NLP - Assignment 1
Nonlinear Regression with Deep Learning

Dhivya Chandrasekaran
Department of Computer Science)
Student ID : 1111017
dchandra@lakeheadu.ca

***Abstract*—In this assignment we intend to build a Nonlinear Regression model using 1 Dimensional Convolutional Neural Network.**

I. Introduction

Non linear regression is a form of regression analysis in which the decision boundary is not a straight line and rather fits the data in a more flexible curve .The aim of this assignment is to build a custom made 1 dimensional convolutional neural network(CNN) using PyTorch. PyTorch is an open source Python library widely used for Natural Language Processing(NLP) tasks. The article provides a brief overview of convolutional neural networks in section II and the experimental analysis of the model is carried out in section III and section IV provides an outline of the results.

II. Background

The one-dimensional convolution is defined as an operation between a vector of weights and a vector of sequential inputs where the weights act as filters for the data. The idea behind convolution is the element-wise multiplication of these vectors to form another vector. Convolution layers are used to capture primitive features in a data. [1] *Maxpooling* is a non-linear subsampling technique that returns only the maximum value in a given set of input. Maxpooling layers are used in convolutional neural networks to reduce the dimensionality of the input data. *Convolutional neural networks (CNN)* are a class of deep learning models which used the operation of convolution in atleast one of its layers. Dr. Yann Lecun invented the LeNet which is the first deep learning model. Later various CNNs models were proposed of which the most prominent are AlexNet, VGGNet,GoogleNet and ResNet. CNNs are widely used in Image Processing and have become the basis of moder Computer Vision [2].
We calculate the $R2\ score$ for the model proposed in this article. R squared error can be defined as the measure of the closeness to the prediction values to the fitted regression line. The score varies between 0 and 1. 1 indicates that the predicted data points are closer to the expected values. We calculate the $L1\ Loss$ in our model which measure the absolute value difference between the desired output and the predictions.

III. Experimental Analysis and Comparison

The entire code for the assignment is provided in Appendix A

A. Dataset:

For this assignment the *Calfornia housing dataset* is used. This dataset contains the details from 1990 Calfornia census. The dataset contains 20,640 instances and 10 features. The 10 different features and the description are given in Table I.Missing data in the dataset is handled by elimination since only 207 instances have missing values.

B. Model building:

The code snippet for the proposed model is show in A. The proposed model has three convolution layer followed by one max pooling layer. The output from the maxpooling layer is fed to a flatten layer inorder to be to a linear layer. The linear layer output is passed to the output layer which provides the predictions. The proposed model also performs Batch Normalisation to handle vanishing gradients

