Optimization Methods for Solving Deep Learning Problems: A Case Study of Adaptive Learning Rate Optimizers

Ganiyu A. Saheed *, Muhammed Abdulkabir, Onanusi A. Babajide

Posted Date: 8 April 2026

doi: 10.20944/preprints202604.0443.v1

Keywords:
optimization methods; deep learning; adaptive learning rate optimizer; hyperparameters; Adam; AdaGrad; RMSProp; SGD

 doi:10.20944/preprints202604.0443.v1

Article

Optimization Methods for Solving Deep Learning Problems: A Case Study of Adaptive Learning Rate Optimizers

Ganiyu A. Saheed [1,*], Muhammed Abdulkabir [2] and Onanusi A. Babajide [3]

[1] Program in Applied Mathematics (GIDP), University of Arizona, Tucson, USA
[2] Institute of Mathematics, University of Silesia in Katowice, Katowice, Poland
[3] Department of Science Education, University of Ilorin, Ilorin, Nigeria
* Correspondence: saheedadisa4@gmail.com

Abstract

In this project, we study the optimization methods impacts on deep learning tasks, where we particularly focus on adaptive learning rate optimizers (e.g., AdaGrad, RMSProp, and Adam) and describe them, stating their strengths, weaknesses, and scenarios where they excel or underperform. We employ an experimental approach to analyze their performance, generalization, computational efficiency, and hyperparameter sensitivity. The study compares the performance of adaptive optimizers against a traditional method (SGD) and a non-tuning machine learning model (LDA). Our empirical results show that Adam performs best both on the train and test set in terms of accuracy, speed, generalization, and computational efficiency.

Keywords: optimization methods; deep learning; adaptive learning rate optimizer; hyperparameters; Adam; AdaGrad; RMSProp; SGD

1. Introduction

Optimization has been an important part of neural network (NN) research for an extended period [1], as its contribution to efficient model training and assurance of convergence to an optimal solution can not be overemphasized. It is noticeable that there are some traditional methods, such as Stochastic Gradient Descent (SDG) are popular in use, but adaptive learning rate methods (such as Adam, RMSProp, and AdaGrad) have gained recognition because of their ability to adjust their learning rate in the process of training, which helps in faster convergence, managing noisy gradient better, and lower the training error.

This project will concentrate on adaptive learning rate optimizers, evaluating their adaptability to various deep learning tasks as well as their strengths and limitations. The performance of adaptive optimizers will be contrasted with traditional methods such as SGD through a series of experiments, with an emphasis on their impact on the overall accuracy of the model, generalization, and training speed.

1.1. Problem Description

In this section, we describe an optimization problem for a supervised learning problem. Suppose that we have a data point $x_i \in \mathbb{R}^{d_x}, y_i \in \mathbb{R}^{d_y}, i = 1, \cdots, n$, where n is the number of training samples, x^i is the feature vector of the i th sample (an object, image, etc.), y^i is the corresponding label. Our goal is to predict y_i based on the information of predictors in x_i, which means that we are interested in learning the underlying mapping that maps each x_i to y_i. To approximate the mapping, we use a neural network

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2026 doi:10.20944/preprints202604.0443.v1

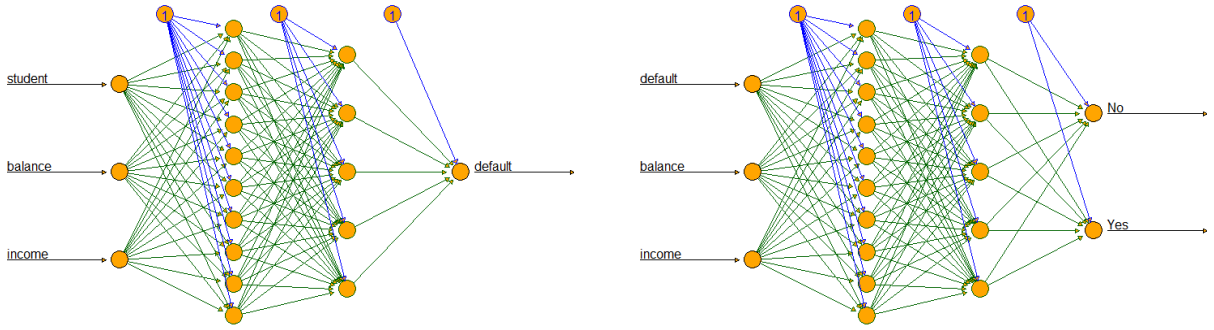
$f_\theta : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$. The general form of the neural network for an L-layer network, the output $f_\theta(x)$ is defined as

