AMATH 582: HOME WORK 4

SUKHJIT KAUR

Department of Mathematics, University of Washington, Seattle, WA
sukhjitk@uw.edu

Abstract. This project explores the classification of images from the FashionMNIST dataset using Fully Connected Deep Neural Networks (FCN). The dataset consists of 60,000 training images and 10,000 validation images, with each sample represented by a 28px by 28px greyscale image. Due to the more challenging nature of this dataset, a more robust and nonlinear model is required to properly train a classifier. A FCN model is designed to intake the reshaped image vector of 784 and output a layer of 10, corresponding to 10 classes. Cross-entropy loss with Stochastic Gradient Descent (SGD) optimizer is used to perform the classification, with adjustable parameters including learning rate and epochs. Validation accuracy and training loss curve are inspected to determine optimal parameters, and finally testing is performed post-training. Hyperparameters are finetuned by testing different learning rates and optimizers.

1. Introduction and Overview

Provided an image set of articles of clothing from Modified National Institute of Standards and Technology (MNIST) dataset, 'FashionMNIST', we attempt to use Fully Connected Deep Neural Networks (FCN) to train a classifier to recognize and sort the images by class. The dataset is split - consisting of 60,000 images in a training set used to train the classifier, and 10,000 images in a test set used for validation of the FCN. The training and test sets, X_{train} and X_{test} are made up of 28px x 28px sized greyscale images. Each pixel in the 28x28 image (784 total pixels) represents a *feature* - this corresponds to the 784 neuron input layer of the model. The clothing type assigned to each image in the training or test set, y_{train}, y_{test} is referred to as a *label* - this corresponds to the output layer of 10 classes. (Figure 1).

By adjusting the number of hidden layers, and neurons, among other parameters like epochs, and learning rate, and by using Cross-entropy loss, the model can be optimized. Including more layers and neurons than necessary increases computational complexity, without much increased benefit. This is important to take into account when designing the FCN model. By iteratively experimenting with the parameter values, the model can be tuned for reasonably high accuracy within a reasonable training time.

2. Theoretical Background

Classification is achieved using a Fully Connected Deep Neural Network model (FCN). FCN is a mathematical method that allows an algorithm to learn patterns within some data, and output a decision based on the inputs.

The neural network models are constructed using artificial neurons, based on *sigmoid neurons*. Each neuron takes an input, x_j, with an associated weight, w_j, and a bias, b, forming the weighted sum $z = x \cdot w + b$. The sigmoid activation function, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

produces an output between 0 and 1, which can be interpreted as a probability. However, due to the limitations of sigmoid activations, such as vanishing gradients, we adopted the ReLU activation function for hidden layers. [2] The Rectified Linear Unit (ReLU) activation function was chosen for the hidden layers due to its computational efficiency and ability to mitigate the vanishing gradient problem. ReLU introduces non-linearity by outputting the input directly if positive, and zero otherwise:

$$f(z) = \max(0, z) \tag{2}$$

This allows the network to learn complex mappings between inputs and outputs. [2]

Date: March 9, 2025.

FIGURE 1. The first 64 images from the FashionMNIST dataset

Cross-Entropy loss was used as the objective function for training, as it is well-suited for multi-class classification tasks. It measures the difference between the true label distribution and the predicted probability distribution, giving a value for training loss.

$$L = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{3}$$

where y_i is the true label (1 for correct class, 0 otherwise) and $\hat{y}_i$ is the predicted probability for class i. This loss pushes the model to output probabilities close to 1 for the correct class and 0 for others. [2]

Three optimization algorithms are explored — Stochastic Gradient Descent (SGD), Adam, and RMSProp. Each optimizer uses different strategies:

- **SGD:** Updates weights using gradients computed from randomly selected mini-batches, often combined with momentum to accelerate convergence.
- **Adam:** Combines the benefits of momentum and adaptive learning rates by maintaining first and second moment estimates of gradients, leading to more efficient updates.
- **RMSProp:** Uses an adaptive learning rate that scales with the moving average of squared gradients, preventing vanishing or exploding gradients.

These methods aim to iteratively adjust weights to minimize the loss function and improve the model's predictive performance.

This math is what describes a feed-forward neural network. The neural network consists of an input layer, with input neurons (w_j and x_j), an output layer with output neurons (σ), and a number of hidden layers in between these layers (no input or output) 3. Information from the input layer is fed only forward, until it reaches the output.

All three methods iteratively adjust weights and biases to minimize the loss function. The update rules for weights w_k and biases b_l in gradient descent are given by:

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k} \tag{4}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l} \tag{5}$$

where η is the learning rate. To reduce computational complexity, Stochastic Gradient Descent (SGD) estimates the gradient using a small random sample (mini-batch) of training data, making updates more efficient.

Iterating over all training data using mini-batches completes one *epoch*, and the process is repeated for a set number of epochs to optimize the model's parameters. This effectively reduces time needed to find the gradient, and perform learning [2].

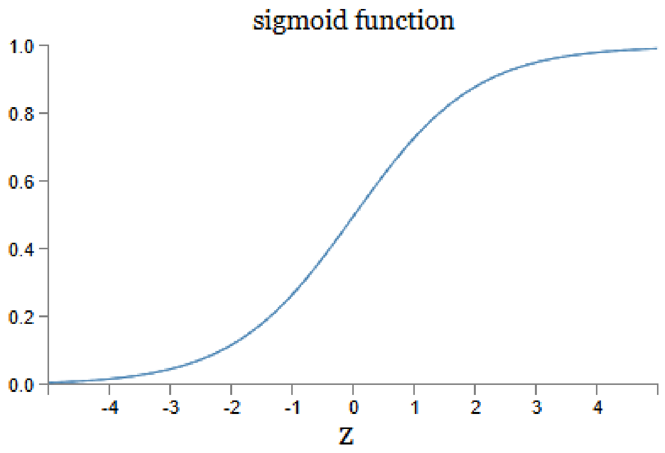


FIGURE 2. The sigmoid function resembles a smooth version of the classic step function. Image credit: Nielsen [2]

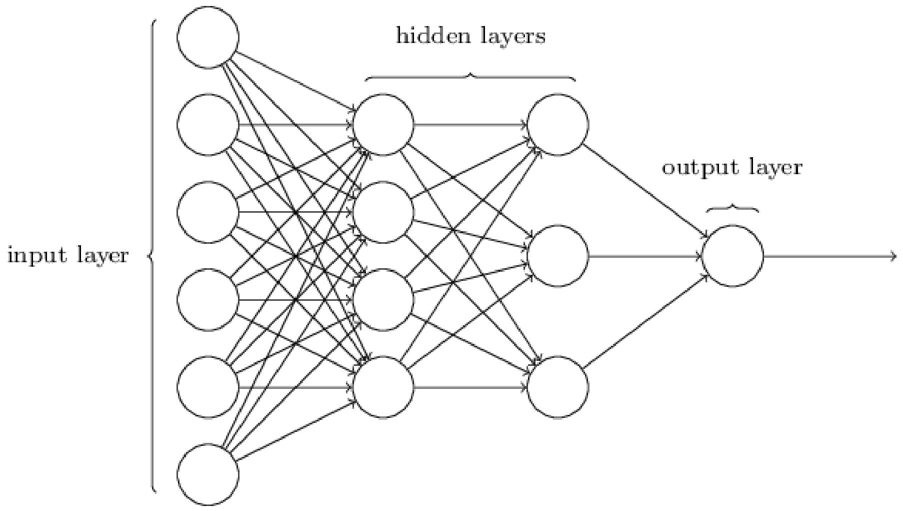
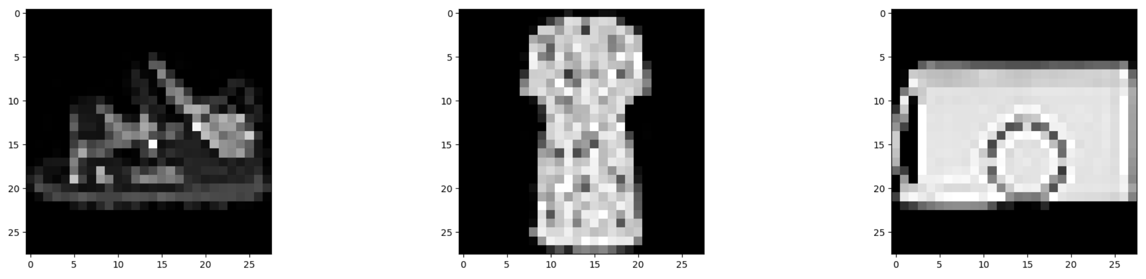
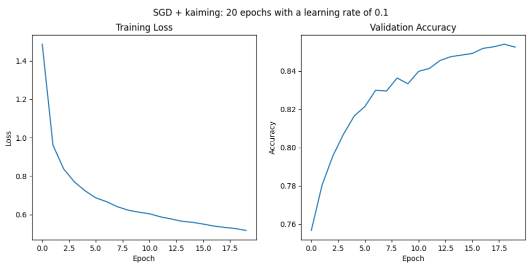


