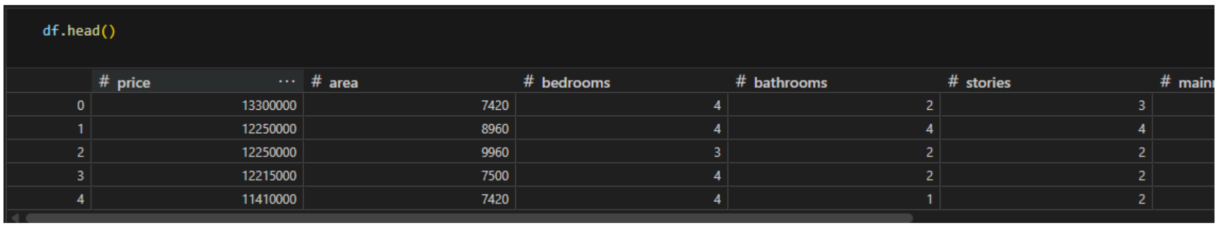
Project X

The goal of this project was to take a copy of one of the github projects of our lecturer and experiment with a simple artificial neural network (ANN) created form scratch, in order to understand how neural networks, learn from data, why they are built this way and what kind of effects arise from certain adjustments.

Dataset

The dataset, that was used in this project contains information about houses, including their selling price and many more significant independent variables. The goal for the artificial neural network is to learn the relationship between the property characteristics and the target variable "price" and then to use this learned relationship to predict new houses, with the given parameters.

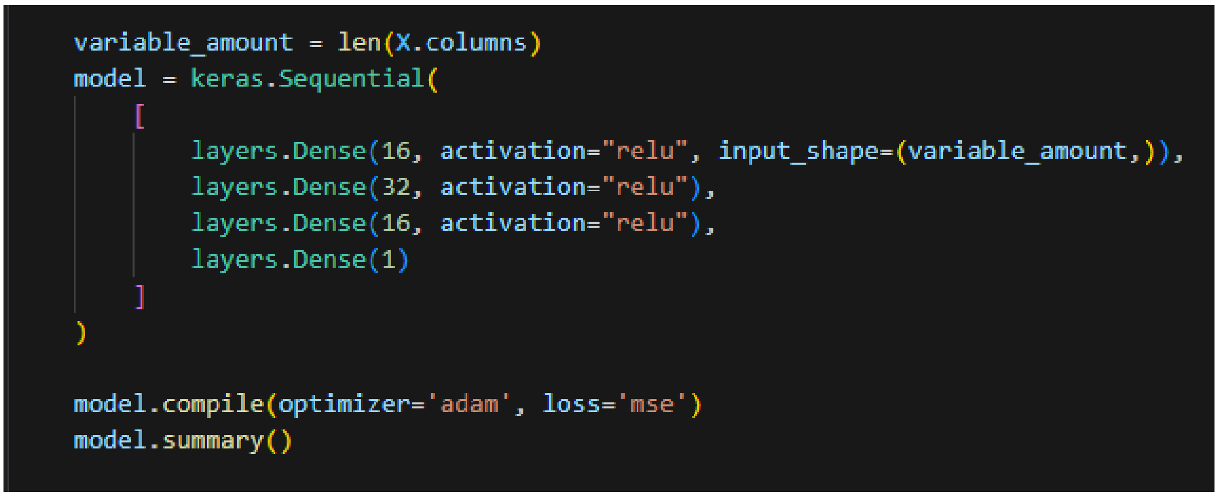


```
df.head()
```

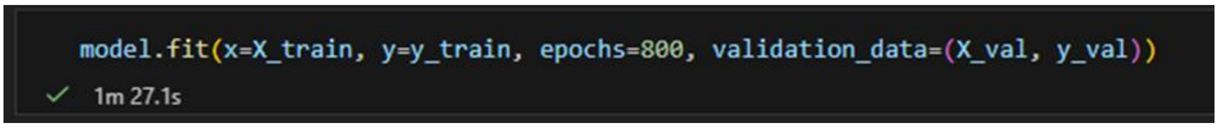
	# price	# area	# bedrooms	# bathrooms	# stories	# main
0	13300000	7420	4	2	3	
1	12250000	8960	4	4	4	
2	12250000	9960	3	2	2	
3	12215000	7500	4	2	2	
4	11410000	7420	4	1	2	

Neural Network Structure

The neural network I have chosen to experiment with is called "ann_regression_example_2.ipynb" from the "lecture2_ann_regression_example" folder.



```
variable_amount = len(X.columns)
model = keras.Sequential(
    [
        layers.Dense(16, activation="relu", input_shape=(variable_amount,)),
        layers.Dense(32, activation="relu"),
        layers.Dense(16, activation="relu"),
        layers.Dense(1)
    ]
)

model.compile(optimizer='adam', loss='mse')
model.summary()
```



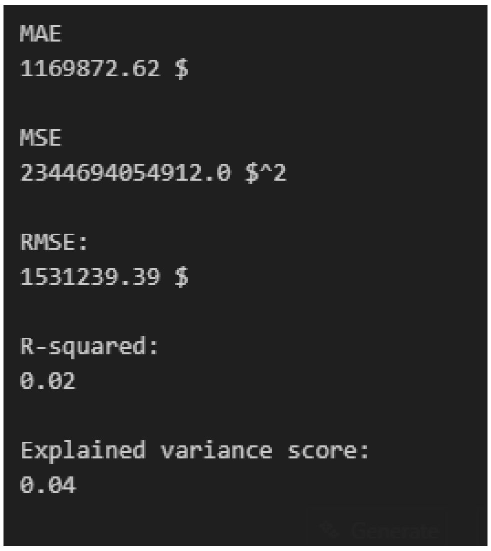
```
model.fit(x=X_train, y=y_train, epochs=800, validation_data=(X_val, y_val))
✓ 1m 27.1s
```

Why only Dense layers?

Since the goal of the project is to predict house prices based on purely numerical data, using only Dense layers is the best architectural choice, because they allow every input feature to connect to every neuron in the network. This enables the model to learn complex relationships between all housing attributes (e.g. number of rooms, area, etc.) and the target variable (price).

Neuron architecture

The 16-32-16-1 architecture is a simple structure for a regression project with 545 data samples. In the theory of Deep Learning, we always differentiate between the input layer, which is a single one, hidden layers, which can be multiple depending on the dataset and an output layer. However, we have to adjust some parameters and change the neural network architecture, to reach a better model quality (R^2-score).

