Neural Network Coursework - Report

Leonardo Garofalo, Karim Khairaz, Nicholas Pfaff, Bradley Stanley-Clamp

Contents

1 Model Description and Justification **2**
- 1.1 Data Analysis . . . 2
  - 1.1.1 Missing Values in the Data . . . 2
  - 1.1.2 Feature/Output Distributions . . . 2
- 1.2 Preprocessing . . . 2
- 1.3 General Model Architecture . . . 3

2 Evaluation Setup **4**
- 2.1 Mean Absolute Percentage Error . . . 4
- 2.2 Median Absolute Percentage Error . . . 4
- 2.3 Mean and Median Percentage Error . . . 4

3 Hyperparameter search **4**

4 Final Evaluation **5**

1 Model Description and Justification

1.1 Data Analysis

Exploratory data analysis was performed to gain insight into the dataset. This revealed useful patterns that influenced the design of the model.

The dataset contains a total of 16512 data points. Each datapoint contains 9 features and 1 output ('median_house_value').

1.1.1 Missing Values in the Data

Data Analysis revealed that the only feature containing missing values is 'total_bedrooms'. No other feature contains missing values.

1.1.2 Feature/Output Distributions

Each feature/output value was plotted against its relative frequency in Figure 1. It was observed that:

- The probability distributions of 'total_rooms', 'total_bedrooms', 'population', 'households', and 'median_income' seem to be Gaussian.
- 'housing_median_age' and 'median_house_value' have a large concentration of data points around the values of 50 and 50000 respectevly. These abnormal frequency spikes do not appear in other features.
- 'ocean_proximity' is the only feature with discrete textual values. All other features contain continuous values.

1.2 Preprocessing

In our model Data preprocessing consists in the follwing steps:

- Handle missing values in the dataset
- Handle textual values in the dataset
- Normalize the numerical values in the dataset
- Scale the output values

Handle Missing Values The preprocessor method was implemented in a way to allow 3 distinct strategies to deal with the missing values in the dataset. These strategies were:

1. Remove all datapoints that contain missing values
2. Replace the missing values with a fixed default value
3. Replace the missing values with the mean of the feature that they belong to

The data analysis performed in subsection 1.1.1 revealed that all the missing values belonged to the 'total_bedrooms' feature. Moreover, subsection of 1.1.2 showed the absence of abnormal frequency spikes in this feature.

Strategy 1 was not chosen because removing datapoints from our dataset would negatively impact the performance of our model, as the model would have less examples to learn from.

Strategy 2 was not chosen as replacing the missing values with a arbitrary default value would affect the probability distribution of the feature.

Strategy 3 was selected as replacing the missing values with the mean of 'total_bedrooms' is the method that affects the distribution of our model the least (from the definition of mean). This method also allowed us to retain all the datapoints.

Handle Textual Values The data analysis performed in Section 1.1 did not indicate any way of assigning a label a priority (i.e. assigning a label a value bigger/smaller than the value assigned to a different label). Therefore, rather than using a strategy like integer encoding (that assumes a metric to measure the relative difference between labels in order to put them on a scale) the chosen approach was one-hot encoding. One-hot encoding treats each label as a distinct, independent feature and hence does not require assigning priority to labels.

