



Huber Loss and Neural Networks Application in Property Price Prediction

Alexander I. Iliev^{1,2}(✉), Amruth Anand¹

¹ SRH Berlin University, Charlottenburg, Germany

² Institute of Mathematics and Informatics, Bulgarian Academy
of Sciences, Sofia, Bulgaria
ailiev@berkeley.edu,
amruthanand24@gmail.com

Abstract. In this paper we aim to explore the Real Estate Market in Germany, and particularly we have taken a dataset of Berlin and applied various advanced neural network and optimisation techniques. It's always difficult for people to estimate what price is best for a property and there are various categorical and numerical features involved in it. And the main challenge is to choose the model, loss function and customize the neural network to best fit for the marketplace data. We have developed a project that can be used to predict the property price in Berlin. Firstly, we have worked on procuring the online current market price of the property by web scraping. Then we did intensive exploratory Data Analysis on it, prepared the best data for experiments. Then we build 4 different models and worked on the best loss functions which can suite our model and tabulated the mean squared and mean absolute errors for the same. We have tested our model with the current on the market properties, and the sample results are plotted. This methodology can be applied efficiently, and the results can be used by the people who are interested in investing in real estate in Berlin, Germany.

Keywords: Huber loss • Hyperparameter Tuning • Exploratory Data Analysis • Recurrent Neural Nets • Convolution Neural Nets • Deep Neural Nets.

1. Introduction

Real Estate is a great investment option, investing on a property is always difficult in Germany due to the lack of visibility in the market data. In this paper we aim to build an AI model which will help make decision for the most suitable price for a property in Berlin [1-5]. In the study carried out we hypothesize that Huber loss function suites best for real estate data. And we used different neural network models to test this hypothesis and compared with a standard mean squared error loss function. During these experiments we also considered using different neural networks to compare which suites the best.

In this paper, we covered collecting data and performing exploratory data analysis in Section 2. In Section 3 we discuss about different neural network models we used to perform the experiments and best optimization in order to select them. In Section 4, we present the results of all our finding and discuss about them. In

Section 5, we state our conclusion for the research topic and suggest future scope for improvement.

2. About the Data

We collected the data by scraping the data from open online resources, for this experiment we have checked the **robots.txt** of those websites and gathered the data. After scraping we had 15,177 properties to work on. And after exploratory data analysis we ended up with 12,743 properties. A heatmap of the data is observed in figure 1.

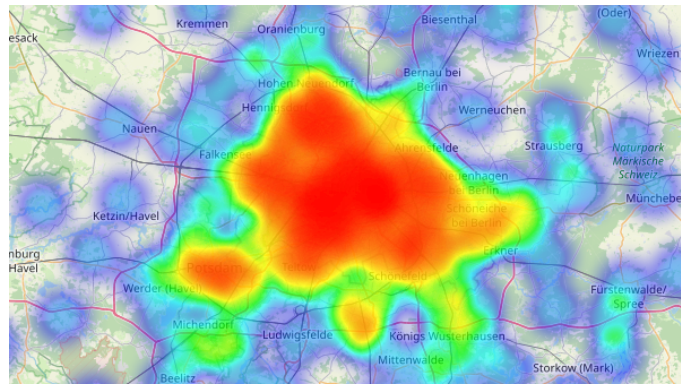


Fig. 1. Heat Map of Dataset

2.1. Data Preparation

Preparation of data plays very important role in every Machine Learning and Artificial Intelligence projects, especially if we are trying to build our own dataset.

Since we gathered information online by web scraping, there was a lot of unstructured data. We did data cleaning in Python to give it a proper structure. The feature for the property we considered are purchase price, living space, number of rooms, floor of the apartment, number of floors in the apartment, number of rooms, construction year, renovation year, list of features in the property, postcode, category to which the property belongs, balcony, garden, terrace, garden, lift, guest toilet, heating type, furnishing quality, garage space, furnishing quality, and address of the property.

Since most of the above-mentioned features were not provided online, we had to exclude them and some of the features were removed because they were correlated as seen in figure 2 [6].

