AMATH 482: HOME WORK 4

TIANRUI ZHOU

Amath Department, University of Washington, Seattle, WA
tianrz5@uw.edu

Abstract. This report is about Image Classification with Deep Neural Networks [1]. The content will be divided into five sections: Introduction and Overview, Theoretical Background, Algorithm Implementation and Development, Computational Results, and Summary and Conclusions. They will elaborate on how I used Neural Networks concept and Machine Learning concept to process the raw data FashionMNIST. I used Python (PyTorch) as my data processing tool and generated visual results with it. I will provide detailed explanations of my understanding of the mathematical foundations of the methods and algorithms. I will present my computational results and my conclusion for this project.

1. Introduction and Overview

I am working on the FashionMNIST dataset, which contains images depict articles of clothing, in order to train Fully Connected Deep Neural Networks (FCNs) to distinguish and classify the images in the dataset. The raw data has 60,000 training images and 10,000 testing images, and each of the clothing belongs to a class from one of 10 classes. I will train my classifier using the training set, and then, use validation set and testing set for evaluation and hyperparameter tuning of the FCN model[1].

My goal of the project is designing my FCN model and doing some experiment on it. I will first use SGD optimizer and relu activation in designing the model, with adjustable number of batches, hidden layers and neurons in each of the layer, learning rate, and number of epochs. Then, I inspect the training loss curve and its corresponding validation accuracy, in terms of the number of epochs. I will find parameters, which result in reasonable loss curve and testing accuracy above 85%, to be my baseline configuration. Lastly, I compare different optimizers, apply dropout regularization, initializations and normalization, for purpose of analyzing the effect on the training process and testing accuracy.

2. Theoretical Background

For loading the dataset from the FashionMNIST dataset, we separate the data in to two parts: training data and testing data. And in training data, we separate this data in to two sets: training set and validation set. The testing data is distinct from the training and validation data in order to evaluate the final model's performance.
For each training and testing process, we first construct a class to set up my FCN model, which has adjustable hidden layers and input and output dimension. I set four hidden layers and with 500, 300, 200, 100 neurons in the layers. I set the number of epoch to be 70, to ensure we have enough training opportunities so that it can achieve better accuracy. We also include the activation formula: The ReLU activation function, defined as $f(x) = \max(0, x)$, and tanh activation function,

Date: March 12, 2024.

defined as $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$
Then, we iterate over epochs, batches. we train and validate the FCN model by using **gradient descent**.

$$W_{k+1} = W_k - \alpha \cdot \nabla J(W_k)$$

In this formula, W_{k+1} represents the updated weights, W_k is the current weights, α is the learning rate, and $\nabla J(W_k)$ represents the gradient of the cost function J. In the loop, we also employ the forward propagation, which is computing the loss through forward pass for a single training example, and also the back propagation, which is computing the gradients of parameters through backward pass for a single training example [11].

The **dropout regularization** effectively spreads the weights, and prevents overfitting in neural networks. Random neurons are dropped with probability p. **Initializations** initializing weights from various distributions plays a role in training. Some frequently used initialization procedures are Random Normal, Xavier and Kaiming. **Batch Normalization** normalizes input into each layer for each training mini-batch, and addresses issue of shifting input distributions over training [12].

3. Algorithm Implementation and Development

In the python code, I used some software libraries and packages to help me do the calculation and analysis.

`Torch` is an open-source machine learning library that provides many tools for training neural networks, such as activation functions, optimizer, gradient computation, tensor operation and so on. `torch.nn.Module` is the base class for all neural network modules [6]. `optimizer.zero_grad()` resets the gradients of all optimized `torch.Tensor` s [8]. `torch.no_grad()` is the context-manager that disables gradient calculation [7].

The `NumPy`[2] library is used for numerical operations and array manipulations. Some built-in functions in this package are really crucial to my program. `np.zeros()` returns a new array of given shape and type, filled with zeros [5]. `np.sum` sum of array elements over a given axis [4].
The `tqdm` is a library that provides a simple and smart way to add progress bars to and iterable objects, and it can also predict the remaining time [13].
The package `Matplotlib`[3] is used to make visualization of my data. `Seaborn` is a Python data visualization library based on `matplotlib`. It provides a high-level interface for drawing attractive and informative statistical graphics [10].
The `sklearn.model_selection` module is intended to assist with cross-validation. `train_test_split` split arrays or matrices into random train and test subsets [9].

4. Computational Results

In this project, I am experimenting on designing my FCN model with adjustable number of batches, hidden layers, neurons in each layer, learning rate for each optimizer, number of epochs. First, I find my baseline of FCN which has reasonable running time, training loss curve, validation and test accuracy, by using the SGD optimizer.