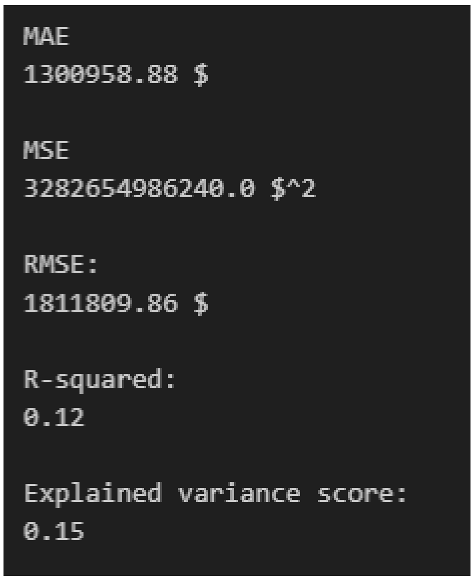


```
MAE
1169872.62 $

MSE
2344694054912.0 $^2

RMSE:
1531239.39 $

R-squared:
0.02

Explained variance score:
0.04
```

Different neuron patterns experiment

I want to try different types of architectures with different neuron patterns and optimizations to see how the model will reach and if the model quality improves overall. The different created neural architectures were **done without any optimizations,** that is why the R²-score is low as shown.

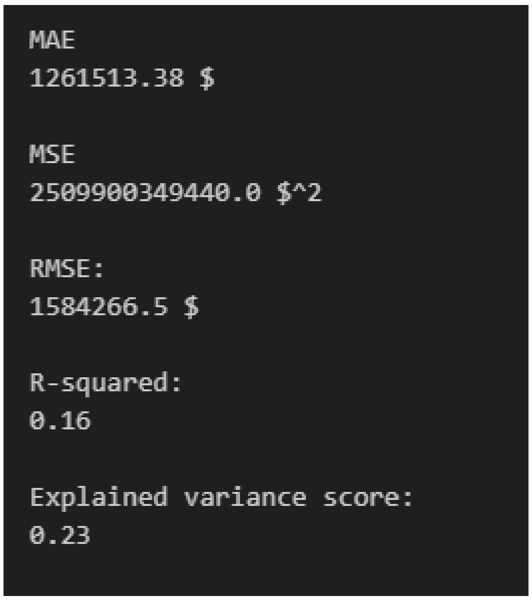
32-24-16-1 (normal, decreasing, decreasing, output)



```
MAE
1300958.88 $

MSE
3282654986240.0 $^2

RMSE:
1811809.86 $

R-squared:
0.12

Explained variance score:
0.15
```

In this architecture, I wanted to try a neuronal downward pattern, but as we can see in the error metrics, the performance was weak, but better than the other neural patterns. I feel like a decreasing neural pattern structure works better in general, but it could also be only good for ANN-Regression projects only.

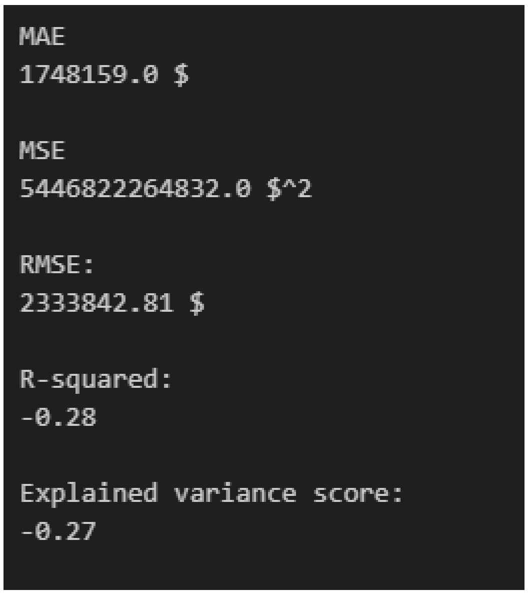
32-16-8-1 (normal, decreasing, decreasing, output)



```
MAE
1261513.38 $

MSE
2509900349440.0 $^2

RMSE:
1584266.5 $

R-squared:
0.16

Explained variance score:
0.23
```

This neural pattern worked the best out of all tested patterns with a decreasing structure but less neurons in the hidden layers as well as in the output layer. This tells me that a clean and consistent decreasing structure works better in general for ANN-Regression projects. In my opinion the narrowing of neurons appears to help the network reduce complexity while still learning important information.

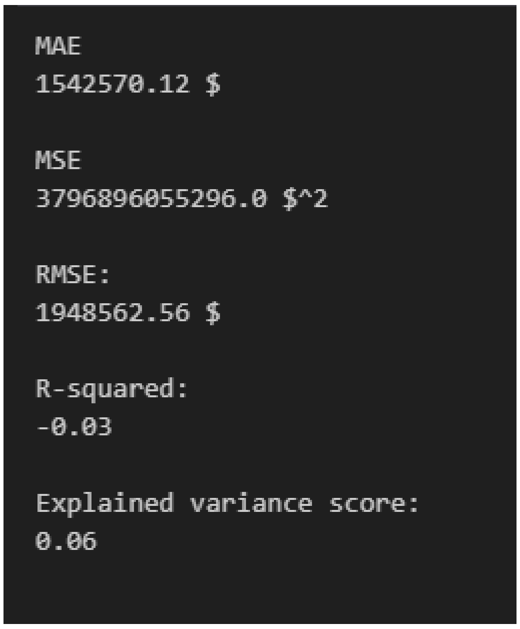
32-16-32-1 (normal, decreasing, increasing, output)



```
MAE
1748159.0 $

MSE
5446822264832.0 $^2

RMSE:
2333842.81 $

R-squared:
-0.28

Explained variance score:
-0.27
```

This was one of the weakest-performing architectures. The R²-score dropped into the negative range (-0.28). The MAE and RMSE were also significantly higher and in my opinion, I think the “decrease -> increase” structure seems to confuse the model, resulting in a bad model training.

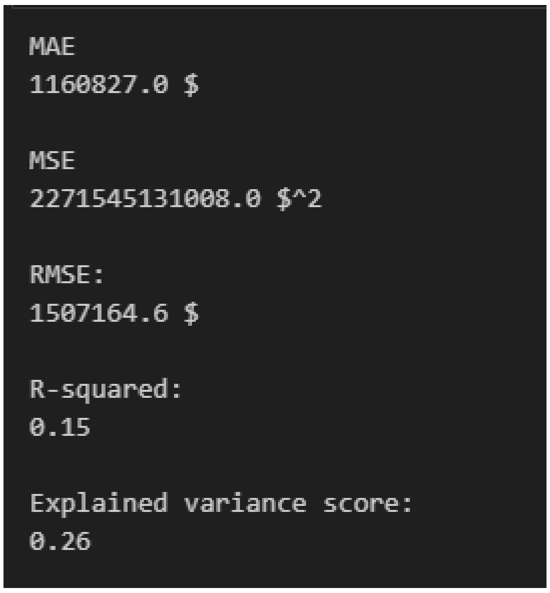
16-24-32-1 (normal, increasing, increasing, output)



```
MAE
1542570.12 $

MSE
3796896055296.0 $^2

RMSE:
1948562.56 $

R-squared:
-0.03

Explained variance score:
0.06
```

Also increasing the neurons with each hidden layer results in a negative R²-score, not being the best option for our ANN-Regression project.

