Coursework 2: Artificial Neural Networks

Introduction to Machine Learning

Katherine Miles, Kaixuan Khoo, Gus Levinson, Barry Quek

March 4th, 2022

2.1 Implement and architecture for regression

To train our model, we used 80% of the 'housing.csv' dataset, splitting out 20% for testing.

The dimensions of our neural network (initially) were as follows: we have a single hidden layer that is 50 neurons in size. (Results detailed in 2.2) The input layer has 13 neurons, corresponding to the number of features/columns in our pre-processed training sets, while the output layer has just 1 neuron. The input size is simply standard practice in feed-forward networks, while the output layer has just 1 neuron because our model we wish for the model to predict a single continuous value for each data instance.

Random weights were assigned to each connection and we used sklearn's RobustScaler to normalize the data. As stated in the docs, this scaler removes the median and scales the data according to the to the interquartile Range) (i.e., between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile)). This was to ensure that the outliers in our data would not affect the model's training and testing algorithms. The dataset had one column ('ocean_proximity') that contained categorical values, which we encoded using one-hot encoding.

We initially trained our model across 100 epochs with a batch size of 32. We found that this was sufficient for the Average Training Loss per epoch (calculated using root mean square error) to plateau at a relatively low value.

2.2 Set up model evaluation

Following model building and training, the deep learning neural network can be used for prediction of median house pricing. The predict method takes in and preprocesses an X (`x_test` or `x_val`) into a tensor and from those values calls the self model and predicts a corresponding Y. The score is then calculated from these predictions as the root MSE between the predicted Y values and the true Y values.

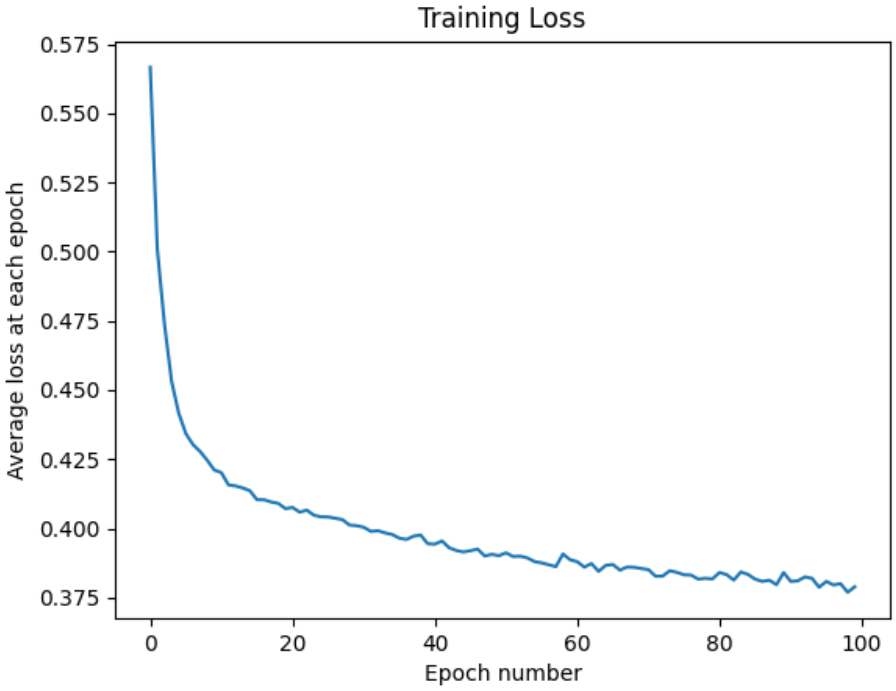

Figure 1: Graph showing training loss decrease over 100 epochs in example main

2.3 Perform hyperparameter turning

For hyperparameter tuning, we chose to adjust our model on the following: learning rate, batch size, number of neurons, type of layers and number of layers. First, we postulated sets of values for each of these hyperparameters that we thought might have a positive effect on the error score. Then, we iterated through each of these sets and trained + scored regressors on them from the training and dataset. More specifically, we did this in a compounding fashion: we began with finding the best learning rate value from our set, then we used this value and found the best batch size, and then found the best model dimensions.

