Major Project Report

To Study and Implement Optimization Algorithms

Project report submitted in partial fulfilment of the requirement for the degree of

Bachelor of Technology

IN

Computer Science and Engineering/Information Technology

BY

Bhavye Sharma (171473)

UNDER THE SUPERVISION OF

Dr.Rajni Mohana

to

Department of Computer Science Engineering and Information Technology **Jaypee University of Information Technology, Waknaghat, Solan -173234, Himachal Pradesh**

Candidate's Declaration

I hereby declare that the work presented in this report entitled "Optimization Algorithms " in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer Science &Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2020 to December 2020 under the supervision of **Dr Rajni Mohana** (Designation and Department name).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Bhavye

Bhavye Sharma

171473

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

[illegible]

Dr Rajni Mohana

Associate Professor

Department of Computer Science & Engineering and Information Technology

Dated:

ACKNOWLEDGEMENT

I wish to express our sense of gratitude towards Dr. Rajni Mohana Department of Computer Science & Engineering, Jaypee University of Information Technology, Waknaghat, my guide, for giving me this wonderful opportunity to work with her. I am grateful for her constant encouragement, motivation, cooperation and support which helped me in finishing this project successfully. Without her expert guidance this would not have been possible. I am also grateful to the people whose works we have referred and the details regarding the same are mentioned in the references. I would also like to thank all our friends and lab assistants for extending their help and support at times when it was needed.

Table of Contents S.NO

Abstract:

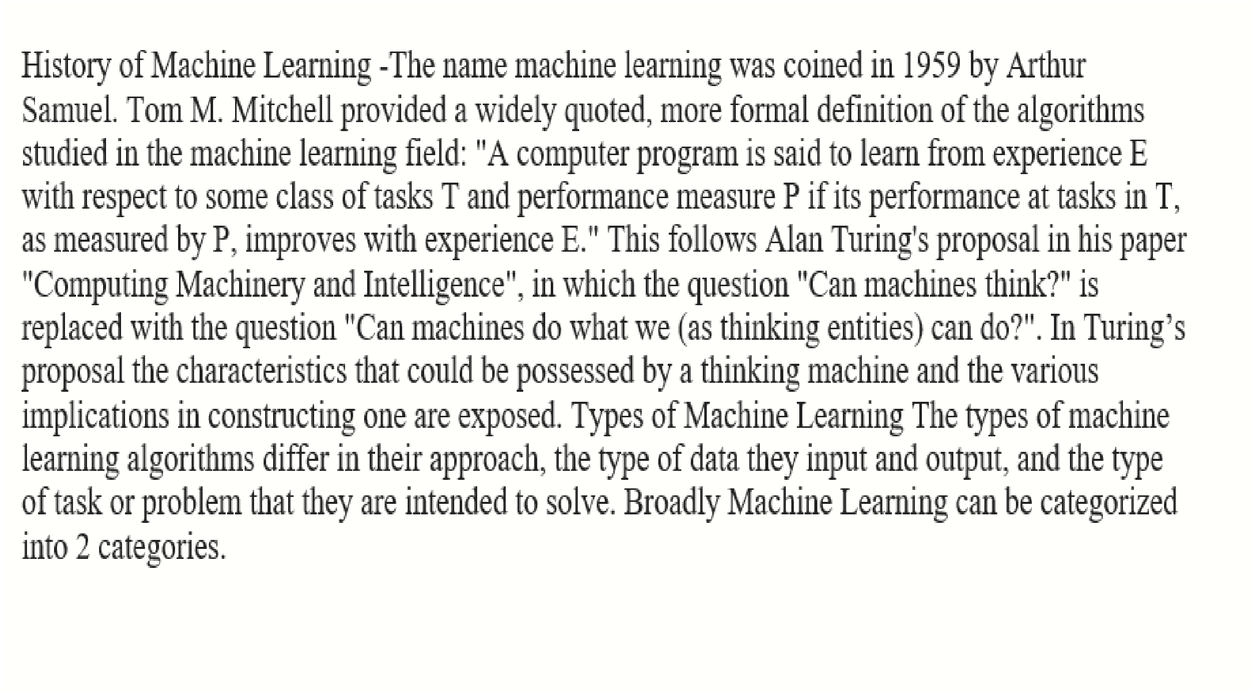
Designing and implementing optimization algorithms in machine and deep learning tends to be very challenging, partly due to the complexity and highly nonlinearity of the problem of interest, partly due to stringent design codes in engineering practice. Conventional algorithms are not the best tools for highly nonlinear global optimization, as they are local search algorithms, and thus often miss the global optimality. In addition, design solutions have to be robust, subject to uncertainty in parameters and tolerance of available components and materials.The optimization algorithms have been implemented.

Chapter 1: Introduction

1.1 Introduction:

To Know about Optimization Algorithms First we should have some background knowledge of Machine Learning and its Types.

Machine Learning is science of making computers learn without being explicitly programmed. It is closely related to the computational statistic, which focuses on making the predictions using the computers. Its application across the business problems, machine learning can also be referred to as predicting analysis. Machine Learning is closely related to the computing statistics. Machine Learning focuses on the development of the computer programs that can be accesed data and can be used to learn themselves. The process of learning which begins with observations or data, such as the examples and the instruction, in order to look for the patterns in the data which makes better decisions for future basis on the examples which are provided. The primary aim is allowing computers to learn automatically without the human interventions or the assistances and adjusting action.


History of Machine Learning -The name machine learning was coined in 1959 by Arthur Samuel. Tom M. Mitchell provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with the question "Can machines do what we (as thinking entities) can do?". In Turing's proposal the characteristics that could be possessed by a thinking machine and the various implications in constructing one are exposed. Types of Machine Learning The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve. Broadly Machine Learning can be categorized into 2 categories.

Types of Machine Learning -

I. Supervised Learning

II. Unsupervised Learning

Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly.

Types of Machine Learning -

I. Supervised Learning

II. Unsupervised Learning

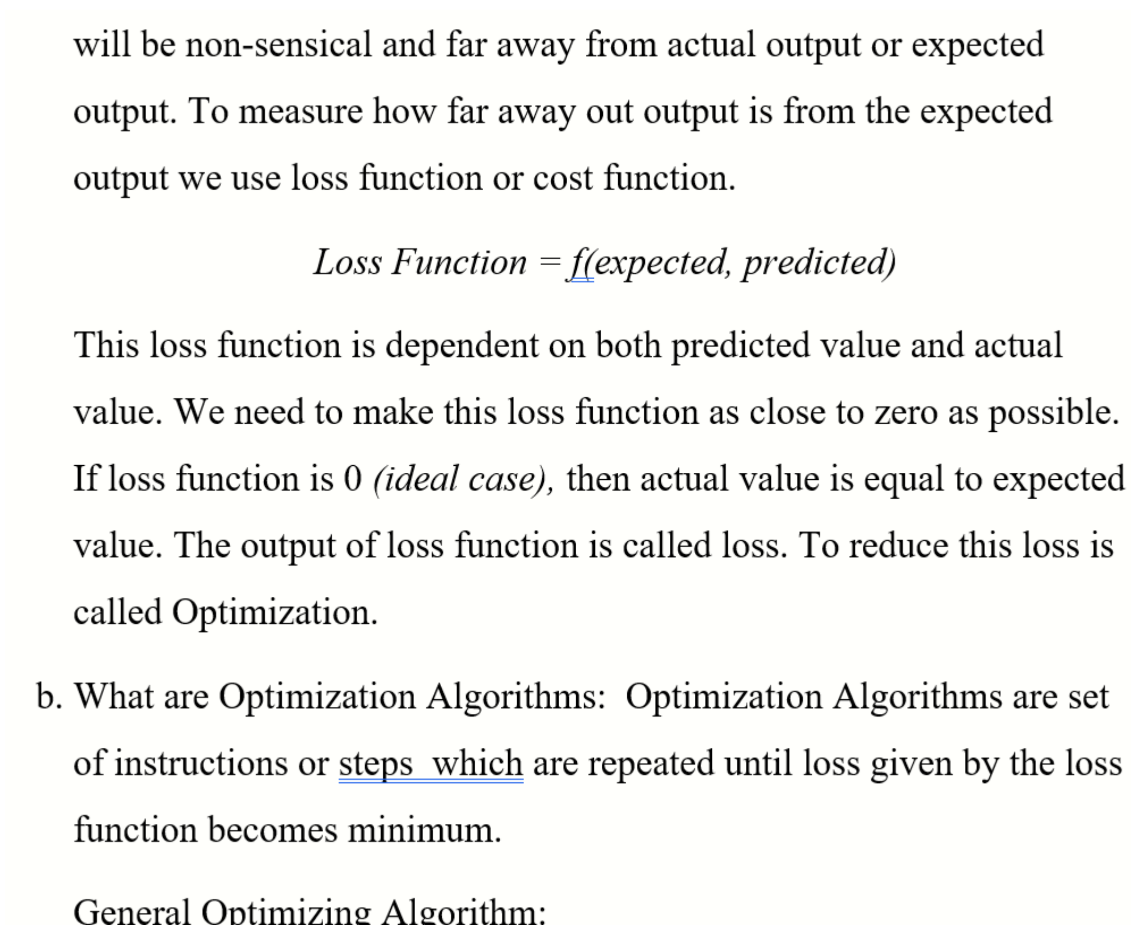
Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly.

1.Supervised Learning -

Supervised Learning is a type of the learning were the data set is given and we know how correct output looks like, having idea that their is a relationship between the input and output. This is a learning a functions which maps inputs to outputs basis on the examples inputs outputs pair. It means that the function from labelling train data consists of set of train examples. Supervised learning problems are categorization .

2.Unsupervised Learning Unsupervised Learning is a type of learning that allows us to approach problems with little or no idea what our problem should look like. We can derive structures by clustering data basis on the relationships amongst different variables in the data.

a. What is Optimization: When we make machine learning model it won't work perfectly on the first go. The output given by the model



will be non-sensical and far away from actual output or expected output. To measure how far away out output is from the expected output we use loss function or cost function.

Loss Function = f(expected, predicted)

This loss function is dependent on both predicted value and actual value. We need to make this loss function as close to zero as possible. If loss function is 0 *(ideal case),* then actual value is equal to expected value. The output of loss function is called loss. To reduce this loss is called Optimization.

b. What are Optimization Algorithms: Optimization Algorithms are set of instructions or steps which are repeated until loss given by the loss function becomes minimum.

General Optimizing Algorithm:

```
b.  predicted_value = model(prams)

    Loop(start):

    predicted_value = changes in model(params)

    Recompute loss: LossFunction(predicted_value, actual_value)

    If loss is less than equal to a threshold: stop loop

    Loop(end)
```

1.2 Problem Statement: To study and implement different optimization algorithms and their working to converge to global minima.

Objective -Implementing various Optimization Algorithms in Machine Learning and understanding the working of the same.

Implemented

1) Momentum -Mini Batch
2) ->Gradient Descent(Batch Gradient)
3) RMSpropagation
4) Adagrad(Adaptive Gradient)
5) Adam

1.3 Objective: To compare different optimization algorithms using graphs and loss.

1.4 Methodology:

i) Construct a dummy and balanced dataset.
ii) Design a Linear Regression algorithm.
iii) Optimize Linear Regression using different optimization techniques.

iv) Plot graphs.

System Design-

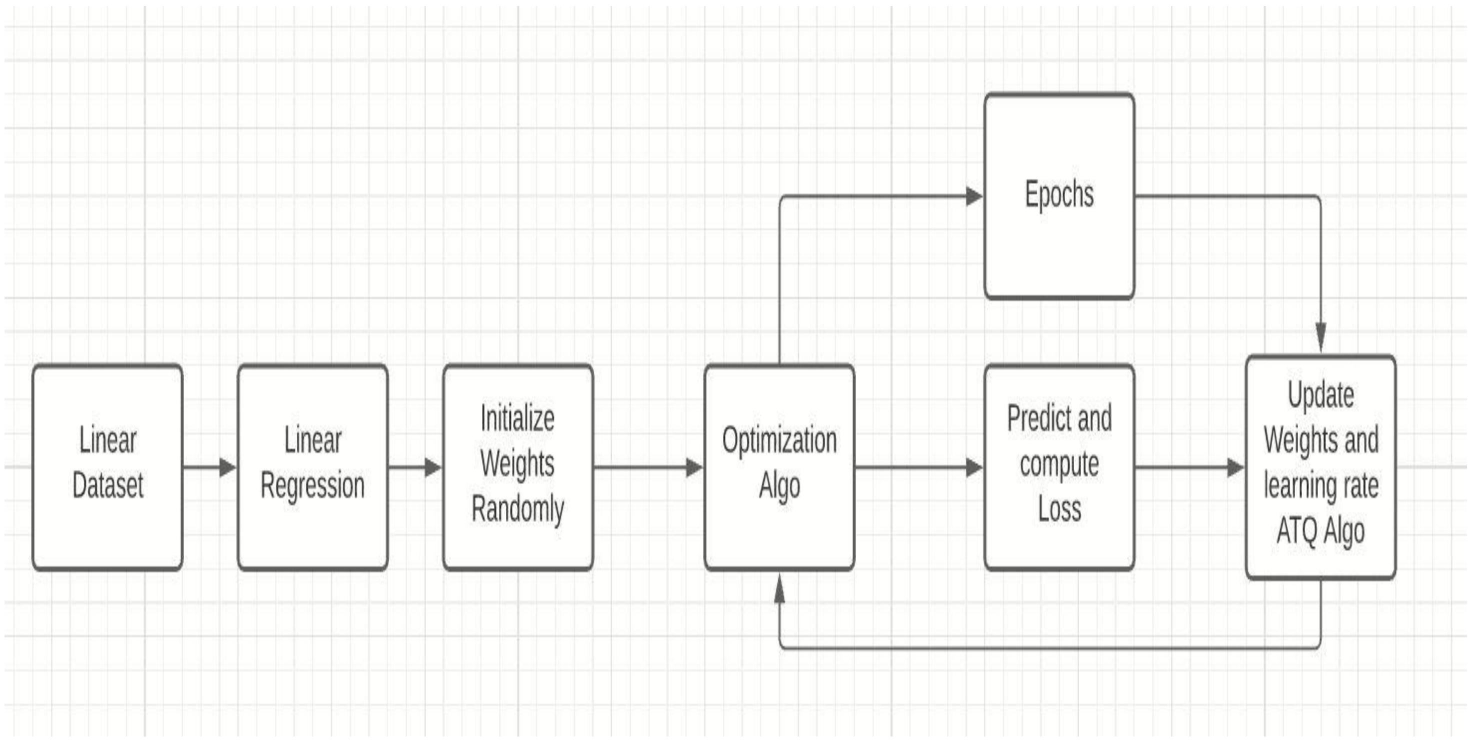


Fig.1

Chapter 2: Literature Survey

1. Hands on Machine Learning using Python by Aurelien Gueron : About Linear Regression and datasets.
2. d2l.ai : About Optimization and optimization algorithms.
3. www.towardsdatascience.com : Working on different algorithms in detail.
4. Bai, Qinghai. "Analysis of particle swarm optimization algorithm." *Computer and information science* 3.1 (2010): 180.
5. Mirjalili, S. and Lewis, A., 2016. The whale optimization algorithm. *Advances in engineering software*, *95*, pp.51-67.
6. Parpinelli, R.S., Lopes, H.S. and Freitas, A.A., 2002. Data mining with an ant colony optimization algorithm. *IEEE transactions on evolutionary computation*, *6*(4), pp.321-332.

- Gradient descent is an optimization algorithm used to minimize function J(θ) that is Mean Squared Error,by updating the parameters which is theta(intercept and slope) and ∇J(θ) w.r.t. to the parameters(theta).
- The learning rate eta determines the size of the steps we take to reach a (local) minimum and eventually a global minima in epochs iterations . In other words, we follow the direction of the slope of the surface created by the objective function until we reach a desired minimum cost.

- We have studied the working of optimization algorithms in machine learning and implemented Gradient Descent using python.
- A linear dataset has been used .
- Various parameters such as theta(slope and intercept) were used, adjusted, updated and observed changes have been plotted .

Loss Function used here-> Mean Squared Error (MSE)was used to calculate cost/error.

- Reference book-Hands On Machine Learning

Tools and Technologies Used

- **Language Used- Python –**

  1.Python is a high level langyage which is easy to implement and is used in various platforms such as data mining ,machine learning etc.

  2.It has important libraries which are powerfula nad helpful in visualizing the data and processing it.

  3.Some of these libraries are Numpy, Pandas, Matplot Lib.

- <u>Libraries used-</u>

1.NumPy-

(a)Numpy is an important library which is used in Python language for working with array data structures.

(b)Numpy Library is which is known as NumPy, helps in mathematical and the logical operations on arrays can be performed.

(c) It also helps in discussing various arrays functioning , types of indexes. An introduction to the Matplotlib helps in providing graphs.

Fig 2

2.Pandas

(a) Pandas is also an important library which is used in the Python language for the working with data frames data structures.

(b) Pandas is an open-source, BSD licens Python libraries providing high performances, easy to using data structures and the data analyzing tools for Python language.

(c) Python with Pandas is used in various ranges of the fields including academic and the commercials domain defining finances, economic, , analytics, and Statistics etc.

Fig 3.

3.MatplotLib-

(a)Matplot Library is an important library which is used in plotting the graphs and visualizing the data which helps in data pre processing and feature engineering .

(b) It provides OOPS API for embedding the plots into the apps using general purposing toolkits such as Tkinter etc which is used in our project.

