VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**MACHINE LEARNING**

**FINAL PROJECT**

*Supervisor*: **Ph.D LÊ ANH CƯỜNG**

*Author*: **NGUYỄN PHƯƠNG TÀI – 521H0480**

Class **: 21H50201**

Class of  **: 25**

**HO CHI MINH CITY, 2023**

VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**MACHINE LEARNING**

**FINAL PROJECT**

*Supervisor*: **Ph.D LÊ ANH CƯỜNG**

*Author*: **NGUYỄN PHƯƠNG TÀI – 521H0480**

Class **: 21H50201**

Class of  **: 25**

**HO CHI MINH CITY, 2023**

ACKNOWLEDGEMENT

To complete this final project, we would like to express our gratitude to Ton Duc Thang University for providing the necessary facilities and to the teachers who have supported us throughout our studies at the university. Moreover, we love to give thanks to our parents who gave us chances to approach knowledge at TDTU and motivated, supported us a lot.

Especially, we would like to thank Ph.D Lê Anh Cường for teaching us with great dedication and detail, so that we have enough knowledge to use for this essay. Due to our limited experience and knowledge, we are sure that there are some mistakes in our work. We sincerely hope to receive feedbacks and constructive criticism from our teacher so that we can complete this essay more effectively.

We would like to express my heartfelt thanks and wish teacher good health.

**THE PROJECT IS COMPLETED**

**AT TON DUC THANG UNIVERSITY**

I hereby declare that this is the product of our project and is guided by Lê Anh Cường. The research content and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluation were collected by the author from different sources and clearly stated in the reference section.

In addition, the project also uses a number of comments, assessments as well as data from other authors and other organizations, all with citations and source notes.

If any fraud is discovered, I will take full responsibility for the content of my project. Ton Duc Thang University is not involved in copyright violations caused by me during the implementation process (if any).

*Ho Chi Minh City, 16th December 2033*

*Authors*

*(Signatures and full names)*

*Nguyễn Phương Tài*

SUPERVISOR’S CONFIRMATION AND EVALUATION

**Supervisor’s confirmation**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ho Chi Minh City, December 16, 2023

(signature and full name)

**Supervisor’s evaluation**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ho Chi Minh City, December 16, 2023

(signature and full name)

SUMMARY

In the realm of machine learning, the main goal of any models is to predict the the outcome based on the given datasets, and as for any prediction, we want the result to have as less error as possible. To combat this in machine learning, a sets of technique called optimization are used to reduce the degree of error, improving the accuracy in many models. In this report, we are going to go through some methods such as Gradient Descent, Momentum, Adagrad, RMSprop, Adaptive Moment Estimation,…

TABLE OF CONTENTS

ACKNOWLEDGEMENT 1

SUPERVISOR’S CONFIRMATION AND EVALUATION 3

SUMMARY 4

TABLE OF CONTENTS 1

CHAPTER 1 – INTRODUCTION 5

1.1 Local and global maxima, minima 5

1.2 Derivative 6

CHAPTER 2 – OPTIMIZATION ALGORITHMS 8

2.1 Gradient Descent 8

2.1.1 Gradient Descent for single – variable problem 8

2.1.2 Gradient Descent for multi – variable problem 10

2.2 Stochastic Gradient Descent 14

2.3 Mini – Batch Gradient Descent 18

2.4 Gradient Descent with Momentum 20

2.5 Adagrad (Adaptive Gradient Algorithm) 24

2.6 RMSProp 26

2.7 ADAM (Adaptive Moment Estimation) 28

2.8 Comparision 32

CHAPTER 3 – CONTINUAL LEARNING AND TEST PRODUCTION 34

3.1 Continual Learning 34

3.1.1 Importance 34

3.1.2 Methods 35

3.1.3 Problems 35

3.2 Test Production 36

3.2.1 Importance 36

3.2.2 Methods 37

REFERENCES 38

**TABLE OF PICTURES, CHARTS AND GRAPHS**

Figure 1.1: Global minima of a function 4

Figure 2.1: Gradient Descent for single – variable problem 8

Figure 2.2: Testing Gradient Descent 9

Figure 2.3: Obtained Result 9

Figure 2.3: Hypothesis function 11

Figure 2.4: Cost function 11

Figure 2.5: Calculate gradient when J = 0,1,2 11

Figure 2.6: Initialization 12

Figure 2.7: Calculation Process 12

Figure 2.8: Result Printing 12

Figure 2.9: Initialization Process 14

Figure 2.10: Update Iteration 15

Figure 2.11: Plotting Graph 16

Figure 2.12: Plotting Result 16

Figure 2.13: Initialization 18

Figure 2.14: Plotting Result 19

Figure 2.15: Gradient Descent Visualization 20

Figure 2.16: Cost Function 21

Figure 2.17: Momentum Gradient Descent 21

Figure 2.18: Initialization 22

Figure 2.19: Momentum Gradient Descent Iteration 1 22

Figure 2.20: Final Iteration Momentum Gradient Descent 23

Figure 2.21: Cost Function 24

Figure 2.22: Adagrad Iteration 24

Figure 2.23: Run Adagrad 25

Figure 2.24: RMSProp Iteration 26

Figure 2.25: Run RMSProp 26

Figure 2.26: RMSProp Result 27

Figure 2.27: Cost Function 28

Figure 2.28: Initialization 28

Figure 2.29: Adam Iteration 29

Figure 2.30: Plotting Adam 30

Figure 2.31: Plotting Result 31

CHAPTER 1 – INTRODUCTION

To understand the concepts of the optimization algorithms, we first need to understand the mathematical theory behind some of the algorithms.

* 1. Local and global maxima, minima

In math, given a function and a range between a and b, denoted as , a local maxima means a point where the function achieve the highest value than any other points in that range, the same can be applied to a local minima but for the smallest function value in that range.

A global maxima or a global minima is a special case of a local maxima and local minima, instead of denoting the point where the function achieve its highest or smallest value in a specific range, it denotes the point where function reaches its highest or smallest value in the entire range range of real values. The two global points can be denoted as follows:

Let be a real – valued function

To better visualize the concepts, we need to have a look at a graph.

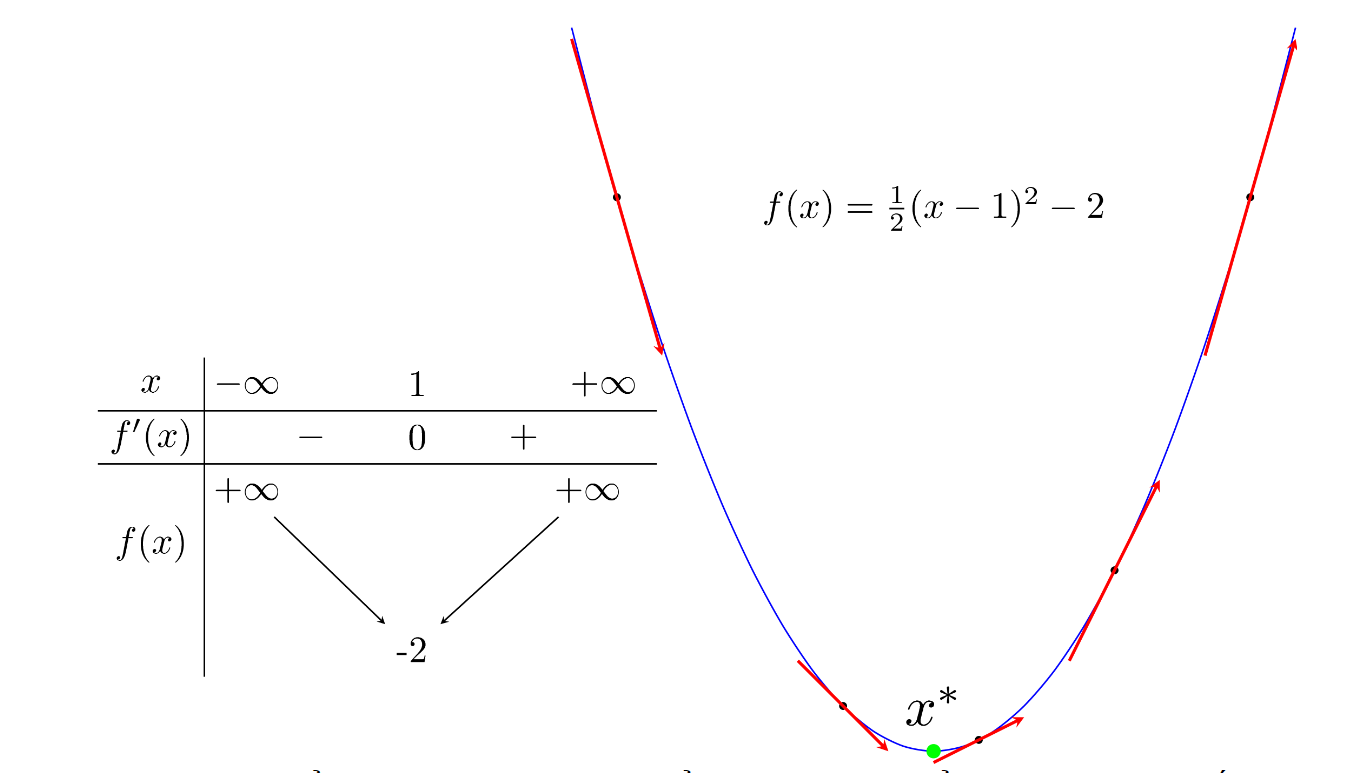


Figure 1.1: Global minima of a function

(Source: machinelearningcoban.com)

In the figure above, we see that the function achieve the smallest value where . There are some concepts related to local, global maxima and minima we need to know.

* The point where the function achieve its smallest or highest value based on a range or the entire real – valued numbers always has the derivative of its function subtituting itself equal to 0.
* In a local minima’s range, the derivatives of the value on the left of the local minima point will always be smaller or equal to 0, while the derivatives of the value on the right of the local minima point will always be higher or equal to 0.
* In a local maxima’s range, the derivatives of the value on the left of the local minima point will always be higher or equal to 0, while the derivatives of the value on the right of the local minima point will always be smaller or equal to 0.
  1. Derivative

The derivative of a function is the rate of value changes of the function as the input value changes. For example, when we have a function and an input , the derivative of this is defined as the limit of the difference quotient as the increment reaches 0. This can be denoted as follows:

There some rules to calculate derivative of a function, here are some of the most popular rules:

* Given then
* Given and then
* Given , then
* Chain rule:

For example, we have some function such as follows:

* , then
* , then

CHAPTER 2 – OPTIMIZATION ALGORITHMS

2.1 Gradient Descent

In machine learning, we usually need to find the the smallest or biggest value based on the function. But the problem is that finding a global minima is very complicated and cumbersome, therefore, we usually find the local minima, and to some degree, it is the value that we need to solve the problem.

Local minimas are the points that when subtitute into the derivative function will give us 0. If we just happen to find someways that can find all of the local minimas, we just need to subtitute each of those values into the function and then find the point where the function achieve the smallest value. But in most cases, this is practically impossible, it can be something related to the complexity of the function, the number of features or there could be too many data points.

One of the most popular approach is to starting from a point that we may consider as a solution to the problem, then use a mathematical operation that gradually reaching the optimal solution.

The main idea of gradient descent is adjusting the variables through multiple iteration to reduce the cost function. It takes repeated steps in the opposite direction of the gradient of the function at the current point. The size of each step is determined by a parameter called learning rate, denoted as .

2.1.1 Gradient Descent for single – variable problem

Let be denoted as local minima and is the point we achieve after the iteration. We need a formula that can bring as close as possible to the value .

