Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CREDIT CARD FRAUD DETECTION

Engineering optimization

Name: Chhavi Jain

Reg. No.: 20BKT0144

ABSTRACT

- Each year, credit card theft costs consumers and financial institutions billions of dollars, and criminals are constantly looking for new laws and strategies to break. Thus, in order to reduce losses, fraud detection systems are becoming crucial for banks and financial institutions.
- Therefore, in this project, we train a model with three distinct sets of optimizers before identifying the optimizer that has the best accuracy and least loss.
- If the fraud detection system is not properly optimized, it may mark a transaction as fraudulent even if it wasn't, putting the customer at danger for that specific credit card company.

AIM:

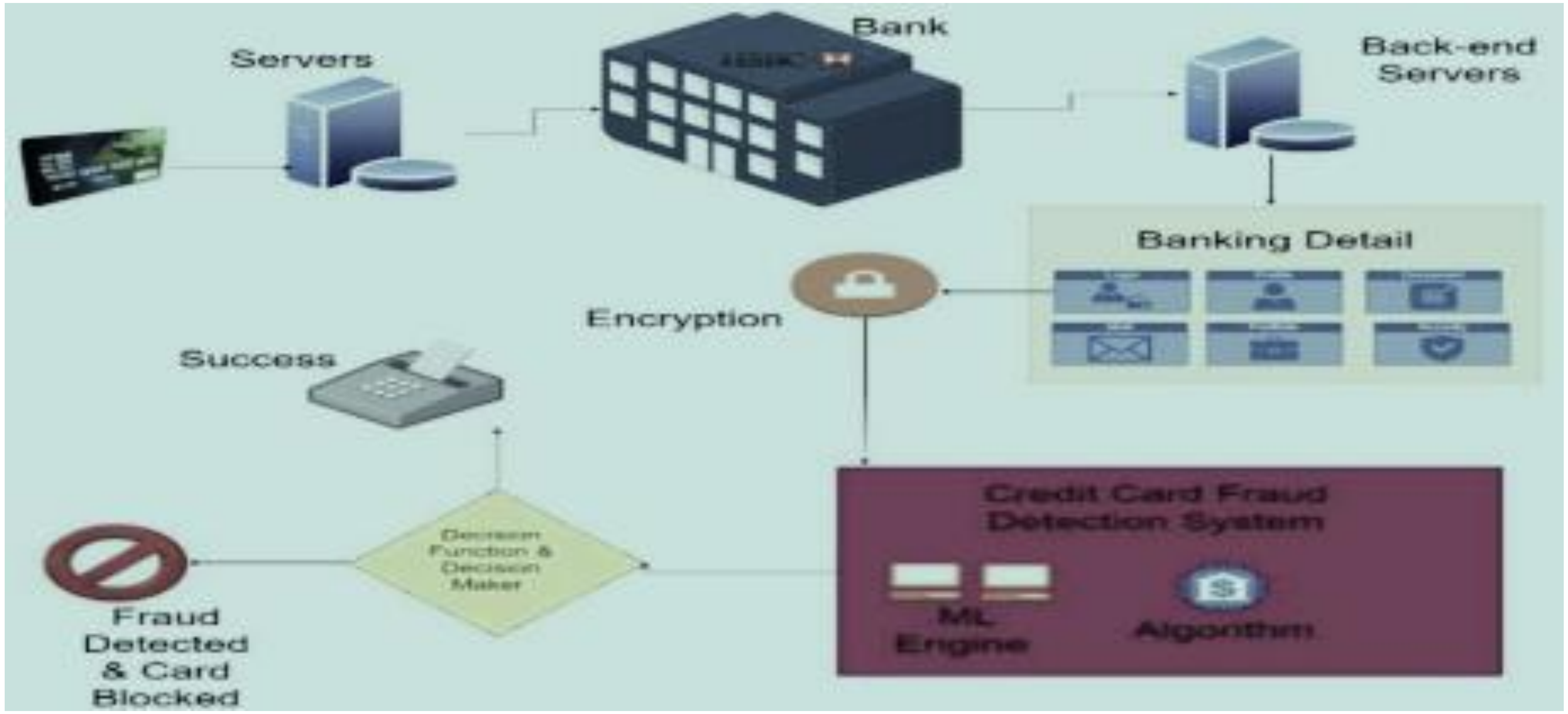
- For this project we will aim for security and honesty of the system. The datasets contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
- We hope to get results within this database which mainly are:
  - ✓ Educating our model to support credit card firms to avoid fraud.
  - ✓ Using three different optimizers, we are able to achieve the stated goal, Finding the most accurate Optimizer by comparing the outcomes of various Optimizers.

CONCEPTS USED:

- For each test in this project, we'll utilize one of three distinct approaches, and we'll compare how accurate each approach is. We will optimize using the following techniques:
  - STOCHASTIC GRADIENT DESCENT METHOD (SGD)
  - ADGRAD (Adaptive gradient algorithm)
  - ADAM OPTIMIZER FOR NEURAL NETWORKS

We will also use JUPYPTER as compiler for python language program to get a better understanding or these concepts while visualising.

OVERVIEW:



