Project Final Report

Advance Special Topics in Computer Science -

Optimization

By -
Gauri Sachin Kulkarni
Abhishek Rohidas Khedekar
Abhishek Milind Patwardhan

Motivation

A key factor in the training and functionality of deep learning models is the selection of optimization techniques. Even though there is a wide range of optimizers on the market today, each optimizer's efficacy might differ greatly depending on the model architecture, dataset, and application. For the following reasons, it is crucial to comprehend how various optimization techniques affect model training and performance:

1. **Improving Convergence Speed:**
   - The effectiveness of training a model depends on how soon the optimizer converges to a good solution. Faster convergence results in lower processing costs and time, which is critical for large-scale datasets and complex models.
   - Identifying the optimizer with the fastest convergence rate can help to expedite the training process.
2. **Maximizing Test Accuracy:**
   - The ultimate goal of every deep learning model is to perform well on new data. Some optimizers excel at generalizing the model's learning, resulting in increased test accuracy.
   - Comparing optimizers helps to identify ones that strike a balance between training efficiency and generalization capabilities.
3. **Understanding Optimizer Behavior**:
   - Diverse optimizers have varying capacities for navigating intricate loss landscapes, avoiding local minima, and managing gradient-related problems including vanishing or bursting gradients.
   - By examining accuracy trends and loss curves, we can gain insight into how various optimizers approach these problems.
4. **Guiding Optimizer Selection**:
   - Practitioners frequently select optimizers without fully comprehending their trade-offs, relying instead on defaults or prior experience.
   - This study attempts to offer empirical information that can direct well-informed decision-making for various tasks by methodically comparing optimizers.
5. **Diversity in Architectures and Datasets**:
   - While datasets like CIFAR-10 and MNIST vary in complexity and scale, models like ResNet50 and AlexNet reflect distinct design philosophies.
   - Analyzing optimizers on a variety of datasets and architectures guarantees that the results are reliable and applicable to a broad range of use scenarios.
6. **Facilitating Research and Innovation**:
   - The study's conclusions may stimulate the creation of fresh optimization methods or enhancements to current ones that are suited to certain difficulties encountered during deep learning training.

By exploring the comparative performance of various optimizers on ResNet50 and AlexNet with CIFAR-10 and MNIST datasets, this project seeks to provide valuable benchmarks and practical recommendations for optimizing deep learning workflows.

Background

By facilitating the creation of models that attain cutting-edge performance on a variety of tasks, such as image recognition, natural language processing, and reinforcement learning, deep learning has completely transformed the area of artificial intelligence. Effective model training is essential to deep learning success and is largely dependent on the selection of optimization techniques. In order to train deep learning models, a loss function that measures the discrepancy between expected and actual outputs must be minimized.

Optimization algorithms play an important part in this process by iteratively modifying model parameters to reduce losses. Gradient descent, the most basic optimization technique, has developed into more complex variants like SGD with momentum and Nesterov momentum, which decrease oscillations and speed up convergence. Adaptivity to learning rates was introduced by optimizers such as Adam, RMSProp, and AdaGrad, which enabled faster convergence and improved performance on a variety of tasks. By addressing issues like over-adaptation to noisy gradients, techniques like AMSGrad and NAdam improve the behavior of adaptive algorithms.

Deep learning architectures like ResNet50 and AlexNet differ in depth, complexity, and design philosophy. ResNet50 uses residual connections to address vanishing gradient problems in deep networks, while AlexNet, a pioneering convolutional neural network (CNN), emphasizes feature extraction and hierarchical learning.

Dataset Characteristics:

- **CIFAR-10**: A dataset of 60,000 32x32 color images in 10 classes, challenging models with its small image size and diverse object categories.
- **MNIST**: A simpler dataset of 70,000 grayscale images of handwritten digits, often used as a benchmark for evaluating model performance on relatively straightforward tasks.
- The differences in dataset complexity and size make it critical to evaluate optimizer performance in both scenarios.

Optimizers Overview -

1. **SGD (Stochastic Gradient Descent)**
   SGD updates parameters by using the gradient of a random subset of the data (mini-batch). It is one of the most widely used optimizers but can be sensitive to the learning rate and might require many iterations to converge.

Update Rule:

$$\theta_{t+1} = \theta_t - \eta g_t$$

Terms:

- θ_t: Current parameter value.
- η: Learning rate.
- g_t: Gradient of the loss function $L(\theta_t)$ with respect to θ_t.

2. **SGDM (SGD with Momentum)**
SGDM improves upon standard SGD by adding a momentum term, which helps accelerate convergence and reduces oscillations during optimization by accumulating the gradients from previous steps.

Update Rule:

$$v_t = \beta v_{t-1} + g_t$$

$$\theta_{t+1} = \theta_t - \eta v_t$$

Additional Term:

- v_t: Velocity, representing the exponentially decaying moving average of gradients.
- β: Momentum factor, determining the contribution of previous velocity values.

3. **SGD with Nesterov Momentum (SGD NM)**
SGD with Nesterov momentum incorporates a "lookahead" mechanism where the current gradient is adjusted based on the previous momentum. This allows the optimizer to anticipate the future path of the gradient, resulting in faster convergence.

Update Rule:

$$\theta' = \theta_t + \beta v_{t-1}$$

$$v_t = \beta v_{t-1} + g(\theta')$$

$$\theta_{t+1} = \theta_t - \eta v_t$$

Additional Term:

- θ': Interim parameter value, shifted in the direction of the previous velocity for better gradient estimation.

4. **Adagrad**
Adagrad is an adaptive optimizer that adjusts the learning rate for each parameter based on past gradient information. It works well when the data is sparse and requires fewer steps to converge in many cases.

Update Rule:

$$r_t = r_{t-1} + g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{r_t + \epsilon}} g_t$$

Additional Term:

- r_t: Accumulated sum of squared gradients.
- ϵ: A small constant added for numerical stability, preventing division by zero.

5. **AdaDelta**
AdaDelta improves on Adagrad by maintaining a moving window of accumulated past gradients, which prevents the learning rate from decaying too rapidly. This method helps stabilize the learning process, especially for complex datasets.

Update Rule:

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho) g_t^2$$

$$\Delta\theta_t = -\frac{\sqrt{E[\Delta\theta^2]_{t-1} + \epsilon}}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Additional Terms:

- $E[g^2]_t$: Exponentially weighted moving average of squared gradients.
- $\Delta\theta_t$: Parameter update step.
- ρ: Decay rate for the moving average.

6. **RMSProp**
RMSProp is an adaptive learning rate optimizer that divides the learning rate by an exponentially decaying average of squared gradients. This optimizer is well-suited for training on non-stationary objectives or noisy data.

Update Rule:

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Additional Term:

- $E[g^2]_t$: Exponentially weighted moving average of squared gradients, discarding older gradient history.

7. **Adam (Adaptive Moment Estimation)**
Adam combines the ideas of momentum and adaptive learning rates, making it highly efficient in handling sparse gradients and noisy problems. It uses both the first moment (mean of gradients) and the second moment (variance of gradients) to

compute parameter updates. This optimizer is widely used for its quick convergence and robustness.

Update Rule:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Additional Terms:

- m_t: Exponentially decaying average of gradients (first moment).
- v_t: Exponentially decaying average of squared gradients (second moment).
- $\hat{m}_t$, $\hat{v}_t$: Bias-corrected estimates of m_t and v_t.

8. **NAdam (Nesterov-accelerated Adaptive Moment Estimation)**
NAdam is an improvement to Adam that integrates Nesterov momentum, which allows the optimizer to look ahead at the future gradients. This results in faster convergence and more efficient learning by adjusting gradients more effectively.

