PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
ESCUELA DE POSGRADO

Ideal Step Size Estimation for the Multinomial Logistic Regression

Tesis para obtener el grado académico de Maestro en Procesamiento de Señales e Imágenes Digitales que presenta:

Gabriel Ramirez Orihuela

Asesor:

Dr. Paul Antonio Rodriguez Valderrama

Lima, 2024

Informe de Similitud

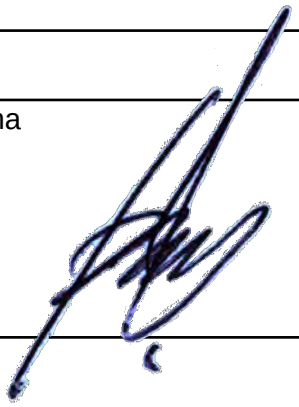
Yo, Paul Antonio Rodriguez Valderrama, docente de la Escuela de Posgrado de la Pontificia Universidad Católica del Perú, asesor de la tesis titulada Ideal Step Size Estimation for the Multinomial Logistic Regression, de el autor Gabriel Ramirez Orihuela, dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 14%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 8/11/24.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de investigación, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha:

Lima, 11 de Noviembre de 2024.

Apellidos y nombres del asesor / de la asesora: Rodriguez Valderrama, Paul Antonio	
DNI: 07754238	Firma
ORCID: 0000-0002-8501-0907	



Resumen

En la base de los problemas de optimización en aprendizaje profundo residen algoritmos como el Gradiente Descendiente Estocástico (SGD, por sus siglas en inglés), el cual emplea un subconjunto de los datos por iteración para estimar el gradiente con el fin de minimizar una función de costo. Los algoritmos adaptativos, basados en el SGD, son ampliamente reconocidos por su efectividad al utilizar la información del gradiente de iteraciones previas, generando un momento o memoria que permite una predicción más precisa de la pendiente real del gradiente en iteraciones futuras, acelerando así la convergencia. No obstante, estos algoritmos aún requieren una tasa de aprendizaje (learning rate o LR) inicial (escalar) así como un programador de LR.

En este trabajo proponemos un nuevo algoritmo de SGD que estima la LR inicial (escalar) mediante una adaptación del tamaño de paso ideal de Cauchy para la regresión logística multinomial; además, la LR se actualiza de manera recursiva hasta un número determinado de épocas, tras lo cual se emplea un programador de LR decreciente. El método propuesto se evalúa en varias arquitecturas de clasificación multiclase bien conocidas y se compara favorablemente con otras alternativas adaptativas (escalares y espaciales) bien optimizadas, incluyendo el algoritmo Adam.

Esta tesis fue presentada en la conferencia indexada en IEEE Xplore LASCAS'24.

Palabras Clave

Aprendizaje profundo, gradiente descendiente estocástico, tamaño de paso adaptativo, regresión logística multinomial.

Abstract

At the core of deep learning optimization problems reside algorithms such as the Stochastic Gradient Descent (SGD), which employs a subset of the data per iteration to estimate the gradient in order to minimize a cost function. Adaptive algorithms, based on SGD, are well known for being effective in using gradient information from past iterations, generating momentum or memory that enables a more accurate prediction of the true gradient slope in future iterations, thus accelerating convergence. Nevertheless, these algorithms still need an initial (scalar) learning rate (LR) as well as a LR scheduler.

In this work we propose a new SGD algorithm that estimates the initial (scalar) LR via an adaptation of the ideal Cauchy step size for the multinomial logistic regression; furthermore, the LR is recursively updated up to a given number of epochs, after which a decaying LR scheduler is used. The proposed method is assessed for several well-known multiclass classification architectures and favorably compares against other well-tuned (scalar and spatially) adaptive alternatives, including the Adam algorithm.

This thesis was presented in the IEEE Xplore indexed conference LASCAS'24.

Keywords

Deep learning, stochastic gradient descent, adaptive step size, multinomial logistic regression.

Contents

Introduction

Deep learning (DL) has become an increasingly popular field in the development of highly accurate models for a wide range of applications in recent years. This surge in popularity is driven by the ability of DL models to automatically extract and learn intricate patterns from a broad range of data, making them effective for tasks such as image classification, speech recognition, natural language processing, and many others [1].

Current deep learning models benefit significantly from parallel programming and advancements in hardware, especially the development of powerful GPUs for multi-core processing. These advancements have played a crucial role in the rapid growth of deep learning by enabling researchers to train larger and more complex models on vast amounts of data [2]. However, training these models still requires significant time.

As the complexity and size of deep learning models and datasets continue to grow, the need for efficient optimization methods becomes increasingly crucial. Adaptive optimization methods are, to date, the main optimization methods used for deep learning algorithms due to their ability to enhance the training process of deep learning models. The effectiveness of these methods is largely attributed to their ability to adjust learning rates dynamically, which allows adaptive optimization methods can navigate complex loss landscapes of DL models more effectively, leading to faster and more reliable convergence [3].

This adaptability is particularly important in deep learning, where the optimal learning rate can vary significantly across different layers and stages of training. Algorithms such as AdaGrad [4], RMSprop [5], and Adam [6] have become pivotal in the DL community due to their robust performance across various tasks and architectures [7].

In this thesis we proposed an adaptive learning rate designed to approximate the ideal step size for Multinomial Logistic Regression (MLR) [8] by means of the Cauchy method [9]. This approach estimates the optimal direction and magnitude of each step at every iteration, thereby enhancing the efficiency of training updates and ensuring that the step size remains close to optimal throughout the training process. This dynamic adjustment is vital for maintaining effective training, particularly with the increasing complexity and scale of deep learning models. When integrated with Stochastic Gradient Descent (SGD), this method can accelerate convergence and improve model accuracy, while maintaining computational costs similar to existing techniques.

This study also evaluates the estimated step size and compares it to the fixed step size used in SGD and its adaptive variants. The goal is to determine if the Cauchy derived step size can enhance convergence speed and accuracy compared to traditional and adaptive methods. In order to do so, an algorithm to compute the Ideal Step Size Estimation (ISSE) for the MLR is developed, then the performance of the algorithm is assessed by comparing it with vanilla SGD and its variants. Our experiment results support our main claim: SGD along with our proposed method has better performance than other existing adaptive step sizes, as well as comparative performance with adaptive methods like Adam [6].

The core ideas of this work were originally presented at the IEEE conference LASCAS'24 [10], in this document an extended version of such ideas are organized using following structure: In Chapter 1, statement of the problem, related theory and current approaches in the state of the art are covered. In Chapter 2, theory behind the logistic regression problem is discussed to understand the ideal step size for the MLR, as well as the study outline. In Chapter 3, the proposed method is calculated and an algorithm is implemented. In Chapter 4, the performance of the algorithm is compared with other adaptive step sizes and adaptive methods, employing the CIFAR-10 [11], CIFAR-100 [11], and SVHN [12] datasets in conjunction with the ResNet-18 [13] and DenseNet-121 [14] convolutional neural networks (CNNs). Finally, in the conclusions are presented and potential improvements for the proposed algorithm are discussed.

Chapter 1

Research Framework

1.1 Statement of the problem

At the core of many learning problems lies the task of optimization, which involves finding the best set of parameters for a given model by minimizing a loss function. Classical learning problems like linear regression [15] use optimization to minimize mean squared error, while more modern models such as Convolutional Neural Networks (CNNs) [16] and Transformers [17] also rely heavily on optimization for backpropagation to estimate the weight parameters.

One of the key factors that contributes to the success of deep learning is the optimization of model parameters using efficient algorithms such as stochastic gradient descent (SGD). SGD is one of the most widely used optimization algorithms and serves as a stochastic approximation of gradient descent (GD) [18].

Consider the following parameter update of GD and SGD:

$$F(\mathbf{u_k}) = \frac{1}{N}\sum_{n=1}^{N} f_n(\mathbf{u_k}) \tag{1a}$$

$$\mathbf{g}_k = \frac{1}{\#\mathcal{B}_k}\sum_{n\in\mathcal{B}_k} \nabla f_n(\mathbf{u_k}) \tag{1b}$$

In (1a), $\mathbf{u}_k$ denotes the model parameters at iteration k and $F(\mathbf{u_k})$ represents the average loss function over the entire dataset, N is the total number of data points, and f_n is the loss associated with the n-th data point.

In (1b), $\mathbf{g}_k$ represents the gradient estimate at iteration k and ∇f_n is the gradient of the loss function with respect to the n-th data point. SGD uses a random subset $\mathcal{B}_k$ of

$\{1,\ldots,N\}$, referred to as a batch. If $\mathcal{B}_k$ covers the entire set, then SGD becomes equivalent to GD.

In the original SGD formulation, the parameter update rule is expressed as follows:

$$\mathbf{u}_{k+1} = \mathbf{u}_k - \alpha \cdot \mathbf{g}_k \tag{2}$$

Here, α denotes the learning rate, which is crucial for influencing both the convergence speed and the final performance of the model. More recent variants of SGD, such as those involving momentum or adaptive learning rates, include additional parameters to enhance the convergence properties and efficiency of the optimization process, these variants adjust the gradient estimates and updates dynamically.

1.2 Related Theory

To determine an ideal step size that enhances the efficiency of the training process, an understanding of optimization methods is essential. For that end, convex functions and the Gradient Descent (GD) iterative algorithm and momentum methods are studied.

