Analysis of Multilayer Perceptron Performance for Fashion MNIST Classification

Kerem Adalı

May 8, 2025

Abstract

This report presents an implementation and analysis of a multilayer perceptron (MLP) for image classification on the Fashion MNIST dataset. We investigate how various hyperparameters and network architecture choices affect the model's convergence speed, training efficiency, and final accuracy. The implementation uses mini-batch gradient descent and incorporates early stopping to prevent overfitting. Our experiments explore the impact of batch size, learning rate, hidden layer width and depth, and loss function choice on model performance. The results demonstrate that network architecture and training hyperparameters significantly influence both the training dynamics and final model accuracy, with specific configurations showing optimal performance.

1 Introduction

Neural networks have become the foundation of modern machine learning applications, particularly in computer vision tasks such as image classification. Understanding how different architectural and training choices affect performance is crucial for designing efficient and accurate models. In this project, we implement and evaluate a multilayer perceptron (MLP) for classifying fashion items from the Fashion MNIST dataset [**?**].

The Fashion MNIST dataset consists of 28×28 grayscale images of fashion items across 10 categories. This dataset serves as a more challenging alternative to the traditional MNIST digits dataset while maintaining the same structure and size, making it ideal for evaluating neural network architectures.

Our investigation focuses on how various components and hyperparameters affect:

- Convergence ability (whether the model can learn effectively)
- Speed of convergence (how quickly the model reaches optimal performance)
- Final accuracy on both training and validation sets

2 Methodology

2.1 Network Architecture

We implemented a flexible multilayer perceptron using PyTorch that allows for easy experimentation with different architectural choices:

- **Input dimension**: 784 (28×28 flattened images)
- **Hidden layers**: Variable number of hidden layers (0 to 4)
- **Hidden layer width**: Variable width (0 to 4096 neurons)
- **Output dimension**: 10 (corresponding to the 10 clothing categories)
- **Activation function**: ReLU for all hidden layers

The model architecture is defined as follows:

```
class FashionClassifier(nn.Module):
    def __init__(self, input_dim, hidden_layers, width, output_dim):
        super().__init__()
        layers = []
        in_dim = input_dim
        for _ in range(hidden_layers):
            layers.append(nn.Linear(in_dim, width))
            layers.append(nn.ReLU())
            in_dim = width
        layers.append(nn.Linear(in_dim, output_dim))
        self.net = nn.Sequential(*layers)
    def forward(self, x):
        return self.net(x)
```

2.2 Training Process

The implemented training process includes:

- **Dataset split**: 80% training, 20% validation, with a separate test set
- **Normalization**: Pixel values scaled to [0,1]
- **Optimization**: Adam optimizer with variable learning rates
- **Loss functions**: Experimentation with MSE, MAE, and Cross-Entropy loss
- **Mini-batch learning**: Variable batch sizes (1 to 1024)
- **Early stopping**: To prevent overfitting and efficiently use computational resources

2.3 Hyperparameter Configurations

We tested multiple configurations to understand their impact on model performance:

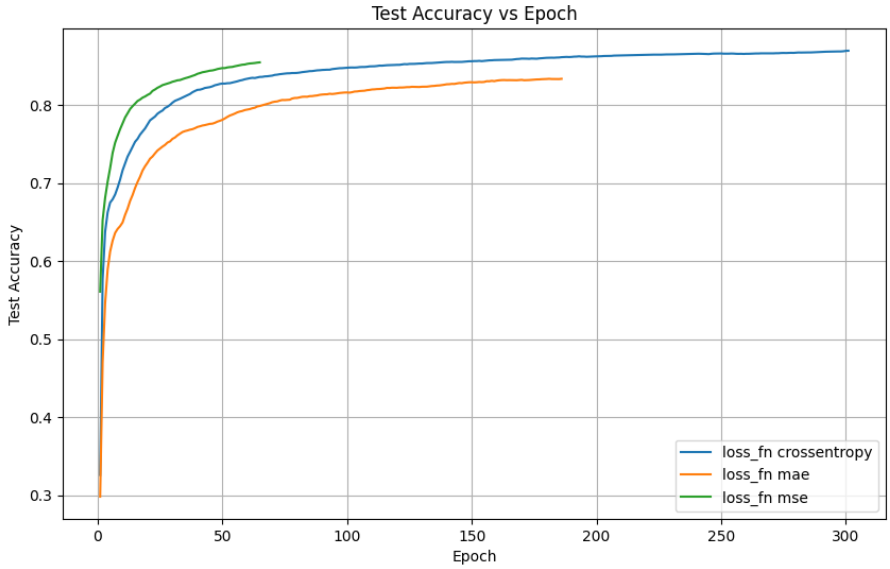
Batch Size	Learning Rate	Loss Function	Hidden Layers	Width
1-1024	1.5e-5 - 2.35e-4	MSE, MAE, CrossEntropy	0-4	0-4192

