How Optimizers Generalize: A Study on Non-Convex Architectures

Mayeul Cassier, Elias Mir, Sacha Frankhauser
Optimization for Machine Learning course, EPFL, Switzerland

***Abstract*—We study the behavior of popular optimization algorithms in non-convex settings, focusing on their impact on generalization. Building on recent theoretical insights [1], [2], we compare optimizers such as SGD, Adam, and PGD across two datasets (SVHN, CIFAR10) and two architectures (MLP, CNN). Our results show that while adaptive methods converge faster, they do not always generalize better. Simpler models like MLPs show little sensitivity to the optimizer, suggesting capacity limitations.**

I. Introduction

Optimization lies at the heart of modern machine learning. While convex optimization has long provided a theoretical foundation for many algorithms, the practical models used today—particularly deep neural networks—are inherently **non-convex**. Understanding how optimizers behave in these non-convex settings is therefore critical.

In this work, we aim to empirically investigate optimization strategies in non-convex landscapes. Our analysis is informed by two foundational contributions: the review *Non-convex Optimization for Machine Learning* [1], which provides a broad theoretical and algorithmic framework, and the empirical study *Adaptive Methods for Nonconvex Optimization* [2], which focuses on the behavior of adaptive optimizers like Adam and RMSProp in such settings.

Using two widely studied image classification benchmarks—SVHN and CIFAR10—we compare a range of optimizers, including gradient-based, accelerated, and constrained methods, across two architectures: a simple MLP and a CNN. Our goal is to better understand how optimizer choice interacts with model complexity and data characteristics in non-convex optimization tasks.

II. Models and Methods

A. Architectures

We evaluate two primary architectures:

- **Multilayer Perceptron (MLP)**: A simple feedforward network with two hidden layers using ReLU activations. The input size adapts to the dataset (e.g., $3 \times 32 \times 32$ for CIFAR-10). It serves as a baseline for optimization dynamics in fully connected settings.
- **Convolutional Neural Network (CNN)**: A modular CNN built from a configurable sequence of convolutional layers (with optional max-pooling), followed by fully connected layers. Batch normalization and ReLU activations are applied after each convolution. The network can be adapted in depth and width by modifying a configuration dictionary passed to the constructor.

Both architectures were trained using cross-entropy loss on classification tasks. All experiments use consistent model architecture across optimizers to isolate their influence.

B. Optimization Algorithms

We organize the optimizers into three functional categories:
Free Methods: These optimizers apply gradient-based updates without projection or second-order information.

- **Gradient Descent (GD)**: A full-batch method, serving as a baseline for convergence speed and generalization.
- **Stochastic Gradient Descent (SGD)**: The mini-batch version of GD, widely used in deep learning.

Accelerated Methods: These methods augment basic gradient descent with additional momentum or curvature information to accelerate convergence.

- **Momentum**: Maintains a velocity vector that accumulates gradients to smooth out oscillations.
- **Nesterov Accelerated Gradient**: A look-ahead version of momentum that anticipates the next step.
- **Adam**: An adaptive optimizer maintaining running estimates of first and second moments. Tends to converge faster, but can overfit.
- **RMSProp**: Similar to Adam but without momentum on the first moment. It normalizes gradients by a running average of their magnitude.

Clipped / Constrained Methods: These optimizers incorporate projection steps to constrain weights after updates.

- **Projected Gradient Descent (PGD)**: Performs a gradient descent step, followed by a projection of weights into a constrained domain. We use the default **clipping** projection to keep weights in $[-1, 1]$.
- **Partial Gradient Descent (PartialGD)**: Updates only a random subset of parameters at each step, effectively introducing sparsity in gradient flow.

All optimizers were implemented as subclasses of PyTorch's `Optimizer` and accept configuration via dictionaries to facilitate fair comparison. For PGD, the projection operator uses the unit-sphere projection.

C. Training Protocol

- All models are trained for a fixed number of epochs (typically 30), or until early stopping on a held-out validation set (if enabled).

- Each configuration is run over 5 different random seeds to account for initialization and batch-order variability.
- For each run, we record: training loss, test loss, and test accuracy at each epoch.
- Additional metrics include: convergence time, variance across seeds, and epochs required to reach 90% of final accuracy.

