TMA4267 - Assignment 3

Student number: 480858

May 7th, 2022

Introduction

We are now going to use a 2-level k-factor experiment to study how neural network parameters will effect its performance. In general, you would want more levels (at least for the learning rate) to find good parameters, but we want to see if a two-level can be used for a small dataset. We also found some research papers, such as the paper by Pontes et al. (2016) and the paper by Laosiritaworn and Chotchaithanakorn (2009), using DOE, so it also seems to have some research interest.

Neural networks are able to detect patterns in the data, then use this to make predictions on new data. Neural networks have been shown to be very powerful for a lot of different type of data, such as tabular data, text and image. However, despite its ability to do a lot of things, they require a lot of tuning and can take a long time to train if the dataset is large. Another concern is also overfitting.

1.1 Dataset

The dataset we will consider is a regression dataset. We will look the Auto MPG dataset[1], where we used the nine features (cylinders, displacement, horsepower, weights, acceleration, model year, origin (Europe, USA, Japan)) to determine the miles per gallon for a car. The dataset consists of 398 samples. We divided the data into train, validation and test set. We trained on the dataset for 100 epochs, but used early stopping to reduce overfitting. When the validation loss had not decreased over 5 epochs we stopped the training, and returned the weights of the model at the best validation loss. We then compared our models performance on the test set. To do this, we used Tensorflow in Python. The code can be found at `https://github.com/rubenmu96/DOE/blob/main/Neural_network.ipynb`.

Factors and levels

In our experiment, we will consider four factors, with two levels each. They are found in Table 1. The learning rate and the optimizer are probably two of the most important parameters in a neural network. If the learning rate is too low, we get very slow convergence and if the learning rate is too large, it might diverge. We will here consider the default[2] Adam and SGD optimizer algorithms. Both of these optimizers can be improved, by for example adding weight decay (to both) and momentum (to SGD), but we will not consider this here. For their default implementation, we expect Adam to perform better, as seen by Kingma and Ba (2014). Since we are dealing with a small dataset, two hidden layers should be enough, and 32 and 64 nodes seems sufficient.

[1] The dataset can be found at `https://archive.ics.uci.edu/ml/datasets/auto+mpg`

[2] The default implementation of Adam and SGD in Tensorflow.

Factors	Level: -1	Level: +1
Optimizer: The optimizer is the algorithm used to update the weights in a neural network. Two common optimizers are Adam and SGD, so we choose them as our two levels	SGD	Adam
Learning rate: The learning rate decides how big the weight update will be	0.005	0.01
Hidden layer 1: The number of nodes in the first hidden layer	32	64
Hidden layer 2: The number of nodes in the second hidden layer	32	64

Table 1: Our chosen factors and their levels.

An issue with neural networks is that we are dealing with non-convex functions, so we have no way of controlling if the chosen levels for the factors are good values. For learning rate, one will often test more values, and if one wants to train large networks, it is also often useful to use a learning scheduler.

There are interactions between our factors. For example, SGD and Adam are both dependent on the learning rate and might prefer different values. From our experience,

Response variable

To measure the performance of a neural network, there are multiple response variables we can use. Since we are using a regression dataset, possible responses are MAE (mean absolute error) or RMSE (root mean squared error). Both of these could have worked well as a response, but we decided to go for the MAE, which is calculated using the formula $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$, where y_i is the true value and $\hat{y}_i$ is the predicted value. MAE will treat all errors equally (in comparison to for example RMSE, which will penalize larger errors more), it also not senstive to outliers.

Design and experiment

4.1 Design

We decided to go for a 2^k fractional design instead of a 2^{k-p} fractional factorial design. Fractional factorial designs are usually done in the case where doing many experiments is expensive, but this is not a concern for us, as each run only takes a few seconds to produce the response variable. We didn't find it necessary to use blocking variables.

4.2 Reproducibility and replication

An issue with neural networks is that they are stochastic, so running a neural network twice will lead to different results. A way to control this is by using a seed. Since we wanted to compare neural networks with different parameters, they should have the same weight initialization. We therefore decided to use a seed.

As expected, when using a different seed, we observed that we got a different result. Choosing the seed that gives the best performance is of course not something that one should do, but testing different seeds can be a good way to test how robust the model is. We ran our R code on two different seeds and noticed that our results (main plot, interaction plot and so on) had some variations. Therefore, we decided to use four replications. We also saw that a similar approach was used in Laosiritaworn and Chotchaithanakorn (2009) (but there was no mention if they used a seed for their two replicates or if every run was ran without a seed).