Update Rule:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \left(\hat{m}_t + \frac{\beta_1}{1-\beta_1^t} g_t \right)$$

Additional Term:

- $\frac{\beta_1}{1-\beta_1^t} g_t$: Interim gradient adjustment based on momentum.

9. **Adamax**
Adamax is a variant of Adam that uses the infinity norm instead of the L2 norm for the parameter updates. This modification often helps in cases where Adam may experience instability in its updates.

Update Rule:

$$u_t = \max(\beta_2 u_{t-1}, | g_t |)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t} m_t$$

Additional Term:

- u_t: Exponentially weighted infinity norm of gradients.

10. **AMSGrad**

AMSGrad is a modification of the Adam optimizer designed to avoid the problem of decreasing learning rates, which could hinder model performance. By using the maximum of past squared gradients, AMSGrad ensures that the learning rate never decreases, leading to more stable convergence.

Update Rule:

$$v_t = \max(v_{t-1}, g_t^2)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

Additional Term:

- v_t: Maximum of past squared gradients, ensuring a non-decreasing step size.

Main Idea

The project is geared towards conducting and comparing several optimization algorithms for deep learning models with the greatest emphasis being on Nesterov Accelerated Shuffling Gradient (NASG). The basic two objectives of this project are:

- Implement several optimizers and compare their performances in terms of training loss, accuracy, and Receiver Operating Characteristic (ROC) curves.
- Implement gradient-based optimization algorithm, NASG, and check its performance at different learning rates.

We aim at checking how well these optimizers with the new NASG optimized models perform in deep learning models as well as how learning rate selection affects the optimization in the process.

1. Implementation of Various Optimizers

To set the baseline, we use implementations of many famous optimizers, like AMSGrad, NAdam, AdaDelta, Adagrad, Adamax, RMSProp, SGD, SGDM (Stochastic Gradient Descent with Momentum), and SGD with Nesterov momentum (SGD NM). The Key Performance Indicators (KPIs) for comparison are:

1. Training Loss: Measures the performance of the model during training.
2. Training Accuracy: Reflects the model's ability to classify correctly on seen data.
3. ROC Curve: Used to assess the trade-offs between the true positive rate and false positive rate, providing an insight into the model's discrimination capability.

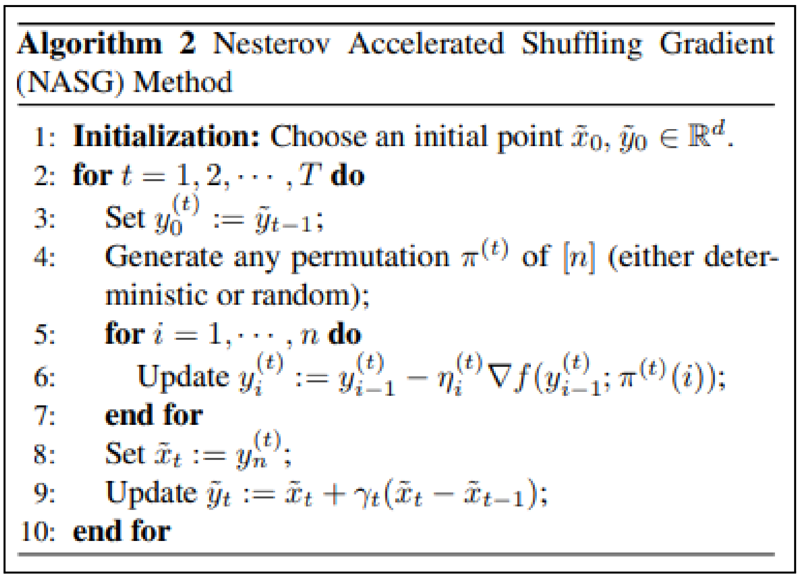
By varying the learning rate for every optimizer we want to reach out at a point that is much better with respect to faster convergence and at higher test accuracy.

Each set of optimizers will be subjected to testing on two established deep learning architectures, ResNet50 and AlexNet, using the CIFAR10 and MNIST datasets as training sets.

2. Nesterov Accelerated Shuffling Gradient (NASG)
A key aspect of this project is the introduction of NASG (Nesterov Accelerated Shuffling Gradient), an innovative approach that merges Nesterov's Accelerated Gradient (NAG) with shuffling-based gradient updates. The NASG algorithm overcomes the shortcomings of conventional momentum-based optimizers and enhances convergence speed, especially in stochastic gradient methods.

NASG is founded on two primary concepts:
Nesterov's Momentum: This technique accelerates the convergence rate of gradient descent by taking into account the future gradient in addition to the current one. Nesterov's momentum achieves a faster convergence rate of O(1/T^2), where T represents the number of iterations/epochs, compared to the traditional gradient descent rate of O(1/T).

Shuffling Gradient Descent: Rather than executing a deterministic update of the gradient based on a fixed sequence of data points, NASG employs random or deterministic shuffling methods. This randomization helps to disrupt symmetry and enhances convergence speed, as empirical research indicates that shuffling typically results in quicker convergence than standard SGD with sequential updates.



Algorithm 2 Nesterov Accelerated Shuffling Gradient (NASG) Method

```
1: **Initialization:** Choose an initial point $\tilde{x}_0, \tilde{y}_0 \in \mathbb{R}^d$.
2: **for** $t = 1, 2, \cdots, T$ **do**
3:     Set $y_0^{(t)} := \tilde{y}_{t-1}$;
4:     Generate any permutation $\pi^{(t)}$ of $[n]$ (either deterministic or random);
5:     **for** $i = 1, \cdots, n$ **do**
6:         Update $y_i^{(t)} := y_{i-1}^{(t)} - \eta_i^{(t)} \nabla f(y_{i-1}^{(t)}; \pi^{(t)}(i))$;
7:     **end for**
8:     Set $\tilde{x}_t := y_n^{(t)}$;
9:     Update $\tilde{y}_t := \tilde{x}_t + \gamma_t(\tilde{x}_t - \tilde{x}_{t-1})$;
10: **end for**
```

Mathematical Formulation of NASG

The NASG method follows these steps:
Gradient Update Step (for each sample at epoch):

$$y_i^{(t)} = y_{i-1}^{(t)} - \eta \nabla f\left(y_{i-1}^{(t)}; \pi^{(t)}(i)\right),$$

where:

- $y_i^{(t)}$ is the updated value for the i-th sample at epoch t,
- $\pi^{(t)}$ is the permutation of data indices at epoch t,
- η is the learning rate, and
- $\nabla f\left(y_{i-1}^{(t)}; \pi^{(t)}(i)\right)$ is the gradient at the current point for the i-th sample.

Momentum Update (after all samples have been processed in epoch):

$$x^{(t+1)} = y_n^{(t)} + \gamma_t\left(y_n^{(t)} - y_n^{(t-1)}\right),$$

where:

- $x^{(t+1)}$ is the final updated value after epoch t,
- γ_t is the momentum parameter, typically defined as:

$$\gamma_t = \frac{t-1}{t+2}.$$

This momentum helps accelerate convergence by adjusting the momentum term based on the number of epochs.

Learning Rate η_t : The learning rate is chosen to control the step size of the gradient updates and is defined as:

$$\eta_t = \frac{\alpha_t}{L},$$

where L is the Lipschitz constant for the gradient of the loss function, and α_t is a scaling factor that may vary across epochs.

