Non Linear Regression w/t DL

Jagmeet Singh Sidhu
Lakehead University
Student ID- 1102275
Email- Jsidhu10@lakeheadu.ca

***Abstract*—This is the assignment report in which we learn Convoloution Neural Network(CNN) of multi-layer, 1 Dimension with PyTorch to perform regression on the dataset that we use can be downloaded from https://github.com/ageron/handson-ml/tree/master/datasets/housing we also connected GPU using Google Colab. We created a custom method to train our new model in batches and let it run for a set number of epochs. Finally, we evaluate our model on the testing dataset to see how it performs using R2 square value and for optimizer function we evaluated using AdamDelta, Adams, SGD, Rprop to see which function performs better. We plotted the the whole dataset using .Subplot() library and calculated the inference time from importing libraries to end where we are printing average R2 square value**

I. Introduction

In this paper we use the Non Linear regression with CNN 1 dimension using PyTorch. We use different optimizer like SGD, Rprop, Adam.

Stochastic Gradient Descent (SGD) performs an update of the parameters for every example of preparation. Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients before updating each parameter for similar examples. When implementing one update at a time, SGD removes this redundancy.
We use Rprop as well. For feedforward artificial neural networks, Rprop, short for robust backpropagation, is a learning heuristic for supervised learning. This is an optimisation algorithm for first order. Similar to the Manhattan update law, Rprop only takes into account the sign of the partial derivative over all trends (not magnitude) and acts on each "weight" independently. For each weight the update value for that weight is multiplied by a factor if there was a sign shift of the partial derivative of the total error function compared to the last iteration.
Adam is an optimization algorithm which can be used to update the network weights iteratively based on training data instead of the traditional stochastic gradient descent technique. Instead of adjusting the parameter learning rates based on the average first moment (mean) as in RMSProp, Adam also uses the average second gradient moments (uncentered variance) as well. The algorithm explicitly calculates an exponential moving average of gradient and squared gradient, and the beta1 and beta2 parameters control the decay rates of these moving averages. The initial value of the moving averages and the near to 1 beta1 and beta2 values results in a distortion of the estimates of moment to Zero.

II. Background

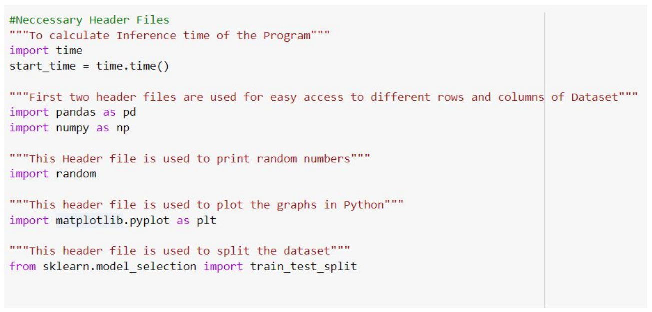
Linear regression models provide a rich and versatile structure that meets various analysts ' needs. Linear regression models are not suitable for all circumstances however. Throughout engineering and sciences there are many problems where the response variable and the predictor variables are related through a known nonlinear process. Which leads to a type of nonlinear regression. When the least squares approach is applied to these models, the resulting normal equations are nonlinear, and usually difficult to solve. The usual approach is for an iterative method to immediately minimise the residual number of squares.

A convolutionary neural network (CNN, or ConvNet) in deep learning is a subset of deep neural networks, most widely applied to visual imaging research. These are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the characteristics of their shared-weights architecture and invariance translation. These have image and video recognition tools, suggest framework, image detection, medical image analysis, and processing of natural language.

III. Proposed Solution

1. Firstly we import libraries

Installing neccessary Libraries to do the data wrangling process.

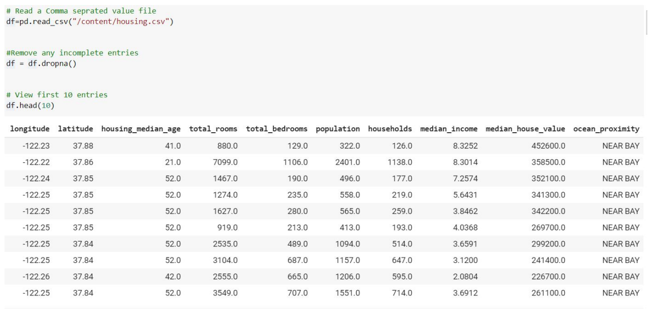


```
#Neccessary Header Files
"""To calculate Inference time of the Program"""
import time
start_time = time.time()

"""First two header files are used for easy access to different rows and columns of Dataset"""
import pandas as pd
import numpy as np

"""This Header file is used to print random numbers"""
import random

"""This header file is used to plot the graphs in Python"""
import matplotlib.pyplot as plt

"""This header file is used to split the dataset"""
from sklearn.model_selection import train_test_split
```

Fig. 1. Importing Libraries

2. Importing and printing Dataset

Here we import the whole dataset from the githb link provided.