We need to make an observe as follows:

* If the deriviative of the function at the point when subtituting has a negative value, which means that is on the left of based on the conclusion we made in section 1.1, then we need to move to the right. The reverse statement is true when is on the right of . In short, we need to move in reverse to the derivative value.
* To move in reverse to the derivative value, we have the following function . Of which, is the value that is in reverse to the derivative function value.
* The further away is from to the left, the smaller becomes, and vice – versa. Therefore, the value must scale with and has to be in reverse, from that, we have the following function:

The value is denoted as the learning rate, it determines the speed of convergence into the local minima, the smaller the value, the slower the convergence process become, but if the value is too large, it can lead to overshooting, an instance where the will by pass into the otherside of the gradient and make it harder for to converge.

To demonstrate this, we have a small Python program.



Figure 2.1: Gradient Descent for single – variable problem

In the above Python program, our chosen function is ,cost is the function we use to calculate when subtituting x and derivative is, by the name suggest, a function where we calculate the derivative value when subtituting x into . The gradientDescent function is used to find the optimal , we set the number of iteration as 1 million, the x\_new value is , we only need to find that when subtituting into the derivative function, any value that is smaller than 0.001 is acceptable and is a optimal solution.

To test this function, we have the following code

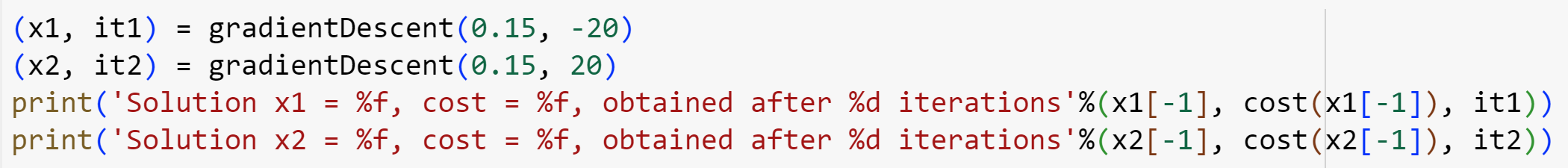


Figure 2.2: Testing Gradient Descent

In the above code, x1 and x2 serve as the of each test, while it1 and it2 serve as the number of iteration to achieve x1 and x2. When the iteration process complete, we print out the value of each test and its cost after a number of iteration.

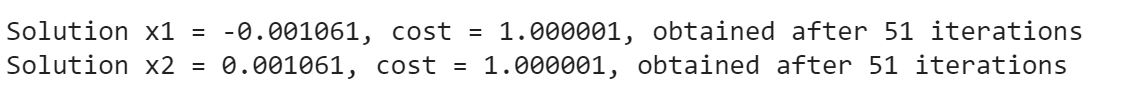


Figure 2.3: Obtained Result

2.1.2 Gradient Descent for multi – variable problem

When we have a dataset consists of n features (attributes) and m training examples, the data can be represented as number of sets consisting of a vector of features. Where is a vector of 𝑛 features for the 𝑖 − 𝑡ℎ example and is the corresponding target as follows.

The hypothesis function for the multi-variables Linear Regression problem can be written as followed:

Where is a vector of parameters or coefficients of the model,

The cost function for the multi-variables Linear Regression can be written as followed:

For each iteration, we need to update the weight parameters (or 𝜃) for each of the feature inside the training example, we can achieve this result by subtracting the current weight parameter by the learning rate multiply with the loss function.

For each parameter , we have:

With 𝛼 as the learning rate. For can also further denote the partial derivative of the cost function with respect to 𝜃𝑗 as followed:

In conclusion, the algorithm needs to be performed by the following steps:

1. Starting of by assigning random values to all of the weight parameters.
2. Update the parameters by subtracting the weight parameter by the partial derivative of the loss function as mentioned above.
3. Repeat step 2 until all of the partial derivative is equal to 0 or close to a very small value.

To demonstrate the above algorithm, we have the following Python program:

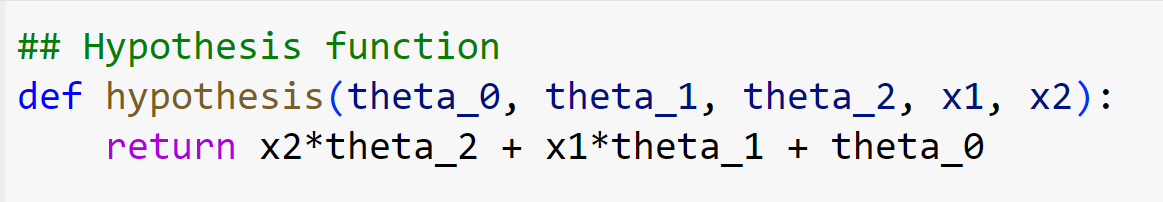


Figure 2.3: Hypothesis function

First, we have a hypothesis function, here, I define the program to have 3 theta values for demonstration purposes.

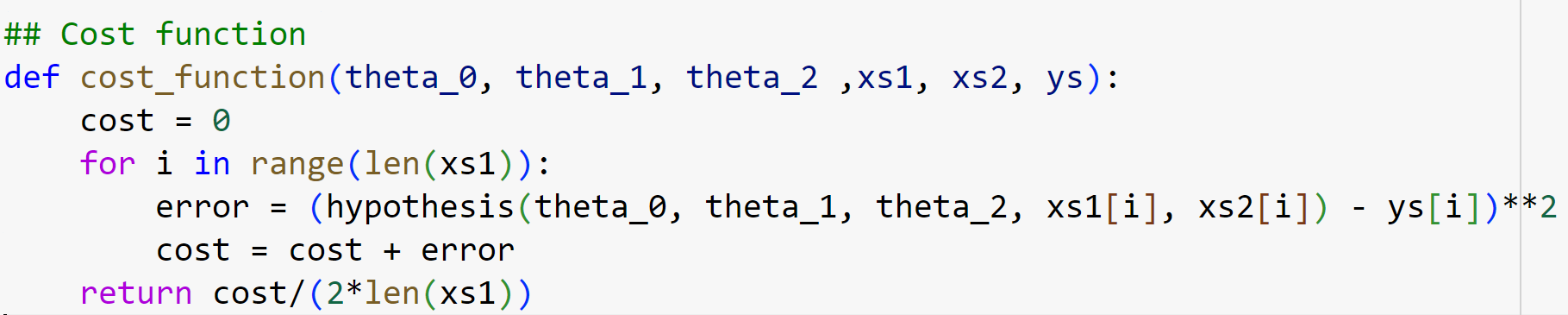


Figure 2.4: Cost function

Cost function based on the 0, 1 and 2.

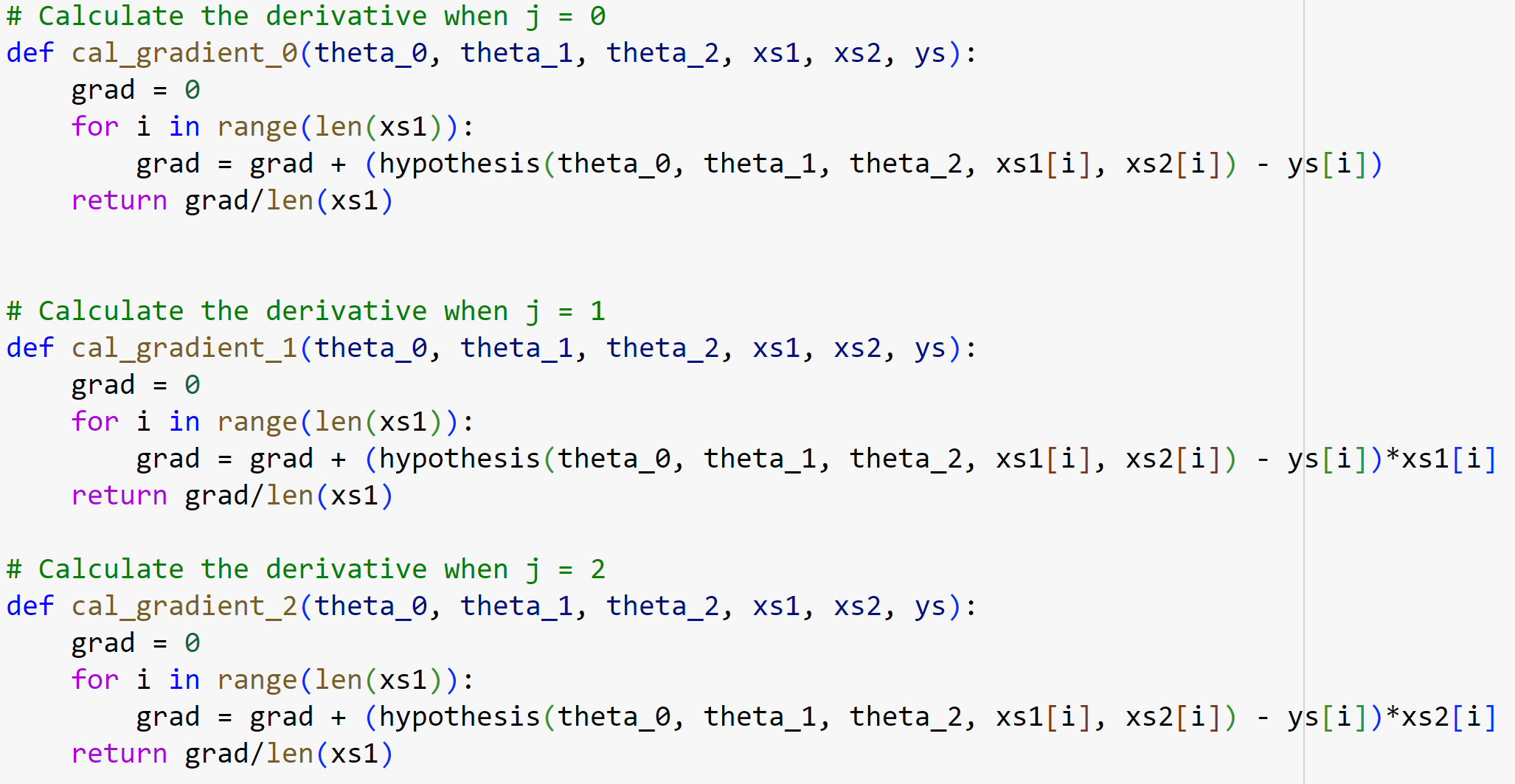


Figure 2.5: Calculate gradient when J = 0,1,2

Here, we have 3 functions used to calculate derivative when j is equal 0, 1 and 2 respectively.