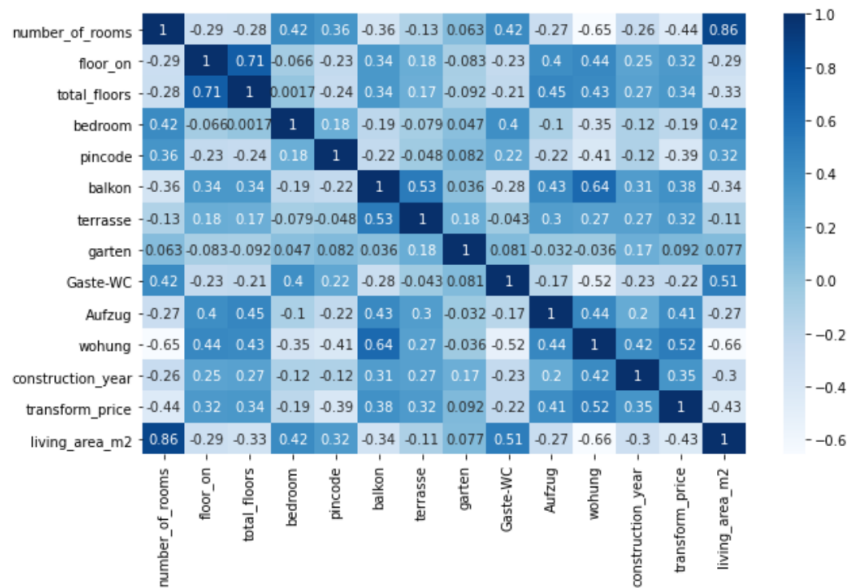


Fig. 2. Heat Map for features correlation

Once after removing the null data, it was important for us to investigate the outliers. First step we did removal of outliers. We used Boxplots to remove the outliers. We considered any point above third quartile by 1.5 times inter quartile range or below first quartile by more than 1.5 times inter quartile range as an outlier.

For some important feature like purchase price, we had a right skewed data, so we used $\log_{1p}$ to make it a normal distribution as seen in figure 3.

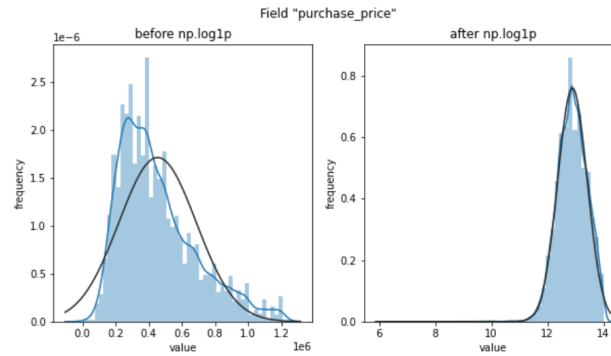


Fig. 3. Normalization

Similarly, we have considered different techniques for different features. And one important thing worth mentioning here is that we need to perform log transform to all the features associated with purchase price.

During EDA, we understood a lot of insights about the real estate market in Berlin.

3. Neural Network Models

We have built 3 different type of neural network models to experiment how the model behaves for the data we have. One was a simple deep neural network, recurrent neural network [7], and convolution neural network. We adjusted these neural nets to work for a regression problem. Like we have in this case.

3.1. Deep Neural Network

Deep neural network is a popular choice for any regression problems, we used a 5-layer neural network model for our 12,743 data, which is shown in figure 4. The reason for 5 layer is because we know the number of hidden layers should be less than twice the size of the input layer. We have 13 features, hence 5 layers.

Initially, we have experimented using 3 layers on our data, the model seems to not learn properly. We also checked out 4 layers, but 5 layers stood out the best.

To select the number of hidden neurons, the thumb rule is it should be between input size layer and output size layer. But to find the best number of hidden layers, it is always a iterative process to do trial and find which is the best.

In our model we have taken, 600, 450, 300, 100, and 50 as several hidden layers with one final layer.

```
tf.random.set_seed(20)
five_layer_model = tf.keras.models.Sequential()
five_layer_model.add(tf.keras.layers.Dense(600, activation='relu', input_dim=X.shape[1:][0]))
five_layer_model.add(tf.keras.layers.Dense(450, activation='relu'))
five_layer_model.add(tf.keras.layers.Dense(300, activation='relu'))
five_layer_model.add(tf.keras.layers.Dense(100, activation='relu'))
five_layer_model.add(tf.keras.layers.Dense(50, activation='relu'))
five_layer_model.add(tf.keras.layers.Dense(1))
```

Fig. 4. Deep Neural Network Model

We have used ReLU activation function, activation functions controls on how well the neural network model learns from the training dataset. As we can see in the figure that we have not used activation function for our output layer, because we need the model to work as a regression problem.

We chose rectified linear activation function (ReLU) because we are using 5 layers and we expect to have vanishing gradients in other activation function. ReLU overcomes these problems and help the model to learn quickly and perform well.

Optimization is a method in neural network which is used to change the weights and learning rate in accordance to reduce the losses, this will also let us get our results much faster [8].

