COMP70050 Coursework 2: Neural Networks Report

James Jun Hao Ong (CID: 01848230), Michal Palič (CID: 01942994),

Václav Pavlíček (CID: 01868933), Matthew Setiawan (CID: 01917068)

25 November 2022

1 Introduction

The goal of this project was the development of a model for the prediction of house prices in California from a set of input features. To these ends, a preprocessor, neural network architecture and optimizer parameters were defined and experimentally optimised.

2 Architecture for regression

2.1 Preprocessor method

The dataset consisted of columns with values spanning multiple orders of magnitude. Given this fact and the usage of tanh as the activation function, it was important to control the range of the input and output data. This normalization step presents benefits with regards to optimizer convergence speed [1]. During training, the minimum, maximum and average values per column were stored in the regressor object. These values were then used to scale the network inputs into the range [-1,1] for the training data and a similar range for the remainder. The network outputs a normalized value, for which the mapping needs to be reversed. The output normalization ensures that the output neuron's range is not exceeded given the previous layer's tanh activation function. One-hot encoding was used for categorical variables in the dataset as it provides a higher accuracy compared to other encoding methods [2].

Entries with missing elements were present within the dataset. The pre-processor identified these elements and replaced them with the mean of the training set entries from the appropriate column[3]. Other established approaches, such as the replacement of missing values using the k-nearest neighbor algorithm were considered, but ultimately abandoned due to the existence of reliable, less processing intensive methods.

Table 1 shows a sample of the unprocessed data and table 2 shows the same data entries after the application of our post-processing method. Samples in table 1 contain numerical values, missing data represented as NaN and a column that contains text label values. Table 2 shows the preprocessed data where all missing values were replaced with means of each column and the column with text labels was replaced with one-hot encoded values.

Longitude	Latitude	Housing Median Age	Total Rooms	Total Bedrooms	Population	Households	Median Income	Ocean Proximity
-117.61	34.13	21	8416	1386	4308	1341	4.461	Inland
-117.37	33.98	**NaN**	201	44	130	24	2.025	Inland
-118.34	**NaN**	36	2274	411	1232	423	*5.373*	<1H Ocean

Table 1: Example unprocessed DataFrame containing missing entries and categorical input values

Longitude	Latitude	Housing Median Age	Total Rooms	Total Bedrooms	Population	Households	Median Income	Inland	<1H Ocean
0.505	1	-1	1	1	1	1	0.455	1	-1
1	-1	0	-1	-1	-1	-1	-1	1	-1
-1	0	1	-0.495	-0.453	-0.472	-0.394	*1*	-1	1

Table 2: Example processed DataFrame with missing entry handling, normalization, and one hot encoding

2.2 Regressor Constructor method

The constructor method was used to initialize the desired architecture. The constructor took a series of hyper-parameters as optional arguments. These included the number of neurons per layer (as a list), dropout rate, and the input dimension to be inferred from the provided data. Our network consisted of a series of linear layers, each followed by tanh activation functions and dropout layers.

The tanh activation function was chosen due to its bounded output range, gradual gradients and its odd symmetric nature, which are believed to improve the speed of convergence compared to other common activation functions such as sigmoid [4]. The computational complexity and gradient diffusion issues [4] were not expected to be an issue due to the small network size.

2.3 Model-training method

The fit method finds the optimal values of weights and biases using loss minimisation. First, the input is normalized and then backpropagation is used to obtain a smaller loss. The maximum number of times a training set member is used in back-propagation is specified by the number of epochs in the constructor. During each epoch, the training set is split into batches and the average loss is calculated for all of the batches. The loss is also calculated on the validation dataset. If the sum of losses over the last 100 validation losses is greater than the sum of validation losses before the last 100 losses, then the fit loop is terminated to prevent unnecessary computation. The mandatory minimum number of runs of the epoch loop could be specified to account for the initial non-monotonic behaviour of the validation loss. The network state of the best-performing epoch was returned.