1.2.1 Convex functions

For the case of study, f represents a cost function that evaluates vectors in $\mathbb{R}^N$, implying: $f : \mathbb{R}^N \to \mathbb{R}$. Said function must be convex to guarantee that it reaches a minimum, for this, the following must be fulfilled:

$$f(\alpha\mathbf{y} + (1-\alpha)\mathbf{u}) \leq \alpha f(\mathbf{y}) + (1-\alpha) f(\mathbf{u}), \quad \forall \alpha \in [0,1], \mathbf{u}, \mathbf{y} \in \mathbb{R}^N \tag{3}$$

The first term represents the value of the convex function evaluated between $\mathbf{u}$ and $\mathbf{y}$, and the second term is the line segment connecting the values of the function in $\mathbf{u}$ and $\mathbf{y}$.

This inequality demonstrates the property of convex functions, where the function value at a convex combination of any two points in the domain does not exceed the corresponding combination of the function values at those points.

Furthermore, a differentiable function is one that lies above its tangent lines at every point where it is defined:

$$f(\mathbf{z}) \geq f(\mathbf{u}) + \langle \nabla f(\mathbf{u}), \mathbf{z} - \mathbf{u} \rangle, \quad \forall \mathbf{z}, \mathbf{u} \in \mathbb{R}^N \tag{4}$$

This relationship expresses that for any two points in the domain of the function, the value at one point is greater than or equal to the value at another point plus the inner product of the gradient of the function at that second point and the vector difference between

the two points. The inequality states that the value of the function f at any point $\mathbf{z}$ is at least as great as the value at another point $\mathbf{u}$ plus the linear approximation of f at $\mathbf{u}$ evaluated at $\mathbf{z}$.

This property allows the use of gradient based iterative methods in convex optimization, as it ensures that the direction of the gradient always points towards a minimum, warranting efficient search for optimal solutions within this framework.

Also, to be convex, the function f must be twice differentiable and have a positive value:

$$\nabla^2 f(\mathbf{u}) \geq 0 \tag{5}$$

This condition indicates that the Hessian matrix of the function f at any point $\mathbf{u}$ is positive semi-definite, which is a key factor in determining convexity at that point. When this condition holds across the entire domain of the function f, the function is globally convex. This property simplifies the optimization process when employing algorithms like gradient descent, as it guarantees that following the path of steepest descent will lead directly to the optimum.

Lastly, to ensure that the convex function will reach a minimum using the gradient descent algorithm with step size α, the function must be Lipschitz continuous, that is:

$$f(\mathbf{z}) \leq f(\mathbf{u}) + \langle \nabla f(\mathbf{u}), \mathbf{z} - \mathbf{u} \rangle + \frac{L}{2} \|\mathbf{u} - \mathbf{z}\|_2^2 \tag{6}$$

This condition implies that the function f has a bounded rate of change, controlled by the Lipschitz constant L, which ensures that the function does not vary too abruptly. Such bound is critical for the convergence of gradient descent because it allows for the selection of a step size α that is sufficiently small to make continuous progress towards the minimum, but large enough to ensure efficient convergence. Particularly, if $0 < \alpha \leq \frac{1}{L}$, gradient descent is guaranteed to converge to the minimum of a convex function, ensuring that each step taken is in a direction that reduces the value of the function.

This Lipschitz continuity, combined with the positive semidefinite condition of the Hessian, forms a robust framework for applying gradient descent to convex optimization problems efficiently.

1.2.2 Optimization algorithms: GD and SGD

Due to memory limitations, calculating the total gradient of the dataset, denoted $F_k(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} f_n(\mathbf{w_k})$ as in (1a), is not generally feasible in practice. This issue arises because, in real world applications $N \to \infty$, making it impractical to compute the gradient for the

entire dataset, therefore a random subset is used by each iteration in SGD [18]. In this context, the gradient is approximated by $\mathbf{g}_k = \frac{1}{\#\mathcal{B}_k} \sum_{n \in \mathcal{B}_k} \nabla f_n(\mathbf{w_k})$ as in (1b).

Given the parameter to be optimized $\mathbf{w}$ and the step size α_k which may be constant or adaptive, the key difference between GD and SGD are highlighted by (7) and (8):

$$\text{GD}: \quad \mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \cdot \nabla F(\mathbf{w}_k) \tag{7}$$

$$\text{SGD}: \quad \mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \cdot \mathbf{g}_k \tag{8}$$

With these approximations, it is possible to compute the cost function and the gradient without limits of size from the dataset and without memory limitations.

1.2.3 Momentum and Nesterov

Since GD and SGD are not efficient against sudden slope changes in multiple dimensions, momentum [19] can help handle it more effectively and reach a minimum of the cost function in fewer iterations by taking accounts of previous gradients in the update rule at each iteration:

$$\text{Momentum:} \quad \mathbf{z}_{k+1} = \gamma \cdot \mathbf{z}_k - \alpha \nabla F(\mathbf{w}_k) \tag{9}$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{z}_{k+1} \tag{10}$$

Momentum accelerates the reduction of the function in gradient direction with large changes and that has maintained its sign in previous iterations, obtaining faster convergence and reduced oscillation. The term γ determines the inertia or memory of the algorithm, if γ is large the momentum will overshoot the target and as γ goes smaller it behaves more like GD, and if γ is much bigger than α, the accumulated gradients will be dominant in the update rule.

A method that is closely related to momentum method is Nesterov Accelerated Gradient (NAG) [20]:

$$\text{Nesterov}: \mathbf{w}_{k+1} = \mathbf{z}_k - \alpha \nabla F(\mathbf{z}_k) \tag{11}$$

$$\mathbf{z}_{k+1} = \mathbf{w}_{k+1} + \gamma_k (\mathbf{w}_{k+1} - \mathbf{w}_k) \tag{12}$$

The difference between Momentum and Nesterov Accelerated Gradient lies in the gradient computation. In Momentum, the gradient is computed using the current parameters $\mathbf{w}_k$, but depending only in the accumulation of past gradients can sometimes be counterproductive. To better anticipate the behavior of the function in future iterations, the

gradient is evaluated at $\mathbf{z}_k$, which incorporates the inertial sequence $\gamma_k(\mathbf{w}_{k+1} - \mathbf{w}_k)$ that controls the extent of the lookahead, reflecting how strongly past gradients influence the current update. By evaluating the gradient at this adjusted position, NAG provides a more informed estimate of the future parameters. This helps improve the direction of the update by accounting for both the current trajectory and the accumulated momentum, ensuring that the update reflects not just the immediate gradient but also the expected future behavior of the optimization process.

1.3 State of the Art

1.3.1 Adaptive Step Sizes for SGD

Besides classical methods, the choice of an appropriate step size for executing SGD algorithms is one of the main challenges in deep learning, since SGD does not work with the conventional line search methodology [21], it is common in SGD to either use a diminishing step size or manually adjust a fixed step size [22].

Conventional adaptive step size strategies such as Cauchy [9], Barzilai-Borwein [23] and Lipschitz-based [24], that are common strategies in the GD context, cannot be directly used in SGD, however there are current studies that adapt them in order to fit the model.

1.3.1.1 Barzilai-Borwein Step Size for SGD

The Barzilai-Borwein (BB) method [23], is quasi-Newton approach that is designed to compute step sizes in optimization without directly calculating the Hessian matrix, instead, the BB method uses an approximation derived from the secant equation. As in (1a), F represents the cost function, and $\mathbf{H}$ is the approximation of the Hessian matrix:

$$\mathbf{H}_k(\mathbf{x}_k - \mathbf{x}_{k-1}) = \nabla F(\mathbf{x}_k) - \nabla F(\mathbf{x}_{k-1}) \tag{13}$$

This effectively captures the curvature of the objective function by considering the gradients and displacements between two consecutive iterations, allowing the BB method to have computational simplicity and robust convergence properties of more traditional, computationally intensive Newtonian methods.

The BB method minimizes the difference between the current and previous iterations gradients and objective variables as follows:

$$\arg\min_{\alpha} \|(\mathbf{x}_k - \mathbf{x}_{k-1}) - \alpha(\nabla F(\mathbf{x}_k) - \nabla F(\mathbf{x}_{k-1}))\|_2^2 \tag{14}$$

Where the step size obtained is:

$$\alpha_k^{BB} = \frac{(\mathbf{x}_k - \mathbf{x}_{k-1})^T (\nabla F(\mathbf{x}_k) - \nabla F(\mathbf{x}_{k-1}))}{\|\nabla F(\mathbf{x}_k) - \nabla F(\mathbf{x}_{k-1})\|_2^2} \tag{15}$$

However, due to the randomness of the stochastic gradients, the step size computed in SGD-BB may change drastically which may cause instability and divergence. An algorithm proposed in [25] addresses this problem by computing a summation of the moving average of the gradient each iteration inside an epoch to approximate the true gradient.

1.3.1.2 Lipschitz constant based step size

The Lipschitz constant L is an essential value in optimization algorithms, especially gradient-based methods like GD, it determines stability and convergence properties [24]:

$$\|f(x) - f(y)\| \leq L\|x - y\| \tag{16}$$

This inequality implies that the function f does not change more rapidly than L times the change in x. In practical terms, a smaller Lipschitz constant means that f is smoother, and thus, gradient descent can use a larger step size without risking divergence. Conversely, a large Lipschitz constant may lead to oscillations or slow convergence, necessitating a more cautious approach with smaller step sizes. Thus, knowing or estimating L accurately is crucial for the effective application of gradient based optimization methods. A method for the Lipschitz constant estimated for MLR in deep learning models is studied in [26].

1.3.2 Adaptive Methods for SGD

Adaptive algorithms [7] typically employ a step size that is either constant or exhibits a decaying pattern over time, they have been designed to enhance the efficiency of optimization methods. Rather than relying solely on current data, gradient information is incorporated from prior iterations, which aids in more accurately estimating the true gradient direction in the upcoming iterations; in the light of (2), this amounts to replacing $\mathbf{g}_k$ by a function of the set $\{\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_k\}$. This approach allows for more informed updates [27].

