AMATH 582: HOMEWORK 5

HUNTER LYBBERT

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

Abstract. In this report we compare and analyze the merits of Convolutional Neural Networks (CNNs) against our previous work with the Fully Connected Network (FCN). An FCN with the best hyperparameter configuration from report 4 is trained with multiple different total number of weights. Additionally, hyper-parameter tuning is conducted for the CNN. Finally different CNNs are trained with different total numbers of weights but with the best hyperparameter configuration from the previous step. An analysis of the costs and benefits of these two models is presented, primarily focused on the number of weights, training time, and model accuracy on the test set.

1. Introduction and Overview

In this report we continue our study of deep learning, the field of neural network based model architecture.

Again, our setup is a common supervised learning problem, given a collection of N data points with labels in classification or target values in a regression setting

$$\{(\boldsymbol{x_0}, y_0), (\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_{N-1}}, y_{N-1})\}.$$

The data is denoted as a matrix X and a vector of target values or class labels $\boldsymbol{y}$. We then are looking for a function f which takes in the training data and most accurately predicts the target values or class labels, written in optimization form we are looking for the following

$$f_{MLE} = \underset{f}{\operatorname{argmin}} \frac{1}{2\sigma^2} ||f(X) - \boldsymbol{y}||_2^2 \tag{1}$$

where σ^2 is the variance of the normally distributed error terms $\epsilon \sim \mathcal{N}(0, \sigma^2)$ defined by $\epsilon_i = y_i - f(x_i)$ So said another way we are trying to minimize our errors in the classification task. The specific class of functions f to be considered to solve the problem is a neural network, specifically we evaluate and compare FCNs to CNNs. We will treat the theoretical background of these methods in the next section 2.

Before proceeding, we would like to acknowledge the critical use of the following packages in our analysis. Namely, Matplotlib was used to create all plots and animations [2]. Addiitonally, PyTorch [1] was crucial for easily implementing the desired architecture while Ray Tune [3] was used to help with hyper parameter tuning.

2. Theoretical Background

For a general theoretical background on neural networks and basic hyperparameters please refer to the report from homework 4. We presented our theoretical background clearly there. Instead of repeating the information about gradient descent, back propogation, optimizers, learning rates, and dropout etc. in this report again, we will focus on the unique architectural considerations of a convolutional neural network.

Date: March 19, 2025.

TODO: What is a convolution?

TODO: Present a visual representation of a convolutional layer

TODO: Present an example architecture, perhaps the one you used for your best CNN

2.1. Architecture. TODO: maybe reuse this section

We will discuss what our own experiments revealed about weight initialization in our computational results section 4.

3. Algorithm Implementation and Development

In order to train a CNN we began by creating a model class using PyTorch's [1] many built in methods and classes. This time we manually experimented with different number of convolutional layers until we settled in on our basic architecture (3 convolutional layers and a fully connected layer).

I did not try to use Ray Tune [3] again, but rather I generalized the model training python class I created in homework 4 to be able to apply it to both the CNN and FCN. This generalizing of the python class led me to relearning a bit about python class inheritance and was a good exercise in trying to generalize my code. This generalized class still managed the experiments for me and recorded results, total number of weights, training time, and other parameter settings to a *json* file. This allowed me to train many models over night and analyze the results the next day and refer back to them in the future as well. It is all easy to use and very reproducible.

TODO: Describe the architecture you ended up settling on (generally)

Due to the large number of hyper parameters it is an overwhelming amount of information to specify in each plot for each loss or accuracy curve. Therefore, I have labeled plots with experiment numbers and mean accuracy across test set. For select models I will detail their exact configuration in the next section 4, while others can be looked up in the repo directly on GitHub.

4. Computational Results

We will primarily let the loss, accuracy curves, and hyperparameter configuration tables speak for themselves. We have included a small sample of the results of our parameter tuning. Please, see Figures 1 and 2 for visualizations of how batch normalization, weight initialization, optimizer choice, and learning rates effected the loss and accuracy curves throughout the training process. After training our 300 plus models and inspecting which configurations produced a model with above 90% test accuracy, we picked the configuration in 1. We trained a model with this configuration for 5 repeated trials and report the test accuracy mean and standard deviation across trials. We also repeated this process for the MNIST Digit classification problem from homework 3. Our best model configuration aligns well with our theoretical background expectations for the optimizer, learning rate, dropout rates, batch normalization, and weight initialization. A dropout rate around 0.5 was best. Adam with a low learning rate barely edged out SGD with a larger of a learning rate. We don't report on RMSprop here (due to space), however, it performed similar to Adam.

We also applied sklearn's K-nearest neighbors to this dataset and had a test accuracy of 0.8554 (with number of neighbors set to 5). Varying the number of neighbors degraded the model slightly.

Ep.	**Arch**	**W. Init**	**B. Norm**	**Drop**	**Optim**	**lr**	$\boldsymbol{\mu}$ **(Test Acc)**	$\boldsymbol{\sigma}$ **(Test Acc)**
50	[1024, 256]	Kaiming	True	[0.5, 0.5]	Adam	0.001	0.9030 ★	0.0021

Table 1. This was the configuration that was best from the last report in hw 04. Therefore it was the base setup for the FCN's trained in this hw 05, however, with the necessary adjustments due to weight count constraints.

