CSC3022F Machine Learning Assignment 1- Artificial Neural Networks

Student Nikita Martin

Introduction

This project investigates the feedforward artificial neural network, training configurations, and the hyperparameters used, to understand how these factors influence the classification performance. The goal of this assignment is to identify an effective ANN topology for the FashionMNIST, and to evaluate the design choices on the classification accuracy. This report outlines the neural networks used to classify FashionMNIST image, covering model design, an explanation of how the networks are trained and validated, and evaluation.

Dataset and Pre-processing

We first start with downloading the FashionMNIST dataset which consists of a collection of 70000 grayscale images of 10 fashion categories, each of size (28x28) pixels. **Dataset**: FashionMNIST (60000 training, 10000 test images) **Image size**: 28x28 grayscale image **Categories**:

- T-shirt/top, Trouser, Pullover, Dress, Sandal, Sneaker, Bag, Ankle boot, Coat, Shirt

0.1 Normalization (Pixel Value Scaling)

Before training, all FashionMNIST images were converted to tensors and normalized to a [0,1] range, by dividing the values by 255. Colour images are treated similarly, with each colour channel being normalized independently.

Justification: This standardization improved training stability by ensuring that the input values remain within a consistent scale. Normalization is essential in deep learning to reduce the impact of differences in input features, improve convergence of the optimization process, and increase generalization by reducing overfitting and thereby strengthening the model more to handle data variations. This generally leads to faster and improved model performance.

0.2 Flattening

Following normalization, each image, which was originally multidimensional (1, 28, 28), was then flattened into a 784-element vector (28*28) which is required by a feedforward network consisting of fully connected (dense) input layers.

Justification: Flattening is necessary when feeding images as input to fully connected layers. These layers expect a 1-D vector, preserving the pixel information while conforming to the input requirements.

During prediction, external JPEG images were converted to grayscale and resized to 28x28, and then inverted to match the dataset's white-on-black style, improving prediction reliability.

0.3 Obtaining the Validation/Training/Test sets

- **Test set**: The test set is used to validate the model's performance after the model has been trained for classification. It contains examples that the model has not seen before.
- **Training set**: The training set consists of the largest subset of the dataset and is used to train the model. This is the set that the model uses to learn to make predictions or decisions.
- **Validation set**: The validation set is used during training to evaluate the model's performance and to help adjust the hyperparameters of the model. This helps us to make adjustments to the model during training, to get an idea of how the model is generalizing to examples that are not in the dataset.

After loading the FashionMNIST dataset with Pytorch, we have two sets of data: the test set and training set. Given these two sets, a third set - the validation set was created by splitting the existing training set - consisting of 60000 samples into the training set containing 50000 samples and a validation set containing 10000 samples.

Building a Feedforward Neural Network (FNN)

A Feedforward Neural Network is a type of artificial neural network whereby information flows in a single direction - from the input layer, through the hidden layers, and then to the output layer. This model is often used for pattern recognition tasks such as image classification.

- Input layer: The input layer receives the raw data, where each neuron represents a single feature of the input. For the FashionMNIST dataset, this means that there are 784 neurons (28x28 pixels per grayscale image). Preprocessing for input: The pixel values are normalized to the [0,1] range by dividing by 255. This normalization helps in faster convergence during training.
- Hidden layers: Hidden layers lie between the input layer and the output layer. Each neuron in a hidden layer computes a weighted sum of its inputs, adds a bias, and then applies an activation function (such as ReLU) to introduce non-linearity. These layers allow the Three hidden layers were tested, to analyze the trade-off between the model complexity and performance.
- Output layer: the output later generates the final classification result. Each neuron corresponds to one of the target classes and generates a probability score, typically using the Softmax function.

0.4 Topology of the networks investigated

0.4.1 Investigation of the Hidden Layers

Investigating the number of input layers on the model performance:

- **Shallow network**: 1 hidden layer , which is a simpler model, that is faster to train
- **Deeper network**: 3 hidden layers which is more expressive and can learn more complex features.