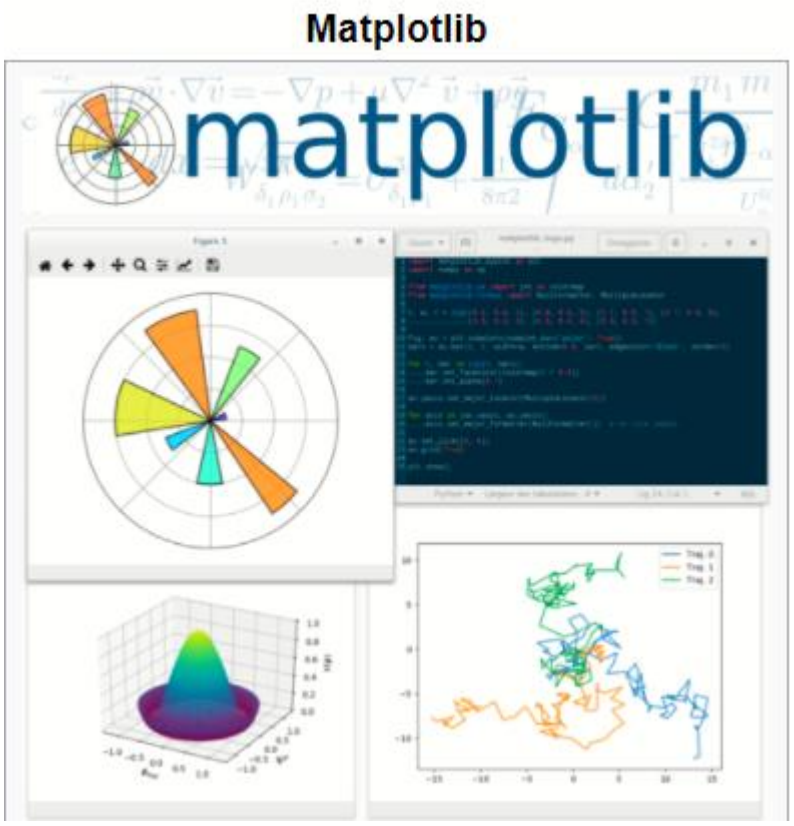


Fig 4

•

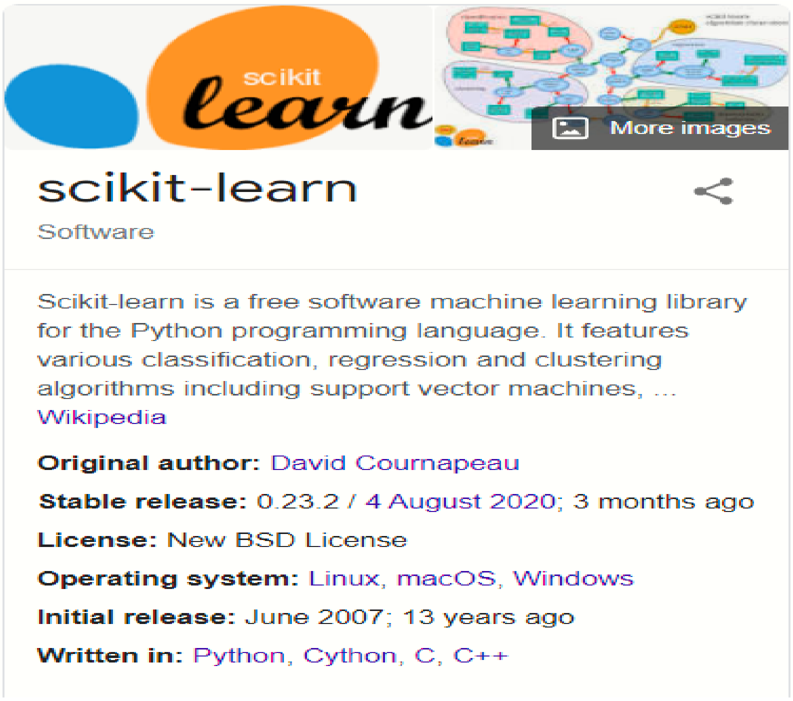
Platform –GoogleColab

(a)Google Colab is an important and powerful tool used in various data warehousing,machine learning and deep learning implementation purposes.

(b)It can be used for remote implementation of the projects whre 2 or more people can work on the project simultaneously.

(c)It supports multiple file formats such as csv, xml, images ,html ,latex etc.

5.Scikit Learn-



scikit-learn

Software

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, ...
Wikipedia

Original author: David Cournapeau

Stable release: 0.23.2 / 4 August 2020; 3 months ago

License: New BSD License

Operating system: Linux, macOS, Windows

Initial release: June 2007; 13 years ago

Written in: Python, Cython, C, C++

Fig5.

Chapter 3: System Development

1- Constructing dummy dataset:

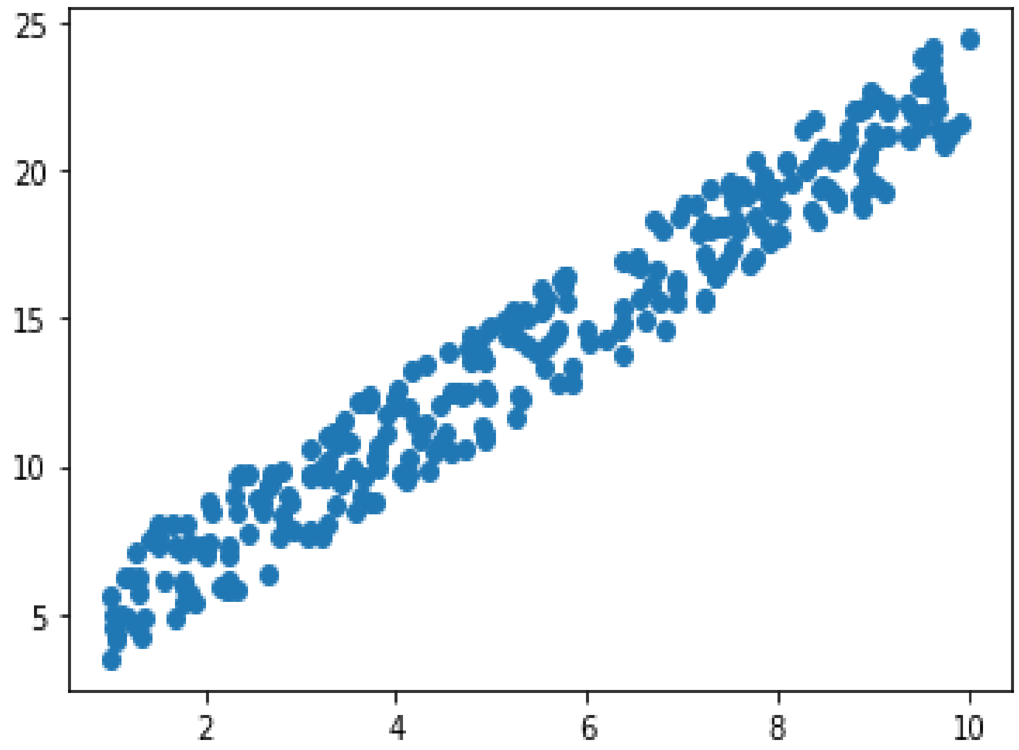
Dummy dataset is constructed by taking some random points on x-axis feeding to a function. In this project we have used 300 points and the function used is of a linear nature. Simultaneously we have added noise to our dataset so that optimization algorithm doesn't stops at the first step.

$$\textit{Dataset: } y = m*x + c + noise$$

Algorithm for generating dataset:

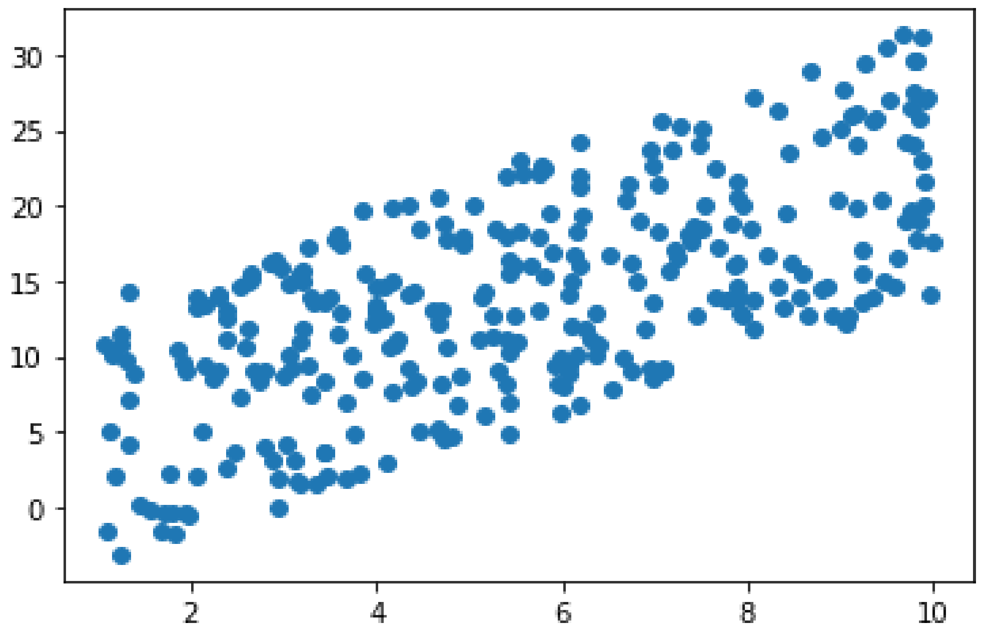
```
def genrate_dataset(slope, intercept, size, noise):
    X = random_points(num_points)
    Y = slope*X + intercept + noise
    Dataset = concatenate(X, Y)
    return dataset
```

Noise Factor=1



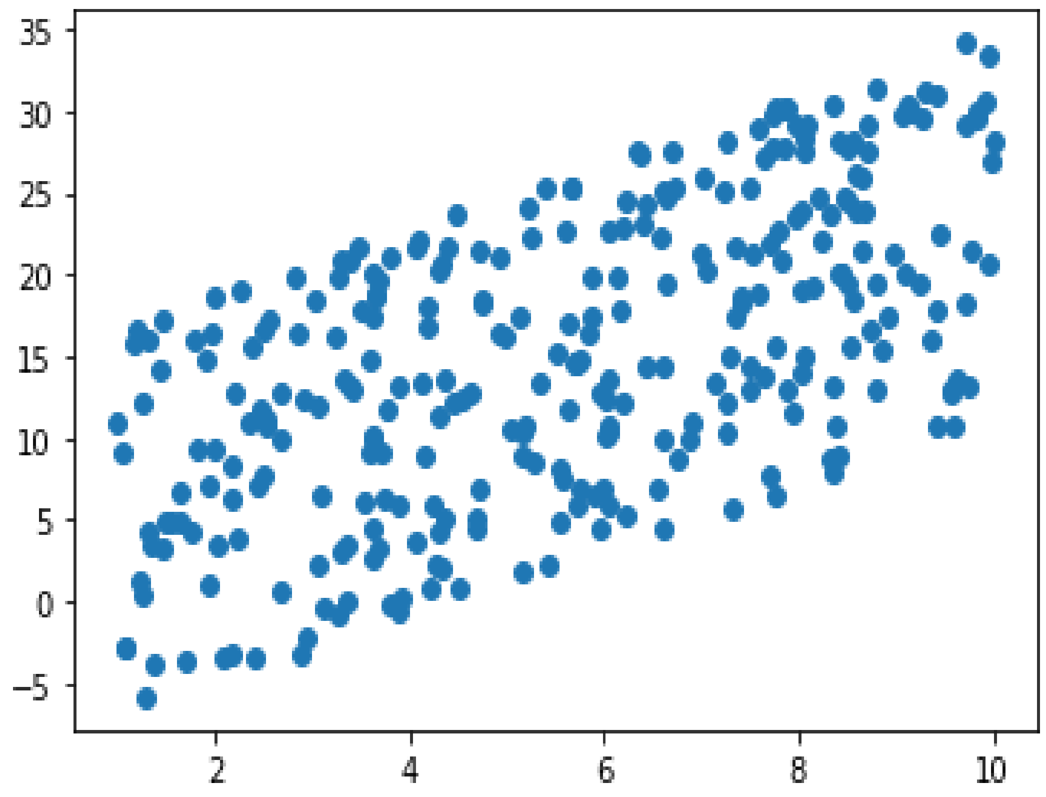
G 1.

Noise Factor 4.5



G 2.

Noise Factor 6

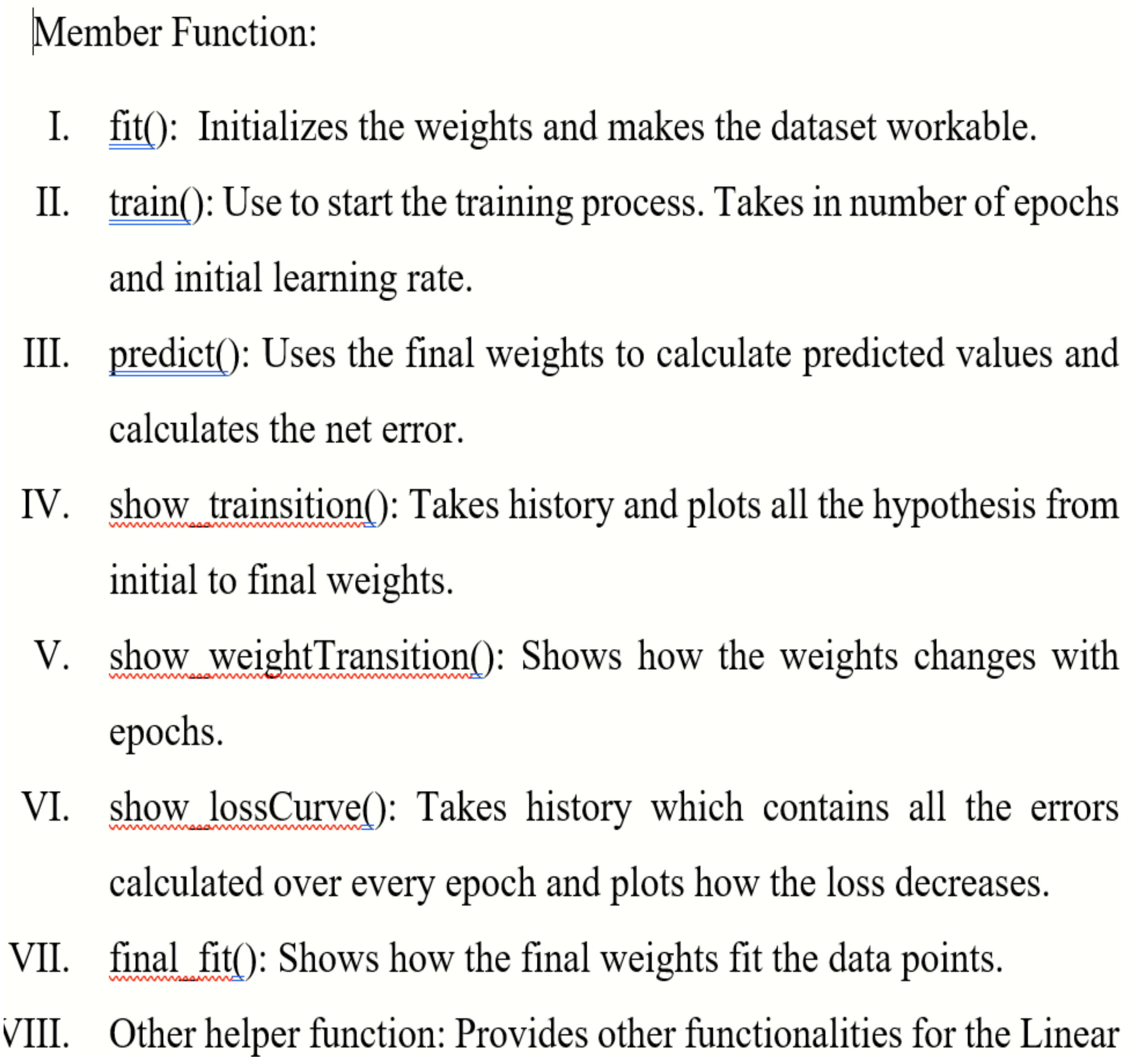


G 3.

a. Make Linear Regression class: Declared basic variables and member functions.

   Variables Declared:

   I. History: stores the all the errors and weights of all the epochs(iterations).
   II. Optimizer: sets the optimizer used to optimize the loss function.
   III. Dynamic Learning Rate: sets the trend which the learning rate follows to reduce.



Member Function:

I. fit(): Initializes the weights and makes the dataset workable.

II. train(): Use to start the training process. Takes in number of epochs and initial learning rate.

III. predict(): Uses the final weights to calculate predicted values and calculates the net error.

IV. show_trainsition(): Takes history and plots all the hypothesis from initial to final weights.

V. show_weightTransition(): Shows how the weights changes with epochs.

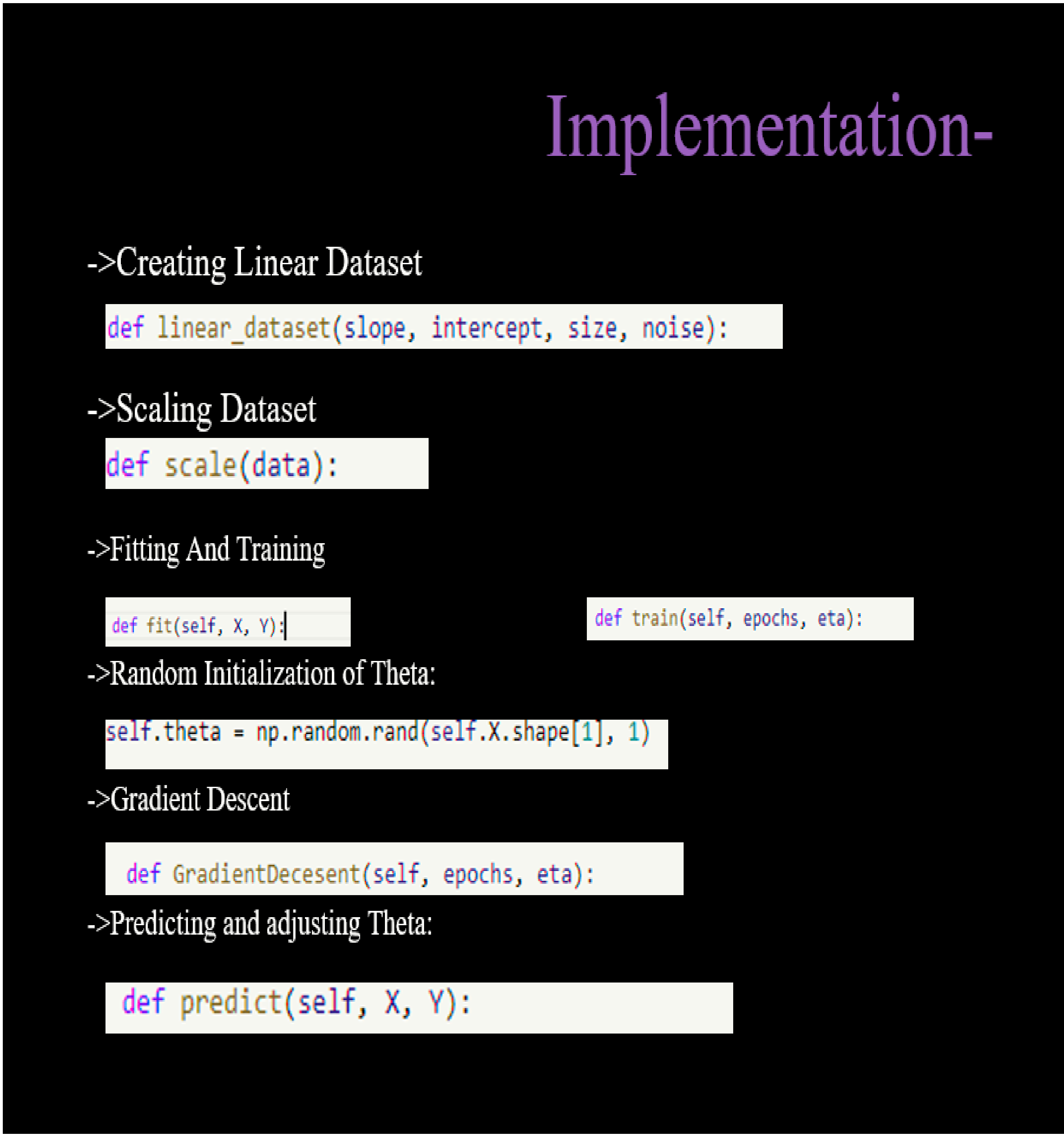
VI. show_lossCurve(): Takes history which contains all the errors calculated over every epoch and plots how the loss decreases.

