UNIVERSITY OF CALIFORNIA

Los Angeles

Sampling is All You Might Need

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Applied Statistics and Data Science

by

Yik Cheung Jeffrey Chan

2025

ABSTRACT OF THE THESIS

Sampling is All You Might Need

by

Yik Cheung Jeffrey Chan

Master of Applied Statistics and Data Science

University of California, Los Angeles, 2025

Professor Guido F. Montúfar Cuartas, Chair

We propose two gradient-free optimization methods to solve Ordinary Least Squares linear regression problems, focusing on the use of Monte Carlo and Bagging Monte Carlo techniques. These methods leverage multiple core processors to iteratively generate sample betas within a restricted search space and record the mean squared error. While exploring the parameter space, our results do not conclusively align with the double descent narrative commonly discussed in prior literature, which primarily links this phenomenon to gradient descent methods and their implicit norm-minimization bias. Our experimental comparisons reveal that these gradient-free approaches yield competitive performance metrics, including mean squared error, R^2, and L^2 Norm, which are on par with those achieved by traditional Ordinary Least Squares and Gradient Descent methods.

The thesis of Yik Cheung Jeffrey Chan is approved.

Xiaowu Dai

Yingnian Wu

David Anthony Zes

Guido F. Montúfar Cuartas, Committee Chair

University of California, Los Angeles

2025

To Natalie Chau, Mei Mei, and other family members...

for all the countless support and ineffaceable and invaluable memory.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Gradient based and gradient free optimization are the two primary classes of optimization methods that we can employ in machine learning. In machine learning, it is common to use gradient based optimization specifically gradient descent or its variants. However, we suggest that gradient free optimizers might be undervalued and that they can have several advantages. For example, gradient free optimizers do not require smoothness or differentiability of an objective function to perform optimization and may have a better chance to avoid poor local minima [5, 7, 9]. Unlike gradient free optimizer, the learning rate in gradient based optimization is a critical hyperparameter that has to be adjusted with care for the algorithm to converge and reach a good solution [5, 11].

One of the driving ideas in contemporary machine learning and in particular neural networks is that the parameter optimization procedures that are used for training play a key role in enabling generalization. Specifically, parameter optimization by gradient descent serves as regularizer that implicitly biases the solutions in a way that is beneficial for generalization. For example, in the case of overparametrized linear models, optimizing the squared error loss by gradient descent is biased towards the solution that is closest to initialization. This means that for zero initialization the algorithm will return the minimum norm solution, which is known to have good generalization properties.

Some recent works have suggested that the driving factor for the good generalization performance of overparametrized models is not necessarily the specific parameter optimization

procedure but rather the fact that the set of good solutions in parameter space has a large volume [4]. Certainly the metric on parameter space influences both the volume of a particular set of solutions and also gradient descent. Towards gaining clarity about this, we investigate what happens if the parameter optimization is conducted in a gradient free manner via randomly sampling values of the search variable.

Double Descent is a phenomenon that is observed in machine learning models as the number of model parameters increases relative to the number of training data points [2]. Here one observes that as the number of parameters increases, the test error first decreases and then increases, as one would expect from traditional statistical learning theory, but then, surprisingly, as the number of parameters continues to increase, the test error decreases again. Double Descent has been characterized in particular for the case of ordinary linear regression [18].

In the present work, we utilize multi-core computational resources to execute two novel gradient free optimizers, Monte Carlo (MC) and Bagging Monte Carlo (BMC), to solve linear regression problems. We compare the performance results with Ordinary Least Squares (OLS) and Gradient Descent (GD) which serve as our baseline. In the meantime, we will capture the Double Descent events by varying the number of data points of the input matrix size, k where $k \in \mathbb{N}$, to create the over and under parametrized scenarios. In addition, we will use OLS, GD, MC, and BMC to build a predictive linear regression model using a ratio of 70% : 30% training and testing data split. Lastly, we report its performances using train and test mean squared error, train and test R^2, train and test L^2 Norm, number of cores used, and the optimizers' configurations.

1.1 Contributions

This research advances the field of machine learning optimization through the introduction of innovative gradient-free methods tailored for solving linear regression challenges. The

key contributions of this work are crafted to demonstrate both theoretical and practical advancements:

- **Development of Novel Optimization Techniques:** We introduce two gradient-free approaches—Monte Carlo (MC) and Bagging Monte Carlo (BMC). These techniques leverage the power of multi-core processors to perform extensive parameter searches within a defined space, significantly enhancing the computational efficiency and robustness of model training processes.

- **Empirical Analysis of the Double Descent Phenomenon:** Through rigorous experimental design, our study not only illustrates how MC and BMC manifest the Double Descent phenomenon but also contrasts these observations with those from traditional methods such as Ordinary Least Squares and Gradient Descent. This comparison sheds light on the conditions under which Double Descent occurs, thereby enriching the current understanding of model complexity and training dynamics.

- **Comprehensive Performance Evaluation:** The efficacy of MC and BMC is evaluated against conventional optimization algorithms. Our results reveal that these new methods achieve comparable, if not superior, performance metrics such as train and test R^2, mean squared error (MSE), and L^2 Norm. Such findings underscore the potential of gradient-free methods to serve as viable alternatives to gradient-based optimization in certain regression tasks.

- **Practical Implications and Scalability:** The implementation of MC and BMC demonstrates their scalability and adaptability to different data regimes and hardware configurations. This aspect is crucial for practitioners seeking efficient solutions in environments with varying computational resources.