2.4 Final Evaluation of Best Model

From this, our optimal model had a learning rate of 0.1, batch size of 64, and two hidden layers with 100 and 50 neurons with layers `ReLU` and `Sigmoid`. This model was an improvement over the previous model, reducing the error by approximately 6%.

In each of the hyperparameter tuning steps, we use an `epoch` size of 100, and vary the `learning_parameter`, finding the model with the lowest mean squared error (MSE). Then, we keep this best value for learning parameter and find the best `batch_size`, again to see how the best value of batch size affects MSE. Finally, to test other network architectures, we've tested other sizes for the hidden layers. Here, the list of integers represent the number of neurons in each hidden layer. From Figure 2, we can see that the best learning rate was 0.001. From Figure 3, we determine that a larger batch size leads to a better model, and a model with architecture of [100, 50] is the best.

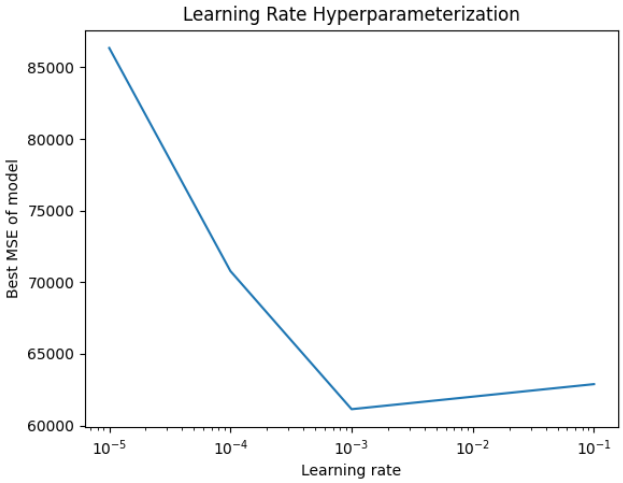

Figure 2: Best learning rate from hyperparameter tuning

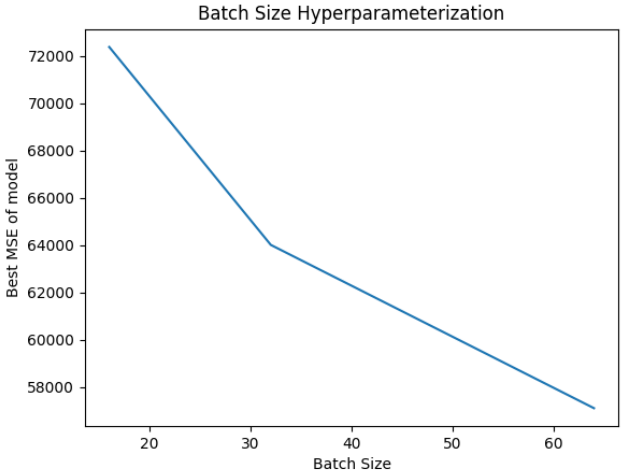

Figure 3: Best batch size from hyperparameter tuning

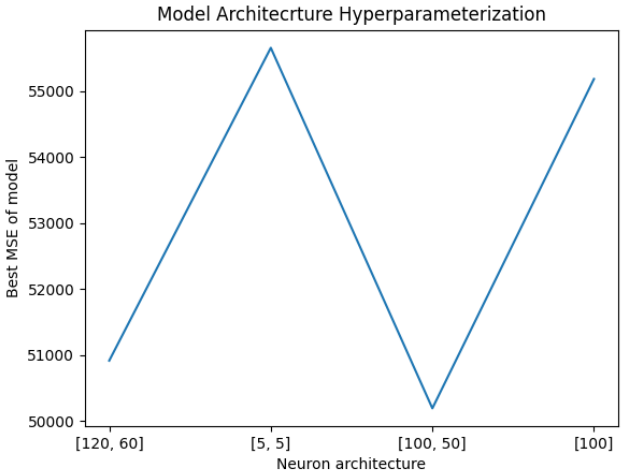

Figure 4: Best neural network architecture from hyperparameter tuning