

Major Project Report

Implementing Optimization Algorithms to Increase Efficiency of Linear Regression Model

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

Bachelor of Technology

IN

Computer Science and Engineering/Information Technology

BY

Achintye Sharma (171331)

UNDER THE SUPERVISION OF

Dr. Ekta Gandotra

to



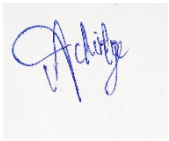
Department of Computer Science Engineering and Information Technology

Jaypee University of Information Technology, Wahnaghat, Solan -173234, Himachal Pradesh

Candidate's Declaration

I hereby declare that the work presented in this report entitled "Implementation of Optimization Algorithms to Increase Efficiency of Linear Regression Model" 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 May 2021 to July 2021 under the supervision of **Dr. Ekta Gandotra** (Designation and Department name).

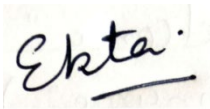
The matter embodied in the report has not been submitted for the award of any other degree or diploma.



Achintye Sharma

171331

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



Dr. Ekta Gandotra

Assistant Professor

Department of CSE & IT

ACKNOWLEDGEMENT

I want to express my sense of gratitude towards Dr. Ekta Gandotra Department of Computer Science & Engineering, Jaypee University of Information Technology, Waknaghat, for providing me the excellent opportunity for working with her. I feel myself thankful for the regular encouraging attitude, cooperation and the motivation as well as the support which kept me up for fully finishing the project properly as well as successfully. The expert guidance which was provided helped me in finishing the project.

Table of Contents

Serial Number	Description	Page No
1.	Candidate's declaration	2
2.	Acknowledgement	3
3.	List of Figures	6
Chapter 1	Introduction	
1.1	Introduction	8
1.2	Types of Machine Learning Algorithms	9
1.3	Problem Statement	10
1.4	Objective	11
1.5	Methodology	11
1.6	System Design	12
Chapter 2	Literature Survey	13
2.1	Gradient Descent	15
2.2	Tools and Technologies used	
Chapter 3	System Development	16

3.1	Feature Selection	17
3.2	Member Functions used	19
3.3	Implementation of the Optimizing Algo	20
3.4	SGD	25
3.5	RMSporopagation	30
3.6	Adagrag	34
3.7	Adam	38
Chapter-4	Performance Analysis	44
4.1	Results	45
4.2	Advantages/Disadvantages	46
Chapter-5	Conclusion	48
6	References	50
7	Appendices	51

List of Figures

Figure Number	Description	Page No
Figure 1.1	System Design	12
Figure 3.1.1	Scatter Plot of Total Rooms vs Population	17
Figure 3.1.2	Scatter Plot of Total Population vs households	18
Figure3.1.3	Scatter Plot for Longitude vs Latitude	19
Figure 3.3.2	Loss Curve for GD	24
Figure 3.3.1	System Design	25
Figure 3.3.3	Weight transition for GD	27
Figure 3.3.4	GD best-fit-line	27
Figure 3.4.1	Loss Curve of Stochastic GD	33
Figure3.4.2	Weight Transition	38
Figure 3.4.3	Best Fit Line	38
Figure 3.8	Timeline of the Project	43
Figure 4.1	Comparison Graph	45

Abstract

Linear Regression model takes Linear dataset (kaggle dataset -California Housing), predicts the Price of the Houses and then we reduce the error with the help of optimization Algorithms.

In this project we will develop and implement different optimization algorithms from scratch, and learn the role of their parameters which will enable us to tune the hyperparameters in a targeted manner to improve the performance of our model. Algorithms are first given random initialization which are then updated over the epoch iterations to achieve out global minima. The Error updated over the iterations is calculated using the Loss Function which is updated over by giving new values to our weights which are slope and intercept.

Various Different Optimization Algorithms are implemented and the Performance Analysis of all the Algorithms implemented is done.

Thus, increasing the efficiency of our model.

Chapter 1

Introduction

1.1 Introduction:

1.1.1 Term Optimization: Training the ML model doesn't work perfectly at once. The output of the ML our model will not be giving correctly and efficiently, it will be away from actual o/p or expected o/p. For knowing how away we are from the o/p we use Mean Squared Error.

$$Func(MSE) = f(expected, predicted)$$

The MSE depends upon both and actual and predicted value. We try to reach the value of this loss function as close as 0. 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.

1.1.2What are Optimization Algorithms: Optimization Algo is nothing but set of instructions/steps which when repeated until error given by our Loss(MSE) function becomes minimum.

Optimization Algo

predict = model(parameter)

(start of the loop)-

predicte = changes_in_model(parameter)

Recomputing the loss (): Loss_Function(predict, actual-val)

If the loss is \leq equal to a threshold - (stopping the loop)

(End of loop)

1.2 Machine Learning-

ML is a study of causing PCs to learn without being expressly program. It is firmly identified with the computational measurement, which centers around making the predicted values utilizing the PCs. Its application across the business issues, AI can likewise be alluded to as anticipating investigation. ML is firmly identified with the registering measurements. ML centers around the advancement of the PC programs that can be accessed information and can be utilized to learn themselves. The way toward realizing which starts with perceptions or data, like the models and the guidance, to search for the examples in the information which settles on better choices for future premise on the models which are given. The essential point is permitting PCs to adapt naturally without the human intercessions or the aids and changing activity.

1.2.1 Machine Learning and Its Types-

1. Unsupervised Learning

2. Supervised Learning

1. Supervised Learning -

Supervised Learning is a type of the learning where the data set is given and we know how correct output looks like, having idea that there is a relationship between the input and output the input and output.

This is a learning of 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 is categorization.

2. 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.

1.3 Problem Statement

To illustrate the implementation of various Optimization Algorithms on Our Linear Regression model.

1.3.1 Application Scenario-

1. It can be applied in Supervised Machine Learning problems.
2. All the datasets that are applicable on Linear Regression.
3. There should be linear relationship between dependent and independent variable.

1.3.2 Use Cases

1. **Linear regressions** can be used in business to evaluate trends and make estimates.
2. Real Estate prices forecasts that follows linear relation.

1.3.3 Implemented Optimizers are:

- 1) **Gradient Descent (GD)(Batch Gradient)**
- 2) **Momentum Stochastic gradient (SGD)(Mini Batch)**
- 3) **RMS-propagation**
- 4) **Adagrad(Adaptive)**

5) Adam

1.4 Objective

Implementing and illustrating various optimization algorithms and compare them for linear regression model .

1.5 Methodology

- (i) Taking a linear –dataset (California Housing) and implementing feature engineering (feature selection).
- (ii) Implement a Linear Regression algorithm.
- (iii) Linear Regression takes Optimizers as the argument with random weights initialized .
- (iv) Optimizing Linear Regression by using different optimization algorithms to reduce the loss function that is the Mean Squared Error.
- (v) Plotting graphs.

1.6 System Design

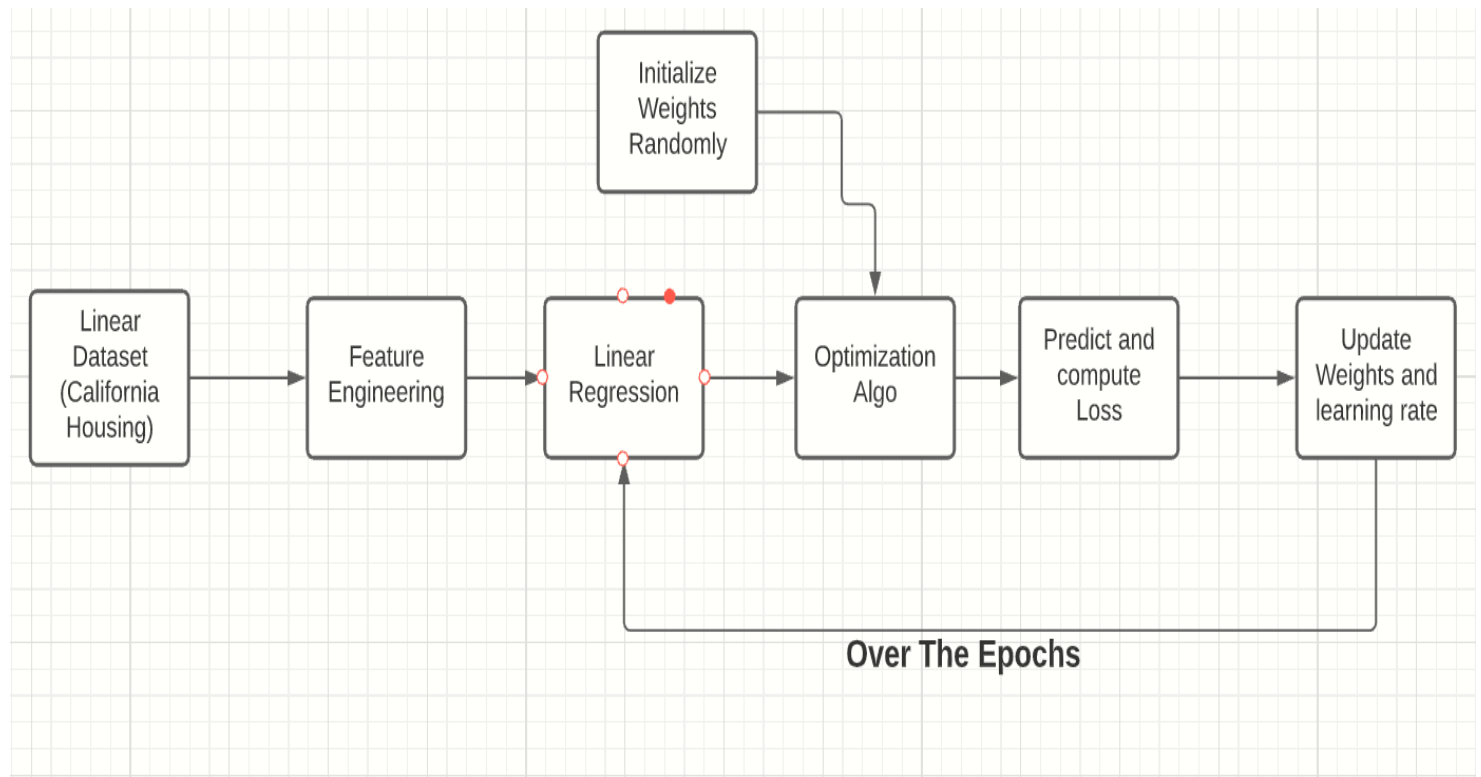


Figure 1.1: System Design

Chapter 2

Literature Survey

1. Working of the Optimization Algorithms [Hands on Machine Learning using Python by Aurelien Gueron].
2. Implementation of the Algorithms [towards datascience]
3. Creating dataset and feature selection [Analytics Vid]

