# Artificial Neural Networks Report

Riya Chard, Vamika Gupta, Anish Narain, Elsa Polo-Laube

November 27, 2023

## 1 Introduction

This report describes a neural network architecture created to infer the median house value of a block group from the value of all other attributes with the California House Prices Dataset. This report includes justifications for model and evaluation metrics used within the code and explains the process of tuning optimal hyperparameters. Finally, the best model is determined through testing and evaluation.

## 2 Description and Justification of the Model

### 2.1 Handling Data

The preprocessor method was implemented first to preprocess the inputs and outputs on the model. This includes the following:

1. **Handling the missing values in the data**, using the lambda function to fill NaN (missing) values with the mean of the respective column. The mean was chosen because after analysing the data, it was found that only the total_bedrooms feature was missing any values. Furthermore, only 1.02% of the values were missing. This was insignificant and the mean was used to fill the gaps to ensure minimal distortion of the overall data distribution. This is a better option than removing values (which would mean the model would have a smaller data-set to train on) or replacing the missing values with an arbitrary default one (which would change the distribution).
2. **Handling the textual values in the data** by mapping from categorical values to 1-hot vectors by separating the categorical and numerical columns from the input x. As the ocean_proximity column was textual, the code uses a LabelBinarizer transformer to convert the categorical values into a one-hot encoded format. During training, the LabelBinarizer is fit on the categorical data and for testing the already fitted LabelBinarizer is used to transform new data into the same one-hot encoded format.
3. **Normalising numerical inputs**, using scalar objects to normalise or standardise numerical data. In order to determine the optimal scalar method, further data analysis was done, as shown in the next section.

### 2.2 Analysing Data and Normalisation Options

To normalise the data, the following scalars were considered:

- **MinMaxScaler**: Scales each feature to a range, typically between 0 and 1. It is sensitive to outliers.
- **MaxAbsScaler**: Scales each feature by its maximum absolute value. It is suitable for data that is centred at zero or sparse data.
- **RobustScaler**: Uses statistics that are robust to outliers. It removes the median and scales the data according to the IQR.

- **StandardScaler**: Standardizes features by removing the mean and scaling to unit variance, assuming the data follows a Gaussian distribution.

The data was then analysed to find the distributions of each feature.

