Miami University

Department of Electrical and Computer Engineering

-Simple Deep Learning-

-SmartNet Lab Report-

Build a simple square prediction model
using Deep Learning from Tensorflow library

Monday, February 8, 2022

Written By:
-Loc Tran-

Date: February 8, 2022

Contents

1 Introduction

In this assignment, I am going to build a simple model to predict square values from each number in a range from 0 to 20. By using Deep Learning knowledge, I will apply a Neural Network architecture which consists of 2 hidden layers to get a result: drawing a line to fit the data set.

2 Pre-processing

To create a model, I need to prepare the dataset for it so that the I can graph them. I created a set of 2 variable X and Y (X is independent variable while Y is dependent variable) in a csv file name *trainingdata.csv*. In this file, there are 2 columns representing for X and Y values and each row is an example (for example: 2, 4).

```
X,Y
0,0
0.5,0.25
1,1
1.5,2.25
2,4
2.25,5.0625
3,9
3.2,10.24
4,16
4.5,20.25
5,25
5.5,30.25
6,36
6.1,37.21
7,49
7.5,56.25
8,64
8.4,70.56
9,81
9.4,88.36
10,100
10.5,110.25
11,121
12,144
13,169
14,196
15,225
16,256
17,289
18,324
19,361
20,400
```

Figure 2.1: Data set

After that, I create an object name *square* to read the data from file using **Panda** library and divide it into 2 different array, X and Y, before reshaping them into 2-dimensional vector. Then, I used them to make a graph for easier visualization.

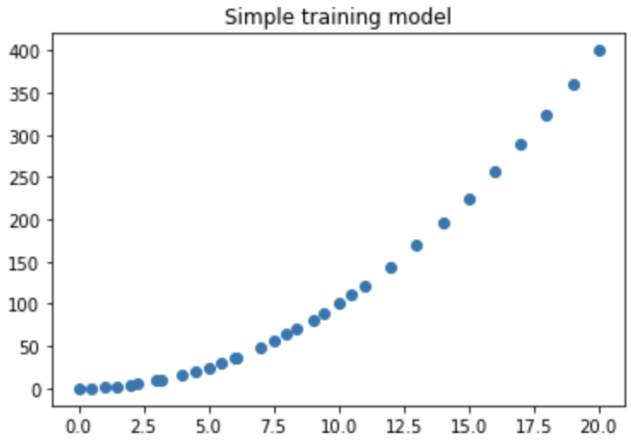

Figure 2.2: Visualization

3 Implement Model

This is the most important part of the project, when I needed to build a model based on existing data set so that it can make further prediction from another example. To build this model, I used **Tensorflow** library as an consistent method to make the prediction with a few lines of code, which help developers design and maintain the project easier. My model was built under Neural Network structure (like human brain) in which each layer will receive inputs from previous, plug them in activation function, calculate the loss function, and return to backward-propagation to find gradient descent, which can optimize the weight(W) and bias(b) parameters.

In the picture below, it can be seen that the model consists of 2 hidden layers. Both of them have **100** units with **relu** as their activation function. To calculate the loss function, I use **Mean Square Error** which measures the amount of error in statistical models. It assesses the average squared difference between the observed and predicted values in order to tell us how close the sketching to the set of points. I also use **Adam** for optimization and set the learning rate to **0.01** so that my prediction can fit better and learn better to the data set. The whole process will be repeatedly implemented in **epoch = 1000** times to get the best result.

```
# Create a 2-hidden layer model to predict the data
"""
    Hidden layer #1 has 100 units and uses relu activation function
    Hidden layer #2 has 100 units and uses relu activation function
"""
model = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation = 'relu'),
    tf.keras.layers.Dense(100, activation = 'relu'),
    tf.keras.layers.Dense(1)
])

# Optimize the model using 'mean squared error' loss function and 'Adam' optimizing method
model.compile(loss = 'mean_squared_error',
              optimizer = tf.keras.optimizers.Adam(0.01))

model.fit(X, Y, epochs = 1000, verbose = False)
```

Figure 3.1: Model Implementation

4 Visualizing Result

After building the model, I came to visualization part to check the result. But before that, I needed to reshape to predicted value *Ypred* because during the process, *Ypred* is a list of predicted output, in which each element is a 2D numpy array. In order to be able to draw the curve, reshaping is necessary so that the predicted values can have the same shapes to those in X (see the attached python file).

The final result is show below:

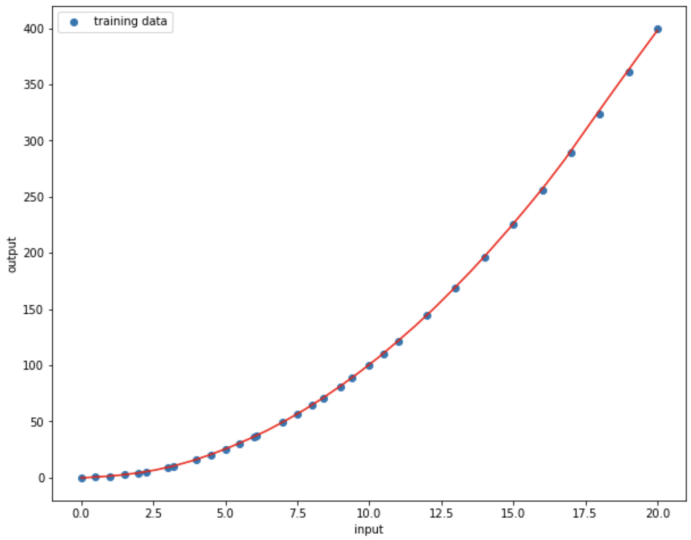

Figure 4.1: Final Result

5 Test Different Parameters and HyperParameters

In this section, I will try to change some values in parameters and hyperparameters to check how is the model look like compared to the above model.

