An Empirical Study of Optimizer Behaviour in Neural Network Training

Observations from Controlled Experiments on MNIST

Krish Patel

1 Introduction

Optimization algorithms play a central role in the training of neural networks, directly influencing convergence speed, stability, and generalization.

Learning rate schedules, training duration, and architectural choices can significantly alter optimization dynamics, making it difficult to attribute performance differences solely to the optimizer itself.

This work presents an empirical study of commonly used optimization algorithms under controlled and intentionally constrained experimental setups. Rather than tuning each optimizer for best possible performance, this study aims to observe and characterize optimizer behavior across different training regimes. By fixing the model architecture, dataset, and training protocol, we isolate the effects of the optimization algorithm and examine how its characteristics emerge under varying conditions.

2 Experimental Setup

This section describes the experimental configuration used across all studies, ensuring reproducibility and fair comparison.

2.1 Dataset

All experiments are conducted on the MNIST handwritten digit dataset, consisting of 60,000 training samples and 10,000 test samples. Images are converted to tensors using a standard `ToTensor` transformation.

2.2 Model Architecture

A fully connected neural network (MLP) is used throughout this study, with the following architecture:

$$784 \to 256 \to 128 \to 64 \to 10$$

ReLU activations are applied between all hidden layers.

2.3 Optimizers

The following optimizers are evaluated:

Optimizer	Learning Rate
SGD	0.1
Momentum	0.1
NAG	0.1
RMSProp	0.001
Adagrad	0.1
Adam	0.001
AdamW	0.001

2.4 Training Protocol and Metrics

All models are trained using cross-entropy loss with a batch size of 64. Training loss and test accuracy are logged after each epoch. A fixed random seed is used to ensure reproducibility across runs.

3 Experiment 1: Early Convergence

The first experiment examines how quickly different optimizers make progress during the initial phase of training.

3.1 Objective

The objective of this experiment is to evaluate early optimization speed, rather than long-term stability or final performance.

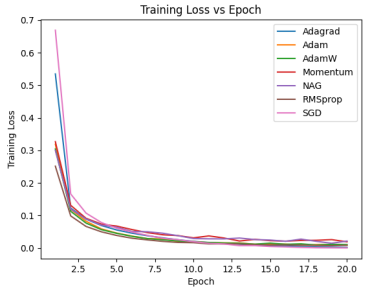
3.2 Setup

All optimizers are trained for 20 epochs using constant learning rates. No learning rate schedules are applied in this experiment.

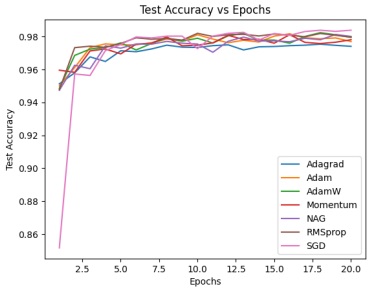
3.3 Observations

Momentum-based methods (Momentum and NAG) achieve high test accuracy within the first few epochs, indicating rapid early descent. In contrast, plain SGD exhibits slower initial progress. Adaptive optimizers such as Adam and RMSProp also converge quickly during this phase.

This experiment highlights differences in early optimization speed rather than long-term stability, explaining why SGD may appear inferior when only short training horizons are considered.



(a) Training loss



(b) Test accuracy

4 Experiment 2: Long Training with Constant Learning Rate

The second experiment investigates optimizer robustness under prolonged training without learning rate scheduling.

4.1 Objective

This experiment asks how optimizers behave when trained beyond their stable regime using a fixed learning rate.

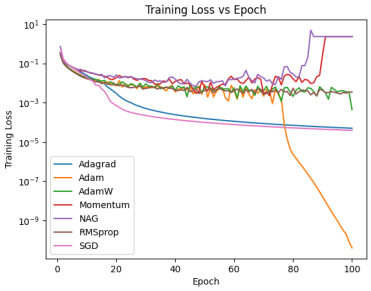
4.2 Setup

All optimizers are trained for 100 epochs with constant learning rates. No learning rate schedules or early stopping mechanisms are employed.