2.1 Gradient descent (GD)

It is an optimization algo is being use to minimise function i.e Loss/Error which here is Mean Squared Error (MSE) by the updation of various other parameter that are θ_0 (intercept) and θ_1 (slope) and derivative of $J(\theta)$ with respect to parameter (θ).

The learning rate η helps us determine size of step involving that is taken to reach our minima and eventually which is global minima in n iteration

The working and Implementation of Algo in machine learning is performed in Python Language.

1. A Linear Regression Model (California Housing) is used.

2. Parameters that is intercept and slope were updated and adjusted and observed changes have been plotted .

Mean Squared Error - Loss Function is calculate errors.

2.2 Technologies and the Tools Used

2.2.1 Languages Being Used-

1. Python

1. Python being high level language can be easily implemented and is being use in one of various platform which are machine learning (ML), data mining etc.

2. It includes one of the important library which are powerful and helps in visualization of the data points and helps process it.

3. Some of the libraries which are included -, Pandas, Matplot-Lib, Numpy.

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 license 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.

3. NumPy

(i)Numpy is most frequently used and one of the important libraries which are being used in the Python lang which helps us for work in data structures array.

(ii)Numpy Lib also known as Num-Py, helps us in doing mathematical operations as well as logical operation on various data structures including arrays.

4.Matplot-Lib:

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

5.GoogleColab:

(a)Google Colab is a significant and useful asset utilized in different information warehousing,machine learning and profound learning execution purposes.

(b)It can be utilized for far off execution of the tasks where at least two individuals can use project at once.

(c)It upholds various file formats arrangements, for example,pictures,html , csv, xml, latex and so forth

6.SK-Learn:

Chapter 3

System Development

3.1 Feature Selection

(a) Data set has been first analysed. Our Dependent variable i.e Y is Median_House.

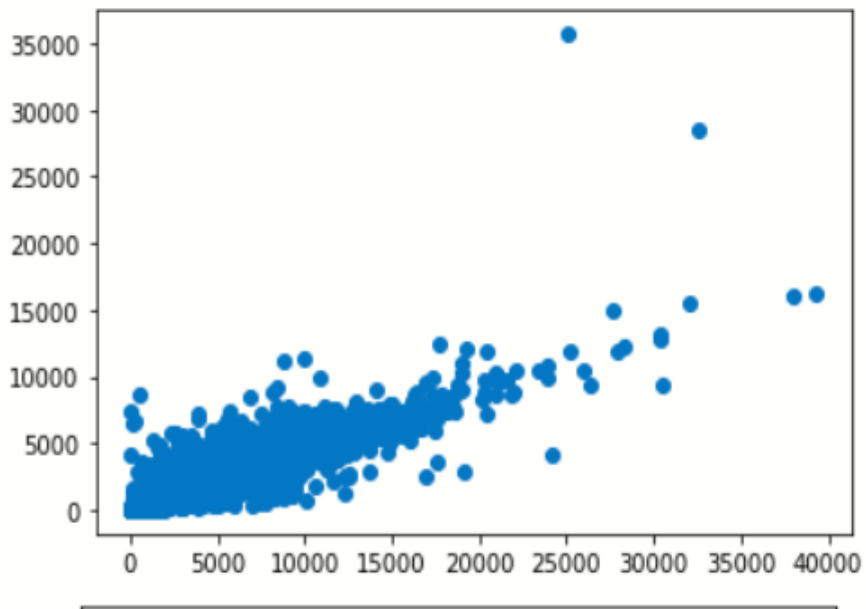
(b) Rest of the Features have been taken and Feature Selection is performed.

(c) In Feature Selection the Highly correlated features i.e Total Rooms and Total Households have been dropped as highly correlated with Total Population Feature.

(d) The features are then passed in GD Gradient Descent which gives weights to each feature and the features with the highest weights are selected.

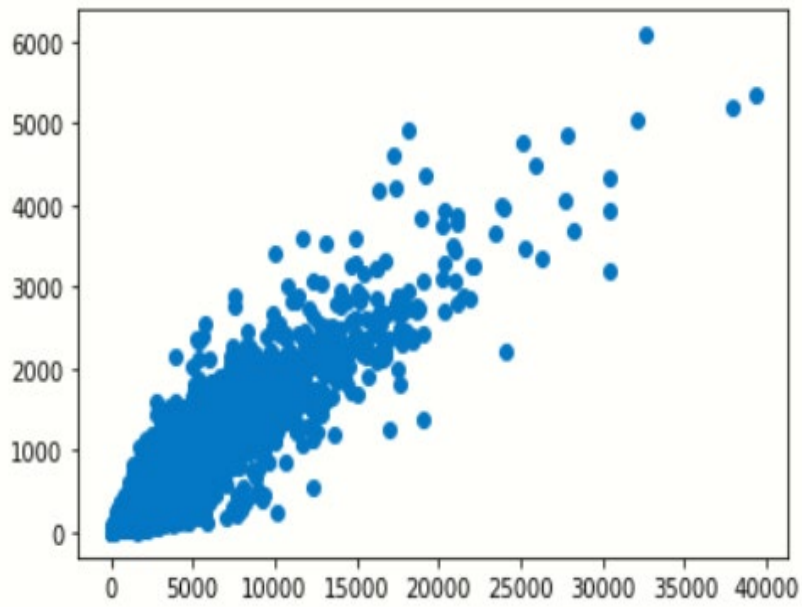
$$\textit{Dataset: } y = m * x + c + \textit{noise}$$

3.1.1 Scatter Plot for the Features



Scatter Plot of Total Rooms vs Population

Figure 3.1.1: Scatter Plot of Total Rooms vs Population



Scatter Plot of Total Population vs HouseHolds

Figure 3.1.2:Scatter Plot of Total Population vs Households

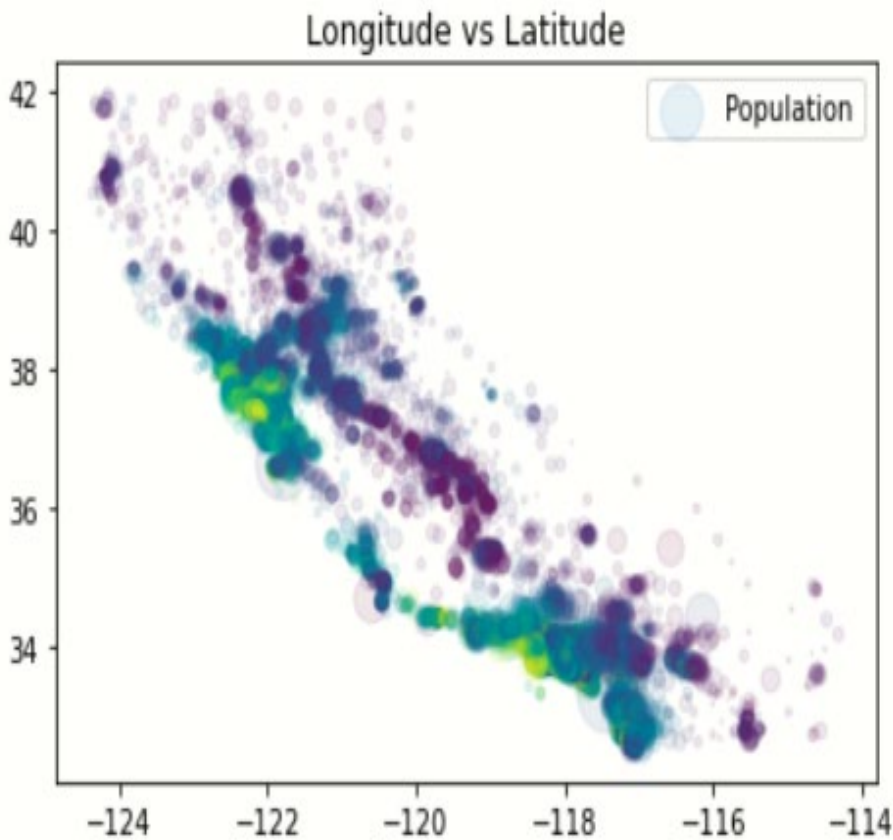


Figure3.1.3:Scatter Plot for Longitude vs Latitude .

