NLP Assignment 1

1st Khanam K.Zainab
dept. of Computer Science
Lakehead University
Ontario, Canada
kkhanam@lakeheadu.ca

***Abstract*—The assignment was completed based on the housing dataset in which my aim was to improve the performance of the model where I have achieved an accuracy of 0.80 as my R^2 score, with my testing dataset.I also addressed the underfitting and overfitting issue by normalizing the dataset and for overcoming Vanishing/exploding gradient issue I have used relu as my activation function since it which does not cause a small derivative to occur and also the number of trainable parameters I have is 374705 with a kernel size of 1and my inference time of 103.99s when I was training my model and for testing the data the time took was 0.211s and have splitted the training and testing data at a ratio of 70:30 with a random state of 2003 as mentioned in the assignment.**

I. Introduction

The model designed from the housing dataset is a non-linear regression model and I have described the steps of how I have done my assignment in the Steps carried out section and the results of my experiment are described in the Experimental Results Section and finally I have concluded that after normalizing the dataset and changing a number of parameters we can achieve a better R^2 score as well as MSE.

II. Background

For my assignement I have used neural network model which includes the Convolutional Neural network (CNN) which is defined as linear operation that uses general matrix multiplication in atleast one of the layers [1]. CNN has been used widely in many real life applications such as for image classification datasets [2]. Also, in my neural network model I have used Adamax as my optimizer. Optimizers are algorithms that can aid to change the attributes of neural network such as weights and learning rate in order to decrease the losses and thus helps to get results faster, some of the optimizers are - Adam, Adamax,Stochastic gradient [3].Artificial neural networks also have activation function of a node in artificial neural networks and it is defined as the output of that node given an input or set of inputs [4].Furthermore, in machine learning, the vanishing gradient problem is a difficulty found in training artificial neural networks with gradient-based learning methods.The major problem is in some cases, the gradient will be vanishingly small, and thus preventing the weight from changing its value [5]. In the worst case, this may completely stop the neural network from further training.

https://github.com/Zainab-Lakeheadu/NlP_assignment

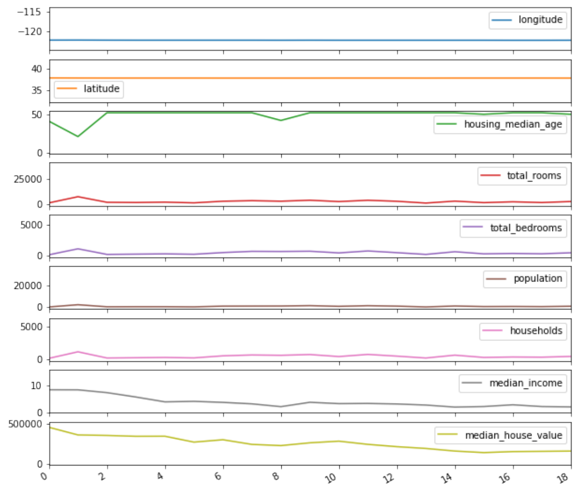

Fig. 1. sub-plots of the first eighteen samples from the dataset in a single figure

Hence, to overcome this issue relu can be used as activation function as it doesn't cause a small derivative and thus the gradient does not become so small.

III. Steps carried out for Completing the Dataset

A. *For Question 1 and 2*

At first, I imported all the necessary libraries that I needed to complete Question 1 and 2. The libraries that I needed are pandas, numpy, matplotlib.pyplot and from sklearn I imported preprocessing.Then I uploaded the csv file into google colab from my local drive which is stated in the appendix section A. After that I have read the file using pandas and printed the first 10 rows. Next, I have deleted the last row which is a categorical column as I have to run the model as a regression model. Then I have deleted any instances which has empty cells and after that I have plotted first eighteen samples from the dataset in a single figure as shown below using the sublpot function as shown in Fig 1.

B. *Steps for the neural network model*

As this model is supposed to be a regression model so I have deleted the last column as it is categorical and dropped the missing values. For avoiding the Vanishing/exploding gradient issue I i have used relu as my activation function and I normalized the data to avoid any anomalous value I have used the Max-Min normalization which is shown in the Appendix D.

