Imperial College
London

Introduction to Machine Learning

Coursework 2, Part 2: Group Report

Shreya Desai
Hana Ali
Alexander Mikheev
Acer Blake

2022
November

1 Model Description

1.1 Model Architecture

The final high-level model is a fully-connected PyTorch neural network with parametric values for the number of hidden layers, neurons per layer, learning rate, batch size, and training epochs. The objective was to minimise the number of hand-crafted constraints on the model architecture, which naturally led to the implementation of a dynamic model. As will be detailed in Section 3, this allowed the hyperparameter search method to guide the discovery of a high-performance regressor. PyTorch was chosen over the neural network implemented for part 1 due to its optimised libraries and compatability with various hyperparameter search facilities such as GridSearchCV.

1.2 Data Preprocessing

While not directly part of the model architecture, data preprocessing significantly influenced the neural network performance, weights, training times, and number of epochs required to achieve convergence. Emphasis was thus placed on cleaning and preparing the data in such a way as to enhance the performance of the model. Five steps were taken to prepare the data before training: one-hot encoding, filling in missing values, normalisation, ensuring robustness, and converting the training data into a tensor dataset.

- **One-hot encoding**: One-hot encoding was performed on the 'ocean proximity' feature, which is the only column in the dataset containing categorical values. Since a neural network can only operate on numerical features we require a method of converting these categories to real numbers. One-hot encoding converts these string-based representations into numerical binary values, which can then be used by the neural network to train.

- **Filling in missing values**: Several cells in the provided dataset contain NaN values, which were replaced to handle both the case in which training data is missing and the case in which test data is missing and a prediction is to be made with sparse information. This was achieved by using the median value for numerical features and the modal value for the categorical 'ocean proximity' feature.

- **Normalisation**: The given dataset contains feature values that range widely in magnitude. For example, entries for the 'total rooms' column are generally on the order of hundreds or thousands, while entries for the 'housing median age' column are on the order of tens. Thus, normalisation is required to bring all of the data within the same scale. Standard scaling was used to normalise feature values so that all values are within the interval [0, 1]. This was done according to the equation $z = \frac{x-\mu}{\sigma}$ where σ is the standard deviation, μ is the mean, and x is an individual feature value. Standard scaling was used as opposed to min-max scaling as it handled outliers more reliably in experiments.

- **Ensuring dimensional robustness:** Due to performing one-hot encoding it is possible that the training and test datasets will contain different numbers of columns. For example, there are only four occurrences of the 'ISLAND' value for the 'ocean proximity' column in the entire dataset. When separating shuffled data into training and test sets then, it is possible for this feature value to be missing in the training set, which results in the addition of an extra column in the test set. To ensure robustness we save the column headers from the training set and impose these on the test set.

- **Tensorising the dataset**: The final stage of data preprocessing entailed converting the training data to the PyTorch tensor datatype. This allows for hardware level optimisation by utilising GPUs, thereby significantly accelerating training and evaluation times.

2 Evaluation Setup

Model performance was evaluated using several metrics, namely the root mean squared error (RMSE), R^2 score, mean absolute difference (MAD), model complexity, and training time. Each of these measures captures different aspects of model performance and when taken together proved a powerful tool for effectively refining the model. A full summary of the metrics and the purpose they serve is given in the table below:

Metric	Definition	Motivation and utility
Root Mean Squared Error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$	Predictive accuracy. RMSE evaluates how well a regression model fits the data. This tells us how well the model has performed over the full training set for a given epoch.
R^2 Score	$\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$	Variance. R^2 evaluates how much of the variability observed in the house price predictions is explained by the regression model.
Mean Absolute Difference (MAD)	$\frac{1}{n}\sum_{i=1}^{n}\lvert\hat{y}_i - y_i\rvert$	Qualitative expressivity. The MAD informs us of the average value by which the predicted house price differs from the actual house price in dollars.

Figure 1: A table containing a summary of the metrics used for evaluating the model.