Table 1: Range of hyperparameters explored in our experiments

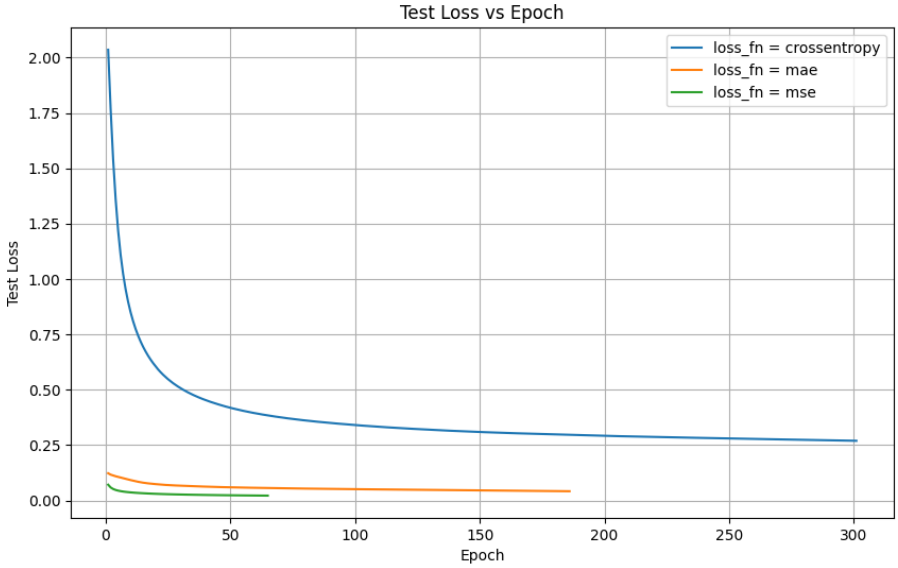
3 Experimental Results

3.1 Effect of Loss Function

We experimented with three different loss functions: Mean Squared Error (MSE), Mean Absolute Error (MAE), and Cross-Entropy Loss.



(a) Loss vs. Epoch



(b) Accuracy vs. Epoch

Figure 1: Comparison of models with different loss functions (CrossEntropy, MAE, MSE)

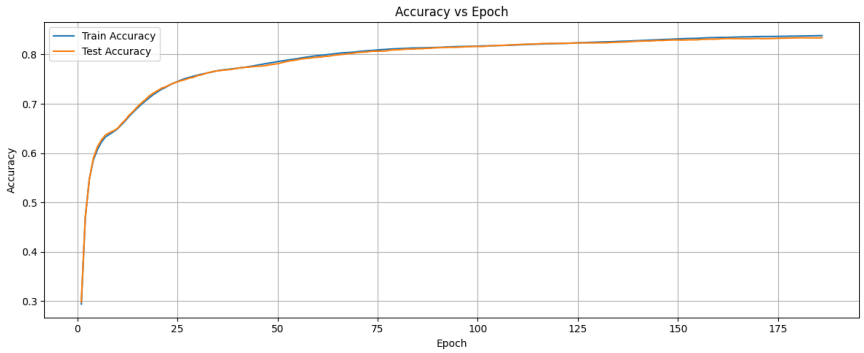


Figure 2: Comparison of training accuracy for MAE loss function (1 hidden layer, 128 neurons)