3.2 Member Functions used

- 1.fit (): Initializes the weights and makes the dataset workable.
 - 2.train (): Use to start the training process. Takes in number of epochs and initial learning rate.
 - 3.predict (): Uses the final weights to calculate predicted values and calculates the net error.
 - 4.show_trainsition (): Takes history and plots all the hypothesis from initial to final weights.
 - 5.show weightTransition(): Shows how the weights changes with epochs.
 - 6.show_lossCurve (): Takes history which contains all the errors calculated over every epoch and plots how the loss decreases.
 - 7.final_fit (): Shows how the final weights fit the data points.
 - 8.Other helper function: Provides other functionalities for the Linear Regression class to work.
- Regression class to work.

3.3 Implementation of the Optimizing Algo:

3.3.1 Gradient Descent (GD):

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 GradientDescent(self, epochs, eta):
```

->Predicting and adjusting Theta:

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

3.3.1.1 Definition

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

2.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.

$$\hat{y}_{predicted} = w_1x_1 + w_2x_2 + \dots + w_nx_n = \sum w_ix_i$$

On differentiation we get,

$$dJ/dw_i = 2*(y_predicted - y_actual)*x_i,$$

$$error = y_predicted - y_actual \text{ so now } dJ/dw_i = 2*error*x_i$$

Gradient = Array of dJ/dw_i .

Update Statement: $w_i = w_i + (learning_rate) * gradient[i]$.

3.3.1 Algorithm of the Gradient Descent (GD)

- The learning rate α is kept constant only the values of the slope(m) and intercept(c) is being randomly initialize and gets update over the iterations.

- The optimal weights are obtained over the iterations and graphs of loss/error vs epoch gets plotted for observation for reaching the minima.
- If error is high, we have to move in the right direction i.e we have to keep on increasing the values in +ve direction of axis.

2.The equations of the Algo is

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

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

Alpha(a) is the learning rate that is kept 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 in which movement is done for reduction of the cost function which is MSE.

3.Loss Function

Mean Squared Error (MSE) is differentiated with respect to the weight for finding the gradient

WE Take all of the data points in every iteration.

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

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

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

Differentiation wrt to m Similarly for c

```
# Algorithm 1
def GradientDescent(self, epochs, eta):
    previous_theta = []
    previous_error = []
    for epoch in range(epochs):
        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 = (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]
```

FIGURE 3.3.1: Algorithm of GD

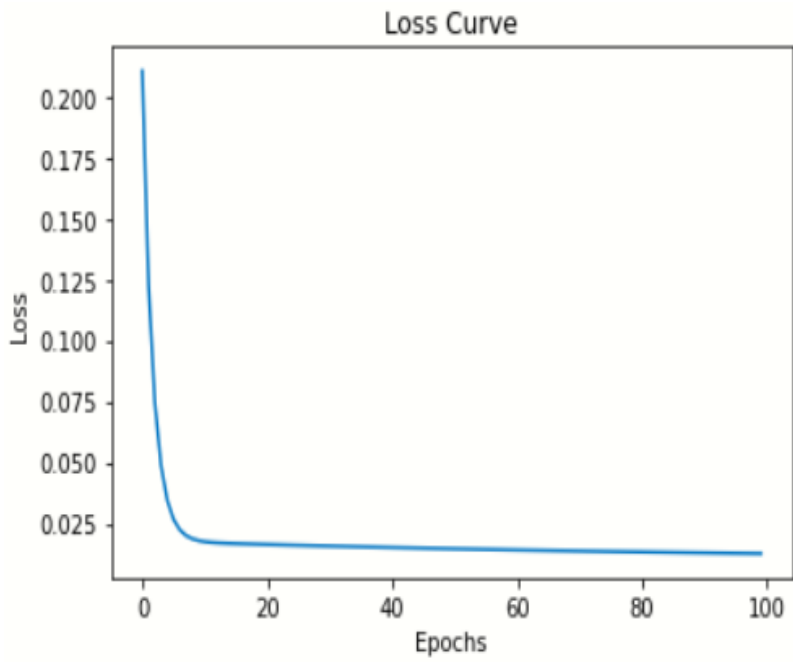
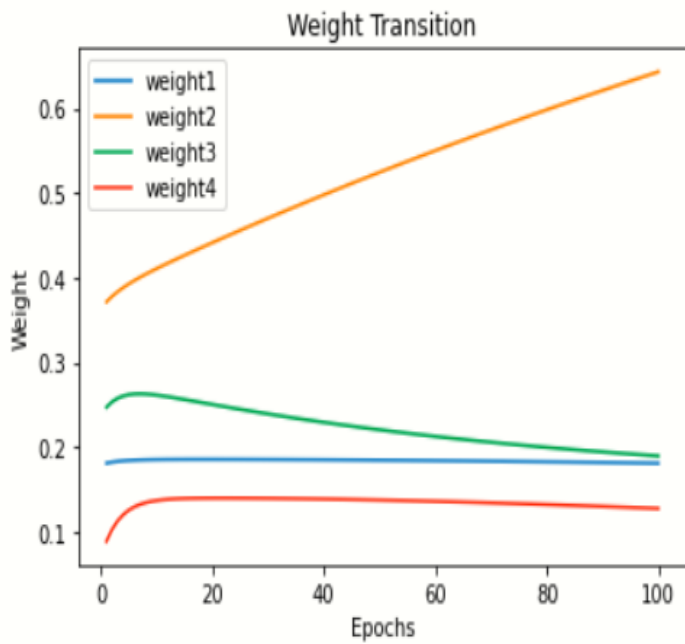


FIGURE 3.3.2: Loss Curve for GD



Error for GD = 0.07310575088828815

Figure 3.3.3: Weight transition for GD

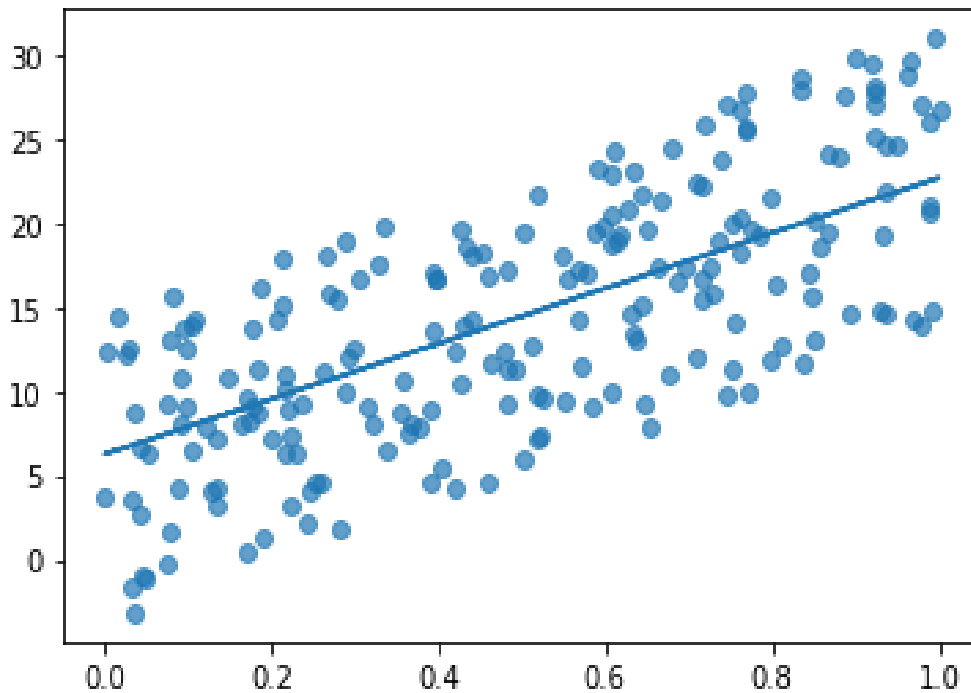


Figure 3.3.4:GD best-fit-line

3.4 Momentum-SGD Stochastic Gradient Descent

1.Linear Regression object takes Momentum as argument to call Gradient Descent.

2.Momentum works on Stochastic Gradient Descent(SGD). SGD is almost same as Gradient Descent (GD).

3.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.

4.Learning rate which is kept const, only the weight i.e slope(m) and the intercept(c) are initialized randomly and update over iterations

5.The converging of the slope is not so smooth as the mini batch oscillates very much which is to be dampen where as gradient descent has smooth convergence of loss curve.

6. The oscillations have been dampened which is achieved by providing the momentum to the weight i.e. exponential weighted averages.

7. Giving the more weights to present iteration gradient and less to previous gradient. Beta=0.95 initializing, alpha(a) is constant

•

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

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

