# Ideal Step Size Estimation for the Multinomial Logistic Regression

Gabriel Ramirez
*Department of Electrical Engineering*
*Pontificia Universidad Catolica del Peru*
Lima, Peru
gabriel.ramirez@pucp.edu.pe

Paul Rodriguez
*Department of Electrical Engineering.*
*Pontificia Universidad Catolica del Peru*
Lima, Peru
prodrig@pucp.edu.pe

***Abstract*—At the core of deep learning optimization problems reside algorithms such as 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.**

***Index Terms*—Deep learning, stochastic gradient descent, adaptive step size, multinomial logistic regression.**

## I. Introduction

Deep learning has become an increasingly popular field in the development of highly accurate models for a wide range of applications in recent years [1]. 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. One of the key factors that contributes to the success of deep learning algorithms is the optimization of the model parameters using efficient algorithms such as stochastic gradient descent (SGD) [2], one of the most widely used algorithms for optimization and the stochastic approximation of gradient descent (GD).

$$F(\mathbf{u}) = \frac{1}{N}\sum_{n=1}^{N} f_n(\mathbf{u}), \quad \mathbf{g}_k = \frac{1}{\#\mathcal{B}_k}\sum_{n\in\mathcal{B}_k} \nabla f_n(\mathbf{u}) \qquad \text{(1a,1b)}$$

Given a loss function as in (1a), SGD uses a random susbset (see (1b)) at each iteration to estimate the true gradient, which in its original formulation, is then used to update the solution as in (2):

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha(\alpha_0, k, "sch") \cdot \mathbf{g}_k \qquad (2)$$

~~This algorithm uses a subset of the data per iteration to estimate the gradient of the cost function, which is then used to update the parameters.~~

reads as a repetition of the previous paragraph

However, the performance of SGD is highly dependent on the choice of the learning rate ($\alpha_0$ in (2)), as well as on its associated scheduling [3] represented by "sch" in (2), which determines the step size taken during the optimization process. Selecting an appropriate learning rate is a challenging task as it can affect the convergence speed and the final performance of the model.

Adaptive algorithms [4] 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, these algorithms incorporate gradient information 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 $\{\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_k\}$.

The multinomial logistic regression (MLR) [5], also known as softmax regression, is a well-known model for multiclass classification, which arguably has better performance than SVMs [6] and decision trees [7]. It is robust to outliers, noisy data, and is differentiable, making it a popular choice in various fields such as image recognition, natural language processing, and speech recognition.

This study proposes a novel adaptive method for the softmax regression that calculates an approximation of the Cauchy step size [8] in the last layer. An algorithm to compute the automatic step size is developed for MLR, then the performance of the algorithm is assessed by comparing it with vanilla SGD and its variants. Our experiment results validate that the SGD along with our proposed method has comparative performance (test accuracy as well as the rate of convergence) with state-of-the-art adaptive methods. Our experiment results support our main claim: SGD along with our proposed method has better performance (final test accuracy as well as the rate of convergence) than other existing adaptive step sizes, as well as comparative performance with adaptive methods like Adam [9].

In Section 2, similar methods regarding adaptive step sizes for SGD are presented. In Section 3, the proposed method is presented and an algorithm is implemented. In Section 4, the performance of the algorithm is compared with other adaptive step sizes and adaptive methods, employing the CIFAR-10

[10], CIFAR-100 [10], and SVHN [11] datasets in conjunction with the ResNet-18 [12] and DenseNet-121 [13] convolutional neural networks (CNNs). Finally, in Section 5 the conclusions are presented and potential improvements for the proposed algorithm are discussed.

## II. Related work

### A. Optimization algorithms: GD and SGD

Due to memory limitations, the calculation of the total gradient of the dataset is not typically feasible, therefore a random subset is used by each iteration in SGD [2]. In this context, given the loss function (1a), the gradient is approximated by (1b), where the random subset $\mathcal{B}_k$ is called "batch", and the variable $\mathbf{w}$ represents the parameters to be optimized.

