AMATH 482/582: HOMEWORK 4

HAI VAN LE

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

Abstract. For this project, we are interested in training Fully Connected Deep Neural Networks (FCN) to classify the images in FashionMNIST data set which has been used extensively in training machine learning models. Here, we design an FCN model and evaluate the accuracy of different optimizers such as RMSProp, Adam and SGD and initializations such as Random Normal, Xavier Normal, Kaiming (He) Uniform.

1. Introduction and Overview

The FashionMNIST[1] database (Fashion Modified National Institute of Standards and Technology database) is a dataset of Zalando's article images—hosting a training set of 60,000 examples and a test set of 10,000 examples. Similar to the MNIST dataset that we examined in homework 3, each example in FashionMNIST is also a 28x28 grayscale image, associated with a label from 10 classes. Since MNIST has been extensively and overly used for training machine learning, FashionMNIST was published to replace its predecessor for benchmarking machine learning algorithms. According to the developers, FashionMNIST shares the same structure of training and testing splits as MNIST. Thus, we trained our classifiers using the training set while the test set is only used for validation/evaluation of our classifiers.

2. Theoretical Background

2.1. Deep Neural Network.

Machine learning is a complex subject that is best explained through visualization such as a deep neural network as seen in Figure 1[1]. According to IBM, a neural network makes decisions in a similar way to the human brain, by using processes that mimic the way biological neurons work together to identify phenomena, weigh options and arrive at conclusions. As illustrated in 1, a neural network consists of layers of nodes or artificial neurons which comprise an input layer, one or more hidden layers, and an output layer are the layers of nodes, or artificial neurons. Every node has a weight and threshold attached to it and is connected to other nodes. For a 28x28 grayscale image, there are $28 \times 28 = 784$ inputs with values between 0 and 255. Training data is essential for neural networks to learn and become more accurate over time. Once we have determined the baseline configuration or set of parameters that trains in reasonable time and results in reasonable training loss curve and testing accuracy, we can apply the neural network model to quickly classify and cluster data.

2.2. Optimizers.

In machine learning, optimizers are important in finetuning the parameters of a model. They are used to minimize the loss function and generally, one optimizer might be better for a model but worse for another so we need to select the best optimizer based on the training data and the model itself. Here in this project, we tested three different popular optimizers:

Date: March 9, 2024.