VII. final_fit(): Shows how the final weights fit the data points.

I. VIII. Other helper function: Provides other functionalities for the Linear Regression class to work.

2-*Implementing Optimizing Algorithms:*

- Gradient Descent:



Implementation-

->Creating Linear Dataset

```
def linear_dataset(slope, intercept, size, noise):
```

->Scaling Dataset

```
def scale(data):
```

->Fitting And Training

```
def fit(self, X, Y):
```

```
def train(self, epochs, eta):
```

->Random Initialization of Theta:

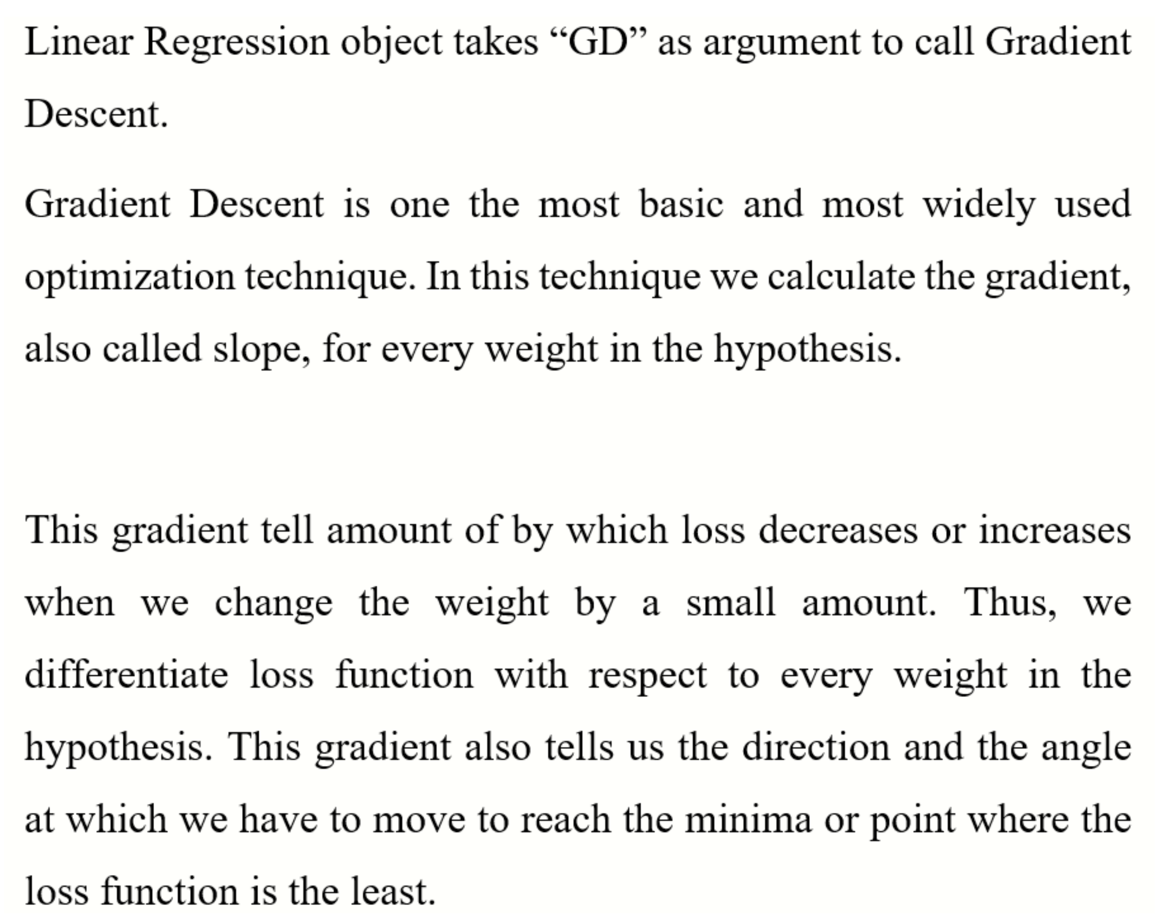
```
self.theta = np.random.rand(self.X.shape[1], 1)
```

->Gradient Descent

```
def GradientDecesent(self, epochs, eta):
```

->Predicting and adjusting Theta:

```
def predict(self, X, Y):
```



Linear Regression object takes "GD" as argument to call Gradient Descent.

Gradient Descent is one the most basic and most widely used optimization technique. In this technique we calculate the gradient, also called slope, for every weight in the hypothesis.

This gradient tell amount of by which loss decreases or increases when we change the weight by a small amount. Thus, we differentiate loss function with respect to every weight in the hypothesis. This gradient also tells us the direction and the angle at which we have to move to reach the minima or point where the loss function is the least.

We add this gradient, with direction, to its respective weight after multiplying it with learning rate which represents the length of step we have to take.

Now, we apply this to one of the simplest loss functions, Mean Squared Error.

MSE(y_predicted) = ((y_predicted – y_actual)^2)/n, where n is the number of instances of data available.

$y\_predicted = w_1x_1 + w_2x_2 + \ldots. + w_nx_n = \Sigma w_ix_i$

On differentiation we get,

$dJ/dw_i = 2*(y\_predicted-y\_actual)*x_i$,

error = y_predicted-y_actual ,so now $dJ/dw_i = 2*error*x_i$.

Gradient = Array of dJ/dw_i.

Update Statement : $w_i = w_i + (learning\_rate)*gradient[i]$.

Algorithm -Gradient Descent

- The learning rate alpha is constant only the weights that is slope and intercept are randomly initialized and updated over the epochs
- The optimal weights are obtained over the epochs and graph of loss vs epochs is plotted to observe to reach the global minima .
- If the loss is high then we move towards right direction ie we increase the weights in the positive direction of x-axis.

- if the loss increases over the epochs then we decrease the weights we move in the negative direction of x-axis
- ***The update equations are:***

$$w = w - \alpha \nabla_w J$$

$$b = b - \alpha \nabla_b J$$

Alpha is learning rate which is constant.

$$\frac{\partial}{\partial w} J(w) = \nabla_w J$$

$$\frac{\partial}{\partial b} J(w) = \nabla_b J$$

- w,b are the weights that we are trying to optimize over epochs by partially derivating to achieve the gradient which is the direction we move to reduce the cost function that is Mean Squared Error **.**

Loss Function

- Mean Squared Error is derivated wrt to weights to find gradient
- Taking all data points in every epochs

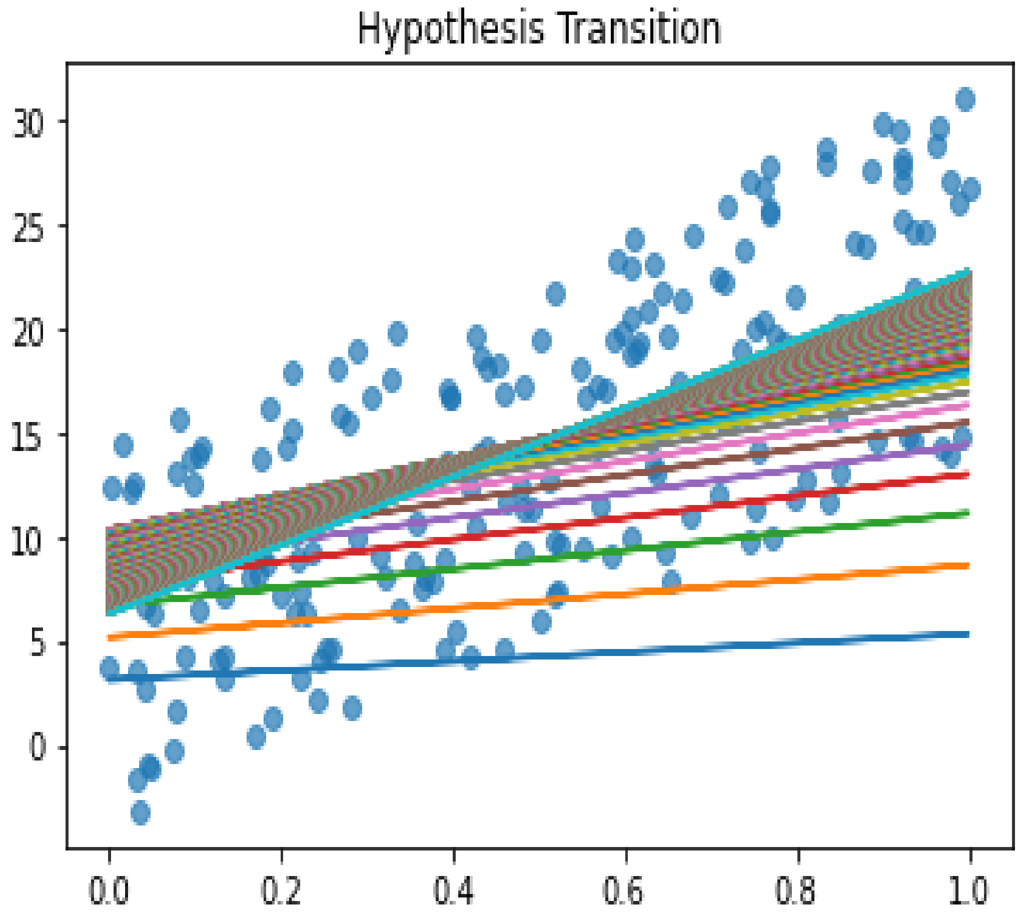
$$E = \frac{1}{n} \sum_{i=0}^{n} (y_i - \bar{y}_i)^2$$

$$D_m = \frac{1}{n} \sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

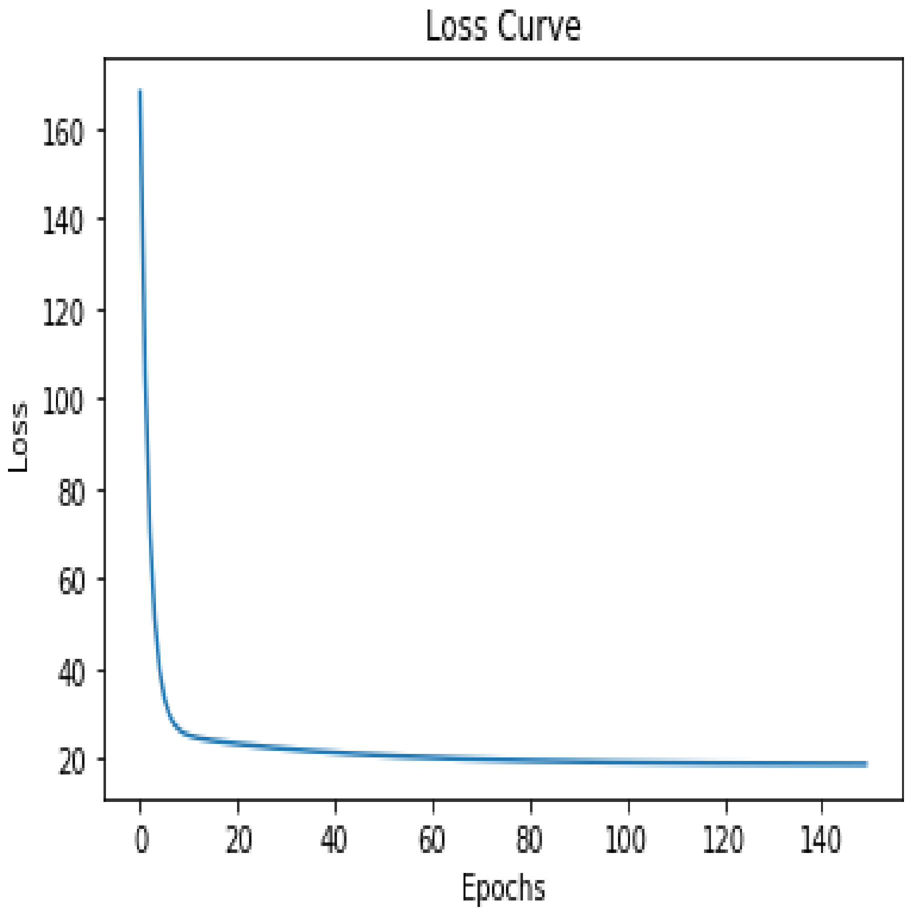
$$D_m = \frac{-2}{n} \sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

Derivative with respect to m Similarly for c

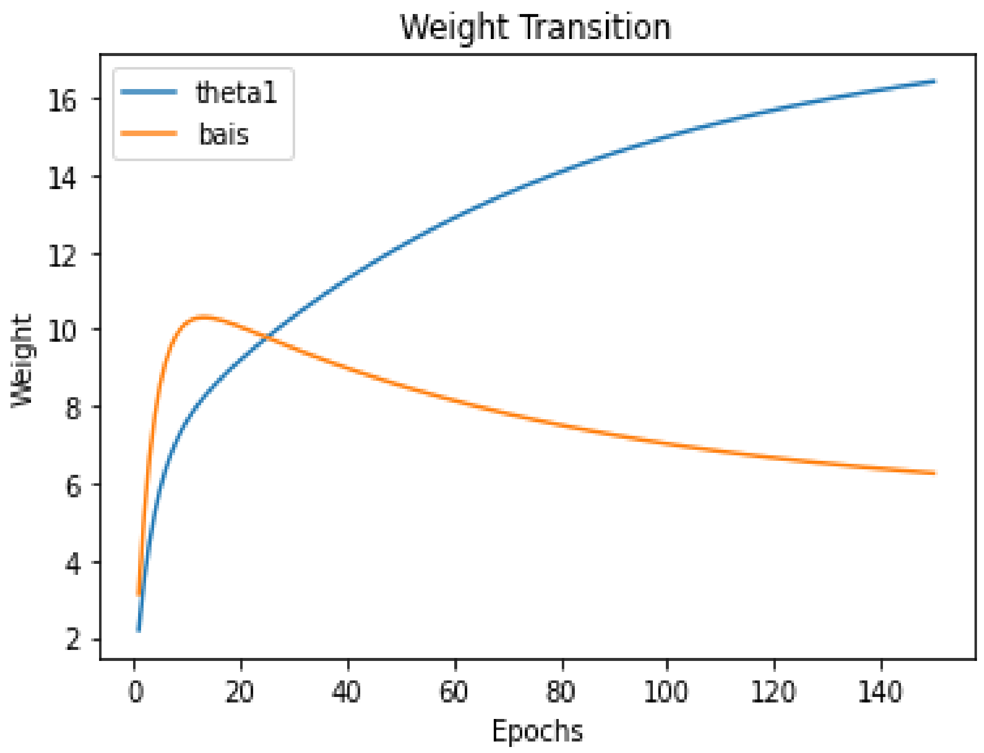
```
def GradientDecesent(self, epochs, eta):
  previous_theta = []
  previous_error = []
  for _ in range(epochs):
    predicted = np.dot(self.X, self.theta)
    error = predicted-self.Y
    gradient = (2/len(self.X))*np.dot(np.transpose(self.X), error)
    self.theta = self.theta - eta*gradient
    error = (np.dot(np.transpose(error), error))/len(X)
    previous_theta.append(self.theta)
    previous_error.append(error[0])
  return [previous_theta, previous_error]
```



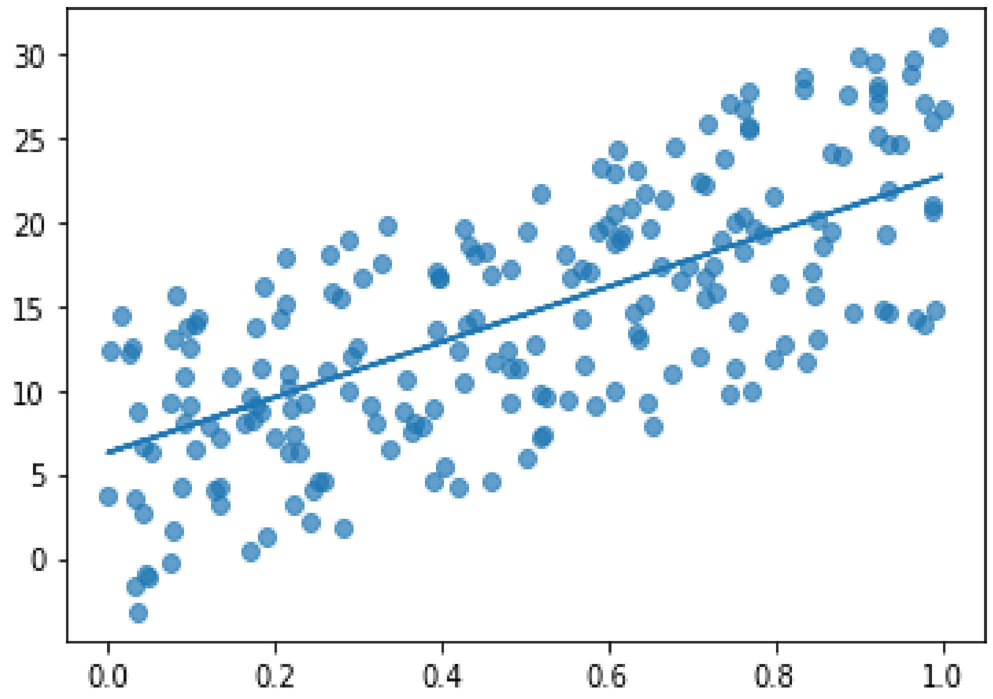
G4.


Loss Curve
Loss
Epochs
160
140
120
100
80
60
40
20
0
20
40
60
80
100
120
140

G 5.


Weight Transition
theta1
bais
Weight
Epochs
2
4
6
8
10
12
14
16
0
20
40
60
80
100
120
140

G 6.


0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0

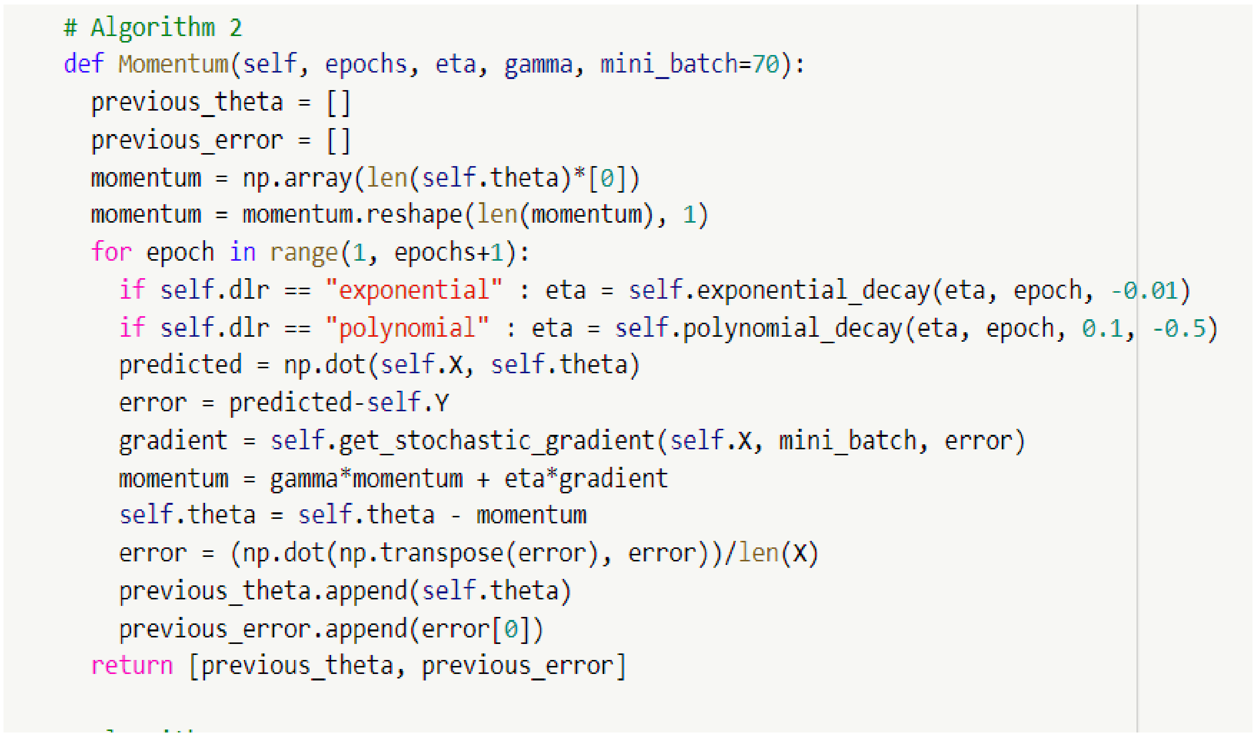
G7.