1.3.2.1 Adagrad

Adagrad [4] is an optimization method that dynamically adjusts the learning rate for each parameter by taking into account the historical squared gradients. The method computes an additional step for updating the parameter by first calculating the accumulated squared gradient vector up to time step k, represented by:

$$\mathbf{v}_k = \mathbf{v}_{k-1} + \mathbf{g}_k^2, \tag{17}$$

where $\mathbf{v}_k$ is the sum of the squares of the past gradients with respect to that parameter up to the current step, and $\mathbf{g}_k$ is the gradient of the loss function at step k. Given this accumulation, the parameter update is executed as follows:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \frac{\alpha}{\sqrt{\mathbf{v}_k + \varepsilon}} \cdot \mathbf{g}_k \tag{18}$$

In this equation, $\mathbf{w}_k$ denotes the parameter value in the current iteration, α is a predefined initial learning rate, and ε is a small constant added to improve numerical stability and prevent division by zero and oversized steps.

Adagrad adjusts the learning rate for each parameter by dividing it by the square root of the accumulated gradients, allowing for an individualized approach to parameter updates as seen in Figure 1.

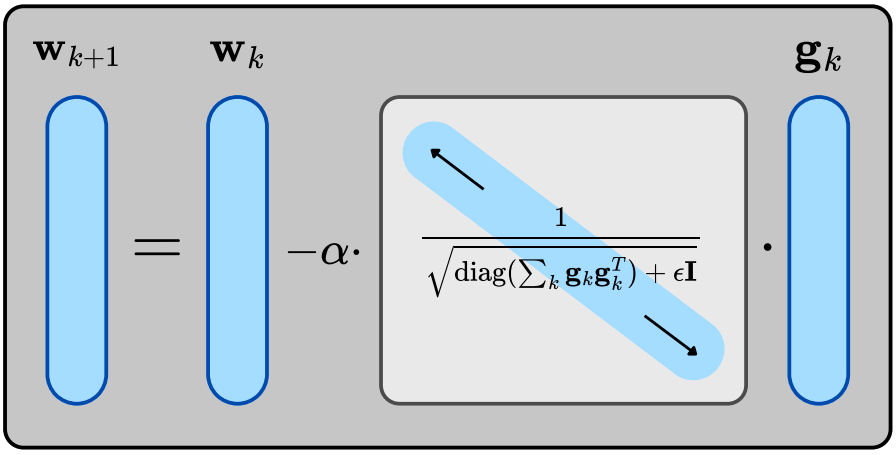


Figure 1: Multi-dimensional Adaptive Step Size Adagrad.

This adaptive learning rate mechanism results in smaller updates for parameters with large gradients and larger updates for those with smaller gradients, which ensures that infrequently occurring but potentially relevant parameters receive sufficient attention. By scaling the updates based on the frequency and magnitude of the gradients, Adagrad promotes faster convergence, especially in datasets characterized by sparse features.

However, a significant limitation of AdaGrad is that its learning rate tends to continuously decrease over time, which can lead to very small updates and, consequently, a premature stopping of the training process. This occurs because the accumulated squared gradient in the denominator can grow excessively large, leading to an overly diminished effective learning rate for each parameter.

Despite this, AdaGrad has influenced the development of subsequent adaptive algorithms, which refine and build upon the initial approach, providing strategies for managing the diminishing learning rates.

1.3.2.2 RMSProp

RMSProp [5] is an adaptive optimization algorithm that modifies Adagrad to overcome the limitation of a monotonically decreasing learning rate. It introduces a decay factor to control the accumulation of past squared gradients:

$$\mathbf{v}_k = \beta \cdot \mathbf{v}_{k-1} + (1-\beta) \cdot \mathbf{g}_k^2 \tag{19}$$

Where β is a decay factor which moderates the influence of past gradients on the accumulation. This adjustment ensures that the accumulated squared gradients $\mathbf{v}_k$ to adapt more dynamically over time, preventing it to continue growing without bound which can result in the learning rate from diminishing too promptly.

This modification means that RMSprop does not just accumulate squared gradients, it maintains an exponential moving average of these gradients. This approach moderates the impact of any individual gradient update on the learning rate. Consequently, unlike Adagrad, RMSprop ensures that the learning rate declines more gradually. This allows for more consistent and effective parameter update in the training process.

In RMSprop, the parameter update rule is modified by the new definition of the variable $\mathbf{v}_k$, and follows the same structure of AdaGrad as in (18).

1.3.2.3 ADAM

The Adam algorithm [6] is designed to combine the benefits of previous presented optimization methods. Adam maintains an exponential moving average of past squared gradients and, additionally, also preserves an exponentially moving average of past gradients similar to momentum.

The first moment is represented by $\mathbf{m}_k$ and can be interpreted as the mean or momentum of the gradients. Alternatively, the second moment represented by $\mathbf{v}_k$, defines the uncentered variance like in RMSprop.

Adam has the following update equations for these moments:

$$\mathbf{m}_k = \beta_1 \cdot \mathbf{m}_{k-1} + (1-\beta_1) \cdot \mathbf{g}_k \tag{20}$$

$$\mathbf{v}_k = \beta_2 \cdot \mathbf{v}_{k-1} + (1-\beta_2) \cdot \mathbf{g}_k^2 \tag{21}$$

To improve the estimates in the early iterations, Adam introduces bias correction terms for both moment estimates, this ensures the estimates are more accurate during the initial phases of training where bias can significantly affect the calculations. The new moments are defined as:

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{\left(1-\beta_1^k\right)} \tag{22}$$

$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{\left(1-\beta_2^k\right)} \tag{23}$$

The bias corrected moments are then used for updating the parameters:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \frac{\alpha \cdot \hat{\mathbf{m}}_k}{\sqrt{\hat{\mathbf{v}}_k + \varepsilon}} \tag{24}$$

Overall, Adam has demonstrated faster convergence and superior performance across a wide spectrum of deep learning tasks, making it a popular choice in many applications [7].

1.3.3 Stochastic Optimization Methods Structure

Building upon the explanation of adaptive step sizes and adaptive methods for deep learning, Figure 2 synthesizes them in a structure. Starting from the left, the root of these methods, the SGD algorithm, is presented. The diagram then depicts the development from momentum adaptations to more recent multidimensional adaptive methods, and also includes the advancements of adaptive step sizes for SGD.

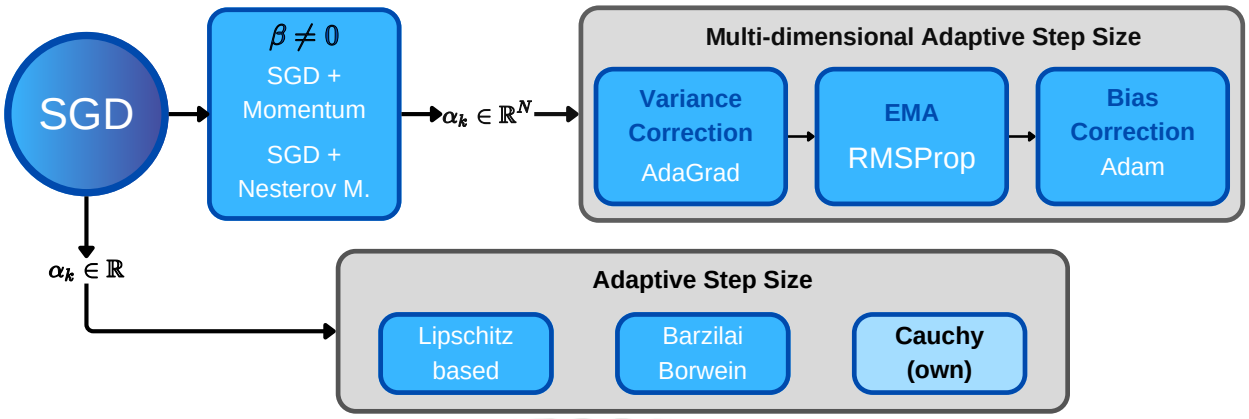


Figure 2: Stochastic Optimization Methods Structure

It it worth noting that the Cauchy Step Size inside the adaptive step size methods in Figure 2, portrayed as own, is the main aim of this work and is being proposed for the multinomial logistic regression.

Chapter 2

Case of Study

In this study, for image classification, the well known multiclass model of Multinomial Logistic Regression (MLR) or softmax [8] is employed, which arguably has better performance than SVMs [28] and decision trees [29]. It is robust to outliers, noisy data, and is differentiable, making it a popular choice in various fields such as image recognition and natural language processing.

We investigate the impact of an automatic step size selection method for the application of MLR for image classification tasks. The focus is to improve the training process by selecting appropriate step sizes for the SGD iterative algorithm, this involves analyzing different optimization techniques and their impact on the convergence and accuracy of the model.

2.1 MLR Cost Function

Starting from the binary logistic regression [30], which consist in classification of data with two classes (the results are bounded between zero and one), the following probability functions categorizes a variable in a binary domain, the sigmoid function:

$$P(y=0 \mid \mathbf{x},\mathbf{w}) = \frac{e^{-\langle \mathbf{x},\mathbf{w}\rangle}}{1+e^{-\langle \mathbf{x},\mathbf{w}\rangle}} \tag{25}$$

$$P(y=1 \mid \mathbf{x},\mathbf{w}) = 1 - P(y=0 \mid \mathbf{x},\mathbf{w}) = \frac{1}{1+e^{-\langle \mathbf{x},\mathbf{w}\rangle}} \tag{26}$$

In this, the vector $\mathbf{x}$ denotes the evaluated values that are part of two possible categories, y is the category or label of said data and $\mathbf{w}$ denotes a vector of weights which will be updated in each iteration and at its optimum, when reaching the minimum of the

cost function, said calculated weights will make the sigmoid function behave with the best possible accuracy.