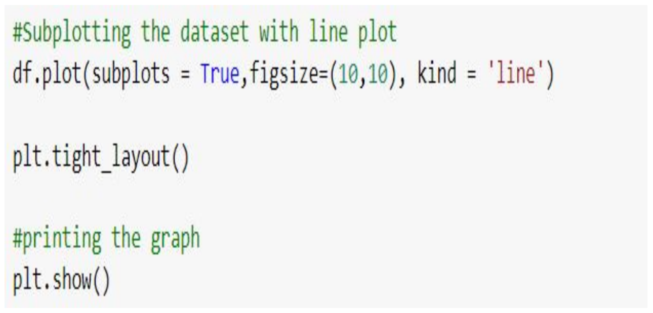


```
# Read a Comma seprated value file
df=pd.read_csv("/content/housing.csv")


#Remove any incomplete entries
df = df.dropna()


# View first 10 entries
df.head(10)
```

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

Fig. 2. First 10 Columns of Dataset

3. Plotting Dataset



```
#Subplotting the dataset with line plot
df.plot(subplots = True,figsize=(10,10), kind = 'line')

plt.tight_layout()

#printing the graph
plt.show()
```

Fig. 3. code to plot

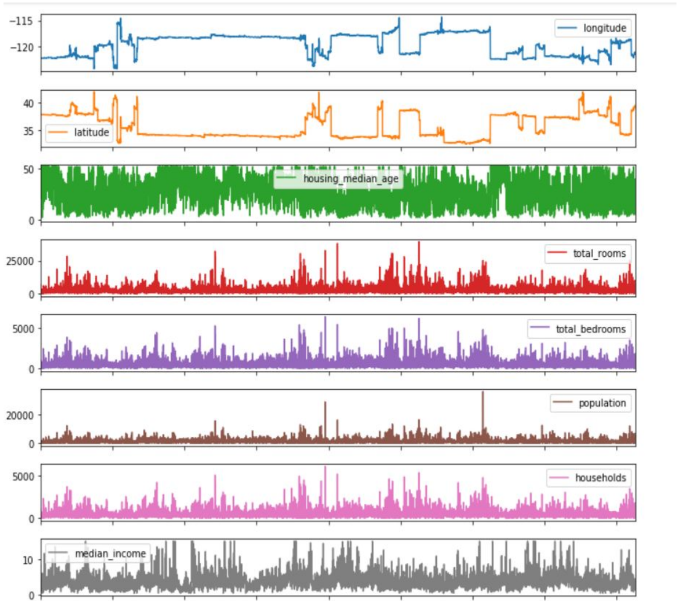


Fig. 4. Line plot

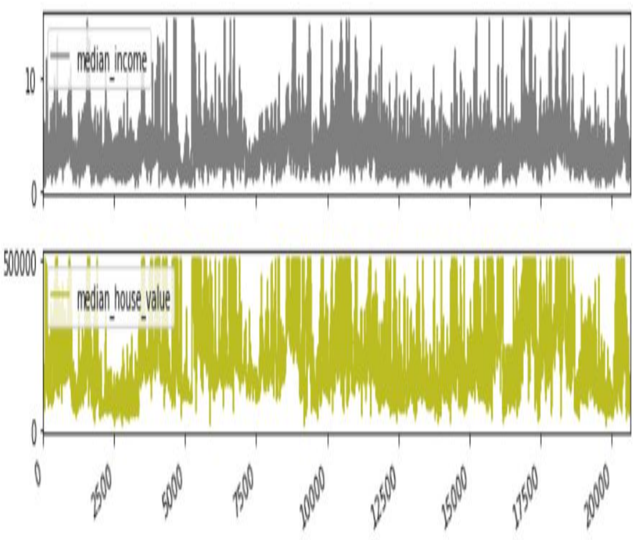
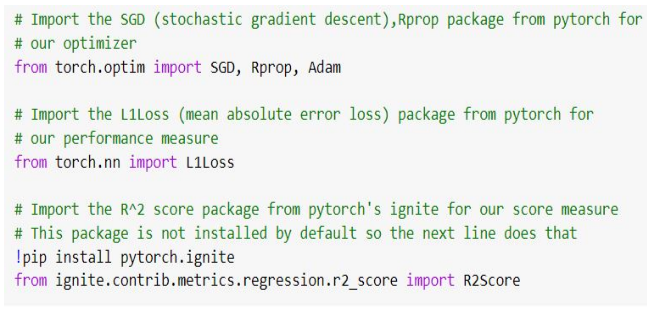


Fig. 5. Line plot

4. Importing Optimizers

Here as you can see below the 3 different optimizrs has been installed.