Momentum: SGD Stochastic Gradient Descent

- Linear Regression object takes “Momentum” as argument to call Gradient Descent.
- Momentum works on Stochastic Gradient Descent. Stochastic Gradient Descent is almost same as Gradient Descent.
- Gradient Descent takes all the point in dataset into consideration to calculate, the process is slower increases time complexity so we take mini batch to make the weights updation faster.
- The learning rate alpha is constant only the weights that is slope and intercept are randomly initialized and updated over the epochs
- The optimal weights are obtained over the epochs and graph of loss vs epochs is plotted to observe to reach the global minima .
- The convergence is not smooth because of the mini batch it is very oscillating which has to be damped where as gradient descent has smooth convergence of loss curve.
- The damping of the oscillations is done by giving the momentum to the weights ie exp weighted average.
- Giving the more weights to present iteration gradient and less to previous gradients. Beta=.95 initialized ,alpha is constant.

$$V_t=\beta\beta(1-\beta)S_{t-2}+\ldots+\beta(1-\beta)S_{t-1}+\ldots+(1-\beta)S_t$$

- $$V_t=\beta V_{t-1}+(1-\beta)\nabla_w L(W,X,y)$$
$$W=W-\alpha V_t$$



```
# Algorithm 2
def Momentum(self, epochs, eta, gamma, mini_batch=70):
  previous_theta = []
  previous_error = []
  momentum = np.array(len(self.theta)*[0])
  momentum = momentum.reshape(len(momentum), 1)
  for epoch in range(1, epochs+1):
    if self.dlr == "exponential" : eta = self.exponential_decay(eta, epoch, -0.01)
    if self.dlr == "polynomial" : eta = self.polynomial_decay(eta, epoch, 0.1, -0.5)
    predicted = np.dot(self.X, self.theta)
    error = predicted-self.Y
    gradient = self.get_stochastic_gradient(self.X, mini_batch, error)
    momentum = gamma*momentum + eta*gradient
    self.theta = self.theta - momentum
    error = (np.dot(np.transpose(error), error))/len(X)
    previous_theta.append(self.theta)
    previous_error.append(error[0])
  return [previous_theta, previous_error]
```

Loss Function;

$$E=\frac{1}{n}\sum_{i=0}^{n}(y_i-\bar{y}_i)^2$$

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

Derivative with respect to m Similarly for c

Mean Squared Error is derivated wrt to weights to find gradient

- Taking Mini Batch of N data points Randomly by using seed

To reduce this, we only take some points from dataset to calculate the gradient.

These points are chosen randomly from the set.

Due to this random selection process the loss function converges but, in a zig-zag manner. The Momentum comes into picture here as it smooths out this zig-zag pattern, and also it helps accelerate the process and help the function converge faster.

As the name suggest, algorithm gains “momentum”. For example, if we throw a ball from top of the curve, it will still have momentum or speed when it reaches local minima, and will pop out of it and continue searching for global minima.

To mathematically understand this, we associate a weight with every gradient. This weight is maximum for current gradient and minimum for the first gradient.

This weight is constant to some power. This constant is beta, with a value between 0 to 1.

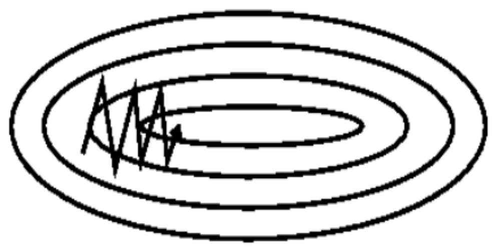


Image 2: SGD without momentum

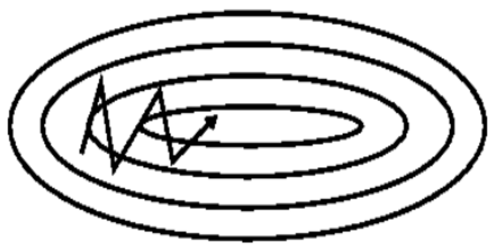


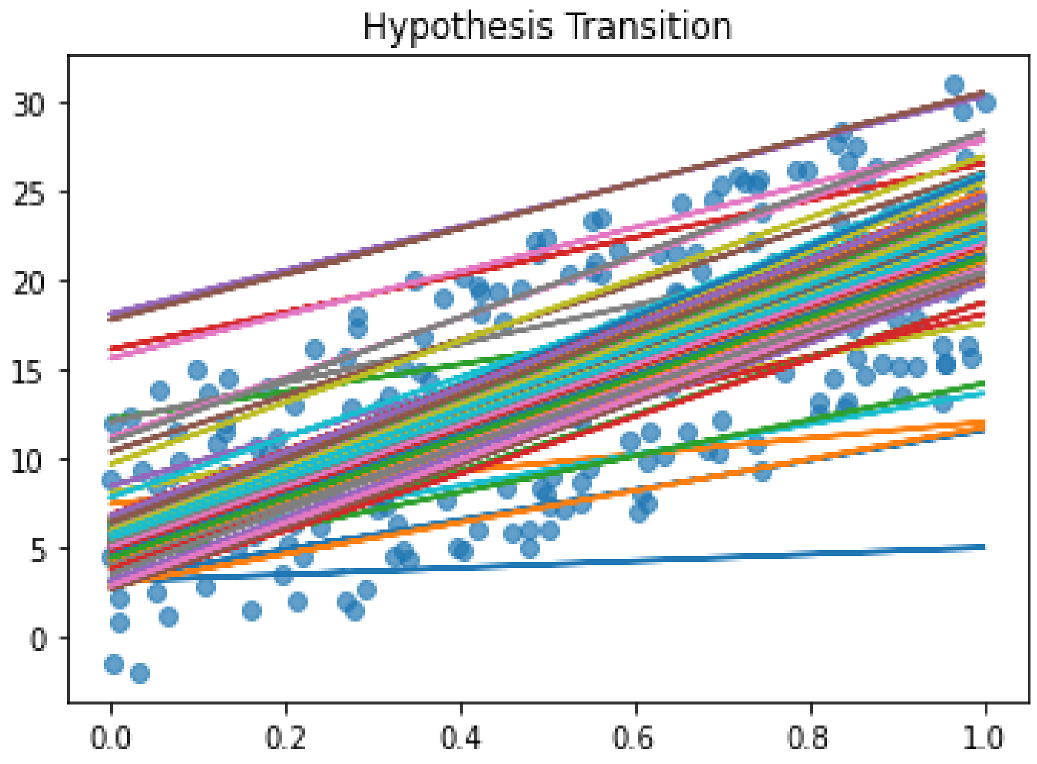
Image 3: SGD with momentum

Gradient = get_gradient(data, points)

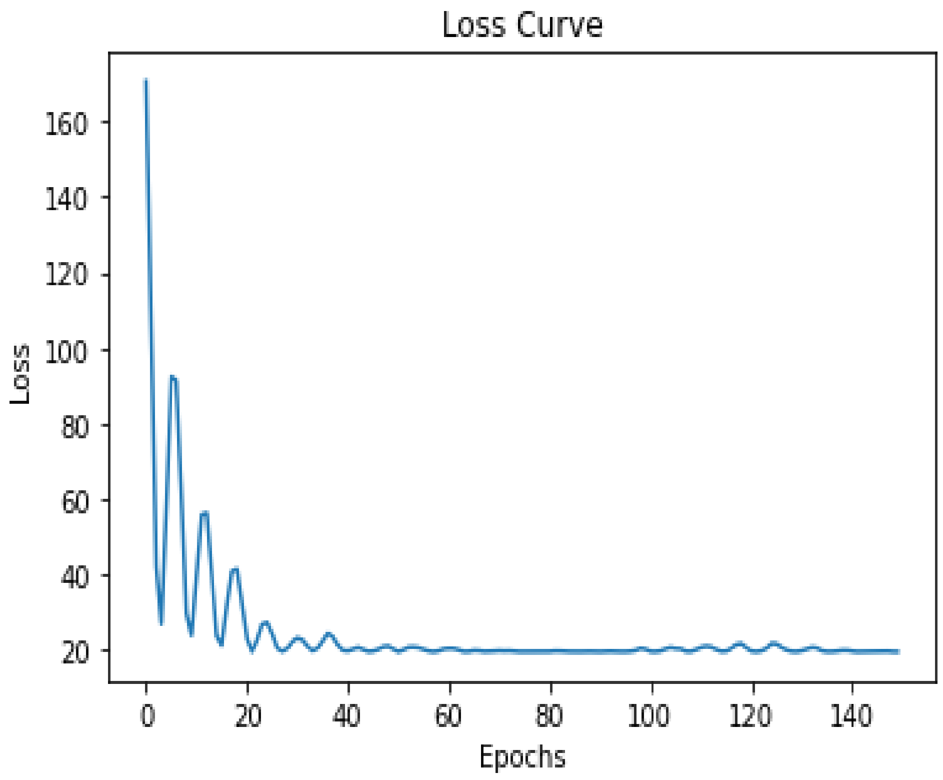
*Momentum = β*Momentum + α*Gradient*

The current gradient has a weight of 1 and rest other gradients are multiplied by beta.

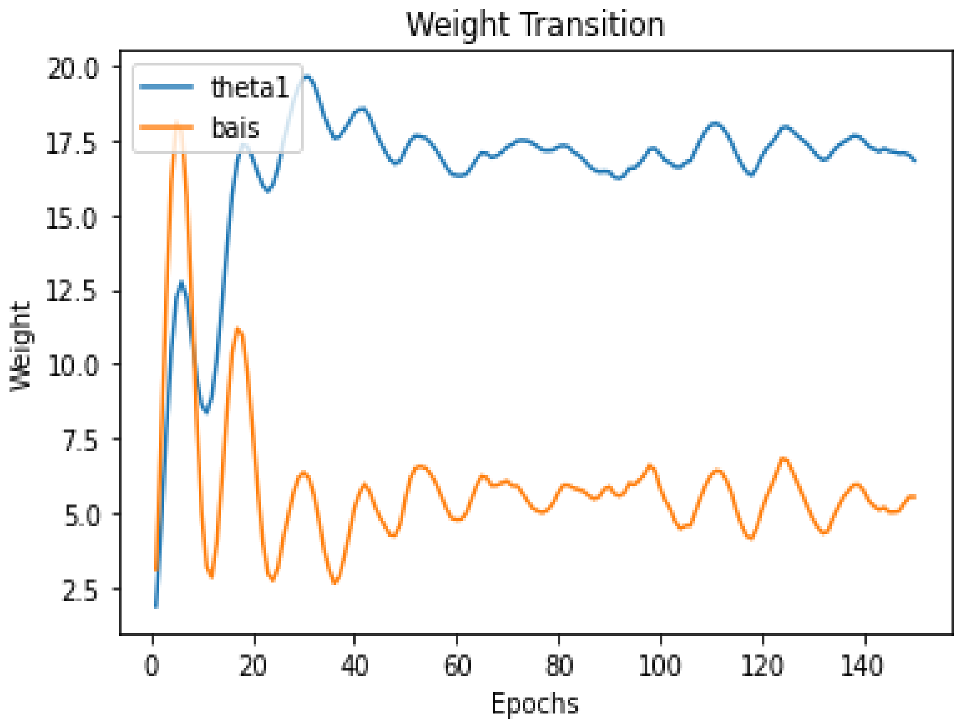
Update Statement : $w_i = w_i + Momentum$



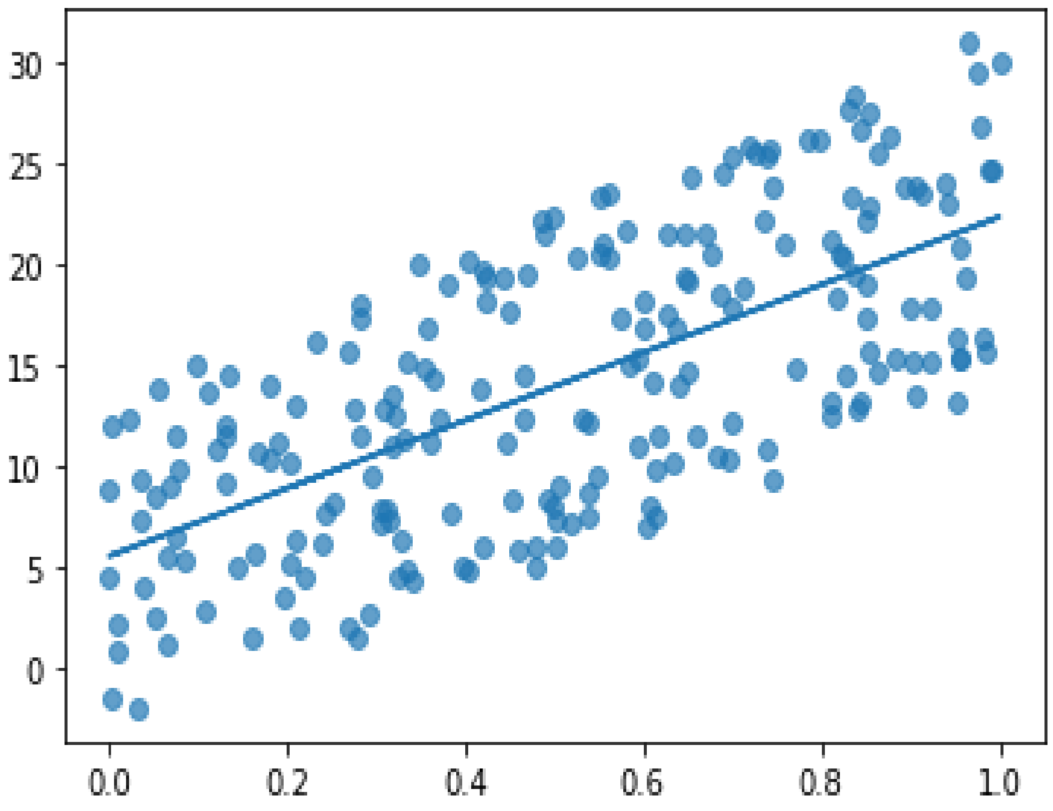
G8.


Loss Curve
Loss
Epochs
160
140
120
100
80
60
40
20
0
20
40
60
80
100
120
140

G9.


Weight Transition
theta1
bais
Weight
Epochs
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
20
40
60
80
100
120
140

G10.



G11.

3- *Root Mean Squared Propagation(RMSProp) :*

Linear Regression object takes "RMSP" as argument to call Gradient Descent.The learning rate is not constant where as it is varied over epochs to converge faster .

The Smoothening Vdw and Vdb is used to contain the fall of the learning rate and for convergence to still occur and reach to global minima otherwise weights wont update and get stuck.

This happens as the previous and the new weights are almost the same so the convergence comes to still. The value of learning rate in Vdw is denoted by beta and is usually set to 0.9 .

- The Vdw and Vdb are initilaised to 0. For every epochs we calculate and

update

$$v_{dw} = \beta \cdot v_{dw} + (1-\beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1-\beta) \cdot db^2$$

1.

weights with the help of velocity component.

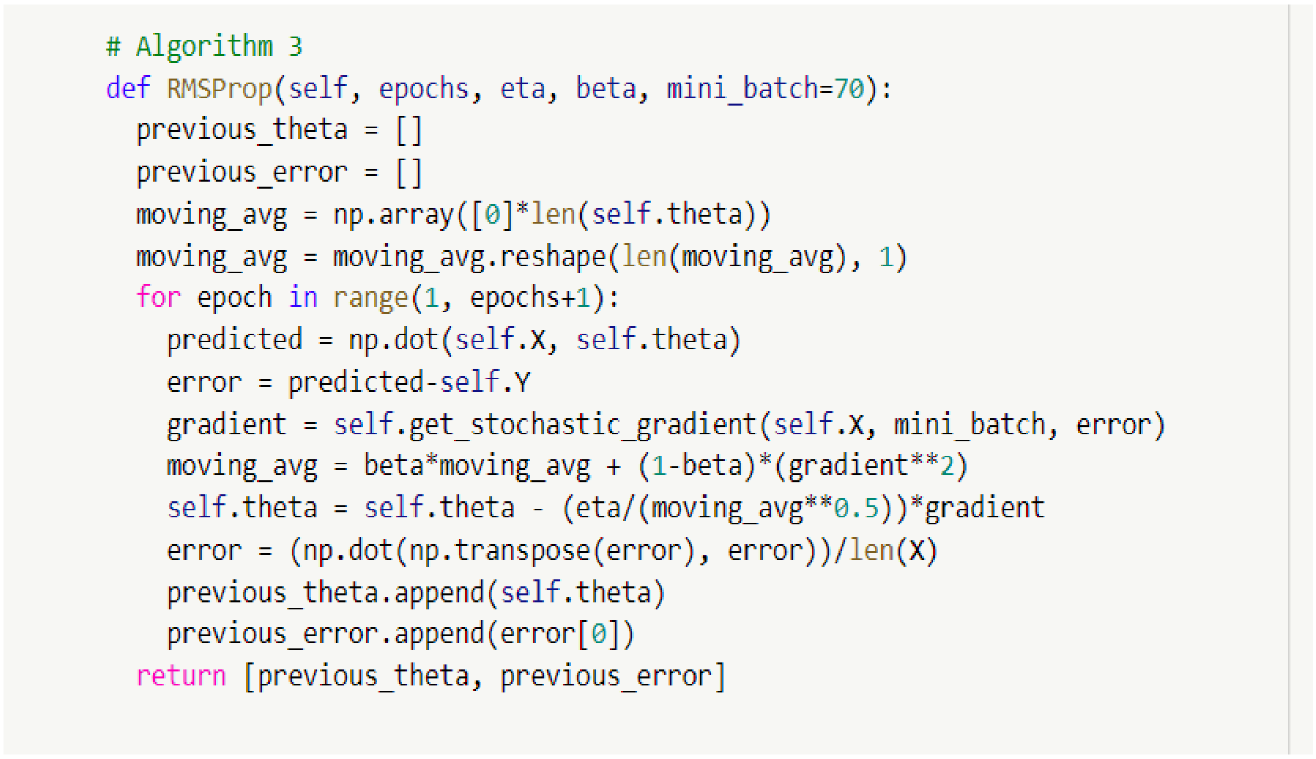
- Weight updation Equation-is as follows--

The learning rate is
update and contained by vdw,vdb

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

Algorithm-



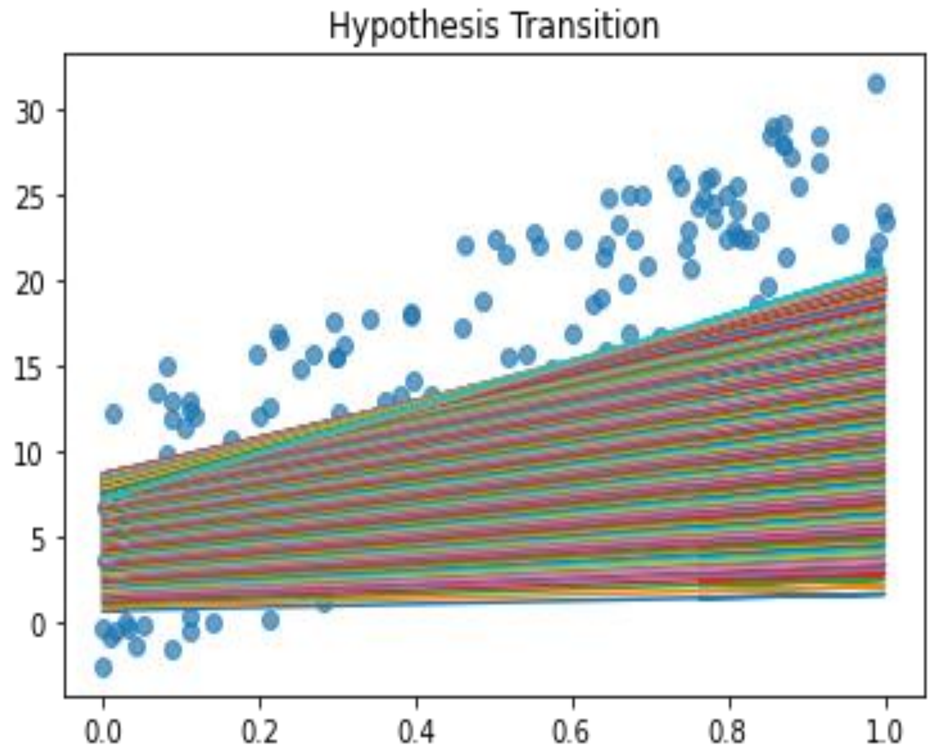
```
# Algorithm 3
def RMSProp(self, epochs, eta, beta, mini_batch=70):
  previous_theta = []
  previous_error = []
  moving_avg = np.array([0]*len(self.theta))
  moving_avg = moving_avg.reshape(len(moving_avg), 1)
  for epoch in range(1, epochs+1):
    predicted = np.dot(self.X, self.theta)
    error = predicted-self.Y
    gradient = self.get_stochastic_gradient(self.X, mini_batch, error)
    moving_avg = beta*moving_avg + (1-beta)*(gradient**2)
    self.theta = self.theta - (eta/(moving_avg**0.5))*gradient
    error = (np.dot(np.transpose(error), error))/len(X)
    previous_theta.append(self.theta)
    previous_error.append(error[0])
  return [previous_theta, previous_error]
```

RMSProp is just RProp used on SGD with some changes. In RMSProp we use exponentially weighted average on squared of gradient. This is done because of averaging gardients.

