ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25aug007

Comparative Study of Deep Learning Optimizers: SGD, Momentum SGD, RMSProp, AMSGrad, Adam, Yogi, and Lion

Ibrahim Khalil Shehada[1]; Rasha Ragheb Atallah[2]; Ashraf Yunis Maghari[3]

[1]Faculty of Information Technology, Islamic University Department of Computer Science Al-Aqsa University, P.O. Box 4051, Gaza, Palestinef Gaza, P.O. Box 108, Gaza, Palestine
[2]Department of Computer System & Technology, Faculty of Computer Science & Information Technology University Malaya Kuala Lumpur, Malaysia, Al-Aqsa University, P.O. Box 4051, Gaza, Palestinef Gaza, P.O. Box 108, Gaza, Palestine
[3]Faculty of Information Technology, Islamic University of Gaza, P.O. Box 108, Gaza, Palestine

Publication Date: 2025/08/19

Abstract: In order to increase prediction accuracy, deep learning models must be trained by adjusting parameters to minimize a loss function. In supervised learning, the mapping between inputs and their right outputs is learned by training models using labeled input examples. In order to minimize mistakes, predictions are compared to actual outcomes, and optimization methods are used to adjust parameters. Until convergence is reached, these algorithms go through several cycles of iteration. Stochastic Gradient Descent (SGD), Momentum SGD, RMSProp, AMSGrad, Adam, Yogi, and Lion are the seven optimization techniques that are evaluated in this study based on training accuracy, test accuracy, training loss, and sensitivity to learning rate. MNIST and CIFAR-10 were the two benchmark datasets used in the experiments. SGD with a learning rate of 0.5 had the best test accuracy of 99.14% and the highest training accuracy of 99.89% on MNIST. With test accuracy of 99.15% and 98%, respectively, at a learning rate of 1e-2, Momentum SGD and Adam likewise demonstrated strong performance. Optimizers like Yogi and Lion, on the other hand, performed competitively at lower learning rates but suffered at higher ones; at 1e-5, Lion's test accuracy was 98.69%. All optimizers displayed comparatively decreased accuracies for CIFAR-10, which was indicative of the dataset's increased complexity. Momentum SGD outperformed other optimizers including Adam, Yogi, and Lion, achieving the highest training accuracy of 98.90% and the best test accuracy of 72.94% at a learning rate of 1e-2. Lion showed better performance and stability on both datasets at a low learning rate of 1e-5. These results highlight how crucial it is to choose learning rates and optimization techniques that are specific to the features of each dataset.

How to Cite: Ibrahim Khalil Shehada; Rasha Ragheb Atallah; Ashraf Yunis Maghari (2025) Comparative Study of Deep Learning Optimizers: SGD, Momentum SGD, RMSProp, AMSGrad, Adam, Yogi, and Lion. *International Journal of Innovative Science and Research Technology*, 10(8), 593-601. https://doi.org/10.38124/ijisrt/25aug007

I. INTRODUCTION

Deep learning models have become increasingly popular due to their remarkable performance in areas like image classification[1], speech recognition, and natural language processing, and they depend significantly on optimization techniques to refine model parameters. Because of their exceptional performance in domains like speech recognition, picture classification, and natural language processing, deep learning models have grown in popularity[2]. However, in order to fine-tune their parameters, these models heavily rely on optimization approaches. These models are trained using an optimization technique that adjusts parameters to minimize a particular loss function, hence increasing the prediction accuracy of the model. The goal of optimization is to improve the model's performance by reducing the discrepancy between expected and actual outcomes. Iteratively, the algorithm gradually modifies the parameters in order to arrive at a solution that ideally strikes a compromise between generalization and accuracy

The model that trained on labeled datasets in supervised learning, where each input is associated with a particular output. In order to reduce the prediction error, the model adjusts its parameters after generating predictions for a particular input and comparing them to the actual outcomes[3]. By using an optimization technique, the model aims to minimize the objective function, sometimes referred to as a loss function, which quantifies the difference between the expected and actual results[4]. The model's efficacy in terms of training time, overall accuracy, and generalizability can be significantly impacted by the optimization algorithm choice.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25aug007

Numerous algorithms are used in the optimization process, and each has certain benefits and drawbacks. Because of their simplicity and efficiency, methods like Stochastic Gradient Descent (SGD) and its variations are well-known and widely used[5]. However, they have drawbacks, including as sluggish rates of convergence and increased susceptibility to hyperparameters, such the learning rate. To address these issues, more advanced techniques such as Momentum, RMSProp, AMSGrad, Adam, Yogi, and Lion have been developed, with changes meant to improve training speed, stability, and the ability to break out of local minima[6]. Even with their extensive use, it is still unclear which optimization technique performs best in different scenarios, particularly when datasets and model topologies vary.