For the case of multinomial logistic regression, there is categorization in multiple classes, it is a generalization of the binomial case and it takes the following form that is denoted by:

$$\begin{bmatrix} P(y=0 \mid \mathbf{x},\mathbf{w}) \\ P(y=1 \mid \mathbf{x},\mathbf{w}) \\ P(y=2 \mid \mathbf{x},\mathbf{w}) \\ \vdots \\ P(y=L-1 \mid \mathbf{x},\mathbf{w}) \end{bmatrix} = \frac{1}{\sum_{l=1}^{L} e^{\langle \mathbf{x},\mathbf{w}_l \rangle}} \begin{bmatrix} e^{\langle \mathbf{x},\mathbf{w}_0 \rangle} \\ e^{\langle \mathbf{x},\mathbf{w}_1 \rangle} \\ e^{\langle \mathbf{x},\mathbf{w}_2 \rangle} \\ \vdots \\ e^{\langle \mathbf{x},\mathbf{w}_{L-1} \rangle} \end{bmatrix} \tag{27}$$

In these, L classes are given, $\frac{1}{\sum_{l=1}^{L} e^{\langle \mathbf{x},\mathbf{w}_l \rangle}}$ normalize the data and the parameter $\mathbf{w}_l$ represents the weights for each of the different classes. The multiclass classification problem with the softmax activation function can be appreciated in Figure 3.

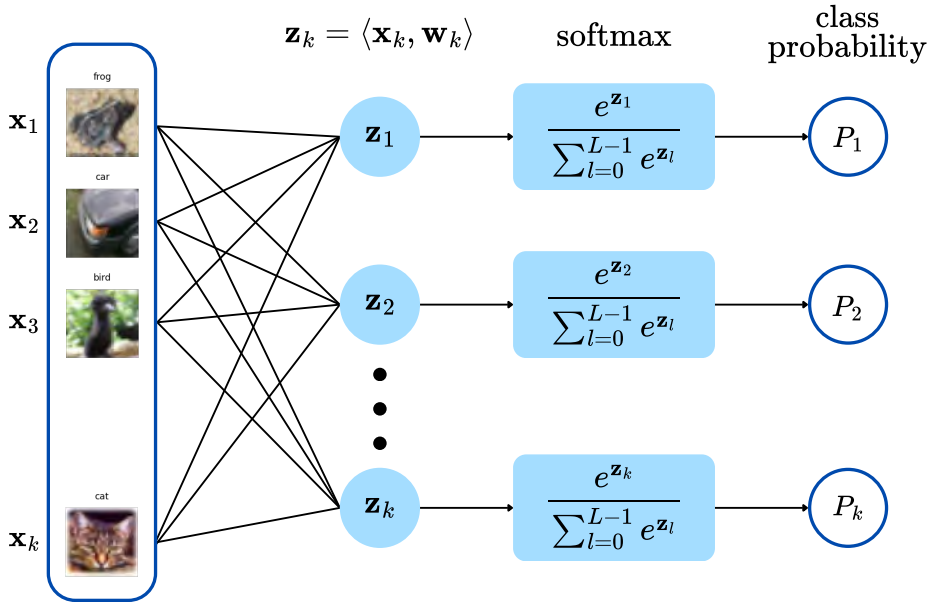


Figure 3: Softmax Activation in Multiclass Image Classification

In general, the following function is given, it is called the softmax function:

$$\mathrm{P}\left(y_n = c \mid \mathbf{x}_n, \mathbf{w}\right) = \frac{e^{\left\langle \mathbf{x}_n, \mathbf{w}_{y_n} \right\rangle}}{\sum_{l=0}^{L-1} e^{\langle \mathbf{x}_n, \mathbf{w}_l \rangle}} \tag{28}$$

For the multinominal logistic regression, which is the case of study, the following structure is employed:

$$\mathbf{X} = \begin{bmatrix} | & | & | & & | \\ \mathbf{x}_0 & \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_{N-1} \\ | & | & | & & | \end{bmatrix}, \mathbf{X} \in \mathbb{R}^{k \times N} \tag{29}$$

$$\mathbf{W} = \begin{bmatrix} | & | & | & & | \\ \mathbf{w}_0 & \mathbf{w}_1 & \mathbf{w}_2 & \dots & \mathbf{w}_{L-1} \\ | & | & | & & | \end{bmatrix}, \mathbf{W} \in \mathbb{R}^{k \times L} \tag{30}$$

$$\mathbf{Y} = \begin{bmatrix} | \\ \mathbf{y}_n \\ | \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{bmatrix}, \mathbf{Y} \in \mathbb{R}^{N \times 1} \tag{31}$$

$\mathbf{X}$ is a matrix containing the dataset which is composed by N columns denoting the number of the individual signals as vectors of data and k rows representing the features for each one, $\mathbf{W}$ is the weight matrix with L columns as vectors of the individual classes and k rows representing the features for each one, finally $\mathbf{Y}$ is a vector containing the respective category or label for each of the data.

To estimate $\mathbf{W}$, the Maximum Likelihood Estimation (MLE) method can be used. MLE is a method for estimating the parameters given the probability function and observed data, this is achieved by maximizing the likelihood function so that the observed data is most probable.

$$\mathbf{w}_{MLE} = \text{argmax}_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \tag{32}$$

The likelihood function $\mathcal{L}(\mathbf{w})$ is the joint probability of data, which in this case, being independent and identically distributed, becomes the multiplication of its known probability density functions:

$$\mathcal{L}(\mathbf{w}) = \prod_{i=0}^{N-1} \mathrm{P}\left(y_n = c \mid \mathbf{x}_n, \mathbf{w}\right) \tag{33}$$

Analogously, one can find the logarithm of the likelihood function given that it is monotonically increasing and maximizing the log-likelihood is equivalent:

$$l(\mathbf{w}) = \sum_{n=0}^{N-1} \log \left[\frac{e^{\langle \mathbf{x}_n, \mathbf{w}_{y_n} \rangle}}{\sum_{l=0}^{L-1} e^{\langle \mathbf{x}_n, \mathbf{w}_l \rangle}} \right] = \sum_{n=0}^{N-1} \left[\left\langle \mathbf{x}_n, \mathbf{w}_{y_n} \right\rangle - \log \sum_{l=0}^{L-1} e^{\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \right] \tag{34}$$

As can be seen, working with log-likelihood is favorable due to the ease it provides when working with multiplication of exponentials.

By taking $\mathbf{w} = -\mathbf{w}$, given that its values are initialized at random, the following cost function and its gradient are obtained:

$$f_c(\mathbf{w}) = \sum_{n=0}^{N-1} \left\langle \mathbf{x}_n, \mathbf{w}_{y_n} \right\rangle + \sum_{n=0}^{N-1} \log \left(\sum_{l=0}^{L-1} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \right) \tag{35}$$

$$\frac{\partial f_c(\mathbf{w})}{\partial \mathbf{w}_{y_n}} = \nabla_n = \sum_{n=0}^{N-1} \left(\mathbf{x}_n I\left(y_n = c\right) - \frac{e^{\left\langle \mathbf{x}_n, \mathbf{w}_{y_n} \right\rangle}}{\sum_{l=0}^{L-1} e^{\langle \mathbf{x}_n, \mathbf{w}_l \rangle}} \right) \tag{36}$$

The goal is to minimize the cost function to find an ideal step size for the SGD and its variants.

2.2 Ideal Step Size for MLR

The optimization of the step size in gradient descent is crucial for efficient learning in models such as multinomial logistic regression. We begin by defining the ideal step size, also known as the Cauchy step size [9], which is derived by minimizing the following function for the step k:

$$\alpha_k^C = \arg\min f\left(\mathbf{w}_n - \alpha \nabla_n\right) \tag{37}$$

To find this ideal step size, we first compute the derivative of the cost function with respect to α and set it to zero, ensuring that the step size minimizes the function along the direction of the gradient:

$$\frac{\partial f_c\left(\mathbf{w}_n - \alpha \nabla_n\right)}{\partial \alpha} = 0 \tag{38}$$

This derivative equation is a critical step as it confirms the point of minimum descent, integrating both the direction and magnitude of the step.

To apply the Cauchy Step Size to MLR, the cost function can be expressed and manipulated as follows:

$$f_c(\mathbf{w} - \alpha \nabla) = -\alpha \left\langle \mathbf{x}_n, \nabla_{y_n} \right\rangle + \log \left(\sum_{l=0}^{L-1} e^{-\langle \mathbf{x}_n, \mathbf{w}_l - \alpha \nabla_l \rangle} \right) \tag{39}$$

To find the ideal step size, we differentiate this function with respect to α and set the derivative to zero:

$$\frac{\partial}{\partial \alpha}\left[f_c(\mathbf{w}-\alpha\nabla)\right]=0 \tag{40}$$

By solving this equation, we obtain an estimate for the ideal step size.

2.3 Study Outline

This study proposes a novel adaptive method for the softmax regression that calculates an approximation of the ideal Cauchy step size [9]. Figure 4 illustrates the optimization process in a Deep Learning model for image classification, in which an adaptive step size is taken in contrast to a constant one.