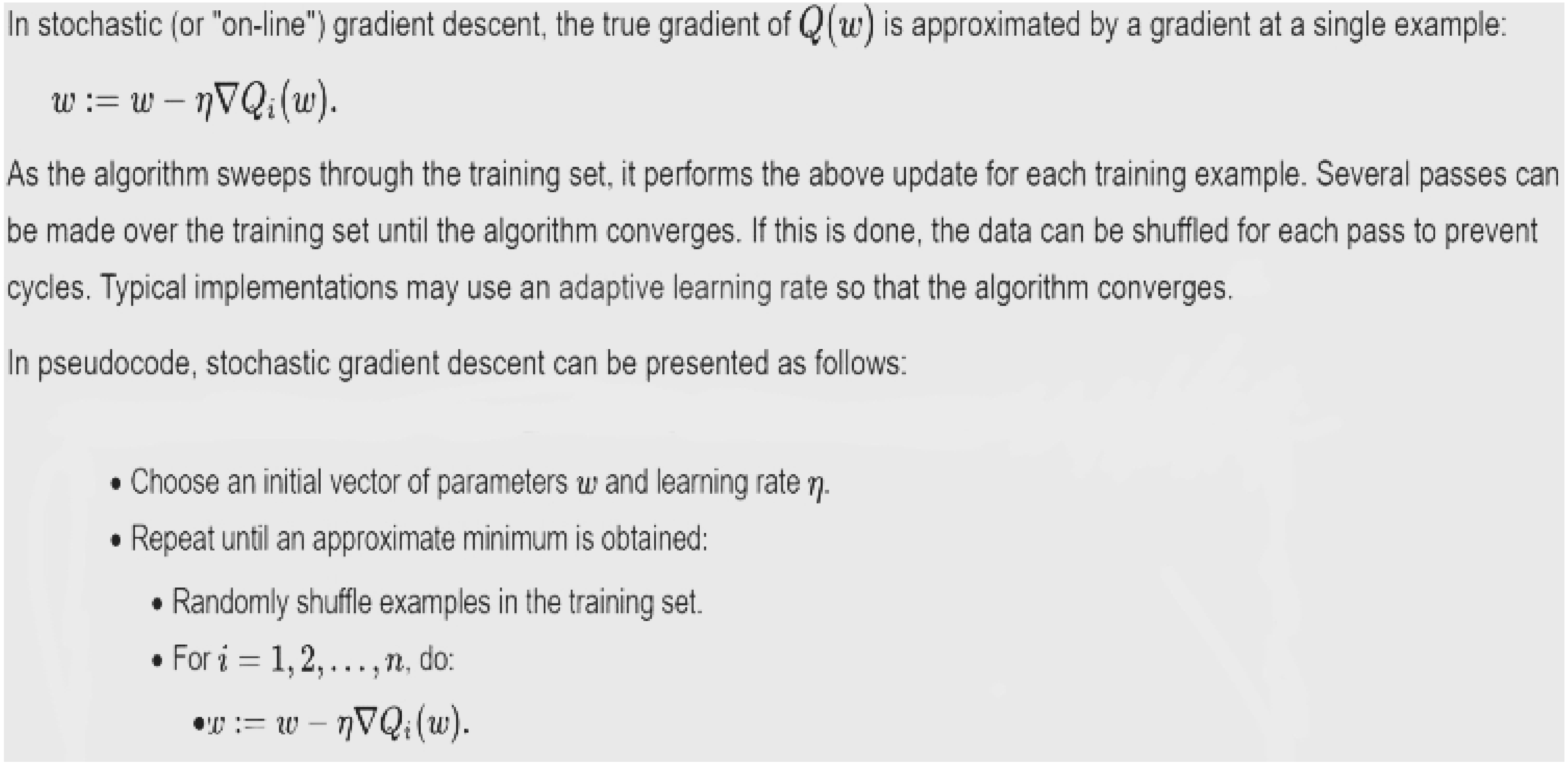
Servers
Bank
Back-end Servers
Banking Detail
Encryption
Success
Credit Card Fraud Detection System
ML Engine
Algorithm
Decision Function & Decision Maker
Fraud Detected & Card Blocked

WHAT IS GRADIENT DESCENT ?

- Gradient, in plain terms means slope or slant of a surface. So gradient descent literally means descending a slope to reach the lowest point on that surface.
- The objective of gradient descent algorithm is to find the value of "x" such that "y" is minimum. "y" here is termed as the objective function that the gradient descent algorithm operates upon, to descend to the lowest point.
- **"Gradient descent is an iterative algorithm, that starts from a random point on a function and travels down its slope in steps until it reaches the lowest point of that function."**

STOCHASTIC GRADIENT DESCENT METHOD:

- An iterative technique for maximizing an objective function with acceptable smoothness qualities is stochastic gradient descent, or SGD (e.g. differentiable or sub differentiable).
- It might betaken into account as a stochastic approximation of gradient descent because it substitutes the real gradient (derived from the complete collection of facts) by an estimate thereof (determined from aa data collection that was chosen at random).
- This decreases the computing burden, particularly for high-dimensional optimization problems. Load, resulting in quicker repetitions at the expense of a lower convergence rate.



In stochastic (or "on-line") gradient descent, the true gradient of $Q(w)$ is approximated by a gradient at a single example:

$$w := w - \eta \nabla Q_i(w).$$

As the algorithm sweeps through the training set, it performs the above update for each training example. Several passes can be made over the training set until the algorithm converges. If this is done, the data can be shuffled for each pass to prevent cycles. Typical implementations may use an adaptive learning rate so that the algorithm converges.

In pseudocode, stochastic gradient descent can be presented as follows:

```
• Choose an initial vector of parameters $w$ and learning rate $\eta$.
• Repeat until an approximate minimum is obtained:
    • Randomly shuffle examples in the training set.
    • For $i = 1, 2, \ldots, n$, do:
        • $w := w - \eta \nabla Q_i(w)$.
```

WHERE CAN WE POTENTIALLY INDUCE RANDOMNESS IN OUR GRADIENT DESCENT ALGORITHM?

"Stochastic", in plain terms means "random".

- It is while selecting data points at each step to calculate the derivatives. SGD randomly picks one data point from the whole data set at each iteration to reduce the computations enormously.
- It is also common to sample a small number of data points instead of just one point at each step and that is called "mini-batch" gradient descent. Mini-batch tries to strike a balance between the goodness of gradient descent and speed of SGD.

ADAGRAD (ADAPTIVE GRADIENT ALGORITHM):

- Adagrad is a method for gradient-based optimization that adjusts the learning rate to the parameters, producing smaller updates (i.e., low learning rates) for parameters connected to often occurring features and bigger updates (i.e., high learning rates) for parameters. Connected to sporadic characteristics.
- Adagrad's principle is to apply various learning rates based on iteration for each parameter. Different learning rates are required because the learning rate for sparse features parameters must be higher than the learning rate for dense features parameters. Due to the less frequent occurrence of sparse features.
- This makes it a good choice for handling sparse data. Adagrad significantly increased the SGD's resilience and used it to train Google's massive neural networks, which, among other things, began to detect cats in videos on YouTube. Additionally, Pennington et al. Training GloVe word embeddings is advised since infrequent words need substantially greater updates than common ones.