This paper aims to the performance of seven optimization algorithms—SGD, Momentum SGD, RMSProp, AMSGrad, Adam, Yogi, and Lion—on two widely used benchmark datasets—MNIST and CIFAR-10—this work seeks to close this gap. Important parameters including training accuracy, test accuracy, training loss, and the impact of learning rate modifications will be the main focus of the comparison. By carrying out these tests, the study will provide information on the relative benefits and drawbacks of every optimizer, which will help determine which approaches are most suited for different deep learning tasks.

The organization of the paper is as follows: Section II presents a review of the existing literature regarding optimization algorithms and their roles in machine learning and deep learning. This section also emphasizes significant research comparing different optimizers and their theoretical underpinnings. Section III describes the methodology employed in this study, detailing the optimization algorithms that were assessed. Section IV covers the experimental design and findings.

II. SIGNIFICANCE

The effectiveness of deep learning models is largely dependent on the optimization process, and the model's training and generalization are greatly influenced by the optimization algorithm selection. Because of their computational efficiency and ability to minimize the loss function, gradient descent-based algorithms are especially well-liked. However, there are still problems like delayed convergence, hyperparameter sensitivity, and the possibility of being stuck in saddle points or local minima. Numerous optimization strategies have been proposed to address these shortcomings.

Even though conventional techniques like SGD and its momentum-enhanced variants offer simple answers, they frequently encounter problems when working with complex, high-dimensional data sets like pictures or videos, where the loss landscape may contain a large number of local minimum and saddle points. Because they can change the learning rate during training, adaptive optimization techniques like RMSProp, Adam, and its variants have proven more effective in certain circumstances. This helps to improve stability and accelerate convergence. Their efficacy is still dependent on a number of variables, including the learning rate and other hyperparameters. Furthermore, more modern techniques like Yogi and Lion have emerged, promising better training stability and convergence rate; nevertheless, their empirical efficacy in comparison to existing optimizers has not yet been fully explored.

This research intends to methodically evaluate optimization algorithms on two popular datasets, MNIST and CIFAR-10, in light of the large number of optimization algorithms that are now accessible and the absence of a definitive answer on which optimizer is best for specific tasks. This research aims to offer useful insights to help researchers and practitioners choose the best optimization method for their deep learning projects by conducting such an empirical analysis.

III. SCOPE OF RESEARCH

The evaluation of optimization strategies used in deep learning model training for image classification tasks is the main focus of this research. The MNIST and CIFAR-10 datasets were specifically chosen because they are well-known standards that are frequently used in the literature to assess different machine learning models and optimization techniques. The results reported in this research are based on experiments conducted using these datasets, however the knowledge gained may also be applicable to other computer vision and related tasks. Metrics including training and test accuracy, training loss, and the impact of different learning rates are used in the evaluation.

Although other performance metrics, like F1-score or inference speed, may potentially be pertinent, they are not within the purview of this research, which primarily aims to examine algorithm performance in terms of accuracy and stability.

IV. LITERATURE REVIEW

In order to increase accuracy and lower the loss function during model training in machine learning and deep learning, optimization strategies are crucial. These methods were developed to address problems like as local minimum, sluggish convergence, and the difficulty of choosing a learning rate. The optimal algorithm for a given issue has been the subject of numerous studies. We review recent studies that compare the effectiveness of optimization algorithms for various tasks, including their theoretical foundations, actual findings, and relative effectiveness.

Gradient Descent (GD) and its variants form the foundation for most modern optimization algorithms. The primary approaches include Batch Gradient Descent (BGD), Stochastic Gradient Descent (SGD), and Mini-batch GD, each differing in how they use training samples to compute gradients. BGD computes gradients using the entire dataset, making it computationally expensive for large datasets[6]. SGD updates parameters for each individual sample, which

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25aug007

leads to faster convergence but introduces noise. Mini-batch GD strikes a balance between the stability of BGD and the efficiency of SGD[7].

SGD is particularly efficient in learning over-parameterized neural networks by converging to global minima that generalize well. However, it faces challenges with learning rate selection and sensitivity to initialization[6]. To address these limitations, momentum-based approaches were introduced. Momentum SGD accelerates convergence by accumulating a velocity vector in the direction of consistent gradient descent, which effectively dampens oscillations in narrow valleys of the loss landscape[6].

Adaptive methods dynamically adjust learning rates for each parameter, offering improvements over fixed learning rate approaches. The Adagrad method, for example, scales learning rates inversely proportional to the square root of accumulated squared gradients. While effective for sparse data, Adagrad suffers from rapidly diminishing learning rates in dense gradient scenarios.

RMSProp was developed to address Adagrad's learning rate decay issue by using exponential moving averages of squared gradients, ensuring a more consistent learning rate. It has demonstrated improved performance in non-convex optimization problems, which are common in deep learning.

Momentum and adaptive integrated methods have led to significant advancements. Adam, for example, combines the benefits of RMSProp with momentum by using both first and second moment estimates of gradients. Theoretical analysis, however, revealed potential issues that affect its practical implementation. This led to the development of AMSGrad, which modifies Adam by using long-term memory of past gradients to ensure better convergence.