$$\begin{aligned} f_\theta(x) &= \text{Output Layer}\left(\text{Hidden Layer}_L(\cdots(\text{Hidden Layer}_1(\text{Input Layer}(X))))\right) \\ &= \begin{cases} \beta_0 + \sum_{k=1}^{m_L} \beta_k A_k^{(L)} & \text{(Regression)} \\ \text{Softmax or Sigmoid}\left(\beta_0 \sum_{k=1}^{m_L} \beta_k A_k^{(L)}\right) & \text{(Classification)} \end{cases} \\ &= \sum_{k=1}^{m_L} \beta_k A_k^{(L)} \quad [\text{ for simplicity }] \end{aligned} \tag{1}$$

where

$$A_k^{(l)} = g\left(w_{k0}^{(l)} + \sum_{j=1}^{m_{l-1}} w_{kj}^{(l)} A_j^{(l-1)}\right) \tag{2}$$

for hidden Layer $l = 1, \ldots, L$, and $A^{(0)} = X$ for input Layer, $g(z)$ is a nonlinear activation function that is declared in advance. θ is a collection of all parameters (β's and w's) need to be estimated from data (see Figure 1).



(a) (b)

Figure 1. This is a neural network diagram having an input layer of three predictors, two hidden layers with 10 and 5 units, respectively, and (a) an output layer of one unit (regression), and (b) an output layer of two units (binary classification) for studentship status.

Now, to predict the $\hat{y}_i = f_\theta(x_i)$ that is close to the true output y_i, we consider the minimization problem.

$$\min_\theta F(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_\theta(x_i)), \tag{3}$$

where $\ell(\cdot, \cdot)$ is a loss function. For regression problems, $\ell(y, z)$ is often chosen to be the quadratic loss function $\ell(y, z) = \|y - z\|^2$. For binary classification problem, a popular choice of ℓ is $\ell(y, z) = \log(1 + \exp(-yz))$

2. Literature Review

In this section, we review some related papers on adaptive learning rate optimizers. We proceed by presenting some basic concepts of those adaptive learning rate methods, where their update rules are provided.

Ruo-Yu Sun discusses the generic optimization methods used in training neural networks, such as stochastic gradient descent and adaptive gradient methods in [1] in recognition of their capability to improve the training of NN. Adaptive learning rate methods such as Adam (Adaptive Moment Estimation), RMSProp (Root Mean Square Propagation), and Adagrad (Adaptive Gradient Methods)

 doi:10.20944/preprints202604.0443.v1

modify the learning rate based on the historical gradients, bringing more effective navigation of the optimization settings. He further notes that there is still a gap in practical performance and theoretical understanding of these methods emphasizing that "bringing theory closer to practice is still a huge challenge for both theoretical and empirical researchers". In a nutshell, adaptive learning rate methods are essential methods to mitigate issues related to gradient explosion/vanishing and improve convergence rate.

Moreover, Diederik P. Kingma and Lei Ba Jimmy [2] introduce the Adam (Adaptive Moment Estimation) method as an adaptive learning rate algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments that effectively combine the strengths of AdaGrad and RMSProp, which make it well-suited for sparse gradients and non-stationary objectives. Their empirical results demonstrate that Adam outperforms traditional optimization methods like SGD and other adaptive methods in consideration of convergence rate and robustness in different machine learning tasks (see Figure 2). Its advantages include low memory requirements and intuitive interpretation of hyperparameters that require little tuning. However, it is noted that it can exhibit instability if the learning rate is not properly adjusted.

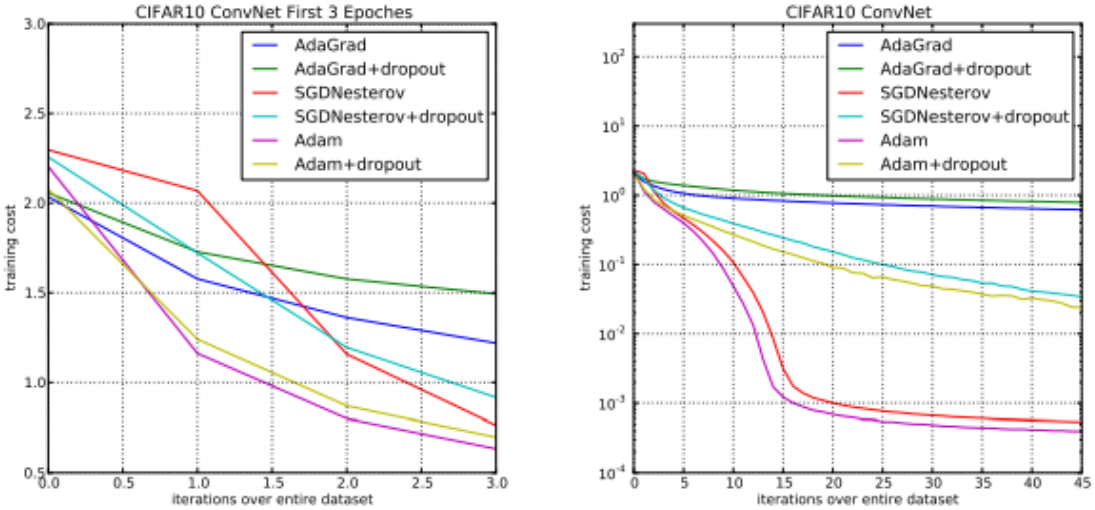


Figure 2. Convolutional neural network training costs. (left) Training cost for the first three epochs. (right) Training costs over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture (Source: Kingma, D. P. [2]).

