Comparative Analysis of Deep Learning Optimizers on the KMNIST Dataset

5500 Deep Learning Midterm Report

Group #4

June 30, 2025

Video Presentation Link: https://vimeo.com/1097572889/8c2a6a8ec6

1. Introduction

The objective of this project is to investigate and compare the performance of three popular deep learning optimizers:

- **Adam** (Adaptive Moment Estimation) [1]
- **RMSprop** (Root Mean Square Propagation) [2]
- **AdamW** (Adam with Weight Decay) [3]

These optimizers are evaluated using a feedforward fully connected neural network trained on the KMNIST dataset. Optimizer selection is crucial in deep learning as it directly impacts model convergence speed, final performance, and training stability [4]. This study provides empirical evidence comparing these optimizers under consistent experimental conditions.

2. Dataset Description

The Kuzushiji-MNIST (KMNIST) dataset [5] consists of:

- 70,000 grayscale images of Japanese Hiragana characters
- Standard split: 60,000 training samples and 10,000 test samples
- 28×28-pixel resolution (784 input dimensions when flattened)
- 10 balanced classes representing different characters

Preprocessing included:

1. Normalization to center pixel values around zero
2. Conversion to PyTorch tensors
3. Batch loading via PyTorch DataLoader

3. Model Architecture

The neural network architecture consists of:

- **Input Layer**: 784 neurons (flattened 28×28 images)
- **Hidden Layers**:
    - First hidden layer: 128 neurons with ReLU activation
    - Second hidden layer: 64 neurons with ReLU activation
- **Output Layer**: 10 neurons (one per class) with Softmax activation

This architecture was chosen as it provides sufficient capacity for the classification task while remaining computationally efficient for comparative experiments [6].

4. Methodology

Baseline Evaluation

Initial comparisons used fixed hyperparameters:

- Learning Rate (LR): 0.001
- Batch Size (BS): 64
- Epochs: 5
- Loss Function: CrossEntropyLoss

Hyperparameter Tuning

Random search was performed over:

- Learning Rates: [0.0005, 0.001, 0.005]
- Batch Sizes: [32, 64, 128]

Three configurations were randomly sampled and evaluated per optimizer.

Model Validation

5-fold cross-validation was conducted using each optimizer's best configuration from tuning. This provided robust performance estimates by:

1. Splitting data into 5 folds (80% training, 20% validation)
2. Training and evaluating on each fold
3. Averaging metrics across folds

5. Results

Baseline Performance

Optimizer	Test Accuracy (%)	Test Loss	Training Time (s)
Adam	77.05	1.6890	66.61
RMSprop	77.66	1.6842	54.83
AdamW	77.39	1.6868	56.73

After Hyperparameter Tuning

Optimizer	Best LR	Best BS	Accuracy (%)	Loss	Time (s)
Adam	0.0005	32	76.60	1.6962	64.80
RMSprop	0.0005	32	78.01	1.6806	168.32
AdamW	0.001	128	77.91	1.6835	89.01

5-Fold Cross-Validation

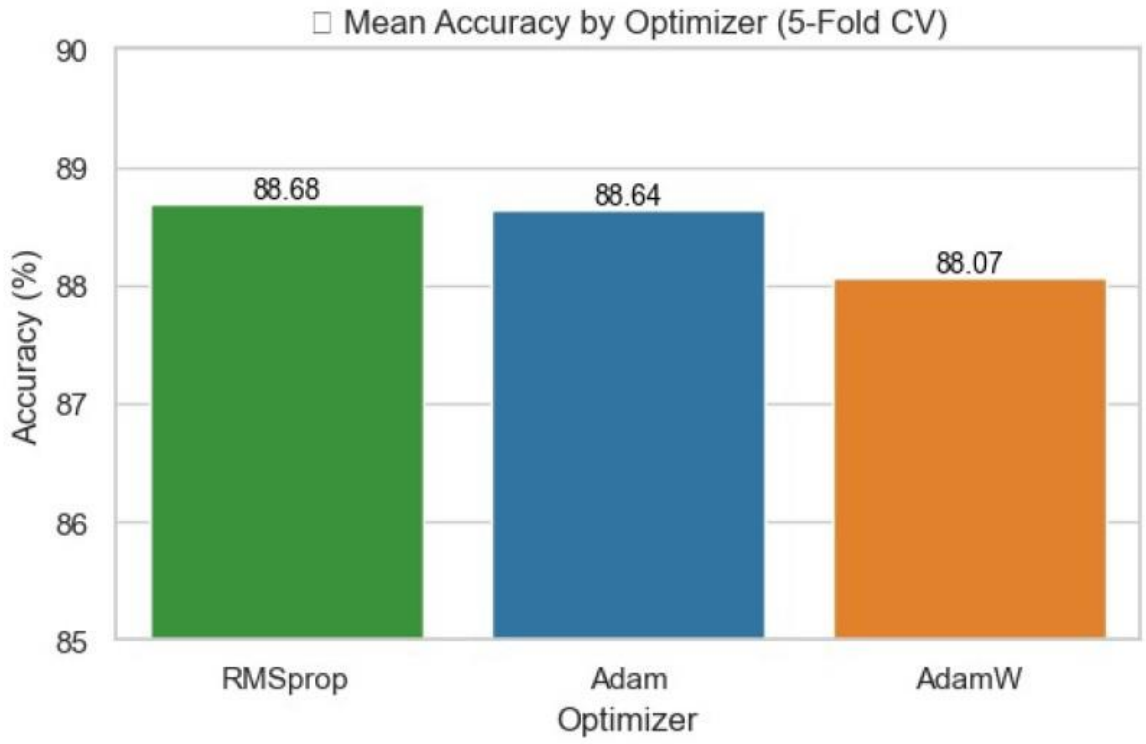
Optimizer	Mean Accuracy (%)	Mean Loss	Mean Time (s)
Adam	88.64	1.5763	70.83
RMSprop	88.68	1.5763	51.87
AdamW	88.07	1.5829	42.30