Convergence Analysis of NASG

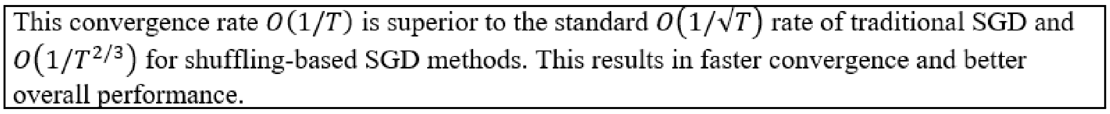
One of the significant advantages of NASG is its improved convergence rate. For convex and smooth objective functions, NASG achieves a convergence rate of:

$$F\left(x^{(T)}\right) - F(x^*) \leq \frac{4\sigma^2}{9LT} + \frac{2L \parallel x^{(0)} - x^* \parallel^2}{T},$$

where:

- $F\left(x^{(T)}\right)$ is the loss at the T-th iteration,
- $F(x^*)$ is the optimal loss,
- σ^2 is the variance at the minimizer,
- L is the Lipschitz constant, and
- $\parallel x^{(0)} - x^* \parallel$ is the initial distance to the optimal solution.

This convergence rate $O(1/T)$ is superior to the standard $O\left(1/\sqrt{T}\right)$ rate of traditional SGD and $O\left(1/T^{2/3}\right)$ for shuffling-based SGD methods. This results in faster convergence and better overall performance.


This convergence rate $O(1/T)$ is superior to the standard $O(1/\sqrt{T})$ rate of traditional SGD and $O(1/T^{2/3})$ for shuffling-based SGD methods. This results in faster convergence and better overall performance.

Comparison of Learning Rates for NASG

The second key aspect of the project involves **comparing different learning rates** for NASG. Since NASG utilizes both momentum and shuffling techniques, the **learning rate** plays a critical role in its performance. We aim to:

- Implement **NASG** with various learning rates to observe its effect on training loss and accuracy.
- Compare the **optimal learning rate** for NASG against the learning rates found to be optimal for other optimizers, such as Adam, SGD, and RMSProp.

Through experimentation, we expect to identify the learning rate that yields the fastest convergence and highest test accuracy for NASG.

Implementation Details

For this study, we conducted image classification experiments on two standard datasets: CIFAR-10 and MNIST. We employed two prominent deep learning architectures: ResNet50 (using a pre-trained model) and AlexNet. For the MNIST dataset, which contains grayscale images, we implemented a preprocessing step to convert the images to RGB format.

We maintained consistent hyperparameters across all experiments to ensure fair comparison. The mini-batch size was set to 128 for all models to balance training stability and memory efficiency. Each model was trained for 50 epochs to evaluate performance and convergence rates.

The learning rates were carefully tuned based on optimizer characteristics:

- For SGD and its momentum variants, we used a learning rate of 0.01
- For adaptive algorithms (Adam, AdaDelta, Nadam, AMSGrad), we set the learning rate to 0.001

Additional hyperparameters were configured as follows:

- Momentum (α) of 0.9 for momentum-based SGD variants
- AdaDelta: $\rho = 0.95$ for squared gradients decay
- For Adam, AdaMax, Nadam, and AMSGrad: $\rho1 = 0.9$ and $\rho2 = 0.999$

We also implemented and evaluated the recent Nesterov Accelerated Shuffling Gradient optimizer. For this optimizer, we conducted experiments with different learning rates:

We also implemented and evaluated the recent Nesterov Accelerated Shuffling Gradient optimizer. For this optimizer, we tested different learning rate configurations across four model-dataset combinations:

CIFAR-10 dataset:

- ResNet50: [1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001]
- AlexNet: [1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001]

MNIST dataset:

- ResNet50: [1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001]
- AlexNet: [0.01, 0.005, 0.001, 0.0005, 0.0001]

Throughout the training process, we monitored both training loss and accuracy across all models for the full 50 epochs to analyze convergence speed and overall performance.

Results

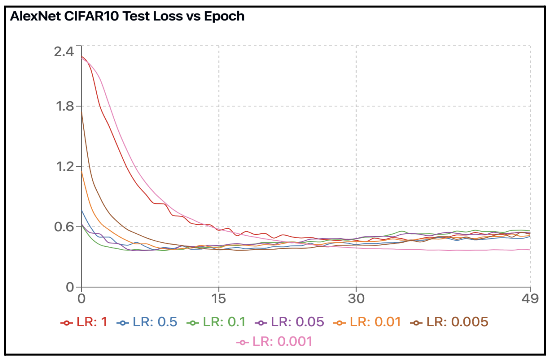
NASG Optimizer

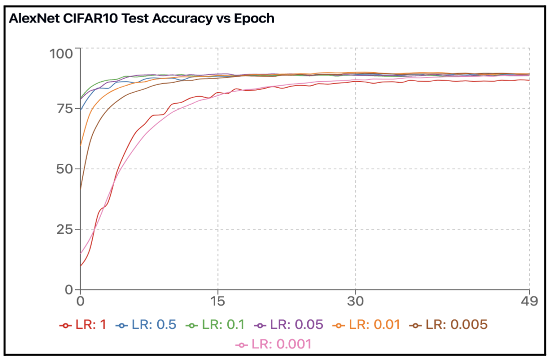
The experiments evaluated two prominent architectures - AlexNet and ResNet50 - across different datasets using various learning rates with the NASG optimizer. For AlexNet on MNIST, the model achieved exceptional performance with test accuracies reaching 99.5-99.6% across multiple learning rates (0.01-0.0001), with 0.01 showing the fastest convergence while maintaining stability. When tested on CIFAR10, AlexNet demonstrated moderate performance, achieving 88-89% test accuracy, with learning rates 0.1 and 0.05 providing the best balance between convergence speed and final accuracy. ResNet50 showed superior performance on both datasets. On MNIST, it achieved remarkable test accuracies of 99.6-99.7%, with learning rates 0.1-0.5 showing optimal performance. On CIFAR10, ResNet50 reached test accuracies of 89-90%, demonstrating better convergence stability compared to AlexNet, particularly with learning rates 0.05-0.1. The NASG optimizer proved effective across both architectures, showing robust performance across different learning rates and datasets.

Model	Dataset	Learning Rate Range	Best Learning Rate	Peak Test Accuracy	Convergence Speed
AlexNet	MNIST	0.01 - 0.0001	0.01	99.60%	10 epochs
AlexNet	CIFAR10	1.0 - 0.001	0.1, 0.05	88.90%	25 epochs
ResNet50	MNIST	1.0 - 0.001	0.1, 0.5	99.70%	15 epochs
ResNet50	CIFAR10	1.0 - 0.001	0.05, 0.1	90.00%	30 epochs

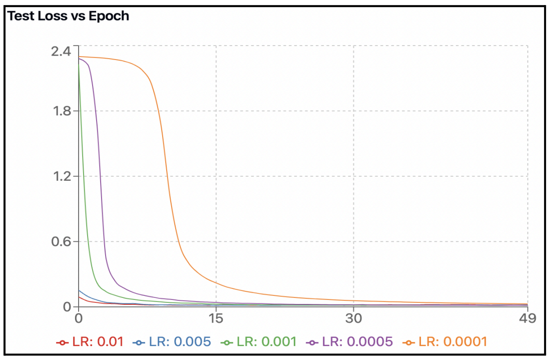
AlexNet with NASG Optimizer for different Learning Rates