Aspect	Shallow Network	Deep Network
Number of Hidden Layers	1	3
Prediction Accuracy Example Architecture	[784] → [256] → [10]	[784] → [512] → [256] → [128] → [10]
Training Time	59.1s - Faster	3m 13.0s - Slower
Overfitting Risk	Lower	Higher
Convergence	Faster	Slower - May require more epochs
Test Accuracy	85,32%	87.80%
Parameter Count	Fewer parameters	More parameters

Table 1: Comparison of Shallow vs Deep Feedforward Neural Networks

When comparing a shallow and deeper feedforward neural network in terms of accuracy, training time, generalization:

- **Shallow network (1 hidden layer with 256 input features)**: achieved a test accuracy of 85.32% significantly lower training time (59.1s). It converged quickly and posed less risk to overfitting due its limited capacity.
- **Deep network (3 hidden layers with 512, 256, 128 units)** achieved a higher test accuracy of 87,80%, however, it increased training time to 3 minutes and 13 seconds. This network posed a higher risk of overfitting without careful regularization or dropout.

0.4.2 Network Architecture

The number of neurons in each hidden layer defines the capacity of the model, and significantly impacts the performance of the model and its ability to learn patterns. Finding a good number of neurons to define the hidden layer is important, as this determines the model performance, and finding the right balance can lead to underfitting whereby the model struggles to learn the patterns and yields a low accuracy, or overfitting whereby the networks essentially memorizes the the training set, resulting in poor generalization, decreasing the prediction accuracy. My network model consisted of:

- **Input Layer**: 784 neurons (flattened 28x28 grayscale image). The FashionMNIST images are 28x28 grayscale images are flattened into a 784-dimension input vector, allowing the network to treat each pixel as a feature.

- **Hidden Layer 1**: 512 neurons, implemented with the ReLU activation function
- **Hidden Layer 2**: 256 neurons, implemented with the ReLU activation function
- **Hidden Layer 3**: 128 neurons, implemented with the ReLU activation function
- **Output Layer**: 10 neurons, with the Softmax activation (one per class)

Final Architecture: [784] - [512] - [256] - [128] - [10]

The hidden layers are implemented in a pyramidal structure, whereby each layer has fewer neurons than the one before it. This design was inspired by the architecture of the Convolutional Neural Network(CNN), where the data gets more focused and detailed as it moves through the layers. Although this is a fully connected network and not a CNN, using fewer neurons in each layer helps the model learn simpler and more useful features step by step. This approach was adopted in an attempt to help the model make better predictions and train more effectively.

0.4.3 Connectedness

Each layer in the network is fully connected to the next, meaning that every neuron in one layer passes information to every neuron in the next layer, allowing the network to learn the relationships between the input features and the output classes.

0.4.4 Activation function

: A series of tests were done with various activation function to determine the best one to be used:

Activation Combo	Layer 1	Layer 2	Layer 3	Test Accuracy (%)
A	ReLU	ReLU	ReLU	88.59
B	Sigmoid	Sigmoid	Sigmoid	84.33
C	Tanh	Tanh	Tanh	88.67
D	ReLU	Tanh	ReLU	88.50
E	ReLU	Sigmoid	ReLU	88.25
F	Leaky ReLU	Leaky ReLU	Leaky ReLU	88.55

Table 2: Comparison of Activation Function Combinations in a 3-Layer Neural Network

Several combinations of activation functions were tested in the hidden layers to evaluate their impact on the model's performance. Surprisingly, although ReLU is commonly used in neural networks due to its efficiency, however, in the experiments above, Tanh consistently performed the best by achieving the highest test accuracy of 88.67% when used in all layers. To confirm that the results were consistent, the tests above were run multiple times. Tanh maps the input to a [-1, 1] range, which appears to stabilize the learning of the model. Based on the tested results, Tanh was selected as the activation function for the hidden layers in the final model.

0.4.5 Batch Normalization

: To improve both the test and validation accuracies, batch normalization was explored and implemented. Batch normalization is a technique that normalizes activation functions of each layer for every mini-batch. This normalization leads to faster convergence, improved training stability by reducing sensitivity to parameter initialization, and better generalization performance. When batch normalization was implemented on the hidden layers, an increase in both training and test accuracies were observed, however this came at a cost of slightly increased time during training due to additional computations that were introduced.

0.5 Loss and Optimization Functions

: Loss and optimization functions are two of the important components in neural networks. The loss function measures how wrong the model's predictions are compared to the actual(true) values in the training data. The optimization function use this loss to update the model's parameters in effort to reduce the loss over time and improving performance.