Taking into account the step size $\alpha_k$ which may be constant or adaptive (see Section II.B for alternative in the non-stochastic case) the key difference between GD and SGD are highlighted by (3) and (4), where $\mathbf{g}_k$ is defined in (1b):

$$\text{GD} : \mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \cdot \nabla F(\mathbf{w}_k) \tag{3}$$

$$\text{SGD} : \mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \cdot \mathbf{g}_k \tag{4}$$

### B. Adaptive Step Sizes

The choice of an appropriate step size for executing SGD algorithms is one of the main challenges in machine and deep learning [4]. Since SGD does not work with the conventional line search methodology, it is common in SGD to either use a diminishing step size or manually adjust a fixed step size [14]. Conventional adaptive step size strategies such as Cauchy [8], Barzilai-Borwein [15], Lipschitz-based [16], and others, which are typically applied in the GD context, cannot be directly applied to SGD as they exhibit significant differences and need to be modified in order to fit the model.

SGD

*1) Barzilai-Borwein Step Size for ~~Stochastic Gradient Descent~~:* The use of Barzilai-Borwein (BB) [15] step size is proposed to automatically generate step sizes. The BB method minimizes the difference between the current and previous iterations from the gradient and objective variables as follows:

$$\arg\min_{\alpha} \|(\mathbf{x}_k - \mathbf{x}_{k-1}) - \alpha(\nabla F(\mathbf{x}_k) - \nabla F(\mathbf{x}_{k-1}))\|_2^2 \tag{5}$$

Where the step size obtained is:

$$\alpha_k^{BB} = \frac{(\mathbf{x}_k - \mathbf{x}_{k-1})^T(\nabla F(\mathbf{x}_k) - \nabla F(\mathbf{x}_{k-1}))}{\|\nabla F(\mathbf{x}_k) - \nabla F(\mathbf{x}_{k-1})\|_2^2} \tag{6}$$

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 [17] addresses this problem by computing a summation of the moving average of the gradient each iteration inside an epoch to approximate the true gradient.

*2) Lipschitz constant based step size:* The Lipschitz constant $L$ and the learning rate $\alpha$ are two essential values optimization algorithms, especially gradient-based methods like GD. These two constants are intrinsically related in the sense that they determine the stability and convergence properties of the optimization algorithm [16]:

$$\|f(x) - f(y)\| \leq L\|x - y\| \tag{7}$$

The Lipschitz constant $L$ bounds how much $f$ can change when $x$ changes, it represents a measure of the curvature of the function $f$, a small Lipschitz constant $L$ allows for a larger learning rate $\alpha$, and a large $L$ a smaller $\alpha$.

A common approach involves determining the step size by taking the inverse of the Lipschitz constant. A method for the Lipschitz constant in deep learning models is studied in [18], where it is estimated for MLR.

## III. Proposed Method

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 [18]. 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 $\alpha$ 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, previously labeled as $f_n$ is fully defined in (8), 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 $\nabla_{y_n} = \nabla f_n(\mathbf{w}_{y_n})$.

$$f_n(\mathbf{w}) = \langle \mathbf{x}_n, \mathbf{w}_{y_n} \rangle + \log(\sum_{l=0}^{L-1} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle}) \tag{8}$$

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

$$f_n(\mathbf{w} - \alpha\nabla) = -\alpha\langle \mathbf{x}_n, \nabla_{y_n} \rangle + \log(\sum_{l=0}^{L-1} e^{-\langle \mathbf{x}_n, \mathbf{w}_l - \alpha\nabla_l \rangle}) \tag{9}$$

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

$$\langle \mathbf{x}_n, \nabla_{y_n} \rangle = \frac{\sum_{l=0}^{L-1} \langle \mathbf{x}_n, \nabla_l \rangle e^{-\langle \mathbf{x}_n, \mathbf{w}_l - \alpha\nabla_l \rangle}}{\sum_{l=0}^{L-1} e^{-\langle \mathbf{x}_n, \mathbf{w}_l - \alpha\nabla_l \rangle}} \tag{10}$$

$$\sum_{l \neq y_n} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \cdot e^{\alpha\langle \mathbf{x}_n, \nabla_l \rangle} = \sum_{l \neq y_n} \frac{\langle \mathbf{x}_n, \nabla_l \rangle}{\langle \mathbf{x}_n, \nabla_{y_n} \rangle} e^{-\langle \mathbf{x}_n, \mathbf{w}_l \rangle} \tag{11}$$