These contributions are pivotal as they not only enhance the algorithmic toolkit available to data scientists but also provide deeper insights into the behavior of learning models under

various optimization regimes. The findings from this research could potentially inform future developments in machine learning optimization, particularly in how we understand and leverage the capabilities of gradient-free algorithms in broader applications.

1.2 Related Work

The exploration of optimization methods in machine learning is vast and variegated, with a rich tapestry of research focusing on both gradient-based and gradient-free techniques. Our review of the literature reveals a dynamic field where the understanding of phenomena such as Double Descent is still evolving and where gradient-free methods are gaining recognition for their robustness and versatility.

Understanding Double Descent: The phenomenon of Double Descent has been scrutinized extensively in recent literature. Belkin et al. (2019) provided a foundational perspective by identifying the occurrence of Double Descent in highly parametrized models, challenging traditional beliefs about overfitting [2]. Nakkiran et al. (2019) expanded on this by demonstrating that Double Descent could occur across various model families, not just deep neural networks, suggesting a more ubiquitous nature of the phenomenon [12, 13]. These studies underscore the complexity of model training dynamics in modern machine learning, setting the stage for our exploration of how gradient-free methods interact with these dynamics.

Gradient-Free Optimizations: While the majority of optimization research has historically centered on gradient-based methods due to their intuitive implementation and effective results in large-scale applications, gradient-free methods have carved out a niche due to their applicability in non-differentiable scenarios. Works by Chiang et al. (2023) and Schaeffer et al. (2024) have begun to peel back the layers on how gradient-free approaches could sidestep some of the pitfalls that ensnare gradient-based methods, particularly in handling noisy or incomplete data landscapes [4, 18].

Comparative Studies and Applications: Our work is particularly inspired by the comparative studies of Buschjäger and Morik (2021), who argued against the existence of Double Descent in Random Forests, suggesting that certain algorithmic frameworks inherently resist such phenomena due to their structural properties [3]. This discourse opens new avenues for investigating how different optimization frameworks—especially less traditional, gradient-free methods—behave under varying data conditions.

In bridging these discussions, our research does not merely replicate existing findings but seeks to expand the dialogue about optimization methods in machine learning. By integrating and building upon these foundational studies, we aim to illuminate the practical and theoretical implications of employing gradient-free methods like Monte Carlo and Bagging Monte Carlo in scenarios traditionally dominated by gradient-based approaches. This not only enriches the field's understanding of Double Descent but also enhances the toolkit available to practitioners dealing with complex, real-world datasets.

Chapter 2

Preliminaries

In this paper, we are comparing the performance of the following optimizers to solve a linear regression problem: Ordinary Least Squares (OLS), Gradient Descent (GD), Monte Carlo (MC) and Bagging Monte Carlo (BMC). Linear regression is

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}. \tag{2.1}$$

In equation 2.1, $\boldsymbol{y}$ represents the vector of dependent variables, $\boldsymbol{X}$ is the design matrix with N rows representing the number of observations and p columns representing the number of features (independent variables), $\boldsymbol{\beta}$ is the vector of coefficients to be estimated, and $\boldsymbol{\epsilon}$ is the vector of random errors, assumed to be normally distributed with mean zero and constant variance σ^2, i.e., $\boldsymbol{\epsilon} \sim N(0, \sigma^2\mathbb{I})$.

The explicit representation of equation 2.1 is as follows:

$$\boldsymbol{y} = \begin{bmatrix} 1 & x_{1,2} & \cdots & x_{1,p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,2} & \cdots & x_{N,p} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_N \end{bmatrix} \tag{2.2}$$

where $\boldsymbol{y}$ is a vector of actual value, $\boldsymbol{\beta}$ is the weight vector, $\boldsymbol{X}$ is a N rows by p columns matrix containing input data points as its rows, and $\boldsymbol{\epsilon}$ is a noise vector with $\boldsymbol{\epsilon} \sim N(0, \sigma\mathbb{I})$. After performing equation 2.2, we will have a vector of response, $\boldsymbol{y}$. In addition, we use the

mean squared error (MSE) as our loss function,

$$\ell(\hat{Y}) = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2, \tag{2.3}$$

where $\hat{Y}_i$ is the predicted value and Y_i is the actual value. It is very common to use the OLS to arrive the optimal coefficient, $\hat{\boldsymbol{\beta}}$ that is the value of $\boldsymbol{\beta}$ in equation 2.1 that minimizes equation 2.3. The essence of OLS is projecting the observed data points onto a subspace defined by the independent variable. In the simple overdetermined case where $\boldsymbol{X}^\top\boldsymbol{X}$ is invertible the minimizer of the squared error loss is given by

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}. \tag{2.4}$$

Meanwhile, GD works by iteratively adjusting the parameters in the opposite direction of the gradient of the loss where we will have an initial guess $\boldsymbol{\beta}$ when $i = 0$. GD can be expressed as follows

$$\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i-1} - \alpha\nabla f(\boldsymbol{\beta}_{i-1}). \tag{2.5}$$

In the case of MSE, $f(\boldsymbol{\beta}) = \ell(\hat{Y}(\boldsymbol{\beta}))$. In addition, $\bar{y}$ is

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{2.6}$$

R^2 is defined as

$$R^2 = 1 - \frac{\sum_i^N (y_i - \hat{y}_i)^2}{\sum_i^N (y_i - \bar{y})^2}. \tag{2.7}$$

Lastly, the Euclidean norm of an n-vector $\boldsymbol{x}$ is

$$\| \boldsymbol{x} \|_2 = \left\{ \sum_i^n |x_i|^2 \right\}^{\frac{1}{2}}. \tag{2.8}$$

2.1 Double Descent

Double Descent is a phenomenon observed in machine learning where the performance of a model on a validation or test set first gets worse as the model complexity increases (overfitting near the interpolation threshold), and then surprisingly improves even as the complexity continues to grow.

