DEEP LEARNING HOMEWORK 5

*PROF:RICARDOCALIX*

# AUTO ENCODER CODE WITH MNIST DATA AND OUTPUT.

## simple auto-encoder

import tensorflow as tf

import numpy as np

import math

import scipy.spatial.distance as distance

from numpy import genfromtxt

import sklearn

from sklearn.preprocessing import StandardScaler

import pandas as pd

import matplotlib.pyplot as plt

###############################################################################

## data

input1 = genfromtxt('mnist\_auto.txt', delimiter=',', usecols=(i for i in range(1,785)) , skip\_header= 1)

df =pd.DataFrame( input1 )

input = df.to\_numpy()

print "input \n" , input

print "shape \n " , np.shape ( input )

noisy\_input = input + 0.2 \* np.random.random\_sample((input.shape)) - 0.1

print "noisy\_input\n" , noisy\_input

output = input

print "output\n" , output

#################################################################################

# normalizing

## scale to [0, 1]

scaled\_input\_1 = np.divide((noisy\_input - noisy\_input.min()),(noisy\_input.max()-noisy\_input.min()))

scaled\_output\_1 = np.divide( (output-output.min()), (output.max()-output.min()))

print "scaled\_input\_1 \n", scaled\_input\_1

print "scaled\_output\_1 \n",scaled\_output\_1

## scale to [-1, 1]

scaled\_input\_2 = (scaled\_input\_1\*2)-1

scaled\_output\_2 = (scaled\_output\_1\*2)-1

print "scaled\_input\_2 \n" ,scaled\_input\_2

print "scaled\_output\_2 \n " ,scaled\_output\_2

################################################################################

## data set to use

input\_data = scaled\_input\_2

output\_data = scaled\_output\_2

###############################################################################

print "input\_data\_shape \n" ,input\_data.shape

print "output\_data\_shape \n" ,output\_data.shape

n\_sample, n\_features = input\_data.shape

###############################################################################

x = tf.placeholder("float", [None, n\_features]) ## (None, 784)

y\_ = tf.placeholder("float", [None, n\_features]) ## (None, 784)

###############################################################################

## nn with 1 hidden layer, neural net 4x3x4

def inference(x):

Wh = tf.Variable( tf.random\_uniform( (784,184), -1.0 / math.sqrt(n\_features), 1.0 / math.sqrt(n\_features) ) , dtype='float32' )

bh = tf.Variable( tf.zeros([184]) , dtype='float32' )

h1 = tf.nn.tanh( tf.matmul(x, Wh) + bh )

#Wy = tf.transpose(Wh) #tied weights

Wy = tf.Variable( tf.random\_uniform( (184,784) , -1.0 / math.sqrt(n\_features), 1.0 / math.sqrt(n\_features) ) , dtype='float32' )

by = tf.Variable( tf.zeros([784]) , dtype='float32' )

y = tf.nn.tanh( tf.matmul(h1, Wy) + by )

return y, Wh, bh, h1

###############################################################################

def loss\_lse(y, y\_):

meansq = tf.reduce\_mean( tf.square(y\_ - y) ) ## lse

return meansq

#############################################################################

def loss\_cross\_entropy(y, y\_):

#cross\_entropy = -tf.reduce\_sum(y\_ \* tf.log(y)) ##cross entropy

#return cross\_entropy

xentropy = tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=y, labels=y\_)

loss = tf.reduce\_mean(xentropy)

return loss

##############################################################################

def train(cost):

train\_step = tf.train.GradientDescentOptimizer(0.05).minimize(cost)

return train\_step

################################################################################

def test():

print "hello"

##############################################################################

y, Wh, bh, h1 = inference(x)

cost\_lse = loss\_lse(y, y\_)

cost\_xentropy = loss\_cross\_entropy(y, y\_) ## alternative

train\_op = train(cost\_lse)

##############################################################################

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

#############################################################################

n\_epochs = 50

batch\_size = min(50, n\_sample) ## (50, 5)

for i in range(n\_epochs):

sample = np.random.randint(n\_sample, size=batch\_size)

batch\_xs = input\_data[sample,:]

batch\_ys = output\_data[sample,:]

result = sess.run(train\_op, feed\_dict={x: batch\_xs, y\_:batch\_ys})

if i % 100 == 0:

print i, sess.run(cost\_xentropy, feed\_dict={x:batch\_xs,y\_:batch\_ys}), sess.run(cost\_lse, feed\_dict={x:batch\_xs , y\_:batch\_ys})

print "output\_data (no noise)"

print output\_data

print "input\_data (noisy data)"

print input\_data

print "predicted\_output\_data (de-noised data)"

predicted\_output\_data = sess.run(y, feed\_dict={x: input\_data})

print predicted\_output\_data

############################################################################################

## this code compares the distances between:

## a) the noisy input and the image without noise

## b) the noisy input and the de-noised image

## the de-noised image should be closer to the noisy image because the denoising AE

## tries to remove some noise

## range: noisy\_image................de-noised\_image.....................no\_noise\_image

dist = 0.0

print "distance between input\_data (noisy) and output\_data (no noise)"

for j in range(n\_sample):

res1 = distance.euclidean(input\_data[j,:], output\_data[j,:])

dist = dist + res1

print dist

dist\_denoise = 0.0

print "distance between input\_data (noisy) and predicted\_output\_data (after de-noising)"

for j in range(n\_sample):

res2 = distance.euclidean(input\_data[j,:], predicted\_output\_data[j,:] )

dist\_denoise = dist\_denoise + res2

print dist\_denoise

###############################################################################################

print "Wh (weights of encoder)"

print sess.run(Wh)

print "bh"

print sess.run(bh)

print "h1 (compressed version of x)"

print sess.run(h1, feed\_dict={x: input\_data})

######################################################################################

print "<<<<<<<<<<<<<<<DONE>>>>>>>>>>>>>>>>>"

**SCREEN SHOT OF THE OUTPUT OF AUTO ENCODER:**