I have taken the 8th column as the target variable and from 1- 7 column as my features and I have splitted the dataset at a ratio of 70:30 and have converted the training and testing dataset as numpy arrays so that the pytorch model can process the data. Next for creating the network I have imported pytorch, Conv1d for convolutional layer, MaxPool1d for Max-pooling, Flatten layer and Linear layer and finally imported the dataloader and tensor dataset from pytorch libraries to work with the housing dataset and then defined the model,and after that I have defined the feed model to feed the parameters from one layer to another which is shown in Appendix E. For training the model, I have imported the Adamax optimizer and the metrics L1 loss and R^2 score as metrics for measuring the performance so a value of R^2 score close to 1 will indicate the model have performed very well. Also, I have initialized the batch size as 64.Then, I have initialized the method for training the model and method for running the batches of the training data through the regression model and fetching THE L1 loss and the R^2 which is shown in detail in Appendix F. I have trained the model with 100 epochs and I have initialized the learning rate as 0.001. I have taken a list which will store the value of my R^2 score so that I can find the index of the epoch which has the best result, as by previously training I got the best result at 100 epochs with a R^2 score of 0.80 as my R^2 score so I have kept 100 epochs for my final experiment which is shown in in Appendix F. Then I received the R^2 score with my testing data with is shown in the table I and I have received R^2 score of 0.80 and code is illustrated in details in Appendix G. In order to address the vanishing gradient issue I have used relu as my activation function as it doesn't cause a small derivative to occur and also I have splitted the dataset at a ratio of 70:30 with a total instances of 20640 so my training data is a total 0f 13760 instances and used random seed of 2003 as mentioned in the assignment. I have changed the number of epochs to 100 as after 100 epochs my R^2 score does not have a significant change. I tried changing the kernel size, however for this dataset changing the kernel size did not work that well. I have used Adamx optimizer, I have added convolutional layers and for my hidden layers and I have used 550 neurons and relu as my activation function and thus by changing this parameters, my model was performing much better than before and I have shuffled the data and used random seed as 2003 so that each time the data was shuffled it would give a consistent result so the model was trained with an ample amount of data and for my training dataset I got an accuracy of R^2 Score of 0.84 approximately and after testing with 1/3 of the data I achieved an accuracy of 0.80. So the difference of my testing and training accuracy is only 4% , which indicate my training data did not face any overfitting issue. The number of trainable parameters I have is 374705 and the code executed is shown in Appendix H. Also, the inferences time for the training data was 103.99s and for testing data the inferences time was 0.211s and code is shown in Appendix F G.

IV. Experimental Results

After normalizing the dataset and changing the parameter my model was performing much better, I have changed the optimizer from stochastic gradient(SGD) optimizer to Adamax and I have increased the number of epochs for training 10 to 100 and also I have changed the learning rate from 1e-5 to 0.001 and I have increased the number of hidden neurons as well from 128 to 550 neurons and added 1 more convolutional(Conv) layer. The table below shows the comparison metric score results of the original experiment and the new experiment conducted by changing parameters and the table below shows the comparison of the parameter and the results achieved from the previous experiment conducted in class and new experiment conducted for the assignment is showed in Table I.

Parameters	Previous experiment	New Experiment
Normalization	None	Max-Min
Epochs	10	100
No. of Hidden Neurons	128	550
Conv Layers	1	2
Learning Rate	1e-5	0.001
Optimizer	SGD	Adamax
Random State	None	2003
MSE	91786	0.066(Normalization)
$\text{R}^2 Score$	0.25	0.80
Inference Time	0.087s	0.211s

TABLE I
COMPARISON OF THE OF THE PARAMETER AND RESULT OF THE OLD AND NEW EXPERIMENT

V. Conclusion

To sum up, the assignment was completed based on the housing dataset in which my aim was to enhance the performance of the model where I have achieved an accuracy of of R^2 score of 0.80 with my testing dataset.I also addressed the underfitting and overfitting issue and Vanishing/exploding gradient issue and have the number of trainable parameters of 374705, kernel size of 1, inference time of 0.211s when I was testing the model and inferemce time of 103.99s when I was training the model and have splitted the training and testing data at a ratio of 70:30 with a random state of 2003.

References

References

[1] Hubel, David H., and Torsten N. Wiesel. "Receptive fields and functional architecture of monkey striate cortex." The Journal of physiology 195.1 (1968): 215-243.

[2] Wang, Jiang, et al. "Cnn-rnn: A unified framework for multi-label image classification." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[3] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[4] Haykin, Simon. Neural networks: a comprehensive foundation. Prentice Hall PTR, 1994.

[5] Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6.02 (1998): 107-116.

APPENDIX A
UPLOADING THE DATASET

```
#uploading the file from local drive
from google.colab import files
uploaded = files.upload()
```

APPENDIX B
CODE FOR QUESTION 1

```
#reading the dataset
dataset = pd.read_csv('/content/housing.csv')
dataset.head(10)
```

APPENDIX C
CODE FOR QUESTION 2

```
#for subplottinng the
#features and target variable
#of the dataset
dataset.plot(subplots=True,
figsize=(10, 10));
plt.legend(loc='best');
plt.xlim(0, 18)
```

APPENDIX D
PREPROCESSING THE DATASET

```
#deleted the last categorical column
del dataset['ocean_proximity']
#dropped the instances having missing values
dataset = dataset.dropna()
#for avoiding the
#Vanishing/exploding
#gradient issue.
#returns a numpy array
x = dataset.values
#initializing the MinMax Scaler
min_max_scaler = preprocessing.MinMaxScaler()
#Normalizing the dataset
x_scaled = min_max_scaler.fit_transform(x)
#converting the datset back to a dataframe
dataset = pd.DataFrame(x_scaled)
```

