Non-Linear Regression with One-Dimension Convolution Neural Network

Jugal Shah
Department of Computer Science
Lakehead University
Thunder Bay, Canada
jshah5@lakeheadu.ca

***Abstract*—Neural Networks adoption has been growing over the last few decades to mimic the functionality of the human brain. Convolution neural networks represent a class of deep neural networks that are well known for classification problems, majorly in the domain of image classification and text classification. CNN is widely used to perform tasks such as handwritten digit classification, object detection, speech recognition, to name a few. This paper focuses on experimenting with the application of a convolution neural network for a regression-based problem. Regression analysis is primarily employed for predicting or forecasting a dependent variable, for example, company sales number from a series of independent variables, for example, weather, competitor price, and many more. This study aims to predict the house median value from the given longitude, median housing age, the total number of rooms, and a few more independent variables using a regression-based CNN model. The California Housing dataset is used as a data source for the training and testing of the model. The paper additionally provides a comparison of different model build for experimentation purposes. All the models are evaluated on their L1Loss and R^{2}Score. Pytorch Library is been used for this study.**

***Index Terms*—CNN, regression, neural network, kernel, learning rate, R^{2}Score, L1Loss, epoch, SGD**

I. Introduction

Regression models have been widely used over the years for forecasting or predicting parameters that can assist in solving real-world problems. It aids in outlining the relationship between a dependent variable and one or more independent variables. However, with the growing complexity of the structure of data and the ever-increasing accumulation of data in a variety of formats, new challenges are identified more often. Simple regression models cannot resolve these challenges. Many regression models cannot fit this new form of data; hence the need for more complex models has arisen. Over the last few decades, the term neural network and deep learning have become highly popular. The concept of Neural networks originated from the biological neural system. Therefore similar to a human brain, a neural network model comprises a stack of neurons organized in a layered fashion. These layers of interconnected neurons operate together in the network to solve complex problems using a mathematical approach. At the initial stage, a neural network model is highly unstable. Training is essential for the model to perform better. During this training period, the model learns by performing multiple calculations and parameter updates such as calculating error, computing the gradient, back-propagating the gradient, updating the network weight, and various other. These updates and computations add to the complexity of the model and also improves its prediction capabilities considerably, making it a good option for regression problems. In this paper, a Convolution neural network, primarily used for image classification problems, is applied to a regression problem to experiment with a regression-based CNN model for prediction. The CNN model is built to operate on tabular data and perform predictions. Cost estimation being of primary importance for business, this paper proposes a CNN model for the prediction of house median value [1]. One-dimension CNN is used for performing the prediction because of the inputs to the model being a one-dimensional array. The model is build using the Pytorch library and executed in the Google Colab environment. Experimenting with different hyperparameters and CNN layer configuration, the paper proposes a model with L1Loss of 41236.96 and R^{2}Score of 0.73.

II. Background

A general convolution neural network architecture consists of numerous convolution and max-pooling layers followed by few fully-connected layers and a softmax layer. Whereas in a regression-based CNN, the last layer is replaced with a fully-connected regression layer. In the paper [2], such architectures are referred to as vanilla deep regression. The absence of a systematic and statistical analysis of such deep regression-based models has led to the practice of trial and error techniques for improving model performance. The paper addresses the above issue by performing a detailed analysis of vanilla deep regression on computer vision problems. It uses the Biwi and FLD dataset to investigate the impact of network optimizer, batch size, loss function, batch normalization, dropout layer, and few more parameters on the model. Each of the individual specifications mentioned in this paper can be used as a guide in fine-tuning a deep regression network. A complex dynamic data may necessitate the use of a complex model such as CNN. However, in [3], Youshan Zhang and Qi Li have brought in to light the issue of the computational overhead that attached to model training. To tackle this issue, in their paper, they have proposed a hybrid model that extracts features from the data using a regression-based CNN, and an SVR is trained using those features. The model is used to forecast monthly electric consumption. The proposed RCNN-SVR model consists of three pairs of convolution and max pooling and an SVR classifier for prediction. In the end, the paper demonstrates a detailed comparison of the application of a standalone RCNN model, standalone SVR model, and the proposed hybrid model RCNN-SVR to the given problem. The proposed model achieves better accuracy compared to other models with an MSE of 0.8564. Another similar research is presented in [4], in which Salim Malek, Farid Melgani, and Yakoub Bazi propose the application of 2D CNN and 1D CNN along with regressor Gaussian process regression and Support Vector Machine in chemometric analyses from spectroscopic data. Existing research proves the application of Conventional CNN to simple regression; however, architectural modifications are required to work on multi-variate regression. In [5], an updated CNN model is proposed, in which the convolution and pooling filters are used in conjunction with a temporal dimension. The features maps generated by the model are consolidated together and then passed as a single input to the neural network regressor.The model operates on segmented multi-variate time series data required for estimating the Remaining Useful Life (RUL) of a subsystem.

