Training Deep Neural Networks for Manhattan Property Sale Price Estimation

Tara Lakdawala

1. **Introduction**

It is well known that New York City is home to some of the most expensive properties in the country. The average person can find themselves living a few blocks from the aptly named “Billionaire Row,” and property transactions contribute to a significant portion of the New York economy. It thus becomes meaningful to be able to predict sale prices within the city. These estimates can help inform economic decisions made by the government, predict the average sale price in a neighborhood, and help create indicators of when the housing market is behaving abnormally in a certain area.

To predict a value, we often use regression. Among the most powerful models for performing regression is the neural network. It is able to expand a feature space, extract important indicators, and learn a model based on these parameters. Neural Networks are commonly used to make predictions about the nature of real estate markets. This method has been used in the past for housing price predictions in Hong Kong.[[1]](#footnote-0) This paper found that neural networks are effective in predicting sale prices over a given time series and that features such as number of units, property price index, and rental index are strong indicators of sale price. We find that one of these features— units— is also available to us in our dataset. This gives a good indication that a neural network, given the right architecture, will be able to learn the transformations of the correct features and generalize well.

Furthermore, we will be attempting to tune our model’s hyperparameters to achieve a faster convergence time. It is well known that neural networks generally will learn over time and perform well. Therefore, loss is not a good indicator of the effect of parameter tuning if we can assume that our model can train for any hundred number of epochs. It has been found that measuring convergence time is a far more effective indicator of neural network performance.[[2]](#footnote-1) Based on this information, it thus becomes our objective to train our model on separate architectures to find out at which point our architecture achieves fastest convergence. The model which is capable of doing this will thus become the model we choose to evaluate our test set on.

1. **Description**

The data sourced for this project was provided by the state of New York. The full dataset consists of 84,548 rows tracking property sales in the city of New York from September 2017 to September 2018. The data includes each of the following 20 features:

1. BOROUGH
2. NEIGHBORHOOD
3. BUILDING CLASS CATEGORY
4. TAX CLASS AT PRESENT
5. BLOCK
6. LOT
7. BUILDING CLASS AT PRESENT
8. ADDRESS
9. APARTMENT NUMBER
10. ZIP CODE
11. RESIDENTIAL UNITS
12. COMMERCIAL UNITS
13. TOTAL UNITS
14. LAND SQUARE FEET
15. GROSS SQUARE FEET
16. YEAR BUILT
17. TAX CLASS AT TIME OF SALE
18. BUILDING CLASS AT TIME OF SALE
19. SALE DATE
20. SALE PRICE

Evidently, this dataset is large and unwieldy. To make it usable, I chose to narrow my task down to the borough of Manhattan, or as it is encoded in the dataset, Borough 1. Second, the creators of the dataset indicated that frequently in the dataset property transfers between family members or property transfer for inheritance occurs for a price much lower than market value, frequently $0 or $1. To avoid biasing my data towards these indicators, I decided that I would only be considering property sales that were greater than or equal to $100,000 as this significantly reduces the likelihood that a property was transferred. Next, I noticed that Land Square Feet, Gross Square Feet, and Apartment Number were missing for over 80% of the data. Because of this, I decided to drop the features as they were unlikely to be helpful. This leaves 12,505 rows of data.

Furthermore, Tax Class Before Sale and Tax Class At Time of Sale were inconsistently labelled. When looking at the data it was clear that “Before Sale” indicates subclasses whereas these were dropped “At Time of Sale”. To avoid generating a false indicator of sale price, I chose to drop these features as inconsistently labelled data can undercut the performance of a model.