[1]https://www.ibm.com/topics/neural-networks

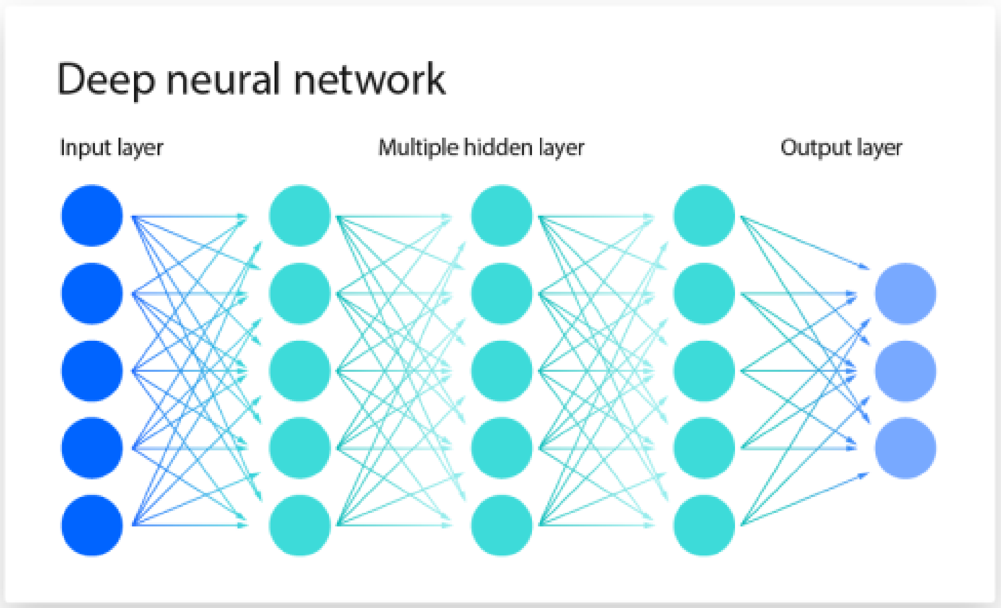

FIGURE 1. Deep Neural Network

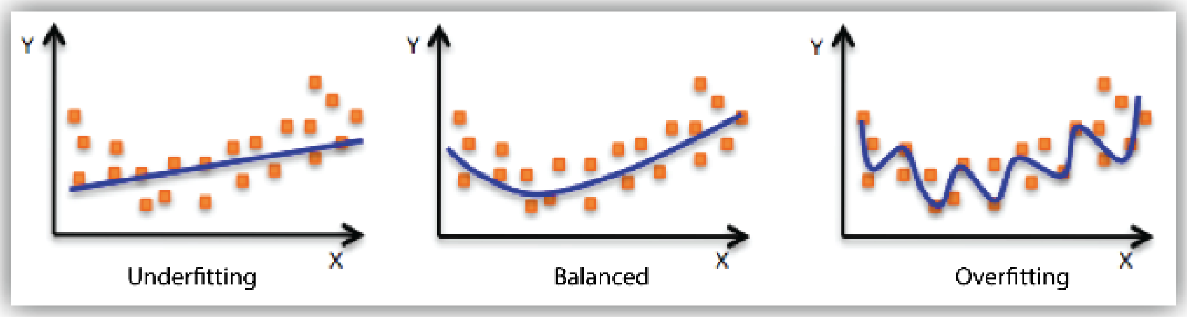

FIGURE 2. Overfitting vs. Underfitting

- Stochastic gradient descent SGD
- Adam
- Root Mean Squared Propagation RMSProp

2.3. **Overfitting and Underfitting.** A picture is worth a thousand words. As we can see from figure 2[2], an underfitted model performs poorly on the training data as it is unable to capture the relationship between the inputs and the targets. Conversely, when the model performs too well on the training data but does not explain the testing data well, it is overfitting since it is memorizing the data it has seen (aka training data) and is unable to generalize to unseen (aka testing data.)

2.4. **Dropout Regularization.** The intuition behind dropout regularization is that it will effectively spread the weights by randomly dropping units to ensure that no units are codepedent with

[2]https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

one another. The recommended probability for dropout is 0.2 to 0.5. To test the effect of dropout regularization, we can compare the training loss curves, validation accuracy curves, and test accuracies with and without dropout regularization. If the model with dropout regularization performs better in terms of validation accuracy and generalizes well to the test set without overfitting, it indicates that dropout regularization improves performance. Conversely, if the model without dropout regularization performs better or if there are not much significant changes, it suggests that dropout regularization might not be necessary for this specific model or configuration.

2.5. **Other parameters.** In this project, we also consider three initializations namely Random Normal, Xavier Normal, Kaiming (He) Uniform and Batch Normalization. All of these are available in PyTorch [3]. As its name suggests, initialization is the method of assigning an initial value to the inputs. Random Normal is an initializer that generates tensors with a normal distribution. Xavier Normal initialize the weights such that the variance of the activations are maintained across every layer [4]. Similarly, Kaiming Uniform initializes the weights and considers the non-linearity of activation functions, such as ReLU. The only normalization we tested, Batch Normalization, is to normalize the inputs by recentering and rescaling that maintains the mean output close to 0 and the output standard deviation close to 1.

3. Algorithm Implementation and Development

This project deploys different library interfaces such as torchvision to load the data, *matplotlib.pyplot* - a plotting library for the Python programming language and its numerical mathematics extension NumPy. torchvision has useful transformations (e.g. normalization) and loads data as Pytorch tensors. Specifically, the training data is split into Training and Validation datasets and loaded into DataLoader as batches. We defined a DataLoader that allows for easy iteration over the batches and samples within it. Batches are useful for setting when weights updates will occur. We then defined the FCN model and tried different set of parameters until we ended up with the baseline configuration. Then, we wanted to test different optimizers such as RMSProp, Adam and SGD with different learning rates (eg. 0.001; 0.01; 0.1) so we automated the process above by defining a function and a *for* loop that takes a list of learning rate and a dictionary of optimizer as parameters. Then, we modified the FCN model for each part of task 3. We also changed the baseline configuration using the best performer to reduce the amount of work my laptop has to do and prevent overheating.

