KING COUNTY HOUSE SALES PRICE PREDICTION
REPORT

Prepared for: COMP534 - Applied AI 2021

Prepared by:

Aditya Naik
Student ID: 201574102
sganaik@liverpool.ac.uk

Reza Fazli
Student ID: 201538160
sgrfazli@liverpool.ac.uk

Yuzhe Qiu
Student ID: 201535477
Y.Qiu30@liverpool.ac.uk

Introduction

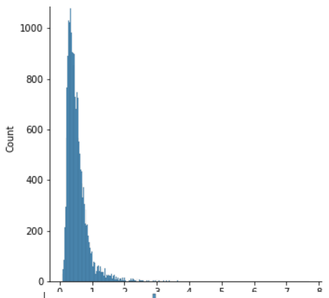
The project aims to answer the question "What will be the price of a house in the King County region?
For our study, we will be using the **House sales in King County, USA dataset**. This data is to explore the relationship between home prices in King County and the various attributes in the dataset between May 2014 and May 2015. We will then make use of this information to build a neural network to perform a regression task of predicting house prices. The complete model and analysis were implemented in the Google Collaboratory environment. A brief description of each variable is provided for those who find the names ambiguous.

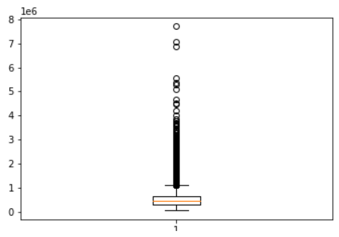
Risk assessment:

Predictive analytics projects involve a range of risks, based on the type of analysis conducted. Overfitting could be a significant problem since many variables could be considered while training the model. It is important to recognise trends in variables such as the city's population, geographic location and jobs forecasted to be able to accurately forecast prices, which may be hard to incorporate into the analysis as we do not have access to this information. Given this fact, we will make necessary assumptions and identify the critical indicators that will drive our analysis.

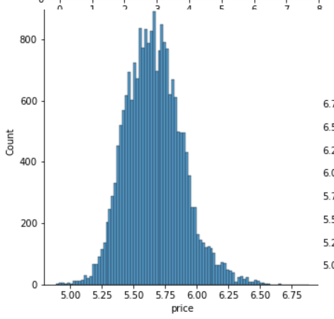
Exploratory Data Analysis:

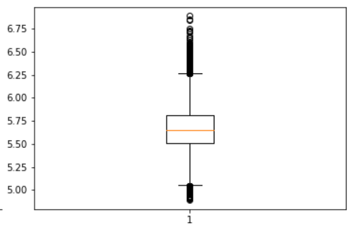
Price (outcome)



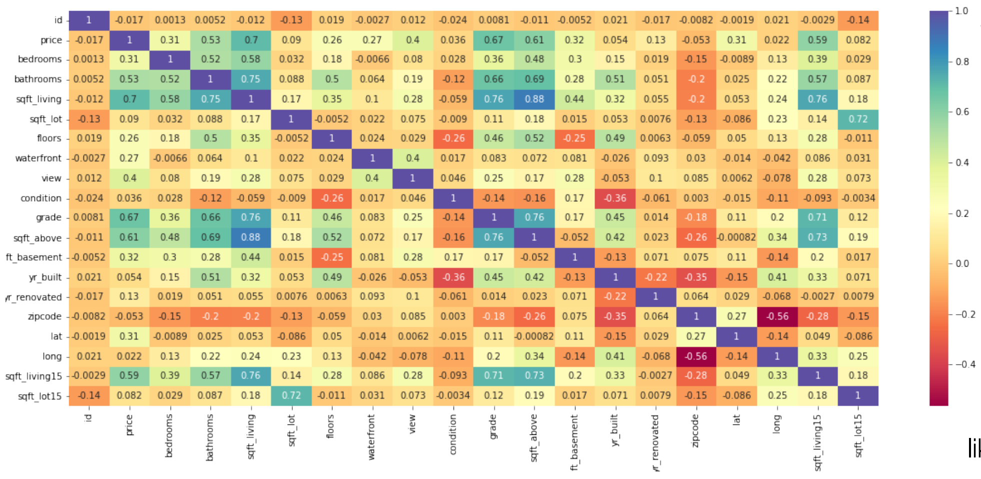


The purpose of this analysis is to forecast house prices. Price variables are highly skewed to the right in the real estate market, with their highest value being more than 10 times greater than the 3rd quartile:





It is essential that we shall not remove these properties of high value from the dataset since they are examples of the luxury market in the region and not anomalies. Data can easily be rebalanced by transforming it with a logarithm to base 10. Our models could be improved by this transformation.



According to the **Correlation Matrix** between the numerical features of the dataset, there is a correlation between the price and:

sqft_living: An increase in living space logically implies a price increase.

sqft_living15: The living for 15 nearest neighbours areas is also linked to the current price. Recent renovations have modified the living area, but the house likely stays in a similar price range.

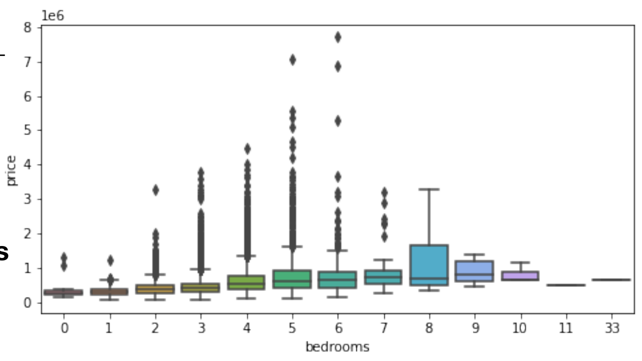
sqft_above: Is defined as the area above ground. It's not surprising to see a correlation with the price as well.

bathrooms: There is a slight correlation between the number of bathrooms and the price.

Some other relationships can be identified:

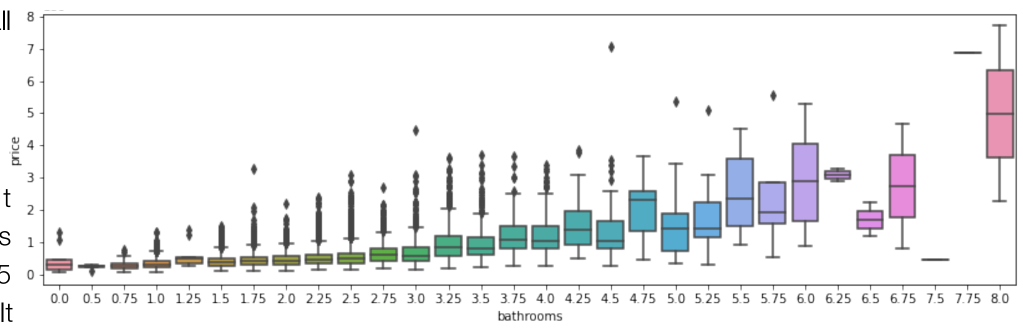
-The sqft_features are linked together, as sqft_living = sqft_above + sqft_basement, and the lot areas are rarely changed within a year.

-Zipcodes and longitudes seem to be related to each other, probably because of the way zip codes are designed in the area.



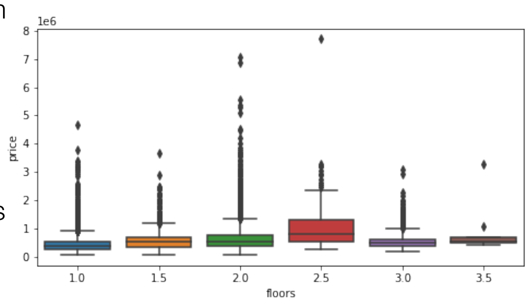
Here are the following charts that shed light on each factor's structure:

-A majority of houses in the region have 3 or 4 bedrooms. An overall trend shows that more bedrooms mean higher prices.