We have used Adam Optimizer [9]; it involves a combination of two gradient descent methodologies:

- Gradient Descent with momentum
- Root Mean Square Propagation (RMSprop)

Adaptive moment Estimation is an algorithm for optimization technique for

gradient descent. This method is efficient when working with large problem involving a lot of data or parameters. It requires less memory and is very effective and efficient.

Cost function or popularly called as loss function plays a very important role in the behavioural of model. It is always important to select the best loss function for our model. For regression problem of ours we had few options and we experimented with 2 of them. One is Mean Squared Error and the other is Huber loss.

Mean Squared Error is a sum of squared distances between our target variables and predicted values.

$$MSE = \frac{\sum_{i=1}^n (y_i - y_i^p)^2}{n}$$

MSE = mean squared error, n = number of data points, y_i = observed value, and y_i^p = predicted value

```
five_layer_model.compile(loss='mse', optimizer='adam',
                        metrics=['mae'])
```

Fig. 5. Loss function MSE

Like we explained above, we have used the same in our code shown in figure 5. Here, to see the performance of the model we have used metrics, MAE. Which is mean absolute error. We could have used any of them. As a result, for this model, we got mean squared error as 0.17 and mean absolute error as 0.27. Since, this is a regression problem, we can't just rely on the metrics. But we also need to see how good the model has learnt itself.

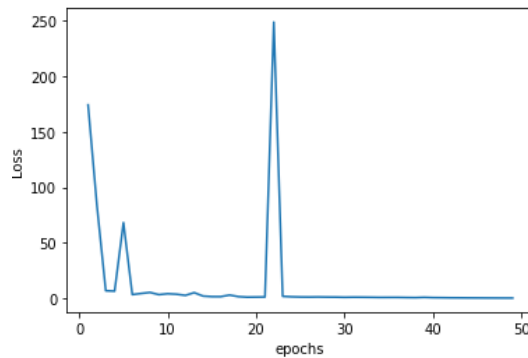


Fig. 6. Loss Curve ANN MSE

We can see in the graph from figure 6 that the model is not learning as expected, and the model is useless. But looking at this we can say that the model is not learning properly, so we decided to change the loss function and try the same model again.

For this experiment we have decided to use the Huber loss function.

$$L_{\delta}(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta, \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{for } |y - f(x)| > \delta, \end{cases}$$

The formula says, if the error is small, we take first condition. Which is error squared divided by 2.

Here we need to define the delta.

To select the delta, we have used hyperparameter tuning, as shown in figure 7, using a function in sklearn GridSearchCV. GridSearchCV will loop through the predefined hyperparameter's and fit the model on our training set.

```
from sklearn.model_selection import GridSearchCV
param_grid = {'epsilon': [1,1.1,1.2,1.25,1.3,1.35,1.4,1.5,1.75,1.95,2,2.25,2.5]}
grid = GridSearchCV(HuberRegressor(),param_grid,refit=True,verbose=2)
grid.fit(X_train,y_train)
print(grid.best_estimator_)
```

Fig. 7. Hyperparameter Tuning for Delta in Huber Loss

With this experiment we found the best delta value for our training data is 1.75.

Then, we used the loss function to our model and training it again as seen in figure 8:

```
five_layer_model.compile(loss=tf.keras.losses.Huber(delta=1.75), optimizer='adam',
                        metrics=['mae'])
```

Fig. 8. Loss function Huber Loss

Using this, we got mean squared error as 0.21 and mean absolute error as 0.31. But most interestingly here we can see that our model has good learning rate per every epoch and there are no noises.

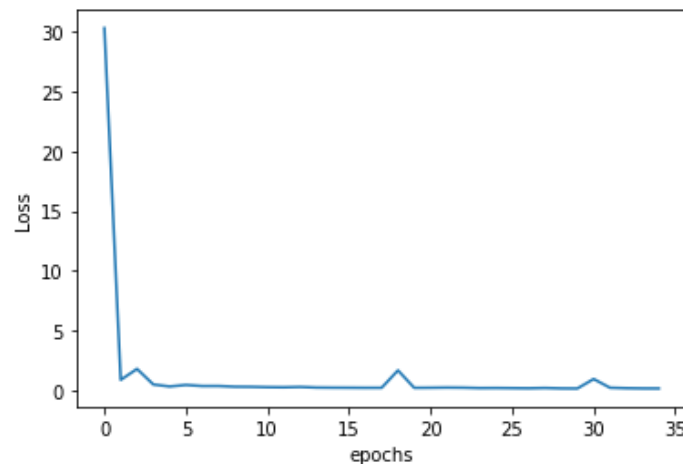


Fig. 9. Loss Curve ANN Huber Loss

As we can see in the graph from figure 9 the model has improved its learning drastically from before. So, using Huber loss was the best option for our training data.

