Non-linear Regression Using 1DCNN.

Kushal Patel
dept. of Engineering
Lakehead University
Thunder Bay, Canada
kpatel57@lakeheadu.ca

***Abstract*—In this paper, idea is to construct one-dimension convolutional neural network for predicting median house value which directly takes values longitude, latitude, housing median age, total number of rooms, total number of bedrooms, population, number of households, and median income considering some parameters such as modular coding, over/under fitting issues, vanishing/exploding gradient issue.**

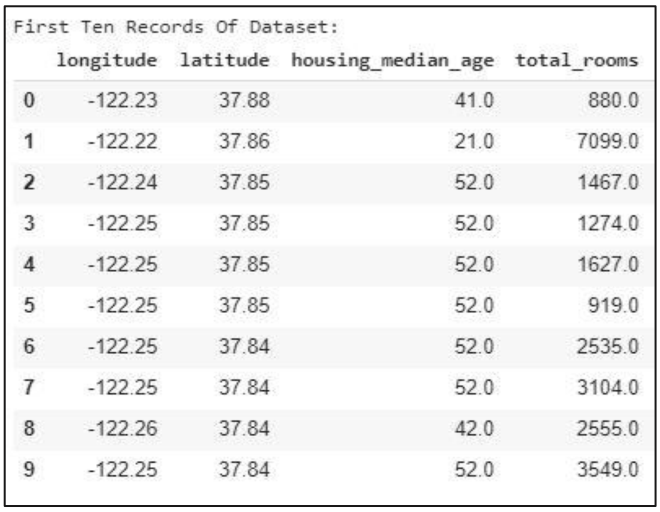
***Keywords*—Nonlinear regression, 1D convolutional neural network, mean square error, r2 score, L1loss.**

I. Introduction

The name of the model is 1104361_con1d_reg and it is predicting the median house value from several data of the housing database using values of longitude, latitude, housing median age, total rooms, total bedrooms, population, households, median income and median house value. Basically, the idea is to implement (1DCNN) one dimensional convolution neural network to predict the value and the output must be non-linear regression. A 1D CNN is very effective when you expect to derive interesting features from shorter (fixed-length) segments of the overall data set and where the location of the feature within the segment is not of high relevance. In statistics, nonlinear regression is a form of regression analysis in which observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables.

II. Modeling

Firstly, there are some libraries which needed to be instaled such as pandas library to read the dataset, sklearn library for preparing our dataset to train and test the model, numpy library to work with and manipulation of the data. To drop any incomplete enteries dropna() method is used so that it will not effect the prediction of value. To print the first 10 enteries of the dataset dataset.head(10) is used and output of it is shown in Fig 1.



First Ten Records Of Dataset:

	longitude	latitude	housing_median_age	total_rooms
0	-122.23	37.88	41.0	880.0
1	-122.22	37.86	21.0	7099.0
2	-122.24	37.85	52.0	1467.0
3	-122.25	37.85	52.0	1274.0
4	-122.25	37.85	52.0	1627.0
5	-122.25	37.85	52.0	919.0
6	-122.25	37.84	52.0	2535.0
7	-122.25	37.84	52.0	3104.0
8	-122.26	37.84	42.0	2555.0
9	-122.25	37.84	52.0	3549.0

Fig. 1. First 10 records of the dataset.

Matplot library is used to plot each feature of the dataset on separate sub-plots which is shown in Fig 2 and each feature is shown clearly in the figure in a different colors. For each sub-plot is implemented using separate Y-axes and a shared X-axes.

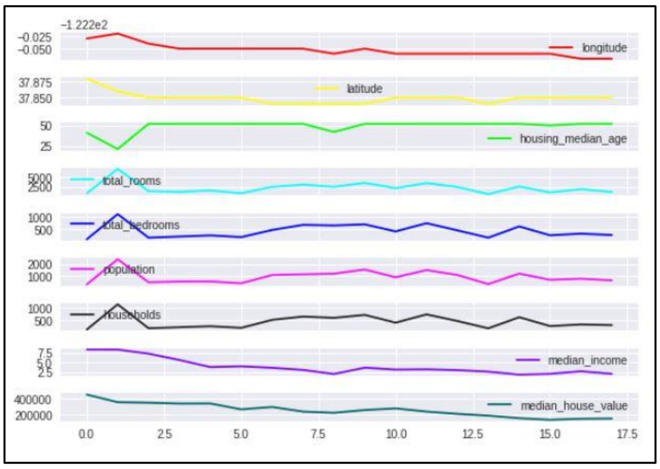


Fig. 2. Visual plotting of each feature of the dataset on separate sub-plots.