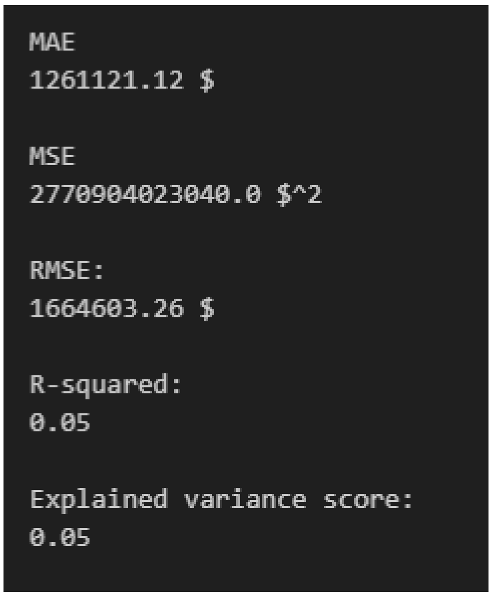
16-32-16-1 (normal, increasing, decreasing, output)



```
MAE
1160827.0 $

MSE
2271545131008.0 $^2

RMSE:
1507164.6 $

R-squared:
0.15

Explained variance score:
0.26
```

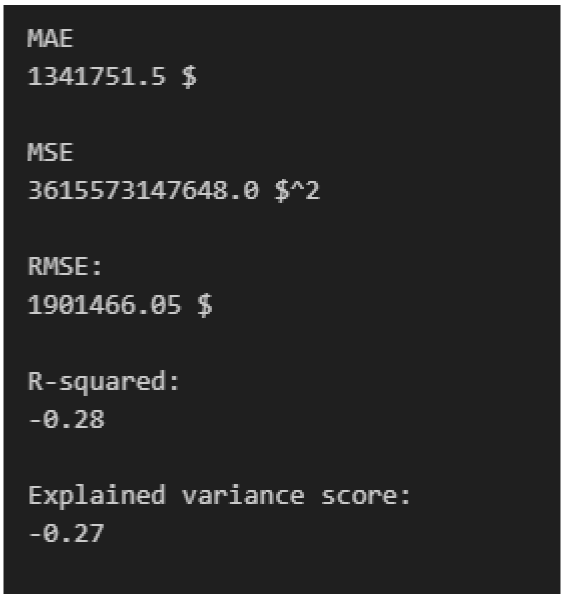
Using the neural model of my lecturer seems to work well. I feel like the narrowing phase of decreasing from one hidden layer with 32 neurons to another hidden layer with 16 neurons works the best in general, so that the model can learn effectively.

32-32-32-1 (all normal, output)



```
MAE
1261121.12 $

MSE
2770904023040.0 $^2

RMSE:
1664603.26 $

R-squared:
0.05

Explained variance score:
0.05
```

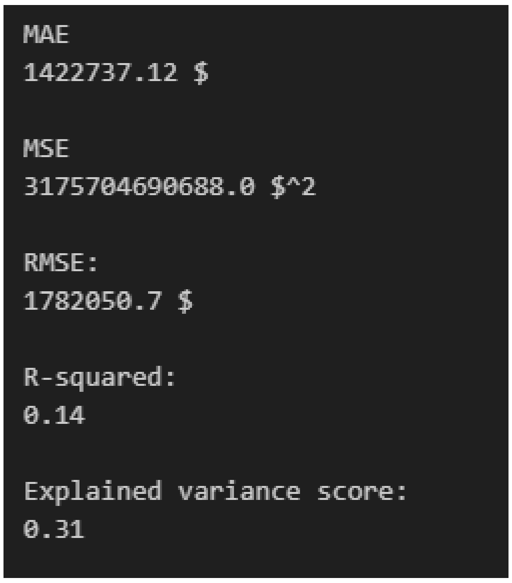
16-16-16-1 (all normal, output)



```
MAE
1341751.5 $

MSE
3615573147648.0 $^2

RMSE:
1901466.05 $

R-squared:
-0.28

Explained variance score:
-0.27
```

Increasing the neurons but keeping the same number of neurons does not work particularly well for model training, but it does not perform worse than the other neural structures with a negative R^2-score. My impression is that symmetric architectures like these one can be too generic and fail to guide the network toward a good compression before the final output. For a model to train and perform better in general there has to be a compression between the hidden layers, for the model to recognize patterns and learn from them effectively.

16-14-12-10-8-4-1 (normal, decreasing slowly until output)



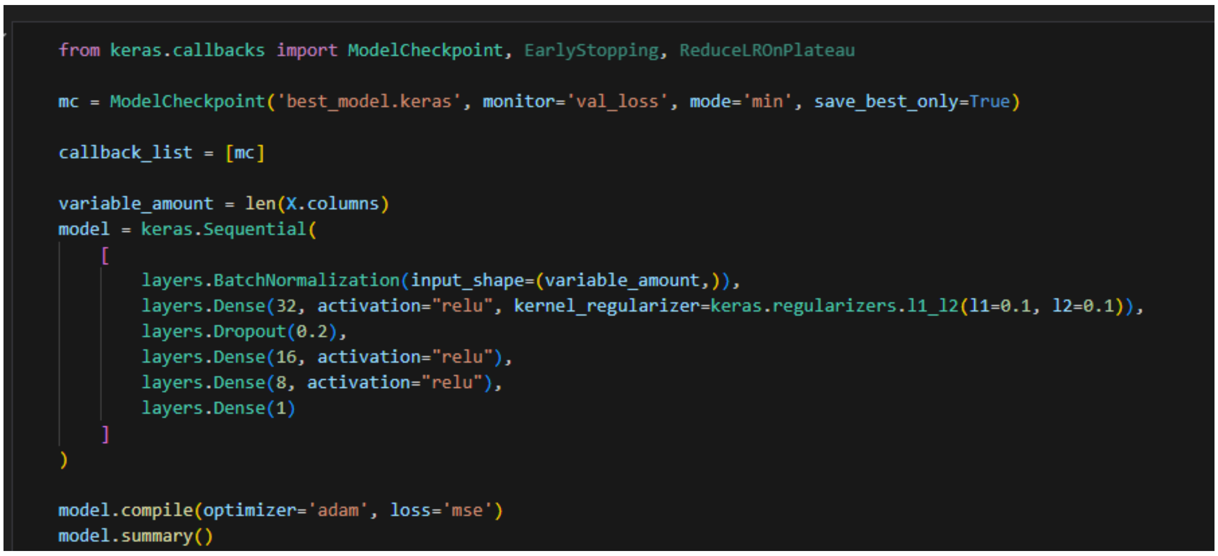
```
MAE
1422737.12 $

MSE
3175704690688.0 $^2

RMSE:
1782050.7 $

R-squared:
0.14

Explained variance score:
0.31
```

Since the compression of the hidden layers, by decreasing the number of neurons for each layer, worked so well, I have tested, that if I do the same decrease pattern, but with more layers this time, I will get a better R²-score and it worked, indicating that when compression and a decreasing neuron pattern exists, we automatically get a better trained model.