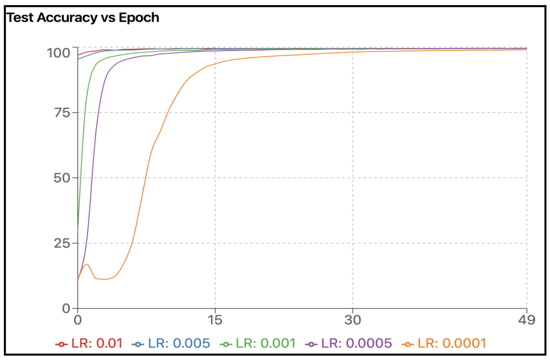
CIFAR-10





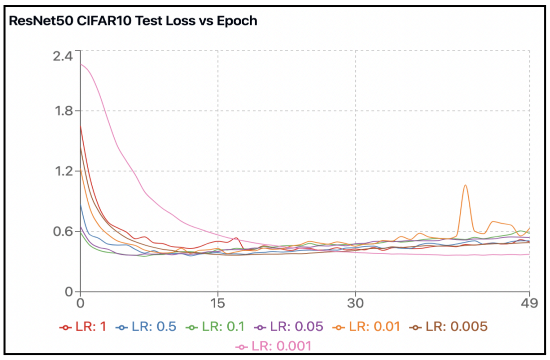
MNIST

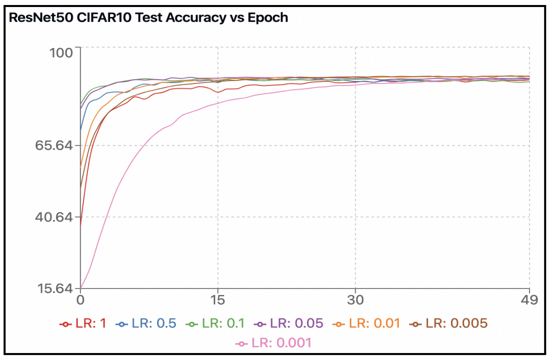




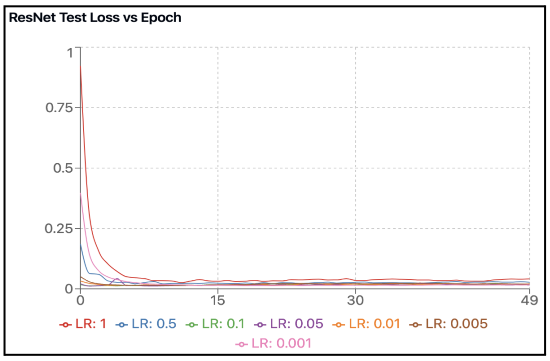
ResNet50 with NASG Optimizer for different Learning Rates

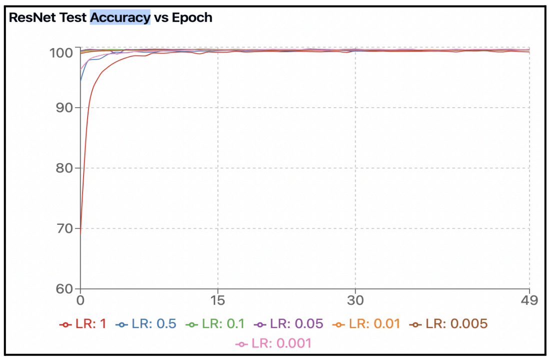
CIFAR10





MNIST





AlexNet

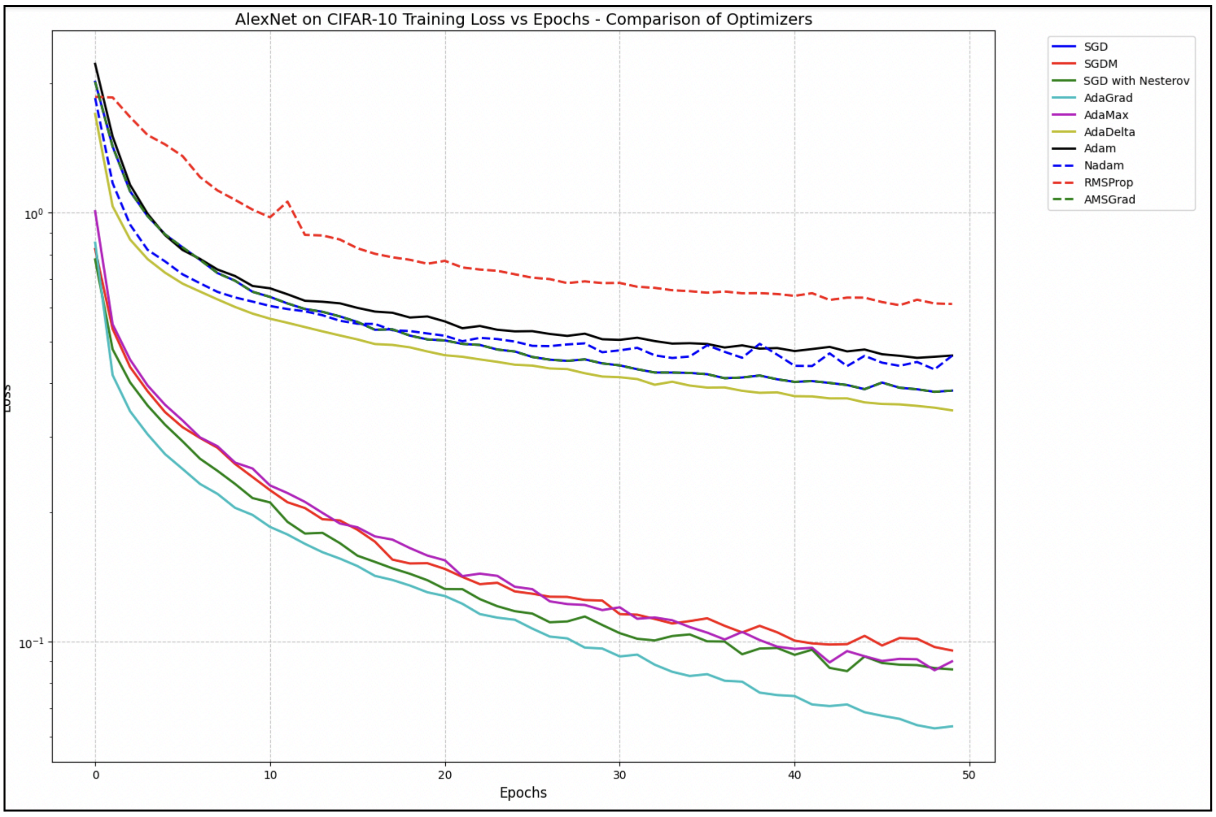
The performance of various optimizers was compared on AlexNet using both CIFAR-10 and MNIST datasets. On CIFAR-10, AdaGrad was the top performer with 97.90% accuracy, while SGD variants also showed strong results (around 97% accuracy). Surprisingly, modern optimizers like Adam and RMSProp underperformed on CIFAR-10, achieving only 84.90% and 80.11% accuracy respectively.

For MNIST, traditional SGD emerged as the best performer with 99.96% accuracy, followed closely by its variants SGDM and Nesterov Momentum (both around 99.75%). While AdaGrad maintained strong performance (99.40%), some modern optimizers struggled - particularly Nadam, which showed significant instability and poor final accuracy (11.09%). Across both datasets, SGD-based methods demonstrated consistently strong and stable performance, while more recent optimizers showed unexpected weaknesses on these specific architectures.

