House Price Prediction Using Feed Forward Neural Network with Hyper Parameter Tuning.

Felix Emmanuel 202122699

Course – Applied Artificial Intelligence - 771767_A21_T2:

University of Hull.

Introduction and aim

This project is aimed at analyzing data with feedforward neural network and experiment systematically with different hyperparameter configurations, e.g. the learning rate, batch size, number of hidden units, layers, etc.

Over the past few years there has been rising waves of interests in Neural Networks and their applications. More of the modern algorithms that are now being deployed are built based on the existing foundation created in the 80s and up to the 90s. Some of these work are centered around using techniques to create performance optimization for the networks without sacrificing the quality or standard of the algorithm to predict on untested dataset. The techniques can be used to focus on a certain aspect of the network or apply globally to an array of network

"There has been significant research that attempts to find techniques to tune hyperparameters such as the number of layers, number of neurons and activation functions etc. These techniques include grid search, random search and Bayesian optimization-based methods" (Farooq, 2018)

Background

Real estate businesses has expanded a lot into virtual space where artificial intelligence is now being incorporated into the sales of properties.

Although most real estate information is available to the public, sifting through public property records to find title documents, purchase prices, and even mortgage liens can be an arduous task. (GOFUS, 2022)

The Project

This project is about building an algorithm that can predict house prices for a real estate company. The dataset was adapted from Kaggle. It was for a competition tagged Zillow's Home Value Prediction (Zestimate).

The Zestimate is an algorithm already built by the real estate agency that makes prediction about prices of homes in the United State by analyzing property data but they wanted data scientist to build an algorithm that will beat their existing prediction model.

The Zestimate uses an algorithm that which runs unique evaluation models based on the information about a property such as the tax assessment, the previous and current transactions to calculate appraisal of a property. (Zillow, 2018)

Zillow Prize was a competition with a $1,000,000 grand prize. The real estate agency challenged the data science community to "help push the accuracy of the Zestimate even further".

I decided to take this project and test it for a classification problem, that is I will be building algorithms that will predict if the price of a property is of average or median price value or not. This will help people when it comes to fitting in a budget to a property. The features I used for this project has reduced input features and the task varied into predicting if a house price stays above or below median value. This is to distinguish it from the work already done in the competition.

To achieve this I will be using feedforward neural network with hyperparameter tuning and also compare it with other algorithms such as Random Forest, Decision Tree, Linear Regression and SVM.

Methodology / Technical scope

Data Pre-processing

This part I had to load in my data into my Jupyter notebook, perform some exploratory data analysis (EDA) which will be explained in the next section of this literature.

Methodology

The aims of my research is to firstly training my algorithms to conduct a classification task, carry out hyperparameter tuning and then comparing my results with that of classification algorithms.

The performance of a network is hugely dependent on the network architecture and its model capability. (Farooq, 2018)

Layers in Neural network impacts on the outcome of a prediction model. For a simpler problem, it is sufficient to have smaller numbers of layers meanwhile for complicated problems larger number of layers are needed.

Inserting regularization layers in a neural network can help prevent overfitting. (Rendy, 2021)

Batch normalization layer – This normalizes the values that passes through it for each batch. This is like using standard scaler in conventional machine learning

The dropout layer drops at random certain number of neurons from a layer. Which means that the dropped neurons are no longer used. The value of the or percentage of neurons that needs to be dropped is set in the dropout rate

Epoch - This is the number of times a dataset is run through a neural network

One epoch simply means that a dataset has been passed through the neural network once. If the epoch size is too small this will lead to underfitting of the dataset because the neural network wouldn't have learned enough. Conversely, if the number of epochs is too many this leads to overfitting because the model then can predict the data too well but will not be able to predict new unseen data good enough. Hence it is good to tune the number of epochs to gain the best result

We will be performing a grid search for model. Grid search is used for hyperparameter optimization.

This technique is available in scikit-learn in the GridSearchCV class. In order to construct this class, we will build a dictionary for the hyperparameters to evaluate from the param_grid argument. This maps the model parameter names to the array to try. This gives us the score that is optimized as the accuracy. (Brownlee, 2016)

When this the outcome of the grid search can be accessed from the result from grid.fit(). Using the best score and the best parameter members we get the best score observed during the gridsearch and the parameters that combined to achieve this best results.

Critical Evaluation

Exploring and Processing the Data

The first thing was to read in the data i.e my CSV file. Then convert them to arrays.

The dataset was then split into the input features i.e X and the label features Y.

Then the data was scaled (normalized) this was to let input features both have the same orders of magnitude after which the dataset was split into training, validation and test sets respectively.

The feature that we would like to predict is in the last column of the dataset. i.e this classifies whether the price is above the average or not (1 – YES, above and 0 – NO, below average price)

In order to train our dataset, I split the dataset into input feature (X) and the feature to predict (Y).

After splitting the dataset, the next step is to make the scale similar. For instance some features like lot area are in thousands, while the score for overall quality are in units and tens, whereas the number of fireplaces are just numbers between 0, 1 or 2.