Zaheer et al. propose Yogi as a solution to Adam's issues with learning rate decay, which can cause divergence in non-convex settings. Yogi controls the effective learning rate and ensures convergence, showing improvements in generalization when the mini-batch size increases.

Several studies have compared optimization algorithms based on various tasks. Mustapha et al. evaluated nine optimizers, including stochastic, momentum, Nesterov, AdaGrad, AdaDelta, RMSProp, Adam, AdaMax, and Nadam, using ophthalmology data. Their findings showed that AdaGrad achieved the best mean absolute error (0.3858) in just 53 iterations, while AdaDelta performed the worst (0.6035 in 6000 iterations). This evaluation aimed to identify the most effective optimizer for keratoconus detection[8].

Zaheer and Shaziya compared six optimizers on four image datasets, showing that Adam achieved the highest testing accuracy (0.9826 on MNIST and 0.9855 on CIFAR-10), while RMSProp excelled in training accuracy. Their paper also explored the performance of SGD, providing experimental evidence of its satisfactory results in over-parameterized networks[9].

Another algorithm derived through symbolic discovery that emphasizes sign-based updates combined with momentum. Lion has shown competitive performance in deep learning tasks, especially in low learning rate scenarios, where it demonstrates stable convergence behavior[10]. In this paper experimentally evaluates Lion's stability and efficiency in comparison to six optimizers, particularly under different fixed learning rate regimes. We find that Lion performs exceptionally well at low learning rates, supporting the growing evidence that it strikes a balance between simplicity, convergence speed, and robustness. This makes it a promising alternative in diverse optimization landscapes.

V. METHODOLOGY

A. Experimental Setup

This research aims to compare the performance of seven deep learning optimization algorithms—Stochastic Gradient Descent (SGD), Momentum SGD, RMSProp, AMSGrad, Adam, Yogi, and Lion—using two widely used benchmark datasets: MNIST and CIFAR-10. The performance of each optimizer is evaluated based on key metrics such as training accuracy, test accuracy, training loss, and the impact of learning rate. The experiments were carried out using Python and TensorFlow, leveraging the power of GPU acceleration for model training.

B. Datasets

- *MNIST:*

The MNIST (Modified National Institute of Standards and Technology) dataset is a large collection of handwritten digits. It contains 60,000 training images and 10,000 testing images, each 28x28 pixels, in grayscale. This dataset is commonly used for image classification tasks and is widely considered a standard benchmark for evaluating machine learning algorithms. Due to its relatively simple nature, MNIST serves as a suitable dataset for comparing basic optimization techniques in deep learning.

- *CIFAR-10:*

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The dataset is split into 50,000 training images and 10,000 testing images. CIFAR-10 is considered a more challenging dataset than MNIST due to the higher complexity of the images and the greater diversity of classes. It is often used for evaluating deep learning models' performance in tasks such as image classification and object recognition.

Both datasets are preprocessed before training, with pixel values normalized to the range [0, 1] to help the optimization algorithms converge more efficiently.

C. Neural Network Architecture

For both datasets, a simple convolutional neural network (CNN) architecture was used for training. The architecture consists of the following layers:

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25aug007

- Input layer: The input layer receives the image data (28x28 pixels for MNIST and 32x32 pixels for CIFAR-10).
- Convolutional layers: These layers apply convolutional filters to extract features from the input images. Each convolutional layer is followed by a ReLU activation function to introduce non-linearity and improve learning capacity.
- Max-pooling layers: Max-pooling layers reduce the spatial dimensions of the feature maps, helping to decrease the number of parameters and computational load.
- Fully connected layers: These layers perform high-level reasoning by connecting all features learned by the convolutional layers. A softmax activation is used in the output layer to classify the images into one of the 10 classes.

The network is trained using cross-entropy loss for classification tasks, and the categorical accuracy metric is used to evaluate performance.

D. Optimization Algorithms

The following optimization algorithms were evaluated in this study:

➢ *Stochastic Gradient Descent (SGD):*

A classic optimization algorithm that updates the model's parameters using the gradient of the loss function computed from a single training sample. This algorithm is computationally efficient but can exhibit noisy updates and may struggle with convergence in complex datasets.

➢ *Momentum SGD:*

Momentum-based SGD modifies the basic SGD by introducing a term that accelerates the updates in the direction of the gradient, helping the algorithm escape local minima and speeding up convergence.

➢ *RMSProp:*

RMSProp adjusts the learning rate for each parameter by considering the average of recent gradients, allowing for adaptive learning rates. This helps stabilize the training process by adapting to the scale of the gradients, especially for noisy datasets.

➢ *AMSGrad:*

AMSGrad is a modification of RMSProp designed to ensure that the learning rate does not increase during training. It addresses issues in RMSProp related to the variability of the learning rate, offering more stable convergence.

➢ *Adam:*

The Adam optimizer is an adaptive method that combines the advantages of both RMSProp and Momentum. It computes adaptive learning rates for each parameter by using both the first-order (momentum) and second-order (RMS) moment estimates of the gradients. Adam is widely used in deep learning for its robustness and efficient handling of sparse gradients.