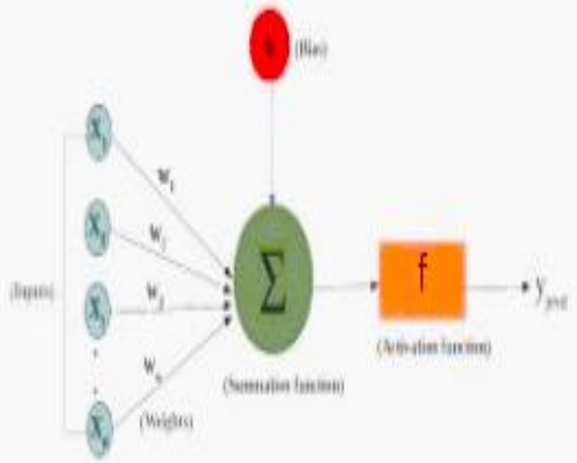
$$\text{SGD} \Rightarrow w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

$$\text{Adagrad} \Rightarrow w_t = w_{t-1} - \eta_t' \frac{\partial L}{\partial w_{t-1}}$$

$$\text{where } \eta_t' = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

ε is a small +ve number to avoid divisibilty by 0

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_{t-1}}\right)^2 \quad \textit{summation of gradient square}$$

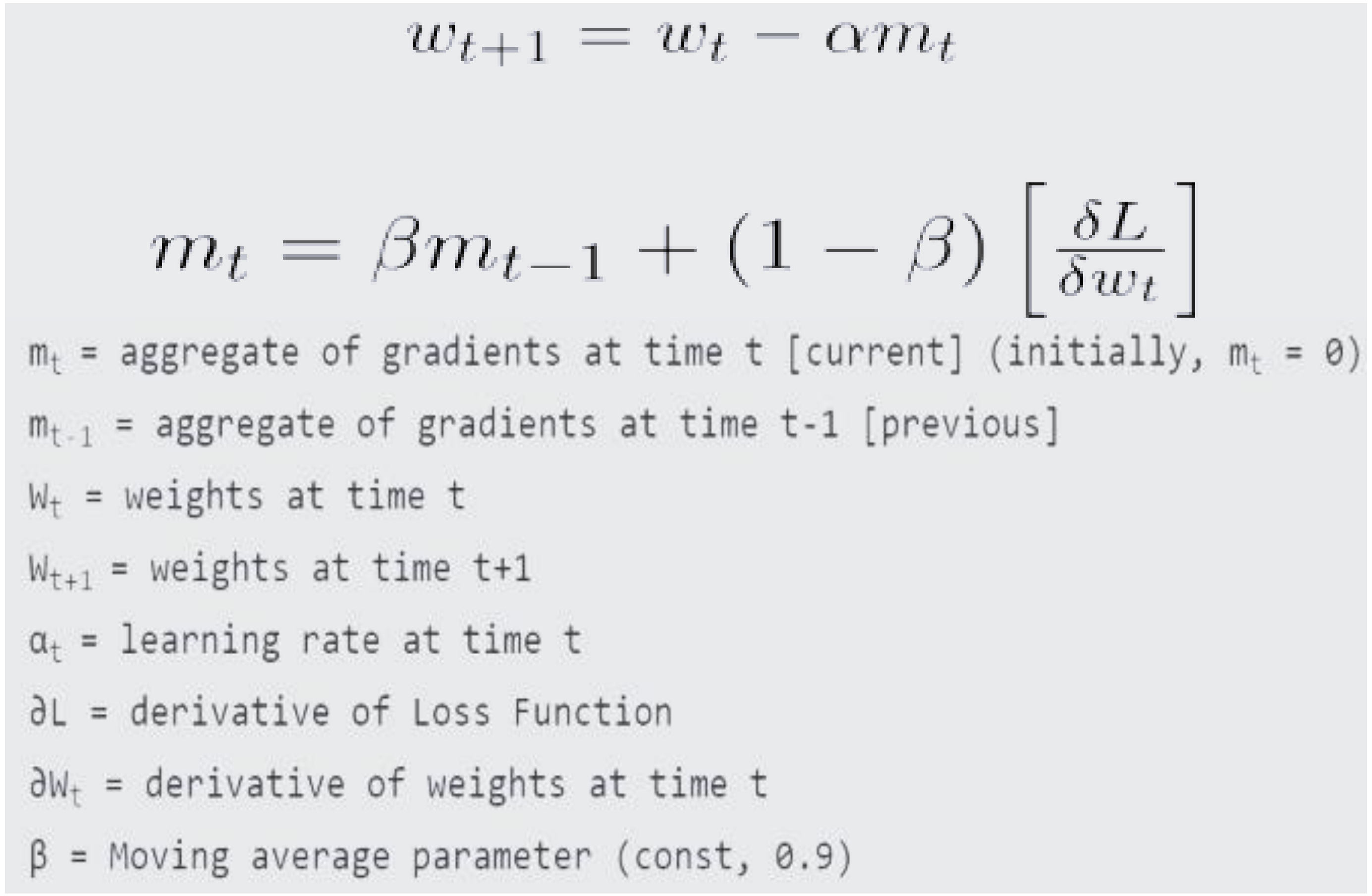


The learning rate has been altered in the image mentioned Adagrad optimizer equation such that it will naturally decrease as time goes on because the total of the previous gradient square will always increase after every time step.

ADAM OPTIMIZER FOR NEURAL NETWORKS:

- Adam is an optimization technique that may be used to iteratively update network weights based on training data instead of the conventional stochastic gradient descent method.
- An approach for gradient descent optimization is called adaptive moment estimation. When dealing with complex problems requiring a lot of data or factors, the strategy is incredibly effective. It is effective and uses little memory. It combines the gradient descent with momentum method and the RMSP algorithm, intuitively.
- Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled "Adam: A Method for Stochastic Optimization". I will quote liberally from their paper in this post, unless stated otherwise.

- By using the "exponentially weighted average" of the gradients, this approach (Adam) is used to speed up the gradient descent process. The technique converges quicker pace to the minima when averages are used.



$$w_{t+1} = w_t - \alpha m_t$$

$$m_t = \beta m_{t-1} + (1 - \beta)\left[\frac{\delta L}{\delta w_t}\right]$$

m_t = aggregate of gradients at time t [current] (initially, m_t = 0)

m_{t-1} = aggregate of gradients at time t-1 [previous]

W_t = weights at time t

W_{t+1} = weights at time t+1

α_t = learning rate at time t

∂L = derivative of Loss Function

∂W_t = derivative of weights at time t

β = Moving average parameter (const, 0.9)

MATHEMATICAL ASPECT OF ADAM OPTIMIZER

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t} \right] v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta w_t} \right]^2$$

- Since mt and vt have both initialized as 0 (based on the above methods), it is observed that they gain a tendency to be ‘biased towards 0’ as both β1 & β2 ≈ 1. This Optimizer fixes this problem by computing ‘bias-corrected’ mt and vt . This is also done to control the weights while reaching the global minimum to prevent high oscillations when near it. The formulas used are:

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t} \widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

- Intuitively, we are adapting to the gradient descent after every iteration so that it remains controlled and unbiased throughout the process, hence the name Adam.