III. Dataset and Preprocessing

In this study, the California Housing dataset is used for predicting the house median value given the longitude, latitude, house median age total rooms, total bedrooms population, households, and the median income. The dimension of the dataset is 20433×10 The last column of the dataset, labeled as ocean proximity, is not considered for prediction as the field only holds textual data and does not contribute to the prediction of the house median value. Pandas library is utilized for reading the dataset. Fig. 1 below shows a sample of all the features of the dataset plotted on separate sub-plots using the matplotlib library.

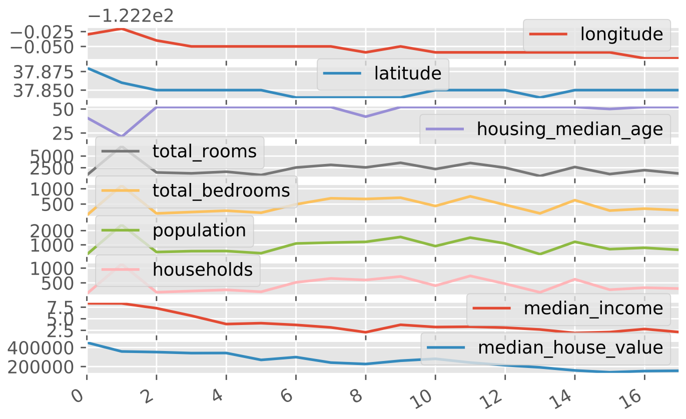

Fig. 1. First eighteen samples of dataset.

As part of data preprocessing, all the missing values are removed from the dataset. Post preprocessing the data is partitioned into training and testing using the train test split functionality form sklearn library. The ratio of training to testing data is 70:30. A random state of 2003 is set so that during each execution, the training and testing data is not altered. The training and testing data is converted to a NumPy array using the NumPy library for the ease of data manipulation.

IV. Proposed Model

In this paper, a regression-based CNN model is proposed for predicting the house median value. Fig. 2 shows the complete architecture of the CNN model. The inputs supplied to the model are one-dimension; therefore we are using the Conv1D package form the PyTorch library. The training and testing data are first converted into batches of fixed size before passing to the model. Due to each feature of the dataset being in a different numeric range, the first layer, i.e., the Batch normalization layer normalizes the input before passing it to the convolution layer. The normalized data is then forwarded to the convolution layer that runs filters of kernel size one over the data to generate features. Activation function Leaky ReLU is applied to the produced features, and then the features are passed to the max pooling layer for down sampling. The output produced from the max pooling layer is then again forwarded to a set of convolution layer, activation function, and max pooling layer. The final output is flattened into a single vector and then passed to a pair of linear layer for linear transformation and achieve a single output. The application of the Leaky ReLU function adds non-Linearity to the model, which is essential to understand the intricate relationship between the input parameters. The proposed model is evaluated using L1Loss and R^2Score. The L1Loss of the model 41236.96 and R^2Score is 0.73.

A. Parameters Configurations

The proposed model performs the best compared to different models that are mentioned in the Experiment Analysis section of the paper. Hyper Parameters and key Configuration that contributed to the performance of the model are shown in the Table 1 below.

TABLE I
Proposed Model Parameters Configurations

Parameter	Configurations
Kernel	1
Learning Rate	0.01
Batch size	64
Epoch	100
Optimizer	Adam
Activation function	Leaky ReLU

B. Model Implementation

From a practical perspective, the model is defined using a python class, with each layer imported from the PyTorch library. The python class is labeled as CnnRegressor, and it inherits the torch.nn.Module. The class encapsulates the complete structure of the model. It consists of below two

