One dimensional convolution based neural network to predict median house value

Tejas Wadiwala
Department of Computer Science
Lakehead University
Thunder Bay, Canada
twadiwal@lakeheadu.ca

***Abstract*—In this paper, the main aim is to propose a one-dimensional convolutional neural network to predict the median house value using nonlinear regression. A model is created by defining the input layers and the connected convolutional layers. There are hyperparameters also defined, which also play an essential role in the outcome. Various experiments have been performed and got different outputs from the various experiments performed. These experiments have been performed by changing the layers of the model, also by changing the hyperparameters. The architecture of the CNN model is defined using the PyTorch libraries. The execution environment i.e., the training and testing of the model is done using Google Colab. For nonlinear regression, ReLU (rectified linear unit) activation function has been applied to different layers. The dataset used in for the above experiments is the modified version of the California Housing Dataset. The more the R^2 Score the better the model. After performing, nonlinear regression on different models, the model that is proposed gave an R^2 Score of 0.7808. This is the maximum R^2 Score that I could attain in this model.**

***Index Terms*—ReLU, epochs, adam, adadelta, learning rate, hyperparameters, dataset, batch size, convolution, flatten, normalization**

I. Introduction

Regression analysis is a set of statistical methods in statistical modeling which is used for measuring the relationships between a dependent variable (which is often known as the outcome variable) and one or more independent variables (sometimes called as features). There are many types of regression analysis methods, of which we are most interested in is linear and nonlinear regression. Linear regression may refer to a regression model that is entirely made up of linear variables. The most straightforward case of linear regression is a single variable linear regression. A single variable regression is a technique that is used to model the relationship between a single input independent variable (feature variable) and the output dependent variable using a linear model, i.e., a line [1]. Nonlinear regression is a type of regression analysis in which the observational data are modeled by a function that is a nonlinear aggregation of the model parameters and depends on one or more independent variables. The data are adjusted by a method of successive approximations.

II. Literature Review

For discovering patterns in big data that would lead to some actionable insights, there various different kinds of algorithms used. On a higher level, it can be classified into two groups on the way they learn the data in order to make prediction, they are supervised and unsupervised. In a supervised learning algorithm, there is an input variable(s) and an output variable and it uses an algorithm to learn the mapping, so that when a new data is provided it can predict its output. It requires data to be labeled, i.e., data should be labeled with correct answers. Supervised learning can also be classified further into regression and classification problems. The goal of both of them is predict the outcome, which is numerical in case of regression and categorical in case of classification. Regression is used if the output which is expected is a real or a continuous value, like salary or weight. There are various models that can be used of which the simplest one is the linear regression. It tries to fit the data through a hyperplane that goes through the points [4].

III. Dataset

The dataset which I am using for nonlinear regression is the modified version of the California Housing dataset. It has the following attributes: longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, median_house_value, and ocean_proximity. The main aim of this analysis is to predict median_house_value using nonlinear regression. I trained the dataset using the attributes longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, and median_income. The output variable is median_house_value.

I have performed pre-processing by removing all the NA values from the dataset. Also, while training the dataset, I have not included ocean_proximity because it has zero to none impact on the output. Further, the data has been split into training and testing; it is done by using the train_test_split package of the sklearn.model_selection library. It is split into 70 percent and 30 percent, i.e., 70 percent data in the training data segment and 30 percent data in the testing data segment. Now, both the data (i.e., training and testing) are converted into a NumPy array using training and testing, the benefit of converting it into a NumPy array is that the data then becomes easy for manipulation. Also, I have set the random_state to 2003.

Fig. 1. below shows each feature of the dataset on separate sub-plots i.e., sub-plots the first eighteen samples from the dataset are shown in a single figure.