Why this is a problem is that it makes it hard for the neural network to initialize. In order to resolve this, I scaled the data and this was done by using the scikit-learn package.

Having scaled the dataset, the next step was to split the dataset into three i.e. training set, validation set and a test set. In splitting my dataset I imported the code from scikit-learn namely ‘train_test_split’. This as the name implies, helps split the dataset into a training set and a test set.

What my code was doing was to let scikit-learn know that it should use 30% of my overall dataset for my val_and_test size. Hence, after splitting I now have six variables namely X train, Y train, X val, Y val, X test and Y test in all for my dataset, which are listed

Building my architectural models

I used the Sequential model. Sequential model basically implies that the layers are described in sequence

First I imported the necessary codes from Keras, built the first model with my first and second hidden layers having Thirty-two neurons each while the third (output) layer had only one neuron.

Therefore to configure the model we use the compile function. For my compiler the hyperparameters used are “Adam” for my optimizer and “Binary cross entropy” for my loss function, this is because our dataset is basically numerical.

At this point training only requires that it fits the code to the model using the fit function. This is to fit the parameters to the data.

So next was to set the size of the batch and number of times it has to train (i.e epochs). I set what my validation data would be in order for the model to tell me on how my structure is performing on the validation data.

Then next is to build the history

After plotting the visualization for loss and accuracy the plot showed that for our second model, it has over-fitted, the training loss was decreasing and the validation loss was increasing. Also plotting the accuracy shows a wide variance between the train and the validation accuracy.

In order to then reduce the overfitting the following steps were taken,

L2 regularization was added, this was done by adding an additional code to each one of our dense layers with the variable kernel regularizer. What this does is to square the values of the parameters of the loss function, then weigh them by 1 percent.

A new dropout layer of 0.3 was also added, this implies that the neurons from previous layer will be dropped by a probability of 0.3 in training.

Then I ran it with the same parameters as I did for the second Model (i.e. the over fitted model).

When this was done and compared with the model_2, we could see from the graph that the techniques have reduced the overfitting of the training set.

Hyperparameter tuning

The next phase was to carry out hyperparameter tuning for our dataset, to see how the different hyperparameters will help our network yield the best result.

I adopted the grid search system for my hyperparameter tuning. This involves defining the hyperparameter tuning for the dataset, doing a grid search for the models hyperparmeters (such as batch size, epochs, learning rate, number of neurons, dropout rate and so on).

The following set of Hyperparameter tuning was carried out and the corresponding outcome for the various experiments extracted from my jupyter notebook computed.

Tuning Batch Size and Epoch

Tuning the batch size exposes the network to patterns before the weights get updated. This also optimizes the network training in the sense that it tells it the amount of pattern to read and store in its memory per time.
The number of epochs means the number times the training dataset was ran through the network while it is training. For this experiment we tried batch sizes in multiples 5 and 10.
Below are the results of the experiments carried out

Experiment 1

Dense Layer = 12

Input Dimension = 8

Batch size value = 10, 20, 40, 60, 80 100

Epoch = 10, 50, 100

Best result = 0.5198 with batch size of 20 and 100 epochs

Experiment 2

Dense Layer = 12

Input Dimension = 8

Batch size value = 5, 10, 15, 20, 25, 30

Epoch = 10, 40, 150

Best result = 0.5548 with batch size of 15 and 150 epochs

Experiment 3

Dense Layer = 12

Input Dimension = 10

Batch size value = 5, 10, 15, 20, 25, 30

Epoch = 10, 40, 150

Best result = 0.8336 with batch size of 5 and 150 epochs

Tuning the Optimizer

I used the Keras suites to tune the optimizer of the model used in training. The following optimizers suite were tested Adagrad, SGD, Adadelta, Adamax, RMSprop, Adam and Nadam.

Experiment 1

Dense Layer = 12

Input Dimension = 8

Best result = Adamax with 0.5253

Experiment 2

Dense Layer = 12

Input Dimension = 10

Best result = Adamax with 0.7493

Tuning Learning Rates and Momentum

Tuning my learning rates I tried with small learning rates and momentum. Learning rates helps control the amount of update on the weight after each batch, while the momentum controls the amount of influence the previous update will have on the current weight.