Using the Taylor Series to clear the desired value $\alpha$ results in the following quadratic equation:

$$\sum_{l\neq y_n} e^{-\langle \mathbf{x}_n, w_l\rangle}\cdot\left(1+\alpha\langle \mathbf{x}_n,\nabla_l\rangle + \frac{\alpha^2}{2}\langle \mathbf{x}_n,\nabla_l\rangle^2\right)$$
$$= \sum_{l\neq y_n}\frac{\langle \mathbf{x}_n,\nabla_l\rangle}{\langle \mathbf{x}_n,\nabla_{y_n}\rangle} e^{-\langle \mathbf{x}_n,w_l\rangle} \qquad (12)$$

In SGD, which can be interpreted as GD with unbiased noise added to the gradient at each iteration [19], 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:

$$\hat{\alpha} = \left|\frac{\sum_{l\neq y_n} e^{-\langle \mathbf{x}_n,\mathbf{w}_l\rangle}\langle \mathbf{x}_n,\nabla_l\rangle}{\sum_{l\neq y_n} e^{-\langle \mathbf{x}_n,\mathbf{w}_l\rangle}\langle \mathbf{x}_n,\nabla_l\rangle^2}\right| \qquad (13)$$

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 square of the gradient in the denominator serves as a normalizing factor that tends to reduce the step size $\alpha$ when the gradient is large and increase it when the gradient is small [20]. 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.

Given that there is no small constant in the denominator where the gradient is squared, typically referred to as $\epsilon$ in adaptive methods [4], 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.

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 averages (ema$(x,\gamma)$) are 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. When the algorithm reaches a maximum step size, the reduction of the step takes place in a magnitude inverse of the square root of the epoch.

SGD with ISSE step size is computed at each iteration inside an epoch, following the proposed algorithm:

Not previously defined

**Algorithm 1** SGD ISSE

```
Require: $f, \mathbf{w}_0, \alpha_{\max}, c, \gamma,$ epoch
 1: Initialize max_step ← False
 2: for $k = 0$ to $K$ do
 3:     $a = \sum_{l\neq y_n} e^{-\langle x,w_l\rangle}\langle x,\nabla_l\rangle^2$
 4:     $b = \sum_{l\neq y_n} e^{-\langle x,w_l\rangle}\langle x,\nabla_l\rangle$
 5:     $\text{ema}_b, \text{ema}_a = \text{ema}([b,a],\gamma)$          expand this
 6:     $\alpha = |\frac{\text{ema}_b}{\text{ema}_a}|$
 7:     $\hat{\alpha} = c \times \text{ema}(\alpha,\gamma)$
 8:     if $(\hat{\alpha} > \alpha_{\max})$ OR max_step then
 9:         max_step = True
10:         $\hat{\alpha} = \frac{\alpha_{\max}}{\sqrt{\text{epoch}-\text{epoch}_{\alpha_{\max}}}}$
11:     end if
12:     $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \hat{\alpha}\cdot\nabla_k$
13: end for
14: return $\mathbf{w}_{k+1}$
```

## IV. Experimental results

To test the ISSE step size in MLR, the experiments compared it to vanilla SGD, SGD with momentum, SGD with Barzilai-Borwein step size [17], SGD with Lipschitz constant [18] and the Adam algorithm [9]. The loss function and accuracy are computed per epoch employing the ResNet-18 [12] and DenseNet-121 [13] CNNs, and were trained and tested using the CIFAR-10, CIFAR-100, and SVHN datasets. REFs

For all the optimizers, weight decay was applied with a value of $5\cdot 10^{-4}$ and learning rate scheduling with a reduction of 0.1 in the 100th epoch. For the ISSE step size, a value of $\gamma$ 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.

Fig. 1. represents the algorithm implemented for the CIFAR-10 dataset, Fig. 2. for the CIFAR-100 dataset and Fig. 3 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.

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 SGD ISSE effectiveness.

