Model Design For Deep Learning In Practice (Project 3)

Vaibhav Chandra
vc2530@columbia.edu

DongRyong Choi
dc3454@columbia.edu

Erica Li
el3018@columbia.edu

Rohan Uppuluri
rku2101@columbia.edu

Abstract

Deep learning in practice can be complicated due to the existence of design choices that go into a neural network model. These choices include network architecture, optimization, initialization, and regularization, and they can have a substantial impact on the performance of trained neural network models. This begs the question of how robust the models are to these design choices. Here, we investigate how these choices can affect out-of-sample performance and whether good choices for one dataset translate into good choices for another dataset.

1 Introduction

The success of deep learning in speech recognition, image captioning, and other complex feature engineering makes it an attractive approach to analyzing data. Design choices of neural network models can substantially affect its performance. This project investigates how these choices affect out-of-sample performance measured by sparse cross entropy accuracy through four segments: (i) network architecture, (ii) optimization, (iii) initialization, and (iv) regularization. To evaluate the robustness of models relative to hyperparameter choices, we conduct this analysis on two different datasets and examine whether the best performing models for one dataset correspond to the best performing models for the other.

1.1 Data

We first analyzed the MNIST digit dataset and then proceeded to analyze the fashion-MNIST dataset.

The MNIST dataset contains 60,000 training images and 10,000 test images of size 28x28 (784 total features) with 10 possible classes representing digits 0-9. We did an 80-20 training/validation split.

Likewise, the fashion-MNIST dataset has 60,000 training images and 10,000 test images of size 28x28 (784 total features) with 10 possible classes. However, these classes represent articles of clothing. We did an 80-20 training/validation split here as well.

2 Methods

2.1 Starting Point

We started by training an unregularized MLP with two hidden layers of 16 hidden units and ReLU activations default optimization and initialization settings as the baseline model configuration for both the MNIST and Fashion MNIST datasets. Note that default optimization and initialization hyperparameters from Keras were used in this case (i.e. Adam with learning rate 0.001 and Glorot Uniform initialization). We refer to the colab notebook submitted for all of the relevant plots of validation accuracy vs. number of epochs since there are numerous plots that could be included in this project.

2.2 Network Architecture

The natural progression for an image classification problem is to use convolutional layers. From the baseline model, we incrementally added more complexity through the addition of layers [1], considering several different candidate network architectures. After trying different orders and combinations of Conv2D (c), Dense (f), Dropout (d), and Maxpooling2D (p), we arrived at three models with the highest validation accuracy: cpcpdff, cfpcfpdff, and ccpdnff. For example, we tried multiplicative of cp pairs as well as cfp pairs and then visualized the validation accuracy curves to decide on the optimal one.

We note that a dropout layer is a regularizer, but we have included it in the models since our empirical experiments have shown it increases accuracy and so we wanted to extract the composition of layers that work well with a dropout layer. With the last model, ccpdnff, we decided to add batch normalization (n) in this step instead of in regularization to study the effects of tuning hyperparameters while taking the batch normalization into account versus tuning without it.

For the Fashion MNIST dataset, we followed the same process and acquired two new model architectures: cfpdndff and cpdnfndff. We kept the ccpdnff architecture from MNIST since it performed well on Fashion MNIST in terms of validation accuracy.

2.3 Optimization

To get us to our goal for optimization, we have implemented a Keras tuner to search through parameter settings that are more heuristic based (e.g. kernel size, number of filters, pooling size, activation function, batch size, number of epochs). We investigated the optimizers SGD, Adam, Nadam, and RMSprop with the choices of learning rate {1e-2,1e-3,1e-4}. Of these hyperparameter choices, we found that our architectures were most sensitive to the choice of optimizer and the choice of learning rate.

We chose SGD, Adam, and RMSprop as they are the most popular choices of optimizers in practice. Nadam is a modified version of Adam that implements Nestorov momentum, which first performs an average of past steps and moves in that direction first before computing the gradient in the new position. This helps prevent the gradient from being stuck in a local minimum. We chose to experiment with Nadam since Adam is often the best first choice of an optimizer, so we wanted to see how beneficial the Nestorov momentum modification would be [3]. The modification did not perform as well as regular Adam on any of our MNIST models whereas one of the models in Fashion MNIST did better with Nadam instead of Adam. Ultimately, we chose RMSprop and SGD for the MNIST models, and we chose Nadam and Adam for the Fashion MNIST models.