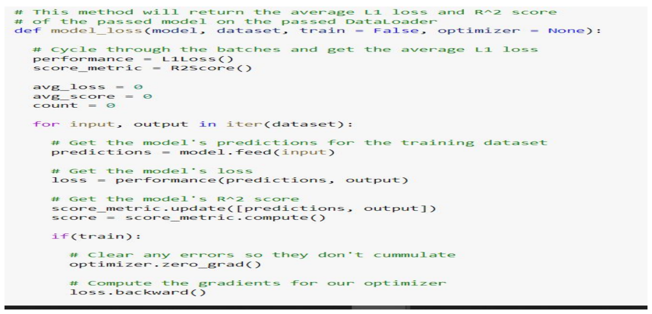


```
# Import the SGD (stochastic gradient descent),Rprop package from pytorch for
# our optimizer
from torch.optim import SGD, Rprop, Adam

# Import the L1Loss (mean absolute error loss) package from pytorch for
# our performance measure
from torch.nn import L1Loss

# Import the R^2 score package from pytorch's ignite for our score measure
# This package is not installed by default so the next line does that
!pip install pytorch.ignite
from ignite.contrib.metrics.regression.r2_score import R2Score
```

Fig. 6. optimizers for Training models

5. Training Model

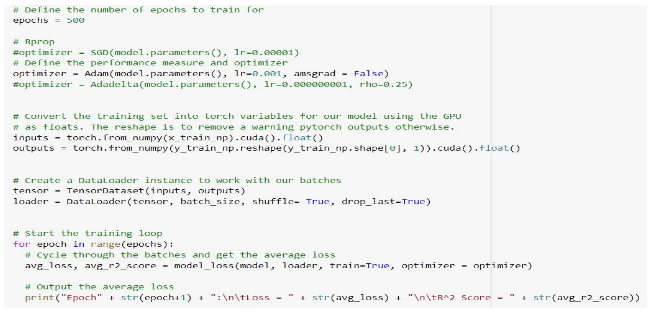
Here we started the actual Training of the model.



```
# This method will return the average L1 loss and R^2 score
# of the passed model on the passed DataLoader
def model_loss(model, dataset, train = False, optimizer = None):

  # Cycle through the batches and get the average L1 loss
  performance = L1Loss()
  score_metric = R2Score()

  avg_loss = 0
  avg_score = 0
  count = 0

  for input, output in iter(dataset):

    # Get the model's predictions for the training dataset
    predictions = model.feed(input)

    # Get the model's loss
    loss = performance(predictions, output)

    # Get the model's R^2 score
    score_metric.update([predictions, output])
    score = score_metric.compute()

    if(train):

      # Clear any errors so they don't cummulate
      optimizer.zero_grad()

      # Compute the gradients for our optimizer
      loss.backward()
```

