Assignment 3 | Using deep learning

Lars Michael Alexander Tallqvist | 1901050

https://github.com/AlexanderTallqvist/AA-DATA-SCIENCE

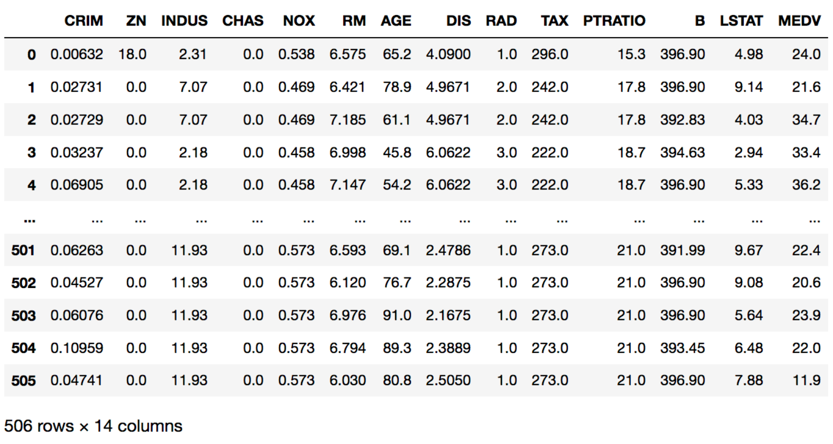
Introduction

This assignment discusses the process of using deep learning to predict housing prices. The data that we were given was Boston housing prices collected by the U.S Census Service, that we were to analyze by using deep learning and neural networks. Deep learning is a term for a neural network with many hidden layers. A neural network can be seen as a mathematical model that looks like the human brain, that can be used to discover patterns in data. In short, these are the tasks that were to be completed for this assignment:

1. Process the given data and make sure everything is in order.
2. Divide the dataset and use a portion of it for training.
3. Evaluate the performance of the network that you trained.
4. Try to predict the median value of owner-occupied homes in $1000's.

The Boston dataset

The data that we were to use contained a total of 506 rows of housing data. The dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass. There was a total of 14 columns of various housing related information. The dataset is described as follows:



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

Image 1: Boston dataset printed out in a Jupyter notebook.

- **CRIM**: per capita crime rate by town.
- **ZN**: proportion of residential land zoned for lots over 25,000 sq.ft.
- **INDUS**:proportion of non-retail business acres per town.
- **CHAS**: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
- **NOX**: nitric oxides concentration (parts per 10 million).
- **RM**: average number of rooms per dwelling.
- **AGE**: proportion of owner-occupied units built prior to 1940.
- **DIS**: weighted distances to five Boston employment centres.
- **RAD**: index of accessibility to radial highways.
- **TAX**: full-value property-tax rate per $10,000.
- **PTRATIO**: pupil-teacher ratio by town.
- **B** : 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
- **LSTAT**: % lower status of the population.
- **MEDV**: Median value of owner-occupied homes in $1000's.

The MEDV value is considered the target value of this dataset, and this was also the value that we were assigned to predict.

I started tackling the dataset by looking through the data and making sure that all the columns and rows had entries in them. I then wanted to get a better understanding of the data that I was working with, so I ended up creating scatterplots and box plots to figure out what features might have the biggest impact when trying to predict the MEDV value.

The scatterplots and the box plots below prove that there is a strong linear correlation between the LSTAT (% lower status of the population) and RM (average number of rooms per dwelling) with the target value MEDV. The LSTAT value has a negative correlation while the RM value has a positive one.

