UNIVERSITY OF LIVERPOOL

COMP 534 – Applied Artificial Intelligence

Assignment 2 - Neural network for solving a regression problem and testing multiple hyperparameters to obtain the most appropriate model

Submitted by,

Name	Student Id	Email
Alwin Joseph Christopher	201594340	sgachri4@liverpool.ac.uk
Mohamed Muradh Maricair	201602133	sgmkader@liverpool.ac.uk

1. Introduction:

The assignment aims to train a neural network to solve regression problems. We use Kaggle's Housing Price Prediction data. We created and trained our neural network with Keras and tested multiple hyperparameters by tuning throughout development to obtain the best network for our problem.

1.1 Libraries Used:

We used the below libraries to realise the Neural network models.

- Keras
- Pandas
- NumPy
- Matplotlib
- Seaborn
- Torch

1.2 Strategies:

- Our idea is to fine-tune one hyperparameter by keeping all other parameters fixed and observing the changes.
- We plotted loss function vs epochs for each tuning of the hyperparameter and chose the parameter which produces minimum loss
- We also calculated the R2 score for each parameter and then tuned the other hyperparameter by keeping the current one fixed.
- Instead of using grid search, we executed the model with different parameters to understand how the model behaves for each hyperparameter tuning and plotted the graph to find the best parameter.

1.3 Values of Hyperparameters tested:

Hyper Parameter	Different Values Tuned
Batch size	25,50,75
Optimizers	Adam , Adagrad, SGD, RMSprop
Learning rate	0.001, 0.01, 0.1
Activation function	relu, linear, LeakyRelu
Loss function	Mean Absolute Error, Mean Squared Error
Hidden Layers	1, 2, 3, 4, 5
Neurons	8, 16, 32, 64, 128

2. Data Analysis and Visualization:

In the Data visualisation and analysis stage, we use a different way to visualise data. Find the relation between independent and dependent (sales price) variables to make an informed decision for the feature engineering process.

- First, we loaded data from a CSV file downloaded from Kaggle (Housing Price prediction), using pandas
- Using IsNull() function in pandas, we verified if any null values or NAN values were present in our dataset, as they affect the final result of the model.
- Viewed the distribution of each feature of data. To know the count, mean, median and std of features

2.1 Pearson correlation matrix

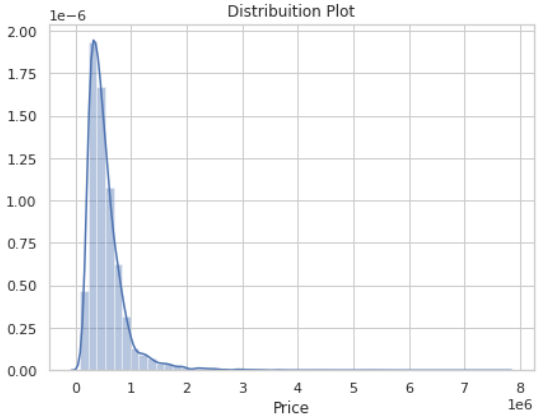
The correlation matrix heatmap can be viewed from *section 2.1* of python notebook (going forth, we will include section number, which means a section in the python notebook).

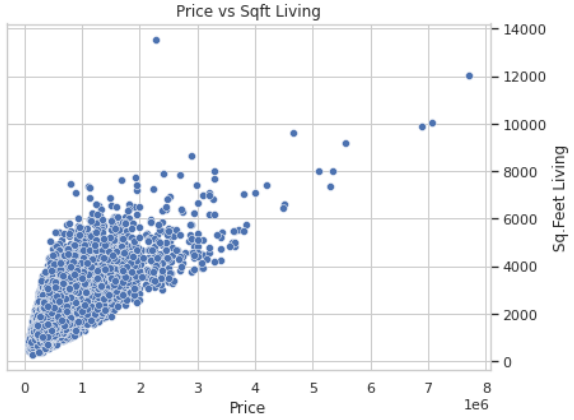
We use the Pearson correlation coefficient to examine the strength and direction of the linear relationship between two continuous variables. The correlation coefficient can range in value from −1 to +1. The larger the absolute value of the coefficient, the stronger the relationship between the variables. An absolute value of 1 indicates a perfect linear relationship for the Pearson correlation. A correlation close to 0 indicates no linear relationship between the variables. The sign of the coefficient indicates the direction of the relationship. If both variables tend to increase or decrease together, the coefficient is positive, and the line that represents the correlation slopes upward. If one variable tends to increase as the other decreases, the coefficient is negative,

and the line that represents the correlation slopes downward. From the heatmap, we can see sqft_living has a correlation coefficient of 0.7.

