Non Linear Regression model using one dimensional Convolution based on Neural Network

Birva Manojkumar Patel
Master of computer science
Lakehead University
ThundeBay, Ontario
bpatel24@lakeheadu.ca

***Abstract*—This report depicts the detailed work of how I implemented the non linear regression model using 1 dimensional convolution based on neural network and got final accuracy of L1Loss value and R2score value. With that I tried to plot the graph for each feature of the dataset on different subplots. Also I gave the basic idea of some key terms used in CNN.**

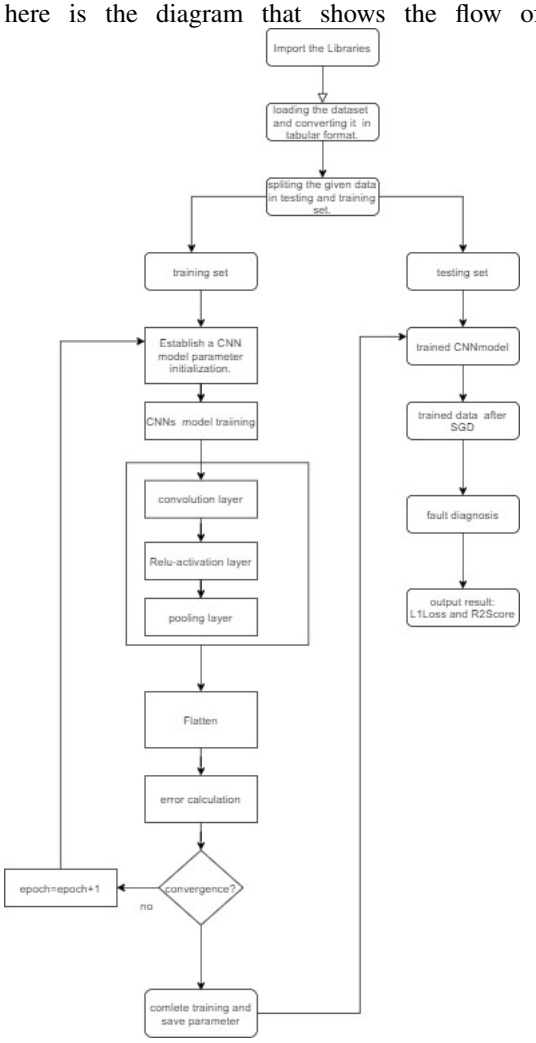
***Index Terms*—neural network,linear-nonlinear regression,natural language processing,deep learning, maxpooling.**

I. Introduction

CNN is now the go-to model on every image related problem. In terms of accuracy they blow competition out of the water. It is also successfully applied to natural language processing and more. The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. For example, given many pictures of cats and dogs it learns distinctive features for each class by itself.

An overview of a convolutional neural network (CNN) architecture and the training process. A CNN is composed of a stacking of several building blocks: convolution layers, pooling layers (e.g., max pooling), and fully connected (FC) layers. A model's performance under particular kernels and weights is calculated with a loss function through forward propagation on a training dataset, and learnable parameters, i.e., kernels and weights, are updated according to the loss value through backpropagation with SGD optimization algorithm. using ReLU rectified linear unit.

PyTorch is an open source machine learning library used for developing and training neural network based deep learning models. It is primarily developed by Facebook's AI research group. Unlike most other popular deep learning frameworks like TensorFlow, which use static computation graphs, PyTorch uses dynamic computation, which allows greater flexibility in building complex architectures. Pytorch uses core Python concepts like classes, structures and conditional loops — that are a lot familiar to our eyes, hence a lot more intuitive to understand. This makes it a lot simpler than other frameworks like TensorFlow that bring in their own programming style.
In implementing this model I am using California housing csv file which is loaded in tabular format and it contains 9 columns/attributes.and at the end model will give the loss based on the testing and training accuracy of the data.
here is the diagram that shows the flow of my work.



II. Background/Literature Review

Typical convolutional layers are linear systems, as their outputs are affine transformations of their inputs. Due to their linear nature, they lack the ability of expressing possible non-linearities that may actually appear in the response of complex cells in the primary visual cortex [25].

Hence, we claim that their expressiveness is limited. To overcome this, various non-linearities have been used as activation functions inside CNNs, while also many pooling strategies have been applied [1].