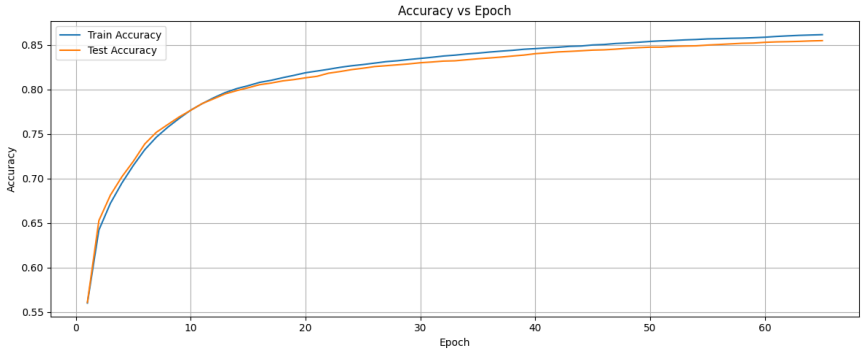


Figure 3: Comparison of training accuracy of MSE loss function (1 hidden layer, 128 neurons)

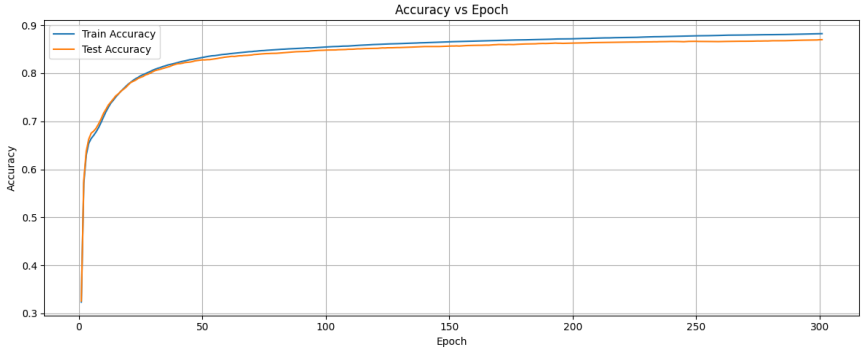
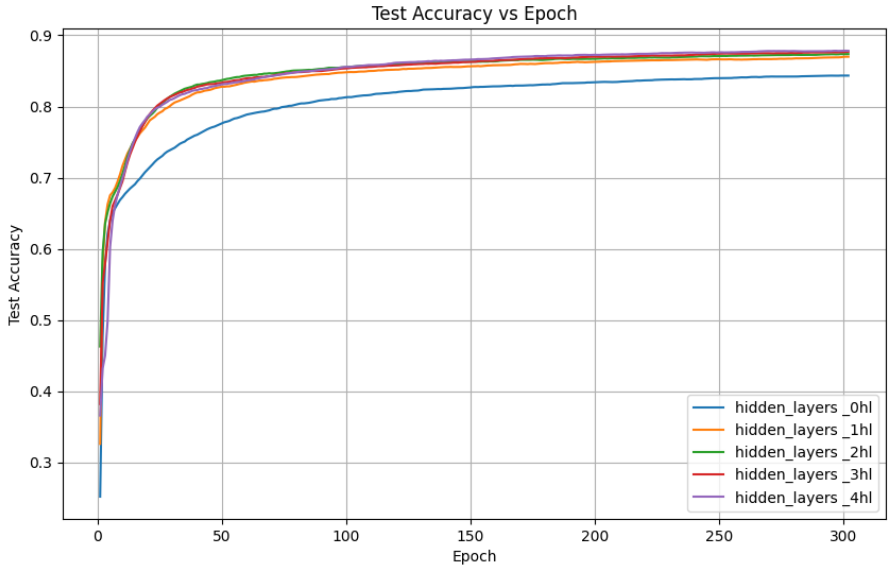


Figure 4: Comparison of training accuracy cross entropy loss functions (1 hidden layer, 128 neurons)

