Final Project Report - Deep Neural Networks for Price Predictions using Used Cars

Shrey Rakesh Kevadia, Fernando Malca Luque, Anthony Mingus, William Widjaja

Code repository: github.com/n-ndo/csc671-final-project

1 Introduction

When purchasing a vehicle from a dealership, customers rightly expect full transparency during their transaction: a complete and accurate history of the vehicle, a clear breakdown of every fee, and fair pricing. This responsibility falls upon the dealership to provide this kind of "honesty" to their buyers. Yet in recent years, many dealerships have engaged in deceptive tactics against their clients, involving hiding fees, inflating add-on costs, and failing to disclose past damages of the vehicle. In 2024, the Federal Trade Commission (FTC) and Illinois regulators fined a group of ten dealerships known as the Leader Automotive Group and its parent dealership, AutoCanada, for engaging in these tactics, which involved odometer tampering and withholding accident histories from their clients. In response, the FTC proposed the "Combating Auto Retail Scams" rule to outlaw any misrepresentation of fees or vehicle condition. However, dealers and consumer advocates alike criticized this rule for potentially increasing paperwork, costs, and time for both parties if it were implemented.

To address the challenges that the rule may bring, our research investigates whether deep neural networks can automatically learn the patterns to provide fairness and transparency to consumers and dealers. We trained and compared four model architectures: Multilayer Perceptron (MLP), Long Short-Term Memory (LSTM), TabNet, and Neural Factorization Machine (NFM). We hope that the models are able to predict an accurate pricing for vehicles given some information about them such as their brand and model.

1.1 Dataset

The dataset we used comes from Kaggle and includes detailed information about used cars. It contains features like the car's model year, mileage, transmission type, fuel type, and manufacturer. Some of the columns were related to color, engine type, and title status. The dataset includes both numbers and text data, and each row represents a single car listing. Before using it, we cleaned and processed this data to remove unnecessary columns and make it ready for training our model.

1.2 Data Preprocessing

In this project, the data preprocessing stage involved several key steps to prepare the used car dataset for training the machine learning model. First, the *price* and *mileage* columns were cleaned by removing symbols like dollar signs ($), commas, and text like "*mi.*", then converted into numerical format. Irrelevant columns such as `model`, `engine`, `clean_title`, `ext_col`, and `int_col` were dropped to reduce noise. The remaining features were separated into numerical and categorical types. For numerical features, missing values were filled using the median and then scaled to a standard range using `StandardScaler`. Categorical features had missing values filled with the most frequent category and were then encoded via one-hot encoding. We then standardized all numerical values and one-hot encoded categorical variables. The dataset was split into 70% training, 15% validation, and 15% test sets. Finally, the processed data was converted into PyTorch tensors to be compatible with the neural network model. This preprocessing ensured clean, consistent, and normalized input for the model to learn effectively.

1.3 Mean Squared Error (MSE) and Mean Absolute Error (MAE) Loss Functions

For all four models, we tracked both the Mean Squared Error (MSE) and the Mean Absolute Error (MAE) during training, validation, and testing to evaluate model performance. In addition, for some models that compared the predicted vs. actual prices, we reported MAE on the test set to quantify average absolute deviation.

Given true targets y_i and predictions $\hat{y}_i$ over N examples:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \qquad \text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|y_i - \hat{y}_i\right|.$$

By evaluating both metrics, we gained complementary perspectives: MSE's sensitivity to outliers highlights large deviations, while MAE conveys the average magnitude of prediction errors in actual dollar amounts.

2 Multi-Layer Perceptron (MLP)

2.1 Model Overview

Our first model is a feed-forward Multi-Layer Perceptron (MLP) designed to learn non-linear relationships between tabular input features and vehicle price. We trained two versions using this architecture. Version 1 served as a baseline: a relatively shallow network trained for a fixed 1,000 epochs using full-batch updates and light dropout. In Version 2, we deepened the architecture, switched to mini-batch training, strengthened dropout regularization, lowered the learning rate, and added an early-stopping criterion to halt training once validation performance stopped improving. We tried to focus on improving accuracy while preventing overfitting and reducing the overall training time.

