Project 3: Neural Networks

Michal Golovanevsky
mgolovan@calpoly.edu

Markelle Kelly
mkelly23@calpoly.edu

Jeremy Whorton
jwhorton@calpoly.edu

Abstract. *Our team developed code for training neural networks as well as standard linear regression. To explore the differences between these methods, we created models predicting housing prices from a dataset on Russian housing sales. We discuss the implementation, optimization, and final performance of the two methods, including a comparison of predictive accuracy and training time. For the purposes of the Russian housing data, we choose neural networks as superior.*

Key words: *Neural networks, backpropagation, linear regression, gradient descent, gridsearch*

Introduction

The 2017 Russian Sberbank dataset consists of approximately 30,000 real estate listings, including details about each property as well as demographic and macroeconomic data related to the time and location of the sale. The goal of our project is to predict the price of each property, using both linear regression and neural networks, in order to compare the two methods.

With this goal in mind, we implemented both linear regression, using the traditional closed form solution for ß, and backpropagation for a fully-connected feed-forward neural network. After data cleaning and feature selection, we optimized both the regression model and neural network for predictive accuracy. We then evaluate and compare the two methods in terms of prediction and training time.

Dataset Creation Process

Overview

We formed our initial dataset by merging the train and macro datasets on time stamp. We then investigated each column, considering its description and hypothesizing whether it would be valuable in predicting house prices. In choosing candidates that seemed most informative, we prioritized those with fewer missing values, or missing values we could fill with a reasonable, intuitive strategy. We especially avoided columns in which more than half of the values were missing. After evaluating the columns, we created a subset of the dataset, including our 39 candidate explanatory variables[A1] and the target variable.

Handling Missing Values

While we aimed to limit missing values in our feature set, our variable selection did not eradicate them completely. We had to fill in missing values for several variables.

life_sq, full_sq: Both full_sq, which represents the full size of the property, and life_sq, which gives the size of living areas of the property, have some missing or incorrect values. In addition to true missing values, there are 440 observations with a value of 0 or 1 for living square meters or full square meters. Since these values represent actual houses and apartments, it is probably safe to assume that a value of 1 or less square meters for space is a

mistake. Of those observations, there are only 21 observations that have these unreasonably small (i.e. ≤1 meters) values for both full and living space. For the remaining 419 observations with one reasonable space value (these were found to all have values over 10 square meters), we set the value of the opposite variable to missing to later fill in with a more helpful estimate.

After this step, we found 32 observations where living square meters is greater than the full square meters. We looked at each of these individually – for some observations, full_sq seemed to have the incorrect value, while life_sq did not make sense for others. For example, one investment property was listed with a full size of five square meters, but 44 meters of living space, but another three-room property with a total size of 79 square meters was recorded as having 7,478 square meters of living space (almost 25,000 square feet!). Overall, however, full_sq seemed to be more consistent. Other than two low values of five and nine, values more or less fell between 30 to 80 square meters, while values of life_sq ranged from tens to hundreds to thousands. Noting that Russian housing is, on average, smaller than American housing – 64% of Russian families live in apartments smaller than 60 square meters[1] – we chose to retain the value of full_sq. We set the two very small full_sq values to missing, but set life_sq to missing for the remaining 30 observations.

These two variables, life_sq and full_sq, appear to have a relatively consistent relationship; the proportion of life_sq to full_sq averages 0.645 and has a standard deviation of 0.166. The correlation of the two variables is 0.442. Additionally, no observation has missing values for both full_sq and life_sq. Therefore, we decided to use these two variables to estimate missing values of each other. Null values of life_sq were filled by multiplying the corresponding value of full_sq by the average proportion of life_sq to full_sq (0.645). Similarly, missing values of full_sq were filled with life_sq / 0.645.