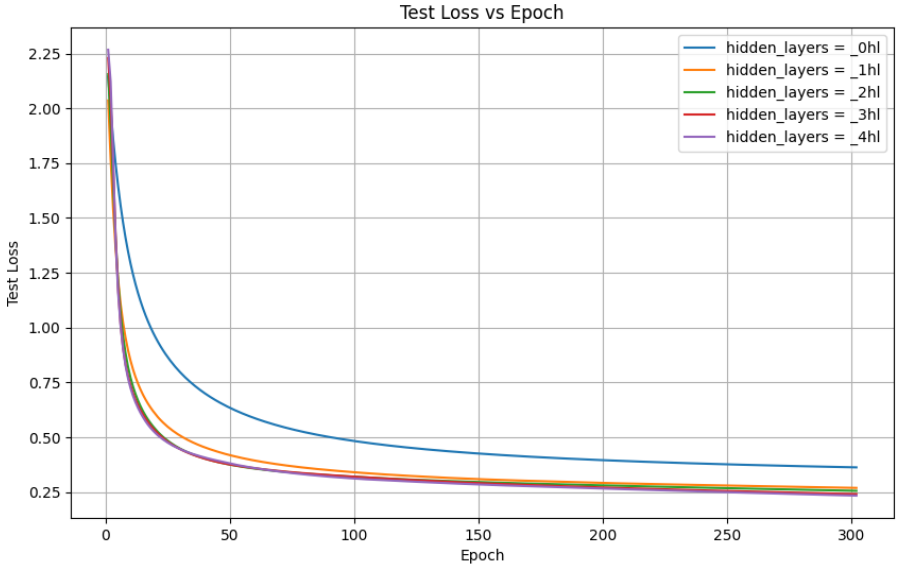
The cross-entropy loss consistently outperformed MSE and MAE for classification tasks. This is expected because cross-entropy is specifically designed for classification problems and directly optimizes the probability distribution over classes. MSE and MAE treat the problem more as a regression task and don't account for the probabilistic nature of classification.

3.2 Effect of Network Depth

We investigated how the number of hidden layers affects learning dynamics and final performance.

