**CREDIT CARD FRAUD DETECTION**

**CODE:**

import pandas as pd

import numpy as np

import tensorflow as tf

import tensorflow.compat.v1 as tf

tf.disable\_v2\_behavior()

import os

from datetime import datetime

from sklearn.metrics import roc\_auc\_score as auc

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

df = pd.read\_csv('creditcard.csv')

df.shape

print("Total time spanning: {:.1f} days".format(df['Time'].max() / (3600 \* 24.0)))

print("{:.3f} % of all transactions are fraud. ".format(np.sum(df['Class']) / df.shape[0]\* 100))

df.head()

df.columns

df.dtypes

plt.figure(figsize=(12,5\*4))

gs= gridspec.GridSpec(5,1)

for i, cn in enumerate(df.columns[:5]):

ax= plt.subplot(gs[i])

(sns.distplot)(df[cn][df.Class ==1], bins=50)

sns.distplot(df[cn][df.Class ==0], bins=50)

ax.set\_xlabel('')

ax.set\_title('histogram of feature: '+ str(cn))

plt.show()

TEST\_RATIO = 0.25

df.sort\_values('Time', inplace = True)

TRA\_INDEX = int((1-TEST\_RATIO) \* df.shape[0])

train\_x = df.iloc[:TRA\_INDEX, 1:-2].values

train\_y = df.iloc[:TRA\_INDEX, -1].values

test\_x = df.iloc[TRA\_INDEX:, 1:-2].values

test\_y = df.iloc[TRA\_INDEX:, -1].values

print("Total train examples: {}, total fraud cases: {}, equal to {:.5f} of total cases. ".format(train\_x.shape[0], np.sum(train\_y), np.sum(train\_y)/train\_x.shape[0]))

print("Total test examples: {}, total fraud cases: {}, equal to {:.5f} of total cases. ".format(test\_x.shape[0], np.sum(test\_y), np.sum(test\_y)/test\_y.shape[0]))

cols\_max = []

cols\_min = []

for c in range(train\_x.shape[1]):

cols\_max.append(train\_x[:,c].max())

cols\_min.append(train\_x[:,c].min())

train\_x[:, c] = (train\_x[:, c] - cols\_min[-1]) / (cols\_max[-1] - cols\_min[-1])

test\_x[:, c] = (test\_x[:, c] - cols\_min[-1]) / (cols\_max[-1] - cols\_min[-1])

cols\_mean = []

cols\_std = []

for c in range(train\_x.shape[1]):

cols\_mean.append(train\_x[:,c].mean())

cols\_std.append(train\_x[:,c].std())

train\_x[:, c] = (train\_x[:, c] - cols\_mean[-1]) / cols\_std[-1]

test\_x[:, c] = (test\_x[:, c] - cols\_mean[-1]) / cols\_std[-1]

learning\_rate = 0.01

training\_epochs = 10

batch\_size = 256

display\_step = 1

n\_hidden\_1 = 15

n\_input = train\_x.shape[1]

X = tf.placeholder("float", [None, n\_input])

weights = {

'encoder\_h1': tf.Variable(tf.random\_normal([n\_input, n\_hidden\_1])),

# 'encoder\_h2': tf.Variable(tf.random\_normal([n\_hidden\_1, n\_hidden\_2])),

'decoder\_h1': tf.Variable(tf.random\_normal([n\_hidden\_1, n\_input])),

# 'decoder\_h2': tf.Variable(tf.random\_normal([n\_hidden\_1, n\_input])),

}

biases = {

'encoder\_b1': tf.Variable(tf.random\_normal([n\_hidden\_1])),

# 'encoder\_b2': tf.Variable(tf.random\_normal([n\_hidden\_2])),

'decoder\_b1': tf.Variable(tf.random\_normal([n\_input])),

# 'decoder\_b2': tf.Variable(tf.random\_normal([n\_input])),

}

# Building the encoder

def encoder(x):

layer\_1 = tf.nn.tanh(tf.add(tf.matmul(x, weights['encoder\_h1']),biases['encoder\_b1']))

return layer\_1

def decoder(x):

# Encoder Hidden layer with sigmoid activation #1

layer\_1 = tf.nn.tanh(tf.add(tf.matmul(x, weights['decoder\_h1']),

biases['decoder\_b1']))

return layer\_1

encoder\_op = encoder(X)

decoder\_op = decoder(encoder\_op)

y\_pred = decoder\_op

y\_true = X

batch\_mse = tf.reduce\_mean(tf.pow(y\_true - y\_pred, 2), 1)

cost = tf.reduce\_mean(tf.pow(y\_true - y\_pred, 2))

optimizer = tf.train.RMSPropOptimizer(learning\_rate).minimize(cost)

save\_model = os.path.join("C:\\Users\\anwar\Desktop\\New folder",'temp\_saved\_model\_1layer.ckpt')

saver = tf.train.Saver()

init = tf.global\_variables\_initializer()

with tf.Session() as sess:

now = datetime.now()

sess.run(init)

total\_batch = int(train\_x.shape[0] / batch\_size)

for epoch in range(training\_epochs):

for i in range(total\_batch):

batch\_idx = np.random.choice(train\_x.shape[0], batch\_size)

batch\_xs = train\_x[batch\_idx]

\_, c = sess.run([optimizer, cost], feed\_dict={X: batch\_xs})

if epoch % display\_step == 0:

train\_batch\_mse = sess.run(batch\_mse, feed\_dict={X: train\_x})

print("Epoch:", '%04d' % (epoch + 1),"cost=", "{:.9f}".format(c),"Train auc=", "{:.6f}".format(auc(train\_y, train\_batch\_mse)),"Time elapsed=", "{}".format(datetime.now() - now))

save\_path = saver.save(sess, save\_model)

print("Optimization Finished!")

print("Model saved in file: %s" % save\_path)

OUTPUT