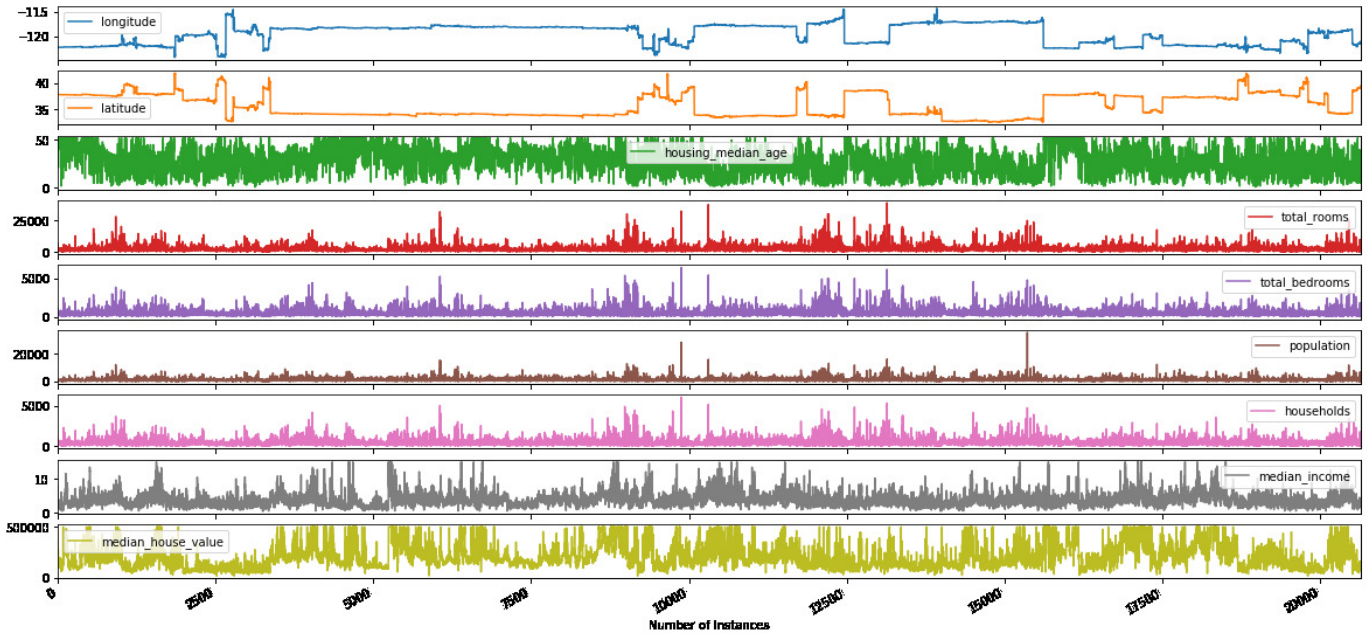

Fig. 1. Line graph depicting features of the dataset

Features	Description
longitude	A measure of how far west a house is; a higher value is farther west
latitude	A measure of how far north a house is; a higher value is farther north
housingMedianAge	Median age of a house within a block; a lower number is a newer building
totalRooms	Total number of rooms within a block
totalBedrooms	Total number of bedrooms within a block
population	Total number of people residing within a block
households	Total number of households, a group of people residing within a home unit, for a block
medianIncome	Median income for households within a block of houses (measured in tens of thousands of US Dollars)
medianHouseValue	Median house value for households within a block (measured in US Dollars)
oceanProximity	Location of the house w.r.t ocean/sea

TABLE I
LIST OF FEATURES IN THE DATASET

and exploding gradients. The input data is passed throw BatchNorm1d layer before passing through the first convolution layer.The model was trained using three different optimizers SGD,ASGD and Adam optimizer. Out of the three, Adam optimizer yielded the best results hence it used in the final model. The model was evaluated with three different activation functions in the convolution layers of the model such as relu, sigmoid and tanh. Sigmoid and TanH caused errors due to vanishing gradients hence ReLU activation function is used in the model. Inorder to take into consideration all the features available the kernel size is maintained at 1.

No of Epochs	Learning Rate	Training R2Score	Testing R2Score	Inference Time in ms
50	0.01	0.719428	0.669038	182
100	0.01	0.732206	0.721175	141
300	**0.01**	**0.758559**	**0.744018**	**139**
500	0.01	0.777081	0.727654	229
1000	0.01	0.779681	0.707925	145

TABLE II
PERFORMANCE COMPARISON AT DIFFERENT NO OF EPOCHS WITH FIXED LEARNING RATE

No of Epochs	Learning Rate	Training R2Score	Testing R2Score	Inference Time in ms
300	0.1	0.684762	0.689833	147
300	**0.01**	**0.758559**	**0.744018**	**139**
300	0.001	0.753317	0.732106	147

TABLE III
PERFORMANCE COMPARISON AT DIFFERENT NO OF EPOCHS WITH FIXED LEARNING RATE

Table II provides the comparison of performance of the model when implemented for different number of epochs at a constant learning rate. Table III provides comparison of performance of the model