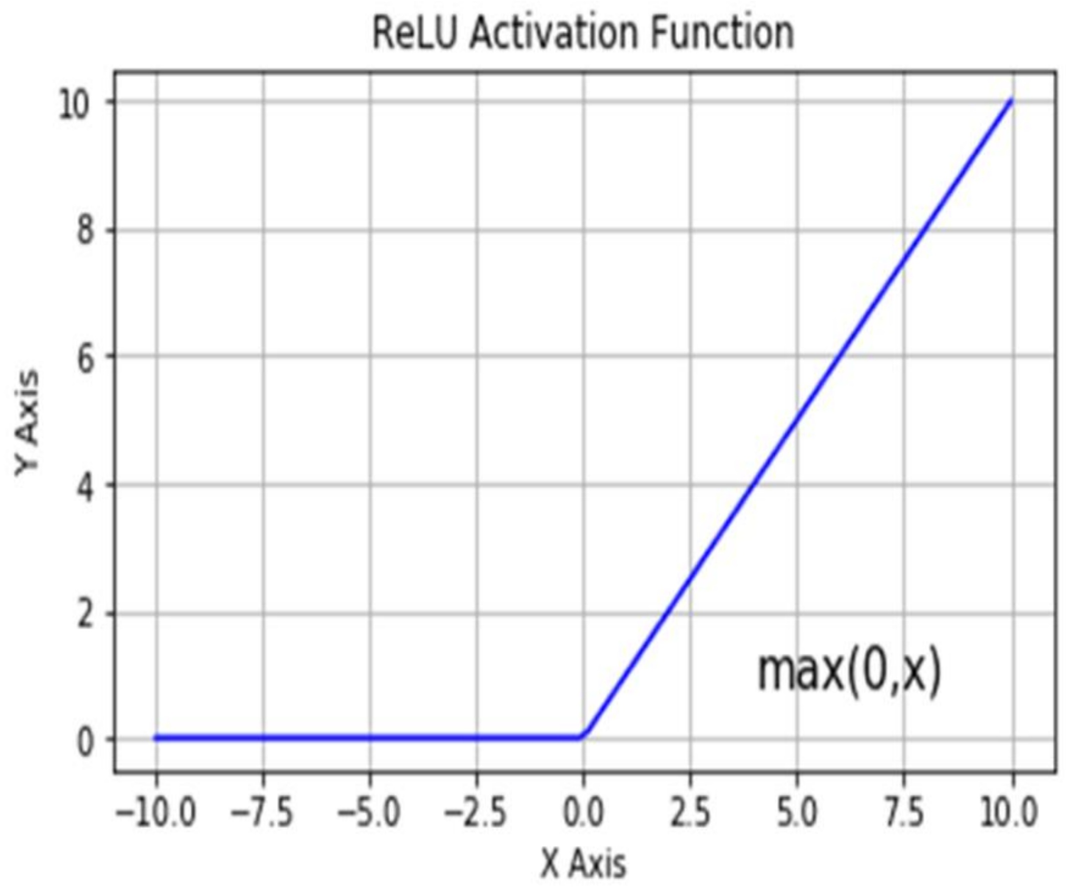
- **What do you mean by weights?**

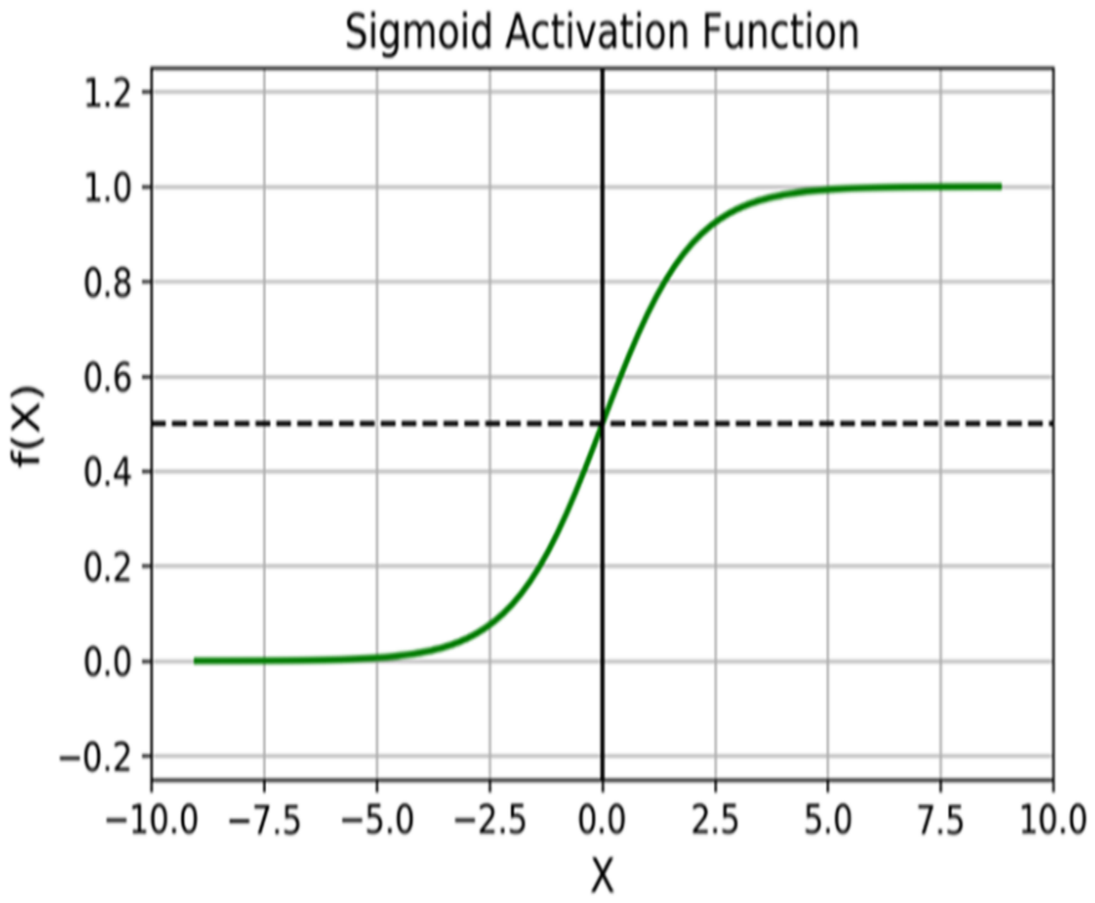
Weight is the parameter within a neural network that transforms input data within the network's hidden layers. A neural network is a series of nodes, or neurons. Within each node is a set of inputs, weight, and a bias value.