2.2 Distribution plot of Price and square feet vs price

We plotted the price distribution (section 2.2); from the below graph, we understand that most data lies in the price range of 0 to 3 million, and a few outliers till 8million. The same can be understood from square feet living vs price plot.



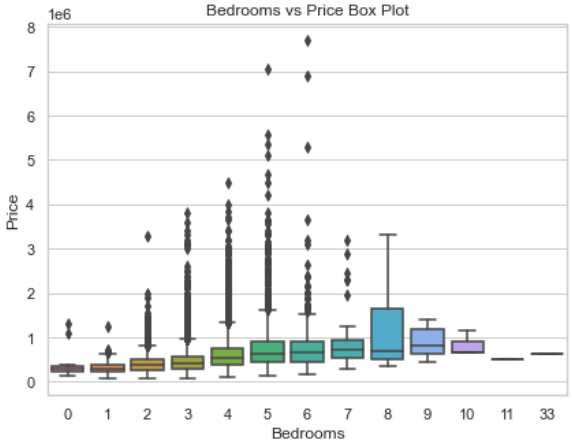


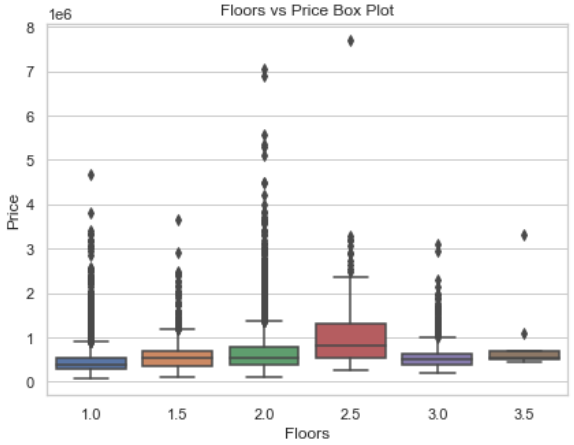
2.3 Count plot of zip code

We did a count plot of zip codes to understand the number of houses present in each area and their effect on the price of the house. In general, house price can change based on where it is located. The image can be viewed in notebook *section 2.4*.

2.4 Boxplot

We did boxplots for number of bedrooms vs price and # of floors vs price. This shows for each discrete value of bedroom and floors price ranges from 0 to 3 million.





3. Feature Engineering:

This section discusses how we modified feature information based on our earlier visualisation.

- Since most of our price ranges from 0 to 3 million, we drop the objects with a cost of more than 3 million. This gives us normally distributed data without outliers. (*Section 3.1*)
- We drop the id column, as it does not impact price information.
- Based on the correlation plot, we removed latitude, longitude, and DateTime features. Also, latitude and longitude feature & zip code has the same relation with house price.
- We dropped DateTime and converted it to the year and month feature. Using this, we found trends in price based on the month. (*Section 3.3*)
- Since the apartment area affects price, we performed one-hot encoding for the zip code feature.
- Year of renovation is essential in determining the price, but only 4% of data contains information and the value is 0 for the rest. So, we changed to the value of 1 for all the renovated houses irrespective of their year.

- From the high-level view of data, we saw bedrooms and floors had values with decimal values like 1.5, 1.75, etc., which is not the general way of describing bedrooms and floors. They should be discrete numbers. Hence, we rounded it up to ceil.

4. Training and Testing process:

In this section, we discuss the process of splitting training data, scaling, model implementation procedure, steps followed in tuning the hyperparameter, and evaluation technique used.

4.1 Training set

- Data has been split into 70:30 ratio. Before splitti,ng we performed feature engineering on the dataset.
- We used Min Max scaler to scale our features, both training and test set.

4.2 Model Implementation

- We initially built the model in PyTorchorch library but later moved to Keras. Though PyTorch is faster than Keras, due to its high-level framework, Keras was easier to build, train and evaluate the model
- We created a generic function to initialise Keras model based on hyperparameters given. This function creates a new model, initialize optimizer and learning rate. Add hidden layers and neurons for each layer. Compile the model with metrics MAE and MSE. (*Section 5.1*)
- Early stopping is added to the model with the patience of 10 epochs; if there isn't any considerable change in the metrics or loss, model training is stopped
- We initially build neural network with 3 hidden layers, and neurons of same size (32) as the number of features and fully connected with the output layer of one neuron. Relu is the activation function used for hidden layers and the final layer. We used adam optimizer with a learning rate of 0.01, mean squared error as the loss function, and ran the model for 50 epochs with a batch size of 100. We observed that the model was overfitted with the loss value obtaining plateau in the 3rd epoch and validation loss was poorer compared to train loss.
- Hence, we added Dropout and batch normalization to the network, and the model improved significantly.
- We fine-tuned each hyperparameter using our strategy to find the best parameters
- Plots the EPOCH Vs Losses for each model trained. It plots both loss and validation loss. Returns us with a model and history of losses and metrics.

4.3 Evaluation methods

We used two approaches to find the optimum or best value for a hyperparameter.

- Plot Epoch vs Losses and min loss of a model vs hyperparameter values. We will understand how losses are reduced over epochs when value is chosen for a hyperparameter from these graphs. Then we get the minimum loss obtained over the epochs in a model and it is plotted against its hyperparameter value, which gives final view of the best conditions.
- For each model, we predict using test data and find the r2 score. It is also termed R squared or the coefficient of determination is a statistical measure of fit that indicates how much variation of a dependent variable is explained by the independent variable(s) in a regression model. R2 score ranges from 0 to 1. Lower the value of R2, the model is worst fit and 1 being the best fit.

5. Evaluation:

Here we discuss hyperparameter tuning, r2 score and losses obtained for each value

.

5.1 Epoch vs Losses

Layers	Neurons	Activation (hid)	Activation (output)	Optimizer	Loss	Batch Size	Epoch	Learning Rate	R2 Score
2	16	Linear	Relu	Adam	MAE	25	100	0.01	0.684

We initialized our model with 2 layers with 16 neurons each, hidden layer used linear activation function, relu for output layer, optimizer adam with learning rate 0.01, loss MAE with batch size 25 and 100 epochs. With this setup we obtained elbow at 20 and 40 epochs, post which there isn't much change in the losses. We found our model obtained R2 score of 68.4%. *(Graphs for this evaluation is in Section 6.1)*

5.2 Batch size vs Losses

Layers	Neurons	Activation (hid)	Activation (output)	Optimizer	Loss	Batch Size	Epoch	Learning Rate	R2 Score
2	16	Linear	Relu	Adam	MAE	25	100	0.01	0.679
2	16	Linear	Relu	Adam	MAE	50	100	0.01	0.684
2	16	Linear	Relu	Adam	MAE	75	100	0.01	0.503

Batch size of 25,50,75 is tuned with parameters used in prior model. For batch size of 25, our model learned gradually and obtained 67.9 % r2 score. Whereas batch size 75 over 100 epochs finds difficult to obtain local or global minimum and their r2 score dropped drastically to 50.3%. As the batch size increases the losses increased and R2 score decreased, it shows model couldn't learn efficiently if batch size is larger. Hence, we fixed batch size as 50 for upcoming parameter tunings. *(Graphs for this evaluation is in Section 6.2)*

5.3 Optimizer and learning Rate vs Losses

Layers	Neurons	Activation (hid)	Activation (output)	Optimizer	Loss	Batch Size	Epoch	Learning Rate	R2 Score
2	16	Linear	Relu	Adam	MAE	50	100	0.001	-2.46
2	16	Linear	Relu	Adam	MAE	50	100	0.01	0.687
2	16	Linear	Relu	Adam	MAE	50	100	0.1	0.697
2	16	Linear	Relu	adagrad	MAE	50	100	0.001	-2.62
2	16	Linear	Relu	adagrad	MAE	50	100	0.01	-2.62
2	16	Linear	Relu	adagrad	MAE	50	100	0.1	-2.27
2	16	Linear	Relu	SGD	MAE	50	100	0.001	0.668
2	16	Linear	Relu	SGD	MAE	50	100	0.01	0.667
2	16	Linear	Relu	SGD	MAE	50	100	0.1	0.735
2	16	Linear	Relu	RMSprop	MAE	50	100	0.001	-2.62
2	16	Linear	Relu	RMSprop	MAE	50	100	0.01	-2.62
2	16	Linear	Relu	RMSprop	MAE	50	100	0.1	-2.24

Here we tuned optimizer combined with learning rate in same model and obtained above results. As we can see since we fixed epoch as 100, for learning rate 0.001, for all optimizers, performs bad, as it did not reached local or global minimum, or model is not trained well, hence their coefficient of determination score is in negative. *(Graphs for this evaluation is in Section 6.3)* Based on our results, we find that SGD and Adam are the two best optimizers for our model. SGD gives us better score because it generalize better than Adam though it combines advantages of Adagrad and RMSprop computing individual adaptive learning rates for different parameters.

5.4 Activation Function vs Losses

Layers	Neurons	Activation (hid)	Activation (output)	Optimizer	Loss	Batch Size	Epoch	Learning Rate	R2 Score
2	16	Relu	Relu	SGD	MAE	50	100	0.1	0.730
2	16	Linear	Relu	SGD	MAE	50	100	0.1	0.680
2	16	LeakyRelu	Relu	SGD	MAE	50	100	0.1	0.715

We obtained relu as the appropriate activation function for the model with R2_score of 73%. We explored with LeakyRelu as it avoids the vanishing gradient problem occurring with relu, and we acquire 71.5% for coefficient of determination.*(Graphs for this evaluation is in Section 6.4)*

5.5 Loss Function vs Losses

Layers	Neurons	Activation (hid)	Activation (output)	Optimizer	Loss	Batch Size	Epoch	Learning Rate	R2 Score
2	16	Relu	Relu	Adam	MAE	25	100	0.01	0.700
2	16	Relu	Relu	Adam	MSE	25	100	0.01	0.736

Our results shows that Mean squared error perform better to Mean absolute error metrics as the R2_score for MSE Loss is 73.6% because MSE will punish the error with square of the error. *(Graphs for this evaluation is in Section 6.5)*

5.6 Hidden Layers vs Losses

Layers	Neurons	Activation (hid)	Activation (output)	Optimizer	Loss	Batch Size	Epoch	Learning Rate	R2 Score
1	16	Relu	Relu	Adam	MSE	25	100	0.1	0.646
2	16	Rel	Relu	Adam	MSE	25	100	0.1	0.775
3	16	Relu	Relu	Adam	MSE	25	100	0.1	0.680
4	16	Relu	Relu	Adam	MSE	25	100	0.1	0.771
5	16	Relu	Relu	Adam	MSE	25	100	0.1	0.729

Hidden layers establish the complexity of the model. Adding more complexity to the model might overfit the dataset. Hence, we experimented with maximum of 5 hidden layers, and we secured the best score for 2 hidden layers. *(Graphs for this evaluation is in Section 6.6)*

5.7 Neurons vs Losses

Layers	Neurons	Activation (hid)	Activation (output)	Optimizer	Loss	Batch Size	Epoch	Learning Rate	R2 Score
2	8	Relu	Relu	Adam	MSE	25	100	0.1	0.809
2	16	Relu	Relu	Adam	MSE	25	100	0.1	0.775
2	32	Relu	Relu	Adam	MSE	25	100	0.1	0.662
2	64	Relu	Relu	Adam	MSE	25	100	0.1	0.681
2	128	Relu	Relu	Adam	MSE	25	100	0.1	0.649

We investigated with various neuron size values belonging to exponentials of 2, and as we increased the neuron sizes, training losses decreased rapidly. This conveys it learned training data well and not generalised it for test data. Hence our best R2_score is obtained at 8 Neurons. *(Graphs for this evaluation is in Section 6.7)*

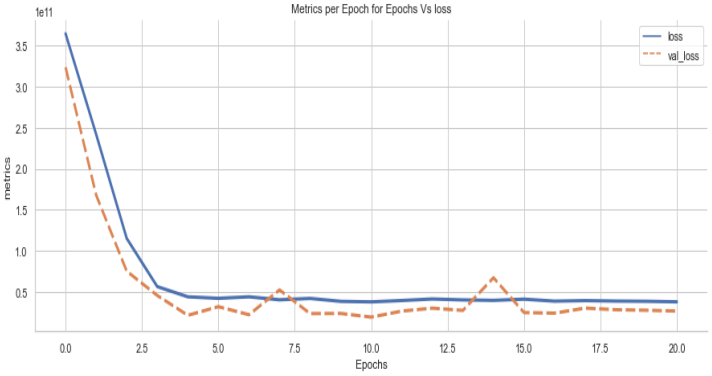
6. Best Parameter:

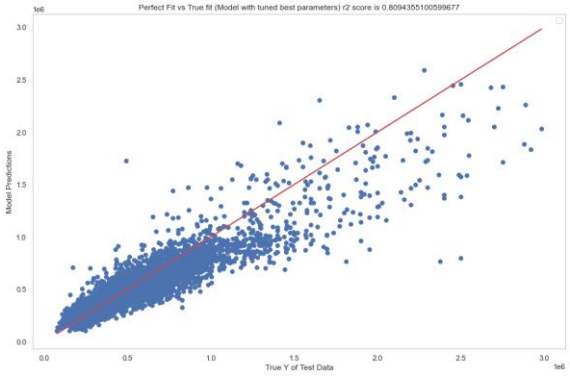
- For our best model, we obtained the R2_score of 80.9% *(section 7.2)*

Layers	Neurons	Activation (hid)	Activation (output)	Optimizer	Loss	Batch Size	Epoch	Learning Rate	R2 Score
2	8	Relu	Relu	Adam	MSE	25	100	0.1	0.809

From our extensive analysis of hyperparameter tuning, the values in the above table are the best parameters for the given dataset. In the Below graph, we can notice that validation loss is less than the training loss, and also model fitness plot shows the fitness for our predicted values(blue dot) lies close to the actual value(red line).

We applied Kfold cross validation for the best model and calculated the R2_score for each fold. *(section 7.1)*






```
------------------------------------------------------------------------
Score per fold
------------------------------------------------------------------------
> Fold 1 - Loss: 51678003200.0 - R2 Score: 47.6971110813361%
------------------------------------------------------------------------
> Fold 2 - Loss: 25822470144.0 - R2 Score: 75.67479132804435%
------------------------------------------------------------------------
> Fold 3 - Loss: 47220666368.0 - R2 Score: 55.701064328538116%
------------------------------------------------------------------------
> Fold 4 - Loss: 22557300736.0 - R2 Score: 79.11127474229599%
------------------------------------------------------------------------
> Fold 5 - Loss: 31989870592.0 - R2 Score: 72.90324154702728%
------------------------------------------------------------------------
> Fold 6 - Loss: 40014557184.0 - R2 Score: 62.48845570385222%
------------------------------------------------------------------------
Average scores for all folds:
> R2 Score: 65.59598978851567 (+- 11.295849333844863)
> Loss: 36547144704.0
------------------------------------------------------------------------
```

The final R2_score obtained for our model is 80.9%.
Hence these are the best parameters for our model.

7. Final conclusions:

Here we discuss about the problems we faced during the development process and mention the task allocated for our team members

7.1 Challenges Faced:

Initially, we implemented the neural network using PyTorch code, but we realised that the number of code lines would be very high when applying tuning for each hyperparameter. Therefore, we built the neural network architecture in Keras, where we could modify the parameters through the data-driven method.

Also, our model was overfitting for all the changes in hyperparameter, then we added dropout and batch normalisation to the network, and we were able to the model overfitting.

7.2 Task Allocated:

We worked together from the beginning during this assignment, discussed the strategy and ideas, and shared our knowledge towards solving every problem encountered in the development process. We always collaborated in person and used google colab for coding where both of us could code parallelly. Hence, we haven't singled out the task for each person. Instead, we completed all the jobs together.

References:

1. https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8 - for loss and activation function
2. https://machinelearningmastery.com/regression-metrics-for-machine-learning/ - Evaluation of regression problem
3. https://www.tensorflow.org/api_docs/python/tf/keras/utils/set_random_seed
4. https://keras.io/api/models/
5. https://keras.io/api/layers/
6. https://keras.io/api/metrics/