2.2 Hyperparameters

We configured two distinct MLP versions to assess the effects of each configuration on its overall performance.

Version 1 (Baseline, 1000-Epochs) This baseline model uses a classical two-layer architecture with full-batch updates to establish a reference performance:

- **Epochs:** 1000
- **Batch size:** Full-batch
- **Hidden layers:** [128, 64]
- **Activation:** ReLU
- **Dropout:** [0.20, 0.10]
- **Learning rate:** 0.001

Version 2 (Optimized with Early Stopping) To improve convergence and prevent overfitting, this version deepens the network, employs mini-batch training, and implements early stopping:

- **Epochs:** up to 1000 with early stopping (patience = 50 epochs)
- **Batch size:** 64 (mini-batch)
- **Hidden layers:** [256, 128, 64]
- **Activation:** ReLU
- **Dropout:** [0.30, 0.20, 0.00]
- **Learning rate:** 0.0005

2.3 Optimizers and Fine Tuning

Both MLP variants employ the `Adam` optimizer with default momentum parameters to update network weights. No extra fine tuning was involved for both versions of the model.

2.4 Results

Version 1 (Baseline, 1000-Epochs) Version 1's MAE fell from about $43,000 to $20,352 on the validation set, plateauing around epoch 600 and exhibiting slight overfitting by epoch 1000 with negligible additional gains.

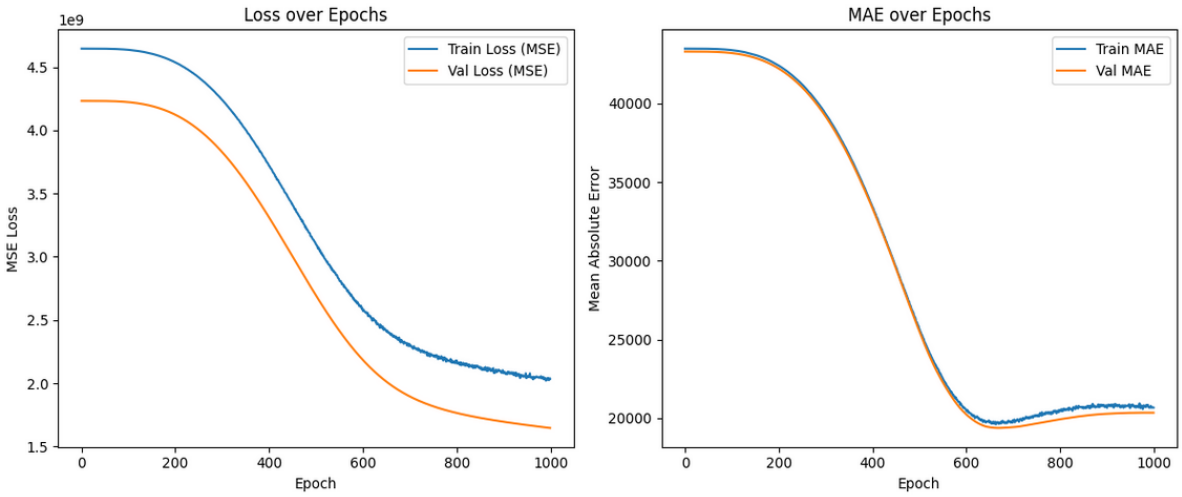

Figure 1: (Version 1) Training and validation MAE over 1000 epochs.

Version 2 (Optimized, Early Stopping) Version 2's MAE plummeted within the first 100 epochs and began to level off around epoch 120. Early stopping triggered at epoch 166, yielding a final validation MAE of approximately $13,024—an improvement over the baseline and in the training time.