Finally, since only 21 observations had 0 or 1 for both variables, these were filled in with the column means.

floor, cafe_avg_price_5000: Since there are less than 300 missing values (less than one percent of observations) for number of floors and average cafe price, we chose to simply fill in missing values with the average of the column.

num_room: There are more missing values for num_room, almost one third of the dataset, so it is preferable to avoid using a simple mean to fill in. Since num_room is very likely related to size of living space, we chose to forward-fill in null values for num_room, sorting by life_sq.

state: The state variable appears to be on a scale of 1 through 4. While there is one value of 33, this seems to be a mistake, so we set this to missing. There are lots of missing values for state - a little under one half of the dataset. Because of this, it would likely hurt the prediction process to try to estimate these values. However, the condition of the house is likely very informative for the houses that do have it recorded. In order to keep this helpful information and not dilute it with estimated values, we designated a value of zero to represent an unknown state. Therefore, missing values of state were filled in with zeros.

Numeric Conversion

Several variables also needed to be converted into numeric types in order to be useful input to our regression and neural networks.

ecology: The ecology variable is ordinal, so we maintained this ordering numerically. We used one for 'poor', two for 'satisfactory', three for 'good', four for 'excellent', and zero for 'no data'.

product_type: We mapped product type to a binary numeric variable, where 1 means the property was an investment and 0 means the property was owner occupied.

nuclear_reactor_raion, big_market_raion, oil_chemistry_raion, detention_facility_raion, railroad_terminal_raion, radiation_raion: These variables' values consisted of "yes" and "no", so we mapped no to 0 and yes to 1.

Normalization

Finally, all variables were normalized with the formula (value – minimum)/(minimum – maximum). This puts all values between 0 and 1, which helps facilitate the convergence of neural networks. It also prevents predicted values from getting too large and causing overflow during the first round of backpropagation for a large network.

Linear Regression Implementation

To perform linear regression, we estimated coefficients using the traditional closed form solution for ß. This formula, which involves matrix inversion, was practical for this project because of the nature of our dataset, which was relatively small in size, did not have multicollinearity issues, and had more rows than columns. If any of these characteristics were missing, an alternative implementation such as QR-decomposition or gradient descent would have been necessary.

Neural Network Implementation

Neural Network Class

To facilitate the initialization and training of our neural networks, we wrote a NeuralNetwork class to store information about a network's input parameters and keep track of node and weight values during backpropagation. This class takes in the number of layers, the number of nodes in each layer, and the activation function for each layer. Based on this information, the NeuralNetwork class randomly initializes all weights from a uniform distribution between -0.25 and 0.25. Initializing relatively small weights helps prevent overflow by limiting the size of initial predicted values.

To perform forward propagation, the NeuralNetwork class contains a forward_prop function that takes in a matrix of input data. This is divided into arrays corresponding to each column (i.e., an array of values of the first variable that functions as the first input node, another array for the second, and so on). For each layer, forward_prop takes the input nodes for that layer, as well as an intercept node it creates, and sums these values multiplied by their respective weights for each node in the next layer. Note that the input nodes are numpy arrays, so the weight multiplication and summing is performed for all observations at once. After this, the corresponding activation function is applied to the arrays of sums and the results are stored as input for the next iteration. Upon reaching the final layer, the predicted values for each x are returned in an array.

Backpropagation and Gradient Descent

The NeuralNetwork class also contains a back_prop function, which takes advantage of the stored node values from forward propagation. The function takes the predicted y values (from forward_prop), the true y values, a learning rate, and a list of derivative functions that matches the network's activation functions. Starting at the final layer, back_prop iterates through each combination of the previous layer's node values and the current layer's delta values, multiplying these to get weight updates. Again, since this process is being performed for multiple observations simultaneously, the result for each weight is an array of different changes to be applied. For each weight, its actual change is the sum of its update array, multiplied by the learning rate. Each weight's change is subtracted from it to create a list of new weights. The back propagation function then computes the new deltas by going through the z values for the previous layer, multiplying the derivative of the activation function

at that point by each old delta and corresponding weight and summing these values. The NeuralNetwork class is then updated to reflect its new weights.

Finally, our NeuralNetwork class includes a function gradient_descent that takes in the arguments for forward_prop and back_prop, as well as batch size and tolerance parameters. For each epoch, this function divides the input data into batches of the specified size, performs forward and back propagation on each batch, and computes the mean square error. This process is repeated until the changes in MSE are smaller than the input tolerance.

