AMATH 482/582: HOME WORK 4

REBECCA WANG

Amath Department, University of Washington, Seattle, WA
lufanw@uw.edu

Abstract. This report explores the design and optimization of a Fully Connected Neural Network (FCN) for FashionMNIST image classification. We implement an adjustable FCN, train it with varying hyperparameters, and evaluate its performance. Key experiments include testing different optimizers, weight initializations, and regularization techniques to improve accuracy. Our findings provide insights into optimizing deep learning models for image classification.

1. Introduction and Overview

Deep learning enables effective image classification, with FashionMNIST serving as a challenging benchmark. This study develops and optimizes an FCN to classify images into ten categories. We first implement a baseline model, then tune hyperparameters such as layer configurations, learning rates, and optimization methods. By analyzing training loss and accuracy, we identify strategies to enhance model performance efficiently.

2. Theoretical Background

2.1. Fully Connected Neural Networks. A Fully Connected Neural Network (FCN) consists of layers where each neuron in one layer is connected to all neurons in the next. It is used for learning hierarchical representations. The forward propagation for layer l follows:

$$h^{(l)} = f(W^{(l)}h^{(l-1)} + b^{(l)}). \tag{1}$$

2.2. Activation Functions. Activation functions introduce non-linearity, preventing FCNs from being simple linear models. Common functions include:

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \text{ReLU}(x) = \max(0, x), \quad \text{LeakyReLU}(x) = \max(ax, x). \tag{2}$$

2.3. Loss Function. For multi-class classification, the **cross-entropy loss** is optimized during training to improve prediction accuracy, defined as:

$$L(y, \hat{y}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i. \tag{3}$$

Date: March 17, 2025.

2.4. Optimization Algorithms. Optimization algorithms iteratively update model weights to minimize the loss function. **SGD, RMSProp, and Adam** are commonly used.

Stochastic Gradient Descent (SGD) updates weights using:

$$w^{(t+1)} = w^{(t)} - \alpha \frac{\partial J}{\partial w}, \tag{4}$$

where α is the learning rate. Adding **momentum** improves convergence:

$$v^{(t)} = \beta v^{(t-1)} + (1-\beta)\frac{\partial J}{\partial w}, \quad w^{(t+1)} = w^{(t)} - \alpha v^{(t)}. \tag{5}$$

RMSProp adapts learning rates per parameter:

$$s^{(t)} = \gamma s^{(t-1)} + (1-\gamma)\left(\frac{\partial J}{\partial w}\right)^2, \quad w^{(t+1)} = w^{(t)} - \frac{\alpha}{\sqrt{s^{(t)}+\epsilon}}\frac{\partial J}{\partial w}. \tag{6}$$

Adam (Adaptive Moment Estimation) combines momentum and adaptive learning rates:

$$m^{(t)} = \beta_1 m^{(t-1)} + (1-\beta_1)\frac{\partial J}{\partial w}, \quad s^{(t)} = \beta_2 s^{(t-1)} + (1-\beta_2)\left(\frac{\partial J}{\partial w}\right)^2. \tag{7}$$

Bias correction ensures stability:

$$w^{(t+1)} = w^{(t)} - \frac{\alpha}{\sqrt{\hat{s}^{(t)}+\epsilon}}\hat{m}^{(t)}. \tag{8}$$

Learning Rate Decay is applied in training to improve convergence stability by gradually decreasing the learning rate:

$$\alpha = d_{\text{epnum}} \cdot \alpha_0. \tag{9}$$

2.5. Overfitting/Underfitting and Regularization. Overfitting occurs when a model memorizes noise instead of patterns, leading to poor generalization. It is indicated by **high training accuracy but stagnant or declining validation accuracy**, along with **increasing validation loss despite decreasing training loss**. **Underfitting** happens when a model is too simple to capture patterns, resulting in **low accuracy and persistently high loss** on both training and validation sets. **Dropout Regularization** mitigates overfitting by randomly deactivating neurons during training, preventing reliance on specific features:

$$\tilde{a}_i = a_i \times \tilde{d}_i/(1-p_d) \tag{10}$$

2.6. Weight Initialization. Proper initialization prevents vanishing or exploding gradients. Common methods include:

Random Normal: Weights are sampled from a standard normal distribution:

$$w \sim \mathcal{N}(0, \sigma^2), \tag{11}$$

where a small standard deviation σ (e.g., 0.01) is used to prevent large initial activations.