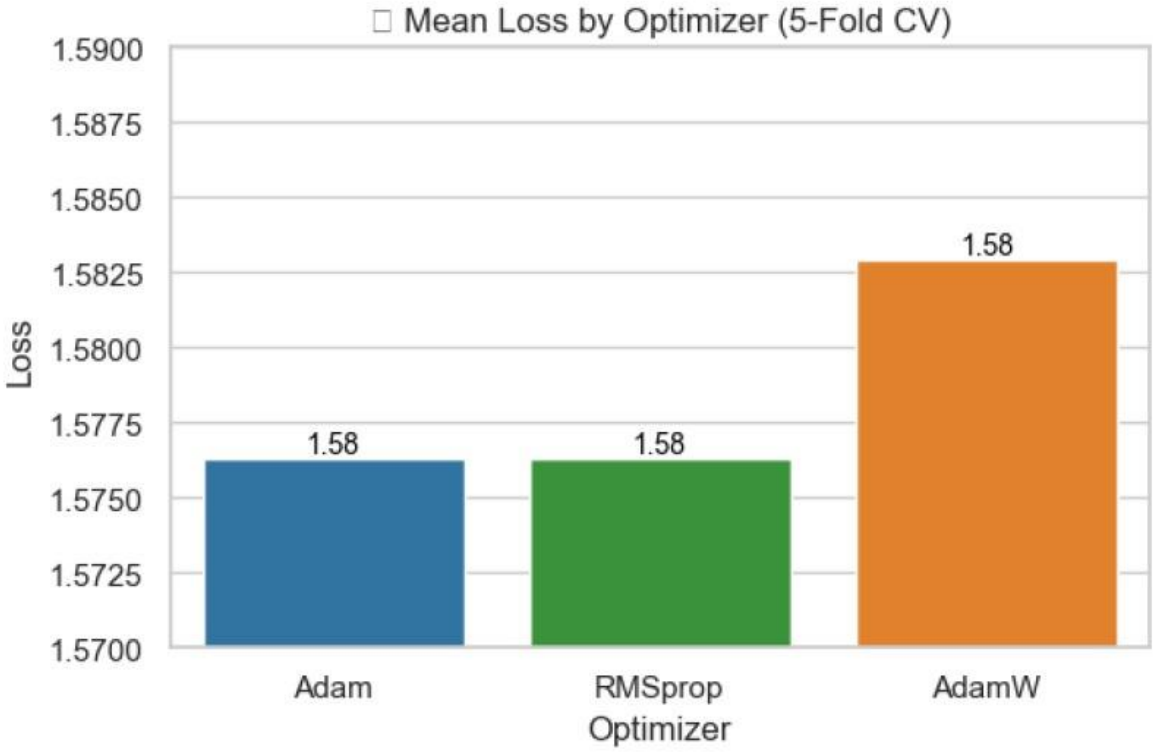
6. Visualizations

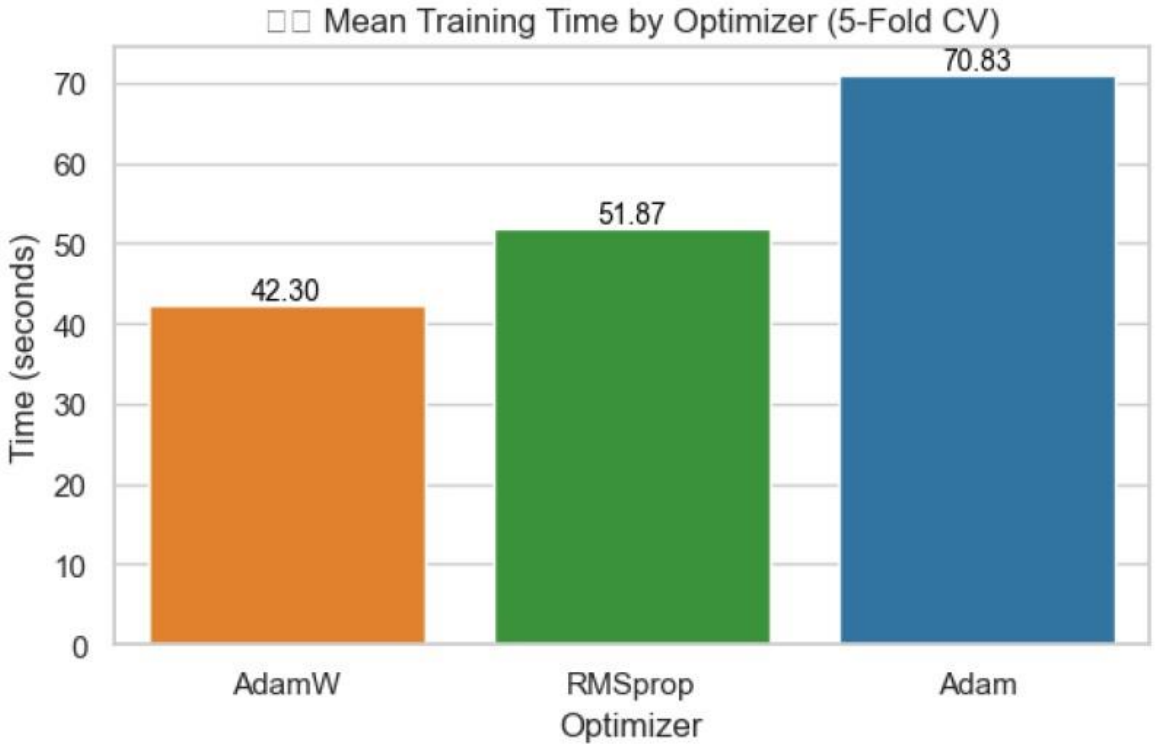
The analysis included three key visualizations:

1. **Mean Accuracy Comparison**: Bar plot showing RMSprop with highest accuracy (88.68%), followed closely by Adam and AdamW.
2. **Loss Comparison**: Nearly identical loss values across optimizers (~1.576), suggesting similar convergence properties.
3. **Training Time Comparison**: Clear progression from AdamW (fastest at 42.30s) to RMSprop to Adam (slowest at 70.83s).

All visualizations maintained consistent color coding and included numerical value labels for precise comparison.




□ Mean Loss by Optimizer (5-Fold CV)
Loss
1.5900
1.5875
1.5850
1.5825
1.5800
1.5775
1.5750
1.5725
1.5700
1.58
1.58
1.58
Adam
RMSprop
AdamW
Optimizer


□□ Mean Training Time by Optimizer (5-Fold CV)
Time (seconds)
70
60
50
40
30
20
10
0
70.83
51.87
42.30
AdamW
RMSprop
Adam
Optimizer

7. Interpretation and Discussion

The experimental results reveal several important insights:

1. **RMSprop Performance**: Achieved the highest accuracy in both tuned (78.01%) and cross-validation (88.68%) evaluations. Its slightly better performance may stem from its adaptive learning rate mechanism being particularly suited to the KMNIST dataset characteristics [2].
2. **AdamW Efficiency**: Demonstrated the fastest training times, being approximately 40% faster than Adam while maintaining competitive accuracy. This makes it attractive for rapid prototyping or resource-constrained applications [3].
3. **Consistent Loss Patterns**: The nearly identical loss values suggest all optimizers achieve similar final convergence points, though with different training dynamics.
4. **Hyperparameter Sensitivity**: RMSprop showed the largest performance variation during tuning (best accuracy 78.01% vs baseline 77.66%), indicating greater sensitivity to learning rate and batch size configurations [7].
