Journal of Theoretical and Applied Information Technology

30th November 2023. Vol.101. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ENHANCING ERROR MINIMIZATION IN MACHINE LEARNING: A NOVEL APPROACH INTEGRATING GRADIENT-BASED AND DICHOTOMY METHODS COMPARED TO GRADIENT DESCENT

ABDELHAMID OUAZZANI CHAHDI[1], WIAM SAIDI[2], KHALID SATORI[3], RAFIK LASRI[4], ABDELLATIF EL ABDERRAHMANI[5]

[1, 2] Researcher, Department of Computer Science, Sidi Mohammed Ben Abdellah University Fez, Morocco

[3, 5] Professor, Department of Computer Science, Sidi Mohammed Ben Abdellah University -Fez, Morocco

[4]Associate Professor, Physics Department, Abdelmalek Essaadi University Larache, Morocco

E-mail: [1]abdelhamid.ouazzanichahdi@usmba.ac.ma, [2]wiam.saidi@usmba.ac.ma,

[3]khlid.satori@usmba.ac.ma, [5]abdellatif.elabderrahmani@usmba.ac.ma

ABSTRACT

This research introduces an innovative technique designed to effectively minimize errors in machine learning, with the intention of subsequently applying it to enhance cloud-computing security. Our approach merges gradient-based optimization with the dichotomy method, streamlining the learning process. Its primary objective is the swift identification of the minimum point of a differentiable and convex cost function. To evaluate its efficacy in comparison to the traditional Gradient Descent approach, we apply it to linear regression models and conduct a comprehensive analysis across various dataset sizes and precision settings. Our experiments reveal significant advantages, including reduced execution time and fewer iterations required for convergence. This research contributes to the advancement of optimization techniques in machine learning and deep learning, promising potential benefits for practitioners, especially in the context of cloud computing security.

Keywords: *Deep Learning, Dichotomous Search, Gradient Descent, Linear regression, Machine Learning, Optimization*

1. INTRODUCTION

In recent years, the fields of machine learning (ML) and deep learning (DL) have witnessed remarkable growth, propelled by advancements in ML algorithms and the availability of extensive datasets [9]. ML and DL offer a multitude of advantages in data analysis tasks such as feature extraction, clustering, classification, regression, prediction, and more [14]. However, a central challenge within these fields revolves around optimizing cost functions, pivotal metrics utilized to quantify the deviation between a model's predictions and the actual values it aims to estimate. Among these metrics, the mean squared error (MSE) stands out as one of the most frequently used indicators for assessing the performance of regression models, encompassing linear regression, neural networks for regression, and polynomial regression [22]. The primary goal during the training of a regression model is to minimize the MSE. This optimization process frequently relies on algorithms such as gradient descent, stochastic gradient descent, mini-batch gradient descent, batch gradient descent, and others [6], each possessing its own advantages and suitability for specific machine learning tasks. Nevertheless, these optimization algorithms come with their limitations, including sensitivity to learning rate selection, sluggish convergence on complex cost surfaces, and computational inefficiencies when dealing with extensive datasets [19]. This paper presents a solution to overcome these constraints by improving solution accuracy and accelerating the learning process. We introduce an innovative method called 'Dichotomous

Gradient Descent' (DGD), designed to efficiently pinpoint minima in convex and differentiable cost functions. DGD offers notable advantages, including rapid convergence, robustness, computational efficiency, adaptability to high-dimensional spaces, and precision. Moreover, it simplifies practical implementation by eliminating the requirement for a learning rate. Nevertheless, we will also explore critical limitations, encompassing susceptibility to local minima, interval constraints, limited parallelism, and sensitivity to cost function shapes. These limitations necessitate careful consideration in specific applications. The following sections of this article are organized as follows: Section 2 provides an in-depth review of related works, offering a comprehensive exploration of gradient descent and the dichotomous method. In Section 3, we introduce our proposed methodology, known as Dichotomous Gradient Descent, and discuss its implementation in linear regression. Section 4 presents the Numerical Results and Discussion.

2. RELATED WORKS

2.1 Gradient Descent And Its Derivatives

Gradient descent stands as a ubiquitous optimization algorithm within the domains of machine learning and deep learning, primarily employed for the minimization of objective functions during the iterative learning process. Its operational principle involves the descent along the negative gradient direction to ascertain the local minima of differentiable and convex cost functions. This methodological framework finds pervasive utilization in machine learning and deep learning paradigms, including applications like linear regression [5].

As delineated in Figure 1, the algorithm advances iteratively by the computation of the subsequent step based on the gradient at the current spatial coordinate. Subsequently, this step undergoes adjustment through a learning rate, and the resultant value is subtracted from the present position according to the prescribed formula:

$$A_{t+1} = A_t - \alpha \nabla f(A_t) \qquad (1)$$

with:

A_{t+1} = Next position

A_t = Current position

α = Learning rate (Step Size)

$\nabla f(A_t)$ = Gradient at current position

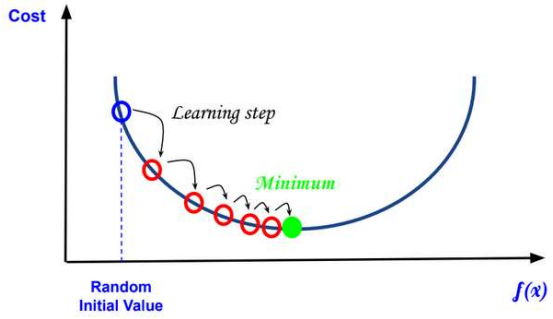

Figure 1: Gradient descent algorithm

There exist three distinct variants of the gradient descent algorithm, primarily differentiated by their data utilization strategy for determining the subsequent step. Batch gradient descent (BGD) methodically leverages the entire dataset in each epoch to compute the next step, while stochastic gradient descent (SGD) draws upon a singular example from the dataset per epoch. Mini-batch gradient descent, on the other hand, employs a fixed batch size of data examples from the dataset to compute the ensuing step during each epoch[11][6].

The gradient descent methodology presents several intricate challenges demanding meticulous consideration. Firstly, the selection of an appropriate learning rate poses a non-trivial task. An excessively small learning rate engenders sluggish convergence, whereas an overly large one impedes convergence and introduces erratic behavior into the loss function, potentially leading to divergence [11]. Secondly, conventional strategies such as learning rate schedules endeavor to dynamically adjust the learning rate during training. However, these adaptations frequently rely on predefined schedules or thresholds rooted in the variations of the objective function between epochs, lacking adaptability to the idiosyncratic characteristics of a given dataset [1]. Lastly, the

uniform application of a singular learning rate to all parameter updates may not be well-suited for datasets characterized by sparsity, featuring features with varying frequencies. In such circumstances, it may prove more advantageous to enact substantial updates for rarely occurring features, while adopting a more cautious approach for frequently encountered ones [15]. In the ensuing sections, we shall delineate several algorithms widely embraced within the Deep Learning community to confront the aforementioned challenges.