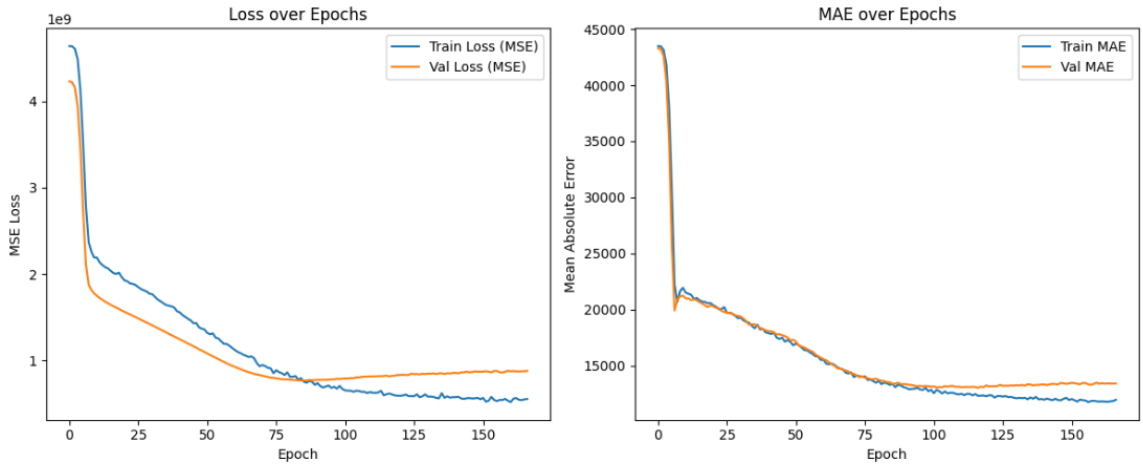

Figure 2: (Version 2) Training and validation MAE with early stopping.

3 Long Short-Term Memory (LSTM)

3.1 Model Overview

For our second model, we decided on using a Long Short-Term Memory model architecture. This LSTM model was largely exploratory, since our used-car dataset lacks an inherent temporal sequence and the order of features carries no sequential meaning. Nonetheless, we structured the input as if each feature vector were a one-step "time series" and applied an LSTM layer to capture complex nonlinear interactions via its gating mechanisms. We hypothesized that the LSTM's ability to learn dependencies might transfer to tabular data, despite the absence of true temporal structure.

3.2 Hyperparameters

We configured the LSTM with these hyperparameters:

- **Epochs:** 1000
- **Batch size:** 64 (mini-batch)
- **Hidden size:** 32 units
- **Dropout:** 0.001 after the LSTM layer
- **Learning rate:** 0.001

3.3 Optimizers and Fine Tuning

We used the `Adam` optimizer with default parameters. During initial experiments, learning rates above 0.001 led to unstable MAE, while rates below 0.001 converged too slowly. Hidden sizes larger than 32 caused overfitting, so we fixed the hidden dimension at 32 and retained 30% dropout to regularize.

3.4 Optimizers and Fine Tuning

For this model architecture, we used the Adam optimizer with default parameters. Additionally, we applied a log transformation to the vehicle prices:

$$y' = \log(y + 1)$$

to spread out small values and compress large ones; we then inverted this transform when computing the final MAE.

During the first training runs of the model, we learned that the learning rates above 0.001 led to unstable MAE, while rates below 0.001 converged too slowly. Hidden sizes larger than 32 caused overfitting, so we fixed the hidden dimension at 32 and retained 30% dropout to regularize.

3.5 Results

During the final training and validation runs of the model, we found out that the MSE for our validation dropped rapidly, which indicated that the model learned the task quickly. After reversing the log transform of the prices, the validation MAE dropped from over $40,000 initially to around $12,500 at convergence.