0.5.1 Loss Function

In context of the FashionMNIST classification task, the loss function influences the model's ability to distinguish between similar categories and to generalize effectively to unseen data. Therefore, selecting an appropriate loss function is important in achieving both high accuracies and stable training behaviour.

The CrossEntropyLoss function, and the combination of LogSoftmax in the output layer followed by the NLLLoss functions were experimented with. CrossEntropyLoss is widely used for multiclass classification as it combines the NLLLoss and LogSoftmax activation into a single function. Surprisingly, the alternative setup which uses the combination of NLLLoss followed by LogSoftmax achieved slightly better test accuracies across multiple runs. This could be due to improved gradient stability and the more direct control that it offers over the output layer. As a result, we selected the latter approach for our final model architecture.

0.5.2 Optimization:

In terms of optimizers, three optimizers were evaluated: SGD without momentum, SGD with Momentum, and Adam.

Optimizer	Learning Rate	Convergence Speed	Test Accuracy	Stability
SGD no Momentum	0.001	Slow	86.14%	Moderate
SGD with Momentum	0.001	Slow	88.79%	High
Adam	0.0001	Fast	88.35%	High

Table 3: Comparison of Optimization Algorithms on FashionMNIST

To evaluate the impact of different optimizers on the model performance, three different optimizers were experimented: Adam, SGD with momentum and SGD without momentum. As shown in the table, Adam demonstrates the fastest convergence with consistent high stability, while SGD with momentum achieved the highest test accuracy with a lower convergence rate, which suggest with the right tuning, the momentum based optimizer may lead to better generalization. However, although the SGD with momentum yielded a slightly higher accuracy, Adam was selected for its better training efficiency.

1 Learning Rates:

The learning rate is a hyperparameter that controls how fast the model learns and updates its weights during training. A higher learning rate causes the model to converge faster but may lead to overfitting, while a lower learning rate converges slower and may lead to underfitting. An ideal learning rates depends on the size of the neural network and the training data. To determine the optimum value, three values were experimented on the Adam optimizer: 0.01, 0.001, 0.0001. The corresponding test accuracies were 88.86%, 88.72%, and 88.83% respectively. Based on these results, a learning rate of 0.01 yielded the highest test accuracy for this model, and was selected.

2 Training and Validation

How the Networks are Trained and Validated: The neural network was trained using mini-batch gradient descent with the Adam optimizer. A loss function called Negative Log Likelihood Loss (NLLLoss) was used, along with with LogSoftmax activation function at the the output layer to convert the prediction into probabilities. During training, the model processed the training data in mini-batches.For each batch, the model made predictions and calculated how far off those predictions were using the loss function, and then updated its parameters based on the loss values.

Throughout the training process, the program kept track of how well the model is doing by measuring both of the prediction accuracies and average loss loss tracked over the entire training set. The prediction accuracy was then calculated using an accuracy function that compared the predicted values to the true labels. After every 200 batches, the training progress updates were printed to monitor how well the model was doing.

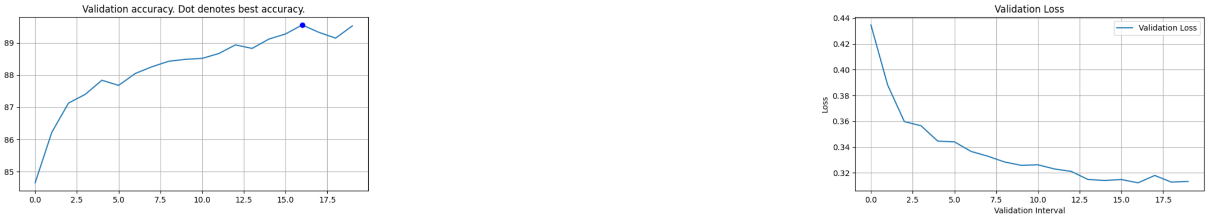
Once training completed, the model was evaluated on a separated validation set to measure how well it generalizes to new and unseen data. To prevent overfitting and and unnecessary time wasted during training, early stopping was used. This helped to monitor the the validation accuracy after each epoch, and then stop training if no significant improvement is made over a set on consecutive epochs (patience). This ensured that the model maintained the best weights that are observed during training.