-The number of bathrooms is difficult to understand without knowing the meaning of the decimals and so we will assume that 0.5 means a toilet without a bathroom. It seems that most houses have values of 2.5 and 1.

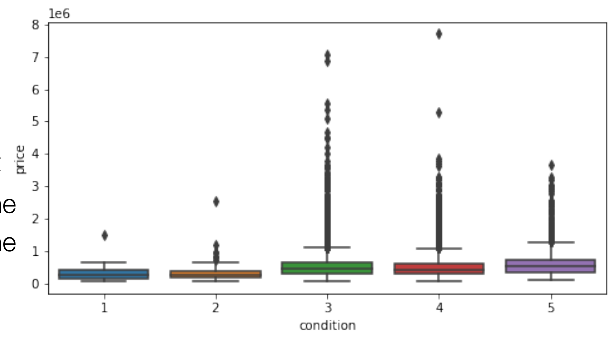
-Most of the houses in the region are in an average condition 3, with 26% of the houses being in a great condition 4, and 8% being in an exceptional condition 5. But the condition does not seem to have a high impact on prices.



-The real estate market in the region seems to offer almost as many houses with two floors as with one floor.

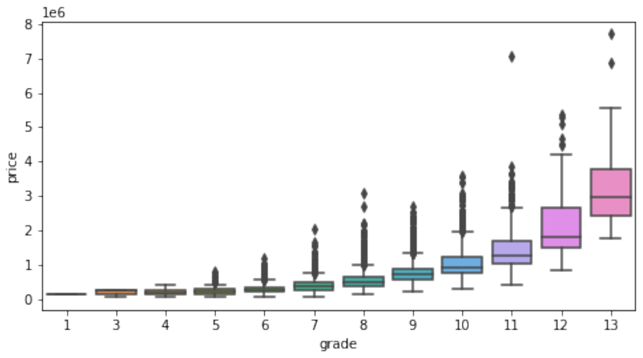
-Houses with a view of the waterfront are more expensive than the ones without.

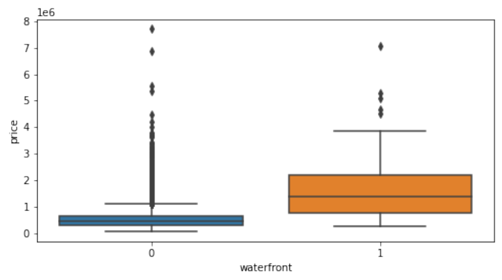
-The grade distribution is similar to the condition (with more levels), but the relation to price looks clearer.

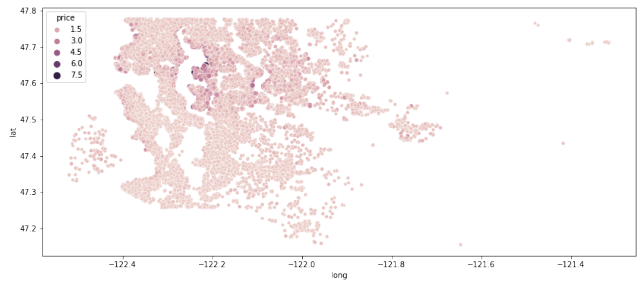


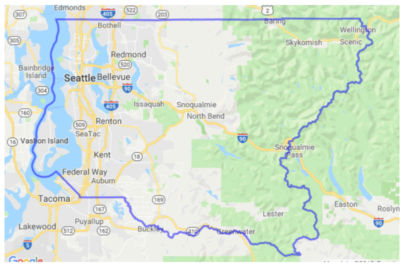
-There are more properties in the North-West of the region, based on latitude and longitude distribution.

-There seems to be a decent number of houses in the dataset across the zip code range, but without a good understanding of the region and zip code structure, we can't comment on this at the moment.