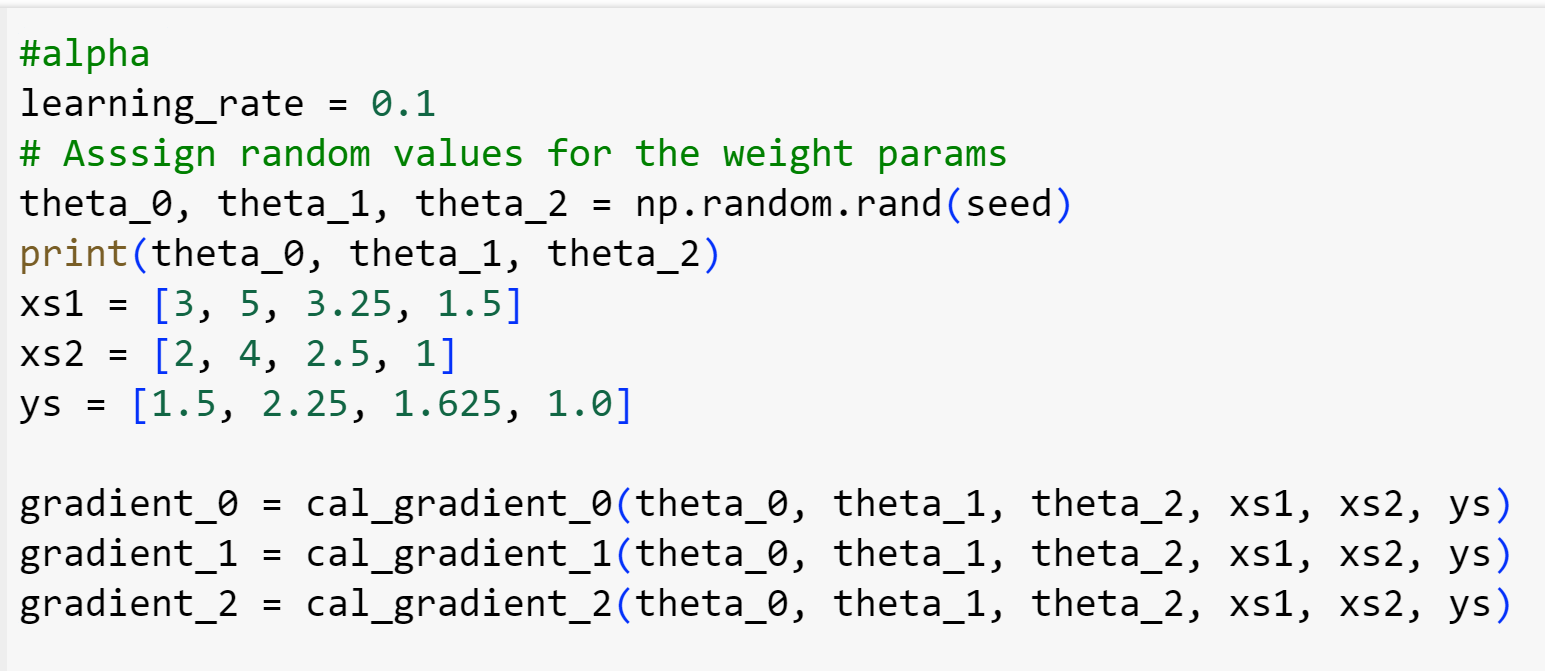


Figure 2.6: Initialization

This is our initialization step, the learning rate we have set at 0.1, the three theta values are set randomly. Xs1, xs2 are the feature values and ys is the intended result value. The program will calculate gradient\_0, gradient\_1 and gradient\_3 to reach an optimal solution through a number of cycles.

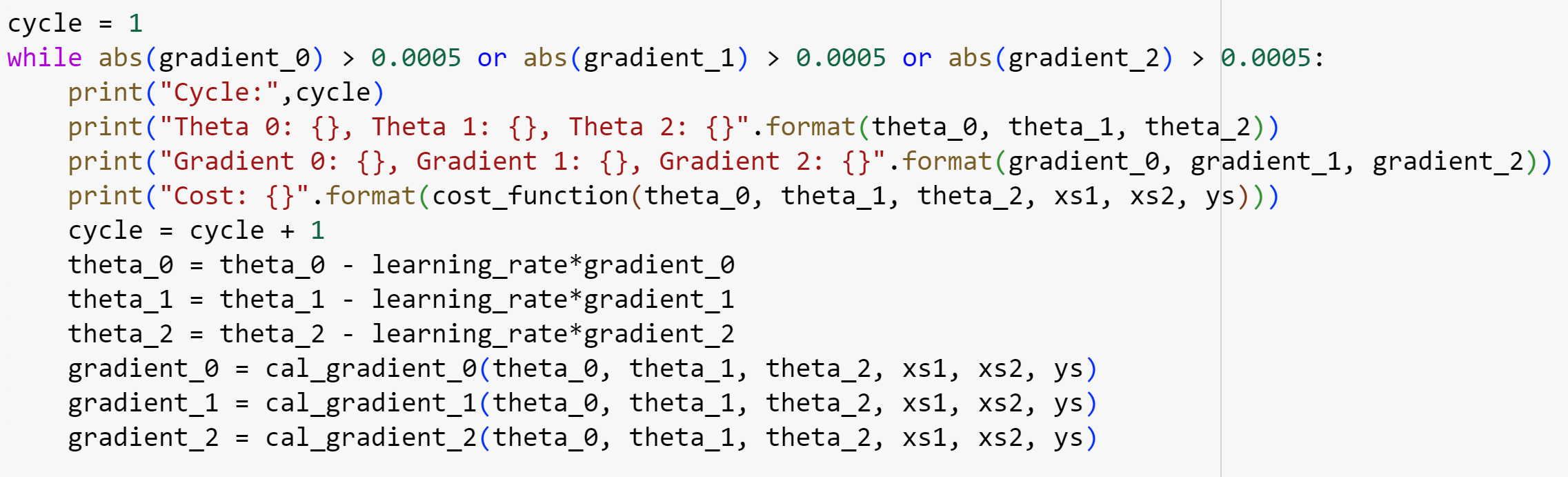


Figure 2.7: Calculation Process

In each iteration, the theta value is updated according to the above formula. After finishing the calculation process, the result will be printed as follows

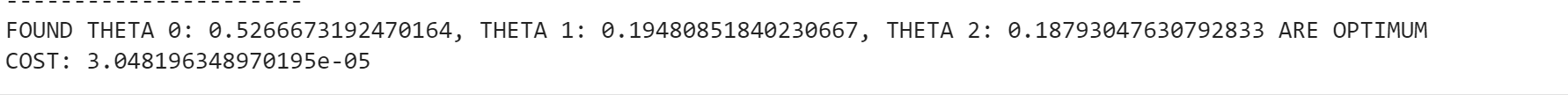


Figure 2.8: Result Printing

2.2 Stochastic Gradient Descent

In this algorithm, the main difference when comparing to the regular Gradient Descent algorithm, is that it is instead of updating the theta value based on all data points, it updates the theta value based on a single data point. This approach introduces noise into the parameters update, which can help to bypass local minima.

Each time the theta value is updated iteratively on every single data point of a data set is called an epoch. In each epoch, the theta value is updated time, as is the number of data point in a data set. At first, this seems slow because instead of performing the updating process once on every epoch, since we have to update times on each epoch. But this process is very effective since Stochastic Gradient Descent only requires a small number of epochs, usually smaller than 20.

The mathematical expression of the updating process of Stocastic Gradient Descent is defined as follows:

In which, is the derivative of the cost function at a pair of data point . The cost function can be defined as follows:

The derivative of the above function is:

is to denote transpotation.

The algorithm is defined as follows:

1. Shuffle the dataset to make sure that the randomness is perserved in the algorithm
2. Iterate through the sets in the shuffled order.
3. Calculate the gradient based of the cost function
4. Update the theta value scaled by the learning rate.
5. Repeat the above steps until we got an optimum solution

To demonstrate the algorithm, we have a Python program which simulates the Stochastic Gradient Descent process

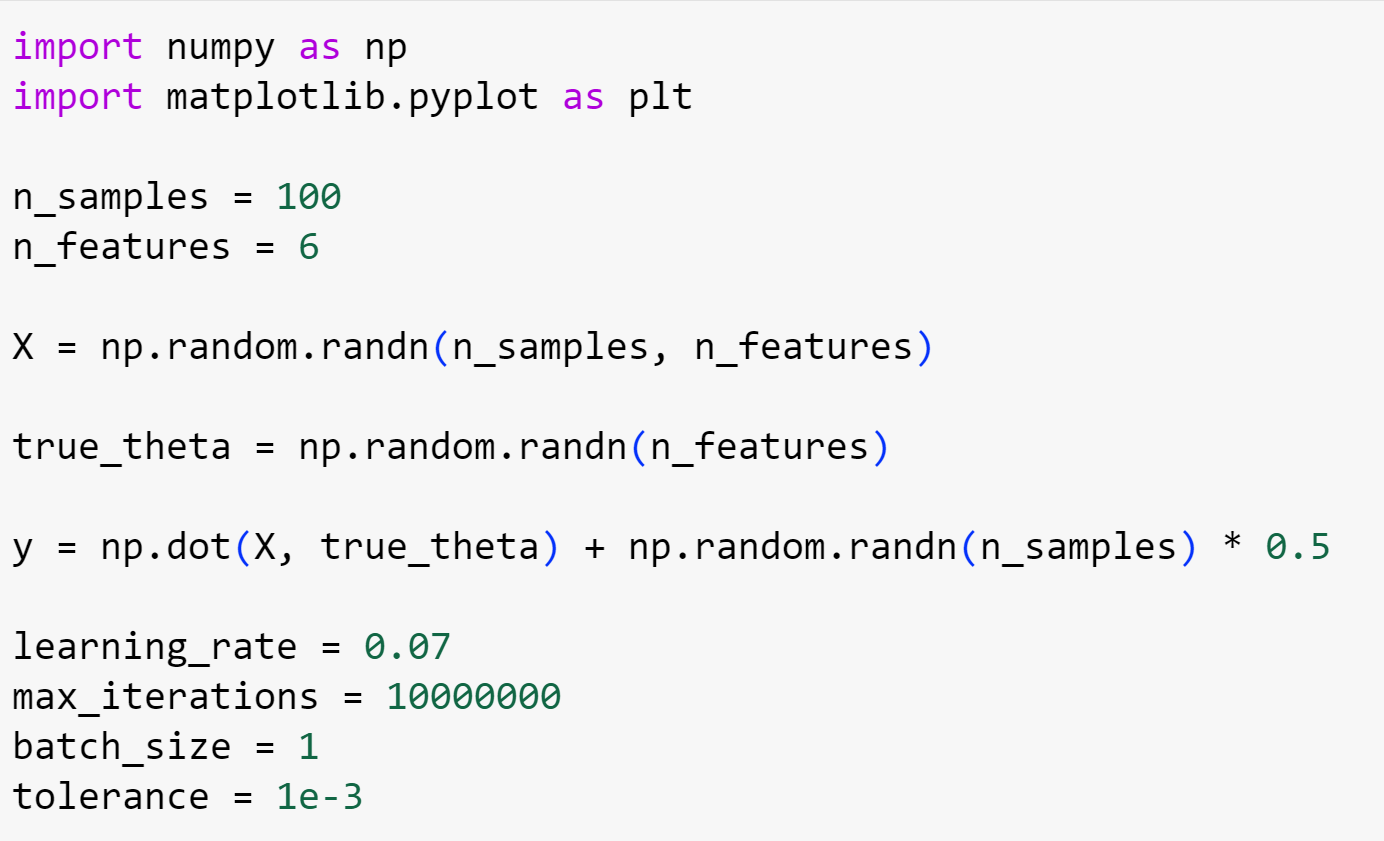


Figure 2.9: Initialization Process

In the initialization process, we are creating a sample dataset X that has 100 rows of data and each row has 6 features. The true\_theta is an array of coefficients that defines the relationship between X and y. In this context, the target array y is calculate as follows:

With noise as a value of random fluctuations to make the data more relatable. The learning rate we have set as 0.07, the maximum iteration is 1 million iterations, the tolerance value is 0.001, which means that if the gradient value is lower than 0.001 then it is acceptable. Finally, the batch\_size value means the number of data point used to update the theta values.