In our notations, $X^{N\times p}$ represents a matrix with N rows and p columns, where N is the number of observations and p is the number of features. This matrix format is used throughout to denote the dimensions of matrices relevant to our discussion.

This phenomenon can be represented through the following figure which illustrates how test error evolves as model complexity increases, as shown in Figure 2.1:

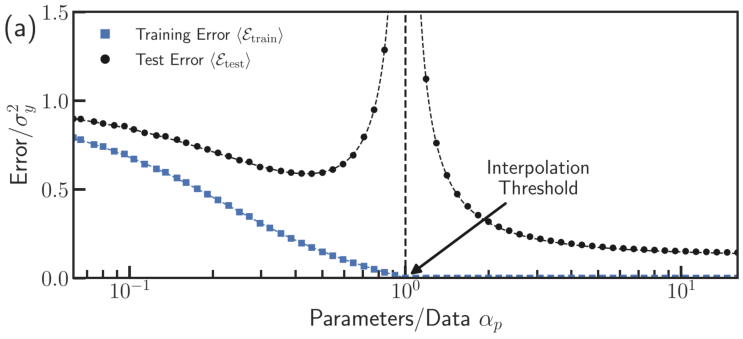

Figure 2.1: Double Descent Curve [16]. Adapted from J. W. Rocks and P. Mehta, “Memorizing without overfitting: Bias, variance, and interpolation in over-parameterized models," available at arXiv:2010.13933.

The Double Descent curve typically shows a "U" shape followed by a second decreasing phase as the model's capacity becomes significantly large relative to the complexity of the task. The first descent corresponds to the traditional U-shaped bias-variance tradeoff, and the second descent occurs when the model enters the overparameterized regime, where models can fit the training data perfectly and still generalize well to new data. This counterintuitive behavior has been noted in settings including but not limited to neural networks, deep trees, and certain linear regression models.

2.2 Gradient Descent

Gradient Descent is a method to find the minimum of a function by iteratively moving towards the steepest descent direction defined by the negative of the gradient. In the context of linear regression, the Gradient Descent algorithm updates the parameters in the opposite direction of the gradient of the loss function with respect to the parameters.

The update rule for the parameters can be expressed as:

$$\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i-1} - \alpha \nabla f(\boldsymbol{\beta}_{i-1}), \tag{2.9}$$

where $\boldsymbol{\beta}_i$ is the vector of coefficients at iteration i, α is the learning rate, and $\nabla f(\boldsymbol{\beta}_{i-1})$ is the gradient of the loss function f with respect to $\boldsymbol{\beta}$ at iteration i.

This process is repeated until convergence, typically when the change in loss is below a predefined threshold or after a fixed number of iterations.

2.3 Monte Carlo Method

The Monte Carlo method is a robust statistical simulation technique widely used across various fields such as algebraic computations, modeling natural processes, and in quantitative disciplines like finance and statistics. This method involves the generation of a significant number of random samples to approximate complex mathematical solutions, making it invaluable for scenarios where analytical or deterministic solutions are impractical or impossible to derive.

The fundamental principle of the Monte Carlo method is to utilize random sampling to estimate properties of a distribution or to approximate the solution to a mathematical problem. This is particularly effective in high-dimensional spaces where the volume of the space explodes exponentially with the number of dimensions, making traditional analysis

Algorithm 1 Capturing the Double Descent Events Pseudocode

Required: $\boldsymbol{X}^{N\times p}, \boldsymbol{Y}^{N\times 1}$
Initialization: $O \leftarrow \varnothing,\ G \leftarrow \varnothing,\ M \leftarrow \varnothing,\ E \leftarrow \varnothing,\ i = 1$
Step size: α, usually set to a small value like 0.01 or adjusted dynamically
Stopping criteria: Threshold for minimal improvement in MSE, typically 10^{-6}
while $k \leq N$ **do**
  while $i \leq 100$ **do**
    $\boldsymbol{A}^{k\times P} \leftarrow \text{sample}(\boldsymbol{X}, \boldsymbol{Y}, \text{size} = k)$
    $\boldsymbol{X}_{train}^{\lfloor k\times 0.7\rfloor\times p}, \boldsymbol{X}_{test}^{\lfloor k\times 0.3\rfloor\times p}, \boldsymbol{Y}_{train}^{\lfloor k\times 0.7\rfloor\times p}, \boldsymbol{Y}_{test}^{\lfloor k\times 0.3\rfloor\times p} \leftarrow \text{splitData}(\boldsymbol{A}^{k\times p})$
    $\hat{\boldsymbol{\beta}}_{OLS} \leftarrow \text{OLS}(\boldsymbol{X}_{train}^{\lfloor k\times 0.7\rfloor\times p}, \boldsymbol{Y}_{train}^{\lfloor k\times 0.7\rfloor\times p})$
    $O \leftarrow (\text{trainLoss}(\hat{\boldsymbol{\beta}}_{OLS}), \text{testLoss}(\hat{\boldsymbol{\beta}}_{OLS}))$
    Initialization for GD: $\boldsymbol{\beta}_0 \leftarrow$ Random or Zero initialization
    $\hat{\boldsymbol{\beta}}_{GD} \leftarrow \text{GradientDescent}(\boldsymbol{X}_{train}^{\lfloor k\times 0.7\rfloor\times p}, \boldsymbol{Y}_{train}^{\lfloor k\times 0.7\rfloor\times p}, \alpha, \text{Stopping criteria})$
    $G \leftarrow (\text{trainLoss}(\hat{\boldsymbol{\beta}}_{GD}), \text{testLoss}(\hat{\boldsymbol{\beta}}_{GD}))$
    $\hat{\boldsymbol{\beta}}_{MC} \leftarrow \text{MonteCarlo}(\boldsymbol{X}_{train}^{\lfloor k\times 0.7\rfloor\times p}, \boldsymbol{Y}_{train}^{\lfloor k\times 0.7\rfloor\times p})$
    $M \leftarrow (\text{trainLoss}(\hat{\boldsymbol{\beta}}_{MC}), \text{testLoss}(\hat{\boldsymbol{\beta}}_{MC}))$
    $\hat{\boldsymbol{\beta}}_{BMC} \leftarrow \text{BaggingMonteCarlo}(\boldsymbol{X}_{train}^{\lfloor k\times 0.7\rfloor\times p}, \boldsymbol{Y}_{train}^{\lfloor k\times 0.7\rfloor\times p})$
    $E \leftarrow (\text{trainLoss}(\hat{\boldsymbol{\beta}}_{BMC}), \text{testLoss}(\hat{\boldsymbol{\beta}}_{BMC}))$
    $i \leftarrow i + 1$
  end while
  $k \leftarrow k + \delta$ ▷ δ is the increment in k to gradually increase task complexity
end while
return R ▷ Returns a record of the training and testing losses

unfeasible.

The Monte Carlo method can be mathematically described as follows: Suppose we want to estimate an integral $\int_D g(x)dx$

$$\hat{g} = m(D)\frac{1}{N}\sum_{i=1}^{N} g(x_i) \quad \forall i \in \{1, \cdots, n\} \tag{2.10}$$

where x_i are drawn uniformly over D, and $m(D)$ is the measure or mass of the support , D, commonly $\int_D dx$. By the law of large numbers, as N, the number of samples, increases, $\hat{g}$ converges to $\int g(x)dx$.

The statistical properties such as the mean μ and variance σ^2 of the underlying process can also be estimated. For random variables $X_1, X_2, \ldots, X_n$ independently and identically

distributed with mean $\mu = \mathbb{E}[X_i]$ and variance $\sigma^2 = \mathrm{Var}[X_i]$, the sample mean

$$M_n = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{2.11}$$

approximates μ, and by the Central Limit Theorem, the distribution of M_n converges to a normal distribution centered at μ with variance $\frac{\sigma^2}{n}$ as $n \to \infty$.

The Monte Carlo estimations rely heavily on the law of large numbers which asserts that the sample average converges in probability towards the expected value as the sample size grows:

$$\lim_{n\to\infty} \mathbb{P}(|M_n - \mu| \geq \epsilon) = 0, \tag{2.12}$$

for any $\epsilon > 0$, ensuring the consistency of the Monte Carlo estimates.

These principles allow the Monte Carlo method to provide reliable estimations even in complex scenarios, supporting its application in fields like mathematical finance, applied statistics, and artificial intelligence for problems that are otherwise analytically intractable [6, 10, 17, 20, 21].

2.4 Bagging Method

The ensemble method, particularly the Bagging (Bootstrap Aggregating) approach, is designed to improve the stability and accuracy of machine learning algorithms by combining multiple models to reduce variance and avoid overfitting [8, 14, 15]. Commonly used in decision tree algorithms and various clustering methods, Bagging is particularly effective in environments where the prediction model is highly sensitive to the variability in its training dataset [1, 19].

Bagging involves generating multiple versions of a predictor by using a random sampling with replacement technique on the training data, training a separate model on each sample, and then averaging the predictions. The ensemble's final output is derived by averaging the predictions from all the individual models to improve the generalization error of the resultant

model.

In Bagging, multiple datasets are created from the original dataset by bootstrap sampling. Each of these datasets is used to train a model $\hat{f}^{(b)}$, where b represents the index of the individual model in the ensemble. The variable $\boldsymbol{x}$ denotes an input vector from the dataset, and B indicates the total number of bootstrap samples or models in the ensemble. The aggregated prediction, $\hat{f}(\boldsymbol{x})_B$, is calculated as follows:

$$\hat{f}(\boldsymbol{x})_B = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{(b)}(\boldsymbol{x}), \tag{2.13}$$

where $\hat{f}^{(b)}(\boldsymbol{x})$ is the prediction made by the b-th model on input $\boldsymbol{x}$. This aggregation helps to mitigate the variance between the predictions of individually trained models, leading to more robust and stable performance across diverse datasets.

The efficacy of Bagging comes from its ability to reduce variance and provide a smoothing effect over the training data, which is particularly beneficial in models that are prone to overfitting, such as decision trees [8, 14, 15].

Chapter 3

Methodology

3.1 Monte Carlo Sampling

Monte Carlo method is a computational technique that generates random points (samples) to execute a numerical experiment for N amount of times. When we exhaust all the samples in the universe, we will arrive to the optimal solution. In this paper, we use C amount of cores to perform Monte Carlo simulation. Each core will run Q amount of individual simulations, for a total of $Q \times C$ samples of the vector $\boldsymbol{\beta}^{p\times 1}$. Next, we split the input data matrix $\boldsymbol{X}$ into training and testing sets, $(\boldsymbol{X}_{train}^{\lfloor k\times 0.7\rfloor \times p}, \boldsymbol{Y}_{train}^{\lfloor k\times 0.7\rfloor \times p})$, with a ratio of 70% and 30%, respectively, using *sklearn* package. The selection of the range for the betas was based on initial exploratory analysis and heuristic adjustments to ensure the optimization process captures a broad spectrum of potential solutions, thereby increasing the robustness of the model findings.

