Housing Price Prediction on Kaggle

Prithvi Raj Singh
Center for Advance Computer Studies
University of Louisiana at Lafayette
Lafayette, US
prithvi.singh1@louisiana.edu

***Abstract*—In this paper, we will be summarizing our work on the Kaggle Housing Prediction competition. We used the D2L book as our reference worked on tuning the hyperparameters and observed the performance of our model. The result of submission on Kaggle for official testing is RMSE. Our work on tuning the hyperparameters didn't necessarily bring any positive change. We have also tried to answer and explain our model, variable standardization, layer count, and regularization. The best performance our system reported was a training error of 0.126395 and, a valid error of 0.146429, but our chart showed that the model was underfitting. We obtained a score of 0.15151 on our official submission on Kaggle.**

***Index Terms*—Hyperparameter, RMSE, Tuning, Standardize Variable.**

I. Introduction

The housing market is a very volatile market, the price of property can swiftly change in a very short time. There have been several efforts in developing advanced Machine Learning techniques that can precisely predict the price of a house at a given time, and location. We basically collect and feed old housing market data with several features to our Machine Learning model and it gives the estimate given a new set of data. In this paper we will participate in an online Kaggle [2] Competition to predict the housing market and explain the technique we used as stated in [1]. We will also explain the different error values we get as we change the hyperparameters and re-run our model. We will be following the instruction from the D2L book [**1**] and explain the exercise problem from Chapter 4 'Prediction of Housing on Kaggle'.

II. Methodology

For our experiment, we built a linear regression model with a single sequential layer. We will be training our model with square loss. The mean squared error gives us the means of squares of errors between labels and prediction. Since the housing market is based on different geographical, and social quantities we are interested in getting the relative error over absolute error. We will use the log RMSE (Root Mean Square Error)to compute the relative error. RMSE calculates the error between the log of the predicted and the log of the label price. For the model that we submitted on Kaggle, we used Adam Optimizer. We perform K-fold cross-validation on our training dataset. Our model gives training and validation error averages based on the times we train. Tuning the hyperparameters of our model we can see changes in our error values. In each fold of the training feature, train label, epochs, learning rate, and weight decay our model will iterate our given fold number (K). Each iteration will have given epochs, learning rate, batch size, and weight decay. Our validation model will give us the train log RMSE and validation log RMSE for each fold. After initial testing and tuning our hyperparameters, we go on to use the entire training dataset for training purposes and use our model on the test dataset for submission on Kaggle.

$$\text{log RMSE} = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(\lg y_i - \lg \hat{y}_i\right)^2}$$

III. Results

In this section, we will explore the performance of the model based on a particular set of hyperparameters used. We will start with the base model as mentioned in the D2L book and also explain the results of our own hyperparameter tuning.

A. Performance on Untuned Hyperparameters

Based on untuned hyperparameters as used in the D2L book [1] we see the train log RMSE and validation log RMSE to be close to what is mentioned in the book. We get the average train log RMSE of 0.165537 and validation log RMSE of 0.170339. Here we are doing 5-fold cross-validation with 100 epochs, a learning rate of 5, no weight decay, and a batch size of 64. The chart shows that the initial model has no overfitting or underfitting.

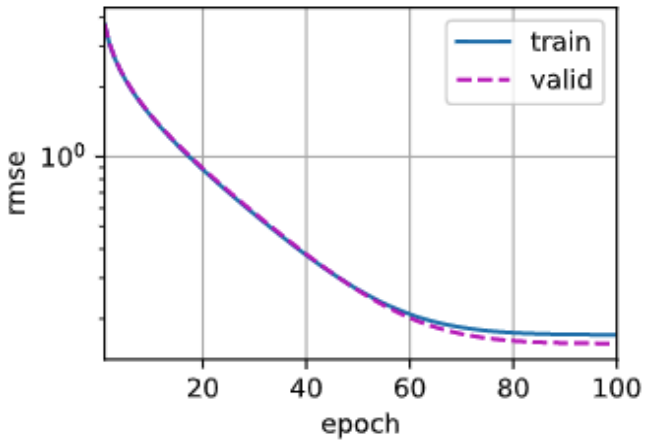

Fig 1: Average Train and Validation log RMSE on untuned hyperparameters.

Self-funded with time and energy!.

B. Performance with Hyperparameter Tuning

We will observe the performance of our model based on tuning the weight decay, learning rate epochs, batch size, and different values for K-fold cross-validation. With 10, 200, 3e-3, 0.3, and 32 as our number of the fold, epochs, learning rate, weight decay, and batch size respectively we will see that the average train and valid errors are 5.086963, and 5.089255 respectively. This is not an ideal result even though our model shows no sign of overfitting. Changing the epochs, and batch size, while keeping the learning rate and weight decay the same as above doesn't produce any competitive results.