- **What are RelU and Sigmoid activation function?**

ReLU (Rectified Linear Unit) is an activation function that maps any number to zero if it is negative, and otherwise maps it to itself. The ReLU function has been found to be very good for networks with many layers because it can prevent vanishing gradients when training deep networks.

Sigmoid is a type of activation function that maps any number between 0 and 1, inclusive, to itself.


ReLU Activation Function
max(0,x)
Y Axis
X Axis
10
8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Sigmoid Activation Function
f(X)
X
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

- **What is epochs** ?

An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the machine learning model. Another way to define an epoch is the number of passes a training dataset takes around an algorithm.

- **What does hidden layer means?**

A hidden layer in an artificial neural network is a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function.

SPECIFICATION

- Hidden Layers used- 1
- Activization Functions- ReLu & Sigmoid
- Epochs- 20
- Optimizers- Adagrad, SGD, ADAM using tensorflow

CODE:

```
#importing various extension for optimiser
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
from tensorflow.keras.models import Sequential,model
from tensorflow.keras.layers import BatchNormalization,Dropout,Dense,Flatten,Conv1D
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import Adagrad
#Data file
credit_df = pd.read_csv("creditcard.csv")
credit_df.head(5)
credit_df.info()
credit_df.describe()
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
```

```
#standard scaling
credit_df['std_Amount'] = scaler.fit_transform(credit_df['Amount'].
values.reshape (-1,1))
#removing Amount
credit_df = credit_df.drop("Amount", axis=1)
sns.countplot(x="Class", data=credit_df)
import imblearn
from imblearn.under_sampling import RandomUnderSampler
undersample = RandomUnderSampler(sampling_strategy=0.5)
cols = credit_df.columns.tolist()
cols = [c for c in cols if c not in ["Class"]]
target = "Class"
```

```
#define X and Y
X = credit_df[cols]
Y = credit_df[target]
#undersample
X_under, Y_under = undersample.fit_resample(X, Y)
from pandas import DataFrame
test = pd.DataFrame(Y_under, columns = ['Class’])
#visualizing undersampling results
fig, axs = plt.subplots(ncols=2, figsize=(13,4.5))
sns.countplot(x="Class", data=credit_df, ax=axs[0])
sns.countplot(x="Class", data=test, ax=axs[1])
```

```
fig.suptitle("Class repartition before and after undersampling")
a1=fig.axes[0]
a1.set_title("Before")
a2=fig.axes[1]
a2.set_title("After")
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_under, Y_unde
r, test_size=0.2, random_state=1)
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],1)
```

```
model=Sequential()
model.add(Conv1D(32,2,activation='relu',input_shape=X_train[0].shape))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Conv1D(64,2,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid’))
model.summary()
model.compile(optimizer=Adam(learning_rate=0.0001),loss='binary_crossentropy',metr
ics=['accuracy’])
history = model.fit(X_train,y_train,epochs=20,validation_data=(X_test,y_test))
```

```
model2=Sequential()
model2.add(Conv1D(32,2,activation='relu',input_shape=X_train[0].shape))
model2.add(BatchNormalization())
model2.add(Dropout(0.2))
model2.add(Conv1D(64,2,activation='relu'))
model2.add(BatchNormalization())
model2.add(Dropout(0.5))
model2.add(Flatten())
model2.add(Dense(64,activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(1,activation='sigmoid'))
model2.summary()
model2.compile(optimizer=SGD(learning_rate=0.0001),loss='binary_crossentropy',metri
cs=['accuracy'])
history2 = model2.fit(X_train,y_train,epochs=20,validation_data=(X_test,y_test))
```

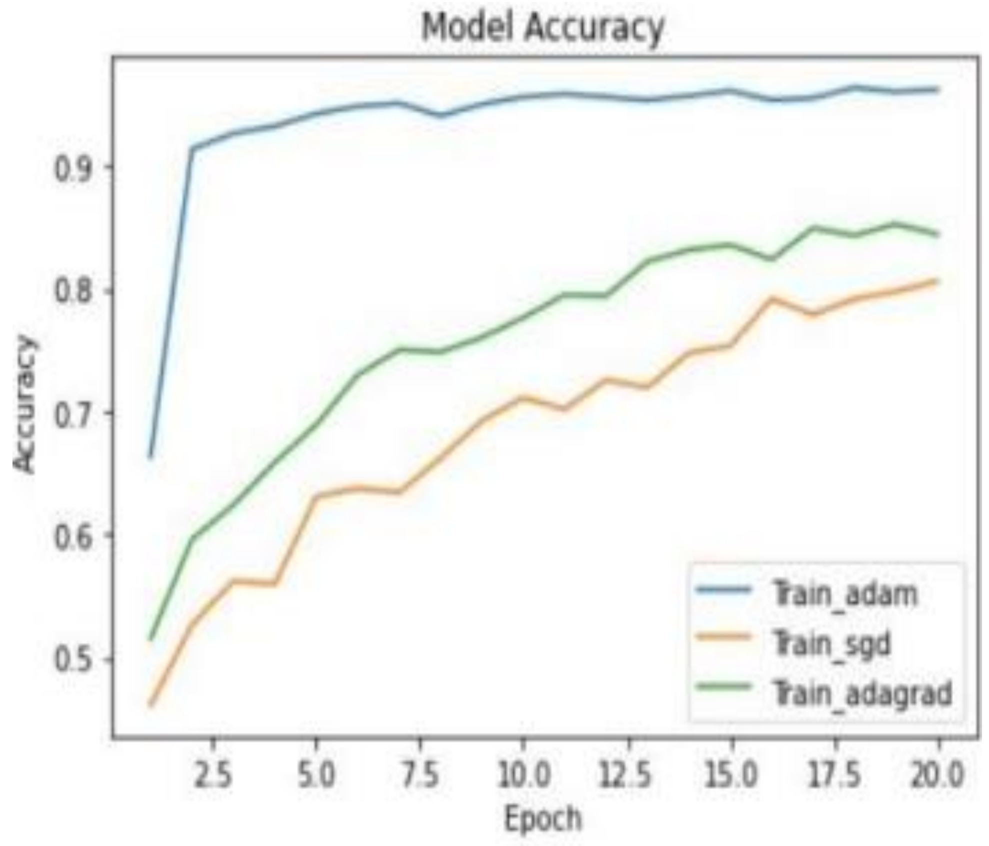
```
def plotLearningCurve(history,epochs):
epochRange = range(1,epochs+1)
plt.plot(epochRange,history.history['accuracy'])
plt.plot(epochRange,history2.history['accuracy'])
plt.plot(epochRange,history.history['val_accuracy'])
plt.plot(epochRange,history2.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train1','Train2','Validation1','Validation2'],loc='upper left')
plt.show()
plt.plot(epochRange,history.history['loss'])
plt.plot(epochRange,history2.history['loss'])
plt.plot(epochRange,history.history['val_loss'])
plt.plot(epochRange,history2.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train1', 'Tain2','Validation1', 'Validation2'],loc='upper left')
plt.show()
```