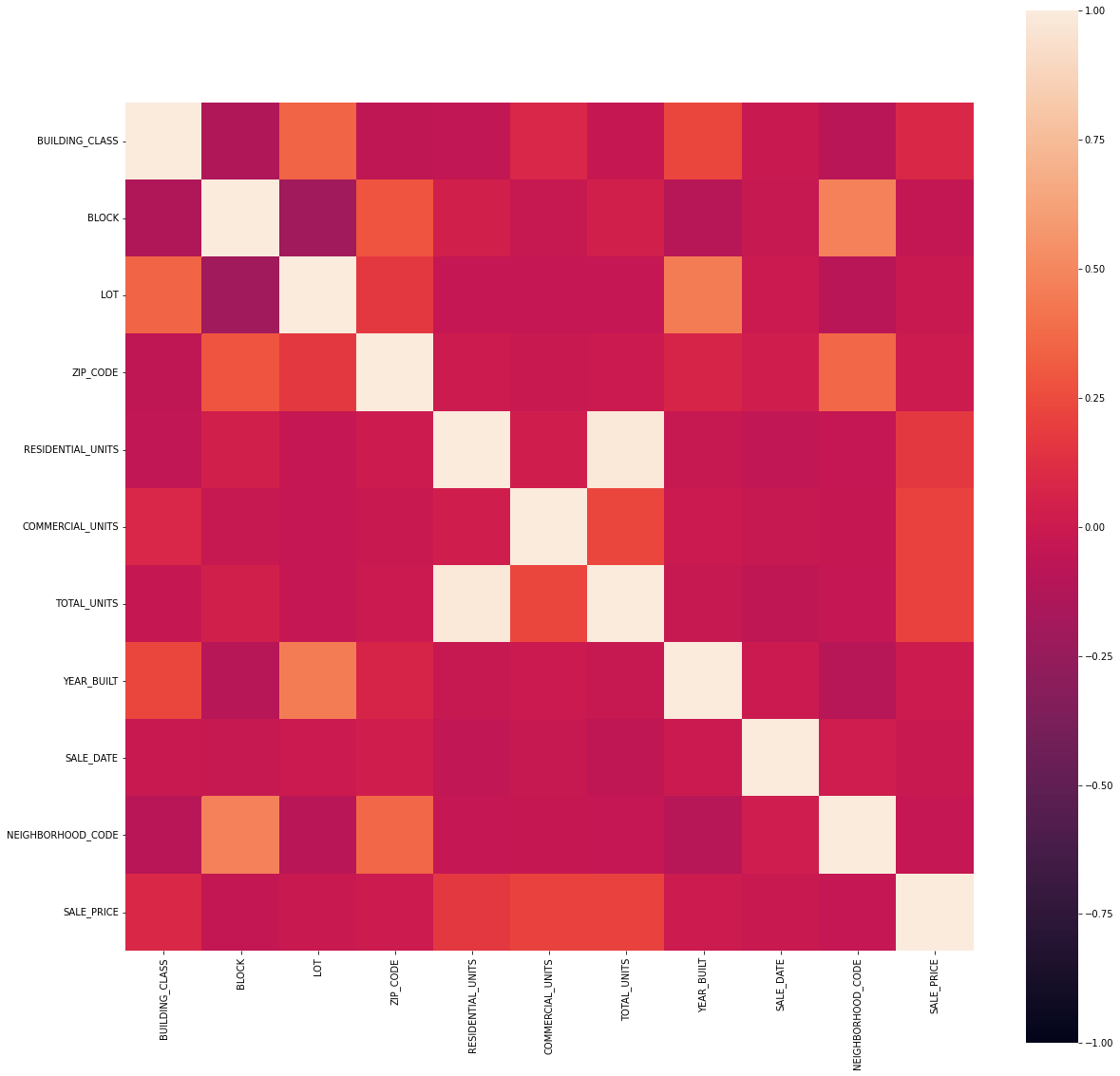
After cleaning my data and dropping poor features, I was left with the following fifteen features:

1. BUILDING\_CLASS
2. BLOCK
3. LOT
4. ZIP\_CODE
5. RESIDENTIAL\_UNITS
6. COMMERCIAL\_UNITS
7. TOTAL\_UNITS
8. YEAR\_BUILT
9. SALE\_DATE
10. NEIGHBORHOOD\_CODE
11. SALE\_PRICE

To begin experimenting, I partitioned my data with an 80/20 split into train and test data. The test data was set aside until all feature engineering and parameter tuning was completed.

Finally, I chose my class value to be Sale Price. This seemed to be the obvious choice for class value as all other features I had available to me are commonly seen as influencers of Sale Price. This feature came with the dataset as one of the main features which is traced by the government when a sale occurs.

Before starting my experiments, I chose to generate a correlation matrix which can be seen below. It gives us an idea of which features are strong indicators of sale price.



*Figure 1. Correlation matrix heatmap between features of the cleaned dataset*

As we can see in the correlation matrix, certain features are clearly categorical. These are not linearly correlated to the sale price in any way. These features include Building Class, Block, Lot, Zip Code, and Neighborhood Code. To appropriately account for this, I performed an encoding of these features to convert the categorical data into binary indicators. Here we see that Residential Units, Commercial Units, and Total Units have the strongest correlation with Sale Price.

Since we are trying to predict a numerical feature, this problem becomes one of regression.

1. **Baseline**

The best baseline for a regression problem is Linear Regression. This was chosen for its strong interpretability. It is easy to understand what the learned parameters mean in this model and it becomes easy to evaluate. Notice that the model chosen for this problem is a deep neural network with relu activation on every layer and linear transformation on the final layer since we are trying to perform regression. With feature engineering and parameter tuning, we can modify our neural network’s architecture to achieve faster convergence. Unfortunately, neural network parameters are not highly interpretable. To create a baseline which makes it easier to understand where to begin with feature engineering, we choose to use linear regression as a substitute. The logic of this choice is that linear regression performs a similar task as the last layer of the neural network by linearly transforming 10 features into one to make a prediction for sale price.

To evaluate our model, we will use Mean Absolute Error Loss. An alternative to this would be to use Mean Squared Error, however, in a dataset such as this one which has outliers in the class value, this would bias our model towards accounting for our outliers more, giving us poor performance. Thus, we will use M5 Linear Regression performed on a 5 fold Cross Validation in Weka.

The baseline model was trained and returned the following output:

**MODEL PARAMETERS:**

SALE\_PRICE =

504027.3928 \* BUILDING CLASS +

-2232.5849 \* BLOCK

-1629.2195 \* LOT +

29205.1621 \* ZIP CODE +

113099.3412 \* RESIDENTIAL UNITS

2084017.0202 \* COMMERCIAL UNITS

-292424696.1307

Mean Absolute Error: 3744315.6301

We see the linear regression model weighs commercial and residential units highly. This makes sense because we saw that there was a high correlation between these features and the sale price. We also see that the building class is weighted highly, giving us an idea that the building class is a strong influencer on the sale price.

The goal now becomes to tune our parameters so that we can help our neural network reach convergence faster. The baseline of our dataset only considers our raw data as input with mean absolute error 3,744,315.63 averaged over the five folds held out in cross validation for the linear regression model.

This error is seemingly high, especially when considered in the context of property value. However, when considered in the context of this specific dataset, we notice that the dataset includes sale prices which range from $100,000 to $2,210,000,000. This range is extremely large, so it is not surprising that we see a mean absolute error in the millions.

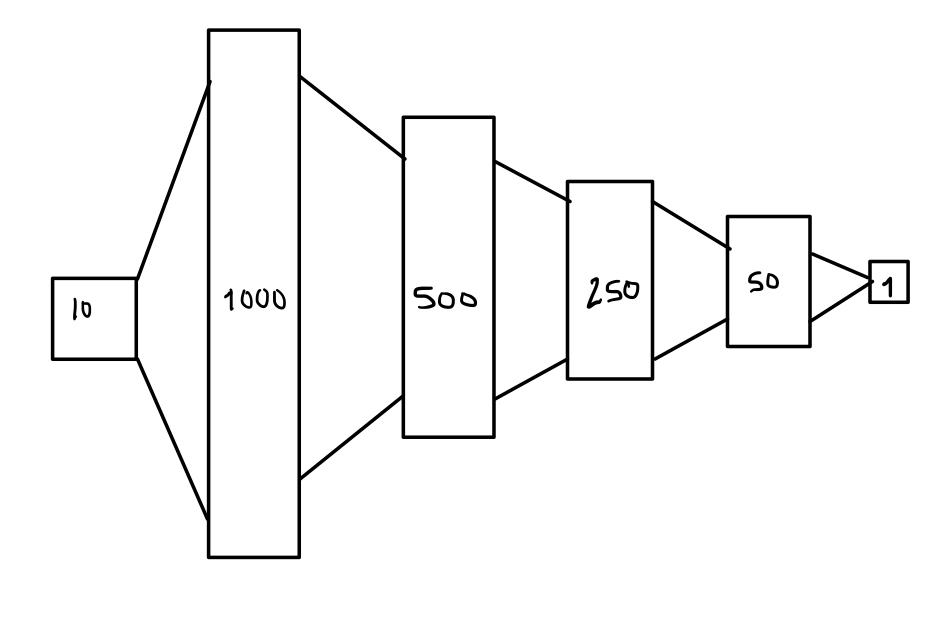
When performing comparison for feature engineering we notice that most frequently block and lot are poor indicators. However, if we restructure our dataset so that it considers these two features together, we hypothesize that we can see an improvement in our performance. We choose to do this to these two features because in New York, buildings are precisely indicated by their block and lot numbers. This can become a strong indicator when considered together. To combine these features ,we perform multilevel modeling where we consider Block and Lot together as Block[Lot] as Lot is a subcategory for Block in New York building codes.