ID	Model	Weights	Time	Ep.	Optim	lr	W. Init	μ (Test Acc)	σ (Test Acc)
49	CNN (100k)	108,814	0:06:15	50	Adam	0.001	Xavier	0.9236 ★	0.0167
56	CNN (10k)	9,980	0:04:02	50	Adam	0.001	Xavier	0.9002	0.0229
57	CNN (20k)	20,071	0:04:15	50	Adam	0.001	Xavier	0.9091	0.0151
58	CNN (50k)	50,392	0:05:19	50	Adam	0.001	Xavier	0.9152	0.0188
59	FCN (50k)	52,842	0:02:30	50	Adam	0.001	Kaiming	0.8802	0.0158
60	FCN (100k)	109,770	0:02:32	50	Adam	0.001	Kaiming	0.8889	0.0211
61	FCN (200k)	202,974	0:02:32	50	Adam	0.001	Kaiming	0.8965	0.0188

TABLE 2. After performing Hyperparameter tuning to determine the best settings for out CNNs, we trained models with 100k, 50k, 20k, and 10k weights. Using the configuration from Table 1 (best from homework 4) we trained 3 FCN's each with 200k, 100k, and 50k weights in each model. Notice that training time increases very rapidly for the CNN as the number of parameters increases, while the FCN training time is almost constant (at least at the scale and for the experiments we conducted. Finally, notice how even the smallest CNN outperforms the FCN on the test set accuracy, let alone how high the test accuracy is with the 100k weights CNN.

TODO: Talk about hyperparameter tuning for the CNN TODO: Talk about the need for regularization and mitigating overfitting!

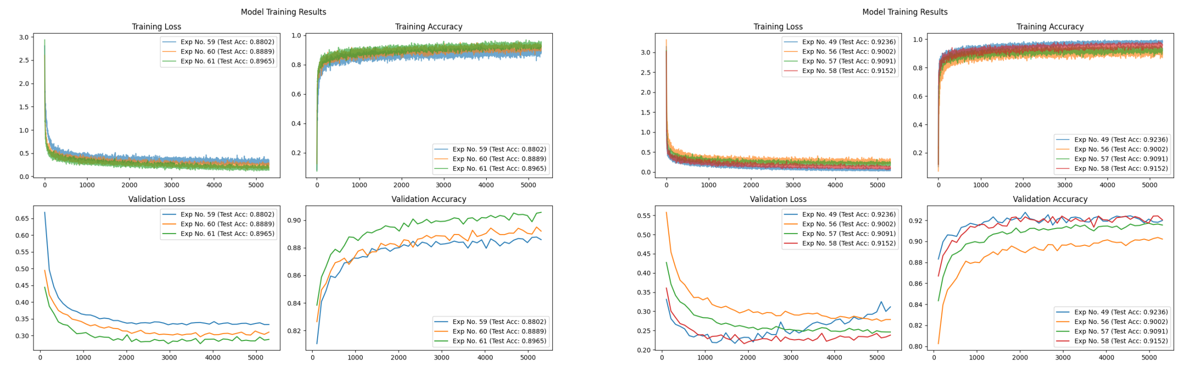

FIGURE 1. On the left we have the 3 different FCNs which are all the same except different total number of weights, compare to Table 2. On the right are the 4 different CNNs trained with the same parameter settings except they vary the total number of weights, compare to Table 2.

ID	Ep.	Arch	W. Init	B. Norm	Drop	Optim	lr	μ (Test Acc)	σ (Test Acc)
111	30	[1024, 256]	Xavier	True	[0.3, 0.5]	Adam	0.001	0.8967	0.0222
123	30	[1024, 256]	Random N.	True	[0.3, 0.2]	Adam	0.001	0.8807	0.0202
135	30	[1024, 256]	Kaiming	True	[0.1, 0.1]	Adam	0.001	0.8912	0.0175

TABLE 3. **TODO: Convert this into something to talk about the hyperparameter tuning of the CNNs**

5. SUMMARY AND CONCLUSIONS

TODO: Conclude which model you would use when based on needing it to train quickly, visual data, etc. The method we settled on limited how we could convey the results

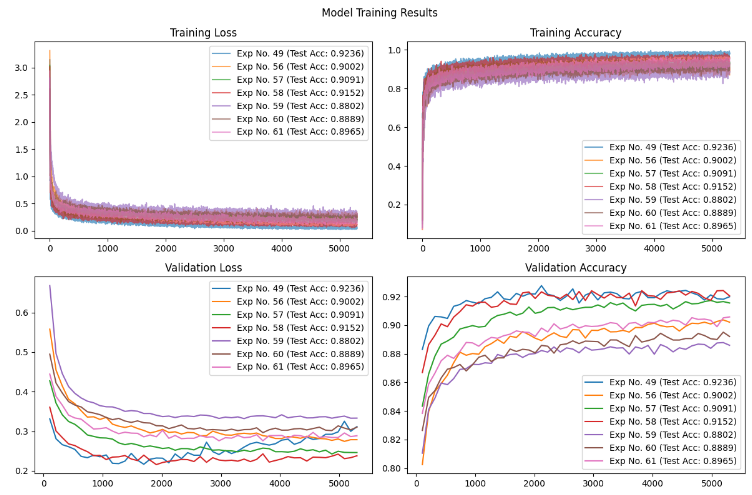

FIGURE 2. This is a visualization of all of all models in Table 2. They were visualized separately in Figure 1.

here in the report, though it recorded thorough information for later reference on GitHub This was an excellent introductory experience into building, training, and evaluating a neural network.

Acknowledgements

The author is thankful to Jaxon Tuggle for offering regular feedback and counsel when interpreting results and clarifying the implications and merits throughout the hyperparameter tuning process. We would also like to thank Professor Eli Shlizerman for carefully instructing us in class.

References

[1] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, M. Suo, P. Tillet, E. Wang, X. Wang, W. Wen, S. Zhang, X. Zhao, K. Zhou, R. Zou, A. Mathews, G. Chanan, P. Wu, and S. Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024.

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.