4.3 Randomization

Randomization is not a concern in our experiment. Since our neural network will produce the same result (due to the seed), it does not matter which order we run the experiment, but we still decided to use FrF2 in R to get an order we didn't choose, as it is good practice to do so. We ended up with the order seen in Figure 1.

	Optimizer	Learning_rate	Hidden1	Hidden2
1	SGD	5e-3	32	32
2	Adam	5e-3	32	32
3	SGD	1e-2	32	32
4	Adam	1e-2	32	32
5	SGD	5e-3	64	32
6	Adam	5e-3	64	32
7	SGD	1e-2	64	32
8	Adam	1e-2	64	32
9	SGD	5e-3	32	64
10	Adam	5e-3	32	64
11	SGD	1e-2	32	64
12	Adam	1e-2	32	64
13	SGD	5e-3	64	64
14	Adam	5e-3	64	64
15	SGD	1e-2	64	64
16	Adam	1e-2	64	64

Figure 1: For each replication, we used this order.

Results and analysis

We will now analyze our results using the different approaches that can be seen in Tyssedal.

5.1 Standard errors and effects

The effects, as stated in Tyssedal, is two times the coefficients. We get the effects found in Figure 2.

```
                          (Intercept)                                Optimizer1
                            4.0350625                                -0.1578750
                        Learning_rate1                                  Hidden11
                           -0.1395000                                 0.0404375
                              Hidden21                     Optimizer1:Learning_rate1
                           -0.0008750                                 0.1813125
                  Optimizer1:Hidden11                           Optimizer1:Hidden21
                           -0.0356250                                 0.0073125
               Learning_rate1:Hidden11                      Learning_rate1:Hidden21
                           -0.0067500                                -0.0010625
                    Hidden11:Hidden21           Optimizer1:Learning_rate1:Hidden11
                           -0.0092500                                -0.0098125
Optimizer1:Learning_rate1:Hidden21                 Optimizer1:Hidden11:Hidden21
                           -0.0133750                                -0.0234375
  Learning_rate1:Hidden11:Hidden21 Optimizer1:Learning_rate1:Hidden11:Hidden21
                           -0.0183125                                -0.0147500
```

Figure 2: The calculated effects

Figure 3 shows the summary of or linear regression model. We see that the coefficients with large effect are also the ones with low p-value. We would say that optimizer, learning_rate, Hidden1 and optimizer:learning_rate are important.

```
                                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                                2.0175313  0.0101500 198.771  < 2e-16 ***
Optimizer1                                -0.0789375  0.0101500  -7.777 4.79e-10 ***
Learning_rate1                            -0.0697500  0.0101500  -6.872 1.16e-08 ***
Hidden11                                   0.0202188  0.0101500   1.992   0.0521 .
Hidden21                                  -0.0004375  0.0101500  -0.043   0.9658
Optimizer1:Learning_rate1                  0.0906563  0.0101500   8.932 8.94e-12 ***
Optimizer1:Hidden11                       -0.0178125  0.0101500  -1.755   0.0857 .
Optimizer1:Hidden21                        0.0036562  0.0101500   0.360   0.7203
Learning_rate1:Hidden11                   -0.0033750  0.0101500  -0.333   0.7410
Learning_rate1:Hidden21                   -0.0005312  0.0101500  -0.052   0.9585
Hidden11:Hidden21                         -0.0046250  0.0101500  -0.456   0.6507
Optimizer1:Learning_rate1:Hidden11        -0.0049063  0.0101500  -0.483   0.6310
Optimizer1:Learning_rate1:Hidden21        -0.0066875  0.0101500  -0.659   0.5131
Optimizer1:Hidden11:Hidden21              -0.0117188  0.0101500  -1.155   0.2540
Learning_rate1:Hidden11:Hidden21          -0.0091563  0.0101500  -0.902   0.3715
Optimizer1:Learning_rate1:Hidden11:Hidden21 -0.0073750  0.0101500  -0.727   0.4710
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0812 on 48 degrees of freedom
Multiple R-squared:  0.8051,    Adjusted R-squared:  0.7442
F-statistic: 13.22 on 15 and 48 DF,  p-value: 2.968e-12
```

Figure 3: Summary of our linear regression model

5.2 Main effect

As expected, the optimizer and learning rate are two important parameters in our neural network. We see that the second hidden layer gives the same performance for both of its level (so it is not significant), while the first hidden layer is not as important as the learning rate and optimizer, but still has some significance.

