Development of AI for Predicting Real Estate Prices (2025)

Bilal Alali and Andrej Sum-Shik
Frankfurt University of Applied Sciences

***Abstract*—This paper presents the development and evaluation of several artificial intelligence models for predicting real estate prices. In this study, three different approaches—linear regression, polynomial regression, and neural networks—were implemented and compared. We describe the challenges encountered during data collection, model implementation, and evaluation under conditions of limited data. Results indicate that the neural network model, despite its complexity, outperforms the regression-based approaches in capturing nonlinear relationships between property features and price. Future work is proposed to expand the dataset and further optimize the neural network architecture.**

***Index Terms*—Artificial Intelligence, Real Estate Price Prediction, Neural Networks, Regression Models, Data Analysis.**

I. Introduction

Real estate is a critical market for both individuals and businesses, with prices influenced by factors such as supply, demand, location, and property condition. In this work, we explore the use of artificial intelligence (AI) to predict real estate prices. Originating from our "Angewandte KI" seminar project, this research provided an opportunity to apply and compare multiple AI models. The project also served as a practical exercise to bridge theoretical machine learning concepts with real-world data challenges. In particular, we aimed to investigate how different modeling techniques—ranging from simpler regression methods to more complex neural networks—could capture intricate relationships among various property features. This introductory section also sets the stage for a detailed discussion of the challenges and practical considerations involved in deploying AI for price prediction. Furthermore, the remainder of this paper elaborates on the methodologies and experimental results, ensuring clarity and reproducibility of our approach.

II. Challenges in Developing the AI

A. Understanding and Implementing the Models

Developing an effective AI model required a thorough understanding of several approaches. We implemented three models:

Linear Regression: As the simplest model, it helped us understand the basic relationship between living space and price. This model assumes a linear relationship between the input parameters and the output parameter. The coefficient of determination was used to evaluate how well the parameters align. Although its assumptions are relatively simple, linear regression remains a cornerstone in understanding data trends. It allowed us to establish a baseline from which the improvements of more complex models could be measured.

Polynomial Regression: To explore more detailed relationships between features, polynomial regression was used. Instead of a simple linear model (i.e., $w_1 p_1 + w_2 p_2 + \cdots =$ output), a higher-degree model was considered (e.g., $w_1 p_1 + w_2 p_2^2 + w_3 p_3^3 + \cdots =$ output) to capture non-linear relationships. This method adds flexibility to model curvature in the data. However, higher-degree models can overfit the training data by capturing noise rather than meaningful patterns. Thus, careful tuning and validation were necessary to avoid these pitfalls, ensuring that the model generalized well to unseen data.

Neural Network: A neural network was developed for deeper modeling of complex relationships. This model learns the relationship between input parameters and the output through backpropagation and network architecture design. Despite being the most complex and requiring more data, it is best for capturing highly non-linear interactions. The ability of neural networks to automatically learn feature hierarchies makes them particularly well-suited for problems where input variables interact in non-trivial ways. Their flexibility, however, comes at the cost of increased computational requirements and a higher risk of overfitting, particularly when data is scarce.

Overall, a methodical approach to understanding these models ensured that each step was carefully validated.

B. Data Collection and Quality

One of the main challenges was obtaining a sufficiently large and high-quality dataset. Due to the unavailability of open data sources for German real estate prices, data were scraped from online platforms and stored in JSON format. The dataset incorporated features such as living space, number of rooms, floor level, price, and an AI-evaluated "location score" (used only for properties in Frankfurt). Occasional null values were replaced by the averaged value of the corresponding parameter.

The data collection process was time-intensive and required careful curation. We conducted multiple rounds of cleaning and normalization to ensure that the dataset was as reliable as possible. Each data point was scrutinized for consistency in units and realistic ranges. The limitations imposed by the dataset size meant that each example was critical in training robust models, emphasizing the need for rigorous pre-processing and validation procedures. Overall, the data

collection process underscored the importance of rigorous preprocessing and consistency checks.

III. Comparison and Selection of Models

After implementing the three models, their performance was compared under consistent training and evaluation conditions:

Linear Regression: Easy to implement and effective for simple relationships (e.g., between living space and price). However, it could not handle complex, non-linear relationships. The coefficient of determination was only around 63% and the mean absolute error (MAE) was approximately 249,000. Nevertheless, this model provided an important reference that underscored the limitations of linear assumptions in complex real-world scenarios.

Polynomial Regression: This model better captured complex relationships, yielding a slight improvement with a coefficient of determination of around 66% and a MAE of roughly 247,000. Polynomial regression showed that introducing non-linearity could improve performance, although the benefits plateaued as the model complexity increased. It served as a bridge between the simplicity of linear models and the complexity of neural networks, demonstrating that even small non-linear adjustments could have a measurable impact.