Experiment 1
Learning rate function = 0.01
Momentum = 0
Dense Layer = 12
Input dimension = 8
Learn rate values for grid search = 0.001, 0.01, 0.1, 0.2, 0.3
Momentum values for grid search = 0.0, 0.2, 0.4, 0.6, 0.8, 0.9
Best result = 0.4623 (46%) with learning rate of 0.3 and momentum of 0.9

Experiment 2
Learning rate function = 0.01
Momentum = 0
Dense Layer = 12
Input dimension = 10
Learn rate values for grid search = 0.001, 0.01, 0.1, 0.2, 0.3
Momentum values for grid search = 0.0, 0.2, 0.4, 0.6, 0.8, 0.9
Best result = 0.5157 (51%) with learning rate of 0.3 and momentum of 0.8

Experiment 3
Learning rate function = 0.01
Momentum = 0
Dense Layer = 12
Input dimension = 10
Learn rate values for grid search = 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5
Momentum values for grid search = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
Best result = 0.5143 (51%) with learning rate of 0.2 and momentum of 0.3

Experiment 4
Learning rate function = 0.03
Momentum = 0.04

Dense Layer = 12
Input dimension = 10
Learn rate values for grid search = 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5
Momentum values for grid search = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
Best result = 0.5157 (52%) with learning rate of 0.3 and momentum of 0.3

Experiment 5
Learning rate function = 0.03
Momentum = 0.2
Dense Layer = 15
Input dimension = 10
Learn rate values for grid search = 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5
Momentum values for grid search = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
Best result = 0.5144 (51%) with learning rate of 0.3 and momentum of 0.1

Tuning Neuron Activation

Under activation function I evaluated the different activations from Keras. Activation are used to control the non-linearity of each neurons in the neural network. The activation suites tested are relu, Softsigntanh, Softmax, hard sigmoid, linear, Softplus and Sigmoid

Experiment 1
Dense Layer = 12
Input dimension = 8

Best Result = 0.5226 (52%)

Experiment 2
Dense Layer = 12
Input dimension = 10

Best Result = 0.7863 (78%)

Tuning Dropout rate for regularization

In this experiment I evaluated the effect of tuning the dropout rate for regulation. Like earlier mentioned dropout and regularization helps to limit overfitting of our data which improves our model's performance. A dropout percentages between 0.0 and 0.9 was used for this experiment

Experiment 1
Dense Layer = 12
Input dimension = 8
Dropout rate = 0.0
Weight constraint values = 1, 2,3,4,5
Dropout Values = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
Best result = 0.5472 (55%) with dropout rate of 0.5 and weight constraint of 3

Experiment 2
Dense Layer = 12
Input dimension = 10
Dropout rate = 0.0
Weight constraint values = 1, 2,3,4,5
Dropout Values = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
Best result = 0.7931 (79%) with dropout rate of 0.2 and weight constraint of 4

Tuning Hidden Neurons

Generally the number of neurons in a layer controls the representational capacity of the network. (Brownlee, 2016)

Under this experiment I tuned the number of neurons in a single layer. Values tested ranged from 1 to 30 in steps of 5.

Experiment 1
Dense = Neurons
Input dimension = 8
Neurons = 1, 5, 10, 15, 20, 25, 30
Best result = 0.5329 (53%) with neuron 1

Experiment 2
Dense = Neurons

Input dimension = 10

Neurons = 1, 5, 10, 15, 20, 25, 30

Best result = 0.7870 (79%) with neuron 15

Experiment 3

Dense = Neurons

Dropout added to Neurons = 0.2 and 0.1

Input dimension = 10

Neurons = 1, 5, 10, 15, 20, 25, 30

Best result = 0.8021 (80%) with neuron 15

Comparing my neural network performance against other algorithms

1 Random Forest

For my Random forest algorithm, I imported the Random forest classifier, for the forest parametes, the n- estimator values were from 500 to 1,000 while the max depth values are 10, 20, 40, 70. Also grid search was carried out on algorithm.

After running the model, my best result was 0.9178 (92%) with max depth of 10 and estimator of 500.

2 Decision Tree

For my Random forest algorithm, I imported the Decision Tree classifier from the sklearn library.

Carried out a grid search for optimization of the model, for my parameters, my max tree values 10, 20, 50 and 70 while my sample leaf values are 2 , 4 and 8.

My best result was 0.9178 (92%) with maximum depth of 70 and minimum leaf of 8.

3 Support Vector Machines (SVM)

My SVM was a simple algorithm. I imported SVC (Support vector classifier) from Sklearn.

Then this model was fitted to my training dataset.

The value of Classifier used was 100.

My model came out with a prediction of 0.9223 (92%).

Conclusion :

After testing all the models and all the hyperparameter tuning my model yielded the best result with the SVM which gave an accuracy score of 0.9223.

References

Brownlee, J., 2016. *Machine Learning Mastery.* [Online]
Available at: https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
[Accessed 18 April 2022].

Farooq, A., 2018. Hyperparameter Optimization on a Feed-Forward Neural Network using. p. 1.

GOFUS, A. E., 2022. *Worldwide Erc.* [Online]
Available at: https://www.worldwideerc.org/news/technology/artificial-intelligence-is-transforming-the-real-estate-market#:~:text=Artificial%20intelligence%20helps%20real%20estate,forefront%20with%20its%20Zestimate%20method.
[Accessed 24 April 2022].

Rendy, 2021. *Analytics Vidhya.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/
[Accessed 19 April 2022].

Zillow, 2018. *Kaggle.* [Online]
Available at: https://www.kaggle.com/competitions/zillow-prize-1/overview
[Accessed March 2022].