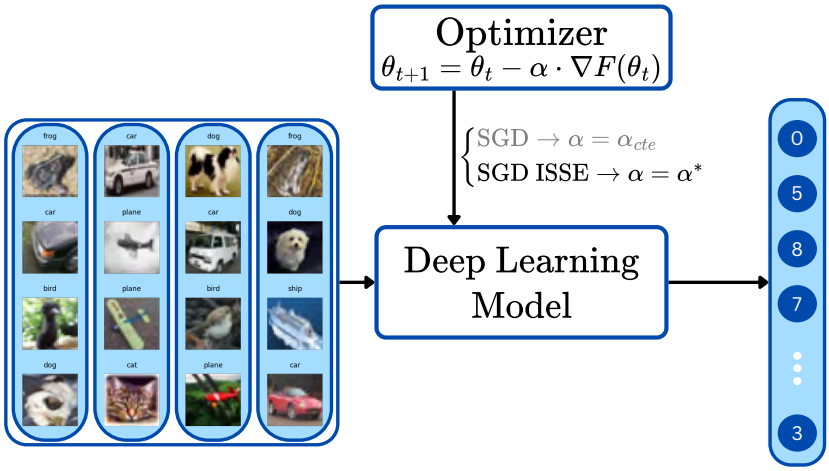


Figure 4: Optimization process in a Deep Learning model.

The approach involves the following steps:

- Theoretical and empirical validation: Demonstrating that an ideal step size can be stochastically approximated using the Cauchy method. This includes theoretical analysis and empirical validation to show the reliability of the method.
- Algorithm development: Creating an algorithm to estimate the ideal step size, dynamically adjusting it during training to ensure optimal parameter updates.
- Implementation in PyTorch: Developing an optimizer in PyTorch using the calculated step size.

- Performance testing: Rigorously testing the algorithm against traditional SGD and its adaptive variants, which involves experimentation on image classification tasks to assess loss, accuracy, and time.

By benchmarking against the presented methods, this study aims to demonstrate the advantages of using an adaptive, Cauchy based step size in deep learning, in order to contribute valuable insights into optimizing learning rates, potentially leading to more efficient and accurate models for diverse applications.

The estimation of a practical Cauchy step size for the MLR in the stochastic context will be derived in the following Chapter 3.

Chapter 3

Proposed Method

After analysing methods such as the SGD algorithm as well as adaptive methods for optimization, the case of approximating an ideal step size for the multinomial logistic regression is studied and implemented in a computational algorithm.

3.1 Approximate ideal step size for MLR in SGD

In neural networks with softmax activation, the magnitude of the gradients decreases as backpropagation moves from the output layer towards the earlier layers, this results in the gradients at the last layer being the largest among all the gradients computed [26]. Thus, the proposed step size is calculated in the last softmax layer with larger gradients that provide the most significant contribution to the update step.

An approximate ideal step size α is derived from the MLR cost function and is implemented in a computational algorithm. The cost function, when considering only a single sample from the entire dataset, is labeled as f_n:

$$f_n(\mathbf{w}) = \left\langle \mathbf{x}_n, \mathbf{w}_{y_n} \right\rangle + \log\left(\sum_{l=0}^{L-1} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \right), \tag{41}$$

where $\mathbf{w}$ represents the weights associated with the model, $\mathbf{x}_n$ is an individual input data sample, and y_n indicates its corresponding class. For a specific class l out of the total L classes, the gradient of the cost function with respect to the weights of that class is represented as $\nabla_l = \nabla f_n(\mathbf{w}_l)$, and for the weights associated with the class of the specific data sample y_n, is denoted by $\nabla_{y_n} = \nabla f_n(\mathbf{w}_{y_n})$.

Following a procedure similar to the one used when deriving the Cauchy step size, the loss function, for an unspecified α step, at the next GD step, is given by:

$$f(\mathbf{w}-\alpha\nabla) = -\alpha\left\langle \mathbf{x}_n, \nabla_{y_n}\right\rangle + \log\left(\sum_{l=0}^{L-1} e^{-\langle \mathbf{x}_n, \mathbf{w}_l - \alpha\nabla_l\rangle}\right) \tag{42}$$

Finding the derivative and equaling to zero $\frac{\partial}{\partial\alpha}[f(\mathbf{w}-\alpha\nabla)] = 0$:

$$\left\langle \mathbf{x}_n, \nabla_{y_n}\right\rangle = \frac{\sum_{l=0}^{L-1}\langle \mathbf{x}_n, \nabla_l\rangle e^{-\langle \mathbf{x}_n, \mathbf{w}_l - \alpha\nabla_l\rangle}}{\sum_{l=0}^{L-1} e^{-\langle \mathbf{x}_n, \mathbf{w}_l - \alpha\nabla_l\rangle}} \tag{43}$$

$$\sum_{l\neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l - \alpha\nabla_l\rangle} = \sum_{l\neq y_n} \frac{\langle \mathbf{x}_n, \nabla_l\rangle}{\left\langle \mathbf{x}_n, \nabla_{y_n}\right\rangle} e^{-\langle \mathbf{x}_n, \mathbf{w}_l - \alpha\nabla_l\rangle} \tag{44}$$

$$\sum_{l\neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l\rangle} \cdot e^{\alpha\langle \mathbf{x}_n, \nabla_l\rangle} = \sum_{l\neq y_n} \frac{\langle \mathbf{x}_n, \nabla_l\rangle}{\left\langle \mathbf{x}_n, \nabla_{y_n}\right\rangle} e^{-\langle \mathbf{x}_n, \mathbf{w}_l\rangle} \tag{45}$$

Using Taylor Series [31] to clear the desired value α :

$$e^{\alpha\langle \mathbf{x}_n, \nabla_l\rangle} = 1 + \alpha\langle x_n, \nabla_l\rangle + \frac{\alpha^2}{2}\langle \mathbf{x}_n, \nabla_l\rangle^2 \tag{46}$$

$$\sum_{l\neq y_n} e^{-\langle \mathbf{x}_n, w_l\rangle} \cdot \left(1 + \alpha\langle \mathbf{x}_n, \nabla_l\rangle + \frac{\alpha^2}{2}\langle \mathbf{x}_n, \nabla_l\rangle^2\right) = \sum_{l\neq y_n} \frac{\langle \mathbf{x}_n, \nabla_l\rangle}{\left\langle \mathbf{x}_n, \nabla_{y_n}\right\rangle} e^{-\langle \mathbf{x}_n, w_l\rangle} \tag{47}$$

In SGD, which can be interpreted as GD with unbiased noise added to the gradient at each iteration [32], the inherent randomness can result in negative or complex values for the step size in the quadratic solution. To counter this, the real part of the step size and its absolute value are considered in the quadratic equation solution. This ensures a positive, real step size, aligning with the principle of SGD by updating in the direction opposite to the gradient thereby maintaining the effectiveness of the method despite its inherent stochasticity. The following Ideal Step Size Estimation (ISSE) for MLR is obtained:

$$\alpha_{ISSE} = \left| \frac{\sum_{l\neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l\rangle}\langle \mathbf{x}_n, \nabla_l\rangle}{\sum_{l\neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l\rangle}\langle \mathbf{x}_n, \nabla_l\rangle^2} \right| \tag{48}$$

The sum over $l \neq y_n$ indicates that the step size adjusts based on how poorly the model performs on classes other than the target class. This can be beneficial for imbalanced classification problems or scenarios where certain classes are more challenging to classify than others.

Since $e^{-\langle \mathbf{x}_n, \mathbf{w}_l\rangle}$ represents the output of the softmax function, these values are higher when the accuracy of the model is low but should decrease over time, it gives more weight to terms where the dot product $-\langle \mathbf{x}_n, \mathbf{w}_l\rangle$ is small. This implies that if the current model is

already fairly confident in its prediction (the dot product is large), then the step size would be smaller, contributing to the stability of the model.

The squared gradient in the denominator serves as a normalizing factor that tends to reduce the step size when the gradient is large and increase it when the gradient is small [4]. This can help prevent the optimization algorithm from taking overly large steps that could push it past a local optimum, while also allowing the algorithm to accelerate when it is far from an optimum.

Because of the stochastic nature of the step size calculation by batch, high variance regarding the components inside the step size quadratic function is to be expected; to address that, exponential moving average (EMA) is taken for the quadratic component a and the linear component b. The resulting step size is also averaged and multiplied by a constant c, given the reduction of the EMAs.

Also, given that there is no small constant in the denominator where the gradient is squared, typically referred to as ε in adaptive methods [7], it could lead to an increase in the step over time if the gradient is too small and eventually diverge, which is why a maximum step size and a decreasing factor are proposed in the algorithm. When the algorithm reaches a maximum step size α_{max}, the reduction of the step takes place in a magnitude inverse of the square root of the epoch.

3.2 Algorithm implementation

This section explains the derivation of the algorithm used to calculate the ISSE step size $\alpha_{ISSE} = \left| \frac{\sum_{l \neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \langle \mathbf{x}_n, \nabla_l \rangle}{\sum_{l \neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \langle \mathbf{x}_n, \nabla_l \rangle^2} \right|$ depicted in (48). For that purpose, the following variables are defined:

- $\mathbf{w}$: Weight parameters adjusted during training to minimize the loss.
- $\mathbf{x}$: Data points fed into the model.
- ∇ : Loss function gradient.
- l : Class identifier.
- y_n : Corresponding class or label.
- α_{max} : Maximum step size.
- c : Constant multiplier.
- γ : Exponential moving average factor.

As can be seen, some calculations involving the numerator b and denominator a are necessary. Given the matrix nature of the problem, a direct calculation of the variable a is represented in Figure 5, there $\mathbf{X}_{k\times R}$ represents the input data of k features and batch size R, and $\mathbf{W}_{k\times L}$ the weight matrix for L classes.