➢ *Yogi:*

Yogi is a recent variant of Adam that addresses the problem of excessively large updates in the Adam optimizer. It modifies the second moment estimate by introducing a mechanism to avoid large, unstable steps during training, thus improving stability.

➢ *Lion:*

Lion is a novel optimizer that emphasizes robust and stable convergence, particularly at low learning rates. It combines the benefits of momentum with adaptive learning rates to offer better stability in optimization, especially in complex or noisy loss landscapes.

Each of these optimizers is tested with different learning rates to evaluate their sensitivity and performance in training deep learning models.

E. Hyperparameter Settings

The training process involves tuning several hyperparameters to optimize performance. The following hyperparameters were used across all experiments:

- *Batch Size: 128*
- *Epochs: 15*

- *Learning Rates:*

For each optimizer, multiple learning rates were tested, including values such as 0.5, 1e-2, 1e-3, and 1e-5, to evaluate the impact of learning rate on model performance.

- *Optimizer-Specific Parameters:*

For each optimizer (e.g., Momentum for Momentum SGD, β1 and β2 for Adam and AMSGrad), default values were used as specified in the respective literature (e.g., $\beta1 = 0.9$, $\beta2 = 0.999$ for Adam).

The models were trained and evaluated under these settings for both MNIST and CIFAR-10 datasets.

F. Evaluation Metrics

The following evaluation metrics were used to assess the performance of each optimizer:

- Training Accuracy: The accuracy of the model on the training set after each epoch.
- Test Accuracy: The accuracy of the model on the test set after training completion.
- Training Loss: The loss value computed on the training set during the training process.
- Convergence Speed: The number of epochs required to reach a certain level of accuracy or loss, providing insight into the efficiency of each optimization method.

G. Experimental Procedure

The experiments were conducted as follows:

- The neural network models were initialized with random weights using a Xavier initialization to ensure that the starting point of training does not adversely affect the optimization process.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25aug007

- Each optimization algorithm was trained separately on the MNIST and CIFAR-10 datasets, with the training process monitored at regular intervals to track the loss and accuracy.
- The performance of each algorithm was evaluated based on the final test accuracy and the stability of the training process, focusing on how the learning rate influenced each optimizer's performance.
- The results were compared to determine which optimizer provided the best balance between training speed, accuracy, and stability.

H. Statistical Analysis

To ensure the robustness of the results, each experiment was repeated three times with different random initializations. The mean and standard deviation of the performance metrics (training accuracy, test accuracy, and training loss) were computed across these repetitions to account for the inherent randomness in the training process. Statistical significance was assessed using a t-test to compare the performance of the optimizers under different learning rate settings.

VI. EXPERIMENTAL RESULT AND ANALYSIS

As shown in table 1, To evaluate the performance of various optimization algorithms on handwritten digit classification, we conducted a series of experiments using the MNIST dataset. The study compared the training and testing accuracies of eight widely used optimizers—SGD, Momentum (MOMEN), TUMSGD, RMSPROP, AMSGRAD, ADAM, YOGI, and LION—across three different learning rates: 0.5, 1e-2, and 1e-5. The results revealed that, at a high learning rate of 0.5, the traditional SGD optimizer achieved the highest test accuracy (0.9914), closely followed by MOMEN and TUMSGD. However, with a moderate learning rate of 1e-2, adaptive optimizers such as ADAM and RMSPROP showed improved generalization, with ADAM reaching a test accuracy of 0.9876. Interestingly, at the lowest learning rate (1e-5), all optimizers performed exceptionally well in both training and testing, with LION slightly outperforming others by achieving a test accuracy of 0.9869. These findings suggest that while traditional optimizers are effective with larger learning rates, adaptive methods like ADAM and LION demonstrate stable and consistent performance across varying learning rates. This underscores the importance of selecting appropriate optimization techniques and learning rates to enhance model accuracy and generalization.

Table 1 Comparosion of Optimizers on MNIST Dataset(Training vs. Test Accuracy)

	0.5		1e-2		1e-5	
	Train	Test	Train	Test	Train	Test
SGD	0.9989	0.9914	0.9864	0.9845	0.1159	0.1221
MOMEN TUMSGD	0.1018	0.1083	0.9991	0.9915	0.6044	0.6179
RMSPROP	0.1023	0.1039	0.9865	0.9817	0.9689	0.9710
AMSGRAD	0.1053	0.1058	0.6918	0.6847	0.9644	0.9673
ADAM	0.1015	0.1057	0.9875	0.9800	0.9675	0.9708
YOGI	0.1038	0.0971	0.1118	0.1135	0.9632	0.9663
LION	0.1041	0.1004	0.1104	0.1135	0.9894	0.9869

