COMP-5413 - Topics in Natural Language Processing
Assignment 1

Jiyanbo Cao
1094314

***Abstract*—The task of Assignment 1 first is to visualize the first ten records of California housing dataset(housing.csv), and plot each feature of the dataset on separate sub-plot. Then implement a one dimensional convolution based neural network to predict median house value using other parameters.And the solution must be nonlinear regression.**

I. INTRODUCTION

In this assignment, the most important part is to create a nonlinear solution for 1d convolution neural network to predict house value. I'm using several Python libraries to complete the task. Pandas is one library that use to load the dataset. Numpy is a library that use to processing arrays. Matplotlib is a library to draw plots, figures.Sklearn.model_selection to split our dataset to training and testing sample. PyTorch is using to build our own convolution neural network. To create a 1D convolution neural network we need to create several layers that PyTorch provide us. torch is the package that need to be first import. The layer classes are from torch.nn, like Conv1d, MaxPool1d and other layers that we need to create the 1d convolution neural network. The dataset is California housing dataset(housing.csv). The dataset include longitude, latitude, housing_median_age, total_room, total_bedrooms, population, households, median_income, median_house_value, and ocean_proximity. longitude, latitude, housing_median_age, total_room, total_bedrooms, population, households, median_income are parameters that we are going to use to predict median_house_value

II. BACKGROUND

For text processing we actually only need to process the text data from one dimension. So a 1d convolution neural network is suitable for this assignment. And the model must consider modular coding, over/under fitting issue, vanishing/exploding gradient issue, number of trainable parameters, kernal size, inference time, and the dataset should be spilt to train/test of the ratio 70:30. The dataset are download form https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv.
The dataset contain 20433 validate data. There are 8 parameters that we can use to predict the house value. For evaluate the performance of the model we create a model loss function which use L1loss, and R2 score. Also using optimizer to reduce the losses of the model. The optimizer I use is Stochastic gradient descent(SGD), SGD tries to update the model's parameters more frequently. so the model parameters are altered after computation of loss on each training example.

III. PROPOSED MODEL

A.

According to the questions, first to load the dataset from specific website.

```
!wget https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv
```

This is the code used to download the dataset to Colab. I choose to download the dataset instead of link to google drive. Next we need to load the dataset use Pandas, Numpy for manipulate the data, and matplotlib for creating plots and figures.

```
import pandas as pd
import numpy as np
import matplotlib as plt
```

After import those packages we can load the dataset and read the first ten records fromt the dataset.
Use pd.read_csv to read dataset form the right path. dropna() is a function to dispose some data that are not completed. head(10) is to show the first ten records of the dataset. And the Fig.1 is the visualization of first ten dataset.

```
# read dataset
dataset = pd.read_csv('/content/housing.csv')
dataset = dataset.dropna()
# show first 10 records
dataset.head(10)
```

B.

Second to plot the features of the dataset on separate sub-plots. Fig.2 is the visualization.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0

Fig. 1. First ten records

```
show = dataset.head(19)
show.plot(subplots=True)
```

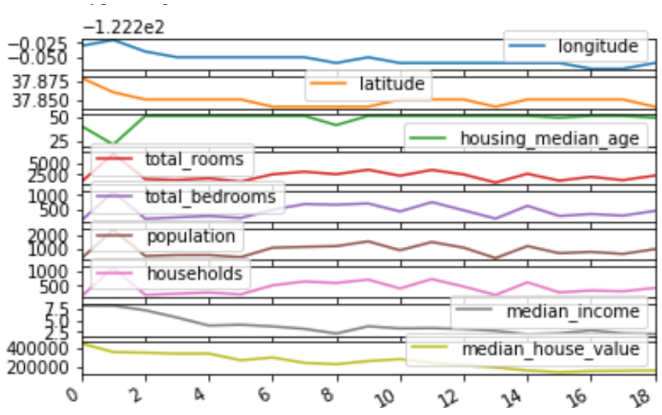

Fig. 2. features of dataset

C.

