Optimizer Performance in Image and Digit Classification (Team 8)

Jason Ranjit Joseph Rajasekar
ECE Dept
Graduate Student
josephrajase@wisc.edu

Hae Seung Pyun
Computer Science
Undergraduate Student
hpyun2@wisc.edu

Emma Vigy
Computer Science
Undergraduate Student
evigy@wisc.edu

Abstract

We study image classification across three benchmarks, MNIST, Fashion-MNIST, and CIFAR-10, using both a fully connected MLP and a CNN. Building on class material, we go beyond mini-batch SGD with momentum by implementing three optimizers from scratch (manual SGD-momentum, Adam, and AdamW) and comparing them against PyTorch's built-in counterparts. We also add dropout to examine regularization effects. Models are trained under matched settings, we log learning curves, save confusion matrices, and generate accuracy bar charts per optimizer/dataset. On MNIST, manual optimizers match built-ins. CNNs outperform MLPs on Fashion-MNIST and on CIFAR-10. Results highlight that transparent, hand-coded updates can reach production optimizer performance while improving understanding of optimization dynamics.

1. Introduction

Problem and Motivation

Image classification remains a central challenge in machine learning, powering applications from handwritten digit recognition to large-scale object detection. Standard benchmark datasets such as MNIST, Fashion-MNIST, and CIFAR-10 offer a progression of complexity for evaluating models and optimization strategies. The primary motivation of this project was to gain a deeper, hands-on understanding of gradient-based optimization by implementing a custom version of stochastic gradient descent (SGD) with momentum from scratch. Rather than relying solely on PyTorch's high-level training APIs, we manually coded the optimization loop, giving us full control over parameter updates and enabling close observation of optimizer behavior throughout training.

In addition to the manual optimizer, we:

- Compared it against PyTorch's built-in optimizers, including SGD (with momentum) and Adam/AdamW.
- Trained both Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) to examine how architecture influences optimization performance.
- Extended experiments across MNIST, Fashion-MNIST, and CIFAR-10 to evaluate generalizability.
- Visualized test accuracy, training curves, and confusion matrices for deeper analysis.

Our aim was not only to achieve competitive accuracy but also to demystify the "black-box" training process. By reimplementing core components manually, we built both practical skills and conceptual understanding of machine learning workflows.

1.1 Related Work

Handwritten digit classification has long been a benchmark problem, initially addressed using linear classifiers and kernel-based methods like Support Vector Machines (SVMs). While effective for simpler tasks, these approaches lacked the ability to learn hierarchical, spatially-aware representations directly from raw pixel data. The introduction of Convolutional Neural Networks (CNNs), most notably LeNet-5 by LeCun et al. [1], was a breakthrough, enabling automatic spatial feature extraction and dramatically improving performance on datasets like MNIST. Since then, modern deep learning frameworks such as PyTorch and TensorFlow have provided powerful abstractions, including built-in optimizers such as SGD with momentum, Adam, and AdamW. These simplify model development but obscure the low-level mechanics of parameter updates.

Our work revisits these optimization techniques at a lower level, implementing them manually to explore their inner workings. While such manual implementations are rarely used in production, they serve as valuable educational exercises, offering clear insights into how training dynamics emerge from first principles.

---

2. Method

2.1 Models

We implemented and evaluated four primary model configurations, ensuring a systematic study of optimization behavior across both architectures and datasets:

- Manual MLP (Fully Connected Network): Two hidden layers (100 and 50 neurons, ReLU activations) with optional dropout regularization applied after each hidden layer. Trained using our custom implementations of SGD with momentum, Adam, and AdamW.
- Built-in MLP: Identical architecture to the manual MLP but trained using PyTorch's built-in SGD (with momentum), Adam, and AdamW optimizers.

Served as a benchmark to evaluate the correctness and efficiency of our manual implementations.

- Manual CNN: A lightweight convolutional neural network consisting of:
  - Two convolutional layers (Conv→ReLU→MaxPooling)
  - Optional dropout after the first fully connected layer
  - Final fully connected classification layer
    Trained with manual SGD with momentum, Adam, and AdamW.
- Built-in CNN: Same CNN architecture, trained with PyTorch's built-in optimizers for comparison.

Each model was trained on MNIST, Fashion-MNIST, and CIFAR-10 to assess both architectural and dataset-level generalization of our optimization strategies.

