Assignment 3 in INF-1600: Neural Network

aal107@uit.no

November 3, 2024

Contents

1 Introduction

This report examines the optimization and overfitting of a neural network classifying images from the FashionMNIST dataset. The goal is to improve performance through architectural and hyperparameter adjustments.

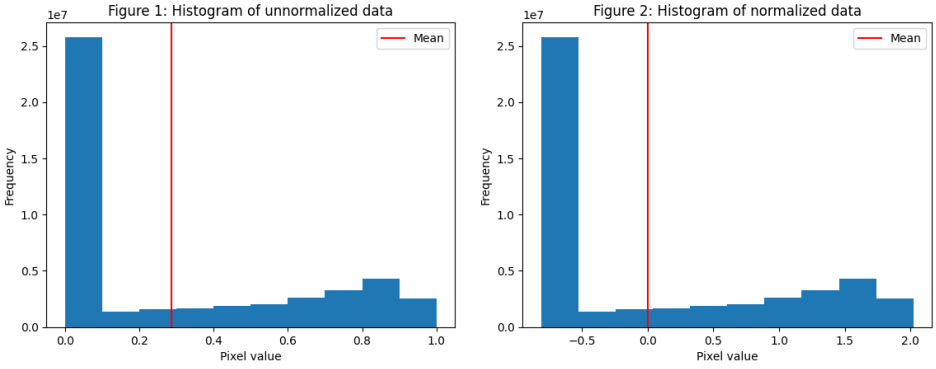
2 Data Preparation

2.1 Normalization in Neural Networks

Data normalization scales inputs to a consistent range, typically with a mean of 0 and standard deviation of 1, improving training efficiency and model generalization by stabilizing gradients[1].

2.2 Pre- and Post-Normalized Data Observations

Figures 1, (A and B) show pixel distributions before and after normalization. In Figure A (unnormalized data), values range from 0 to 1, with most pixels concentrated near 0, resulting in a darker overall distribution. Figure B (normalized data) centers values around 0, spreading them across a standardized range from -1 to 2. This normalization balances pixel intensities, helping the network by ensuring all inputs are on a similar scale, which improves convergence and stability.



(a) Before Normalization (b) After Normalization

Figure 1: Data Distribution Before and After Normalization

3 Tweaking Hyperparameters

3.1 Effect of Adjustable Parameters on Model Training

Each hyperparameter influences model accuracy, convergence, and training time[2]:

- **Batch Size:** A smaller batch size can provide a more detailed gradient estimation, improving model accuracy at the cost of increased training time. Larger batch sizes reduce training time but may lead to less precise convergence.
- **Learning Rate:** Controls the step size in the optimization process. A higher learning rate speeds up training but risks overshooting optimal values, while a lower rate ensures finer adjustments, potentially leading to better convergence but longer training times.
- **Optimizer:** The choice of optimizer (e.g., SGD, Adam) affects how the model adjusts weights. SGD is simple but may converge slowly, while optimizers like Adam can improve convergence speed and stability.
- **Loss Function:** The loss function measures prediction error. Different loss functions are suited to different tasks and can affect model accuracy and convergence behavior.
- **Hidden Neurons:** The number of hidden neurons in layers determines the model's capacity. More neurons allow the model to capture complex patterns but can lead to overfitting and increased computation time.
- **Epochs:** The number of epochs controls how many times the model processes the training data. More epochs allow the model to learn better but may lead to overfitting and longer training time if over done.

3.2 Base Model Training and Testing Performance

The base model achieved a training accuracy of 45.28%, shown in Table 1. Figure 2 shows a downward trend in training loss with a slight test loss increase.

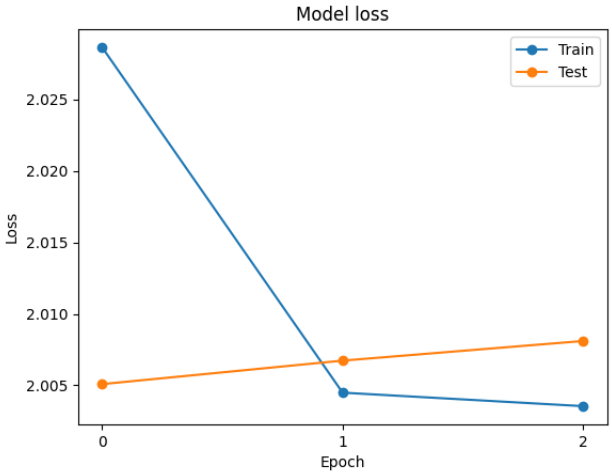


Figure 2: Model loss for the base program

3.3 Hyperparameter Tuning Experiments

Table 1 summarizes the outcomes of tuning key parameters, including batch size, learning rate, number of hidden neurons, epochs and optimizer, with the goal of enhancing model accuracy.