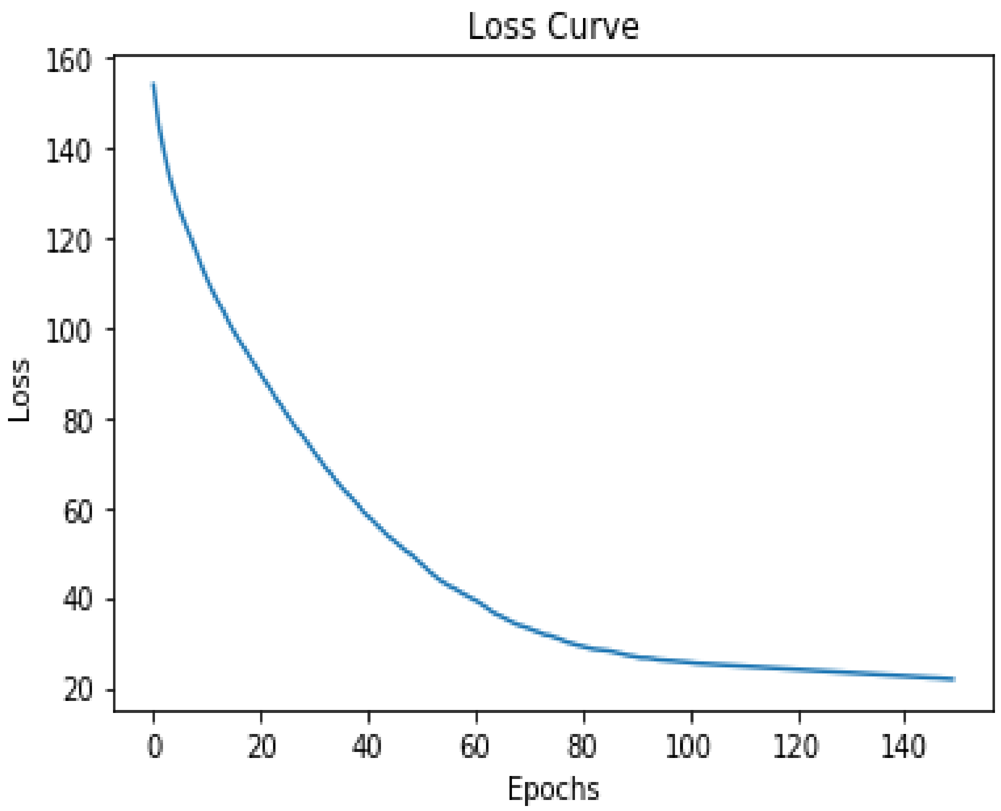
$$Moving\_average = \beta * Moving\_average + (1-\beta) * Gradient^2$$

Update Statement:

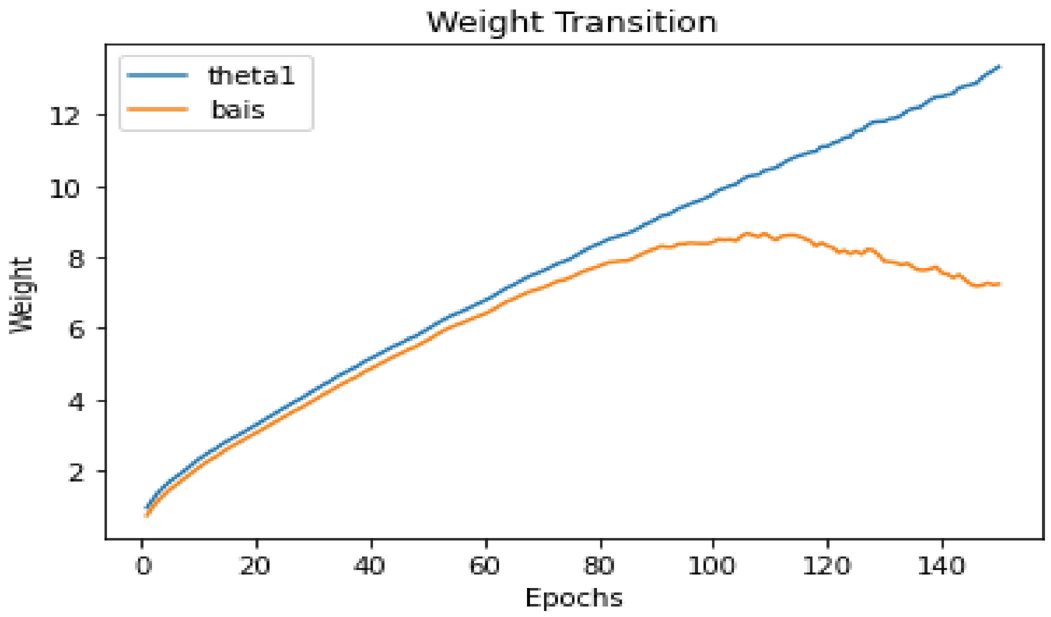
$$w_i = w_i - (leraning\_rate/Moving\_average^{0.5)} * Gradient$$



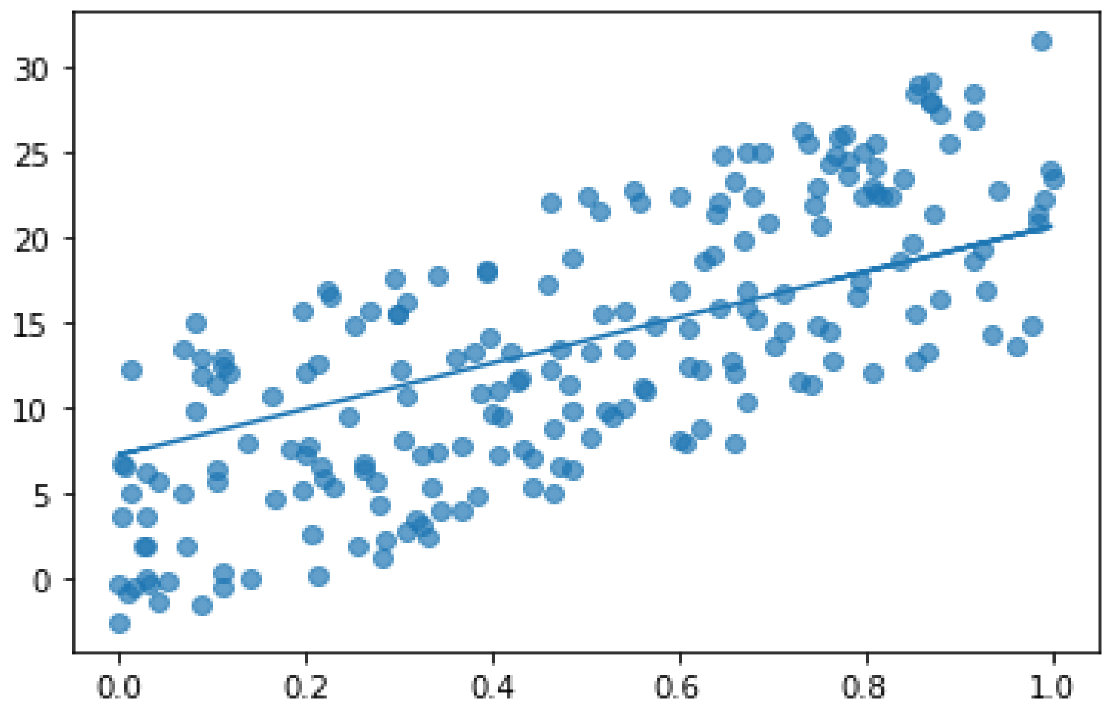
G12.


Loss Curve
Loss
Epochs
160
140
120
100
80
60
40
20
0
20
40
60
80
100
120
140

G13.


Weight Transition
theta1
bais
Weight
Epochs
12
10
8
6
4
2
0
20
40
60
80
100
120
140

G14.



G15.

4) Adagrad- Adaptive Gradient ie adaptive learning learning rate

.For the different epochs and iterations the learning rate gets updated because in

The bag of words the mini batch may have sparse or dense classes to even out the distribution we use varied learning rates.

Till now what we implemented till SGD with momentum was that the learning rate stays steady that is constant and fixed

- There is no momentum directly used ,it is more in line with rmsprop where the learning rate is constant and vdw is used for faster convergence just gradient squared is used.
- The problem with this is that over epochs the alpha squared will become very very large and learning rate very small so the weights will not be updated and get stuck in the local not the global minima.

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

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

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

$$\varepsilon \text{ is a small } +\text{ve number to avoid divisibilty by } 0$$

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

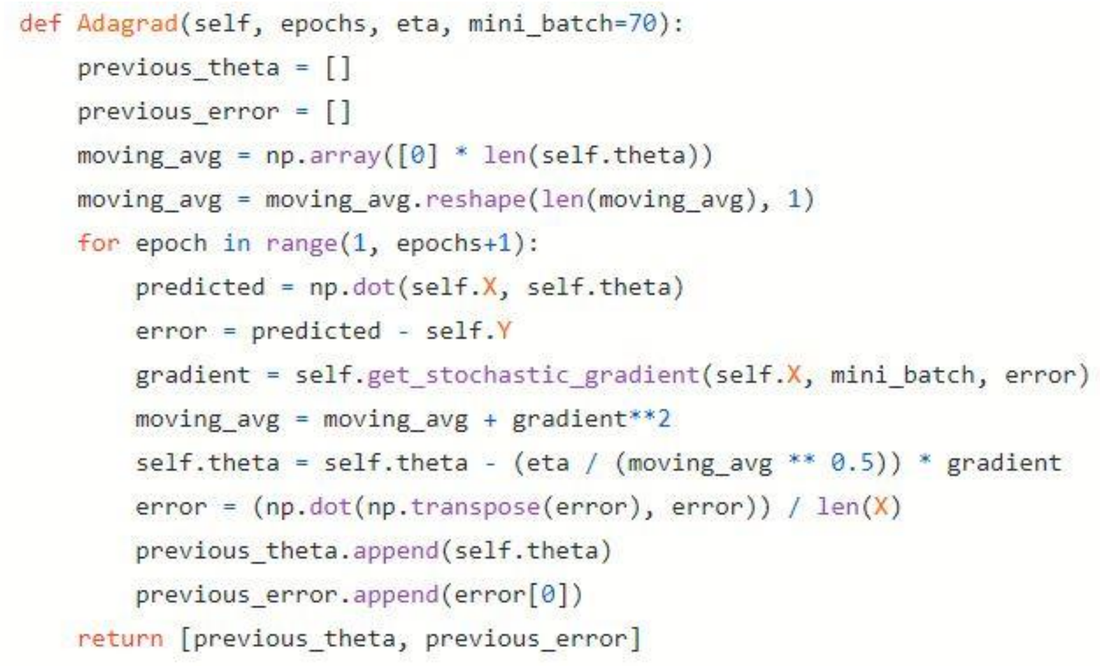
- The N stands for learning rate which is variable.
- W(t) is the weights updation equation .
- N' is the equation how the learning rate is updated for every epoch .
- The very large alpha squared is the drawback which is dealt in Rmsp
- The Loss Function is same as before that is Mean Squared Error.

Hence the rmsprop or adadelta is used where the weighted average is used instead of the alpa to contain the rise of it and thus moving towards the global minima.

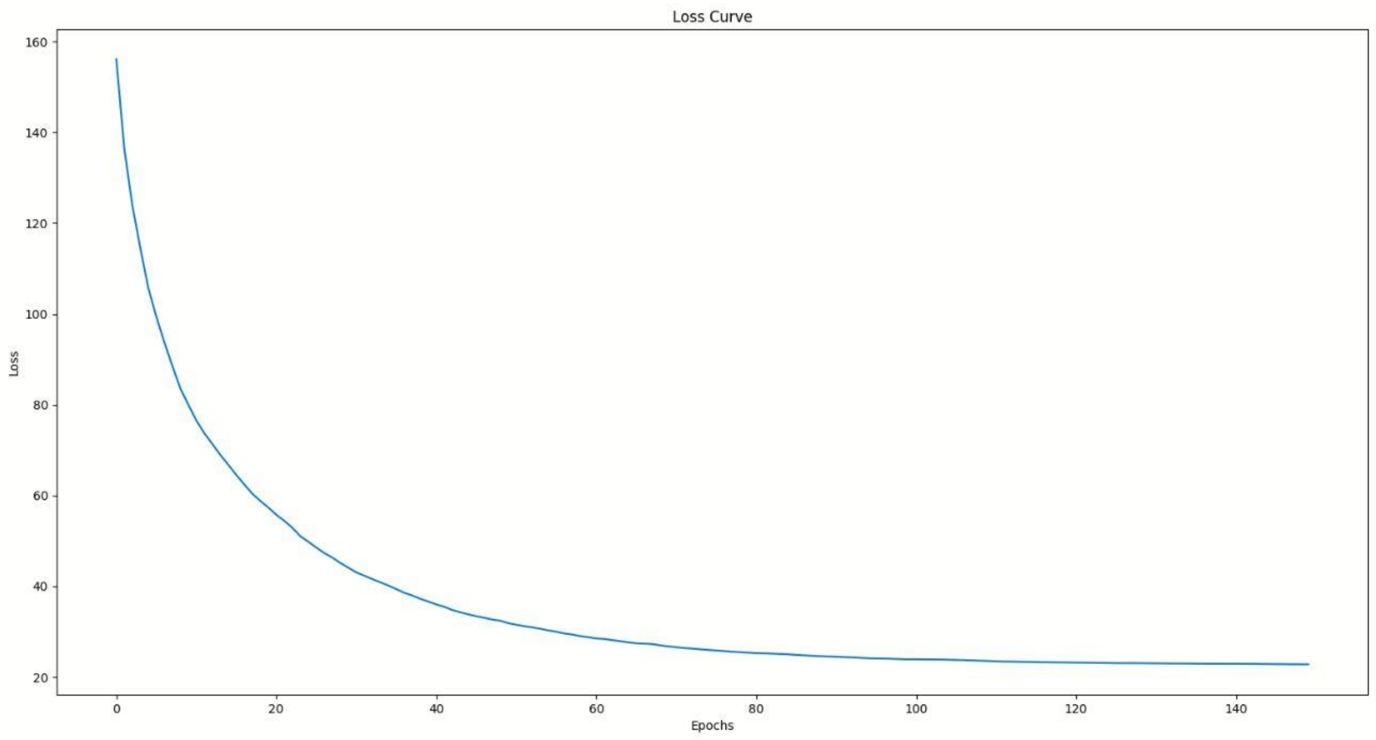
$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

$$b_t = b_{t-1} - \eta \frac{\partial L}{\partial b_{t-1}}$$

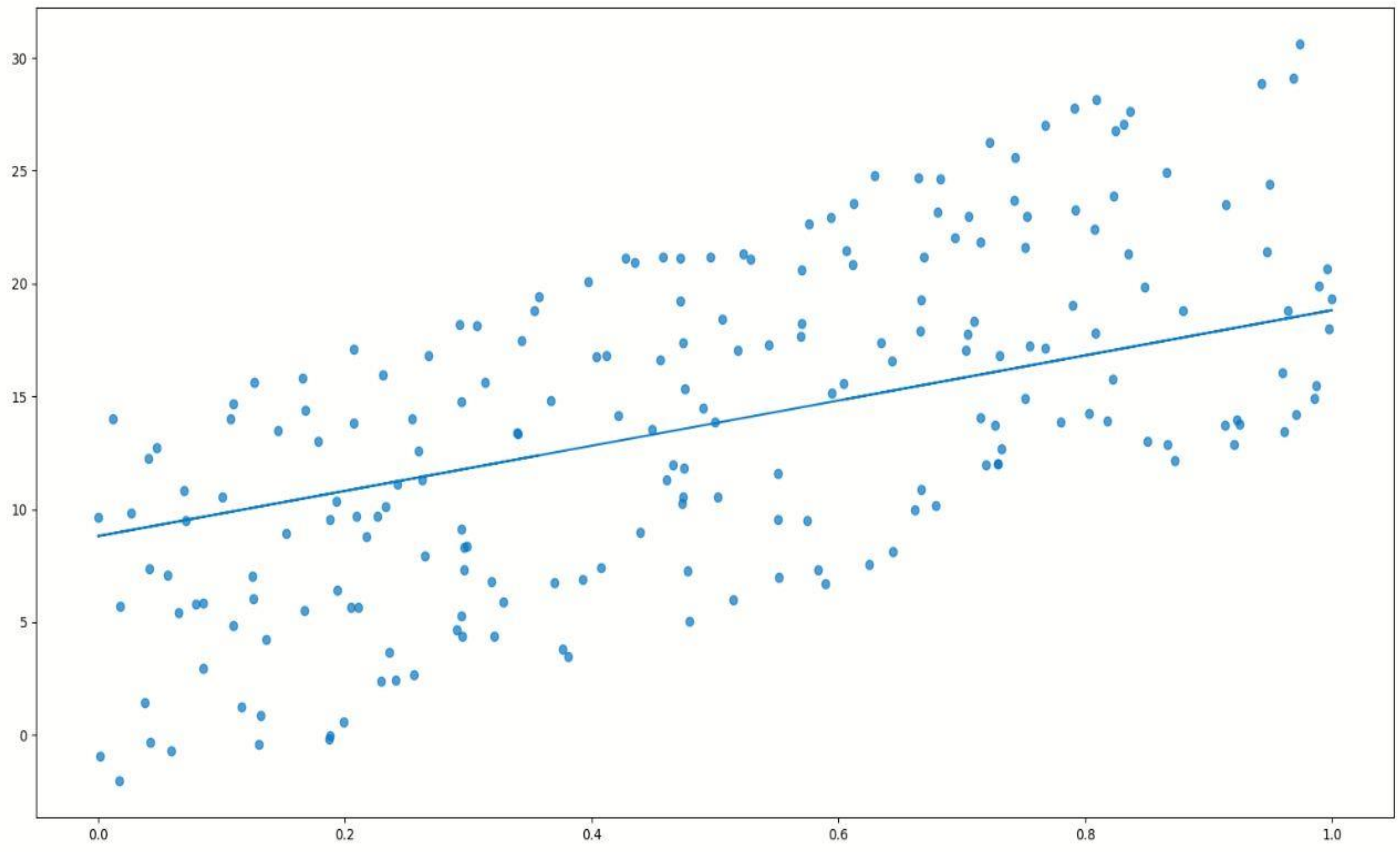
Where the eta is updated by the above mentioned equation.



```
def Adagrad(self, epochs, eta, mini_batch=70):
    previous_theta = []
    previous_error = []
    moving_avg = np.array([0] * len(self.theta))
    moving_avg = moving_avg.reshape(len(moving_avg), 1)
    for epoch in range(1, epochs+1):
        predicted = np.dot(self.X, self.theta)
        error = predicted - self.Y
        gradient = self.get_stochastic_gradient(self.X, mini_batch, error)
        moving_avg = moving_avg + gradient**2
        self.theta = self.theta - (eta / (moving_avg ** 0.5)) * gradient
        error = (np.dot(np.transpose(error), error)) / len(X)
        previous_theta.append(self.theta)
        previous_error.append(error[0])
    return [previous_theta, previous_error]
```


Loss Curve
Loss
Epochs
160
140
120
100
80
60
40
20
0
20
40
60
80
100
120
140

G16.