As noted earlier, optimization is crucial for deep learning training. The traditional optimizer has been SGD coupled momentum because of its simplicity, low computational cost, and good generalizability. But it needs fine-tuning of a constant learning rate, which can cause slow convergence or suboptimal solutions in some cases. Adaptive learning rate optimizers like Adam, RMSProp, and AdaGrad modify the learning rate accordingly based on the gradient history, which often leads to faster convergence, especially when you have a large dataset or in tasks with sparse gradients. We first discuss the gradient descent method (GD) briefly for smooth transitioning of understanding since some of these adaptive learning rate optimizers are based on it

Gradient Descent (GD): A GD method is an iterative process, that starts from an arbitrary point on the loss function and moves down its slope in steps until it reaches the minimum point of the loss function. Its update rule is given as

$$\theta_{t+1} = \theta_t - \alpha_t \nabla F(\theta_t), \tag{4}$$

 doi:10.20944/preprints202604.0443.v1

where α_t is the learning rate and $\nabla F(\theta_t)$ is the corresponding gradient of $F(\theta_t)$. This method converges at a linear rate; its solution is globally optimal for the convex objective function, while its downside is high computational cost as it uses the whole data for gradient computation.

Stochastic Gradient Descent (SGD): SGD was introduced to address the issue of the high computational cost of GD for large-scale data, as it (GD) passes the whole data at once in each iteration, but instead, SGD randomly samples one data point x_i or a mini-batch from the training dataset for each update. Let us rewrite equ(3) as

$$\min_{\theta} F(\theta) \triangleq \frac{1}{B}\sum_{i=1}^{B} F_i(\theta), \tag{5}$$

where $F_i(\theta)$ represents the sum of training loss for a mini-batch of training samples and B is the total number of mini-batches. Its update's rule is given as

$$\theta_{t+1} = \theta_t - \alpha_t \nabla F_i(\theta_t). \tag{6}$$

This method converges at a sublinear rate, which saves computation costs, while its downside is that the solution may be stuck at the saddle point in some cases.

SGD with Momentum: Shiliang Sun et al [3] note that the concept of momentum is derived from the mechanics of physics, which simulates the inertia of objects, and the idea of applying momentum in SGD is to preserve the influence of the previous update direction on the next iteration to a certain degree. When SGD randomly picks i at t-th iteration, its update rule is given as

$$\begin{cases} g_t = \nabla F_i(\theta_t) \\ m_t = \beta m_{t-1} + (1-\beta) g_t, \\ \theta_{t+1} = \theta_t - \alpha_t m_t. \end{cases} \tag{7}$$

Diederik P. Kingma and Lei Ba Jimmy [2] call this method the stochastic versions of the heavy-ball method and accelerated gradient method, which are also known as "momentum methods" in deep learning.

Adaptive Gradient Methods (AdaGrad): AdaGrad is an improvement of SGD [4], it modifies the learning rate dynamically using historical gradients from previous steps. Its update rule is given as

$$\begin{cases} g_t = \nabla F_i(\theta_t) \\ v_t = \sum_{j=1}^{\mathrm{T}} g_j \circ g_j, \\ \theta_{t+1} = \theta_t - \alpha_t v_t^{-1/2} \circ g_t, \quad t = 0, 1, 2, \cdots, \end{cases} \tag{8}$$

where $\circ$ represents the element-wise product. The method is suitable for dealing with sparse gradient problems but is not suitable for dealing with non-convex problems [3].

Root Mean Square Propagation (RMSProp): RMSProp was introduced in order to correct the drawback in AdaGrad and Rprop (Resilient Propagation) [**?**]. This leads to a new definition of v_t. So, at the t-th iteration of RMSProp i is randomly selected and the update rule is given as

$$\begin{cases} g_t = \nabla F_i(\theta_t) \\ v_t = \beta v_{t-1} + (1-\beta) g_t \circ g_t, \\ \theta_{t+1} = \theta_t - \alpha_t v_t^{-1/2} \circ g_t, \quad t = 0, 1, 2, \cdots, \end{cases} \tag{9}$$

 doi:10.20944/preprints202604.0443.v1

This method improves the drawback in the late stage of AdaGarad and is suitable for non-stationary and non-convex optimization problems. But the downside of it is that the update process may be repeated around the local minimum in the late stage.