Huber loss is less sensitive to the outliers, and we see that in real estate, some of the data points are subjected as outliers.

In Mean squared error the gradient decreases whenever the loss gets closer to the

minima, which makes it more precise, but in Huber loss whenever the gradient decreases it curves around the minima.

3.2. Recurrent Neural Network

Our model for the Recurrent Neural Network is displayed in figure 10. RNN is also considerably a good choice for house price prediction, as this type of model is used for sequential data.

Problems:

- Inputs, and outputs with has different shape
- Doesn't share features learnt across different position

In RNN, the parameters it uses for each time step are shared, so there will be a set of parameters which we will discuss now, But the parameters governing the connection from $x^{<1>}$ to the hidden layer will be some set of parameters, we will be writing them as W_{ax} and is the same parameters W_{ax} that is used for every time step.

The activations, the horizontal connections will be governed by some set of parameters W_{aa} and for every time step. So, in this Recurrent Neural Network, when making the prediction for $y^{<3>}$, it gets the information not only from $x^{<3>}$ but also the information from $x^{<1>}$ and $x^{<2>}$ because the information on $x^{<1>}$ can pass through this way to help to prediction with $y^{<3>}$.

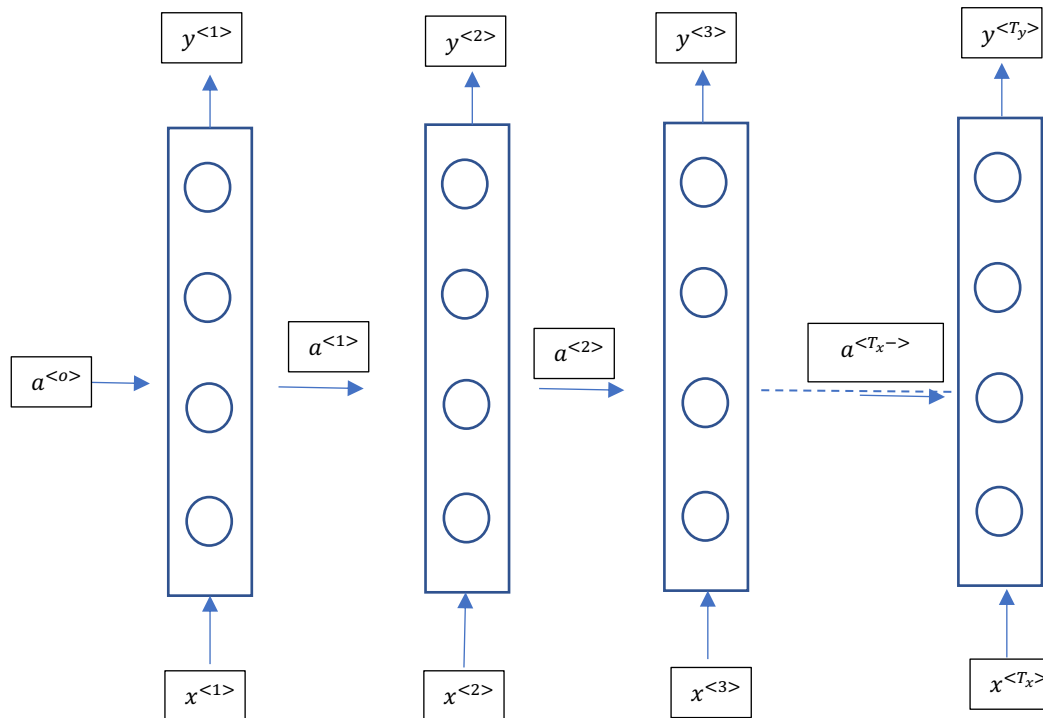


Fig. 10. RNN Forward Propagation

The activations in the choice of RNN are tanh and sometimes ReLU are also used although the tanh is a common choice, and we have other ways of preventing the vanishing gradient problems. Depending on output function 'Y<I>', if binary classification then we should have used sigmoidal function, SoftMax or K-way classification.

Simplified RNN notation.

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

Let's say, $a^{<t>} \rightarrow 100D$, $x^{<t>} \rightarrow 10000D$

And it is stacked horizontally, then

$$W_{aa} \rightarrow (100, 100)$$

$$W_{ax} \rightarrow (100, 10000)$$

$$W_a \rightarrow (100, 10100)$$

$$\text{Similarly, } [a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ \vdots \\ x^{<t>} \end{bmatrix}$$

$$\text{Finally, } [W_{aa}, W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^t \end{bmatrix} = W_{aa}a^{<t-1>} + W_{ax}x^{<t>}$$

Therefore, more generally we end up with this equation,

$$a^{<t>} = g(W_{ax}x^{<t>} + W_{aa}a^{<t-1>}) + b_a \text{ for tanh/relu}$$

$$y^{<t>} = g(W_{aa}a^{<t>} + b_y) \text{ for Sigmoidal}$$

To perform Back Propagation, we need cost/cross entropy

$$L^{<t>}(y_i^{<t>}, y^{<t>}) = -y^{<t>} \log(y_i^{<t>}) - (1 - y^{<t>}) \log(1 - y_i^{<t>}) \text{ This is for an element in the sequence.}$$

Overall Function,

$$L(y_i, y) = \sum_{t=1}^{t_y} L^{<t>}(y_i^t, y^t), \text{ This will be the equation for Back Propagation through time.}$$

Here, we have 4 types of RNN:

1. One-to-One
2. One-to-Many
3. Many-to-One
4. May-to-Many

For our use case, for price prediction we have used Many-to-One.

Our RNN neural network snippet looks like this


```

tf.random.set_seed(20)
model = tf.keras.models.Sequential([
    tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis = -1), input_shape = [None]),
    tf.keras.layers.SimpleRNN(100, return_sequences = True),
    tf.keras.layers.SimpleRNN(80, return_sequences = True),
    tf.keras.layers.SimpleRNN(40),
    tf.keras.layers.Dense(1)
])
model.compile(loss = 'mse', optimizer = 'adam', metrics = ['mae'])

```

Fig. 11. RNN Neural Network

As we can see from the code snippet displayed in figure 11, we have our first layer as a lambda layer. We have used lambda layer to use the arbitrary expression as a layer during our experimentation. These layers are generally used for a sequential model. In our use case we had a simple expression to experiment with. Hence, we used it. With this model we achieved mean squared error as low as 0.089 and mean absolute error as 0.222

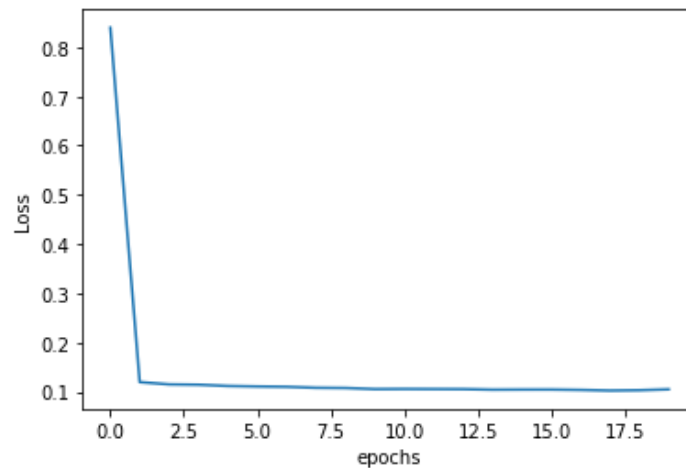


Fig. 12. Lose Curve Simple RNN

This graph in figure 12 shows us how the model learns from every epoch, and we see that the model improved in a very less epochs and got stabilised later.

We have used hyperparameter tuning for the learning rate in our RNN model, this way we can control the speed with which the model learns as evident from figure 13.

```

lr_schedule = tf.keras.callbacks.LearningRateScheduler(
    lambda epochs: 1e-5 * 10**(epochs / 20))
model.compile(loss = 'mse', optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-5), metrics = ['mae'])

```

Fig. 13. RNN Learning rate

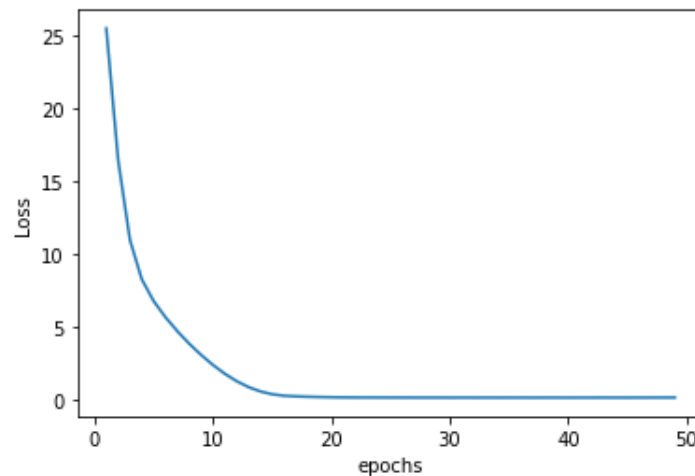


Fig. 14. Loss Curve RNN using Hyperparameter Tuning

After implementing this, we can see that the model is improving by looking at the way it is learning in this graph Fig. 14. When we train a model, we expect a smooth bell-shaped curve which we achieved with this experiment.