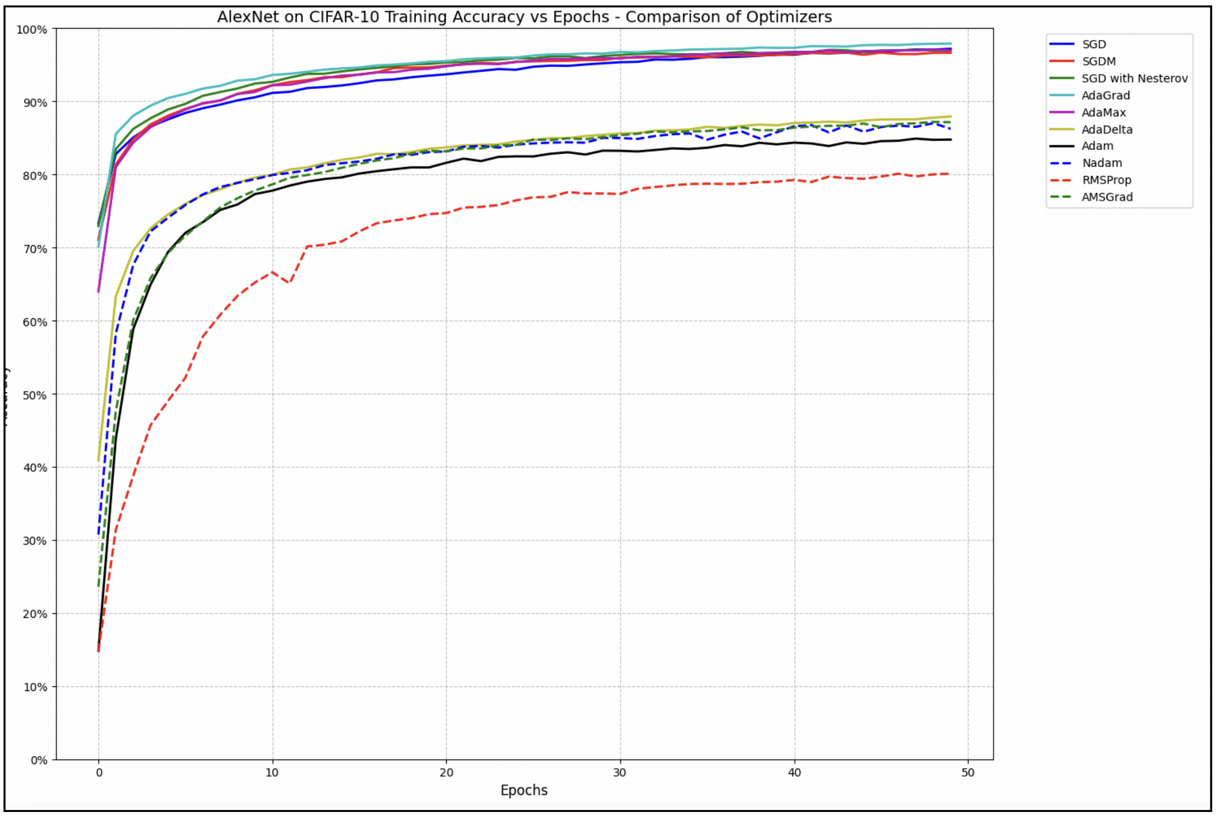
AlexNet on CIFAR10 Dataset

Optimizer	Best Loss (Epoch)	Best Accuracy (Epoch)	Early Convergence (Epoch)
AdaGrad	0.0635 (48)	**97.90%** (49)	15
SGD with Nesterov	0.0862 (49)	97.04% (47)	20
SGD	0.3849 (49)	97.19% (49)	25
SGDM	0.0954 (49)	96.66% (43)	22
AdaMax	0.0900 (49)	97.08% (48)	18
AdaDelta	0.3462 (49)	87.93% (49)	35
Adam	0.4647 (49)	84.90% (47)	40
Nadam	0.4641 (49)	87.01% (48)	38
RMSProp	0.6127 (49)	80.11% (49)	45
AMSGrad	0.3849 (49)	87.18% (48)	30

AlexNet on CIFAR10 Training Loss vs Epochs - Comparison of Optimizers



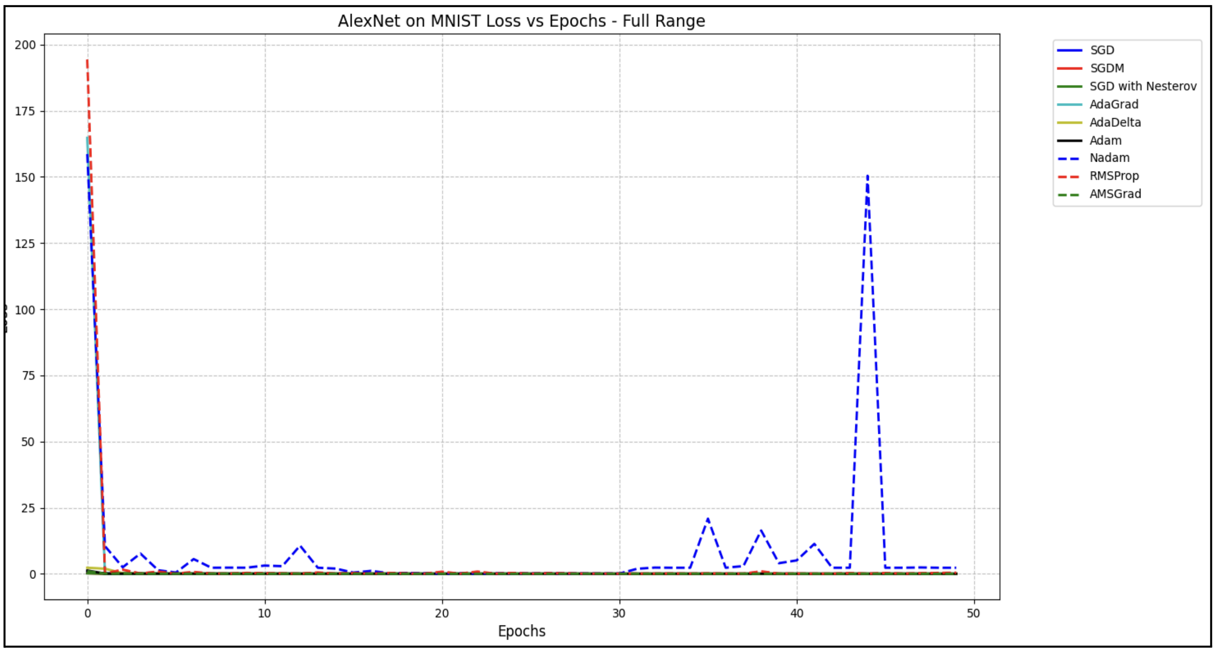
AlexNet on CIFAR10 Training Accuracy vs Epochs - Comparison of Optimizers