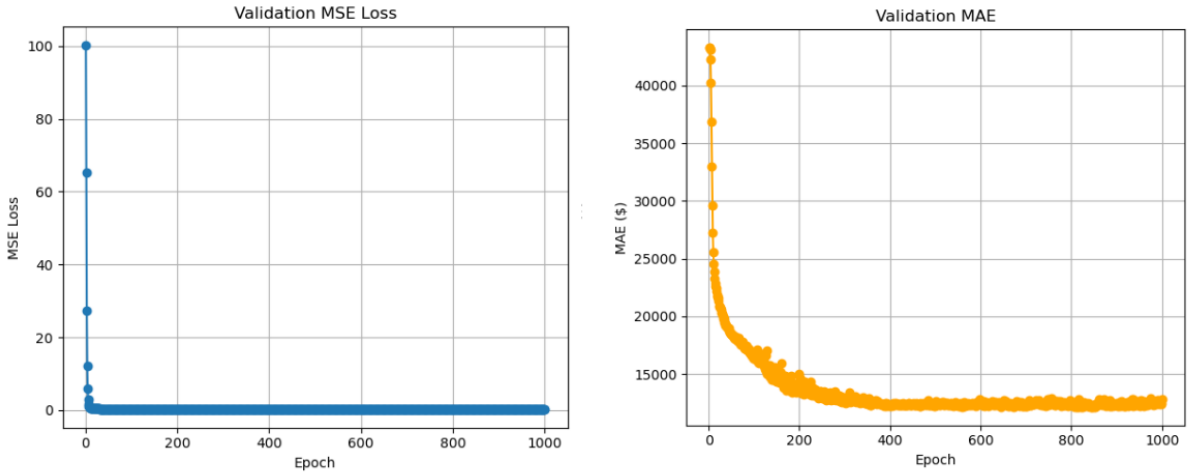

Figure 3: (LSTM) validation MSE (log scale) and validation MAE over epochs, plus actual vs. predicted prices.

Using the Actual vs Predicted plot, we saw that the LSTM model reliably estimates prices for the bulk of listings under $100,000—these points are close to the line—while consistently failing to predict higher-end vehicles (¿$100,000), likely due to their scarcity in the training data.

The model achieved a final test MAE of approximately $12,700. These results highlight that, although the LSTM captures the overall pricing trend, it struggles with extreme outliers and offers limited benefit over simpler, non-sequential architectures for this task.

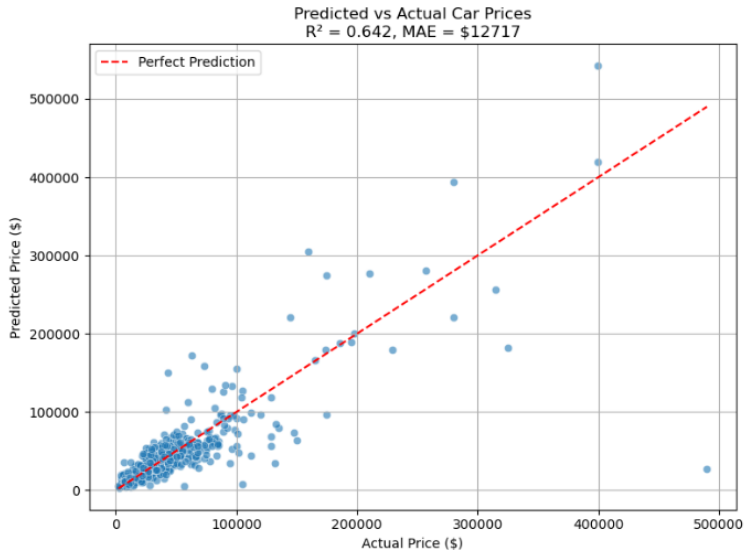

Figure 4: (LSTM) Actual vs. Predicted Prices on the test set.

4 TabNet

4.1 Model Overview

For our third model, we decided to use a TabNet model architecture. TabNet, introduced by Google Cloud researchers in 2019, is specifically designed for tabular data by combining decision-step feature masking with sequential attention. Unlike traditional MLPs, which apply the same transformations to all features at each layer, TabNet's attentive masks select the most informative subset of features at every decision block. This allows the network to focus on relevant feature interactions sequentially, improving interpretability and efficiency when learning from structured datasets like ours.

4.2 Hyperparameters

We initialized TabNet with decision and attention dimensions $n_d = n_a = 16$, and these hyperparameter configurations:

- **Learning rate:** 0.02
- **Max epochs:** 1000
- **Early stopping patience:** 20
- **Batch size:** 256
- **Virtual batch size:** 64

n_d is the number of decision units that are used to transform the inputs to the final prediction layers. n_a is the number of attention units that decide which features to focus on at each decision step.

