AMATH 482/582: HOME WORK 5

XINYUE LYU

Amazing Department, University of Washington, Seattle, WA
xlyu6@uw.edu

Abstract. This report explores the classification performance of Fully Connected Networks (FCNs) and Convolutional Neural Networks (CNNs) on the FashionMNIST dataset under constrained weight budgets. I design and evaluate FCN models with approximately 100K, 50K, and 200K parameters, followed by CNN models with 100K, 50K, 20K, and 10K parameters. The study examines the impact of model size on classification accuracy, training time, and efficiency. The results highlight the advantages of CNNs in image classification tasks and provide insights into the trade-offs between model complexity, accuracy, and computational efficiency.

1. Introduction and Overview

Fully Connected Networks (FCNs) and Convolutional Neural Networks (CNNs) play a crucial role in image classification. This report examines their performance under different weight numbers using the FashionMNIST dataset. I compare FCNs with 50K, 100K, and 200K parameters and CNNs with 10K, 20K, 50K, and 100K parameters. The study evaluates test performance and training efficiency. By analyzing these models, we highlight the trade-offs between accuracy and complexity, obtaining insight into designing efficient deep learning models for image classification.

2. Theoretical Background

a. **Neural networks and ReLU**: This model might be inspired by how neurons work in our brain. It consists of neurons(or perceptrons) as input and output. Between the input and output layer, there are hidden layers that are linear combinations of neurons in the former layer with some weights and bias.

Putting the different results of the linear combination to an activation function(add non-linearity to the model), we get neurons that make another layer. In this assignment, I used ReLU as the activation function for each layer. This activation function outputs the input if the input is positive, and zero otherwise.

Fully connected neural network(FCN) refers to the situation that every neuron in one layer is connected to every neuron in the next layer.

The Convolutional Neural Network (CNN) contains "convolutional layers", extracting features, and "pooling layers", reducing spatial dimensions.

b. **Cross Entropy loss**: It works as a loss function to calculate the differences between predictions generated from the model and the real data. Cross Entropy loss measures the difference between the predicted probability distribution and the true (or actual) distribution of labels. It is calculated as:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i))$$

Date: March 21, 2025.

where N is the number of instances, y_i is the true label for instance i, and p_i is predicted probability by the model.

c. **Gradient Descent**: A gradient decent algorithm helps people find the best performance of a model by minimizing the loss function. In this assignment, I performed Stochastic Gradient Descent(SGD), Root Mean Square Propagation(RMSProp), and Adaptive Moment Estimation(Adam) optimizer with different learning rate (the step size during training).

d. **Overfitting/underfitting**: If a model fits very well to the training data and can not generalize for another unseen data, it is overfitting. If a model is too simple, it is underfitting, resulting in poor performance in training and testing data.

Dropout regularization randomly sets some nodes to zero with probability p during each forward pass. It assists in reducing overfitting.

e. **Kaiming(He) Initialization**: It is a technique for initializing the weights of neural networks. The Kaiming initialization is calculated as a random number with a Gaussian probability distribution with a mean of 0.0 and a standard deviation of $\sqrt{(\frac{2}{n})}$, where n is the number of inputs to the node[**?**].

f. **Batch Normalization**: This is a technique that normalizes activations within each mini-batch. It adjusts and scales activations using:

$$\hat{x} = \frac{x - \mu}{\sigma}$$
$$y = \gamma \hat{x} + \beta$$

where μ, σ are the mean and standard deviation of the batch and γ, β are learnable parameters.

3. Algorithm Implementation and Development

In this assignment, I use the FCN model from the last homework and change the number of neurons. Additionally, I also build CNN model and perform hyper-parameters tuning with lists of parameters. Then I utilize the CNN that performs best for 100K weights to access how the model performs with 50K, 20K, and 10K weights. In order to achieve this implementation, I used **PyTorch**, providing efficient tensor computations and automatic differentiation. The **torch.nn** module facilitates defining network layers, applying Dropout, and integrating Batch Normalization. Beside, **NumPy** aids in numerical operations.

4. Computational Results

Number of Weights of FCN	Test Accuracy	Time
50K	87.69%	1m54s
100K	88.15%	2m2s
200K	88%	2m1s

Table 1. This table shows the test accuracy and training time of FCN models with different numbers of weights to train.

I have tuned the FCN from the last assignment. My FCN model with 100K weights gives a test accuracy of 88.15%. Based on this model architecture, the test accuracy changed when change the number of weights to 50K and 200K. For a FCN with 50K weights, the test accuracy is 87.69%, and for a FCN with 200K weights, the test accuracy is 88%, as shown on Table 1. The only

Optimizer	0.01 LearningRate	0.005 LearningRate	0.001 LearningRate
SGD	0.9155	0.9171	0.8966
Adam	0.9058	0.9093	0.8932

Table 2. This table shows the test accuracy(between zero and one) for two optimizers with three different learning rates, aiming to find the most suitable combination of parameters for the CNN model.

Number of Weights of CNN	Test Accuracy	Time
10K	78.75%	2m29s
20K	79.38%	2m35s
50K	89.88%	2m45s
100K	91.71%	NA

Table 3. This table shows the test accuracy and training time for CNN models with different number of weights. The training time is not calculated since it is in the process of hyper-parameter tuning process.

difference of these FCN models is the number of weights. All other parameters such as batch size and learning rate are the same. As the number of weights increases, the test accuracy increases at first and then decreases. Doubling the number of weights, the test accuracy doesn't change a lot, decreasing about 0.1%. It is not efficient to use 200K weights. The decreasing trend might infer to the tendency of overfitting. In addition, The time generally increases as the more weights to be trained.

Then, by hyperparameter tuning, results shown on Table 2, I found that the CNN model with 0.005 learning rate and SGD optimizer performs best for training 100K weights. This combination of optimizer and learning rate generates 91.71% test accuracy. Also, all test accuracy generated by the combinations of different optimizers and learning rates are higher than the test accuracy rate of FCN with the same number of weights(100K). When the number of weights to be trained be 20K and 10K, the test accuracy is below 80%. This might because of underfitting.

Comparing Table 1 and Table 3, I find that the test accuracy for CNN with 50K weights (89.88%) is higher than FCN with 50K weights (87.69%). The test accuracy of FCN with 100K weights is still around 2% away from the test accuracy of CNN with only 50K weights.

5. Summary and Conclusions

Generally, the CNN models are more efficient than FCN models. Comparing the two types of models, CNN models need less weights to achieve a high test accuracy. The time for these two types of models is not different much, within 1 minute.

Acknowledgements

The theoretical background is mostly the same as HW4. The author is thankful to Dr.Frank for instructing the concepts of convolutional neural network, gradient descent, and the coding techniques along with the textbook[1].

References

[1] J. N. Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data.* OUP Oxford, 2013.