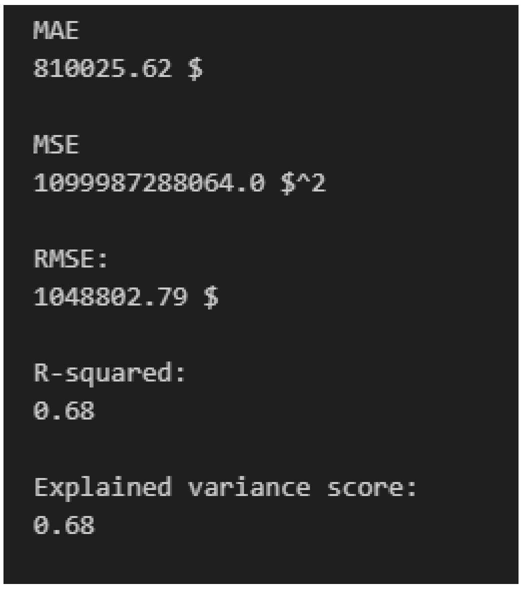
Result of different neuron patterns

The best performing architecture in my experiments was the slowly decreasing model structure **("16-32-16-1", "32-16-8-1", "16-14-12-10-8-4-1").** In my view this gradual reduction of neurons across layers helped the network compress information, recognize patterns and learn more effective in model training.

Adding optimizations (BatchNormalization, Regularization (l1,l2), Dropout)



```python
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

mc = ModelCheckpoint('best_model.keras', monitor='val_loss', mode='min', save_best_only=True)

callback_list = [mc]

variable_amount = len(X.columns)
model = keras.Sequential(
    [
        layers.BatchNormalization(input_shape=(variable_amount,)),
        layers.Dense(32, activation="relu", kernel_regularizer=keras.regularizers.l1_l2(l1=0.1, l2=0.1)),
        layers.Dropout(0.2),
        layers.Dense(16, activation="relu"),
        layers.Dense(8, activation="relu"),
        layers.Dense(1)
    ]
)

model.compile(optimizer='adam', loss='mse')
model.summary()
```



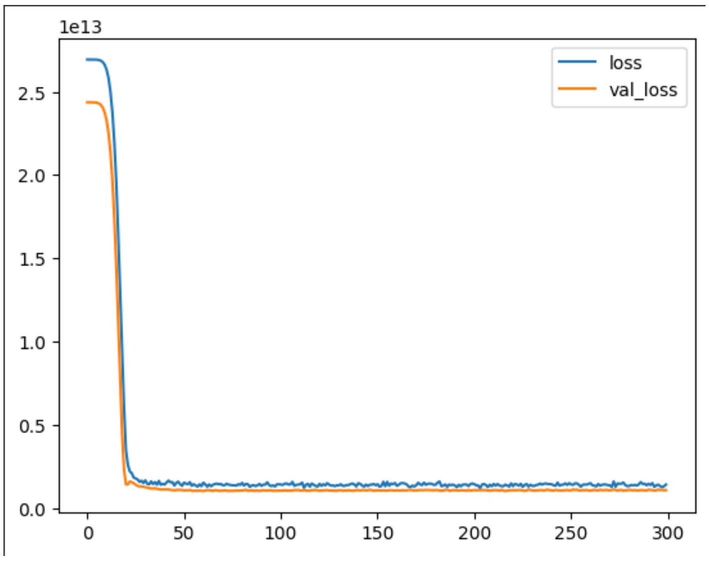
```
MAE
810025.62 $

MSE
1099987288064.0 $^2

RMSE:
1048802.79 $

R-squared:
0.68

Explained variance score:
0.68
```

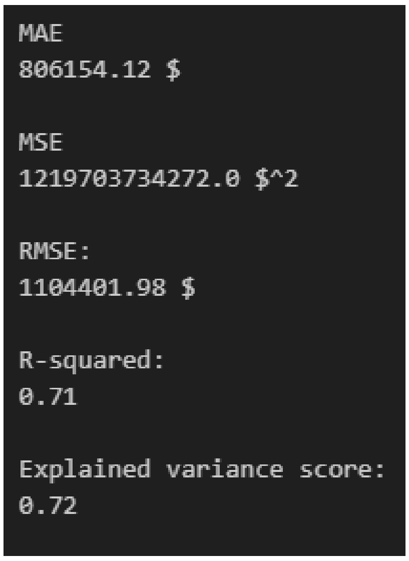
After experimenting with the neural network architecture, I used different optimizations to improve the model quality (R^2-score), so that I can experiment with the following questions:

Does the change of learning rate affect the training process?

To compare the effect of different learning rates, I have experimented with 6 different kinds of learning rates to see how they affect the model training, to see if the change of the learning rate affects the training process. I used the same number of epochs and we can see the differences with the visualization and the error metrics.

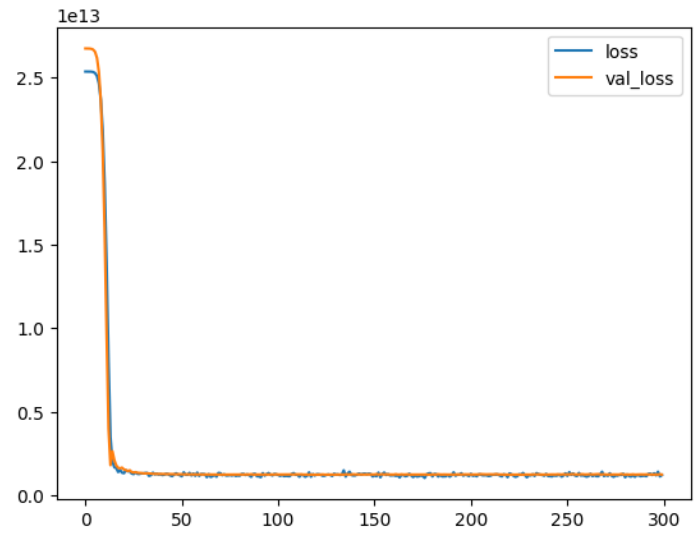
Learning_rate = 0.005 / Epochs = 300

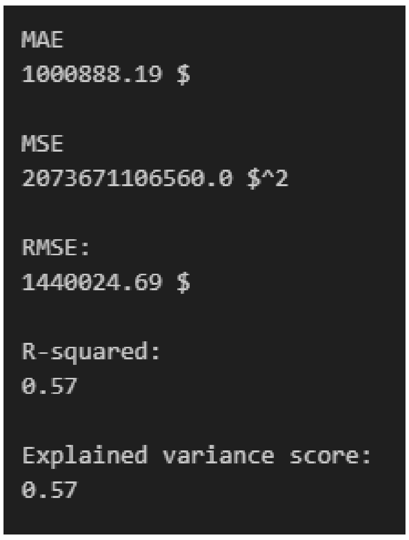




```
MAE
806154.12 $

MSE
1219703734272.0 $^2

RMSE:
1104401.98 $

R-squared:
0.71

Explained variance score:
0.72
```