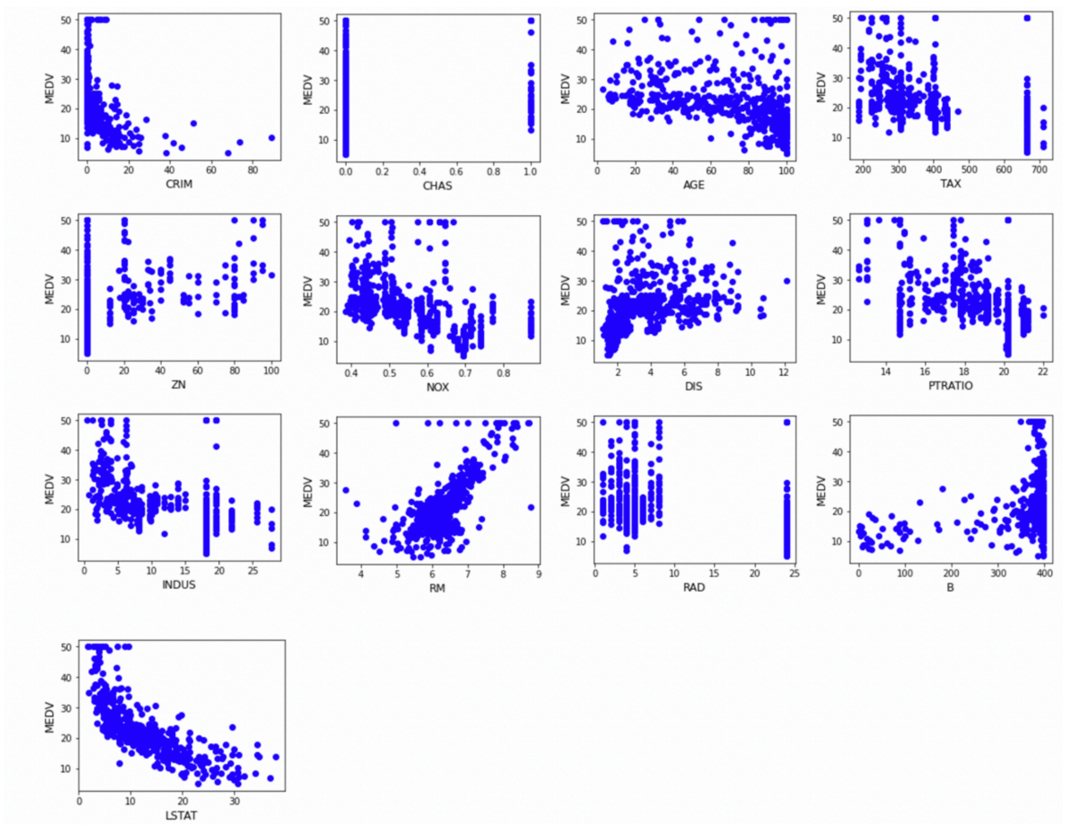


Image 2: Scatterplots of all the data columns. The MEDV value is used as the Y -column.