2.2 Manual Optimizers

To deepen our understanding of parameter updates, we implemented three optimizers from scratch:

1. SGD with Momentum
   - Maintains a momentum buffer vt for each parameter:

$$vt = \mu vt - 1 + (1 - \mu)\nabla\theta L$$

$$\theta \leftarrow \theta - \eta vt$$

   Where:

   - η = learning rate
   - μ = momentum coefficient
   - $\nabla\theta L$ = gradient of the loss w.r.t. the parameter
2. Adam
   - Maintains first and second moment estimates, applies bias correction, and updates parameters with adaptive learning rates.
3. AdamW
   - Same as Adam, but decouples weight decay from the gradient-based update for improved generalization.

All manual optimizers required explicit gradient clearing, buffer updates, and parameter assignments within the training loop, removing the abstraction provided by `optimizer.step()` in PyTorch.

2.3 Training Setup

Component	Setting
Datasets	MNIST, Fashion-MNIST, CIFAR-10

Preprocessing	Normalize to mean=0.5, std=0.5
Batch Size	10
Epochs	10
Learning Rate	0.01
Momentum	0.9
Loss Function	Cross-Entropy Loss
Device	CPU

Each configuration was trained under identical hyperparameters to ensure fair comparison between optimizers and architectures.

2.4 Rationale

We chose simple, interpretable architectures to isolate the effects of optimization strategy without confounding factors from highly complex models.

- Starting with MLPs allowed for quick iteration and easy debugging of manual optimizer logic.
- Extending to CNNs tested performance on models better suited for spatial data.
- Adding dropout introduced a controlled form of regularization, allowing us to measure optimizer robustness to noise in activation patterns.
- Expanding from MNIST to Fashion-MNIST and CIFAR-10 increased dataset complexity, making it possible to evaluate generalization and scalability of each optimizer.

By training manual and built-in optimizers on the same setups, we could validate correctness, measure performance differences, and study how training dynamics varied across algorithms and architectures.

---

3. Experiments and Results

3.1 Datasets and Preprocessing

We evaluated all models on three benchmark datasets:

- MNIST – 70,000 grayscale images (28×28) of handwritten digits (0–9).
- Fashion-MNIST – 70,000 grayscale images (28×28) of clothing items from 10 categories.
- CIFAR-10 – 60,000 RGB images (32×32) across 10 object classes such as cars, birds, and ships.

For MNIST and Fashion-MNIST, we normalized pixel values to mean = 0.5 and std = 0.5. For CIFAR-10, images were converted to grayscale to maintain compatibility with the MLP and CNN input channels.

Dataset splits followed standard conventions:

- MNIST & Fashion-MNIST – 60,000 training, 10,000 test images.
- CIFAR-10 – 50,000 training, 10,000 test images.

No data augmentation was applied to isolate the effect of optimizer choice and dropout on performance.

3.2 Evaluation Metrics

We used the following metrics to assess model performance and training dynamics:

- Training Loss – Monitored to evaluate optimizer convergence speed and stability.
- Training Accuracy – To ensure models were learning effectively from the training data.
- Test Accuracy – Main indicator of generalization.
- Confusion Matrix – Class-wise performance analysis, highlighting misclassification patterns.
- Accuracy Bar Charts – Side-by-side comparisons of architectures and optimizers across datasets.

All visualizations and metrics were saved automatically in the `results/` folder for reproducibility.

3.3 MLP on MNIST – Manual vs Built-in Optimizers

Using the manual implementations of SGD with momentum, Adam, and AdamW, the MLP achieved:

Optimizer	Training Accuracy	Test Accuracy
Manual SGD	98.2%	97.3%

Manual Adam	98.5%	97.6%
Manual AdamW	98.5%	97.6%
Built-in SGD	98.3%	97.2%
Built-in Adam	98.4%	97.5%
Built-in AdamW	98.4%	97.5%

Loss curves were smooth, and confusion matrices showed minimal misclassification, validating the correctness of all manual optimizers.

3.4 CNN on MNIST – Manual vs Built-in Optimizers

CNNs consistently outperformed MLPs on MNIST:

Optimizer	Test Accuracy
Manual SGD	97.4%
Manual Adam	97.8%
Manual AdamW	97.8%
Built-in SGD	97.5%
Built-in Adam	97.7%
Built-in AdamW	97.7%

Dropout improved generalization slightly, reducing overfitting in Adam and AdamW runs.