2.1.1 Momentum

Momentum gradient descent is a variant of the gradient descent optimization algorithm used in machine learning and deep learning. It incorporates a momentum term to help accelerate convergence, particularly when dealing with complex and high-dimensional optimization problems. As delineated in Fig. 2, the update to the model's parameters at each iteration is influenced not only by the current gradient of the loss function but also by a moving average of past gradients. This moving average, often referred to as "momentum," helps to smooth out variations in the gradient and allows the optimization process to continue in the same direction even when the gradient changes direction frequently[17][3].

The formula for updating the parameters in momentum gradient descent typically looks like this:

$$V_{t+1} = \gamma V_t - \alpha \nabla f(A_t)$$
$$A_{t+1} = A_t - V_{t+1} \quad (2)$$

With:
V_t = is the velocity or momentum at time step t.
A_t = Current position
γ = is the momentum coefficient, often set between 0 and 1 [17].

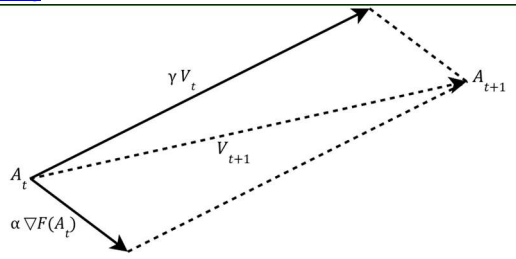

Figure 2: Momentum Algorithm

2.1.2 Nesterov accelerated gradient

Nesterov Accelerated Gradient (NAG), commonly referred to as Nesterov Momentum, stands as a specialized variant of the Momentum optimization algorithm with widespread adoption in the domains of deep learning and optimization. Diverging from the conventional Momentum methodology, as elucidated in Fig. 3, Nesterov Momentum initiates its process by computing an initial step that advances in the direction of the prevailing momentum vector. Subsequently, it refines this step by incorporating the gradient information at the updated position. This distinctive 'look-ahead' capability endows Nesterov Momentum with the capacity to proactively anticipate forthcoming updates, thereby resulting in expedited convergence, particularly in cases characterized by cost functions that feature elongated and narrow valleys. Nesterov Momentum adeptly strikes a harmonious balance, harnessing the advantages of momentum to accelerate convergence while maintaining precision during proximity to the minimum [21]. The computation of NAG can be succinctly represented through the following update rules:

$$V_t = \gamma V_{t-1} + \alpha \nabla f(A_n - \gamma V_{t-1})$$
$$A_{n+1} = A_n - V(t) \quad (3)$$

Here are the key components:
V_t = is the velocity or momentum at time step t.
A_n = Current position
A_{n+1} = Next position
γ = is the momentum coefficient, often set between 0 and 1 [17].

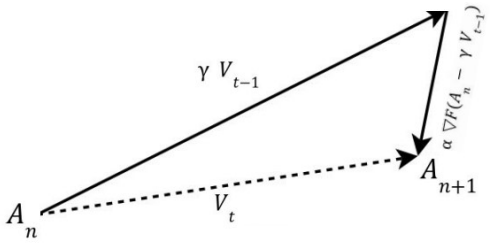

Figure 3: Nesterov Accelerated Gradient Algorithm

2.1.3 Adagrad

Adagrad, an acronym denoting Adaptive Gradient Algorithm, is a fundamental optimization technique extensively employed in the realms of machine learning and deep learning. Its prominent feature lies in its adaptability to dynamically tailor learning rates for individual model parameters during the training process. This adaptability bestows upon Adagrad a particular utility in scenarios characterized by sparse data or problems where certain features exert a significantly more pronounced influence than others. The algorithm achieves this by modulating the learning rate for each parameter based on the historical gradients encountered, thereby effectuating smaller updates for frequently changing parameters and more substantial updates for those exhibiting less frequent alterations [11][20][19]. The AdaGrad update rules can be succinctly articulated as follows:

$$A_{t+1} = A_t - \frac{\alpha}{\sqrt{\epsilon + \Sigma_{i=1}^{t} \nabla f(A_i)}} \nabla f(A_t) \qquad (4)$$

with:
A_{t+1} = Next position
A_t = Current position
α = Initial learning rate (Step Size)
$\nabla f(A_t)$ = Gradient at current position
ϵ = is typically set to a small positive value (e.g.,$1e-8$) to prevent division by zero in case the denominator, $\Sigma_{i=1}^{t} \nabla f(A_i)$, equals zero.

However, one notable drawback of AdaGrad is its accumulation of squared gradients in the denominator. Since these accumulated terms are always positive, the learning rate can become excessively small, potentially causing a slowdown and, in some cases, hindering convergence to the minimum [11][10].

2.1.4 Adadelta

Adadelta, an extension derived from the Adagrad optimization paradigm, introduces a more robust optimization strategy by effectively addressing the issue of monotonically decreasing learning rates inherent in Adagrad. In stark contrast to Adagrad, which accumulates all historical squared gradients, Adadelta employs a limited-size window mechanism to maintain a decaying average of prior gradients. This adaptive algorithm eliminates the necessity of specifying an initial learning rate, as it dynamically adapts the learning rate in response to the historical gradients and updates. The inherent stability and adaptability of Adadelta render it a particularly valuable choice for the training of neural networks, especially in scenarios marked by fluctuating gradient magnitudes. As a result, it significantly enhances the convergence and overall optimization performance [19].

2.1.5 RMSprop

Root Mean Square Propagation (RMSprop) constitutes an evolutionary step beyond AdaGrad, meticulously designed to counteract the issue of diminishing learning rates. Notably, both RMSprop and Adadelta emerged as independent solutions, nearly concurrently, to address the challenge of Adagrad's rapidly declining learning rates. In contrast to Adagrad, which accumulates gradient sums for each dimension, RMSprop computes the learning rate through a process involving a decaying average of gradients, significantly enhancing its computational efficiency. By harnessing this decaying average of gradients, RMSprop effectively attenuates the influence of extreme historical values, thereby fostering expeditious convergence, particularly in scenarios characterized by convex structures. Consequently, RMSprop has earned widespread acclaim as an exceedingly efficient and widely adopted optimization technique, particularly within the domain of deep neural networks, cementing its status as a gold standard among practitioners in the field of deep learning [10][12].