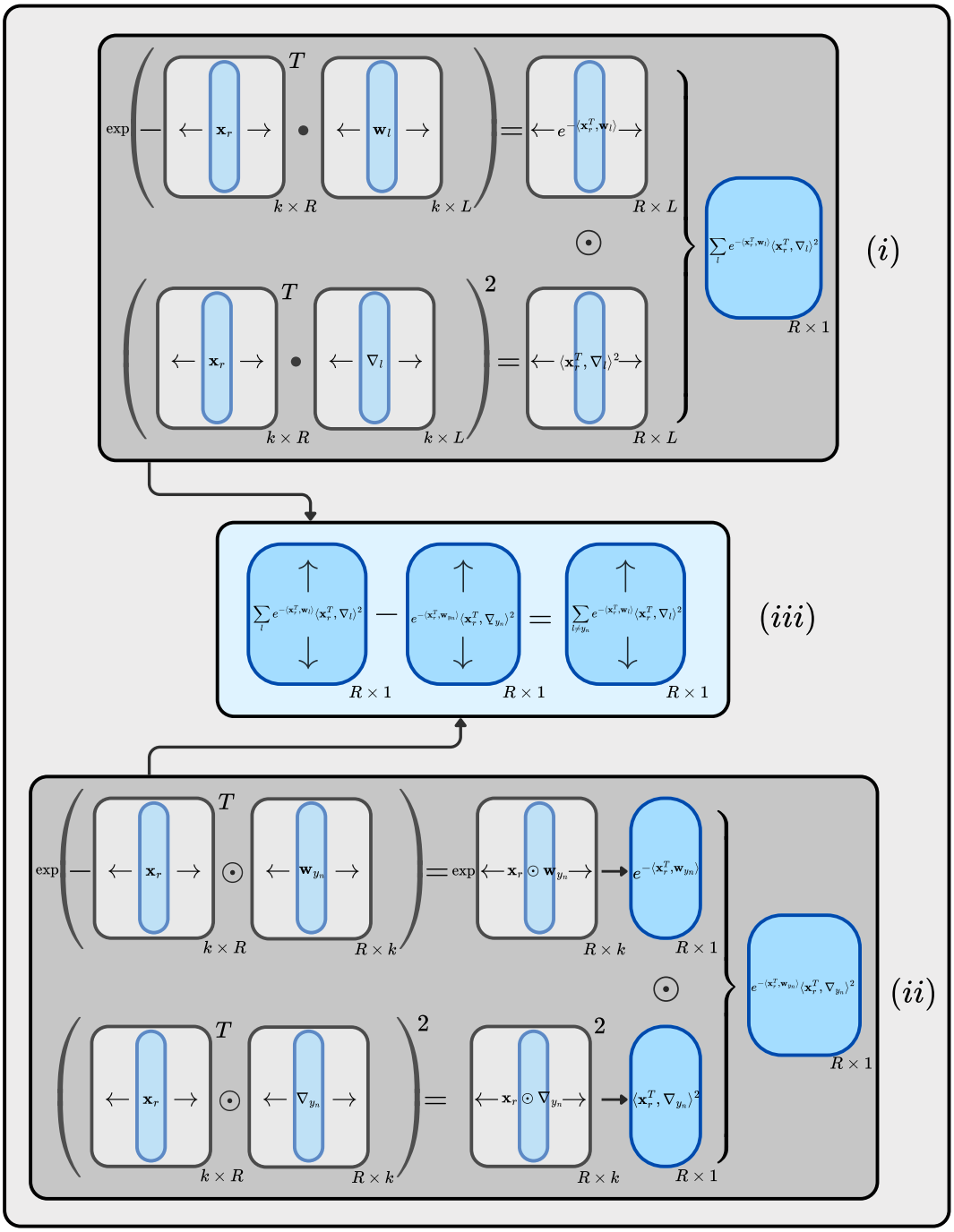


Figure 5: Matrix operations required to compute the ISSE step size.

To generate the algorithm, first small terms as $\sum_{l\neq y_n} e^{-\langle \mathbf{x}_r^T \mathbf{w}_l \rangle}$ are determined and then joined and computed. It is necessary to find the exponential of the negative of inner product of $\mathbf{X}$ and $\mathbf{W}$, then, by adding values alongside the horizontal axis, the summation of the exponential of $\mathbf{X}$ and $\mathbf{W}$ is obtained, which gives $\sum_l e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \langle \mathbf{x}_n, \nabla_l \rangle^2$, as seen in (i). Then, to find $e^{-\langle \mathbf{x}_n, \mathbf{w}_{y_n} \rangle} \langle \mathbf{x}_n, \nabla_{y_n} \rangle^2$, element-wise multiplication, multiplying each of the date with is respective weight is performed by $\mathbf{X}$ and $\mathbf{W}_{y_n}$, the sum of the elements of the matrix is as seen in (ii). Finally, both values are subtracted to find $\sum_l e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \langle \mathbf{x}_n, \nabla_l \rangle^2 - e^{-\langle \mathbf{x}_n, \mathbf{w}_{y_n} \rangle} \langle \mathbf{x}_n, \nabla_{y_n} \rangle^2 = \sum_{l\neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \langle \mathbf{x}_n, \nabla_l \rangle^2$ in (iii). A vector of $\mathbb{R}^{R\times 1}$ is obtained. Same procedure can be followed to obtain b, without the squared inner product.

With theses calculations, the ISSE step size is computed at each iteration inside an epoch, following the proposed Algorithm 1.

Algorithm 1 SGD ISSE for the Softmax

Require: $\mathbf{w}_0, \alpha_{\max}, c, \gamma,$ epoch

for $k = 0$ to K **do**
  $a = \sum_{l\neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \langle \mathbf{x}_n, \nabla_l \rangle^2$, $\hat{a} = \gamma \hat{a} + (1-\gamma) a$
  $b = \sum_{l\neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \langle \mathbf{x}_n, \nabla_l \rangle$, $\hat{b} = \gamma \hat{b} + (1-\gamma) b$
  $\alpha = |\frac{\hat{b}}{\hat{a}}|$
  $\hat{\alpha} = c \cdot (\gamma \cdot \hat{\alpha} + (1-\gamma) \cdot \alpha)$
  if $(\hat{\alpha} > \alpha_{\max})$ OR max_step **then**
    max_step = True
    $\hat{\alpha} = \frac{\alpha_{\max}}{\sqrt{\text{epoch} - \text{epoch}_{\alpha_{\max}}}}$
  end if
  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \hat{\alpha} \cdot \nabla_k$
end for
return $\mathbf{w}_{k+1}$

The SGD algorithm is coded with the calculated step size, training with datasets for classification with multinomial logistic regression. Finally, a comparison is generated between the developed method and adaptive algorithms currently used for optimization methods, as seen in the next chapter.

Chapter 4

Experimental Results

To test the proposed step size in MLR, the experiments were structured as follows:

- Optimizers to be assessed:
  - Vanilla SGD.
  - SGD with momentum.
  - SGD with Barzilai-Borwein step size [25].
  - SGD with Lipschitz constant [26].
  - The Adam algorithm [6].
  - ISSE step size (own).
- Metrics:
  - Trainining and testing loss function.
  - Trainining and testing accuracy.
- Models employed:
  - ResNet-18 [13].
  - DenseNet-121 [14].
- Datasets used:
  - CIFAR-10 [11]: A dataset of 60,000 32x32 color images in 10 classes, with 6,000 images per class, commonly used for object recognition tasks.
  - CIFAR-100 [11]: An extension of CIFAR-10 with 100 classes containing 600 images each, providing more detailed classification.

- SVHN [12]: The SVHN dataset consists of over 600,000 labeled digits collected from house numbers, useful for digit recognition tasks.

Code is available at https://github.com/gabrielrori/ISSE-Step-Size.

For all the optimizers, weight decay was applied with a value of $5 \cdot 10^{-4}$ and learning rate scheduling [33] with a reduction of 0.1 in the 100th epoch. For the ISSE step size, a value of γ between 0.99 and 0.999 was considered, a scalar $0 < c < 10$ to counter the EMA reductions and $\alpha_{\max} = 0.25$.

The implementation was carried out on a computer with an i7 12700k processor, 64 GB of DDR4 RAM, and a Nvidia RTX 4090 graphics card with 24 GB of VRAM. The algorithm was tested in Ubuntu 22.04 operating system with Python 3.10.