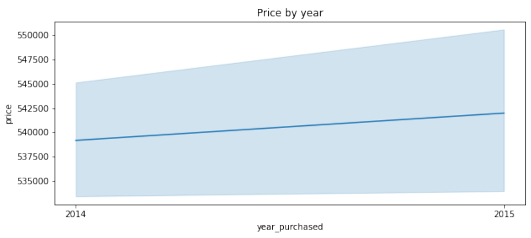


Feature Engineering [3][4]

Latitude and longitude

On the map and the spatial plot, we can see that the prices of the houses tend to be higher in Bellevue and Redmond in Seattle which overlooks the water body as compared to the ones located farther away. Most probably, this is the main city centre, indicated by the dense roads and grey colour indicating the presence of a large number of buildings as opposed to the interior of the map which appears to have more vegetation. The distance from Seattle appears to play a significant role in judging the price of a house, so we introduce a new variable called distance_from_seattle into the training data.

Date



The date of selling the house is represented by this date variable. Here, we will only look at the year, as it is an important factor in judging the price of a house. Using this variable, we will extract the year and store it in a variable called ‘year_purchased’.

Bathrooms, Bedrooms and Floors

Considering the bathrooms, There are several values below one, which do not make any sense, we remove all those instances where the number of bathrooms is less than one. The correlation heat map indicates that this variable is highly correlated with bedrooms, floors, sqft_living, grade, sqft_above and sqft_above15. The bedrooms also appear to have some anomalous values like 33, and 11 which we will eliminate. We will not delete zero values for bedrooms, which represent studio apartments. Likewise, the Floor attribute does not seem to provide much information about the price. To resolve this issue, we will create a new variable using floors, bedrooms, and bathrooms to reduce the multicollinearity of bedrooms and bathrooms (bath/floor' and ‘bedrooms/bath’).

Grade

The grade of the building seems to have an impact on the cost of a house which is shown by the increasing level of the median for each category. But at the same time, it is also highly correlated to the sqft_above variable (77%) we will try to modify this a bit in an attempt to reduce its multicollinearity. In the modified attribute:1 : Bad quality, * : Below average, * : Good , * : Best

Year Renovated

Traditionally, it has been observed that a newly renovated house will cost more than one that was renovated years ago. Thus, we took this attribute and turned it into an ordinal variable that ranges from 0 to 7 depending on how close the year of the renovation was to the year of sale.

Data Preparation

It appears that very few houses have changed hands more than once during the period, so the id and date should not be considered in this analysis.
In the context of house prices, location is one of the most important factors, so Zip code is a useful variable in that sense. It is a categorical variable that will have to be converted into one-hot vectors by using pandas dummies.
In the end, based on the feature engineering and analysis we did and the correlation matrix, we concluded that the best features that are worth capturing are Distance_from_Seattle, lat, long, view, sqft_above, bath/floor, grade, year_purchased, new_renovated, sqft_basement, bedrooms/bath, dummy zip codes.

Libraries used

The Pandas library was implemented to load the CSV data as a DataFrame and manipulate the data. The NumPy library was used to make calculations and manipulate the data arrays. Matplotlib and Seaborn for plotting. The sklearn library was the most extensively used right from the StandardScaler to using RandomizedSearchCV for hyper-parameter tuning. The TensorFlow and Keras libraries were used to build the neural networks.

Hyper-parameter Tuning

The hyperparameters were tuned using sklearn’s RandomizedSearchCV. The hunt for the best values was done by inputting the model and a dictionary of possible values in the RandomizedSearchcv object such that while searching for one hyper-parameter the others were kept fixed enabling us to get a good understanding of how the model’s performance changes based on the parameter alone. Among the list of values for the hyper-parameter being tuned the n_iter value was set to 4 which means out of the given list of possible values, 4 will be sampled at random. Based on these 4 values the models were tested using 3-fold cross-validation with negative mean absolute error as the scoring function. But, since the neural network was built using TensorFlow and Keras we will be making use of the KerasRegressor object which is a thin wrapper around the Keras model that will enable us to use sklearn’s RandomizedSearchCV. The search was performed and evaluated for only 15 epochs and a default batch size of 32.

