Using a Machine Learning Model for Determining Housing Prices

Connor Rawls
C00211180@louisiana.edu
University of Louisiana, Lafayette, Louisiana,

***Abstract*—This paper performs an investigation on the ability of training and utilizing a machine learning model for the prediction of a commonplace socioeconomic problem: determining the price of houses based upon their specific features. An original model was provided as a base-line to test the performance. Several alterations such as hyperparamaters and topology were explored in the attempt of increasing model performance and explore the affect of each dynamic. The results show that a lightly-tuned version of the original model performs the best, with an error rate of 15.33%.**

I. Introduction

MACHINE learning provides a powerful tool for conquering the questions that inhabit our daily lives. In the case of this paper, that question is the pricing of houses based upon their respective features, such as, year of construction, roof type, and street. This problem is presented through and guided by *Dive into Deep Learning* [1]. The purpose of this paper involves developing, training, testing, and evaluating a machine learning model for the proposed problem. This paper hypothesizes that the by altering the original model's hyperparameters and topology, the model's error rate will decrease, in turn, increasing inference accuracy.

II. Methodology

A. Data

The data used for training and testing purposes come from *Kaggle*, an online machine learning resource that introduces open competitions as an incentive to develop capable models [2]. The training dataset contains 1460 records, with each entry possessing 80 features. The entries are representative of a housing unit containing certain characteristics. The testing dataset contains 1459 entries with the same amount of features as the training set. Kaggle is additionally used as an online scoring mechanism during the testing phase of the experiment.

B. Hyperparameter Tuning

After a prototype model was operational, the model's hyperparameters were tuned. This was done through K-fold cross validation. Specifically, the a study was done to examine the effect k, epoch number, and learning rate have on the model's performance. During these tests it is important to minimize the model's training and valid Root-Mean-Squared-Error, while additionally minimizing the difference between the two values. The reason behind minimizing the difference between the two is to avoid overfitting the training data, leading to poor generalization. The default hyperparameter values can be observed in Fig. 1, with the tuned hyperparameters in Fig. 2.

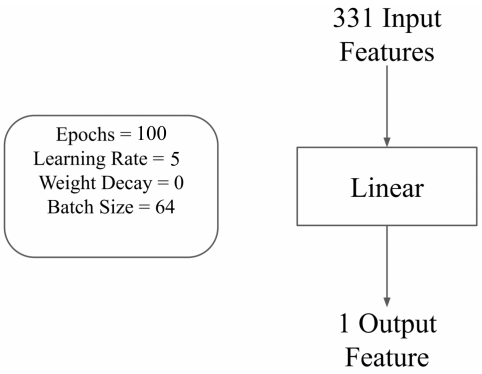


Fig. 1: Prototype model hyperparameters topology.

C. Regularization and Topology Exploration

In this experiment, the regularization of the model was examined through the possible values for weight decay. The model development was aided by referring to [1]. The prototype model's topology can be viewed in Fig. 1. After running the preliminary tests and tuning the hyperparameters, the model's topology was altered in seek of improving its accuracy. This final model's design can be observed in Fig. 2. The amount of layers, as well as the type of layers, were explored in the attempt to develop a better model topology than the default one.

D. Standardized Continuous Numerical Features

In the original experimental tests, the input variables were standardized. Generally, this procedure is done to expose the relative error of the model, rather than the absolute error. The relative results, in this case, provide a good depiction of how usable our results are. A comparative study was done to show how this standardization affects the accuracy of the different models used in this experiment.