FIGURE 3. The neural network consists of many layers - input, hidden, and output. Image credit: Nielsen [2]

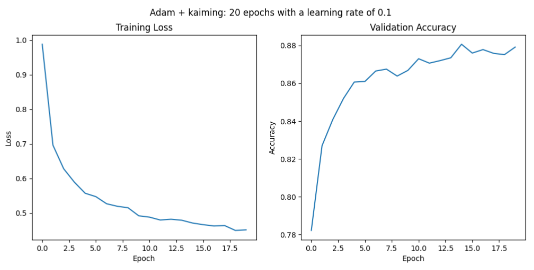


(A) (B) (C)

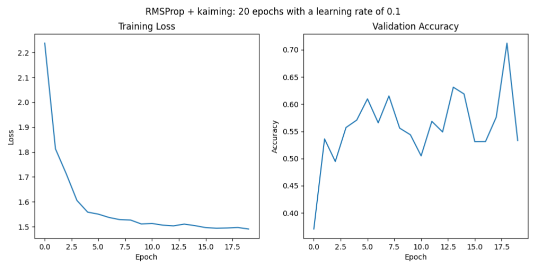
FIGURE 4. The first sample in the first few batches



(A) SGD



(B) Adam



(C) RMSProp

FIGURE 5. Various optimizer types with a learning rate of 0.1, initialized with Kaiming (He). Adam achieves a validation accuracy of 87% (b), compared to 85% by SGD (a), and 53% by RMSProp (c).

3. Algorithm Implementation and Development

The architecture of the FCN comprised an input layer of size 784, corresponding to the 28x28 pixel images in the FashionMNIST dataset, and an output layer of size 10, representing the 10 distinct clothing categories. The hidden layers were designed to be adjustable in both size and number, and ReLU activation functions were employed to introduce non-linearity, thereby enabling the model to capture more complex relationships within the data.

For training, PyTorch's built-in functions were utilized [1]. CrossEntropy loss was selected as the loss function, given its suitability for multi-class classification problems. Various optimizers — including SGD, Adam, and RMSProp — were implemented to compare their impact on the model's performance. The learning rate was treated as a crucial hyperparameter, as it directly influenced the speed of convergence and the final accuracy of the model.

To monitor the model's progress, training loss and validation accuracy were recorded at each epoch. Additional techniques such as Dropout regularization were incorporated to mitigate overfitting, while different weight initialization methods — including Random Normal, Xavier Normal, and Kaiming — were explored to assess their influence on model convergence. Batch Normalization was also integrated to stabilize and accelerate training. A logging system was implemented to record the optimizer type, learning rate, weight initialization method, training loss, and validation accuracy for each training run. The results were saved into a CSV file to allow for comparison of the various model configurations.

