Implement 1D Convolution Layer Neural Network for Non-linear Regression

Bhargavkumar Patel

1106925

[1]Lakehead University

bpatel22@lakeheadu.ca

Abstract—*This paper justifies the work and the implementation performed throughout the completion of the non-linear problem for 1D Convolution Network.[2] Non Linear regression problem. Nonlinear regression models are important tools because many real estate datasets and entries are better represented by nonlinear than linear models. Fitting nonlinear models is not a single-step procedure but an involved process that requires careful examination of each individual step. Depending on the objective and the application domain, different priorities are set when fitting nonlinear models; these include obtaining acceptable parameter estimates and a good model fit while meeting standard assumptions of statistical models. The steps followed are as follows: 1. Define a Regression Model. 2. Initialize parameters like batch size, input, output size. 3. Define the model with the input filter size/Kernel size and batch size. 4. Apply maxpooling layer to the model for dimension reduction. 5. Flatten the layer to compute the vector operation of the inputs and the recent output with filtersize. 6. Feed the network the values required like input, batch size and the number of kernel so that the model can train itself. 7. Compute the output with L1Loss and R2Score for testing the model and calculate the accuracy of the model. I hope that this contribution will clarify some of the difficulties and confusion with the task of using nonlinear models and to gain decent accuracy with some feature alteration and desired operations.*

IndexTerms: c*nn, adam, maxpooling, kernel, batch, normalization, LSTM*

1. INTRODUCTION

Convolutional Neural Network (CNN) models were developed for image classification, in which the model accepts a two-dimensional input representing an image's pixels and color channels, in a process called feature learning.

This same process can be applied to one-dimensional sequences of data. The model extracts features from sequences data and maps the internal features of the sequence. A 1D CNN is very effective for deriving features from a fixed-length segment of the overall dataset, where it is not so important where the feature is located in the segment. The 1D cnn works well with: 1. Analysis of a time series of sensor data. 2. Analysis of signal data over a fixed-length period, for example, an audio recording. 3. Natural Language Processing (NLP), although Recurrent Neural Networks which leverage Long Short Term Memory (LSTM) cells are more promising than CNN as they take into account the proximity of words to create trainable patterns.

For this experiment of housing dataset for solving 1D convolution layer network we have created a network that extracts the values from the dataset namely: longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, median_house_value, and ocean proximity where ocean proximity is the output we want to test after the training of the model.

1.1. Background

For this implementation of the model I have used the PyTorch[3] framework . PyTorch is a Python-based library that provides functionalities such as: 1. TorchScript for creating serializable and optimizable models. 2. Distributed training to parallelize computations. 3. Dynamic Computation graphs which enable to make the computation graphs on the go.

The Libraries used during the implementation are as follows:

- Pandas
- Sklearn
- Numpy
- Matplotlib
- Torch
- Time
- Ignite

These are some libraries used. I have used Pandas for reading the dataset from the github using the url and function read.csv(). Sklearn library for splitting the dataset into train and test respectively. The other use of Sklearn is done for normalizing and scaling the datapoints in range [0,1] so that they are available uniform for training and testing. Matplotlib is used for plotting the subplots of the different features of the dataset. The use of time library is to keep track on the time elapsed during the computation of the training model. Ignite helped extract the R^2score (accuracy) of the model. Torch library is uses the neural network packages required for designing the model for regression.

2. PROPOSED MODEL

The proposed model is designed to solve the 1D convolution layer non regression problem.

Steps to follow implementation:

1. Connect to a GPU runtime from Google Colab and process housing dataset.
2. Process the dataset : housing.csv from the link given below

   "https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv"

3. Create Custom model ,from scratch using 1D convolution layers.
4. Create a custom method to train our new model in batches and let it run for a set of epochs.
5. In the final step evaluate the model on the testing dataset to see how it performs.

2.1. Experimental Analysis and modifications

1. The code has been framed with two hidden layers namely "CONV_LAYER" and "CONV_LAYER1" in which the output of the first layer is given as input for the second newly formed layer.

```
self.conv_layer = Conv1d(batch_size, 128, 1)
self.conv_layer1 = Conv1d(128, 128, 1)
```

2. Apart from this since new convolution layer was created so activation relu function is applied to the last formed layer so as to overcome the overfitting of the data.
3. The data is normalised using the MinMax Scaler function of sklearn library that scales the data into range of [0,1].
4. Subplots are generated for every feature
5. For inference time the library time is used to calculate the time elapsed during computation.
6. Adam Optimizer is used insetad of SGD to yield better results.[3]
7. To remove the Vanishing Gradient problem ,the code has code :
   Optimizer.zero_grad() that will remover the gradient with respect to parameters and
   Loss.backward() to recover the gradient with respect to parameters.

```
optimizer.zero_grad()

loss.backward()
```

8. Parameters are altered to gain high accuracy.

3. RESULTS

All the results from the implementation are as follows:

1. Loading the dataset from the given link

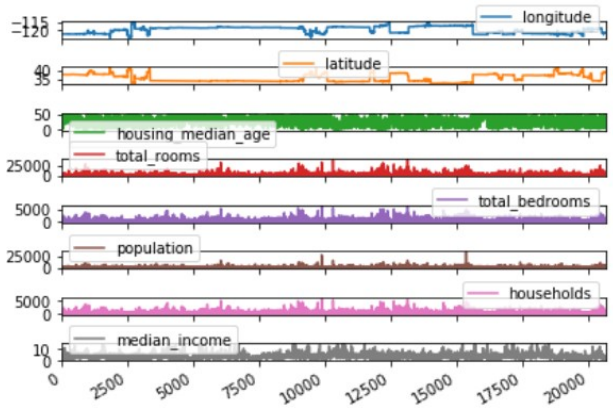
```
[ ]  #reads comma-seperated data (csv file)
     dataset = pd.read_csv('https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv')
```

2. Print the first 10 records

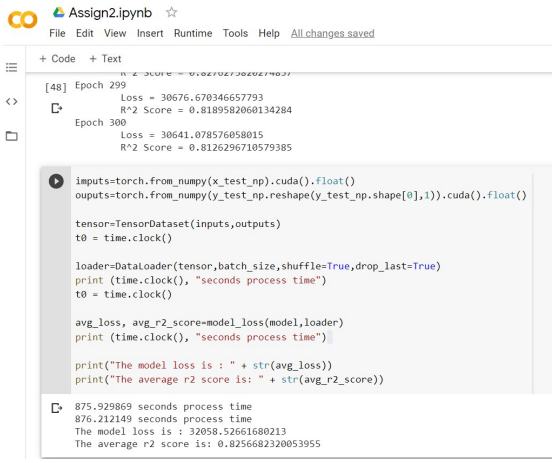
```
#view the first five rows of the dataset
dataset.head(10)
```

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

3. Plotting Graph for features



4. Average Loss and Average R^2 Score



5. CONCLUSION

In this Assignment I have created a 1D convolution model that trains the model on basis of certain parameters and tests the model with a loss of approximately 32000 and average R^2 score approximately upto 0.8256 (82.56%) on final notes. Apart from this it helped me gain a vast knowledge of how modular programming is executed and its importance when working with large codes, how to create a CNN model from scratch and how altering the parameters results in drastic change to accuracy and the loss detection. It gave me a great hands on experience by formulating different aspects of code and terminologies to design a user friendly and efficient program. The biggest challenge I faced is to design a CNN model with two hidden layers and the parameters it requires to process the model. Also I realised that the only way you can turn an inefficient code to efficient code is by performing error solving method. For information on source code and to learn I have created a git repository:

https://github.com/Bhargav0312/1Dconvolutionnetwork

6. COMPARISON MATRIX

Parameters	Unchanged Values	Changed Values
Number of epochs	10	300
Learning rate	1e-5	1e-2,1e-3
Batch size	64	64
Average R^2 Score	.34(approx)	.8256(approx)
Average Model Loss	70000 (approx)	32000(approx)

REFERENCES

[1] https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/
[2] https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/
[3] https://pytorch.org/docs/stable/optim.html
[4] Nonlinear Regression Models and Applications in Agricultural research