III. Results

A. Hyperparameter Exploration

The results of the hyperparameter tests can be viewed in Fig. 3. The result columns are *avg train rmse*, *avg valid rmse*,

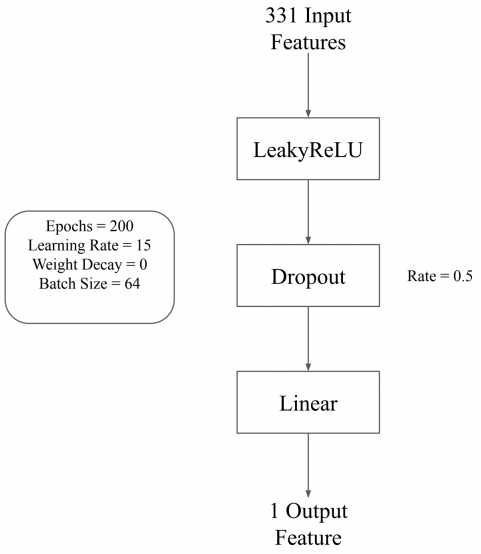
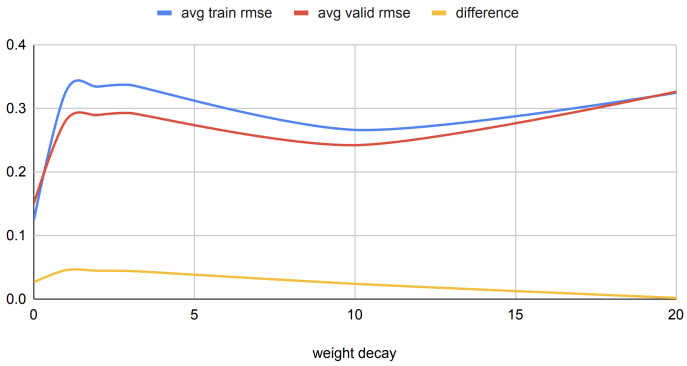


Fig. 2: Complex model's tuned hyperparameters and topology.

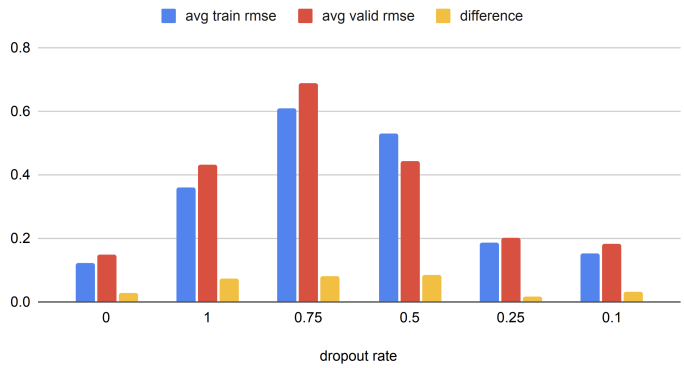
Hyperparameter Exploration

k	epochs	learning rate	weight decay	batch size	avg training rmse	avg valid rmse	difference	test score
5	100	5	0	64	0.165522	0.170639	0.005117	0.16727
2	100	5	0	64	0.194062	0.199454	0.005392	
10	100	5	0	64	0.164563	0.168028	0.003465	
5	**50**	5	0	64	0.264878	0.268189	0.003311	
5	**150**	5	0	64	0.153759	0.162586	0.008827	
5	**200**	5	0	64	0.127743	0.147683	0.01994	
5	**250**	5	0	64	0.124733	0.149438	0.024705	
5	**300**	5	0	64	0.122679	0.15154	0.028861	
5	100	**2**	0	64	0.362646	0.366001	0.003355	
5	100	**10**	0	64	0.145483	0.155973	0.01049	
5	100	**15**	0	64	0.134964	0.148958	0.013994	
5	100	**20**	0	64	0.129858	0.147998	0.01814	
5	100	**25**	0	64	0.127056	0.14898	0.021924	
5	100	**30**	0	64	0.124725	0.149659	0.024934	
5	**200**	**15**	0	64	0.123569	0.149997	0.026428	0.15333

Fig. 3: Effect of hyperparameters on model performance.



(a) Weight decay.



(b) Dropout layer with variable rates.

(c) Activation functions.