Adaptive Moment Estimation (Adam): Adam [2] combines RMSProp and momentum methods, It makes use of both first- and second-order momentum estimation of the gradient to modify the step size dynamically of each parameter. Its update rule is given as

$$\begin{cases} g_t = \nabla F_i(\theta_t), \\ m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \\ v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t \circ g_t, \\ \theta_{t+1} = \theta_t - \alpha_t v_t^{-1/2} \circ m_t, \quad t = 0, 1, 2, \cdots, \end{cases} \tag{10}$$

where β_1 and β_2 are exponential decay rates. The Adam method is relatively stable in gradient processes and suitable for most non-convex problems with high-dimensional datasets.

3. Methodology/Algorithm Description

In this section, we first describe our methodology, then we proceed to describe some of the algorithms of adaptive learning rate method discussed in section 2

In our study, we employ an experimental methodology to evaluate the performance of various adaptive optimizers (Adam, RMSProp, and AdaGrad) in comparison to traditional optimizers (SDG) and a non-tuning parameter machine learning method (LDA: linear discriminant analysis) on deep learning tasks. The analysis will concentrate on the efficiency of resources, generalization, and the speed of convergence. We evaluate the performance of optimizers by utilizing handwritten zip-code (see Figure 3), which have training data of 7291×256 dimensions and test data of 2007×256 dimensions. Our tools include R programming (version 2023.09.1 Build494), Keras(Version: 2.13.0), Tensorflow, and convolutional neural networks (CNN) as frameworks for model implementation and experimentation using HP Pavilion x360 Convertible (processor: 8×intel@ corei5-8250U CPU @1.60GHZ, memory: 8.00 GIG of RAM) on Windows 11.

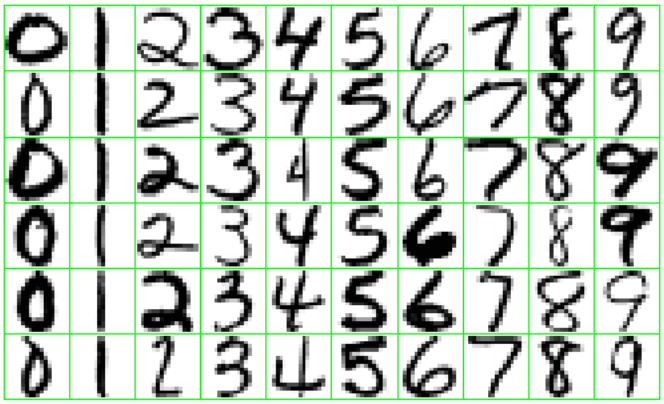


Figure 3. Examples of handwritten digits from U.S. postal envelopes [5].

The following are the description of the algorithms

 doi:10.20944/preprints202604.0443.v1

Algorithm 1 Stochastic Gradient Descent (SGD)

Require: Objective function $F(\theta)$, initial parameters θ_0, learning rate α_t, number of iterations T

Initialize θ_0
for $t = 1$ to T (And θ not converge) **do**
    Randomly select $i \in \{1, 2, \ldots, N\}$
    Compute gradient $g_t = \nabla F_i(\theta_t)$
    Update parameters $\theta_{t+1} = \theta_t - \alpha_t \cdot g_t$
end for
return θ_t

Algorithm 2 AdaGrad General Algorithm

Require: Step size η, stochastic objective function $f(\theta)$, initial parameter vector θ_1

for $t = 1$ to T (And θ not converge) **do**
    Evaluate $f_t(\theta_t)$
    Get and save g_t
    $v_t \leftarrow \sum_{\tau=1}^{t} g_\tau g_\tau^\top$
    $\theta_{t+1} \leftarrow \theta_t - \eta v_t^{-1/2} g_t$
end for
return θ_t

Algorithm 3 RMSProp Algorithm

Require: Step size η, decay rate β, stochastic objective function $f(\theta)$, initial parameter vector θ_1, small constant $\epsilon > 0$

Initialize $v_0 = 0$
for $t = 1$ to T (And θ not converge) **do**
    Evaluate $f_t(\theta_t)$
    Compute gradient $g_t = \nabla f_t(\theta_t)$
    Update the moving average of squared gradients:

$$v_t \leftarrow \beta v_{t-1} + (1 - \beta) g_t \circ g_t$$

    Update parameters:

$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} \circ g_t$$

end for
return θ_t

Algorithm 4 Adam: Stochastic Optimization Algorithm

Require: Step size α, exponential decay rates $\beta_1, \beta_2 \in [0, 1)$, stochastic objective function $f(\theta)$ with parameters θ, initial parameter vector θ_0

