Neural Networks – Multilayer Perceptron

Sivuyise Matwa
Department of Computer Science
sivuyisematwa15@gmail.com

Table of Contents

1. INTRODUCTION

This project explores the design, training and validation of Multilayer Perceptron, implemented with the neural network architecture using the Fashion-MNIST image dataset. The objective of this project is to pre-process the data to be ready for training, build a feed forward architecture from stretch in PyTorch, and apply backpropagation to optimize the model's parameters. Through the systematic experiment, this report examines how various hyperparameters, optimization method and choice of activation methods affect the models performance in terms of validation accuracy on validation dataset. The fashion-MNIST dataset which consist of grayscale images representing different categories of clothing, provide a suitable benchmark for classification models. In this report, design choice, tuning of parameters, and results are discussed for understanding the behaviour of neural networks.

2. DATA PREPROCESSING

The fashion-MNIST dataset consists of training set of sixty thousand images and labels and testing set of ten thousand images and labels of twenty-eight by twenty-eight pixels. The data was loaded using the pandas library, and the testing set was split into training data and validation data using the twenty-eighty rule, and in both sets of data, the labels and images were separated. This implementation was done as to ensure the models train on input data and the true labels. The datasets were normalised so as to prevent feature dominance using the min-max scaling and turned into tensors for the Pytorch model training.

3. MODEL DESIGN - Multilayer perceptron NN

The Multilayer perceptron neural network is implemented using the object oriented design for easy structuring of the data building up the model logic. The network follows a fully connected feedforward architecture consisting of an input layer, two hidden layers, and an output layer. The input layer receives flattened image vectors of size 784 (28×28 pixels), which are passed through two hidden layers with 128 and 64 neurons respectively. The output layer contains 10 neurons corresponding to the 10 clothing categories in the dataset.

Each hidden layer is followed by an activation function, either ReLU or Sigmoid, depending on the hyperparameter configuration being tested. The model's output layer uses the softmax activation to convert logits into probability distributions across the output classes.

The network's parameters are optimized using gradient descent-based methods either Adam, SGD, or RMSprop, depending on the hyperparameter configuration being tested, with different learning rates explored during experimentation. The model is trained using cross-entropy loss, which is well-suited for multi-class classification problems. The design emphasizes flexibility, allowing different combinations of hyperparameters, the learning rate, activation function, and optimizer to be evaluated to identify the best-performing configuration. The defaults batch size of sixty-four was adopted in all trainings of the models.

4. TRAINING AND VALIDATION

4.1.1. The Base Model

To benchmark the models trained with different hypermeter configurations, a benchmark base model was developed, with parameters, the learning rate 0.001, optimizer being the Adam method, and ReLu function being the activation function in all the networks layers. Due to computational resources, the number of iterations a model was allowed to train on were limited to only ten. This hyperparameter configuration was chosen on bases that it represents a well-balanced effective starting point for training neural networks, with the Adam method being an adaptive optimizer, the ReLU functions being the simplest of activation functions and being computational efficient, and the learning rate at 0.001 normally works well across many tasks in models development.

4.1.2. Hyperparameter Optimisation Experiment

For the purposes of this project, hyperparameter configuration combination being tested include the learning rates 0.001, 0.01 and 0.1, the activation functions namely the ReLU function and Sigmoid function, and the optimizers being the Adam, SGD, and RMSprop, and with the number of iterations for the model trainings kept at ten. These combinations are chosen mainly based on intuition and in avoidance to computational complexity due to the machine being used in development being the limiting factor (Windows i3, the Celeron CPU). However, these hypermeters combinations also allows us to benchmark our models against the base model, at the same time giving us possibly best performance in terms of validation accuracy.

The combination of two activation functions, three learning rate values and three optimizers, gives together a total of 18 models, and thus being trained. This step comes heavy on computational efficiency, with a total running time averaging at around 10 minutes. This is due in effort of automating the training process of the above-mentioned combinations, coming down to an architectural design of three nested loops. Each loop for each hyperparameter. In training, the hyperparameter combination at that point is stored in an array and the model is also stored in an array for reporting purposes.

5. DISCUSSION

5.1.Base Model

The base model archived not a surprisingly, 88.08% validation accuracy, in 10 training iterations. This serves as the reference point for assessing the impact of various hyperparameter combinations. Call this model B.

5.2.Benchmarking models

5.2.1. Best vs. Baseline Performance

With the above-mentioned combinations, the model achieving the lowest validation was the combination of hyperparameters, ReLU, RMSprop and learning rate of 0.1, achieving an accuracy of 10.98%. Call this model P. Whilst best performing model was the combination of ReLU as activation function, RMSprop as optimizer, and learning rate of 0.001, achieving a validation accuracy of 88.32% (see the notebook). Call the best performing model C.

Comparing the base model to our best performing model of the 18 models trained, it is observed that the is only a difference of 0.24% validation accuracy between the 2 models. That said, throughout the experimentation, the combination of the base model hyperparameters would sometimes yield the best performing model in the 18 trained models, while model C would come in second, again not with so much difference between the validation accuracy of C and B. Overall, on many repeated trainings, model C would outperform model B. Highlighting model C as best performing model in the given hyperparameter combinations given.

5.2.2. Hyperparameter Influence

By comparing models C and P which had the same hyperparameter configuration except for learning rate, it is clear that the learning rate had the most significant influence on performance. Model P, trained with a high learning rate of 0.1, achieved only 10.98% accuracy, showing that large step sizes cause the optimizer to overshoot the optimal weights, preventing convergence.

While a smaller learning rate of 0.001 allowed the model to make gradual, stable updates to its parameters. This steadier progression enabled the network to reach a lower loss and achieve higher validation accuracy.

These results suggest that moderate learning rates help balance speed and stability, meaning fast enough to learn efficiently but small enough to avoid fast divergence. While activation functions and optimizers also affected results, their effectiveness depended largely on a well-tuned learning rate.

5.2.3. Overfitting Analysis

The comparison between the base and best-performing models shows that hyperparameter tuning effectively reduced overfitting. In the base model, the validation loss remained consistently higher than the training loss, indicating limited generalization. However, in the best-performing model, both training and validation losses decreased smoothly and remained aligned across epochs. This smaller gap between the two curves demonstrates improved generalization and model stability. Overall, model C achieved lower losses and better balance between training performance and validation accuracy, confirming that the adjustment of optimizer mitigated overfitting.

5.2.4. Error Analysis

Looking at the confusion matrix of the best performing model, C, the class that was mostly wrongly predicted was class 6. Class 6 was confused with class 0 by the model. The second mostly wrongly predicted class was class 2, the model confused it with class 6. Visualizing these classes, we observe shirts and t-shirts are represented by these classes. This wrongly classification by the model would likely due to the difference in sleeve lengths and the shape. Other misclassifications occurred, suggesting that the model failed to learn the edges of the pictures and mostly differentiating features.

6. CONCLUSION

This project successfully trained a multilayer perceptron on the Fashion-MNIST dataset and explored how hyperparameters affect performance. We found that learning rate had the largest impact, with smaller values stabilizing training and improving validation accuracy. Activation functions and optimizers also influenced results, but less significantly.

Hyperparameter tuning improved performance over the baseline and reduced overfitting, as shown by smoother loss curves. Confusion analysis highlighted visually similar classes that were often misclassified, revealing limitations in feature distinction.

Overall, the project demonstrates the importance of systematic hyperparameter exploration and provides a foundation for further improvements, such as dropout, batch size tuning, or deeper architectures.