Parameters selected:

Activation functions: [tf.keras.layers.PReLU(), tf.keras.layers.LeakyReLU(),'sigmoid', 'tanh', 'relu', 'elu', ‘selu']
Number of hidden layers : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Neurons per layer: [79, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140]
Optimisers : ['Adam','RMSprop', 'Adagrad', 'Adadelta', 'Adamax', ‘Nadam']
Loss function: ['mse','mae', ‘huber_loss']
Batch size : [5, 10, 20, 32, 40, 50, 60]

Initial learning rate : [0.1, 0.01, 0.001, 0.0001]
We will not be using a constant learning rate as this might cause the model to train very slowly if it's small or cause it to oscillate and never converge to the global minimum. To avoid this we will be using a learning rate scheduler [9] while training the final model. This helps take advantage of the high initial learning rate at the beginning and progressively drops its value as it progresses resulting in a fine-tuned search for the minimum [4].

Epochs: For tuning the epochs we will be using the Early stopping callback while training the final model based on the above-tuned hyper-parameters. We will choose a sufficiently high number of epochs and start training the model with the early stopping callback with patience = 10 indicating that if the validation loss doesn't improve for 10 successive epochs the training will stop. This will help us save time and resources that would be lost in using grid search or Randomised search for training a large number of epochs [4].

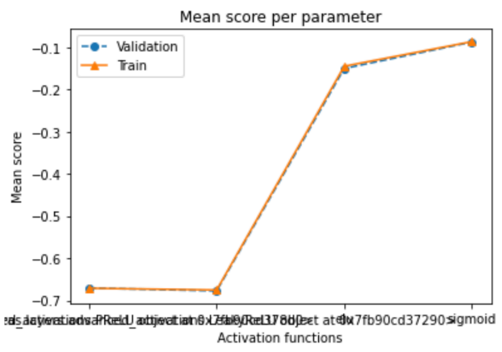
Training and Testing process

For the purpose of training and testing the cleaned dataset was shuffled and split using sklearn’s train_test_split method where the training data had about 80% of the shuffled data points while the test data made up 20% of the dataset. The test data was left untouched until the final model was built and trained. The training set was further shuffled and split using the same method into the final training data and validation data in the same ratio. This validation data was set aside to be used while training the final model as we will be using callbacks for tuning the epochs which requires validation loss to be monitored.

Evaluation

The model was developed by implementing a function with default parameters which we estimated to be a good model to start with. As we go further we will be turning all the parameters one by one using Randomised search Cross-validation.
To build this neural network we will be using batch normalisation [7] which normalises the inputs before sending them into the next layer. This tends to speed up training by avoiding the vanishing gradients problem [4]. We will not be using this method for layers with the self-activation function as these layers are self-normalising. In addition to this, we also use the He_normal initialisation strategy for ReLU activation functions and its variants [10][4] and the Glorot initialization for sigmoid and tanh functions [4] and LeCun initialisation for the selu function [4] instead of random as random initialisation tends to bring in instability at the early stages of training resulting in longer training times and causing the network to get stuck in local optima [4].

