Median Housing Price Prediction using NLP

Vikas Trikha
Department of Computer Science
Lakehead University
Thunder Bay, Canada
vtrikha@lakeheadu.ca

***Abstract*—This paper attempts to explain the concepts of neural networks using various layers and try to build a non-linear regression model to predict the median house value using other column's data from the provided dataset. The solution has been proposed considering several typical issues in mind, out of which few are modular coding, overfitting or underfitting problem, exploding gradient issue, kernel size, inference time, and several trainable parameters. The problem statement has been tackled by building an 8-layered neural network in which a sequence of convolutional layer, linear layer, max-pooling layer, average pooling layer, flatten layer is used along with input and output layer. The performance of this model is evaluated on two qualitative parameters which are R^2 and MSE score, aiming to increase the accuracy.**

***Index Terms*—Non-Linear regression model, overfitting, underfitting, modular coding, convolutional layer, linear layer, max-pooling layer, average-pooling layer, neural network, R^2 score, MSE score**

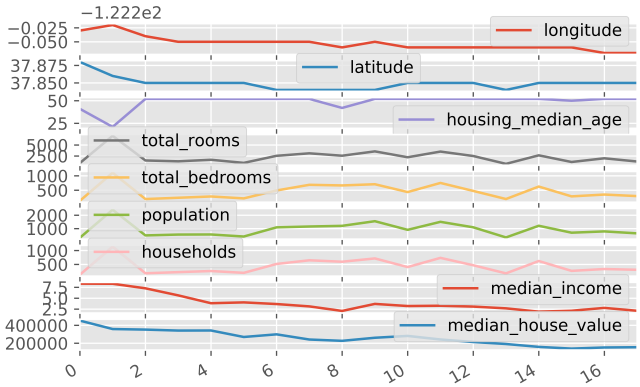

Fig. 1. Example plot of matplotlib using "ggplot"

I. Introduction

Before beginning with developing a non-linear regression deep learning model, the first and most important task to load the dataset in the google colab directory which could be done easily with the help of read_csv command. Once the dataset is loaded, data cleaning and pre-processing can be done in which all the missing values (N/A values) can either be filled or dropped out which entirely depends upon the model requirements. Once data is loaded, data visualization operations can be applied on data for getting more clarity about the data that has to be dealt with. This data visualization can be done with the help of python and pandas libraries which in this case is "matplotlib". Matplotlib is extensively used in python for visualizing numerical mathematics and acts as an object-oriented API for embedding plots using a package known as ggplot. Fig. 1 displays the example plot that is embedded by using the matplotlib library. For a model to be effective, it is extremely necessary to slice the data into two parts which are train data and test data and make sure to never train on test data which can eventually lead to surprisingly high scores in the evaluation matrix. The training set has to be examined exactly considering overtraining the data would appear in overfitting or underfitting which in any case has to be avoided. Overfitting refers to a model that models the training data extremely well whereas Underfitting refers to a model that can neither model the training data nor conclude to fresh data. . To inaugurate with the implementation of a one-dimensional convolution-based network, let's discuss the dataset attributes. The dataset provided in file housing.csv contains 10 columns with 20,641 records based upon which a neural network has to be build which predicts median house value using other columns data like longitude, latitude, housing median age, total number of rooms, total number of bedrooms, population, number of households, and median income. The first step before devising was to break the data in the ratio of 70:30 in which 70% as training set and 30% was test set, and that can be obtained with the help of python's NumPy library. Further, going on convolutional neural network, it indicates that the system engages a mathematical operation called "convolution" which uses a special kind of linear activity in neural networks instead of general matrix multiplication. CNN generally consists of an input layer along with several hidden layers and an output layer. Now data is inputted in a neural network in the form of batches where input will be convolved and would pass the result to the next layer.[1] To reduce the dimensionality of the data, convolutional layers are accompanied by pooling layers, which can be varied as per the requirements such as a max-pooling layer or average pooling layer. CNN is a fully connected feed-forward neural network, which usually appends a fully connected layer after flattening the matrix to connect every neuron in one layer to every neuron in another layer, which is similar to the policy of Multilayer perceptron neural network.

II. Literature Review

A tremendous amount of work has been done on artificial neural networks, and most of it is into implementation. [5] Regression techniques are widely employed to solve the goal of predicting values from a given set of continuous values. For optimizing neural network loss functions, gradient descent was the first-order iterative optimization algorithm used to get derivative-free optimizations. But over time, different optimizers were used with an adaptive learning rate such as Adagrad, RMSprop, Adadelta, and Adam. Adaptive Moments (Adam) is an update to RMSprop where momentum is incorporated whereas in RMSprop is a modification of Adagrad, which shrinks the learning rate and considers the new gradients. Adadelta is an extension of Adagrad, which doesn't require any manual tuning of the learning rate. With a bunch of modifications acquainted in the optimizer, something was necessitated to fasten the confluence in the network and normalizes the data, and that function was Batch Normalization. To calculate the efficiency of a system, the loss function is employed at the training time. The baseline loss functions can be MSE (mean squared error), MAE (mean absolute error), or the Huber loss. Since convolutional networks are having a high number of parameters, it is recumbent to overfitting, and thereby conventional regularization techniques have to be used, and that is where loss and batch normalization appears in to strike. Following the bottom to top approach, Supervised machine learning is broadly categorized into classification and regression where regression analysis is a form of predictive modeling procedure which investigates the association within an independent and dependent variable in a given dataset whereas classification is used to distribute the data into two classes based upon the dataset provided. There are numerous regression techniques popularly practiced, out of which few are linear regression, logistic regression, polynomial regression, stepwise regression, or ridge regression. Linear regression aims to define the direct relationship between the independent and dependent variables, and it subtly becomes difficult in case of complex outliers the presence of multiple independent variables lineages towards the forward selection or backward elimination approach for selection of most significant independent variables. Non-linear regression consists of observational data are reduced by a function, which is a nonlinear combination of the model parameters and depends on one or more independent variables. The data are fitted by a method of succeeding approximations.

III. Proposed Methodology

Several neural networks are applied in real-time applications like face detection, pattern recognization, or prediction of any values. In this paper, a proposed methodology has been discussed to predict the house median value, and evaluation has been performed using various optimizers or activation functions to measure the difference of accuracy between the results upon changes being adhered to in the network. [2] Batch normalization is used along with the input layer to diminish the measure by what the hidden unit values shift around (covariance shift) and to increase the learning power of a network. Batch normalization enables each layer of a network to study by itself a concisely more independently of other layers. To improve the resistance of a neural network, batch normalization normalizes the output of a previous activation layer by deducting the batch mean and dividing by the batch standard deviation. However, if SGD optimizers are applied with batch normalization, it undoes the normalization if intended to minimize the loss function, Thereby in this assignment, adam optimizer has been used with batch normalization to increase the accuracy of the proposed artificial neural network. Along with checking on activation functions and optimizers, some tests were conducted on changing the learning rate, too, since it is an essential hyperparameter of the machine learning model. [3] After holding various runs with some changes in the learning rate, it was concluded that sometimes tremendous learning rate could result in unstable training and tiny learning rate result in failure to train the dataset thoroughly, thereby a balanced learning rate is required to train the model correctly and achieve the best results for the same. There are multiple reasons to avoid using sigmoid and tanh activation function in this particular assignment, out of which one of the most prominent reasons was to prevent exploding or vanishing gradient problems. Therefore Rectified Linear Unit (ReLu) activation function was preferred over others. [4] The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. The rectified linear activation is the default activation when developing multilayer Perceptron and convolutional neural networks. Besides this, there are some additional layers in the convolutional network which are pooling layer and flattening layer. Pooling is the process of merging which basically aims to reduce the size of the data. On moving the stride from one window to another either max pooling or average pooling layer can be applied. [5] Flattening layer is the last stage of CNN which performs the conversion of data in to a 1-Dimensional array for inputting it to the next layer. Linear layer has been added in the model to add linear transformation for the data which means a single layer of linear neurons has been added to the network. After the model is built, it comes the time to train it by passing a number of epochs. Epochs in machine learning generally refer to the number of passes through the entire training set the machine learning algorithm has completed.

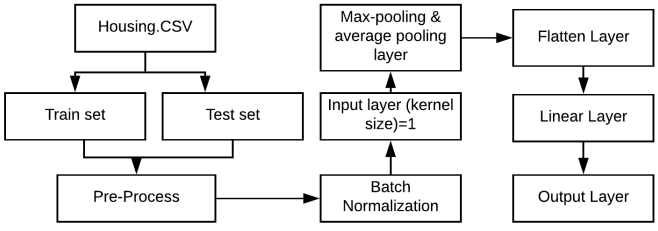

Fig. 2. Overview of Proposed Method

IV. Results & Evaluation

Moving on to the next phase, the developed model was tested on two basic qualitative parameters, which were MSE error and R$\hat{2}$ score. Initially, the model was trained with SGD optimizer, which gave an accuracy of about 30%. Later different optimizers were applied like adam and adadelta, which improved the accuracy a bit, but the major enhancement was observed when batch normalization was added on top of the model along with adam optimizer. Later, some more experiments were performed by changing the learning rates of the optimizer to confirm the full utilizations of the optimizers.

TABLE I
EVALUATION MATRIX USING SGD AND ADAM OPTIMIZER

Optimizer	Learning Rate	R^2 Score
SGD	0.01	Nan
SGD	0.001	Nan
SGD	0.00001	-3.2
Adam	0.01	0.76
Adam	0.001	0.71
Adam	0.00001	-3.07

The final results are based upon two factors, which are MSE and R^2 score, where MSE stands for Mean Standard Error, and it can never be in negative since the values are being squared and then is divided by the total number of samples. Refer to the equation below for more precision:

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

Whereas R^2 score is the coefficient of determination, which is calculated by the sum of the second errors and subtracting the derivation from 1. Find below the equation for the same:

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

R-squared is a statistical measure of how nearby the data are to the fitted regression line. Using batch normalization with adam optimizer at learning rate= 0.01 with batch size= 256 gave the best R^2 score which was 76%. Specific changes were committed in batch size, too, to analytically see the impact of changing batch size on the R-squared value which is precisely explained in table II, The optimizer used was adam only with Learning rate = 0.01. The model was trained throughout with 100 epochs to avoid the issue of overfitting and underfitting.