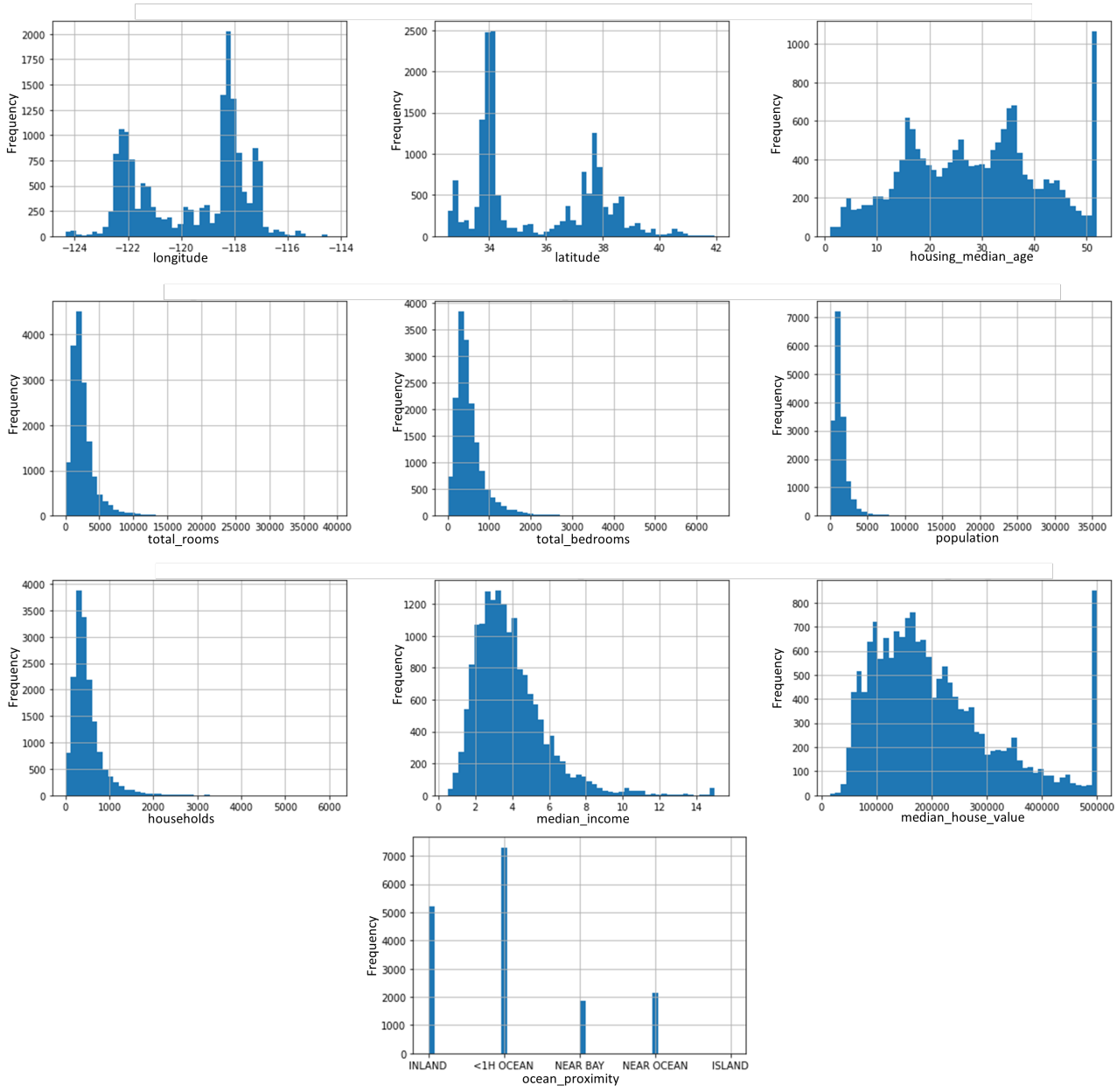

Figure 1: feature/output value was plotted against its relative frequence

Normalize Numerical Values Gradient descent is used as the optimization technique for our model. Different feature ranges lead to different update step sizes. As the step size in the update step of gradient descent depends on feature ranges, it is important that all the features are scaled down to the same range of values (i.e. normalized).

The preprocessor method implements two ways of performing features normalization:

- Min-Max Normalization
- Standardization

The data analysis performed in subsection 1.1.2 revealed that most of the features seem to follow a Gaussian-like distribution. As standardization works best when the features follow a Gaussian distribution, it was chosen as our normalization strategy. Min-Max normalization could have also been a reasonable choice, but since only a few features are not Gaussian-like, standardization was preferred.

Scale the Output Values Experimental analysis showed that large values of 'median_house_value' interfered with the training of our model. This is because the MSE would reach very high values, causing problems with the calculations. The solution was to scale down the values of 'median_house_value' by a factor of 100000.

1.3 General Model Architecture

The complete model architecture is implemented in the 'Regressor' class and is based on PyTorch. The number of neurons in the input layer is fixed to the number of features in the housing data and is determined dynamically in the

constructor. The number of neurons in the output layer is fixed to one and the activation function at the output layer is linear (identity) as the task is regression. We tried to design everything else as dynamically as possible to make hyperparameter tuning easier. For example, the constructor takes an optional list containing the number of neurons per linear layer and another containing the corresponding activation functions. Supported activation functions are ReLu, Tanh, Sigmoid, and identity (no activation function). These are then combined sequentially into a neural network. By a similar fashion, one can choose one of stochastic gradient descent, Adadelta, or Adam as the optimizer. The loss function used is mean squared error. The network is trained using mini-batch gradient descent.

2 Evaluation Setup

There are a number of evaluation metrics to choose from which measure performance of the trained model by comparing the predicted and actual values. These metrics are used when training a model and comparing models with different hyperparameters:

- To train a model, a metric has to be chosen as the loss function. Since the evaluation metrics compute the difference between the model and the actual values, minimising this function leads to better performance of the model. This same metric can then be used to tune the hyperparameters.
- Using the loss function alone to evaluate how good the final model is is tricky as for example a single MSE result is hard to understand. It is affected by the output number range and hence can take arbitrarily large values. Thus, when evaluating the final model, we can choose multiple metrics which provide a well rounded insight into the performance of the trained model.