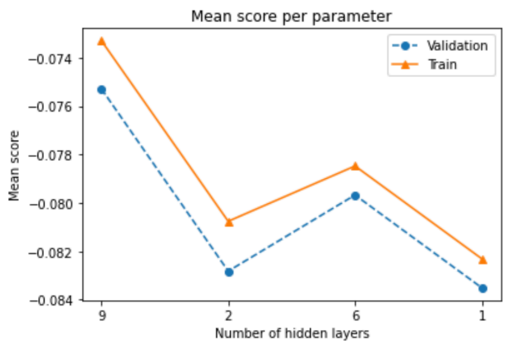
Activation Function



Selected parameter: 'sigmoid'
The sigmoid function is normally not very popular due to the fact that in the long run, it saturates to 0 or 1 which happens due to the fact that the mean value of the sigmoid function is 0.5 and not 0 which tends to push the input further and further as we go deeper in the layer causing it to saturate [6]. In this condition, no learning can take place or even if it does it will happen very slowly as the derivative in the saturated region is zero [4].

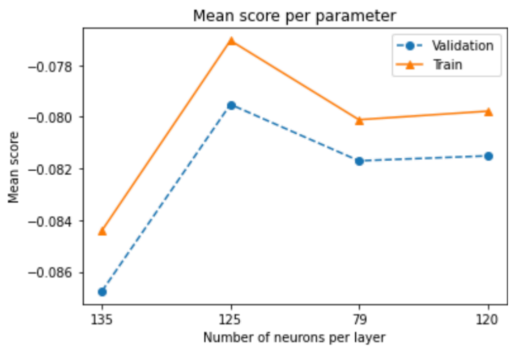
We alleviate this problem by making sure that the variance of the outputs of each layer should be equal to the variance of its inputs. This is done by using the glorot initialisation at the beginning of the training and by the batch normalisation layer we used in the build function. This special layer performs normalisation on the outputs of one layer before feeding it to the next layer thereby restricting the variance and eliminating the 'dead neuron' effect [4].

Hidden Layers



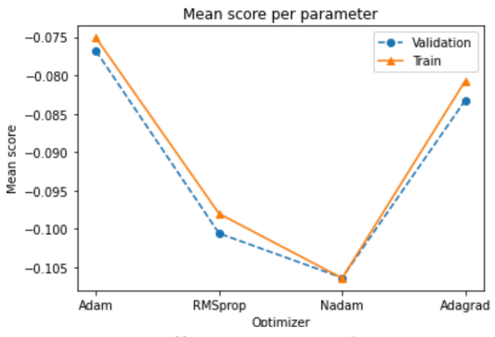
The higher the number of hidden layers in the network enables the model to detect more patterns and hence learn better which is why among the chosen values for the number of layers 9 was selected as it helped the model learn more important information which was not possible with a shallower network [4][5]. But at the same time, a very deep network may cause overfitting and also increase the computational time and resources [5]. We select 9 because it has the highest validation and training score with no signs of overfitting as the difference between the scores is not high.

Number of neurons per layer



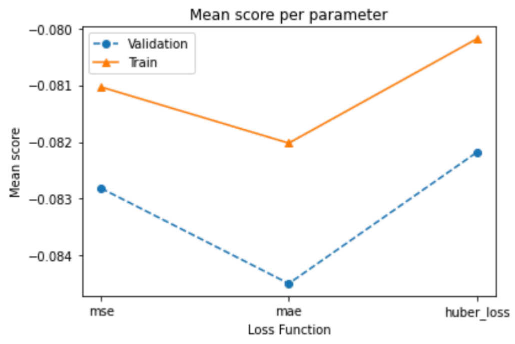
Similar to the number of layers, the number of nodes in a layer is also important. Too many could overfit the data by detecting unnecessary patterns among the variables itself while too few tend to result in a high bias due to under-fitting[4][5]. In this case, we choose 125 as it has the highest training and validation score compared to the others.

