Machine Learning Coursework 2
Neural Networks

Indraneel Dulange, Omar Zeidan

Shaheen Amin, Shaanuka Gunaratne

November 2022

0. Introduction

The goal of this project was to create an artificial neural network by training on the California House Prices Dataset to infer the median house value of a block group from the value of all other attributes. A neural network architecture was implemented, and the model was trained on the dataset using the PyTorch neural network library.

1. Implementing an Architecture for Regression

1.1 Pre-processor Method

The pre-processor method converts the data into a format that can be used by the neural network. A Pandas Dataframe was used to process the data as it allowed for easy manipulation of 2D data structures. Firstly, any null datapoints were replaced by the averages of their respective columns using the Pandas 'fillna' method. The mean was used for continuous data, median for discrete values and the mode for categorical data. Secondly, the textual values in the 'Ocean_proximity' column had to be converted into a numerical format. As these were categorical datapoints, numbers or any other magnitude-based representations could not be used as their magnitudes would bias the model (i.e. the model may develop a bias towards larger numbers). The solution was to use a format that classified the values without adding a magnitude bias, which was one-hot encoding. This method converted the values into a binary representation of 0s followed by a single 1; the position of the '1' determined the categorical value. The LabelBinarizer class in scikit-learn was used to implement this. Finally, all other datapoints were scaled and normalised within a range of 0 to 1. This was done to ensure that there was no bias in the model due to large deviations in the data, but also to make numerical calculations during forward propagation simpler, which would lead to a faster convergence.

1.2 Constructor Method

The constructor initialises all the attributes used by the model. We amended the parameters to include the default hyperparameter values which were the maximum epochs, learning rate, the neuron architecture (which specifies the number of hidden layers in the network as well as how many neurons per layer), batch size and the minimum improvement factor. Any variable that needed to be

used outside of their methods was initialised here as an attribute. The binarizer and binarizer classes (for the ocean proximity column) were hardcoded in-case the training set does not include all the categories. The remaining attributes, 'allowance', 'count' and 'prevLoss' need initial values and are used in the 'earlyStop' static method. The artificial neural network itself was not initialized in the constructor as that would prevent hyperparameter tuning from changing its properties, since that process occurs after initialization.

1.3 Model Training Method

The model training is defined in the fit method of our regressor class. Both precision and efficiency were taken into consideration when designing the model.
Firstly, a simple sequential model was defined taking 13 neurons as input to match the 13 columns of data, followed by hidden layers defined by the hyperparameter 'neuronArchitecture'. The network then converged to a single output neuron as we desired one output (median house value).

Choosing a good optimizer was important as this dictated the change in attributes and weights applied in the model. From the optimizers that were available in the PyTorch library, the SGD and Adam optimizers were deemed the most reliable for this task. The Adam optimizer was used over SGD as it was known to have faster convergence and a lower risk of noise bias.

The mean square error was used to calculate the loss during gradient descent. This was the optimal choice for a continuous dataset that focused on precision. The squared factor ensured that larger errors in the predictions were penalised to a higher degree. It made sense to focus on larger errors rather than smaller ones because the model takes in real data and house prices are not consistent.

Mini-batch gradient descent was implemented to provide a balance between faster convergence and computational efficiency. Furthermore, using a large enough batch size would reduce the impact of noise on the model.

Additionally, during fitting, if a validation set is passed in (usually used during hyperparameter tuning), the 'early stop' function will become active. It is designed to prevent the model from overfitting and stops training when it discovers that the model is degrading or when the percentage improvement between epochs is insufficient. This 'percentage improvement' factor was manually tuned and set to 0.005%. Additionally, the model must fail to meet this percentage improvement requirement a certain number of times in a row for training to stop. This is defined by the 'allowance' attribute in the constructor and is currently set to 5. This ensures spikes in RMSE do not halt the training process until it is definitive the model is not improving.

2. Evaluation Methodology

2.1 Prediction Method

As the neural network has been trained on normalised data, any predictions made by the neural network must also be normalised - this is done by the preprocessor described in Section 1. Once the data has been normalised, it is passed forward to produce the predicted value using the PyTorch Sequential container that describes our neural network. The data returned from this is the prediction.

2.2 Evaluation Method

Among the popular metrics MSE, MAE and RMSE, our chosen metric was RMSE (root mean squared error) – this means that lower values are better which suits the GridSearchCV function later used in hyperparameter tuning. RMSE punishes large errors (as the error terms are squared) and the units of the final value error term are the same as the original units of the target value that is being predicted, in other worlds, using RMSE in a house price prediction model gives the error in terms of house price, which is an intuitive value for determining the model's performance.

2.3 Determining Error

To capture a representative RMSE value, we ensured that prior to testing, we split the dataset to retain a validation set which will be used to evaluate the model. This validation set is a left-out dataset, and therefore to the model it is unseen data and works as a reliable indicator to the model's performance.