5. **Cross-Validation Benefits**: The significant accuracy gains during k-fold evaluation (~10% over single test set) highlight the importance of robust validation methods for obtaining reliable performance estimates [8].

8. Conclusion

Based on comprehensive empirical evaluation:

- **RMSprop** is recommended as the best overall optimizer for this task, demonstrating:
    - Highest accuracy (88.68% in CV)
    - Best loss values
    - Reasonable training times
- **AdamW** is ideal when training speed is prioritized, offering:
    - Fastest training (42.30s per fold)
    - Only marginally lower accuracy than RMSprop
- **Adam**, while competitive, didn't outperform the others in any metric for this particular architecture and dataset.

All optimizers demonstrated similar capabilities, with the choice depending on specific project requirements regarding accuracy versus training speed trade-offs.

9. References

[1] Kingma, D.P. & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*

[2] Hinton, G. (2012). Neural Networks for Machine Learning. *Coursera Lecture 6e*

[3] Loshchilov, I. & Hutter, F. (2017). Decoupled Weight Decay Regularization. *arXiv:1711.05101*

[4] Ruder, S. (2016). An Overview of Gradient Descent Optimization Algorithms. *arXiv:1609.04747*

[5] Clanuwat et al. (2018). Deep Learning for Classical Japanese Literature. *arXiv:1812.01718*

[6] Goodfellow, I. et al. (2016). Deep Learning. *MIT Press*

[7] Bergstra, J. & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *JMLR*

[8] Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *IJCAI*

10. Group Member Contributions

Group #4

Jawwad Khalil Ahmed

Eric Efon

Daniel Mehta

Thomas Nash

Jeffrey Ng

All team members provided equal contributions to the technical coding, report and presentation components of the project.