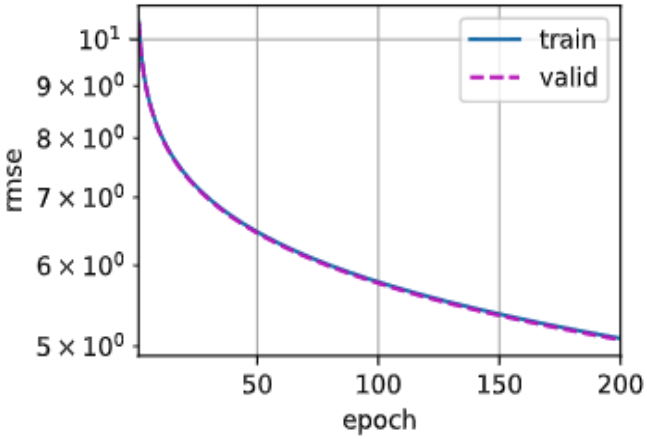

Fig 2: Avg train and valid error tuned as mentioned above.

If we radically increase our learning rate to 3e1, reduce the weight decay to 1e-2, and have a batch size of 128 keeping everything else the same will give us considerably better and competitive results. We will have the average train and validation log RMSE of 0.126395, and 0.146429 respectively. The chart (Fig 3) shows a linear decline in error value and a slight deviation of train and validation error. Our Chart shows that the model is overfitting.

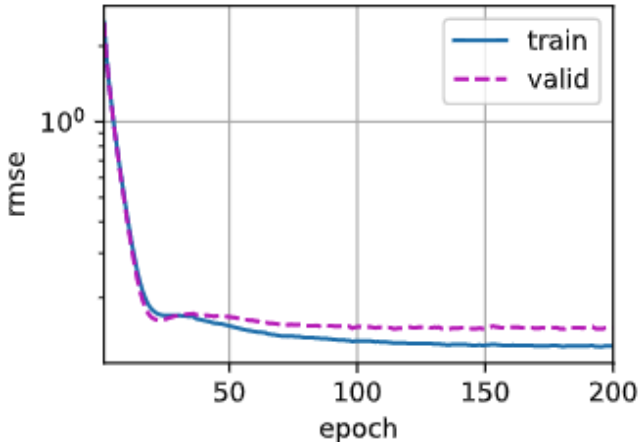

Fig 3: Avg train and valid error tuned as mentioned above.

Tuning the learning rate, weight decay, and batch size doesn't necessarily give us the best result we are looking for.

Changing the hyperparameters won't necessarily give us any better results. In the experiment we conducted by adding a number of layers in the model with L2 regularization, we did see any improvement. In fact, there was massive overfitting when we added one extra dense layer. The results were 0.122865 and 0.214159 for average train and validation log RMSE. We used 10, 100, 5, 0, and 64 as our k, epochs, learning rate, weight decay, and batch size. Refer to Figure 4.

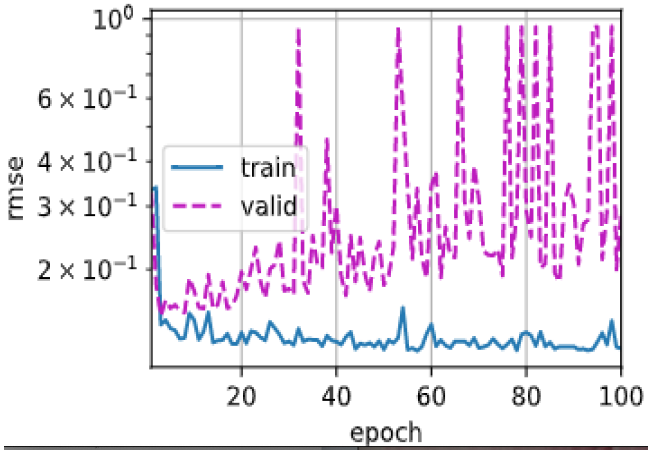

Fig 4: Avg train and valid error tuned as mentioned above.

When we added two extra layers and a dropout layer we got appropriate fitting but the results were way higher and noncompetitive. We achieved average train log RMSE and valid log RMSE of 2.051237 and 2.051151 respectively. Any extra layer addition just worsens the accuracy.

When experimenting with the optimizer we changed the Adam optimizer to SGD optimizer and kept the learning rate to 0.01. The result is huge overfitting even though the average log RMSE of 5-fold validation is close.

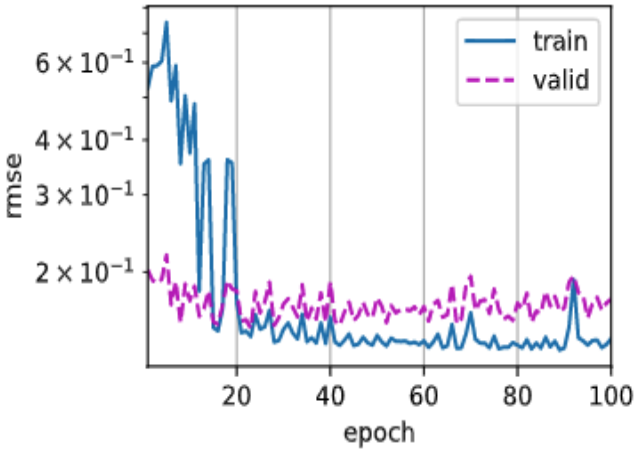

Fig 5: Avg train and valid error with SGD optimizer used.