2.1.6 Adam

Adam, an abbreviation for Adaptive Moment Estimation, represents a widely adopted optimization algorithm within the domain of deep learning and machine learning research. It harmoniously amalgamates the virtues of both momentum-based optimization and RMSprop. Adam meticulously maintains two distinct moving averages: the first-order moment, encapsulating the mean of historical gradients, and the second-order moment, encapsulating the uncentered variance of historical gradients. These moving averages hold the pivotal role of tailoring the learning rate for each parameter individually, thereby facilitating swift convergence, even when faced with the challenges posed by gradients characterized as sparse or noisy. The distinctive attributes of Adam, including its adaptive learning rate and momentum components, imbue it with exceptional efficiency, rendering it highly applicable across a wide spectrum of optimization tasks. Consequently, it has emerged as the preferred choice for training deep neural networks and other machine learning models. Its remarkable versatility and robust empirical performance have solidified its position as a preeminent optimization algorithm, garnering widespread adoption and endorsement among both researchers and practitioners in the field [16].

2.1.7 AdaMax

AdaMax, an optimization algorithm derived from the Adam algorithm, has been engineered to strategically address select limitations inherent to its predecessor. In contrast to Adam, which relies on the management of two moving averages (specifically, the first and second moments), AdaMax simplifies this process by exclusively considering the first moment (representing the mean of gradients) while introducing the concept of the infinity norm (characterizing the maximum absolute value) for the second moment. This strategic adaptation imparts enhanced robustness, particularly in the presence of gradients marked by noise or sparsity. AdaMax's design notably excels in scenarios typified by significant variability in gradient updates. By employing this approach, AdaMax deftly maintains adaptive learning rates tailored to each parameter, thereby fostering efficient convergence, especially in the intricate landscape of non-convex optimization problems frequently encountered within the domain of deep learning. The algorithm's inherent simplicity, coupled with its compelling empirical performance, establishes AdaMax as a promising and viable alternative to Adam. It thus furnishes researchers and practitioners with a valuable tool for the training of neural networks and the optimization of various machine learning models [16].

2.1.8 Nadam

Nadam, an acronym signifying Nesterov-accelerated Adaptive Moment Estimation, represents an advanced optimization algorithm that harmoniously amalgamates the robust features of Nesterov's accelerated gradient descent with the adaptive learning rate mechanisms inherent to Adam. Notably, it introduces Nesterov momentum, thereby expediting convergence by computing gradients with an enhanced estimate of future positions. Nadam maintains the practice of retaining moving averages of historical gradients and their uncentered variance, a hallmark feature akin to Adam. However, it innovatively incorporates the Nesterov update scheme for gradient calculation. This fusion of methodologies engenders a highly efficient optimization algorithm, exquisitely tailored for navigating the intricate terrains typified by deep neural networks and complex optimization landscapes. Empirically, Nadam consistently exhibits competitive performance metrics in terms of training speed and convergence accuracy, firmly establishing itself as a prominent choice among researchers and practitioners operating within the realms of machine learning and deep learning [13].

2.2 The Dichotomy Method (Bisection Method)

The dichotomy method, also recognized as the bisection method, stands as a straightforward yet formidable technique utilized for the minimization of convex functions. Its efficacy is particularly pronounced when employed on functions that exhibit a confirmed convexity property over a

closed interval. As visually represented in Fig. 4, this method initiates with an initial interval encompassing the sought-after minimum value and subsequently subdivides the interval systematically while assessing the function's value at the midpoint. It sequentially discerns which segment of the interval contains the minimum and progressively refines the search domain until a predefined tolerance level is satisfied [8]. This iterative procedure ensures a convergence trajectory towards the global minimum for convex functions by perpetually narrowing the search space, thus methodically homing in on the optimal solution. The dichotomy method, characterized by its computational efficiency, finds broad utility in the domain of optimization, particularly in circumstances where function evaluations entail high computational cost or exhibit noise. Its convergence is linear, and it operates autonomously of gradient information. Nevertheless, its foremost constraint lies in its applicability, which is principally confined to one-dimensional optimization tasks. Consequently, its suitability diminishes when confronted with intricate, high-dimensional, or non-convex optimization problems that frequently emerge in contemporary research undertakings [2].

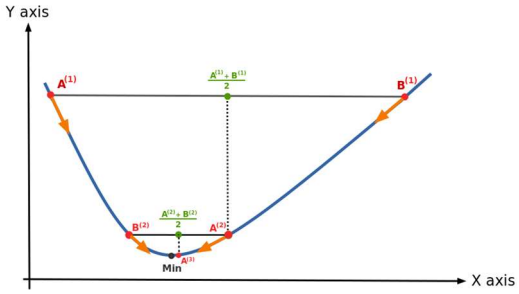

Figure 4: Bisection Method

3. THE PROPOSED APPROACH

3.1 Methodology Overview

Optimizing convex and differentiable functions plays a pivotal role in various machine learning and deep learning applications. In this context, we introduce DGD, an innovative approach that blends Gradient-Based and Dichotomy methods to efficiently minimize such functions. Our method involves crucial steps, including level set determination, gradient computation, intersection point identification, and iterative updates. DGD offers a robust and efficient means of optimizing convex and differentiable functions, delivering precise solutions while minimizing iteration requirements. Moreover, our algorithm seamlessly integrates with a range of machine learning and deep learning models, with a primary focus on reducing learning errors for convex and differentiable functions. For a visual representation of our approach, refer to the flowchart in Fig. 5, which illustrates the iterative steps in the process.

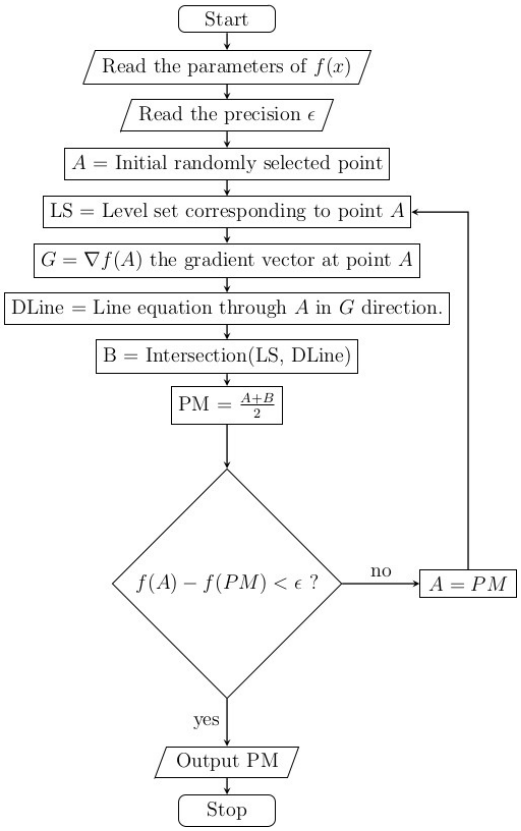

Figure 5: Dichotomous Gradient Descent Algorithm

Our approach begins with an initial iterated point $A^{(1)} \in E$ where E defines a differentiable and convex function f, and a precision tolerance $\epsilon > 0$. Our method establishes a sequence of iterates, $A^{(2)}, A^{(3)}, A^{(4)}, \ldots \in E$, with the condition that $f(A^{(i+1)}) < f(A^{(i)})$, ensuring that the precision criterion of $f(A^{(i)}) - f(A^{(i+1)}) < \epsilon$ is met. The

transition from it passes from $A^{(i)}$ to $A^{(i+1)}$ involves the following steps:

1. Selecting an initial point at random and denoting it as $A^{(i)}$, where $i = 0$ (see Fig. 6).

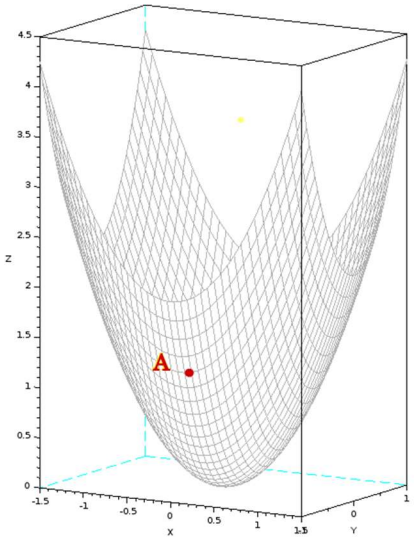

Figure 6: Random Starting Point A

2. Identifying the level set LS passing through the random point $A^{(i)}$ (see Fig. 7).

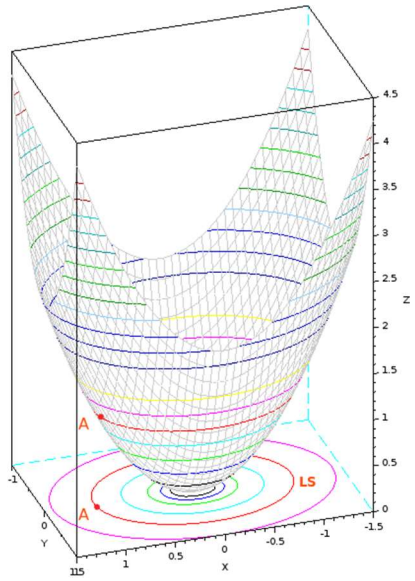

Figure 7 : Level Set LS Passing Through A

3. Computing the gradient vector $G \leftarrow \nabla f(A^{(i)})$(see Fig. 8).

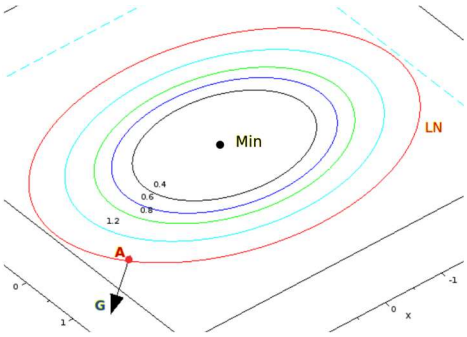

Figure 8: Gradient Vector G At Point A

4. Determining the equation of the line D passing through $A^{(i)}$ in the direction of vector G (see Fig. 9).

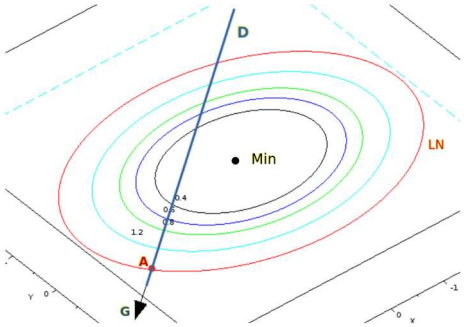

Figure 9: Line D Passing Through A In The Direction Of Vector G

5. Calculating the coordinates $B^{(i)}$, which represent points of intersection between line D and the level set LS (see Fig. 10).

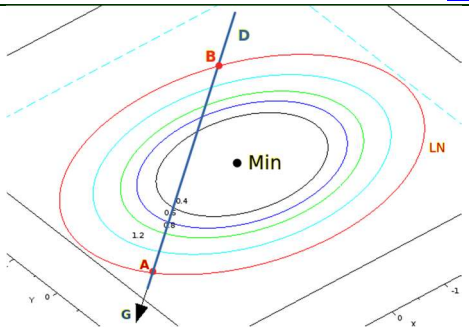

Figure 10: Point B: Intersection Points Between Line D And The Level Set LS

6. Finding the midpoint coordinates $PM^{(i)}$ of the segment $[A^{(i)}, B^{(i)}]$, $PM^{(i)} \leftarrow \frac{A^{(i)}+B^{(i)}}{2}$ (see Fig. 11).

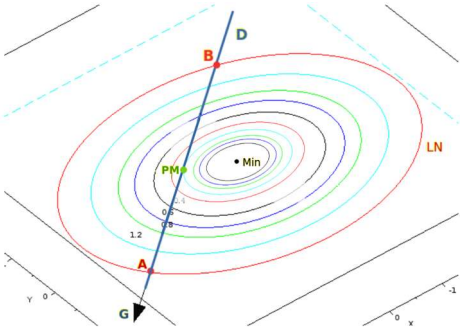

Figure 11: Midpoint Pm (x_M, y_M)

7. Updating the iterate: $A^{(i+1)} \leftarrow PM^{(i)}$ and repeat step 2 until the desired level of precision for the minimum is achieved (see Fig. 12).

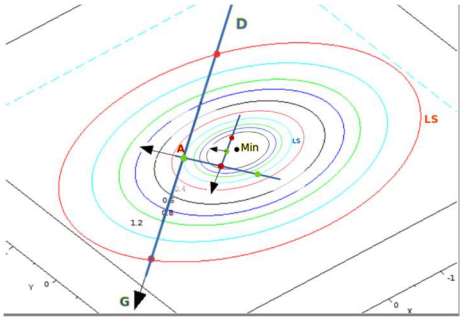

Figure 12: Convergence Of Dichotomous Gradient Descent Algorithm

To demonstrate the effectiveness of our approach, we have tested the algorithm on a basic linear regression model.