2.4 Initialization

Borrowing from empirical results and common machine learning practice [2], we investigated several initialization strategies. The following initialization/activation pairs worked particularly well: Glorot Normal/ReLU, He/ReLU, and variance scaling/tanh activation. Glorot Normal initialization usually works well because it helps in avoiding the weights to either diminish or explode by allowing for scaling of the weight distribution on a layer-by-layer basis. It essentially draws form a normal distribution with a centered mean and standard deviation scaled to the layer's number of input and output neurons — thus keeping them in a reasonable range across the entire network. He Normal initialization also adapts the same idea of the balancing of the variance of the activation as does Glorot, but is aimed to work optimally with ReLU to handle the non-differentiability of the ReLU activation function at $x = 0$ [4]. Variance scaling is a generalization of these initialization methods, allowing us more flexibility and customization with weight initialization.

2.5 Regularization

Our regularization methodology employs batch normalization, layer weight regularization, and data augmentation. Our experiments demonstrated that batch normalization layers yield the best performance when deployed in these sequences: (i) flatten/batch normalization/output layer, and (ii) maxpooling/batch normalization/conv2d layer. A Keras tuner was used to find the best hyperparameters for layer weight regularizers (L1, L2), but it did not improve on the validation accuracy. Data augmentation was found to prevent the model from overfitting the training data and yielded better validation accuracy for one of the Fashion MNIST models. Data augmentation involved trying different settings for rotation range, zoom range, shear range, width shift range, height shift range, and vertical flip. We incrementally added one new adjustment in each run and combined adjustments that seemed to improve validation accuracy. We also made sure that our augmentation preserved the images so that the objects would still be visible.

None of the above methods except for batch normalization and dropout seemed to improve any of the models for MNIST and Fashion MNIST. However, for one of our models in Fashion MNIST, namely ccpdnff (Adam), data augmentation also helped. Also, for Fashion MNIST, reducing learning rate callbacks seemed to help stabilize the validation accuracy over the epochs.

3 Results

We achieved test accuracies for MNIST and Fashion MNIST with the new models achieved after going through the four criterion steps separated by the line as follows:

Table 1: Model Test Accuracy

Models	MNIST(%)	Fashion-MNIST(%)
Baseline	95.21	85.89
cpcpdnfff (RMS)	99.33	91.29
cfpncfpdnff (RMS)	99.23	91.12
ccpdnff (SGD)	**99.35**	**92.23**
ccpdnff (Adam)	–	**92.84**
cfpdndff (Nadam)	–	92.36
cpdnfndff (RMS)	–	92.52

Note that the best performing models are shown above. They were chosen based on highest sparse categorical accuracies on the validation set while we tried each model across different settings. Sparse categorical accuracy is a good metric for success in this case because our goal is to evaluate which model we create is best at predicting a target class (digits / fashion articles).

3.1 MNIST Dataset

After going through the four criteria laid out in the previous section, we found that the best performing models for MNIST were cpcpdnfff (RMS), cfpncfpdnff (RMS), and ccpdnff (SGD) described as follows. Table 1 summarizes our test accuracies, which range from 99.2-99.4%.

- **cpcpdnfff (RMS)**: The network architecture as described by the name of this model involves 2x[Conv2D(filters=96, kernel_size=5), MaxPooling2D], 1x[Dropout(0.1)], 1x[BatchNormalization], and 2x[Dense(units=64)] layers (excluding output layer). RMSprop with learning rate 0.0001 was used as the optimizer. The specifications for each layer and for the learner have been optimized with the Keras Tuner. Initialization / activation pairs are Glorot Normal with ReLU. Regularization techniques employed were batch normalization and callback for early stopping (patience=10). Data augmentation (including but not limited to width shift, height shift, zoom, and ZCA) actually decreased the validation accuracy even when the number of epochs and batch sizes were adjusted, so we did not proceed with training this model with augmented data.
- **cfpncfpdnff (RMS)**: The network architecture here involves 2x[Conv2D(filters=128, kernel_size=5), Dense(units=36), MaxPooling2D(pool_size=2)] separated by a 1x[BatchNormalization], followed by 1x[Dropout(0.1)], 1x[BatchNormalization], and 1x[Dense(units=36)] layers (excluding output layer). Initialization / activation pairs are He Normal initialization with ReLU. This model also worked best when RMSProp with learning rate 0.0001 was used as the optimizer, and batch normalization and callback for early stopping (patience=10) were employed for regularization. Specifications for each layer and for the learner have been optimized with the Keras Tuner.
- **ccpdnff (SGD)**: The network architecture here consists of the following. We used 1x[Conv2D(filters=32,kernel_size=5], 1x[Conv2D(filters=128,kernel_size=5], MaxPooling2D[(pool_size=2)], 1x[Dropout(0.1)], 1x[BatchNormalization], and 1x[Dense(units=128)] layer (excluding output layer). SGD with learning rate 0.1 was the best optimizer choice, and we found that the best initialization/activation pair for this model was variance scaling with relu activation. For regularization we included dropout and batch normalization, and we used early stopping with patience=10.