Xavier Initialization: Used with Sigmoid, Tanh, and linear activations. Designed for activation functions with zero-centered outputs, it maintains variance across layers:

$$w \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}+n_{\text{out}}}\right) \quad \text{(Normal)}, \quad w \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}}+n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}}+n_{\text{out}}}}\right) \quad \text{(Uniform)}. \tag{12}$$

Kaiming (He) Initialization: Optimized for ReLU-based activations to preserve variance:

$$w \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right) \quad \text{(Normal)}, \quad w \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}}}}\right) \quad \text{(Uniform)}. \tag{13}$$

2.7. Batch Normalization. **Batch Normalization** stabilizes training by normalizing activations within each mini-batch:

$$\hat{x} = \frac{x - \mu}{\sigma}, \quad y = \gamma \hat{x} + \beta, \tag{14}$$

where μ and σ are batch statistics, and γ, β are learnable parameters. This allows for higher learning rates and faster convergence. **Batch Size** defines the number of samples per iteration, influencing model convergence and generalization. A common batch size is:

$$J = \frac{1}{m} \sum_{i=1}^{m} L(\tilde{y}^{(i)}, y^{(i)}). \tag{15}$$

2.8. Evaluation Metrics. Model performance is evaluated using:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}. \tag{16}$$

This measures both training progress and validation/test performance.

3. Algorithm Implementation and Development

3.1. Data Processing and Model Architecture. The FashionMNIST dataset is preprocessed using **Torchvision** with normalization to improve stability. A validation set is created using **Scikit-learn**, and data is processed in mini-batches using **PyTorch's DataLoader**. The **FCN** consists of an input layer, multiple hidden layers with **ReLU activation**, and an output layer for classification. **Dropout Regularization** is applied to enhance generalization.

3.2. Training and Optimization. The model is trained using **SGD** with momentum and **Cross-Entropy Loss**. Hyperparameters such as batch size, learning rate, and number of epochs are adjusted for efficiency. The training process includes forward propagation, loss computation, and weight updates via backpropagation. Validation accuracy is tracked to assess overfitting, and performance is evaluated by analyzing training loss and accuracy trends.

3.3. Hyperparameter Tuning and Model Improvements.

3.3.1. Optimizer Comparison. **SGD**, **RMSProp**, and **Adam** are evaluated with different learning rates to identify the most effective optimization method. Each optimizer is tested on a fresh FCN model, and results are analyzed based on convergence, training loss, and validation accuracy.

3.3.2. Overfitting and Regularization. **Dropout Regularization** is applied to assess its impact on overfitting. Different dropout rates are tested to determine their effect on generalization and test accuracy, improving model robustness by reducing co-adaptation of neurons.

3.3.3. Weight Initialization Strategies. The sensitivity of the model to weight initialization is studied using **Random Normal**, **Xavier Normal**, and **Kaiming Uniform**. Each method is applied to a fresh FCN model and evaluated for training stability and final test accuracy.

3.3.4. Normalization and Training Stability. **Batch Normalization** is introduced to improve training stability and mitigate internal covariate shift. Its effect on loss convergence and generalization is compared against a baseline model without normalization.

3.3.5. Performance Evaluation and Analysis. Results from tuning experiments, including test accuracy, validation accuracy trends, and training loss curves, are analyzed to identify the best-performing optimizer, regularization method, initialization strategy, and normalization technique.

4. Computational Results

4.1. Model Performance and Parameter Selection. The **FCN model** is designed with adjustable **hidden layers and neurons**, using **ReLU activation** for efficient feature extraction. It has **784 input dimensions** and **10 output classes** for **FashionMNIST classification**. The model employs **Cross-Entropy loss** and **SGD (LR=0.01)** for optimization. **Mini-batch training** (512 samples for training, 256 for testing) ensures stable updates. Trained for **10 epochs**, it achieves a peak **validation accuracy of 81.82%** with minimal overfitting, as indicated by the **steady validation accuracy and decreasing loss**. A **final test accuracy of 80.90%** confirms strong generalization. Figure 1 (left) illustrates the loss and accuracy trends, highlighting effective learning and stability.