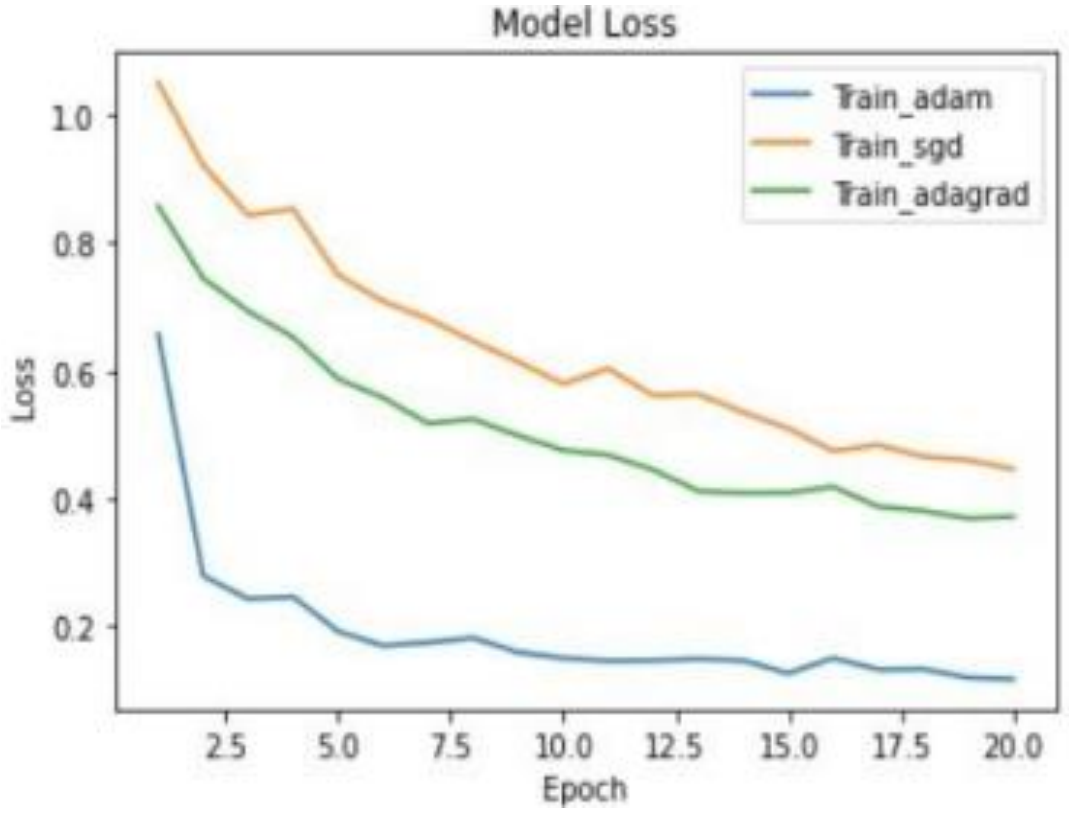
```
model3=Sequential()
model3.add(Conv1D(32,2,activation='relu',input_shape=X_train[0].shape))
model3.add(BatchNormalization())
model3.add(Dropout(0.2))
model3.add(Conv1D(64,2,activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))
model3.add(Flatten())
model3.add(Dense(64,activation='relu'))
model3.add(Dropout(0.5))
model3.add(Dense(1,activation='sigmoid'))
model3.summary()
model3.compile(optimizer=Adagrad(learning_rate=0.0001),loss='bi
nary_crossentropy',metrics=['accuracy'])
history3 = model3.fit(X_train,y_train,epochs=20,validation_data=(X_test,y_test))
```

```
def plotLearningCurve(history,epochs):
epochRange = range(1,epochs+1)
plt.plot(epochRange,history.history['accuracy'])
plt.plot(epochRange,history2.history['accuracy'])
plt.plot(epochRange,history3.history['accuracy'])
plt.plot(epochRange,history.history['val_accuracy'])
plt.plot(epochRange,history2.history['val_accuracy'])
plt.plot(epochRange,history3.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy’)
plt.legend(['Train_adam','Train_sgd','Train_adagrad','Validation_adam','Validation_sgd','V
alidation_adagrad'],loc='botom right')
```