3 Model evaluation

To set up the datasets, the provided data was shuffled according to a fixed seed to remove any ordering in the dataset. The data was then split into train, test and validation sets with a split of 80/10/10. During training, the model was evaluated after every epoch using the validation set, and the mean squared error loss noted. This was used for early stopping as well as for the selection of the model with the best performance generalization following training. One of the main advantages of using root means squared error, is that large values are penalized more heavily, which ensures that the highest effort goes into the removal of the most significant errors. When calculating the score for a model the input is initially normalized by the preprocessor to values between -1 and 1 based on the training data. The prediction is generated using a forward pass through the network. The output, is generated in normalized form, which is reversed before the predictions are returned.

4 Hyperparameter tuning

Neural network models have a large number of hyperparameters related to the architecture, as well as the optimization process itself. The performance of the network is highly dependent on these parameters, so their optimization is crucial. Due to the relatively long training time, it was not feasible to optimize the architectural and training hyper-parameters jointly. Grid search was used to methodically sweep architectural hyperparameter combinations, which was followed by the individual optimization of training parameters.

From preliminary experimentation, it was noted that the final loss of the trained model was highly dependent on the initial random state of the weights and biases. It was deemed that training a network on a set of hyperparameters only once would exhibit a large amount of noise making informed decision-making difficult. Therefore an ensemble of 8 networks was trained with differing random weight initialisation for the evaluation of a given hyperparameter set. The best validation loss was taken to be representative of the relative architecture performance with proper training hyperparameters.

4.1 Architectural hyperparameters

The default training hyperparameters were set as: dropout: 0.2, learning rate: 0.0003, batch size: 32, and max epochs: 2000 for the optimization of architectural parameters. This was empirically based on a good performance during preliminary experimentation. The number of epochs was variable. During training, the optimizer tracked the validation loss and stopped training when the validation loss was observed to be rising. Due to the noise on the validation loss, an average across the last 200 epochs was used in deciding early training termination. The loss of the best-performing epoch was taken.

The approximate values for the network hyperparameters had to be estimated in order to set the bounds of the grid search. Similar regression problems were identified in the literature. Lee et. al. [5] used a network of 2 hidden layers, 6 neurons each to solve a multiple regression problem with 4 inputs and 2 outputs and Azmathullah et.al. [6] used a network of 1 hidden layer with up to 82 neurons to solve a regression problem of 3 outputs and 5 inputs. As such, the hyperparameter search was restricted to architectures of 1, 2 and 3 hidden layers. The search also included 3 layers as the problem being addressed here was slightly larger than those found in the literature.

Figure 1 shows the performance of the model with 1 hidden layer on the validation data set as a function of the hidden layer size. It can be seen that the model can predict values to an extent. There is a trend towards

increasing performance with an increased size of the hidden layer. The optimum for this subset of architectures was 64 neurons, with a best case RMSE of 69611.

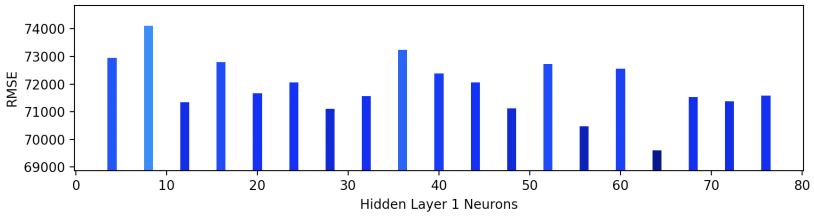


Figure 1: Best ensemble 1 layer search

Figure 2 shows the best-case performance of the model with 2 hidden layers. Values from 4 to 40 neurons per layer varied with a stride of 4. The results show that for this kind of network within the parameter range, where exists a minimum with the first hidden layer containing 24 neurons and the second hidden layer containing 36 hidden neurons. The best case RMSE was 61348.