Regression Optimization and Evaluation

Model Selection

To develop the most accurate linear regression model, we implemented forward-stepwise variable selection based on the minimization of mean square error. Starting with no variables in the model, the feature that lowers the MSE the most is added, and this process is continued until the MSE ceases to improve. For each variable subset, we computed the MSE with five-fold cross validation to ensure that the model actually had improved accuracy for new predictions and was not overfitting the data.

This method removed 12 of our originally chosen variables. The remaining 27 columns became our final dataset[A2] which we used for training both our regression models and our neural networks.

Evaluation

We chose to prioritize accuracy (i.e., MSE) in developing this model to have a meaningful baseline to compare with our neural network. We planned to compare the two strategies in terms of accuracy, not interpretability, so it made sense to make both as accurate as possible.

In addition to MSE, however, we also monitored R^2 values across our linear regression models. Our final model performed well based on both of these metrics, with an MSE of 0.00126 and an R^2 of 0.461.

Neural Network Optimization and Evaluation

Random Search

In order to understand the effects of different neural network parameters on its prediction accuracy, we performed random search across the number of layers, the number of nodes in each layer, and the activation functions for each layer. We used random search instead of grid search because in situations like ours where several hyperparameters need to be tuned, each with many possible values, an exhaustive grid search becomes incredibly time consuming and wasteful of resources. Although this means our final model may not be the absolute best model, random search allows us to explore the hyperparameter space and choose a solid network with good parameter values.

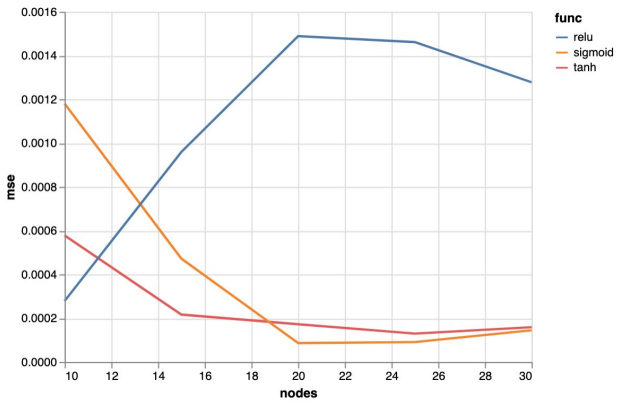
For our search, each network was assigned a random number of layers between one and five, random numbers of nodes for each layer from the options [5, 8, 10, 15, 20, 25, 30, 40, 50], and a random list of activation functions, choosing from relu, sigmoid, or tanh. After training about 75 different networks, we picked out the best models (by test set MSE) to compare them. There was quite a variety of parameter values across the best models. Most notably, we had good models across the full range of numbers of layers. Specifically, the networks with the three best MSEs had one, three, and four layers. This suggests that it is the layer quality, not quantity, that is more important in prediction. Since the simple network with one layer had comparable accuracy to our most complex networks, we chose to iterate further with the single-layer network. Choosing this simpler neural network, which uses 15 nodes with a sigmoid activation function, allowed us to train more efficiently and more clearly see the effects of parameter tuning.

In order to try as many random networks as possible, we set the convergence criteria slightly looser during this step, with a stopping condition of a change in MSE between epochs of less than $5x10^{-8}$. This was enough to give us a good idea of which parameter combinations were promising, but it meant the output models could be slightly improved. To remedy this, we re-trained our chosen model, optimizing to a stricter stopping condition. After this more intense optimization, the network converged with an MSE of 0.0000146.

For comparison, we also ran this optimization on our four-layer neural network[A3], which performed even better, reaching an MSE of 0.0000056. For the purposes of parameter tuning, however, we iterate on the simpler model.

Tuning Parameters

After selecting this model, we made some final optimizations by performing a grid search across slight changes in the numbers of nodes and activation functions. In addition to fine-tuning performance, this helped develop our understanding of the way the parameters change in the behavior and effectiveness of the model.