By reviewing research paper [2] I get to know that 1D CNN is useful for text not on images as the 2D CNN. so complex problems like image recognition can only be solve by 2D CNN. Deep 2D CNNs with many hidden layers and millions of parameters have the ability to learn complex objects and patterns providing that they can be trained on a massive size visual database with ground-truth labels. With a proper training, this unique ability makes them the primary tool for various engineering applications for 2D signals such as images and video frames.

III. PROPOSED MODEL

- By downloading california housing csv file and loading it, the first step begins with the dataset setup. I am using google colab to implement the 1 dimensional convolution neural network. Moving forward with installing packages and loading the library.
  Pandas = A library for data wrangling and data manipulation.
  From Sklearn, sub-library model_selection, I've imported the train_test_split so I can well split dataset to training and test sets
  NumPy = A library of numerical computations.

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
```

- The input data is in CSV format where columns are separated by whitespace. The basic process of loading data from a CSV file into a Pandas is achieved using the "read_csv" function in Pandas. Then printing the first ten row of the data using head() function.

```
dataset=pd.read_csv('housing.csv')
dataset=dataset.dropna()
print("here are the first five row of the data set")
dataset.head(10)
```

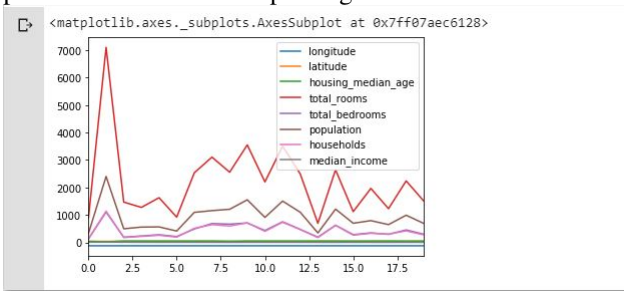
- plot the graph to show comparisons between the dataset values for example longitude, latitude, housing median age, total number of rooms, total number of bedrooms, population, number of households, and median income. Pandas provide a unique method to retrieve rows from a Data frame. Dataframe.iloc[] method is used when the index label of a data frame is something other than numeric series.

```
x = dataset.iloc[0:20]
```

```
df = pd.DataFrame(x, columns = ['longitude', 'latitude',
                                'housing_median_age',
                                'total_rooms', 'total_bedrooms',
                                'population',
                                'households','median_income'])
```

```
df.plot()
```

- the following diagram is the output of the plotting dataset values.



- retrieving the median_house_value for the parameter y which will used for training and it will be helpful to predict the right value and other parameters are included in testing set which is x. We are loading from longitude column to the median_income.

```
Y=dataset['median_house_value']
#TheremainderofthecolumnswillbeusedtopredictY
#Selectfromthe"longitude"columntothe"median_income"column
X=dataset.loc[:,'longitude':'median_income']
```

- train_test_split function splits numpy arrays or pandas DataFrames into training and test sets with or without shuffling.Diving dataset into 80% with training set and 20% testing set

```
x_train,x_test,y_train,y_test =
train_test_split(X, Y ,test_size=0.2)
x_train_np= x_train.to_numpy()
y_train_np= y_train.to_numpy()

x_test_np= x_test.to_numpy()
y_test_np= y_test.to_numpy()
```

- Import the torch and then from that import 1D convolution,maxpooling 1D,flatten, linear model,relu, DataLoader and TensorDataset. relu:ReLU stands for rectified linear unit, and is a type of activation function.Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

```
import torch
from torch.nn import Conv1d
from torch.nn import MaxPool1d
from torch.nn import Flatten
from torch.nn import Linear
from torch.nn.functional import relu
from torch.utils.data import DataLoader,TensorDataset
```

- Defining the CnnRegressor class which initialize the batchsize,input_layer,max_pooling_layer, conv_layer and so on

  batch size: is a term used in machine learning and refers to the number of training examples utilized in one iteration.

  Max pooling: is a sample-based discretization process. The objective is to down-sample an input representation, reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

  convolution layer:A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function. the convolution operation is performed on the input data with the use of a filter or kernel to then produce a feature map.

  flatten layer: is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector.