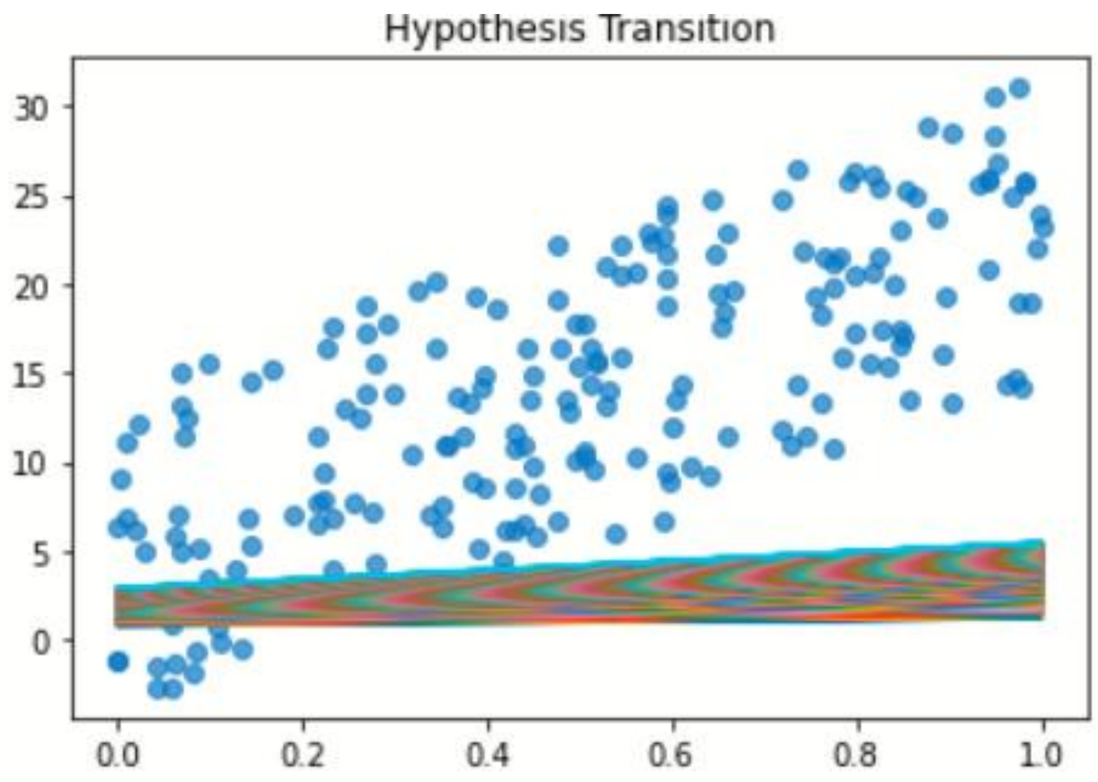


G17.

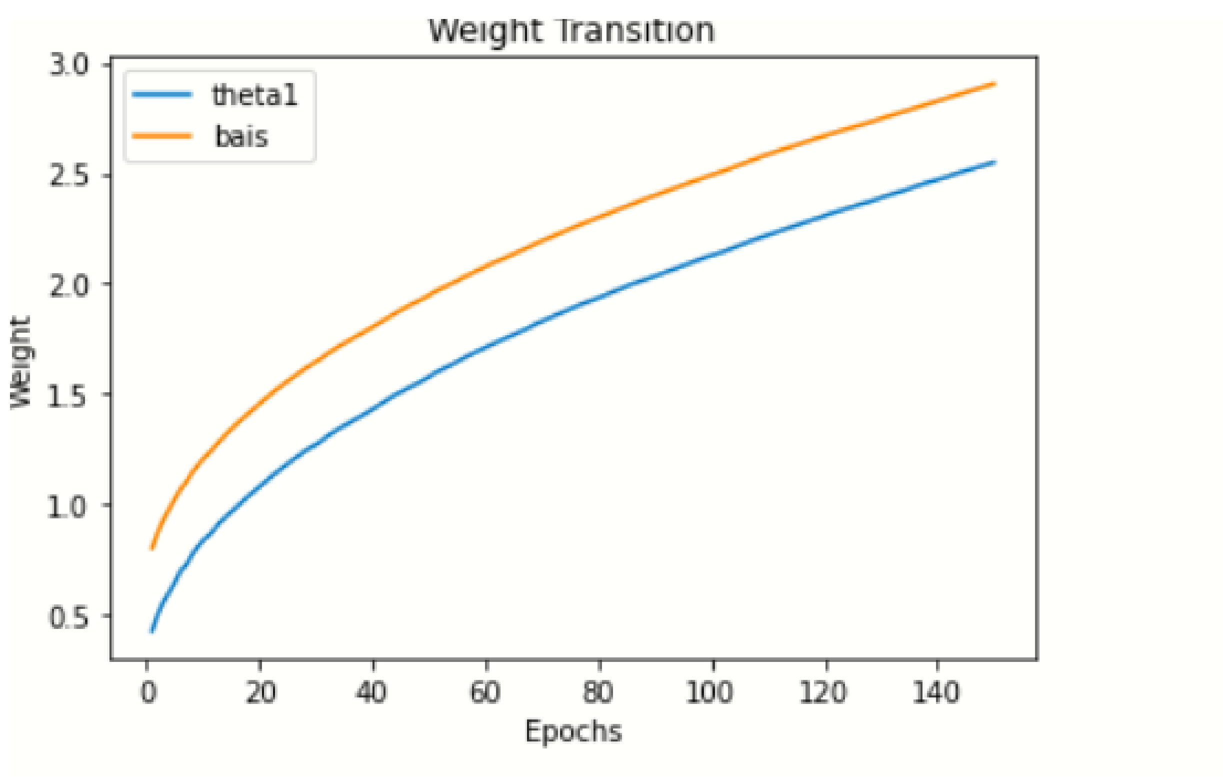
The above graph shows the best fit line model with the most optimum weights reached over epochs and optimized.

The next graph shows the weight transition over epochs and that plotted .

All weights to show the all weights with lines and to demonstrate the optimization happening such way over the epochs .

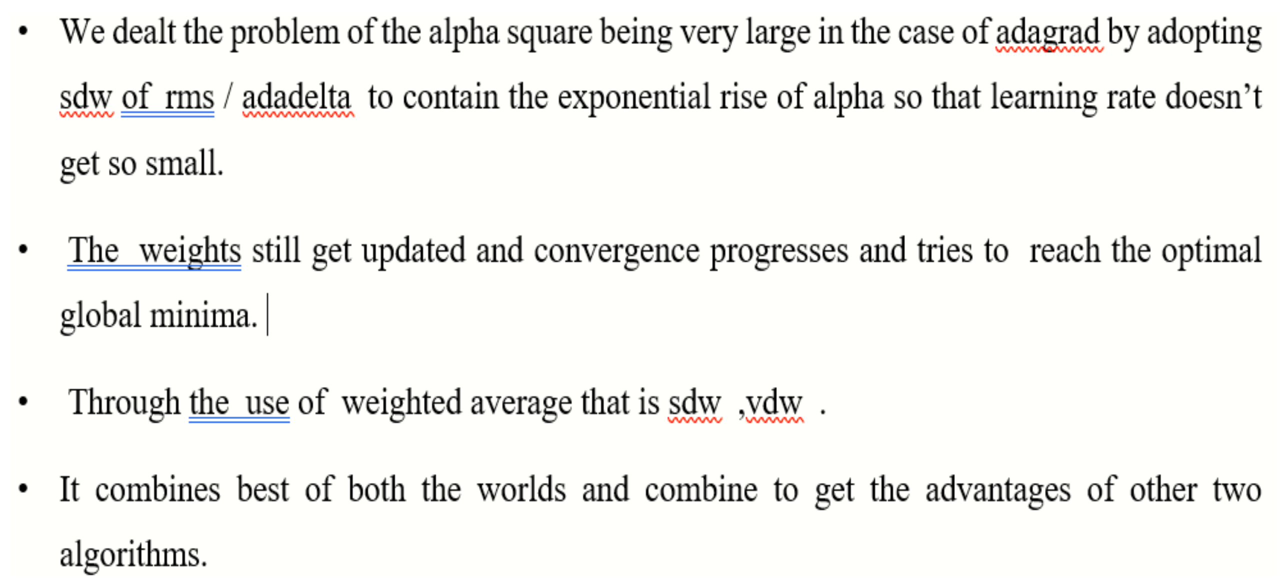


G18.



G19.

5)Adam-



- We dealt the problem of the alpha square being very large in the case of adagrad by adopting sdw of rms / adadelta to contain the exponential rise of alpha so that learning rate doesn't get so small.
- The weights still get updated and convergence progresses and tries to reach the optimal global minima.
- Through the use of weighted average that is sdw ,vdw .
- It combines best of both the worlds and combine to get the advantages of other two algorithms.

-
- Hence a very optimised approach is adopted so that we may vary our learning rate dynamically using Sdw and along with that we also dampen the oscillations with the Vdw component and converge faster and smoothly .

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

1)The Vdw ,Vdb for the momentum is used and denoted as m(t)

As the exponential weighted gradient to dampen and smoothen the oscillations and convergence.

2)The Sdw,Sdb are used for the adaptive learning rates and are denoted as V(t) and are computed for every epoch for the current mini batch as the exponential weighted gradient squared.

3)g(t) is denoted as the gradient derived by the loss function MSE.|

On current Mini batch and epoch ,

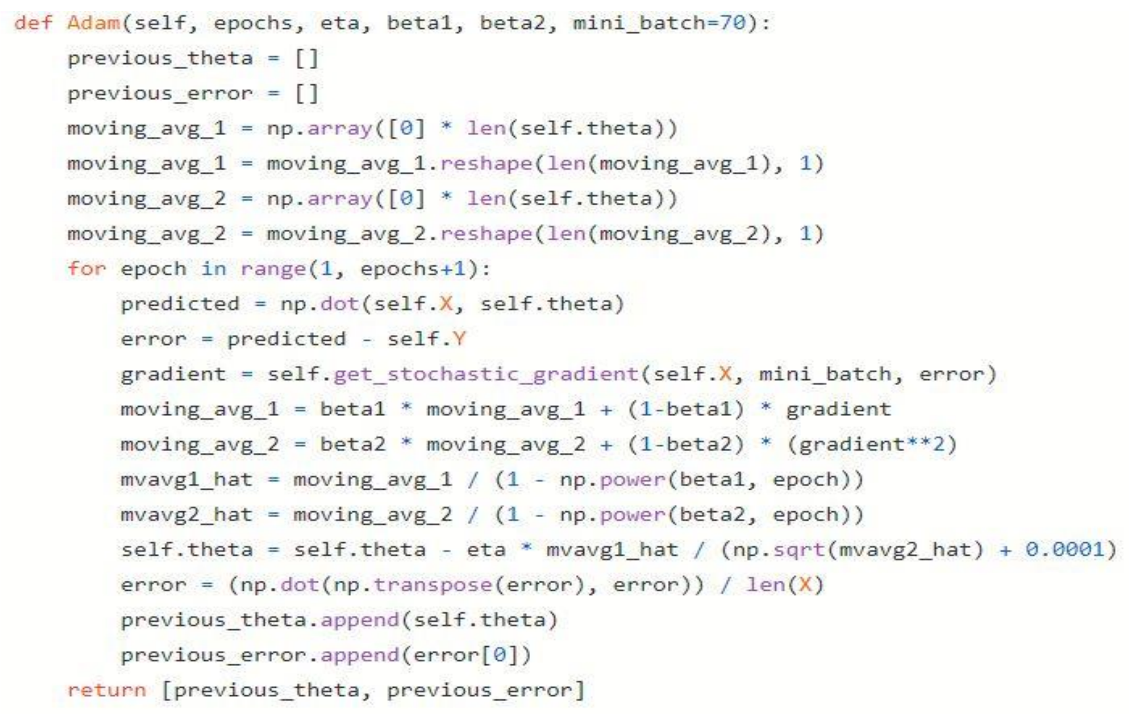
- *m(t),v(t) are computed and fed to weight update equation;*

Over the epochs the error is reduced as the slope converges in the search of the optimal minima