It appears that, for the sigmoid and tanh activation functions, increasing the numbers of nodes in a layer improves accuracy up to a certain point (around 20 nodes), then ceases to help, even becoming detrimental to the network. For the relu activation function, adding nodes seems to worsen the MSE until a much higher number of nodes (around 30). Similarly, it appears that the sigmoid activation function is the most useful activation function for larger numbers of nodes, while relu and tanh are more applicable to smaller layers.

Based on this grid search, we updated our model to use 20 nodes instead of 15, and kept the sigmoid activation function.

Choosing Learning Rate and Batch Size

Finally, to ensure optimal convergence, we performed grid search over learning rates and batch sizes. Since learning rate and batch size are used more for model tuning rather than model creation, we felt it was appropriate to tune them separately. Additionally, since there were only two parameters, with fewer potential values, we felt an exhaustive grid search was appropriate.

Only a small range of learning rates (between 0.001 and 0.005) was tested - any smaller and the network took unreasonably long to converge, any larger and the network was at risk of overflow issues. Batch sizes between 50 and 500 were tested.

batch rate	**0.001**	**0.005**
50	54	41
100	53	44
500	57	42

This chart shows the number of epochs to convergence under different learning rates and batch sizes. While batch size did not seem to make a notable difference on convergence, the slightly higher learning rate did speed it up considerably.

Evaluation

Our final neural network, using all these parameters, was then evaluated with five-fold cross validation. The network's average MSE was 0.0000453 and its R^2 was 0.642.

Results and Comparison

In terms of accuracy, while both the regression model and the neural network are useful for prediction, there is no competition. Our final neural network's MSE was less than one-tenth that of the linear regression, and its R^2 was almost 0.2 higher.

	MSE	**R^2**
Linear Regression	.00126	0.461
Neural Network	.000045	0.642

Of course, linear regression has the advantage of speed – the model itself trains nearly instantaneously, and stepwise feature selection with five-fold cross validation runs in just a couple of minutes. On the other hand, training one neural network takes a couple minutes and running a full grid search to optimize all parameters could go on for days. However, this problem is less dramatic for simple neural networks – for example, just one hidden layer of 15 nodes with a sigmoid activation function. Even this incredibly simple network achieves a better accuracy than our best linear regression within one or two epochs (less than five seconds). In less than a minute, this network achieves an MSE of 0.0007, which is a significant improvement upon our regression model. Therefore, it seems that even if time is a constraint, it is worthwhile to test out some simple neural networks in addition to linear regression.

In general, therefore, we prefer neural networks for modeling these data. They have a much higher accuracy, increasing our confidence in how closely we are fitting the data and thus in the validity of our predictions, making the increase in training time worthwhile.

Conclusion

After implementing neural networks for modeling Russian home prices, training them to maximize predictive accuracy, and comparing them to the baseline of a linear regression model, we feel we have gained a valuable understanding of neural networks, in the details of their implementation as well as their advantages and disadvantages.

If we were to move forward, we would investigate a more thorough parameter search for our network, which could potentially improve its accuracy even more. It would also be interesting to compare our results in modeling this dataset to results from other machine learning algorithms.

Bibliography

1. Georgy Manaev, "What does the typical Russian home look like?", Russia Beyond, 2018.

Appendix

A1. Data Dictionary of Original Selected Features (39)

full_sq*	total area in square meters, including loggias, balconies and other non-residential areas
life_sq*	living area in square meters, excluding loggias, balconies and other non-residential areas
floor*	for apartments, floor of the building
num_room*	number of living rooms
state*	apartment condition
product_type*	owner-occupier purchase or investment
raion_popul	Number of municipality population. district
green_zone_part	Proportion of area of greenery in the total area
indust_part	Share of industrial zones in area of the total area
children_preschool	Number of pre-school age population
children_school	Population of school-age children
healthcare_centers_raion	Number of healthcare centers in district
university_top_20_raion	Number of higher education institutions in the top ten ranking of the Federal rank
culture_objects_top_25_raion	Number of objects of cultural heritage
shopping_centers_raion	Number of malls and shopping centres in district
oil_chemistry_raion*	Presence of dirty industries
radiation_raion*	Presence of radioactive waste disposal
railroad_terminal_raion*	Presence of the railroad terminal in district
big_market_raion*	Presence of large grocery / wholesale markets
nuclear_reactor_raion*	Presence of existing nuclear reactors
detention_facility_raion*	Presence of detention centers, prisons
work_all	Working-age population