Experiment	Batch Size	Learning Rate	Optimizer	Hidden Neurons	Epochs	Accuracy
Base	32	2	SGD	5	3	45.28%
1	32	0.01	SGD	50	10	83.70%
2	64	0.001	SGD	100	20	75.42%
3	128	0.0005	SGD	200	30	70.57%
4	128	2	SGD	200	10	78.02%
5	32	0.01	SGD	100	50	86.62%
6	32	0.01	SGD	150	50	86.57%
7	**64**	**0.001**	**Adam**	**128**	**10**	**88.27%**
8	**64**	**0.001**	**Adam**	**128**	**20**	**88.07%**

Table 1: Hyperparameter Tuning Results with SGD and Adam Optimizers

3.4 Final Model Performance

The highest model accuracy, 88.27%, was achieved in Experiment 7 using a batch size of 64, learning rate of 0.001, Adam optimizer, 128 hidden neurons, and 10 epochs. This setup notably outperformed both the base model and all SGD configurations.

Figure 3 illustrates the training process of this optimized model. Training loss steadily declines over the epochs, and test loss initially drops before stabilizing, suggesting minimal overfitting. Adam's adaptive learning rate accelerates early convergence, evidenced by the rapid loss reduction, underscoring its advantage over SGD.

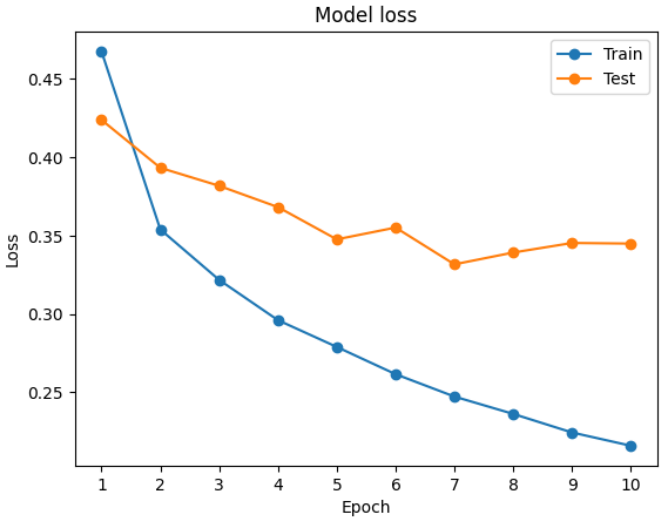


Figure 3: Training process of the final optimized model (experiment 7)

3.5 Impact of Replacing SGD with Adam

Switching from SGD to the Adam optimizer enhanced accuracy, with only minor adjustments to hyperparameters.

- **Optimizer**: Replaced SGD with Adam to allow for adaptive learning rates, leading to more efficient convergence.
- **Batch Size**: Increased to 64 for improved gradient estimation.
- **Hidden Units**: Expanded to 128 to enhance model complexity and feature learning.
- **Epochs**: Set at 10(Experiment 7) and later tested with 20(Experiment 8), though extended epochs showed no accuracy gains.

Experiment 7 (10 epochs) achieved 88.27% accuracy, while Experiment 8 (20 epochs) slightly decreased to 88.07%, indicating potential overfitting with prolonged training.

The adoption of Adam resulted in a substantial accuracy increase from approximately 45% to over 88%. Extended training provided minimal benefit, highlighting the balance between optimization and overfitting risks.

4 Overfitting

4.1 Understanding Overfitting

Overfitting occurs when a model learns the training data too well, capturing noise and specific patterns that do not generalize to new data. This results in high accuracy on the training set but poor performance on unseen data. In a neural network trained on the FashionMNIST dataset, overfitting could mean the model learns to recognize specific textures or patterns unique to the training images, rather than general features of clothing. Consequently, the model may struggle to correctly classify new, unseen images, limiting its practical utility in real-world applications.