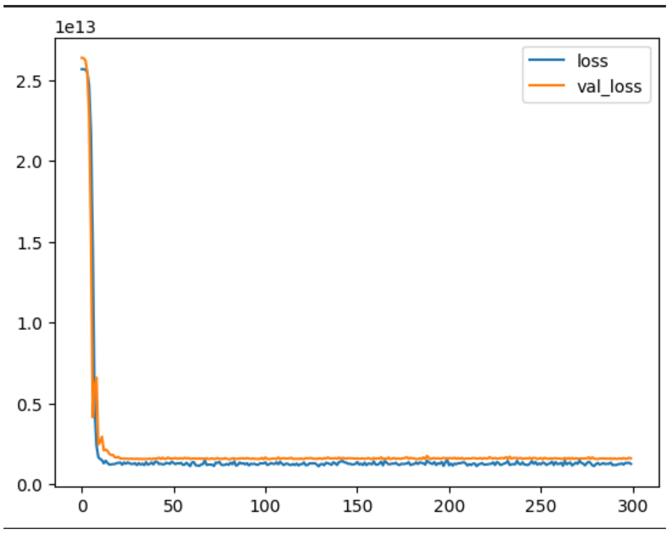
Learning_rate = 0.01 / Epochs = 300

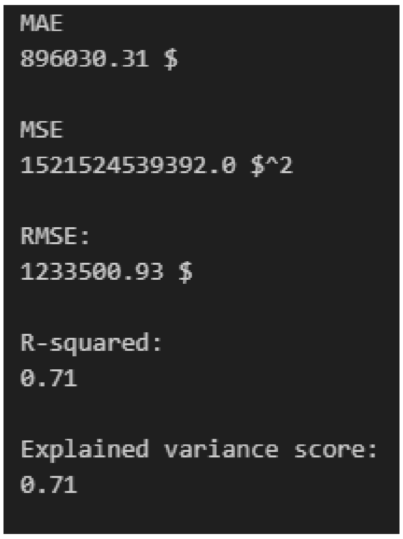




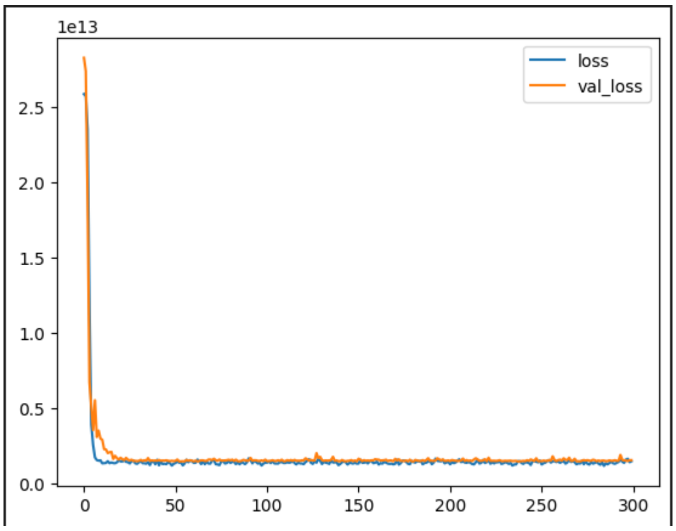
```
MAE
1000888.19 $

MSE
2073671106560.0 $^2

RMSE:
1440024.69 $

R-squared:
0.57

Explained variance score:
0.57
```

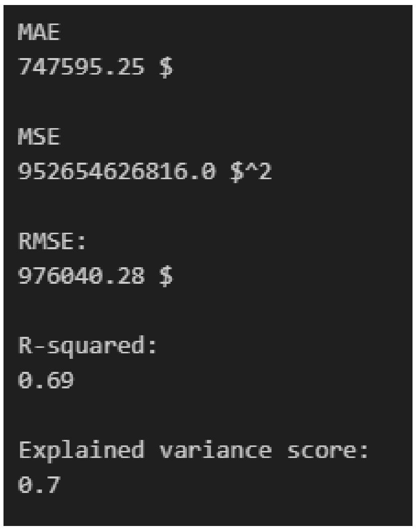
Learning_rate = 0.02 / Epochs = 300



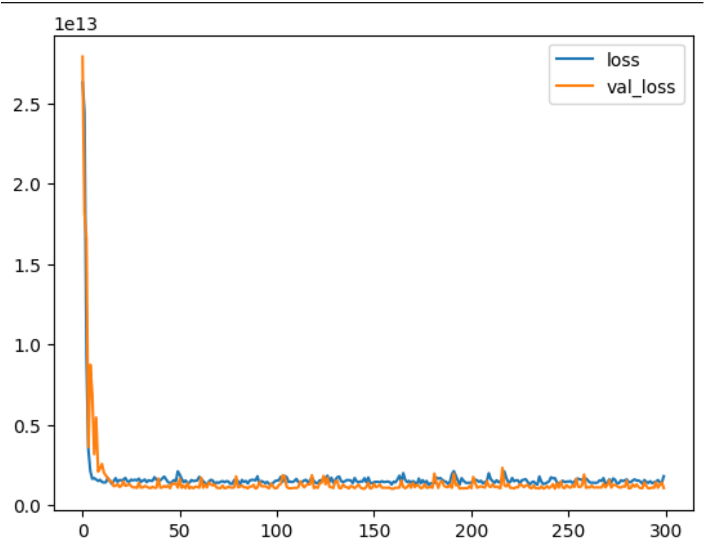

```
MAE
896030.31 $

MSE
1521524539392.0 $^2

RMSE:
1233500.93 $

R-squared:
0.71

Explained variance score:
0.71
```

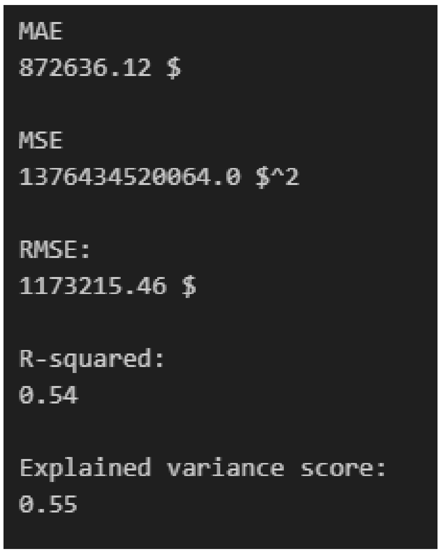
Learning_rate = 0.05 / Epochs = 300




```
MAE
747595.25 $

MSE
952654626816.0 $^2

RMSE:
976040.28 $

R-squared:
0.69

Explained variance score:
0.7
```