Before create 1d convolution neural network, we have to split our dataset to traning and testing dataset by using sklearn.model_selection. And define X to be the parameter that used to predict house value Y. Then we spit the dataset.

```
from sklearn.model_selection import
train_test_split
X = dataset.loc[:,"longitude":
"median_income"]
Y = dataset["median_house_value"]
# training/testing 70/30
x_train, x_test, y_train, y_test =
train_test_split(X ,Y, test_size=0.3)
```

Also we need to convert the dataset to numpy array to fit in PyTorch model.

```
x_train_np = x_train.to_numpy()
x_test_np = x_test.to_numpy()
y_train_np = y_train.to_numpy()
y_test_np = y_test.to_numpy()
```

After preparation we can import the package for creating the 1d convolution neural network model. Because this is a 1d convolution neural network so only need to import Conv1d and MaxPool1d. Flatten for convert output to 1 column. Linear is a signal node. Dropout is to prevent model overfitting. Leaky_relu is activation function. DataLoader and TensorDataset are used to work with dataset.

```
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 Dropout
from torch.nn.functional import leaky_relu
from torch.utils.data import DataLoader,
TensorDataset
```

To create your own model form PyTorch, your model class must inherit from *torch.nn.Module*. So the parameter of *Cnn_1d* is *torch.nn.Module*, for initialization function we define parameters as *batch_size*, *inputs*, *output*. *batch_size* is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Then we define the model with 2 convolution layer, one maxpool layer, 1 dropout layer, one flatten layer and three linear layer. Define a forward function to structure our model. First reshape the input to right size that can be fit in our network. Input layer is convolution layer, the paremeters are inputs, batch_size, and kernal size is 1. Here I use leaky_relu function as activation function to reduce linear problem. After this is maxpool layer. The kernal size of maxpool layer is also 1. Maxpool layer is used to down-sample the input,reducing its dimensionality.Dropout layer is used to prevent overfitting. Then next hidden layer is another convolution layer. Next is flatten layer to convert the output feature to 1d. Thus it can be fed into neural network. The last three layer are linear layer, to learn the relation between inputs and outputs.

```
class Cnn_1d(torch.nn.Module):
 def __init__(self, batch_size, inputs, outputs):
  super(Cnn_1d, self).__init__()
  self.batch_size = batch_size
  self.inputs = inputs
  self.outputs = outputs

  # (in_channels, out_channels, kernal_size)
  self.input_layer = Conv1d(inputs,
  batch_size, 1)
  self.max_pool1 = MaxPool1d(1)
  self.drop1 = Dropout(0.25)
  self.conv = Conv1d(batch_size, 128, 1)
  self.flatten = Flatten()
  self.linear1 = Linear(128, 64)
  self.linear2 = Linear(64, 32)
```

```
  self.output_layer = Linear(32, outputs)

 def forward(self, input):
  input = input.reshape((self.batch_size,
  self.inputs, 1))
  output = leaky_relu(self.input_layer(input))
  output = self.max_pool1(output)
  output = self.drop1(output)
  output = leaky_relu(self.conv(output))
  # output = self.max_pool2(output)
  # output = self.drop2(output)
  output = self.flatten(output)
  output = self.linear1(output)
  output = self.linear2(output)
  output = self.output_layer(output)

  return output
```

Next we can train the model with optimizer(Stochastic gradient descent). The evaluation function is L1Loss and R2_score.

```
from torch.optim import SGD
# performance measure
from torch.nn import L1Loss
!pip install pytorch-ignite
from ignite.contrib.metrics.regression.r2_score
import R2Score
```

I define the batch_size to 128. Set the model instance to use GPU for processing.

```
batch_size = 128
model = Cnn_1d(batch_size, X.shape[1],1)
# set the model to use GPU for processing
model.cuda()
```

The evaluation class is modified. The function $model_loss(model, dataset, train = False, optimizer = None)$ accepts four arguments: *model* is the neural network model, dataset is the dataset we use, *train* is Boolean, decide whether execute training process. *optimizer* is whether use optimizer. For loss function will return the average loss and R2 score. Fig.3 and Fig.4 are some training and testing result.

```
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.forward(input)
   loss = performance(predictions,output)
   score_metric.update([predictions,output])
   score = score_metric.compute()

   if(train):
    #  clear any errors
     optimizer.zero_grad()
     loss.backward()
     optimizer.step()

   avg_loss += loss.item()
   avg_score += score
   count += 1
 return avg_loss / count, avg_score / count
```

```
Epoch294:
        Loss=72497.54493947072, R2 Score=0.30082772354779036
Epoch295:
        Loss=72357.60768581081, R2 Score=0.3074066973518501
Epoch296:
        Loss=72006.23155968469, R2 Score=0.32065888121543806
Epoch297:
        Loss=73005.05729166667, R2 Score=0.28818702181095013
Epoch298:
        Loss=72822.77238175676, R2 Score=0.2977566501018745
Epoch299:
        Loss=72464.43697212837, R2 Score=0.3015146277956283
Epoch300:
        Loss=72111.21794059685, R2 Score=0.31540952109804654
```

