Developing a Neural Network for Regression Problem Set:
Predicting Housing Prices in King County, USA

Dataset: https://www.kaggle.com/datasets/harlfoxem/housesalesprediction/code
Code file: CA2-HousingNN.ipynb

Michael Ross (SID: 201589412)
Shuhei Ishiwatari (SID: 201603195)
University of Liverpool
COMP 534: Applied AI
Assignment 2
8 April 2022

Background

The accompanying dataset was provided by Kaggle and provides over 23,000 instances of house sales in King County of Washington, USA. This dataset provides features that uniquely describe a particular house and serves as factors that contributed to the price a house was sold for. The aim of this project is to design a basic neural network to predict, as closely as possible, the sale price of the houses in the dataset.

Libraries Used:

Data Pre-processing

- Pandas
- Numpy
- Sklearn - StandardScaler

Hyperparameter Tuning

- Talos
- Sklearn - mean_squred_error

Modeling

- Tensorflow/Keras
- Sklearn - LinearRegression
- Sklearn - train_test_split\
- Matplotlib

Data Processing

Our initial observation of the dataset was that there were several features that were either categorical or had to be interpreted in a different way that would make sense for the neural network to understand. In other words, we had to adapt the dataset in a way that provided some context for our neural network. Our group decided to implement one-hot encoding on categorical features such as waterfront view, view, and zipcode.

Next, we observed that the 'yr_renovated' feature had many instances of a 0, which is supposed to indicate that the house had never been renovated, but we determined that this would be interpreted as year 0 for our model. To fix this, we changed every instance of a 0 in this feature to the year the house was built. We then replaced 'yr_built' and 'yr_renovated' to the difference between these years and the year of the sale. If a house had never been renovated, the value in these two new columns would be the same.

Finally, after dropping irrelevant features, we scaled the dataset using sklearn's StandardScaler as the mean of each feature varied significantly. We also experimented with the MinMaxScaler, but observed a noticeable improvement in our models using the StandardScaler. For training, we used an 80:20 train/test split which is considered a very common split for machine learning purposes.

Justification for dropped features

'id' - this is a unique, categorical, identifier that will not impact price

'date' - date has been reformatted to days since sale

'lat/long' - US zipcodes work in a similar fashion as clustering. Two neighboring communities located in the same city, but of vastly different living standards will be identified with different zipcodes, but share nearly identical grid coordinates. Zipcode should be used as a better indicator of house valuation in relation to location over latitude/longitude.

Hyperparameter Testing

One of the many challenges in implementing even the most basic of neural networks is that there are many hyperparameters that need to be tested to output the best results for the model. This presents two primary challenges: 1. There are only general guidelines, or recommendations, to determine a bespoke set of hyperparameters for a given model that usually involves testing different combinations of parameters to observe the impact they have on a model's performance. 2. Testing hyperparameters requires a significant amount of time and resources (very demanding of CPU/GPU).

Table 1. Hyperparameters of NN to test in the current experiment.

Hyperparameter	Meaning	Testing values
learning_rate	It defines the step size at each iteration of the gradient descent. If the value is too large, it can overshoot the minimum of a cost function. Otherwise the learning can be slow.	[0.1, 0.01]
num_neurons	The number of neurons in each hidden layer.	[1, 8 ,55]
hidden_layers	The number of the hidden layers in our neutral network	[1, 3]
batch_size	The number of training objects that will propagated through neutral network at each iteration	[10, 100]
epochs	It determines how many times all the training objects will be propagated through the neutral network.	[10, 30]
optimizer	Is a function where the objective is to update weights (and learning rate) in such a way that it reduces the overall cost.	[Adam, RMSprop]
loss	Loss/cost function. It determines an error (difference between the predicted and actual values). An optimizer updates the weights and bias in such a way that the loss function is minimized.	[mean_squared_error]
activation	It determines how a linear combination of weights and inputs is transformed into an output. This adds non-linearity to the neutral network.	[sigmoid, relu]

<u>Strategy for testing</u>

The table 1 shows the hyperparameters of our interest and concrete values of each parameter to be tested in the current experiment. To observe the impact of a parameter, all the other parameters need to be fixed while testing multiple values of the parameter. That is, every possible combination of the values are examined. From the current values of each parameter, all the combinations are given by:

$$3 \times 2^6 \times 1 = 192$$

192 neutral networks need to be created to test the impact of each parameter in the dataset. To achieve this, we have utilized *talos* which is a hyperparameter optimization library for Keras and TensorFlow. Talos locks other parameters while testing multiple values of a parameter. For example:

Figure 1. An example of hyperparameter tuning with talos.

	loss	val_loss	activation	batch_size	epochs	neurons	hidden_layers	loss	lr	optimizer
0	0.3	0.3	sigmoid	10	10	4	1	MSE	0.1	Adam
1	0.4	0.4	sigmoid	10	10	4	1	MSE	0.1	RMSprop

As shown in Figure 1, tuning of two optimizers (Adam and RMSprop) with talos was done while all the other parameters remained the same. This way, we can truly investigate how changes in each parameter affect the final result.

