CCFraudDetectionThruDeepLearning

Applying Deep Learning thru TensorFlow to predict Credit Card Fraud on dataset @Kaggle

Capstone Proposal - Credit Card Fraud Detection

By - Amber, Jagan, Kevin https://github.com/amberved/CCFraudDetection (https://github.com/amberved/CCFraudDetection)

Proposal

Domain Background

Big and high profile credit cards and data breaches have been dominating the headlines in the past couple of years across the world. These problem of Credit cards breaches in U.S. alone is responsible for 47 percent of the worldâ€™s card fraud as of 2014. 15.4 million US consumers were affected by these kind of fraud in 2016, which is nearly 2 million more than in 2015. Dollar amount associated with such activities is 16 Billion in 2016 alone and significantly increasing at the rate of 61 Percent. Hence it is a big clear and present problem that needs Smarter soultions and machine learning can help.

Figure 1

According to data from the Federal Reserve, Credit card Fraud only impacts a fraction of all purchases made with Credit Cards but it represents one of the biggest concerns among consumers and also results into billions of dollars of losses to fininial companies be it bank, credit card companies, retailers & governments.

One can find lot of research and real world implementation done around this area like outlined at https://www.research.ibm.com/foiling-financial-fraud.shtml (https://www.research.ibm.com/foiling-financial-fraud.shtml) http://www.ulb.ac.be/di/map/adalpozz/pdf/Dalpozzolo2015PhD.pdf (http://www.ulb.ac.be/di/map/adalpozz/pdf/Dalpozzolo2015PhD.pdf)

Problem Statement

Credit card Fraud is a complex problem with large number of marid financial aspects. Consequently we need automatic systems able to support fraud detection and fightback. These systems are essential since it is not always possible or easy for a human analyst to detect fraudulent patterns in transaction datasets, often characterized by a large number of samples, many dimensions and online update.

The design of fraud detection machine learning algorithms is however particularly challenging due to the non-stationary distribution of the data, the highly unbalanced classes distributions and the availability of few transactions labeled by fraud investigators.

Listed below are few crucial issues any machine learning model will encounter and should attemp to address i) Why and how undersampling is useful in the presence of class imbalance (i.e. frauds are a small percentage of the transactions) ii) How to deal with unbalanced and evolving data streams (non-stationarity due to fraud evolution and change of spending behavior) iii) How to assess performances in a way which is relevant for detection and iv) How to use feedbacks provided by investigators on the fraud alerts generated.

Credit Card Fraud can in theory to detected based on various features like time of use, place of use, frequncy, unsual amount of spending, frequency of transection and many more such featues. But in general all this can be converted to mathematical values and machine learning models can be applied. In generic terms Credit Card Fraud identification can be treated as classicifcation problem.

Datasets and Inputs

I am planning to use Kaggel dataset on â€œCredit Card Fraud Detectionâ€ found at https://www.kaggle.com/dalpozz/creditcardfraud (https://www.kaggle.com/dalpozz/creditcardfraud)

This dataset presents Credit Card transactions, where it has 492 frauds out of 284,807 transactions. It contains around 30 features for each transection which are PCA transformation. All Features V1, V2, ... V28 are the principal components obtained with PCA. Only 2 features which have not been transformed with PCA are 'Time' and 'Amount'.

The intereseting aspect about using this data set is that ratio of frauds vs total transactions is very low. Secountly, all the data is already convered into PCA so we can not judge the importance of any feature over other and have will force us to work with all of them equally and focus on the r2 or similar mathamatical relationships in place of human intution. Last point about this data set is that this is really good dataset as data preprocessing and cleansing is already done and with few changes can be fed into many supervised learning algorithms.

The major challenge with this data set is that it that fraud transactions like in real time is very small faction of over all transactions, hence we will have to find good strategy for data splitting in training/validation/testing subsets. In this case we will use *Resampling the dataset* methods Essentially this is a method that will process the data to have an approximate 50-50 ratio. One way to achieve this is by OVER-sampling, which is adding copies of the under-represented class (better when you have little data). Another method could be UNDER-sampling, which deletes instances from the over-represented class (better when he have lot's of data). Hence we will try this schemes and measure performance by compare model with resampling and when not using it.

Solution Statement

This problem of identifying frudulant transection from valid credit card, can we taken as a classical ML Classfication problem. Simply put classification problems are tasked of identifying to which of a set of categories a new observation may belongs, on the basis of a training set of data containing observations whose category membership is known.

This problem of identifying the fraulent transections can be broken into 3 steps. 1) Imbalanced in data - The ratio of valid vs fraud transection data avaliable to us. 2) Classifiction -

Benchmark Model

Evaluation Metrics

Classifier performance depends greatly on the characteristics of the data to be classified. There is no single classifier that works best on all given problems. The data set we are using for this problem is highly imbalanced and hence traditional popular measures like "precision and recall" and "receiver operating characteristic (ROC)" may not be very effective for this classification algorithms.

As a performance metric, I think the uncertainty coefficient will have a advantage over simple accuracy in that it is not affected by the relative sizes of the different classes. The uncertainty coefficient is useful for measuring the validity of a statistical classification algorithm and has the advantage over simpler accuracy measures in that it is not affected by the relative fractions of the different classes, i.e., P(x). It also has the unique property that it won't penalize an algorithm for predicting the wrong classes, so long as it does so consistently (i.e., it simply rearranges the classes). This is useful in evaluating clustering algorithms since cluster labels typically have no particular ordering.

Suppose we have samples of two discrete random variables, X and Y. By constructing the joint distribution, PX,Y(x, y), from which we can calculate the conditional distributions, PX|Y(x|y) = PX,Y(x, y)/PY(y).

The uncertainty coefficient or proficiency is defined as: U(X|Y)={{H(X)-H(X|Y)}{H(X)}}={{I(X;Y)}{H(X)}},

Where The entropy of a single distribution is given as: {H(X)=-\sum *{x}P*{X}(x)\log P*{X}(x),} H(X)=-\sum* {x}P*{X}(x)\log P*{X}(x), while the conditional entropy is given as: {H(X|Y)=-\sum *{x,~y}P*{X,Y}(x,~y)\log P*{X|Y}(x|y).} H(X|Y)=-\sum* {{x,~y}}P*{{X,Y}}(x,~y)\log P*{{X|Y}}(x|y).

Project Design

I intentd to follow the strategy to approach this problem:

---

cited and references

https://www.javelinstrategy.com/coverage-area/2017-identity-fraud (https://www.javelinstrategy.com/coverage-area/2017-identity-fraud)

https://www.businesswire.com/news/home/20170201005166/en/Identity-Fraud-Hits-Record-High-15.4-Million (https://www.businesswire.com/news/home/20170201005166/en/Identity-Fraud-Hits-Record-High-15.4-Million)

https://www.kaggle.com/dalpozz/creditcardfraud (https://www.kaggle.com/dalpozz/creditcardfraud)

http://blog.kaggle.com/2016/07/21/approaching-almost-any-machine-learning-problem-abhishek-thakur/ (http://blog.kaggle.com/2016/07/21/approaching-almost-any-machine-learning-problem-abhishek-thakur/)

https://en.wikipedia.org/wiki/K-means_clustering (https://en.wikipedia.org/wiki/K-means_clustering)

https://en.wikipedia.org/wiki/Mixture_model (https://en.wikipedia.org/wiki/Mixture_model)

http://www.cs.cmu.edu/~guestrin/Class/10701-S07/Slides/clustering.pdf (http://www.cs.cmu.edu/~guestrin/Class/10701-S07/Slides/clustering.pdf)

http://www.ele.uri.edu/faculty/he/PDFfiles/ImbalancedLearning.pdf (http://www.ele.uri.edu/faculty/he/PDFfiles/ImbalancedLearning.pdf)

http://www.ulb.ac.be/di/map/adalpozz/pdf/Dalpozzolo2015PhD.pdf

In [18]:
```
import os
import pandas as pd
import numpy as np
import tensorflow as tf
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
%matplotlib inline
```

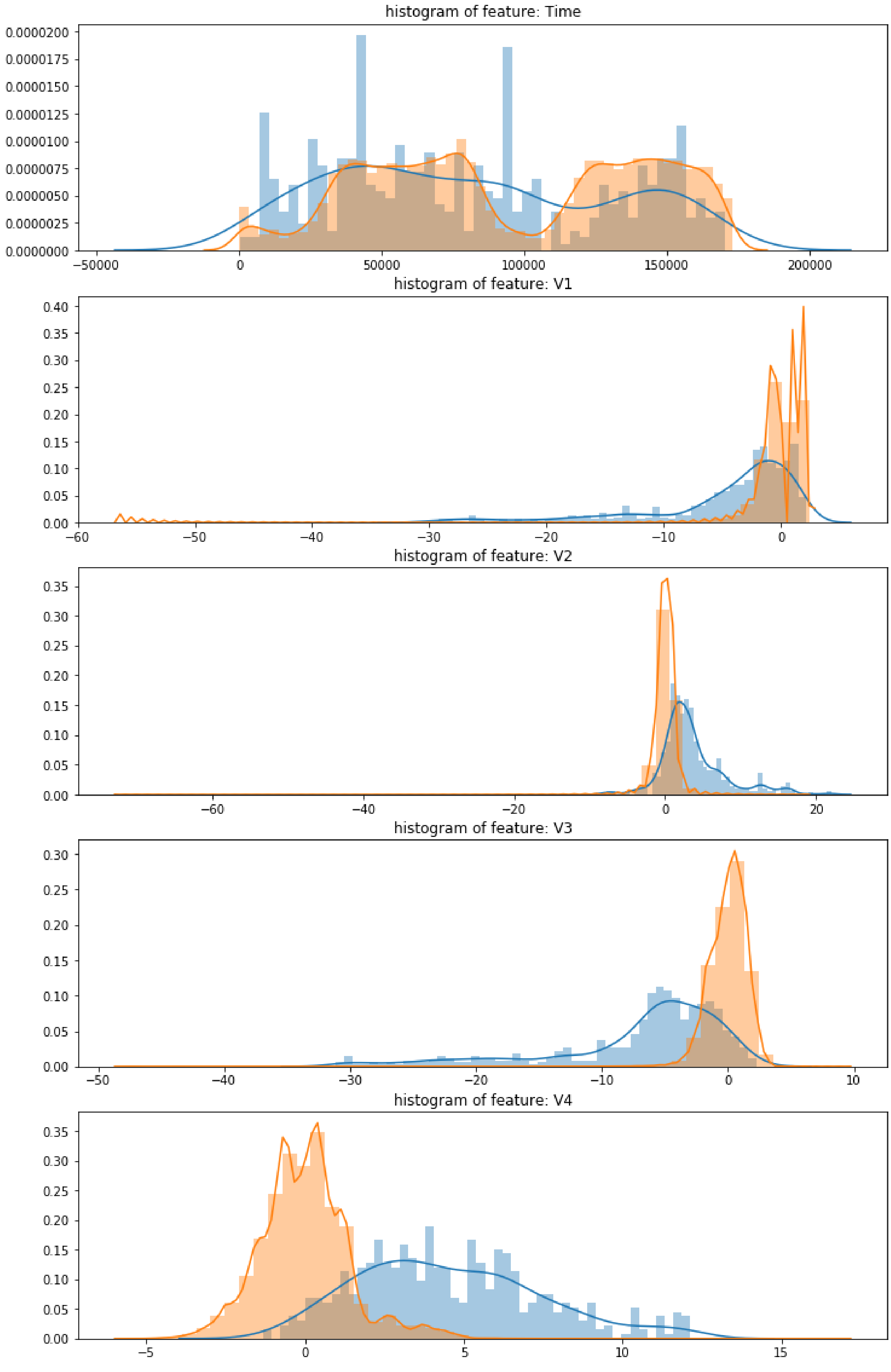
In [6]:
```
df = pd.read_csv('creditcard.csv')
df.shape
print("Total time spanning: {:.1f} days".format(df['Time'].max() / (3600 * 2
4.0)))
print("{:.3f} % of all transactions are fraud. ".format(np.sum(df['Class'])
/ df.shape[0] * 100))
df.head()
df.columns
df.dtypes
```

```
Total time spanning: 2.0 days
0.173 % of all transactions are fraud.
```