Algorithm 2 Search Space Function Pseudocode

Required: $\boldsymbol{X}, \alpha = 3$
$lower \leftarrow \lfloor min(\boldsymbol{X}_{tr}^{\lfloor N\times 0.7\rfloor \times p})/\alpha \rfloor$
$upper \leftarrow \lfloor max(\boldsymbol{X}_{tr}^{\lfloor N\times 0.7\rfloor \times p})/\alpha \rfloor$
return $lower$, $upper$

Once we have generated the sample $\boldsymbol{\beta}$, $\boldsymbol{\beta} \sim U(lower, upper)$ and the lower and upper values are retrieved from algorithm 2, we compute the loss using equation 2.3 for each sample $\boldsymbol{\beta}$ with the same set of input data $(\boldsymbol{X}_{train}^{\lfloor N\times 0.7\rfloor \times p}, \boldsymbol{Y}_{train}^{\lfloor N\times 0.7\rfloor \times p})$. The following is the pseudo

code for the Monte Carlo method and the following is the basic notation: $\boldsymbol{X}_{train}^{\lfloor N\times 0.7\rfloor \times p}$ as train input matrix, $\boldsymbol{Y}_{train}^{\lfloor N\times 0.7\rfloor \times p}$ as train output vector, $\boldsymbol{X}_{test}^{\lfloor N\times 0.3\rfloor \times p}$ as test input matrix, $\boldsymbol{Y}_{test}^{\lfloor N\times 0.3\rfloor \times p}$ as test output vector, and we calculate train MSE and store it to into a variable a. Lastly, the algorithm will store the sample beta and both MSEs as a tuple into the set R.store the sample beta and both MSEs as a tuple into the set R.

Algorithm 3 Monte Carlo Optimizer Pseudocode

Required: $Q,\ \boldsymbol{X}_{tr}^{\lfloor N\times 0.7\rfloor \times p},\ \boldsymbol{Y}_{tr}^{\lfloor N\times 0.7\rfloor \times p}$
Initialization: $R \leftarrow \varnothing,\ i = 1$
while $i \leq Q$ **do**
  Initialization: $\hat{\boldsymbol{\beta}}_{MC}^{p\times 1} \leftarrow \varnothing, a \leftarrow 0$
  $lower,\ upper \leftarrow search\_space(\boldsymbol{X}_{train}^{\lfloor N\times 0.7\rfloor \times p})$
  $\hat{\boldsymbol{\beta}}_{MC}^{p\times 1} \leftarrow generate\_betas(lower,\ upper)$
  $a \leftarrow MSE(\boldsymbol{X}_{train}^{\lfloor N\times 0.7\rfloor \times p},\ \boldsymbol{Y}_{train}^{\lfloor N\times 0.7\rfloor \times p})$
  $R \leftarrow R \cup \{(\hat{\boldsymbol{\beta}}_{MC}^{p\times 1},\ a)\}$
  $i \leftarrow i + 1$
end while
return R

Lastly, to use the standard multi core procedure, we use the built in multi processing package from Python to execute the algorithm 3 for C amount of times. Once all the cores are completed with the simulation, we will combine all the results and find the lowest MSE weight vector from algorithm 3 with input data ($\boldsymbol{X}_{train}^{\lfloor N\times 0.7\rfloor \times p}$, $\boldsymbol{Y}_{train}^{\lfloor N\times 0.7\rfloor \times p}$) as our final $\hat{\boldsymbol{\beta}}$.

3.2 Bagging Monte Carlo Optimizer

In our Bagging Monte Carlo optimizer, BMC, it uses the same structure as algorithm 2 and 3 with a few modifications. In our experiment, we use C cores to build C models and average C predictions as our output value.

$$\hat{f}_{bag}(\boldsymbol{x}) = \frac{1}{C}\sum_{i=1}^{C}\hat{f}^{i}(\boldsymbol{x}) \tag{3.1}$$

To generate C sets of data, we will be using *sklearn* resample function for bagging sampling with replacement argument as true and number of samples will be the amount of rows, N, of the data matrix $\boldsymbol{X}$, divided by C number of cores.

Algorithm 4 Bagging Sampling Pseudocode

Required: $\boldsymbol{X}, \boldsymbol{Y}, C$
Initialization: $\boldsymbol{X'} \leftarrow \varnothing,\ i = 1,\ n = \lfloor \frac{N}{C} \rfloor$
while $i \leq C$ **do**
    $\boldsymbol{X'} \leftarrow \boldsymbol{X'} \cup \{(resample(X,\ replacement = True,\ number\_of\_sample = n))\}$
    $i \leftarrow i + 1$
end while
return $\boldsymbol{X'}$

Once we have the Bagging Sampling function ready, we use algorithm 3 and 4 to perform the simulation and then apply equation 3.1 to calculate the predicted value.

3.3 Data Processing

In our experiments, we have used the energy and wine data sets from University of California, Irvine and California Housing data set from Sklearn. Instead of letting the optimizers do its job, we first perform a standard exploratory data analysis and transform certain features. Meanwhile, linear regression is very sensitive to the outlier and multi-collinearity; as a result, we first run the Variance Inflation Factor (VIF) and keep all the features that have a VIF less than 11.