3.3. Hybrid Neural Network

In our last neural network, we have worked on the hybrid neural network [10, 11] with the combination of 1 lambda layer, 1 conv1d, 2 LSTM layers, 2 Simple RNN layers, 3 dense layers, and 1 final output dense layer, as shown in figure 15.

We wanted to experiment with the hybrid model and see the performance of it, with our data and compare how it will work compared to others. In the hybrid model we have also used hyperparameter for the learning rate for Adam optimizer.

```
tf.random.set_seed(68)
model = tf.keras.models.Sequential([
    tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis = -1), input_shape = [None]),
    tf.keras.layers.Conv1D(filters = 128, kernel_size = 5, strides = 1, padding = 'same', activation = 'relu'),
    #tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis = -1)),
    tf.keras.layers.LSTM(128, return_sequences = True),
    tf.keras.layers.LSTM(64, return_sequences = True),
    tf.keras.layers.SimpleRNN(80, return_sequences = True),
    tf.keras.layers.SimpleRNN(40),
    tf.keras.layers.Dense(100, activation = 'relu'),
    tf.keras.layers.Dense(50, activation = 'relu'),
    tf.keras.layers.Dense(20, activation = 'relu'),
    tf.keras.layers.Dense(1)])
lr_schedule = tf.keras.callbacks.LearningRateScheduler(
    lambda epochs: 1e-5 * 10**(epochs / 20))
model.compile(loss = 'mse', optimizer = 'adam', metrics = ['mae'])
```

Fig. 15. Hybrid Neural Network

Most of the layers which are been used here are being explained in the above sections, 3.1 and 3.2.

The metrics we have used for calculation the loss for all the above models were mean squared and mean absolute errors, and the values we have got from this neural net was 0.12 and 0.26.

The loss curve for this hybrid model was also impressive as shown in figure 16, as we see there is no noise in its trends.

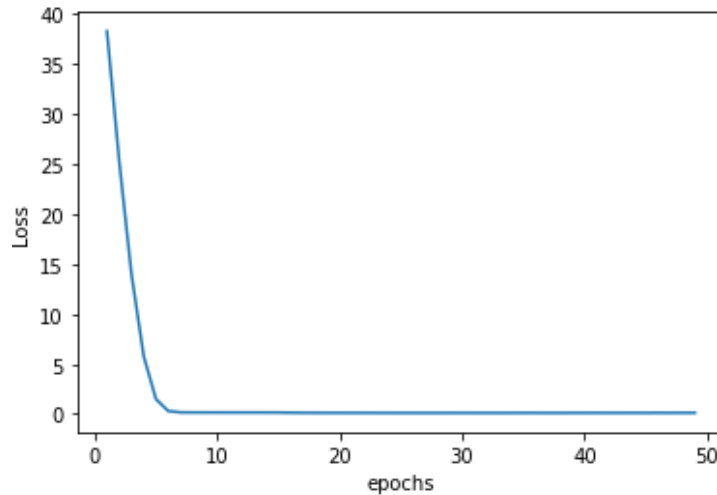


Fig. 16. Loss Curve Hybrid Neural Network

4. Results

The main objective of this paper was to find the best neural network for the real estate market in Berlin, Germany.

We have worked on multiple models and tried to improvise the models using different loss functions, figuring the best delta value for Huber loss function, using hyperparameter tuning for Adam optimizer. In this section, we have discussed and tabulated the results and the tests we have done. We have taken the properties online in the market and tried to compare the models on how they are efficient on the real-world data. We have considered doing various pre-processing steps to make the data suitable for the models to learn from and we have explained them briefly in section 2 of the paper. Different models and their respective errors are shown in Table 1:

Table 1. Metrics comparison for models

Model	Mean Squared Error	Mean Absolute Error
Deep Neural Nets with MSE loss function	0.177	0.271
Deep Neural Nets with Huber Loss function	0.215	0.3163
Simple RNN	0.102	0.235
Simple RNN with hyperparameter tuning Adam Optimizer	0.106	0.241
Hybrid Neural Network	0.124	0.268

We have also discussed about the loss curve of these models in section 3, to understand how the models are learning. A comparison of the test data for each model is shown in Table2.

Table 2. Test Data Comparison

Test Data	Model 1	Model 2	Model 3	Model 4	Model 5
8.545	8.584	8.782	8.598	8.530	8.613
8.370	8.283	8.475	8.398	8.261	8.434
8.694	8.577	8.632	8.408	8.253	8.412
7.717	7.777	7.390	7.866	7.861	7.909
7.740	7.651	7.719	7.783	7.664	7.743

NOTE: These are the log values directly from the model which has normalized log values.