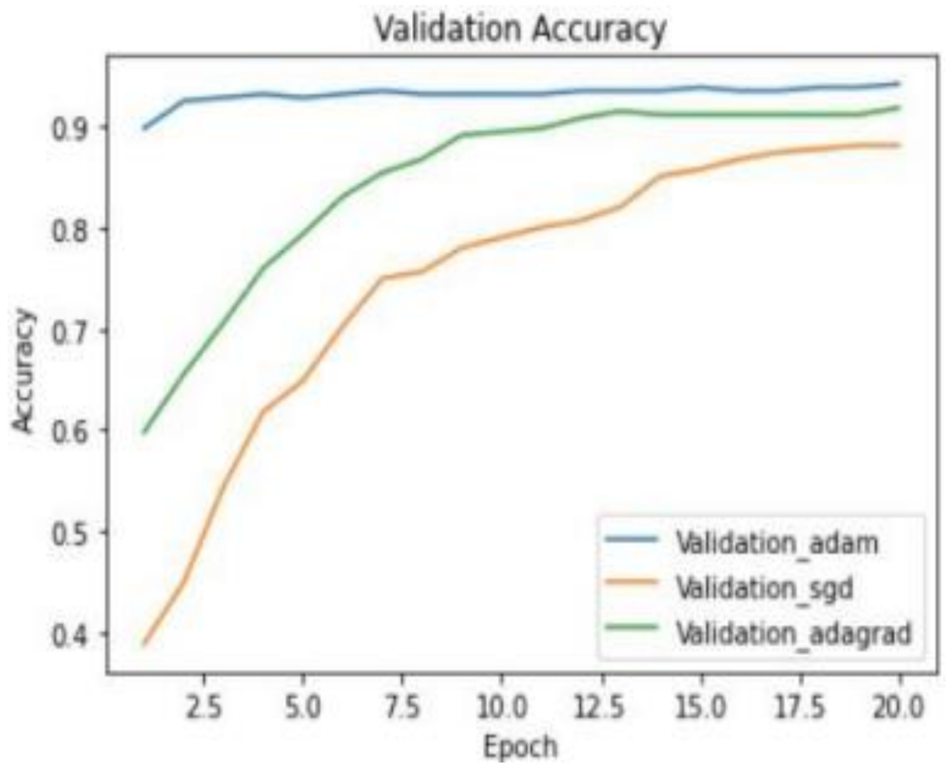
```
Plt.show()
plt.plot(epochRange,history.history['loss'])
plt.plot(epochRange,history2.history['loss'])
plt.plot(epochRange,history3.history['loss'])
plt.plot(epochRange,history.history['val_loss'])
plt.plot(epochRange,history2.history['val_loss'])
plt.plot(epochRange,history3.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train_adam','Train_sgd','Train_adagrad','Validation_adam','Validation_sgd','V
alidation_adagrad'],loc='upper right')
plt.show()
plotLearningCurve(history,20)
```

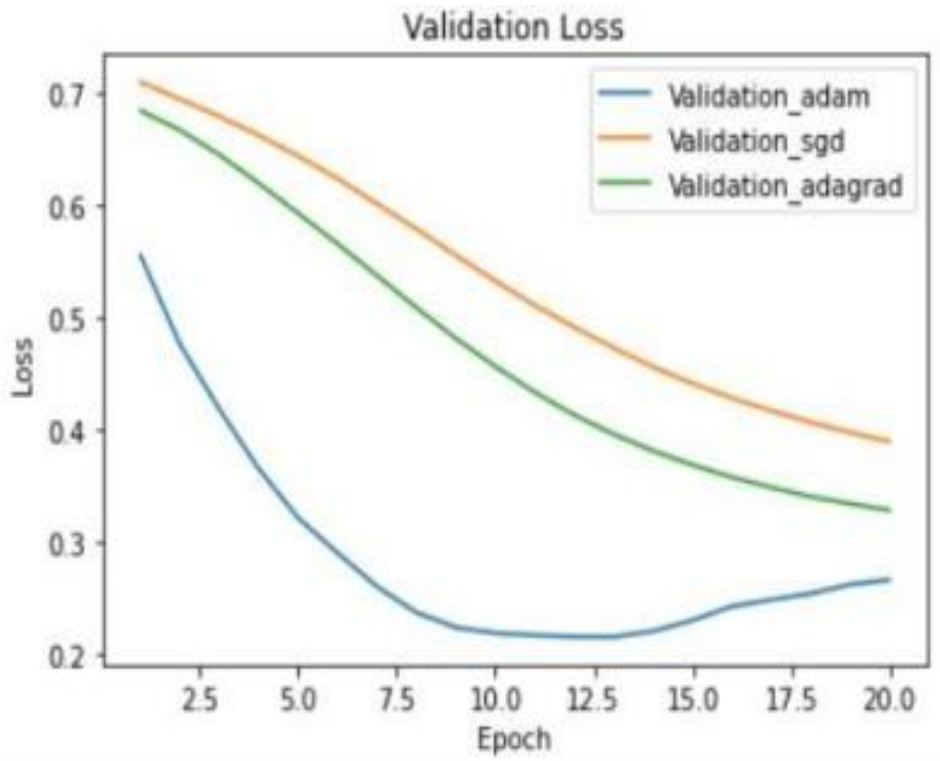
TRANING DATA SET





VALIDATION DATASET





RESULTS:

- In any optimization technique the rate of convergence depends on how we efficiently update the weights and also on how efficiently we optimize the step size.
- In Adagrad, if the value of the gradient square,

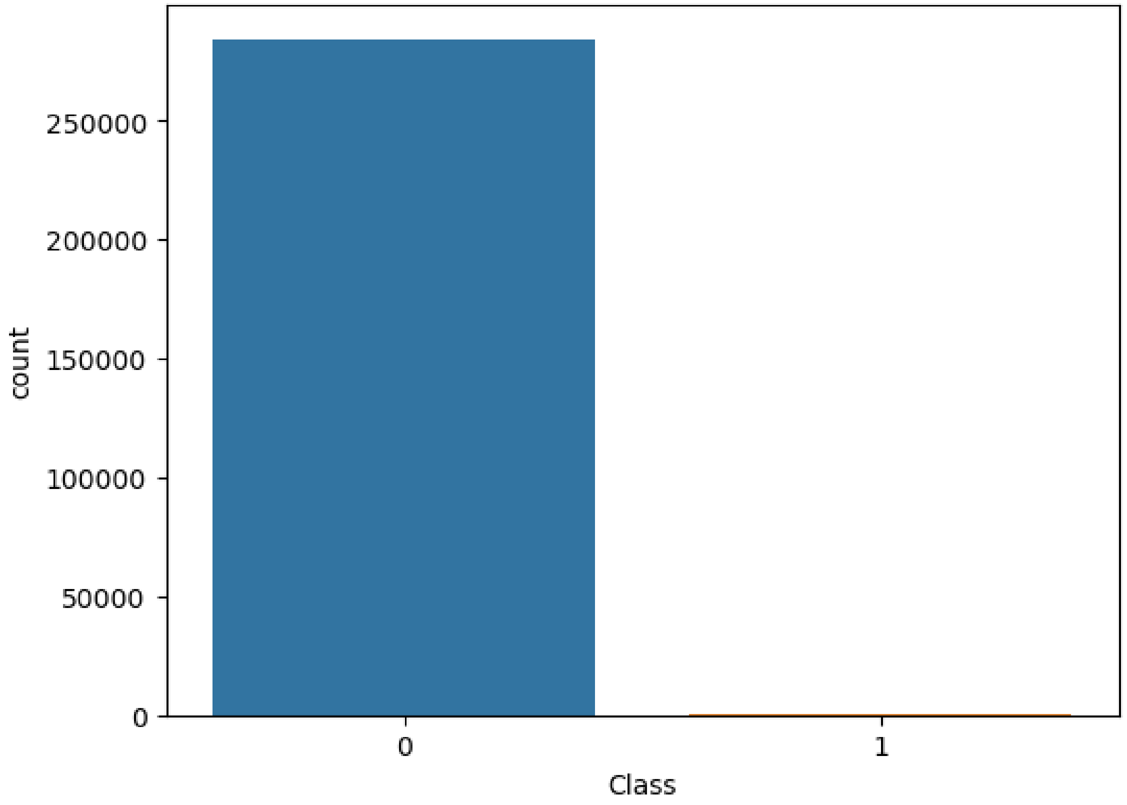
$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial \mathrm{L}}{\partial \mathrm{w}_{t-1}}\right)^2 \quad \textit{summation of gradient square}$$