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

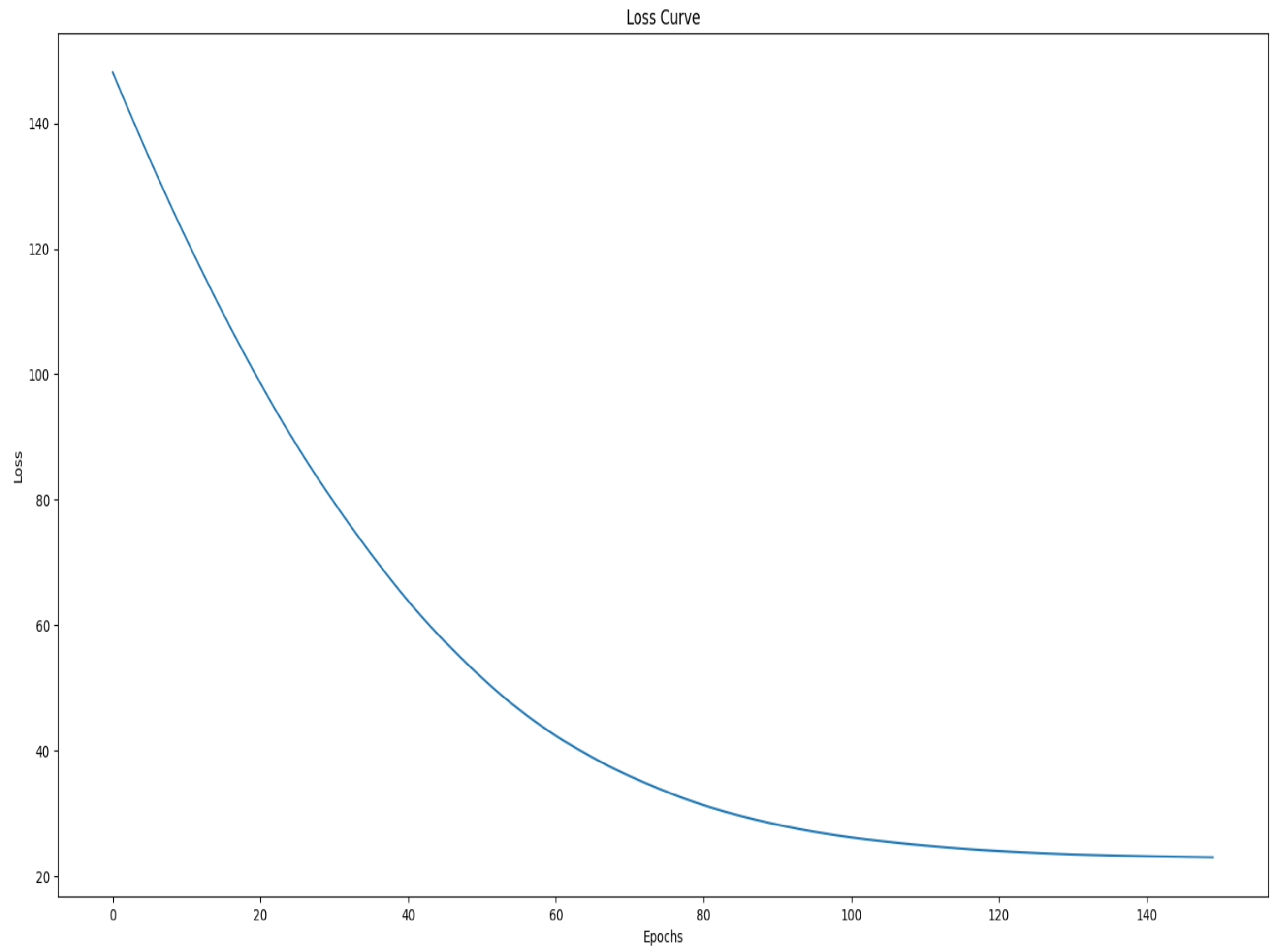
$$\downarrow$$

$$+ne$$



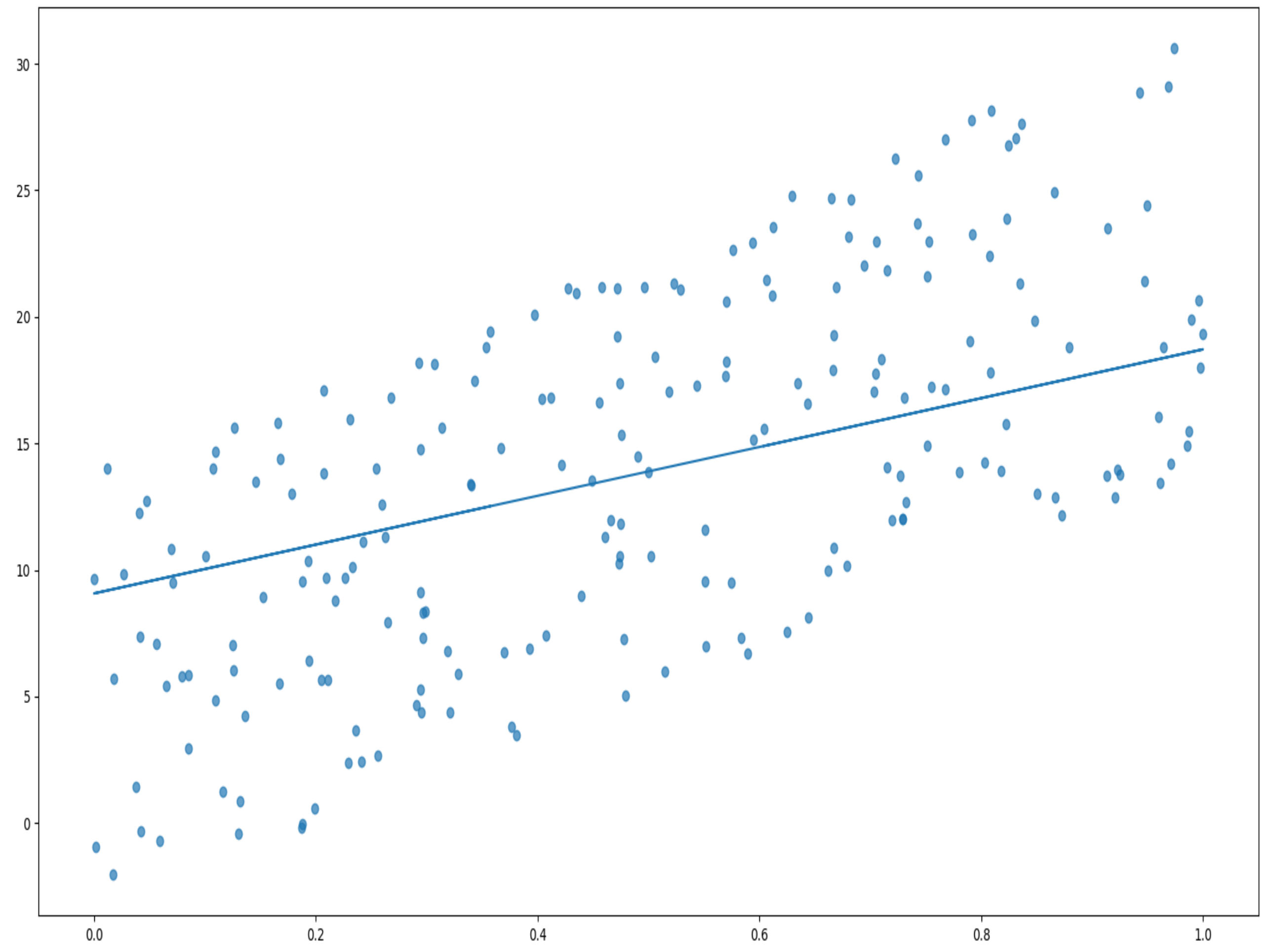
```
def Adam(self, epochs, eta, beta1, beta2, mini_batch=70):
    previous_theta = []
    previous_error = []
    moving_avg_1 = np.array([0] * len(self.theta))
    moving_avg_1 = moving_avg_1.reshape(len(moving_avg_1), 1)
    moving_avg_2 = np.array([0] * len(self.theta))
    moving_avg_2 = moving_avg_2.reshape(len(moving_avg_2), 1)
    for epoch in range(1, epochs+1):
        predicted = np.dot(self.X, self.theta)
        error = predicted - self.Y
        gradient = self.get_stochastic_gradient(self.X, mini_batch, error)
        moving_avg_1 = beta1 * moving_avg_1 + (1-beta1) * gradient
        moving_avg_2 = beta2 * moving_avg_2 + (1-beta2) * (gradient**2)
        mvavg1_hat = moving_avg_1 / (1 - np.power(beta1, epoch))
        mvavg2_hat = moving_avg_2 / (1 - np.power(beta2, epoch))
        self.theta = self.theta - eta * mvavg1_hat / (np.sqrt(mvavg2_hat) + 0.0001)
        error = (np.dot(np.transpose(error), error)) / len(X)
        previous_theta.append(self.theta)
        previous_error.append(error[0])
    return [previous_theta, previous_error]
```

The graph shows the loss curve over the epochs which gets smoothened and converge.



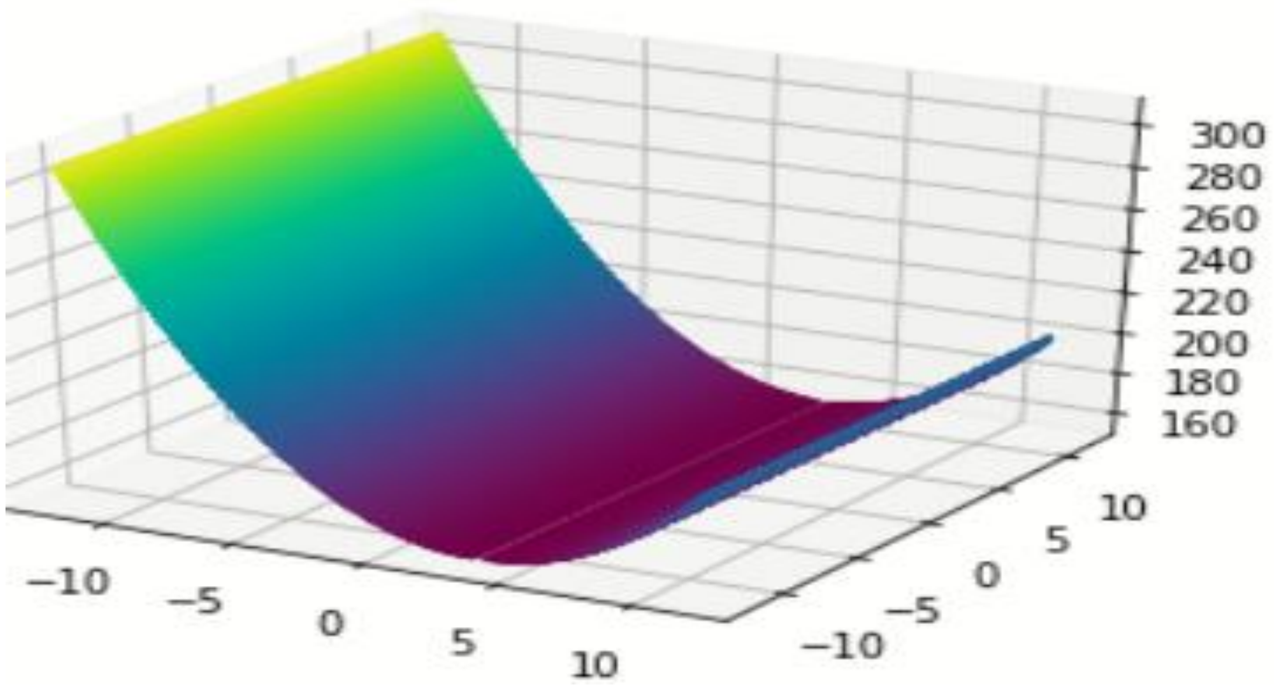
G21.

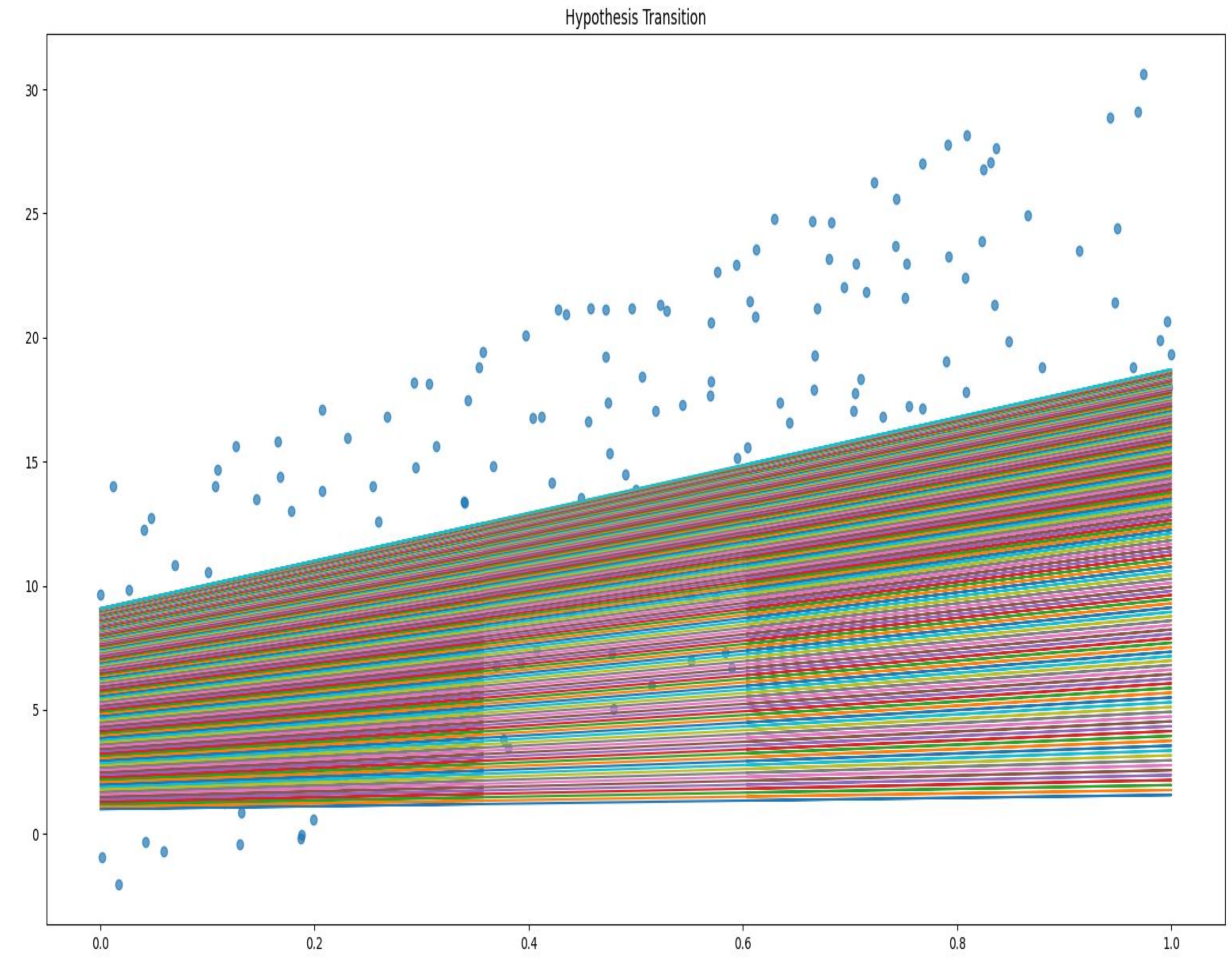
The above graph shows the best fit line model with the most optimum weights reached over epochs and optimized.

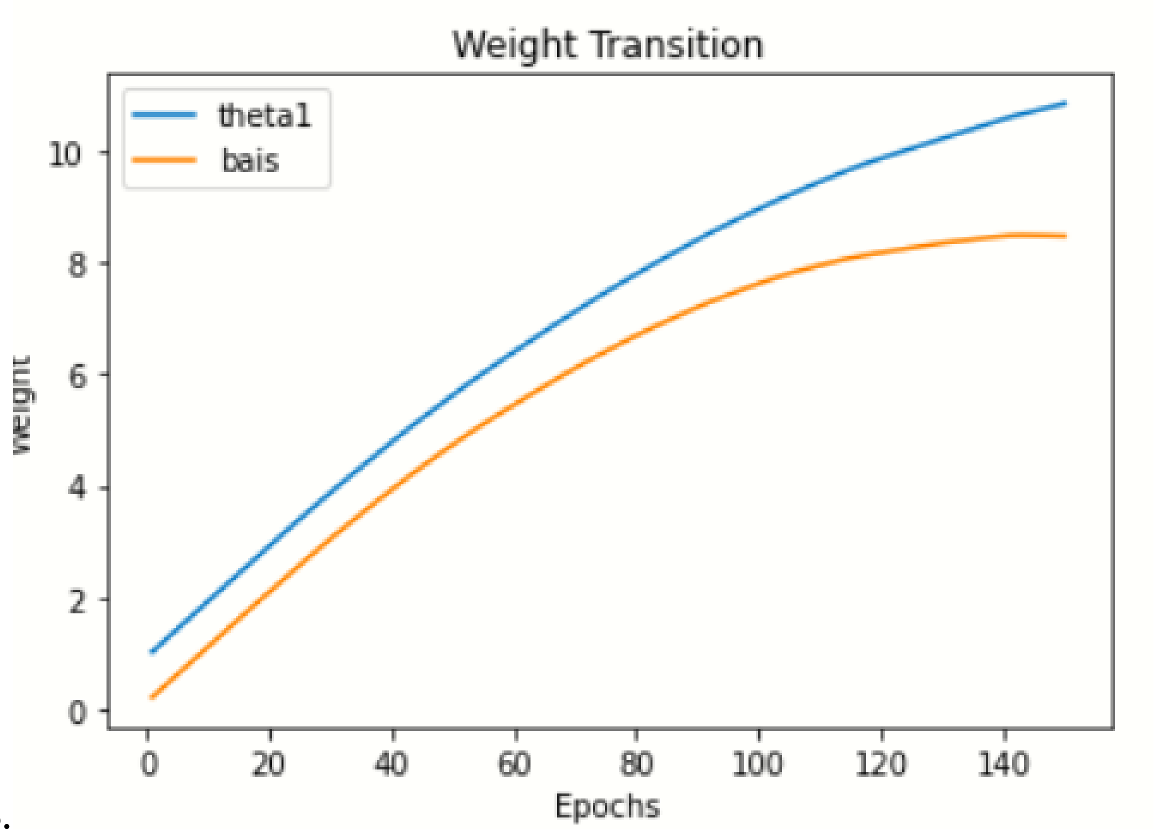


G22.

All weights to show the all weights with lines and to demonstrate the optimization happening such way over the epochs .



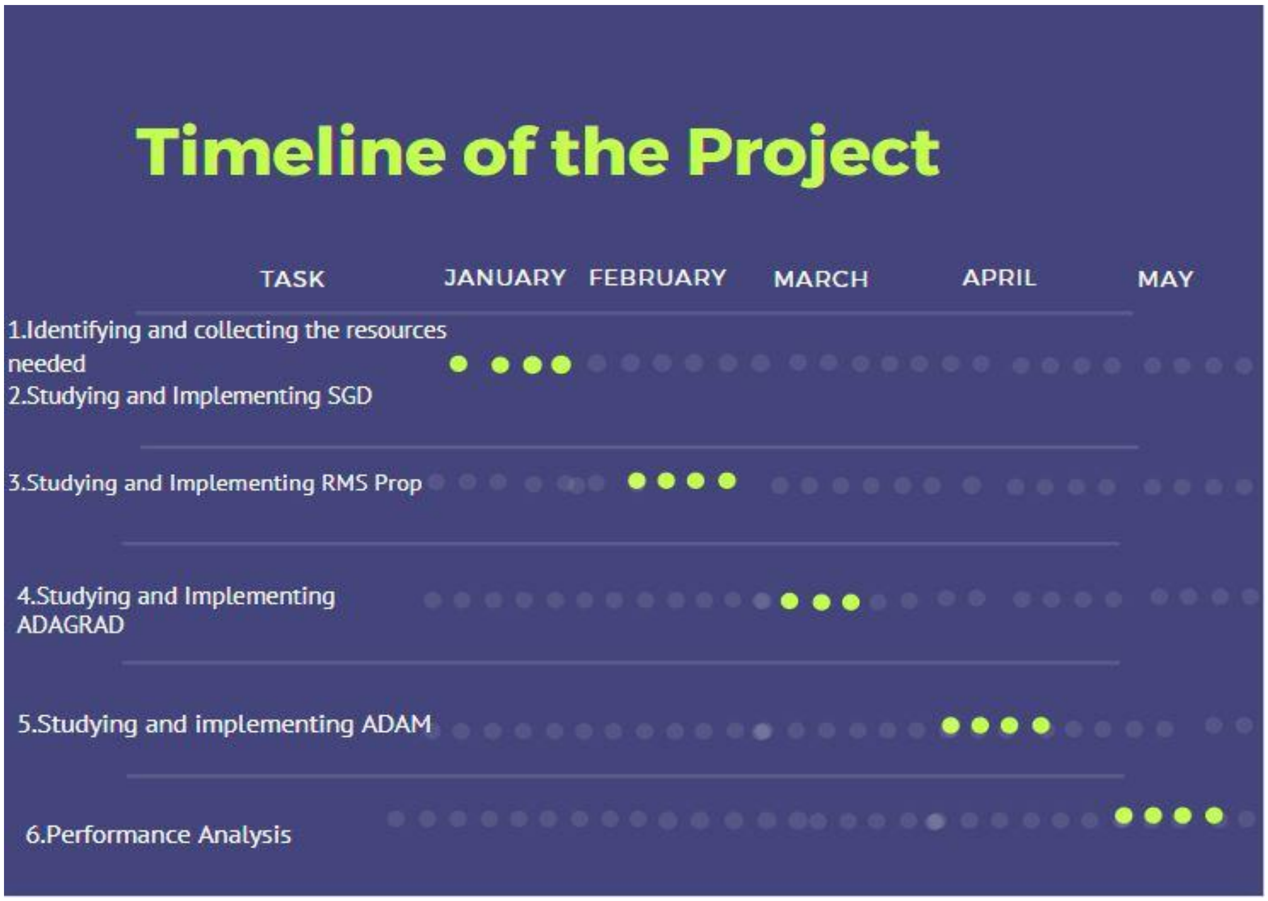

Hypothesis Transition
30
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0


Weight Transition
theta1
bais
weight
Epochs
0
2
4
6
8
10
0
20
40
60
80
100
120
140

G23.

TimeLine of the Project-

Gantt Chart-



Chapter 4: Performance Analysis

The results obtained by this project are in form of loss given by the model.