D. Datasets

We use two standard image classification datasets:

- **SVHN**[3]: A simpler digit classification task with lower intra-class variability.
- **CIFAR-10**[4]: A moderately complex dataset with 10 classes and color images.

Preprocessing includes normalization and tensor conversion. Dataloaders are reused across seeds and models to ensure fairness.

E. Evaluation and Visualization

The following evaluation pipeline is used:

- Accuracy per optimizer is summarized using mean and standard error across seeds.
- Test Loss curves are visualized across epochs for each optimizer and dataset, with confidence bands.

III. Results

All results are averaged over 5 seeds and optimized over the learning rate l_r and the number of epochs for each optimizer individually. We report test loss evolution during training and final test accuracies.

A. CIFAR10

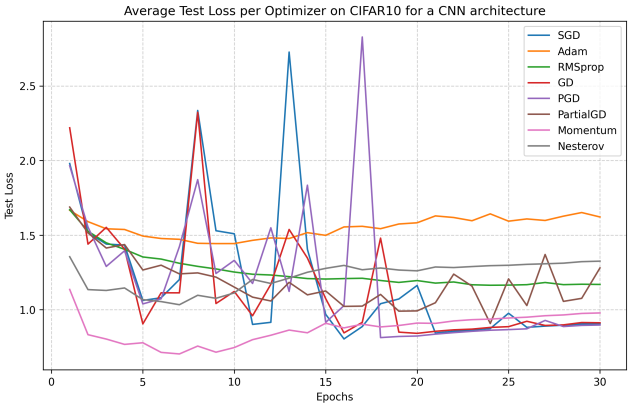

Fig. 1. Test loss over epochs for various optimizers on CIFAR10 using a CNN architecture

The plots in Figures 1 and 2 show the evolution of the test loss over training epochs for each optimizer.

Final Test Accuracy (CIFAR10): Table I summarizes the final test accuracy (averaged over 5 seeds) along with the standard deviation and learning rate used for each optimizer.

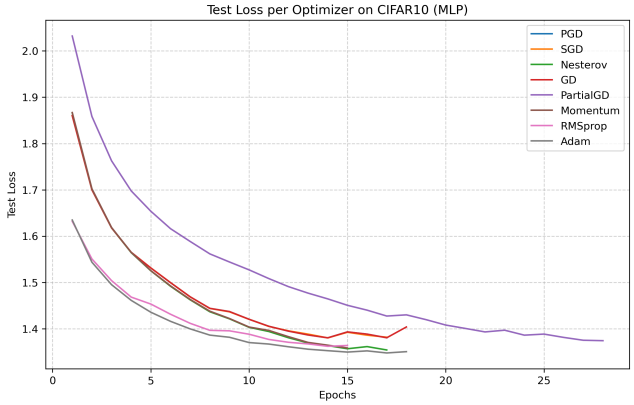

Fig. 2. Test loss over epochs for various optimizers on CIFAR10 using an MLP architecture

Optimizer	**CNN Acc (%)**	**MLP Acc (%)**	l_r (CNN)	l_r (MLP)
Gradient-based Methods				
GD	83.20 ± 0.31	51.09 ± 1.51	$5 \cdot 10^{-2}$	10^{-2}
SGD	83.02 ± 0.08	51.50 ± 0.82	$5 \cdot 10^{-2}$	10^{-2}
Accelerated Methods				
Momentum	82.55 ± 0.25	52.35 ± 0.24	$5 \cdot 10^{-3}$	10^{-3}
Nesterov	68.40 ± 29.20	52.52 ± 0.25	10^{-2}	10^{-3}
Adam	48.69 ± 14.74	52.84 ± 0.47	10^{-4}	10^{-4}
RMSProp	60.30 ± 9.26	52.17 ± 0.23	10^{-5}	10^{-4}
Clipped / Constrained Methods				
PGD	82.98 ± 0.22	51.56 ± 0.80	$5 \cdot 10^{-2}$	10^{-2}
PartialGD	64.68 ± 6.70	51.78 ± 0.30	$5 \cdot 10^{-3}$	10^{-2}

TABLE I
FINAL TEST ACCURACY FOR EACH OPTIMIZER ON CIFAR10 USING CNN AND MLP, GROUPED BY OPTIMIZER TYPE. LEARNING RATES l_r ARE SHOWN FOR BOTH MODELS.