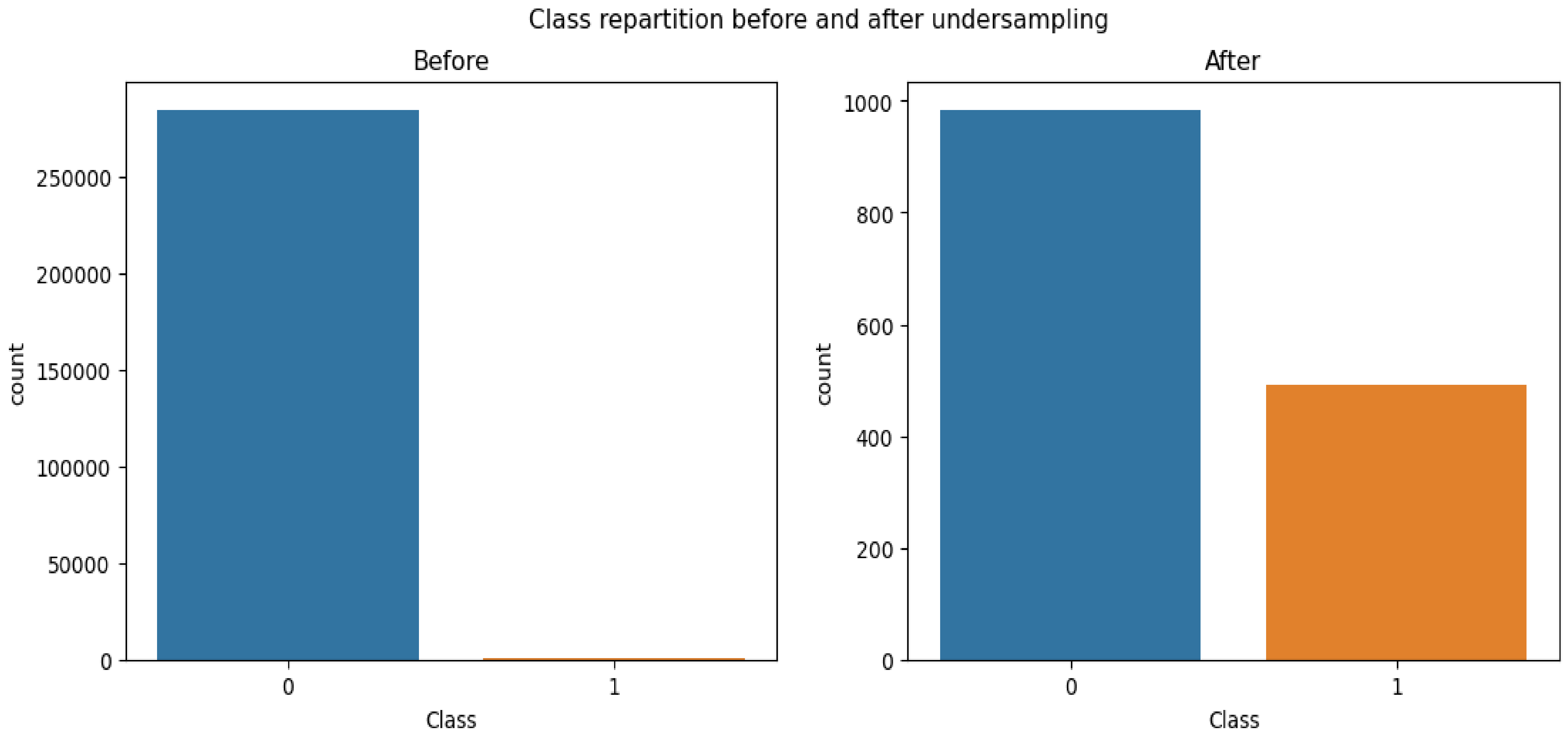
becomes very high, then the value of learning rate,

$$\eta_t^{'} = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

becomes very low. This results in the poor optimization weights.

- As in the Adam, which is the combination of Adagrad and RMSProp, ensures the efficient optimization on the weights.
- So, its because of these reasons Adam is giving comparatively better results than Adagrad and SGD


250000
200000
150000
100000
50000
0
count
0
1
Class


Class repartition before and after undersampling
Before
After
count
Class
0
1
250000
200000
150000
100000
50000
1000
800
600
400
200

CONCLUSION

- From the above output slide we can clearly see that the best accuracy and the minimal loss is shown by the Adam Optimizer which is then followed by Adagard and then SGD.
- It clearly proves that the best optimizer for this Dataset isAdam and it will give the highest efficiency for successful fraud detections

REFERENCES:

- Credit Card Fraud Detection Based on Transaction Behaviour -by John Richard D. Kho, Larry A. Vea published by Proc. of the 2017 IEEE Region 10 Conference (TENCON), Malaysia, November 5-8, 2017.
- CLIFTON PHUA1, VINCENT LEE1, KATE SMITH1 & ROSS GAYLER2 A Comprehensive Survey of Data Mining-based Fraud Detection Research published by School of Business Systems, Faculty of Information Technology, Monash University, Wellington Road,Clayton, Victoria 3800, Australia.
- Research on Credit Card Fraud Detection Model Based on Distance Sum by Wen-Fang YU and Na Wang published by 2009 International Joint Conference on Artificial Intelligence.
- Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy published by IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL. 29, NO. 8, AUGUST 2018 .
- Credit Card Fraud Detection-by Ishu Trivedi, Monika, Mrigya, Mridushi published by International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 1, January 2015

THANK YOU

Worked under:

prof. RAGHUNATHAN T