4.2. Optimized Configuration. The **FCN model** is optimized for FashionMNIST, balancing **accuracy and efficiency**. **Mini-batch training** (128 samples for training, 256 for testing) ensures stable updates. **SGD with momentum (LR=0.01)** enables smooth convergence. Figure 1 (right) shows a consistent **loss decrease** and **accuracy increase**, peaking at **88.83% validation accuracy at epoch 8**. The minimal gap between training and validation loss confirms **strong generalization**, with a final **test accuracy of 88.21%**, surpassing the 85% benchmark. The stability in validation accuracy suggests a well-optimized baseline for further tuning.

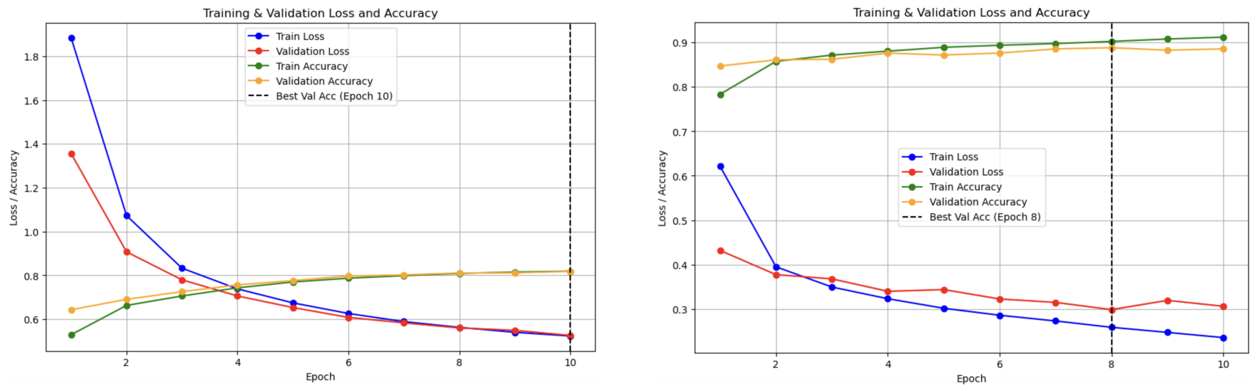

Figure 1. Training & Validation Loss and Accuracy for Baseline Models

4.3. Hyperparameter Tuning.

4.3.1. Optimizer Comparison. To evaluate optimizer performance, I tested **SGD, RMSprop, and Adam** across different learning rates, selecting the best configuration for each. Figure 2 presents validation accuracy trends for each optimizer across different learning rates, where **Adam (LR=0.001)** and **RMSprop (LR=0.001)** achieving the highest validation accuracy of **89.53%**, surpassing **SGD (LR=0.01)** at **89.33%**. Figure 3 highlights optimizer trends with best LR: **SGD** demonstrated stable improvement but slower convergence, **RMSprop** converged quickly but showed mild overfitting, and **Adam** achieved the highest and most stable validation accuracy with smooth convergence. Loss curves confirm **Adam** maintains the best balance between training and validation loss, ensuring strong generalization. The final test accuracy of **Adam (89.00%)** also outperforms **RMSprop (88.33%)** and **SGD (87.57%)**, making it the most effective optimizer with superior stability and generalization.

