Study of Gradient Descent

Ujjwal Mishra [23162]
Vasu Goel [23166]

Indian Institute of Information Technology Una

This study explores the application of the gradient descent algorithm in optimizing a linear regression model with noisy data. The primary objective was to investigate the effectiveness of gradient descent in identifying optimal model parameters that minimize the loss function. By applying the algorithm to a synthetic dataset, we derived parameter estimates of $a = 0.2046$ and $c = -4.3583$, which led to a model with a high accuracy of approximately 95.97%.

1. Introduction

Gradient descent, first introduced by Augustin-Louis Cauchy in 1847 and independently proposed by Jacques Hadamard in 1907, stands as a cornerstone in optimization theory. Despite its early mathematical origins, the method's practical significance remained largely unexplored until Haskell Curry's groundbreaking analysis of convergence properties in 1944. The algorithm belongs to the class of first-order optimization methods, distinguished from local search techniques by its utilization of derivative information. Its elegance lies in the intuitive principle of following the path of steepest descent in a multidimensional space. While initially conceived for pure mathematical optimization, the method has found renewed prominence in contemporary machine learning, particularly in neural network optimization and statistical modeling. Modern variants, including stochastic and mini-batch implementations, have extended its applicability to large-scale optimization problems. The subsequent sections provide a detailed examination of the algorithm's mathematical formulation, its variants, and the associated loss functions that guide the optimization process.

1.1. Algorithm

The gradient descent algorithm represents a fundamental optimization technique in computational mathematics. Consider an objective function $J(\Theta)$, which defines a hypersurface over the parameter space. In lower dimensions, one can intuitively visualize $J(\Theta)$ as a surface above the parameter space Θ, and this geometric interpretation extends naturally to higher dimensions. The algorithm's primary objective is to locate the global minimum of this surface through iterative refinement. Beginning at an arbitrary point in the parameter space, the method computes the gradient vector $\nabla J(\Theta)$, which indicates the direction of steepest ascent. The algorithm then proceeds in the opposite direction of this gradient by a predetermined step size η, effectively descending toward lower function values. This process continues iteratively, with each step following the path of steepest descent, until convergence criteria are satisfied. In the subsequent sections, we present explicit formulations of the gradient descent algorithm for both univariate ($\Theta \in \mathbb{R}$) and multivariate ($\Theta \in \mathbb{R}^n$) objective functions, along with their respective convergence properties.

1.1.1 Gradient descent in 1 dimension

We start by considering gradient descent in one dimension. Assume $\Theta \in \mathbb{R}$, and that we know both $J(\Theta)$ and its first derivative with respect to Θ, $J'(\Theta)$. Here is pseudo-code for gradient descent on an arbitrary function f. Along with f and its gradient f', we have to specify the initial value for parameter Θ, a step-size parameter η, and an accuracy parameter ϵ.

1D-GRADIENT-DESCENT(Θ_{init}, η, f, f', ϵ)
1. $\Theta^{(0)} = \Theta_{init}$
2. $t = 0$
3. *repeat*
4. $\quad t = t + 1$
5. $\quad \Theta^{(t)} = \Theta^{(t-1)} - \eta f'(\Theta^{(t-1)})$
6. *until* $|f(\Theta^{(t)}) - f(\Theta^{(t-1)})| < \epsilon$
7. *return* $\Theta^{(t)}$

Note that this algorithm terminates when the change in the function f is sufficiently small. There are many other reasonable ways to decide to terminate, including:

1. Stop after a fixed number of iterations T, i.e., when $t = T$.

2. Stop when the change in the value of the parameter Θ is sufficiently small, i.e., when $|\Theta^{(t)} - \Theta^{(t-1)}| < \epsilon$.

1.1.2 Gradient descent in multiple dimensions

The extension to the case of multi-dimensional Θ is straightforward. Let's assume $\Theta \in \mathbb{R}^m$, so $f : \mathbb{R}^m \to \mathbb{R}$. The gradient of f with respect to Θ is

$$\nabla_\Theta f = \begin{bmatrix} \partial f/\partial \Theta_1 \\ \vdots \\ \partial f/\partial \Theta_m \end{bmatrix}$$

The algorithm remains the same, except that the update step 5 becomes

$$\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$$

and any termination criteria that depended on the dimensionality of Θ would have to change. The easiest thing is to keep the test in line 6 as $|f(\Theta^{(t)}) - f(\Theta^{(t-1)})| < \epsilon$, which is sensible no matter the dimensionality of Θ.

1.2. Loss Functions

In statistical learning, loss functions serve as fundamental metrics for measuring model performance. They quantify the discrepancy between predicted values and ground truth, enabling model optimization and evaluation.

1.2.1 Mean Absolute Error (L1 Loss)

The Mean Absolute Error (MAE), also known as L1 Loss, is defined as the average of absolute differences between predicted values and actual observations. For a dataset with n observations, where $\hat{y}_i$ represents the predicted value and y_i represents the actual value, MAE is expressed as:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i| \tag{1}$$

MAE demonstrates particular utility in scenarios with potential outliers, as it penalizes deviations linearly, making it more robust to extreme values. The absolute value operation ensures that positive and negative deviations do not cancel each other during aggregation.

1.2.2 Mean Square Error (L2 Loss)

The Mean Square Error (MSE), or L2 Loss, employs quadratic penalization of deviations. For a dataset with n observations, MSE is calculated as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{2}$$

MSE is particularly effective for datasets with relatively uniform distribution and minimal outliers. The quadratic nature of this loss function makes it especially sensitive to larger deviations, making it optimal for scenarios where outliers are genuine data points rather than noise.

1.2.3 Root Mean Square Error (RMSE)

The Root Mean Square Error represents a variant of MSE that provides measurements in the same units as the target variable. It is defined as:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2} \tag{3}$$

RMSE maintains the same fundamental properties as MSE but offers improved interpretability due to its scale alignment with the original data. Like MSE, it is particularly sensitive to outliers and is most appropriate for normally distributed data.

1.3. Types of Gradient Descent

Gradient Descent comes in three main types: Batch Gradient Descent, Stochastic Gradient Descent, and Mini-Batch Gradient Descent. Each method has its own way of calculating the gradient, affecting the speed and stability of convergence.

1.3.1 Batch Gradient Descent

In Batch Gradient Descent (BGD), we use the entire dataset to compute the gradient of the cost function. Given an objective function $J(\Theta)$, Batch Gradient Descent updates the parameters as follows:

$$\Theta := \Theta - \eta \nabla J(\Theta),$$

where η is the learning rate and $\nabla J(\Theta)$ represents the gradient calculated over all training examples. This approach gives stable convergence but is slow, especially for large datasets, as it requires processing all data at each step.

1.3.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) takes a different approach by updating the parameters after each training example, rather than the entire dataset. For the i-th training example, the update rule becomes:

$$\Theta := \Theta - \eta \nabla J(\Theta; x^{(i)}, y^{(i)}),$$

where $(x^{(i)}, y^{(i)})$ is the i-th training example and its label. Since updates are done on each example, SGD introduces more variability in the path toward the minimum. This can make convergence faster but less stable, often "bouncing" around the minimum. However, this variability can also help the algorithm escape local minima, making it useful for complex models.

1.3.3 Mini-Batch Gradient Descent

Mini-Batch Gradient Descent combines ideas from both BGD and SGD. It calculates the gradient using small random subsets of the data called "mini-batches." For a mini-batch of size m, the update rule is:

$$\Theta := \Theta - \eta \frac{1}{m} \sum_{j=1}^{m} \nabla J(\Theta; x^{(j)}, y^{(j)}),$$

where $(x^{(j)}, y^{(j)})$ are examples within the mini-batch. This approach balances stability and speed, as mini-batches reduce the noise of SGD without the full computational load of BGD. Mini-Batch Gradient Descent is often used in training neural networks, where it can take advantage of modern hardware like GPUs.

2. Methodology

This section outlines the methodology employed in the study, encompassing data generation, model construction, and optimization approach. We begin by describing the data generation process, followed by the mathematical formulation and implementation of gradient descent for parameter estimation.

2.1. Data Generation

We have generated or own dataset that consists of values generated from a linear relationship with added noise. Given x values in the range $[-100, 400]$ with increments of 5, the corresponding y values are calculated using:

$$y = 0.2x - 5 + \epsilon \tag{4}$$

where ϵ is random noise uniformly distributed in the range $[-18, 19]$. This noise term introduces variability to the data, making it more realistic by deviating from the exact linear relationship. The table below shows the data generated.

S.No	x	y
1	-100	-33.0
2	-95	-38.0
3	-90	-28.0
4	-85	-12.0
5	-80	-37.0
6	-75	-15.0
7	-70	-37.0
⋮	⋮	⋮
95	365	83.0
96	370	61.0
97	375	89.0
98	380	60.0
99	385	69.0
100	390	78.0
101	395	75.0

Table 1: First 7 and Last 7 Values of the Dataset

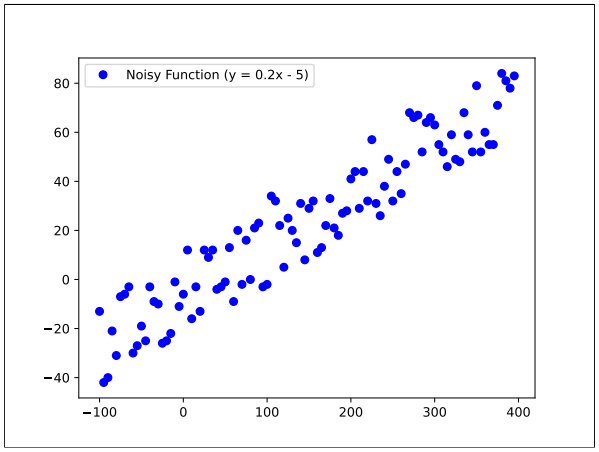

Figure 1: Generated data.

2.2. Model

To model the data, we assume a linear relationship represented by:

$$y = ax + c, \tag{5}$$

where a and c are parameters that we seek to estimate using gradient descent. We initialize these parameters randomly, with the objective of optimizing them to minimize the error between the model's predictions and the actual data.

2.3. Gradient Descent Optimization

The optimization objective is the Mean Squared Error (MSE), defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2, \tag{6}$$

where $\hat{y}_i$ denotes the predicted value, and y_i represents the actual value from the dataset. The al-

gorithm iteratively updates the parameters a and c using the gradients of the MSE until convergence is achieved.
The optimized parameters obtained through this process were approximately $a = 0.2046$ and $c = -4.3583$. These parameters result in a model that closely fits the generated data.

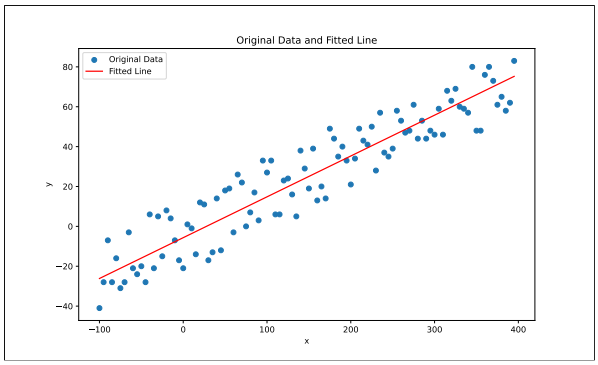

Figure 2: Best Fit Line obtained for the generated data.

The figure above illustrates the path taken by the gradient descent optimization process on the error surface, showing how the parameters converge to their optimal values.

3. Results

This section summarizes the key outcomes of the optimization process.

3.1. Optimization Summary

1. **Total Steps:** 330 iterations
2. **Varying Parameters:** 2 (parameters a and c)
3. **Final Parameters:** $a = 0.2046$, $c = -4.3583$
4. **Accuracy Improvement:**
   (a) Initial Cost: 3.0302
   (b) Final Cost: 0.1216 (Mean Squared Error)
   (c) Accuracy: 95.97%

3.2. Parameter Estimates

Table 2 lists the values of a, c, and the cost function at selected iterations:

Iteration	a	c	Cost
0	-0.7610	0.1571	3.0302
30	0.0109	0.0857	0.9871
60	0.4319	0.0467	0.3791
90	0.6616	0.0255	0.1982
120	0.7869	0.0139	0.1444
150	0.8552	0.0076	0.1283
180	0.8925	0.0041	0.1236
210	0.9128	0.0023	0.1222
240	0.9239	0.0012	0.1217
270	0.9300	0.0007	0.1216
300	0.9333	0.0004	0.1216
330	0.9351	0.0002	0.1216

Table 2: Parameter values a, c, and cost across selected iterations

3.3. Error Surface Visualization

Figure 3 shows the error surface for the proposed function.

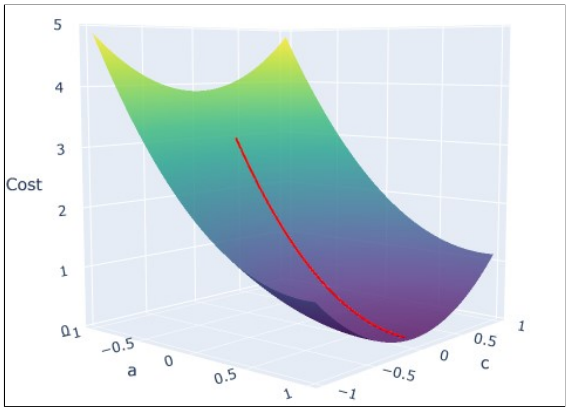

Figure 3: Error surface for the proposed function.

The red line indicates the trajectory of the optimization process, illustrating how the cost function decreased over iterations.

4. Conclusion

In this study, we have demonstrated the effectiveness of gradient descent in optimizing a linear regression model with noisy data. We observed that the algorithm yielded parameter estimates of $a = 0.2046$ and $c = -4.3583$, resulting in an impressive accuracy of approximately 95.97%. The visualization of the error surface and the optimization path effectively highlighted the convergence behavior of the algorithm.

5. Acknowledgements

We are grateful to Dr. Shivdutt Sharma, Discrete Mathematics Professor, IIIT Una for assigning this

project of "Gradient Descent" and his constant facilitation of guidance. We appreciate the opportunity to delve into this topic and the valuable resources[1] he provided, which greatly enriched our learning experience.

References

[1] MIT. Lecture notes 6.390, 2022.