Report Title:	**Fashion-MNIST and MLP**
Author Name:	**王小慧**
Student ID:	**21307110415**
Institute:	**信息科学与工程学院**
Report Date:	**2024/11/11**

Contents

1 Introduction

In this experiment, we explore the use of Multi-Layer Perceptron (MLP) for improving classification performance on Fashion-MNIST dataset, and we also leverage advanced techniques such as Dropout and Learning Rate Decay to further boost accuracy and generalization.

1.1 Fashion-MNIST

The Fashion-MNIST dataset, a collection of 28x28 grayscale images representing 10 clothing categories, provides a good benchmark for evaluating image classification models.

1.2 Multi-Layer Perceptron (MLP)

MLP is one of the simplest forms of neural networks, consisting of input, hidden, and output layers as Figure 1. Each neuron in a layer is connected to every neuron in the previous and subsequent layers, forming a fully connected network. The output of each neuron is computed using an activation function, such as the ReLU, which introduces non-linearity into the network.

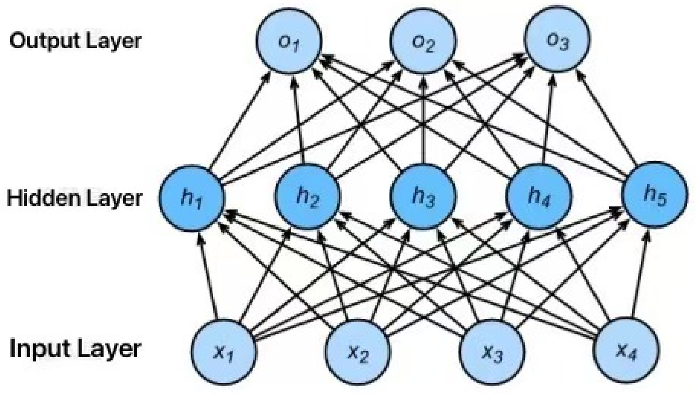


Figure 1 Multi-Layer Perceptron

The output y of a single-layer perceptron can be expressed as:

$$y = f(wx + b) \tag{1}$$

where w is the weight matrix, x is the input vector, b is the bias, and f is the activation function (e.g., ReLU or sigmoid).

An MLP learns to map input features to output labels by adjusting the weights and biases through backpropagation using an optimization technique like gradient descent.

1.3 Dropout

Dropout is a regularization technique used to prevent overfitting in neural networks. During training, Dropout randomly sets a fraction of the neurons to zero in the network at each training step. This prevents neurons from learning too much and forces the network to learn more robust features.

Mathematically, for a given layer, if x_i is the input, the output y_i is given by:

$$y_i = \begin{cases} x_i/p & \text{, if } i \text{ is kept} \\ 0 & \text{, if } i \text{ is dropped} \end{cases} \tag{2}$$

where p is the probability of a neuron being kept (usually between 0.5 and 0.8). The scaling factor $1/p$ ensures that the expected sum of activations remains the same during training and inference.

Dropout has been shown to significantly improve generalization, especially in deep networks with large numbers of parameters.

2 Experimental Setup

2.1 Environment Preparation

- **Programming Language:** Python 3.8
- **Deep Learning Framework:** torch 1.13.1
- **Other Libraries:** torchvision(dataset utilities), sklearn(confusion matrix computation), matplotlib(visualization)

2.2 Dataset Preparation

- **Source:** The dataset was downloaded using *torchvision.datasets.FashionMNIST*. The *download=True* parameter ensures automatic fetching and storage in the specified directory.
- **Classes:** T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle Boot
- **Training and Testing Split:**
  - Training Set: 60,000 images with labels.
  - Test Set: 10,000 images with labels.

3 Experimental Procedure

3.1 Data Processing

In the data processing phase, several augmentations were applied to the dataset to improve the model's generalization capabilities. Data augmentation techniques were employed to increase the diversity of the training data, including random horizontal flips and random rotations. These transformations help the model learn to be more robust to variations in the input data, thus improving its ability to generalize to unseen samples.

