NMIST Kaggle

Cole Killian

November 02 2019

Implementation of the model

We tried many different model implementations. We started out with a dense neural network and scored around 70% accuracy. Then we upgraded to a convolution neural network and experimented with a many different architectures. This meant testing out 1-4 convolutions layers, 2-3 dense layers, and 1-5 dropout layers in an attempt to prevent overfitting. The total parameters of these models varied from ~70,000 to ~950,000.

Results

Our first submission scored a 84%. We found that models with more parameters tended to perform better as long as there were enough dropout layers to prevent overfitting. We were aiming for a model that took 25 or fewer epochs to train, so if it went 25 epochs without the training accuracy pulling away from the validation accuracy, we weakened the dropout layers. Eventually we found a best model that included three convolutional layers, two dense layers, two pooling layers, two batch normalization layers, and three dropout layers for a total of 635,882 trainable parameters!. This model's summary is displayed on the following page.

Challenges

This is more of a superficial challenge, but a challenge none the less: not taking advantage of GPU acceleration. Too much time was spent waiting for models to train that could have been spent optimizing them. On the positive side, eventually I did find out about GPU acceleration, and will get into the habitat of activating it every time I open Google Collab.

I was challenged by my disuse of a plan when hyperparameter training. In hindsight, I should have taken a more systematic approach instead of randomly changing hyperparameters and hoping for improvements in results. In reality this makes no sense because with my current level of understanding of convolutional neural networks, the chance of the model improving its performance after one of my tweeks is probably close to 50%. Writing a program to test a variety of different architectures and hyperparameters and letting it run over night is what I should have done. But at least I will value this strategy more highly next time.

Conclusion

Overall, this Kaggle was a fantastic learning experience. I got to make many mistakes and will learn from them as best as I can. My takeaways include that I should spend more time researching the goal before blinding coding an implementation, make sure that I am taking advantage of all possible resources (looking at you GPU acceleration), and that I should develop a systematic approach to solving problems rather than a spontaneous one. Also, it is so satifying when an "improvement" actually works!

```
Model: "sequential_22"
_____
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_55 (Conv2D)           (None, 28, 28, 32)        320
_____
conv2d_56 (Conv2D)           (None, 28, 28, 32)        9248
_____
max_pooling2d_43 (MaxPooling (None, 14, 14, 32)        0
_____
dropout_53 (Dropout)         (None, 14, 14, 32)        0
_____
conv2d_57 (Conv2D)           (None, 12, 12, 64)        18496
_____
batch_normalization_12 (Batc (None, 12, 12, 64)        256
_____
max_pooling2d_44 (MaxPooling (None, 6, 6, 64)          0
_____
dropout_54 (Dropout)         (None, 6, 6, 64)          0
_____
flatten_22 (Flatten)         (None, 2304)              0
_____
dense_55 (Dense)             (None, 256)               590080
_____
batch_normalization_13 (Batc (None, 256)               1024
_____
dropout_55 (Dropout)         (None, 256)               0
_____
dense_56 (Dense)             (None, 64)                16448
_____
dense_57 (Dense)             (None, 10)                650
=================================================================
Total params: 636,522
Trainable params: 635,882
Non-trainable params: 640
```

Figure 1: Model Summary

Individual Contributions

Cole Killian:

I wrote the code for automating the downloading, unzipping and loading into memory of the data files, as well as for saving the submission to a csv (although this was adapted by code shared by Rick in the Slack).

I built the keras model architectures and performed hyperparameter tuning manually. That being said, I was heavily influenced by the following very helpful articles and the starter code from Slack:

1. https://mlexplained.com
2. https://towardsdatascience.com
3. https://www.youtube.com/watch?v=lOZGYzTn9Z8
4. https://www.kaggle.com/pavansanagapati/a-tutorial-cnn-model-fashion-mnist

Link to Github Repository:

https://github.com/Ruborcalor/CNN_NMIST