4.3 Optimizers and Fine Tuning

We used the `Adam` optimizer with default parameters for this model. Additionally, we applied a similar log transformation to the vehicle prices as seen for the LTSM model to improve the distribution of the dataset and stabilize training.

4.4 Results

The model's training MSE dropped sharply within the first 100 epochs, showing us rapid initial learning. A spike in loss during those early iterations—likely due to the higher initial learning rate—was followed by a smooth decrease and stabilization, indicating solid convergence. Meanwhile, the validation MAE computed in log-price space leveled off between 0.4 and 0.5, demonstrating consistent predictive performance:

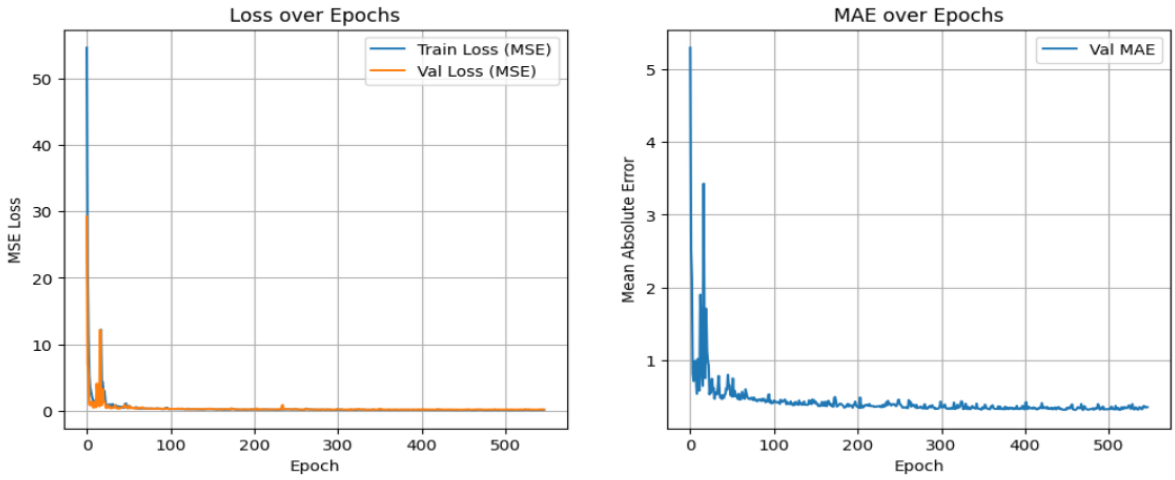

Figure 5: (TabNet) training MSE and validation MAE (log scale) over epochs.

Although training was permitted up to 1000 epochs, early stopping halted the run around epoch 550 once the patience threshold was reached, confirming that further epochs yielded diminishing returns. It is worth noting that performance measured in log space does not directly map to dollar accuracy, so we also examined the Actual Vs. Predicted plot:

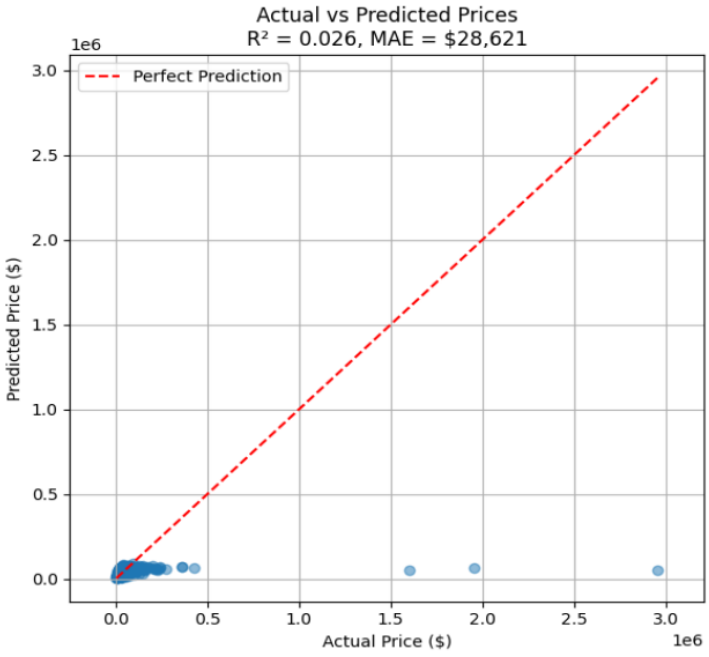

Figure 6: TabNet: Actual vs. Predicted prices on the test set.