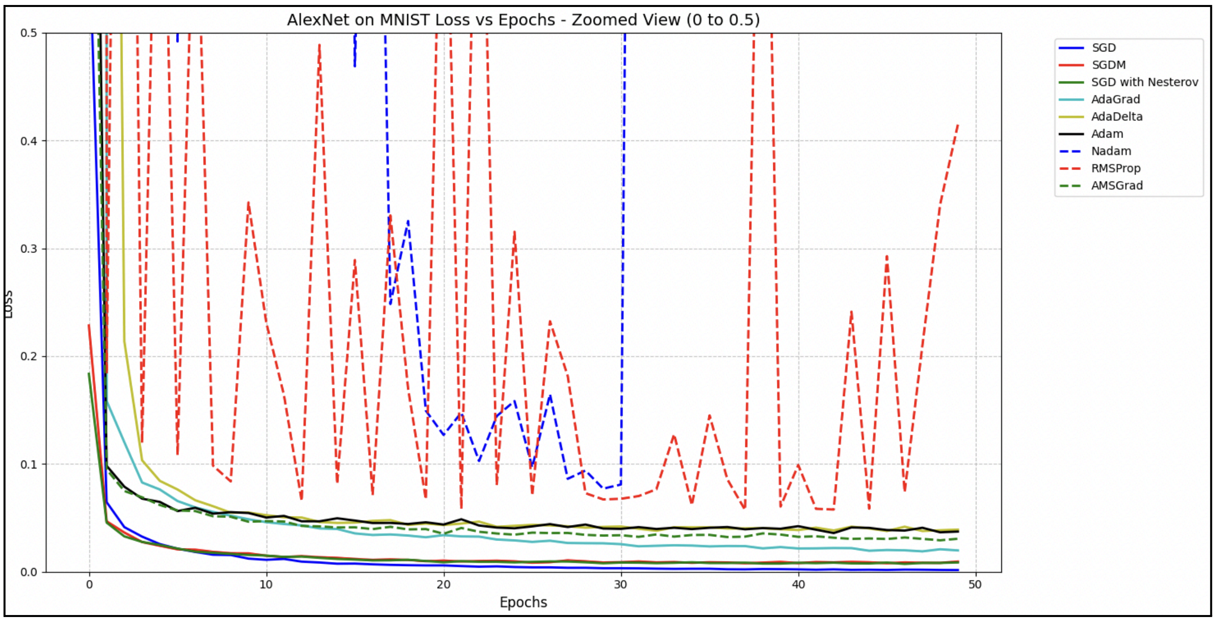
AlexNet on MNIST Dataset

Optimizer	Best Loss (Epoch)	Best Accuracy (Epoch)	Early Convergence (Epoch)
AdaGrad	0.0016 (49)	**99.96%** (46)	10
SGD with Nesterov	0.0079 (45)	99.75% (45)	12
SGD	0.0073 (46)	99.77% (44)	12
SGDM	0.0189 (47)	99.40% (47)	15
AdaDelta	0.0378 (45)	98.91% (42)	20
Adam	0.0359 (42)	99.00% (42)	18
Nadam	0.0771 (29)	97.80% (29)	35
RMSProp	0.0574 (37)	98.45% (37)	25
AMSGrad	0.0292 (48)	99.12% (44)	15

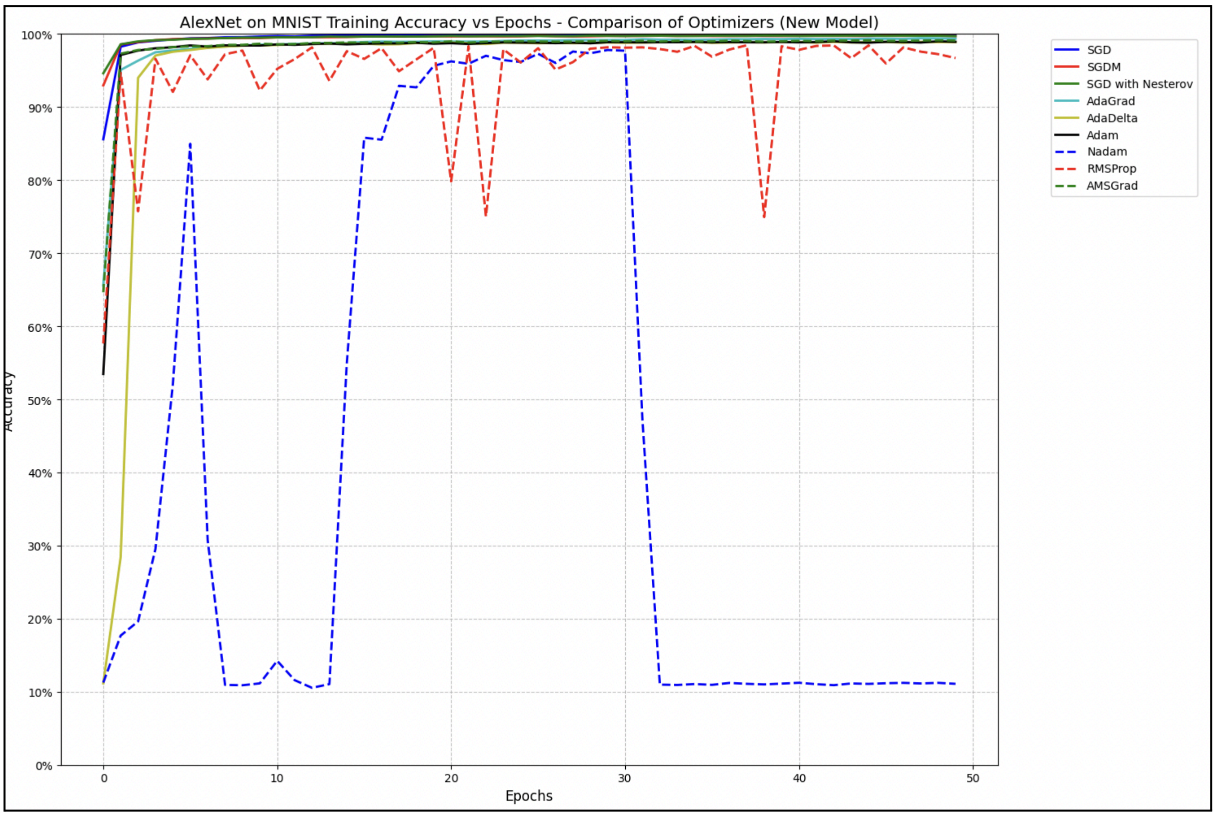
AlexNet on MNIST Training Loss vs Epochs - Comparison of Optimizers



Zoomed View (0 to 0.5)



AlexNet on MNIST Training Accuracy vs Epochs - Comparison of Optimizers



ResNet50

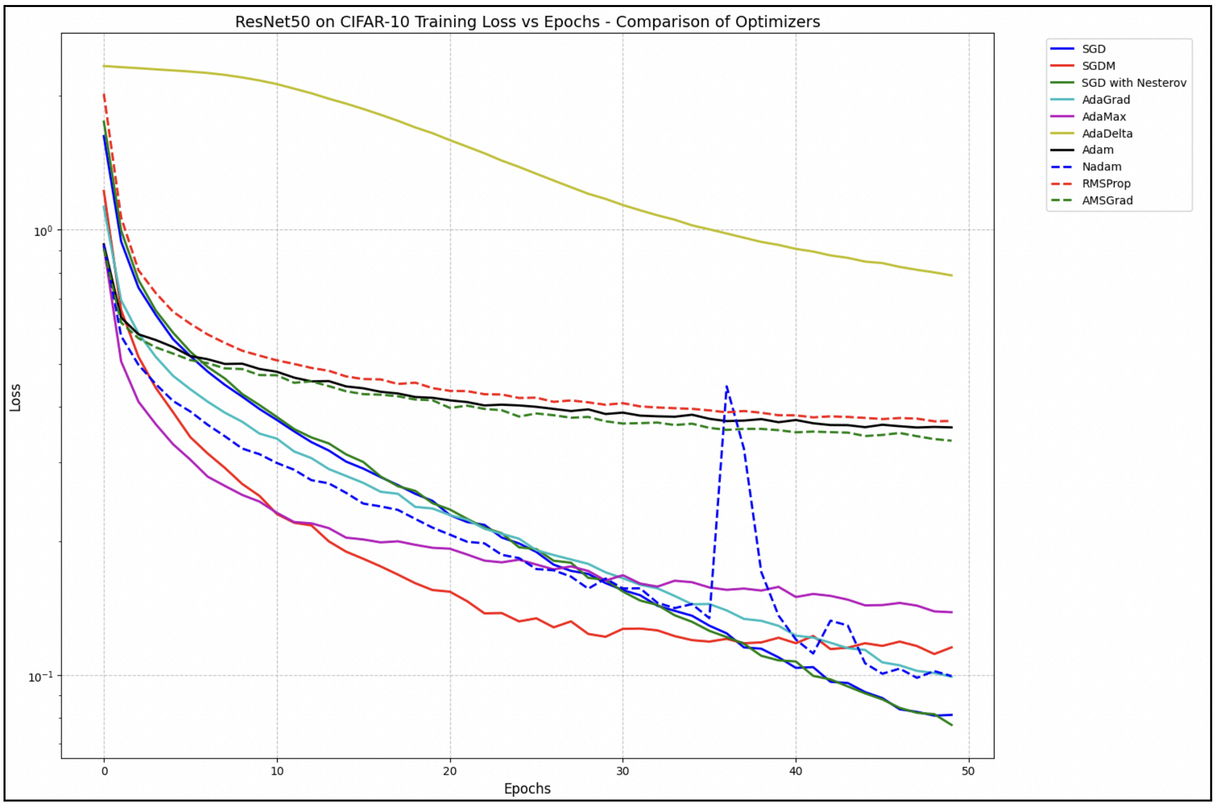
For MNIST, ResNet50 achieved exceptional performance across most optimizers, with SGD-based methods showing particularly strong results. The model reached very high accuracy levels (>99%) quickly, demonstrating effective learning on this simpler dataset. SGD achieved the highest accuracy of 99.96%, while modern optimizers like Adam and RMSProp showed stable but slightly lower performance.