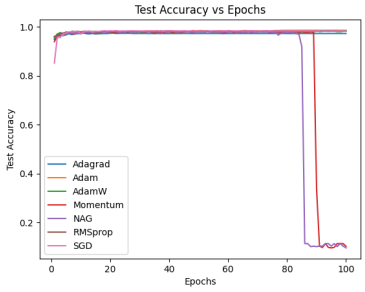
4.3 Observations

SGD, Momentum, and NAG exhibit late-stage instability, characterized by exploding training loss and a collapse in test accuracy. In contrast, Adam, AdamW, and RMSProp remain stable throughout training. Adagrad converges smoothly but plateaus early due to aggressive learning rate decay.

The observed divergence in SGD-based methods is not a bug, but a consequence of using a constant learning rate beyond its stable regime. This experiment emphasizes robustness rather than optimal usage.



(a) Training loss



(b) Test accuracy

5 Experiment 3: Optimizers in Their Intended Regime

The third experiment evaluates optimizers under configurations that align with their intended usage.

5.1 Objective

This experiment seeks to understand how optimizer behavior changes when appropriate learning rate strategies are applied.

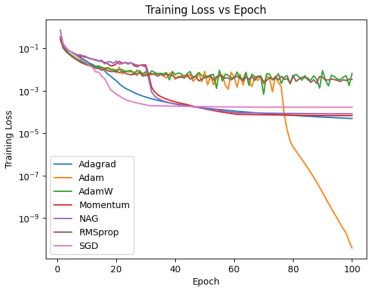
5.2 Setup

SGD, Momentum, and NAG are trained using a StepLR scheduler with step size 30 and decay factor 0.1. Adaptive optimizers (Adam, AdamW, RMSProp, Adagrad) are trained using constant learning rates. All models are trained for 100 epochs.

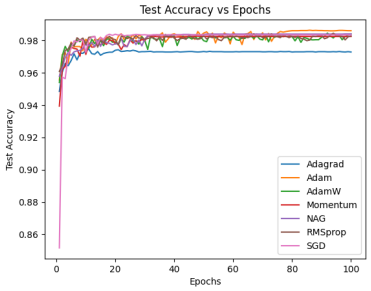
5.3 Observations

Under learning rate decay, SGD-based methods no longer exhibit instability and achieve competitive test accuracy. Generalization improves, and differences between optimizers become more subtle. Adam no longer appears dominant, highlighting that its apparent superiority in earlier experiments was regime-dependent.

This experiment reveals optimizer characteristics that are obscured when optimizers are evaluated under mismatched training conditions.



(a) Training loss



(b) Test accuracy

6 Comparative Discussion

Across experiments, distinct optimizer characteristics emerge. Adam optimizes aggressively and achieves rapid early progress, but often saturates without corresponding generalization gains. SGD benefits significantly from controlled step size reduction, demonstrating improved stability and competitive performance when used correctly. Momentum accelerates early descent, while NAG introduces additional sensitivity to curvature. Adagrad trades optimization speed for stability, resulting in early convergence but limited late-stage improvement.

These results indicate that optimizer performance is inherently regime-dependent and cannot be meaningfully summarized by a single ranking.

7 Limitations and Future Work

This study is limited to a single dataset, a single random seed, and a fully connected architecture. Hyperparameter sweeps and multi-seed averaging were not performed.

Future work(which i'll prolly never do) includes extending the analysis to convolutional architectures, harder datasets, learning rate sensitivity studies, and averaging results across multiple random seeds to improve statistical robustness.

8 Conclusion

This work demonstrates that there is no universally best optimizer. Optimizer behavior is conditional on training regime, learning rate strategy, and evaluation. Empirical analysis across multiple controlled experiments was necessary to meaningfully understand optimizer characteristics.

The implementation is available at: GitHub Repo