A. Splitting Dataset Into X and Y

For input and output of the model I am splitting the dataset as X an input and Y an input as shown in code snippet.

```
#output Y as prediction of median house value
Y = dataset['median_house_value']

#selecting rest of inputs as X
X =dataset.loc[:,'longitude':'median_income']
```

B. Split dataset into training and testing portions.

dataset is splitted 70\% for training and 30\% for testing which is shown in code snippet.

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

C. Converting Numpy Arrays To Work With Pytorch Model.

For the next step we are converting the datasets to numpy so that we can work with the pytorch model and it is shown in code snippet.

```
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()
```

D. *Calculating mean square error*

The mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. MSE is mentioned in code snippet.

```
model = linear_model.LinearRegression()
model.fit(x_train, y_train)
predictions = model.predict(x_test)
mse = mean_squared_error(y_test, predictions)
print("The model's mean squared error is: " +
str(mse))
```

E. *Importing Important Libraries For Model.*

Next step is to import important libraries for the model which are torch, dataloader, tensordataset and some of the layers such as 1 dimensional convolutional neural network, max pooling layer, flatten layer, linear layer, relu activation layer which are briefly shown in code snippet.

```
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
```

F. *Defining Model.*

For defining the model class must be a subclass of torch.nn.Module and initialization method is been defined and batch size, inputs and outputs are given as an argument as Fig 8 presents.

```
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
```

Defining different layers as presented in code snippet as I have used different combination of layers to increase the accuracy.

```
self.input_layer = Conv1d(inputs, batch_size, 1)
    self.max_pooling_layer = MaxPool1d(1)
    self.conv_layer = Conv1d(batch_size, 512, 1)
    self.max_pooling_layer1 = MaxPool1d(1)
    self.conv_layer1 = Conv1d(512, 256, 1)
    self.max_pooling_layer2 = MaxPool1d(1)
    self.conv_layer2 = Conv1d(256, 128, 1)
    self.flatten_layer = Flatten()
    self.linear_layer = Linear(128,64)
    self.output_layer = Linear(64, outputs)
```

Then after I have defined a method called feed for input. Each layer is called in sequence and output of each layer is given as an input to the next layer which increases extra hidden layers to 8 instead of only one. Every convolution layer and input layer is passed through relu activation layer. And the final output of the last layer is returned through feed method.

III. Training the Model

kernel size used for the model is 1 and batch size of the model is 64 which trains the model after 64 entries. To train the model Adam optimizer is used. I have used different types of optimizers and collected data of each optimizers and the accuracy of the model for example, Stochastic gradient descent (SGD) gave me accuracy up to 40 percent and L1loss was also high around seventy thousand which is not good for a model. So I came up with the Adam optimizer from Gradient Descent, Mini Batch Gradient Descent, Momentum, Adagrad etc. To set the model to use GPU while runtime cuda() is used. For calculation purpose L1loss and R2Score packages are installed. After that to calculate L1loss and R2Score one method called model_loss is defined and these scores are stored in a counter and it is been returned in last.

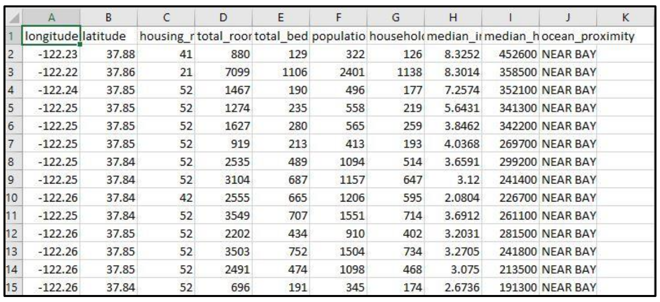
To get the better accuracy correct combination of epochs and batch size should be used. In my model epochs are set to 600 as considering overfitting and underfitting should not happen. Learning rate of the model is 1e-4 as considering epochs and batch size. For the training of the model every time code is been shuffled as you run the whole code so that we can not give the model same values every time while training the model. For every epoch L1loss and R2Score is been calculated and printed as shown in code snippet.

```
epochs = 600
optimizer = Adamax(model.parameters(),
                   lr = 1e-4)
inputs =
torch.from_numpy(x_train_np).cuda().float()
outputs =
torch.from_numpy(y_train_np.reshape(y_train_np.s
hape[0],1)).cuda().float()
tensor = TensorDataset(inputs, outputs)
loader = DataLoader(tensor,batch_size,
shuffle=True, drop_last=True)
time1 = time.time()
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^2 Score = " +
str(avg_r2_score))
```

```
time2 = time.time()
print("Total time taken is "+str(time2 - time1)
+ " seconds " )
```