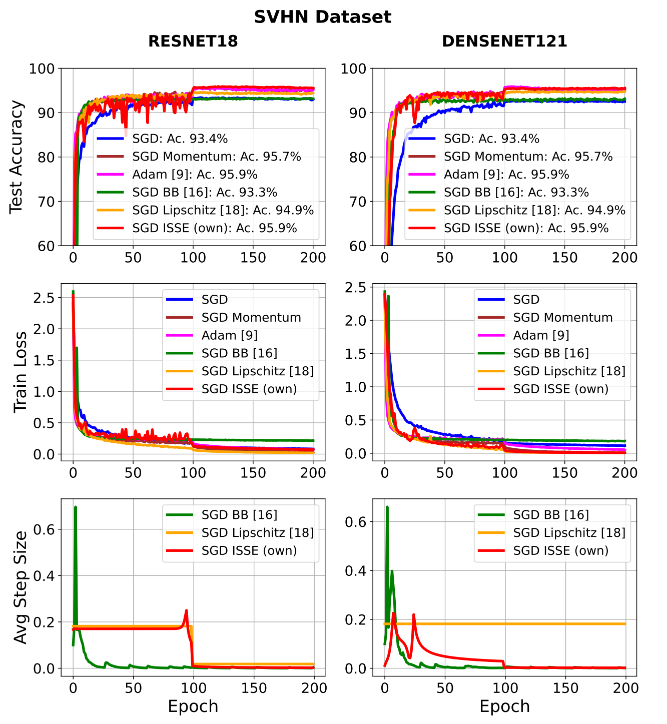

Fig. 1. Performance Metrics for SVHN Dataset. Rows represent metrics: (top) training loss, (middle) test accuracy, and (bottom) step sizes. Columns represent architectures: (left) ResNet-18 and (right) DenseNet-121.

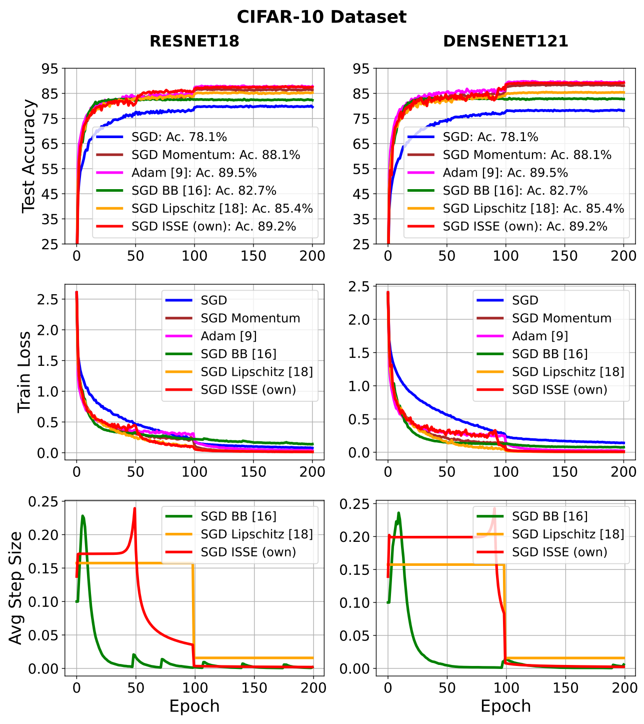

Fig. 2. Performance Metrics for CIFAR-10 Dataset. Rows represent metrics: (top) training loss, (middle) test accuracy, and (bottom) step sizes. Columns represent architectures: (left) ResNet-18 and (right) DenseNet-121.

add a simple comment about time performance

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 [21], increasing the likelihood of more optimal solutions faster.

The step size in SGD ISSE is observed to increase until it reaches a predefined upper limit, denoted as $\alpha_{max}$, 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 [22], 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 attributed to a learning rate scheduling.

Additionally, as seen in Fig. 4, SGD, Adam and SGD ISSE exhibit comparable training epoch durations. In neural network training, the predominant computational demands are attributed to backpropagation and gradient calculations [23], 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.

## V. Conclusions

In this work, we have proposed a novel method for automatically computing an ideal step size for multinomial logistic regression employing SGD. The proposed method uses the first-order derivative of the loss function and computes an adaptive step size. To evaluate the performance of the proposed method, we conducted experiments using ResNet-18 and DenseNet-121 on the CIFAR-10, CIFAR-100 and SVHN datasets using multinomial logistic regression.

The results demonstrated that the ideal step size outperformed other adaptive step size methods like BB and Lipschitz in terms of convergence rate. It outperformed both vanilla SGD and SGD with momentum and had comparable performance to Adam.

