Implementation of One Dimension Convolution Neural Network for Prediction of Housing Data

Pankti Joshi
Department of Computer Science
Lakehead University
Thunder Bay, Canada
pjoshi@lakeheadu.ca

***Abstract*—There are a lot of advancements and researches going on with Artificial Neural Network along with multiple layers. This is often referred to as Deep Neural Network as it deals with more profound and multiple hidden layers. It has proved to be an efficient method for processing a massive amount of data in complex ways. The most popular concept in Deep Neural Network is Convolution Neural Network (CNN). CNN is known to have multiple layers, where each layer has its unique properties. The segments included in CNN are Convolution Layer, Non-Linear Layer, Pooling Layer, and Fully Connected Layer. CNN has a wide application in Machine Learning, Computer vision, Deep Learning, Recommendation Systems as well as Natural Language Processing. In this paper, CNN is explained using an example where the data is passed in all the hidden layers to predict median house value. There is a brief explanation regarding the hidden layers and comparative analysis of how do various parameters affect CNN efficiency.**

***Index Terms*—Convolution Neural Networks, Pooling**

I. Introduction

Convolution Neural Network (CNN) has achieved great results for dealing with extensive data, and its applications are enormous in the areas of visual recognition art and systems. Also, it has given definite results in handling massive data and providing accurate output in all the fields ranging from voice recognition to image classification. CNN is made up of a large number of neurons with learnable weights and biases. The primary function of CNN comprises of the neurons which receive inputs, takes a weighted sum of the data, and passes it to the activation function, which is then sent up as an output layer. This whole process has a loss function, which reduces the strength of the output signal. The CNN model helps by reducing the number of parameters in the Artificial Neural Network, which allows researchers to compute complex tasks and mechanisms. This would help to achieve more accurate results. Due to multiple layers in the CNN model, feature abstraction is a significant field that would help to gain definite results for predicting the data and getting the output. Thus, various layers are vital elements to be studied on CNN and its execution. All the layers on CNN have their significance and help to improve the accuracy of the model. Section II provides a short literature review of the work conducted in the research area related to one-dimensional CNN. The details related to the dataset used in the proposed model are provided in Section III. Section IV explains the Proposed model and its architecture. A brief experimental result is provided in Section V, which is followed by Results and Conclusion, after which Appendix is mentioned, which provides explanations for the essential code snippets used in the proposed model.

II. Literature Review

Neural Network is not a new thing in this era, where millions of new researches and explorations are done in the same field. While there are data analytics and predictions are going in this field. Artificial Neural Network and Convolution Neural Network held high importance while solving a lot of real-time problems. It has its roots in all the applications of Computer Science. There are several IEEE resources from which the concept of a one-dimensional convolution neural network was studied. Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi did one of the remarkable concept presentations in the paper " Understanding of a convolution neural network." There is a brief description of the different layers in the CNN model. There are in-depth explanations about convolutions, stride, Pooling layers, Padding, fully-connected layer, as well as CNN architecture [1].

The applications of CNN have wider roots in application fields of Computer Vision, Deep Learning, Pattern Recognition, and more. It is essential to understand the basic architecture of the CNN model, and there can be many variations in the basic model though the basic concepts remain the same [2].

Smith, Alice E., and Anthony K. Mason did research on the real-time example of Cost Estimation. The analytics basically differentiates the output from basic methods- Neural Network and Regression model. The actual dataset was used for the comparison of predictive models in both the arena. The researcher clearly states that the regression model provides accuracy in predicting the cost in terms of scalability, model execution, and creation[7].

Thus there a lot of online tools and software, literature, blogs as well as videos to learn about the Convolution Neural Network Models. There are multiple methods to train a single model and get the output. The following section briefs about the dataset used in the proposed model to get a predicted output.

The dataset has been obtained from the California Housing Dataset from GitHub, available from the University of Porto.

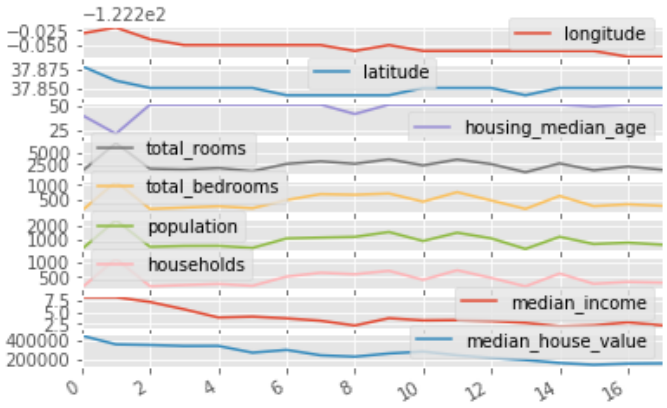

Fig. 1. Features of 18 Data Sample figuree

This data set consists of a range index of 20640 data entries and ten columns with attributes names longitude, latitude, house median age, total rooms, total bedrooms, population, households, median income, median house value, and ocean proximity. The last column, "ocean proximity," is ignored as it consists of textual data and does not contribute towards the output. Thus, the dimension of the dataset is 20433 ×10. The problem that is addressed in the paper is to make a non-linear regression model based on the data provided and implement a one-dimensional convolution-based neural network model using the PyTorch library. The model developed is to predict median house value from parameters longitude, latitude, median housing age, the total number of rooms, the total number of bedrooms, population, number of households, and median income as parameters. The dataset is divided into training and testing data in the ratio of 70:30, and the random state considered as 2003. The primary purpose is to understand different layers and how do various segments affect the accuracy of the model. The trial and error method has been adopted to get a decent score. The following section gives a detailed explanation of the proposed model.

III. Proposed Model

In this paper, the regression-based one-dimensional convolution model has been implemented to predict the median housing values from the parameters provided, which maximum accuracy. A brief description of the dataset has been provided in section III of the paper. The data that is provided is one dimensional; hence the model is implemented using 1D Convolution Neural Network using Pytorch Library. There are different layers through which information has been passed to obtain a higher accuracy score. Primarily, the data has been divided into training and testing data in the ratio of 70 to 30. These data are converted into the fixed batch size and then passed in the model. For the proposed model, the batch size considered is 250. For the proposed model, Adamax optimizer has been used to optimize the output. There are two Pooling layers used while executing the model for output efficiency. MaxPooling and AveragePooling are used to get better results. Fig. 1 depicts the Proposed Architecture of the CNN model with all the layers through which the input data passes through to get the output results.

The first layer in the model is Batch Normalization Layer, which stabilizes the input data for the faster training process. This normalized data is passed to the first convolution layer.

The convolution layer extracts the feature of the data provided, by taking as image matrix and filter or kernel as input. This data is then passed through an activation function, which would map the input to the response variable. In the proposed model, the Relu activation function has been applied to the extracted features. After this, the data is passed to the MaxPooling layer to extract the maximum value from each patch of the feature map. This output from the MaxPooling layer is fed into another 1D Convolution layer, after which additional Relu activation function is applied to the output data for a better score. This data is passed through the dropout layer with a probability of 0.2. The dropout layer would reduce the overfitting problem caused. The optimized output is passed through the second Pooling Layer, AveragePooling, which would extract average values from each patch of the feature map. The output obtained from the Pooling Layer is send through two layers of the flattening layer. The flatten layer would convert the pooled feature map to a single column to be passed to the fully connected layer. The output is passed through a softmax layer; this would allow the neural network to run through a multi-class function. This would enable a clear understanding of the identification of various objects in the data. These obtained features are then passed through a pair of linear layers to reduce to achieve a Linear output. The Loss in the data is calculated using the L1Loss() function; this would compute the Loss value in the output values of the proposed model. R2Score is used to calculate how close the output is to the ideal values. For the proposed model, after passing through all the layers mentioned above, the R2Score and L1Loss() of the model computed comes to 0.77 and 36068.4505. The proposed model consists of 11 layers. The key specifications and major parameters used for the optimized performance of the proposed model are listed in Table 1. For the technical aspect, the proposed model is implemented in Python. The one dimension convolution neural network has been implemented using the PyTorch Library in Python.

TABLE I
Parameters Configurations of Proposed Model

Parameter	Configurations
Optimizer	Adamax
Activation function	ReLU
Normalization	Batch Normalization
Pooling Layers	MaxPooling;AveragePooling
Kernel	1
Learning Rate	0.01
Batch size	250
Epoch	500
Softmax(dimension)	1
Dropout(Probability)	0.2

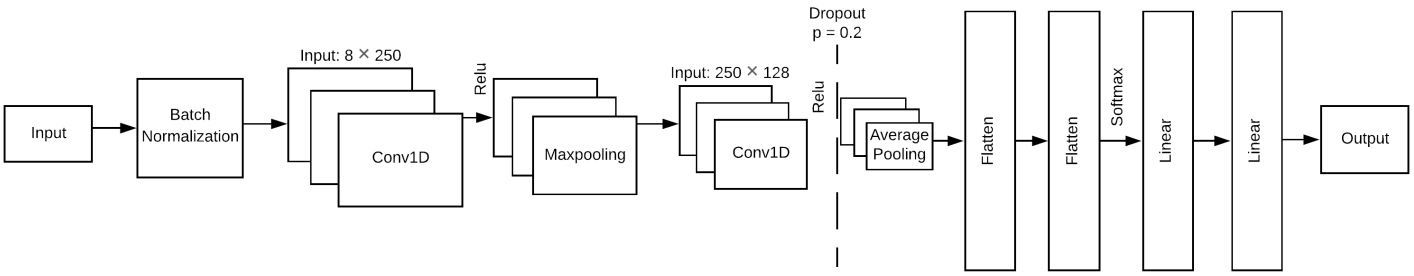

Fig. 2. Proposed CNN Architecture.

A python class is defined as CnnRegressor, which inherits torch.nn.Module. This class has the whole model structure. The init(self, batch size, inputs, outputs) and feed(self, input) function are stated in the python class. The init() function initializes all the CNN layers, and take batch size as the input. The feed() function is used to feed in the data in the model, process the data in all CNN layers and gives the output. The proposed model has been evaluated using the function model loss(model, dataset,train=False,optimizer=None). This function takes the data in batches, runs through the model, and computes out the L1Loss and R2Score of the epochs while training. It passes the batch input to feed() function from which the output computed is compared with the ideal output to compute the L1Loss() and R2Score.

IV. Experimental Calculation

Though different kinds of variations in parameters, we can achieve different scores. This section provides a brief about the equations used and required to compute the L1Loss, R2Score, and Mean Squared Error values. These three are the parameters based on which the output is computed. The higher the R2Score, more accurate results are obtained. Similarly, the L1Loss and Mean Squared Error must be as low as possible to obtain the precise output. Equation (1) provides the formula for computing the L1Loss from the given model. L1Loss function refers to the Loss encountered while executing the proposed model. To get predicted values, the data passes through multiple layers while calculating the output. Here, the equation has the summation from 1 to 'n' specified value, computes the mod of difference of the predicted value from the true value obtained while computing the data. This provides the average L1Loss of the given proposed model.

$$L1Loss = \sum_{i=1}^{n} (y_t - y_p) \tag{1}$$

Equation (2) provides the basic formula for computing the R2Score. R2Score refers to the output precision of the proposed model. The higher the R2Score, the better the accuracy achieved by the proposed model.

$$R2Score = 1 - \frac{First\ sum\ of\ errors}{Second\ sum\ of\ errors} \tag{2}$$

While computing the prediction results, MSE is an important parameter to estimate the average value of squares of errors. Ideally, MSE refers to the average squared difference between the actual and estimated values obtained. It can be referred to as a risk function that corresponds to the expected value of squared error loss.

$$MSE = \frac{1}{N} \sum_{i=1}^{n} (y_i - \hat{y})^2 \tag{3}$$

In all, the lesser the MSE error, the more accurate the computation values and the better the performance of the model. For the proposed model, the MSE computed is 3388064512.0. The model's L1loss and R2Score are 37247.91796875 and 0.76, respectively.

V. Experimental Analysis

To achieve better outputs and to compute the predicted values nearer to the ideal values, various parameters such as number of epochs, learning rates, batch size, number of layers, optimizer used, dimensions of the input and output can be altered. As mentioned in the previous sections, computing better results are done by hit and trial method- by setting different values for various parameters. Thus, the proposed model did go through several use cases to obtain the highest R2Score. Multiple cases experimented are discussed in the section.

Primarily, the proposed model was executed without the Average Pooling layer and 200 epochs. All the other parameters were the same, as mentioned in the proposed model. The model gave R2Score of 0.75 with L1Loss() as 36786.34, while with the average pooling layer, it was 0.74 and 36376.4152, respectively. While changing the value of Dropout to 0.5 along with 200 epochs and without Average Pooling Layer R2Score reduced to 0.72 while L1Loss() came to 36786.34. While reducing epochs to 100 along with the above-changed parameters, the score came to 0.76 while the Loss increased to 38366.39.

Secondly, experimenting with Adam as the optimizer and with 100 epochs, keeping all the other parameters constant as mentioned in the proposed model, the R2Score was 0.75 while Loss increased to 37890.75. Using Adamax as the optimizer and learning rate to 1e-7, L1Loss comes to 38635.50, while

R2Score remains 0.74. While removing the softmax layer to this case, Loss hikes up to 206852.24, and R2Score decreases to -3.18. The scenario mentioned above becomes the worst case compared to all the instances experimented.

For understanding the importance of the softmax layer, just the softmax layer was removed from the mentioned proposed model, and it was observed that Loss enhanced to 40256.72 while R2Score became 0.73.

Removing one flatten layer from the ideal model score remains 0.74, while the L1Loss values increase to 38359.99. Along with that removing the Dropout layer would result in 0.77 and 37584.46. Thus, it can be inferred that the fully-connected layer, like softmax, flatten layers, as well as Dropout, are essential to prevent the loss of data while dealing with CNN models. Additionally, removing the average pooling layer as well as reducing the batch size to 100 would result in 0.73 Score and 38461.57 Loss.

Lastly, removing one linear layer to the ideal proposed model gives the R2Score as 0.71 while L1Loss increases to 40752.87. While removing just a MaxPooling Layer from the proposed model would result in a score of 0.75 while L1Loss again hikes up to 38026.81. Thus, MaxPooling and Linear Layer, too, contributed to preventing the Loss of the data in the proposed model. In all, it can be inferred that all the layers in the model played an essential role in the proposed model of non-regression one-dimensional convolution layer to achieve an optimum output of 0.77 Score value and 36068.4505 Loss.

VI. Conclusion

To conclude the research work, one-dimensional CNN, based on the non-linear regression model. The proposed model used in-total of 11 layers, including two layers, each of the convolution layers, flatten layer, and linear layers. The Adamax optimizer is used in the proposed model. The Max-Pooling and AveragePooling Layers have been implemented in the proposed model. Relu activation function, Dropout with probability 0.2 and softmax with dimension one is used for better lower Loss score and accurate prediction values. The proposed model is 77 percent specific in terms of predicting the median house value.

VII. Appendix

A. CnnRegressor function

In the Python class, CnnRegressor that consists of two functions init () and feed(), is the basic building block of the whole proposed model. The function init() takes batch size as the input parameter. The data are then passed in the convolution neural network model for computing the output values. The size of the input is the batch size provided and passes it through the layers specified in the function. The input data goes through the Batch normalization, where the data is normalized. This 250 size data goes through the convolution layer, max pooling layer, convolution layer. After which the output is reduced in size to 128. Relu activation function is applied after every convolution layer. This data is then filtered through the dropout layer with 0.2 probability. After this, it passes through the average pooling layer, two corresponding Flatten Layers, and Softmax layer. The data is then passed through the Linear layer, where the dimension of the channels decreases from 128 to 64 and 64 to 32 at the output layer. The feed() function runs the input through the model and provides the computed output.

```
#major python class that takes in the input and
    provides the processed output
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
    self.batch_normaliztion_1=BatchNorm1d(inputs)
    self.input_layer = Conv1d(inputs, batch_size,
    kernel_size=1)
    self.max_pooling_layer = MaxPool1d(1)
    self.conv_layer = Conv1d(batch_size, 128, 1)
    self.drop_out_1=Dropout(0.2)
    self.average_pooling_layer = AvgPool1d(1)
    self.flatten_layer = Flatten()
    self.flatten_layer1 = Flatten()
    self.softmax = Softmax(dim=1)
    self.linear_layer = Linear(128, 64)
    self.linear_layer1 = Linear(64, 32)
    self.output_layer = Linear(32, outputs)

  def feed(self, input):
    input = input.reshape((self.batch_size, self.
    inputs, 1))
    output=self.batch_normaliztion_1(input)
    output = relu(self.input_layer(output))
    output = self.max_pooling_layer(output)
    output = relu(self.conv_layer(output))
    output= self.drop_out_1(output)
    output = self.average_pooling_layer(output)
    output = self.flatten_layer(output)
    output = self.flatten_layer1(output)
    output = self.softmax(output)
    output = self.linear_layer(output)
    output = self.linear_layer1(output)
    output = self.output_layer(output)
    return output
```

Listing 1. CnnRegressor Python class

B. Model Loss Functionn

The model loss function calculates the Loss that the model encounters while the data goes through all the layers and finally gives the output. There is always some in data, as it undergoes through several layers to get output data. The accuracy in output is measured based on three functionalities like L1Loss, R2Score and Mean Squared Error values.

```
#model loss is computed while computing the output
def model_loss(model, dataset, train = False,
    optimizer = None):
  performance = L1Loss()
  performance1 = MSELoss()
  score_metric = R2Score()

  avg_loss = 0
  avg_loss1 = 0
  avg_score = 0
  count = 0

  for input, output in iter(dataset):
```

```
    predictions = model.feed(input)
    loss = performance(predictions, output)
    loss1 = performance1(predictions, output)
    score_metric.update([predictions, output])
    score = score_metric.compute()
    if(train):
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()

    avg_loss += loss.item()
    avg_score += score
    avg_loss1 += loss1.item()
    count += 1
  return avg_loss / count, avg_loss1 / count,
    avg_score / count
```

Listing 2. Loss function

REFERENCES

[1][online]. Available: https:// github.com/ageron/handson-ml/tree/master/datasets/housing.csv.

[2]Saad Albawi, Tareq Abed Mohammed, Saad Al-Zawi, Understanding of a convolutional neural network, August 2017.

[3][online]. Available: https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8.

[5][online]. Available: https://keras.io/layers/normalization/.

[6]Babu, Giduthuri Sateesh, Peilin Zhao, and Xiao-Li Li. "Deep convolutional neural network-based regression approach for estimation of remaining useful life." In International conference on database systems for advanced applications, pp. 214-228. Springer, Cham, 2016.

[7]Smith, Alice E., and Anthony K. Mason. "Cost estimation predictive modeling: Regression versus neural network." The Engineering Economist 42, no. 2 (1997): 137-161.

[8] Lathuilière S, Mesejo P, Alameda-Pineda X, Horaud R. A comprehensive analysis of deep regression. IEEE transactions on pattern analysis and machine intelligence. 2019 Apr 11.

[9][online]. Available: https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf.

[10][online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[11][online]. Available: http://cs231n.github.io/convolutional-networks/.

[12][online]. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.