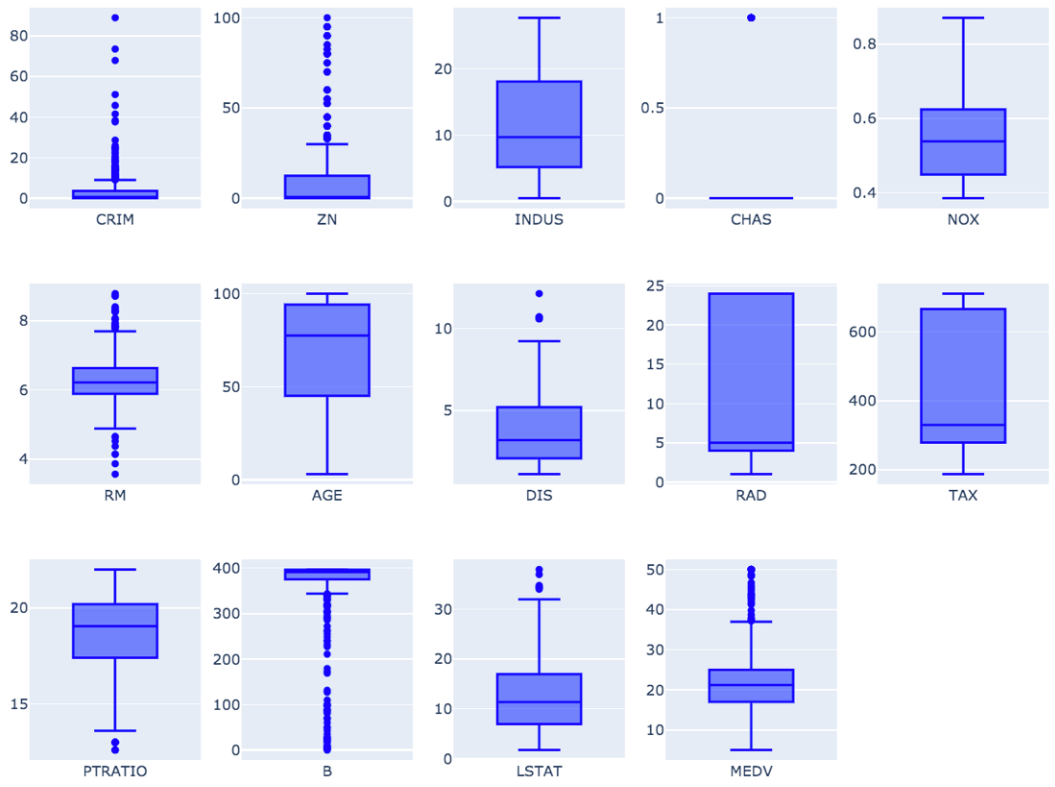
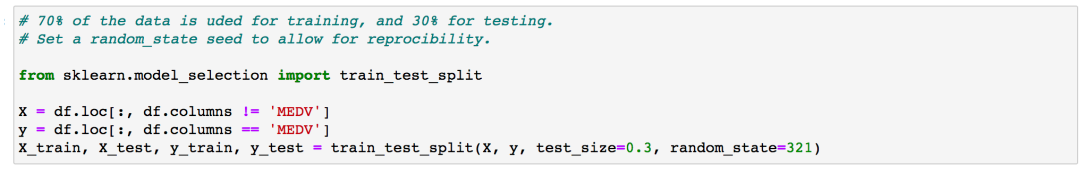


Image 3: Box plots of all the data columns. The MEDV value is used as the Y -column.

Data processing and modelling

After I felt like I had a better understanding of the data, it was time to start processing the data and train the neural network. I used a split of 70% and 30%, where the former part was used for training the network, and the latter for testing.



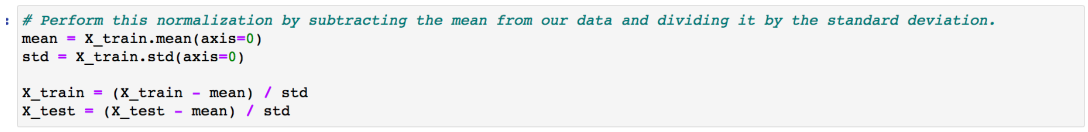
```
# 70% of the data is uded for training, and 30% for testing.
# Set a random_state seed to allow for reprocibility.

from sklearn.model_selection import train_test_split

X = df.loc[:, df.columns != 'MEDV']
y = df.loc[:, df.columns == 'MEDV']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=321)
```

Image 4: Splitting the data for training and testing.

In order to reduce the difference in scale that may arise from existent features, it is a good idea to provide standardized inputs to the neural network. I performed a normalization of the dataset by subtracting the mean from our data and dividing it by the standard deviation.