Fig. 4: Effects of regularization (a) and layer design (b c) on the model's performance.

difference, and *test score*. The *rmse* columns show the average Root-Mean-Squared-Error of the model after going through K-fold cross validation. The *difference* column shows the absolute difference between the two rmse values. This metric helps visualize the intensity of the model's generalization. The *test score* column depicts the models accuracy during the testing phase. This phase occurs through *Kaggle*'s interface and they provide a score. The blue and green highlighted rows depict the default and best hyperparameter configuration, respectively, while the two green boxed cells show the parameters that provided the greatest improvement. The number of epochs and learning rate value were chosen as providing the greatest improvement as they had they minimized both the training rmse and difference. For the sake of not bogging-down *Kaggle*'s servers, the testing was only done utilizing the best configuration of hyperparameters found through K-fold variation. The results show that the tuned model holds a 8.33% increase in test score over the original.

B. Design Exploration

Utilizing the tuned hyperparameter configuration of the model, the model's regularization and topology was explored. The reason behind testing the topology design only on the tuned hyperparameters and not on the original design, is that the ultimate goal of this experiment is to maximize the model's performance (minimize the *test score*). As we know the tuned hyperparameters hold a better performance over the original configuration, we may simply test only this tuned option for the sake of brevity. The preliminary test results showing the effects of utilizing regularization and dropout layers in the network can be viewed in Fig. 4. Additionally, this figure also shows the effect on performance different activation functions possess. The green-highlighted row represents the hyperparameter-tuned configuration from the previous tests in Section III.A. It can be seen that, in all three tests, the original, hyperparameter-tuned model holds the best performance. For the sake of testing different topologies, the best performing dropout rate and activation function were used in the continuation of the experiment. Out of the complex

Model Topology

linear	LeakyReLU	dropout	avg train rmse	avg valid rmse	difference	test score
1	0	0	0.123569	0.149997	0.026428	0.15333
1	1	0	0.126759	0.154227	0.027468	
1	0	1	0.530154	0.443807	0.086347	
1	**1**	**1**	**0.216886**	**0.220882**	**0.003996**	**0.23184**
2	0	0	0.298358	0.30655	0.008192	
2	0	1	0.231874	0.686821	0.454947	
2	1	1	3.365504	3.601524	0.23602	
3	0	0	5.032032	4.95297	0.079062	
3	1	0	3.626251	3.770322	0.144071	
3	1	1	6.161998	6.318375	0.156377	
3	2	0	6.466961	6.522843	0.055882	
3	0	2	7.401005	7.446148	0.045143	
3	2	2	9.840349	9.809041	0.031308	

Fig. 5: Comparison of various combinations of possible layers in model.

topology configurations, the LeakyReLU activation function performed the best. With consideration of the dropout rate, the rate of 0.1 performed the best. However, for the topology tests, a dropout rate of 0.5 was chosen as this provided a modest middle-ground showcasing the best performing rate while still exhibiting the behavior of including a dropout layer.

Alongside exploring the possible choices for regularization and utilizing different layers, the model's topology itself was also tested. The different topologies compared can be viewed in Fig. 5. As in the figures before, the green-highlighted row shows the hyperparameter-tuned-only topology. The red-highlighted row shows the best performing topology out of the complex topology configurations. It can be seen that the best performing complex topology shows a 33.86% performance degradation over the hyperparameter-tuned topology.

C. Standardization

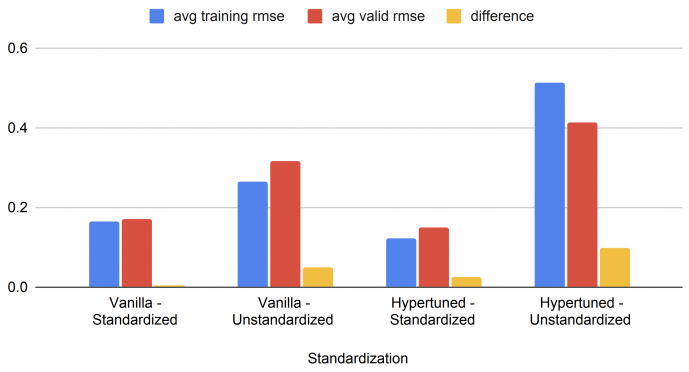
The use of standardizing the continuous numerical features was tested using the three models: *vanilla* (Fig. 1), *hypertuned* (Fig. 3), and *complex* (Fig. 2). The results of these tests can be viewed in Fig. 6. One can observe that in all cases, using unstandardized features decreases model accuracy as well as the difference between training and validation error.

