Nonlinear Regression with 1D Convolutioanl Neural Network

Karan Atulkumar Shah
Dept. of Computer Science
1100690
Lakehead University
Thunder Bay, Ontario
kshah8@lakeheadu.ca

***Abstract*—This report shows how we can use one dimensional convolutional neural network for nonlinear regression result. For this I have built one neural network to California Housing dataset and got 79.23 testing accuracy.**

Keywords—Conv 1D, Nonlinear Regression, housing data set,

I. INTRODUCTION

This report shows the depth implementation of one dimensional convolutional neural network to train and test the California housing dataset. The data contains information from the 1990 California census. In this report I have mentioned all the steps to train and test the dataset using convolutional neural network like. Data pre-processing, how to remove outliers or unnecessary data, how to use this data to train and test CNN model.

Convolution is a mathematical operation where you "summarize" a tensor or a matrix or a vector into a smaller one. If your input matrix is one dimensional then you summarize along that on dimensions, and if a tensor has n dimensions then you could summarize along all n dimensions. Conv1D and Conv2D summarize (convolve) along one or two dimensions.[1].

In this report I have introduce one dimensional convolutional neural network, where I have used non linear activation function Relu activation which gives us a nonlinear result.

II. IMPLEMANTATION

A. Preprocessing and Loading data

As I mentioned earlier i worked on California housing dataset which is available at GitHub. In reference section I have mentioned the GitHub link.

In this data set we have total 10 Feature among this we have nine independent Feature and one Dependent Feature which is median house value.

	longitude	latitude	...	median_house_value	ocean_proximity
0	-122.23	37.88	...	452600.0	NEAR BAY
1	-122.22	37.86	...	358500.0	NEAR BAY
2	-122.24	37.85	...	352100.0	NEAR BAY
3	-122.25	37.85	...	341300.0	NEAR BAY
4	-122.25	37.85	...	342200.0	NEAR BAY

As you can see I have created 4 function for data pre-processing task. The first method shows how to get the dataset from GitHub URL and how to read it using python panda's library.

```
def getDataset():
  url = "https://raw.githubusercontent
.com/ageron/handson-
ml/master/datasets/housing/housing.csv"
  dataset = pd.read_csv(url)
  print(dataset.head())
  return dataset
```

The second methods show the data visualization of the dataset.

```
def dataVisualization(dataset):
  print("First Ten recoreds of dataset")
  print(dataset.head(n=10))
  print("Plot Each Feature of the data
set on the Seperate Sub-plot")
  print(dataset.plot(subplots=True))
```

Third method checking the null or missing value in the dataset and if there is a missing value than how to remove all missing values from dataset using simple drona function.

```
def checkingAndRemovingNullValues(dataset):
  # Checking Missing Values
  print("Checking Missing Values")
  print(dataset.isnull().sum())
  print("Removing Missing Values")
  dataset = dataset.dropna()
  print(dataset.isnull().sum())
  return dataset
```

The forth method shows the structure of the dataset and find outliers form it and also shows how to remove those values.

```
def removingOutliers(dataset):
  dataset.describe()
    plt.figure(figsize=(10,6))
  sns.distplot(dataset['median_house_value'],color
='green')
  plt.show()
```

```
   dataset[dataset['median_house_value']>450000]['median_house_value'].value_counts().head()
   dataset=dataset.loc[dataset['median_house_value']<500001,:]
   dataset=dataset[dataset['population']<25000]
   return dataset
```

After Pre-process the data, now we can say that we are ready to split our data set into training and testing part. To split the dataset, I have taken all the feature from longitude to median income as independent variable and for dependent variable I have taken median housing value. And then perform data normalization on it to scale all the value between 0 and 1. I have split dataset in to 70:30 ratio as 70% training data and 30%testing data.

```
def loadDataSet(dataset):
   Y = dataset['median_house_value']
   X = dataset.loc[:,'longitude':'median_income']
   x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.3,random_state=2003)
   return x_train,x_test,y_train,y_test,X
->Data normalization.
def normalizingData(x_train,x_test,y_train,y_test):
  sc = StandardScaler()
  x_train_np=sc.fit_transform(x_train)
  x_test_np=sc.transform(x_test)
  y_train_np = y_train.to_numpy()
  y_test_np = y_test.to_numpy()
  return x_train_np,x_test_np,y_train_np,y_test_np
```

B. My Proposed Convolutional Method.

In my proposed convolutional neural network, I have used Four layers in this network. One Input later, Two hidden layer and final output layer.

To get Nonlinear result I have Relu activation function and with this I have used one maxpool and one flatten layer Maxpool layer to compress the input vector dimensionality and to get all metrics in one vector I have used flatten layer.