2.1 Mean Absolute Percentage Error

This evaluation metric is an average of the absolute percentage difference between the actual and predicted values. Since this metric involves an absolute difference, outliers aren't punished as much with higher weighting as in Mean Squared Error (squaring magnifies large differences due to outliers). However, since the metric is an average, we get an overall understanding of the performance for all data points which will include outliers and extremities. When interpreting the result of this metric for different models, the percentage allows us to understand the average fractional accuracy error of the models. This lets us intuitively grasp the advantage of one model over another as a percentage is easier understood than some large MSE value.

2.2 Median Absolute Percentage Error

The Median Absolute Percentage Error is calculated by finding the median of all the absolute percentage differences between actual and predicted values [1]. Since we are using median or the middle value, the metric is unaffected by values (or outliers) at the extremities. Compared to the Mean Absolute Percentage Error, we get a snapshot of the performance of the model for the bulk of the values without considering the inaccuracy brought about by outliers. Moreover, it allows us to understand the effect that outliers have on the performance of our model.

2.3 Mean and Median Percentage Error

The advantages offered by Mean Percentage Error and Median Percentage Error are similar to their Absolute Percentage Error counterparts. However, this time we are summing both positive and negative values. The possibility of a negative sign will help us grasp whether the model is overestimating or underestimating the actual values, indicating a bias. If the value is close to zero, then we are approximately over- and underestimating by equal amounts.

3 Hyperparameter search

We identified the following hyperparameters (not including different preprocessing options):

- number of linear layers
- number of neurons per linear layer
- activation functions between linear layers
- batch size
- optimizer
- learning rate (if optimizer requires it)

Finding the optimal hyperparameters would require trying all possible hyperparameter combinations which is unfeasible due to time constraints. The most important hyperparameters to tune appeared to be the number of linear layers and the corresponding number of neurons per layer. Hence, all other hyperparameters were fixed to reasonable values and remaining hyperparameters were optimised by searching through a defined search space. Next, the fixed hyperparameters were indivudally optimised using three-fold cross validation.

- The activation function was fixed to ReLu for all layers due to its simplicity and speed.
- The batch size was set to 2000 as this was be a reasonable middle value. Bigger batches result in a better estimation of the gradient and hence smoother stepping towards the minimum. Smaller batches result in a noisier estimation and hence might allow us to avoid getting stuck in local minimums.
- The optimizer was set to Adadelta as it was thought that an adaptive learning rate would work better than a fixed one for a range of layer and neuron distributions. Moreover, using an adaptive learning rate meant that the learning rate didn't need to be fixed.
- The number of neurons in the input layer were fixed to work for number of features.
- The number of neurons in the output layer were fixed to one while the output activation function must be linear (identity), as this is a regression task.

The distribution of neurons per layer were limited to a linear increase/decrease from the first hidden layer to the last hidden layer. This means that we only had to set the number of neurons in the first hidden layer and the number of neurons in the last hidden layer. All other layers would be determined by this linear increase/decrease constraint. This allowed further constrain the search space of the number of layers and of neurons per layer.

More simplification was applied by discretising the search space via the number of neurons in the first and last hidden layers to 10, 30, 60, or 100.

The number of hidden layers was varied from 0 to 10. To improve the consistency of results without significantly increasing the execution time, three-fold cross validation was used.