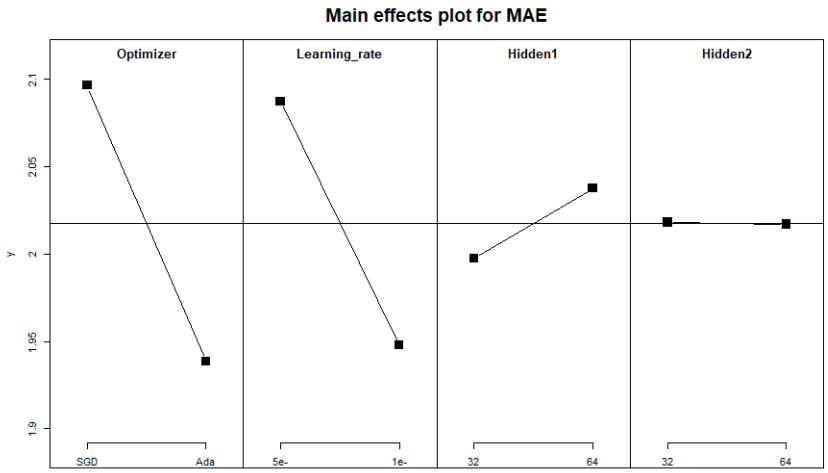

Figure 4: Main effect of our factors

5.3 Interaction plot

We see that there is interaction between optimizer and learning rate. The others seems to have no or very little interaction.

Figure 5: Interaction plot

5.4 Normal plot

The factors shown are the ones that are significant, where the further away from 0 (on the x-axis), the more significance it will have. From our normal plot we see that optimizer, learning rate, Hidden1, and optimizer:Learning_rate are significant. This matches with what we observed above, where Hidden1 has some significance, but not as much as the three others.

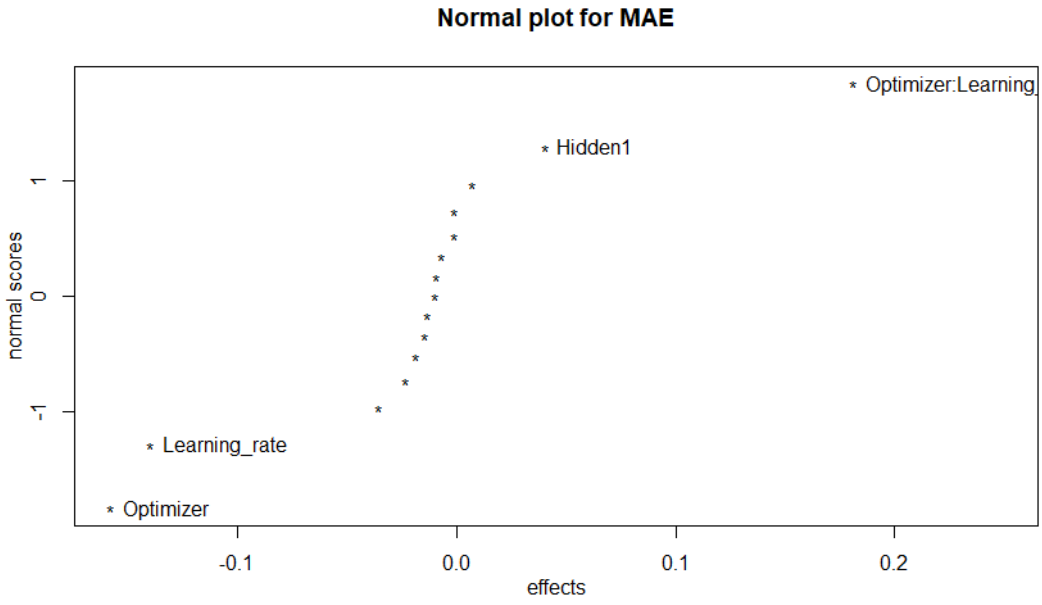

Figure 6: Normal plot

5.5 Linear regression and assumptions

When using linear regression, we are making certain assumptions on the data. To see if these assumptions hold, we look at plots seen in Figure 7. We make the following remarks

- Residuals vs fitted: This plot shows that the assumption of linearity holds, as the red line is pretty straight. The residuals also seems to be independent.
- Scale-location: This plot shows that the red line is almost straight. This indicates that the condition of equal variance (or homoscedasticity) is satisfied.
- QQ-plot: We see that most points lies on the straight line, however, there are some deviations (but these deviations are not too big). We are not quite sure, but we are going to say that the data looks normally distributed.

From these remarks, we would say that the assumptions of linear regression is not violated.