```
# 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 + (1 - gamma) * gradient
        self.theta = self.theta - eta * 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]
```

1. 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

MSE is diff w.r.t to the weights(w) finds slope.

1. Implementing Mini Batch of N data points Randomly using seed

2. For decreasing this, taking a data point from dataset for calculation of the slope. These points are picked arbitrarily from the set.

3. 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.

4. The name itself suggest, the algo gains momentum. For instance, off chance that we toss a ball from the curve, it will in any case have speed when it arrives at minima, and will jump out of it and keep looking for global minima.

5. To numerically get this, we partner a load with each inclination. This weight is most extreme for current slope and least for the principal inclination.

These weights are constant to the power. The constant beta(b), has a value between [0,1].

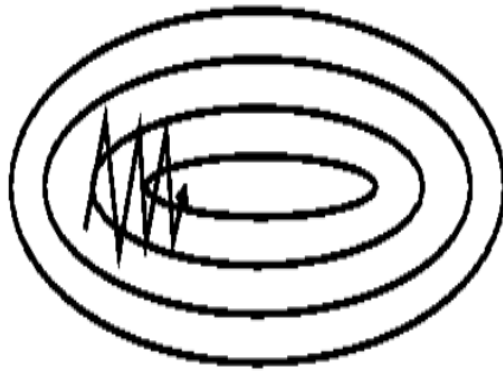


Image 2: SGD without momentum

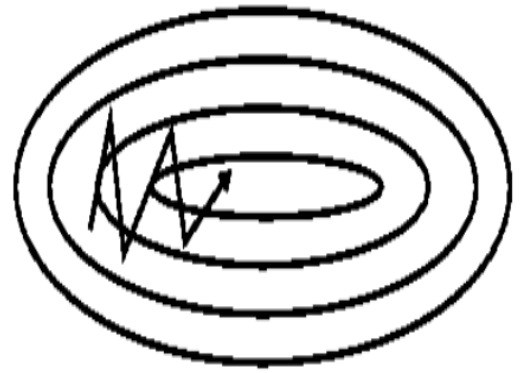


Image 3: SGD with momentum

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 = $w_i = w_i + \text{Momentum}$

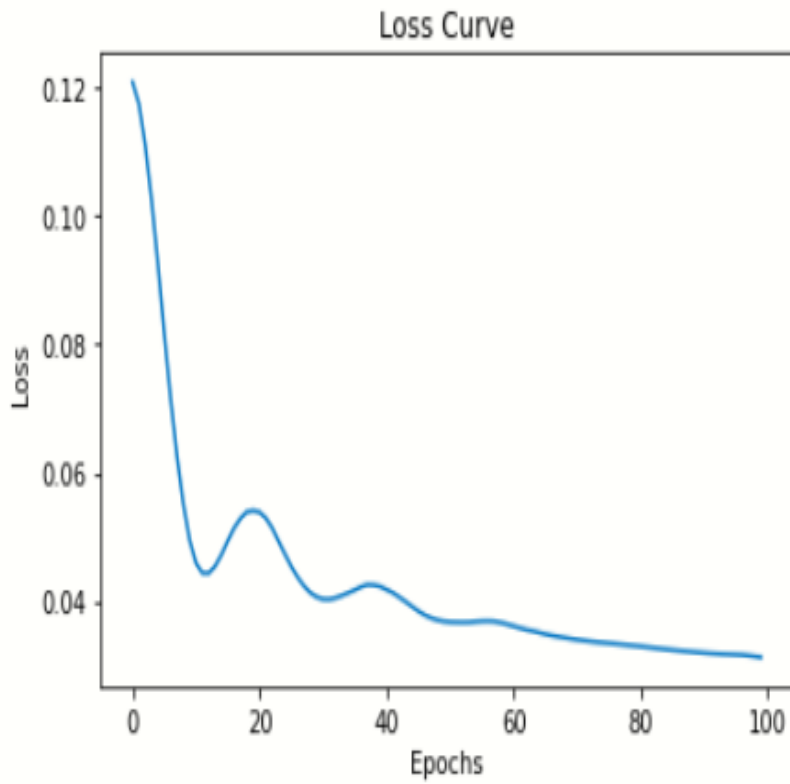


Figure 3.4.1: Loss Curve of Stochastic GD

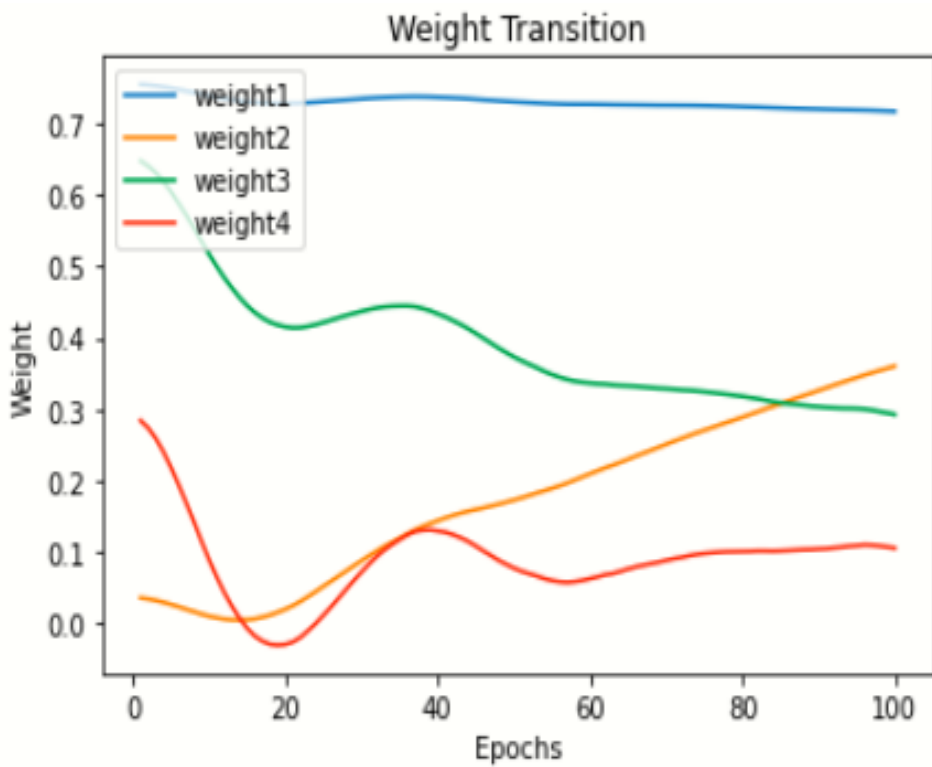


Figure 3.4.2: Weight Transition

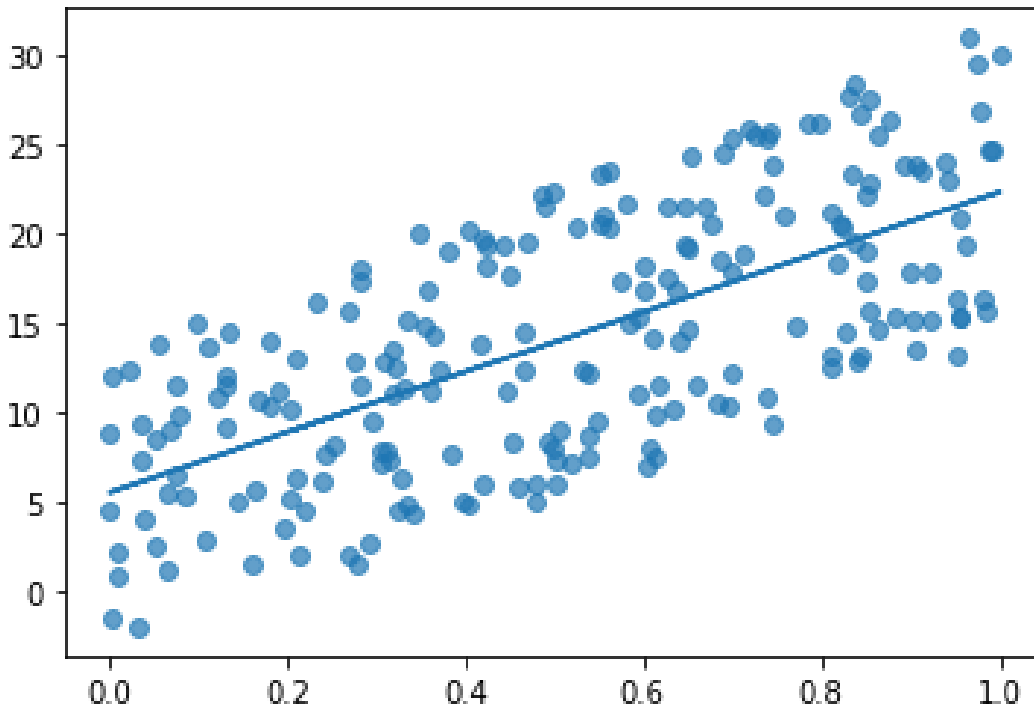


Figure 3.4.3 :Best Fit Line

3.5 Root Mean Square Propagation

1. Linear Regression object takes “RMSP” as argument to call Gradient Descent.
2. The learning rate is not constant where as it is varied over epochs to converge faster.
3. The Smoothing 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.
4. This happens as the previous and the new weights are almost the same so the convergence comes to still.
5. The value of learning rate in Vdw is denoted by beta and is usually set to 0.9 .

The v_{dw} and v_{db} are initialised to 0. For every epochs we calculate and update

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

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

weights with respect to weight components.

3.5.1 Weight Equation

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

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

3.5.2 RMS Algo-

```
# 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]
```

RMS-Propogation is R-Prop combined with Stochastic(SGD) having changes. In RMS-Propogation is using the exponential weighted averages that is square of the gradients. The reason behind it being values of average gardient.

$$\text{Moving_average} = \beta * \text{Moving_average} + (1-\beta) * \text{Gradient}^2$$

Update Statement:

$$w_i = w_i - (\text{leraning_rate} / \text{Moving_average}^{0.5}) * \text{Gradient}$$

3.5.3 Outcomes

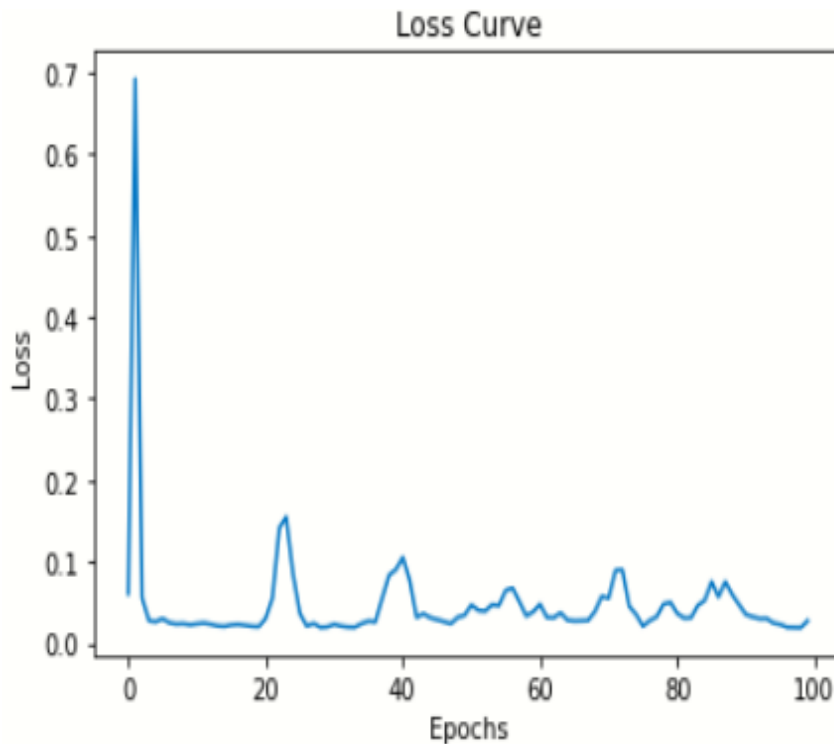


Figure 3.5.1: Loss Curve

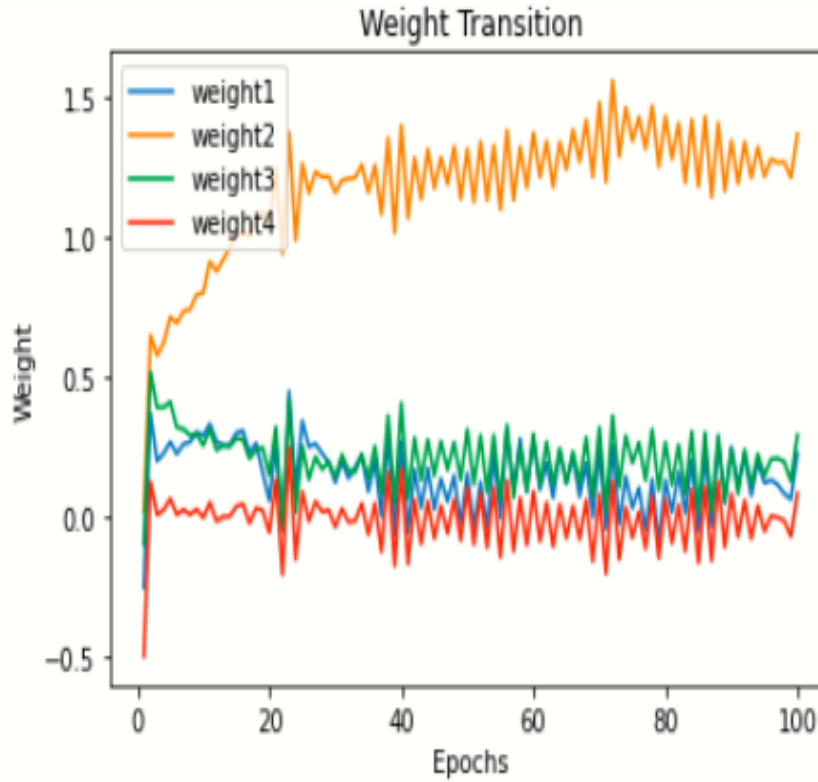


Figure 3.5.2: Weight Transition

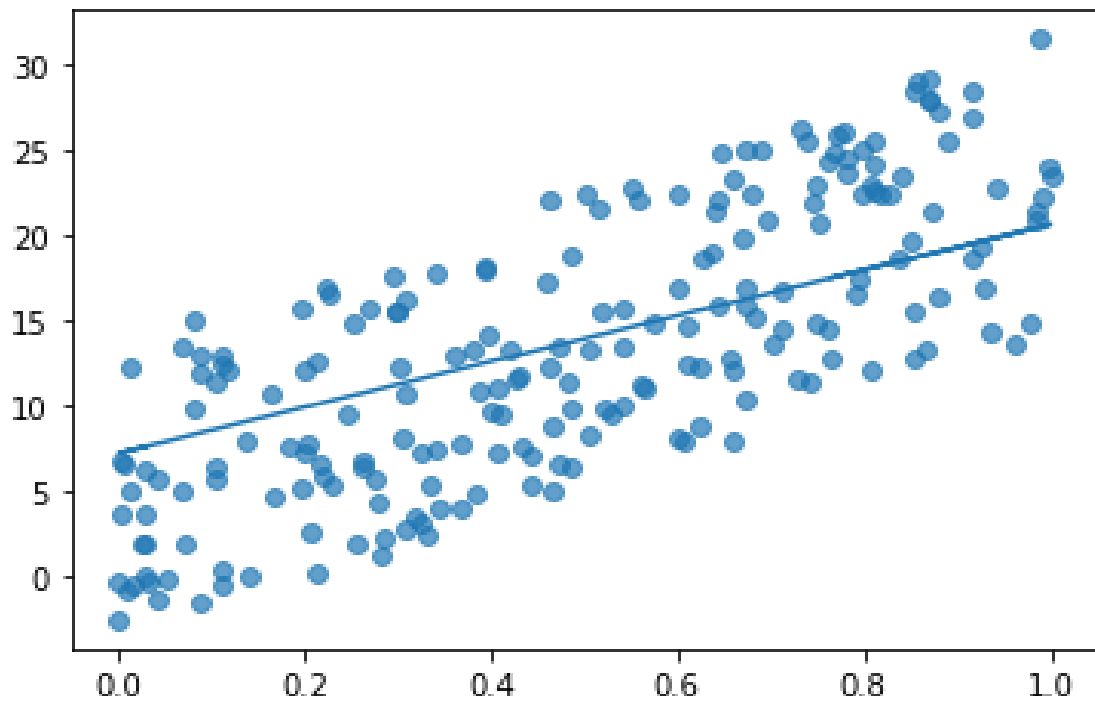


Figure 3.5.3: Best Fit Line

3.6 Adagrad

1. Adaptive Gradient that is adaptive learning learning-rate
2. Given the repeated iterations and the epoch learning rate(α) can be updated as the
3. Data points of the mini batch can have sparse /dense classes for evening out distributions that we used in our changing learning rates.
4. The algo which we have executed till Stochastic(SGD) is the learning rate(α) remains consistent which is constant and fixed
5. There is no momentum directly used, it is more in line with Rmsprop where the learning rate is constant and α is used for faster convergence just gradient squared is used.

3.6.1 Disadvantage

The problem with this is that over epochs the α squared will become very large and learning rate very small so the weights will not be updated and get stuck in the local not the global minima.

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

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

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

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

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

1. Here η means the learning rate which is variable.

$W(t)$ are the weights of the updated equations.

2. Large value of α squared is disadvantage which can be dealt in Rms-propagation.

MSE equation or the loss function is the same as before.

Thus rmsprop and adadelta are used wherever weighted averages are used instead of α so that containing the rise of weights which helps moving us in the direction of the global minima.

$$W_t = W_{t-1} - \eta \frac{\partial L}{\partial W_{t-1}}$$

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

3.6.2 Adagrad Algo