On the test set, the model achieved an R^2 score of 0.026, indicating it explains only 2.6% of the variance in actual prices—a very low value. The MAE was $28,621, meaning that on average the model's predictions were off from the actual prices by that amount.

5 Neural Factorization Machine (NFM)

5.1 Model Overview

For our fourth model architecture, we implemented a Neural Factorization Machine (NFM), which combines the efficient second-order interaction modeling of a Factorized Machine (FM) with the ability of a deep neural network model to learn non-linear, higher-order patterns. We chose this architecture due to the dataset containing many high-cardinality categorical fields that would otherwise explode into a huge, sparse matrix.

5.2 Factorization Machines

Factorization Machines excel in handling sparse, high-dimensional inputs by learning a low-rank embedding $\mathbf{v}_i \in R^k$ for each feature x_i, then modeling every pairwise interaction via dot products. Given $x \in R^n$, the FM predicts:

$$\hat{y}_{\text{FM}}(x) = w_0 + \sum_{i=1}^{n} w_i\, x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle\, x_i\, x_j,$$

where w_0 is the global bias, w_i the linear weight, and $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ the embedding interaction.$\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ ensures that only non-zero features are considered in the prediction.

Despite the efficiency of Factorization Machines, they fail to accurately compute predictions when utilizing real-world data. This is because FMs remain linear in those embedding dot-products, whereas real-world data is often highly non-linear and contains higher-order patterns. For example, FMs fail to accurately capture how the combined relationships of brand, age, and mileage influence the overall price.

5.3 Neural Extension

This is where the idea of Neural Factorization Machines comes in to deal with the limitations of real-world data. Firstly, NFMs embed each feature x_i into a low-dimensional vector $\mathbf{v}_i \in R^k$, scaling it by its value x_i. All pairwise, element-wise products of the scaled embeddings are then pooled into a bi-interaction vector:

$$P \;=\; \sum_{i=1}^{n} \sum_{j=i+1}^{n} \big(x_i\, \mathbf{v}_i\big) \circ \big(x_j\, \mathbf{v}_j\big),$$

where $\circ$ denotes the Hadamard (element-wise) product. This vector P summarizes all second-order relationships among active features. Afterwards, the NFM feeds P through a compact multi-layered perceptron—consisting of fully connected layers with activations and dropouts—to learn the nonlinear and high-order patterns that FMs fail to capture. The final prediction given by NFM can be written as:

$$\hat{y}_{\text{NFM}}(x) = w_0 + \sum_{i=1}^{n} w_i\, x_i + f(P).$$

5.4 Hyperparameter Grid Search

We ran a grid search over six key hyperparameters, training each configuration for up to 2000 epochs:

- **Embedding dimension:** {8, 16, 32, 64}
- **MLP hidden layers:** {[64,32], [128,64], [256,128,64], [512,256,128,64]}
- **Dropout rate:** {0.1, 0.2, 0.3, 0.4}
- **Learning rate:** {0.1, 0.01, 0.001, 0.0001}
- **Batch size:** {32, 64, 128, 256}
- **Weight decay:** {0, 0.1, 0.01, 0.001}

The best combination after approximately two hours of search was:

- **Embedding dimension:** 16
- **MLP layers:** [256, 128, 64]
- **Dropout rate:** 0.2
- **Learning rate:** 0.001
- **Batch size:** 128
- **Weight decay:** 0.001

To confirm that the hyperparameters were the best possible configuration, we decided to run the model using values outside of the grid-search space. After around 30 extra minutes of training and validating, the values of our grid search yielded the best configuration values, as the values outside of the grid search space either outperformed or resulted in little to no improvement to the model's overall performance.

For the final training and validation runs, we capped training at 2,000 epochs and applied early stopping with a 150-epoch patience, helping to avoid overfitting and prevent unnecessary computations.