The root mean squared error was chosen as a measure of predictive accuracy due to the nature of the prediction task. The magnitude of the error between predicted house prices and the actual house prices may initially be on the order of hundreds of thousands. Therefore, in the early stages of training, the regular mean squared error commonly used to evaluate regression models often produces results which give rise to infinite values and overflow errors. Taking the square root of this metric prevents such issues from arising.

The R^2 score was used to provide another perspective on model performance. While the RMSE is unbounded, R^2 provides a measure of the performance of the regressor within the interval $[0, 1]$, with a score of 1 representing a model which perfectly fits the data. This provides a more interpretable measure of how well the model is performing in absolute terms.

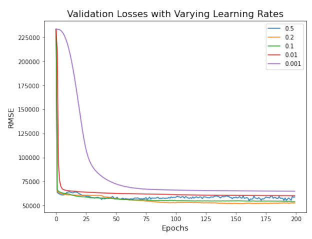
In addition to the two metrics defined above, the mean absolute difference was used to give a qualitative meaning to the predictive accuracy of the model. The mean absolute difference tells us what, on average, the difference in dollars will be between the model's prediction for a house price and the actual house price. While not used as a loss function for training the network, the mean absolute difference provides a useful, human-interpretable way of reasoning about whether the model is accurate in absolute terms.

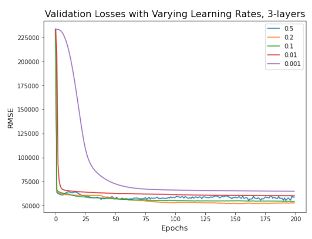
Finally, it should be noted that we use two further informal metrics to compare various models: model complexity and training time. Throughout many experiments it emerged that different model architectures had near identical performance, with some pairs of equivalently performing models being significantly more complex than others. In such cases we have favoured models with fewer hidden layers and fewer neurons per layer.

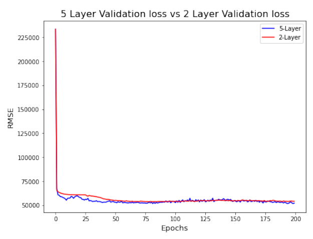
3 Hyperparameter Search

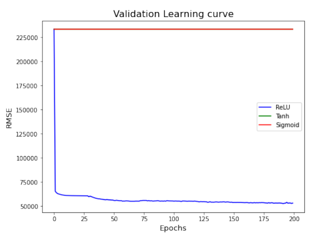
The hyperparameters were optimised using a combination of manual tuning and combinatorial search over a pre-defined search space using GridSearchCV (a technique that exhaustively tries every combination of parameters with cross-validation). Even a narrow search space resulted in a total run time on the order of tens of hours, therefore it was necessary to constrain the range of hyperparameters being tested. The process of constraining the search space was guided by a combination of experimentation and theoretical research. Justifications for the search space over each parameter are given below:

- **Activation function**: $\in \{\text{ReLU}\}$
  - From a theoretical standpoint, ReLU is the clear choice of activation function between layers for regression tasks, because it prevents outputs being inappropriately constrained. We verified this by also trying tanh and sigmoid. Figure 2d illustrates that tanh and sigmoid are completely unsuitable. Our output activation function is linear, since this is a regression task.
- **Optimizer**: $\in \{\text{Adam}\}$
  - Stochastic gradient descent (SGD) was tested and simply did not perform nearly as well as Adam. AdamW was tested as well, however the extension of weight decay was not useful because of the low number of training epochs we used. Moreover, with fixed weights we rarely ran into overfitting issues.
- **Batch size:** $\in \{32, 64\}$
  - Through manual experimentation lower batch sizes suffered exceedingly long training times and slower convergence.
  - Higher batch sizes resulted in poorer model performance due to numerical instability.
- **Learning rate:** $\in \{0.01, 0.2, 0.1\}$
  - Observing Figures 2a and 2b, a learning rate of 0.001 converges to values $\approx$ 20,000 RMSE higher than the other curves, and is hence too low. A learning rate of 0.5 produces a ‘jagged’ curve from overshooting, leading to unstable values which are often too high. Figure 2a shows training on a 2 layer model, while Figure 2b trains a 3 layer model; we plot both to demonstrate that this outcome was consistent across several manual experiments.
- **Epochs:** $\in \{100, 200\}$
  - Across all manual experiments conducted the learning curve converged well before 200 epochs, thus there is no benefit to increasing this further. Figure 2 illustrates this.
- **Number of neurons per layer:** $\in \{16, 32, 64, 96\}$
  - The aim was to have enough variety to achieve noticeable differences. Through experimentation we observed that a change by a constant factor of 2 gave this effect.
- **Number of hidden layers:** $\in \{1, 2, 3, 4\}$
  - Through experimentation we observed that 5 layer models achieved no further accuracy gains, and was thus a redundant quantity. Figure 2c illustrates the learning curve for a 5 layer model, which achieves a similar validation RMSE as the 2 layer models (Figure 3).