Neural Network: The neural network delivered the best results, particularly in modeling complex interactions between features such as location, room count, and price. Its flexibility allowed for better data analysis, though it required significantly more computational power and training data. The reported MAE ranged between 120,000 and 200,000. These results underline the neural network's strength in capturing hidden patterns and interactions that simpler models could not, highlighting the potential of deep learning in real estate valuation.

This comparative analysis not only validated our initial assumptions but also provided valuable insights for further refinements.

IV. Dataset Description

The dataset comprised 102 data points. In some instances, missing values—particularly for the "Etage" (floor level) parameter—were replaced by the corresponding average. For evaluation purposes, the dataset was split (for example, using data points 1–88 for training and 89–102 for evaluation). Although limited in size, the dataset provided valuable insights into model performance under constrained conditions.

In total, the features included:

- **Living Space (in m^2):** A direct numeric value describing property size.
- **Number of Rooms:** An integer that significantly impacts property valuation.
- **Floor Level (Etage):** Important for properties in multi-story buildings, often influencing price.
- **Price:** The final numeric target for prediction.
- **Location Score:** An AI-generated score reflecting local amenities, public transport, and overall desirability.

Although the dataset is limited in size, it has served as a critical foundation for testing and evaluating the models.

V. Evaluation Metrics

Evaluation was primarily based on the mean absolute error (MAE), which measures the average absolute difference between predicted and actual prices. In addition, the coefficient of determination (R^2) was used to assess the proportion of variance explained by the model. MAE, being less sensitive to outliers than the mean squared error (MSE), proved particularly useful given the occasional presence of extremely high or low values in the dataset. This section also discusses the trade-offs between using MAE and MSE, with MAE providing a more intuitive measure of average error that better aligns with practical expectations in price prediction. These metrics were selected to offer a balanced perspective on both the accuracy and robustness of the models.

VI. Neural Network Architecture

Our neural network model consists of four layers:

- **Input Layer:** Four neurons corresponding to the features—number of rooms, living space, location score, and floor level.
- **Hidden Layers:** Two hidden layers, each with six neurons and using ReLU activation functions, which help introduce non-linearity into the model.
- **Output Layer:** A single neuron with a sigmoid activation function to scale outputs between 0 and 1. During testing, predicted outputs were rescaled by the maximum price value to obtain absolute price predictions.

This structure was chosen after careful consideration of the dataset's size and complexity. The ReLU activation function was particularly beneficial presenting non-linearity, ensuring that the network learned efficiently. This architecture was chosen to balance computational efficiency with the capability to model complex data patterns.

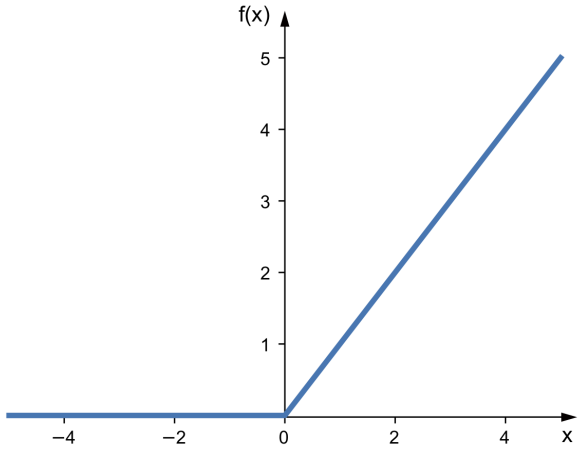

Fig. 1. The non-linear RELU function

Additional aspects include using the Mean Squared Error (MSE) as the default loss function in early experiments and scaling all prices between 0 and 1 to facilitate smooth gradient descent. A detailed analysis of the architecture reveals that even minor adjustments in the network layers could lead to significant changes in prediction accuracy.

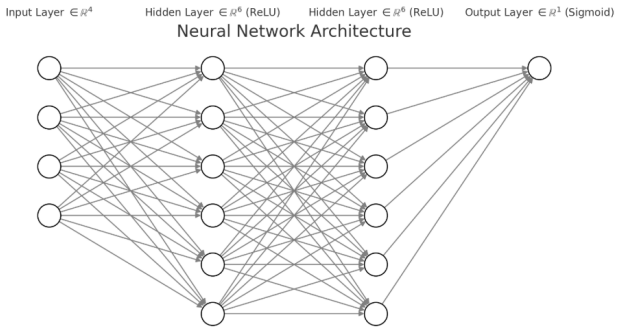

Fig. 2. Neural Network Architecture

VII. Implementation with TensorFlow

To further optimize the neural network, TensorFlow was employed. Key modifications included:

- **Optimizer:** The use of the "adam" optimizer instead of standard gradient descent, which provided adaptive learning rates and faster convergence, due to Adam using the average gradient instead of the "latest" one. The Adam optimizer follows these equations:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
(First moment estimate - moving average of gradients) (1)

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
(Second moment estimate - moving average of squared gradients) (2)

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
(Bias-corrected first moment estimate) (3)

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
(Bias-corrected second moment estimate) (4)

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$
(Parameter update rule) (5)

where g_t is the gradient at time t, β_1 and β_2 are exponential decay rates (typically 0.9 and 0.999, respectively), α is the learning rate, and ϵ is a small constant to prevent division by zero.
- **Dropout:** Implementation of dropout (0.2%) in both hidden layers to prevent overfitting by randomly deactivating neurons during training.
- **Minibatch Processing:** A batch size of 8 was used to enhance training efficiency and stability, balancing the trade-off between noisy gradient estimates and computational demands.
- **Feature Normalization:** Each feature was normalized by its mean and standard deviation to ensure that all input variables contributed proportionately to the learning process.

Each modification was carefully integrated to enhance model performance without compromising the overall structure.

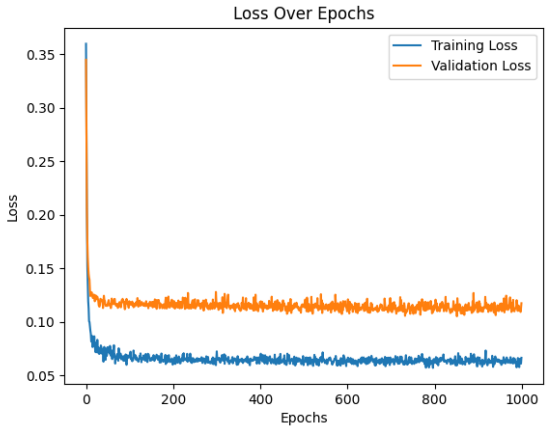

Fig. 3. Loss over Epoch Comparison: Training vs. Evaluation

These modifications were validated through a series of experiments in which both training and validation losses were monitored closely. The iterative process of adjusting the learning rate and dropout rate helped us arrive at an optimal configuration that minimized overfitting while ensuring effective learning.

VIII. Hyperparameter Tuning and Practical Observations

During the training process, numerous hyperparameters were experimented with to maximize performance:

- **Learning Rate:** We tested values ranging from 10^{-4} to 10^{-2}. A higher learning rate led to rapid convergence but also risked overshooting minima, while a lower learning rate required more epochs but yielded a smoother loss curve.
- **Number of Epochs:** Training for 100 to 500 epochs revealed that while additional epochs sometimes offered marginal improvements, they could also lead to overfitting, especially when the model began memorizing noise in the training data.
- **Network Depth and Width:** Although the final architecture utilized two hidden layers with six neurons each, additional experiments with three layers and varying neuron counts were conducted. These tests underscored that a deeper network, more or less neurons did not necessarily equate to better performance given the dataset size.

These observations contributed to a deeper understanding of the model's behavior during training.

IX. Discussion

In addition to the aforementioned analysis, further observations were made during the experimental phase that provide deeper insights into our model's performance. We

observed that the training dynamics varied considerably between the simpler regression models and the neural network. For instance, while the linear and polynomial regression models quickly converged to stable solutions, their performance plateaued at a relatively high error level. In contrast, the neural network required a longer training period and careful tuning to prevent overfitting, yet ultimately demonstrated a greater capacity to capture the complex, non-linear patterns present in the data.

Moreover, the neural network showed a marked sensitivity to hyperparameter adjustments. Small changes in learning rate or dropout rate often resulted in significant fluctuations in validation error, emphasizing the need for a systematic tuning process. We also noted that the choice of training/evaluation split played a crucial role in model robustness; the neural network maintained a more balanced error distribution compared to the biases observed in the regression models. These observations reinforce the importance of domain-specific feature engineering and further suggest that augmenting the dataset, or employing transfer learning techniques, could be beneficial in future work. Overall, the experimental findings reinforce the importance of iterative testing and validation in model development.

X. Loss Function Comparison

Although MSE is differentiable everywhere—a key advantage for training—the squaring of errors excessively penalizes outliers. MAE, on the other hand, treats all errors equally, often resulting in more consistent performance for datasets with significant variance. In scenarios where extremely high-priced properties could skew the average loss, MAE proved more robust and aligned better with real-world expectations.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \qquad MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Fig. 4. MAE (left) and MSE (right) formulas.

$$\frac{\partial MAE}{\partial \hat{y}_i} = \begin{cases} \frac{1}{n}, & \text{if } \hat{y}_i < y_i \\ -\frac{1}{n}, & \text{if } \hat{y}_i > y_i \\ \text{undefined}, & \text{if } \hat{y}_i = y_i \end{cases} \qquad \frac{\partial MSE}{\partial \hat{y}_i} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)$$

Fig. 5. MAE derivative (left) and MSE derivative (right) formulas.