ekder_all	Population older than working age
park_km	Distance to park
public_transport_station_km	Distance to the public transport station (walk)
big_road1_km	Distance to Nearest major road
fitness_km	Distance to fitness
big_church_count_5000	The number of big churches in 5000 meters zone
mosque_count_5000	The number of mosques in 5000 meters zone
cafe_avg_price_5000*	Cafes and restaurant average bill in 5000 meters zone
office_count_5000	The number of office space in 5000 meters zone
ecology*	Ecological zone where the house is located
salary	Average monthly salary
cpi	Inflation - Consumer Price Index Growth
usdrub	Ruble/USD exchange rate
mortgage_rate	Weighted average rate of mortgage loans
unemployment	Unemployment rate
bandwidth_sports	Capacity of sports facilities
rent_price_2room_eco	rent price for 2-room apartment, economy class
price_doc	sale price (this is the target variable)

*Corresponds to variables that were modified or filled. Please see above to understand transformations or how missing values were handled.

A2: Final Dataset Dictionary (27)

life_sq*	living area in square meters, excluding loggias, balconies and other non-residential areas
floor*	for apartments, floor of the building
num_room*	number of living rooms
product_type*	owner-occupier purchase or investment
raion_popul	Number of municipality population. district

green_zone_part	Proportion of area of greenery in the total area
indust_part	Share of industrial zones in area of the total area
children_preschool	Number of pre-school age population
children_school	Population of school-age children
healthcare_centers_raion	Number of healthcare centers in district
university_top_20_raion	Number of higher education institutions in the top ten ranking of the Federal rank
culture_objects_top_25_raion	Number of objects of cultural heritage
railroad_terminal_raion*	Presence of the railroad terminal in district
big_market_raion*	Presence of large grocery / wholesale markets
nuclear_reactor_raion*	Presence of existing nuclear reactors
work_all	Working-age population
ekder_all	Population older than working age
park_km	Distance to park
big_road1_km	Distance to Nearest major road
fitness_km	Distance to fitness
big_church_count_5000	The number of big churches in 5000 meters zone
mosque_count_5000	The number of mosques in 5000 meters zone
office_count_5000	The number of office space in 5000 meters zone
ecology*	Ecological zone where the house is located
cpi	Inflation - Consumer Price Index Growth
mortgage_rate	Weighted average rate of mortgage loans
bandwidth_sports	Capacity of sports facilities
price_doc	sale price (this is the target variable)

*Corresponds to variables that were modified or filled. Please see above to understand transformations or how missing values were handled.

A3: Three best models from random search

1. {'nlayers': 3, 'nnodes': [24, 8, 20, 25, 1], 'activations': [<function relu at 0x7f321d41f400>, <function tanh at 0x7f321d42a048>, <function relu at 0x7f321d41f400>, <function sigmoid at 0x7f321d41fea0>, <function relu at 0x7f321d41f400>]}
2. {'model': {'nlayers': 4, 'nnodes': [24, 40, 5, 10, 15, 1], 'activations': [<function relu at 0x7f321d41f400>, <function tanh at 0x7f321d42a048>, <function sigmoid at 0x7f321d41fea0>, <function relu at 0x7f321d41f400>, <function sigmoid at 0x7f321d41fea0>, <function relu at 0x7f321d41f400>]}
3. {'model': {'nlayers': 1, 'nnodes': [24, 15, 1], 'activations': [<function relu at 0x7f321d41f400>, <function sigmoid at 0x7f321d41fea0>, <function relu at 0x7f321d41f400>]}