Predicting rental costs in the Indian housing market: a deep learning approach

Tyler Brown 01/1355033

Introduction

Deep learning is an important alternative to ordinary least squares (OLS) for regression tasks. Compared to OLS, deep neural networks can better model non-linear functions and constitute a more flexible tool for prediction. In this project, I use both approaches to predict cross-sectional rent prices on housing in India using data on house characterstics and location[1]. After hyperparameter tuning, I deploy a three-layer deep neural network (DNN) to predict prices with significantly lower error compared to a naive OLS regression model. I am able to reduce the mean squared error (MSE) of the model from 0.64 in OLS to 0.17 with a DNN, indicating that deep learning is indeed applicable for this task and can better predict rental prices than simpler models.[2]

Analysis

The raw data contains no missing values and comprises 11 potential features and one target variable (rent price). The data contains relatively few numerical features (number of bathrooms, total number of rooms, date of listing, and size). The remaining features are either ordinal (furnishing status) or categorical (area type, floor out of total floors, area locality, city, preferred tenant type, and point of contact). The distribution of rental prices across select categorical features is presented in Fig. 1, where we see that generally units in carpet areas, in Mumbai, with furniture, agent-listed, and geared towards families have the highest prices.

[1]The data, available from Kaggle, are sourced from Magicbricks, an online rental platform.

[2]Replication data and code for this project can be found at the link here: https://github.com/tylerjamesbrown7/sl_final_project

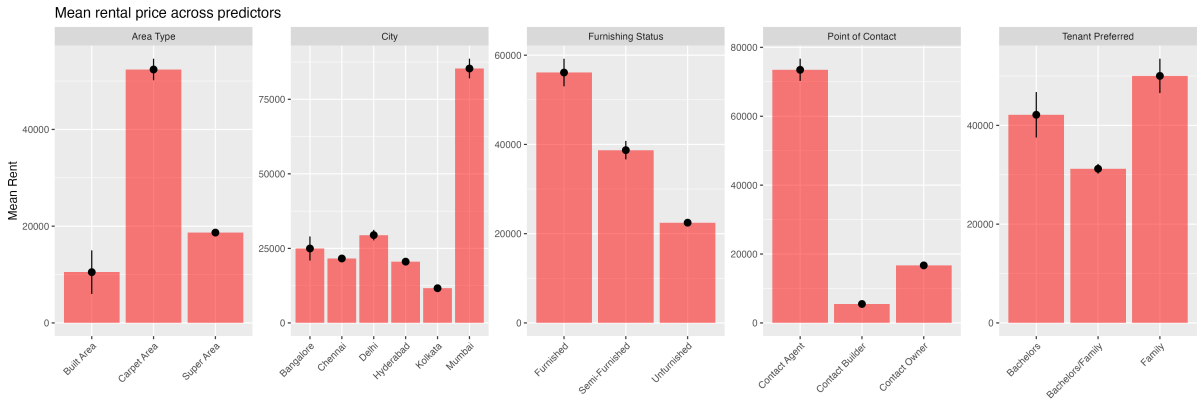


Figure 1: Mean rental price across select categorical predictors

In order to prepare the data for passing into the neural network, I first scale all numerical features. Second, I convert furnishing status to a binary indicator (semi- or fully-furnished as 1, or non-furnished as 0) and expand all categorical variables into one-hot coded vectors for each level, so that each level is represented as a vector of binary indicators. Additionally, I include the listing date as a (scaled) numerical predictor to account for time trends in rental prices. I also apply a log transformation on rental price to stabilize model predictions and make results across OLS and the neural network more comparable. The data is then split into a training and test set with an 80-20 split, and the final matrix contains 4746 observations and 2734 predictors (3796 training, 950 test).