4.3.2. Dropout Regularization Analysis. To evaluate dropout's impact on generalization, we trained the FCN with dropout rates of **0, 0.3, and 0.5**. Table 1 shows **Dropout = 0** achieves the highest validation accuracy (**88.87%**) but overfits, as seen by the widening gap between training and validation accuracy after epoch 5. **Dropout = 0.3** improves generalization, maintaining stable

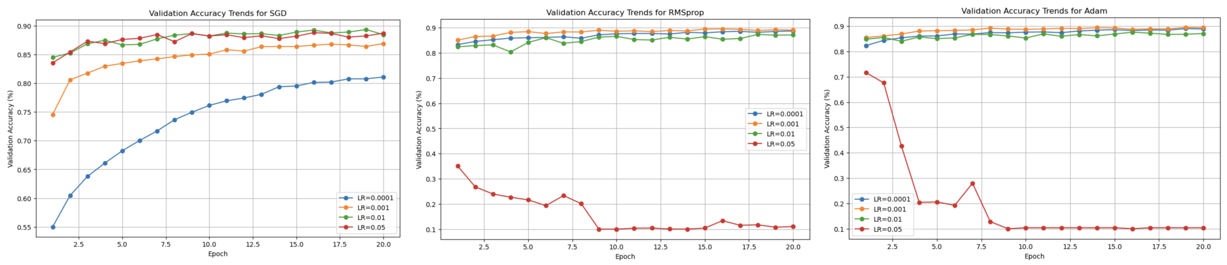

FIGURE 2. Validation Accuracy Across Different Learning Rates for Optimizers

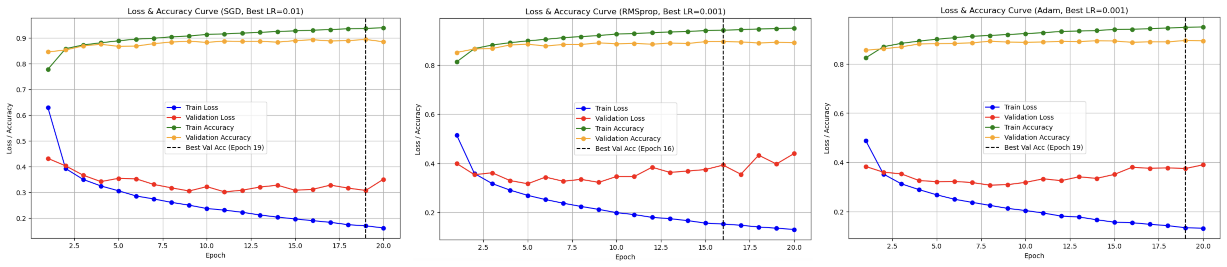

FIGURE 3. Training & Validation Loss and Accuracy with Different Optimizers

validation accuracy (**88.47%**), while **Dropout = 0.5** underfits, reaching only **87.53%** due to excessive regularization. Figure 4 confirms these trends: the **No Dropout model** overfits, the **Dropout = 0.5 model** struggles to converge, and the **Dropout = 0.3 model** finds a balance. Final test accuracies further validate this, with **Dropout = 0 (88.34%)** outperforming **Dropout = 0.3 (87.69%)** and **Dropout = 0.5 (86.92%)**, confirming that **Dropout = 0.3** is optimal for reducing overfitting while preserving accuracy.