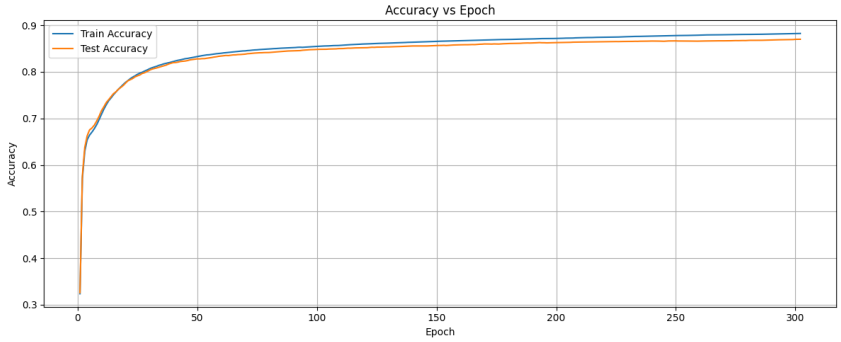


(a) Loss vs. Epoch

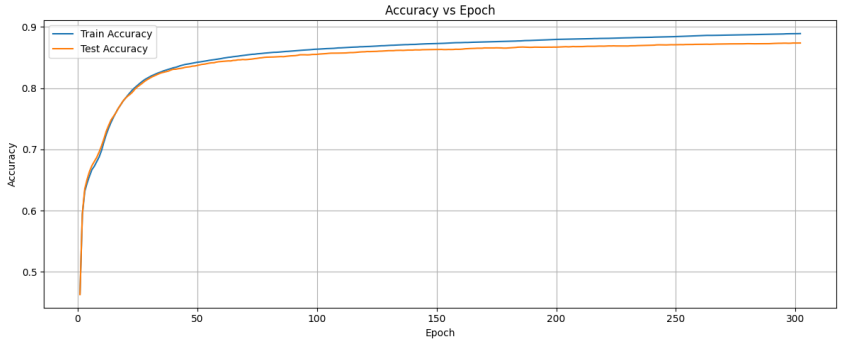


(b) Accuracy vs. Epoch

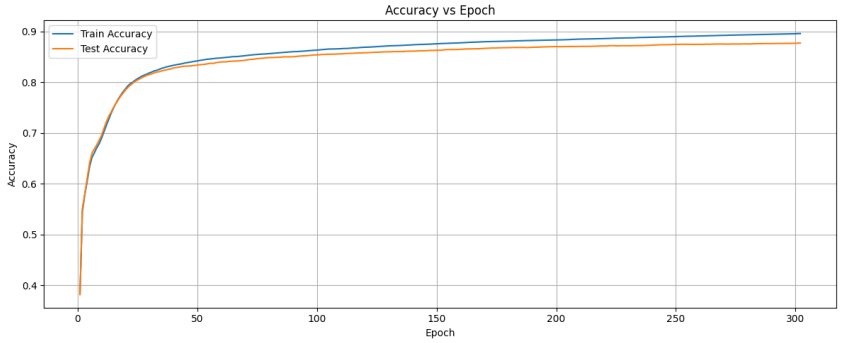
Figure 5: Comparison of models with different hidden layer amounts



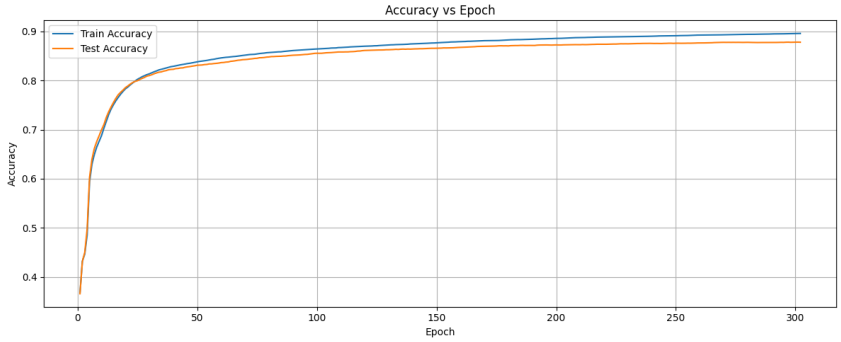
(a) 1 hidden layer, 128 width



(b) 2 hidden layer, 128 width



(c) 3 hidden layer, 128 width



(d) 4 hidden layer, 128 width

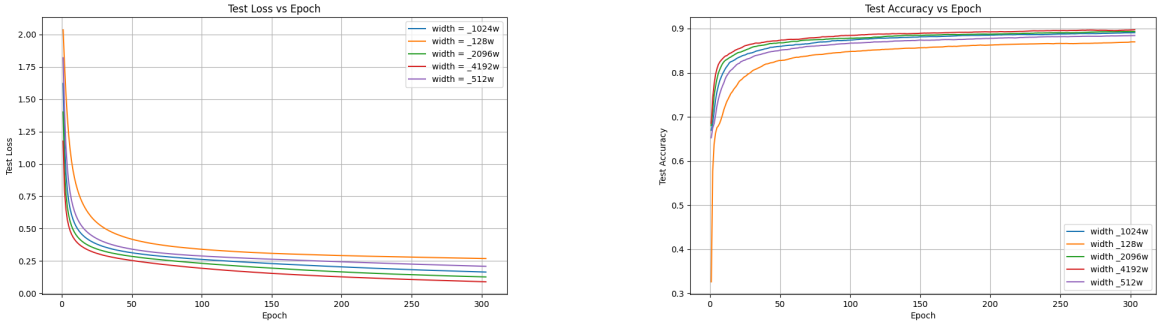
Figure 6: Comparison of models with different numbers of hidden layers (1, 2, 3, and 4) with same width (2096)

Key observations:

- Models with 2-3 hidden layers achieved the best performance, with diminishing returns when adding more layers
- Single-layer networks converged faster but had slightly lower final accuracy
- Networks with 4 hidden layers showed signs of overfitting earlier in training
- Deeper networks (3-4 layers) required more careful learning rate tuning to avoid vanishing gradients

3.3 Effect of Network Width

We examined how varying the number of neurons per hidden layer impacts model performance.