Here are the augmentation techniques we used in this experiment:

```
train_transform = transforms.Compose([
    transforms.RandomRotation(10),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

3.2 Parameter Configuration

Several changes were made to the original model's architecture from the slides and training parameters to improve its performance:

- **Additional Hidden Layers and Dropout Layer:** We increased the complexity of the baseline MLP model by adding two hidden layers. This allowed the model to capture more intricate patterns and improve its expressiveness. We also applied Dropout regularization to the model to prevent overfitting. This technique randomly disables a fraction of neurons during training, forcing the network to generalize better by not relying on specific neurons too heavily.

- **Learning Rate Decay:** The learning rate was adjusted during training using a StepLR scheduler. This approach gradually decreases the learning rate at specified intervals to improve convergence and avoid overshooting the optimal solution.

- **Optimizer Change:** Instead of using the Adam optimizer, we switched to Stochastic Gradient Descent (SGD) with momentum.

```
class NeuralNetwork(nn.Module):
    def __init__(self, dropout_rate=0.2):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(p=dropout_rate),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

```
optimizer = torch.optim.Adam(model.parameters(), lr=args.learning_rate)
scheduler = StepLR(optimizer, step_size=10, gamma=0.1)
```

4 Results and Analysis

4.1 Baseline

The baseline experiment used a MLP model with one input layer, one hidden layer, and one output layer. The model was trained using the default setup without data augmentation, learning rate decay, or dropout. The Adam optimizer was employed with a fixed learning rate of 0.005. This simple architecture and training configuration provided a clear reference point for evaluating the impact of various modifications on model performance.

- **Test Accuracy:** 86.7%
- **Overfitting:** The model exhibits signs of overfitting, as evidenced by the increasing test loss while the training loss continues to decrease as shown in Figure 2.
- **Observation:** Although the baseline setup converged quickly, it struggled with generalization, which was evident from Figure 3. Specifically, the model frequently confused categories such as ”shirt” and ”T-shirt,” as well as ”pullover” and ”coat,”

resulting in a higher error rate. This suggests a need for improvements in regularization, optimizer adjustments, and data augmentation to enhance the model's performance and robustness.

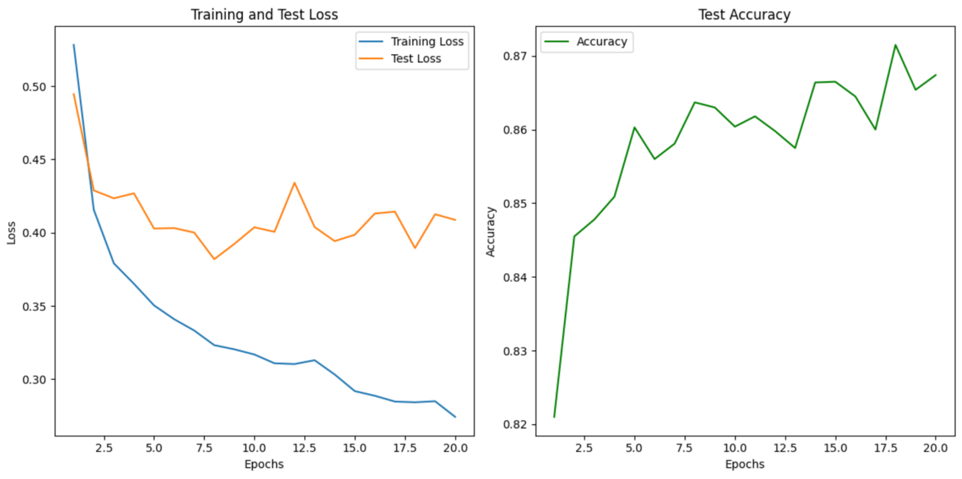
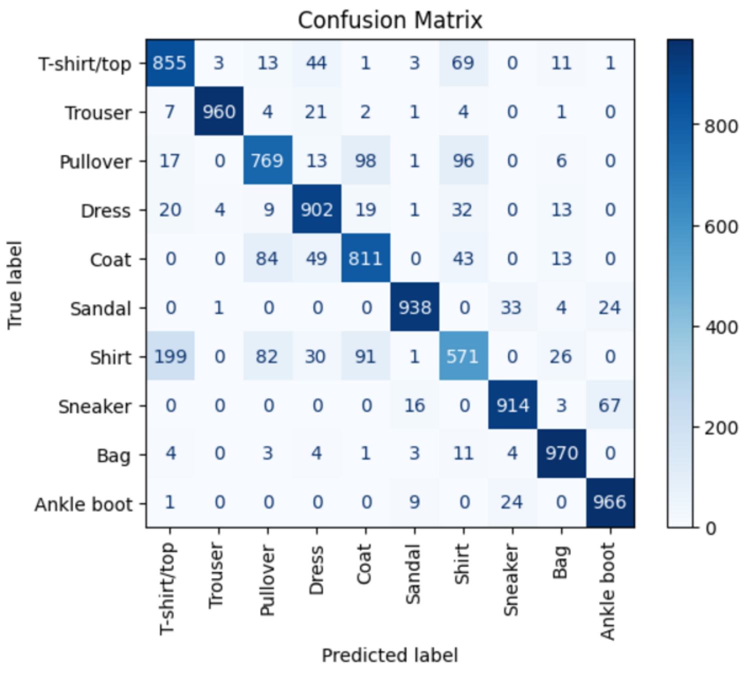