Future research aims to test the algorithm on a broader range of networks and datasets, incorporating features from adaptive algorithms such as variance reduction and momentum-based updates, reduce the need for hyperparameters and generate adaptive step sizes for different loss functions and models employing CNNs.

## References

[1] B. Mahesh, "Machine learning algorithms - a review," in *International Journal of Science and Research (IJSR)*, 2020.
[2] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks*. Cambridge University Press, 1998.
[3] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Proceedings of the 31st International Conference on Neural Information*

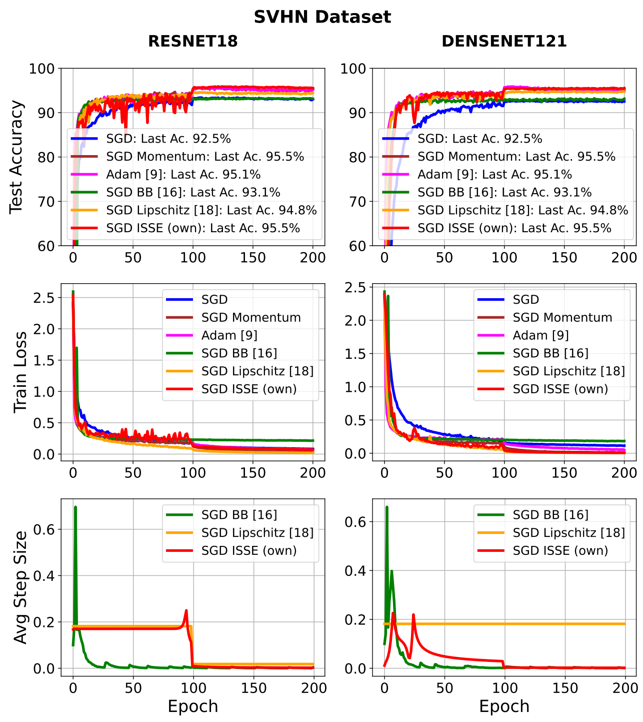

Fig. 3. Performance Metrics for CIFAR-100 Dataset. Rows represent metrics: (top) training loss, (middle) test accuracy, and (bottom) step sizes. Columns represent architectures: (left) ResNet-18 and (right) DenseNet-121.

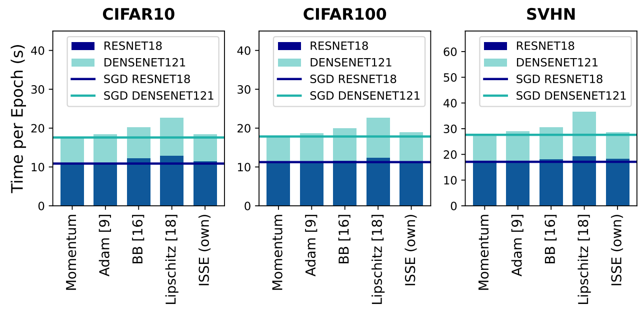

Fig. 4. Time per epoch in training for the optimizers in ResNet-18 and DenseNet-121. Columns represent the CIFAR-10, CIFAR-100 and SVHN datasets

*Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 4151–4161.
[4] S. Ruder, "An overview of gradient descent optimization algorithms," in *CoRR*, 2016.
[5] Y. Ren, P. Zhao, Y. Sheng, D. Yao, and Z. Xu, "Robust softmax regression for multi-class classification with self-paced learning," 08 2017, pp. 2641–2647.
[6] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.
[7] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees," *Wadsworth International Group*, vol. 37, no. 15, pp. 237–251, 1984.
[8] A. Cauchy, "Methode generale pour la resolution des systemes d'equations simultanees," in *C.R. Acad. Sci. Paris*, 1847, pp. 536–538.
[9] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2014.
[10] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.
[11] 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.
[12] 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.
[13] G. Huang, Z. Liu, L. van der Maaten, and K. Weinberger, "Densely connected convolutional networks," 07 2017.
[14] 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.
[15] 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.
[16] N. K. Vishnoi, *Algorithms for Convex Optimization*. Cambridge University Press, 2021.
[17] 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.
[18] 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
[19] 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.
[20] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," in *J. Mach. Learn. Res.*, 2011.
[21] 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
[22] L. Smith, "Cyclical learning rates for training neural networks," 03 2017, pp. 464–472.
[23] 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.