.X = tf.placeholder("float", [None, self.input\_size],name="X")**

**self.Y = tf.placeholder("float", [None, self.classes\_no],name="Y")**

**pred\_y=self.fprop(self.X)**

**cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=pred\_y, labels=self.Y),name='cost') # compute costs**

**train\_op = tf.train.GradientDescentOptimizer(learning\_rate=self.lr).minimize(cost,name='train') # construct an optimizer**

**sess.run(tf.global\_variables\_initializer())**

**for i in range(epoch):**

**sess.run('train', feed\_dict={'X:0':trainset,'Y:0':Y\_labels})**

**if(output & (i%printInterval==0)):**

**print(sess.run('cost:0',feed\_dict={'X:0':trainset,'Y:0':Y\_labels}))**

**if (save):**

**saver=tf.train.Saver()**

**saver.save(sess, self.saverDir+self.saverName)**

**tf.reset\_default\_graph()**

**sess.close()**

**return**

def model(self,input,i):

h=tf.nn.sigmoid(tf.matmul(input, self.weights[i])+self.biases[i])

return h

def fprop(self,input\_d):

for layer in range(len(self.weights)):

input\_d=self.model(input\_d,layer)

return input\_d

def use(self,useset):

predict\_op = tf.argmax(self.fprop(useset), 1)

return self.sess.run(predict\_op)

def accuracy(self,testset,y):

correct\_prediction = tf.equal(tf.argmax(y, 1), self.use(testset))

return self.sess.run(tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32)))

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

x,y=dh.createSetFromCSV('training.csv')

testset,y\_=dh.createSetFromCSV('test.csv')

ann=NeuralNetwork(sizes=[50],lr=1)

ann.train(x,y,1000,printInterval=100,output=True,save=True,load=True)

#print("Accuracy:",ann.accuracy(testset,y\_))

#dh.csvOutput(ann.use(testset),'output//output\_ann.csv')