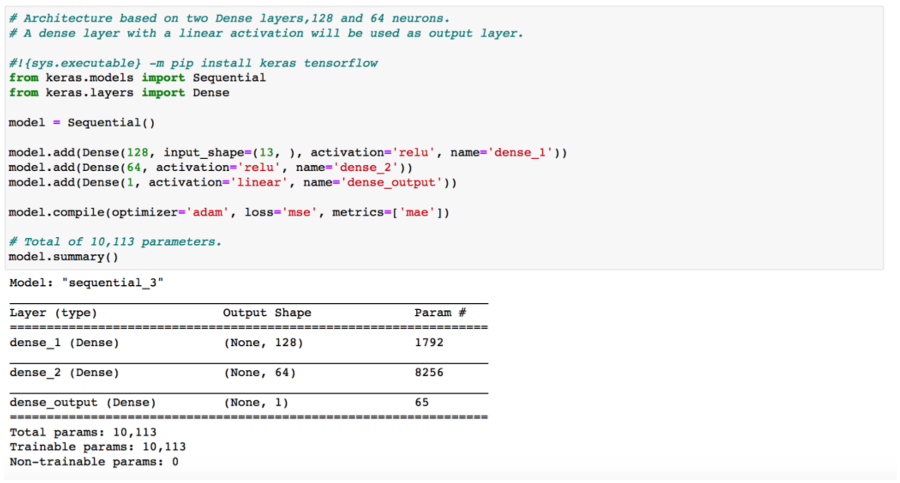


```
# Perform this normalization by subtracting the mean from our data and dividing it by the standard deviation.
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)

X_train = (X_train - mean) / std
X_test = (X_test - mean) / std
```

Image 5: Normalizing the data.

I chose to adopt an architecture based on two “Dense layers”, the first with 128 and the second with 64 neurons. Dense layers are your typical layers where every neuron from the previous layer is connected to every neuron of that dense layer. Hence making it a dense connection as opposed to a sparse connection like in a convolutional neural network. This produced a total of 10133 parameters.



```
# Architecture based on two Dense layers,128 and 64 neurons.
# A dense layer with a linear activation will be used as output layer.

#!{sys.executable} -m pip install keras tensorflow
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

model.add(Dense(128, input_shape=(13, ), activation='relu', name='dense_1'))
model.add(Dense(64, activation='relu', name='dense_2'))
model.add(Dense(1, activation='linear', name='dense_output'))

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Total of 10,113 parameters.
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               1792
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_output (Dense)         (None, 1)                 65
=================================================================
Total params: 10,113
Trainable params: 10,113
Non-trainable params: 0
```

Image 5: Setting up Dense layers with 128 and 64 neurons.

I then fitted my model with both labels and features for a total amount of 100 epochs, while keeping apart 5% of the samples as a validation set. Plotting the mean average error and the loss of the model shows that the model was capable of learning patterns in the data without overfitting (which might sometimes be a problem when the datasets are too small).

```
# Plotting both loss and mean average error.
fig = go.Figure()
fig.add_trace(go.Scattergl(y=history.history['loss'], name='Train'))
fig.add_trace(go.Scattergl(y=history.history['val_loss'], name='Valid'))
fig.update_layout(height=500, width=700, xaxis_title='Epoch',yaxis_title='Loss')
fig.show()
```