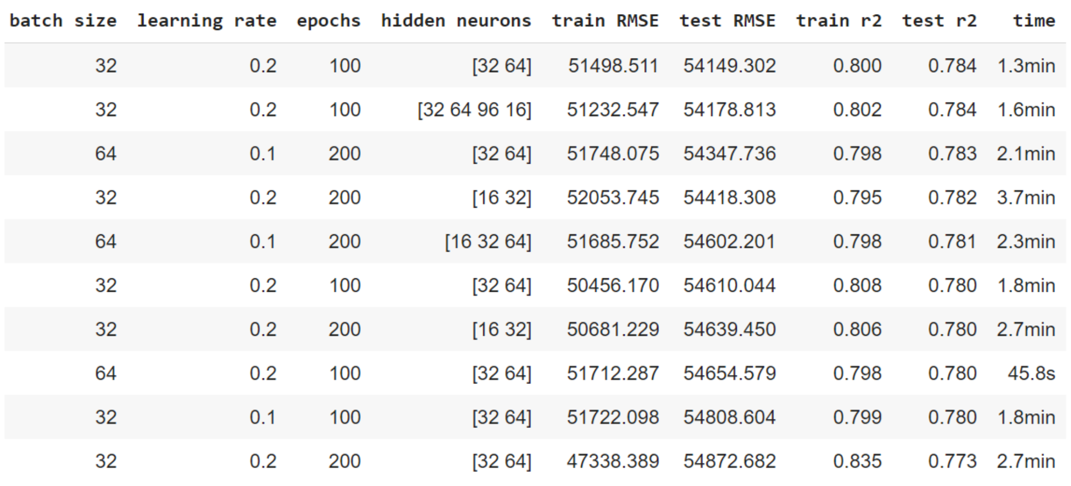




(a) Varying LR, 2 layer (b) Varying LR, 3 layer (c) 2 vs 5 layer model (d) Varying Activation

Figure 2: Plots for various manual tuning experiments.

At each step of the grid search a configuration of parameter values was selected and 3-fold cross validation was performed to evaluate the performance of the model with these parameters fixed. The best models were then tracked over many hundreds of trials and the top performing architectures were identified and further analysed. The performance of the top ten performing models is given in the figure below.



batch size	learning rate	epochs	hidden neurons	train RMSE	test RMSE	train r2	test r2	time
32	0.2	100	[32 64]	51498.511	54149.302	0.800	0.784	1.3min
32	0.2	100	[32 64 96 16]	51232.547	54178.813	0.802	0.784	1.6min
64	0.1	200	[32 64]	51748.075	54347.736	0.798	0.783	2.1min
32	0.2	200	[16 32]	52053.745	54418.308	0.795	0.782	3.7min
64	0.1	200	[16 32 64]	51685.752	54602.201	0.798	0.781	2.3min
32	0.2	100	[32 64]	50456.170	54610.044	0.808	0.780	1.8min
32	0.2	200	[16 32]	50681.229	54639.450	0.806	0.780	2.7min
64	0.2	100	[32 64]	51712.287	54654.579	0.798	0.780	45.8s
32	0.1	100	[32 64]	51722.098	54808.604	0.799	0.780	1.8min
32	0.2	200	[32 64]	47338.389	54872.682	0.835	0.773	2.7min

Figure 3: Top ten performing models found by GridSearchCV, averaged across folds.

