Basic Neural Networks on MNIST Dataset

Changsoo Jung, Vishwajeet Bhosale, Amanda Cohen
Colorado State University

Introduction

MNIST dataset is one of the most famous datasets to learn the basics of neural networks. It has 70k images of handwritten digits, each of dimension 28x28. In this assignment, we used Tensorflow's beginner.ipynb notebook as a starting point and performed various experiments to analyze performance of trained models. We analyzed failure cases of the basic model first, then we trained models with various epochs, widths and depths. Also, we examined the effect of preprocessing the images with Sobel edge detection.

Failure Cases

We analyzed the failure cases using OpenCV since it supports putText() function and hconcat() function that help to add text and merge multiple images easily. We expanded each image to add a text legend for checking a correct label and the basic model's prediction.

As can be seen from figure 1 some of the digits appear ambiguous even for human vision. For example, the first image looks like a number six, but the actual label is a number five. This handwritten image is confusing since the number five and the number six have similar features such as an arc at the bottom and a curved line at the top. Guessing a number using similarity of features happens in neural networks as well. The basic model predicts a number through recognized features, so it can make a wrong prediction if a handwritten image has similar features with multiple numbers.

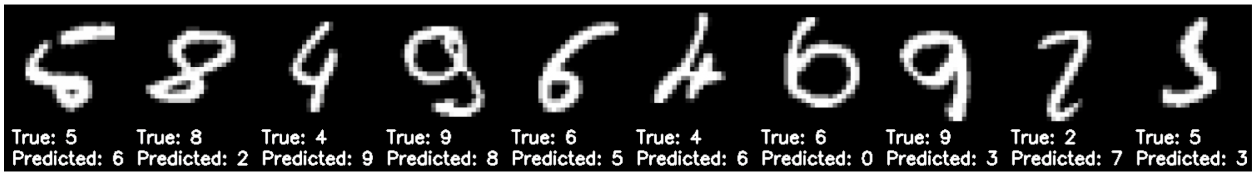

Figure 1 | Failure cases of the basic model.

The failure cases showed how a neural network works. In the first hidden layer, each unit captured a small feature of an input image[1]. Then, multiple features were merged in the next hidden layer, so the merged features could represent big characteristics that include circles or lines.

Effect of varying epochs

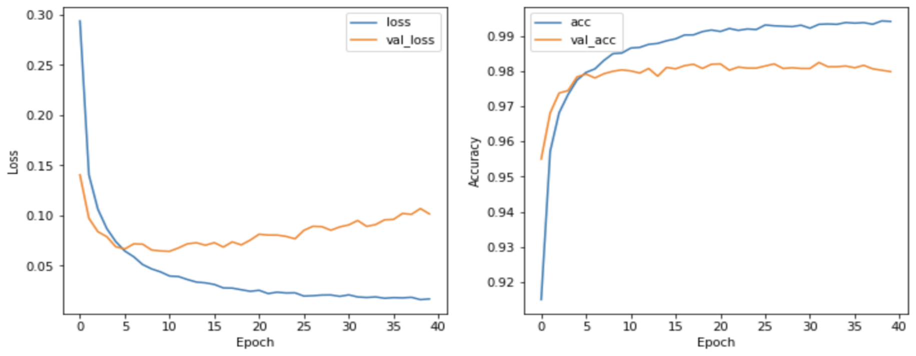

Figure 2 | The performance of the basic model that trained with 40 epochs. Although the validation accuracy stayed above 97%, the validation loss increased after around 10 epochs.

We trained the basic model with 2, 5, 10, 20, 40, 100 and 500 epochs. In general, we observed that increasing epochs up to a threshold improves validation accuracy but if epochs are increased beyond that, then accuracy does not increase and the model starts to ovefit. Validation loss tended to increase after around 10 epochs. Figure 2 shows well how the model starts to be over-trained since loss on the test set increased while the loss of the training set was decreasing.

Effect of varying width and depth of NNs

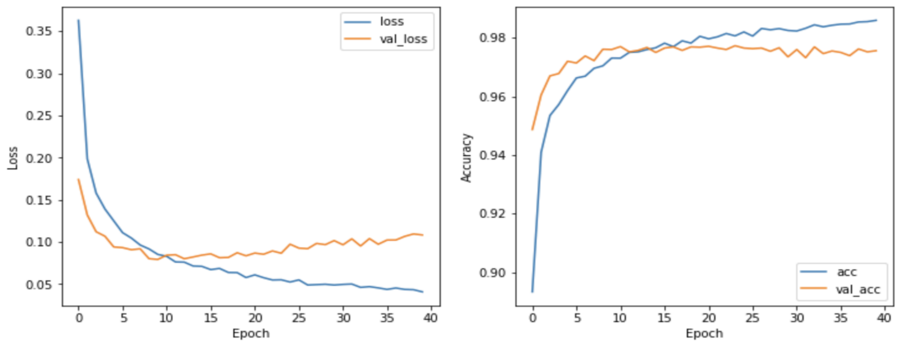

Figure 3 | The performance of the model that used one hidden layer with 64 units. A model that used 64 units for a hidden layer produced a similar result to the basic model.