When combining these features, we see some improvement in performance. However, while this is the best feature improvement we can get from combining features, we do not see significant improvement. Thus we find that there isn’t much we can do for our feature engineering aside from our encoding of categorical variables.

1. **Parameter Tuning**

Three different model architectures were experimented with.

**Model 1 is a four layer neural network with the following architecture:**

Total params: 3,194,351

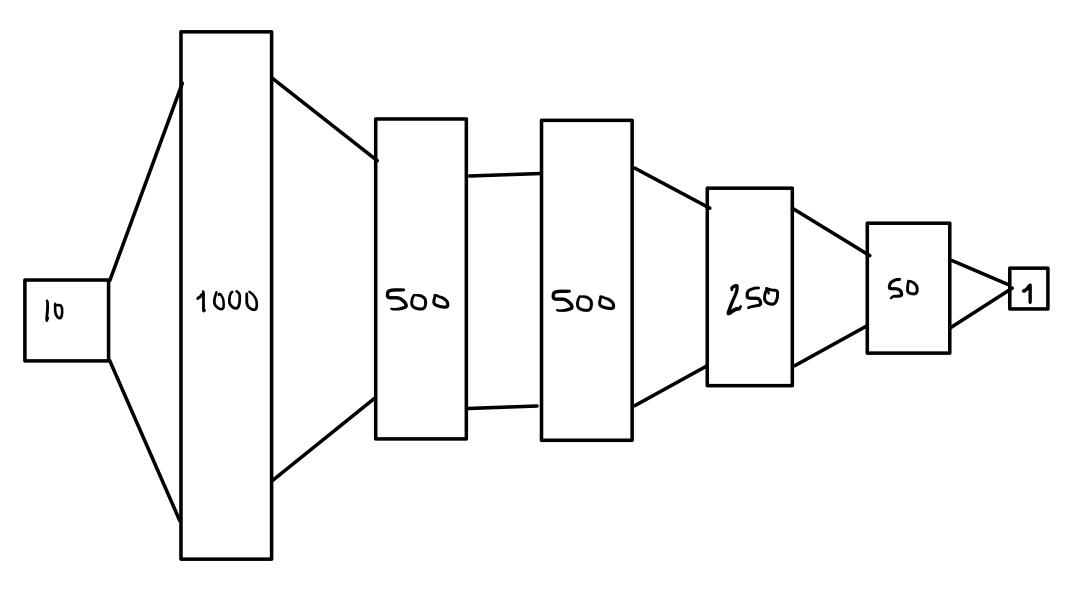
Trainable params: 3,194,351

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Figure 2. Architecture summary of Model 1*