Figure 1 is the training loss and validation accuracy curve of my baseline of my FCN, using SGD optimizer. By varying adjustable parameters, I log my result of my baseline configuration, which shows in Table 1. It has reasonable running time of **3 minutes and 15 seconds**, as well as decreasing training loss curve and increasing validation accuracy curve. The testing accuracy, **87.56%**, is above 85%.

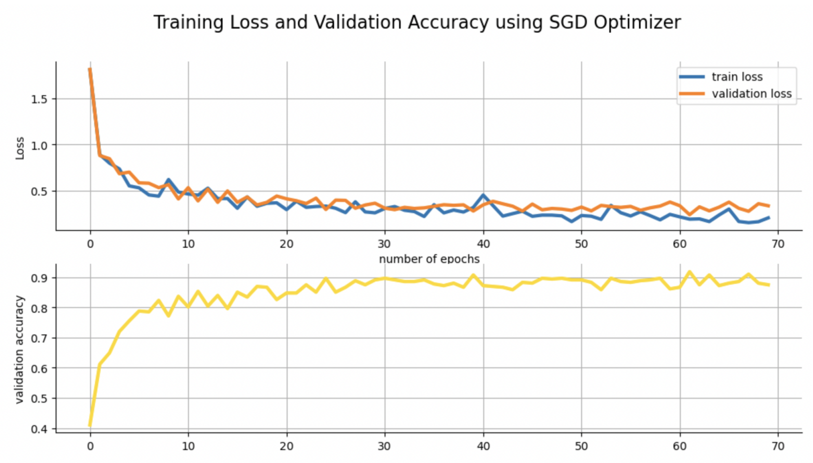


FIGURE 1. Training loss curve and validation accuracy curve using SGD optimizer, with learning rate 0.03

Learning Rate	Number of Epochs	Validation Accuracy (%)
0.03	70	87.50%

Testing Accuracy (%)	# Hidden Layers	Running Time (min)
87.56%	4	3.25

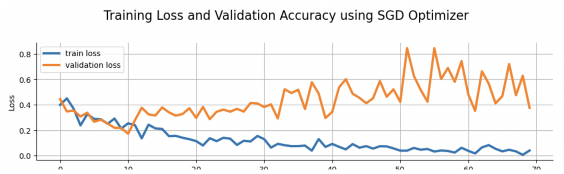
TABLE 1. SGD Results-baseline configuration

TABLE 2. Validation and Test Accuracy for Different Optimizers and Learning Rates

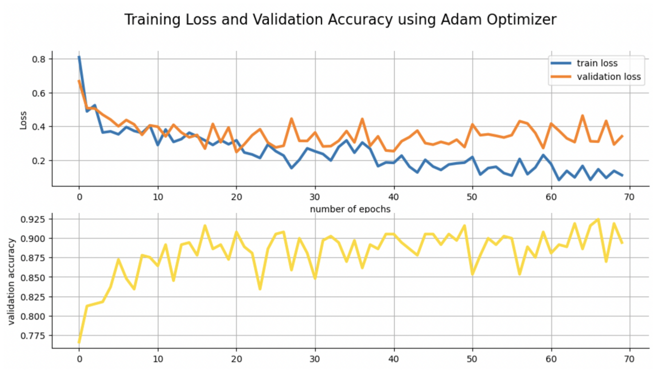
Optimizer	Learning Rate	Accuracy (%)	
		Validation	Testing
SGD	0.03	87.50 %	87.56 %
	0.001	63.32 %	65.68 %
	0.0001	8.42 %	11.75 %
Adam	0.03	10.60 %	9.91 %
	0.001	93.48 %	89.44% $\pm$ 2.75%
	0.0001	88.86 %	89.70% $\pm$ 1.94 %
RMSProp	0.03	19.02 %	19.60 %
	0.001	90.49 %	88.14% $\pm$ 2.82%
	0.0001	89.95 %	88.73% $\pm$ 2.22%

Then, I apply hyperparameter tuning from my baseline. I investigate how the choice of optimizer and learning rate affects the performance of my FCN. I tried learning rate of **0.03**, **0.001**, and **0.0001** on each of the three optimizer: **SGD**, **RMSProp** and **Adam**. I record the accuracy result of validation and testing on Table 2, and plot the training loss curve.

According to table 2, SGD exhibits relatively high performance with a validation accuracy of 87.50% and a testing accuracy of 87.56%. However, SGD optimizer seems difficult to reach high accuracy using smaller learning rate such as 0.001 and 0.0001. Both Adam and RMSProp optimizer demonstrate high accuracy with learning rate of 0.001 and 0.0001. The testing accuracy of Adam gets above 89%, which is higher than the testing accuracy 88.14% and 88.73% of RMSProp; Also, the standard deviation of Adam is 2.75% and 1.94%, which both smaller than the standard deviation of 2.82% and 2.22% from RMSProp; In Adam optimizer, the choice with learning rate of 0.0001 has higher testing accuracy and standard deviation. Besides, we can tell from the training and validation loss curve 2b and 2a that the one with learning rate of 0.0001 is less overfitting **??**. Therefore, **Adam optimizer with learning rate 0.0001** is the most suitable for my FCN.