For CIFAR-10, the performance was more varied across optimizers. Traditional SGD and its variants (SGDM, SGD NM) showed strong performance, reaching accuracies above 97%. AdaGrad and Nadam also performed well, while AdaDelta struggled significantly, only reaching about 73% accuracy. The more complex nature of CIFAR-10 led to lower overall accuracies compared to MNIST.

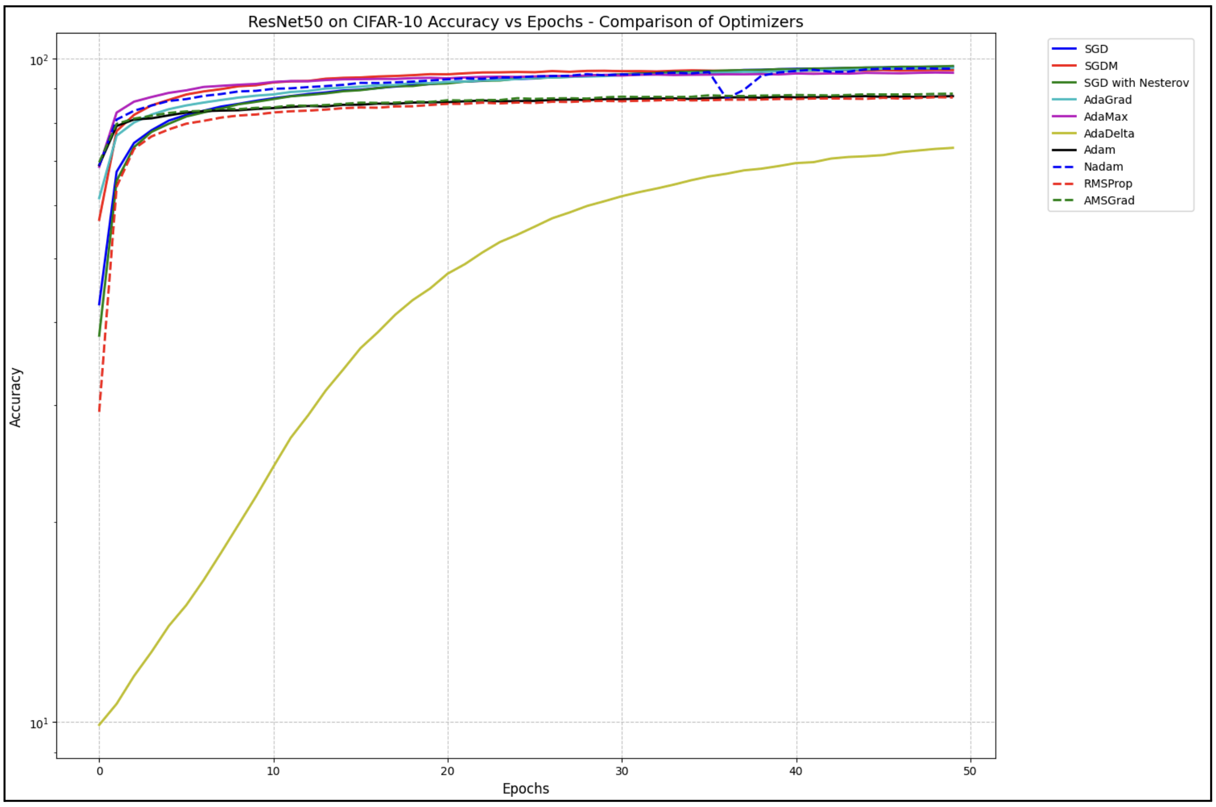
ResNet50 on CIFAR-10 dataset

Optimizer	Best Loss (Epoch)	Best Accuracy (Epoch)	Early Convergence (Epoch)
AdaGrad	0.0991 (49)	96.74% (49)	7
SGD with Nesterov	0.0772 (49)	**97.41%** (49)	11
SGD	0.0813 (49)	97.18% (49)	10
SGDM	0.1153 (49)	96.11% (42)	8
AdaMax	0.1383 (49)	95.23% (48)	5
AdaDelta	0.7900 (49)	73.32% (49)	25
Adam	0.3598 (49)	87.73% (49)	6
Nadam	0.0994 (49)	96.66% (47)	8
RMSProp	0.3719 (49)	87.41% (48)	12

ResNet50 on CIFAR-10 Training Loss vs Epochs - Comparison of Optimizers



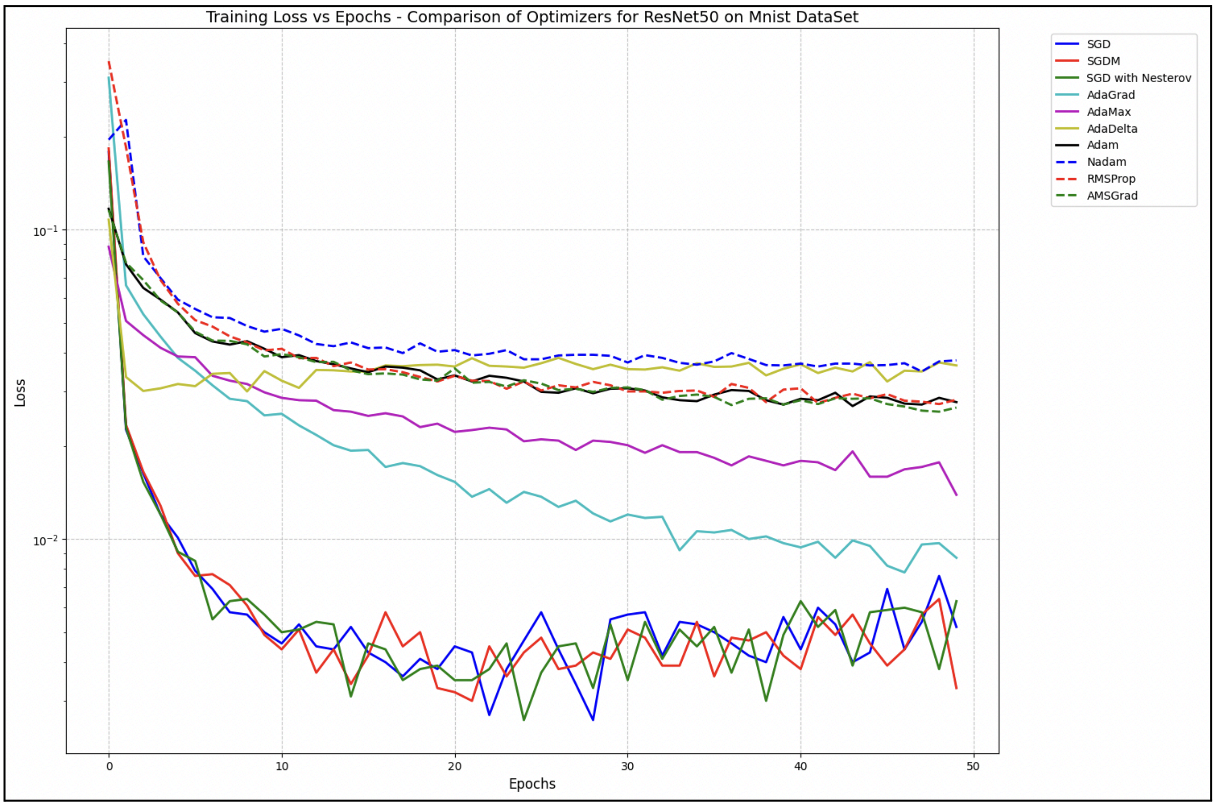
ResNet50 on CIFAR-10 Training Accuracy vs Epochs - Comparison of Optimizers