Initialize $m_0 = 0$ (First moment vector)
Initialize $v_0 = 0$ (Second moment vector)
Initialize $t = 0$ (Timestep)
while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Compute gradients at timestep t
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ▷ Update biased first moment estimate
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ ▷ Update biased second raw moment estimate
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Bias-corrected first moment estimate
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Bias-corrected second raw moment estimate
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ ▷ Update parameters
end while
return θ_t ▷ Resulting parameters

 doi:10.20944/preprints202604.0443.v1

4. Numerical Experiment

In this section, we present and analyze our numerical experiment results following the methodology described in section 3. Generally, we set our number of epochs to 30 in training CNN in R Programming Language using the Keras library. We will not give much detail about our CNN setup since our focus is on the optimization methods. We tune the important hyperparameters of each discussed optimizer, respectively. Each digit of the handwritten zip code is 16×16 grayscale image. This data was collected by the neural network group at AT&T research labs and they noted that the test set is notoriously "difficult", that any trained model with 2.5% an error rate is excellent [5].

4.1. Experiment on Hyperparameter Tuning

In this subsection, we present results on tuning the combination of some relevant hyperparameters for each discussed optimizer.

Figure 4 shows the test and training accuracy of AdaGrad optimizer by tuning its hyperparameters (learning rate α and batch size) using CNN to classify handwritten zip-code. The test (green) and train (red) accuracy are plotted as a function of learning rate (α) for each batch size. It is noticeable in each batch size plot there is a little discrepancy between train and test accuracy, indicating that there is no overfitting going on, that is, the model performs well on unseen data, and our best-tuned hyperparameters for learning rate and batch size are $\alpha = 0.05$ and 64 respectively with test accuracy of 96.76%.

Figure 5, 6, and 7 account for the test and train accuracy of RMSProp, Adam, and SGD optimizers, respectively; see Table 1 for their respective best-tuned hyperparameters and accuracy. All the optimizers perform well in terms of generalizability and accuracy, but the Adam method performs best.

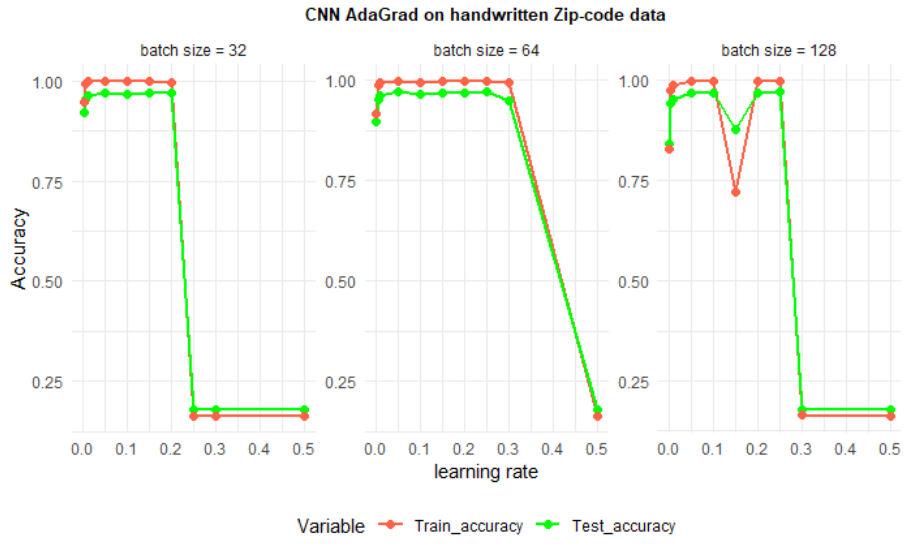


Figure 4. This plot shows the test (green) and training (red) accuracy as a function of the learning rate for each batch size, using the AdaGrad optimizer in CNN to predict handwritten zip codes.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2026 doi:10.20944/preprints202604.0443.v1