Chapter 4

Results

4.1 Double Descent Observation

In this section, we demonstrate the result of result, algorithm 1, of the Double Descent observations by varying the input size or amount of rows, k, of the data matrix and for each k, we collect 100 mean squared error values to generate all the plots in section 4.1 where as the shaded region is the standard error of the calculated mean squared error. First, lets

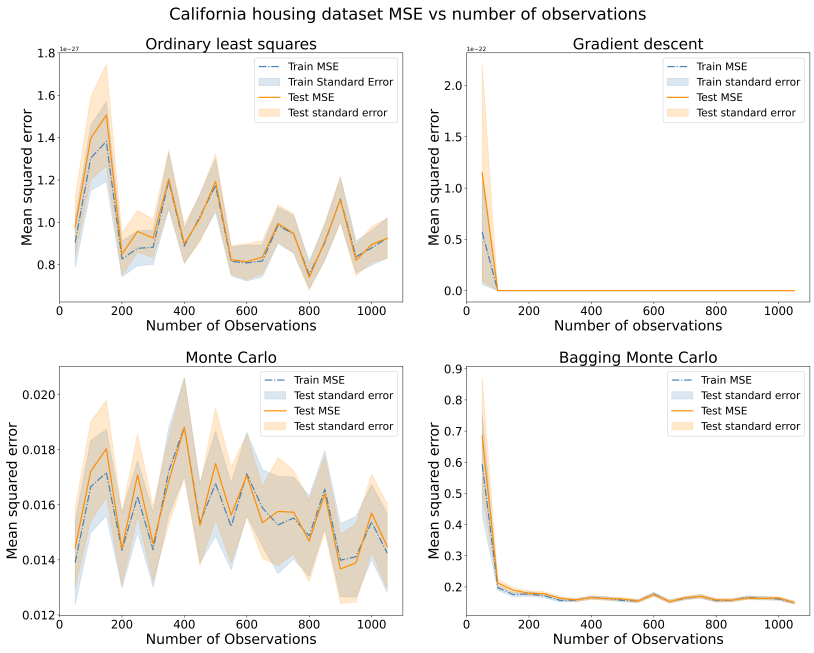

Figure 4.1: Double Descent Experiment Using California Housing Dataset

dive into the California housing data set. In figure 4.1, all the optimizers have a consistent decreasing trend across different values of k. In addition, there are no strong Single and Double Descent patterns in OLS, MC, BMC, and GD. Also, both MC and BMC optimizers achieve comparable MSE with OLS and GD. On the other hand, the BMC optimizer has the smallest standard error across all values k and its MSE quickly stabilized after k is beyond 250.

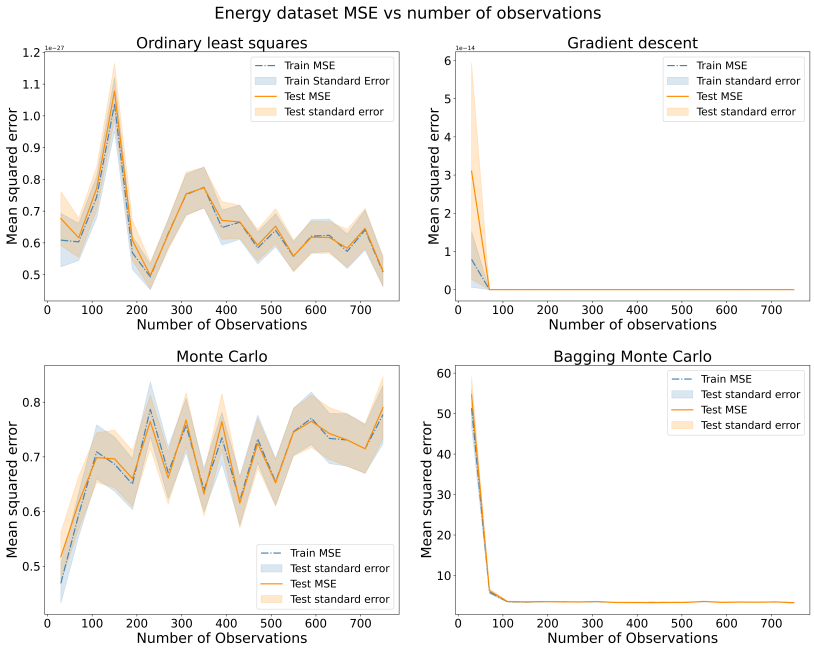

Figure 4.2: Double Descent Experiment Using Energy Dataset

In figure 4.2, none of the optimizers are showing any evidence of Single and Double Descent patterns. Moreover, OLS, GD, MC, and BMC optimizers have low and steady MSEs across all k values, whereas BMC is performing consistently when the k value exceeds 100. Meanwhile, BMC has low MSE with minimal variability that converges quickly. Lastly, MC has an interesting plot. It has the smallest MSE when the number of observations is smaller than 100. When k is greater than 100, the MSE starts to hover around 0.68. In figure 4.3, all four experiments share the same pattern and nuance on the MSE throughout the variety of value k. Moreover, BMC has the smallest standard error among other optimizers. Lastly,