To dive deeper into diagnosing our models, we analyze the test predictions against the actual labels. The illustrations below show examples that our best model ccpdnff (SGD) predicted correctly (Figure 1) and incorrectly (Figure 2). It turns out that our model predicts most digits quite well (99%+ precision and 99% recall for all digits except 5). In some cases, the test images were difficult for human eyes to distinguish as well, for example True 6 / Pred 0 and True 2 / Pred 7. We believe that the digit 5 has the most similarities with other digits so this is the category that had the worst prediction accuracy. Overall, we are content with the results of our models in this section for MNIST given they all yield above 99.2% test accuracy.

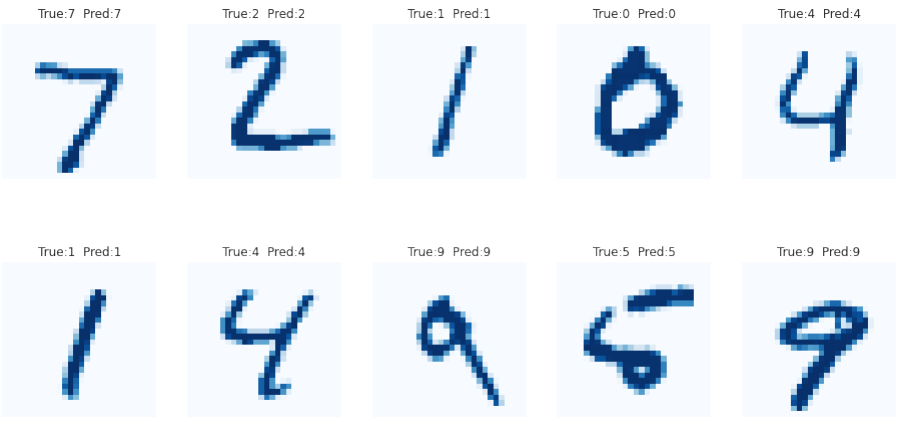

Figure 1: Test images that ccpdnff (SGD) predicted correctly

Figure 2: Test images that ccpdnff (SGD) predicted incorrectly

3.2 Fashion-MNIST Dataset

We first trained our three best models mentioned in Section 3.1 on the Fashion MNIST data. After that, we kept one of the architectures, ccpdnff and made two new architectures, cfpdndff and cpdnfndff. With these three architectures, we found that the best performing models for Fashion MNIST were ccpdnff (Adam), cfpdndff (Nadam), and cpdnfndff (RMS) described as follows. Table 1 summarizes our test accuracies, which range from 92.3-92.8%.

- **ccpdnff (Adam)**: The network architecture here consists of the following. We used 1x[Conv2D(filters=32,kernel_size=3], 1x[Conv2D(filters=128,kernel_size=5], MaxPooling2D[(pool_size=2)], 1x[Dropout(0.1)], 1x[BatchNormalization], and 1x[Dense(units=128)] layer (excluding output layer). Adam with learning rate 0.0001 was the best choice of optimizer. We found that the default initialization (Glorot uniform) along with relu activation for the Conv2D layers and sigmoid activation for the dense layer performed best. For regularization we included dropout and batch normalization, used early stopping with patience=10, and performed data augmentation on the Fashion MNIST dataset to improve performance. The batch size was 32, and the number of epochs was 30.

- **cfpdndff (Nadam)**: The network architecture here consists of the following. We used 1x[Conv2D(filters=112,kernel_size=3], 1x[Dense(units=112)], MaxPooling2D[(pool_size=2)], 1x[Dropout(0.1)], 1x[BatchNormalization], 1x[Dropout(0.1)], and 1x[Dense(units=112)] layer (excluding output layer). Nadam with learning rate 0.0001 was the best choice of optimizer with the reducing learning rate with the factor of 0.9. We found that the default initialization setting with ReLU was sufficient and data augmentation did not help improve the performance. The batch size was 256, and the number of epochs was 50.

- **cpdnfndff (RMS)**: The network architecture here consists of the following. We used 1x[Conv2D(filters=112,kernel_size=5], MaxPooling2D[(pool_size=2)], 1x[Dropout(0.1)], 1x[BatchNormalization], and 1x[Dense(units=112)], 1x[BatchNormalization], 1x[Dropout(0.1)], 1x[Dense(units=112)], layer (excluding output layer). RMSprop with learning rate 0.0001 was the best choice of optimizer with reducing learning rate factor of 0.9. We found that the default initialization setting with ReLU was sufficient and data augmentation did not help improve the performance. The batch size was 256, and the number of epochs was 50.