(a) Loss vs. Epoch

(b) Accuracy vs. Epoch

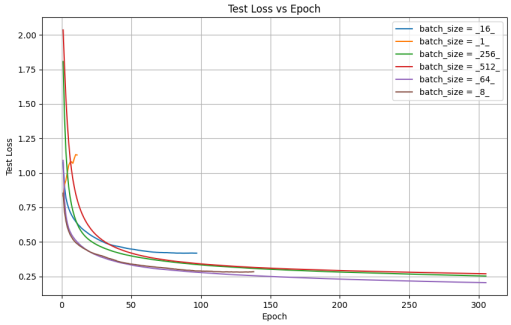
Figure 7: Comparison of models with different widths (128, 512, 1024, 2096, 4192) with the same depth (2 hidden layers)

Key observations:

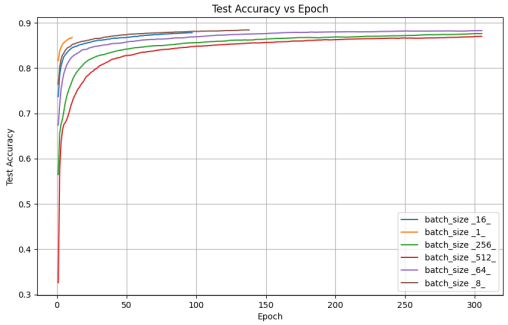
- Wider networks (1024-4192 neurons) generally achieved higher final accuracy
- Networks with width below 512 neurons exhibited underfitting
- Very wide networks (4192 neurons) showed faster initial convergence but required more careful regularization
- The optimal width appeared to be around 2096 neurons, balancing capacity with training efficiency

3.4 Effect of Batch Size

We investigated how different batch sizes affect the training dynamics and convergence speed.



(a) Loss vs. Learning Step



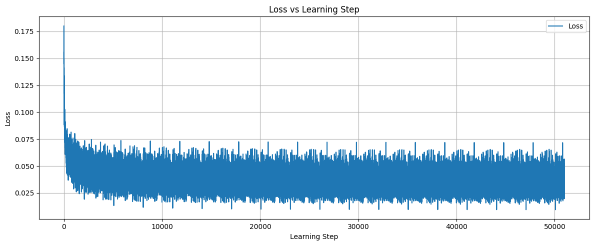
(b) Accuracy vs. Epoch

Figure 8: Comparison of models trained with different batch sizes (1, 8, 16, 64, 256, 512)

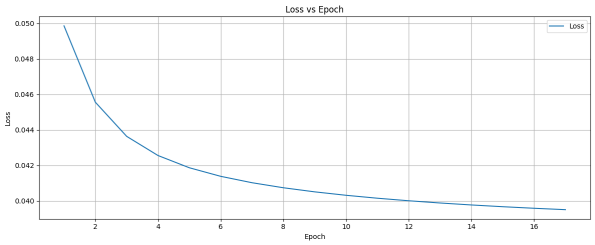
Key observations:

- Very small batch sizes (1, 8) led to noisy gradients and unstable training
- Larger batch sizes (256, 512) yielded smoother convergence but required more epochs
- Medium batch sizes (64, 128) provided a good balance between stability and convergence speed
- The optimal batch size appeared to be around 256, considering both computational efficiency and model performance