IV. DATASET

The dataset used to train the model is the California housing data. Different attributes of Housing.CSV file is given as an input and the model is predicting the median house value. Dataset is shown in the Fig 3.



	A	B	C	D	E	F	G	H	I	J	K
1	longitude	latitude	housing_r	total_roor	total_bed	populatio	househol	median_ii	median_h	ocean_proximity	
2	-122.23	37.88	41	880	129	322	126	8.3252	452600	NEAR BAY	
3	-122.22	37.86	21	7099	1106	2401	1138	8.3014	358500	NEAR BAY	
4	-122.24	37.85	52	1467	190	496	177	7.2574	352100	NEAR BAY	
5	-122.25	37.85	52	1274	235	558	219	5.6431	341300	NEAR BAY	
6	-122.25	37.85	52	1627	280	565	259	3.8462	342200	NEAR BAY	
7	-122.25	37.85	52	919	213	413	193	4.0368	269700	NEAR BAY	
8	-122.25	37.84	52	2535	489	1094	514	3.6591	299200	NEAR BAY	
9	-122.25	37.84	52	3104	687	1157	647	3.12	241400	NEAR BAY	
10	-122.26	37.84	42	2555	665	1206	595	2.0804	226700	NEAR BAY	
11	-122.25	37.84	52	3549	707	1551	714	3.6912	261100	NEAR BAY	
12	-122.26	37.85	52	2202	434	910	402	3.2031	281500	NEAR BAY	
13	-122.26	37.85	52	3503	752	1504	734	3.2705	241800	NEAR BAY	
14	-122.26	37.85	52	2491	474	1098	468	3.075	213500	NEAR BAY	
15	-122.26	37.84	52	696	191	345	174	2.6736	191300	NEAR BAY	

Fig. 3. *dataset*

V. TESTING THE MODEL

For testing of the model I am converting the testing set into torch variables. So average performance of the model is given by avg_loss and avg_r2_score. In our case R2Score and L1loss is shown in code snippet.

```
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("Loss = " + str(avg_loss) + "\nR^2 Score =
" + str(avg_r2_score))
```

VI. COMPARISION OF DIFFERENT MODELS

For testing of the model I have tested different types of optimizers with the combination of different layers as already shown above and I have generated measuring parameters for 50 epochs which are shown in below Table one. By looking at the results R2Score of RMSprop is good but for the higher number of epochs as 600 Adam optimizer is giving best output measures like R2Score is equal to 0.6831815409053447 , loss is equal to 44847.28375822368, mean squared error is equal to 4778844240.921353 and the inference time is 564.6949813365936 seconds which is the best model accuracy I have achieved.

TABLE I. COMPARISION OF DIFFERENT MODELS

Model Optimizer	Avg_R2Score	Avg_L1Loss	MSE	Inference Time
Adam	0.3680549187	67315.652	0.4998e6	45.30
RMSprop	0.4175867325	68131.767	0.4909e6	42.53
Adamax	0.2685680069	75832.675	0.4853e6	53.85
Adadelta	-0.915217697	114621.94	0.4875e6	48.25

Fig. 4. *Different values of different model*

REFERENCES

[1] Ali Haidar, and Brijesh Verma, “Monthly Rainfall Forecasting Using One-Dimensional Deep Convolutional Neural Network,” Central Queensland University, Sydney, NSW 2000, Australia.

[2] Yunlei Sun,” The Neural Network of One-Dimensional Convolution-An Example of the Diagnosis of Diabetic Retinopathy ,”College of Computer & Communication Engineering, China University of Petroleum (East China), Qingdao 266580, China.

[3] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman, “1D Convolutional Neural Networks and Applications – A Survey Professor, Department of Electrical Engineering, Qatar University, Qatar.

[4] T akilan,”Lecture Notes,”, Lakehead Univesity, Thunder bay P7B 5E1, Ontario, Canada.