Laboratory 5

Variant 4 – Fashion MNIST dataset

Lab group 105

Group 9 – Aditya Kandula, Martyna Wielgopolan

27.04.2025

I. Introduction

This project implements a Multilayer Perceptron (MLP) neural network for image classification on the FashionMNIST dataset.
The solution uses manual training with mini-batch gradient descent and enables controlled experimentation with different hyperparameters such as learning rate, batch size, number of hidden layers, width of hidden layers, and loss functions.
The aim of the project is to evaluate how various hyperparameters and components of the neural network affect convergence, training speed, and final accuracy on both the training and validation datasets.
Results are saved as text files and visualized through loss and accuracy plots after each experiment.

How run the experiments

1. Install the required libraries:

```
pip install -r requirements.txt
```

2. Run the main program:

```
python main.py
```

3. Choose an experiment from the menu by entering the corresponding number:
    - 1 – Test different learning rates
    - 2 – Test different mini-batch sizes
    - 3 – Test different numbers of hidden layers
    - 4 – Test different widths of hidden layers
    - 5 – Test different loss functions
4. Wait for the training and validation to complete – it can take a couple of minutes for each test.
5. After the experiment finishes, review the generated outputs:
    - Text files containing experiment summaries are saved in the results/ folder.
    - Training and validation loss and accuracy plots are saved in the plots/ folder.
6. Repeat the process for other experiments if needed.

II. Implementation

Project Structure

The project is organized into the following files and directories:

- **main.py**
  Displays a menu and allows the user to select and run different experiments.
- **model.py**
  Contains the implementation of the Multilayer Perceptron (MLP) class.
- **train_utils.py**
  Contains helper functions for training, validation, and saving experiment results.
- **test_learning_rates.py**
  Runs experiments to evaluate the impact of different learning rates.
- **test_batch_sizes.py**
  Runs experiments to evaluate the impact of different mini-batch sizes.
- **test_hidden_layers.py**
  Runs experiments to evaluate the impact of different numbers of hidden layers, including zero hidden layers (linear model).
- **test_widths.py**
  Runs experiments to evaluate the impact of the width (number of neurons) of hidden layers.
- **test_loss_functions.py**
  Runs experiments to evaluate the impact of different loss functions
- **plots/**
  A directory where the generated training and validation plots (loss and accuracy) are saved.
- **results/**
  A directory where the textual summaries of each experiment are saved.

III. Discussion

Influence of Learning Rate

The experiments clearly show that the learning rate has a significant impact on the learning speed and final accuracy.

- A **small learning rate** (0.001) led to slow convergence and relatively poor final accuracy.
- A **medium learning rate** (0.01) allowed the model to converge faster and achieve decent accuracy.
- The **highest learning rate** (0.1) achieved the best results, reaching 89.1% train accuracy and 87.6% validation accuracy.
  Thus, a higher learning rate within a reasonable range improves both the training speed and final performance.

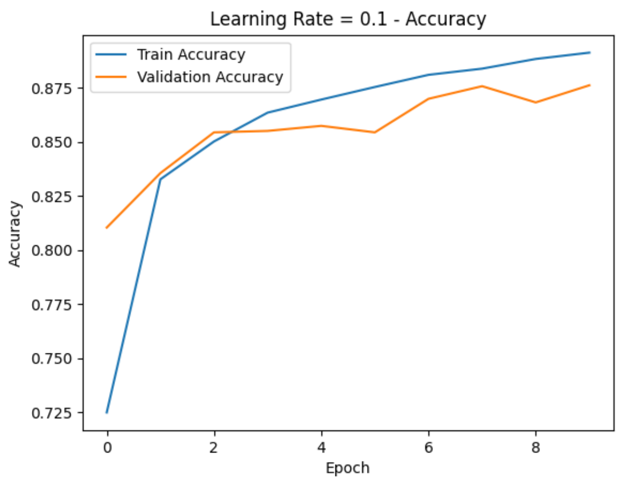

Figure 1 Training and validation accuracy for learning rate = 0.1 (best performing setting among tested learning rates).

Influence of Batch Size

Batch size also played a crucial role:

- **Batch size 1** achieved the best validation accuracy (~83.7%) but at the cost of a much longer training time.
- **Batch size 32** achieved slightly lower accuracy (~76.2%) but with a better balance between speed and generalization.
- **Batch size 128** performed the worst, indicating that large batches can harm generalization.

Thus, smaller batch sizes lead to better results at the cost of computation time.

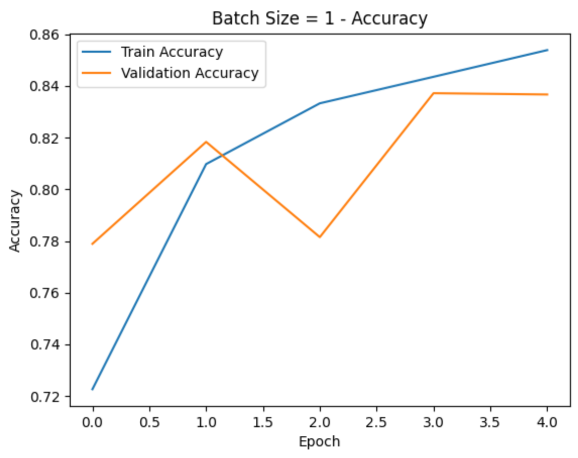

Figure 2 Training and validation accuracy for batch size = 1 (best performing among tested batch sizes).