3.2 Application To Linear Regression Model

Linear regression, a classical statistical method, coexists synergistically within the domains of ML and DL. While ML extends the capabilities of linear regression for more sophisticated predictive modeling tasks, DL, employing deep neural networks, offers unparalleled complexity for tasks like image recognition and natural language processing. Nonetheless, linear regression retains its pivotal role as a foundational model for interpretability in ML and DL projects, underscoring its enduring relevance in the context of data-driven research and practical applications [18].

At its core, linear regression seeks to elucidate the optimal linear equation that encapsulates the relationship between a dependent variable (the outcome variable) and independent variables (predictors). Mathematically, this relationship is articulated as: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \varepsilon$ where Y is the dependent variable, β_0 denotes the intercept, $\beta_1, \beta_2, \ldots, \beta_n$ signify the coefficients, $X_1, X_2, \ldots, X_n$ denote the independent variables, and ε symbolizes the error term. Linear regression encompasses various forms, including simple, multiple, polynomial, and regularized regressions, each tailored to address distinct research needs [7]. The analytical journey involves fundamental steps such as data acquisition, model specification, training, evaluation, inference, and real-world application across diverse fields, ranging from economics to medicine. It mandates vigilant consideration of key challenges including overfitting, model interpretability, adherence to underlying assumptions, and the robust handling of outliers [4].

In essence, the linear regression model stands as the foundational representation of the intricate relationship between the dependent variable Y and the independent variables X through a linear function.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$f(x) = ax + b \quad (5)$$

3.2.1 Dataset

The algorithm is tested on a simple linear regression model from a randomly generated dataset with a single variable x.

We will therefore have a dataset with m examples and $n = 1$ features (Fig. 13)

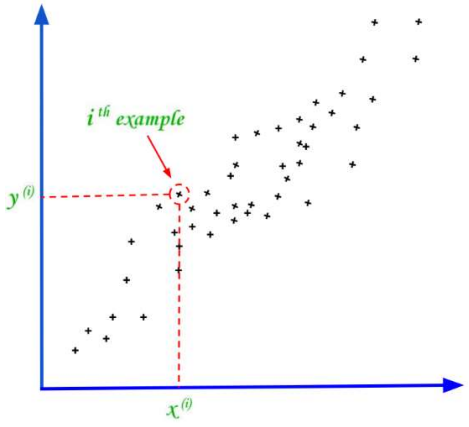

Figure 13: Random Dataset

3.2.2 Mean squared error function

In the multitude of potential lines, our objective is to pinpoint the line that achieves the minimal average of squared errors. In our approach, we will endeavor to find the minimum of the MSE cost function $J(a, b)$:

$$J(a,b) = \frac{1}{2m}\Sigma_{i=1}^{m}\left(ax^{(i)} + \mathrm{b} - \mathrm{y}^{(\mathrm{i})}\right)^2 \quad (6)$$

The J cost function, illustrated in Fig. 14, serves as a measure of this average error. Referred to as the Mean Squared Error, this function depends solely on two parameters: 'a' and 'b'. Importantly, The MSE is a convex and differentiable function, a critical attribute ensuring the convergence of our method to the minimum.

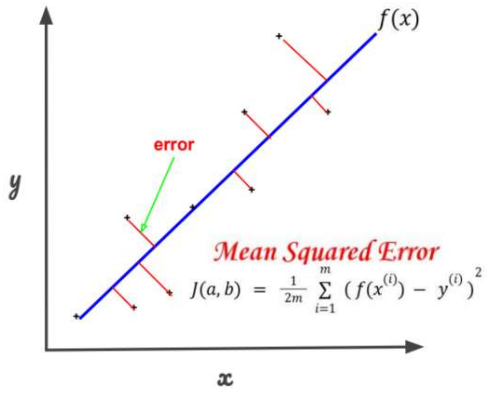

Figure 14: Linear Regression Line

We will employ the coordinates (x, y, z) in place of (a, b, c), resulting in the transformation of the function J to:

$$J(a,b) = \frac{1}{2m}\Sigma_{i=1}^{m}\left(xA^{(i)} + \mathrm{y} - B^{(\mathrm{i})}\right)^2 \quad (7)$$

where $\left(A^{(i)}, B^{(i)}\right)$ is our dataset, $A^{(i)}$ are the features and $B^{(i)}$ are the targets.

3.2.3 Utilizing Our Approach DGD to Minimize the MSE Function

1. Selecting an initial point at random, denoted as $A(x_A, y_A, z_A)$, from within the domain of the function's definition, where:

$$Z_A = J(x_A, y_A) \quad (8)$$

2. Identifying the level set LS passing through the random point A, defined by:

$$J(x,y) = Z_A$$
$$\frac{1}{2m}\Sigma_{i=1}^{m}\left(xA^{(i)} + \mathrm{y} - B^{(\mathrm{i})}\right)^2 = Z_A \quad (9)$$

3. Computing the gradient vector $G \leftarrow \nabla f(A)$: To compute the gradient G of the function J(x, y), we needed to establish the partial derivatives of J with respect to each parameter, x and y, denoted as $\nabla J = \left(\frac{\partial J(x,y)}{\partial x}, \frac{\partial J(x,y)}{\partial y}\right)$ where:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$\frac{\partial J(x,y)}{\partial x} = \frac{1}{m}\Sigma_{i=1}^{m}\left(xA^{(i)} + y - B^{(i)}\right)A^{(i)}$$
$$\frac{\partial J(x,y)}{\partial y} = \frac{1}{m}\Sigma_{i=1}^{m}\left(xA^{(i)} + y - B^{(i)}\right) \quad (10)$$

4. Determining the equation of the line D passing through A in the direction of vector G: Consider $A(x_A, y_A)$ as a point on the straight line D, and let $G(\alpha,\beta)$ a directional vector of D. A point $M(x,y)$ belongs to line D if and only if the vectors $AM(x - x_A, y - y_A)$ and $G(\alpha,\beta)$ are collinear. This condition is expressed as:

$$\beta(x - x_A) - \alpha(y - y_A) = 0$$
$$y = \frac{\beta}{\alpha}x + \left(y_A - \frac{\beta}{\alpha}x_A\right) \quad (11)$$

This equation can be reconfigured into the form $y = ax + b$ with: $a = \frac{\beta}{\alpha}$ and $b = \left(y_A - \frac{\beta}{\alpha}x_A\right)$.