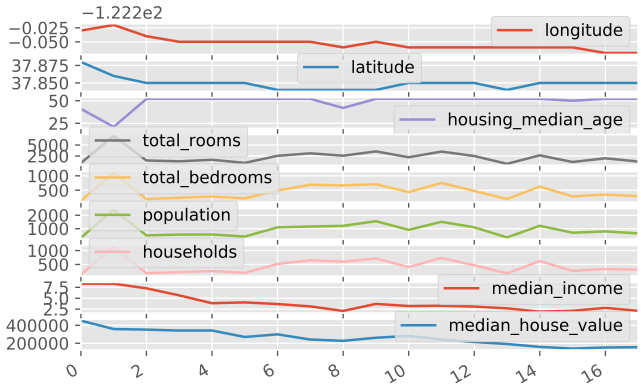


Fig. 1. Sub-plots of first eighteen samples from the dataset in a single figure

IV. Proposed Model

The model which I have used while doing this experiment has many different layers. The first layer is the batch normalization layer. Batch normalization is used to normalize the input layer by adjusting and scaling the activations. It is a technique that can be used for improving the stability, performance, and speed of artificial neural networks. The second layer is the conv1d layer, which is a part of the torch.nn package. It applies a 1D convolution over an input signal composed of several input planes. The next two layers which I have used in my model is the average pooling layer. A pooling layer merges the data present and reduces the size as well. Average pooling layer is a layer that applies a 1D average pooling over an input signal composed of several input planes [2]. The next layer is conv1d. The primary purpose of the convolution and pooling layers is feature extraction. After this, I have used the flatten layer whose primary goal is to convert the data into a 1-dimensional array for inputting it to the next layer [3]. After the flatten layer, I have applied three linear layers to my model connected to the final output layer. Linear layer applies a linear transformation to the incoming data. The output would contain the median house value.

Fig. 2. below shows the proposed model summary

For defining the model, the function *CnnRegressor* is used. It should be a subclass of torch.nn module. First, define the initialization method *__init__* which then calls the superclass to store the parameters, which is followed by the definition of the input layer. Then, *feed* whose main function is to feed the inputs through a model. The actual code snippet is given in Appendix A.

With the proposed model, the main aim is to find out the mean squared error and r^2 score for our model.

The Mean Squared Error is calculated using the following formula:

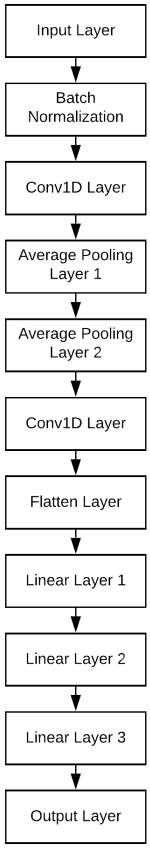


Fig. 2. Model Summary

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y})^2 \tag{1}$$

The R^2Score is calculated using the following formula:

$$R - squared = 1 - \frac{First\ sum\ of\ errors}{Second\ sum\ of\ errors} \tag{2}$$

V. Experimental Analysis

While working with the code, several different combinations of methods were applied to it. These methods mean changing different parameters in the code, which include: optimizer, number of input layers, parameters of input layers, batch_size, learning rate, and more. The parameters mentioned above are also known as hyperparameters. I experimented by changing a lot of different hyperparameters, like, number of input layers, parameters of the input layer, batch size, learning rate, and more. The most impact my model had was when I changed the learning rate, batch size, epochs, and optimizer. In all of the experiments performed, epochs were set to 100. The actual code snippet of defining the model is shown in Appendix B.

Table I below shows the changes in the R^2 Score with respect to learning rate and batch size. R^2 Score obtained in the above table was when the optimizer was set to Adam.