Methods

Deep neural network

Network structure

To estimate non-linearities in rental prices, I deploy a three-layer neural network. The nodes in each layer pass a linear combination of values from the previous layer through a rectified linear unit (ReLU) activation function, which allows the network to model functions of arbitrary complexity. The network structure is summarized in the table below:

Layer	Nodes	Activation Function
Input	**2734**	
Hidden 1	4000	ReLU
Dropout 1		
Hidden 2	1000	ReLU

Layer	Nodes	Activation Function
Dropout 2		
Hidden 3	100	ReLU
Dropout 3		
Output	**1**	**Linear**

Model training

One issue with this data is that the number of predictors is roughly half the number of observations, making overfitting a risk in model training, as the model may be overparametrized relative to the amount of training data. Accordingly, I use a dual approach to combat overfitting. First, I deploy dropout layers after each hidden layer, which randomly selects certain nodes to censor before passing it to the next hidden layer. This shows the network less information in each pass than in the absence of dropout, allowing it to train on more generalizeable structures within the predictors. Likewise, I additionally use an L2 regularizer in the loss function, penalizing the magnitude of the network weights and trading off a small increase in bias for a large decrease in the variance of the final estimator.

I set the dropout rate fixed at 0.25 for all dropout layers. In order to identify appropriate hyperparameters (i.e. the learning rate and the regularization parameter λ), I run a model search across a parameter grid, with learning rates selected from $\{0.001, 0.01, 0.1\}$ and λ from $\{10^{-6}, 10^{-5}, 10^{-4}\}$. I additionally adjust the model structure, starting with a larger number of nodes in the second and third hidden layers and removing nodes while overfitting was still present after 20 epochs. In order to accelerate the training process, I use a batch size of 32, which I do not tune further.

After training the nine possible models, the most stable model that does not indicate overfitting with the lowest validation error had a learning rate of 0.01 and $\lambda = 10^{-5}$. Models with a higher learning rate achieve a lower validation error sporadically, but this is not necessarily a reliable result, as the learning curves are highly variable. Likewise, a decrease in the regularization parameter leads to increasing validation error in later epochs. The learning curves (Fig. 2) demonstrate 1) that a higher learning rate is unstable (preventing effective gradient descent), and a low learning rate trains the model too slowly. Likewise, we see that the model is not particularly sensitive to λ; this makes sense, since dropout layers already have a regularizing effect. It makes further sense that the training curves have higher error than the validation curves because training errors are calculated *with* dropout present, meaning the network evaluating the data is smaller than the final network evaluated on the test set.

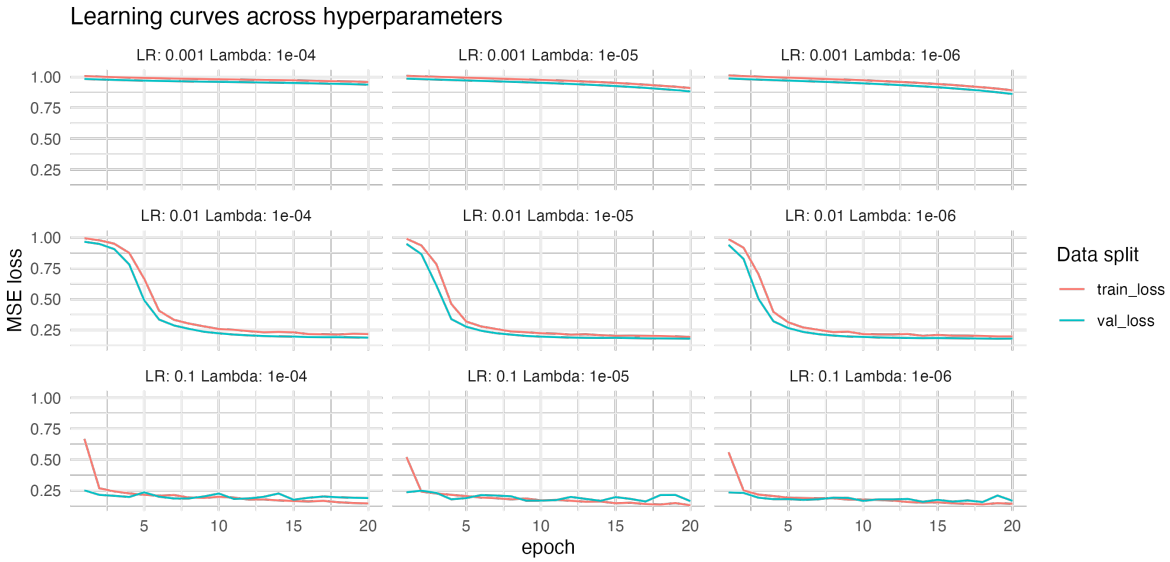


Figure 2: Learning curves across model search grid

OLS Model

The OLS baseline model includes all predictors passed to the neural network and uses a standard linear estimation procedure. As the model solves the optimization problem in one step, the training time is considerably faster than for the neural network.