5. Calculating the coordinates B, which represent points of intersection between line D and the level set LS: To determine the coordinates of the second intersection point B, we need to identify the points where the line D intersects the level set LS. This involves solving the following system of equations:

$$\frac{1}{2m}\Sigma_{i=1}^{m}\left(xA^{(i)} + y - B^{(i)}\right)^2 = Z_A$$
$$y = a_D x + b_D \quad (12)$$

where a_D and b_D are the line D parameters By replacing y in the level set equation by $a_D x + b_D$:

$$\frac{1}{2m}\Sigma_{i=1}^{m}\left(xA^{(i)} + a_D x + b_D - B^{(i)}\right)^2 = Z_A$$
$$\frac{1}{2m}\Sigma_{i=1}^{m}\left(x\left(A^{(i)} + a_D\right) + \left(b_D - B^{(i)}\right)\right)^2 = Z_A \quad (13)$$

The equation can be reduced to the form $ax^2 + bx + c = 0$:

$$\left[\frac{1}{2m}\Sigma_{i=1}^{m}\left(A^{(i)} + a_D\right)^2\right]x^2 + \left[\frac{1}{m}\Sigma_{i=1}^{m}\left(A^{(i)} + a_D\right)\left(b_D - B^{(i)}\right)\right]x + \left[\frac{1}{2m}\Sigma_{i=1}^{m}\left(b_D + B^{(i)}\right)^2 - \mathrm{Z_A}\right] = 0 \quad (14)$$

where x is the variable, and:

$a = \frac{1}{2m}\Sigma_{i=1}^{m}\left(A^{(i)} + a_D\right)^2$,

$b = \frac{1}{m}\Sigma_{i=1}^{m}\left(A^{(i)} + a_D\right)\left(b_D - B^{(i)}\right)$ and

$c = \frac{1}{2m}\Sigma_{i=1}^{m}\left(b_D + B^{(i)}\right)^2 - \mathrm{Z_A}$

This equation is quadratic, and the number of solutions depends on the discriminant $\Delta = b2 - 4ac$:

If $\Delta > 0$: The equation has two distinct solutions, representing our starting point A and the sought-after point B.

If $\Delta = 0$: The equation has only one solution, which corresponds exactly to our starting point A (the minimum).

If $\Delta < 0$: While it was unexpected, the equation has no solution. In practical applications, the dichotomy method, a fundamental component of our approach, repeatedly divides the interval [a, b] into two equal segments. However, as the difference between 'a' and 'b' diminishes, there is a possibility that the midpoint of the interval may align precisely with one of the endpoints (a or b). Due to the finite precision of floating-point calculations, issues related to numerical stability can arise when subtracting values that are very close, potentially causing Δ to become negative and, in turn, potentially causing the optimization process to stall.

6. Finding the midpoint coordinates $PM(x_M, y_M)$ of the segment $[A, B]$:

$$x_M = \frac{x_A + x_B}{2}$$
$$y_M = \frac{y_A + y_B}{2} \quad (15)$$

7. Assign the value of PM to the new coordinates of point A and repeat step 2 until the desired level of precision for the minimum is achieved.

4. NUMERICAL RESULTS AND DISCUSSION

4.1 Test Environment Description

The tests and experiments detailed in this research article were conducted within an environment characterized by the specifications provided in Table. 1 below:

Table 1: Specifications Of The Experimental Environments

Operating System	Linux Mint 21 Cinnamon Version 5.4.8
Linux Kernel	5.15.0-41-generic
Processor	Intel© Xeon© CPU E5-1650 v2 @ 3.50GHz × 6
Memory	62.7 GiB
IDE	Spyder Python IDE for scientific Version 5.4.3 (conda)
Programming language	Python 3.11.3 64-bit

It should be noted that the tests can be conducted in environments with characteristics that are less robust than those specified in Table 1, particularly with regard to RAM and processor capabilities.

4.2 Experiments

In the scope of our research, we conducted an extensive comparative analysis between our proposed methodology, DGD, and the conventional Gradient Descent (GD) technique. Our primary aim was to conduct a comprehensive assessment of the performance of these two algorithms and elucidate the potential ramifications of these findings within the context of machine learning applications.

We executed a quantitative performance evaluation contrasting our DGD algorithm with the GD algorithm when applied to the linear regression model, as outlined in Section 3.2. This comparative analysis was carried out through two distinct experiments.

4.2.1 Experiment 1

In this experiment, we monitored the cost evolution using a random dataset of size 10^3 examples for both algorithms, DGD and GD, with a learning rate of 0.1 for GD. We tracked the cost's progression throughout the entire execution, recording it until it reached a specific precision of $\epsilon = 10^{-1}$. The numerical results will be compared using two criteria: the number of iterations and CPU time. The results of experiment 1 are summarized in the table 2 below:

Table 2: Cost Evolution up to Iteration Number and CPU Time for Precision $\epsilon = 10^{-16}$

Algorithm	Iterations	CPU Time	Final cost
GD	1129	81.0 ms	0.4874839408897289
DGD	10	1.6 ms	0.4874839408897289

Fig. 15 and Fig. 16 respectively display a contour plot and a 3D plot for both algorithms, GD and DGD, providing a visual representation of how the theta parameter values evolve over iterations during the optimization of the MSE cost function towards its minimum.

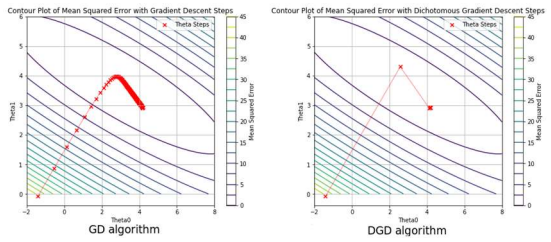

Figure 15: Contour Plot Comparing MSE Convergence Of GD And DGD Algorithms

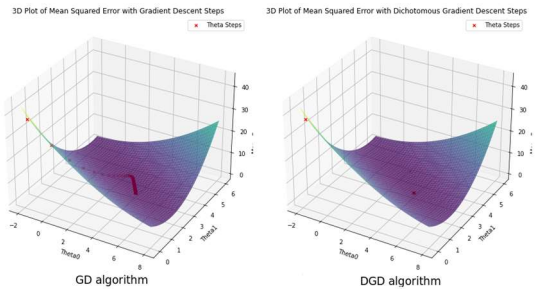

Figure 16: 3D Plot Comparing MSE Convergence Of GD And DGD Algorithms

Fig. 17, depicts the comparison of cost evolution between the DGD and GD algorithms for the first 10 iterations.

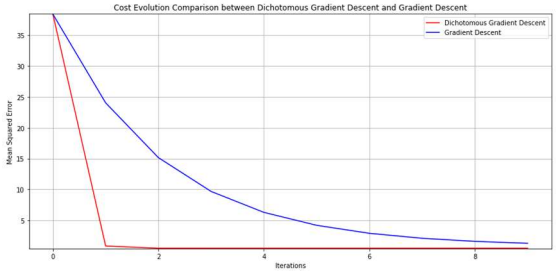

Figure 17: Cost Evolution Comparison Between DGD And GD

Our observation reveals a significant difference between DGD and GD in terms of the number of iterations required to converge. DGD exhibited much faster convergence with only 10 iterations, while GD required 1129. This finding suggests that DGD may be a preferred choice when rapid convergence is a critical criterion.