```
class CnnRegressor(torch.nn.Module):
    def __init__(self,batch_size,
                 inputs,
                 outputs):
        super(CnnRegressor,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.conv_layer=Conv1d(batch_size,128,1)
        self.flatten_layer=Flatten()
        self.linear_layer=Linear(128,64)
        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.conv_layer(output))
        output=self.flatten_layer(output)
        output=self.linear_layer(output)
        output=self.output_layer(output)
        return output
```

- installing pytorch-ignite and importing contribution metrics for regression to find r2_score

```
from torch.optim import SGD
!pip install pytorch-ignite
from torch.nn import L1Loss
from ignite.contrib.metrics.regression.r2_score import R2Sc
```

- defining batch size and creating object for CnnRegression with cuda() function.

```
batch_size=64
model=CnnRegressor(batch_size, X.shape[1], 1)
model.cuda()
```

- Initializing the L1Loss and R2Score function using feed function of CnnRegressor. This function returns the value for avg_loss and avg_score

```
def model_loss(model,dataset,
               train=False,
               optimizer=None):
    performance=L1Loss()
    score_metric=R2Score()

    avg_loss=0
    avg_score=0
    count=0
    for input,output in iter(dataset):
        predictions=model.feed(input)
        loss=performance(predictions,output)
        score_metric.update([predictions,output])
        score=score_metric.compute()
        if(train):
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        avg_loss += loss.item()
        avg_score += score
        count += 1
    return avg_loss/count,avg_score/count
```

- Now define the epoch size and use the SGD optimizer to for learning the training data.

  epoch: it indicates the number of passes through the entire training dataset

  optimizer:It defines some kind of loss function/cost function and ends with minimizing the it using one or the other optimization routine. The choice of optimization algorithm can make a difference between getting a good accuracy in hours or days.

  looping each epochs according to given model and printing the num of epoch with its loss and R2score.

```
epochs = 100
optimizer = SGD(model.parameters(),lr=1e-5)
inputs=torch.from_numpy(x_train_np).cuda().float()
outputs=torch.from_numpy(y_train_np.reshape
                         (y_train_np.shape[0],1))
                          .cuda().float()
tensor= TensorDataset(inputs,outputs)
loader= DataLoader(tensor,batch_size,shuffle=True,
                   drop_last=True)
```

```
for epoch in range(epochs):
  avg_loss,avg_r2_score=model_loss(model,loader,
                                   train=True,
                                   optimizer=optimizer)
  print("Epoch"+str(epoch+1)+":\n\tLoss="
  +str(avg_loss)+"\n\tR^2Score="
  +str(avg_r2_score))
```

- Final step includes the testing which determines by a_test_np and y_test_np. now using TensorDataset put inputs and outputs then using DataLoader pass the batch size, shuffle the data and drop the last unnecessary data. this function prints the final desired output of L1Loss and R2Loss.

```
inputs = torch.from_numpy(x_test_np).cuda().float()
outputs = torch.from_numpy(y_test_np.reshape
                           (y_test_np.shape[0], 1))
                            .cuda().float()
tensor = TensorDataset(inputs,outputs)
loader = DataLoader(tensor, batch_size,
                    shuffle=True,
                    drop_last=True)
avg_loss, avg_r2_score = model_loss(model, loader)
print("The model L1 loss is:" + str(avg_loss))
print("The model R^2 loss is:" + str(avg_r2_score))
```

IV. Experimental Analysis/Comparisons

Most recent studies use hand-crafted feature extraction techniques, such as texture analysis, followed by conventional machine learning classifiers, such as random forests and support vector machines [3]. There are several differences to note between such methods and CNN.

1) CNN does not require hand-crafted feature extraction.
2) Second, CNN architectures do not necessarily require segmentation of dataset by human experts
3) Third, CNN is far more data hungry because of its millions of learnable parameters to estimate, and, thus, is more computationally expensive, resulting in requiring graphical processing units (GPUs) for model training.

V. Conclusion

To conclude,1D CNN works well with the text dataset and in proposed method you can identify that it was not much complex problem so by using 1D CNN we can easily find the L1Loss and R2Score (which are the errors and calculated it using cost function)for given housing dataset. Also I explained Some of the important parameters which plays important part in convolution model for example, learning rate, epochs, pooling, flatten etc.

References

[1] G. Zoumpourlis, A. Doumanoglou, N. Vretos, and P. Daras, "Non-linear convolution filters for cnn-based learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4761–4769.

[2] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *arXiv preprint arXiv:1905.03554*, 2019.

[3] C. Fan, "Survey of convolutional neural network," *Sl: sn*, 2016.