(A) Training loss curve and validation loss curve using Adam optimizer, with learning rate 0.001



(B) Training loss curve, validation loss and accuracy curve using Adam optimizer, with learning rate 0.0001

FIGURE 2. Performance visualization for Adam optimizer with different learning rates

Then, I chose Adam to be my FCN model optimizer. With learning rate of 0.0001, the validation loss curve is above the training loss curve according to figure 2b, so the model is currently **overfitting**. In order to try to fix the overfitting situation, I apply the *Dropout* regularization with dropout probability of 0.5 for each hidden layer. According to Figure 3, the result is that **the variance between the two loss curves becomes smaller and it is less overfitting after using the dropout regularization**, compares to Figure 2b. The validation accuracy is 91.03%, and testing accuracy is 89.70% ± 1.92%.

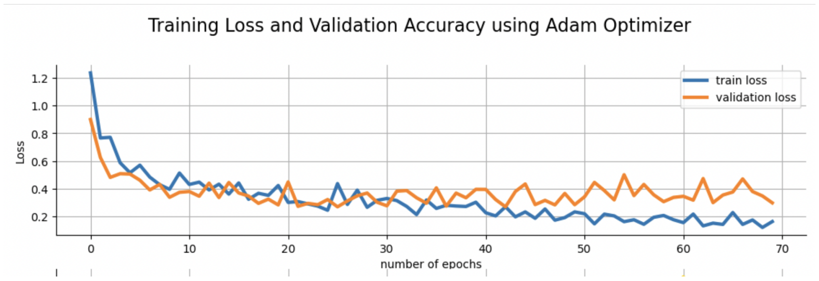
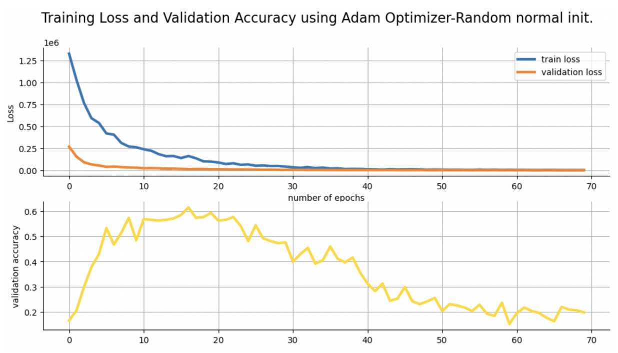
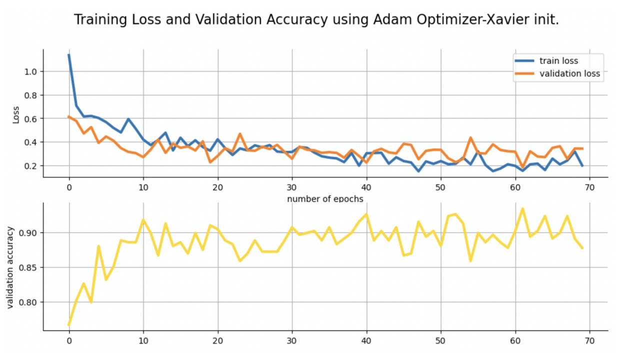


FIGURE 3. Training and validation loss curve using Adam optimizer after using dropout regularization, with learning rate 0.0001

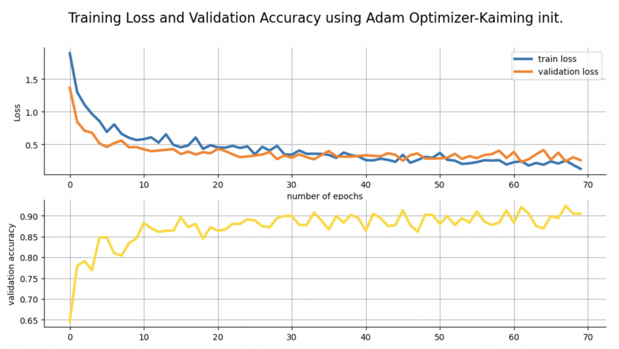
Then, I apply different Initializations, such Random Normal, Xavier Normal, Kaiming (He) Uniform, to discover their effect on the training process and the accuracy of my FCN model.