The CPU time data confirms the advantage of DGD. DGD managed to achieve convergence in just 1.6 milliseconds, while GD took 81 milliseconds. This significant reduction in computation time is crucial in real-time applications or when dealing with large datasets.

Despite differences in the number of iterations and CPU time, both algorithms reached an identical final cost of 0.4874839408897289. This indicates that both methods converge to solutions of comparable quality in terms of cost.

In the field of machine learning, the results of our experimentation have significant implications. By opting for the DGD approach, practitioners can benefit from faster convergence and substantial reduction in computation time without compromising the quality of the final solution. This decision can be crucial in real-time applications or when processing massive datasets.

4.2.2 Experiment 2

Experiment 2 records the number of iterations and execution times for random datasets of varying sizes (number of examples), while maintaining a fixed precision level of $\epsilon = 10^{-16}$ (see Table. 3). In this study, we evaluate the performance of both GD and DGD algorithms in relation to dataset size, all while consistently maintaining $\epsilon = 10^{-16}$ precision. We document the iteration count and CPU times for datasets of different sizes.

Table 3: Comparison Of GD And DGD Performance Across Different Dataset Sizes

	GD		DGD	
Data set size	Iterations	CPU Time (ms)	Iterations	CPU Time (ms)
10^{+1}	993	50.5	6	0.842
10^{+2}	979	51.8	11	1.73
10^{+3}	1129	83.5	10	2.2
10^{+4}	1103	175	8	2.29
10^{+5}	1084	541	8	8.46
10^{+6}	1117	7180	13	215
10^{+7}	1131	101000	15	3040

In this table comparing GD and DGD across various data set sizes, it's evident that DGD consistently outperforms GD in terms of convergence efficiency and computational time. For smaller data set sizes, DGD requires fewer iterations to converge, and as the data set size increases, the computational advantage of DGD becomes even more pronounced, with significantly reduced CPU time. These results highlight the potential benefits of adopting DGD over GD in machine learning tasks, particularly when dealing with larger datasets or scenarios where computational efficiency is critical.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

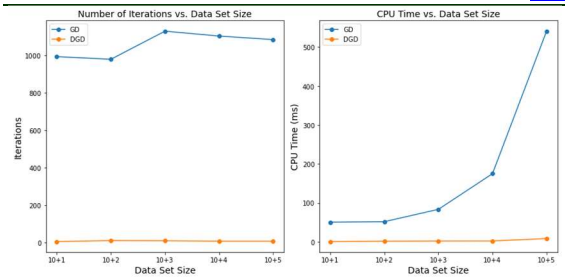

Figure 18: Comparison Of GD And DGD Performance Across Different Dataset Sizes

Fig. 18 provides a comparative analysis of two optimization algorithms: GD and our approach, DGD, across various dataset sizes.

On the left side, the plot titled "Number of Iterations vs. Data Set Size" illustrates the number of iterations required for both GD and DGD to converge as the dataset size increases. It is evident that DGD consistently outperforms GD, requiring significantly fewer iterations to achieve convergence. This advantage is particularly crucial in deep learning tasks where model training can be computationally intensive.

On the right side, the graph titled "CPU Time vs. Data Set Size" displays the computational time (in milliseconds) for GD and DGD. Once again, DGD demonstrates its efficiency by substantially reducing CPU time, even as the dataset size grows. This efficiency is of paramount importance in machine learning, where faster convergence and reduced computation times can expedite model development and deployment.

The results presented in this graph underscore the practical advantages of employing DGD over GD in machine learning and deep learning scenarios, highlighting the potential for accelerated training and more efficient model optimization, especially when working with large datasets.

4.3 Advantages And Limitations Of The Method

4.3.1 Advantages of the method

The newly proposed method combines gradient-based optimization with the dichotomy method to efficiently locate the minimum of convex and differentiable cost functions, offering several potential advantages:

Fast Convergence: The combination of gradient and dichotomy enables rapid convergence towards the global minimum of the cost function, particularly benefiting convex functions and reducing the required number of iterations.

Robustness: This approach exhibits robustness and versatility, accommodating a wide range of cost functions, including nonlinear ones.

Computational Efficiency: By effectively utilizing the gradient to guide the search, the method reduces the computational cost associated with evaluating the cost function, especially in resource-intensive scenarios.

Dimension Adaptability: It adapts well to high-dimensional problems as gradient-based optimization efficiently exploits the function's structure, while the dichotomy method effectively handles searches over wide intervals.

Numerical Stability: The combination of these two methods mitigates numerical stability issues, particularly those related to vanishing gradients, which can occur when relying solely on the gradient.

Guaranteed Convergence: For convex functions, the method guarantees convergence to the global minimum, offering a substantial advantage.

Broad Applicability: This method finds applications across various domains, including mathematical optimization, machine learning, engineering, finance, and other scientific disciplines.

When compared to gradient descent and its derivatives, a significant advantage of this method lies in its inherent independence from the requirement for a learning rate, a feature commonly associated with traditional gradient descent approaches. This absence of a learning rate bestows several advantages:

No Need for Learning Rate Selection: Eliminating the requirement to select a learning rate significantly simplifies the method's implementation. In traditional gradient descent, choosing an appropriate learning rate can be challenging, often involving trial and error. This new method circumvents this delicate step.

Enhanced Stability: The challenge of learning rate selection is closely tied to the stability of convergence. Inappropriate learning rates can lead to divergence or slow convergence. By removing this parameter, the method tends to exhibit greater stability and robustness.

Inherent Adaptability: By employing a combination of gradient and dichotomy, the method naturally adapts to the local characteristics of the cost function without the need for global learning rate fine-tuning.

In summary, the absence of a learning rate represents a significant advantage of this method. It streamlines implementation, enhances stability, and offers a more natural adaptation to local cost function characteristics, making the method user-friendly and robust for a wide range of optimization problems.

4.3.2 Limitations of the Method

While the novel method we propose combines gradient-based optimization with the dichotomy approach, offering several notable advantages for efficiently locating the minimum of convex and differentiable cost functions, it is essential to address certain limitations that may affect its applicability and performance. We delve into these limitations, shedding light on potential challenges that users and practitioners should consider when employing this method:

Local Minimum Challenge: One of the primary concerns associated with our method is its susceptibility to local minima. Depending on the specific characteristics of the cost function, the method may become trapped in suboptimal solutions, making it vital for users to assess the presence of such minima in their optimization problems.