TABLE II
EFFECT OF CHANGING BATCH SIZE ON R^2 SCORE

Batch Size	R^2 Score
64	73.44
128	75.03
256	76.06
512	76.63

V. conclusion

The problem statement was solved using Convolutional Neural network to compute the predicition of housing mean value. Concluding it, the network designed consisted of input and output layer along with seven other layers to improve the efficiency of the layer with relu activation function. The model was rigorously tested by changing some factors aiming to improve the overall accuracy of the model. The changes have mainly done consisted of addition and removal of batch normalization layer, adding multiple layers, changing optimizers with varied learning rates and changing batch size, and after all the modifications, the best accuracy achieved was 76 percent in R-squared value. The issues like overfitting and underfitting or exploding gradient issues have been taken into consideration. This non-linear regression model works well with a combination of layers and results in a good fit for prediction problems.

Appendix

```
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
# defining batch normalization layer
    self.batch_normalization= BatchNorm1d(inputs)
# defining input layer with kernel size=1
    self.input_layer = Conv1d(inputs, batch_size,
    kernel_size=1)
# defining max pooling layer
    self.max_pooling_layer = MaxPool1d(1)
# defining average pooling layer
    self.avg_pooling_layer = AvgPool1d(1)
# defining convolution layer with 128 channels
    self.conv_layer = Conv1d(batch_size, 128, 1)
# defining flatten layer to reduce dimensionality
    self.flatten_layer = Flatten()
#defining linear layer to derive linearity
    self.linear_layer = Linear(128, 64)
#Defining another linear layer to reduce channels
    self.linear_layer2 = Linear(64, 32)
#defining output layer
    self.output_layer = Linear(32, outputs)

  def feed(self, input):
    input = input.reshape((self.batch_size, self.
    inputs, 1))

    output=self.batch_normalization(input)
#applying relu activation function to get the output
    output = relu(self.input_layer(output))
#Giving the output of one layer as input to the
    other layer
    output = self.max_pooling_layer(output)

    output = self.avg_pooling_layer(output)

    output = relu(self.conv_layer(output))

    output = self.flatten_layer(output)

    output = self.linear_layer(output)

    output = self.linear_layer2(output)

    output = self.output_layer(output)
```

```
    return output

    def model_loss(model, dataset, train = False,
    optimizer = None):
  performance = MSELoss()
  score_metric = R2Score()

  avg_loss = 0
  avg_score = 0
  count = 0

  for input, output in iter(dataset):
#get the model's prediction for training dataset
    predictions = model.feed(input)
#getting model's loss
    loss = performance(predictions, output)

    score_metric.update([predictions, output])
    score = score_metric.compute()

    if(train):
      #clearing errors
      optimizer.zero_grad()
      #compute gradients
      loss.backward()

      optimizer.step()
#store loss and increment counter
    avg_loss += loss.item()
    avg_score += score
    count += 1

  return avg_loss / count, avg_score / count

  # Declare number of epochs you want to train the
    model for
epochs = 100
# declare optimizer for performance
optimizer = Adam(model.parameters(), lr = 1e-2, eps
    =1e-08)

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 to compute with batches
tensor = TensorDataset(inputs, outputs)
loader = DataLoader(tensor, batch_size, shuffle =
    True, drop_last = True)

lossData = []
R2ScoreData = []
epochData = []
#Start the training loop
for epoch in range(epochs):
  avg_loss, avg_r2_score = model_loss(model, loader,
     train = True, optimizer = optimizer)
#output the losses
  print("Epoch " + str(epoch + 1) + ":\n\tMSELoss =
    " + str(avg_loss) + "\n\tR^2Score = " + str(
    avg_r2_score))

  epochData.append(epoch + 1)
  lossData.append(avg_loss)
  R2ScoreData.append(avg_r2_score)
```

Listing 1. Code for CNN Regressor and computing model loss and training data

REFERENCES

[1] "Convolutional neural network". wikipedia.org. https://en.wikipedia.org/wiki/Convolutional_neural_network#Convolutional (Accessed Feb 11, 2020).

[2] F D. "Batch Normalization". towardsdatascience.com. https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c (Accessed Feb11, 2020)

[3] Jason Brownlee "dynamics of learning rate". machinelearningmastery.com. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (Accessed Feb 11, 2020)

[4] Jason Brownlee "ReLu activation function". machinelearningmastery.com. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (Accessed Feb 11, 2020)

[5] Jiwen Jeong "CNN". towardsdatascience.com. https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480 (Accessed Feb 11, 2020)