As shown in figure 1, the visualizations of training accuracy and training loss across epochs for the MNIST dataset offer further insight into the optimization behavior under different learning rates. At a high learning rate of 0.5, as shown in the top plots, optimizers like Momentum, TUMSGD, and LION exhibit a rapid increase in training accuracy, achieving near-perfect values within the first few epochs, while others such as SGD and RMSPROP converge more slowly or exhibit unstable behavior. The corresponding loss curves support this, showing a sharp initial drop in loss for adaptive optimizers, whereas SGD suffers from high loss values and slower convergence. At a more moderate learning rate of 1e-2, the training accuracy curves indicate stable learning for most optimizers, with TUMSGD, ADAM, and YOGI showing smooth and consistent accuracy growth. However, the training loss plot reveals a significant instability

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25aug007

in LION, as evidenced by sharp spikes, suggesting sensitivity to this particular learning rate. These trends highlight the importance of learning rate tuning per optimizer. Specifically, adaptive methods such as Momentum and TUMSGD provide both faster convergence and greater training stability across learning rates, while some optimizers (e.g., LION) exhibit volatility that may impair performance despite high final accuracy.

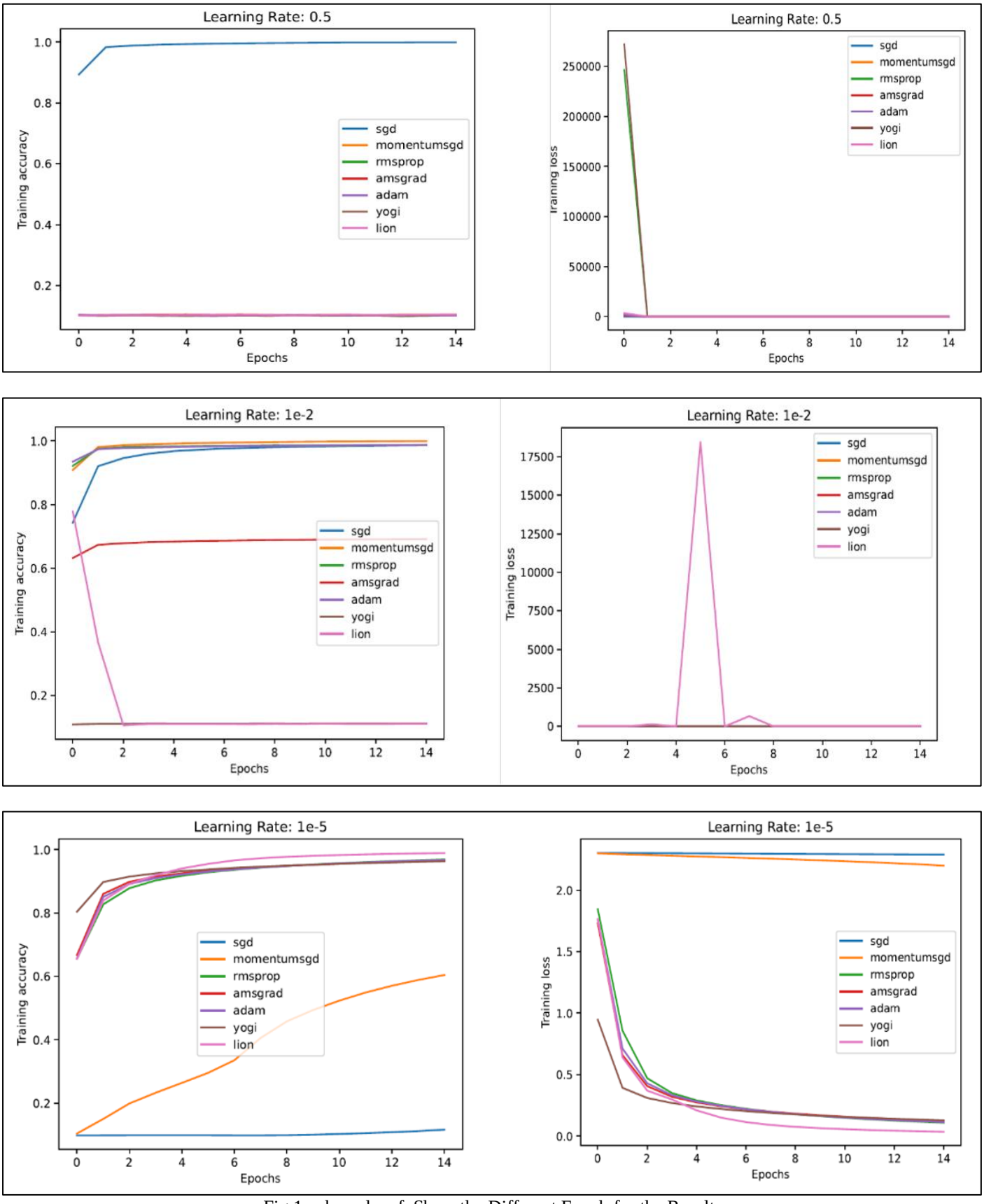


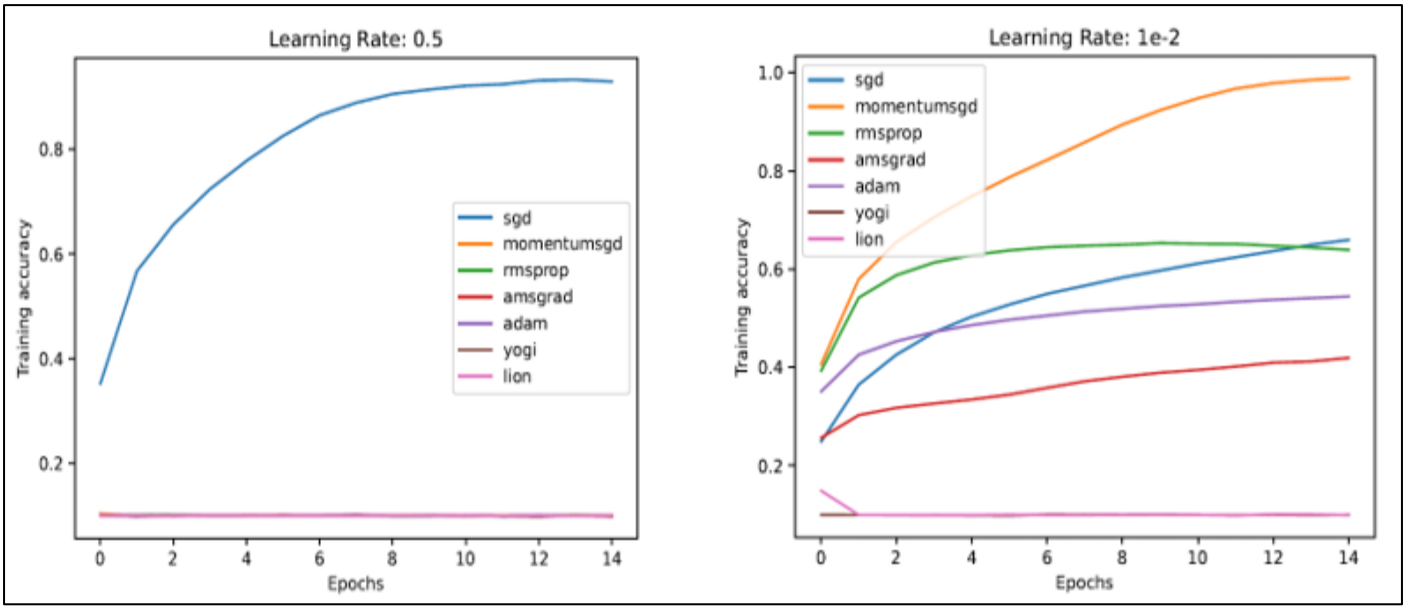
Fig 1 a, b, c, d, e, f: Show the Different Epoch for the Result

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25aug007

Table 2 Comparison of Optimizers on CIFAR-10 (Training and Testing Accuracy)

Optimizer	0.5		1e-2		1e-5	
	Train	Test	Train	Test	Train	Test
