Introduction to Machine Learning
Coursework 2 Report

He Liu, Kejian Shi, Qianyi Li, Shitian Jin

November 26, 2021

1 Description of the model

1.1 Preprocessor

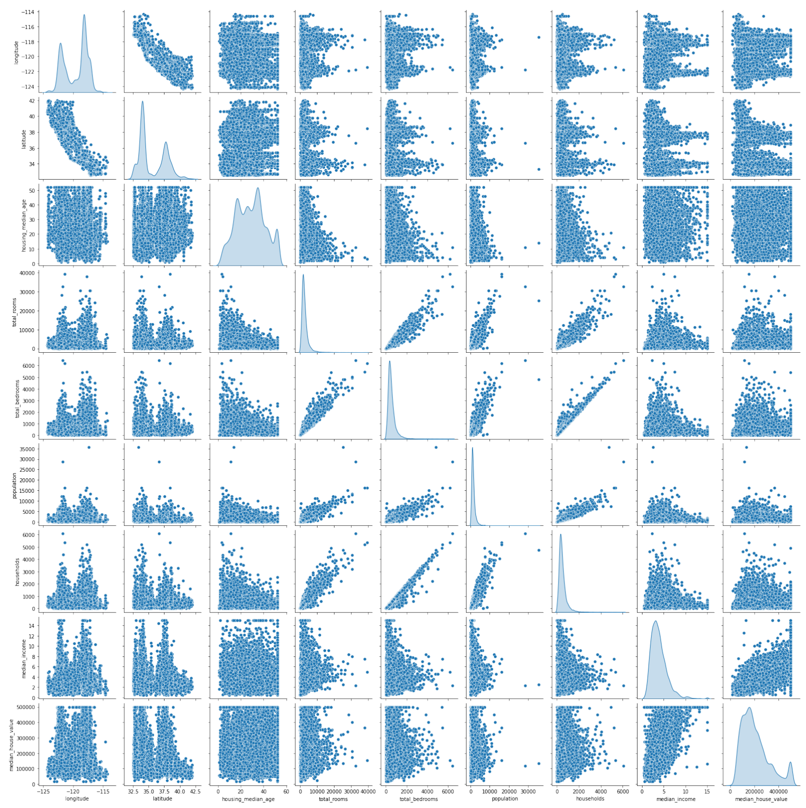


Figure 1: **Scatter Matrix to Investigate the Data**

The first thing we did is to investigate the data. Here are some interesting facts we found:

- The negative correlation in **Longitude** and **Latitude** is not expected, the correlation shown in scatter matrix is the shape of California.
- Not all the features have good shape of norm distributions, we can apply the feature scaling to pre-process the data, for example:

$$z = \frac{(x - \mu)}{\sigma}$$

  where μ is the mean of the data attribute and σ is the standard deviation.

- We use MSE as loss function, overflow may happen during training since the value of original labels (median_house_value) in the training data are large (14999-500001). Hence, we use log(label) instead of label to compute loss.
- Some features are strongly correlated to others. For example, **the number of rooms** and **the number of bedrooms** are strongly positively correlated, which makes sense intuitively. This correlation suggests an alternative way to impute the missing data for **the number of bedrooms** based on **the number of rooms** by developing another linear regression model.

1.1.1 Missing data

We filled in any missing data with the mean value of that feature. We first loaded up the housing dataset using preprocessor, if there's any missing value, we fill in it using the mean of all numeric attributes of this feature by **fillna** method in Pandas package.

1.1.2 Categorical Attribute

We converted the categorical attribute (i.e. ocean_proximity) to one-hot by **get_dummies** method in Pandas package. This has to be done because categorical attributes cannot be passed into neural networks, we have to convert it to one-hot first.

1.2 Architecture

In this part we implemented the regression model. We created a **Fully Connected Neural Network (FCNN)** with **4 hidden layers** by setting the input size according to the dimension of training data, and setting output size as one to solve the linear regression problem. For the input layer and all the first 3 hidden layers, a **ReLU** is applied after each of them as an activation function since ReLU can speed up network training and achieve good results at the same time. **Dropout layers** are also added after each of the first five layers to avoid overfitting during training. Here are other architecture methods we used worth mentioning:

- We used a neuron decay method, which means the neurons' number will decay with increasing of layer number. The number of neurons in each layer is the number of neurons of first layer divided by the layer number(i.e. Neuron number of a layer = $\frac{Neuron\ number}{layer\ number}$, where **number of neurons** is a hyper-parameter we will tune layer).
- Momentum(=0.9): Momentum is implemented to prevent converging to local minimum.
- Mini-batch(batch size:32) is applied during training, instead of updating the parameters with every sample in dataset, we update the parameters after every batch. This can boost up our training process while keeping its accuracy.
- L2 can keep the magnitudes of the weight values small adding a term to the error function, to prevent overfitting. The dropout method and L2 are both applied to ensure a good performance of model.[2](2014,Srivastava,Nitish)

2 Description of the evaluation setup

In this part we evaluated the neural network with Coefficient of Determination(R^2 score) from scikit-learn library. It calculates the score with the following equation:

$$R^2 = 1 - \frac{RSS}{TSS}$$

where RSS is the sum of squares due to regression (explained sum of squares) calculated by $RSS = \sum_{i=1}^{n}(y_i - f(x_i))^2$, which measures how well the regression model represents the data that were used for modeling. And TSS is the total sum of squares, calculated by $TSS = \sum_{i=1}^{n}(y_i - \overline{Y}))^2$, which measures the variation in the observed data.

The reason choosing R^2 score instead of MSE or MAE is that the latter measurements range from 0 to +infinity, thus it is hard to determine the performance of regression from a single value of them. Instead, R^2 score range from 0 to 1(by our definition there will be negative R^2 scores but that means really bad performance of regression so we only care about the range [0,1]) so that by calculating the R^2 score we can determine how well the data fits the model[1](2021, Davide Chicco). In other wards, how good are given features (Input data) mapped to the houses' prices (Output). The closer to 1 the score is, the better performance of the model, and a 0 score means the regression model represents none of the data.

However, it is not wise to rely solely on the measure in R^2, since it depends on several factors, for instance the applied data transformation(Pre-processor). Thus we have to check the learning outcome using the training and validation errors. We divided the whole dataset into training set and validation set with a fraction of 8:2. Then we used the training data to train the model and evaluated the R^2 score using validation set after training.

The average of training and validation loss was also measured using MSE after every epoch to illustrate that the regression model is fitting well as plotted in Figure 2.

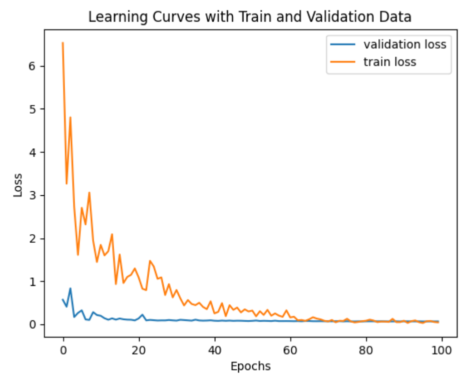


Figure 2: **Data loss with respect to epochs from train and validation set**

3 Information about the hyperparameter search

In this section we performed a grid search for parameter tuning. Here are the parameters we investigated:

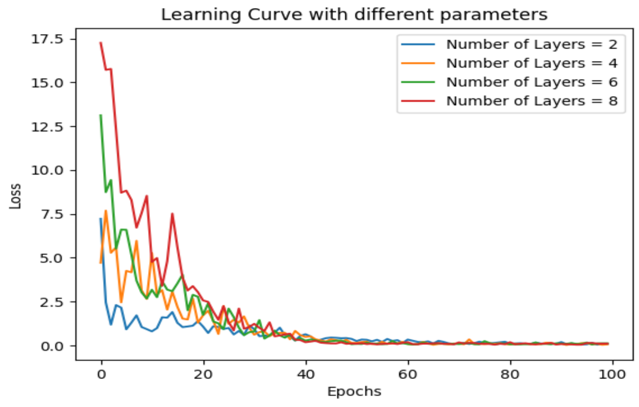
Hyperparameters				
Number of layers	2	4	6	8
Number of Neurons(First layer)	16	32	64	128
Learning Rate	0.01	0.001	0.0005	
Scaler Method	Standard	MINMAX	Robust	
Batch Size	32	64		
Activation Function	Relu	tanh		
Drop Out Num	0.2	0.4		
Weight Decay(L2)	0.0	1e-5		
Momentum	0.9	no		