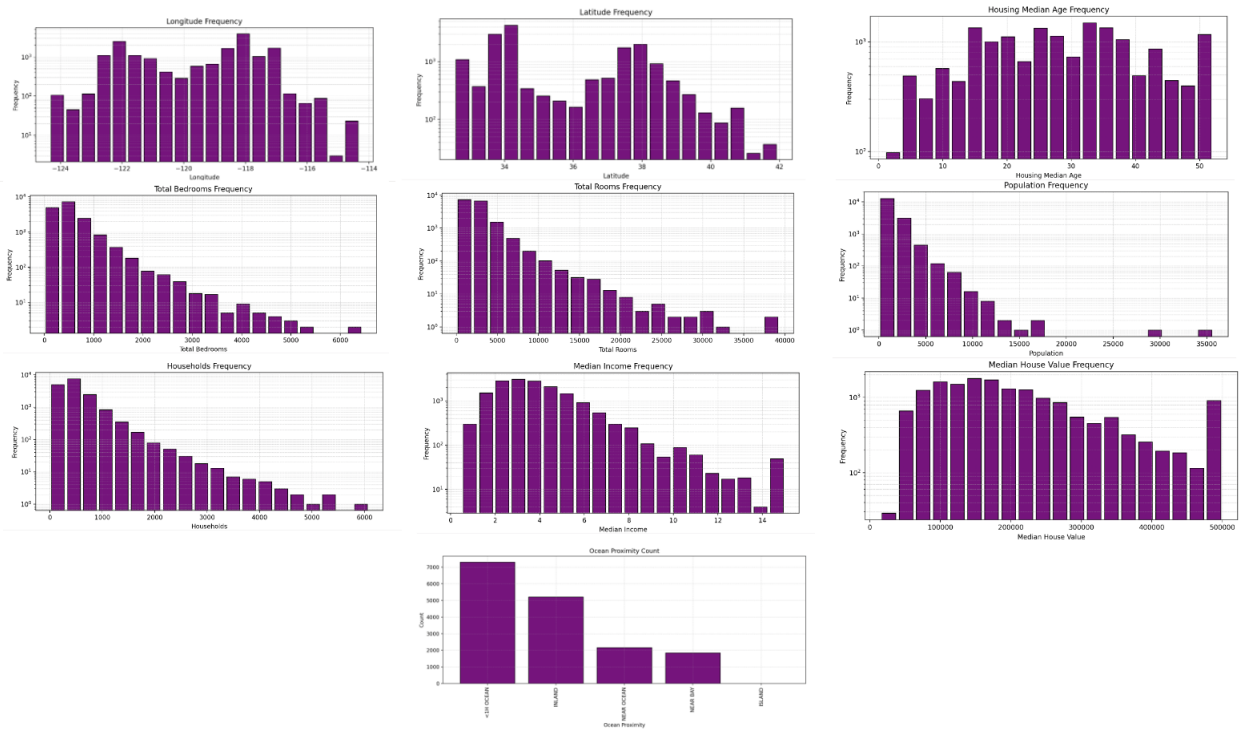


Figure 1: Analysing the distribution of the dataset

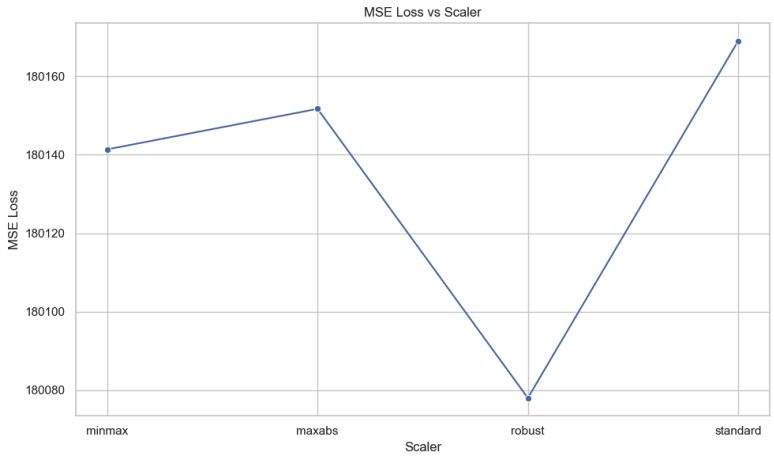


Figure 2: Analysing the impact of using different scalers on the accuracy of its predictions

**Prediction:** Given the histograms in *Figure 1*, the **RobustScaler** was suggested to be the most appropriate as it can handle outliers and is less influenced by the presence of non-Gaussian distributions in the data.

**Result of Hyperparameter Search:** As can be seen in *Figure 2* above, we observed a marginal impact on the MSE loss by varying the scaler used to normalise the data. Consequently, we considered the context of the problem. As *Figure 1* revealed that median house values did not converge to a nor-

mal/uniform distribution that would be the optimal scenario to use a StandardScaler/MinMaxScaler, we opted to use the **RobustScaler** due to its property of being resistant to outliers.

### 2.3 Architecture Overview

The architecture of the layered neural network, capable of processing both numerical and categorical data, is set up using the **Regressor** class. In the initialisation phase, the number and type of neurons in each layer are left customisable so that they can be tuned in the hyperparameter search. The architecture ends with adding a linear output layer to map the processed features to a continuous target variable.

Before training, the dataset is preprocessed. Numerical features in the data are scaled to normalise their range and variance and categorical features are encoded. Missing numerical values are imputed with the mean values of that feature to maintain the integrity of the data without adding a significant bias to the data. The choice of the scaler is also left for further tuning in the hyperparameter search (as described in *Section 2.2*).

The model is then trained using a backpropagation algorithm. The number of epochs which define the duration of the training process is left to be tuned during the hyperparameter search. Mini-batch gradient descent is utilised to update the model parameters, enhancing the convergence speed and model performance. After training, the model's performance can be evaluated using the regression metrics explained in *Section 3* against the separated test data.

## 3 Evaluation Setup

The evaluate set is created using both a prediction method and an evaluation method. PyTorch tensors were used to reduce the number of Python loops. Various evaluation metrics were chosen to evaluate the performance of the model to provide a broad view of performance:

1. **Mean Squared Error** calculates the average from the squared errors. MSE is sensitive to errors, enforcing a higher importance on them. As MSE is in the same units as the target variable, it is also easy to interpret.
2. **Median Absolute Error** takes the median of the absolute differences between the target and the prediction as MedAE is not as sensitive to outliers as MSE, it is useful to observe failures of the model and is more robust.
3. **R2 Error** is a regression error metric that justifies the performance of the model. This evaluates how well the model's predictions match the actual data. Additionally, as a normalized metric, it can be used to evaluate different models.
4. **Mean Absolute Percentage Error** measures the average magnitude of error produced by a model, or how far off predictions are on average. As this is an average, it helps to review the overall performance of the model including outliers and extremities.
5. **Explained Variance Score** is a metric used in regression analysis to assess how well a model captures the variation of the dataset. The metric is is easy to interpret and useful for model comparison.