**Model 2 is a five layer neural network with the following architecture:**



Total params: 3,507,601

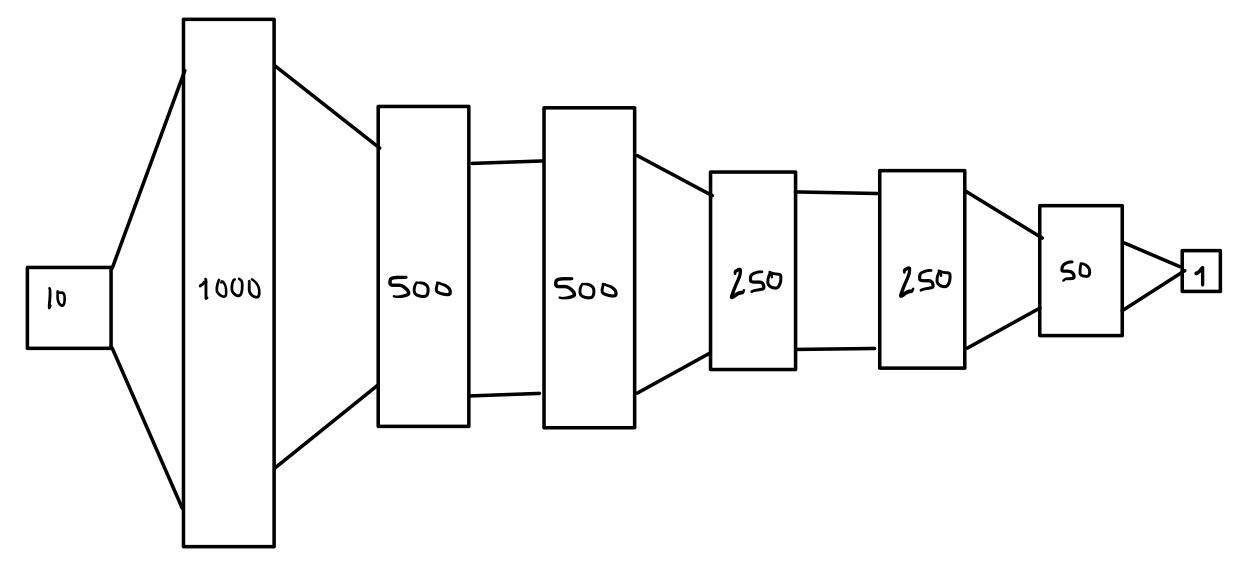
Trainable params: 3,507,601

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Figure 3. Architecture summary of Model 2*

**Model 3 is a six layer neural network with the following architecture:**



Total params: 3,444,851

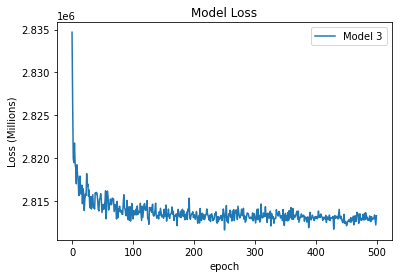
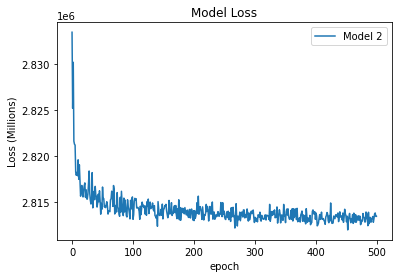
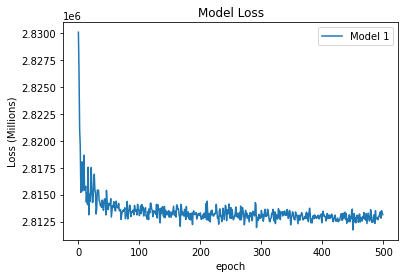
Trainable params: 3,444,851

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Figure 4. Architecture summary of Model 3*

Having established the three architectures we will be using for our hyperparameter tuning, we perform an initial run of all three models across 500 epochs to get an idea of when each model converges.



*Figure 5. Validation loss for 500 epochs across models 1, 2, and 3*

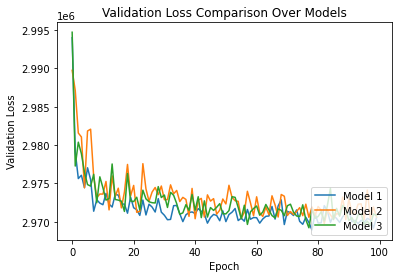
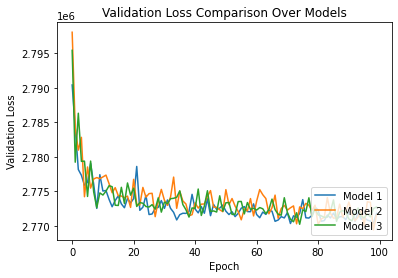
In order to gain a better basis of comparison for model performance, we now proceed to perform a five fold cross validation on the training data for all three models defined above to achieve a basis of comparison. We run each for 100 epochs. This number was chosen after an initial run of all three models showed convergence was achieved before 100 epochs.