The performance of the top ten models consistently scored an RMSE averaged over 3-folds $\approx$ 54,000, and an R^2 score in the interval [0.79, 0.82]. While the RMSE metric can be difficult to interpret, the R^2 score demonstrates that the best models perform exceedingly well.

Performance aside, one of the primary architectural takeaways from the grid search was that it confirmed our initial findings regarding the number of hidden layers, namely that increasing the model complexity by adding additional layers did not materially improve performance, and following a certain point caused a performance decrease. We thus decided on a simpler model with 2 layers.

Moreover, the grid search revealed that varying the batch sizes between 32 and 64 had a negligible difference with respect to performance, but a batch size of 64 typically led to faster training times. This can be seen from Figure 3 by observing rows which differ in their parameters only with respect to their batch sizes. Similarly, with respect to the number of training epochs, for the configurations of the remaining parameters we tested it was found that it was seldom useful to train for more than 200 epochs. Doing so with a high learning rate occasionally gives rise to overfitting. Weighing model complexity, performance and training time, we decided to use the 2 hidden layers model with [32,64] dimensions. A number of other architectural choices were considered when constructing the search space which were ultimately not implemented, in order to keep the runtime feasible.

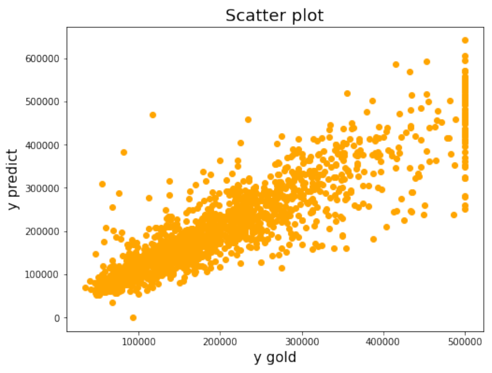
After finalising the model we performed further experiments with dropouts and a decaying learning rate to test if this would lead to further performance gains. Implementing dropouts in combination with the parameter ranges of the final model caused very early training plateaus, and ultimately failed to converge to an acceptable RMSE score. An initial analysis suggests that one potential cause of this performance degradation is that the networks constructed over the grid search are too shallow for dropouts to be effective. Learning rate decay was also tested. Increasing to 500 epochs (in which case the learning rate is sufficiently small) could lower the RMSE by $\approx$ 500, at the cost of tripling the training time. This trade-off was considered to be unrewarding, thus we omit learning rate decay in the final model.

4 Final Evaluation

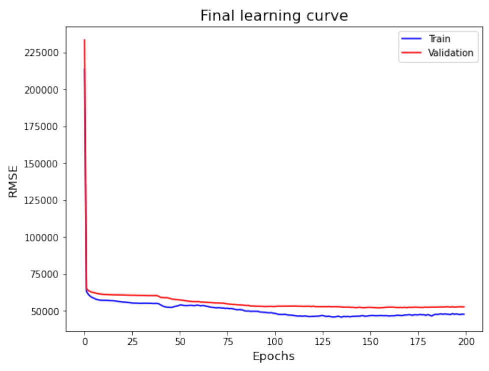
The final parameters for the best performing model are given below:

- **Batch size:** 64
- **Learning rate:** 0.2
- **Epochs:** 200
- **Number of hidden layers:** 2
- **Neurons per layer:** 32 64
- **Activation function: ReLu**
- **Optimiser:** Adam

The final model achieves a training accuracy $\approx$ 48,000 RMSE, and a final accuracy on useen data $\approx$ 50,000 RMSE, with small variablity across runs. This achieves a final R^2 value of 0.81 on unseen data, which is approximately $32,000$ dollars absolute difference per prediction.



(a) Scatter plot of predictions against labels



(b) Learning curve for the final model

Figure 4: Plots demonstrating the performance of the final, best performing model.

Figure 4a shows that the model performs best when predicting house prices with low values and worse on the higher value houses. An approximately linear relationship can be seen clearly in the interval between 0 and 300,000, demonstrating high performance, and thereafter we see greater dispersion in the graph with model predictions becoming progressively worse. This is something to investigate further for future work.