3. Hyperparameter Tuning

3.1 Tuning Methodology

Our approach to hyperparameter tuning assumes that nothing is known about what the optimum configuration would look like, and therefore we use a more general approach. The only fixed parameters are the input layer (of size 13 to match the input feature size after binarizing), the output layer of size 1, and the maximum number of epochs 'nb_epoch' to prevent potential models from taking too long to train (The tuning for this parameter is detailed later). We opted to use Scikit-learn's GridSearchCV function to train multiple models and pick the best candidates. This was chosen over RandomizedSearchCV as our methodology requires being able to train new models based off previous iterations, which is more suited for GridSearchCV. The steps for this are as follows:

1. Initially, the first set of hyperparameters are set as follows (neuronArchitecture describes the number of neurons per hidden layer): LearningRate : [0.001, 0.01, 0.1, 1], neuronArchitecture: [[9], [9,9], [9,9,9]], batchSize: [64, 128, 256, 512].
2. Use GridSearchCV to generate every possible model with the above hyperparameters.
3. Determine the top 5% of candidates, according to their mean test score (3-fold cross validation was used to ensure this average was valid), and if it is the first iteration, determine the following: of the two highest performing neuron architectures, determine the number of layers to be used, and prefer less layers; the two highest performing learning rates; the two highest performing batch sizes.
4. From the learning rate and magnitude, calculate the logarithmic mean and generate four parameters within the range randomly, varying by at most 0.6 orders of magnitude. Also generate 4 new neuron architectures with the number of layers equal to the value discovered in 3a.
5. Repeat from step 2 with the new hyperparameters until there are no more iterations. By default, two iterations are used, the first for determining the magnitude of the parameters and the second for fine tuning the architecture after finding the correct magnitudes.

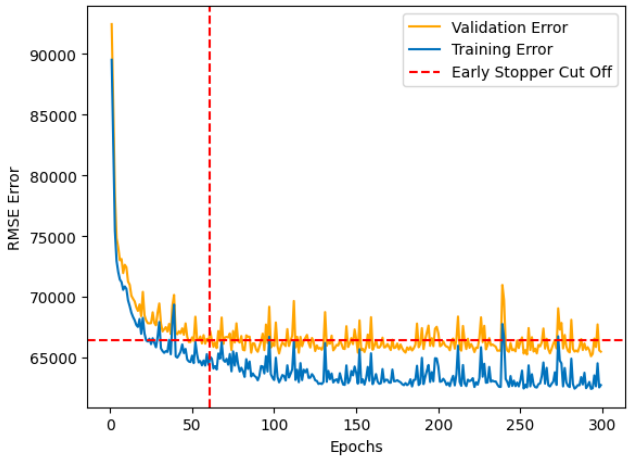
This is a general approach and while it appears to be computationally expensive, models that take too long to train are eliminated early due to the early stop function described in Section 1, and as such this method takes under an hour to perform two iterations whilst producing good results. The Regressor Class was given two new methods that enable it to be compatible with sklearn's GridSearchCV, which are get_params() and set_params(), used for retrieving and setting the hyperparameters in the model. As GridSearchCV sets the parameters after the model has been instantiated, it is crucial that the model is built after this point and not in the init function. This is done at the start of the fit function.

Additionally, as the early stop method is active during the hyperparameter tuning (as we have a validation set), we can infer the 'nb_epoch' hyperparameter by taking the epoch at which the best-performing model stopped at. This is to ensure that models trained without the use of a validation set do not overfit to the training data.

3.2 Analysis of Hyperparameter Tuning Results

To help better quantify the impact of the different parameters, we show the impact of varying the parameters on the optimum configuration: batchSize: 256, learningRate: 0.1, neuronArchitecture: [9,9,9] – which gives an RMSE of 65911. The reported epoch cut offs by the early stopper were 55, 58 and 69 (there is a range due to the 3-fold cross validation). We take the average as 61.

Learning Rate	RMSE
0.01	65867
0.1	64995
1	67832
Batch Size	**RMSE**
512	65485
256	64995
128	68553
64	66022
Hidden Layers	**RMSE**
1 Layer	70298
2 Layers	65548
3 Layers	64995



From these results, it shows that varying just one of the hyperparameters from the optimum is not enough to ruin the performance of the model, however the worst configuration had an RMSE of 164052 (batchSize: 512, learningRate: 0.01, neuronArchitecture: [9]), demonstrating that hyperparameter tuning is required to an extent. Additionally, from the graph, the early stopper cut off works correctly as it terminates training at epoch 61 and past that point there does not seem to be any significant reduction in RMSE despite training up to 300 epochs. This validates the 'minimum percentage improvement' factor hyperparameter described in Section 1, set as 0.005%.

Following the approach of the previous section, the hyperparameters generated after the first iteration were: learningRate: [0.14, 0.09, 0.65, 0.54], neuronArchitecture: [[13, 10], [11, 8], [10, 9], [10, 7]], batchSize: [60, 73, 85, 103]

From these parameters, it can be inferred during the first iteration the algorithm discovered the optimum learning rate lies in between 0.1 and 1, the optimum batch size to be between 64 and 128 and the optimum layer count to be 2. Based off these values, it then generated the 4 slightly randomised parameters within each category to use for the next grid search which are the parameters shown above.

It should be noted that it is possible that the 'best' base parameters may not be within the range the algorithm determines to be the optimum, this is because the algorithm opts to use parameters that are more common amongst the top performing candidates, rather than an uncommon combination that would be reported as the 'best'. For instance, while the best performing model has a batch size of 256, the optimum range is 64-128.

After evaluating these models, the mean RMSE was 72190 with a range of 98141, meaning that there is a large variation in performance depending on which initial hyperparameters are chosen, which is expected.

4. Final Model Evaluation

After the second iteration of hyperparameter tuning, the optimum performance was given by the following parameters: batchSize: 103, learningRate: 0.09, neuronArchitecture: [10, 7], and this model achieved an RMSE of 65633, averaged across 3 folds. The epoch cut offs reported by the early stopper were 78, 82 and 86, yielding an average of 82. Therefore, 82 is the final value for the nb_epoch hyperparameter. During the second iteration of tuning, the average RMSE was 67189 and the range was 4421, demonstrating that during the second iteration, all models perform very similarly and thus reaffirming that only two iterations are needed.

Given that the median house values in the dataset range from 15000 to 500000, an RMSE of 67189 is acceptable, and we can conclude that the regressor is able to produce meaningful predictions about median house prices given the dataset it was trained on.