Here, in this table

- Model 1 = Deep Neural Nets with MSE loss function
- Model 2 = Deep Neural Nets with Huber Loss function
- Model 3 = Simple RNN
- Model 4 = Simple RNN with hyperparameter tuning Adam Optimizer
- Model 5 = Hybrid Neural Network

The number in the tables are the property price per square feet. And the values are the log values.

These results are straight from the model on the test data, we have divided our data into 80% train and 20% validation. And in the pre-processing, we have normalised the data to get the better performance out of the model. And we have tabulated the log values directly from the model.

In this section we take the real time data from online and try to find which model works best on them.

These are the real properties which we used to test our model.

	no_of_rooms	floor_on	total_floor	bedroom	zipcode	balcony	terrace	garten	guest_toilette	Lift	wohnung	year	living_sqm
0	3.0	3	5	2	12489	1	0	0	0	1	1	2022	4.184947
1	3.0	4	5	2	12489	0	0	0	0	1	1	2022	4.184947
2	3.0	0	3	2	12527	0	0	0	0	0	1	1905	4.219508
3	2.0	2	0	1	12489	1	0	0	0	0	1	1900	4.217741
4	3.0	3	0	2	12459	0	0	0	0	0	1	1910	4.501364
5	5.0	2	4	4	10437	1	0	0	0	1	1	2003	4.897840
6	2.0	1	3	1	13189	1	0	0	0	0	1	1923	4.154028
7	1.5	3	6	1	10435	1	0	0	0	0	1	1886	3.700067
8	2.0	0	0	1	10119	0	0	0	0	1	1	2013	4.213756
9	2.0	2	0	0	12489	1	0	0	0	0	1	1900	4.217741
10	3.0	4	0	1	10437	1	0	0	0	1	1	1900	4.779123
11	3.0	3	5	2	10439	1	0	0	0	0	1	1908	4.590767
12	3.0	5	5	2	10439	1	0	0	0	0	1	1908	4.538496
13	3.0	1	0	1	10439	1	0	0	0	0	1	1900	4.350278
14	3.0	4	5	1	14163	1	0	0	0	1	1	2009	4.529584

Fig. 17. Online Real Time Property Data

Note: In figure 17 we have taken the log values for living_sqm field because our

model is trained with log values.

To do that, we just need to use `log1p` from the NumPy library.

Here, we will compare two ANN models, 2 RNN models and Hybrid models separately just to observe how the changes are behaving.

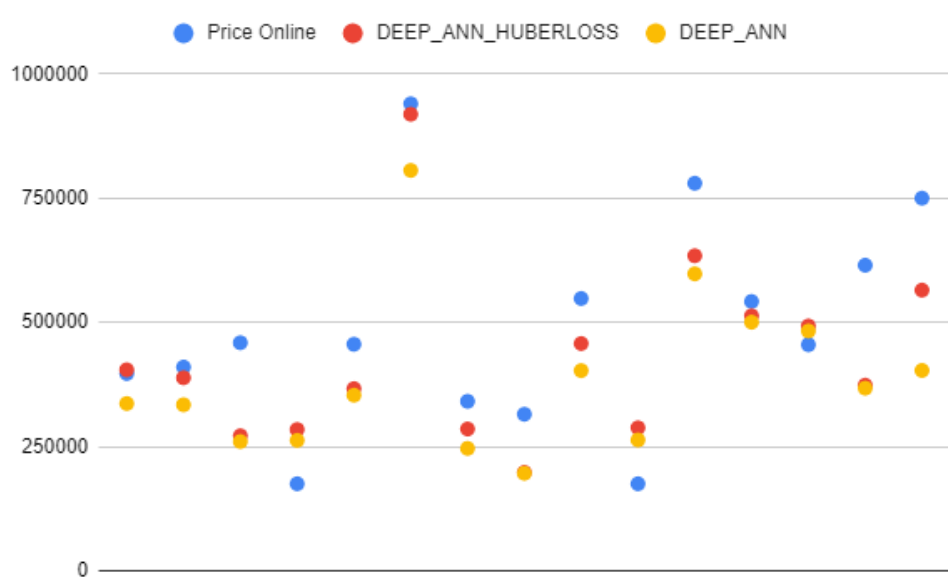


Fig. 18. Comparison of ANN Models

In the graph figure 18, we have plotted the online price, deep Ann model price, and a model with Huber loss.

It is evident in the plot that Deep ANN model performs a lot better than the Deep Neural Network Model.