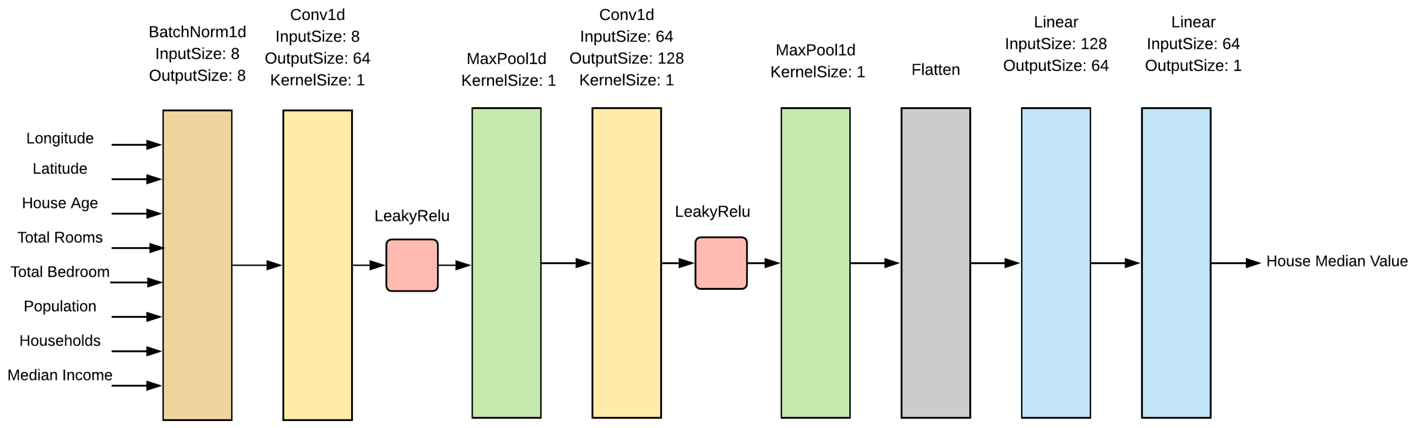

Fig. 2. Proposed CNN Architecture.

functions.The actual code snippet for the functions is given in the Appendix.

- __init__ (self,batch size, inputs, outputs): *The __init__ (self,batch size, inputs, outputs)* function is called for the initializing all layers of the CNN model, accepting as input the batch size, the input size, and the output size.
- feed(self, input): The *feed(self, input)* function feeds the inputs through the model. It takes the data, process it through all the CNN layers, and returns the output.

C. Model Evaluation

As stated previously R^2 Score and L1Loss are used to evaluate the model. Equation (1) denotes the R^2 Score formula. In (1) A_j is the ground truth, P_j is the predicted value, and $\bar{A}$ is the mean of the ground truth.

$$1 - \frac{\sum_{j=1}^{n}(A_j - P_j)^2}{\sum_{j=1}^{n}(A_j - \bar{A})^2} \tag{1}$$

Equation (2) denotes the L1Loss Score formula. In (2) input x is compared against output y for a batch size of N

$$\ell(x,y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = |x_n - y_n| \tag{2}$$

To evaluate the proposed model the function *model_loss(model,dataset,train=False,optimizer=None)* runs batches of data through the model and return the L1Loss and R^{2}Score for each of the epoch during training. It passes the input as a batch to the feed function. Upon receiving the output from the feed function, it compares the output with the expected output to calculate L1Loss and R^{2}Score. When the function is called with the *train* parameter set as true, it performs the below additional steps for each batch to ensure effective training of the model.

- Clear the errors to they do not cummulate
- Compute the gradient for the optimizer
- Update the model parameters based on the gradient using the optimizer

V. Experiment Analysis

The proposed model was built after experimenting with different combinations of all the parameters mentioned in previous section and the organization of the layers. During this study, multiple models were trained, and test results were compared. In this section, the analysis would be narrowed down to four CNN models, including the proposed model, each of which has been a breakthrough and served as a platform for additional achievement. The following sections give an overview of each of the model.

A. Base Model

In the base model CNN architecture, the batch normalization layer and the last max pooling layer are absent. In this model Stochastic gradient descent optimizer was used. Table 2 below shows the hyper-parameters combination set for the base model. The model gave an L1Loss of 72462.75 and an R^{2}Score of 0.34.

TABLE II
Base Model Parameters Configurations

Parameter	Configurations
Kernel	1
Learning Rate	0.00001
Batch size	64
Epoch	100
Optimizer	SGD
Activation function	ReLU

B. Model A

In model A a batch normalization layer as the first layer and dropout layers were added between the convolution layer and the max pooling layer post-activation function. A dropout of 0.2 was added. The dropout helps in preventing over-fitting by deactivating 20 percent of neurons in each iteration. However, the model performed extremely bad with L1Loss of 206671.68 and an R^{2}Score of -3.17, needing a high count of epoch to train better. Table 3 below shows the hyper-parameters combination set for model A.

TABLE III
MODEL A PARAMETERS CONFIGURATIONS

Parameter	Configurations
Kernel	1
Learning Rate	0.00001
Batch size	64
Epoch	100
Optimizer	SGD
Activation function	ReLU

C. Model B

Model B architecture was similar to model A. However, the optimizer was modified to Adam, and the learning rate was updated to 0.01. This model performed extremely well compared to model B and gave an L1Loss of 48756.32 and an R^2Score of 0.60. The Table 4 below shows the hyper-parameters combination set for model B.