5.1 Change number of layer units

Try setting number of layer units = 10:

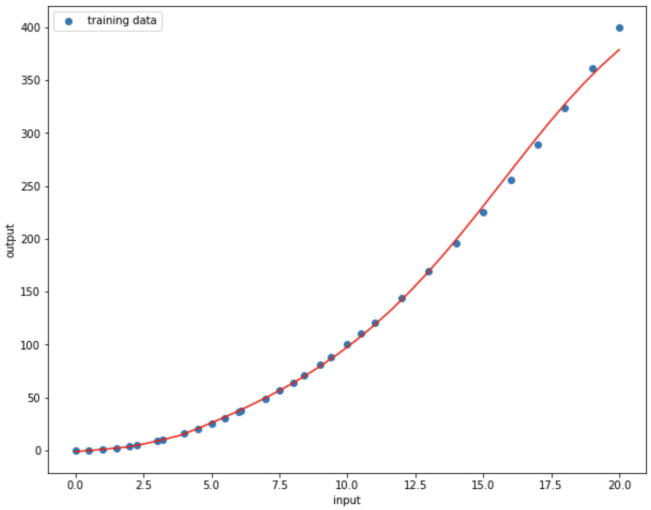

Figure 5.1: N = 10

5.2 Change other activation function

Try setting 1st function = "sigmoid" and 2nd function = "sigmoid":

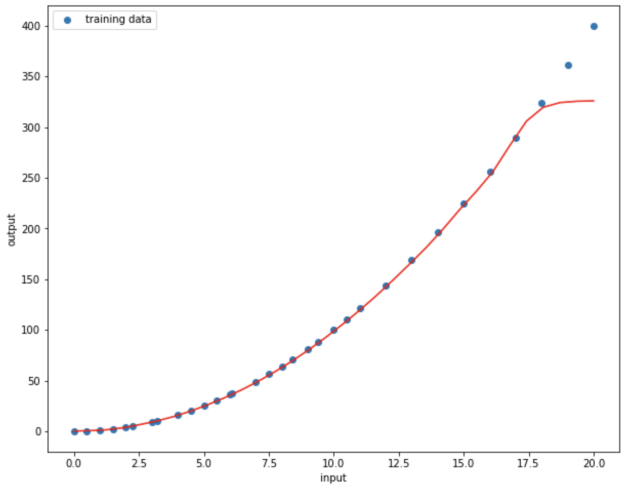

Figure 5.2: activation functions = "sigmoid"

5.3 Change epoch size

Try setting epoch = 100:

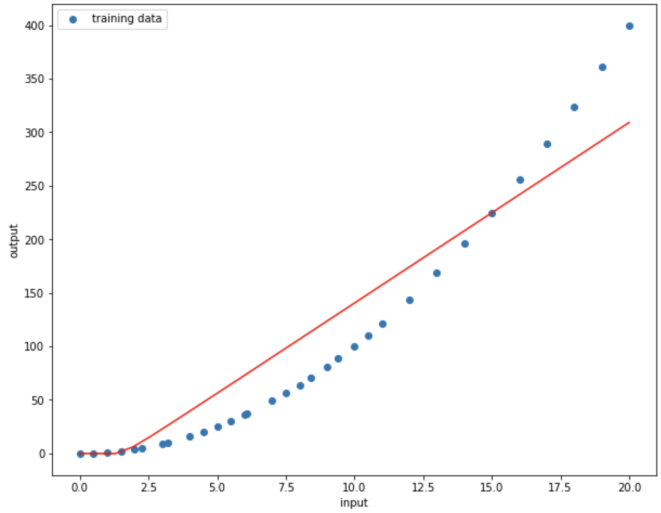

Figure 5.3: epoch = 100

5.4 Worst case scenerio

Try setting epoch = 100, activation functions = "sigmoid", number of layer units = 10:

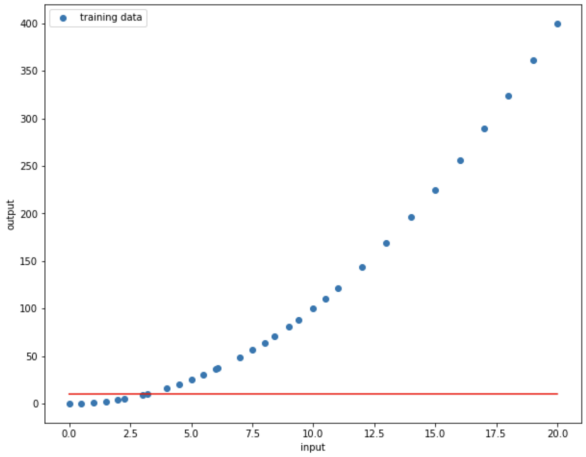

Figure 5.4: worst case scenerio

6 Conclusion

From the given data and the corresponding graph in each case from the previous section, I can conclude that in order to make a good data-fitting model, using appropriate parameters and hyperparameters is very important. The number of layer units should be set to 100, the activation function should be "relu" and the epoch size should be at least 1000. Changing the epoch size might cause the worst model when it change the whole curve into a linear model. Beside, some value which play crucial roles in fitting a good prediction like the learning rate, kind of loss function, or optimization function like Adam also need to be paid attention.