As we have explained in the section 3.1 that Huber loss is more robust to the outliers than the usual MSE loss function. We can see for the property 9 and 15 that the online prices are high, and the model performs well for the outliers, which here means that the property is highly priced than usual.

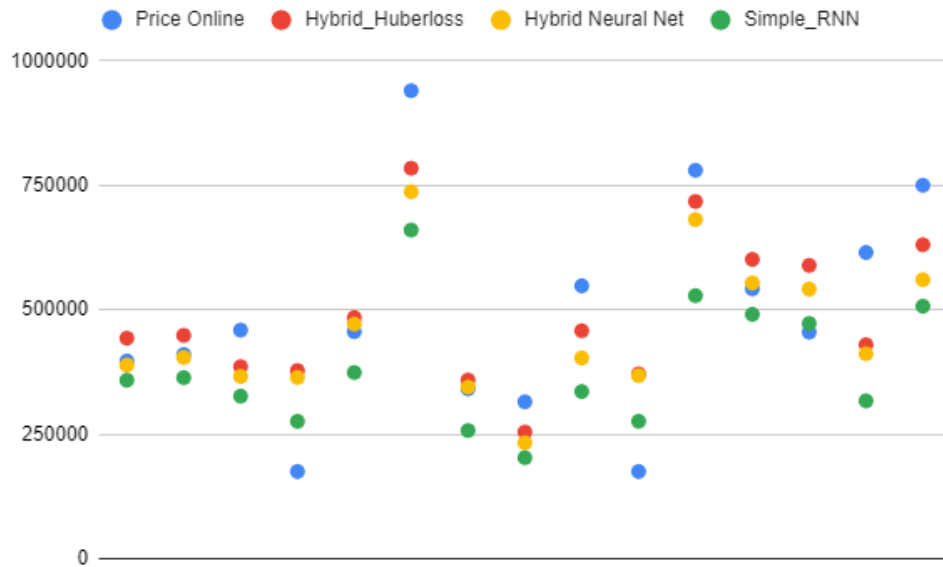


Fig. 19. Comparison of RNN and Hybrid Neural Network

This graph in figure 19 is a comparison of RNN and Hybrid model. In this figure, we can clearly observe that the hybrid model with Huber loss is more accurate compared to the conventional neural networks.

5. Conclusion

The main object of this paper was to experiment on neural nets, optimizers, and loss functions to see which suite the best for property price prediction regression problem statement. And we have considered all the combination and discussed the results in Section 5. From all the experiments carried out on this topic, we can conclude that

- hyperparameter tuning the learning rate of Adam optimizer
- Using the Huber loss function with delta value 1.75
- Using Hybrid Neural Network

Is this best combination for this use case.

References

1. Linda Kauskale (2017) Integrated Approach of Real Estate Market Analysis in Sustainable Development Context for Decision Making Elsevier Procedia Engineering pp. 505-512
2. Shashi Bhushan Jha (22 Aug 2020) Machine Learning Approaches to Real Estate Market Prediction Problem: A Case Study arXiv:2008.09922
3. Nunez Tabales, Julia M. (2013) Artificial Neural Networks for Predicting Real Estate Prices Revista De Metodos Cuantitativos para la economia y la empresa pp. 29-44

4. Hamzaoui, Y.E. and Hernandez (2011) Application of Artificial Neural Networks to predict the selling Price in the real estate valuation 10th Mexican International Conference on Artificial Intelligence pp. 175-181
5. Kauko, T., Hooimajer, P., and Hakfoort, J. (2002) Capturing housing market segmentation: An alternative approach based on neural network modelling Housing Studies, 17(6) pp. 875-894
6. Kenji Fukumizu (May 2017) Influence Function and Robust Variant of Kernel Canonical Correlation Analysis Neurocomputing pp. 304-307
7. Iman Rahimi, Reza Bahmanesh (January 2012) Using Combination of Optimized Recurrent Neural Network with Design of Experiments and Regression for Control Chart Forecasting International Journal of Science and Engineering Investigations vol. 1, issue 1 pp. 24-28
8. D.F. COOK, C.T. RAGSDALE, R.L. MAJOR (2000) Combining a neural network with a genetic algorithm for process parameter optimization, Eng.Appl.Artif. Intell. 13, pp. 391-396.
9. Diederik P.Kingma (2015) Adam: A method for stochastic optimization International Conference on Learning Representations
10. E.W.M. Lee (May 2004) A Hybrid Neural Network Model for Noisy Data Regression IEEE transaction on sybernetics pp. 951-960
11. J. R. Williamson (1996) Gaussian artmap: A neural network for fast incremental learning of noisy multidimensional maps, Neural Netw., vol. 9, no. 5, pp. 881-897