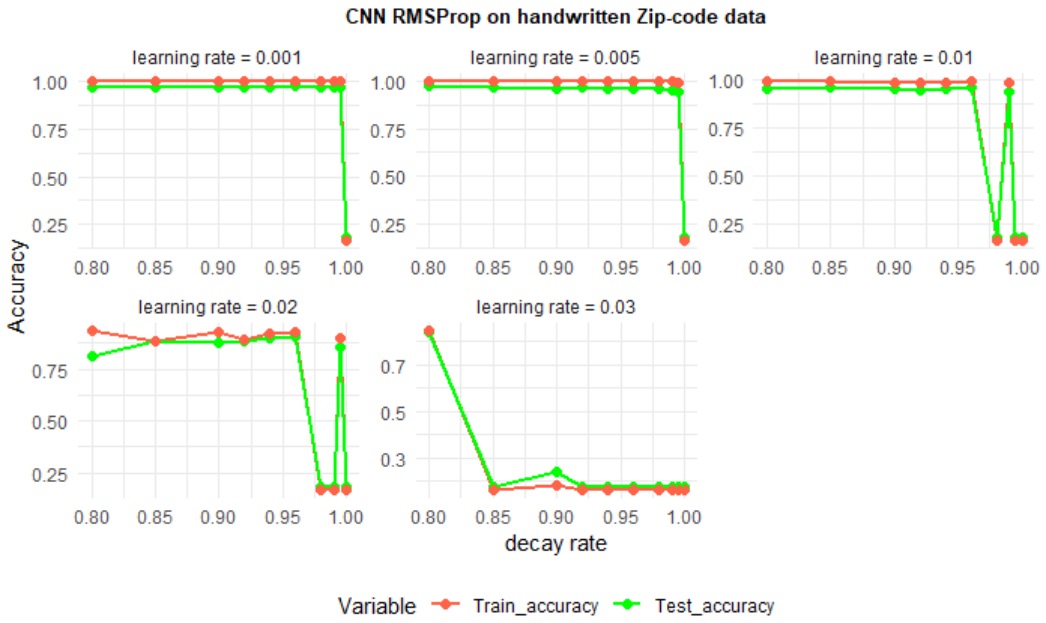


Figure 5. This plot shows the test (green) and training (red) accuracy as a function of the learning rate for each batch size, using the RMSProp optimizer in CNN to predict handwritten zip codes.

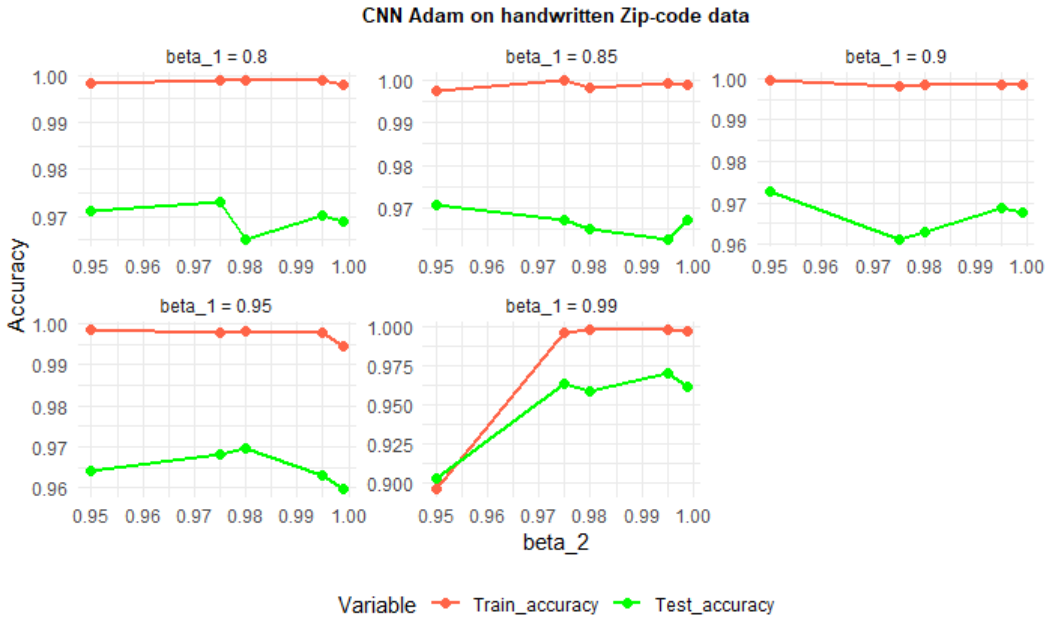


Figure 6. This plot shows the test (green) and training (red) accuracy as a function of learning rate for each batch size, using the Adam optimizer in CNN to predict handwritten zip codes.