Out[6]:
```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

In [4]:
```
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()
```

```
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/amber.ved/anaconda/envs/tf-cpu/lib/python3.5/site-packages/matplotli
b/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```


histogram of feature: Time
0.0000200
0.0000175
0.0000150
0.0000125
0.0000100
0.0000075
0.0000050
0.0000025
0.0000000
−50000
0
50000
100000
150000
200000
histogram of feature: V1
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−60
−50
−40
−30
−20
−10
0
histogram of feature: V2
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−60
−40
−20
0
20
histogram of feature: V3
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−50
−40
−30
−20
−10
0
10
histogram of feature: V4
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−5
0
5
10
15

In [11]:
```
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
```

In [12]:
```
print("Total train examples: {}, total fraud cases: {}, equal to {:.5f} of t
otal cases. ".format(train_x.shape[0], np.sum(train_y), np.sum(train_y)/trai
n_x.shape[0]))
print("Total test examples: {}, total fraud cases: {}, equal to {:.5f} of to
tal cases. ".format(test_x.shape[0], np.sum(test_y), np.sum(test_y)/test_y.s
hape[0]))
```

```
Total train examples: 213605, total fraud cases: 398, equal to 0.00186 of t
otal cases.
Total test examples: 71202, total fraud cases: 94, equal to 0.00132 of tota
l cases.
```

In [14]:
```
'''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_mi
n[-1])
    test_x[:, c] =  (test_x[:, c] - cols_min[-1]) / (cols_max[-1] - cols_min
[-1])'''
```

Out[14]:
```
'cols_max = []\ncols_min = []\nfor c in range(train_x.shape[1]):\n    cols_
max.append(train_x[:,c].max())\n    cols_min.append(train_x[:,c].min())\n
  train_x[:, c] = (train_x[:, c] - cols_min[-1]) / (cols_max[-1] - cols_min
[-1])\n    test_x[:, c] =  (test_x[:, c] - cols_min[-1]) / (cols_max[-1] -
cols_min[-1])'
```

In [15]:
```
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]
```

In [16]:
```
# Parameters
learning_rate = 0.001
training_epochs = 10
batch_size = 256
display_step = 1

# Network Parameters
n_hidden_1 = 15 # 1st layer num features
#n_hidden_2 = 15 # 2nd layer num features
n_input = train_x.shape[1] # MNIST data input (img shape: 28*28)
data_dir = '.'
```

In [19]:
```
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):
    # Encoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.tanh(tf.add(tf.matmul(x, weights['encoder_h1']),
                                   biases['encoder_b1']))
    # Decoder Hidden layer with sigmoid activation #2
    #layer_2 = tf.nn.tanh(tf.add(tf.matmul(layer_1, weights['encoder_h2']),
                                   #biases['encoder_b2']))
    return layer_1


# Building the decoder
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']))
    # Decoder Hidden layer with sigmoid activation #2
    #layer_2 = tf.nn.tanh(tf.add(tf.matmul(layer_1, weights['decoder_h2']),
                                  # biases['decoder_b2']))
    return layer_1

# Construct model
encoder_op = encoder(X)
decoder_op = decoder(encoder_op)

# Prediction
y_pred = decoder_op
# Targets (Labels) are the input data.
y_true = X

# Define batch mse
batch_mse = tf.reduce_mean(tf.pow(y_true - y_pred, 2), 1)

# Define loss and optimizer, minimize the squared error
cost = tf.reduce_mean(tf.pow(y_true - y_pred, 2))
optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(cost)

# TRAIN StARTS
save_model = os.path.join(data_dir, 'temp_saved_model_1layer.ckpt')
saver = tf.train.Saver()

# Initializing the variables
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)
    # Training cycle
    for epoch in range(training_epochs):
        # Loop over all batches
        for i in range(total_batch):
            batch_idx = np.random.choice(train_x.shape[0], batch_size)
            batch_xs = train_x[batch_idx]
            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([optimizer, cost], feed_dict={X: batch_xs})

        # Display logs per epoch step
        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))

    print("Optimization Finished!")

    save_path = saver.save(sess, save_model)
    print("Model saved in file: %s" % save_path)
```

```
Epoch: 0001 cost= 1.307971597 Train auc= 0.953205 Time elapsed= 0:00:01.032
010
Epoch: 0002 cost= 0.700766861 Train auc= 0.954684 Time elapsed= 0:00:01.988
340
Epoch: 0003 cost= 0.728679121 Train auc= 0.955931 Time elapsed= 0:00:02.920
764
Epoch: 0004 cost= 0.587579608 Train auc= 0.957877 Time elapsed= 0:00:03.842
213
Epoch: 0005 cost= 0.571035564 Train auc= 0.956579 Time elapsed= 0:00:04.782
450
Epoch: 0006 cost= 0.509554029 Train auc= 0.955808 Time elapsed= 0:00:05.703
540
Epoch: 0007 cost= 0.757638276 Train auc= 0.955505 Time elapsed= 0:00:06.626
110
Epoch: 0008 cost= 0.415440291 Train auc= 0.956136 Time elapsed= 0:00:07.573
955
Epoch: 0009 cost= 0.394643486 Train auc= 0.956757 Time elapsed= 0:00:08.490
186
Epoch: 0010 cost= 0.540096283 Train auc= 0.957051 Time elapsed= 0:00:09.462
723
Optimization Finished!
Model saved in file: ./temp_saved_model_1layer.ckpt
```