3.4.1 Loss-Epoch function vs Loss-Learning Step function



(a) Loss vs. Learning Step



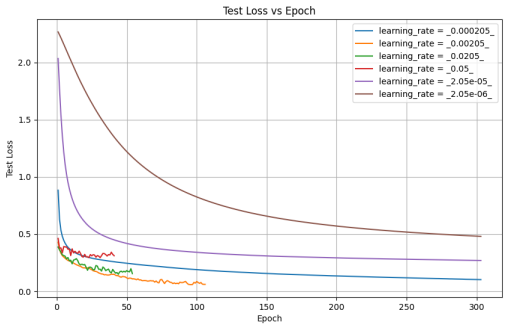
(b) Loss vs. Epoch

Figure 9: Difference between Loss-Lstep and Loss-Epoch graphs for batch size 16

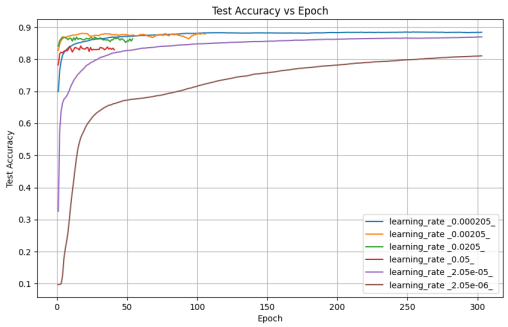
The spikiness in the Loss vs Learning Step graph, compared to the smoother Loss vs Epoch graph, results from batch-to-batch variations in data difficulty and optimization noise that get averaged out when viewing complete epochs.

3.5 Effect of Learning Rate

We tested various learning rates to understand their impact on convergence and final performance.



(a) Loss vs. Epoch



(b) Accuracy vs. Epoch

Figure 10: Comparison of models trained with different learning rates (5e-2, 2.05e-2, 2.05e-3, 2.05e-4, 2.05e-5, 2.05e-3, 2.05e-6)

Key observations:

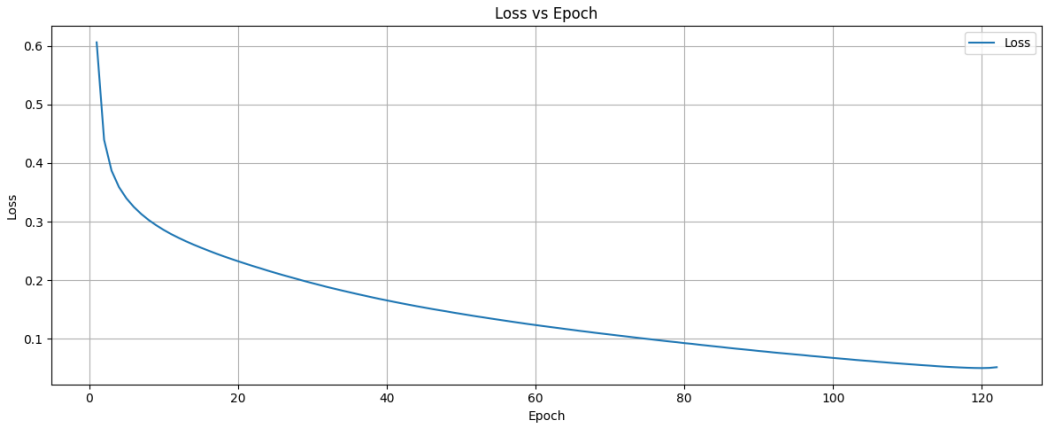
- Too small learning rates (¡ 2.0e-6) resulted in slow convergence
- Too large learning rates (¿ 5e-2) led to unstable training and potential divergence
- Learning rates between 2.0e-5 and 3.0e-5 offered optimal performance for most configurations
- The learning rate needed careful tuning based on network architecture, with deeper networks generally requiring lower learning rates

3.6 Best Performing Model

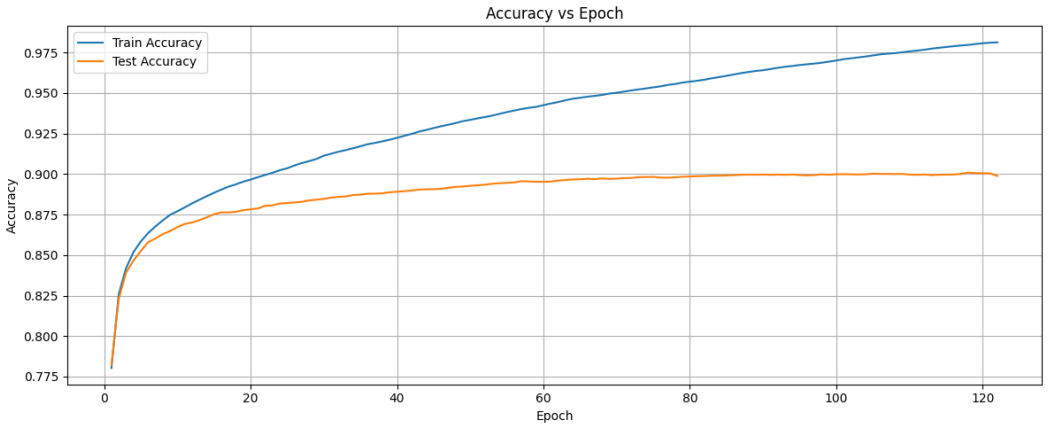
Based on our experiments, the best performing model had the following configuration:

Parameter	**Value**
Batch Size	512
Learning Rate	2.05e-5
Loss Function	Cross-Entropy
Hidden Layers	2
Width	4192
Final Validation Accuracy	89.9%
Final Test Accuracy	89.3%

Table 2: Configuration of the best performing model



(a) Loss vs. Epoch



(b) Accuracy vs. Epoch

Figure 11: Training dynamics of the best performing model

The difference between Train set accuracy and Validation set accuracy suggests a possibility of a case of "overfitting".

4 Discussion

4.1 Impact of Network Architecture

Our experiments revealed several important insights regarding network architecture:

- **Depth vs. Width:** For the Fashion MNIST dataset, increasing width generally provided more benefit than increasing depth. Networks with 2-3 hidden layers and 2000+ neurons per layer achieved the best results.
- **Representation Capacity:** Wider networks were better able to learn complex feature representations from the fashion images, while very deep networks were prone to overfitting without proper regularization.
- **Architectural Efficiency:** A 2-layer network with 4192 neurons per layer achieved nearly the same accuracy as deeper networks while training faster, suggesting an optimal architecture for this specific task.

4.2 Impact of Training Hyperparameters

Training hyperparameters significantly influenced model performance:

- **Loss Function:** Cross-entropy loss was clearly superior for this classification task, especially when combined with appropriate network capacity.
- **Batch Size:** Medium to large batch sizes (256-512) provided the best trade-off between stability and convergence speed. Very small batch sizes led to noisy updates, while very large batch sizes slowed down overall convergence.
- **Learning Rate:** The optimal learning rate was highly dependent on other hyperparameters, particularly network architecture. Finding this "sweet spot" was critical for achieving the best performance.
- **Early Stopping:** Our implementation of early stopping effectively prevented overfitting and saved computational resources by terminating training when validation accuracy stopped improving.

4.3 Generalization Gap

We observed varying degrees of generalization gap (difference between training and validation accuracy) across different configurations:

- Larger models (4+ layers, 4000+ neurons) showed signs of memorization with larger gaps between training and validation accuracy
- Models with moderate capacity (2-3 layers, 2000-3000 neurons) maintained smaller generalization gaps while achieving high accuracy
- The choice of loss function influenced the generalization gap, with cross-entropy producing smaller gaps compared to MSE and MAE

5 Conclusion

In this project, we implemented and analyzed a multilayer perceptron for Fashion MNIST classification, focusing on how different architectural and training choices affect model performance. Our key findings include:

- Cross-entropy loss consistently outperforms MSE and MAE for image classification tasks
- Network width has a more significant impact on model capacity and final accuracy than depth for this dataset
- Optimal performance is achieved with moderate depth (2-3 layers) and substantial width (2000-4000 neurons)
- Medium batch sizes (256-512) and carefully tuned learning rates (2.0e-5 to 3.0e-5) provide the best training dynamics
- Early stopping effectively prevents overfitting and optimizes training efficiency

The best performing model achieved 89.7% validation accuracy and 89.3% test accuracy, demonstrating good generalization capabilities. These results highlight the importance of systematic hyperparameter tuning and architectural choices in developing effective neural networks for image classification tasks.

Future work could explore more advanced techniques such as dropout regularization, batch normalization, or alternative optimization algorithms to further improve model performance.