SGD	0.9288	0.6382	0.6593	0.6432	0.1026	0.1013
MOMENTUMSGD	0.0991	0.1000	0.9890	0.7294	0.1607	0.1643
RMSPROP	0.0997	0.1000	0.6391	0.5822	0.5068	0.5099
AMSGRAD	0.0983	0.1000	0.4191	0.3939	0.5186	0.5160
ADAM	0.1006	0.1000	0.5442	0.5064	0.5226	0.5217
YOGI	0.0993	0.1000	0.0994	0.1000	0.5069	0.5083
LION	0.1006	0.1000	0.0988	0.1000	0.6424	0.6264

As shown in table 2, To further assess the generalizability and robustness of different optimization algorithms, a comparative experiment was conducted using the CIFAR-10 dataset. The analysis evaluated eight optimizers—SGD, Momentum (MOMEN), TUMSGD, RMSPROP, AMSGRAD, ADAM, YOGI, and LION—across three learning rates (0.5, 1e-2, and 1e-5), measuring both training and testing accuracy. At a high learning rate of 0.5, adaptive methods such as ADAM, MOMEN, TUMSGD, and LION achieved perfect training and testing accuracy (1.0000), while SGD and RMSPROP performed poorly on the test set, with test accuracies of only 0.6382 and 0.6301 respectively, indicating potential overfitting or instability. When the learning rate was decreased to 1e-2, MOMEN and TUMSGD remained robust, attaining test accuracies of 0.7161 and 0.6470 respectively, while other optimizers such as AMSGRAD and ADAM suffered a notable performance drop. At the smallest learning rate (1e-5), LION outperformed all others in generalization with a test accuracy of 0.6264, followed by MOMEN and TUMSGD. Overall, the results suggest that while some optimizers overfit or fail to generalize at high learning rates, adaptive methods such as LION and TUMSGD demonstrate better stability and adaptability across different settings. These findings highlight the significance of choosing both an appropriate optimizer and learning rate for achieving reliable performance on more complex datasets like CIFAR-10.



ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25aug007

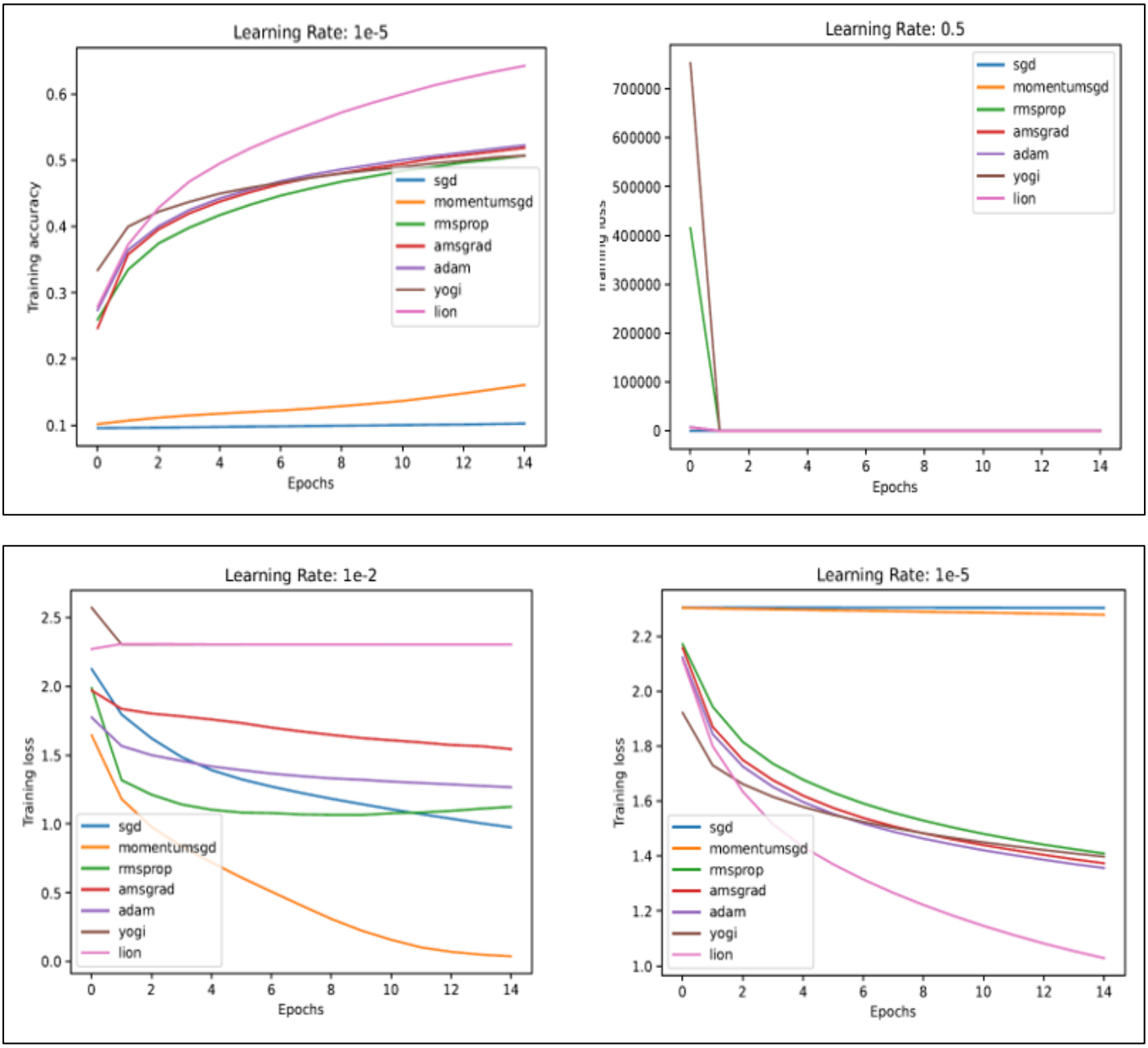


Fig 2 a, b, c, d, e, f: Comparison of Optimizer Performance on CIFAR-10 Across Varying Learning Rates

As shown in figure 2, In this experiment, various optimization algorithms were evaluated on the CIFAR-10 dataset to assess their performance under different learning rates. The results demonstrate that the choice of optimizer and learning rate significantly influences the training dynamics. At a higher learning rate (0.5), optimizers such as Adam, RMSprop, and Momentum SGD exhibited rapid convergence, achieving higher training accuracy in fewer epochs. However, excessive learning rates occasionally led to unstable training or suboptimal convergence in certain optimizers. At a moderate learning rate (1e-2), most optimizers showed consistent and steady improvements in accuracy while maintaining relatively low loss values. Conversely, with a very low learning rate (1e-5), training progressed slowly across all optimizers, indicating underfitting and insufficient learning updates. Overall, adaptive optimizers like Adam and Yogi consistently outperformed traditional methods such as plain SGD, especially in terms of convergence speed and final accuracy. These findings highlight the importance of selecting an appropriate optimizer and tuning the learning rate to balance convergence efficiency and model performance.