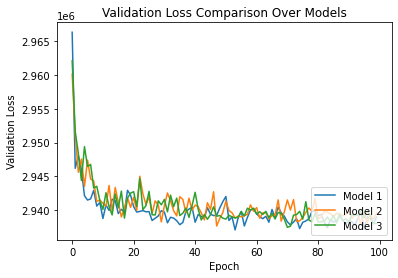
| *Fold/Model* | Model 1 | Model 2 | Model 3 |
| --- | --- | --- | --- |
| Fold 1 | 2771809.7500 | 2771571.7500 | 2772726.0000 |
| Fold 2 | 2970554.2500 | 2970613.7500 | 2970815.7500 |
| Fold 3 | 2938949.2500 | 2939549.0000 | 2937995.0000 |
| Fold 4 | 2444870.0000 | 2444866.0000 | 2444486.2500 |
| Fold 5 | 2943433.2500 | 2944746.0000 | 2944555.5000 |

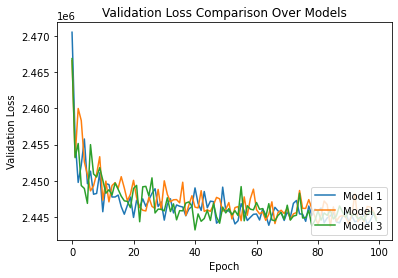
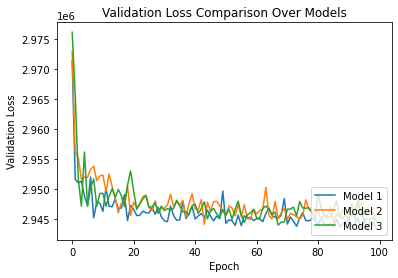
*Table 1. 5-fold cross validation results for models 1-3*

Notice that there is no significant difference in the averaged mean absolute error of the three models across five folds. However, this is expected as a Neural Network is expected to converge after a sufficient number of iterations.

Convergence times are mapped in the following graphs displaying the mean absolute error loss for each fold in the following graphs.







*Figure 6. Validation loss across 5 folds (arranged from 1-5 from left to right, top to bottom)*

Despite training a significantly higher number of parameters in our 5 and 6 layer neural networks, there is not a significant difference (p = 0.9912 between models 1 and 2; p = 0.9951 between models 1 and 2) in the convergence time of the three models. This is a good indicator that our simpler four layer architecture converges quickly. Thus, with fewer features, we can achieve the same results as a more complex model. This is a strong indicator that our four layer architecture will perform well by converging quickly.

1. **Final Evaluation**

In this paper, we found that for regression on sale prices of property in Manhattan, we were able to train a model which has strong performance in comparison to our baseline. In the end, we ran Model 1 after training it for 500 epochs and tested against our test set which was 20% of our initial data held out before trying hyperparameter tuning and feature engineering. When we tested our model on the test set, the resulting mean absolute error was reduced from 3744315.6301 in our baseline to 2819709.7500.

Notice again that our mean absolute error may appear to be high, however our data covers a range of values from $100,000 to $2,210,000,000. Given this range, our mean absolute error indicates strong performance in the presence of outliers. Ultimately we see an improvement over the baseline loss of 3744315.6301 by 24.7%. This performance improvement is statistically significant, indicating that our model improved through our feature engineering and parameter tuning.

1. **Discussion**

Through this project I have learned a lot about the significance of considering multiple factors when evaluating a model’s performance. For example, because we can expect a neural network to converge no matter what, we change our perspective in evaluation and consider if we can help it converge more rapidly. This is a useful tool as it saves significantly on compute time. A quicker train time can give us valuable results faster. In future iterations of model training on this data, I would take more time to analyze the effects of outliers on the performance of the model. Because property value is so vastly varied, I would consider weighting certain features more than others to see if we can use it to create better indicators of property value. Another idea would be to attempt more transformation of the features, for example, scaling or taking the natural log, to help deal with outliers, especially those in the sale price.

Furthermore, I have learned a lot about feature engineering over large datasets. This generally requires much more time and automation (Lightside crashed many times and I had to perform the feature engineering through code). I also learned the importance of cleaning data to avoid including poor indicators in model training.

1. Xin, J. Ge, and G. Runeson. "Modeling property prices using neural network model for Hong Kong." *International Real Estate Review* 7.1 (2004): 121-138. [↑](#footnote-ref-0)
2. Kohler, Michael, and Sophie Langer. "On the rate of convergence of fully connected very deep neural network regression estimates." arXiv preprint arXiv:1908.11133 (2019). [↑](#footnote-ref-1)