4. Computational Results

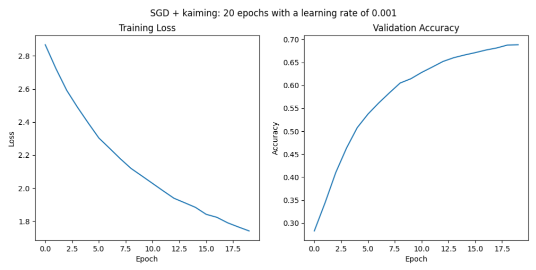
For optimizer comparison, experiments were conducted using SGD, Adam, and RMSProp. Each optimizer was paired with multiple learning rates (ranging from 0.001 to 0.1) to observe how convergence speed and final accuracy were affected (Figure 6b). The training loss and validation accuracy were recorded for each epoch, and the results were visualized through loss curves and accuracy plots. Although, over all epochs, RMSProp appeared to converge quickly, its validation accuracy was not as high as Adam (Figure 5b). This is likely due to the adaptive learning rate mechanism of Adam. However, SGD showed more stable updates, particularly at lower learning rates.

The effect of weight initialization was also explored by testing Random Normal, Xavier Normal, and Kaiming (He) Uniform initializations. Kaiming initialization yielded the most stable training curves and the highest final accuracy, aligning with its theoretical advantage for ReLU activation functions. Xavier initialization performed reasonably well, but Random Normal often resulted in slower convergence and less stable training (Figure 7a).

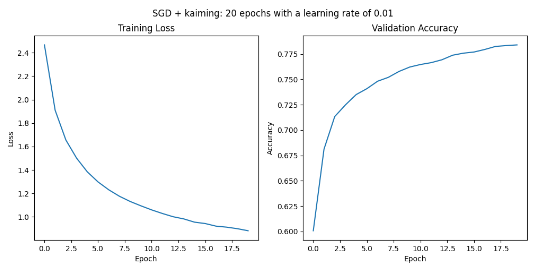
Regularization techniques were incorporated to address overfitting - dropout layers with a dropout rate of 0.5 were added to the hidden layers. The models with Dropout demonstrated improved generalization, as evidenced

optimizer	initialization	learning rate	validation accuracy	test accuracy
SGD	Kaiming	0.001	0.6883	0.687
SGD	Kaiming	0.01	0.7855	0.778
SGD	Kaiming	0.1	0.8525	0.8488
SGD	Xavier	0.01	0.7768	0.7765
SGD	Random	0.01	0.8695	0.858
Adam	Kaiming	0.1	0.8715	0.8667
Adam	Random	0.01	0.8940	0.8830
Adam	**Kaiming**	**0.01**	**0.8938**	**0.8858**
Adam	Xavier	0.01	0.8918	0.8841
RMSProp	Kaiming	0.1	0.533	0.5248

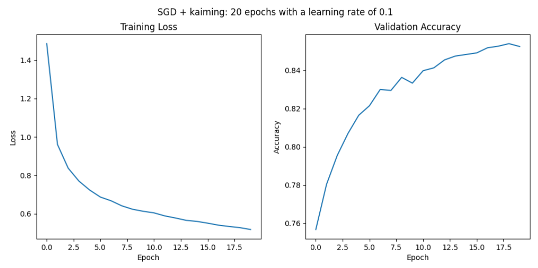
Table 1. Comparison of various configurations of the neural network model, showing the Adam optimizer was consistently superior.



(a) LR = 0.001



(b) LR = 0.01



(c) LR = 0.1

Figure 6. Comparison of different learning rates shows 0.01 (b) converges quickly and is more stable than other rates tested.

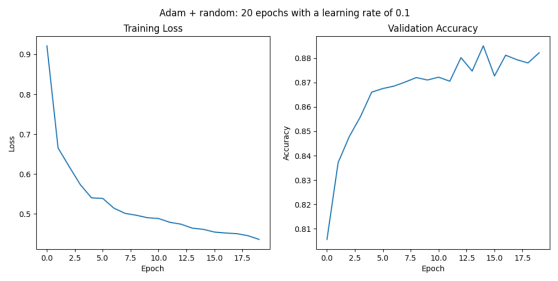
by a reduction in the gap between training and validation accuracy. Additionally, Batch Normalization was tested and found to accelerate training and stabilize learning, especially when combined with higher learning rates (Figure 4c).