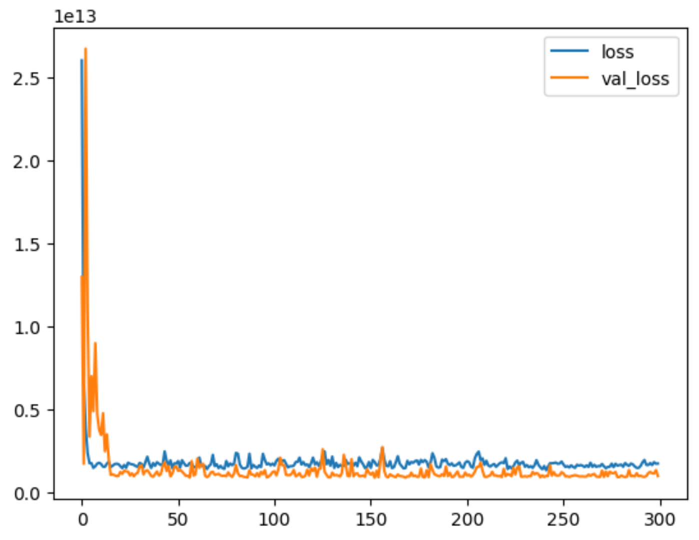
Learning_rate = 0.10 / Epochs = 300

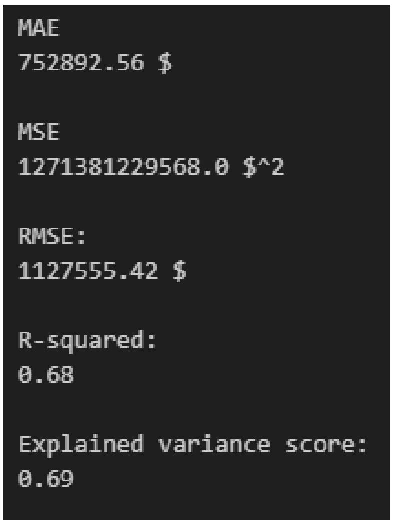




```
MAE
872636.12 $

MSE
1376434520064.0 $^2

RMSE:
1173215.46 $

R-squared:
0.54

Explained variance score:
0.55
```

Learning_rate = 0.25 / Epochs = 300





```
MAE
752892.56 $

MSE
1271381229568.0 $^2

RMSE:
1127555.42 $

R-squared:
0.68

Explained variance score:
0.69
```

Result of learning rate experiment

The learning rate experiments showed differences in how the model converged and how stable the training process was. Very small learning rates such as 0.005 produced smooth and stable loss curves with strong performance metrics (R^2=0.71), although they required slightly more epochs to reach convergence. Slightly higher learning rates like 0.01 and 0.02 still trained successfully, but their R^2 scores dropped noticeably, indicating that the model learned less efficiently. The best overall performance in terms of stability and accuracy was achieved between 0.005 and 0.05.

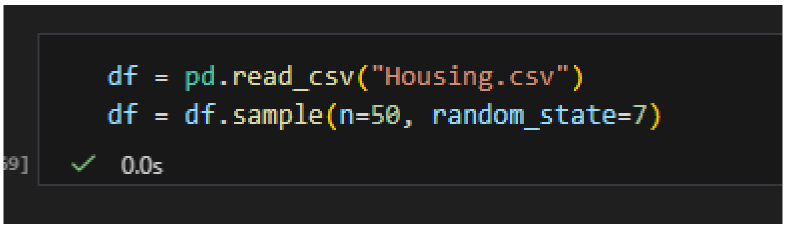
When the learning rate became too large, such as 0.10 or 0.25, the training curves became noisier and less stable, showing that very aggressive learning steps can harm the optimization process.

We can clearly see that the learning rate has a significant effect on the training of the neural network. A well chosen learning rate leads to smooth convergence, meaning better training, better performance and therefore better results.

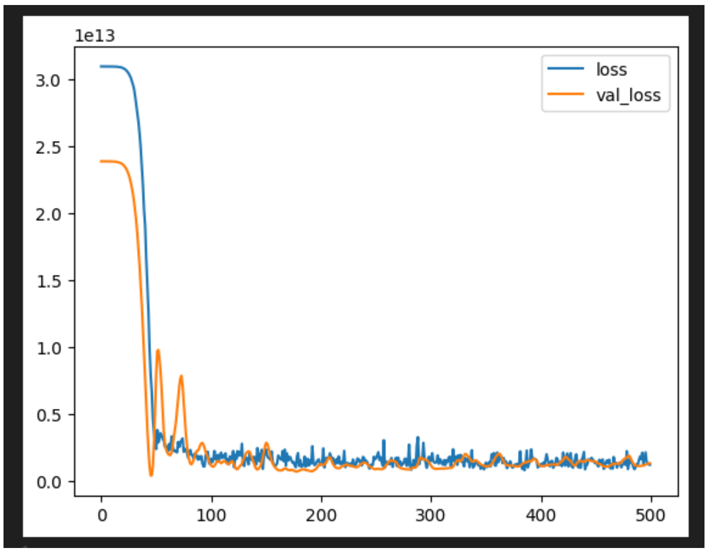
Alternating dataset size experiment (epochs=500)

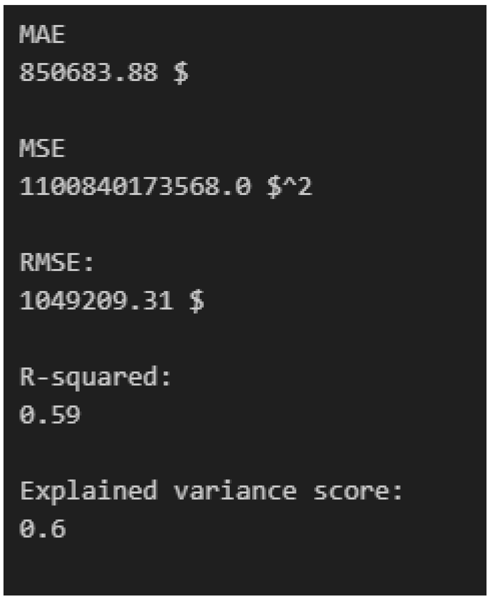
In this part of my project, I wanted to see how the size of the dataset affects the performance of my neural network. Since the original dataset contains 545 samples, I tested smaller sizes ranging from 50 to 300 rows to compare how the model behaves with more or less information. All experiments were trained for 500 epochs so that the results would be easy to compare.