IV. Discussion

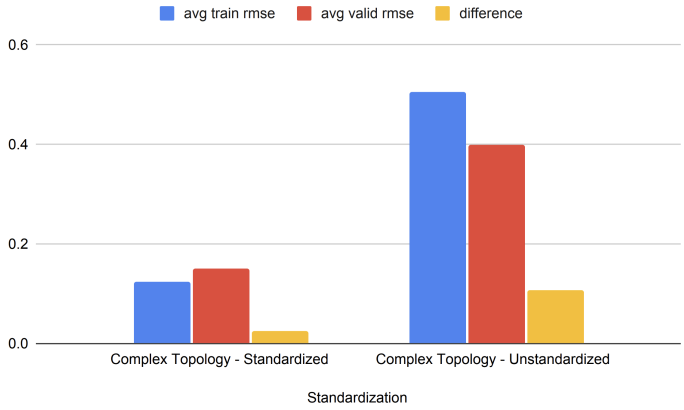
From the previous results, it can be noticed that as the number of epochs or learning rate increases, the training rmse decreases. However, past a certain point, the valid rmse begins to increase, in turn, increasing the difference. Therefore, there is a sweet-spot that provides the best accuracy while maintaining proper generalization.

It can be observed that while increasing the weight decay decreases the model's training accuracy, it also decreases the difference between the training and validation error. This can be viewed as a positive quality, as the model's generalization becomes more robust. Additionally, it was observed that, during low weight decay tests, there existed a high variance in the model's accuracy between training epochs. Alongside this characteristic, increasing the layer count also exhibited similar behaviors.

The dropout rate affected the model's accuracy negatively. As the dropout rate decreased, the error rate would drop as well. This implies that removing the dropout layer increases model performance, with the dropout rate approaching 0.



(a) Standardization comparison in consideration of hyperparameters.



(b) Standardization comparison in consideration of model topology.

Fig. 6: Effects of variable standardization.

The standardization of the continuous numerical features in our tests proved important, as when this procedure was removed, the model accuracy decreased. This behavior is likely due to the fact that, as the model received some input data, if it is non-regularized, it will have no basis on how closely related it is to the norm. As in, it is unsure of how close the value is to the average of that specific feature.

V. Conclusion

In this paper, the exploration of hyperparameters, model topology, and feature standardization was conducted to observe the affect each dynamic has on machine learning inference. The original hypothesis claiming that by tuning the original model's hyperparameters and topology will increase the model's performance only proved correct in regard to the hyperparameter dynamic. However, while this conclusion might imply that this model's topology is near-optimal for the task at hand, it should be noted that a large-scale optimization search in the topology search space provides a possible solution in increasing the model's performance even further.

VI. Future Work

It was concluded that the topologies tested in this experiment showed sub-par performance in comparison to the original model's topology. However, this work explored a

rather shallow pool of possible layers. It is very possible that a network containing far more layers as well as layer complexity will provide better performance. Specifically, this experiment did not humor the idea of including multiple heterogeneous activation functions in the network. An interesting future work would be to include different configurations of the best performing functions in the same network, such as LeakyReLU, Sigmoid, and ReLU.

Another possible section of a future work could be the inclusion of a convolutional layer. It was attempted in this experiment, but all configurations including this layer resulted in a failure. It would prove interesting to include a functional version of a convolutional layer to observe the characteristics it induces on the model's performance.

REFERENCES

[1] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.

[2] "Kaggle house prices - advanced regression techniques," https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview, accessed: 2022-2-21.