These are respective losses given by the model:

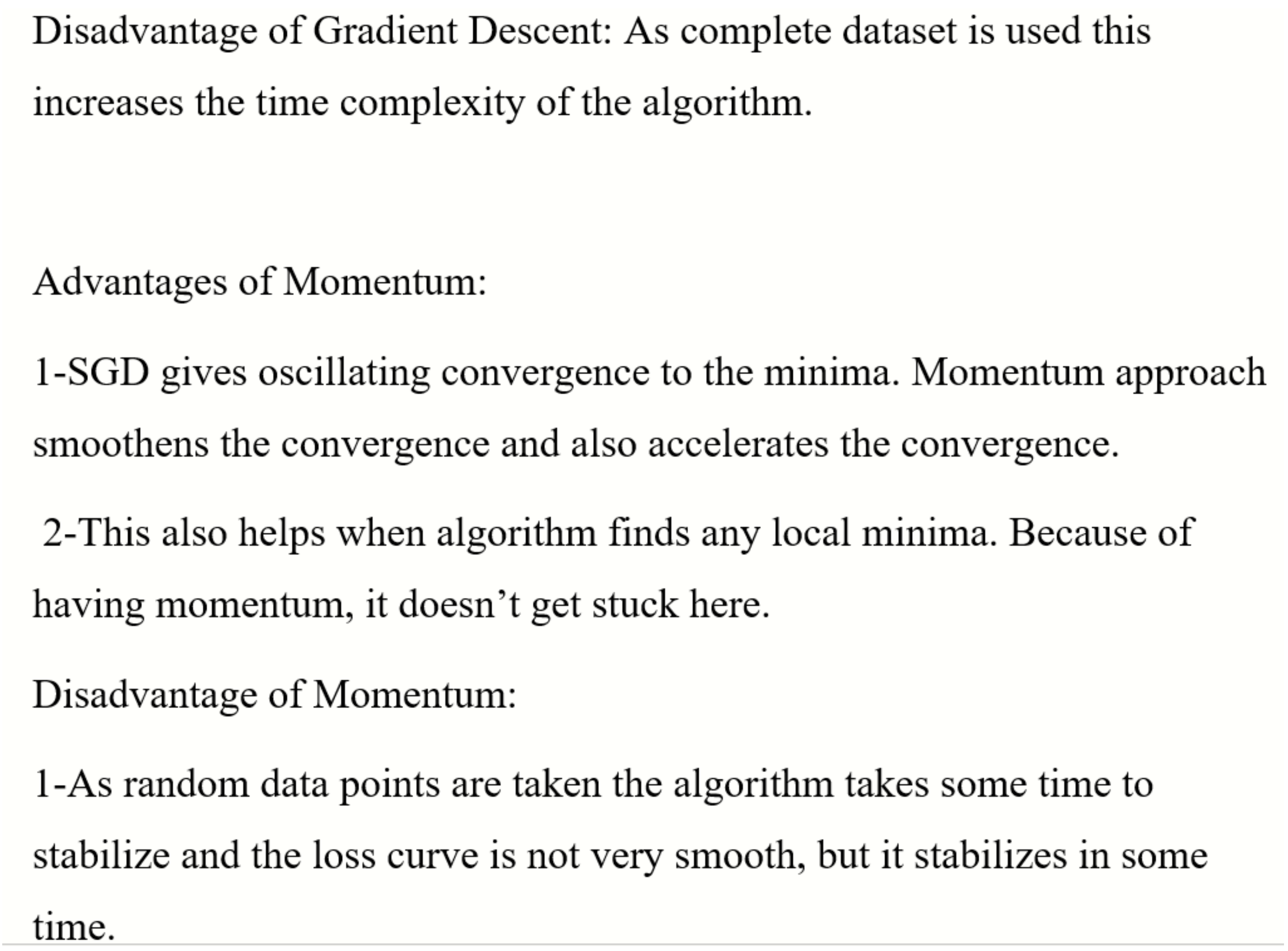
- ✓ Gradient Descent: 60.6
- ✓ Momentum: 57.06
- ✓ RMSProp : 51.4
- ✓ Adagrad:47.3
- ✓ Adam:33.2

i. The best efficiency is of Adam and adagrad as these implement adaptive learning rate along with smoothening of loss curve over epochs using momentum.

ii. Analysis on the basis of advantages and disadvantages:

Advantages of Gradient Descent: Gradient Descent convergence of the weight is very as can be seen in the graph. This is because gradient for each weight is averaged for the entire dataset.

2-Also, as it has momentum it can sometimes overshoot the global minima.



Disadvantage of Gradient Descent: As complete dataset is used this increases the time complexity of the algorithm.

Advantages of Momentum:

1-SGD gives oscillating convergence to the minima. Momentum approach smoothens the convergence and also accelerates the convergence.

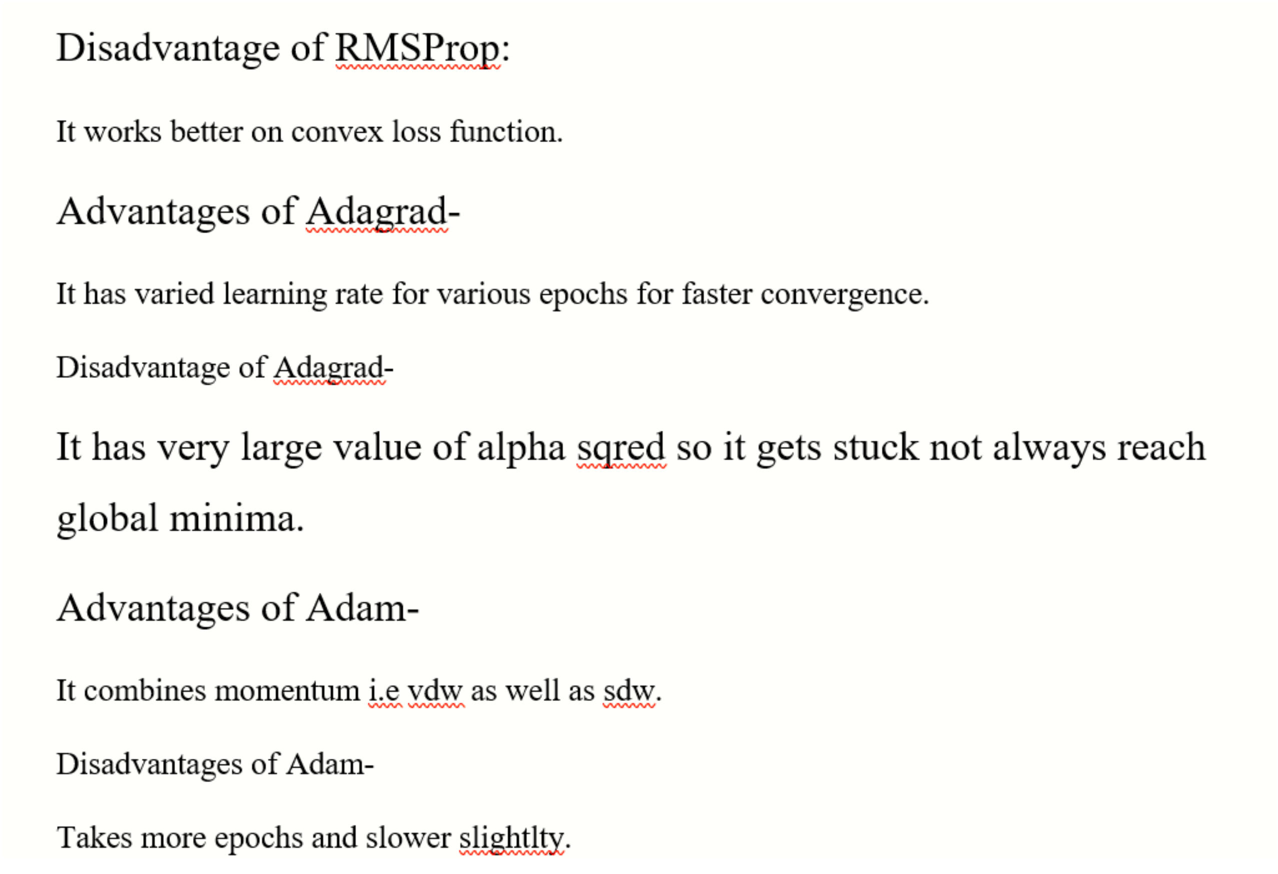
2-This also helps when algorithm finds any local minima. Because of having momentum, it doesn’t get stuck here.

Disadvantage of Momentum:

1-As random data points are taken the algorithm takes some time to stabilize and the loss curve is not very smooth, but it stabilizes in some time.

Advantages of RMSProp:

1-RMSProp converges faster than the Gradient Descent as it works on Stochastic Gradient Descent . RMSProp has knowledge about the previous gradients. The loss curve is smooth.



Disadvantage of RMSProp:

It works better on convex loss function.

Advantages of Adagrad-

It has varied learning rate for various epochs for faster convergence.

Disadvantage of Adagrad-

It has very large value of alpha sqred so it gets stuck not always reach global minima.

Advantages of Adam-

It combines momentum i.e vdw as well as sdw.

Disadvantages of Adam-

Takes more epochs and slower slightltly.

Chapter- 5: Conclusion

Conclusion:

We completed our Project and learnt different methods and algorithm in optimization of the machine learning algorithms. The work carried out came out to be successful and I got to the exposure to implement various Optimization Machine Learning Algorithms. I got to know that Optimization Algorithms in Machine Learning is a technique of improvising the already existing high complexity algorithms and training the models to perform efficiently and reach and converge to global minima and not limited slower and convergent upto local minimas.

Until now we found out that Gradient Descent algorithm works fine but take too much time computationally and converges slowly as it propagates in zig zag manner . It also gets stuck on local minima. But Momentum solves both the problem it converges faster and also comes out of local minima. RMSProp doesn’t overshoots and doesn't fluctuate like momentum and reaches the global minima .Adagrad has large values of alpha square that is dealt by using Adam.

Future Scope:

There are much more better and efficient algorithms to this project like AdaGrad, AdaDelta, Adam, RProp etc. that can be applied to it.

The already existing implemented optimization algorithms such as Gradient Descent, Momentum and RMS Propagation,Adam and Adagrad are efficient in themselves and the efficiencies of the implemented can be increased by using other optimization algorithms mentioned above and eventual comparison can be drawn out for better understanding of the algorithms .

References

1.Parpinelli, R.S., Lopes, H.S. and Freitas, A.A., 2002. Data mining with an ant colony optimization algorithm. *IEEE transactions on evolutionary computation*, *6*(4), pp.321-332.

2. Bai, Q., 2010. Analysis of particle swarm optimization algorithm. *Computer and information science*, *3*(1), p.180

3. Mirjalili, S. and Lewis, A., 2016. The whale optimization algorithm. *Advances in engineering software*, *95*, pp.51-67.

4. Schlipf, M. and Gygi, F., 2015. Optimization algorithm for the generation of ONCV pseudopotentials. *Computer Physics Communications*, *196*, pp.36-44.

5. Rajabioun, R., 2011. Cuckoo optimization algorithm. *Applied soft computing*, *11*(8), pp.5508-5518.

6. Trelea, I.C., 2003. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information processing letters*, *85*(6), pp.317-325.

7. Pelikan, M., Goldberg, D.E. and Cantú-Paz, E., 1999, July. BOA: The Bayesian optimization algorithm. In *Proceedings of the genetic and evolutionary computation conference GECCO-99* (Vol. 1, pp. 525-532).

8. Jiang, Y., Hu, T., Huang, C. and Wu, X., 2007. An improved particle swarm optimization algorithm. *Applied Mathematics and Computation*, *193*(1), pp.231-239.

9. Finkel, D., 2003. *DIRECT optimization algorithm user guide*. North Carolina State University. Center for Research in Scientific Computation.

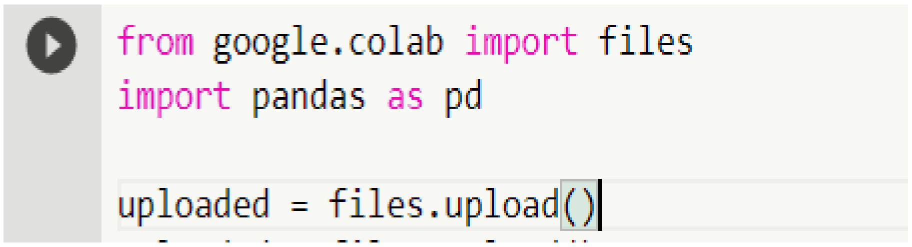
10. Wang, D., Tan, D. and Liu, L., 2018. Particle swarm optimization algorithm: an overview. *Soft Computing*, *22*(2), pp.387-408.

11.https://www.geeksforgeeks.org/optimization-techniques

12.https://en.wikipedia.org/wiki/Category:Optimization_algorithms_and_methods

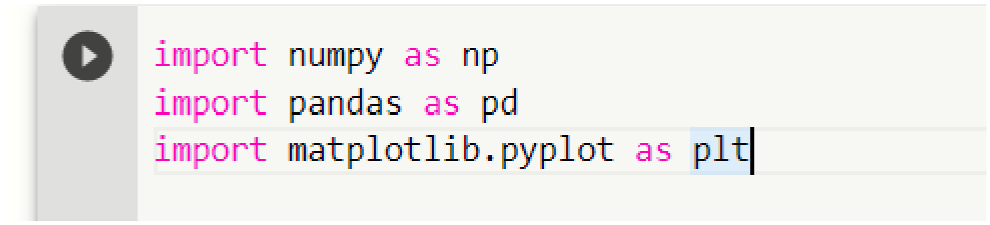
APPENDICES

1.Importing the data set in Google Colab



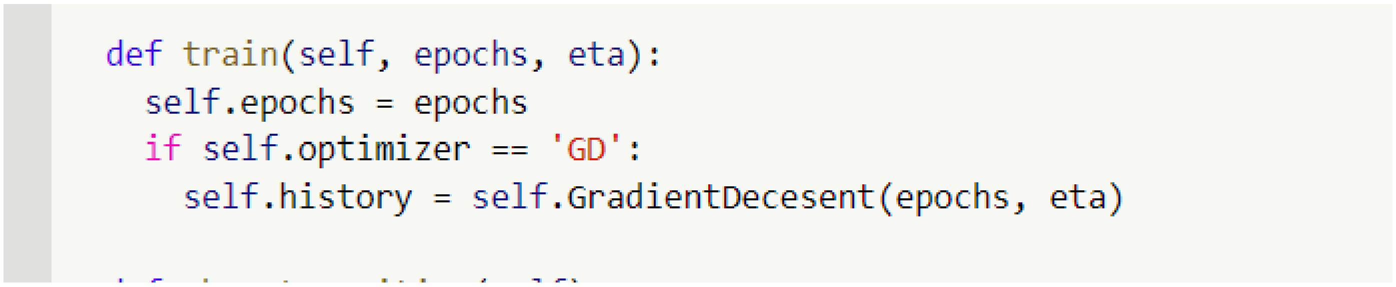
```python
from google.colab import files
import pandas as pd

uploaded = files.upload()
```

2.Importing the important and necessary libraries.



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

3.Training the data set.



```python
def train(self, epochs, eta):
  self.epochs = epochs
  if self.optimizer == 'GD':
    self.history = self.GradientDecesent(epochs, eta)
```


JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: …24/06/2021……………………….
Type of Document (Tick): ->Report B.Tech Project Report Paper

Name: _____Bhavye Sharma______________________ __**Department:** ___Computer Science And Information Technology______________ **Enrolment No** ___171473______

Contact No. ________7696299860______________________E-mail. _______bhavyesharma33@gmail.com__and __171473@juitsolan.in___________________________

Name of the Supervisor: _Dr Rajni Mohana____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): ___To study and_Implement Optimization Algorithms___________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

- Total No. of Pages =65
- Total No. of Preliminary pages =5
- Total No. of pages accommodate bibliography/references = 3

Bhavye

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ………18………..(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

[illegible]

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

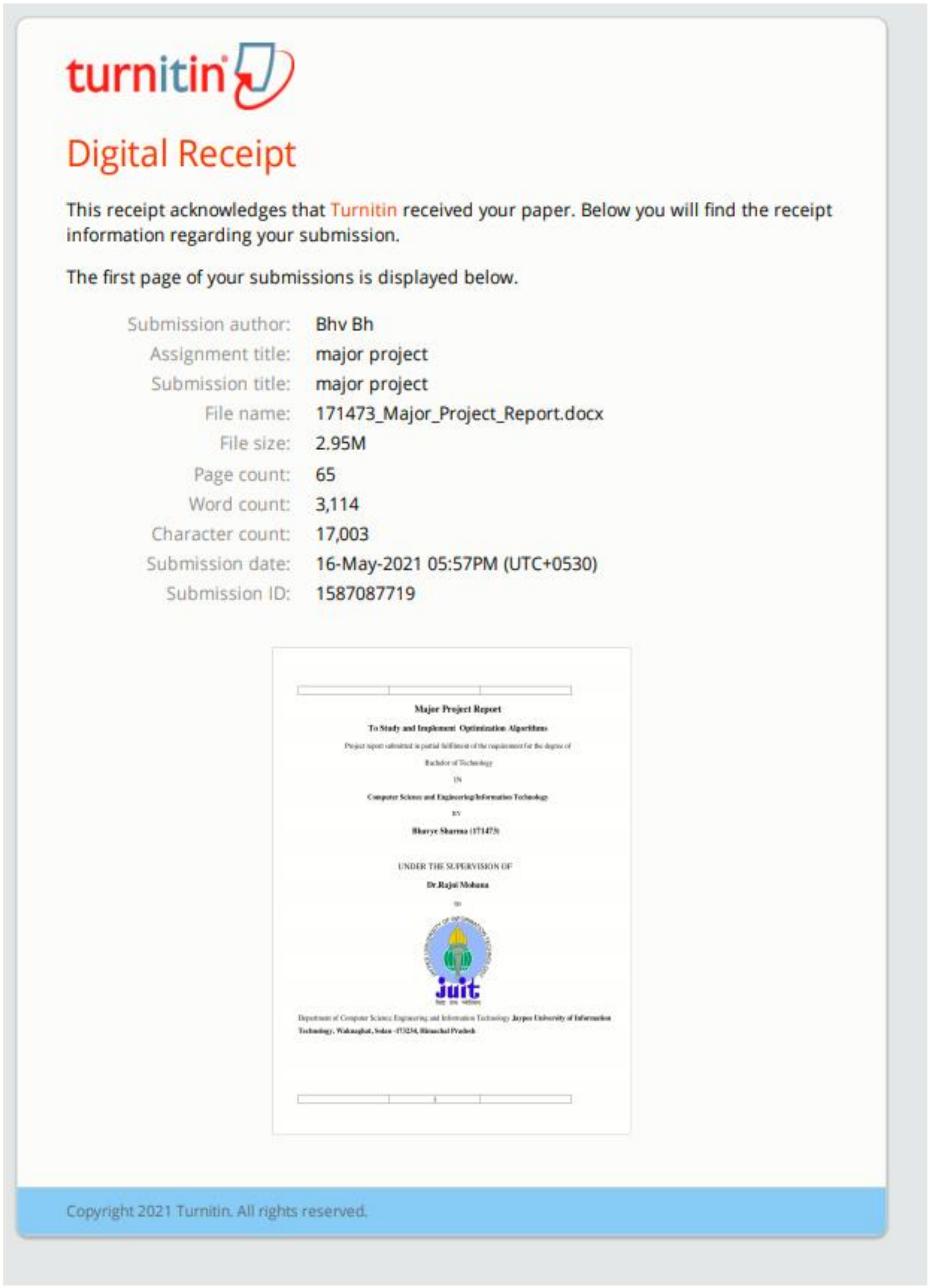
The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index**	**Abstract & Chapters Details**

		(%)		
	• All Preliminary Pages • Bibliography/ Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Page counts	
			File Size	

Checked by
Name & Signature **Librarian**

……



turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Bhv Bh
Assignment title:	major project
Submission title:	major project
File name:	171473_Major_Project_Report.docx
File size:	2.95M
Page count:	65
Word count:	3,114
Character count:	17,003
Submission date:	16-May-2021 05:57PM (UTC+0530)
Submission ID:	1587087719

Major Project Report

To Study and Implement Optimization Algorithms

Project report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

IN

Computer Science and Engineering/Information Technology

BY

Bharye Sharma (171473)

UNDER THE SUPERVISION OF

Dr.Rajni Mohana

to

Department of Computer Science Engineering and Information Technology Jaypee University of Information Technology, Waknaghat, Solan -173234, Himachal Pradesh