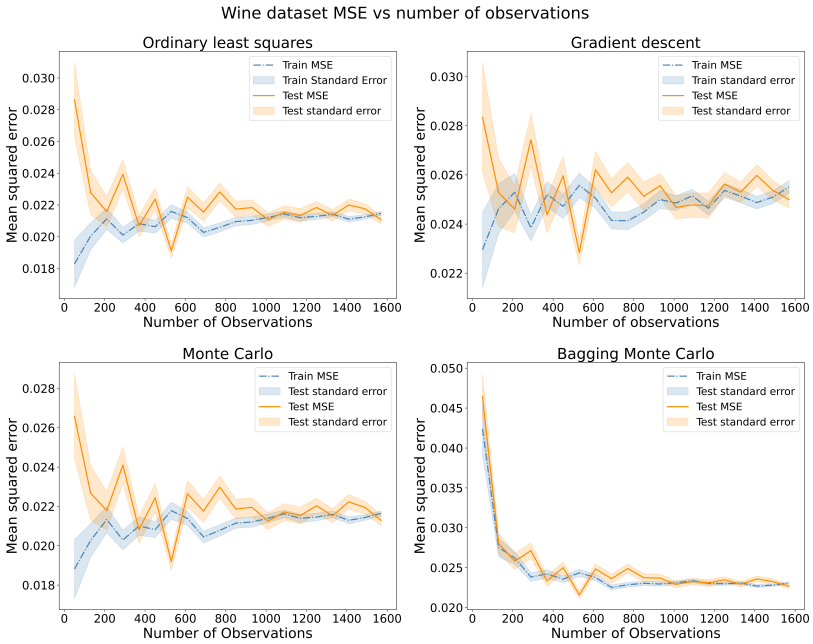

Figure 4.3: Double Descent Experiment Using Wine Dataset

the double descent experiment unable to reveal double descent phenomena, that is test error increases dramatically where the training error reaches the minimal point.

In our examination of the double descent phenomenon, we focused on varying the sample size k of the training data across multiple datasets. It is important to note that the traditional double descent curve is more pronounced when the number of training samples k includes values both smaller and larger than the dimensionality of the data. However, our experimental setup primarily involved sample sizes that were generally larger than the data dimensions, which could potentially obscure the observation of the double descent phenomenon. Future studies could benefit from a deliberate variation in k to span a range more comprehensively, potentially revealing more distinct patterns of double descent.

4.2 Performance on Real Data Set

In this section, we illuminate the Ordinary Least Squares (OLS), Monte Carlo (MC), Bagging Monte Carlo (BMC), and Gradient Descent (GD) performances in a table with three real world data sets. The performance metrics include the train and test mean squared error

Table 4.1: California Housing Dataset Performance

Optimizer	Tr ϵ	Ts ϵ	Tr R^2	Ts R^2	Tr Norm	Ts Norm	Time (s)	Cores	Itr
OLS	84.956	76.308	0.447	0.472	367.418	227.960	0	1	-
GD	84.957	76.310	0.447	0.472	367.418	227.963	0.992	1	1e5
MC	85.130	76.273	0.446	0.472	367.792	227.908	3.191	12	6e5
BMC	86.926	78.518	0.445	0.468	367.987	228.756	6.476	12	6e5

The table includes the train error (Tr ϵ), test error (Ts ϵ), train and test R^2, train and test L^2 Norm, computation time in seconds (Time), number of cores used (Cores), and the number of iterations (Itr.) performed by each optimizer. 'Itr.' denotes the total iterations performed during the optimization process, essential for achieving convergence in iterative methods like Gradient Descent.

denoted as error, train and test R^2, Time used in seconds to perform the optimization, and the optimizers' configuration. In addition, GD uses a learning rate of 0.0001 and iterates 100,000 times with an early stopping threshold at 0.000001. Lastly, we are using 70% of the data to train the model and 30% as the test set to test the model. In Table 4.1, the proposed optimizers, BMC and MC, demonstrate highly competitive results compared to the benchmarks, OLS and GD. Specifically, BMC and MC train and test R^2 differ by at most 0.40% from the OLS train and test R^2. On the other hand, the train and test MSE for BMC and MC are within 2.40% of the baseline. Furthermore, the test L^2 Norms of BMC and MC exceed the baseline by 0.35%.

In Table 4.2, proposed optimizers, BMC and MC, illustrates that the train and test R^2 deviates by at most 1.5% from the baseline, OLS. Moreover, the MC and BMC train and test MSE are insignificant. On the other hand, the test L^2 Norms of BMC and MC surpass the baseline by 16.04 and 17.45, respectively.

In Table 4.3, the proposed optimizers, BMC and MC, performance metrics are similar to

Table 4.2: Energy Dataset Performance

Optimizer	Tr ϵ	Ts ϵ	Tr R^2	Ts R^2	Tr Norm	Ts Norm	Time (s)	Cores	Itr
OLS	0.000	0.000	1.000	1.000	0.000	0.000	0	1	-
GD	0.000	0.000	1.000	1.000	0.234	0.163	0.658	1	1e5
MC	1.286	1.318	0.986	0.985	26.279	17.451	3.217	12	6e5
BMC	2.600	2.498	0.987	0.987	25.206	16.044	6.757	12	6e5

The table includes the train error (Tr ϵ), test error (Ts ϵ), train and test R^2, train and test L^2 Norm, computation time in seconds (Time), number of cores used (Cores), and the number of iterations (Itr.) performed by each optimizer. 'Itr.' denotes the total iterations performed during the optimization process, essential for achieving convergence in iterative methods like Gradient Descent.