In [20]:
```
save_model = os.path.join(data_dir, 'models/temp_saved_model_1layer.ckpt')
saver = tf.train.Saver()

# Initializing the variables
init = tf.global_variables_initializer()

with tf.Session() as sess:
    now = datetime.now()

    saver.restore(sess, save_model)

    test_batch_mse = sess.run(batch_mse, feed_dict={X: test_x})

    print("Test auc score: {:.6f}".format(auc(test_y, test_batch_mse)))
```

```
INFO:tensorflow:Restoring parameters from ./temp_saved_model_1layer.ckpt
Test auc score: 0.940300
```

Using Model Multilayer Perceptron Model

In [43]:
```
import os
import pandas as pd
import numpy as np
import tensorflow as tf
import random
```

Reading Data

In [44]:
```
df = pd.read_csv('creditcard.csv')
df.shape
```

Out[44]: (284807, 31)

Defining Train data and Test Data

In [45]:
```
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
```

Printing Train and Test Examples

In [46]:
```
print("Total train examples: {}, total fraud cases: {}, equal to {:.5f} of t
otal cases. ".format(train_x.shape[0], np.sum(train_y), np.sum(train_y)/trai
n_x.shape[0]))
print("Total test examples: {}, total fraud cases: {}, equal to {:.5f} of to
tal cases. ".format(test_x.shape[0], np.sum(test_y), np.sum(test_y)/test_y.s
hape[0]))
```

```
Total train examples: 213605, total fraud cases: 398, equal to 0.00186 of t
otal cases.
Total test examples: 71202, total fraud cases: 94, equal to 0.00132 of tota
l cases.
```

Preparing data for train and test model

In [47]:
```
noofdatapoints = train_x.shape[0]
labels_zeros = np.zeros(shape=(noofdatapoints,2))
for i,row in enumerate(labels_zeros):
    row[train_y[i]-1] = 1
train_y = labels_zeros

#####

noofdatapoints_ = test_x.shape[0]
labels_zeros = np.zeros(shape=(noofdatapoints_,2))
for i,row in enumerate(labels_zeros):
    row[test_y[i]-1] = 1
test_y = labels_zeros
```

In [48]:
```
'''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_mi
n[-1])
    test_x[:, c] =  (test_x[:, c] - cols_min[-1]) / (cols_max[-1] - cols_min
[-1])'''
```

Out[48]:
```
'cols_max = []\ncols_min = []\nfor c in range(train_x.shape[1]):\n    cols_
max.append(train_x[:,c].max())\n    cols_min.append(train_x[:,c].min())\n
  train_x[:, c] = (train_x[:, c] - cols_min[-1]) / (cols_max[-1] - cols_min
[-1])\n    test_x[:, c] =  (test_x[:, c] - cols_min[-1]) / (cols_max[-1] -
cols_min[-1])'
```