TABLE I
EXPERIMENTAL ANALYSIS USING DIFFERENT HYPERPARAMETERS WHEN OPTIMIZER IS ADAM

Learning Rate	Batch Size	R^2 Score
0.0007	256	0.7341
0.007	512	0.7508
0.005	512	0.7808
0.005	256	0.7570
0.005	128	0.7025
0.005	90	0.7468
0.003	512	0.7581
0.003	128	0.7744
0.001	64	0.6880

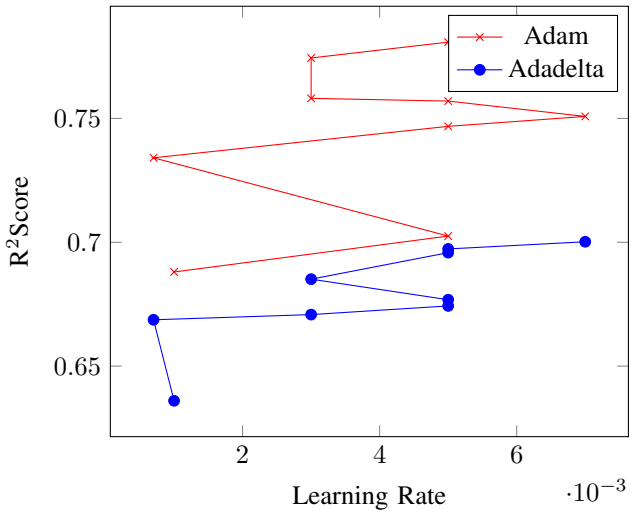
Table II below shows the changes in the R^2 Score with respect to learning rate and batch size. R^2 Score obtained in the above table was when the optimizer was set to Adadelta.

TABLE II
EXPERIMENTAL ANALYSIS USING DIFFERENT HYPERPARAMETERS WHEN OPTIMIZER IS ADADELTA

Learning Rate	Batch Size	R^2 Score
0.0007	256	0.6687
0.007	512	0.7002
0.005	512	0.6973
0.005	256	0.6958
0.005	128	0.6768
0.005	90	0.6743
0.003	512	0.6851
0.003	128	0.6708
0.001	64	0.6360

Adam v/s Adadelta using different hyperparameters is a graph that plots the result (R^2 Score) obtained by changing the learning rate and the batch size. The blue line indicates all the readings of R^2 Score that are obtained when the optimizer is Adadelta, whereas the red line shows all the readings of R^2 Score that are obtained when the optimizer is Adam. Through the plot, we can clearly see that our model performs the best when the optimizer is set to Adam as its lowest reading of R^2 Score is also near the highest reading of R^2 Score when the optimizer is Adadelta.

Adam v/s Adadelta using different hyperparameters



A. Model Performance

According to the above Table I and Table II, the best performing model is the model that has the following hyperparameters:

- Optimizer: Adam
- Learning Rate: 0.005
- Batch Size: 512
- Epochs: 100
- Activation function: ReLU

This model has an R^2 Score of 0.7808.

Fig. 3. below is a graph of Epochs v/s R^2 Score, i.e., the x-axis shows the number of epochs, and the y-axis shows the R^2 Score. The graph was plotted using the code snippet that is shown in Appendix C. This graph was plotted when my model got an accuracy of 0.7808.

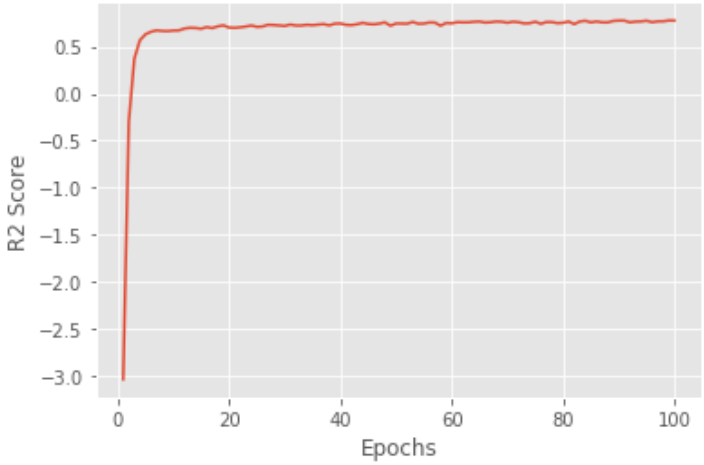


Fig. 3. Epochs v/s R^2 Score Graph

VI. CONCLUSION

In this paper, nonlinear regression has been performed on the modified version of the California Housing Dataset. This