For a network architecture with three hidden layers, the combinations of layer sizes were varied for values ranging from 4 to 20 with a stride of 4. Figure 3 shows the losses as a function of the second and third layer sizes, with layer 1 size fixed as 4 neurons. In the figure, a minimum can be seen with the configuration of 4 neurons in the first hidden layer, 12 in the second and 12 again in the third. Here the best case validation RMSE was 61820.

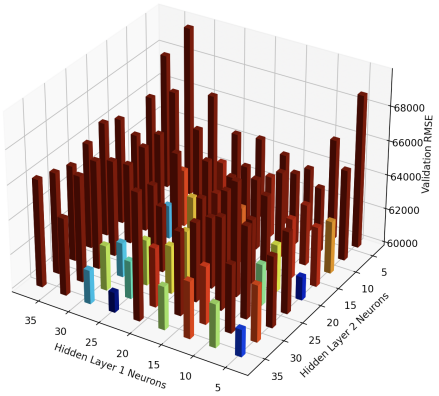


Figure 2: Best ensemble 2 layer search

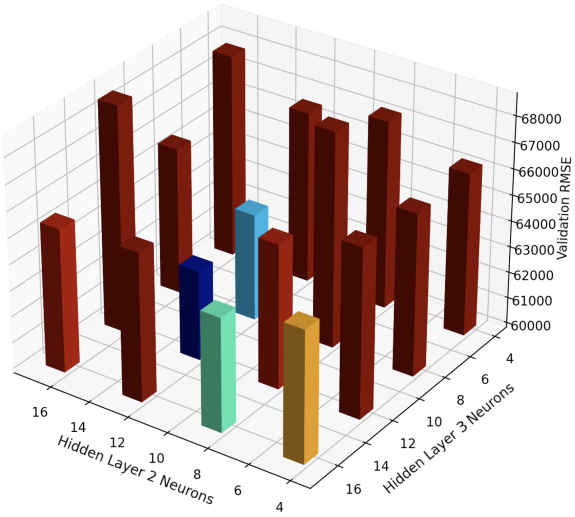


Figure 3: Best ensemble 3 layer search; layer 1 fixed at 4 neurons

Of the three types of architectures trialed, the two layer architecture was the most performant. While it is possible that this is not the true optimum due to the initialization noise and a relatively small sample size due to time constraints, we are confident that this architecture is relatively close to the optimum and should be adequate for the purposes of this report. We note that some of the validation scores are lower than the final results, we attribute this to the generation of a few exceptional solutions when running the close to 1000 training cycles, as well as the use of a different consistent shuffle on the data. As we have only been comparing between networks using the same datasets, this analysis is still valid.

4.2 Optimiser hyperparameters

For our optimization runs, the number of epochs was not fixed, merely given a large upper bound. A moving average of the validation losses of the last 200 epochs was kept and used to detect when the validation loss minimum was reached, which allowed for training to be halted automatically.

Two optimisers were trialed for the training of the final network, SGD and Adam. Figures 4 and 5 show that there were clear performance and training time advantages when using Adam over SGD. This was attributed to the Adam's more advanced per parameter learning rates and moment estimation [7]. It should be noted that

the training losses shown, were based on the forward passes with dropout present and as such exceed validation losses during certain sections of the training process.

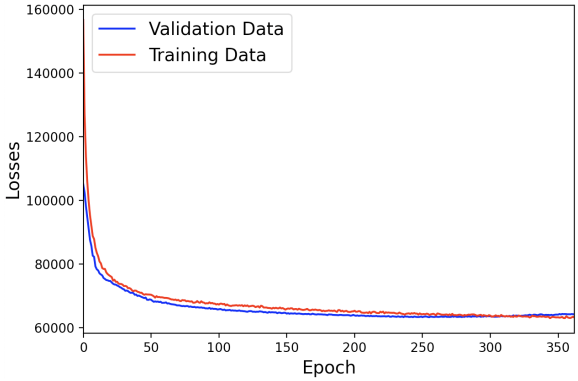


Figure 4: Sample Adam performance

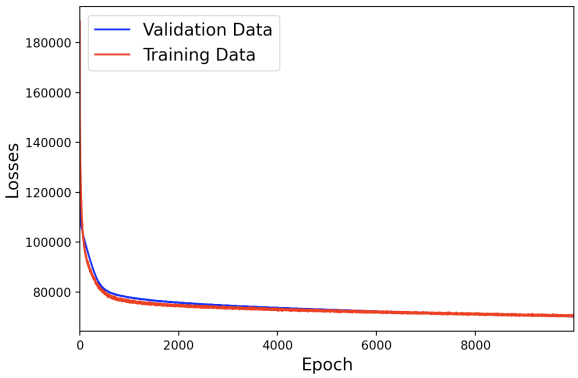


Figure 5: Sample SGD performance

One of the key methods used to limit overfitting and improve the performance of the network on missing data was dropout. Dropout encourages the network to prevent the co-adaptation of feature detectors, decoupling the influence of individual input parameter, by randomly setting neuron activations to 0 at training time [8]. The probability of zeroing neurons is an important hyperparameter and was analyzed experimentally as can be seen in figure 6. An optimum was found at a dropout rate of 0.06. However, this particular result was not repeatable across seed values. Going forward, the value of the second minimum at 0.2 was taken as the results were capable of being reliably replicated. We attribute this behaviour to the exceptionally favourable random nature of dropout and layer initialization in the first minimum.

Another hyperparameter that was optimized was the batch size. Smaller batch sizes lead to more frequent but noisier gradient updates. Generally, this noise is a negative as it can prevent the optimiser from settling on a minimum, but in small amounts such noise can help prevent overfitting [9]. This parameter was varied and its impact on the network performance was observed. Figure 7 shows that the network's loss was at a minimum with a batch size of 32.

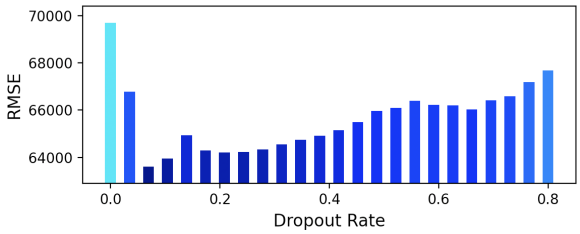


Figure 6: Dropout variation results

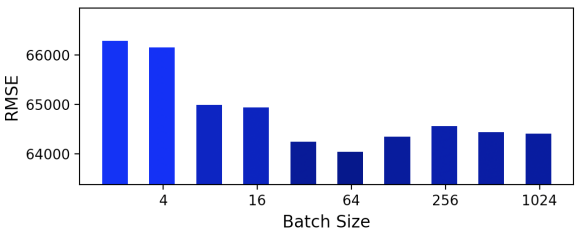


Figure 7: Batch size variation results

The impact of the random weight initialization was noted in earlier stages of exploration, and as such 16 variations of the model with different random initializations were trained. Figure 8 shows the histogram of the achieved validation losses across the 16 variations of the model. It can be noted that on this occasion the variation appears rather low, likely due to the good optimiser hyperparameters. In general, it was common for the model to get stuck in a local minimum with a validation loss approximately 30% worse than the reached minimum.

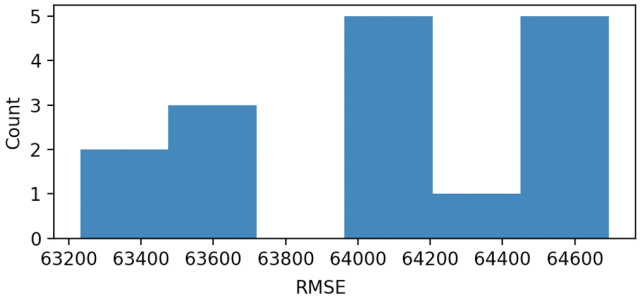


Figure 8: Histogram of validation losses for varying seeds

5 Performance evaluation

A held-out test set was used for the final performance evaluation. This constituted 10% of the total data. The RMSE loss for the final model was *61783* for the training set, *63232* for the validation set, and *63424* for the held-out test set.

Figure 9 shows the predicted and actual house prices on a scatter plot. There is a clear correlation between the predictions and the actual values, with the bulk of the data points in the neighborhood of the central line. It should be noted that the dataset contained a disproportionately a large number of houses with prices of 500001. These data points were predicted poorly by our model. It is believed that this is the consequence of any prices exceeding 500000 being truncated.

Figure 9: Comparison of predictions and gold values

In an attempt further improve performance, an ensemble of 16 models with identical hyperparameters but differing weight initialization before training was trialled. This gave a test RMSE of 63821, which was marginally worse than the single best model. We believe this to be the consequence of the low diversity in the ensemble as for such a small network the optimiser brought the weights and biases to a near identical state. As such this approach was ultimately rejected.

An attempt was also made to retrain the model with the validation dataset included within the training set. Here the best performing optimiser and architectural parameters were used. The model was allowed to train for 255 epochs, which at which time the best performance was previously reached. With the held out test set, the performance was again marginally worse at a RMSE of 63513. It is believed that our training process would have benefited from a decaying learning rate and these results are the consequence of the dropout and the batching introducing a slight dependence on the exact epoch when training is stopped due to the large optimiser steps. As such this approach was also rejected and the initial model was kept.

References

[1] K. Sree and C. Bindu, "Data analytics: Why data normalization," *International Journal of Engineering and Technology (UAE)*, vol. 7, pp. 209–213, 2018.

[2] C. Seger, "An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing," 2018.

[3] T. Makaba and E. Dogo, "A comparison of strategies for missing values in data on machine learning classification algorithms," in *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, pp. 1–7, 2019.

[4] S. S. Liew, M. Khalil-Hani, and R. Bakhteri, "Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems," *Neurocomputing*, vol. 216, pp. 718–734, 2016.

[5] J. Lee and K. Um, "A comparison in a back-bead prediction of gas metal arc welding using multiple regression analysis and artificial neural network," *Optics and Lasers in Engineering*, vol. 34, no. 3, pp. 149–158, 2000.

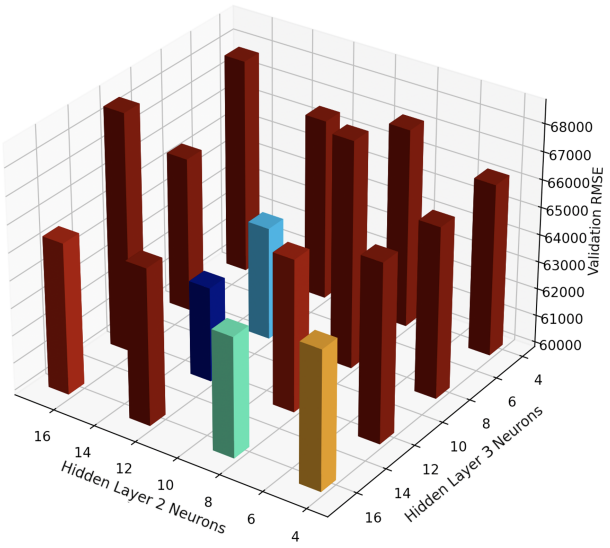
[6] H. M. Azmathullah, M. Deo, and P. Deolalikar, "Neural networks for estimation of scour downstream of a ski-jump bucket," *Journal of Hydraulic Engineering*, vol. 131, no. 10, pp. 898–908, 2005.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

