Assignment Deep Learning Summer School

Bas Machielsen

July 2022

Introduction

In this short report, I outline my efforts for the Deep Learning Summer School assignment. Our job was to predict housing prices on the basis of 29 explanatory variables. I used the TensorFlow library implemented in Python (3.7.6) to train my deep learning model.

I then downloaded the files, kept the test data, and splitted the train data into training and validation data.

Building the Neural Network

The first thing I noticed is that we cannot do pre-processing since the dependent variable of the test data is unavailable to us. Hence, we have to use normalization procedures within the network. Then, I created and compiled a network. I opted for several dropout layers and L1-regularizations to compensate for the impossibility to standardize the data. The dropout layers did not seem to work well, irrespective of the rest of the structure.

I experimented with the number of layers using a ceteris paribus number of nodes in each layer. It turned out that three intermediate layers worked best. Then, I experimented with the number of nodes in each layer. It turned out that a 'large to small' structure worked better than a 'small to large' structure. By this, I mean that the layers with a larger number of nodes came as one of the first layers, and the layers closer to the output layer contained fewer nodes. I used only linear activiation functions, since we have data that is positive everywhere, and also, because we are dealing with a relatively simple prediction problem on the basis of numerical, non-spatial data.

For the same reasons, I did not explore fancier and deeper structure such as recurrent or convolutional neural networks, since they seemed unsuitable for the task.

I opted for the first layer with 15 nodes, the second layer with 10, and the third layer with 5 nodes, leading to the output layer with 1 node. Including the biases, this makes for a total of 671 parameters. The output layer is of course not regularized, since we should not restrain the predictions.

For brevity, I don't show the code used to create and compile the model, but here is a summary:

```
model.summary()
```

```
## Model: "sequential"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense (Dense)                (None, 15)                450
## _________________________________________________________________
## dense_1 (Dense)              (None, 10)                160
## _________________________________________________________________
## dense_2 (Dense)              (None, 5)                 55
## _________________________________________________________________
## dense_3 (Dense)              (None, 1)                 6
```

```
## ================================================================
## Total params: 671
## Trainable params: 671
## Non-trainable params: 0
## ----------------------------------------------------------------
```

I trained the model in 1000 epochs, using the Adam optimizer with a learning rate of 0.01. The learning rate and the number of epochs seemed to face a trade-off, with a lower learning rate necessitating more epochs to reach a kind of convergence, and a higher learning rate necessitating less epochs. A higher learning rate, however, increased the risk of non-convergence, which I experienced several times in trial.

Aside from Adam, I also experienced with different optimizers and more parameters, specifically RMSprop and the momentum parameter, but found that these generally performed much worse, with a much higher loss, or failing to converse. I also experimented with the *beta*'s and `amsgrad` option in Adam, but these parameters did not seem to matter that much, so I opted for the default option.

As for the loss, I (of course) specified logarithmic MSE loss as the objective function. As mentioned before, I splitted the data into train and validation data, and part of my exploration of the hyperparameter space was based on the performance on the validation sample. The results that I got were about a 4% mean squared logarithmic error in the training data, and 5% in the validation data. On the basis of this model, I submitted my predictions.

Conclusion

After submitting, it turned out the accuracy was lower on the testing dataset, on which I received a logarithmic MSE of 0.22, meaning the predictions were 22% off on average. Considering the small difference in performance between my training and validation samples, I do not think overfitting is the reason. There might be different reasons for the lack of generalizability. Maybe the sample is materially different, or the parameter space has been insufficiently explored to create truly generalizable predictions.

The code for this assignment can be found in an Rmd file **here**. Thank you for reading!