

Implementing Convolutional Neural Network for a Housing Data

Vishnu Sudheer Gannamani

Student Id: 1110200

Department of Computer Science Engineering
Lakehead University, ThunderBay

I. ABSTRACT

The document contains the things that are involved in the Assignment-1 in the course Natural Language Processing. Convolutional Neural Network have gained a very good popularity in the fields of Artificial Intelligence and Deep Learning. This document provides a complete review on implementing a One-Dimensional Convolutional based Neural Network that helps in predicting the house median values by using few parameters like longitude, latitude, housing median age, total number of rooms, total number of bed rooms, population, number of households and median income. The main reason for implementing the functionality is to generate a Non-Linear Regression model using Deep Learning techniques. We have used a California housing data set namely housing.csv file for implementing the prototype. Coming to implementation part we have done lot of tasks here like printing the first ten rows of the data-set and after that plotting each individual feature of the given values in a sub plot. We have used a lot of functionalities in this process like loading the data-set which is used in preparing the both testing and training data-set. The main Aim of this functionality was to get a very good Mean Squared Error and R-Score values. So, we have done lot of operations like changing the kernel size, changing the learning rate, changing the batch size and changing the number of layers in it. We have represented a step by step process that involved in implementing the code with its corresponding outputs.

keywords used- Epochs, learning rate, batch size, convolutional layer, max pooling layers, input layer, output layer, linear layer, flatten layer, kernel size, average score, R2 Score and ReLu activation function.

II. INTRODUCTION

The Coding part has done on the CO-LAB platform which is provided by the Google. Google CO-LAB usually uses GPU because it has a very fast processing time. First of all we need to connect to the GPU and open the notebook for entering the code and once after finishing the code run the code. we need to import the housing.csv file and upload it.

Artificial Neural Network (ANN)- Artificial Neural

Network (ANN) technique is used for modeling. Artificial Neural Network (ANN) are inspired by the biological neural networks. Artificial Neural Network (ANN) is based on collection of nodes (or) units which are connected known as Artificial Neurons. An Artificial Neuron receives the signal and process it. The main Aim of Artificial Neural Network (ANN) is to solve problems the same way how our human brain does. Artificial Neural Network (ANN) is used in many applications like medical diagnosis, computer vision, speech recognition, social network filtering etc. By using Artificial Neural Network (ANN), we can predict the values and performance of the process with very high accuracy, adequacy etc. Artificial Neural Network (ANN) completely depends on the representative data. It is also confirmed that ANN (Artificial Neural Networks) based models provide efficiency and robustness.

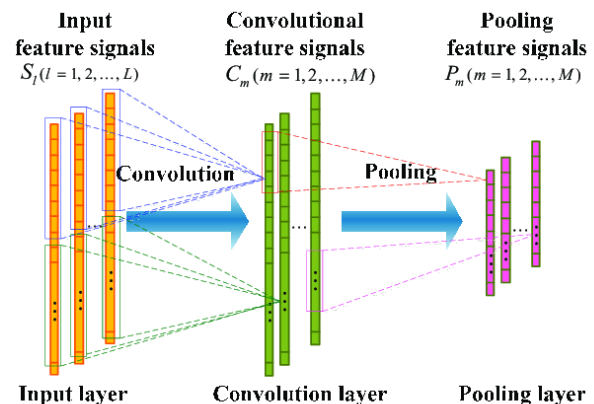


Fig. 1. Figure of CNN

Convolutional Neural Network (CNN):- Convolutional Neural Network (CNN) is a part of Deep Learning. CNN is mainly used in analyzing the imaginary concepts. They have lot of applications in medical images analysis, image classification, image and video recognition and in Natural Language Processing. For the past many years, research on Convolutional Neural Network (CNN) has rapidly increased because of the real-world changes that are taking place in many computer resources and in the memory constraints. The most common use of Convolutional Neural Network (CNN)

is image classification where it identifies the satellite images that identifies the path. A Convolutional Neural Network (CNN) can also be implemented as a U-Net Architecture. A Convolutional Neural Network (CNN) contains an input layer, output layer and multiple hidden layers. The Hidden Layers of a Convolutional Neural Network (CNN) contains a set of Convolutional layers. Usually the Convolutional layer takes the input and pass the result to the next layer. Figure 1 shows the CNN model

Linear Regression- Linear Regression can be defined as a linear approach that is used in modelling the relationship between the dependent variable and independent variables. In case of one Variable it is known as Simple Linear Regression. In case of two variables it is known as Multiple Linear Regression. In Linear Regression, Linear Predictor Functions are used in modeling. Linear Regression has lot of applications which are practical. By using the Least Squares approach the models of Linear Regression are fitted. For not Linear Models, least squares approach is used for fitting the models. A Linear Regression has an equation of $Y = a + bX$. Where a and b are the intercepts, X is the independent variable and Y is the dependent variable. Linear Regression always consists of best fitting straight line all the way through the points. Regression Line is the best fitting line. Figure 2 shows the Linear Regression model

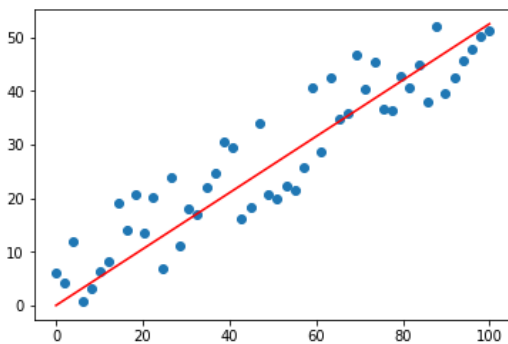


Fig. 2. Example of Linear regression

Kernels- Kernels are important concepts in both Deep Learning and Convolutional Neural Networks. Kernel is mainly used in Statistical Analysis. Kernels are also known as filters as they act as filters on input. Kernel has two dimensions namely height and width. We can change the kernel size based on our output. In my code I have made Kernel size as 1.

Non-Linear Regression- A Non-Linear Regression can be defined as the regression analysis in which the data observed is modeled with the help of the function that is a combination of parameters and on one or more independent variables. By using the successive approximations, the data is fitted. There are several common models used in Non-Linear

Regression like Asymptotic Regression and Growth Model. As the relationship between dependent and independent are not linear these models are known as Non-Linear Regression. Fig 3 shows the non-linear regression model

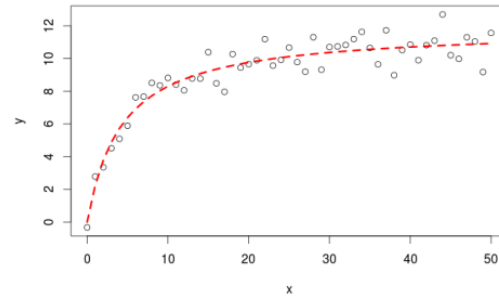


Fig. 3. Example of non linear regression

1-D Convolutional neural network- In order to overcome few limitations that occurs in the 2-D Convolutional Neural Network, 1-D Convolutional Neural Networks comes to the picture. By using 1-D Convolutional Neural Network are used in many real-world applications. Both complexity and computations performance were reduced, so that it is very good to use. 1-D Convolutional Neural Network has lot of advantages. Firstly, its Architecture which is very easy to train the model. Secondly, it has very few hidden layers and neurons.

Inference Time- Inference Time can be defined as the amount of time taken by the source code on two places namely GPU and CPU. The formula for calculating the Inference Time is $X+Y$. X is GPU time and Y is the CPU time. GPU has many advantages than compared to CPU because the GPU runs faster than compared to the CPU.

ReLU- ReLu stands for rectified linear unit. ReLu is an activation function which is defined as $y = \max(0, x)$ mathematically. It is the most common activation function used. ReLu converts the non-linear model to linear model.

R2Score- R2Score is a statistical value that provides us the information about the goodness of a model fit. R2Score is also used as a statistical measure how well the regression predictions the data points.

L1 Loss- L1 Loss functions is a function used in machine learning that is used in minimizing the error. L1 Loss function stands for Least Absolute Derivations.

PyTorch- Py-Torch is a Machine Learning library which is based on the Torch Library. It is an open source. It is written in Python, C++ and CUDA.

Libraries used in our code - We have used a lot of libraries in our code. They are:

- **pandas**-pandas library is used in manipulating the data and for Numeric value analysis.
- **sklearn**- sklearn supports scientific and numeric libraries of Python.
- **numpy**- numpy is used in adding and supporting the large and multi-dimensional arrays or matrices.
- **matplotlib**-numpy is used in adding and supporting the large and multi-dimensional arrays or matrices.
- **torch**- torch provides the N-Dimensional Array.

Literature review: I have chosen the following paper and completely reviewed and did my analysis part.

Paper title- Fatih Ertam, Galip Aydin. Data classification with deep learning using Tensorflow”, IEEE, 2017

Deep Learning has achieved a very high and good success rate in many applications. Deep Learning is preferred where we have a very big data sets because Deep Learning it provides very fast and accurate results. In this paper they have also used Tensorflow which is one of the most famous deep learning libraries used to classify the data-sets. By using the Tensor-flow the authors have studied and compared the effects based on the classification results. The following functions are used like Rectified Linear Unit (ReLU), Hyperbolic Tangent (tanh), Exponential Linear Unit (ELU), Sigmoids, softplus and softsign. In the deep study both Convolutional Neural Network (CNN) and SoftMax classifier are used mainly for deep learning artificial neural network. The results they got at the end are very accurate.

III. PROPOSED MODEL

Deep Learning has achieved a very high and good success rate in many applications. Deep Learning is preferred where we have a very big data sets because Deep Learning it provides very fast and accurate results. In this paper they have also used Tensorflow which is one of the most famous deep learning libraries used to classify the data-sets. By using the Tensor-flow the authors have studied and compared the effects based on the classification results. The following functions are used like Rectified Linear Unit (ReLU), Hyperbolic Tangent (tanh), Exponential Linear Unit (ELU), Sigmoids, softplus and softsign. In the deep study both Convolutional Neural Network (CNN) and SoftMax classifier are used mainly for deep learning artificial neural network. The results they got at the end are very accurate. The key steps involved are:

- **Prerequisites (Importing all the required packages)**
- **Processing our dataset**
- **Creating the Network**
- **Training the model**
- **Testing the model**

In this model we need to import all the prerequisites packages that are necessary for functions. First connect to

GPU runtime and then download the ‘housing.csv’ dataset from D2L and we should upload it in our co-lab notebook and import all the packages like pandas, sklearn, numpy and matplotlib. First of all, lets read and clean our dataset. So to remove any duplicate entries (or) fields we are using dropna() function. we are printing the first 10 rows of the table. So, we have used head (10) and printing it. Then we need to plot the information in the format of the graph Next, we need to split the dataset into the input (X) and output (Y) datasets. We will be predicting the median house value column and the remaining columns will be coming under Y. Finally, we need to split into training and testing portions. We are splitting the dataset so 70 percentage is used for training and 30 percentage is used for testing. After that convert the datasets to numpy arrays using .to numpy function.

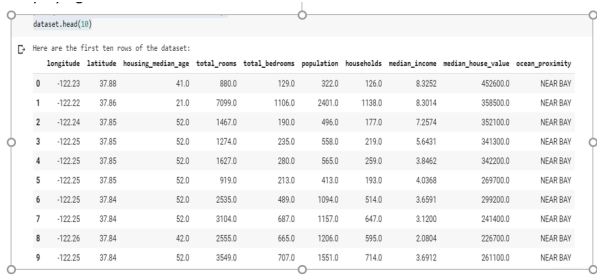
First, let us import all the libraries to build our network. So, importing torch, Conv1d, MaxPool1d, Flatten, Linear, relu, DataLoader and TensorDataSet libraries. Next, we need to define our model. so, we need to define the initialization method and then initialize the super class and store all the parameters. Define the input layer which contains the parameters like channels, output channels and kernel size, Define the max pooling layer, defining two Convolution layers, Defining the flatten layer, Defining the linear layer and Defining the output layer. Define a method which is used to feed the inputs through the model and reshaping the entry so that it can feed the input layer. Next we need to get the output from each and every layer and running them individually.

In Training the models Firstly, we need to import the optimizer and the performance measure packages that we are going to use. We are importing the SGD package for optimisation and L1Loss package for performance measure. Import the R2 score package also. Next we need to define the model by providing the batch size and setting the model to use GPU for processing. I have given batch size as 128. Next let us create a method which is used for running the batches of data via our models. model loss() will return the average L1 Loss and R2 Score. We will be getting the models predictions for the training dataset, models loss and models R2 Score. Next we need to define the number of epochs to train for. I have given 500 epochs. Define the performance measure and optimizer. Next converting the training set into torch variables for our models using GPU as floats. reshape() is used in removing the warning PyTorch outputs. Create a DataLoader to work with our batches and start the loop. It starts calculating the average loss and R2Score for every epoch and generated the output. Finally printing the avg loss and R2Score.

Over /under fitting issue: The scenario in which the model performs well in training time but, doesn't in testing time is know as Over fitting. To avoid this condition I reduced the number of epochs.

Under fitting happens when the model is not performing well in both training and testing phases. To get rid of under fitting, These are the over and under fitting issues

Loading Dataset: To load the first 10 rows of the data sets the read command need to be initiated and the path of the file should be mentioned using csv file.head command is passed and the number of lines (10) should be passed in the parameters to display the first records of the dataset.



	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	222.0	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

Fig. 4. Here are the first ten rows of the dataset:

Loading Graph: The next task was to plot the each feature of the data sets on the separate subplot. In order to do this i used the matplotlib library which is simple and efficient to generate a good graph .

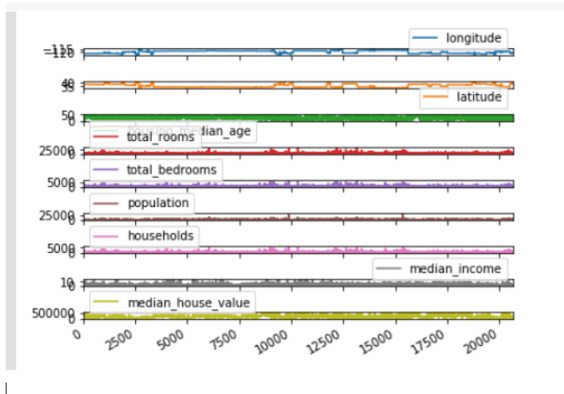


Fig. 5. Plotted graph

Training parameters

The training parameters that are considered for non linear regression model are

- WC = Number of weights of the Convolutional layer.
- K = Size of kernels used in the Convolutional layer
- N = Number of kernels.
- C = Number of channels of the input image.
- Testing the model

Kernel size used : 1

Batch size= 128

Learning rate= 1e-2

Epoch size= 200

IV. DATASET USED

<https://github.com/ageron/handson-ml/tree/master/datasets/housing>

V. APPENDIX

1.Here we define the number of layers and given its corresponding parameters to it. Following is the code for it

```
#verifying the outputs
self.outputs = outputs
#Defining the input layer that has input channels, output c
self.input_layer = Conv1d(inputs, batch_size, 1)
#Defining the max pooling layer that has kernel size
self.max_pooling_layer = MaxPool1d(1)
#Defining the first conv layer
self.conv_layer = Conv1d(batch_size, 128, 1)
#Defining the second conv layer
self.conv_layer2 = Conv1d(128, 64, 1)
#Defining the flatten layer
self.flatten_layer = Flatten()
#Defining the liner layer
self.liner_layer = Linear(64, 32)
#Defining the output layer
self.output_layer = Linear(32, outputs)
#Defining a method feed which is used to feed the inputs to t
def feed(self, input):
    #We are reshaping the entry so that it can feed to the input
    input = input.reshape((self.batch_size, self.inputs, 1))
    #It gets the output from the first layer and run it through
    output = relu(self.input_layer(input))
    #It gets the output from the max pooling layer
    output = self.max_pooling_layer(output)
    #It gets the output from the first convolution layer and ru
    output = relu(self.conv_layer(output))
    #It gets the output from the second convolution layer and r
    output = relu(self.conv_layer2(output))
    #It gets the output from the flatten layer
    output = self.flatten_layer(output)
    #It gets the output from the linear layer and runs it throu
    output = self.liner_layer(output)
```

Fig. 6. Defining all the layers

2.Assigning the epochs and learning rate - Figure 7 represents

```
#It defines the performance measure and the optimizer
optimizer = torch.optim.RMSprop(model.parameters(), lr=1e-2)
#It converts the training set into the torch variables to our model using the GPU as floats
inputs = torch.from_numpy(x_train_np).cuda().float()
#The reshape is used in removing a warning PyTorch outputs
outputs = torch.from_numpy(y_train_np.reshape(y_train_np.shape[0], 1)).cuda().float()
```

Fig. 7. Assigning the epochs and learning rate

3. Assigning the batch size- Figure 8 represents

```
#It Defines the batch_size
batch_size = 128
#It has the parameters like batch_size, X columns
model = CnnRegressor(batch_size, X.shape[1], 1)
#It has the parameters like batch_size, X columns
```

Fig. 8. Assigning the batch size

VI. CONCLUSION

We got the best R2Score for the given model.

The model's L1 loss is: 47853.886884973406

The model's R2 score is: 0.6625881693835639

VII. REFERENCES

- [1] Fatih Ertam, Galip Aydin. **“Data classification with deep learning using Tensorflow”**. IEEE, 2017
- [2] Sunil Kumar Prabhakar, Harikumar Rajaguru. **“EM based non-linear regression and singular value decomposition for epilepsy classification”**. IEEE, 2016
- [3] C. Szegedy, W. Liu, Y. Jia et al. **“Going Deeper with Convolutions”**. IEEE, 2014