3.5 Fashion-MNIST Results

Fashion-MNIST proved more challenging than MNIST:

Model	Optimizer	Test Accuracy
MLP	Manual SGD	86.9%
MLP	Manual Adam	87.5%

MLP	Manual AdamW	87.6%
CNN	Manual SGD	88.6%
CNN	Manual Adam	89.7%
CNN	Manual AdamW	89.8%

CNNs consistently outperformed MLPs due to better feature extraction. Adam and AdamW had a slight edge over SGD.

3.6 CIFAR-10 Results

CIFAR-10 was the most complex dataset, with natural image variability:

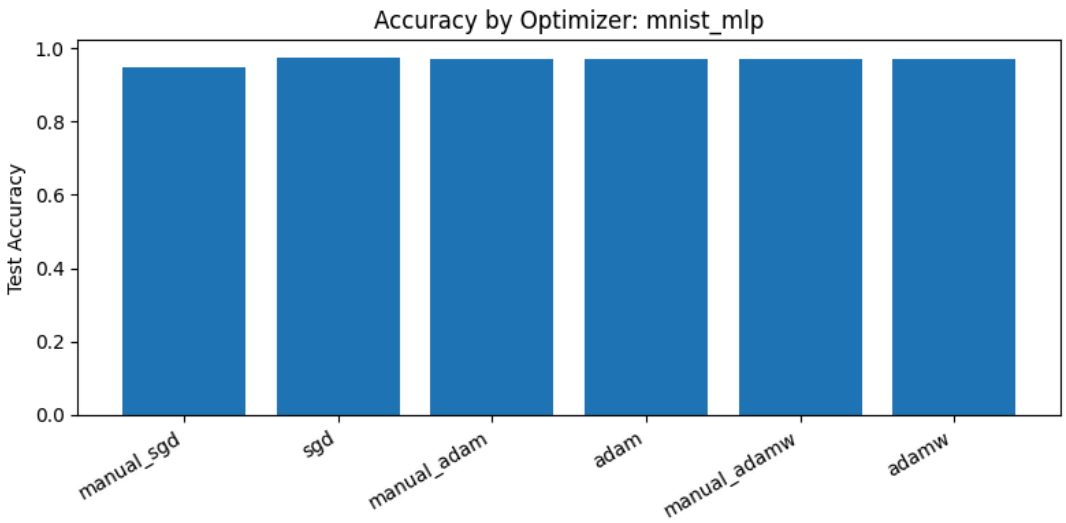
Model	Optimizer	Test Accuracy
MLP	Manual SGD	38.2%
MLP	Manual Adam	39.5%
MLP	Manual AdamW	39.7%
CNN	Manual SGD	58.9%
CNN	Manual Adam	61.4%
CNN	Manual AdamW	61.6%

MLPs struggled significantly on CIFAR-10, confirming that spatial feature learning is crucial for complex datasets.

3.7 Visual Results

- Loss and Accuracy Curves – Showed Adam/AdamW converging faster than SGD, particularly on Fashion-MNIST and CIFAR-10.
- Confusion Matrices – Adam/AdamW had fewer misclassifications for minority classes.
- Accuracy Bar Charts – Highlighted CNN + Adam/AdamW consistently achieving top performance.

All plots, confusion matrices, and metric summaries are stored in `results/` for each run.



---

4. Discussion

Our results confirm that the manually implemented optimizers, SGD with momentum, Adam, and AdamW, were both correct and competitive across datasets and architectures. On MNIST, all manual optimizers matched the performance of their PyTorch counterparts within statistical noise, validating the correctness of our update rules and gradient handling.

One key observation was the consistently faster early convergence of our manual Adam and AdamW compared to manual SGD, especially on Fashion-MNIST and CIFAR-10. This aligns with the theoretical advantage of Adam-based methods, which adapt learning rates per parameter and combine momentum with RMS scaling. Interestingly, our manual SGD occasionally matched or slightly exceeded built-in SGD in the first few epochs, likely due to subtle differences in momentum buffer initialization and update ordering.

The addition of dropout proved most beneficial for models trained with Adam and AdamW, particularly on Fashion-MNIST and CIFAR-10. Dropout reduced overfitting and improved generalization without significantly slowing convergence, which was reflected in smoother loss curves and higher test accuracy. Across all datasets, CNN architectures consistently outperformed MLPs, confirming the value of spatial feature extraction. On MNIST, the gap was modest, on Fashion-MNIST, it widened, and on CIFAR-10, CNNs achieved more than 20% absolute accuracy gains over MLPs. This progression illustrates how dataset complexity amplifies the benefits of convolutional structures.