(A) Training loss curve and validation accuracy curve using Random Normal Initialization



(B) Training loss curve and validation accuracy curve using Xavier Initialization



(C) Training loss curve and validation accuracy curve using Kaiming Initialization

FIGURE 4. Training loss curve and validation accuracy curve using three different Initializations methods

TABLE 3. Initialization Comparison

Initialization	**Validation Accuracy (%)**	**Running Time**	**Testing Accuracy (%)**
Random Normal	19.84	3:34	$18.19\% \pm 2.27\%$
Xavier Normal	87.77	3:44	$88.70\% \pm 1.82\%$
Kaiming (He)	92.39	3:33	$89.16\% \pm 1.79\%$

According to table 3, Random Normal Initialization yields the lowest validation accuracy of 19.84% and lowest testing accuracy 18.19%. For the comparison between Xavier and Kaiming, Kaiming has shorter running time, and higher validation and testing accuracy than that of Xavier. So, **Kaiming Initialization seems to be the most suitable for my FCN**. And by observing Figure 4c, it has less overfitting situation after the initialization, compares to the one 3 after dropout. Using Kaiming Initialization, the validation accuracy increase. Although the testing accuracy slightly decrease, the standard deviation decrease by 0.13%, which means it becomes more stable.

Last but not least, I include *Batch Normalization* to my current FCN model, with figure 5. It has validation accuracy of 88.58% and testing accuracy of $89.55\% \pm 1.73\%$. The testing accuracy is higher than the accuracy after using Kaiming initialization, and the standard deviation is smaller. But the running time 3:46 is longer.

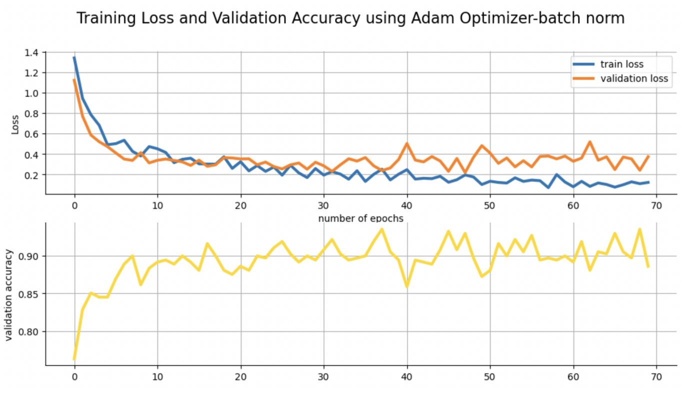


Figure 5. Training and validation accuracy curve using Batch Normalization

5. Summary and Conclusions

This project is about working on the FashionMNIST dataset, which contains images depict articles of clothing, and training Fully Connected Deep Neural Networks (FCNs) to learn to distinguish and classify the images in the dataset, and output 10 categories. I design a FCN model and first use SGD optimizer to set up a baseline configuration, which has reasonable running time accuracy. Through hyperparameter tuning from the basline, I explore how the choice of different optimizer, learning rate, dropout regularization, initialization and normalization affect the performance of my FCN model. As a result, Adam optimizer with learning rate 0.0001, and Kaiming Initialization are the most suitable for my FCN among other optimizer and initialization.

Acknowledgements

The author is thankful to Prof. Eli Shlizerman for useful information in Lectures and Recitation about some important details about this project, and the useful information about how to use PyTorch. We are also thankful to peer Ziyi Zhao and all the classmates on Discord for meaningful discussion of logic of the coding part, and helping me debug my code successfully.

References

[1] Directions, reminders and policies. PDF, 2024. File: AMATH482582$_{H}omework4.pdf$.
[2] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.
[3] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
[4] NumPy. numpy.sum documentation. `https://numpy.org/doc/stable/reference/generated/numpy.sum.html`.
[5] NumPy. numpy.zeros documentation. `https://numpy.org/doc/stable/reference/generated/numpy.zeros.html`.
[6] PyTorch. torch.nn.module documentation. `https://pytorch.org/docs/stable/generated/torch.nn.Module.html`.
[7] PyTorch. torch.no_grad documentation. `https://pytorch.org/docs/stable/generated/torch.no_grad.html`.
[8] PyTorch. torch.optim.optimizer.zero_grad documentation. `https://pytorch.org/docs/stable/generated/torch.optim.Optimizer.zero_grad.html`.
[9] scikit-learn Developers. scikit-learn.train_test_split documentation. `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`.
[10] Seaborn Developers. Seaborn documentation. `https://seaborn.pydata.org/`.
[11] E. Shlizerman. Amath 482 lecture 19 notes, 2024.
[12] E. Shlizerman. Amath 482 practice3 pytorch optimization hyperparam, 2024.
[13] tqdm Developers. tqdm documentation. `https://tqdm.github.io/`.