Interval Constraints: The dichotomy method, integral to our approach, continuously divides the interval [a, b] into two equal segments. However, as the difference between a and b narrows, there arises the risk of reaching a point where the midpoint of the interval aligns precisely (within floating-point precision) with one of the interval's endpoints (a or b). This situation can lead to a stagnation of the optimization process

Limited Parallelism: Our method may not fully exploit parallel processing capabilities, which can potentially restrict its convergence speed when executed on modern hardware architectures. This limitation underscores the importance of assessing computational efficiency, especially for large-scale optimization problems.

Function Shape Dependency: It is crucial to recognize that our method relies on the cost function's properties, specifically its convex and differentiable nature. When dealing with cost functions that significantly deviate from these characteristics, such as non-convex or non-differentiable functions, the method's effectiveness may be compromised. Users should exercise caution and consider alternative approaches for such scenarios.

In conclusion, while our method offers several advantages, it is essential to acknowledge and address these limitations to make informed decisions about its application. Understanding these limitations will enable users to harness the strengths of our method effectively while mitigating potential challenges.

5. CONCLUSION

In this research, we introduced a novel approach called "Dichotomous Gradient Descent" (DGD), which combines gradient-based optimization with the dichotomy method to efficiently locate minima in convex and differentiable cost functions. Our method is designed to significantly speed up learning in machine learning and deep learning models. To assess its effectiveness, we conducted experiments using the Mean Squared Error (MSE) cost function in the context of linear regression and compared DGD to traditional Gradient Descent

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(GD). The results showed outstanding performance, with fewer iterations needed for convergence and reduced execution time. This underscores the importance and effectiveness of our approach in accelerating learning, promising significant improvements in machine learning.

Our method offers advantages like fast convergence, robustness, computational efficiency, adaptability to high dimensions, precision, improved numerical stability, ease of implementation, and guaranteed convergence for convex functions. Additionally, it doesn't require a learning rate, simplifying practical application.

However, it's essential to consider limitations, including susceptibility to local minima, interval constraints, limited parallelism, and dependence on cost function shape. Users should carefully assess these limitations in specific applications, potentially making adjustments or exploring alternatives for non-convex or non-differentiable cost functions.

In summary, our method is versatile and effective across various domains, offering speed, precision, and stability for complex optimization problems. Users must understand both its advantages and limitations to use it effectively. Future research can focus on tailored adaptations and improvements to overcome these limitations and further expand the method's potential.

REFERENCES

[1] Xiong, Y., Lan, L. C., Chen, X., Wang, R., & Hsieh, C. J. (2022, January). Learning To Schedule Learning Rate With Graph Neural Networks. In International Conference On Learning Representation (ICLR).

[2] Etesami, R., Madadi, M., Mashinchi, M., & Ganjoei, R. A. (2021). A New Method For Rooting Nonlinear Equations Based On The Bisection Method. Methodsx, 8, 101502.

[3] Liu, Z., Feng, R., Li, X., Wang, W., & Wu, X. (2021). Gradient-Sensitive Optimization For Convolutional Neural Networks. Computational Intelligence And Neuroscience, 2021, 1-16.

[4] Filzmoser, P., & Nordhausen, K. (2021). Robust Linear Regression For High-Dimensional Data: An Overview. Wiley Interdisciplinary Reviews: Computational Statistics, 13(4), E1524.

[5] Eftekhari, A., Vandereycken, B., Vilmart, G., & Zygalakis, K. C. (2021). Explicit Stabilised Gradient Descent For Faster Strongly Convex Optimisation. BIT Numerical Mathematics, 61, 119-139.

[6] Haji, S. H., & Abdulazeez, A. M. (2021). Comparison Of Optimization Techniques Based On Gradient Descent Algorithm: A Review. Palarch's Journal Of Archaeology Of Egypt/Egyptology, 18(4), 2715-2743.

[7] Maulud, D., & Abdulazeez, A. M. (2020). A Review On Linear Regression Comprehensive In Machine Learning. Journal Of Applied Science And Technology Trends, 1(4), 140-147.

[8] Moheuddin, M. M., Uddin, M. J., & Kowsher, M. (2019). A New Study To Find Out The Best Computational Method For Solving The Nonlinear Equation. Applied Mathematics And Sciences An International Journal (Mathsj), 6(3), 15-31.

[9] Ray, S. (2019, February). A Quick Review Of Machine Learning Algorithms. In 2019 International Conference On Machine Learning, Big Data, Cloud And Parallel Computing (Comitcon) (Pp. 35-39). IEEE.

[10] Mukkamala, M. C., & Hein, M. (2017, July). Variants Of Rmsprop And Adagrad With Logarithmic Regret Bounds. In International Conference On Machine Learning (Pp. 2545-2553). PMLR.

[11] Ruder, S. (2016). An Overview Of Gradient Descent Optimization Algorithms. Arxiv Preprint Arxiv:1609.04747.

[12] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[13] Dozat, T. (2016). Incorporating Nesterov Momentum Into Adam. ICLR Workshop, 1, 2013-2016.

[14] Aggarwal, C. C. (2015). Data Mining: The Textbook (Vol. 1). New York: Springer.

[15] Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014). Identifying And Attacking The Saddle Point Problem In High-Dimensional Non-Convex

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Optimization. Advances In Neural Information Processing Systems, 27.

[16] Kingma, D. P., & Ba, J. (2014). Adam: A Method For Stochastic Optimization. Arxiv Preprint Arxiv:1412.6980.

[17] Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, May). On The Importance Of Initialization And Momentum In Deep Learning. In International Conference On Machine Learning (Pp. 1139-1147). PMLR.

[18] Su, X., Yan, X., & Tsai, C. L. (2012). Linear Regression. Wiley Interdisciplinary Reviews: Computational Statistics, 4(3), 275-294.

[19] Zeiler, M. D. (2012). Adadelta: An Adaptive Learning Rate Method. Arxiv Preprint Arxiv:1212.5701.

[20] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods For Online Learning And Stochastic Optimization. Journal Of Machine Learning Research, 12(7).

[21] Nesterov, Y. E. E. (1983). A Method Of Solving A Convex Programming Problem With Convergence Rate Ok2. In Doklady Akademii Nauk (Vol. 269, No. 3, Pp. 543-547). Russian Academy Of Sciences.

[22] Theobald, C. M. (1974). Generalizations Of Mean Square Error Applied To Ridge Regression. Journal Of The Royal Statistical Society: Series B (Methodological), 36(1), 103–106.Doi:10.1111/J.2517-6161.1974.Tb00990.X