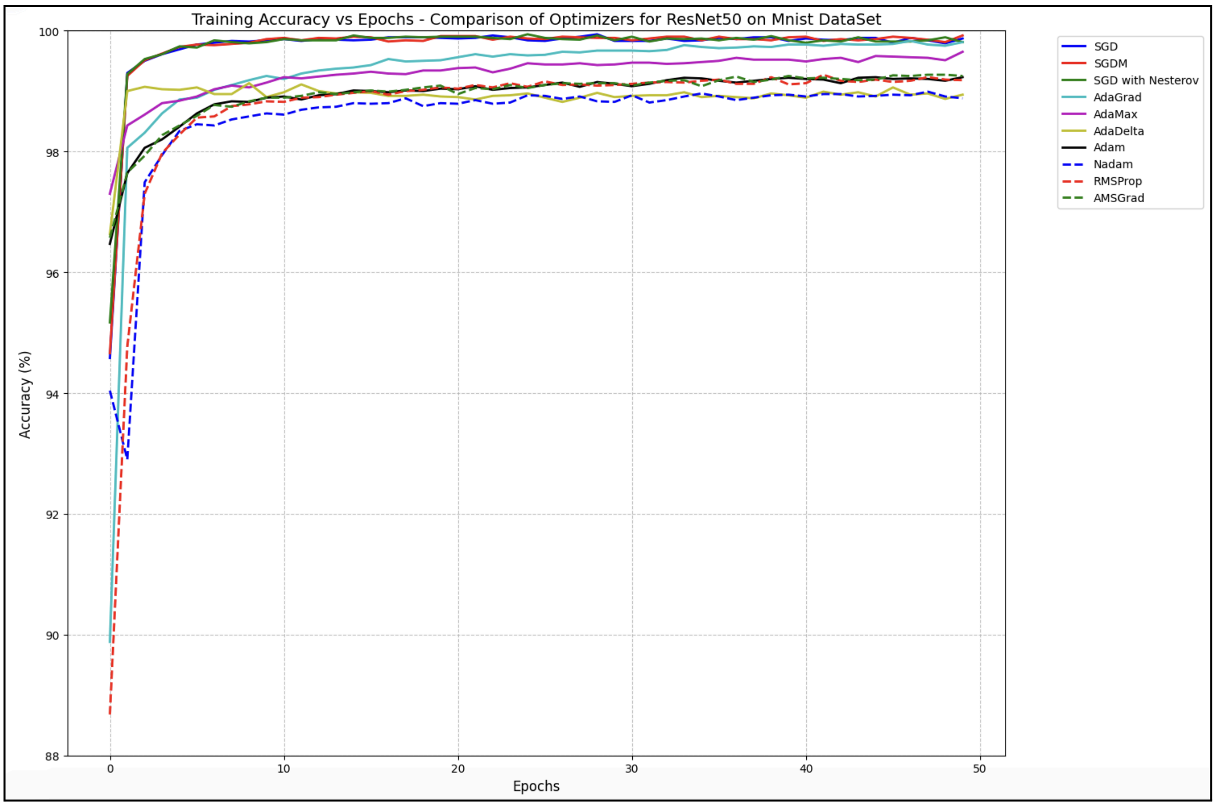
ResNet50 on MNIST Dataset

Optimizer	Best Loss (Epoch)	Best Accuracy (Epoch)	Early Convergence (Epoch)
AdaGrad	0.0991 (49)	99.83% (46)	3
SGD with Nesterov	0.0772 (49)	99.92% (14)	5
SGD	0.0813 (49)	**99.94%** (29)	5
SGDM	0.1114 (48)	99.92% (49)	4
AdaDelta	0.1383 (49)	99.65% (49)	2
Adam	0.7900 (49)	99.13% (41)	8
Nadam	0.3598 (49)	99.23% (49)	3
RMSProp	0.0994 (49)	98.99% (47)	3
AMSGrad	0.3719 (49)	99.27% (41)	4

ResNet50 on MNIST Training Loss vs Epochs - Comparison of Optimizers



ResNet50 on MNIST Training Accuracy vs Epochs - Comparison of Optimizers



When analyzing the optimizer performance across different architectures, NASG showed varied but notable results. On AlexNet with CIFAR10, NASG achieved respectable test accuracies of 88-89% using optimal learning rates of 0.1 and 0.05, surpassing modern optimizers like Adam (84.90%) and RMSProp (80.11%). However, traditional optimization methods including AdaGrad (97.90%) and SGD variants (approximately 97%) demonstrated superior performance for this specific architecture-dataset combination. The results were particularly impressive when testing NASG with ResNet50, where it achieved excellent accuracy of 99.7% on MNIST and 90% on CIFAR10. The optimizer's convergence speed was noteworthy, requiring only 15 epochs for MNIST and 30 epochs for CIFAR10. This performance was especially remarkable given NASG's consistent stability across a wide range of learning rates, a characteristic that sets it apart from many other optimization methods.

Conclusion

While NASG may not have achieved the highest absolute accuracy in all scenarios, it demonstrated remarkable consistency and stability across different architectures and datasets. Its ability to maintain good performance across various learning rates makes it particularly practical for real-world applications where optimal hyperparameter tuning might not be feasible.

The results successfully replicated the findings of the original paper, particularly in demonstrating NASG's robust performance across different learning rates and its strong convergence properties. The consistency between our experimental results and the paper's findings strengthens the validity of NASG as a reliable optimization algorithm.

References

1. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012).
2. He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
3. Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.
4. LeCun, Yann. "The MNIST database of handwritten digits." *http://yann. lecun. com/exdb/mnist/* (1998).
5. Soydaner, Derya. "A comparison of optimization algorithms for deep learning." *International Journal of Pattern Recognition and Artificial Intelligence* 34.13 (2020): 2052013.
6. Tran, Trang H., Katya Scheinberg, and Lam M. Nguyen. "Nesterov accelerated shuffling gradient method for convex optimization." *International Conference on Machine Learning*. PMLR, 2022.

Code

1. https://github.com/gaurisk17/NASG-Codes
2. https://github.com/gaurisk17/AST-CODES-MNIST
3. https://github.com/gaurisk17/AST-Codes-CIFAR10

Member Contribution

Member Name	Contributions	% of Contribution
Gauri Kulkarni	● Implemented ResNet and AlexNet architectures for CIFAR-10/MNIST ● Implemented and evaluated NASG (Nesterov Accelerated Shuffling Gradient) optimizer with various learning rates (1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001) ● Developed a Shell script for parallel model execution on CUDA servers, which would have significantly improved the efficiency of running multiple experiments with different learning rates	40
Abhishek Khedekar	● Focused on AlexNet architecture for both CIFAR-10 and MNIST ● Implemented all standard optimizers including: 1. SGD (with momentum variants) 2. SGDM (Stochastic Gradient Descent with Momentum) 3. Adaptive optimizers (Adam, AdaDelta, Nadam) 4. AMSGrad	30
Abhishek Patwardhan	● Focused on ResNet50 architecture for both CIFAR-10 and MNIST ● Similar to Khedekar, implemented all standard optimizers: ● SGD and variants ● SGDM ● Adaptive optimizers through AMSGrad	30