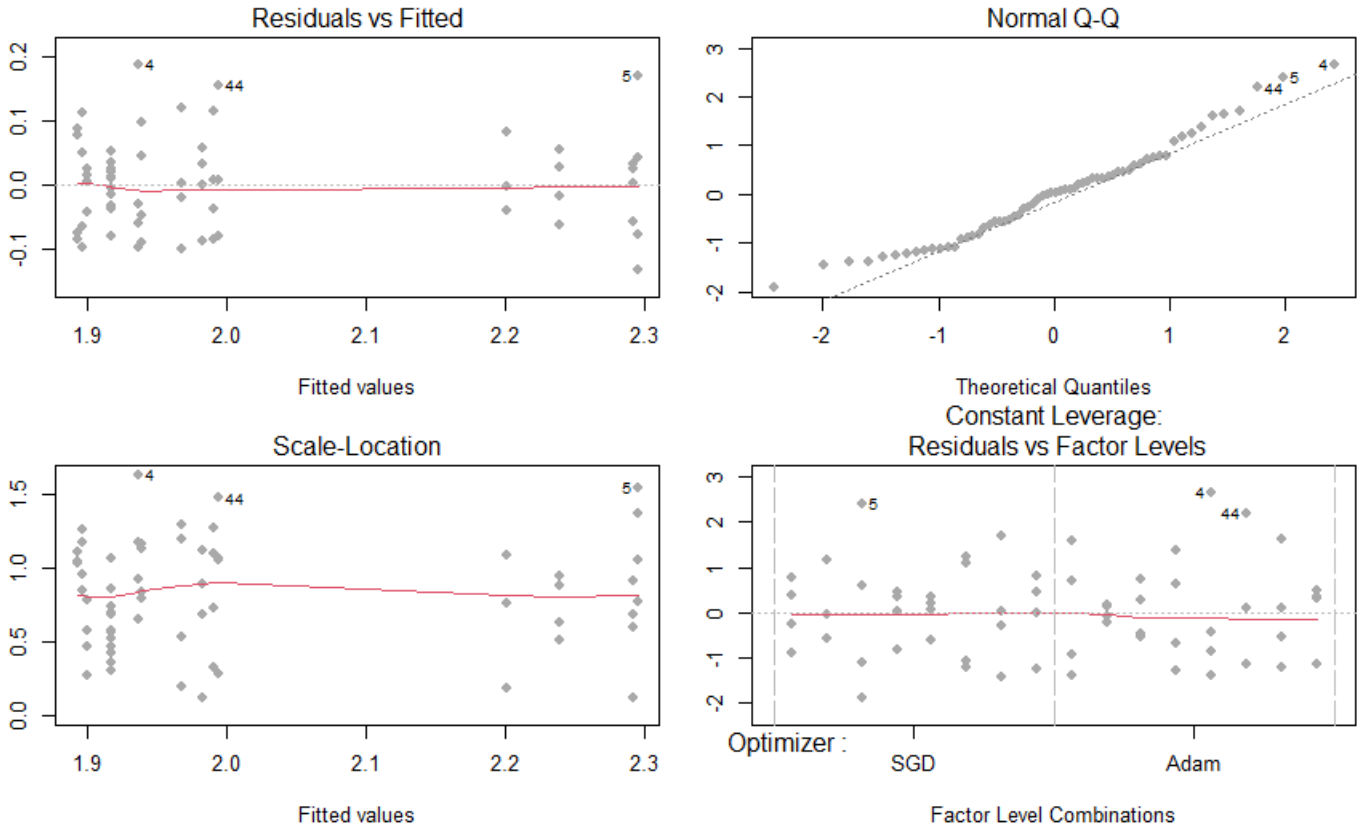

Figure 7: In this figure we consider different plots to see if the linear regression assumption is met.

Conclusion

We observe that the two factors, optimizer and learning rate, were important for the response. The number of nodes in our first hidden layer did have some significance, but the number of nodes in the second layer had very little effect on the response. We also observed that the interaction between optimizer and learning rate were important. This matches well with what we expected. We might have expected the effect of the optimizer to be a bit lower. Adam is less sensitive to learning rate tuning, and the chosen learning rates might have not been too good for SGD, so it might explain the result.

References

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Wimalin Laosiritaworn and Nantakarn Chotchaithanakorn. Artificial neural networks parameters op-

timization with design of experiments: An application in ferromagnetic materials modeling. *Chiang Mai Journal of Science*, 36(1):83–91, 2009.

Fabrício José Pontes, GF Amorim, Pedro Paulo Balestrassi, AP Paiva, and João Roberto Ferreira. Design of experiments and focused grid search for neural network parameter optimization. *Neurocomputing*, 186:22–34, 2016.

John Tyssedal. Design of experiments.