paper explains the supervised machine learning algorithms. Regression is a type of supervised machine learning algorithm which produces an output variable on the basis of the trained input variable(s), which is in continuous/numeric form. Classification is also a type of supervised machine learning algorithm that functions similarly to regression, and it produces categorical output. After rigorous experimental analysis, the best model has been found out, which gives an R^2 Score of 0.7808.

APPENDIX

A. CnnRegressor

```
#Model definintion
class CnnRegressor(torch.nn.Module):
  def __init__(self, batch_size, inputs, outputs):
    #Defining the superclass to store the parameters
    super(CnnRegressor, self).__init__()
    self.batch_size = batch_size
    self.inputs = inputs
    self.outputs = outputs

    self.batch_normalization = BatchNorm1d(inputs)

    #Defining the input layer
    self.input_layer = Conv1d(inputs, batch_size,
    kernel_size=1)

    self.avg_pooling_layer = AvgPool1d(1)

    self.avg_pooling_layer_2 = AvgPool1d(1)

    self.conv_layer = Conv1d(batch_size, 256, 1)

    self.flatten_layer = Flatten()

    self.linear_layer = Linear(256, 128)

    self.linear_layer_2 = Linear(128, 64)

    self.linear_layer_3 = Linear(64, 32)

    self.output_layer = Linear(32, outputs)

  #Defining methods to feed the input through the
    model
  def feed(self, input):

    #Reshape the entry so that it can be put in the
    input layer
    #Even though we are using 1D convolution, it is
    still expecting a 3D arrary
    #to process in a 1D fashion
    input = input.reshape((self.batch_size, self.
    inputs, 1))

    output = self.batch_normalization(input)

    #Getting the output of the first layer and run
    it through ReLU activation
    #function
    output = relu(self.input_layer(output))

    output = self.avg_pooling_layer(output)

    output = self.avg_pooling_layer_2(output)

    output = relu(self.conv_layer(output))

    output = self.flatten_layer(output)

    output = self.linear_layer(output)

    output = self.linear_layer_2(output)

    output = self.linear_layer_3(output)

    output = self.output_layer(output)

    return output
```

Listing 1. CnnRegressor

B. Model Definition

```
#Defining our batch size
batch_size = 512

#Pass the batch size, X columns, and Y columns
model = CnnRegressor(batch_size, X.shape[1], 1)

#Used for setting the model to use the GPU for
    processing
#It also outputs our model summary
model.cuda()
```

Listing 2. Defining the model

C. Plotting the graph

```
#Plotting epochs vs r2 score
plt.plot(epochData, R2ScoreData)

#Defining the y-axis
plt.ylabel('R2 Score')

#Defining the x-axis
plt.xlabel('Epochs')

#Used tho plot the graph
plt.show()
```

Listing 3. Plotting the graph

REFERENCES

[1] George Seif. "5 Types of Regression and their properties" . towardsdatascience.com. https://towardsdatascience.com/5-types-of-regression-and-their-properties-c5e1fa12d55e (Accessed Feb 11, 2020).

[2] "PyTorch Documentation". pytorch.org. https://pytorch.org/docs/stable/nn.html (Accessed Feb 12, 2020).

[3] Jiwon Jeong. "The Most Intuitive and Easiest Guide for Convolutional Neural Network". towardsdatascience.com. https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480 (Accessed Feb 10, 2020).

[4] Sagar Shukla. "Regression and Classification — Supervised Machine Learning". geeksforgeeks.org. https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/ (Accessed Feb 10, 2020).