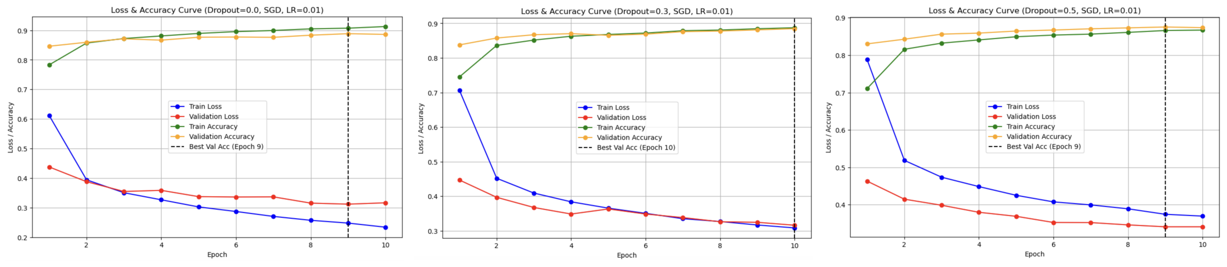

FIGURE 4. Training & Validation Loss and Accuracy with Different Dropout Rates

4.3.3. Impact of Weight Initialization. To assess weight initialization effects, we trained the FCN with **Random Normal, Xavier Normal, and Kaiming Uniform**. Figure 5 and Table 2 show **Kaiming Uniform** achieves the highest validation (**89.08%**) and test accuracy (**88.55%**), ensuring stable learning. **Xavier Normal** performs moderately (**88.50%** validation, **87.90%** test), showing slight overfitting, while **Random Normal** has the lowest accuracy (**88.90%** validation, **87.88%** test), indicating slower convergence. Weight initialization impacts gradient stability and convergence speed, with **Random Normal** causing fluctuations, **Xavier Normal** balancing variance but overfitting slightly, and **Kaiming Uniform** optimizing stability, making it the best choice.

4.3.4. Effect of Batch Normalization. To assess the impact of **Batch Normalization (BN)** on training, we trained an FCN with and without BN for **10 epochs**. Table 3 and Figure 6 highlight its effects on model performance. The FCN **without BN** achieves a peak validation accuracy of

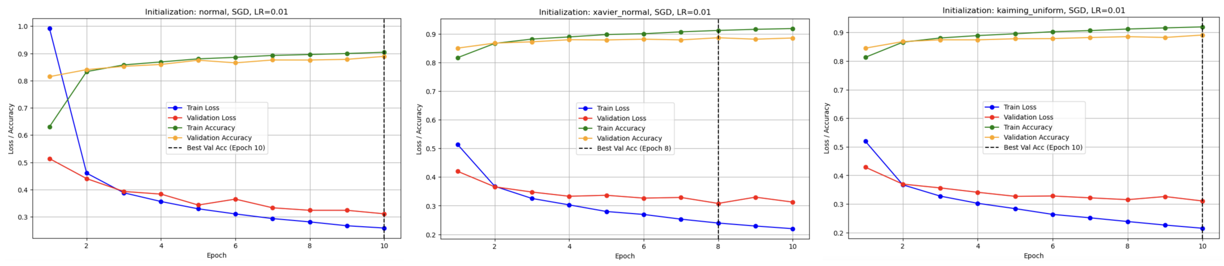

FIGURE 5. Training & Validation Loss and Accuracy with Different Initializations

Epoch	0	0.3	0.5
1	0.8462	0.8377	0.8302
2	0.8595	0.8575	0.8427
3	0.8713	0.8670	0.8562
4	0.8667	0.8702	0.8590
5	0.8767	0.8653	0.8645
6	0.8773	0.8687	0.8672
7	0.8763	0.8760	0.8705
8	0.8835	0.8777	0.8730
9	**0.8887**	0.8810	**0.8753**
10	0.8862	**0.8847**	0.8735

TABLE 1. Validation Accuracy for Dropout Rates

Epoch	Random	Xavier	Kaiming
1	0.8145	0.8497	0.8450
2	0.8402	0.8672	0.8678
3	0.8523	0.8717	0.8742
4	0.8593	0.8790	0.8740
5	0.8748	0.8782	0.8778
6	0.8652	0.8808	0.8783
7	0.8757	0.8783	0.8822
8	0.8755	0.8858	0.8852
9	0.8780	0.8808	0.8823
10	**0.8890**	**0.8850**	**0.8908**

TABLE 2. Validation Accuracy for Initializations

Epoch	No BN	With BN
1	0.8382	0.8603
2	0.8595	0.8677
3	0.8708	0.8635
4	0.8712	0.8758
5	0.8708	0.8855
6	0.8842	0.8888
7	0.8800	0.8855
8	**0.8873**	0.8867
9	0.8865	**0.8893**
10	0.8848	0.8867

TABLE 3. Validation Accuracy for Batch Normalization

88.73%, while the **BN model slightly improves to 88.93%**. Test accuracy is also marginally higher with BN (**87.75%** vs. 87.48%), indicating minimal generalization benefits. Figure 6 shows that **without BN**, loss and accuracy curves converge smoothly but plateau early, suggesting slight overfitting. **With BN**, training loss decreases faster, stabilizing learning, though validation accuracy fluctuates slightly, reflecting sensitivity to batch statistics. While BN accelerates convergence and reduces internal covariate shift, its impact on final test accuracy remains limited in this setting. Overall, BN enhances training stability but does not significantly improve generalization in FCN.