Figure 2 Training and Testing Losses in Baseline Setup



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	855	3	13	44	1	3	69	0	11	1
Trouser	7	960	4	21	2	1	4	0	1	0
Pullover	17	0	769	13	98	1	96	0	6	0
Dress	20	4	9	902	19	1	32	0	13	0
Coat	0	0	84	49	811	0	43	0	13	0
Sandal	0	1	0	0	0	938	0	33	4	24
Shirt	199	0	82	30	91	1	571	0	26	0
Sneaker	0	0	0	0	0	16	0	914	3	67
Bag	4	0	3	4	1	3	11	4	970	0
Ankle boot	1	0	0	0	0	9	0	24	0	966

Figure 3 Confusion Matrix in Baseline Setup

4.2 Impact of Additional Hidden Layers

In this experiment, we employed an MLP model consisting of an input layer, three hidden layers, and an output layer. As the model's complexity increased, we reduced the learning rate to 0.004 to mitigate overfitting. As a result, the test accuracy improved to **87.8%**. The corresponding training and testing loss curves are shown in Figure 4.

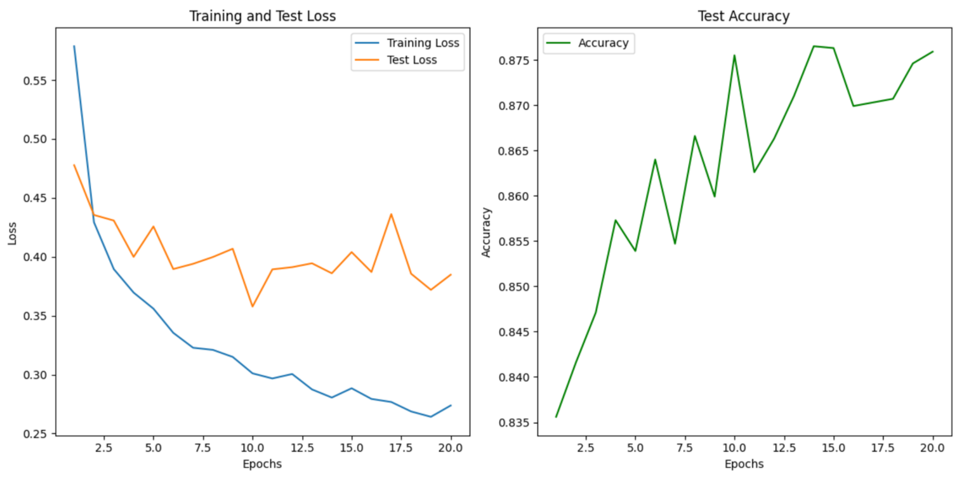


Figure 4 Training and Testing Losses Using Additional Hidden Layers

4.3 Impact of Learning Rate Decay and Dropout

In this experiment, we initially employed learning rate decay as a strategy to mitigate overfitting. While the test accuracy did increase to **89.1%**, overfitting still persisted, with a noticeable gap between the training and test losses as shown in Figure 5.