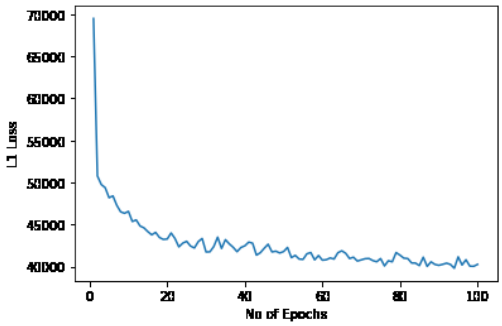

Fig. 2. Loss Curve

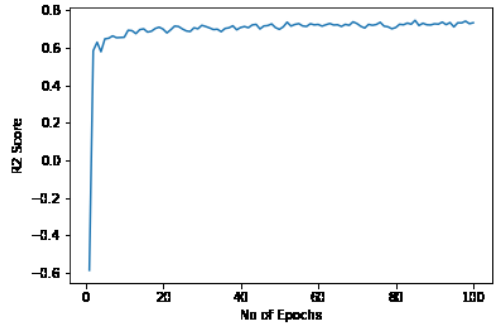

Fig. 3. R2 Score Curve

with different learning rates executed with the most effective number of epochs from Table II. Based on the tabulated results the optimal learning rate for the model is 0.01 and the best results are produced when repeated for 300 epochs.

IV. Results

The data is mounted using the code shown in A and the code to display the first ten records of the dataset is shown in C. The output line graph is shown in Fig.1. The optimal solution for the model is obtained using Adam Optimizer, ReLU activation function, learning rate 0.01, repeated for 300 epochs. The training r2 score is . The testing r2 score is 0.744018 and the L1 Loss is 38706.48 . The total number of trainable parameters in the model are 74,961. The best inference time achieved is 139ms. The loss curve and the r2 score curve are shown in Fig 2 and Fig 3 respectively.

References

[1] Hu, Baotian, Zhengdong Lu, Hang Li, and Qingcai Chen. "Convolutional neural network architectures for matching natural language sentences." In Advances in neural information processing systems, pp. 2042-2050. 2014.

[2] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2414-2423. 2016.

Appendix

A. Mounting data to Google Colab

```
#Mount the google drive and provide
    authentication
from google.colab import drive
drive.mount('/content/drive')
```

B. Importing necessary libraries for data processing

```
from sklearn import linear_model
from sklearn.model_selection import
    train_test_split
from sklearn.metrics import mean_squared_error
    ,r2_score
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

C. Importing necessary libraries for data processing

```

dataset = pd.read_csv('/content/drive/My Drive
    /housing.csv')
print("The first ten rows of the dataset are:"
    )
```

D. Read dataset and print the first 10 rows

```
dataset = pd.read_csv('/content/drive/My Drive
    /housing.csv')
print("The first ten rows of the dataset are:"
    )
dataset.head(10)
```

E. Plot each feature of the dataset on separate subplots

```
axes = dataset.plot.line(subplots=True,figsize
    =(20, 10))
plt.xlabel("Number of Instances")
```

F. Data cleaning by removing null values

```
dataset = dataset.dropna()
```

G. Split data into features and predictions and split the data into training and test in the ratio 70:30

```
Y = dataset['median_house_value']
X = dataset.loc[:,'longitude':'median_income']

# Split the data into training and test sets
    with 70% training samples and 30% test
    samples
```

```
x_train, x_test, y_train,y_test =
    train_test_split(X,Y,test_size=0.3,
    random_state=2003)
```

H. Convert the data to numpy array to be compatible with PyTorch

```
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()
```

I. Import necessary libraries for model execution using Pytorch

```
import time
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 import BatchNorm1d
from torch.nn.functional import relu #
    activation function
from torch.utils.data import DataLoader,
    TensorDataset
from torch.optim import SGD
from torch.optim import Adam
from torch.optim import ASGD
from torch.optim import Adamax
from torch.nn import L1Loss