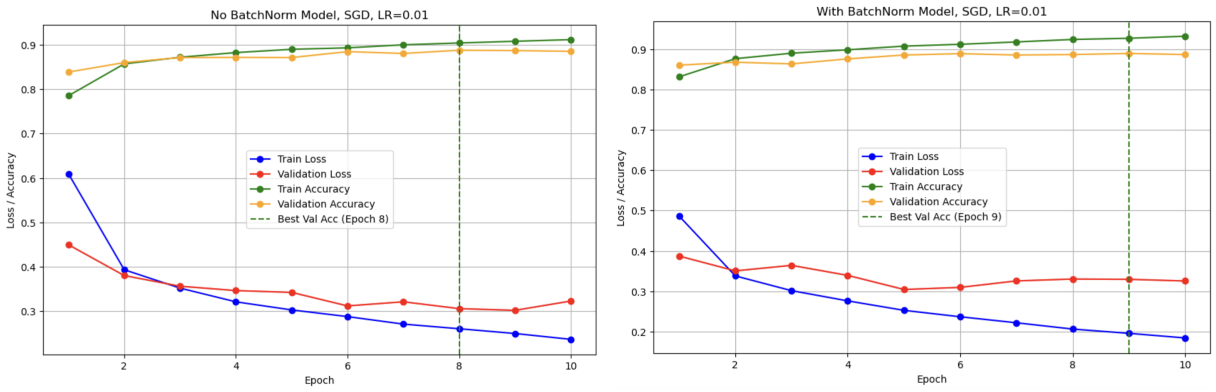

FIGURE 6. Training Loss and Validation Accuracy for FCN with/without BN

4.4. Hyperparameter Tuning for FashionMNIST and MNIST. To achieve high test accuracy on **FashionMNIST** and **MNIST**, we optimized an **Extended FCN** with three hidden layers (**1024, 512, 256 neurons**) and **dropout (0.3)** to prevent overfitting. **Xavier Normal** initialization ensured stable training, while **Batch Normalization** improved convergence. We used the **Adam optimizer (LR=0.001)** with **30 epochs** and a learning rate scheduler to refine performance. As a baseline comparison, we applied a **K-Nearest Neighbors (KNN)** classifier with $k = 3$ to FashionMNIST. The **Extended FCN** achieved a test accuracy of **90.21% on FashionMNIST** and **98.86% on MNIST**, surpassing the target thresholds of **90% and 98%**, respectively. In contrast, the **KNN classifier** reached **85.41%**, significantly lower than the FCN due to its non-parametric nature and lack of learned feature representations. These results confirm that deep networks with proper initialization, normalization, and adaptive learning significantly outperform traditional classifiers for image classification tasks.

5. Summary and Conclusions

This report explores optimizing a Fully Connected Neural Network (FCN) for FashionMNIST classification through hyperparameter tuning, regularization, weight initialization, and normalization. Adam emerged as the most effective optimizer, offering a balance between stability and convergence. Dropout mitigated overfitting by enhancing generalization, while Kaiming Uniform initialization provided the best training stability. Batch Normalization further improved training efficiency but had a limited impact on test accuracy. An extended FCN with optimized hyperparameters achieved strong classification performance, surpassing traditional methods like KNN, demonstrating the advantages of deep learning with proper tuning.

Acknowledgements

This author thanks Professor Frank for providing useful information to tackle the problem through the hints in the assignment, as well as the HW4-Helper.ipynb code. The author is also grateful to TA Rohin Gilman for providing theoretical background to assist in constructing my code. Finally, my peers Liam Lin and Zhaohan Yin were also helpful when I ran into issues related to coding.

References

[1] J.N. Kutz, *Methods for Integrating Dynamics of Complex Systems and Big Data*, Oxford, 2013.
[2] N. Frank, *Lecture Notes*, March 2025.