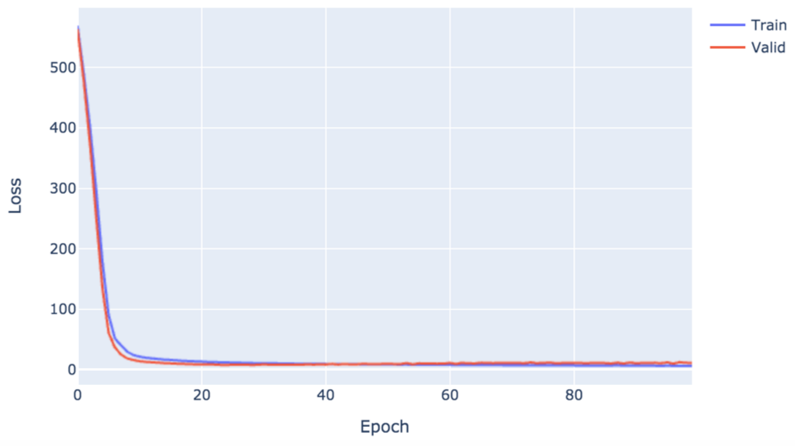


Image 6: Train and validation losses for our trained model.

```
fig = go.Figure()
fig.add_trace(go.Scattergl(y=history.history['mae'], name='Train'))
fig.add_trace(go.Scattergl(y=history.history['val_mae'],name='Valid'))
fig.update_layout(height=500, width=700, xaxis_title='Epoch', yaxis_title='Mean Absolute Error')
fig.show()
```

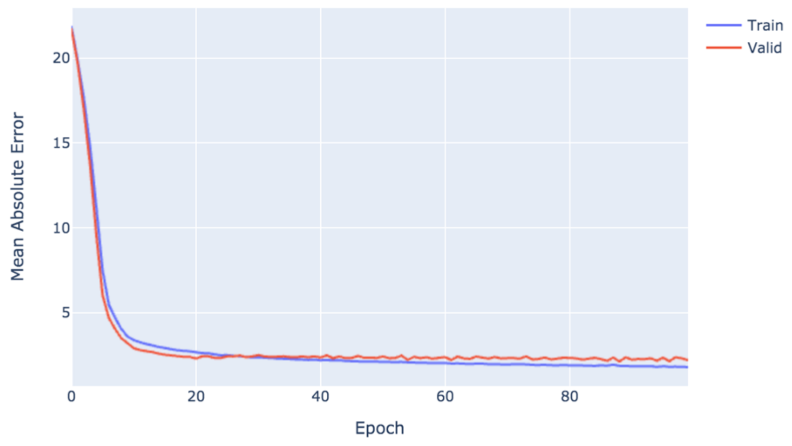
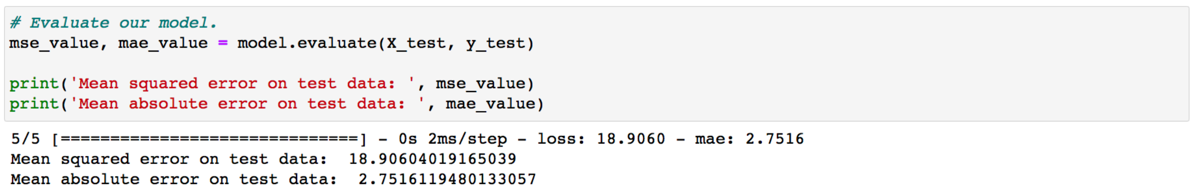


Image 7: Train and validation Mean Absolute Error for validation and training sets.

Results

Lastly, I had to assess if the model was capable of working on with real data, which in this case meant the test set. I used the models evaluate method with the data features and the and the test set target. The model produced an MSE (mean square error) of 18,9 and MAE (mean absolute error) if 2,75. What this means is that the model by average can guess the price of a given house with an accuracy of about 3k, which in this case can be seen as excellent results.



```
# Evaluate our model.
mse_value, mae_value = model.evaluate(X_test, y_test)

print('Mean squared error on test data: ', mse_value)
print('Mean absolute error on test data: ', mae_value)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 18.9060 - mae: 2.7516
Mean squared error on test data:  18.90604019165039
Mean absolute error on test data:  2.7516119480133057
```

Conclusion

Using deep learning and neural networks can be a powerful tool when used correctly, and they can easily outperform traditional Machine Learning methods by a good margin. The models that I ended up creating turned out to be very accurate and could probably be used in a real-world scenario if the data was scaled to prices of today's housing market. I had never worked with deep learning or neural networks before, so I can definitely say that I've learned a lot. All in all this assignment was both fun and challenging, and I look forward to applying my newly acquired knowledge in the future.