5.5 Optimizing and Fine Tuning

We decided to use `AdamW` for this model as it kept weight penalties separate from the learning step, allowing us to efficiently regularize dense embedding vectors without corrupting gradient updates. We then wrapped `AdamW` in a `CosineAnnealingWarmRestarts` scheduler, periodically (every 200 epochs) resetting the learning rate via a half-cosine decay, so the optimizer could break free from shallow minima and keep exploring

A key change in our data preprocessing involved replacing one-hot encoded categorical features with integer indices passed through an embedding layer, enabling the NFM to learn low-dimensional representations of high-cardinality variables efficiently. As with our LSTM model, we trained on log(price + 1) and then inverted the transform during final evaluation, which helped stabilize the data and improve accuracy.

5.6 Results

After our final training and validation run—using the selected hyperparameters, `AdamW` optimizer with `CosineAnnealingWarmRestarts` scheduler, and fine-tuning—the NFM showed rapid convergence and consistent error reduction. In the first 150 epochs, both training and validation MSE (on the log-price scale) dropped sharply before plateauing, and the corresponding MAE curves mirrored this behavior, confirming that the model was effectively minimizing large errors while steadily reducing average absolute error.

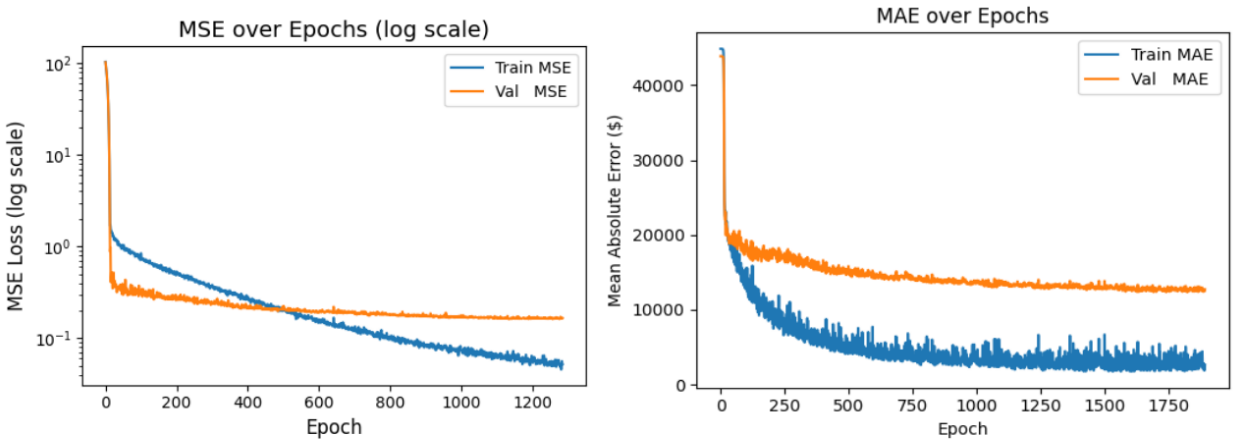

Figure 7: (NFM) Training and validation MSE (log scale) and MAE over epochs

The Actual vs. Predicted plot shows underprediction of high-priced vehicles, with most points below the line, while low and mid-priced listings are estimated more accurately. The model's MAE is $15,714, indicating underfitting at the high end and the need to address extreme values (outliers) in the dataset.