B. SVHN

Figures 3 and 4 display the test loss over epochs for the different optimizers, respectively for CNN and MLP models.

Final Test Accuracy (SVHN): Final test accuracies are reported in Table II, together with the learning rates used for both CNN and MLP.

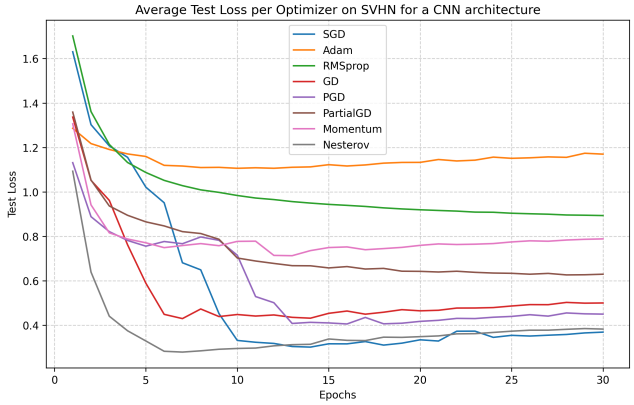

Fig. 3. Test loss over epochs for various optimizers on SVHN using a CNN architecture

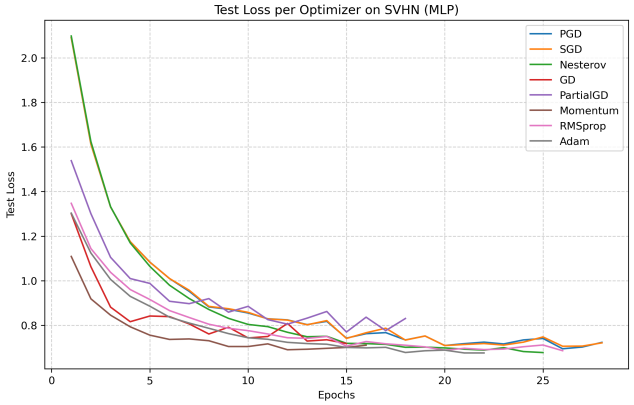

Fig. 4. Test loss over epochs for various optimizers on SVHN using an MLP architecture

Optimizer	**CNN Acc (%)**	**MLP Acc (%)**	l_r (CNN)	l_r (MLP)
Gradient-based Methods				
GD	87.13 ± 7.28	80.32 ± 0.59	$5 \cdot 10^{-2}$	10^{-1}
SGD	93.10 ± 0.14	80.85 ± 1.38	$5 \cdot 10^{-2}$	10^{-2}
Accelerated Methods				
Momentum	75.96 ± 34.63	80.53 ± 0.70	10^{-2}	10^{-2}
Nesterov	92.92 ± 0.11	82.05 ± 0.36	$5 \cdot 10^{-3}$	10^{-3}
Adam	60.03 ± 14.24	81.95 ± 0.25	$5 \cdot 10^{-5}$	10^{-4}
RMSProp	70.42 ± 12.64	81.80 ± 0.58	$5 \cdot 10^{-6}$	10^{-4}
Clipped / Constrained Methods				
PGD	90.13 ± 5.89	80.81 ± 1.28	$5 \cdot 10^{-2}$	10^{-2}
PartialGD	79.05 ± 10.53	76.83 ± 1.88	$5 \cdot 10^{-3}$	10^{-1}

TABLE II
FINAL TEST ACCURACY FOR EACH OPTIMIZER ON SVHN USING CNN AND MLP, GROUPED BY OPTIMIZER TYPE. LEARNING RATES l_r ARE SHOWN FOR BOTH MODELS.

IV. DISCUSSION

Our experimental results reveal several key trends in optimizer behavior across architectures and datasets.

Learning Rate Sensitivity of Accelerated Methods

A consistent observation is that **accelerated methods** (Momentum, Nesterov, Adam) generally require **smaller learning rates** than standard Gradient Descent variants. This trend is particularly pronounced for the **MLP**, where optimizers such as Nesterov or Adam perform best with $l_r \leq 10^{-3}$, while GD and PGD succeed with much larger values (up to 10^{-1}). This may be attributed to the increased sensitivity of momentum-based updates[5], where large steps can destabilize convergence due to gradient accumulation or poor initialization[6].