Evaluation

Table 2 and 3 provide a summary of the best and worst 3 neutral networks out of 192 models that have been created in the first step of our experiment. The training and testing MSE are subject to 5-fold cross validation in the final step of the experiment.

Table 2. Top 3 NN architectures that we obtained in the current experiment.

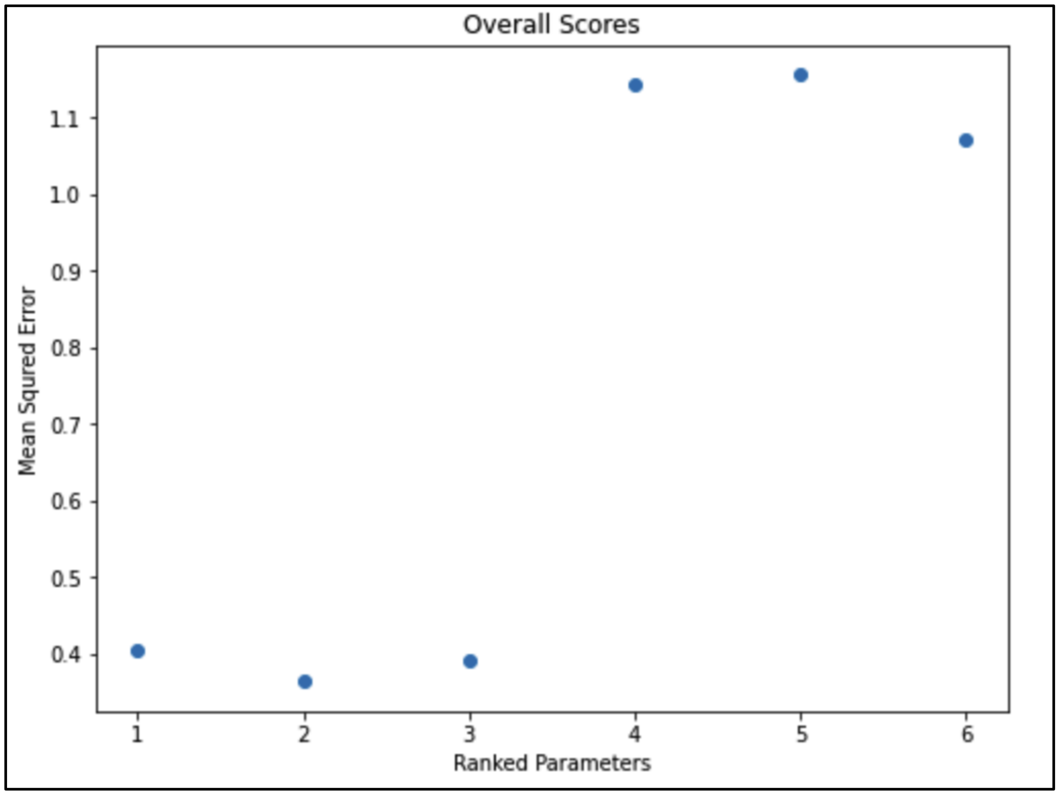
Model	Top 3 Neutral Network Architecture										
	Hidden Layers	Number of Neurons		Activation Function		Batch Size	Epoch	Optimizer	Learning Rate	Training MSE	Testing MSE
		Input	Hidden Layers	Hidden Layers	Output Layer						
Model 1	1	92	55	Relu	Linear	10	30	Adam	0.1	0.0915	0.1135
Model 2	3	92	55	Relu	Linear	10	30	Adam	0.1	0.0912	0.1147
Model 3	3	92	55	Relu	Linear	10	30	RMSprop	0.1	0.1110	0.1182
Note: activation function in the output layer is fixed (linear function) since this is a regression problem. Mean squared error was selected as the cost function. Training & testing MSE rounded to 4 decimal places.											

Table 3. Bottom 3 NN architectures that we obtained in the current experiment.

Model	Bottom 3 Neutral Network Architecture										
	Number of Hidden Layers	Number of Neurons		Activation Function		Batch Size	Epoch	Optimizer	Learning Rate	Training MSE	Testing MSE
		Input	Hidden Layers	Hidden Layers	Output Layer						
Model 1	1	92	8	Sigmoid	Linear	10	10	Adam	0.001	3.2334	3.1330

Model 2	1	92	8	Sigmoid	Linear	10	10	Adam	0.001	3.1442	3.0524
Model 3	3	92	8	Sigmoid	Linear	100	30	Adam	0.001	2.5023	2.4170
Note: activation function in the output layer is fixed (linear function) since this is a regression problem. Mean squared error was selected as the cost function. Training & testing MSE rounded to 4 decimal											

Figure 2. Comparison of results using different parameter combinations, where 1-3 are the best parameters as defined in Table 2 and 4-6 are the worst performing defined in Table 3



Linear Regression Results:
Model score: 0.8203610411159351
MSE: 0.28107077318474344

Analysis of Results

Keeping in mind the limited amount of initial hyperparameters selected to train in our model, our decision to use Talos to output the best combinations of hyperparameters paid off. As shown in Figure 2, there is a clear distinction in mean squared error scores between the three best performing hyperparameter combinations and the three worst combinations.