The result of this search space analysis is presented in figure 2

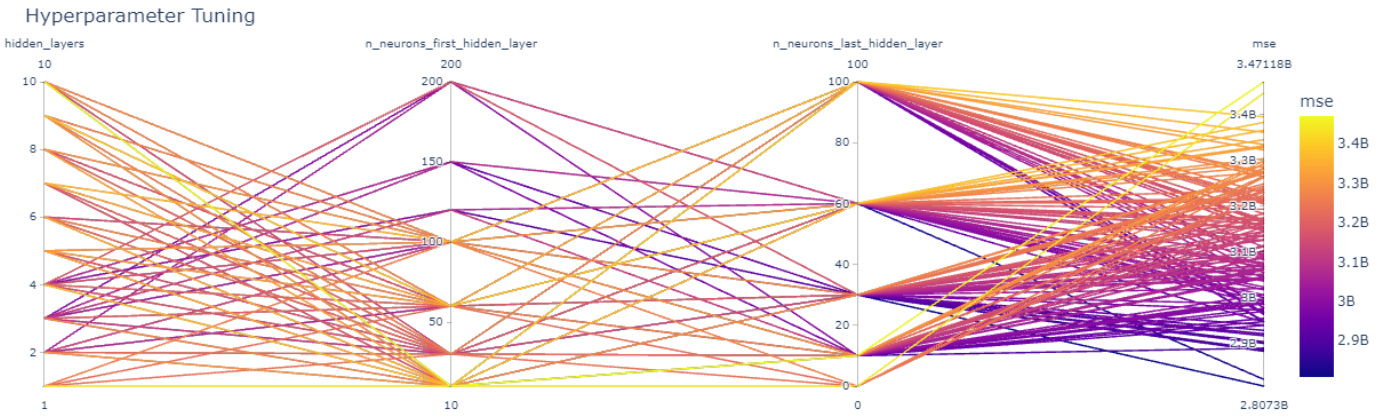

Figure 2: A Parallel Coordinates Plot of Hyperparmeters of the Neural Network

On analysis of figure 2, one key trend was identified; A linear decrease in neurons from first to last hidden layer is generally better. This can be seen by lower valued 'mse' lines (darker purple) having negative gradients between 'n_neurons_first_hidden_layer' and 'n_neurons_last_hidden_layer'. From initial results, optimum ranges of hyperparameters were identified and analysed in more detail. This included running more tests with more neurons in the first hidden layer as the previous optimum corresponded to the maximum value of neurons in the first hidden layer tested. However, these additional tests did not reveal a new optimum. This resulted in an optimal network of 3 layers consisting of 100, 65 and 30 neurons respectively.

Next the optimum activation function was identified as tanh (see Figure 3). The batch size was then varied and a size of 2000 produced the lowest MSE.

As indicated by Figure 3 the optimal number of epochs for our model is around 500. It is also clear from the graph that anything less than around 500 epochs underfits the data, whereas anything more than around 500 epochs overfits it.

4 Final Evaluation

The final evaluation returned a Mean Absolute Percentage Error of 17.5% and a Median Absolute Percentage Error 12.3% (refer to table 1). As described in the evaluation set-up, we expect the Mean Error to be larger than the Median Error due to presence of the outliers described in the data analysis section. The Mean accounts for the percentage error of all values and considers the large outliers which occur at the extremities of the data . The Median, on the other hand, computes the middle value. Since the outliers at both tails are not considered, this metric reveals the models accuracy for the bulk of the data which lies in the middle. Thus, we observe that the presence of outliers results in approximately a 5.2% loss of accuracy in our model. Meaning that for a few house groups, we are having bigger errors

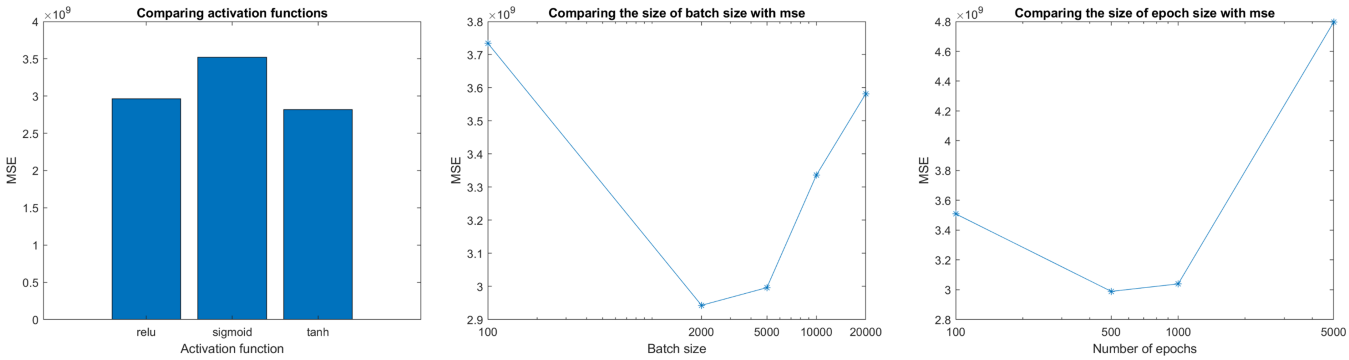

Figure 3: Optimising activation function, batch size and number of epochs

than for most house groups. The Mean Percentage Error is 0.3% and the Median Percentage Error is 3.0% (refer to table 1). These values being close to zero indicates that we are under- and overestimating similarly. The values being positive reveals that our model has a small bias where it underestimates (metric calculated as Y-true - Y-predicted) values. The mean being close to zero while the median is slightly positive indicates that including magnitude of error, we are under- and overestimating equally. However, when not considering magnitude, we are underestimating slightly more often than we are overestimating the house price (there are more positive than negative error values).

Evaluation metric	Metric value
mean_squared_error	2800335578.6526017
mean_absolute_percentage_error	0.17512425077654317
mean_percentage_error	0.0032880333314586744
median_absolute_percentage_error	0.12306712776102045
median_percentage_error	0.03006980194723506

Table 1: Evaluation of the final network

References

[1] *3.3. Metrics and scoring: quantifying the quality of predictions*. en. URL: https://scikit-learn/stable/modules/model_evaluation.html (visited on 11/25/2021).