Optimisers



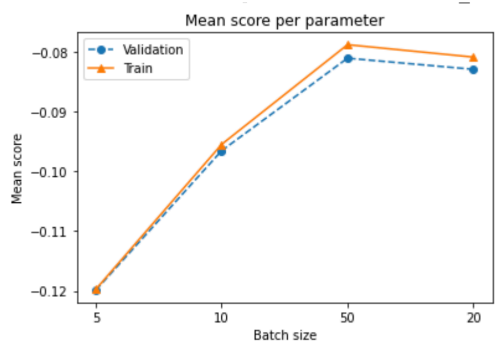
Adam is the best optimiser for this task as is indicated by its high training and validation score. It has the advantage of using the idea of momentum and also keeps track of an exponentially decaying average of past gradients which helps in quick optimisation 8[]. Adam tends to combine the benefits of RMSprop and AdaGrad and can handle sparse data well [1]. Since our data is also sparse owing to a large number of one-hot vectors it's no doubt that Adam has the best performance.

Loss Function



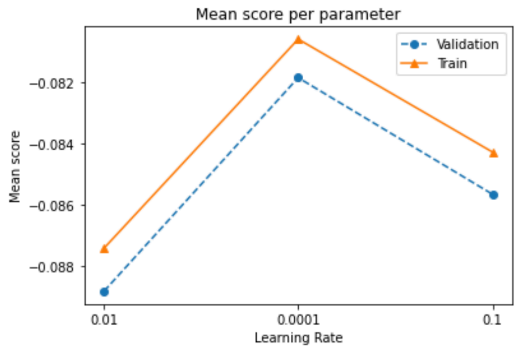
The mean squared error is a good loss function where outliers are important as even if the model makes a single wrong prediction it will shoot up the loss as it is squared. Mean absolute error on the other hand gives equal weightage to the regular values as well as outliers by taking the absolute difference and averaging it which ignores the effect of the outliers [2]. As we discussed before our dataset has a few important outliers indicated by the luxury houses sold at high premium prices while most of the other houses are regular houses so we would like to give some importance to the outliers but not too much importance. In this case, the Huber loss is the best function as it takes the advantages of both the MSE and MAE by introducing a loss threshold value above which MAE is used otherwise MSE is used resulting in a higher training and validation score.

Batch Size



Larger batch sizes have the benefit of taking advantage of hardware accelerators like GPUs as long as they can fit in the memory. This implies the model can see more training instances per second. But very large batch sizes could induce instabilities at the start of training [4]. Plot 50 has the highest training and validation score owing to the fact that the code was run in the Google Collaboratory environment which makes use of Tesla GPUs. Generally, the best default value for this is 32 [4] but since our Random search didn't select it among the candidates we will stick to 50.

Initial Learning Rate [9]

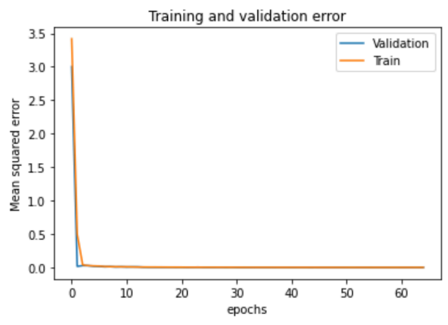


We will be using a learning rate scheduler while training the final model [4]. This helps take advantage of the high initial learning rate at the beginning and progressively drops its value as training goes on resulting in a fine-tuned search for the minimum. The best initial value seems to be 0.0001 which is the smallest value among the chosen ones with the highest training and validation scores. The reason for this is as we are using the prices on the log 10 scale the values vary from just below 5.0 to about 7.0 which is quite a short range and would result in the error values having an even smaller range. Using a larger learning rate with such small errors would never allow the model to converge to the global minimum quickly, it would have to wait for the learning rate scheduler to reduce its value and only once it reaches that level would it start moving towards the minimum resulting in an extended number of epochs.

Epochs

For tuning the epochs we will be using the Early stopping callback while training the final model based on the above-tuned hyper-parameters. We will choose a sufficiently high number of epochs i.e. a value of 150, and start training the model with the early stopping callback with patience = 10 indicating that if the validation loss doesn't improve for 10 successive epochs the training will stop. This will help us save time and resources that would be lost in using grid search or Randomised search for training a large number of epochs [4].

