SCIENTIFIC TEST & ANALYSIS TECHNIQUES
CENTER OF EXCELLENCE

Fundamentals of Gradient Descent

August 2024

Joseph Lazarus, Ctr

To develop and apply independent, tailored Scientific Test & Analysis Techniques solutions to Test and Evaluation that deliver insight to inform better decisions.

About this Publication:
This work was conducted by the Scientific Test & Analysis Techniques Center of Excellence under contract FA8075-18-D-0002, Task FA8075-21-F-0074.

For more information:
Visit, www.AFIT.edu/STAT
Email, AFIT.ENS.STATCOE@us.af.mil
Call, 937-255-3636 x4736

Technical Reviewers:
Corinne Stafford
Anthony Sgambellone

Scientific Test & Analysis Techniques Center of Excellence
2950 Hobson Way
Wright-Patterson Air Force Base, Ohio

Version: 1, FY24

Abstract

This paper presents a clear explanation of the gradient descent algorithm, a fundamental optimization technique widely used in machine learning. By breaking down the mathematical principles and historical context, the paper aims to enhance understanding among practitioners, managers, and executives of how gradient descent functions in optimizing machine learning models. A practical demonstration using a simple linear regression problem illustrates the iterative process of minimizing a loss function, providing a tangible example of how models learn.

Keywords: Gradient Descent, Machine Learning, Optimization Techniques, Loss Function, Decision-Making in AI

Table of Contents

Introduction

Artificial Intelligence (AI) and Machine Learning (ML), particularly Neural Networks (NNs), have demonstrated remarkable capabilities across various domains, from image and speech recognition to game playing and autonomous driving. However, the intricacies of how these technologies operate often remain opaque, especially to non-technical stakeholders such as managers and executives. The complexity of neural network architectures and their training algorithms can obscure understanding, making it challenging to evaluate their performance and potential applications. This paper aims to demystify one of the foundational algorithms in machine learning, gradient descent, to enhance the comprehension of ML concepts among key stakeholders.

Purpose

The primary purpose of this paper is to educate test and evaluation practitioners, managers, and executives about the fundamental mechanisms and principles underlying gradient descent. By breaking down this concept into more accessible terms and demonstrating its practical applications, this paper seeks to empower decision-makers with the knowledge to better assess and leverage AI technologies within their organizations.

Overview of Content

This paper will begin by providing background information on the basic concepts of AI and ML, an introduction to optimization techniques. It will then delve into the gradient descent algorithm, explaining its purpose, basic functioning, and historical context, referencing seminal works by Lemaréchal (2010) and Boyd (2016). Following this, the paper will explore how gradient descent is integrated to enhance learning and accuracy in machine learning models. The paper will summarize the key points and provide a comprehensive list of references. In the appendix, the paper offers a practical demonstration that illustrates key concepts of gradient descent.

Background

Basic Concepts of Artificial Intelligence and Machine Learning

AI, as defined by Russell and Norvig (2020) in "Artificial Intelligence: A Modern Approach," is the study of rational agents. These agents perceive their environment through sensors and take actions to maximize their chances of achieving specific goals. AI systems, therefore, mimic cognitive processes like learning, reasoning, problem-solving, perception, and language understanding. This broad definition encompasses a variety of AI approaches. These include areas such as natural language processing, robotics, and computer vision. Machine Learning (ML) is a subset of AI that focuses on the development of algorithms and statistical models that allow computers to learn from and make predictions or decisions based on data. Within ML, neural networks (NNs) are a critical technique, inspired by the structure and function of the human brain.

Neural networks are computational models composed of interconnected layers of nodes, or neurons, that process input data to generate outputs. These networks are designed to recognize patterns and relationships in data through a process of learning, which involves adjusting the weights of the connections between neurons to minimize the difference between the actual output and the desired output.

Introduction to Optimization Techniques

Optimization is a critical technique for machine learning, involving the selection of the best parameters for a model to minimize or maximize a specific objective function. In the context of neural networks, optimization techniques are used to adjust the weights and biases to reduce the error between the predicted and actual outputs. Gradient descent is one of the most widely used optimization algorithms in ML.

Understanding Gradient Descent

Introduction to the Algorithm

Gradient descent is an iterative optimization algorithm used to minimize a function by moving in the direction of the steepest descent as defined by the negative of the gradient of the loss function, which is defined in the mathematical principles section. The primary objective of gradient descent is to find the set of parameters that minimizes the loss function, which measures the difference between the model's predictions and the actual target values, see Figure 1.

The gradient descent algorithm involves three main steps: initializing the parameters, computing the gradient of the loss function with respect to the parameters, and updating the parameters in the direction of the negative gradient. This process is repeated iteratively until the algorithm converges to a minimum of the loss function.

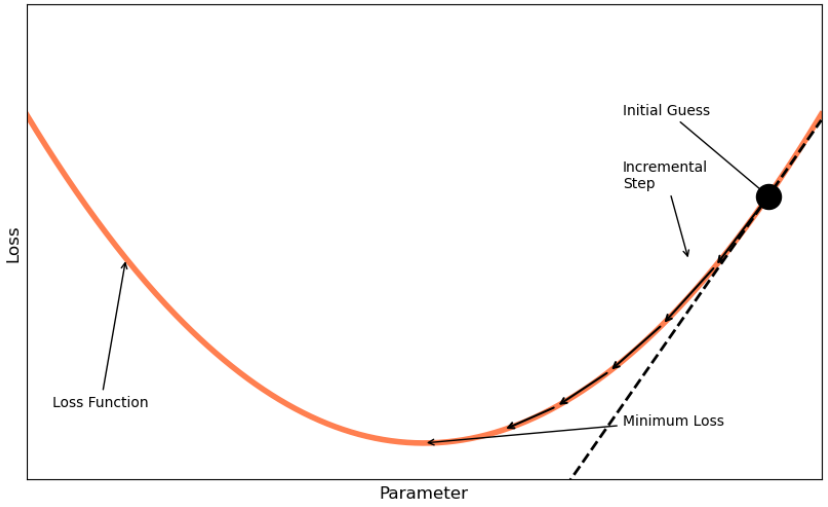

Figure 1
Gradient Descent Visualization

Historical Context

The concept of gradient descent dates to the 19th century, with the work of Augustin-Louis Cauchy in 1847. His paper, "A Method for the Solution of Certain Problems in Least Squares," introduced the gradient method for solving optimization problems (Lemaréchal, 2010). This foundational work laid the groundwork for modern optimization techniques.

In more recent years, the application of gradient descent in machine learning and neural

networks has been popularized through seminal works such as "Learning representations by back-propagating errors" by Rumelhart, Hinton, and Williams (1986), and "Gradient-Based Learning Applied to Document Recognition" by LeCun, Bottou, Bengio, and Haffner (1998). These works demonstrated the effectiveness of gradient-based optimization techniques in training neural networks.

Mathematical Principles

The mathematical foundation of gradient descent is rooted in calculus, specifically the gradient of a function. The gradient is a vector that points in the direction of the steepest increase of a function. In context of a function with a single variable, the gradient corresponds to the derivative, which represents the rate of change or slope of the function. For functions with multiple variables, the gradient is a vector of partial derivatives with respect to each variable. By moving in the opposite direction of the gradient, the algorithm aims to find the minimum of the loss function. This means that for each step or iteration, the parameters are adjusted in a way that reduces the value of the loss function, ideally leading to its minimum.

Mathematically, the update rule for gradient descent is given by:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \, \nabla_{\theta} L(\theta)$$

where theta (θ) is a vector representing the parameters, eta (η) is the learning rate, and $\nabla_{\theta} L(\theta)$ is the gradient of the loss function with respect to the parameters. Note: The learning rate is often denoted by alpha (α) or eta (η). This paper will use eta (η) as the learning rate which controls the step size in the gradient descent algorithm.

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated, depicted as the incremental step in Figure 1. It is crucial to choose an appropriate learning rate; if it is too large, the gradient descent algorithm may diverge or converge too quickly to a suboptimal solution, while if it is too small, the convergence process can be very slow. When setting the learning rate:

- Start with a small value: common initial values range from 0.001 to 0.1
- Consider using an adaptive method: techniques like rate schedules or adaptive learning rate algorithms (e.g., Adam, RMSporp) can adjust the learning rate during training for better performance
- Monitor convergence: plot the loss function to ensure it decreases smoothly; if the loss fluctuates or diverges, the learning rate might be too high; if the plot decreases too slowly, the learning rate might be too low

Break Down of the Process

Gradient descent is accomplished through the following steps.

1. **Initialization:** Start with an initial guess for the parameters.
2. **Compute Gradient:** Calculate the gradient of the loss function with respect to the parameters.
3. **Update Parameters:** Adjust the parameters by subtracting the product of the learning rate and the gradient.

4. **Check Convergence Criteria:** Determine when to stop the iteration process. The algorithm can be stopped when one of the following conditions is met:
    - The change in the loss function between iterations is smaller than a predefined threshold epsilon (ϵ): This criterion checks the difference in the value of the loss function (e.g., mean squared error) between consecutive iterations. If the change is smaller than a predefined value ε, it indicates that further iterations are not significantly reducing the error, suggesting convergence. Mathematically, this can be expressed as:

      $$|L(\theta_{\text{new}}) - L(\theta_{\text{old}})| < \epsilon$$

      This ensures that the model has reached a point where the error is minimal and is not improving appreciably with further iterations. This focuses on improvement in the model's performance, (i.e., how much the error or loss has decreased)

    - The change in the parameter values between iterations is smaller than a predefined threshold: This criterion focuses on the magnitude of the adjustments made to the parameters themselves. If the changes in the parameters θ between iterations are smaller than a predefined value, it indicates that the algorithm is making very small updates and is likely near the minimum of the loss function. This can be mathematically expressed as:

      $$|\theta_{\text{new}} - \theta_{\text{old}}| < \epsilon$$

      This ensures that the parameters have stabilized and are no longer changing significantly, indicating convergence. This focuses on the adjustments made to the model parameters (i.e. how much the parameters have changed)

    - A maximum number of iterations is reached: This criterion sets a cap on the number of iterations the algorithm will perform. It acts as a safeguard to prevent the algorithm from running indefinitely in cases where convergence is slow or difficult to achieve. Typical maximum numbers of iterations can vary depending on the problem and dataset but often range from 1000 to 10,000 iterations. For example, in simple linear regression problems, fewer iterations may be needed, whereas complex neural network training might require many more iterations.
5. **Repeat:** Continue repeating steps 2-4 until the loss function converges to a minimum value or the maximum number of iterations is reached.

Applications of Gradient Descent in Machine Learning

While gradient descent can be applied to various optimization problems, including linear regression, it is not typically the preferred method when a closed-form solution is available. Linear regression problems can be optimized with a single equation and do not require the iterative computations of gradient descent. When a closed form solution exists, it is computationally efficient and provides exact results without the need for iterative updates.

Gradient descent is particularly useful in machine learning problems with:

1. High Dimensionality: When the parameter space is very large, direct methods often become computationally infeasible due to the high cost of matrix inversion. Gradient descent can handle large-dimensional problems by iteratively updating the parameters.
2. Non-Linear Models: Models like neural networks are non-linear and have complex loss surfaces. Gradient descent and its variants (e.g., stochastic gradient descent, mini-batch gradient descent) are well suited for training these models due to their ability to navigate high-dimensional, non-convex loss landscapes.

Neural networks are a prime example where gradient descent is essential. They consist of multiple layers of interconnected neurons, and the objective is to minimize the loss function, which measures the discrepancy between the network's predictions and the actual target values. The high dimensionality and non-linear nature of neural networks make gradient descent an ideal optimization method.

Neural networks often have millions of parameters, making direct optimization methods impractical. Additionally, the loss function in neural networks is differentiable, which means we can calculate the gradient of the loss with respect to each parameter and iteratively update the parameters to minimize the loss. This process is known as backpropagation. For an in-depth explanation of backpropagation and an Excel simulation of a neural network, see "Fundamentals of Backpropagation" (Lazarus, 2024).

Conclusion

Understanding the mechanics of gradient descent is essential for comprehending how machine learning models learn and improve their performance. This paper has aimed to clarify the gradient descent algorithm by breaking down its fundamental principles, historical context, and mathematical underpinnings. By examining the gradient computation, parameter updates, and convergence criteria, this paper aimed to provide a clear and accessible explanation of how gradient descent operates.

Gradient descent's significance in the field of machine learning cannot be overstated. It is the cornerstone of training various models, enabling them to adjust their parameters to minimize errors and enhance accuracy. The algorithm's introduction and subsequent development by pioneers like Rumelhart, Hinton, and LeCun have paved the way for the remarkable advancements in AI and deep learning that we witness today.

For managers and executives, gaining a solid understanding of gradient descent equips them with the knowledge to make informed decisions about adopting and implementing AI technologies within their organizations. This comprehension allows them to better evaluate the capabilities and limitations of machine learning models, ensuring that they can leverage these powerful tools effectively to drive innovation and achieve mission objectives.

As AI and machine learning continue to evolve, the principles of gradient descent will remain foundational to the development of more sophisticated and capable models. By demystifying this algorithm, the paper aims to equip leaders with the necessary understanding to effectively apply AI and machine learning algorithms within their operations contexts.

To further illustrate the concepts discussed, the appendix contains a practical example that demonstrates gradient descent through a simple linear regression model. This example will

provide a hands-on understanding of how gradient descent works in practice, highlighting the step-by-step process of training a model, adjusting parameters, and minimizing error to achieve accurate predictions.

References

Boyd, S. (2016). *Convex Optimization*. Cambridge University Press. (Original Work published 2004). https://doi.org/10.1017/CBO9780511804441

LeCun, Y. Bottou, L., Bengio, Y., & Haffner, P. (1998). *Gradient-based Learning Applied to Document Recognition*. Proceedings of the IEEE, 86(11), 2278-2324

Lemaréchal, C. (2010) *Cauchy and the Gradient Method*. Doc. Math. Extra Vol. ISMP, 251-254.

Rumbelhart, D. E., Hinton, G.E., & Williams, R.J. (1986). *Learning Representation by Backpropagating Errors*. Nature, 323(6088), 533-536.

Russell, S., Norvig, P. (2020). *Artificial Intelligence*: A Modern Approach (4th ed). Pearson.

Appendix

Illustrative Example

Example Scenario

Consider a simple dataset where the x-axis represents weight, and the y-axis represents height. By fitting a line to this data, weight can be predicted for any given height. The equation of a line is given by $y = mx + b$, where y is the dependent variable (height), x is the independent variable (weight), m is the slope of the line, b is the intercept. The process of gradient descent aims to fit such a line to the data by finding the optimal values for the intercept (b) and the slope (m) which minimize the sum of squared residuals. Table A1 and Figure A1 show the data that will be fit.

Table 1

Height and Weight Data

Weights (units) (x_i)	Height (units) (y_i)
0.5	1.4
2.3	1.9
2.9	3.2

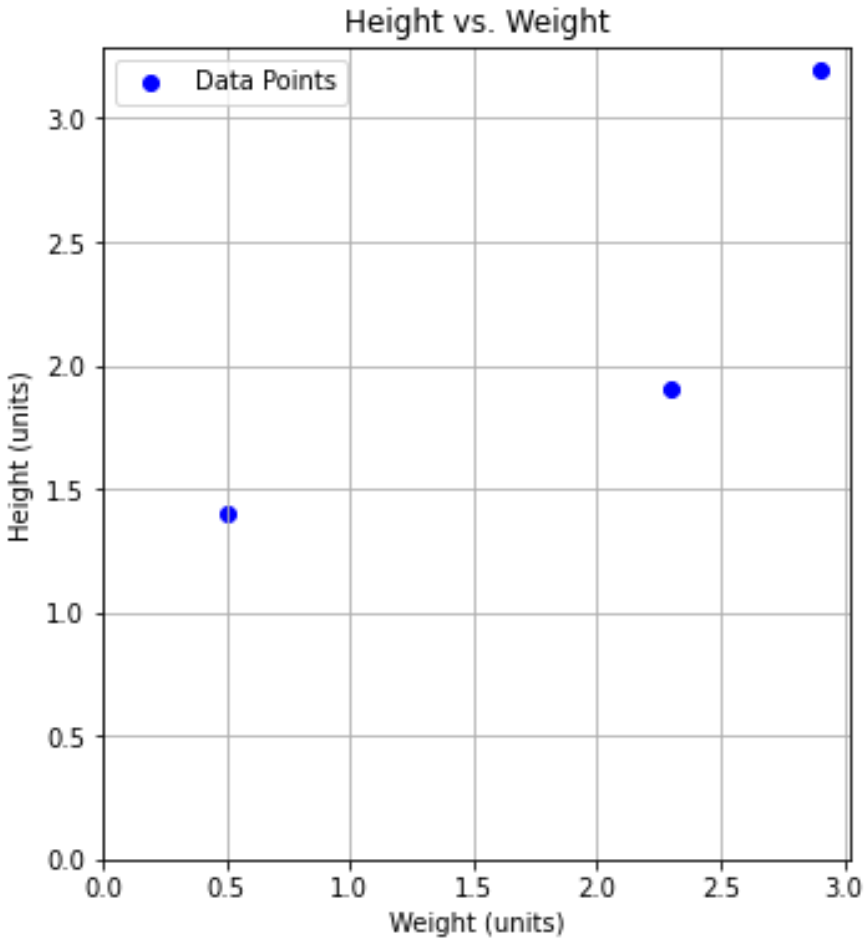

Figure A1

Plot of Height and Weight Data

If we were using the sum of squared residuals (SSR) method to solve for the optimal intercept and slope, we would find where the derivative (slope) of the error function equals zero. In contrast, gradient descent finds the minimum value of the loss by taking iterative steps from an initial guess until optimal values are reached. This iterative approach makes gradient descent particularly useful in scenarios where it is not possible to directly solve for where the derivative equals zero, enabling its application in a wide range of situations.

First Iteration of Gradient Descent

Initial Setup

We start with initial guesses for the intercept (b) and slope (m). In this case, we'll begin with $b = 0$ and $m = 1$. We also define a learning rate (η), which controls the size of our steps. Let's set $\eta = 0.01$.

Step 1: Calculate Model Predictions

Calculate predicted values using our initial guesses, we calculate the predicted height ($\hat{y}$) using 1 for the slope (m) and 0 for the intercept (b) for each weight:

$$\hat{y}_1 = 1 \cdot 0.5 + 0 = 0.5$$
$$\hat{y}_2 = 1 \cdot 2.3 + 0 = 2.3$$
$$\hat{y}_3 = 1 \cdot 2.9 + 0 = 2.9$$

The prediction versus the data points is pictured in Figure A2.

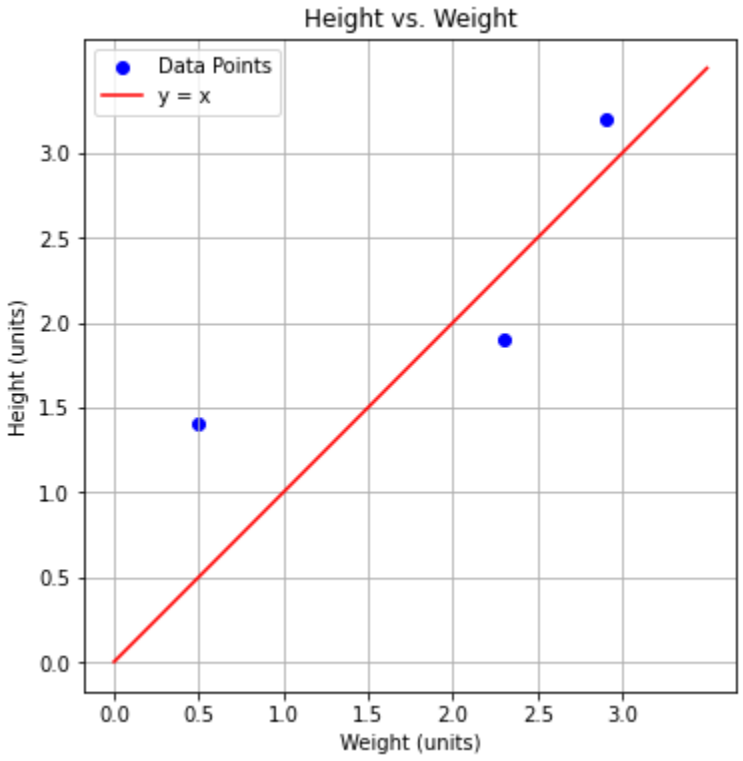

Figure A2
Predicted versus Actual for First Iteration

Step 2: Compute Residuals

Next, we calculate the residuals, which are the differences between the actual heights and the predicted heights:

$$\text{Residual}_1 = 1.4 - 0.5 = 0.9$$
$$\text{Residual}_2 = 1.9 - 2.3 = -0.4$$
$$\text{Residual}_3 = 3.2 - 2.9 = 0.3$$

The residuals are pictured in Figure A3.

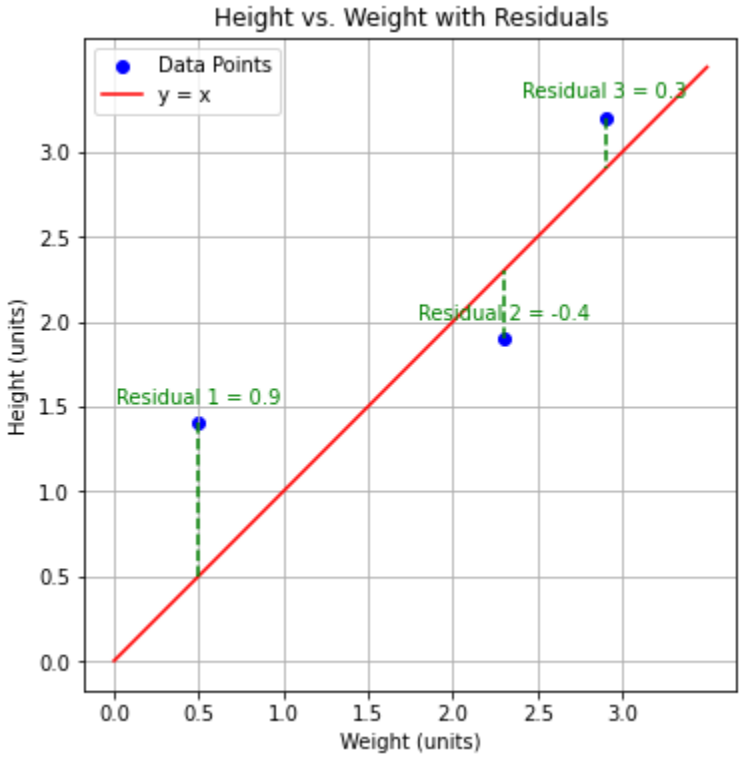

Figure A3
Residuals for First Iteration

Step 3: Calculate the Sum of Squared Residuals (SSR)

The SSR is the sum of the squares of the residuals:

$$\text{SSR} = \sum (y_i - \hat{y}_i)^2 = (0.9)^2 + (-0.4)^2 + (0.3)^2 = 0.81 + 0.16 + 0.09 = 1.06$$

Step 4: Compute the Gradient

We compute the partial derivatives of the SSR with respect to the intercept and slope (together comprising the gradient), which indicate how much the SSR changes with respect to changes in b and m:

$$\frac{\partial \text{SSR}}{\partial b} = -2\sum(y_i - \hat{y}_i) = -2(0.9 - 0.4 + 0.3) = -1.6$$
$$\frac{\partial \text{SSR}}{\partial m} = -2\sum(y_i - \hat{y}_i) \cdot x_i = -2(0.9 \cdot 0.5 - 0.4 \cdot 2.3 + 0.3 \cdot 2.9) = -0.8$$

Step 5: Update the Parameters

We update our guesses for b and m using the gradient and the learning rate:

$$b_{\text{new}} = b_{\text{old}} - \eta \cdot \frac{\partial \text{SSR}}{\partial b} = 0 - 0.01 \cdot (-1.6) = 0.016$$
$$m_{\text{new}} = m_{\text{old}} - \eta \cdot \frac{\partial \text{SSR}}{\partial m} = 1 - 0.01 \cdot (-0.8) = 1.008$$

We repeat these steps, updating b and m each time. With each iteration, the SSR decreases, and our line better fits the data.

Second Iteration of Gradient Descent

Let's continue with the next iteration of gradient descent to see how we are getting closer to minimizing the error.

Recap of Initial Values and First Update

From the first iteration:

- Initial intercept (b) = 0
- Initial slope (m) = 1
- Learning rate (η)

After the first iteration:

- Updated intercept (b) = 0.016
- Updated slope (m) = 1.008

Step 1: Calculate Predicted Values

Using the updated parameters, calculate the predicted height $\hat{y}$ for each weight:

For the first weight (0.5 units):

$$\hat{y}_1 = 1.008 \cdot 0.5 + 0.016 = 0.524$$

For the second weight (2.3 units):

$$\hat{y}_2 = 1.008 \cdot 2.3 + 0.016 = 2.347$$

For the third weight (2.9 units):

$$\hat{y}_3 = 1.008 \cdot 2.9 + 0.016 = 2.937$$

Step 2: Compute Residuals

Calculate the residuals, which are the differences between the actual heights and the predicted heights:

For the first weight (0.5 units):

$$\text{Residual}_1 = 1.4 - 0.524 = 0.876$$

For the second weight (2.3 units):

$$\text{Residual}_2 = 1.9 - 2.347 = -0.447$$

For the third weight (2.9 units):

$$\text{Residual}_3 = 3.2 - 2.937 = 0.263$$

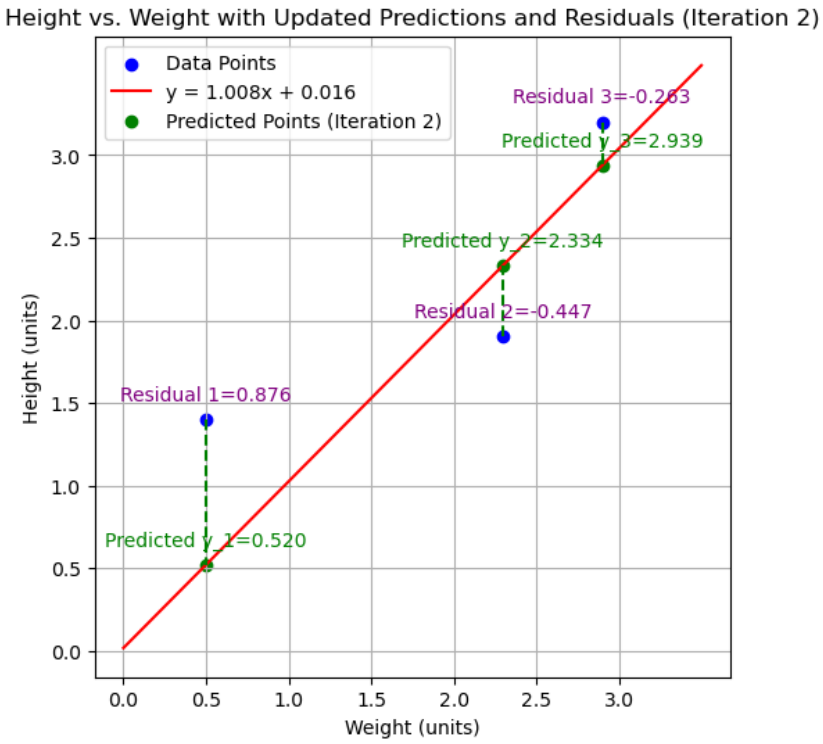

Figure A4
Residuals for Second Iteration

Step 3: Calculate the Sum of Squared Residuals (SSR)

$$\text{SSR} = (0.876)^2 + (-0.447)^2 + (0.263)^2 = 0.767 + 0.200 + 0.069 = 1.036$$

Notice how the SSR (1.036) is slightly lower than the SSR from the first iteration (1.06), indicating that we are moving in the right direction.

Step 4: Compute the Gradient

The partial derivatives of the SSR with respect to the intercept and slope are:

$$\frac{\partial \text{SSR}}{\partial b} = -2\sum(y_i - \hat{y}_i) = -2(0.876 - 0.447 + 0.263) = -1.384$$

$$\frac{\partial \text{SSR}}{\partial m} = -2\sum(y_i - \hat{y}_i) \cdot x_i = -2(0.876 \cdot 0.5 - 0.447 \cdot 2.3 + 0.263 \cdot 2.9) = -0.344$$

Step 5: Update the Parameters

Update the parameters using the gradient and the learning rate:

$$b_{\text{new}} = b_{\text{old}} - \eta \cdot \frac{\partial \text{SSR}}{\partial b} = 0.016 - 0.01 \cdot (-1.384) = 0.016 + 0.01384 = 0.030$$

$$m_{\text{new}} = m_{\text{old}} - \eta \cdot \frac{\partial \text{SSR}}{\partial m} = 1.008 - 0.01 \cdot (-0.344) = 1.008 + 0.00344 = 1.011$$

Explanation of the Process

With each iteration, the parameters (intercept b and slope m) are adjusted slightly to reduce the loss function, in this case SSR. In the second iteration, the SSR decreased from 1.06 to 1.036, showing our updated line fits the data better than the previous iteration.

The iterative nature of gradient descent ensures that with each step, we move closer to the optimal values for b and m that minimize the SSR. The adjustments become smaller as we get closer to the minimum, indicating convergence toward the optimal solution.

As we get closer to the minimum, the slope of the loss function flattens out, meaning the gradient magnitude decreases. A smaller gradient indicates that the error function is not changing as rapidly with respect to the parameters. Consequently, the adjustments to the parameters get smaller.

Mathematically, the parameters updates are calculated as follows:

$$b_{\text{new}} = b_{\text{old}} - \eta \cdot \frac{\partial \text{SSR}}{\partial b}$$

$$m_{\text{new}} = m_{\text{old}} - \eta \cdot \frac{\partial \text{SSR}}{\partial m}$$

Here, $\frac{\partial \text{SSR}}{\partial b}$ and $\frac{\partial \text{SSR}}{\partial m}$ become smaller as we approach the minimum, leading to smaller changes in b and m.

This process continues until the changes in SSR are negligible, meaning we have found the best possible line that fits our data.

Gradient Descent Final Answer and Comparison to Closed-form Solution (SSR)

To determine how many iterations are needed to achieve the optimal solution using gradient descent, the python programming language was used to iterate the process on the data set. The results from gradient descent can then be compared to the closed-form solution.

Gradient Descent Process:

- Initial parameters: Intercept (b) = 0.0, slope (m) =1.0
- Learning rate (η) = 0.01
- Maximum Iterations = 10,000
- Convergence tolerance = 1 x 10^{-6}

  Convergence tolerance is a threshold that determines when the gradient descent algorithm should stop iterating. For this example, a convergence tolerance of 1 x 10^{-6} means the

algorithm stops when the change in parameter values iterations is less than 0.000001. This ensures the algorithm halts when further updates have a negligible effect on reducing the error, indicating near-optimal solutions.

After running the gradient descent algorithm, the parameters converged to the following values.

Gradient Descent Final Parameters:

- Intercept (b): Approximately 0.9486
- Slope (m): Approximately 0.6410
- Number of iterations: 580

Sum of Squared Residuals solution

- Intercept (b): Approximately 0.9487
- Slope (m): Approximately 0.6410

The parameters obtained from gradient descent are very close to the closed-form solution (SSR). This demonstrates that gradient descent can effectively find the optimal solution through iterative updates, even though it required a significant number of iterations to converge. The closed-form solution is easy to obtain for linear regression problems, however, the power of gradient descent is realized on problems where it is not possible to solve for when the gradient equals zero.

Figure A5 shows the minimization of loss over the course of the iterations.

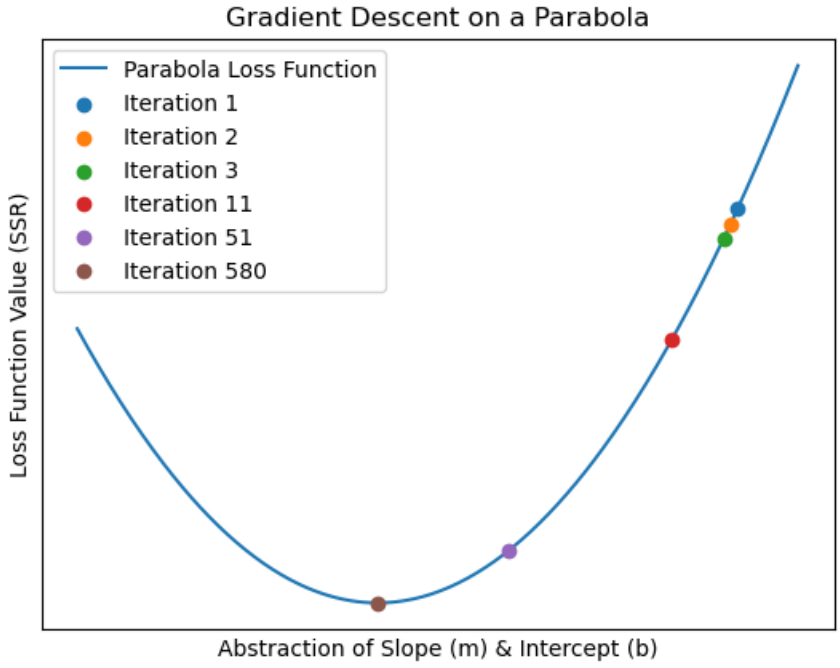

Figure A5
Loss Mimimization Over Iterations

Figure A5 is an educational tool designed to help the reader understand the essence of gradient descent. Although both the slope and intercept influence the loss function, their combined effect

along a single axis is abstracted to make the concept more accessible. The plot captures the key idea that gradient descent is an optimization process, moving toward a minimum error, which is visually represented by the lowest point on the parabola.

Figure A6 illustrates the final solution to the example sum of squared residual problem solved using gradient descent. The blue points represent the original data, and the red shows the best fit line obtained using gradient descent. Green points denote predicted values, and dashed green lines indicated the residuals between the actual and predicted values.

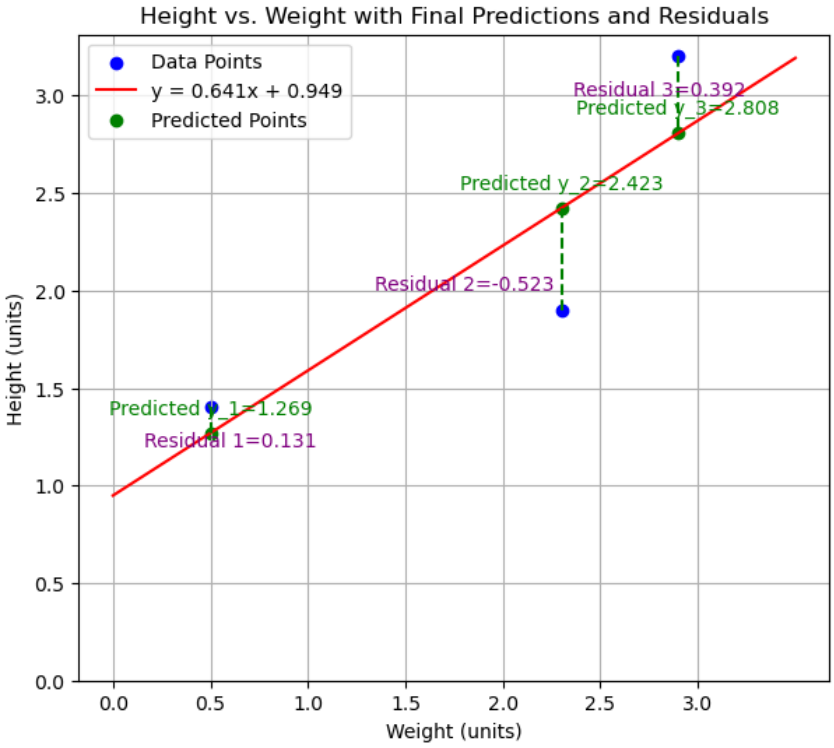

Figure A6
Final Solution