n=50



```
df = pd.read_csv("Housing.csv")
df = df.sample(n=50, random_state=7)
✓ 0.0s
```

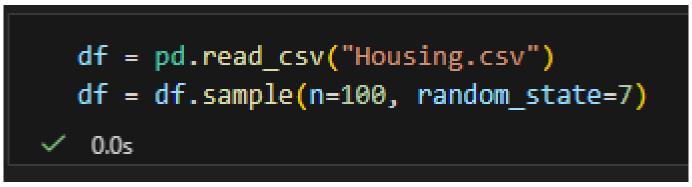




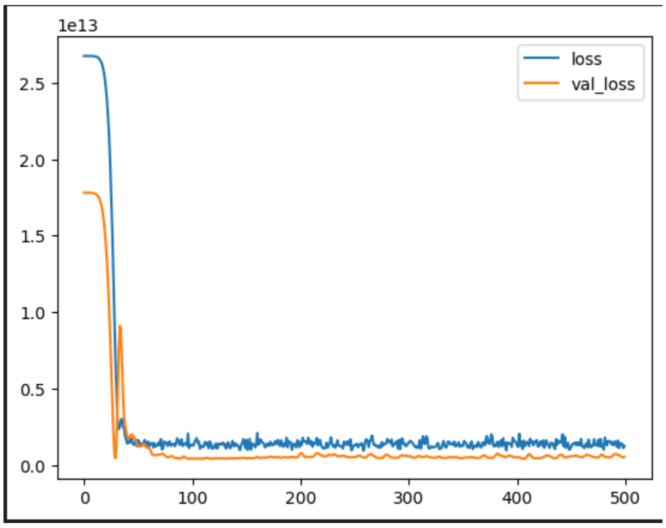
```
MAE
850683.88 $

MSE
1100840173568.0 $^2

RMSE:
1049209.31 $

R-squared:
0.59

Explained variance score:
0.6
```

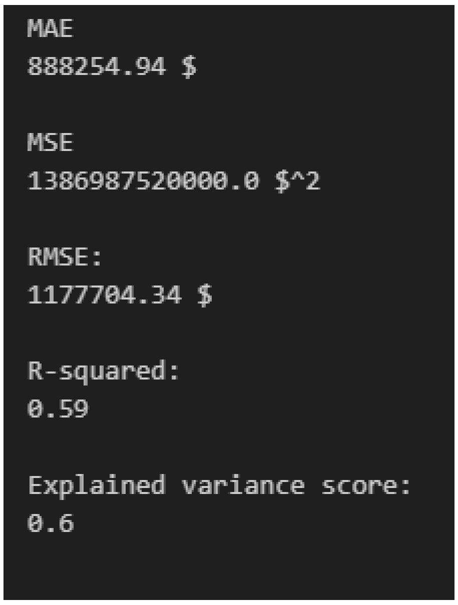
In my opinion, the model did surprisingly well, but with such a small dataset, the network does not have enough variation to learn the full complexity of housing prices.

n=100



```
df = pd.read_csv("Housing.csv")
df = df.sample(n=100, random_state=7)
✓ 0.0s
```

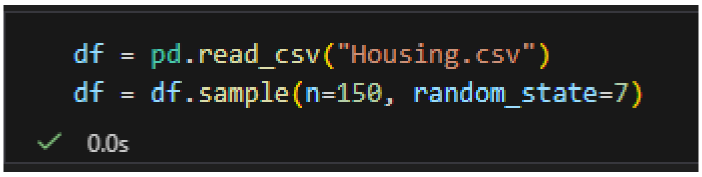




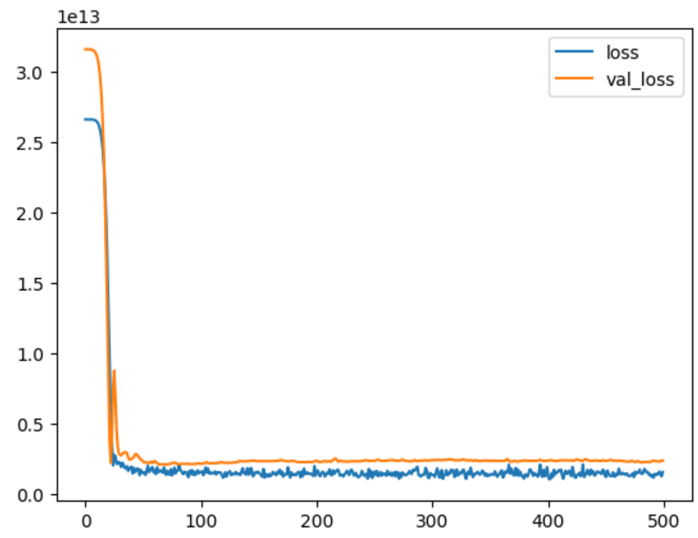
```
MAE
888254.94 $

MSE
1386987520000.0 $^2

RMSE:
1177704.34 $

R-squared:
0.59

Explained variance score:
0.6
```

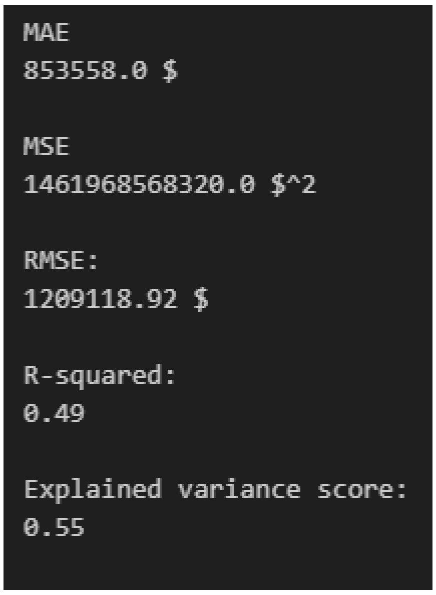
The R^2-score is the same as before with 50 samples, this means doubling the dataset from 50 to 100 rows does not make a big difference.

n=150



```
df = pd.read_csv("Housing.csv")
df = df.sample(n=150, random_state=7)
✓ 0.0s
```

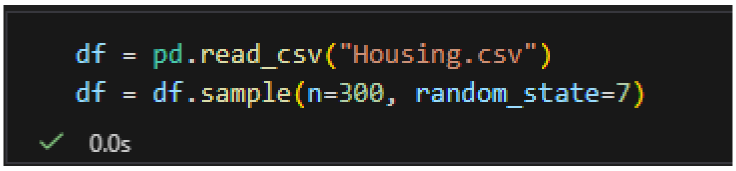




```
MAE
853558.0 $

MSE
1461968568320.0 $^2

RMSE:
1209118.92 $

R-squared:
0.49

Explained variance score:
0.55
```