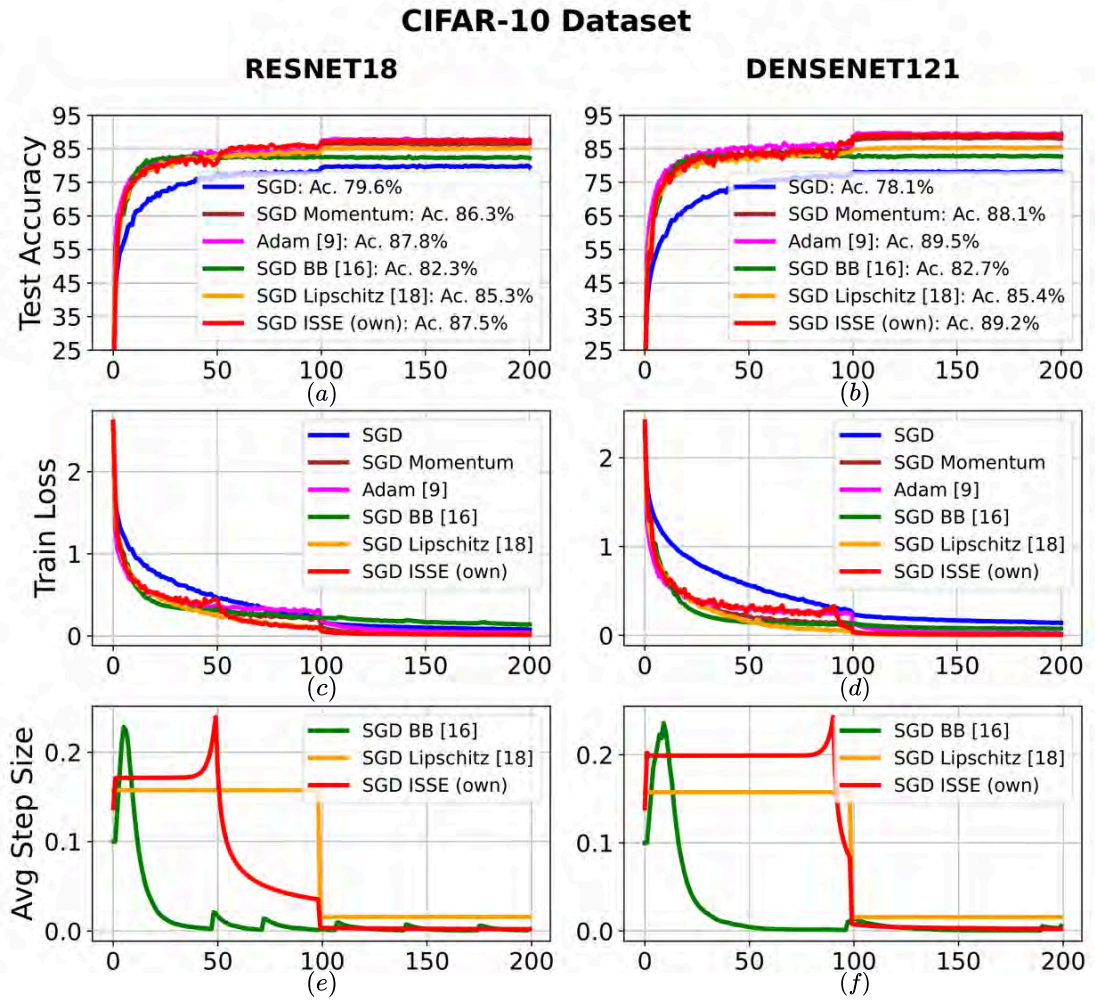


Figure 6: Performance Metrics for CIFAR-10 Dataset by epoch. Rows represent metrics: (a) and (b) training loss, (c) and (d) test accuracy, (e) and (f) step sizes. Columns represent architectures: (left) ResNet-18 and (right) DenseNet-121.

Fig. 6. represents the algorithm implemented for the CIFAR-10 dataset, Fig. 7. for the CIFAR-100 dataset and Fig. 8 for the SVHN dataset.

As seen from the figures, the proposed SGD ISSE optimizer demonstrates superior performance when compared to other adaptive step sizes, vanilla SGD and SGD with momentum. While SGD Lipschitz keeps an almost invariant (it evolves in an order of magnitude around 10^{-5}), the Barzilai step size exhibits similar behavior to ISSE step size, increasing the step for early iterations and then decreasing it, nevertheless, the step size can sometimes peak excessively high or drop too low, rendering it counterproductive.

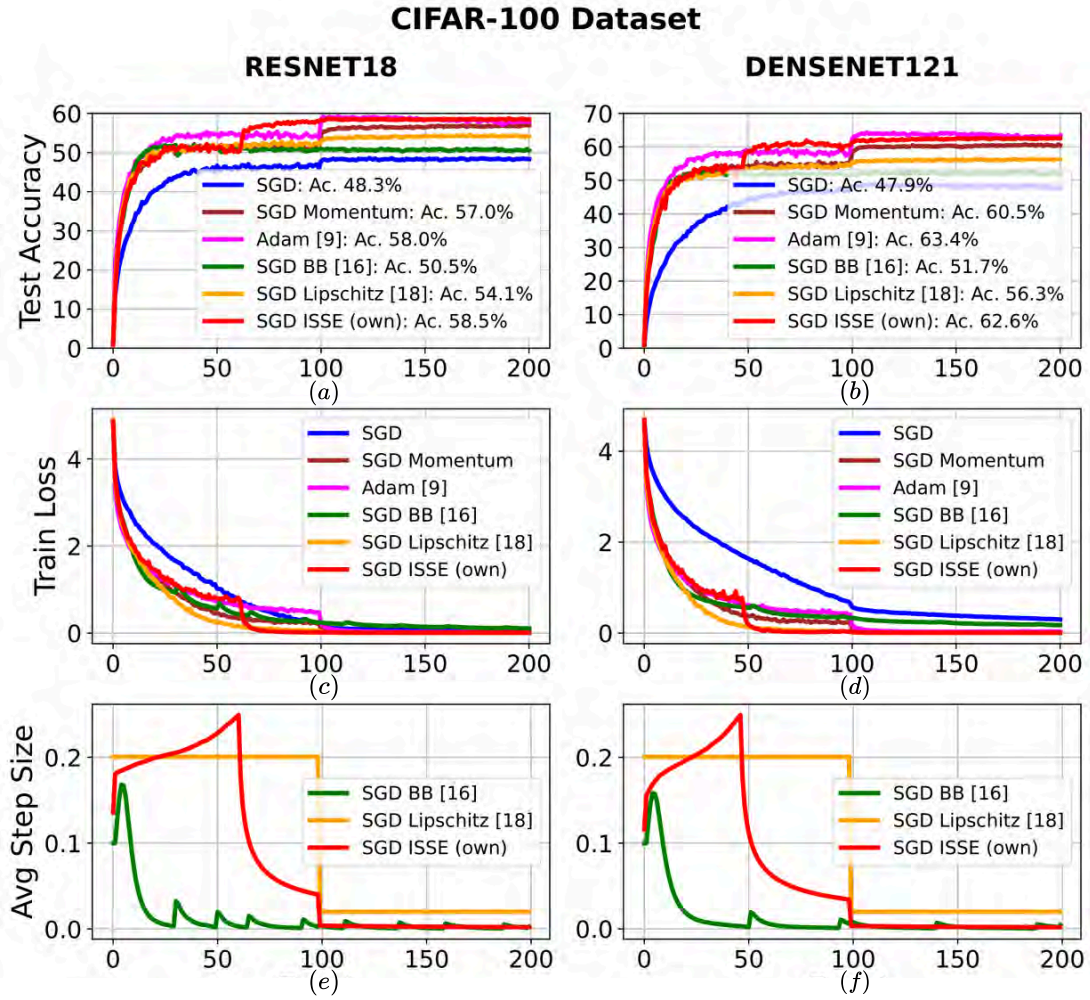


Figure 7: Performance Metrics for CIFAR-100 Dataset by epoch. Rows represent metrics: (a) and (b) training loss, (c) and (d) test accuracy, (e) and (f) step sizes. Columns represent architectures: (left) ResNet-18 and (right) DenseNet-121.

Remarkably, the SGD ISSE optimizer exhibits performance that is comparable to Adam, which is widely regarded as one of the most effective optimizers. This is especially

noteworthy considering that SGD ISSE operates with a uniform step size. Integrating features from Adam, such as variance reduction and momentum based updates, could further enhance its effectiveness.

ISSE step size has a higher learning rate in the initial stages of training which can significantly enhance optimization, it allows the model to converge more rapidly by enabling the optimizer to bypass suboptimal local minima and saddle points [34], increasing the likelihood of more optimal solutions faster.