Immediately, we are able to observe that the performance of our network is highly influenced by the activation function selected as Relu consistently outperformed the Sigmoid activation function. It is speculated that the difference in the performance stemmed from the underlying mathematical mechanisms of two functions.

Given that in the backpropagation algorithm, weights are updated as follows:

$$W^L \leftarrow W^L - \alpha \frac{\partial J}{\partial W^L} \text{ where } \frac{\partial J}{\partial W^L} = \sigma(Z^{L-1}) \cdot (\delta^L)^T$$

Here Z^{L-1} is a linear combination of weights and inputs in the previous layer (L-2) whereas $\sigma(x)$ represented the selected activation function. Given that δ^L is given by

$$\delta^L = \sigma'(Z^L) \cdot W^{L+1}\delta^{L+1}$$

The value of δ^L depends on the derivative of the activation function $\sigma'(Z^L)$. In the case of sigmoid function, its derivative is $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$, where the value is some fraction between 0 and 1. This could make an overall gradient smaller, leading to the vanishing gradient problem, making marginal changes in the weights in the lower layer. However, the derivative of ReLu function is $\sigma'(x) = \{0 \; if \; x < 0, 1 \; if \; x > 0\}$. Since the gradient contains multiple products of 1's, the gradient descent does not suffer from the vanishing gradient. This simple functionality of ReLu could explain its better performance in the current experiment.

Additionally, we observed that the best optimizer could not clearly be defined. Though our charts demonstrate that Adam was the best optimizer, Adam also performed the worst when other hyperparameters were altered. Mainly, we theorize that this could be the result of the learning rates selected as RMSprop and Adam take different approaches in minimizing the loss function.

Both RMSprop and Adam have the potential of convergence at a much faster rate than standard gradient descent. However, RMSprop, as an extension of gradient descent, uses a decaying average of partial gradients to adapt the learning rate automatically by using the most recently observed partial gradients. When observing the top three performing hyperparameter combinations, we see that RMSprop did in fact perform almost as well as Adam and could be because the initial learning rate, or step size, of 0.1 in congruence with a greater number of epochs lead to a better minima, thus reducing our error rate.

With the Adam optimizer, we combine RMSprop with the weighted average of momentum, in which momentum subtracts the exponential decaying average of gradients. In theory, Adam could converge faster than RMSprop, but we also acknowledge that many factors contribute to the validity of this statement. Our group theorizes that both RMSprop and Adam could work in creating our neural network, but that we would need to test additional hyperparameters, such as greater epochs and step sizes to distinguish between the two.

Finally, with the current set of hyperparameters tested in our model, some additional patterns we observed is that increasing the number of neurons in our hidden layers and epochs resulted in better performance, while smaller step sizes did not. Again, we clearly see the correlation between optimizers selected and the number of epochs/learning rates tested, but can not clearly define the best relationship until further testing of additional hyperparameters are conducted.

Final Conclusions

This assignment was an incredible opportunity for our group to implement a neural network for the very first time and develop a deeper appreciation of the complexities of neural networks. Although we were successful in creating our model, we acknowledge that there are limitations in the structure of our network, mainly in our selection of hyperparameters and the inability to freely test a more robust set of hyperparameters due to the time it takes to test them.

Our group had no option but to use Google Colab as our primary IDE for this project in order to take advantage of greater computing resources, but also due to the fact that deep learning libraries such as Tensorflow have issues running properly on the newer Apple Macbook M1 models. Though we had no issues using these libraries on Colab, we often encountered issues with our Colab sessions timing out during the midst of our hyperparameter tuning using Talos, which forced our group to reduce the parameters tested.

Despite the challenges of this project, we do take away the fact that neural networks do have the potential to give better predictions in regression models in datasets with the depth that we used in this project. With over 23,000 records, even our basic neural network structure outperformed a linear regression model as shown below figure 2. We also understand the difficulties in choosing the best parameters is the exact reason why there is no set-in-stone set of rules for creating neural networks. Still, our group finished this project with a great level of enthusiasm for developing our skills and gaining a deeper understanding of deep learning moving forward.

Project Allocation

	Michael Ross	Shuhei Ishiwatari
Data Cleaning	x	x
Feature Selection	x	x
Hyperparameter Selection	x	x
Hyperparameter Tuning - Talos		x
Hyperparameter Testing	x	x
Neural Network Design	x	x
Linear Regression Model	x	
Plotting Results	x	
Tables for report		x
Report Writing	x	x