MLP: Uniformity Across Optimizers

Interestingly, the **MLP exhibits test accuracies that are remarkably stable across all optimizers**, with a variation of less than 2%. This suggests that the model capacity, rather than optimizer choice, is the main bottleneck. In other words, the simplicity of the MLP architecture limits the potential gains from optimization, since most optimizers already reach the model's capacity in terms of generalization[7], [8].

CNN on CIFAR10: Chaotic Loss Curves and Fast Convergence

For the **CNN on CIFAR10**, the test loss curves are often **chaotic**, especially for GD, PGD, and PartialGD. These curves show sudden increases or local maxima, which can be explained by the fact that the model quickly reaches a minimum and then oscillates around it. This behavior is typical of non-convex optimization problems, where small shifts in data batches can lead to learning different features. In contrast to the MLP, the CNN often achieves good performance within just a few epochs, especially on CIFAR10 [9], possibly due to its higher expressiveness and inductive biases (e.g., local connectivity and weight sharing).

Test Loss Dynamics: MLP vs. CNN

The MLP generally produces **smooth and monotonically decreasing test loss curves**, with a well-defined minimum. This is consistent with a model that requires more optimization effort to achieve generalization, potentially due to its lack of structural priors. Conversely, CNNs on CIFAR10 often display early and sharp improvements, followed by irregular behavior. This contrast further supports the hypothesis that CNNs learn faster[9], but are harder to control, especially without explicit regularization.

SVHN vs. CIFAR10: Dataset Complexity

Overall, **SVHN leads to higher test accuracies** than CIFAR10 across both MLP and CNN. This is expected, as CIFAR10 is a more complex dataset requiring spatial feature extraction, for which MLPs are clearly underpowered. That said, some accelerated methods perform poorly with CNNs on CIFAR10, often showing large variances across runs. One plausible reason is the **absence of weight decay** in our experiments. Without regularization, some optimizers overfit or diverge more easily, especially when coupled with early stopping on noisy loss curves.

V. FINAL TAKEAWAYS

In conclusion, our results suggest that:

- **Accelerated methods are harder to fine-tune**, especially in non-convex settings;
- **Gradient-based and constrained optimizers (GD, PGD)** perform as well—or even better—in some cases, while being more stable and easier to tune;
- **Model complexity and regularization** play a major role in loss dynamics and final accuracy;
- For practical use, especially with simple architectures like MLPs, **vanilla Gradient Descent may suffice**.

Overall, this empirical study confirms that many of the observations made in convex or simplified settings **extend well to non-convex deep learning problems**.

Future work could explore the impact of these optimizers in larger-scale regimes, under more aggressive regularization, or in combination with modern training practices such as learning rate scheduling and data augmentation.

REFERENCES

[1] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, "Non-convex optimization for machine learning," *arXiv preprint arXiv:1712.07897*, 2017. [Online]. Available: https://arxiv.org/abs/1712.07897

[2] D. Zhou, T. Tang, P. Yang, and Q. Gu, "Adaptive methods for nonconvex optimization," in *Advances in Neural Information Processing Systems*, vol. 32, 2020. [Online]. Available: https://papers.neurips.cc/paper_files/paper/2019/file/f46eab1e3f2cfb03f6c1cb5a5f00f6d4-Paper.pdf

[3] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/

[4] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[5] Z. Xie, X. Wang, H. Zhang, I. Sato, and M. Sugiyama, "Adaptive inertia: Disentangling the effects of adaptive learning rate and momentum," 2022. [Online]. Available: https://arxiv.org/abs/2006.15815

[6] C. Liu and M. Belkin, "Accelerating sgd with momentum for over-parameterized learning," 2019. [Online]. Available: https://arxiv.org/abs/1810.13395

[7] R. M. Schmidt, F. Schneider, and P. Hennig, "Descending through a crowded valley - benchmarking deep learning optimizers," 2021. [Online]. Available: https://arxiv.org/abs/2007.01547

[8] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Sensitivity and generalization in neural networks: an empirical study," 2018. [Online]. Available: https://arxiv.org/abs/1802.08760

[9] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," 2018. [Online]. Available: https://arxiv.org/abs/1712.09913