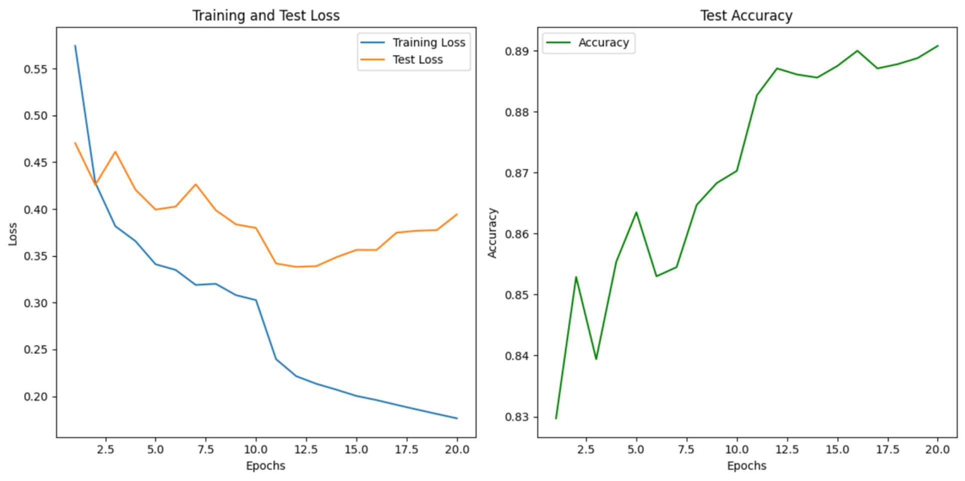


Figure 5 Training and Test Losses Using StepLR

To address this, we introduced dropout in conjunction with learning rate decay. While the combination of LR decay and dropout resulted in a slight decrease in the test accuracy which was **88.2%**, the effect on overfitting was significant. The gap between the training and test loss was significantly reduced, and the test accuracy steadily improved in the final epochs without fluctuations as shown in Figure 6.

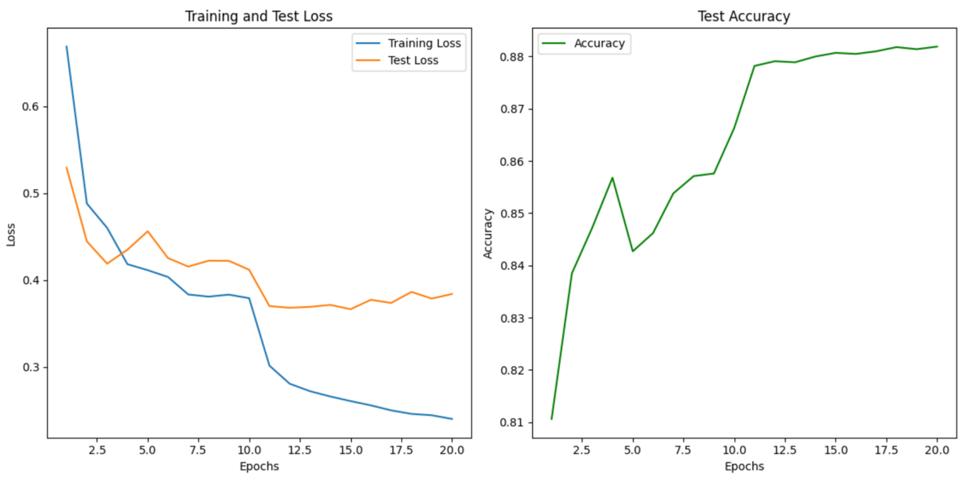


Figure 6 Training and Testing Losses Using StepLR and Dropout

4.4 Impact of SGD

When switching from Adam to SGD, a notable difference was observed in the loss curves. Specifically, the training and test loss curves became significantly smoother, as shown in Figure 7. While Adam typically exhibits more rapid fluctuations in the early stages of training, SGD produces a more gradual and stable decrease in both training and test losses.

Additionally, it was observed that SGD required a higher learning rate than Adam to achieve comparable training performance. This is a typical characteristic of SGD due to its simpler update mechanism and lack of adaptive learning rates.