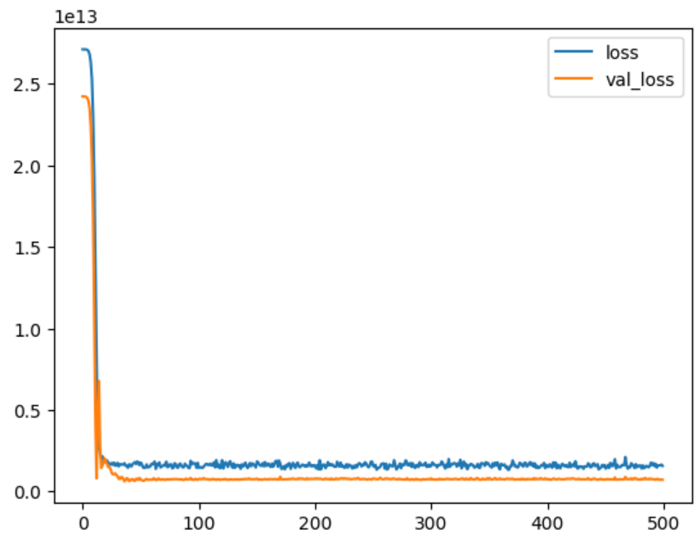
At 150 samples, the performance actually dropped a bit to 0.49. I personally think, that this might be because of the random sampling. We can also see that the val_loss graph was fluctuating even more this time.

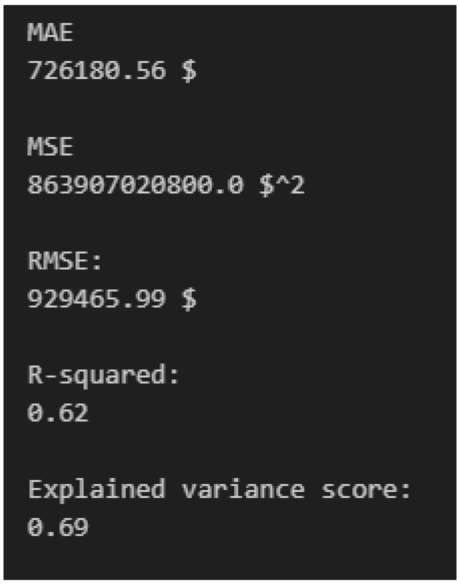
n=300



```
df = pd.read_csv("Housing.csv")
df = df.sample(n=300, random_state=7)
```

✓ 0.0s

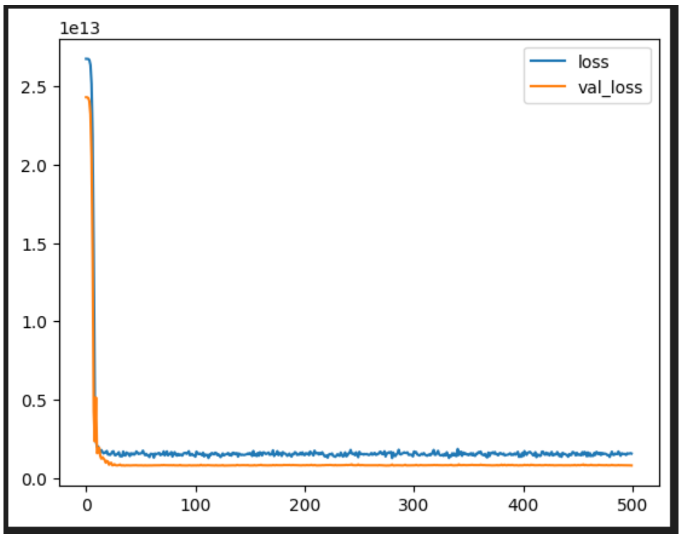


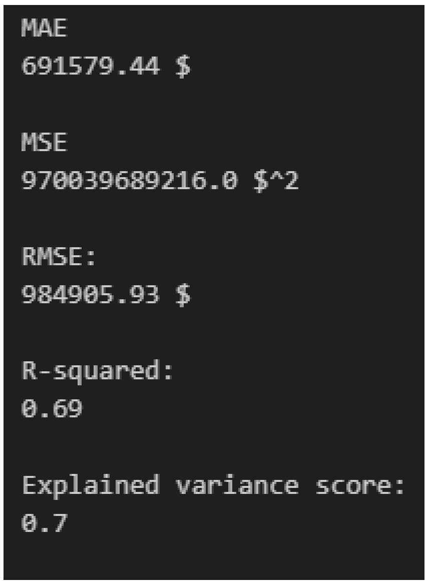


```
MAE
726180.56 $

MSE
863907020800.0 $^2

RMSE:
929465.99 $

R-squared:
0.62

Explained variance score:
0.69
```

The R^2-score rose to 0.62 and the loss curves became a bit smoother, indicating that increasing the dataset size can automatically improve the model's performance.

n=545 (original dataset size)





```
MAE
691579.44 $

MSE
970039689216.0 $^2

RMSE:
984905.93 $

R-squared:
0.69

Explained variance score:
0.7
```

Using the original dataset size produced the best result. The R²-score reached 0.69, and the MAE and RSME values were the lowest out of all experiments. We can also see, that the loss and val_loss curve was the most stable one. In my opinion, this clearly shows, the more data we have available to train the model with, the better the neural network understands the relationship between the variables of the house price dataset.

How can a small change to the neural model can make a drastic change in the performance?

After doing my experiments, even small changes to a neural network can lead to drastic differences in performance, because the model is highly sensitive to its architecture and parameters. For instance, in my neuronal pattern experiment, structures that increased and then decreased, performed much better than those that increased only. Also a low learning rate of 0.005 produced far more stable and accurate results than high parameters like 0.10 and 0.25.

How does the model learn something from the given data?

Usually in Deep Learning a neural network learns from the data through the process of back propagation, like it is said in the lecture's slides (DeepLearning_Part1_deeplearning). It is an interesting process where after the forward pass the model produces a prediction, the model calculates the error using the loss function. In my chosen architecture (16-32-16-1), this means that the error information flows from the output layer back through the 16-neurons and 32-neurons hidden layers, updating the parameters layer by layer. Over iterations in the training, these adjusted weights and bias reduce the loss and that is where the neural model learns patterns in the given dataset.

How does the neural network model create its predictions after training?

After the training is done, the neural network makes its predictions by simply applying what it has learned to new data. When I give the model fresh inputs, it runs them through all the layers, multiplies them with the weights it has learned, adds the biases and applies the activation functions (ReLU). Because the weights were optimized during training, the model can now use them to estimate the final output. In my case, the last layer produces one number, which becomes the predicted house price. So basically, after training, the network just uses its learned knowledge to make predictions.