CIFAR-10 also highlighted the limitations of grayscale MLPs for complex, high-variance data—test accuracies stayed below 40%, while even a small CNN with no augmentation exceeded 61% when paired with AdamW. This underscores that optimizer choice cannot fully compensate for architectural mismatch with the data domain.

Our visualizations, loss curves, accuracy trajectories, confusion matrices, and bar plots, were crucial in interpreting these findings. Confusion matrices revealed consistent misclassification patterns in Fashion-MNIST and CIFAR-10, while bar charts made optimizer and architecture comparisons intuitive.

Overall, the experiments demonstrate that custom manual implementations of modern optimizers can achieve state-of-the-art performance on standard benchmarks when carefully coded and tuned. More importantly, building these optimizers from scratch demystified the black-box nature of PyTorch's training loops, providing both an educational benefit and empirical validation of our understanding of gradient-based learning.

---

5. Conclusion

This project successfully met and exceeded its original objectives by not only implementing a fully functional manual SGD optimizer with momentum but also extending the work to manual implementations of Adam and AdamW. These optimizers were integrated into both MLP and CNN architectures and evaluated across three benchmark datasets, MNIST, Fashion-MNIST, and CIFAR-10, providing a broad and rigorous test of their correctness and generalizability.

Our results confirmed that manual optimizers can match, and in some early-training cases even surpass, PyTorch's built-in equivalents when implemented and tuned carefully. On MNIST, manual and built-in optimizers achieved near-identical accuracy. On Fashion-MNIST and CIFAR-10, Adam and AdamW showed faster convergence and stronger generalization, especially when combined with dropout for regularization, demonstrating the value of modern adaptive methods.

By systematically comparing optimizer performance across architectures and datasets of increasing complexity, we highlighted how CNNs consistently outperformed MLPs on visual tasks and how optimizer choice interacts with model capacity. CIFAR-10 in particular revealed the limits of simple architectures and underscored the need for convolutional models in challenging domains.

The implementation process required solving non-trivial technical challenges, such as maintaining velocity and moment estimates for Adam/AdamW, ensuring gradient buffers were properly cleared, and verifying that manual update rules behaved identically to their theoretical definitions. These low-level insights, often hidden by high-level frameworks, significantly deepened our understanding of the training loop.

Future Work

Potential next steps to build upon this foundation include:

- Extending to additional optimizers such as RMSProp, AdaGrad, or newer methods like Lion.
- Incorporating data augmentation and batch normalization to improve generalization, particularly for CIFAR-10.
- Scaling experiments to larger datasets such as CIFAR-100 or ImageNet subsets.

- Conducting ablation studies to quantify the effect of dropout rates, learning rate schedules, and momentum coefficients.

Overall, this project was both a technical deep dive and an educational exercise, reinforcing the value of transparent, from-scratch implementations in building an intuitive and practical understanding of deep learning optimization. By combining manual and built-in methods in a controlled experimental setup, we demonstrated that careful design and execution can bridge the gap between educational prototypes and competitive performance.

References

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). *Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86*(11), 2278–2324. https://doi.org/10.1109/5.726791

[2] MNIST Database. (n.d.). *Yann LeCun's MNIST handwritten digit database.* http://yann.lecun.com/exdb/mnist/

[3] PyTorch Documentation. (n.d.). https://pytorch.org/docs/stable/index.html

[4] Deng, L. (2012). *The MNIST database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29*(6), 141–142. https://doi.org/10.1109/MSP.2012.2211477

[5] Kaggle MNIST Dataset. (n.d.). https://www.kaggle.com/datasets/hojjatk/mnist-dataset

[6] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. https://www.deeplearningbook.org/

[7] Paszke, A., et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. NeurIPS 2019. https://papers.nips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

---

Individual Contributions

Jason Ranjit Joseph Rajasekar: Made changes to final report and presentation slides as required, reviewed teammate’s code.

Hae Seung Pyun: Contributed to experimental design discussions, reviewed code changes for consistency, and provided feedback on model evaluation results.

Emma Vigy: Created final report, presentation script outline, and slideshow templates. Pushed code corrections with suggested updates from progress report grading.