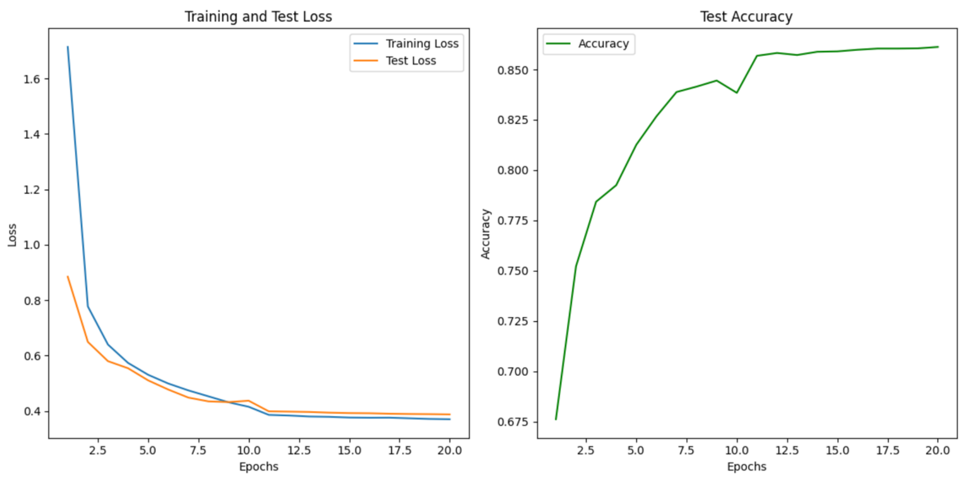


Figure 7 Training and Testing Losses Using SGD

The differences in the characteristics of Adam and SGD, such as loss curves and learning rate requirements, are due to the fundamental differences in how each optimizer adjusts the learning rate and updates the model parameters.

- **Why Adam leads to fluctuations in the loss curve:** Adam uses an adaptive learning rate for each parameter by combining both the momentum (exponentially decaying average of past gradients) and scaling by the variance of past gradients (adaptive adjustment). This strategy allows the optimizer to take large steps when the gradients are small and smaller steps when the gradients are large. This dynamic adjustment can lead to quicker, more aggressive updates, especially in the early stages of training. As a result, it may cause oscillations or fluctuations in the loss curve.

- **Why SGD leads to smoother loss curves:** SGD, on the other hand, uses a fixed learning rate for all parameters and updates them based on the gradient of the loss function. Since SGD uses a fixed learning rate and updates all parameters uniformly, it tends to make more uniform, gradual progress during training.

- **Why Adam requires a lower learning rate:** The adaptive nature of Adam reduces the need for large learning rates since it adjusts per parameter. It essentially adjusts the learning rate for each parameter during training, allowing faster convergence.

- **Why SGD needs a higher learning rate:** Without the variance scaling that Adam provides, SGD requires a higher learning rate to push the parameters sufficiently toward the optimal solution. If the learning rate is too low, SGD may converge very slowly or fail to escape local minima, whereas Adam can get away with smaller learning rates because of its adaptive mechanism.

4.5 Impact of Data Augmentation

To evaluate the impact of data augmentation, we conducted experiments using both MLP and CNN models, comparing their performance with and without augmentation techniques such as random rotations and horizontal flips. Interestingly, we observed that both models have shown a slightly performance drop with augmentation when using the same parameter settings, as shown in Table 1. This phenomenon can be attributed to two main factors:

- **The Need for Adjusted Model Settings:** Data augmentation introduces variability into the training data, which is often necessary to increase the learning rate or adjust the model complexity to fully utilize the benefits of augmentation. Without these adjustments, the additional variability may slow down convergence and degrade performance under fixed parameter settings.

Table 1 The Impact of Data Augmentation

Model	Data Augmentation	Test Accuracy
MLP	without	88.4
MLP	with	87.6
CNN	without	93.2
CNN	with	92.8

- **The Amount of the Fashion-MNIST Dataset:** Fashion-MNIST already contains 60,000 training images, which provides a sufficiently large dataset for training. In such cases, adding more data through augmentation may not significantly enhance the learning process, as the models already have ample data to generalize effectively.

- **The Nature of the Fashion-MNIST Dataset:** Fashion-MNIST is a grayscale dataset that includes categories with subtle differences, such as shirt vs. t-shirt or pullover vs. coat, requiring fine-grained classification. While augmentation techniques like rotations and flips are effective for RGB datasets by introducing diverse variations, their impact on grayscale images is more limited. In the case of Fashion-MNIST, these global transformations may introduce noise or distortions that obscure critical localized features, potentially leading to a minor drop in performance when augmentation is applied.

In summary, while data augmentation is a powerful tool to improve generalization, its effectiveness depends on proper tuning of the model's parameters and alignment with the characteristics of the dataset.

5 Conclusion

In this study, we evaluated the performance of a Multi-Layer Perceptron model on the Fashion-MNIST dataset, experimenting with different configurations such as varying model complexity, learning rates, data augmentation techniques, and optimization algorithms. The MLP model showed promising results for the Fashion-MNIST dataset, but there are inherent limitations when it comes to handling more complex transformations like data augmentation. Future work could involve exploring alternative architectures, such as CNN, which are better suited for image classification tasks that involve spatial transformations.