To see the impact of different numbers of units, we used 2, 8, 64, 256 and 512 units on the basic model. The models with 2 and 8 units produced low accuracies on the test set, 61% for 2 units and 88% for 8 units, since insufficient units couldn't take into account all features of the inputs. The model that used 64 units showed 64 units are enough to take the input's features (See figure 3). From the model with 64 units, the model's accuracy increased as the number of units increased, but these accuracies were generated due to overfitting.

To understand the relationship between a depth of neural network and a model performance, we built various networks with no hidden layer, two hidden layers, and four hidden layers etc. Without a hidden layer, max accuracy of 93% was observed. We tried to apply two hidden layers with a dropout layer right after each hidden layer, but it resulted in a higher loss on the test set since the last dropout layer set the meaningful values to zero[2]. Then, we tried to remove all dropout layers, but the model was easily overfitted (as shown in figure 4) since the model didn't prevent overfitting by the dropout layer. Therefore, we applied one dropout layer right after the first hidden layer, so we could get a great accuracy of 98.22%. The four hidden layers with 128 units didn't show any improvements than the basic model. Moreover, we tried to build three networks with gradually decreasing, gradually increasing, and concave units. The models that used gradually decreasing units and concave units produced the best results of 98.4% and 98.66%, but the model which used gradually increasing units showed a similar performance to the basic model.

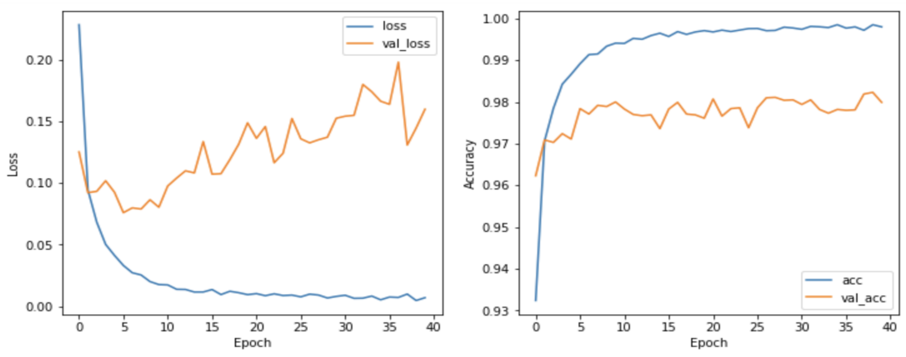

Figure 4 | The performance of the model without any dropout layer. The model is overfitted since the model doesn't drop any value of a unit in the hidden layers.

Preprocessing images with Sobel Edge detector

Since edge detection can be helpful as a preprocessing step, we applied Sobel edge extraction on the MNIST dataset. We trained models with Sobel X, Sobel Y and Sobel X+Y as input images. Through this experiment, we tried to see whether the edge extraction is helpful for the MNIST dataset or not.

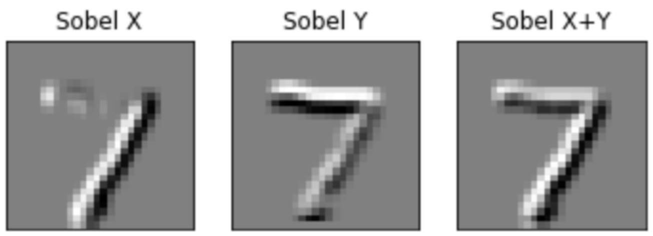

Figure 5 | Examples of Sobel edges.

To extract the Sobel edges, we applied a 3 x 3 kernel since an input image size, 28 x 28, is small. We performed the same experiments with these preprocessed datasets. Overall, using Sobel X and Sobel X+Y didn't improve accuracy compared to baseline. If anything, they underperformed in some cases compared to baseline with validation accuracy of around 96.07% (compared to 98%). Also, the model trained with Sobel X+Y was usually overfitted with a 20% loss on the test set. However, the model trained with Sobel Y edges showed only slightly better accuracy of 98.28% than the basic model, and it was never overfitted.

Conclusion

Using the basic model in beginner.ipynb, we could observe that classifying numbers with a neural network works similar to human vision since a neural network compares features of numbers like humans. Also, we learned how our modifications influence model performance in the various experiments. Having a lot of epochs was not always good for performance because we needed to be aware of overfitting. A small number of units were insufficient to hold features of the input, whereas a large number of units helped to increase the validation accuracy by capturing many features of the input up to a point. The model used two hidden layers with a dropout layer right after the first hidden layer performed better than the basic model, but the model without any hidden layer or with four hidden layers didn't produce better results than the basic model. Gradually decreasing shape and concave shaped networks were helpful to increase accuracy on the test set. In the preprocessing experiments, we applied Sobel X, Sobel Y, and Sobel X+Y, but only Sobel Y helped to improve the performance only slightly.

References

[1] https://youtu.be/aircAruvnKk
[2] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout