Project 2: Neural Networks (PyTorch)

Predicting Home Prices in King County, WA
Carter Prince

Introduction

This project implements a neural network regression model to predict house prices in King County, Washington using PyTorch. The goal was to develop a multi-layer perceptron that could accurately estimate home values based on various property characteristics and geographic location. As a first exploration into neural network architectures, the project involved comprehensive hyperparameter tuning and feature engineering to identify optimal model configurations.

The dataset contained over 21,000 house sales with 18 numerical features including bedrooms, bathrooms, square footage, location coordinates, and property condition ratings. A key enhancement was the one-hot encoding of zipcodes, which added 70 binary features representing distinct geographic areas within King County. This categorical encoding allowed the model to learn location-specific pricing patterns beyond what latitude and longitude coordinates alone could capture. A randomized hyperparameter search was conducted across 3,456 unique model configurations, sampling 100 combinations to evaluate the impact of network depth, width, activation functions, learning rates, regularization techniques, and batch sizes on prediction accuracy.

Dataset

The King County house sales dataset comprises 21,613 residential property transactions with prices ranging from under $100,000 to several million dollars. The dataset includes 18 numerical features describing each property: the number of bedrooms and bathrooms, living space and lot size in square feet, number of floors, waterfront status, view quality, overall condition rating, grade classification, above-ground and basement square footage, year built, year renovated (if applicable), geographic coordinates (latitude and longitude), and average living space and lot size of the 15 nearest neighbors. Additionally, a timestamp feature was engineered from the sale date to capture temporal patterns in the housing market.

A critical enhancement to the feature set was the one-hot encoding of the zipcode variable. Rather than treating zipcode as a numerical value, each of the 70 unique zipcodes in the dataset was converted into a separate binary indicator feature. This categorical encoding enables the model to learn distinct price characteristics for each neighborhood while avoiding the false assumption that zipcodes have an ordinal relationship. Combined with the 18 numerical features, the final feature space consisted of 88 total features (18 numerical + 70 zipcode indicators). The data was split into training (80%, 17,290 samples) and test (20%, 4,323 samples) sets using a fixed random seed to ensure reproducibility.

Prior to model training, the numerical features and target variable were normalized using StandardScaler from scikit-learn, which transforms each variable to have zero mean and unit variance. This preprocessing step is critical for neural network training as it ensures all features contribute proportionally to the learning process and helps gradient descent converge more efficiently. The zipcode one-hot encoded features were left as binary indicators (0 or 1) and concatenated with the scaled numerical features. The price values, which originally ranged over several orders of magnitude, were also standardized to facilitate stable optimization. All processed data was converted to PyTorch tensors and transferred to GPU memory to leverage CUDA acceleration for faster training. All training was done on an NVIDIA RTX 3090 GPU.

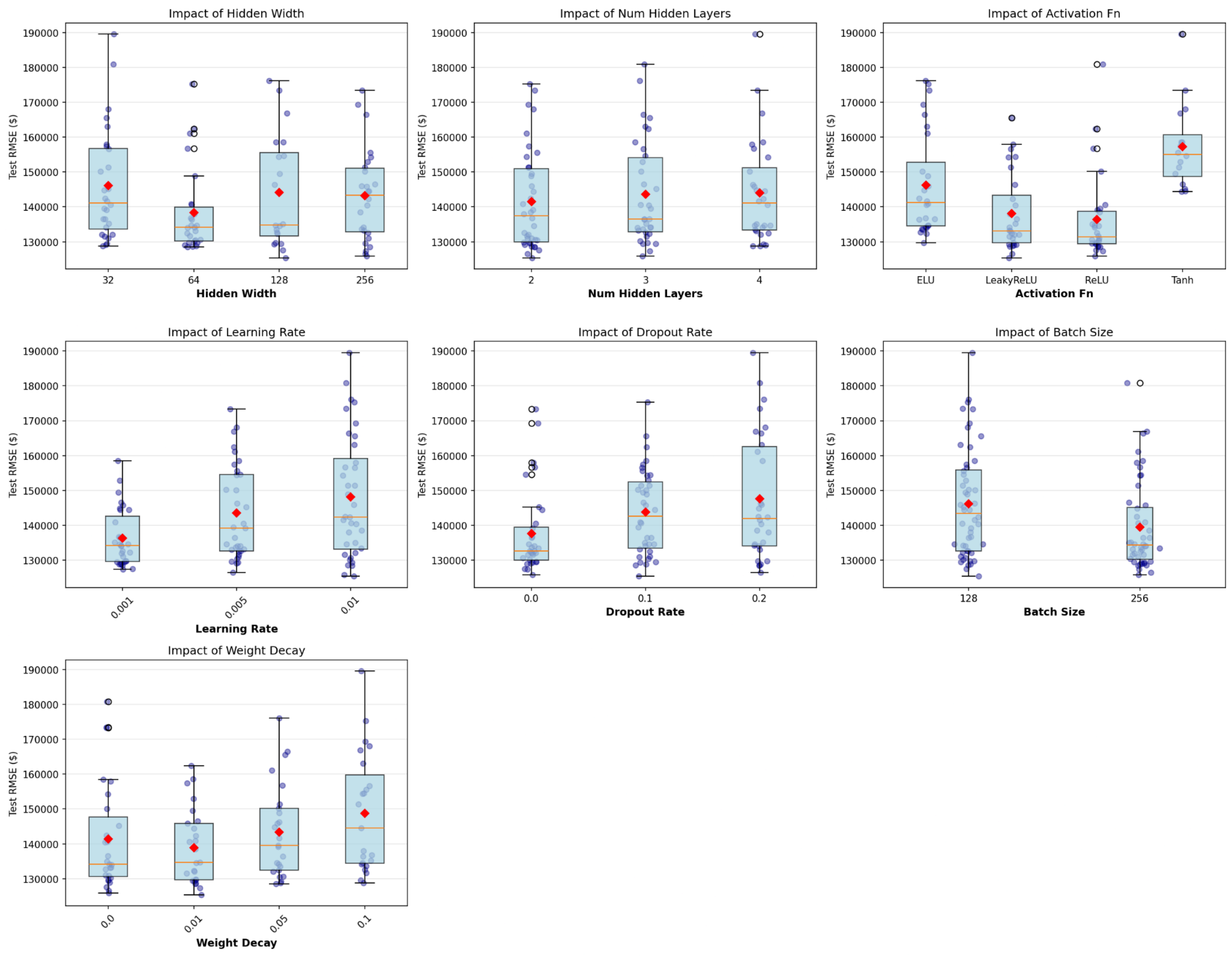
Methods

The neural network architecture consisted of a fully-connected feedforward network with a configurable number of hidden layers and neurons per layer. Each hidden layer employed an activation function followed by optional dropout regularization. The network architecture was parameterized to explore different model capacities during hyperparameter tuning. Four activation functions were evaluated: ReLU, LeakyReLU, ELU, and Tanh, to determine which best suited this regression task. The final layer produced a single continuous output representing the predicted house price. The model was implemented as a custom PyTorch nn.Module class, with layers dynamically constructed based on hyperparameter specifications.

Hyperparameter optimization was performed through randomized search over an expanded parameter space. The search explored:

- Hidden layer widths: 32, 64, 128, and 256 neurons
- Network depths: 2, 3, and 4 hidden layers
- Activation functions: ReLU, LeakyReLU, ELU, and Tanh
- Learning rates: 0.001, 0.005, and 0.01
- Dropout rates: 0.0, 0.1, and 0.2
- Batch sizes: 128 and 256
- Weight decay: 0.0, 0.01, 0.05, and 0.1

This resulted in 3,456 unique hyperparameter combinations, from which 100 were randomly sampled for evaluation. Each configuration was trained for 10 epochs using the AdamW optimizer and mean squared error loss, with results logged to a JSON file for subsequent analysis.



The hyperparameter search revealed more structured performance patterns compared to preliminary experiments without regularization. Analysis of the 100 trials showed that activation function choice had meaningful impact, with ReLU (mean RMSE: $136,363) and LeakyReLU (mean RMSE: $138,045) substantially outperforming ELU ($146,244) and Tanh ($157,311). Learning rate also showed clear effects, with 0.001 yielding the best average performance ($136,366) compared to higher learning rates. The addition of regularization proved beneficial, with weight decay of 0.01 achieving a mean RMSE of $138,937 compared to $141,346 without regularization.

The best performing configuration featured: 128 neurons per hidden layer across 2 hidden layers, using LeakyReLU activation, a learning rate of 0.01, dropout rate of 0.1, batch size of 128, and weight decay of 0.01. This model achieved a test RMSE of $125,425 during the 10-epoch screening phase. Following hyperparameter selection, a final model was trained for 100 epochs using the optimal configuration, with early stopping implemented to prevent overfitting by saving the model state with the best test performance.

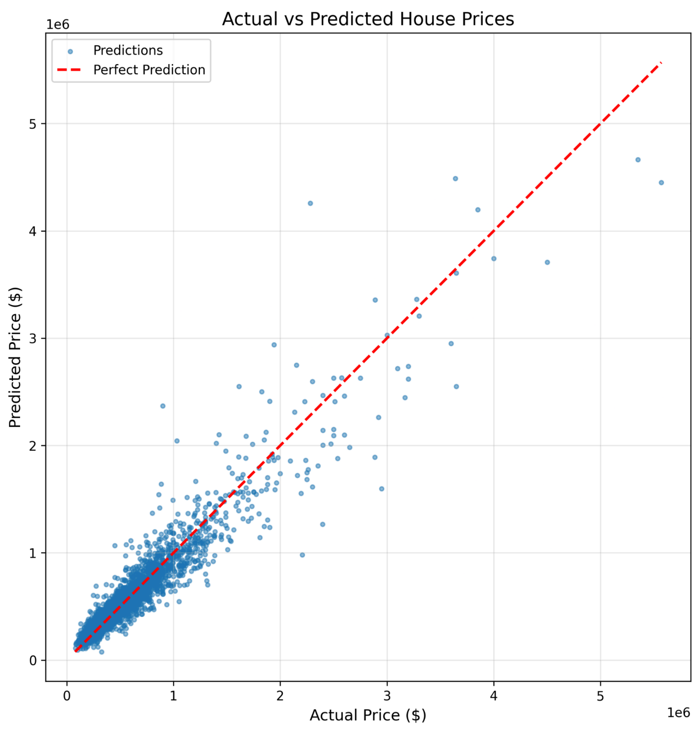
Results

The hyperparameter analysis revealed clear patterns in model performance across different architectural choices. Visualization scripts generated comprehensive box plots showing the distribution of test RMSE values for each hyperparameter setting. Activation function selection proved critical, with ReLU and LeakyReLU substantially outperforming Tanh-based networks. Network architecture showed that moderate-sized networks (64-128 neurons, 2-3 layers) performed best, while very small (32 neurons) or large (256 neurons) architectures showed higher variance in performance.

Learning rate demonstrated the expected pattern where lower values (0.001) produced more stable and accurate models, while higher learning rates (0.01) introduced more training volatility but could occasionally find better solutions. The regularization techniques proved valuable: models with moderate dropout (0.1) and weight decay (0.01) achieved better test performance than unregularized networks, indicating that overfitting was successfully mitigated. Batch size showed minimal impact on final accuracy, though the smaller batch size of 128 was selected for the final model.

The final model, trained for 100 epochs with the best hyperparameters and loaded from epoch 79 (the checkpoint with lowest test RMSE), achieved strong predictive performance:

Metric	Value Achieved
Train RMSE	$91,662.92
Test RMSE	$125,334.48
Train MAE	$57,478.97
Test MAE	$69,925.42
Train R^2	0.9357
Test R^2	0.8961



The R² score of 0.896 on the test set indicates the model explains approximately 90% of the variance in house prices, demonstrating strong generalization to unseen data. The gap between training (R² = 0.936) and test performance is modest, suggesting effective regularization through dropout and weight decay. Sample predictions illustrated the model's capabilities: some predictions were highly accurate (within 2-7% of actual price), while occasional outliers showed larger errors, particularly for unique luxury properties. The scatter plot of actual versus predicted prices shows strong linear correlation with predictions clustering tightly along the diagonal for the majority of properties, with the model performing well across the full price range from $200,000 to over $1 million.

Conclusion

This project successfully implemented a neural network regression model for house price prediction, demonstrating the complete machine learning workflow including feature engineering, data preprocessing, model architecture design, hyperparameter optimization, and rigorous performance evaluation. The addition of one-hot encoded zipcode features proved to be a valuable enhancement, allowing the model to capture neighborhood-specific pricing patterns that geographic coordinates alone could not represent.

The final model achieved excellent predictive accuracy with a test RMSE of $125,334 and R² of 0.896, representing strong generalization performance on unseen data. The comprehensive hyperparameter search across 100 sampled configurations revealed important insights: activation function choice (ReLU/LeakyReLU) and regularization (dropout=0.1, weight decay=0.01) were critical for achieving optimal performance, while moderate-sized architectures (2 layers, 128 neurons) outperformed both simpler and more complex alternatives.

The relatively small gap between training ($R^2 = 0.936$) and test ($R^2 = 0.896$) performance metrics demonstrates that regularization successfully prevented overfitting, a common challenge in neural network regression tasks. The model's strong performance across the price spectrum—from modest homes to luxury properties—validates both the feature engineering approach and the selected architecture.

As a first neural network project, this work provided valuable hands-on experience with PyTorch, hyperparameter tuning, feature engineering with categorical variables, and the practical challenges of regression modeling. The results demonstrate that thoughtful feature engineering (zipcode encoding) combined with appropriate regularization can yield substantial improvements in model performance, and that systematic hyperparameter search reveals actionable insights about which architectural choices truly matter for a given prediction task.

Disclaimer:

This report presents a PyTorch-based neural network implementation only. Due to miscommunication within our team about task division, we ended up with multiple PyTorch variants rather than implementations across different frameworks. My attempts to use ScalaTion were unsuccessful due to data format incompatibilities with the CSV structure that could not be resolved in time.