House Price Prediction using Nonlinear Regression

Shreya Reddy

Department of Computer Science
Lakehead University
sreddy@lakeheadu.ca

I. Abstract

To implement a one dimensional convolution based neural network (1D-CNN) to predict median house value using longitude, latitude, housing median age, total number of rooms, total number of bedrooms, population, number of households, and median income.

II. Introduction

Convolution Neural Network(CNN) in deep learning is a subset of deep neural networks, most widely applied to visual imaging research. To train and evaluate CNN models, each input image must move through a series of filters (Kernels), pooling, fully connected layers (FC) convolution layers and applying an activation function to classify an object with probabilistic values between 0 and 1 [1].
A 1D CNN is a type of CNN that has a hidden that works over a 1D series. The different types of CNN layers are describe below.

Convolution Layer: Its the first layer to extract features from an input image.

Strides: Its the number of pixels shifts over the input matrix.

Pooling: There are different types of pooling layers, but in this assignment we are using max pooling. Max pooling basically takes the largest element from the rectified feature map. Pooling is used to reduces the number of parameters in our model by a process called down-sampling[3].

Activation Function: Rectified Linear Unit(ReLu) is one of the activation functions we are using for this assignment. It is a non-linear function and we use it to know if CNN accepts non-negative linear values or not[2].

For this assignment, I have created a one dimensional convolution (Conv1D)-based neural network for predicting median house value using longitude, latitude, housing median age, total number of rooms, total number of bedrooms, population, number of households, and median income. I have used Google Colab to train and test the model.

III. Experimental Setup

A. Importing the Libraries

Before loading any dataset we first need to import the necessary libraries/packages. Pandas, numpy, matplotlib.pyplot for plotting the graph, sklearn.model_selection for training/testing the model and torch were imported.

B. Loading the Dataset & Data Cleaning

For this assignment, I have used the modified version of the California Housing dataset that's available on Github. To import the dataset we use the pandas library. After loading the dataset I have cleaned up the data to remove any row with NaN values. Figure 3(its dispplayed on page 3) shows the first 5 of the dataset after dropping the NaN values .

C. Visualization

After the data has been loaded, to get a better understanding about the data, there should be some visualization. Figure 1 shows the subplots of the first eighteen samples from the dataset.

To perform this visualization, I used the pandas library to plot the line graph.

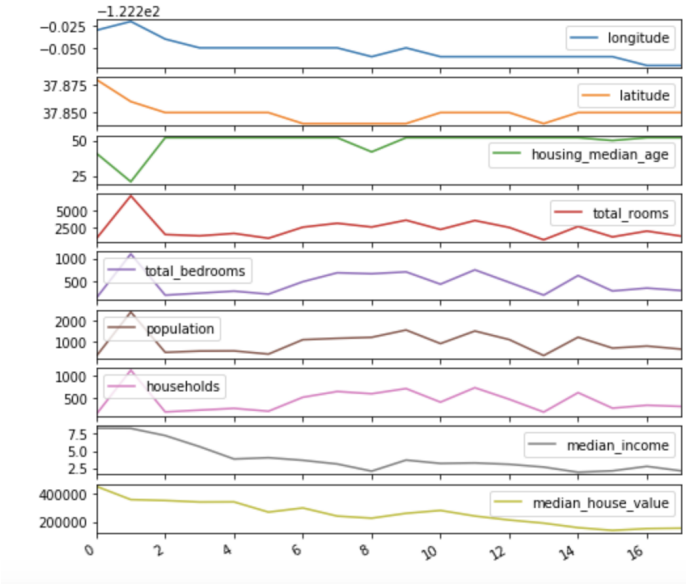

Fig. 1. Subplots of first eighteen samples from the dataset.

D. Training & Testing the dataset

To perform the splitting we import the sklearn library. Then, we define the predict labels (the ones we want to use) and the values we want to predict. Using the train_test_split method we make the split. For this assignment, I am using a train/test split ratio of 70:30.

The test_size=0.3 inside the function indicates the percentage of data that needs to be tested and declaring the random state as 2003. With the outputs of the shape() function we can see that it has 6130rows in the test data and 14303rows in the train data.

After training and testing the dataset, I have plotted a histogram for the trained and tested dataset.

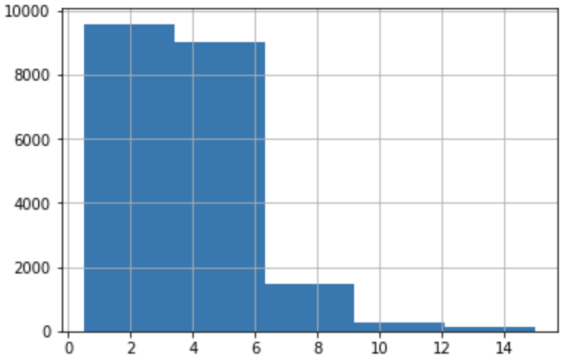

Fig. 2. Histogram for the trained and tested dataset.

E. Over/under fitting issue

Over-fitting means that the model we trained has trained ”too well,” and is now too closely aligned with the training dataset. This model will be very reliable on the training data but on untrained or new data will probably not be very accurate.

Under-fitting means that the model does not fit the data for the training and therefore lacks the data trends. It also means the model cannot be generalized to new data.

F. Creating the model

We need to set the nn.Module, which will define the CNN that we will be using to train. Here we are creating the class which inherits from the nn.Module super class with PyTorch.

The CNN Regressor module has a 1-Dimensional convolution based network. It has 1 max pooling layer, 1 flatten layer, 1 linear layer and 1 output layer. The feed function contains the code to re-shape the data, and the output for a non-linearity output is given to the activation function ReLu.

G. Model Loss

For calculating the loss, L1Loss performance is used and R^2Score metric to calculate the R-Squared Score.

H. Training the model

We need to create an instance of the CnnRegressor class that was created, and then define the loss function and optimizer. Here I have used two optimizers: SGD and Adamax.

I. Testing the model

To test the model, we are first convert the testing set into torch variables using GPU. Then we use a dataloader and predicting the model based on the model trained.

J. Saving the model

I used the torch library to save the trained model.

IV. Results & Conclusion

After changing the epochs to 1000 and adding Adamax optimizer. I have received R^2 score of 0.710 and L1loss of 44087.51 . This implies it is good to predict the median house value.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	me
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

Fig. 3. First five records datasets.

References

[1] Convolutional neural networks tutorial in pytorch. https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-in-pytorch/.

[2] https://medium.com/@raghavprabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

[3] https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6.