## 4 Hyperparameter Search

To optimise our regression model, we conducted an exhaustive search for the optimal hyperparameters. This involved first partitioning the dataset into a training, validation, and testing set, and,

due to time constraints, our search was focused on 3-5 values per hyperparameter. Subsequently, we scored 1152 regressors, each tuned with a different combination of hyperparameters, by finding their root-mean-squared error against their predictions on the validation data, and saved the results in a CSV file to conduct analysis.

We later fixed the ideal values we found from the initial search and then varied one hyperparameter at a time to visualise the trends, which can be seen in *Figure 3* below.


Figure 3: Visualisation of the impact of varying different hyperparameters on the accuracy of its predictions

### 4.1 Parameter Choices

The hyperparameter search included testing the following parameters:

- **Activation Functions**: The activation functions tested were ReLU, Sigmoid and Tanh. The ReLU activation function demonstrated a significantly lower MSE error compared to its alternatives. This aligns with our understanding of ReLU's ability to learn quickly, a quality that proved beneficial for our model which was being trained on a relatively small dataset within a constrained timeline. Therefore, we identified **ReLU** to be the sole activation function we would proceed with.
- **Scaler**: As discussed in *Section 2.2*, we opted for the **RobustScaler** due to its property of being resilient to outliers.
- **Learning Rates**: The learning rates we chose to test were 0.01, 0.1, 0.25, 0.75. Higher learning rates helped the model converge faster but at the risk of potentially missing the minimum loss.
- **Batch Sizes**: We tested batch sizes of 128, 256, 512, 1024 and 2048 neurons. Larger batch sizes led to faster computation but also led to worse model performance.
- **Number of Epochs**: The number of epochs we trialled varied between 100, 500, 1000 and 5000. More epochs can lead to a better-trained model but also increase the risk of overfitting.
- **Number of Neurons per Layer**: For testing purposes, we simplified our model by assigning an equal number of neurons to each layer within the hidden layer, and we chose to test varying between 64, 128, 256, 512 and 1024 neurons. While increasing the number of neurons had the potential to improve the network's ability to capture complex patterns, this must be approached cautiously to avoid overfitting.

### 4.2 Hyperparameter Tuning Results

Given the findings from our initial search detailed in the above section, we then conducted a second tuning where we fixed those values which showed distinct improvements and further refined the other hyperparameters. Our resulting best model, stored in our *part2_model.pickle* file, features the following hyperparameters:

- Number of Hidden Layers: 5
- Number of Neurons per Layer: 64
- Activation Function: ReLU
- Scaler: Robust
- Batch Size: 128
- Number of Epochs: 500
- Learning Rate: 0.1

## 5 Final Evaluation

We conducted a final evaluation of our best model against the testing dataset we separated initially. The results are detailed in *Table 1* below:

| | |
|---|---|
| **Root Mean Squared Error** | 52745.08 |
| **Median Absolute Error** | 22678.467 |
| **Mean Absolute Percentage Error** | 19.69 |
| **Median Absolute Percentage Error** | 13.97 |
| **R2 Error** | 0.7906 |
| **Explained Variance Score** | 0.7914 |

Table 1: Evaluation metrics of best model

Our model has a **Root Mean Squared Error** (RMSE) of 52745.08, which is well under the LabTS threshold of 90000. A lower RMSE implies that the model's predictions are, on average, close to the actual median house prices. However, it is important to note that the RMSE is sensitive to outliers, and our analysis in *Section 2.2* did highlight extreme values in the dataset. The **Mean Absolute Percentage Error** is 19.67%, while the **Median Absolute Percentage Error**, which is less affected by outliers, drops to 13.97%, providing an idea of the accuracy of our model. The **R2 Error**, a key indicator of the model's goodness of fit, stands at 0.7907, suggesting that approximately 79.06% of the variance in the median house prices is explained by our model. Lastly, the **Explained Variance Score** of 0.7914 aligns with the R2 Error and reinforces the model's ability to explain and predict the observed variability.

Overall, while our neural network demonstrates satisfactory performance, if we had more time and fewer CPU constraints, we would consider further optimisation to enhance specific aspects of prediction accuracy and reduce errors. We would be interested in unravelling some simplifications we made, including varying the numbers of neurons within each layer, using different activation functions at different layers, and further fine-tuning our hyperparameters to refine the model and potentially achieve even better results in future iterations.