After comparing all combinations of hyper-parameters, we found the regression model performs best with such parameters below. A reminder is that the neurons in each layer is decaying with respect to its layer number.

Scaler	Layer	Neuron(first layer)	Activation	Dropout	Optimizer	Learning Rate	L2	momentum
Robust	6	64	Relu	0.2	Adam	0.001	1e-5	no

There are more than 1000 combinations of these hyper parameters. For each combination of hyper-parameters, we fitted model using training set and evaluated its performance using validation set. We used the R^2 score explained in Q1 to evaluate the performance along with the learning curve(i.e Figure 3). Another factor: the consumption of time is considered as well.

To help illustrate the difference on performance with varying hyper-parameters, we provided the 'Parallel Contrast', which evaluates the performance of different choices of one single hyper-parameter while keeping all others the same. An example result is shown in Figure 3a, 4a and 5a:

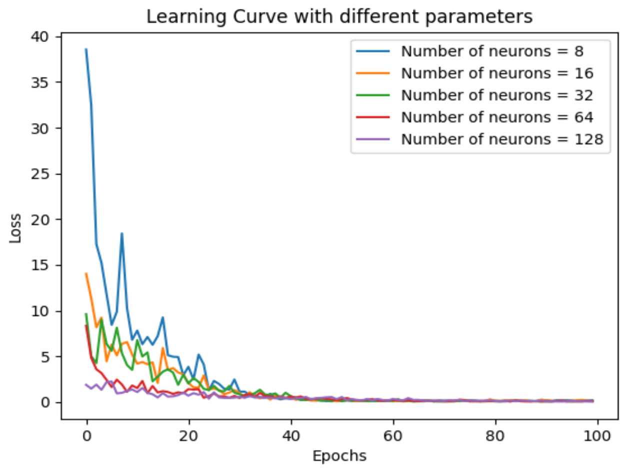


(a) learning curves of different number if layers

No. Layers	Time(s)	score
2	59	0.686
4	91.3	0.717
6	120	0.736
8	251	0.744

(b) Evaluations with different number of layers

Figure 3: num_neurons(first layer) = 64, Batch size = 32, Activation = 'Relu', L2 - 1e-5, Dropout = 0.2,lr = 0.01

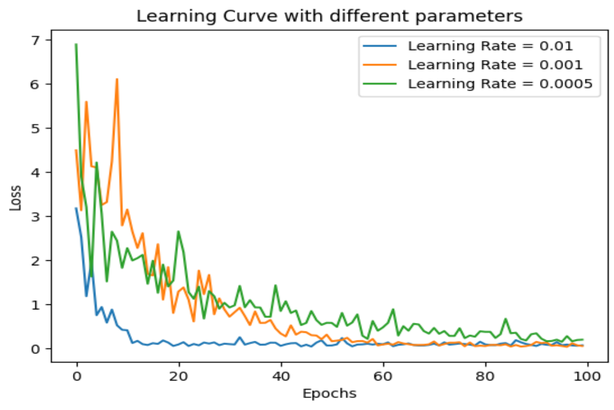


(a) learning curves of different neuron numbers

No. 4-layer Neurons	Time(s)	score
8	46.1	0.636
16	46.7	0.718
32	55.2	0.752
64	64.4	0.770
128	93.9	0.770
256	174.0	0.770

(b) Evaluations with different number of neurons

Figure 4: Layer = 4, Batch size = 32, Activation = 'Relu', L2 - 1e-5, Dropout = 0.2,lr = 0.01