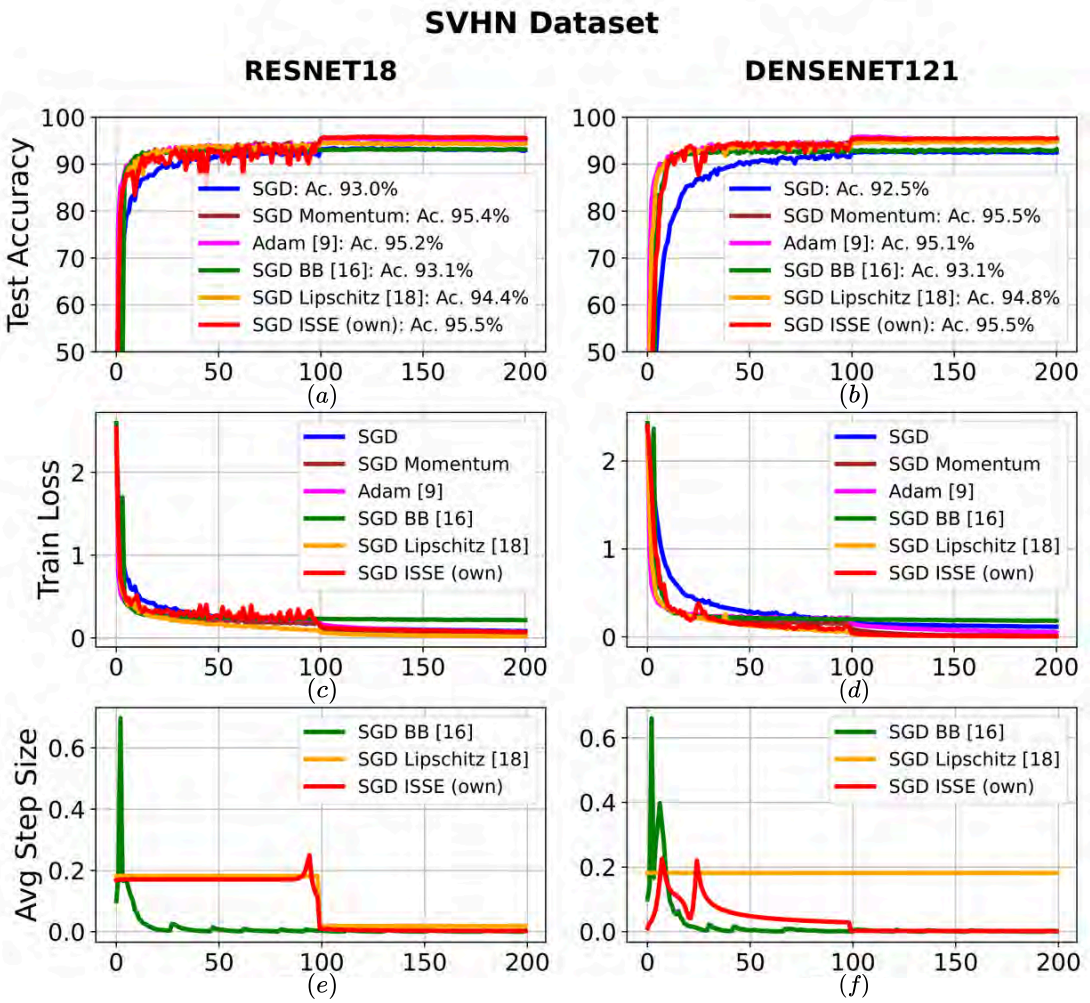


Figure 8: Performance Metrics for SVHN Dataset by epoch. Rows represent metrics: (a) and (b) training loss, (c) and (d) test accuracy, and (e) and (f) step sizes. Columns represent architectures: (left) ResNet-18 and (right) DenseNet-121.

The step size in SGD ISSE is observed to increase until it reaches a predefined upper limit, denoted as α_{max}, beyond this point, the step size begins to decrease. Reducing the learning rate in the later stages of training enables more precise fine tuning of the model

parameters [33], facilitating convergence to potentially improved minima. This adaptive approach ensures that as the model nears an optimal solution, the optimizer is able to make adjustments improving the model performance. At epoch 100, there is another observable reduction in the step size, this adjustment is due to a learning rate scheduling.

Additionally, as seen in Fig. 9, SGD, Adam and SGD ISSE exhibit comparable training epoch times. In neural networks training, the predominant computational demands are attributed to backpropagation and gradient calculations [35] this substantial portion of the training process supersedes the complexity differences in the update rules mechanisms of the optimizers. Thus, even with the ISSE step size calculation in the last layers, its impact on epoch time is minimal compared to SGD and Adam.

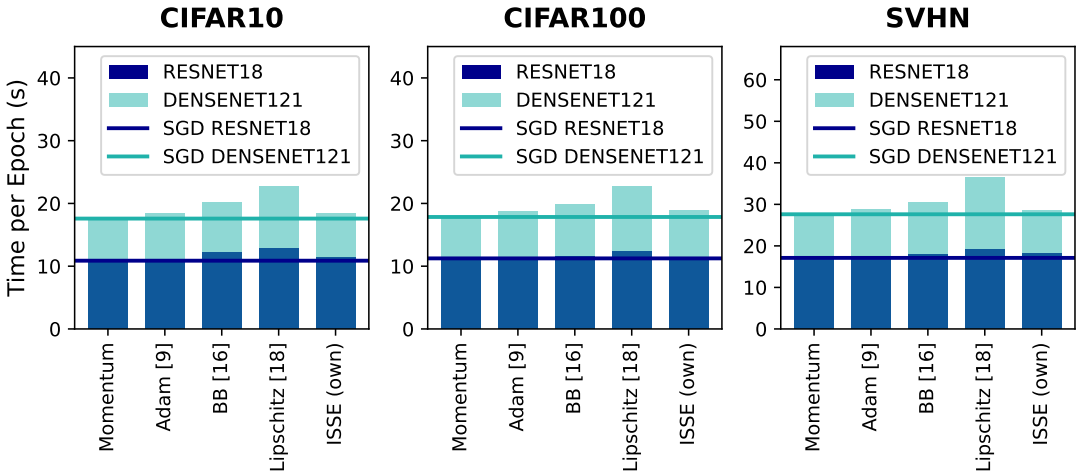


Figure 9: Time per epoch in training for the optimizers compared to vanilla SGD in ResNet-18 and DenseNet-121. Columns represent the CIFAR-10, CIFAR-100 and SVHN datasets.

Conclusions

In this study, we introduced a new technique for automatically determining the optimal step size for Stochastic Gradient Descent (SGD) when applied to Multiple Linear Regression. Our proposed approach leverages the first-order derivative of the loss function to calculate an adaptive step size dynamically. We assessed the effectiveness of our method through experiments on neural network architectures including ResNet-18 and DenseNet-121, utilizing datasets such as CIFAR-10, CIFAR-100, and SVHN.

The empirical results indicate that our Ideal Step Size Estimation (ISSE) method surpasses existing adaptive step size techniques used in the stochastic context, such as Barzilai-Borwein (BB) and methods based on the Lipschitz constant, particularly in terms of the convergence rate. Notably, ISSE not only outperformed standard SGD and SGD with momentum in acceleration of convergence but also demonstrated performance on par with the Adam optimizer.

The core ideas of this thesis were published at the IEEE conference LASCAS'24 [10] and, looking ahead, our future research will explore the application of this algorithm across a more extensive array of neural network models and datasets. We also aim to develop capabilities to generate adaptive step sizes tailored to different loss functions and convolutional neural network (CNN) models. This ongoing research will potentially broaden the applicability of adaptive step size strategies in optimizing neural network training processes.

References

[1] B. Mahesh, "Machine learning algorithms - a review," in *International Journal of Science and Research (IJSR)*, 2020.

[2] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. a. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf

[3] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A survey of optimization methods from a machine learning perspective," 2019. [Online]. Available: https://arxiv.org/abs/1906.06821

[4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," in *J. Mach. Learn. Res.*, 2011.

[5] G. Hinton, "Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude," https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2012, coursera Lecture.

[6] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2014.

[7] S. Ruder, "An overview of gradient descent optimization algorithms," in *CoRR*, 2016.

[8] J. Long, *Regression Models for Categorical and Limited Dependent Variables*, ser. Advanced Quantitative Techniques in the Social Sciences. SAGE Publications, 1997. [Online]. Available: https://books.google.com.pe/books?id=CHvSWpAyhdIC

[9] A. Cauchy, "Methode generale pour la resolution des systemes d'equations simultanees," in *C.R. Acad. Sci. Paris*, 1847, pp. 536–538.

[10] *2024 Latin American Symposium on Circuits and Systems Proceedings*. Punta del Este, Uruguay: IEEE, 2024.

[11] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[12] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[14] G. Huang, Z. Liu, L. van der Maaten, and K. Weinberger, "Densely connected convolutional networks," 07 2017.

[15] X. Yan and X. Su, *Linear Regression Analysis: Theory and Computing*, ser. G - Reference,Information and Interdisciplinary Subjects Series. World Scientific, 2009. [Online]. Available: https://books.google.com.pe/books?id=5pdpDQAAQBAJ

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[18] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks*. Cambridge University Press, 1998.

[19] B. Polyak, "Some methods of speeding up the convergence of iteration methods," in *USSR Computational Mathematics and Mathematical Physics*, 1964, pp. 1–17.

[20] Y. E. Nesterov, "A method of solving a convex programming problem with convergence rate $0(1/k^2)$," in *Soviet Mathematics. Doklady*, 1983, pp. 372–376.

[21] J. Nocedal and S. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. [Online]. Available: https://books.google.com.pe/books?id=VbHYoSyelFcC

[22] A. S. Sebag, M. Schoenauer, and M. Sebag, "Stochastic gradient descent: Going as fast as possible but not faster," *ArXiv*, vol. abs/1709.01427, 2017.

[23] J. BARZILAI and J. M. BORWEIN, "Two-Point Step Size Gradient Methods," *IMA Journal of Numerical Analysis*, vol. 8, no. 1, pp. 141–148, 01 1988.

[24] N. K. Vishnoi, *Algorithms for Convex Optimization*. Cambridge University Press, 2021.

[25] C. Tan, S. Ma, Y.-H. Dai, and Y. Qian, "Barzilai borwein step size for stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2016.

[26] R. Yedida, S. Saha, and T. Prashanth, "Lipschitzlr: Using theoretically computed adaptive learning rates for fast convergence," *Applied Intelligence*, vol. 51, no. 3, pp. 1460–1478, 3 2021. [Online]. Available: https://doi.org/10.1007/s10489-020-01892-0

[27] S. Ray, "A quick review of machine learning algorithms," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 35–39.

[28] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[29] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees," *Wadsworth International Group*, vol. 37, no. 15, pp. 237–251, 1984.

[30] S. Gortmaker, D. Hosmer, and S. Lemeshow, "Applied logistic regression," in *Contemp Sociol.*, 2013.

[31] B. Taylor and W.), *Methodus incrementorum directa & inversa. Auctore Brook Taylor....* typis Pearsonianis, 1715. [Online]. Available: https://books.google.com.pe/books?id=Kb46vgAACAAJ

[32] J. Wu, W. Hu, H. Xiong, J. Huan, V. Braverman, and Z. Zhu, "On the noisy gradient descent that generalizes as sgd," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20. JMLR.org, 2020.

[33] L. Smith, "Cyclical learning rates for training neural networks," 03 2017, pp. 464–472.

[34] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with warm restarts," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=Skq89Scxx

[35] S. Wiedemann, T. Mehari, K. Kepp, and W. Samek, "Dithered backprop: A sparse and quantized backpropagation algorithm for more efficient deep neural network training," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 3096–3104.