Final Model



For the final model we choose the following parameters:

Hidden layers = 9,
Neurons per layer = 125,
Activation function = 'sigmoid',
Optimiser = tf.keras.optimizers.Adam(),
Loss function = 'huber_loss',
Initial Learning rate = 0.0001,
batch size = 50

The final model was trained for about **77 epochs** and was stopped by the early stopping callback as validation performance did not seem to improve much after this. The total training loss is 0.0038, the validation loss is 0.0035 and the final learning rate was **3.9e-07**. The above plot indicates the training and validation loss decreased together throughout the training and there are no signs of any under-fitting or over-fitting. This would be otherwise indicated by diverging training and validation loss curves in case of overfitting or a high training and validation loss at the end of the epochs.

Model evaluation

Upon evaluating the model on the test set the loss was 0.0036

But what does this mean in terms of the real world?

If we recall our target variable, the price was converted to the log 10 space to deal with the problem of the luxury houses. So now we test our model on previously unseen data and find out what would be the actual difference between the price predicted and the actual price of the house. So we convert the predictions and the labels back to the regular prices from the log10 space and the average error between the predicted prices and the labels was found to be **$78985.47.**

Conclusion

So our models predicted prices seem to charge clients $78985.47 more than the actual cost. This shows that our model's performance is not very good as it would overcharge future customers resulting in a loss of business. Generally, machine learning algorithms are subject to both reducible and irreducible errors. Regardless of the algorithm used, there is an irreducible error or bias. Bias describes the difference between true values and predicted values according to a model based on a training sample. The variance is caused by the sensitivity of a model to small fluctuations in the training set. While we have managed to greatly reduce the effects of high variance, the effect of high bias still remains which is most likely due to the fact that, in this project we did not consider the effects of inflation, fluctuating bank interest rates for home loans, and crime rates in the area which could affect the prices of accommodation in the region. This could be one of the reasons for the disparity in the predicted cost value The best option, in this case, would be to get more data or to get more features considering the above while collecting the data which would have a higher impact on the price prediction. Additionally a much deeper network could be explored to see if it has an influence over the prediction accuracy.

Challenges of the project

The main challenge of the project was deciding on the relevant features which would directly impact our models' performance. We had to spend a considerable amount of time manipulating the variables and transforming them in order to provide more meaningful information to the model. The next major task was in using the randomised search for hyper-parameter tuning since we were using only the basic version of Google Collaboratory it was quite time-consuming due to the high level of computation and resources required.

Task allocation

Aditya Naik: Data cleaning, Feature engineering, Activation function, Batch size, Loss function, Final model implementation and evaluation, Training and validation plots, Report writing
Reza Fazli: Data analysis, Data cleaning, Data visualisation, Feature engineering, Hidden layers, Optimiser tuning and analysis, Report writing
Yuzhe Qiu: Hidden units, Learning rate and epochs tuning and analysis.

References

[1] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[2]https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3

[3]Alice Zheng, Amanda Casari, "Feature Engineering for Machine Learning by", O'Reilly Media, Inc.

[4] Aurelion Geron, E 2020," Hands-on machine learning with Scikit-learn, Keras and TensorFlow", O'Reilly Media, Inc

[5] https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

[6] Glorot, X. and Bengio, Y. (n.d.). Understanding the difficulty of training deep feedforward neural networks. [online] Available at: https://homl.info/47.

[7] Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*. [online] Available at: https://homl.info/51.

[8] Kingma, D.P. and Ba, J. (2017). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. [online] Available at: https://homl.info/59.

[9]Senior, A., Heigold, G., Marc', A., Ranzato, K. and Yang (n.d.). AN EMPIRICAL STUDY OF LEARNING RATES IN DEEP NEURAL NETWORKS FOR SPEECH RECOGNITION. [online] Available at: https://homl.info/63.

[10] He, K., Zhang, X., Ren, S. and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*. [online] Available at: https://homl.info/48.