```
# Algorithm 4
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]
```

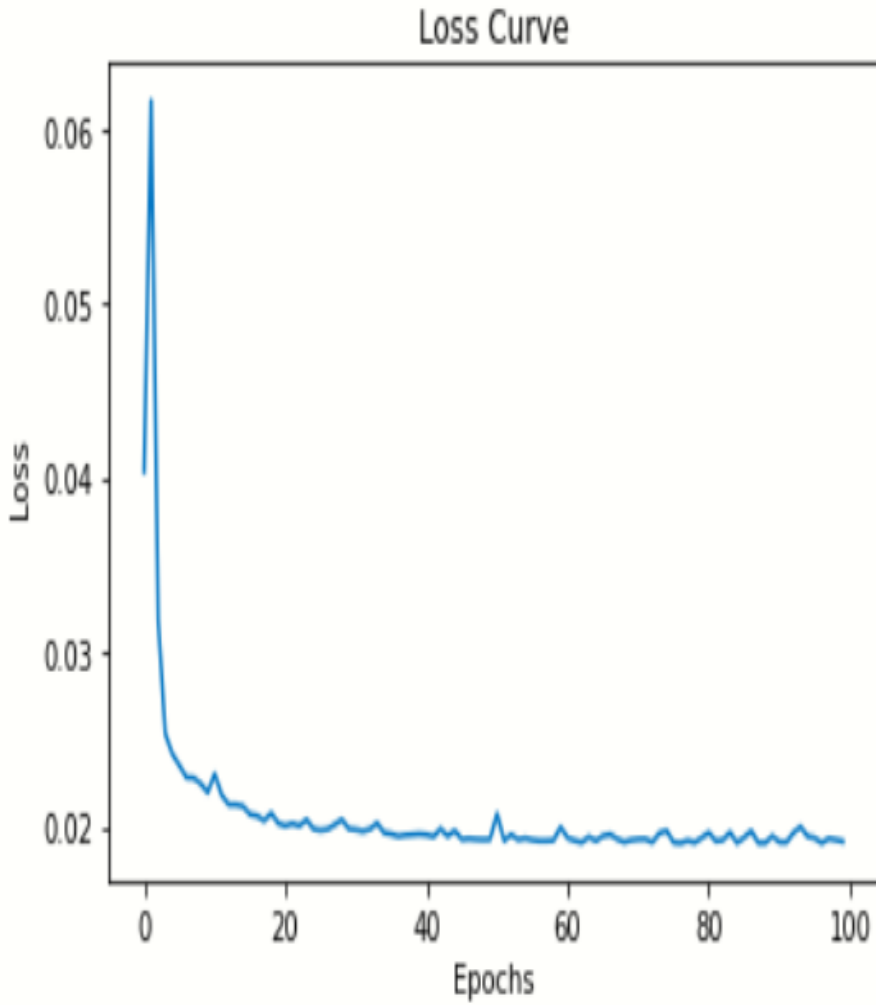


Figure 3.6.1: Loss Curve

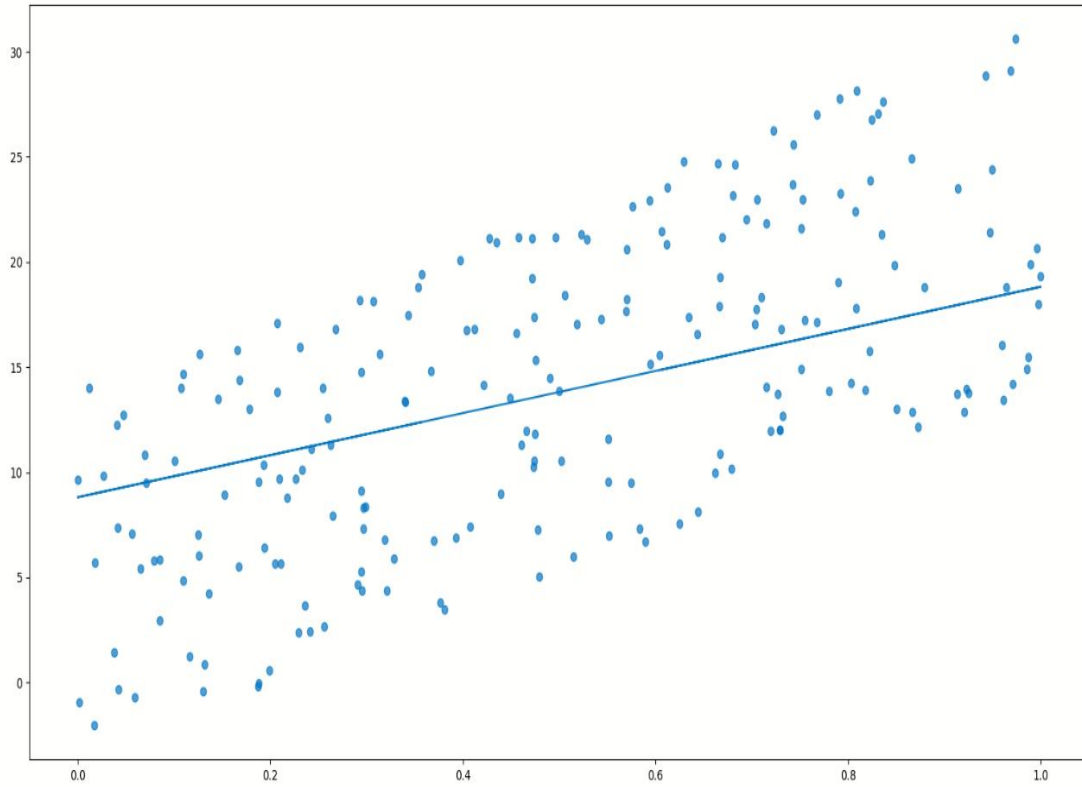
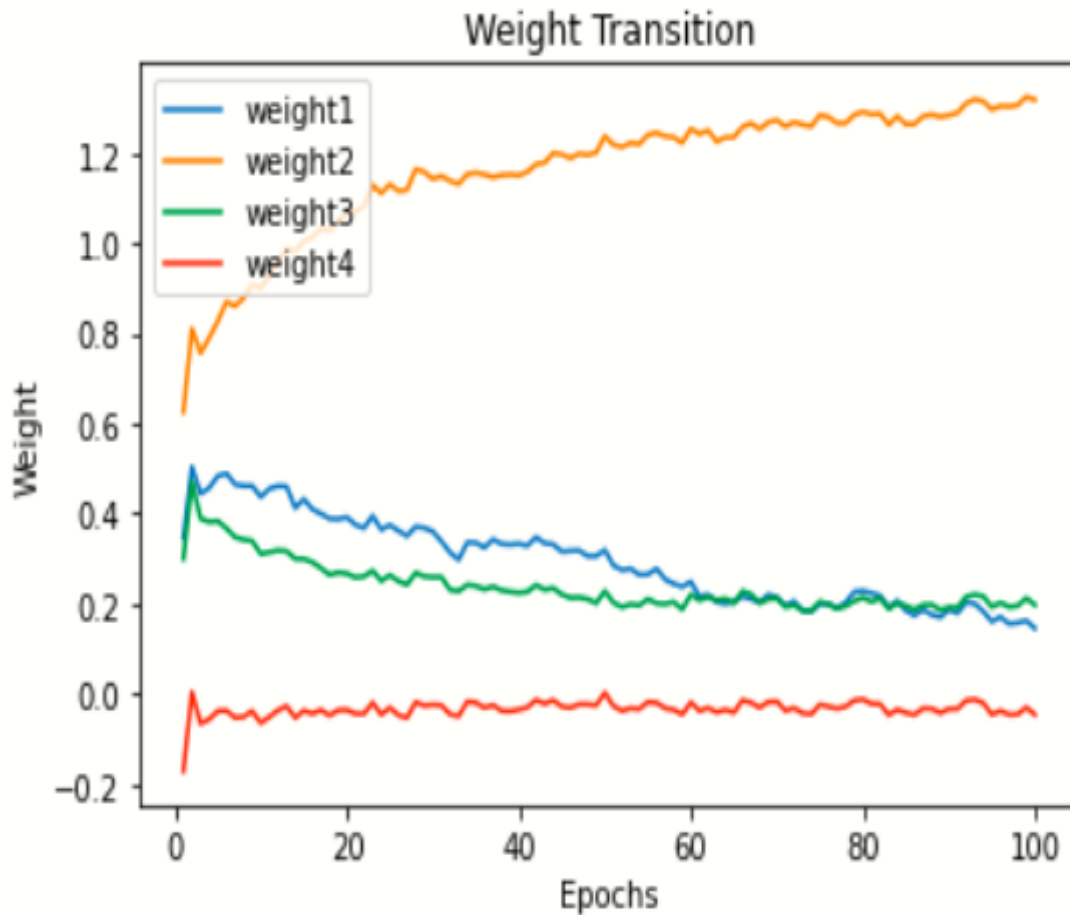


Figure 3.6.2: Best Fit Line

The graph above shows us best-fit-line model w.r.t the weights reaching over the iterations getting optimize.



Error for Adagrad = 0.05943616960408406

Figure 3.6.3: Weight Transition

3.7 Adam

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

3.7.1 Equation

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

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

1)The β_1 and β_2 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 β_1 and β_2 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 Mean Squared (MSE).

4)For the values of mini batch and the iterations –
M and V values are generated and given to our Mean Squared (MSE)

5)With every iteration the loss gets reduced because of the searching global minima values.

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