The addition of layer reduces the average train loss but the model always shows overfitting. The use of a dropout layer or weight attenuation doesn't really help. We can't achieve appropriate fitting or better accuracy tuning hyperparameters. In one of the final tries, we got an average train error of 0.134213 and a validation error of 0.200266. Our model shows significant overfitting as shown below.

TABLE I

K	Epochs	lr	wt. decay	batch size	RMSE	
					Train	Test
5	200	2	0.03	32	0.134	0.200
5	200	3	0.3	128	0.116	0.174
10	100	5	0	64	0.122	0.214
10	100	4e1	1e-2	128	0.129	0.143
5	100	5	0	64	0.165	0.170
10	200	3e-3	0.3	32	5.086	5.089

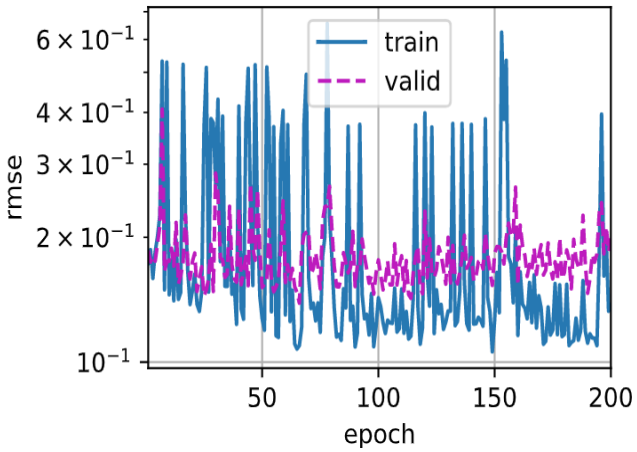

Fig 6: Representing model with activation function and dropout layer.

When we used the whole dataset for our model to train on we got the train log RMSE of 0.136248, with the 200 epochs, learning rate of 5, weight decay of 0, and batch size of 64.

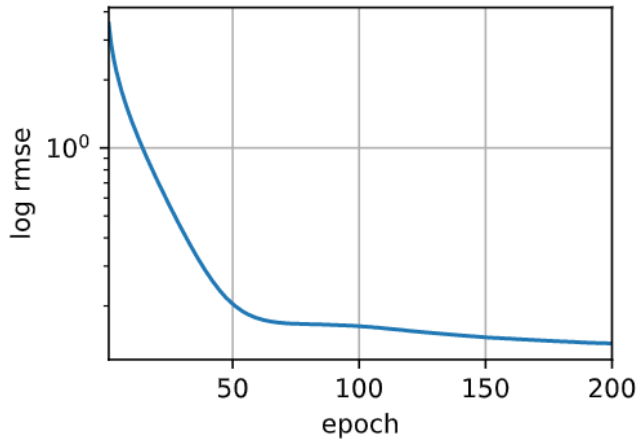

Fig 7: Our final result using who data for training.

Our model has used Standardized Variables to balance the feature size. If we don't use standardizing variable we will get improper feature size that is different for different features. We will get an excess amount of multicollinearity. If a standardized variable isn't used we are at risk of getting both missing statistically significant results and producing misleading results.

IV. Discussion

Our attempt was to do some hyperparameter tuning and increase the efficiency of our model. We even tried to optimize our model by increasing the layer count and implementing regularization, but our model didn't show any significant overall improvement. The results of experimenting with hyperparameters weren't exactly or even close to our expectations. Adding any extra sequential layer and regularization made performance worse. We have a feeling that the dataset given is very simple and might not be good for working with many layers of deep networks. If we can use other datasets with similar features we can train our model on bigger datasets and see how the regression model performs. We have to test our model with so many different numbers for hyperparameters to tune our model to have peak performance.

V. Conclusion

Our target was to minimize the error optimally so that we could achieve a higher ranking on the Kaggle Competition, but we failed. Our experiment didn't perform as expected. The use of more layers, regularization, and dropout worsen the performance of the model instead of making it better. We have a general understanding that increasing the layer count might not be the best thing to do for this problem. The only proper results we achieved were the average train log RMSE of 0.165537 and validation log RMSE of 0.170339, which isn't an optimal result to win us the Kaggle competition. The use of ReLU should be thought of since it reduces the gradient descent of the model. Experimenting with the layer count and regularization didn't increase our performance. Our result of submission on Kaggle was 0.15151

VI. Future Work

One of the most important things for us will be the full implementation of our own code for the Housing Prediction problem. We believe doing our own coding implementation, filtering out unnecessary features in the dataset, and adding features that may be needed can significantly peak the model performance. We shall also try using different optimizers. In the future, we shall also focus on layer counts and regularization despite these things not working as our expectation.

References

[1] Zhang, Aston, Zachary C. Lipton, Mu Li, and Alexander J. Smola. "Dive into deep learning." arXiv preprint arXiv:2106.11342 (2021).

[2] "Kaggle: Your Machine Learning And Data Science Community". Kaggle.Com, 2022, https://www.kaggle.com/.