!pip install pytorch-ignite
from ignite.contrib.metrics.regression.
    r2_score import R2Score
```

J. Initialize and define the CNN model by overwriting the CnnRegressor Class

```
    #Initialize the superclass
    super(CnnRegressor,self).__init__()
    self.batch_size = batch_size
    self.inputs = inputs
    self.outputs = outputs

    self.input_layer = Conv1d(inputs,
    batch_size,1)
    self.batch_norm = BatchNorm1d(8)
    #First Convolution Layer: Filters 128
    self.conv_layer = Conv1d(batch_size,128,1)
    #Second Convoluion Layer: Filters 128
    self.conv_layer_2 = Conv1d(128,128,1)
    #Third Convoluion Layer: Filters 256
    self.conv_layer_3 = Conv1d(128,256,1)
    #Max pool layer
    self.max_pooling_layer = MaxPool1d(1)
    #Flatten layer
    self.flatten_layer = Flatten()
    #Linear layer
    self.linear_layer = Linear(256,64)
    #Ouput layer
    self.output_layer = Linear(64,outputs)

  def feed(self,input):

    input = input.reshape((self.batch_size,
    self.inputs,1))
    output = self.batch_norm(input)
    output = relu(self.input_layer(output))
    #Three convolution layers
    output = relu(self.conv_layer(output))
    output = relu(self.conv_layer_2(output))
    output = relu(self.conv_layer_3(output))
    #max pooling layer
    output = self.max_pooling_layer(output)
    #Flatten layer
    output = self.flatten_layer(output)
    #Linear Layer
    output = self.linear_layer(output)
    #Output Layer
    output = self.output_layer(output)

    return output #return the output
```

K. Build the model

```

# Build the defined model with batch size 64

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

L. Define model_loss function to calculate the performance metrics

```
# Function to calculate the average L1 Loss
    and R2 Score for every epoch.

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
```

M. Train the model

```
epochs = 300

optimizer = Adam(model.parameters(),lr=0.001)

torch.cuda.synchronize()
tr_st_time = int(round(time.time()*1000))

inputs = torch.from_numpy(x_train_np).cuda().float()
outputs = torch.from_numpy(y_train_np.reshape(y_train.shape[0],1)).cuda().float()

r2_score_plot = []
loss_plot = []

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)
  r2_score_plot.append(avg_r2_score)
  loss_plot.append(avg_loss)
  print("Epoch" +str(epoch+1)+":\n\tLoss =" + str(avg_loss)+"\n\tR2_Score = " + str(avg_r2_score))

torch.cuda.synchronize()
tr_time_elapsed = int(round(time.time()*1000)) - tr_st_time
```

N. Save the model

```
torch.save(model.state_dict(), '111101_1dconv_reg.pt')
```

O. Plot the results

```

x_label = np.arange(1,(epochs+1))
plt.figure()
plt.xlabel("No of Epochs")
plt.ylabel("L1 Loss")
plt.plot(x_label,loss_plot)
plt.figure()
plt.xlabel("No of Epochs")
plt.ylabel("R2 Score")
plt.plot(x_label,r2_score_plot)
plt.show()
```

P. Test the model

```
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)

torch.cuda.synchronize()
ts_start_time = int(round(time.time()*1000))
avg_loss, avg_r2_score = model_loss(model, loader)

torch.cuda.synchronize()
ts_time_elapsed = int(round(time.time()*1000)) - ts_start_time
```

Q. Print the results

```
print("Training time elapsed: {} ms".format(tr_time_elapsed))
print ("Test time elapsed: {} ms".format(ts_time_elapsed))

print("The model's L1 loss is: %.2f " % avg_loss)
print("The model's R^2 score is : %.6f" % avg_r2_score)
model_parameters = filter(lambda p: p.requires_grad, model.parameters())
params = sum([np.prod(p.size()) for p in model_parameters])
print("The number of trainable parameters =", params)
```