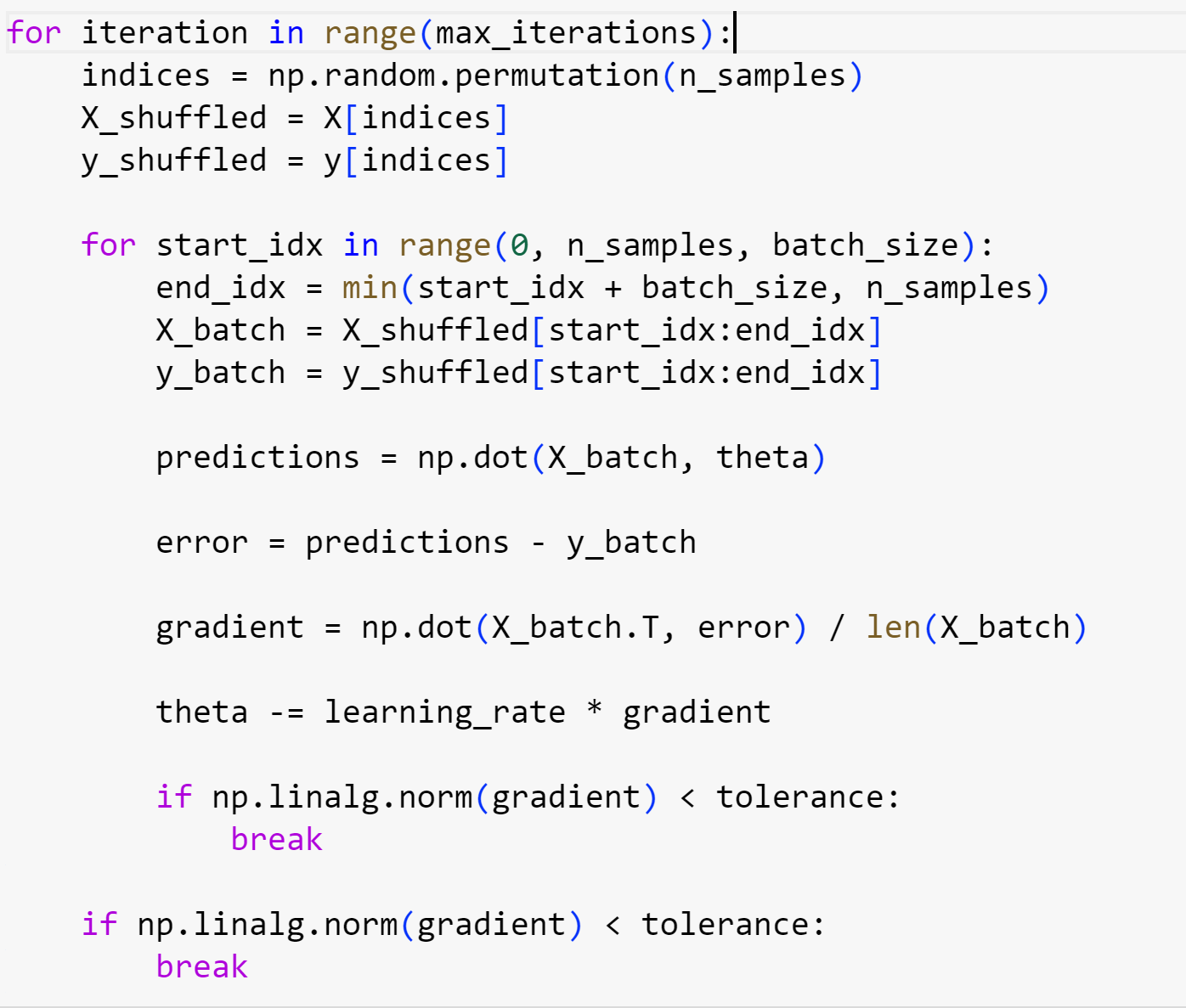


Figure 2.10: Update Iteration

In each of the iteration process, we first shuffle the datasets, then select a number of samples inside a batch, as for Stochastic Gradient Descent, a batch only contains 1 sample. Therefore, in each iteration, we take the sample one by one and update the theta accordingly. If the gradient value is smaller than the tolerance threshold, we stop the program and return the theta value.

To better visualize the convergence, we also plot a graph using the matplotlib library as follows.



Figure 2.11: Plotting Graph

The above code will plot a graph consisting of the blue points of prediction versus actual data, and the red points of the newly predicted points of the new data. The X – axis show the actual values and the Y – axis show the predicted value.



Figure 2.12: Plotting Result

In the graph, the blue points show how the model has learn from the training data while the red points show how the model behave on new data. All of the colored points follow a similar trends suggest that the model has generalize to the new data.

2.3 Mini – Batch Gradient Descent

Mini – Batch Gradient Descent is like the middle – man between Stochastic Gradient Descent and the regular Gradient Descent. Instead of taking all of the data set like ordinary Gradient Descent or each data set like the Stochastic Gradient Descent, this variation only takes a hanful of features of each data point. Each of the selected mini – batch has the number of data points ranging between 1 to as is the number of data points. In each of the updating process, the algorithm takes out a mini – batch and update the theta value based on the derivative cost function of that mini – batch.

The formula for the following updating process can be noted as follows:

In which, is the data points from to .

We need to make sure that, after every updating process, the data points must be shuffled to preserve the randomness of the algorithm.

The algorithm is defined as follows:

1. Shuffle the dataset to make sure that the randomness is perserved in the algorithm
2. Iterate through the subsets in the shuffled order for each epoch
3. Calculate the gradient based of the cost function
4. Update the theta value scaled by the learning rate
5. Repeat the above steps until we got an optimum solution

As we can see, the algorithm is almost the same as the Stochastic Gradient Descent Algorithm, the only real difference is in the way which we select the data to compute.

To demonstrate the algorithm, we will just need to adjust the batch\_size code from the Stochastic Gradient Descent algorithm to your desire number of batch – size. Please notice that the batch\_size must be smaller the number of samples. If the samples have 100 rows then the possible batch\_size could be ranged from 1 to 100.

For demonstration purposes, I have set the number of batch\_size to 20, because 20 is a factor of 100, we usually choose the batch\_size to be the factor of the number of samples to ensure all data must be used to compute.

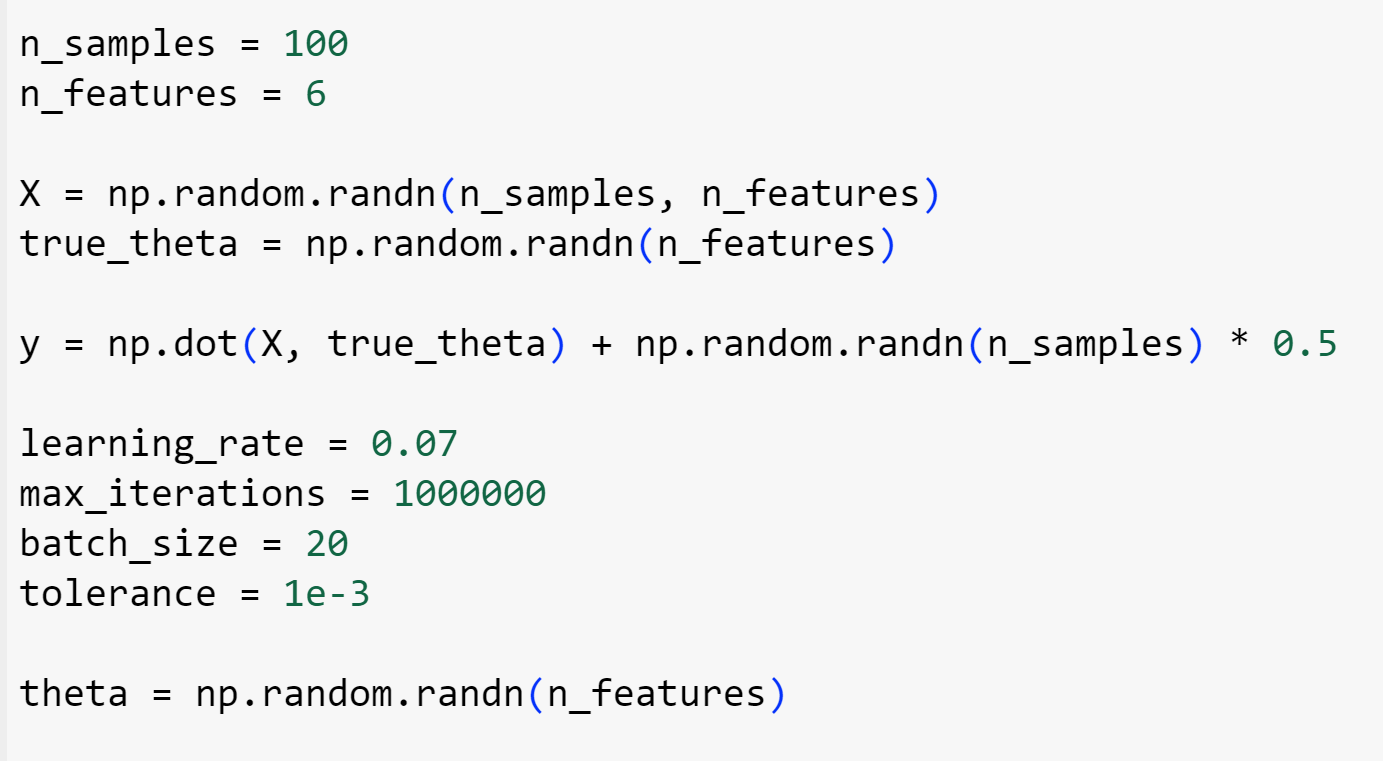


Figure 2.13: Initialization

Here, we have set the batch\_size to 20. The rest of the code is the same as the Stochastic Gradient Descent algorithm.

After running the algorithm, we achieve the following plot.

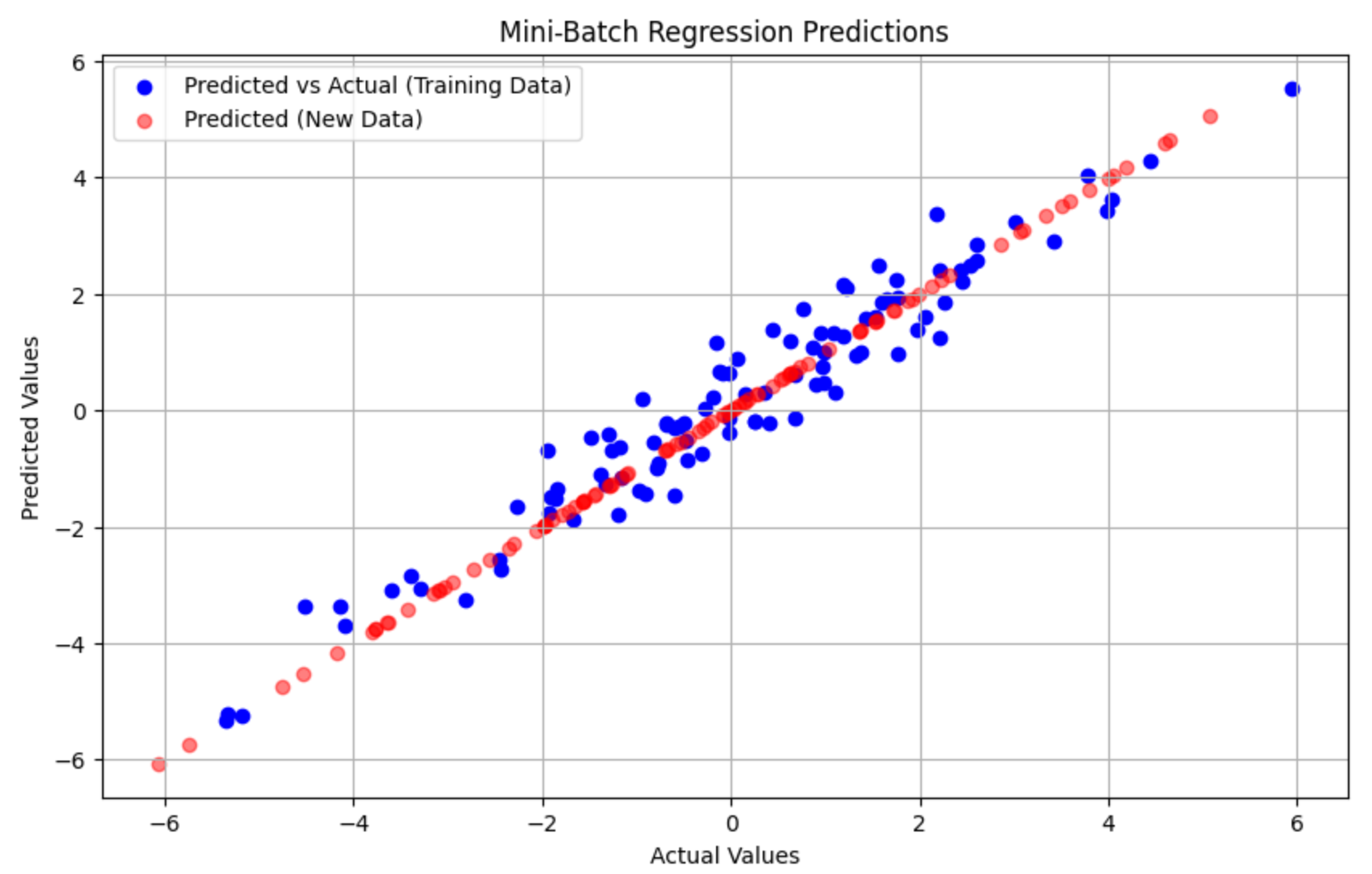


Figure 2.14: Plotting Result

The graph that we achieve through Mini – batch Gradient Descent has similar pattern as the Stochastic Gradient Descent, showing a close cohesion between the seen and unseen data.

2.4 Gradient Descent with Momentum

In the regular Gradient Descent, the most common problem associate with this algorithm is that it usually stuck in the local minima zone, to visualize this, I have the following graph to demonstrate this problem in terms of physics. The visualization will demonstrate all of the possible outcome of the Gradient Descent algorithm

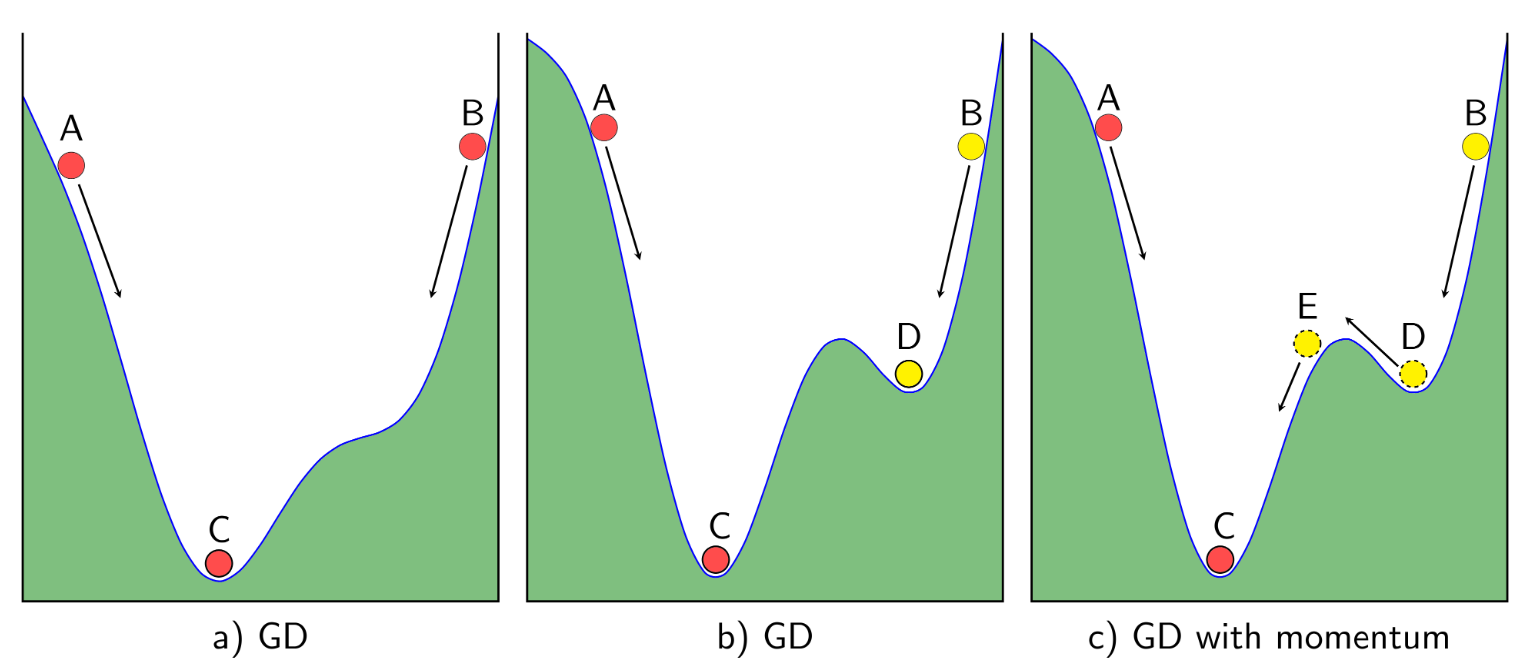


Figure 2.15: Gradient Descent Visualization

(Source: machinelearningcoban.com)

In the visualization, the first subfigure is the case where the algorithm is most sufficient, either dropping from A or B, the ball will always reaches C. In the subfigure, the same goes when you drop from A, but if you start from the B position, you will only reaches D, a local minima point. If only we have some solution to make the ball continues to roll to the E position and then roll the C position.

Momentum is the solution to this problem, we will give the ball a momentum just like when we drop a marble into a downward slope, the marble will gradually gain momentum as it is moving and lose the momentum when encounter an upward slope.

We will need to assign a variable to represent the velocity of the marble, the newly generated formula to update the theta value is as follows.

Of which, the value represent the velocity at the point in time. To calculate the value, we need to formulate the calculation to base on the slope, which is the deriviative of the cost function, and the previous velocity. Note that we will asume the initial velocity to be 0, . We will come to a conclusion of a formula like this.

Of which, the is usually the value of 0.9, the previous velocity . The rest of the formula is the same.

To demonstrate the algorithm, we have setup a Python program with a complex slope.

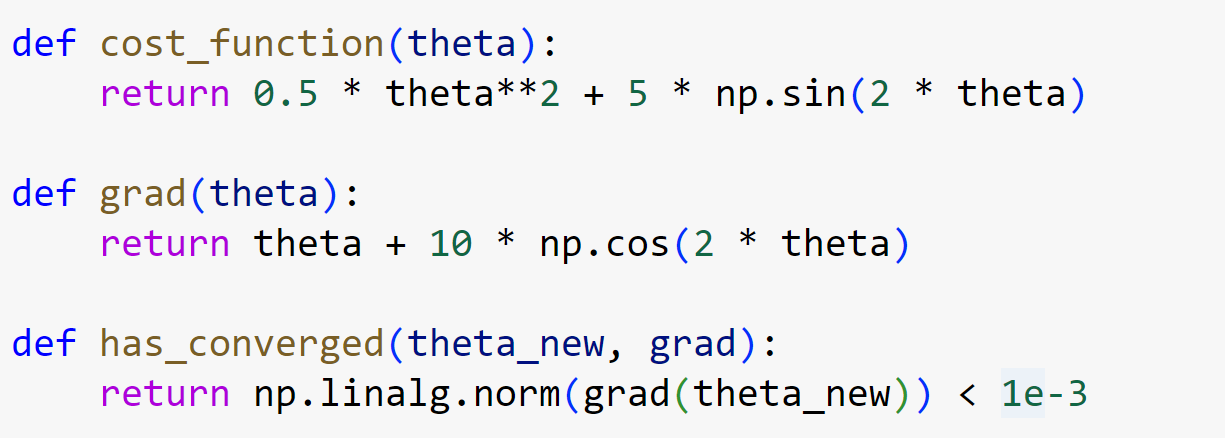


Figure 2.16: Cost Function

We have define the cost function to have a simulated function of . From the cost function, we have the gradient function of . We also have a has\_converged function to determine if the theta value is below the acceptable threshold.

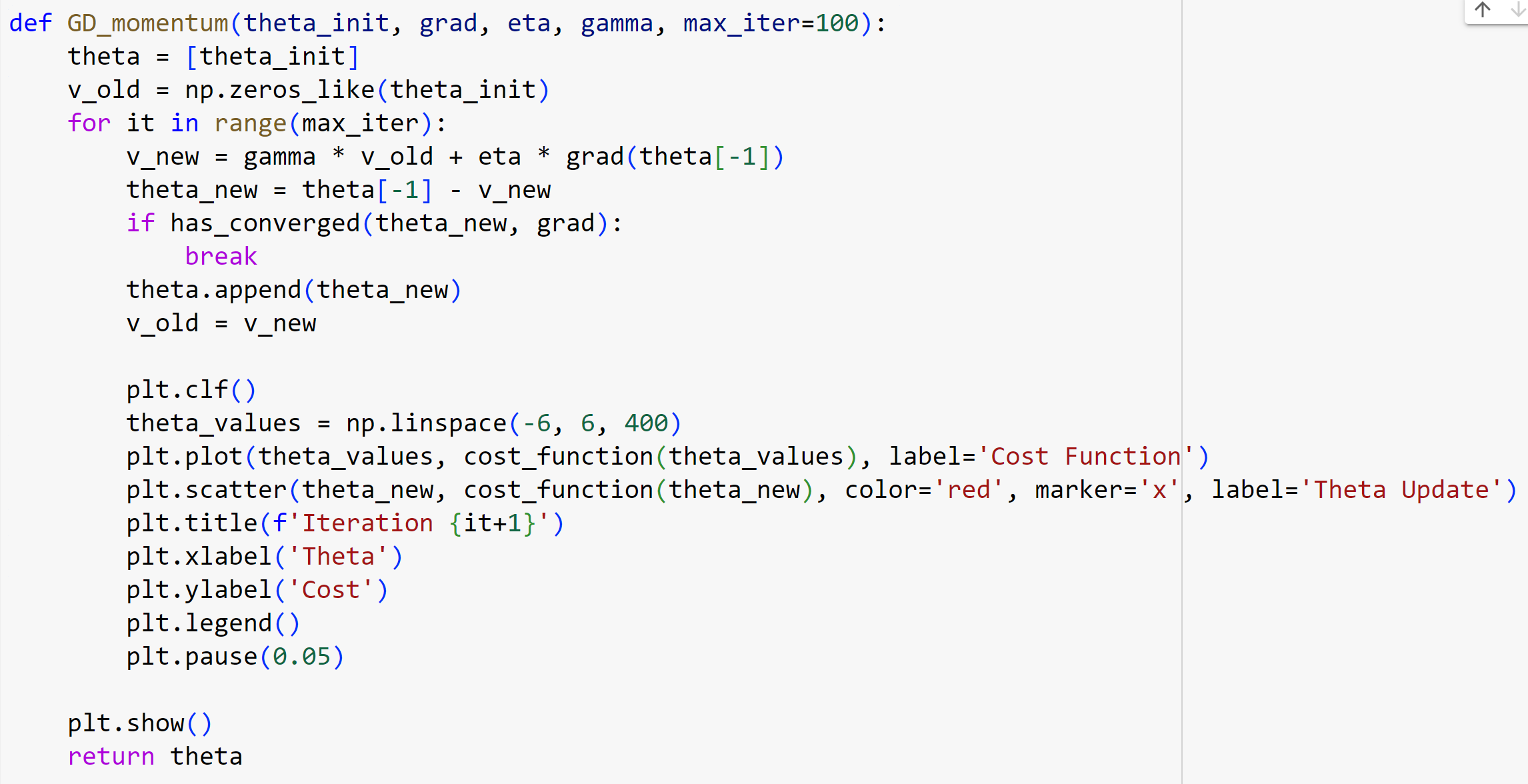


Figure 2.17: Momentum Gradient Descent

In the main Gradient Descent with momentum function, we are going to update the new theta value in each iteration, then check if the value converges. After each step, we also re – assign the velocity value and plot a graph to see where the point is on the graph.