In [49]:
```
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]
```

Setting Hyperparameters

In [50]:
```
# Parameters
noofdatapoints = train_x.shape[0]
print("Number of data points: ", noofdatapoints)

learning_rate = 0.01
training_epochs = 10
batch_size = 256
display_step = 1
noofbatches = noofdatapoints//batch_size
print("Number of batches: ", noofbatches)

# Network Parameters
n_hidden_1 = 128 #  15 # 1st layer num features
n_hidden_2 = 64 # 2nd layer num features
n_input = train_x.shape[1]
print("Input: ", n_input)
data_dir = '.'
n_output = 2
```

```
Number of data points:  213605
Number of batches:  834
Input:  28
```

Defining X and Y as placeholders

In [51]:
```
X = tf.placeholder(tf.float32, [None, n_input])
Y = tf.placeholder(tf.float32, [None,n_output])
print(X.shape)
print(Y.shape)
```

```
(?, 28)
(?, 2)
```

Initializing Weights and Biases

In [52]:
```
weights = \
{
        'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
        'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
        'out': tf.Variable(tf.random_normal([n_hidden_2, n_output]))
}

biases = \
{
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_output]))
}
```

Defining model using sigmoid activation function

In [53]:
```
def multiperceptron(x):
    w1 = tf.Variable(tf.random_normal([28, 128]))
    b1 = tf.Variable(tf.ones([128])/10.0)
    y1 = tf.nn.sigmoid(tf.matmul(x, w1) + b1)

    w2 = tf.Variable(tf.random_normal([128, 2]))
    b2 = tf.Variable(tf.ones([2])/10.0)
    y2 = tf.nn.sigmoid(tf.matmul(y1, w2) + b2)
    return y2

model = multiperceptron(X)
print(model.shape)
print(Y)
```

```
(?, 2)
Tensor("Placeholder_5:0", shape=(?, 2), dtype=float32)
```

Defining cost, optimizer and accuracy

In [54]:
```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = model
, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_min = optimizer.minimize(loss)

correct_prediction = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

init = tf.global_variables_initializer()
```

Defining getbatch function

In [55]:
```
# A function to get a batch of data
def getbatch(xval, yval, arraylength, batchsize=512):

    randstart = random.randint(0, arraylength-batchsize-1)
    xx = xval[randstart:randstart+batchsize]
    yy = yval[randstart:randstart+batchsize]

    return (xx, yy)
```

Running the session

In [56]:
```
with tf.Session() as sess:
    sess.run(init)

    for epoch in range(training_epochs):
        for batch_count in range(noofbatches):
            batch_x, batch_y = getbatch(train_x,train_y, noofdatapoints, bat
chsize = 512)
            sess.run(train_min, feed_dict={X:batch_x, Y:batch_y})
        # Validate after every epoch
        batch_x, batch_y = getbatch(train_x,train_y, noofdatapoints, batchsi
ze = 512)
        losscalc, accuracycalc = sess.run([loss, accuracy], feed_dict={X:bat
ch_x, Y:batch_y})
        print("Epoch: %d, Loss: %0.4f, Accuracy: %0.4f"%(epoch, losscalc, ac
curacycalc))

    # When the training is complete and you are happy with the result
    accuracycalc = sess.run(accuracy, feed_dict={X: test_x, Y: test_y})
    print("Testing accuracy: %0.4f"%(accuracycalc))
```

```
Epoch: 0, Loss: 0.3133, Accuracy: 1.0000
Epoch: 1, Loss: 0.3140, Accuracy: 0.9980
Epoch: 2, Loss: 0.3152, Accuracy: 0.9980
Epoch: 3, Loss: 0.3140, Accuracy: 0.9980
Epoch: 4, Loss: 0.3133, Accuracy: 1.0000
Epoch: 5, Loss: 0.3133, Accuracy: 1.0000
Epoch: 6, Loss: 0.3133, Accuracy: 1.0000
Epoch: 7, Loss: 0.3133, Accuracy: 1.0000
Epoch: 8, Loss: 0.3167, Accuracy: 0.9941
Epoch: 9, Loss: 0.3133, Accuracy: 1.0000
Testing accuracy: 0.9996
```

In [ ]: