Technical Report: Gradient Descent Optimization for Wine Quality Prediction

**Course: Mathematics and Statistics for Data Science.**

**Submitted by: Harpreet Kaur**

**Date:05/03/2025**

# 1. Introduction

**Goal of the Project**  
The goal of this project is to implement and compare various gradient descent variants for optimizing a linear regression model. The Wine Quality Dataset was chosen for this project as it is a regression task, where we aim to predict wine quality scores based on physicochemical properties of the wine.

**Importance of Gradient Descent in Machine Learning**  
Gradient Descent is a fundamental optimization technique in machine learning, used to minimize the cost function. It iteratively updates the model’s parameters by calculating gradients (partial derivatives) of the cost function with respect to the model parameters. Understanding the behavior of gradient descent variants is essential for improving model convergence, speed, and stability.

**Dataset Description**  
The Wine Quality Dataset contains 1,599 samples with 11 physicochemical properties of red wines (e.g., fixed acidity, volatile acidity, citric acid, etc.), and the goal is to predict the wine quality score, which is a continuous variable ranging from 0 to 10.

# 2. Methodology

**Feature Normalization**  
To ensure that the gradient descent converges more efficiently, the features were standardized using StandardScaler from scikit-learn. This scaling operation transforms the dataset into a zero-mean and unit-variance format.

**Cost Function**  
We used the Mean Squared Error (MSE) as the cost function for this regression task. The MSE measures the average squared difference between the predicted and actual wine quality scores.  
J(θ) = (1/2m) \* ∑(yi - ȳi)^2  
Where:  
 J(θ) is the cost function  
 m is the number of samples  
 yi is the true value  
 ȳi is the predicted value

**Implemented Methods**  
We implemented three gradient descent variants:  
1. \*\*Batch Gradient Descent (BGD) : The algorithm computes the gradient using the entire dataset.  
2. \*\*Stochastic Gradient Descent (SGD): The gradient is computed for each data point, resulting in faster but noisier updates.  
3. \*\*Mini-batch Gradient Descent (MBGD):\*\* A compromise between BGD and SGD, where the gradient is computed on small batches of data.

Learning Rate Schedules  
We explored different learning rate schedules to improve the efficiency of convergence:  
- \*\*Constant Learning Rate\*\*  
- \*\*Time-based Decay:\*\* Learning rate decreases over time as lr / (1 + decay\_rate \* epoch)  
- \*\*Step Decay:\*\* The learning rate is dropped by a factor every few epochs.

# 3. Results

**Convergence Plots**We plotted the cost function against the number of iterations for each gradient descent variant. This allows us to visualize the convergence of each method.

**Comparison of Final Costs**  
We evaluated the final cost after 1000 iterations for each gradient descent method and observed the following:  
- \*\*BGD\*\* typically showed slower convergence but a stable reduction in cost.  
- \*\*SGD\*\* converged faster initially but exhibited more oscillations in the cost curve.  
- \*\*MBGD\*\* provided a balanced approach, converging faster than BGD with less noise than SGD.

A graph with a blue line

Description automatically generated

**Performance Across Different Learning Rates**  
We experimented with various learning rates (0.001, 0.01, 0.1) and analyzed their impact:  
- A \*\*too high learning rate\*\* caused the cost to diverge.  
- A \*\*too low learning rate\*\* resulted in slow convergence.  
- An \*\*optimal learning rate\*\* balanced speed and stability.

**Table: Convergence Speed and Stability**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Final Cost | Convergence Speed | Stability |
| BGD | 0.256 | Slow | Stable |
| SGD | 0.259 | Fast | Unstable |
| MBGD | 0.258 | Moderate | Balanced |

# 4. Discussion

When Each Method is Suitable  
- \*\*BGD\*\* is suitable for smaller datasets where we need precise and stable updates.  
- \*\*SGD\*\* works well for large datasets where speed is essential, but it may require more tuning to mitigate the noise.  
- \*\*MBGD\*\* is ideal for situations where we need a balance between computational efficiency and stability, and it's widely used in practice.

**Challenges Encountered**  
Selecting the right learning rate and batch size was crucial to achieve convergence without overshooting or slow progress.  
SGD's noisy updates required additional techniques like learning rate schedules to stabilize the convergence.

**Impact of Learning Rate and Batch Size**  
The learning rate and batch size directly impacted convergence speed and stability. An optimal learning rate improved both, while too large a batch size caused the algorithm to converge slowly. Smaller batch sizes sped up learning but increased computational cost per iteration.

# 5. Conclusion

**Which Method Performed Best and Why**  
- \*\*Mini-batch Gradient Descent (MBGD)\*\* was the most effective method overall, offering a good balance between speed and stability.  
- \*\*Batch Gradient Descent (BGD)\*\* performed well but was slower compared to MBGD.  
- \*\*Stochastic Gradient Descent (SGD)\*\* converged quickly in the beginning but exhibited high variance in the cost function.

**Recommendations for Future Improvement**  
- Incorporating advanced techniques like \*\*Adam optimizer\*\* could improve the speed and stability of training.  
- Tuning the \*\*decay rate\*\* and \*\*learning rate schedules\*\* could further enhance convergence efficiency.