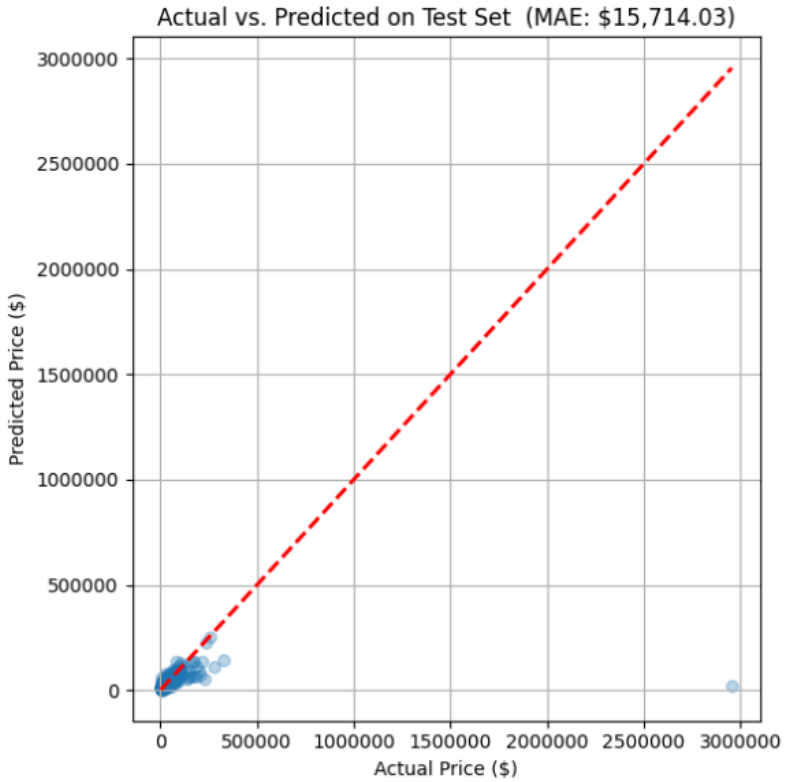

Figure 8: (NFM) Actual vs Predicted Prices on the test set (MAE: $15,714.03).

6 Conclusion

Through extensive testing and iteration, the Long Short-Term Memory (LSTM) model proved most effective at predicting used car prices, achieving the lowest validation MAE of $12,717 by capturing complex feature interactions even without a true temporal sequence. The Multi-Layer Perceptron (MLP) plateaued around an MAE of $13,000, revealing that a simple feedforward network struggles to model the intricate relationships in our data. The TabNet architecture delivered interpretable, attention-guided feature selection and stable convergence, but would have an MAE $28,621, which meant that it underfitted high-end vehicles ($28,621). Finally, the Neural Factorization Machine (NFM) leveraged pairwise embedding interactions within a deep MLP framework to reach an MAE of $15,714, which still underpredicts extreme vehicle pricing.

While the LSTM model is currently the best-performing, its MAE is still not low enough for deployment in a commercial setting, especially in contexts where precise car pricing is critical, such as combating fraudulent pricing in the used car market. Therefore, it remains necessary to refine our current model, conduct further testing, and explore additional architectures to close the performance gap.

The ultimate goal is to develop a model that can consistently predict car prices with a sufficiently low error margin to be reliable for both consumers and businesses in a real-world environment. We must continue to explore and test additional models and refine our existing approach to achieve a level of accuracy that meets the standards required for commercial applications.

6.1 Usage of AI for Exploration

During our model selection process, we used AI tools-such as ChatGPT-to identify promising architectures like LSTM, TabNet, and NFM. While these suggestions provided a good starting point, we still conducted a thorough review and extensive online research to confirm the strengths and limitations of each model architecture. In addition, other model architectures were recommended to us by AI tools for future considerations, such as Random Forest and DCN-V2.

It is worth adding that while AI chatbots are able to give out less conventional or emerging model architectures, such as NFM or DCN-V2, the proposals for these architectures should still be personally validated as some of the models might not work well for the task you're trying to complete using a deep neural network.

Works Cited

- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* Advances in Neural Information Processing Systems, 32.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12, 2825–2830.
- Federal Trade Commission. (2024, December 19). *FTC, Illinois Take Action Against Leader Automotive Group for Overcharging and Deceiving Consumers Through Add-Ons, Junk Fees, Bogus Reviews.*
- He, X., & Chua, T.-S. (2017, August 16). *Neural factorization machines for sparse predictive analytics.* arXiv.
- Sharma, D. (2018, November 15). *LSTM networks: A detailed explanation.* Towards Data Science.
- Arik, S., & Pfister, T. (2021). *TabNet: Attentive interpretable tabular learning.* Proceedings of the AAAI Conference on Artificial Intelligence, 35(8), 6679–6686. 10.1609/aaai.v35i8.16826