During this evaluation, the model's predictions were made without updating any parameter (weights). The average loss and prediction accuracy on the validation set were calculated to measure generalization performance on unseen data.

3 Overcoming Overfitting

3.1 Regularization Techniques

To improve generalization and reduce overfitting, several techniques were implemented in the model architecture, including dropout, early stopping, and batch normalization. Dropout was applied to the hidden layers to prevent the model from relying too much on the neurons in the hidden layers. To determine the dropout value, various dropout rates were tested [0.1, 0.2, 0.3, 0.4 , 0.5] and 0.5 yielded the best performance and generalization, reducing overfitting. Early stopping was used to stop training once the validation accuracy stopped improving which avoided the model from overfitting unnecessary. Additionally, batch normalization was introduced on each of the hidden layers before the activation functions were applied, which stabilized and accelerated the training process while improving the training and validation accuracies.



(a) Validation Accuracy

(b) Validation Loss

Figure 1: Model Training Performance

4 Final Evaluation

After implementing and refining the model architecture, the final model was trained on the full training dataset and tested on the test set. The model achieved a test accuracy of 88.79, demonstrating the strong performance of the FashionMNIST classification task. Additionally, to further test the model's generalisation capabilities, the model was tested on external grayscale jpegs which were resized to 28x28 pixels, and then predicted the classes of these images, showing the it can generalize to input data beyond the original data set, provided that the inputs are processed. These results validate the effectiveness of the model with the architectural choices made, and show that the trained model can be reliably used for image classification involving similar grayscale items.

References

@miscfeedforwarsneuralnetwork, author = Anonymous, title = Feed Forward Neural Networks, howpublished = `https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network#:~:text=The%20architecture%20of%20a%20feedforward,layers%20are%20interconnected%20by%20weights.`, note = Accessed: 2025-05-02

@mischiddenlayerneurons2024, author = Tiancheng Deng, title = Effect of the Number of Hidden Layer Neurons on the Accuracy of the Back Propagation Neural Network, year = 2023, howpublished = `https://www.researchgate.net/publication/377755645_Effect_of_the_Number_of_Hidden_Layer_Neurons_on_the_Accuracy_of_the_Back_Propagation_Neural_Network`, note = Accessed: 2025-05-03

@misclossfunctions2022, author = Vishal Yathish, title = Loss Functions and Their Use In Neural Networks, year = 2022, howpublished = `https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a47`
note = Accessed: 2025-05-03

@misclearningrates2018, author = Sayak Paul, title = Introduction to Cyclical Learning Rates, year = 2018, howpublished = `https://www.datacamp.com/tutorial/cyclical-learning-neural-nets`, note = Accessed: 2025-05-05

@miscthelearningrate2023, author = Mohit Mishra, title = The Learning Rate: A Hyperparameter That Matters, year = 2023, howpublished = `https://mohitmishra786687.medium.com/the-learning-rate-a-hyperparameter-tha`

~:text=A%20higher%20learning%20rate%20will,of%2Dthe%2Dart%20results., note = Accessed: 2025-05-05

@miscbatchnormalisation, author = deepchecks, title = Batch Normalization, howpublished = https://www.deepchecks.com/glossary/batch-normalization/#:~:text=Batch%20normalization%20is%20a%20deep,that%20can%20occur%20during%20training., note = Accessed: 2025-05-05

@miscFeedforwardNeuralNetworks(FNN), author = zilliz, title = Understanding Feedforward Neural Networks (FNNs): Structure, Benefits, and Real-World Application, howpublished = https://zilliz.com/glossary/feedforward-neural-networks-(fnn), note = Accessed: 2025-04-29

@miscneuralnetworkimplementation2025, author = geeksforgeeks, title = How to implement neural networks in PyTorch?, howpublished = https://www.geeksforgeeks.org/how-to-implement-neural-networks-in-pytorch/, note = Accessed: 2025-04-28

@miscneuralnetworktechniques, author = Minhajul Hoque, title = Demystifying Neural Network Normalization Techniques, year = 2023, howpublished = https://medium.com/@minh.hoque/demystifying-neural-network-normalizat
note = Accessed: 2025-05-04