 doi:10.20944/preprints202604.0443.v1

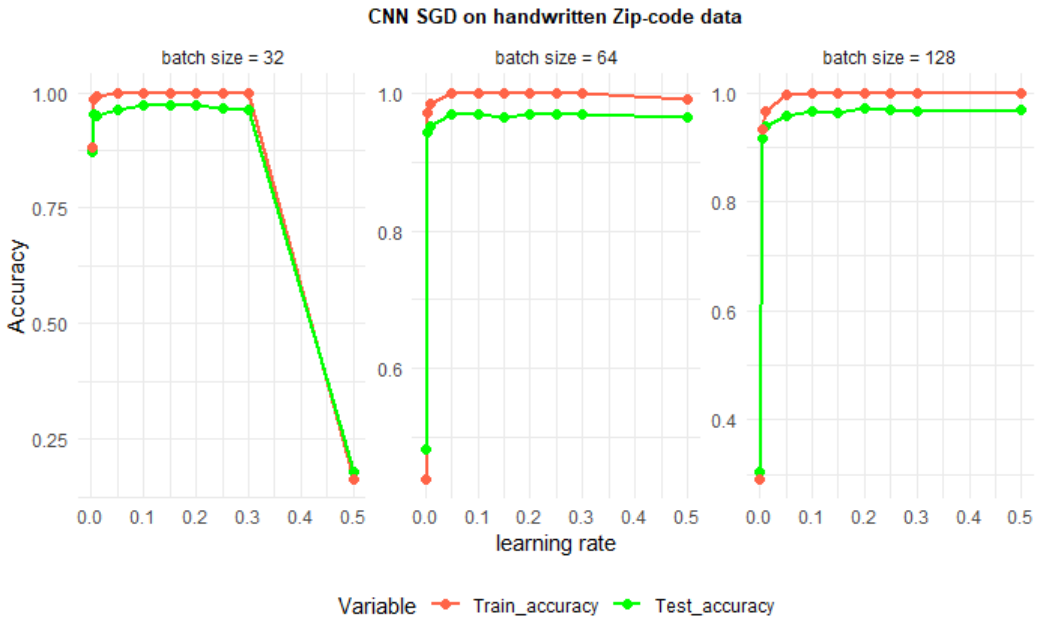


Figure 7. This plot shows the test (green) and training (red) accuracy as a function of learning rate for each batch size, using the SGD optimizer in CNN to predict handwritten zip codes.

4.2. Experiment on Best-Tunned and Default Hyperparameters

Continuing our study, we train our model using our best-tuned and default hyperparameters to compare their accuracy and performance. We plot the test (green) and training (red) accuracy as a function of epochs for each method, and we can see that all the models are robust as their test and train accuracy are closed together for each epoch (see Figure 8 and 9). Table 1 shows the best-tuned (black text) and default (green text) hyperparameters, train and test accuracy, test error, test loss function, and running time (seconds). Considering the test accuracy, all the methods perform better with best-tuned hyperparameters than with default hyperparameters except for the RMSProp which is slightly lower (see Table 1). Generally, the Adam methods give the best performance out of all the considered methods, its test error is 2.99% which is closer to the mentioned excellent error of 2.5% earlier. The last row in the table shows the result of our considered non-tuning machine learning model (LDA). It is very fast in terms of speed but the accuracy is lower than deep learning models.

 doi:10.20944/preprints202604.0443.v1

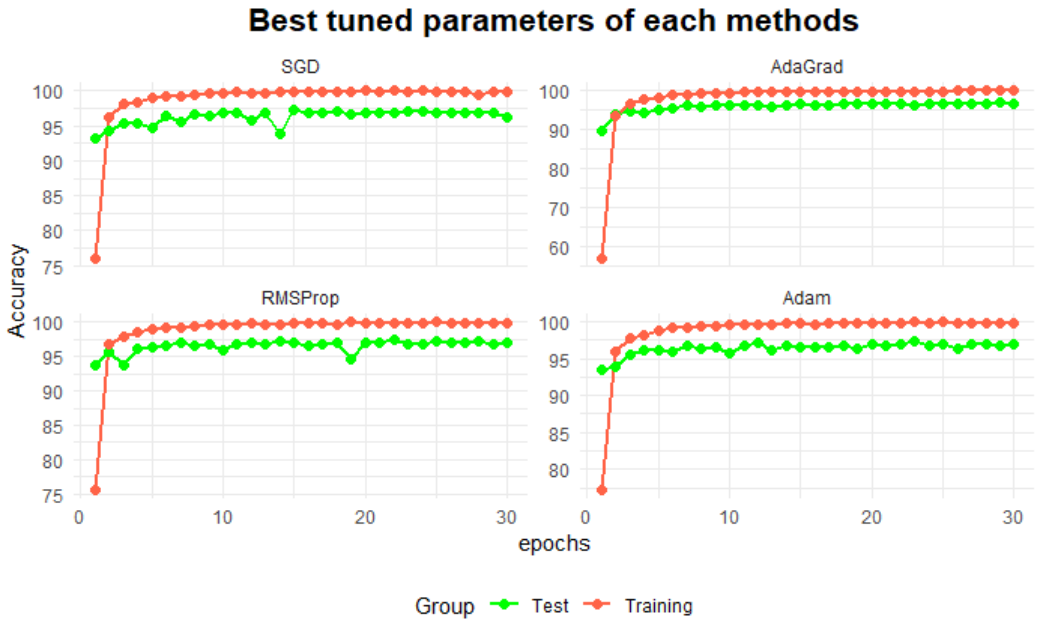


Figure 8. This plot shows the test (green) and training (red) accuracy as a function of epochs for each method with their respective best-tuned parameters in CNN to predict handwritten zip codes.