To dive deeper into diagnosing our models, we analyze the test predictions against the actual labels. The illustrations below show examples that our best model ccpdnff (Adam) predicted correctly (Figure 3) and incorrectly (Figure 4). It turns out that our model predicts some categories quite well (99% precision and 98% recall for "Trouser" and "Sandal") but fails to distinguish between other categories like "Shirt" vs. "T-shirt/top" (precision and recall in the 80%'s). Upon further examination,

it is quite difficult even for human eyes to categorize an article of clothing as a "T-shirt/top" or just a "Shirt", so we are content with the results of our models in this section for Fashion-MNIST.

Figure 3: Test images that ccpdnff (Adam) predicted correctly

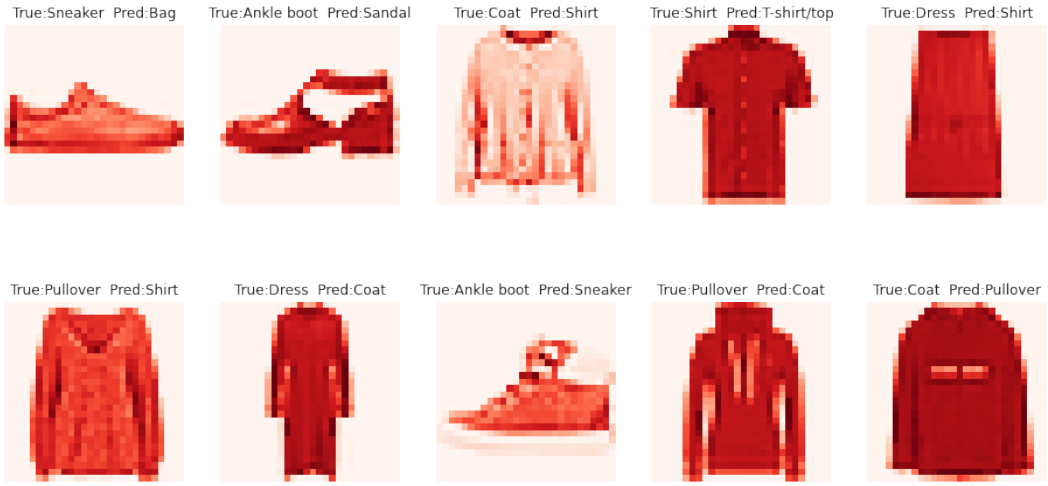

Figure 4: Test images that ccpdnff (Adam) predicted incorrectly

4 Discussion

We observed that MNIST models yielded much higher test accuracy than the Fashion MNIST models mainly because the Fashion MNIST data has more complicated features. An area of further work would be improve the categories of clothing that our models predicted relatively poorly, namely "Shirt" and "T-shirt/top".

In terms of robustness, the best models for MNIST did perform well on Fashion MNIST in terms out of sample accuracy (all of them over 91%) given that most test accuracies we could find for other models online are in the range of 90-95% for Fashion MNIST. We were able to improve 0.5-1%+ from the best models for MNIST by going through those 4 criterion again with the Fashion MNIST data to get test accuracies of over 92%.

For some empirical insights, we found that ReLU as an activation function for both convolutional and fully connected layers seemed to work out well in addition to using a dropout layer after a flatten layer. The position of batch normalization layer and the choice of optimizer function depended on each case. We also found that it was better to include batch normalization layers before finding optimal hyperparameters than first finding optimal hyperparameters and then adding batch normalization layers later on. Also, the optimal batch size for MNIST was relatively smaller than that of Fashion MNIST (30 for all 3 MNIST models vs. 256 for 2 of the Fashion MNIST models). We believe allowing for less noisy weight updates with a larger batch size improved the ability for our models to learn the additional complexity in the Fashion MNIST dataset.

Another opportunity for further work is testing the robustness of our best models on other kinds of datasets with more complex features than MNIST and Fashion MNIST, such as the SVHN dataset. If our models perform well, then we can try to understand what the model has learned that is relevant to the new dataset which causes it to perform well. If it does not perform well, we can study what needs to be changed to get better performance.

Finally, as we tried to rationalize and systematize the process of those four steps - architecture, optimization, initialization, and regularization - we found that there is still a lot of room for empiricism that is absolutely necessary to achieve the best (if we knew) performing models. Even with the Keras Tuner, as we observed each combination of parameters, we narrowed down the choices of parameters in the next grid search to save time and so on. We cannot completely automate this process of finding the best model in the most efficient way and hence we absolutely need our domain knowledge of each choice in building the model to achieve this task.

5 References

[1] Bettilyon, T. (2018) For-loop approach to finding the optimal number of dense layers in network architecture. *How to classify MNIST digits with different neural network architectures*

[2] Mendels, G. (2019) Initialization and activation combinations used in practice by Machine Learning Engineers. *Part 2: Selecting the right weight initialization for your deep neural network*

[3] Ruder, S. (2016) An overview of gradient descent optimization algorithms. *Nadam*

[[4] He et al. (2015) He initialization. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*