The combination of Adam optimizer, a learning rate of 0.01, Kaiming initialization, and the inclusion of both Dropout and Batch Normalization produced the most promising results. Adam was still the most accurate across the board, even when choosing other learning rates or weights (Table 1).

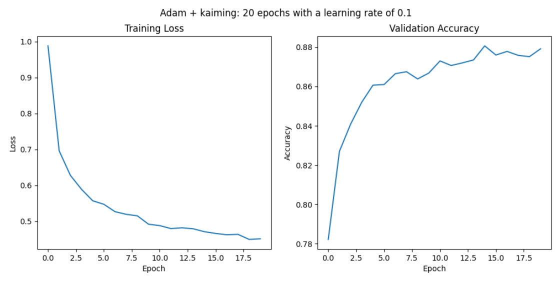
5. Summary and Conclusions

A Fully Connected Neural network (FCN) was developed and tuned to classify the FashionMNIST dataset. The model's flexibility allowed for hyperparameter tuning, enabling investigation of the effects of various optimizers, learning rates, weight initializations, and regularization techniques. Accuracy of greater than 85% was achieved using specific configurations, and surprisingly, some configurations performed poorly.

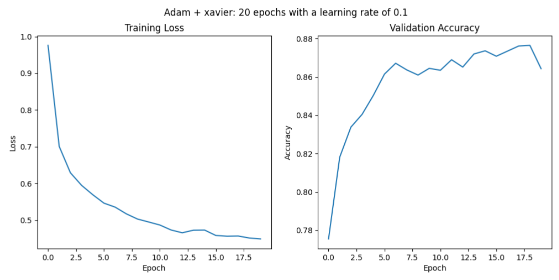
To achieve optimal performance, it was important to select appropriate hyperparameters. Adam optimizer consistently outperformed others in terms of convergence speed and final accuracy, while Kaiming initialization proved to be the most effective for stabilizing training with ReLU activation functions. Regularization through Dropout successfully reduced overfitting, and Batch Normalization further improved training efficiency and accuracy. The best performing configuration consisted of the Adam optimizer, a learning rate of 0.01, initialization of the Kaiming



(A) Random

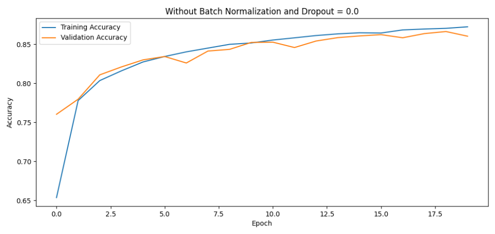


(B) Kaiming

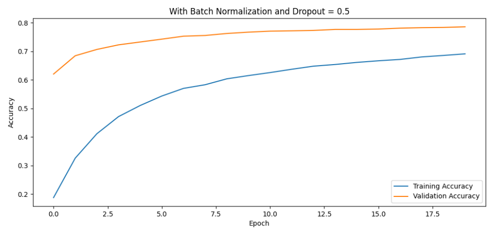


(C) Xavier

FIGURE 7. Initializing with Kaiming resulted in the most stable training (b).



(A) No regularization



(B) With regularization

FIGURE 8. Using Batch Normalization and a Dropout factor of 50%, the model reduces overfitting to the training data.

weight, and inclusion of Dropout and Batch Normalization. This combination resulted in high validation and test accuracy, showing the model's ability to generalize well to unseen data.

This project provided valuable insights into the design and optimization of Fully Connected Neural networks. The systematic exploration of hyperparameters not only improved model performance but also deepened the understanding of how different components interact during training. These findings can inform future work on neural network design and optimization, laying a solid foundation for tackling more complex classification tasks.

Acknowledgements

The author acknowledges the use of the Modified National Institute of Standards and Technology 'Fashion' dataset. The author also appreciates the use of Large Language Models for code debugging resolution and further explanations regarding neural networks.

References

[1] T. L. Foundation. Pytorch documentation.
[2] M. Nielsen. Neural networks and deep learning.