To get more detail you can see my model class bellow.

```
class _1100690_1dconv_reg(torch.nn.Module):
   def __init__(self,batch_size,inputs, outputs):
     super(_1100690_1dconv_reg,self).__init__()
     self.batch_size = batch_size
     self.inputs = inputs
     self.outputs = outputs

     self.input_layer = Conv1d(inputs,batch_size,1)
      self.max_pooling_layer = MaxPool1d(1)
     self.con_layer = Conv1d(batch_size,128,1)
      self.con_layer1 = Conv1d(128,64,1)
      self.flatten_layer = Flatten()
      self.linear_layer = Linear(64,64)
     self.output_layer = Linear(64,outputs)

   def feed(self,input):
     input = input.reshape((self.batch_size,self.inputs,1))
     output = relu(self.input_layer(input))
     output = self.max_pooling_layer(output)
     output = relu(self.con_layer(output))
     output = relu(self.con_layer1(output))
     output = self.flatten_layer(output)
     output = self.linear_layer(output)
     output = self.output_layer(output)
     return output
```

C. Training The model

TO TRAIN THE MODEL AND GET MORE ACCURACY I HAVE USED LIST OF TRAINABLE PARAMETERS LIKE..

1. Adam Optimizer
   This is the most reliable SGD's optimizer to use In CNN networks.
2. Number of epochs
3. To get more equate result I have used 300 epochs over the training dataset.
4. Batch_size:
   I have used bach_size as 32, so in every iteration I am going to use 32 training example to train model.
5. Kernal size:
   I have taken kernel size equal to 1 because the input size is One in my example you can take kernel size as you like but it should be less then input size. Generally, we are initializing the kernel size in between 3 to 5.

   So after training the model I have got R^2 score approximately 80%.

➢ Here is my proposed one dimensional convolutional neural network.

```
_1100690_1dconv_reg(
  (input_layer): Conv1d(8, 32, kernel_size=(1,), stride=(1,))
  (max_pooling_layer): MaxPool1d(kernel_size=1, stride=1, padding=0, dilation=1, ceil_mode=False)
  (con_layer): Conv1d(32, 128, kernel_size=(1,), stride=(1,))
  (con_layer1): Conv1d(128, 64, kernel_size=(1,), stride=(1,))
  (flatten_layer): Flatten()
  (linear_layer): Linear(in_features=64, out_features=32, bias=True)
  (output_layer): Linear(in_features=32, out_features=1, bias=True)
)
```

III. RESULTS

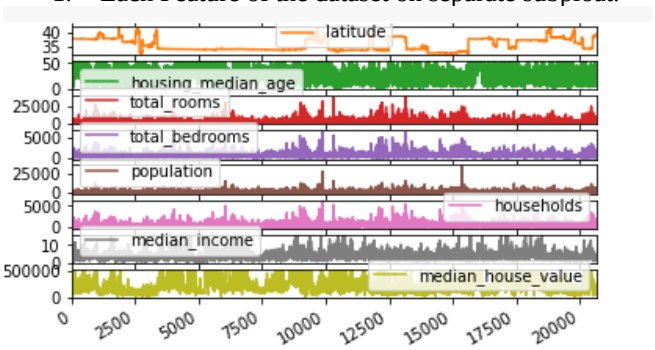
Here Is my training and testing results.

	R^score
Training	80.43%
Testing	79.23%

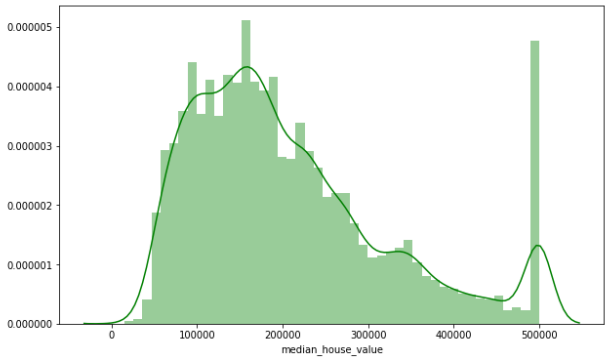
We can boost this accuracy by performing the auto encoder or by batch normalization techniques.

Here are some plot which I have plot during the implementation.

1. Each Feature of the dataset on separate subploat.



2. Outliers



IV. Conclusion and Futurework.

This challenge helped me to extend my knowledge towards deep learning fundamentals. It also gives me a basic knowledge about when and where to use particular deep learning model in particular situations. As I maintained earlier we can improve our accuracy by performing various techniques like auto encoder and batch normalization.

Here is my github link for this assignment : https://github.com/karanshah10/NLP-Nonlinear-Regression-using-1D-CNN

References

[1] https://en.wikipedia.org/wiki/Convolutional_neural_network.

[2] https://pytorch.org/docs/stable/nn.html

[3] https://pytorch.org/docs/stable/optim.html

[4] https://pandas.pydata.org/

[5] https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/

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