schuler_5779_ece_527_report_08_pt1

November 29, 2024

1 GMU ECE 527 - Computer Exercise #8 Part 1 - Report

Stewart Schuler - G01395779
20241205

Logistic Regression & Neural Networks The first classifier designed to be used as a baseline for future tests is a *logistic regression* (LR) classifier. A LR model can be built using *keras* by connected an Input layer directly to the output layer with no hidden layers between.

This model will be used as a baseline for comparison with other *neural network* (NN) models. The LR model has 7850 parameters to be learned, and when trained on the *MNIST* digits dataset it acheived a training accuracy of 0.933 and a test accuracy of 0.927. A summary of the LR classifiers can be found in *Table 1*.

Classifier	Shape	Parameters	Training Acc	Test Acc
Logistic Regression	(I 728, O 10)	7,850	0.933	0.927
1 Hidden NN	(I 728, 28, O 10)	22,270	0.990	0.966
2 Hidden NN	(I 728, 26, 20, O 10)	21,160	0.990	0.965

Table 1. Classifier Comparion on *MNIST* Digits.

The next classifer tested was a 1 hidden layer NN. As shown in *Table 1*, the 1 hidden NN model is able to acheive a training set accuracy of 0.99 and test accuracy of 0.966. Both notable above their respective metics for the LR model. However the cost of doing so was using nearly 3 times as many parameters, meaning slower training and inference time for the model. The hidden layer of 28 neurons was the smallest value I was able to find that could acheiving a ≥ 0.99 training accuracy.

Finally a 2 hidden layer NN was training on the same dataset. To get nearly identical performance to the 1 hidden NN model using two hidden layers it was found that the first and second layers should have 26 and 20 neurons respectivly. This works out the a model with a nearly equal number of parameters and the 1 hidden model.

10 Layer Neural Network Next we consider the provided model with 10 hidden layers each containing 10 neurons. The model as provided is missing both a flattening and an output *softmax* activation layer. Once added the model can be propperly trained and evaluted. The model was found to have the results sumarized by *Table 2*.

Next the table was modifed to have each hidden layer contain 32 neurons. As seen in *Table 2* it drastically increases the ammount of parameters to be learned. When compared with the 1 or 2 NN

models used in part 1, this model performs comparable with significantly more parameters, leading to the conclusion this architecture isn't well suited for the problem dataset.

It can be see by comparing *Tables 1* and *2* that models with a similar number of parameters have very similar accuracies even though they have drastically different architectures.

Classifier	Shape	Parameters	Training Acc	Test Acc
10 Hidden NN (10)	(I 728, 10x10, O 10)	8,950	0.942	0.923
10 Hidden NN (32)	(I 728, 32x10, 20, O 10)	34,954	0.991	0.966

Table 2. 10 Hidden Layer NN Comparion on *MNIST* Digits.

Convolutional Neural Network Lastly we consider a *convolutional neural network*. The smallest model I was able to find that was able to achieve ≥ 0.99 is defined by the summary in *Figure 1*. The model was able to reach a training accuracy of 0.990 and a test set accuracy of 0.987. Making it the best performing model so far. Of note the model only uses 3090 parameters. Which demonstrates that the inclusion of the convolutional and pooling layers can be use useful for out performing the parameters to accuracy relationship observed in the traditional NN models.

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 24, 24, 5)	130
max_pooling2d_12 (MaxPooling2D)	(None, 12, 12, 5)	0
conv2d_13 (Conv2D)	(None, 10, 10, 5)	230
max_pooling2d_13 (MaxPooling2D)	(None, 5, 5, 5)	0
flatten_16 (Flatten)	(None, 125)	0
dense_50 (Dense)	(None, 20)	2,520
dense_51 (Dense)	(None, 10)	210

Figure 1. CNN Model Summary

When training the model it was observed that increasing the batch size lead to a faster trained model with a reduced training accuracy. Decreasing the batch size improved the training accuracy at the cost of training time. For the *MNIST* digits dataset I found a batch size of 128 samples was a reasonable balance between the two tradeoffs.

A dense layer is a fully connected layer of neurons. One is required to be the output layer with a *softmax* activation. Any more than the output layer are not strictly required however I found performance greatly increased by including a second dense layer imediately before the output layer.

Batch normalization is a way of regularizing the weight values by, within a training batch, computing the mean/variance values of each layer and normalizing by it. This prevent one weight from becoming dominate. I found including it in my model made very little difference.

Dropout is a form of regularization where within a training batch some neurons are randomly unused, that is set to a 0 weight. The effect this has is it helps to prevent the case of a few neurons being weighted so heavily that they become dominate. The other neurons must learn during cases

where the dominate neurons are randomly disabled. This can help prevent over fitting. In the case of my model since the dense layers are rather small and there aren't many parameters to learn overfitting through heavy weights is unlikely. So dropout would not be needed in the case. It may be useful for a very deep neural network approach to this same dataset.