4. Computational Results

For task 1-2, our baseline configuration is:

- Number of hidden layers: 2
- Neurons in each hidden layer: [400, 400]
- Learning rate: 5e-2
- Number of epochs: 15

The training loss and validation accuracy curve for task 2 is shown in Figure 3. For task 3, part a, to compare the performance of each optimizer, we logged the training loss, validation and test accuracy. The comparison is illustrated in Table 2. A plot is also provided in Figure 4. After part a, we can also update the baseline configuration to using Adam as the optimizer at the learning rate of 0.001.

In part b, we added *Dropout* regularization at a probability of 0.5, but this actually decreased the performance of all optimizers. Figure 5 shows testing accuracy for different variants. A possible explanation is that there is just not enough training data so adding a dropout regularization might

[3]https://pytorch.org/docs/stable/nn.init.html

[4]https://cs230.stanford.edu/section/4/

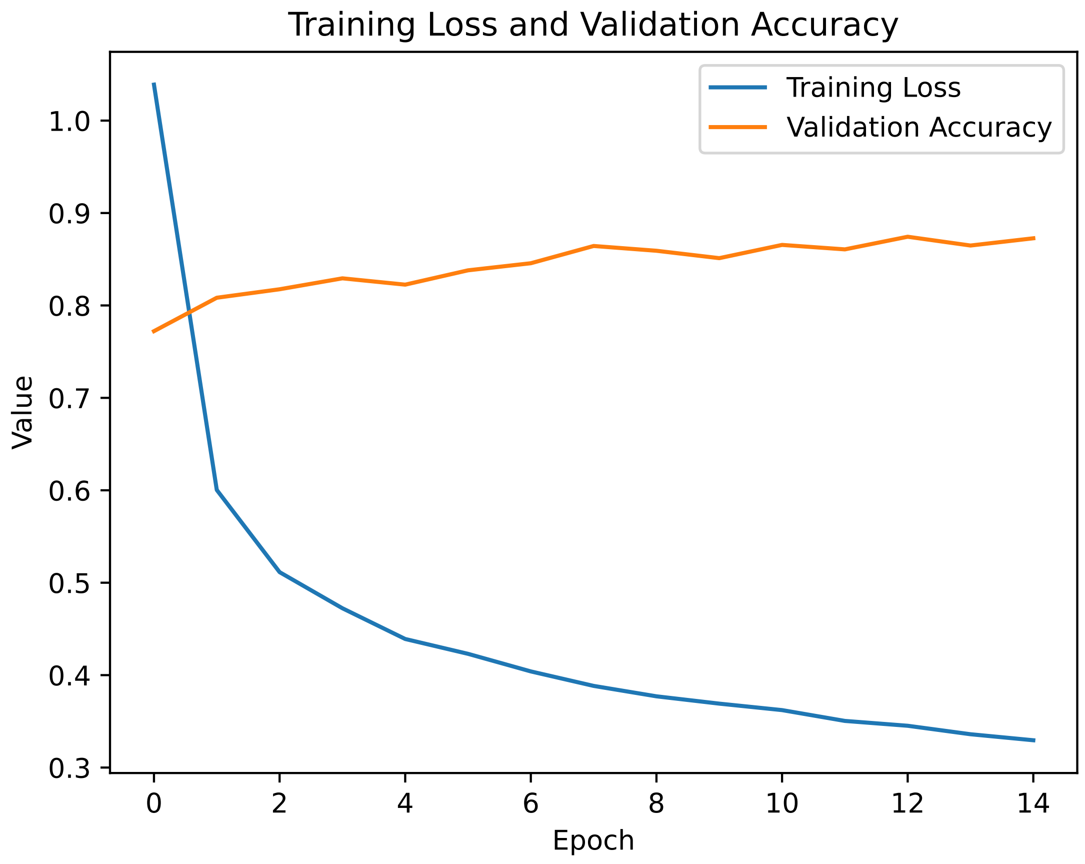

Figure 3. Training Loss vs. Validation Accuracy

Table 1. Optimizers Testing Accuracies

Optimizers and Learning Rate	**Test Accuracies**
SGD (lr=0.001)	67.21%
SGD (lr=0.010)	82.32%
SGD (lr=0.100)	86.58%
RMSProp (lr=0.001)	88.69%
RMSProp (lr=0.010)	84.72%
RMSProp (lr=0.100)	17.88%
Adam (lr=0.001)	88.87%
Adam (lr=0.010)	87.06%
Adam (lr=0.100)	19.61%

not be necessary for this specific task or configuration. Our best-performing configuration is still Adam with learning rate 0.001.

In part c, to save my laptop from burning of running too many different variants, we'd stick with Adam and learning rate of 0.001 but test out different initializations. The testing accuracies are exactly the same: 88.96% for each initialization which suggests a similar idea as in part a: lack of data. More specifically, the dataset may not contain enough complexity to require different initialization methods to learn distinct features. It seems that the data distribution is relatively

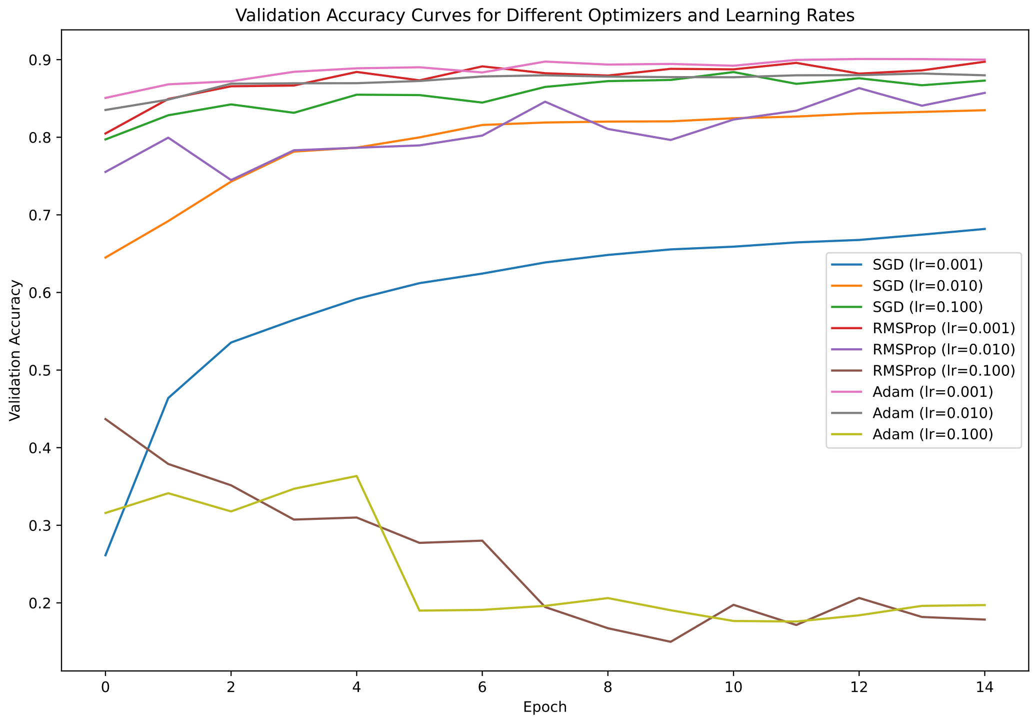

FIGURE 4. Comparison of Optimizers at Different Learning Rate

simple and can be adequately learned by the model regardless of the initialization method; thus, the differences in initialization may not have a significant impact on the final performance. So far, 88.96% is the best testing accuracy.

For part d, we wanted to test each initialization with Batch normalization. The best performer is the model with Adam optimizer, learning rate 0.001, initialization Xavier Normal, and Batch Normalization with a testing accuracy of 89.45%. All in all, we found that the best combination of parameters are:

- Number of hidden layers: 2
- Neurons in each hidden layer: [400, 400]
- Learning rate: 0.001
- Number of epochs: 15
- Optimizer: Adam
- Initialization: Xavier Normal
- Normalization: Batch Normalization

5. Summary and Conclusions

This project attempted to build an FCN model to differentiate images from the FashionMNIST dataset. This project tested different optimizers at different learning rate and found that Adam is the best performer for low learning rates. We also found that Xavier Normal is the best initialization for the model along with Batch Normalization though one limitation is that we only tested one normalization.

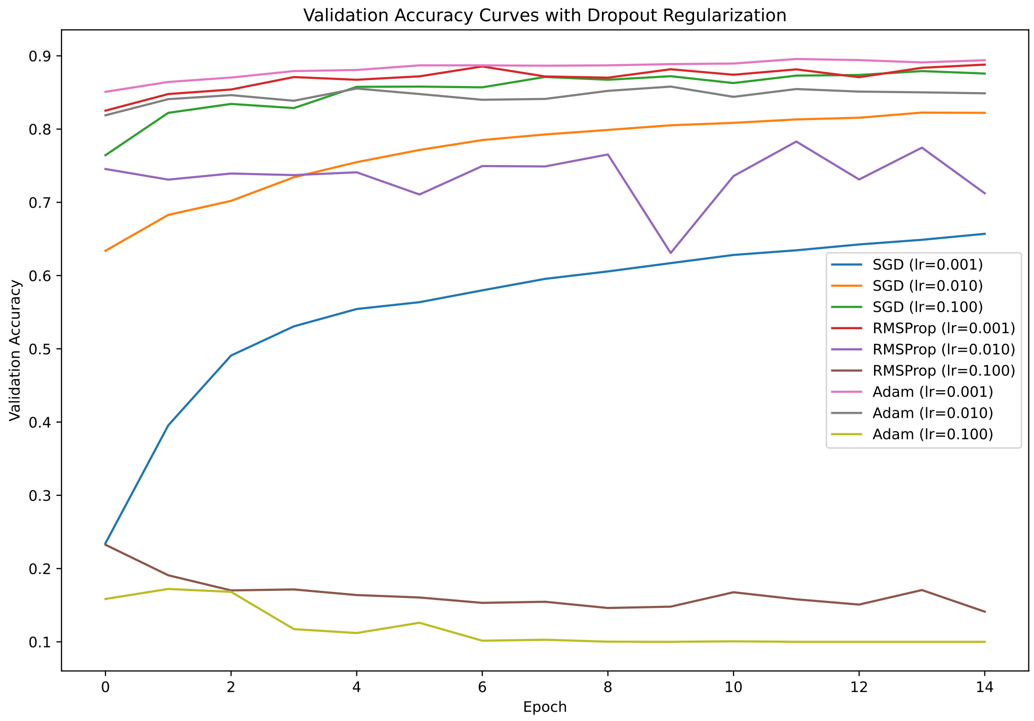

FIGURE 5. Dropout Regularization for Different Variants

TABLE 2. Optimizers Testing Accuracies With Dropout Regularization

Optimizers and Learning Rate	**Test Accuracies**
SGD (lr=0.001)	64.18%
SGD (lr=0.010)	81.27%
SGD (lr=0.100)	86.53%
RMSProp (lr=0.001)	87.58%
RMSProp (lr=0.010)	70.56%
RMSProp (lr=0.100)	14.22%
Adam (lr=0.001)	88.46%
Adam (lr=0.010)	83.81%
Adam (lr=0.100)	10.01%

ACKNOWLEDGEMENTS

The author is thankful to Prof. Eli for useful discussions about the SVD algorithm and Python libraries. She is also thankful to the TAs who answered all questions regarding class materials and resources.

REFERENCES

[1] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.