Table 4.3: Wine Dataset Performance

Optimizer	Tr ϵ	Ts ϵ	Tr R^2	Ts R^2	Tr Norm	Ts Norm	Time (s)	Cores	Itr
OLS	0.020	0.025	0.258	0.255	4.716	3.449	0.000	1	-
GD	0.022	0.028	0.191	0.169	4.923	3.642	0.860	1	1e5
MC	0.020	0.025	0.253	0.256	4.731	3.447	3.155	12	6e5
BMC	0.021	0.026	0.255	0.260	4.723	3.438	6.496	12	6e5

The table includes the train error (Tr ϵ), test error (Ts ϵ), train and test R^2, train and test L^2 Norm, computation time in seconds (Time), number of cores used (Cores), and the number of iterations (Itr.) performed by each optimizer. 'Itr.' denotes the total iterations performed during the optimization process, essential for achieving convergence in iterative methods like Gradient Descent.

the benchmarks, OLS and GD. Specifically, the train and test R^2 for BMC and MC differ by less than 0.3% from OLS. Similarly, the train and test MSE for BMC and MC are marginal compared to the OLS train and test MSE. Lastly, the train and test L^2 Norm for BMC and MC are at most 0.011.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this paper, we have developed two gradient free optimizers that utilize sampling and multiple cores processors to administrate the optimization process. In addition, the Monte Carlo method requires random sampling technique to search for the weight vector within a predefined search space. On the other hand, Bagging Monte Carlo exploits the ensemble method that requires bagging sampling for the input data for C individual models, and in each model, it randomly generates Q amount of weight vectors and selects the lowest MSE weight vector as the final weight. Lastly, Bagging Monte Carlo aggregates Q results and returns the averaged predicted value. In section 4.1, we have experiments with multiple datasets to observe the Double Descent event with OLS, GD, MC, and BMC optimizers. On the other hand, GD, BC, and BMC are considerably stable across all k values. In addition, BMC has the lowest with most stable MSE with minimal variability across all values k. Section 4.2 has shown that BMC and MC are highly comparable with the baseline, OLS. In two out of three experiments, BMC and MC have lower train and test MSEs than the GD. Moreover, the robustness of the train and test R^2 for BMC and MC are highly similar to the OLS R^2. Furthermore, both BMC and MC are exploiting the 12 cores resource with manageable computation time that generates 600,000 samples that deliver highly competitive performance accuracies in both train and test datasets.

5.2 Future Work

For future work, researching and discovering the model behaviors that involve more than five features is necessitated. Second, both Monte Carlo and Bagging Monte Carlo optimizers show promising performance in solving linear regression problems. However, it would be interesting to investigate if both optimizers will do well in a non-convex problem. Third, the search space plays a key role in the process of Monte Carlo and Bagging Monte Carlo optimization and it is crucial to understand how we should define a search space that yields to a high accuracy. Lastly, it is common to have a multiple cores machine that has more than 12 cores in practice. We can leverage more cores to mitigate each core workload. However, it is intriguing to test the algorithm on an accelerator like GPU.

References

[1] Abulkarim Faraj Alqahtani and Mohammad Ilyas. A machine learning ensemble model for the detection of cyberbullying, 2024.

[2] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[3] Sebastian Buschjäger and Katharina Morik. There is no double-descent in random forests, 2021.

[4] Ping-yeh Chiang, Renkun Ni, David Yu Miller, Arpit Bansal, Jonas Geiping, Micah Goldblum, and Tom Goldstein. Loss landscapes are all you need: Neural network generalization can be explained without the implicit bias of gradient descent. In *The Eleventh International Conference on Learning Representations*, 2023.

[5] Alexandru Crăciun and Debarghya Ghoshdastidar. On the convergence of gradient descent for large learning rates, 2024.

[6] Julian Hofstadler and Daniel Rudolf. Consistency of randomized integration methods. *Journal of Complexity*, 76:101740, June 2023.

[7] Feihu Jin, Yin Liu, and Ying Tan. Derivative-free optimization for low-rank adaptation in large language models, 2024.

[8] Azal Ahmad Khan, Omkar Chaudhari, and Rohitash Chandra. A review of ensemble learning and data augmentation models for class imbalanced problems: combination, implementation and evaluation, 2023.

[9] Gawel Kus and Miguel A. Bessa. Gradient-free neural topology optimization: Towards effective fracture-resistant designs, 2024.

[10] David Luengo, Luca Martino, Mónica Bugallo, Víctor Elvira, and Simo Särkkä. A survey of monte carlo methods for parameter estimation. *EURASIP Journal on Advances in Signal Processing*, 2020(1), May 2020.

[11] Luke Metz, C. Daniel Freeman, Samuel S. Schoenholz, and Tal Kachman. Gradients are not all you need, 2022.

[12] Preetum Nakkiran. More data can hurt for linear regression: Sample-wise double descent, 2019.

[13] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt, 2019.

[14] Giang Ngo, Rodney Beard, and Rohitash Chandra. Evolutionary bagging for ensemble learning. *Neurocomputing*, 510:1–14, October 2022.

[15] Sheikh Md. Mushfiqur Rahman and Nasir U. Eisty. Introducing ensemble machine learning algorithms for automatic test case generation using learning based testing, 2024.

[16] Jason W. Rocks and Pankaj Mehta. Memorizing without overfitting: Bias, variance, and interpolation in overparameterized models. *Physical Review Research*, 4(1), March 2022.

[17] Daniel Sanz-Alonso and Omar Al-Ghattas. A first course in monte carlo methods, 2024.

[18] Rylan Schaeffer, Zachary Robertson, Akhilan Boopathy, Mikail Khona, Kateryna Pistunova, Jason William Rocks, Ila R Fiete, Andrey Gromov, and Sanmi Koyejo. Double

descent demystified: Identifying, interpreting & ablating the sources of a deep learning puzzle. In *The Third Blogpost Track at ICLR 2024*, 2024.

[19] Jake A. Soloff, Rina Foygel Barber, and Rebecca Willett. Bagging provides assumption-free stability, 2024.

[20] J.-C. Walter and G.T. Barkema. An introduction to monte carlo methods. *Physica A: Statistical Mechanics and its Applications*, 418:78–87, January 2015.

[21] Oleg Yavoruk. How does the monte carlo method work?, 2020.