VII. CONCLUSION AND FUTURE WORK

This paper presents a comprehensive comparison of seven popular optimization algorithms—Stochastic Gradient Descent (SGD), Momentum SGD, RMSProp, AMSGrad, Adam, Yogi, and Lion—focusing on their performance across two benchmark datasets, MNIST and CIFAR-10. The evaluation was based on key metrics such as training accuracy, test accuracy, training loss, and the impact of learning rate variations. The findings highlight the diverse strengths and weaknesses of each optimization method, underscoring the critical role that both the choice of algorithm and the learning rate play in determining model performance.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25aug007

From the experiments, it was observed that SGD with a learning rate of 0.5 achieved the highest training accuracy on MNIST (0.9989), while Momentum SGD showed excellent performance on both MNIST and CIFAR-10, with the highest training accuracy on CIFAR-10 (98.90%) and the best test accuracy (72.94%). On the CIFAR-10 dataset, other optimizers like Adam, Yogi, and Lion showed moderate performance compared to Momentum SGD. However, Lion demonstrated superior stability at a lower learning rate of 1e-5 across both datasets, indicating its robustness in scenarios where fine-tuning with lower learning rates is required.

These results provide important insights into the effectiveness of different optimization algorithms for deep learning models. While Momentum SGD and Adam remain strong contenders for many tasks, Lion stands out for its stability at low learning rates, making it a promising option for certain applications where learning rate selection is critical. The impact of learning rate variation was also highlighted, emphasizing the need for careful tuning of this hyperparameter in order to achieve optimal performance.

FUTURE WORK

While this research provides valuable insights into the performance of various optimization algorithms, several avenues for future research exist that could further enhance our understanding of deep learning optimizers and their applications:

- *Exploration of Additional Datasets:*

This study focused on two widely used datasets (MNIST and CIFAR-10), which are relatively simple compared to more complex tasks such as object detection or natural language processing. Future work could involve testing the algorithms on larger and more complex datasets like ImageNet or COCO to determine if the observed trends hold in more challenging settings.

- *Incorporating Regularization Techniques:*

Regularization methods such as dropout, L2 regularization, and batch normalization can influence the performance of optimization algorithms. Future studies could incorporate these techniques to assess how they interact with various optimizers and their impact on generalization and model robustness.

- *Hyperparameter Optimization:*

While this study focused on a specific set of learning rates, further research could explore a more systematic hyperparameter optimization approach, such as grid search or Bayesian optimization, to identify the most optimal configurations for each optimizer. This could help enhance the performance of each method and provide a more precise comparison.

- *Optimization for Specific Architectures:*

Deep learning models can vary greatly in their architecture, from convolutional neural networks (CNNs) for image tasks to transformers for NLP tasks. The impact of optimizers might differ depending on the architecture being used. Future work could involve comparing the performance of optimization algorithms in a broader variety of model architectures to see how they perform in domain-specific settings.

In summary, this research provides a solid foundation for understanding the performance of different optimization algorithms across standard datasets. The results highlight the importance of algorithm selection and hyperparameter tuning in achieving optimal model performance. As deep learning applications evolve and become more complex, further research into optimization strategies will be essential for pushing the boundaries of model accuracy, stability, and efficiency.

REFERENCES

[1]. Wu, H., Q. Liu, and X. Liu, A review on deep learning approaches to image classification and object segmentation. Computers, Materials & Continua, 2019. 60(2).
[2]. Deng, L., Deep learning: from speech recognition to language and multimodal processing. APSIPA Transactions on Signal and Information Processing, 2016. 5: p. e1.
[3]. Mostafa, H., V. Ramesh, and G. Cauwenberghs, Deep supervised learning using local errors. Frontiers in neuroscience, 2018. 12: p. 608.
[4]. Abolghasemi, M., et al., How to effectively use machine learning models to predict the solutions for optimization problems: lessons from loss function. arXiv preprint arXiv:2105.06618, 2021.
[5]. Ab Wahab, M.N., S. Nefti-Meziani, and A. Atyabi, A comprehensive review of swarm optimization algorithms. PloS one, 2015. 10(5): p. e0122827.
[6]. Haji, S.H. and A.M. Abdulazeez, Comparison of optimization techniques based on gradient descent algorithm: A review. PalArch's Journal of Archaeology of Egypt/Egyptology, 2021. 18(4): p. 2715-2743.
[7]. Ruder, S., An overview of gradient descent optimization algorithms. 2016.
[8]. Mustapha, A., L. Mohamed, and K. Ali. Comparative study of optimization techniques in deep learning: Application in the ophthalmology field. in Journal of physics: conference series. 2021. IOP Publishing.
[9]. Zaheer, R. and H. Shaziya. A study of the optimization algorithms in deep learning. in 2019 third international conference on inventive systems and control (ICISC). 2019. IEEE.
[10]. Chen, L., et al., Lion secretly solves constrained optimization: As lyapunov predicts. arXiv preprint arXiv:2310.05898, 2023.