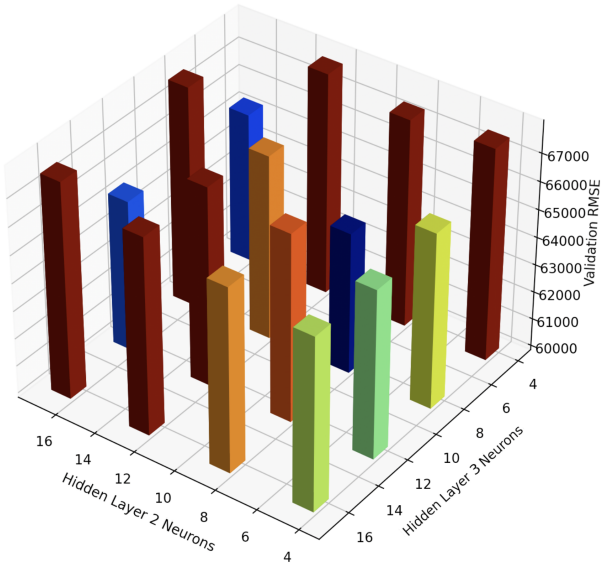
[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[9] A. Devarakonda, M. Naumov, and M. Garland, "Adabatch: Adaptive batch sizes for training deep neural networks," *arXiv preprint arXiv:1712.02029*, 2017.

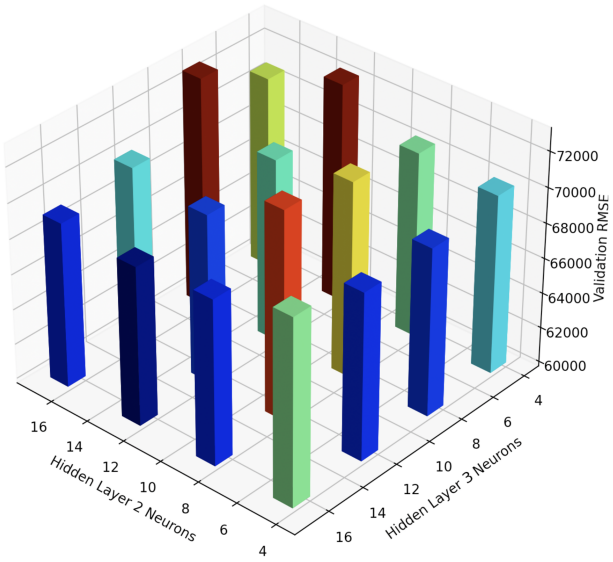
A Appendix: Raw three hidden layer optimization results



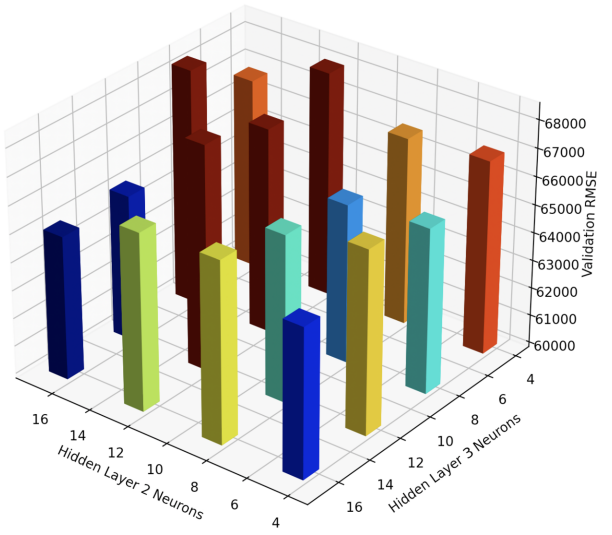
(a) Validation losses for hidden layer 1 with 4 neurons



(b) Validation losses for hidden layer 1 with 8 neurons



(c) Validation losses for hidden layer 1 with 12 neurons



(d) Validation losses for hidden layer 1 with 16 neurons

Figure 10: The average and standard deviation of critical parameters: Region R4