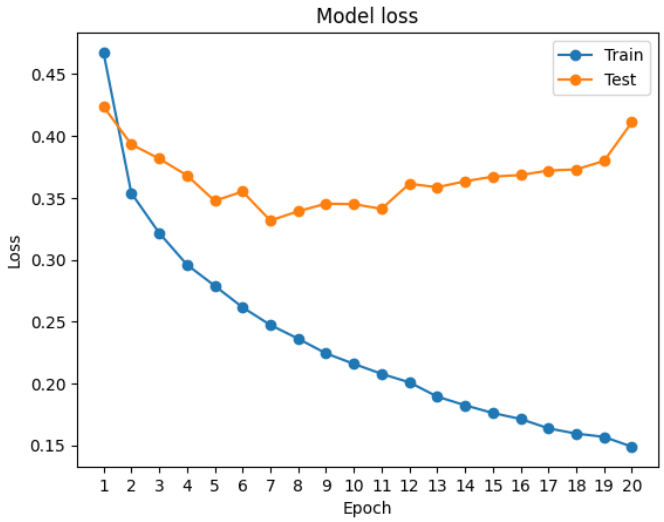


Figure 4: Experiment 8 example of overfitting

In Figure 4 we can see an example of a slight overfitting in the experiment. This conclusion is made from the observation of a steadily decrease of model loss over each epoch on the training data. But in the test data the model decreases at first but starts to rise again around epoch 7. This suggest that the hyperparameters effects the neural network in a way to overfitting occurs on test data, meaning it memorizes the traning data and performs worse on the test data.

4.2 Inducing Overfitting Through Hyperparameters

To induce overfitting in a neural network the following parameters would play a big role for instance:

- **Batch Size:** Using a very small batch size can make the model overly responsive to individual examples, leading it to memorize specific details rather than generalizing patterns.
- **Hidden Neurons:** Increasing the number of hidden neurons gives the model more capacity to capture complex patterns, which can lead it to memorize training data specifics rather than general features.
- **Epochs:** A higher number of epochs allows the model to train longer on the same data, increasing the risk of memorizing the training set and reducing generalization to new images.

5 Neural Network Architecture

5.1 Details of the Neural Network Architecture

The neural network in this assignment is defined by the `NeuralNetwork` class and it's architecture consist of the following:

- **Input Layer:** The input layer consists of a `Flatten` operation that reshapes each 28x28 image into a 784-dimensional vector, allowing it to be processed by the fully connected layers.
- **Hidden Layer:** The model has one hidden layer, which is a fully connected layer with a specified number of hidden units (`hidden_units`). This layer has an input size of 784 (corresponding to the flattened image) and an output size equal to the number of hidden units. The `ReLU` activation function is applied to introduce non-linearity.
- **Output Layer:** The output layer is a fully connected layer with 10 output units, corresponding to the 10 classes in the FashionMNIST dataset. This layer uses the `softmax` activation function to produce probabilities for each class, as this is a multiclass classification problem.

5.2 Improving the Model's Architecture for Image Classification

To improve the model's architecture for image classification, we can use a Convolutional Neural Network (CNN) instead of a fully connected neural network. CNNs are specifically designed for image data and are highly effective for tasks involving spatial hierarchies, such as recognizing patterns and features in images[3].

CNNs are more effective than fully connected networks in extracting and interpreting visual features, leading to improved performance in image classification tasks such as those in the FashionMNIST dataset.

6 Conclusion

This report analyzed how various techniques, including data normalization, hyperparameter tuning, and neural network architecture adjustments, affect the performance of a neural network trained on the FashionMNIST dataset. Data normalization proved to play a big role for stable training, and careful tuning of hyperparameters significantly improved accuracy. The highest performance, reaching 88.27% accuracy, was achieved using the Adam optimizer with optimized batch size, learning rate, and hidden neurons. This demonstrated Adam's advantage over SGD in enhancing model convergence and accuracy.

Furthermore, implementing a Convolutional Neural Network (CNN) could be a future direction to enhance model performance by leveraging spatial hierarchies in image data. These findings underscore the importance of optimizing both hyperparameters and model architecture to maximize neural network performance in image classification tasks.

7 References

- Brownlee, J. (2021). *Using Normalization Layers to Improve Deep Learning Models*. Retrieved from https://machinelearningmastery.com/using-normalization-layers-to-improve-deep-learning-models/

- Pranoy, R. (2019). *What Are Hyperparameters and How to Tune Them in a Deep Neural Network*. Towards Data Science. Retrieved from https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

• Wikipedia Contributors. (2024). *Computer Vision: Recognition.* Retrieved from https://en.wikipedia.org/wiki/Computer$_v$*isionRecognition*