```

# Algorithm 5
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]

```

This graph below represents loss curve over the iterations getting smooth converges.

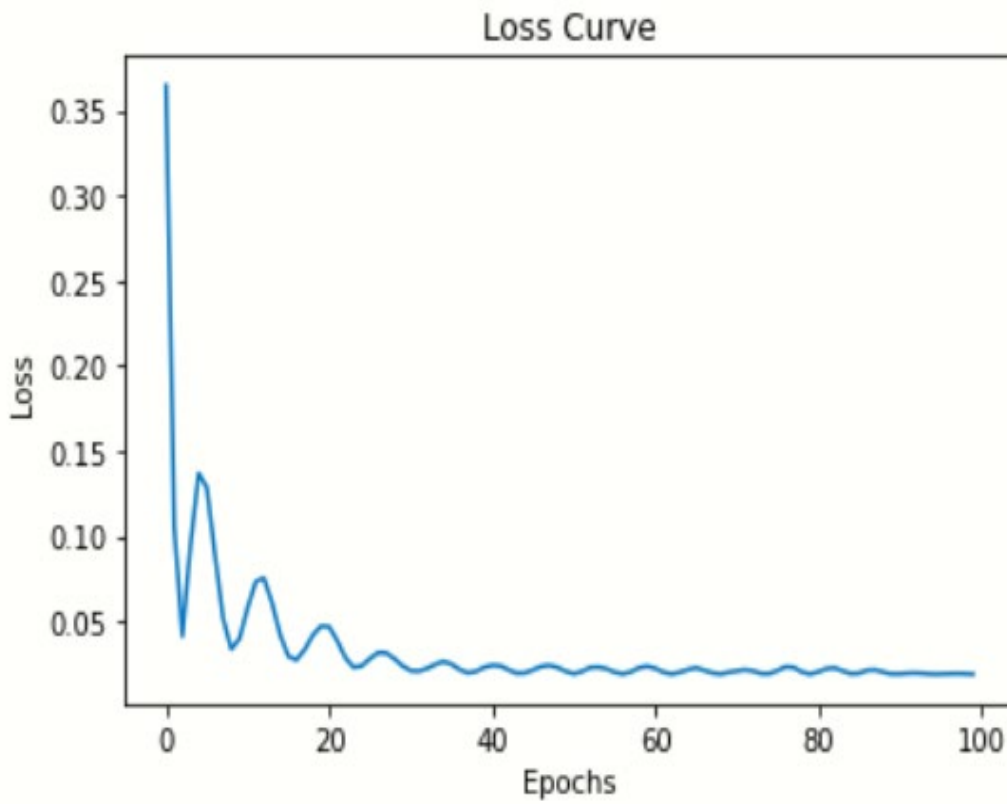


Figure 3.7.1: Loss Curve

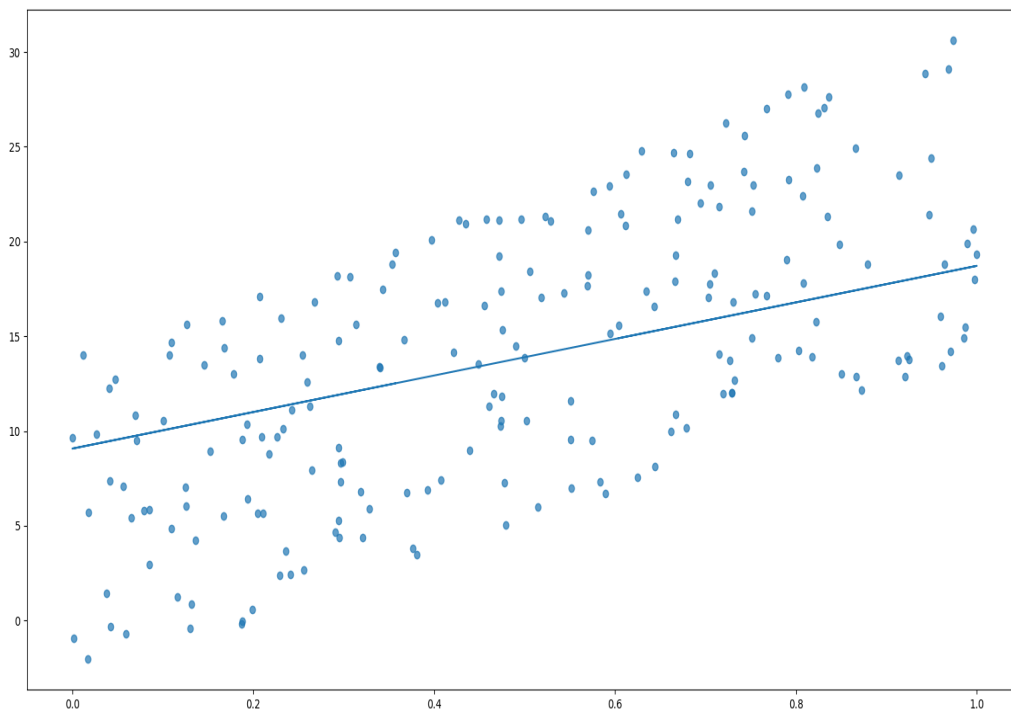
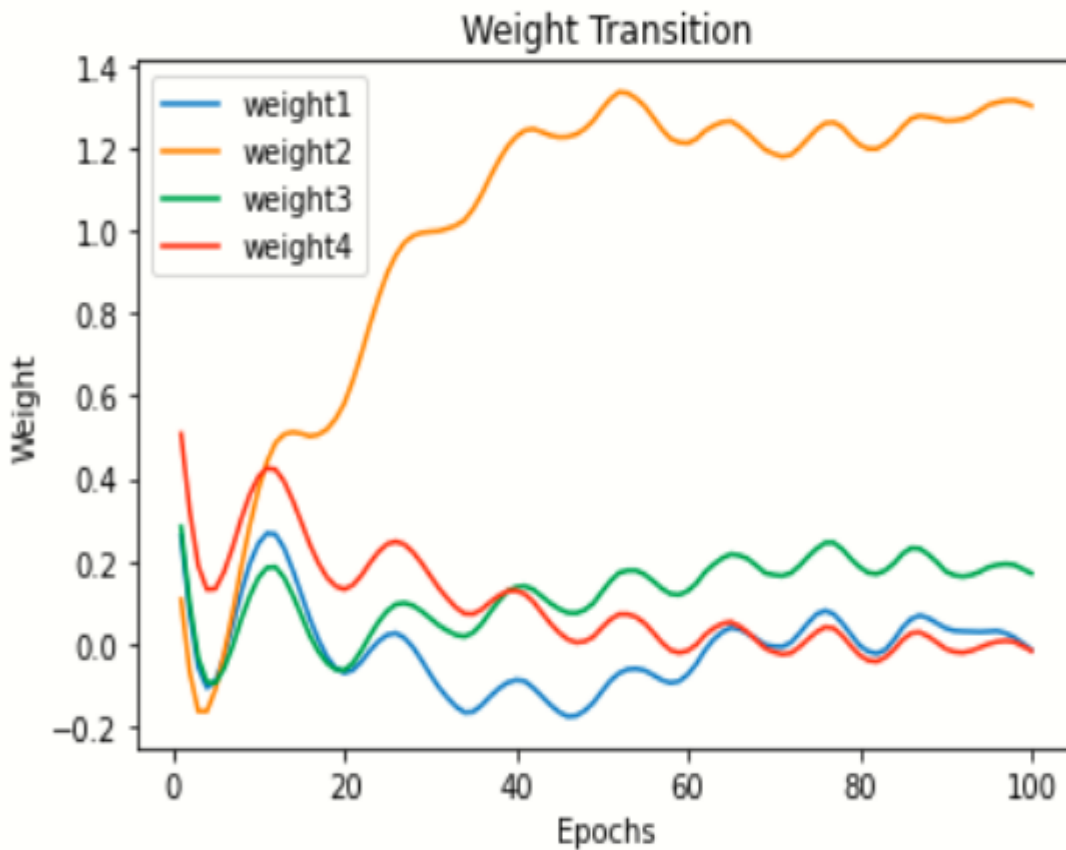


Figure 3.7.2: Best Fit Line

The graph above shows the best-fit-line models.

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

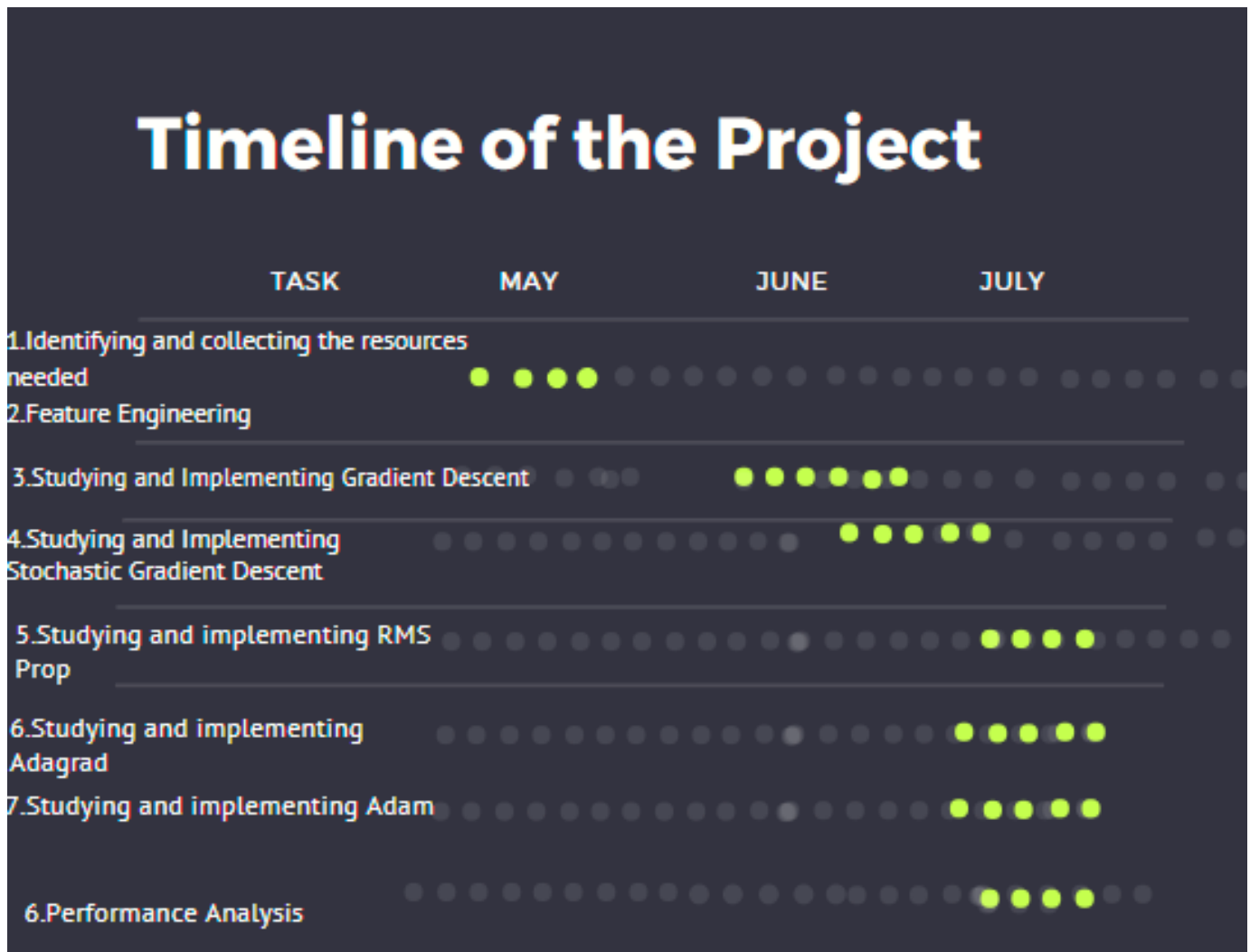


Error for Adam = 0.05915336762570325

Figure 3.7.3: Weight Transition

3.8 Timeline of Project

1. Gantt Chart



Chapter 4

Performance Analysis

4.1 Results

The results of this project are measured in terms of the form of error we get from these models.

The Error/losses of the models are-

(a)Error for GD =0.07310575088828815

(b)Error for Momentum = 0.09272319639976813

(c)Error for RMSP = 0.11781556388946751

(d)Error for Adagrad = 0.05943616960408406

(e)Error for Adam = 0.05915336762570325

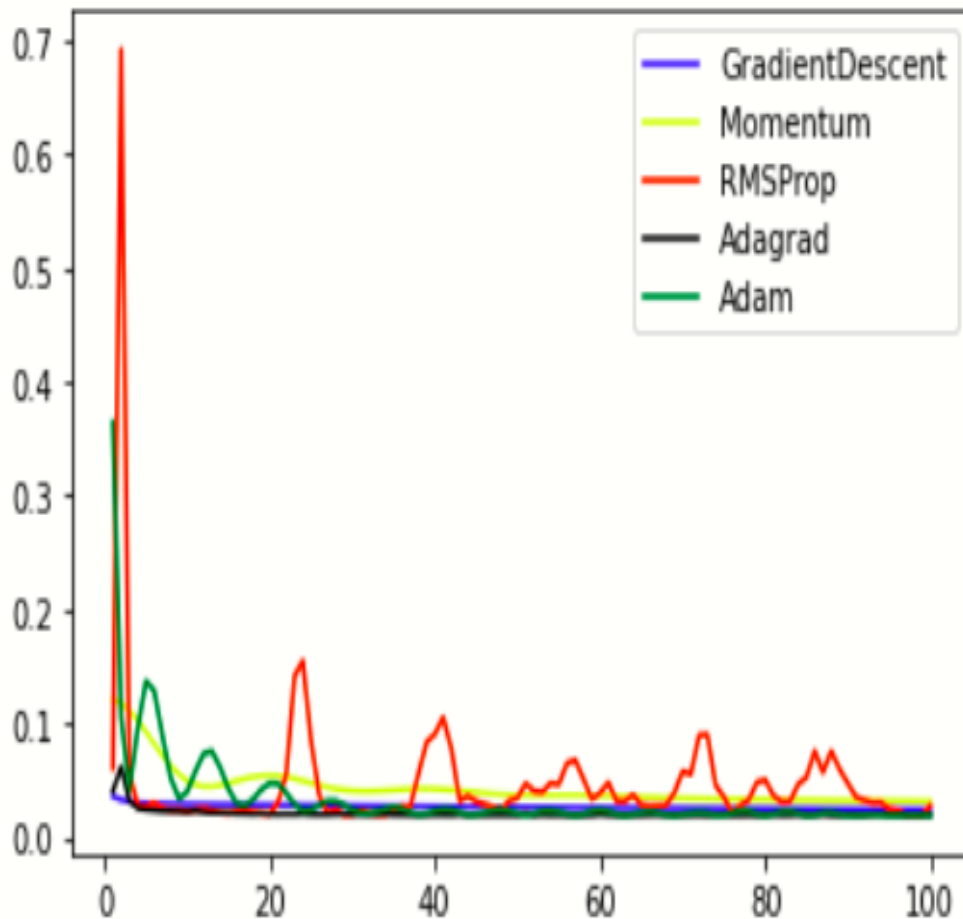


Figure 4.1: Comparison Graph

1. The best efficiency of the model we get is from adagrad and Adam because the models implemented use the adaptive-learning with that loss-curve is smoothed over iterations by usage momentum.

4.2 Advantages/Disadvantages of the Algos:

1. Advantages-Gradient Descent-

Gradient Descent (GD) weights converging is large that can be seen into the graphs in outcomes. The reason behind is that the values of gradients i.e the weights assigned are average of the whole dataset's points.

Because of the moment it is probable that it overshoots of its global minima .

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

3.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.

4.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.

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

5.Advantages-RMSPropogation-

RMS-Propogation converging rate is much more fast in comparison to Gradient Descent (GD) because it has its working basis on SGD . RMS-Propogation keeps the data of the previous gradient values and stores it . The curve related to loss is also much smoother.

6.Disadvantage of RMSProp:

It works better on convex loss function.

7.Advantages of Adagrad

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

8. Disadvantage of Adagrad

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

9. Advantages of Adam

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

10. Disadvantages of Adam

Takes more epochs and slower slightity

Chapter 5

Conclusion

5.1 Conclusion:

I completed the Project and have learned how to implement algorithms from scratch without the use of any lib function for the optimization algorithms to be implemented. The project which has been implemented has been successful and from this project the understanding of the algorithm's functionality and usage of these Optimization Algo is learnt. I learnt that Optimization of the model in the ML models are used to improve the algo and testing our model to perform in more productive reaching and converging minima

Gradient Descent (GD) algo shows a smooth converge line but the disadvantage is that it is computationally very expensive and resource heavy therefore it moves slowly to minima and can stop at local minim. Momentum on the other hand has solved both of the problems that we face from GD i.e converging faster,also it can come out local minimas. Advantage that we get from RProp is that it does not overshoot or shows fluctuation as Momentum (SGD) also achieves global minima easily. Adagrad(Adaptive) has much more larger values of α which can be dealt by the use of Adam(Algo).

The efficiency of the model is increased by Implementing various Optimization Algorithms by reducing the error.

5.2Future Scope:

1.We can implement algo such as Ada-Delta ,Ada-Grad ,R-Prop which can give us more better results.

The Algo implemented can be used in Supervised Learning where there is Linear Regression.

2.Linear regressions can be used in business to evaluate trends and make estimates.

Real Estate prices forecasts that follows linear relation.

References

1. Cui, Z., Zhang, J., Wang, Y., Cao, Y., Cai, X., Zhang, W., & Chen, J. (2019). A pigeon-inspired optimization algorithm for many-objective optimization problems. *Sci. China Inf. Sci.*, 62(7), 70212-1.
2. Pan, Q.K., Sang, H.Y., Duan, J.H. and Gao, L., 2014. An improved fruit fly optimization algorithm for continuous function optimization problems. *Knowledge-Based Systems*, 62, pp.69-83.
3. Rao, R. Venkata, and Vivek Patel. "An improved teaching-learning-based optimization algorithm for solving unconstrained optimization problems." *Scientia Iranica* 20, no. 3 (2013): 710-720.
4. Rao, R. V., Savsani, V. J., & Balic, J. (2012). Teaching-learning-based optimization algorithm for unconstrained and constrained real-parameter optimization problems. *Engineering Optimization*, 44(12), 1447-1462.
5. Luh, Guan-Chun, and Chun-Yi Lin. "Structural topology optimization using ant colony optimization algorithm." *Applied Soft Computing* 9, no. 4 (2009): 1343-1353.
6. Chen, Huiling, Yueting Xu, Mingjing Wang, and Xuehua Zhao. "A balanced whale optimization algorithm for constrained engineering design problems." *Applied Mathematical Modelling* 71 (2019): 45-59.
7. <https://en.wikipedia>.
8. <https://www.geeksforgeeks.org>
9. <https://machinelearningmastery.com/tour-of-optimization-algorithms>
10. <https://towardsdatascience.com/understanding-optimization-algorithms-in-machine-learning-edfdb4df766b>

APPENDICES

1.Data set upload.