The careful comparison of loss functions further highlighted the trade-offs inherent in each approach.

XI. Effect of Location Parameter

An additional experiment replaced the AI-evaluated "location score" with hashed location names to test the effect on model performance. Despite an increase in dataset size by over 50 new data points, the model's accuracy decreased. This outcome is attributed to the more complex, non-linear relationship between raw location data and price, emphasizing that the simplified "location score" provided a more effective linear correlation in this context.

Furthermore, hashed location names fail to capture geographical or socio-economic similarities between neighboring areas. This finding underscores the importance of carefully engineered features in achieving reliable predictions. In future iterations, incorporating more nuanced location data, such as proximity to essential services, might further improve model performance. This experiment clearly illustrates the significance of feature engineering in achieving accurate predictions.

XII. Conclusion and Future Work

This study compared three modeling approaches for predicting real estate prices. Although the linear and polynomial regression models provided valuable baseline insights into feature relationships, the neural network outperformed them by capturing complex non-linear interactions. The challenges encountered—particularly regarding data quality and the limited dataset—highlight the need for larger, more diverse datasets.

Future work will focus on:

- **Data Expansion:** Acquiring a broader range of property listings from multiple cities and sources to improve model generalization.
- **Refined Network Architectures:** Experimenting with deeper networks, alternative activation functions (e.g., Leaky ReLU), and enhanced regularization strategies.
- **Alternative Models:** Investigating ensemble methods (e.g., Random Forests) and kernel-based approaches (e.g., Support Vector Machines) to further improve prediction accuracy.
- **Enhanced Feature Engineering:** Incorporating additional variables such as proximity to schools, crime rates, and economic indicators to capture more aspects of real estate valuation.

In summary, the study highlights the value of combining traditional and modern AI techniques to address complex real estate pricing challenges.

References

[1] YouTube, "Video on AI Model Implementation," Available: https://youtu.be/Ilg3gGewQ5U?si=FnKcMWslfMrANdij.

[2] YouTube, "Understanding TensorFlow Optimizers," Available: https://youtu.be/w8yWXqWQYmU?si=SZJkRL8EmakgBzfT.

[3] YouTube, "Backpropagation and Neural Networks," Available: https://youtu.be/IHZwWFHWa-w?si=Yib9aRHqXMZj8qNR.

[4] Immowelt, "Real Estate Data Platform," Available: https://www.immowelt.de.

[5] GeeksforGeeks, "ML Linear Regression – Types and Applications," Available: https://www.geeksforgeeks.org/ml-linear-regression/#types-of-linear-regression.

Appendix A
Neural Network for Price Estimation

Parameters: 16 biases and 66 weights (total: 72 variables)

Forward Propagation

$$\begin{aligned} X &= [x^{(1)}, x^{(2)}, \ldots, x^{(100)}]^T, \\ Z^{(1)} &= W^{(1)}A^{(0)} + b^{(1)}, \quad A^{(0)} \equiv X, \\ A^{(1)} &= \mathrm{ReLU}(Z^{(1)}), \\ Z^{(2)} &= W^{(2)}A^{(1)} + b^{(2)}, \quad A^{(2)} = \mathrm{ReLU}(Z^{(2)}), \\ Z^{(3)} &= W^{(3)}A^{(2)} + b^{(3)}, \quad A^{(3)} = \mathrm{softmax}(Z^{(3)}). \end{aligned}$$

Backpropagation

Gradient at Output Layer:
$$dZ^{(3)} = 2\left(A^{(3)} - y\right) \cdot \delta'(Z^{(3)}),$$
$$dW^{(3)} = \frac{1}{100}\, dZ^{(3)}\, \left(A^{(2)}\right)^T,$$
$$db^{(3)} = \frac{1}{100} \sum dZ^{(3)}.$$

Hidden Layer Gradients:
$$dZ^{(2)} = \left(W^{(3)}\right)^T dZ^{(3)} \cdot g'(Z^{(2)}),$$
$$dW^{(2)} = \frac{1}{100}\, dZ^{(2)}\, \left(A^{(1)}\right)^T,$$
$$db^{(2)} = \frac{1}{100} \sum dZ^{(2)},$$
$$dZ^{(1)} = \left(W^{(2)}\right)^T dZ^{(2)} \cdot g'(Z^{(1)}),$$
$$dW^{(1)} = \frac{1}{100}\, dZ^{(1)}\, X^T,$$
$$db^{(1)} = \frac{1}{100} \sum dZ^{(1)}.$$