TABLE IV
MODEL B PARAMETERS CONFIGURATIONS

Parameter	Configurations
Kernel	1
Learning Rate	0.01
Batch size	64
Epoch	100
Optimizer	Adam
Activation function	ReLU

D. Proposed Model

In the proposed Model the dropout layer is removed. The training results were compared against testing results to ensure no over-fitting has occurred.Table 5 below presents a comparison of all the models

TABLE V
MODEL COMPARISON

Model	L1Loss	R^2 Score
Base Model	72462.75	0.34
Model A	206671.68	-3.17
Model B	48756.32	0.60
Proposed Model	41236.96	0.73

VI. CONCLUSION

This paper presents the application of one-dimensional convolution neural networks, to solve the regression problem of predicting house median value. The study highlights the fact that mere changes to factors like network architecture, hyper-parameters can offer notable outcomes. The proposed model was a result of rigorous experimentation that revolved around working with numerous combinations of hyperparameters and the organization of the CNN layers. A comprehensive analysis of training and testing results produced in various trails helped in understanding the impact of each configuration, which further served essential in building a better model.

APPENDIX

A. CnnRegressor

```
#CnnRegressor Inherit the torch.nn.Module
class CnnRegressor(torch.nn.Module):

  # Define the initialization method
  def __init__(self,batch_size,inputs,outputs):

    #Call to the Constructor of torch.nn.Module
    super(CnnRegressor,self).__init__()

    #Initialization
    self.batch_size=batch_size
    self.inputs=inputs
    self.outputs=outputs

    #Batch Normalization
    self.batch_normaliztion_1=BatchNorm1d(inputs)
    #Convolution Layer(input channels,output
    channels,kernel size)
    self.input_layer=Conv1d(inputs,batch_size,1)
    #MaxPooling Layer
    self.max_pooling_layer_1=MaxPool1d(1)

    #Convolution Layer(input channels,output
    channels,kernel size)
    self.conv_layer=Conv1d(batch_size,128,1)
    #MaxPooling Layer(kernel size)
    self.max_pooling_layer_2=MaxPool1d(1)

    #Flatten Layer
    self.flatten_layer=Flatten()
    #Linear Layer(input channels,output channels)
    self.linear_layer=Linear(128,64)
    #Output Layer(input channels,output channels)
    self.output_layer=Linear(64,outputs)


  #Function to feed input through the model
  def feed(self,input):

    #Reshaping the input is required as Conv1d Layer
     expectes a 3D input
    input=input.reshape((self.batch_size,self.inputs
    ,1))

    #Batch Normalization
    output=self.batch_normaliztion_1(input)

    #Convolution Layer -> LeakyReLU function->
    MaxPooling Layer
    output=leaky_relu(self.input_layer(output))
    output=self.max_pooling_layer_1(output)

    #Convolution Layer -> LeakyReLU function->
    MaxPooling Layer
    output=leaky_relu(self.conv_layer(output))
    output=self.max_pooling_layer_2(output)

    #Flatten->Linear Layer -> Linear Layer
    output=self.flatten_layer(output)
    output=self.linear_layer(output)
    output=self.output_layer(output)

    #Model Output
    return output
```

Listing 1. CnnRegressor Class

B. *model_loss*

```
def model_loss(model,dataset,train=False,optimizer=
    None):

  performance=L1Loss()
  meansquareperf=MSELoss()
  score_metric= R2Score()
  avg_loss=0
  avg_meansquareloss=0
  avg_score=0
  count=0

  for input,output in iter(dataset):
    #Get the model prediction for the training
    dataset
    predictions=model.feed(input)
    #Get model Loss
    loss=performance(predictions,output)
    meansquareloss=meansquareperf(predictions,output
    )
    #Get Model R^2Score
    score_metric.update([predictions,output])
    score = score_metric.compute()
    if(train):
      #Clear the errors to they do not cummulate
      optimizer.zero_grad()
      #Compute the gradient for the optimizer
      loss.backward()
      #Update the model parameters based on the
    gradient using the optimizer
      optimizer.step()

    #Store the Loss and R2Score and update the
    counter
    avg_loss+=loss.item()
    avg_meansquareloss+=meansquareloss.item()
    avg_score+=score
    count+=1

  return avg_loss/count, avg_score/count ,
    avg_meansquareloss/count
```

Listing 2. model_loss function

REFERENCES

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

[2] 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.

[3] Zhang, Youshan, and Qi Li. ”A regressive convolution neural network and support vector regression model for electricity consumption forecasting.” In Future of Information and Communication Conference, pp. 33-45. Springer, Cham, 2019.

[4] Malek, Salim, Farid Melgani, and Yakoub Bazi. ”One-dimensional convolutional neural networks for spectroscopic signal regression.” Journal of Chemometrics 32, no. 5 (2018): e2977.

[5] 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.