(a) learning curves of different learning rate

Learning Rate	Time(s)	score
0.01	77	0.71
0.001	73	0.76
0.0005	81	0.71

(b) Evaluations with different learning rate

Figure 5: Layer = 4, num_neurons(first layer) = 64, Batch size = 32, Activation = 'Relu', L2 - 1e-5, Dropout = 0.2

The above graphs and tables are just part of the evaluations we did in this section. What we want to emphasize here is the choice of hyper-parameters from time consumption perspective. It is noted in Figure 3b, where the score for 8 layers is clearly a little bit higher than 6 layers, but the time spent is twice longer. In this case we will choose 6 layers because with similar performance, the time for training is much shorter. Another fun fact to know is that there is some kind of consistent between layer numbers and number of neurons for each layer. A Neural Network(NN) with more layers(i.e. 6 layers) will require more neurons for each layer to give a good performance(i.e. 64 in the first layer). And similarly, a NN with less layers(i.e. 2) perform well with small number of neurons in each layer but badly with larger neuron numbers. After comparison we still believe 6 layers with neuron number 64 (In the first layer, decay in each subsequent layer) can give us the best performance.

4 Final evaluation of the best model

As illustrated in Figure 3a, 4a and 5a, manipulating the number of layers and the number of neurons in the layers will alter the performance of the regression model. Therefore, we have explicitly tuned the architecture of the neural network by selecting the one that produces the smallest loss on the validation set, as shown in Section 3. After we got our model with best performance, we run 5 replications of the model to see whether the performance is stable, as shown in Figure 6:

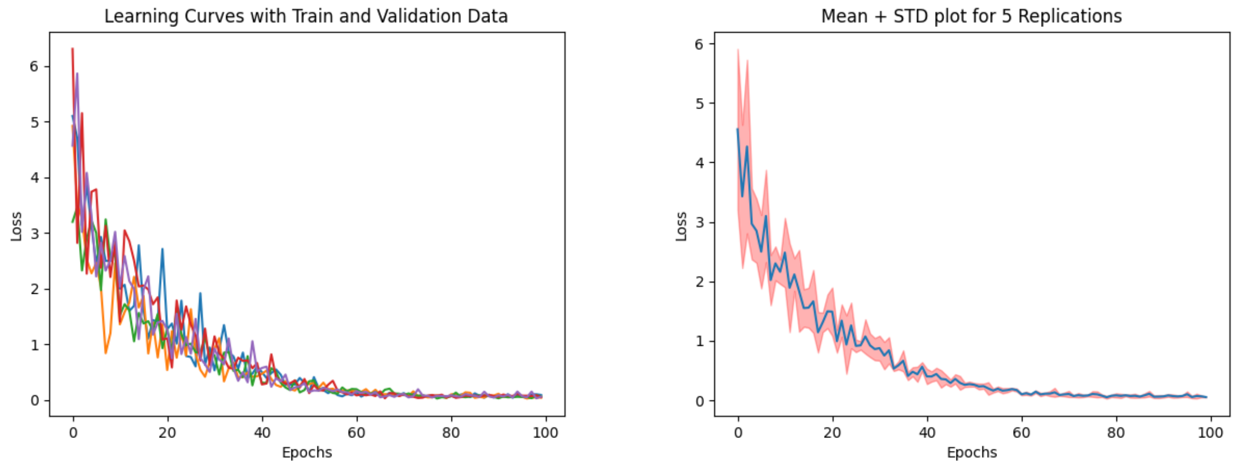


Figure 6: **Learning Curve with Best Model with 5 Replications**

And also the R^2 scores shown in Table 1:

0.746	0.772	0.780	0.778	0.777

Table 1: Score with 5 replications of best model

which we can confidently say that our model is stably performing well!

References

[1] Chicco D, Warrens MJ, Jurman G. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. PeerJ Comput Sci. 2021 Jul 5;7:e623. doi: 10.7717/peerj-cs.623. PMID: 34307865; PMCID: PMC8279135.
[2]Nitish Srivastava,Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014) 1929-1958