Influence of Network Depth (Hidden Layers)

Surprisingly, the linear model (no hidden layers) achieved the best validation accuracy (~78.1%).

- Adding one hidden layer [128] or two layers [128, 64] did not improve the result and slightly worsened performance.
- This indicates that for simple datasets like FashionMNIST, deeper models are not necessarily beneficial.

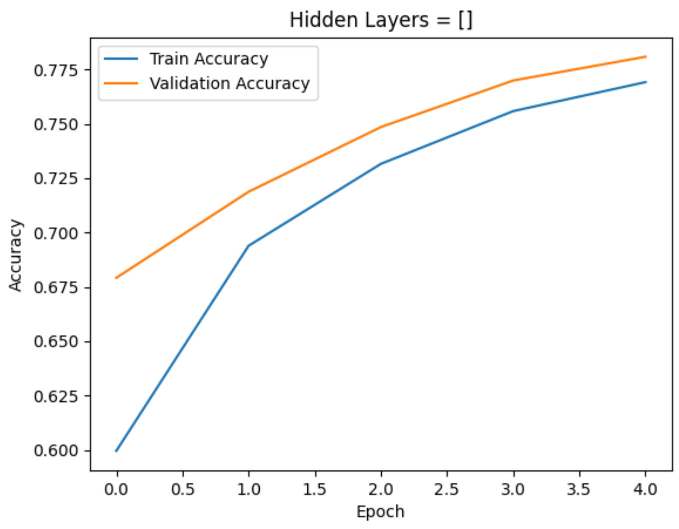

Figure 3 Training and validation accuracy for a linear model (best performing among tested hidden layer configurations).

Influence of Width (Number of Neurons)

Increasing the width of the layers slightly improved performance:

- Width 32 → validation accuracy 68.4%
- Width 128 → validation accuracy 68.9%
- Width 256 → validation accuracy 70.6%

However, the improvements were relatively minor, suggesting that increasing width only modestly helps on simple tasks.

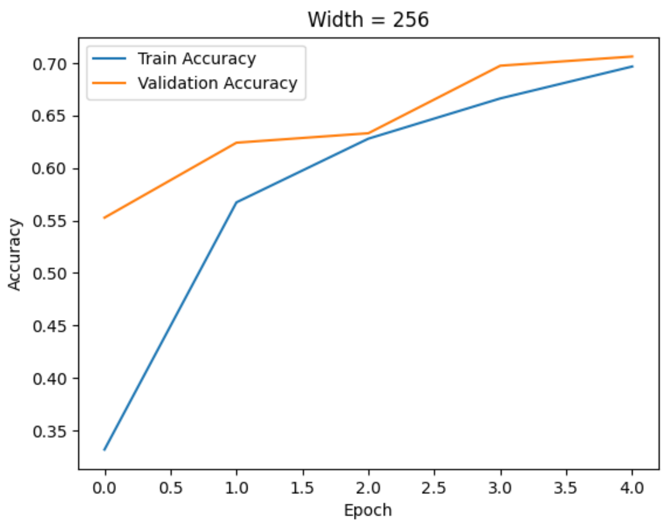

Figure 4 Training and validation accuracy for width = 256 (best performing among tested widths).

Influence of Loss Functions

As expected:

- **CrossEntropyLoss** achieved the best performance for classification (67.8% validation accuracy).
- **MSELoss** was less suitable for classification (63.5%).
- **L1Loss** performed very poorly (41.3%).

CrossEntropyLoss remains the recommended loss function for classification tasks like FashionMNIST.

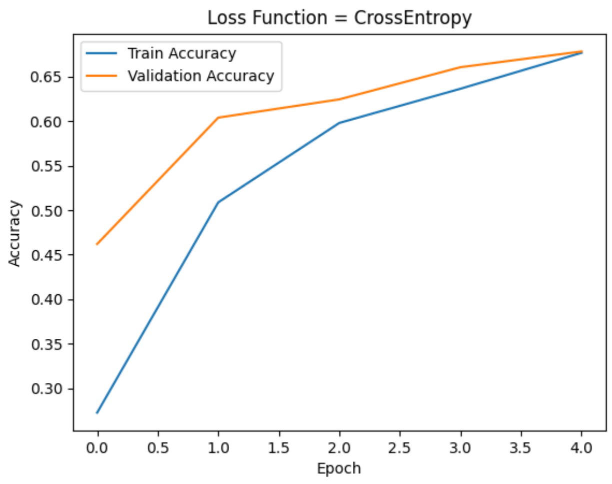

Figure 5 Training and validation accuracy using CrossEntropy loss (best performing among tested loss functions).

IV. Conclusions

The conducted experiments allow us to conclude the following:

- Proper tuning of hyperparameters (especially learning rate and batch size) has a **strong impact** on final model performance.
- **Higher learning rates** (0.1) and **smaller batch sizes** (1) yield the best results but must be used carefully to avoid instability.
- For **simple classification tasks**, a **shallow and narrow network** is sufficient.
- **CrossEntropyLoss** is crucial for classification tasks, outperforming regression-based losses like MSE and L1.
  These findings align with theoretical expectations and practical deep learning practices.
  Proper hyperparameter selection allows even simple MLP models to perform well on datasets like FashionMNIST.