```
import sklearn.datasets as ds
from google.colab import files

uploaded = files.upload()
```

No file chosen
Saving housing.csv to housing.csv

2.Libraries used .

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import warnings
from mpl_toolkits import mplot3d
```

3. Data set being trained.

```
def train(self, epochs, eta, mini_bacth=70):
    self.epochs = epochs
    self.mini_bacth = mini_bacth
    if self.optimizer == "GD":
        self.history = self.GradientDecesent(epochs, eta)
```

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: 24/07/21

Type of Document (Tick): PhD Thesis M.Tech Dissertation/ Report B.Tech Project Report Paper

Name: Achintye Sharma Department: Enrolment No 171331

Contact No. E-mail. 171331@juitsolan.in

Name of the Supervisor: Dr.Ekta

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters):

Implementing Optimization Algorithms to Increase Efficiency of Linear Regression

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.

Complete Thesis/Report Pages Detail:

- Total No. of Pages = 51
- Total No. of Preliminary pages = 8
- Total No. of pages accommodate bibliography/references = 51

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at9.....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(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 (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none">• All Preliminary Pages• Bibliography/Images/Quotes• 14 Words String		Word Counts	
Report Generated on		Submission ID	Character Counts	
			Total Pages Scanned	
			File Size	

Checked by
Name & Signature

Librarian

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com