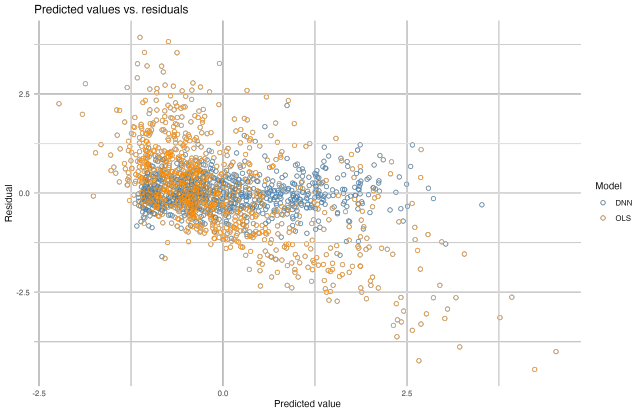
Results

In order to contextualize the performance of the deep neural network (DNN), I evaluate metrics against a baseline OLS regression of log rent on all predictors included in the DNN. The mean absolute error (MAE) and mean squared error (MSE) of the two models are presented in the table in Fig. 3.

From the errors, it is clear that the DNN performs significantly better in both MSE and MAE compared to OLS on the test set. Likewise, when we look at the distribution of residuals, problems with the linear regression model become more apparent, as the normality assumption in the error term is likely not met. The DNN, on the other hand, makes similar errors across the range of predicted values and has a much smaller residual variance. It is also noteworthy that the error is similar across training and test sets in the DNN, implying that our empirical risk estimate closely approximates the true risk. However, the training error for OLS is much lower than for the test set, suggesting overfitting. Likewise, the MAE is lower than the MSE,

Figure 3: Model performance, OLS vs. DNN

Metric	OLS	DNN	
MAE	0.539	0.308	Test
MSE	0.641	0.174	Test
MAE	0.112	0.308	Train
MSE	0.038	0.162	Train



perhaps indicating that outliers are problematic for the OLS estimation but better handled by the DNN.

As a robustness check, I also run a ridge regression, which penalizes the coefficient magnitudes from OLS and aims to prevent overfitting. While the ridge regression performs better than the naive OLS model, it is only able to reach an MSE of 0.20 on the test set with $\lambda = 0.01$, meaning that even regularized linear models do not perform as well as the deep neural network. Accordingly, deep learning is advantageous over traditional models in this setting, but not entirely superior. First, these data suffer from multicollinearity (e.g. neighborhood is correlated with city, and number of bathrooms is correlated with total number of rooms); second, the relationship between the predictors and rental price is plausibly non-linear in parameters—both of which motivate a more complex structure. However, given the similarity in MSE for the ridge regression and DNN, one potentially faces a trade off between interpretability and computation time versus predictive accuracy and generalizeability.

Reflection

This project assessed the applicability of deep learning to a regression task as compared to OLS using data on rental prices in the Indian housing market. I find that a three-layer network including dropout layers and L2 regularization in the loss function performs significantly better to a baseline OLS regression, both in MSE and MAE after hyperparameter tuning.

Addressing data constraints and overfitting are two ways to improve this model in the future. To address data constraints, I would perform data augmentation on the existing training set, adding random noise to existing predictors to allow more passes of the data to the network. Exploring alternative (smaller) architectures with more epochs could be a way to address overfitting. Likewise, if we made multiple observations of the same rental unit over time, we

could employ a recurrent neural network to not only model current prices statically but also predict future prices.

As an approach for predicting complex, non-linear data, deep learning offers clear advantages over regression approaches in terms of its ability to model functional relationships. It nonetheless suffers in terms of interpretability, as we cannot cohesively analyze the impact of a given factor on rental price or determine which predictors most strongly determine price. Extensions of OLS such as ridge regression may be sufficiently accurate and require less computational overhead, meaning that the necessity of deep learning depends strongly on the use case.