Fig. 3. Training result

```
test_loss=76450.64328457447,     R2_Score_test=0.2607691243361396
test_loss=74941.22315492021,     R2_Score_test=0.2854900251782188
test_loss=74380.90508643616,     R2_Score_test=0.2819851377215988
test_loss=73719.17495013298,     R2_Score_test=0.3116021066252173
test_loss=74215.40176196808,     R2_Score_test=0.3361419450763868
test_loss=73411.13572140958,     R2_Score_test=0.31710976775938554
test_loss=73638.7069481383,      R2_Score_test=0.31814285493038424
```

Fig. 4. Testing tesult

Now can train the model, I use 300 epochs for training. Set the learning rate to 1e-6.

```
epochs = 300
optimizer = SGD(model.parameters(),lr=1e-6)

tensor_train = TensorDataset(inputs_train,
outputs_train)
loader_train = DataLoader(tensor_train,
batch_size,
shuffle=True, drop_last=True)
tensor_test = TensorDataset(inputs_test,
```

```
outputs_test)
loader_test = DataLoader(tensor_test,
batch_size, shuffle=True, drop_last=True)

Y_train_loss = []
Y_train_R2 = []
Y_test_loss = []
Y_test_R2 = []
# strat the training loop
for epoch in range(epochs):
  avg_loss, avg_r2_score = model_loss(model,
  loader_train,
  train=True, optimizer=optimizer)
  Y_train_loss.append(avg_loss)
  Y_train_R2.append(avg_r2_score)
  avg_loss_test, avg_r2_score_test =
  model_loss(model, loader_test)
  Y_test_loss.append(avg_loss_test)
  Y_test_R2.append(avg_r2_score_test)

  print('Epoch'+str(epoch+1)+':\n\tLoss='+
  str(avg_loss)+',\tR2 Score='+str(avg_r2_score)
  +'\t---\ttest_loss='+str(avg_loss_test)+
        ',\tR2_Score_test=
        '+str(avg_r2_score_test))
```

Fig.5 and Fig.6 are the line chart of the train, test loss and R2 score.

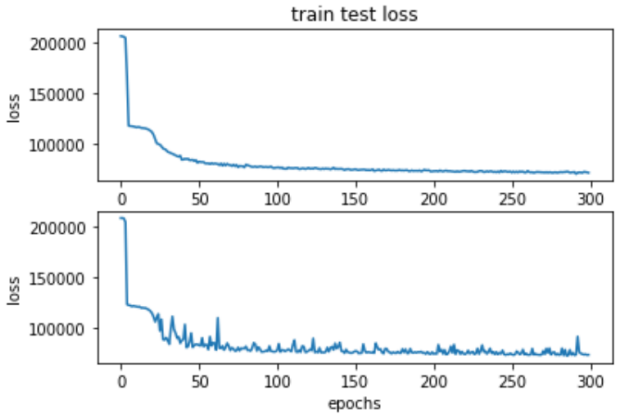

Fig. 5. Train, Test Loss

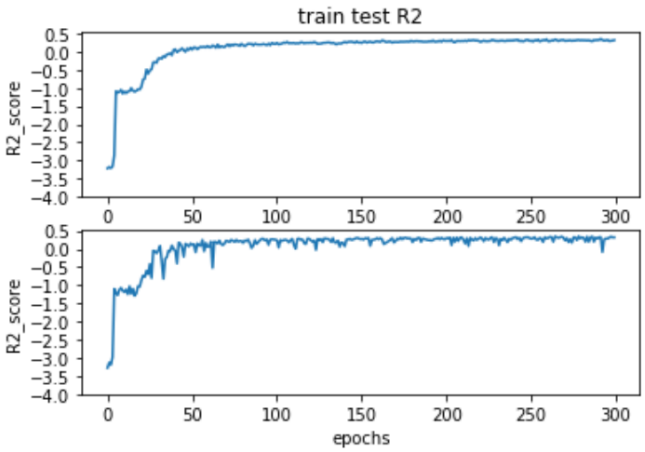

Fig. 6. Train,Test R2 score

IV. Conclusion

There are several things can be change in this model, include the batch size, number of epochs, activation function, number of layers, optimizer. This model with the epochs of 300, the R2 score is around 0.3, loss is around 70000.
Github: https://github.com/AooRobot/house-value-prediction

References

[1] idea from lab2