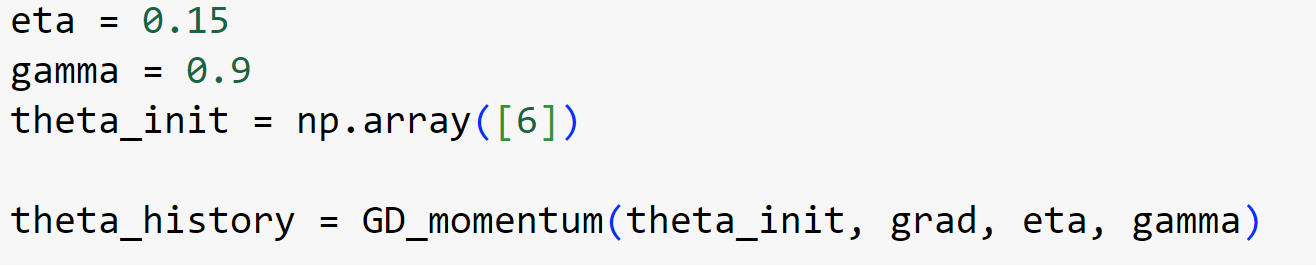


Figure 2.18: Initialization

In the initialization of the Momentum Gradient Descent algorithm, we start assigning the learning rate as 0.15, the value as 0.9 and starting the theta value from 6.

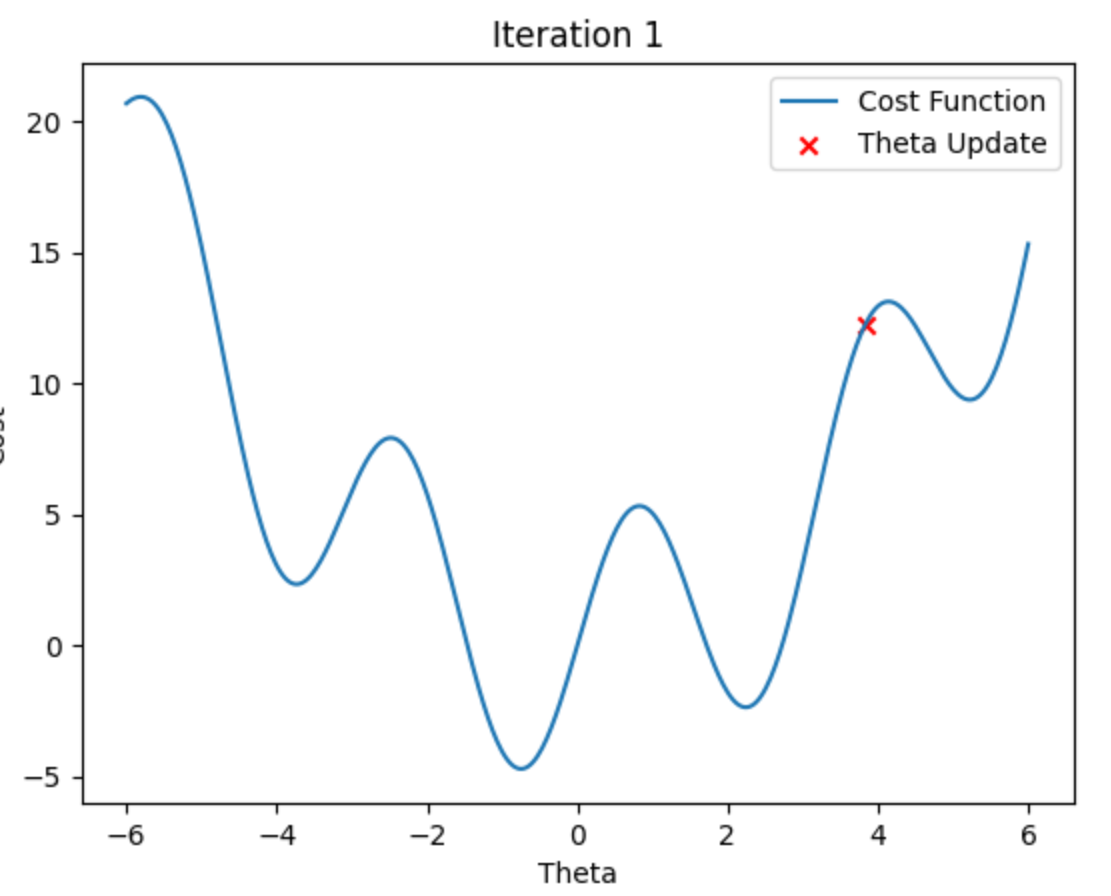


Figure 2.19: Momentum Gradient Descent Iteration 1

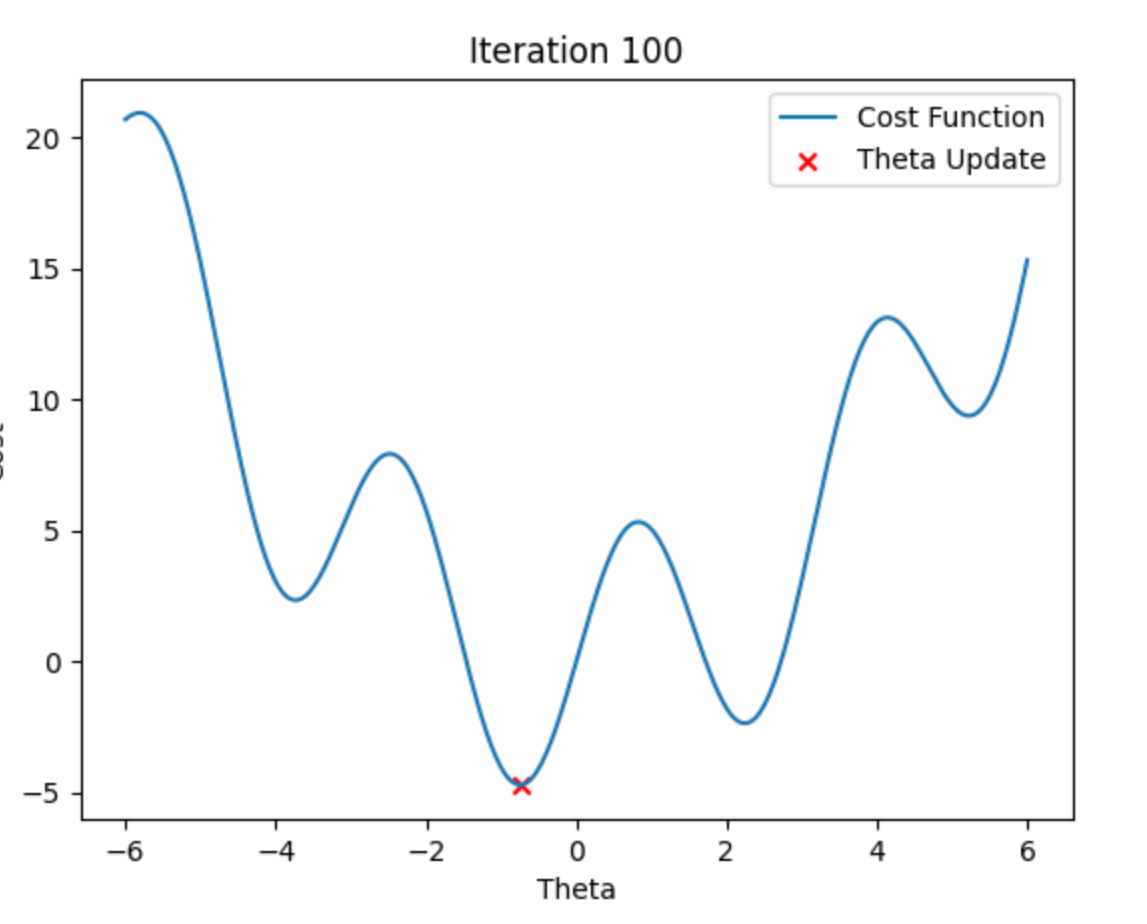


Figure 2.20: Final Iteration Momentum Gradient Descent

In the final iteration, the gradient descent value has converge into the global minima position, instead of being stuck in a local minima postion, achieving the main goal of the Momentum Gradient Descent.

2.5 Adagrad (Adaptive Gradient Algorithm)

One of the weaknesses of the above group of the Gradient Descent algorithms is that we have to manually determine the value of the learning rate, at first this seems like innocent since we can see the changes and adjust the value accordingly, but for some complex function, we need some ways to better adjust the learning rate based on the gradient value.

To achieve this, an algorithm called Adagrad or Adaptive Gradient Algorithm is used. This algorithm will recalculate the learning rate based on the gradient value in each iteration. The formula to update the theta for Adagrad is defined as follows.

Of which, is the current learning rate, is the cumulative sum of square gradients, is a value to prevent the denominator of becoming zero, is a N – ary multiplication operator.

To demonstrate the algorithm, we have the following Python program to simulate the Adagrad calculation steps.

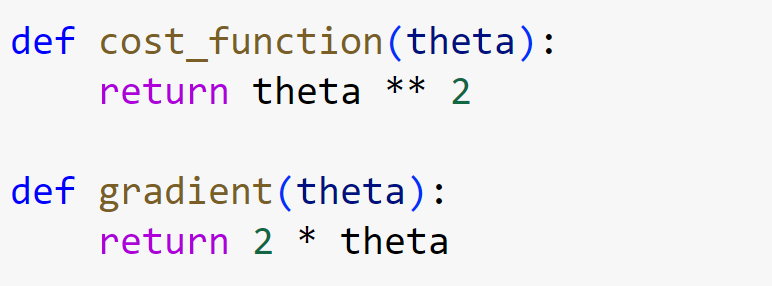


Figure 2.21: Cost Function

Here, our chose cost or loss function is , and the gradient function will now become .

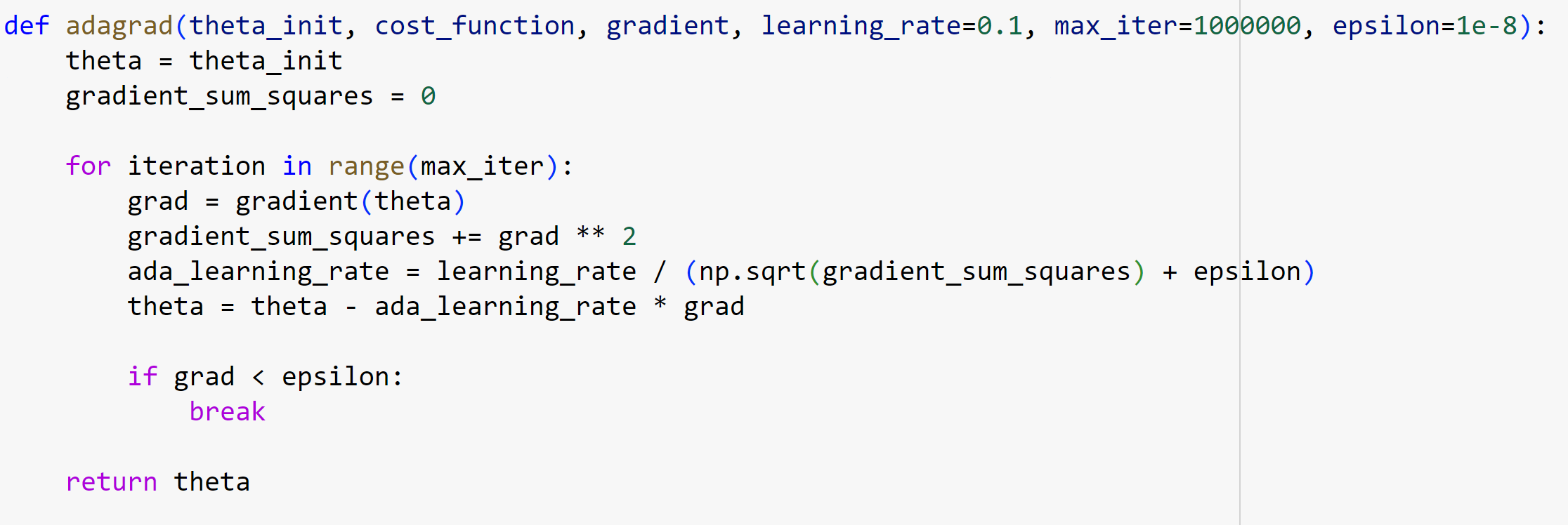


Figure 2.22: Adagrad Iteration

In the Adagrad iteration steps, we initialize the initial theta value and the sum of square gradients as zero. Our operation will calculate the updated learning rate and update the new theta value according to the new learning rate. The function will stop when the gradient value is smaller than .

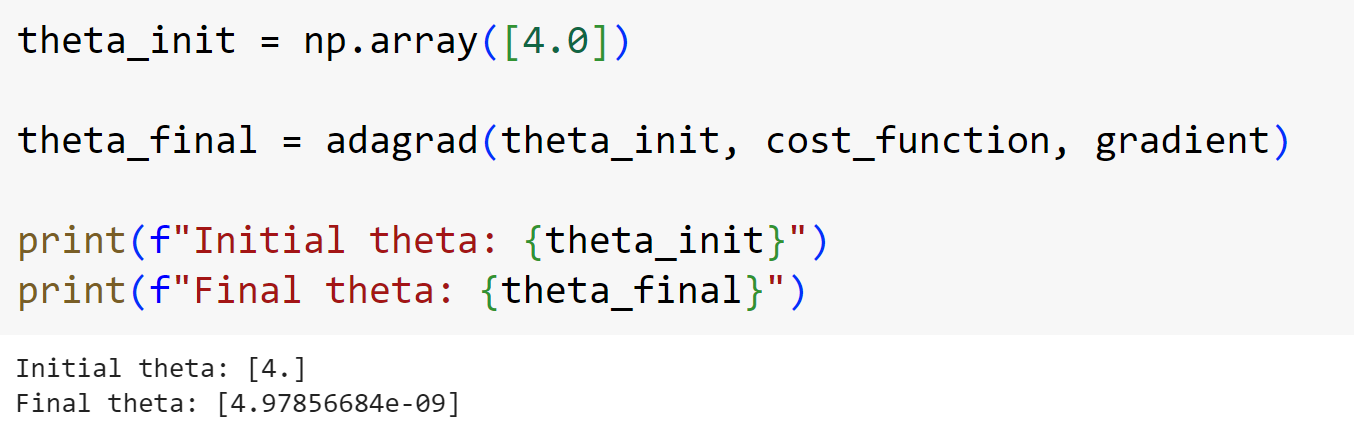


Figure 2.23: Run Adagrad

The final theta value will be printed when the Python program is completed.

2.6 RMSProp

The problem that is most visible when using the Adagrad optimization algorithm is that the learning will gradually drop in the final stages of the algorithm, which means that the learning process will come to a halt or not reaching an optimal solution.

To solve the above problem, we come to an algorithm called RMSProp, or Root Mean Square Propagation, instead of adjusting the learning rate monotically for parameters, RMSProp adjust the learning rate base on the gradient of each parameter.

The main difference of the algorithm compare to Adagrad is that it introduces some factor to prevent the learning rate from becoming excessively small, it also uses a moving average of squared gradients values, which balancing out the need for decreasing the step size for parameters with big gradient values and the need of not letting the learning rate drop too quick.

To update the theta parameter, we have the following formula.

Of which, is the moving average of the squared gradients in iteration . To calculate this parameter, we have a following formula.

In the formula, the decay factor is usually 0.9 so we are going put here to better generalize the formula, is the square of the gradient at iteration .

To demonstrate the RMSProp optimization algorithm, we have the following Python program, compared to the Adagrad code, we will reuse the cost and the gradient function.

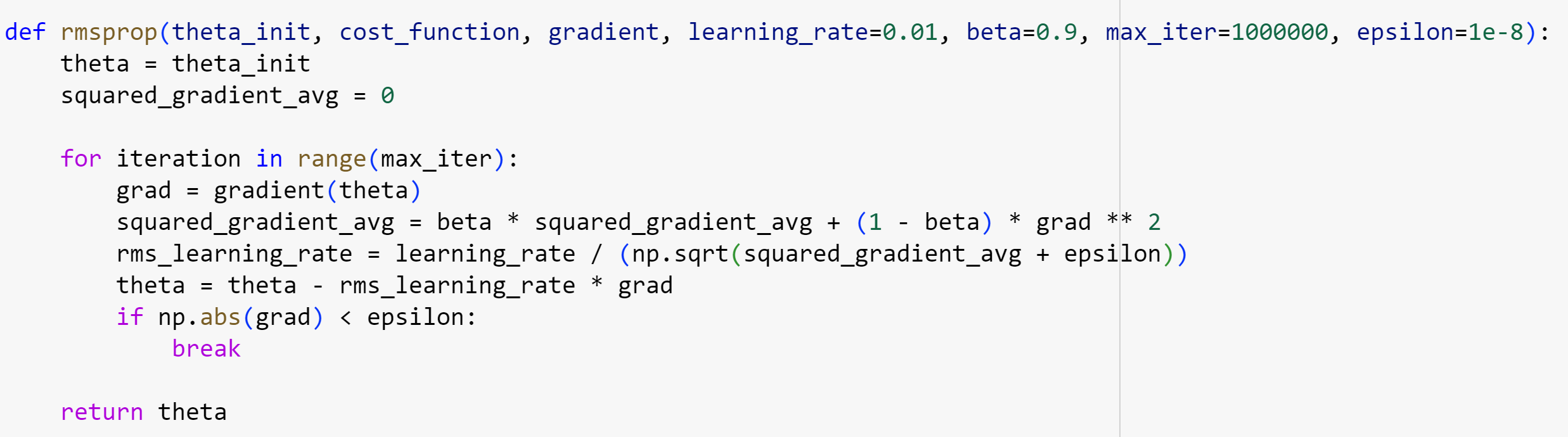


Figure 2.24: RMSProp Iteration

In the RMSProp function, we first initialize the first theta and the first squared gradient average. In each of the iteration step, we simulate the updating step based on the gradient and value.

If the gradient is smaller is smaller than the epsilon threshold then we will stop the function and return theta.

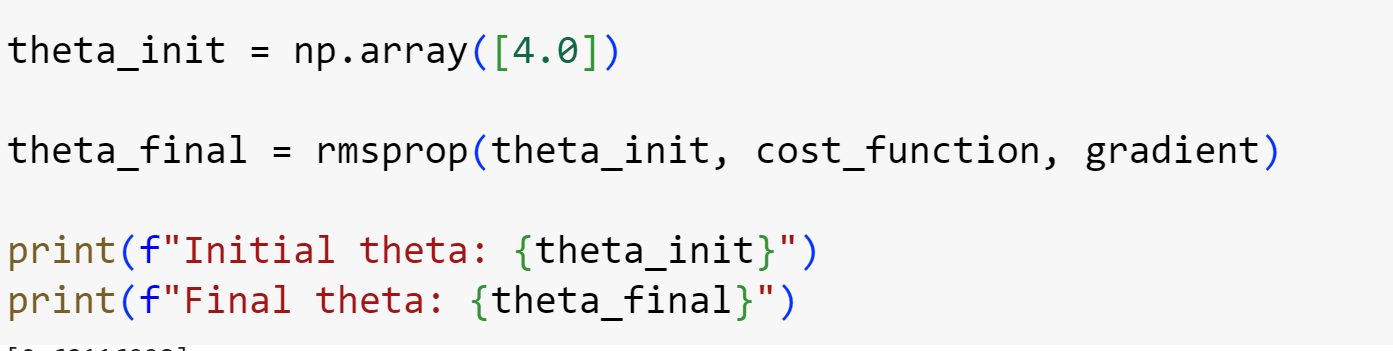


Figure 2.25: Run RMSProp

To run RMSProp, we need to first choose a initial theta value, here we set it as four, and the program will print out the theta value for comparision in the console.

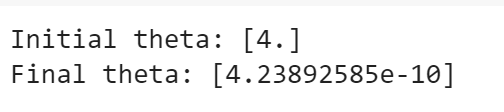


Figure 2.26: RMSProp Result

2.7 ADAM (Adaptive Moment Estimation)

The problem with the RMSProp algorithm is that it sometimes can only returns you the value of the local minima. To combat this weakness, we have combined the Momentum mechanic with RMSProp’s ability to generate adaptive learning rate based on the moving average of squared gradients.

Adam works by updating the network weights based on the training data. It calculates the exponential moving average of gradients and squared gradients. The parameters controls the moving average.

The formula will first calculate the the first moment gradient estimate and the second raw moment estimate. The formulas are as follows.

After computing the first moment gradient estimation and the second raw moment estimation, we now compute the bias – corrected first moment estimate and the bias – corrected second raw moment estimate. The formulas are as follows.

Finally, the theta value will be updated through the following formula.

Of which, serves as the learning rate, are the decay rates for moment estinmates, usually close to 1 or equal to 0.9 for and 0.999 for . is the small constant to prevent a division by zero.

To demonstrate the algorithm in Python, we will choose the cost and the gradient function as follows

The derivative with respect to is:

And the derivative with respect to is:

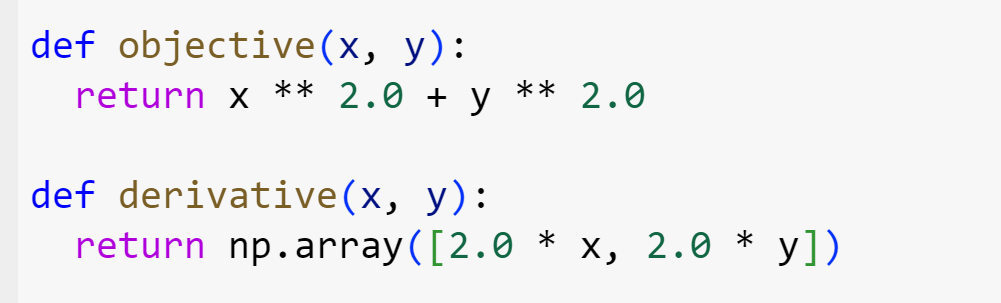


Figure 2.27: Cost Function

The derivative function will save the derivative values based on and respectively.

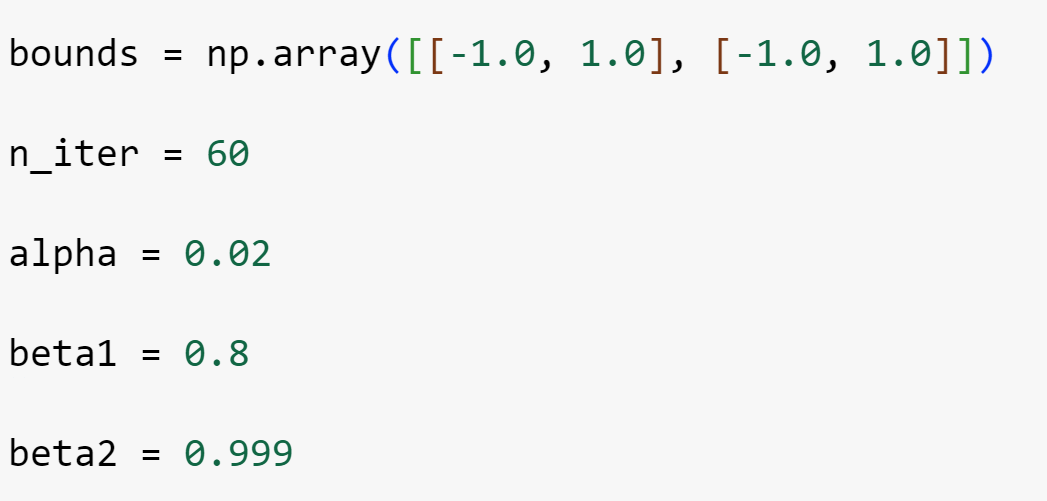


Figure 2.28: Initialization

The maximum number of 60 iteration, the learning rate is at 0.02 and .



Figure 2.29: Adam Iteration

For this iteration steps in the Adam simulation function, we will calculate the theta value as , the scores list is used to store the value of the cost function and the trajectory is used to store the value of the theta value.

After calculating everything, the Adam function will return the theta values, the list of scores of the cost function and the list of trajectory.

To show the achieved trajectory, we will demonstrate by running the algorithm, extracting the results, then we will plot the graph on a 3 – D graph.

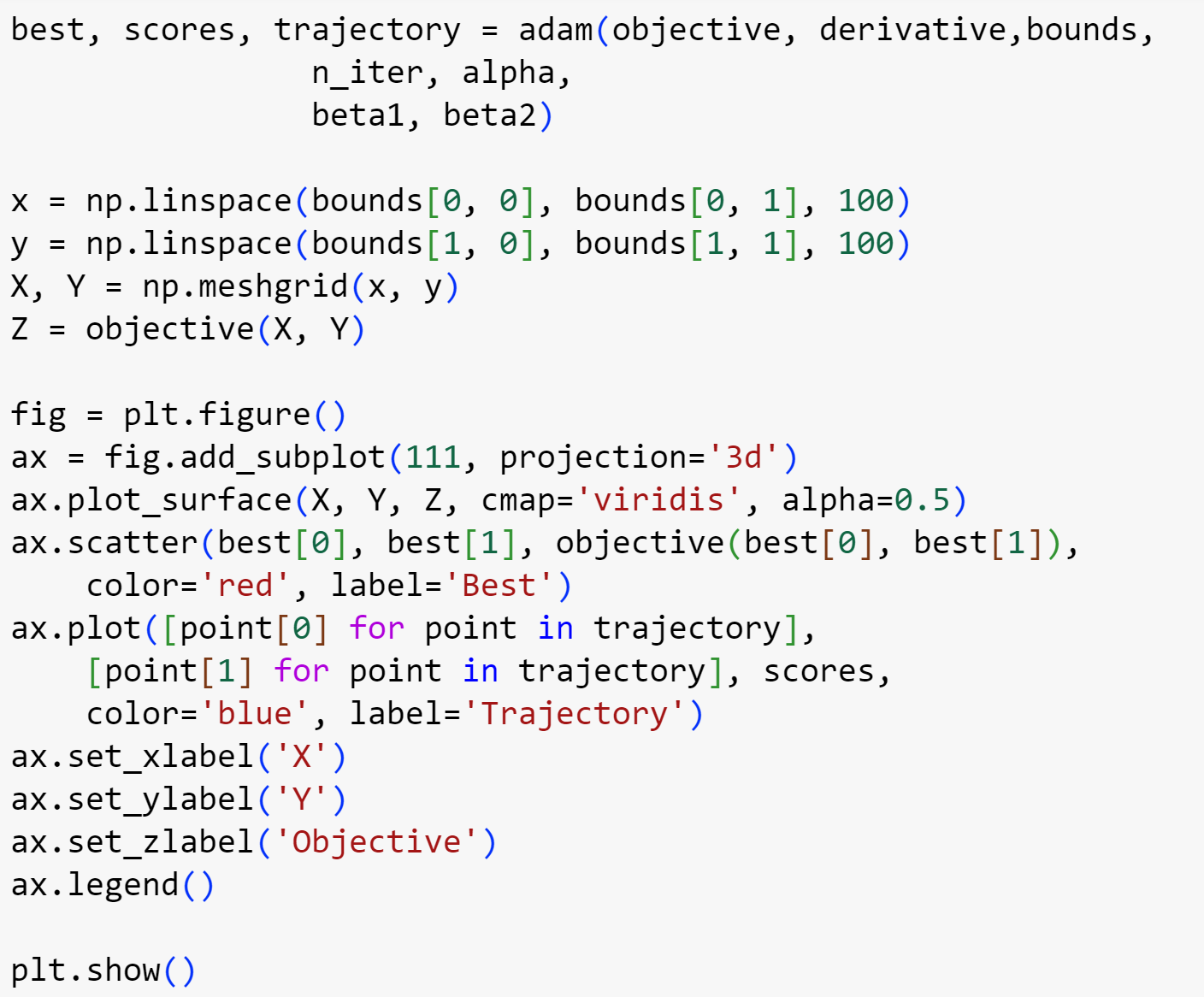


Figure 2.30: Plotting Adam

To plot the Adam algorithm convergence, we first draw a 3 – d representation of the cost function . The we will plot the red point as the best found theta value on the graph, and the trajectory to visualize the movement of the theta value.

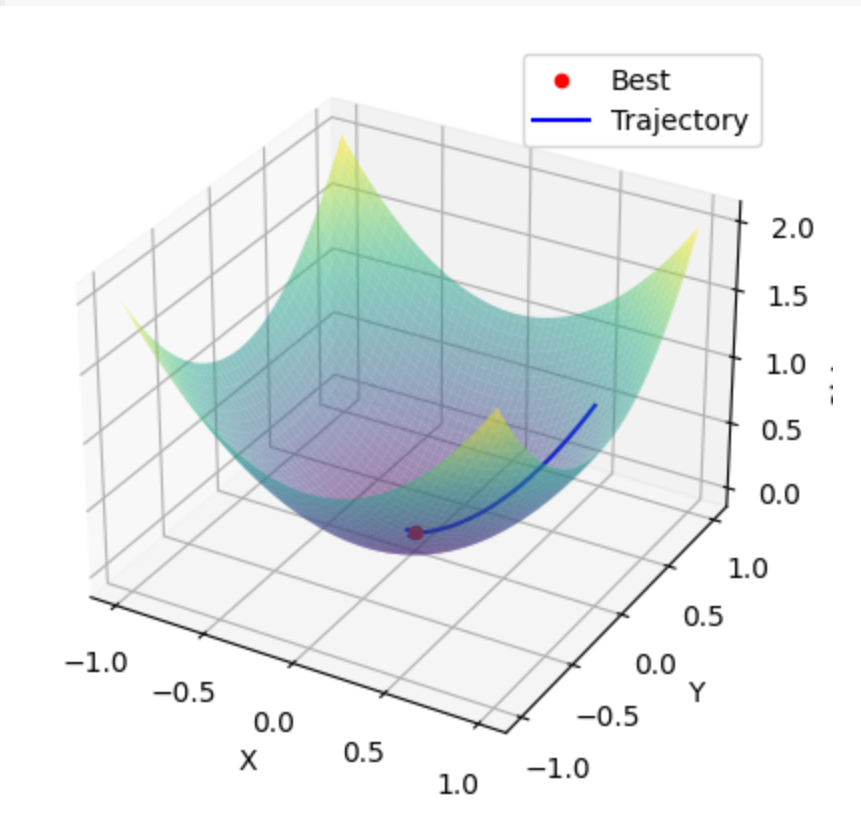


Figure 2.31: Plotting Result

The result here show the movement of the theta value on the graph of the cost function, as we can see, the best theta value is located as the deepest point of the graph, indicating that the Adam has achieved its goal.

2.8 Comparision

In conclusion, most of the optimization algorithm will directly use some form of the regular gradient descent algorithm function. The regular algorithm will use all of the data in each epoch to update the theta value, the Stochastic version of it will only use one data point to update the theta value and the mini – batch version will use a number of data points to update the theta value.

To solve the problem of determining a suitable learning rate, some adaptive learning rates algorithm has been implemented such as RMSProp and Adagrad. The problem with those two approaches mainly comes from the learning rate values becoming too small, such in the case of Adagrad, and the momentum of the algorithm is not enough to reach an optimal minima.

To combat this, Adam is an algorithm that combines Momentum with RMSProp, which makes the algorithm sufficient with many datasets, as it is one of the most used algorithm for optimization.

CHAPTER 3 – CONTINUAL LEARNING AND TEST PRODUCTION

3.1 Continual Learning

In machine learning, when creating a new model, it is important for your machine learning model to be adaptive to new data without explicitly programmed to do so. Machine learning, as the name suggest, is a field where we find solutions for computer to behave like human, the main goal of machine learning is to make computer to be responsive to changes in the data input. In short, Continual Learning is an approach to make models to learn continously and accumulate data overtime. This is one of the most important aspect of machine learning, helping the field to become what it is known for today.

3.1.1 Importance

Continual learning is a concept which refers to the ability to adapt to new data of a machine learning model. In normal days’ life, we are constantly being provided with new data, as a human, we can process those data and make decision based on them. For example, when we were students, teachers would constantly giving us new lesson in every class, our job is to process those new information to solve homework, do exams,… While learning the new materials, we must not forget what we had learned as well, our main goal is to integrate the old information with new one to continously learn and improve everyday. A machine learning model must be adaptive if it wants to be relevant and to be called a “learning” model.

There are few reasons to be adaptive as a machine learning model, some of those are:

* Being Efficient: A machine learning model needs to be efficient when encountering new data. Being resourceful when learning new information.
* Being Adaptive to new information: Ensure the model to have the capability to be learning new data.
* Practicality: The model must have an ability when encountering new information and keeps the model from being obsolete.

3.1.2 Methods

To implement continual learning for a machine learning model, there are a handful of ways to help the model to achieve its goal, of which includes:

* Incremental learning: Refers to the ability of an algorithm to learn continously, adding new information while retaining the previous information. It involves the process of updating a model without the need of retaining the data from scratch. It is most useful when the data stream is continuously being generated.
* Transfer learning: Refers to the ability of a model to initally used for a problem, but can also transfer to another problem, which has to be related. This type of learning is most present in the fields of computer vision and natural language processing.
* Lifelong learning: Refers to the ability of a model to learn continuously from different information overtime, accumulate the new data and refining its knowledge. This is most present in problems where the data is changing over a period of time.

3.1.3 Problems

The problems that most will face when creating a continual learning model is when we have to face with the following problems:

* Catastrophic Forgetting: An instance where the model completely overwrite previous learning information for new data.
* Resource Efficiency: An instance where the developers have to balance out the resource when training on a new data.
* Evaluation Method: Creating an effective method to assess the model based on new information.

To tackle those problems, we have developed a number of methods to preserve the relevance of a model, those can includes:

* Data Rehearsal: Mixing the new data with a subset of old data when training.
* Dynamic Architecture: Expands the model capacity when new problems are introduced, this is to incorporate the new information to the prior knowledge. This can be found in Progressive Neural Networks.

3.2 Test Production

Test Production refers to the process of testing out a model when completing the development process, this can involves multiple phases such as testing on a real – world data for accuracy, determining the performance,…

3.2.1 Importance

Production testing is a phase in the development lifecycle where developers try out the theorectical aspect of the machine learning model in comparision to the complexity of real – world information and data.

The following aspects of the testing phase are the plot points of why we need the phase of testing:

* Ensuring performance: Every application or model when develop has to take in account the ability to scale to the real – world needs.
* Accuracy assessment: Ensure the model capabilities to perform well in terms of accuracy based on new or unforseen data.

3.2.2 Methods

To assess a model, developers usually use a number of evaluation methods to determine whether a model is efficient enough to be released. These can includes:

* A/B Testing: A testing process of comparing a new model to an existing one, determing whether the new model is better or worse in comparision to the old model.
* Canary Releases: A process where the model is used on a small subsets or in a limited environment.
* Shadow Mode: A process where a testing model is running parallelly with the current production model.

And various other evaluation methods, please notice that based on the model and the availability of datasets that we produce the testing phase using different techniques.

REFERENCES

ENGLISH

* <https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/>
* <https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/>
* <https://www.geeksforgeeks.org/how-to-implement-adam-gradient-descent-from-scratch-using-python/>

VIETNAMESE

* <https://machinelearningcoban.com/2017/01/12/gradientdescent/>
* <https://machinelearningcoban.com/2017/01/16/gradientdescent2/>
* <https://d2l.aivivn.com/chapter_optimization/rmsprop_vn.html>
* <https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8>