Fig. 7. Training Model

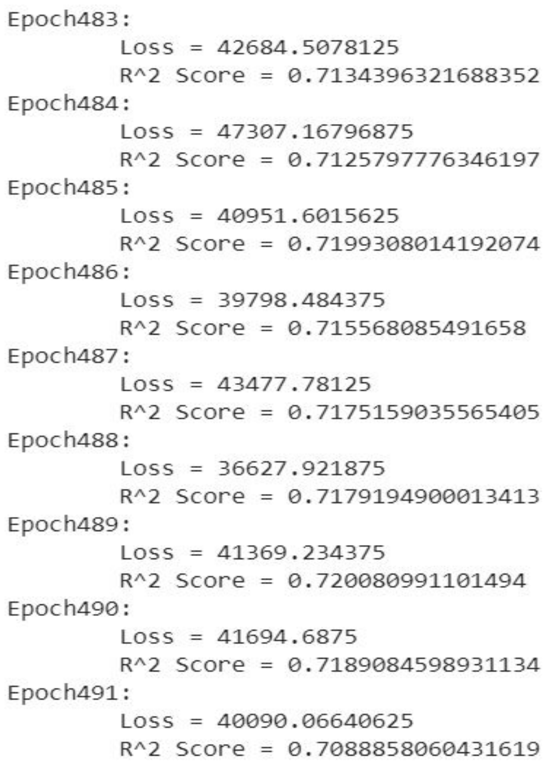
6. Setting Epochs



```
# Define the number of epochs to train for
epochs = 500

# Rprop
#optimizer = SGD(model.parameters(), lr=0.00001)
# Define the performance measure and optimizer
optimizer = Adam(model.parameters(), lr=0.001, amsgrad = False)
#optimizer = Adadelta(model.parameters(), lr=0.000000001, rho=0.25)


# Convert the training set into torch variables for our model using the GPU
# as floats. The reshape is to remove a warning pytorch outputs otherwise.
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()


# Create a DataLoader instance to work with our batches
tensor = TensorDataset(inputs, outputs)
loader = DataLoader(tensor, batch_size, shuffle= True, drop_last=True)


# Start the training loop
for epoch in range(epochs):
  # Cycle through the batches and get the average loss
  avg_loss, avg_r2_score = model_loss(model, loader, train=True, optimizer = optimizer)

  # Output the average loss
  print("Epoch" + str(epoch+1) + ":\n\tLoss = " + str(avg_loss) + "\n\tR^2 Score = " + str(avg_r2_score))
```

Fig. 8. Setting total 500 Epochs

7. Printing result of Epochs and R square values

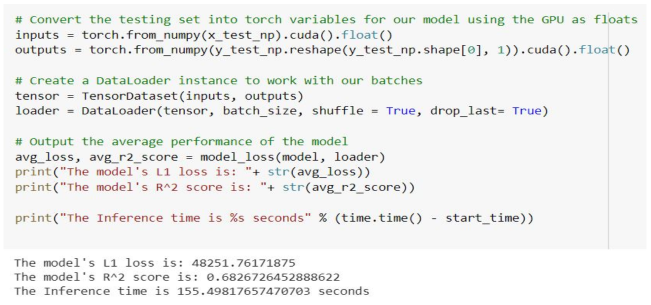


```
Epoch483:
        Loss = 42684.5078125
        R^2 Score = 0.7134396321688352
Epoch484:
        Loss = 47307.16796875
        R^2 Score = 0.7125797776346197
Epoch485:
        Loss = 40951.6015625
        R^2 Score = 0.7199308014192074
Epoch486:
        Loss = 39798.484375
        R^2 Score = 0.715568085491658
Epoch487:
        Loss = 43477.78125
        R^2 Score = 0.7175159035565405
Epoch488:
        Loss = 36627.921875
        R^2 Score = 0.7179194900013413
Epoch489:
        Loss = 41369.234375
        R^2 Score = 0.720080991101494
Epoch490:
        Loss = 41694.6875
        R^2 Score = 0.7189084598931134
Epoch491:
        Loss = 40090.06640625
        R^2 Score = 0.7088858060431619
```

Fig. 9. R square value of every epoch

8. Printing Final Result and Inference Time

After training and testing the final result is printed in the below image shown.



```
# Convert the testing set into torch variables for our model using the GPU as floats
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()

# Create a DataLoader instance to work with our batches
tensor = TensorDataset(inputs, outputs)
loader = DataLoader(tensor, batch_size, shuffle = True, drop_last= True)

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

print("The Inference time is %s seconds" % (time.time() - start_time))
```

```
The model's L1 loss is: 48251.76171875
The model's R^2 score is: 0.6826726452888622
The Inference time is 155.49817657470703 seconds
```

Fig. 10. Final Results

IV. Results

I tried three optimizers in this SGD, Rprop, Adam. I experimented various things. Firstly I added the new layer this exponentially increase the R square value from 0.3 to 0.5 but remains same after adding one more. Then I increase the the batch size from 64 to 256 it increase certain accuracy from 0.5 to 0.6. After this I start experimenting with optimizers SGD didn't perform well decrease the accuracy from 0.6 to 0.4. Then I tried Rprop it again increase the valaue to 0.6. But at the end I tried Adam the R square value increases from 0.6 to 0.67 with lower Inference time.

References

[1] Pramod Singh, Avinash Manure - Neural Networks and Deep Learning with TensorFlow.

[2] Gallant - Non Linear Regression.

[3] Deboleena Roy, Priyadarshini Panda, Kaushik Roy - Non Linear Regression.

[4] Shreyans Pathak, Shashwat Pathak - Data Visualization Techniques, Model and Taxonomy