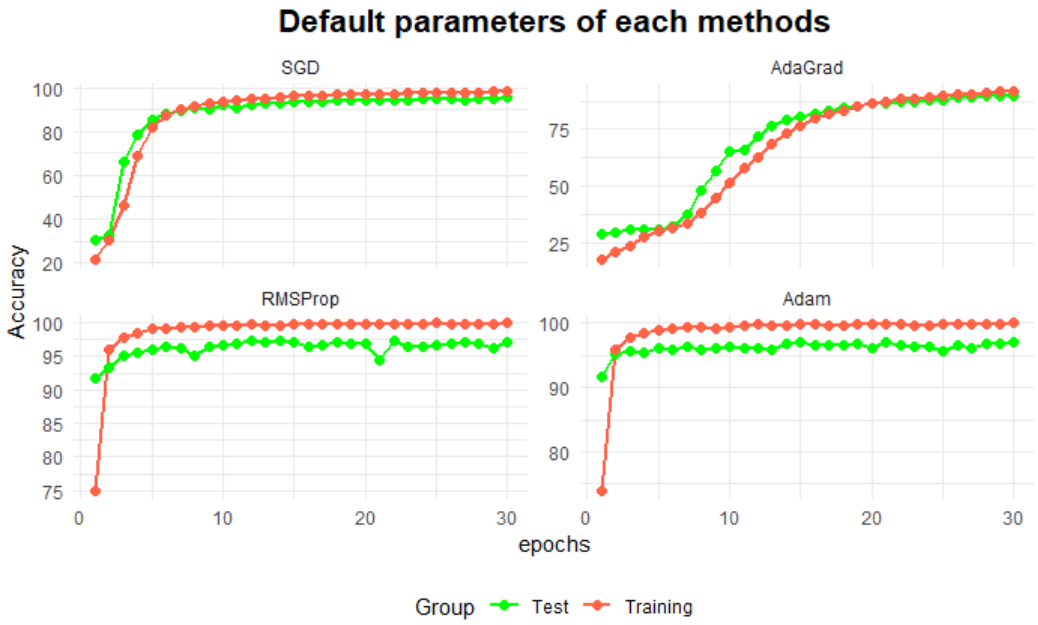


Figure 9. This plot shows the test (green) and training (red) accuracy as a function of epochs for each method with their respective default parameters in CNN to predict handwritten zip codes.

 doi:10.20944/preprints202604.0443.v1

Table 1. This table shows the performance of each method for both best-tuned (in black) and default (in green) parameters. The best value under each column is in bold format.

Methods	Best-tuned/Default parameters	Train accuracy	Test accuracy	Test error	Test loss function	Time(s)
AdaGrad	$\alpha = 0.05, 0.001$ batch size $= 64, 64$	**0.9997** 0.9156	0.9676 0.9008	0.0324 0.0992	0.2007 0.3250	530.67 **478.11**
RMSProp	$\alpha = 0.001, 0.001$ $\beta = 0.96, 0.9$	0.9977 0.9993	0.9691 **0.9716**	0.0309 **0.0284**	0.2259 0.2634	513.33 494.53
Adam	$\alpha = 0.001, 0.001$ $\beta_1 = 0.8, 0.9$ $\beta_2 = 0.975, 0.999$	0.9989 **0.9994**	**0.9701** 0.9691	**0.0299** 0.0309	0.2287 0.2037	**492.42** 487.96
SGD	$\alpha = 0.15, 0.01$ batch size $= 32, 64$	0.9981 0.9827	0.9621 0.9551	0.0379 0.0448	0.2216 0.1801	609.13 488.12
LDA		0.9380	0.8854	0.1145		2.84

5. Concluding Remarks

In our study, we are able to describe each considered adaptive learning rate optimizer, stating their weakness, strengths, and suitability cases. We proceed to explore their hyperparameter effects (such as learning rate, decay rate, etc.). We conclude by analyzing their performance on accuracy, error, convergence speed, and generalization. Our empirical results show that Adam has the best performance in terms of test accuracy, speed, and generalization, which is suitable for optimizing non-stationary and non-convex problems.

There are a number of promising avenues for further investigation into adaptive learning rate optimizers in deep learning that have not been explored in this work. Firstly, consider the theoretical guarantee of convergence of those methods instead of just the empirical approach alone. Secondly, consider a more dense grid for hyperparameter tuning. Lastly, consider tuning-free methods and compare them with those adaptive learning rate methods.

References

1. Sun, R.Y. Optimization for deep learning: An overview. *Journal of the Operations Research Society of China* **2020**, *8*, 249–294.
2. Kingma, D.P.; Jimmy, L.B. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* **2014**.
3. Sun, S.; Cao, Z.; Zhu, H.; Zhao, J. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics* **2019**, *50*, 3668–3681.
4. Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* **2011**, *12*.
5. Hastie, T.; Tibshirani, R.; Friedman, J. The elements of statistical learning: data mining, inference, and prediction, 2017.