APPENDIX E
DEFINING THE MODEL

```
#have taken the 9th column as the target variable
#which the median house value and from first to
#8th column I have taken as features
X = dataset.loc[:, 0:7]
Y = dataset[8]

#Splitted my training and testing dataset
x_train, x_test, y_train, y_test =
train_test_split(X, Y,
test_size = 0.3,
random_state=2003, shuffle = True)

#Converted the testing and training datasets
#to numpy arrays to work with  PyTorch model
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()

#imported the libraries
import torch
from torch.nn import Linear
from torch.nn import Flatten
from torch.nn.functional import relu
from torch.utils.data import DataLoader,
TensorDataset
from torch.nn import Conv1d
from torch.nn import MaxPool1d

class CnnRegressor(torch.nn.Module):
  #defining the method for initialization
  def __init__(self, batch_size, inputs, outputs):
    super(CnnRegressor, self).__init__()
    self.batch_size = batch_size
    self.inputs = inputs
    self.outputs = outputs
    #Defining the input layer
    self.input_layer = Conv1d(inputs, batch_size,1)
    #Defining the max pooling layer
    self.max_pooling_layer= MaxPool1d(1)
    #Adding 2 convolutional layer with 550
    #as number of hidden neurons
    self.conv_layer = Conv1d(batch_size,
    550, 1)
    self.conv1_layer = Conv1d(550,
    550, 1)
    #defining the Flatten layer
    self.flatten_layer = Flatten()
    #defining the linear layer
    self.linear_layer = Linear(550,64)
    #defining the ouput layer
    self.outputs_layer = Linear(64, outputs)
  #initializing method to feed
  #the paprameters in to the model
  def feed(self, input):
    #reshaping the input as it expects
    #a 1D or 2D layer so specifying
    #it as 1 and feeding the parameters
    #to the output layer
    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 = relu(self.conv1_layer(output))
    output = self.flatten_layer(output)
    output = self.linear_layer(output)
    output = self.outputs_layer(output)
    return output
```

APPENDIX F
TRAINING THE MODEL

```
from torch.optim import Adamax
from torch.nn import L1Loss
!pip install pytorch-ignite
from ignite.contrib.metrics.regression.r2_score
import R2Score

#initializing the batch size
```

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

def model_loss(model,dataset , train = False ,
optimizer = None):
  #Iterates through the batches
  #and fetches the L1 loss
  #and R^2 score
  performance = L1Loss()
  score_metric = R2Score()

  avg_loss = 0
  avg_score = 0
  count = 0
  #gets the model prediction for
  #the training dataset
  #and gets the model L1 Loss and R^2 score
  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):
      #clears out any errors and
      #calculates the gradient
      #and stores the loss before
      #updating the counter
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
    avg_loss += loss.item()
    avg_score += score
    count += 1
  return avg_loss / count ,
  avg_score / count

#time library is imported to
#record the training time
import time
#initializing the number of epochs
epochs = 100
#initializing the optimizer and the learning rate
optimizer = Adamax(model.parameters(), lr = 0.001)
#convert the training datat into torch variables
#for model using the GPU
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)
#Dataloader to process the batches
loader = DataLoader(tensor , batch_size ,
shuffle= True , drop_last=True)
#initializing a list to store the score
maxR2_score = []
#initializing the start time
start = time.time()
#loop defined for training the data
for epoch in range(epochs):
  avg_loss , avg_r2_score = model_loss(model , loader ,
  train = True , optimizer=optimizer)
  maxR2_score.append(avg_r2_score)
  print("Epoch" + str(epoch + 1) + ":\n\tLoss" + str(avg_loss)+
  "\n\tR^2 Score =" + str(avg_r2_score))
print( "value is " + str(max(maxR2_score)) + " at epoch "
+ str(maxR2_score.index(max(maxR2_score))+1) )
#initializing the stop time
stop = time.time()
print(f"Training time: {stop - start}s")
```

APPENDIX G
TESTING THE MODEL

```
#convert the testing datat into torch variables for
#model using the GPU
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)
#Dataloader to process the batches
loader = DataLoader (tensor , batch_size ,
shuffle= True , drop_last=True)


#print the average performance
avg_loss , avg_r2_score = model_loss(model , loader)
print("the model's L1 loss is" + str(avg_loss))
print("the R^2 score is" +  str(avg_r2_score))
```

APPENDIX H
NUMBER OF PARAMETERS

```
def number_of_trainable_parameters(model):
  return sum(p.numel() for
  p in model.parameters()
  if p.requires_grad)
x = number_of_trainable_parameters(model)
print("number of number_of_trainable_parameters "
+ str(x))
```