Deep Learning Approach for RSS Prediction

Priyansh
2021183
priyansh21183@iiitd.ac

Rajat Kumar
2021185
rajat21185@iiitd.ac.in

28 April,2024

Abstract

In the project, we created a neural network using the Keras API in Python. The following report compares the results of the data tested with the neural network with the one tested with the linear regression model. Based on the results, we will conclude whether to use the linear regression or neural network model for the given problem.

1 Introduction

In the growing age of AI, the neural network plays a crucial role. We saw this project as an opportunity to delve deeper into the neural network model. Thus, we decided to use Keras API for the project. We decided to use this approach based on the following advantages of Keras:

1. Based on Python, a familiar coding language that gives a better understanding of the code.
2. Compatibility with TensorFlow, which is also important in implementing the neural network.
3. Keras provides industry-strength performance and scalability: it is used by organizations including NASA, YouTube, or Waymo.

This introduction sets the stage for exploring neural networks through the lens of the Keras API, aiming to gain insights and practical experience in this rapidly evolving field.

2 Related Works

For a better understanding about the Keras API, we have gone through the paper 'Keras and TensorFlow: A Hands-On Experience' published by Fredin Joe John Joseph, Sarayut Nonsiri and Annop Monsakul. The paper was extremely useful in understanding the API and building a basic neural network model. We have also explored some websites to get a better understanding. Links to the paper and websites are cited below.

3 Methodology

We have used Keras API to build a neural network. The model consists of one input layer, two hidden layers and one output layer. Both the hidden layers are activated with 'relu' activation function and the output layer is activated with linear activation function. We have passed the log distance numpy array in the Input layer of the model after necessary pre-processing, which is discussed in the next section. The input layer can take batch of variable size, thus providing flexibility. The input size is one because we only pass the log distance array. The first hidden layer will also have an input size of 1, passed by the input layer. The output size of the first hidden layer is 720, equal to the neurons in this layer. Thus, the input size in the second hidden layer is 720 and output is 480, equal to the number of neurons in this layer. Similarly, the output layer will have an input size of 480 and an output size of 1 because it gives only RSS values as output.

We have used the Adam optimizer, which adjusts the learning rates for each parameter individually based on the past gradients and squared gradients. This adaptiveness helps in training deep neural networks more effectively. We saved the model in history, which was later used to test the data. Additionally, we have used a fitted linear regression model to compare results with that of neural network output. The fitted linear regression model was created in one of the previous assignments.

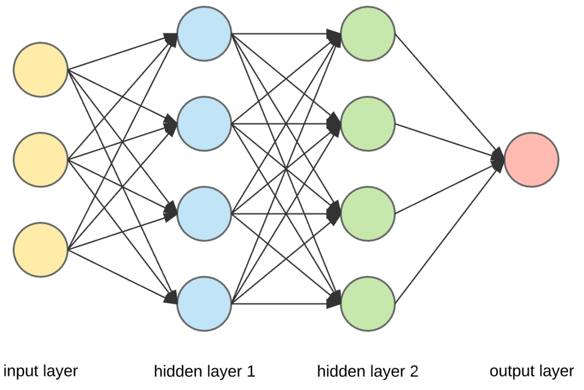
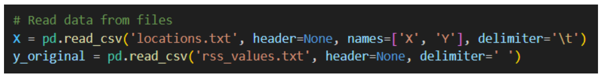


Figure 1: Neural network architecture

4 Evaluations and Results

4.1 Data Preprocessing

Here we have used pandas library to read data from .txt file as follows:



```python
# Read data from files
X = pd.read_csv('locations.txt', header=None, names=['X', 'Y'], delimiter='\t')
y_original = pd.read_csv('rss_values.txt', header=None, delimiter=' ')
```

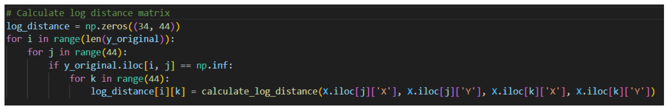
Figure 2: Taking Input from .txt files

For reading locations:

- *header=None*: Indicates that the data file does not contain a header row.
- *names=['X', 'Y']*: Specifies the column names as X and Y for the DataFrame. This is useful as the data file does not have any header row.
- *delimiter='\t'*: Specifies that the data is tab-delimited. Each line in the file is split into columns based on the tab character.

For reading rss values:

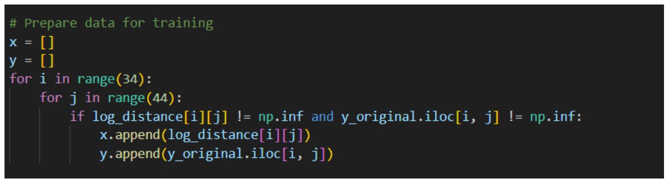
- *header=None*: Indicates that the data file does not contain a header row.
- *delimiter=' '*: Specifies that the data is space-delimited. Each line in the file is split into columns based on the space character.



```python
# Calculate log distance matrix
log_distance = np.zeros((34, 44))
for i in range(len(y_original)):
    for j in range(44):
        if y_original.iloc[i, j] == np.inf:
            for k in range(44):
                log_distance[i][k] = calculate_log_distance(X.iloc[j]['X'], X.iloc[j]['Y'], X.iloc[k]['X'], X.iloc[k]['Y'])
```

Figure 3: Calculating log distance

Here we are first checking which transmitter is transmitting then computing log distance between other receivers by providing x and y coordinates of transmitter and receiver.



```python
# Prepare data for training
x = []
y = []
for i in range(34):
    for j in range(44):
        if log_distance[i][j] != np.inf and y_original.iloc[i, j] != np.inf:
            x.append(log_distance[i][j])
            y.append(y_original.iloc[i, j])
```

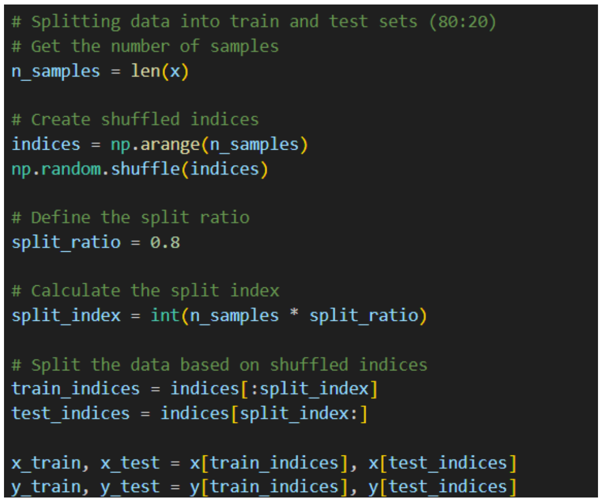
Figure 4: Preprocessing data for training

Here, we have taken the x list as the log distance matrix and y as the RSS matrix by iterating over the original log distance and RSS matrix with infinity elements in between and checking if there is an infinity element in the list. If there are We are not appending our log distance and RSS matrix.

Here We have first randomly shuffling our x, y list then taking 80:20 for training and testing.

4.2 Training and Testing

As mentioned in the previous section, we have used 80 per cent of the total data for training the model and the remaining 20 per cent for testing the model. While training, the model readjusts weights and biases of the hidden



```python
# Splitting data into train and test sets (80:20)
# Get the number of samples
n_samples = len(x)

# Create shuffled indices
indices = np.arange(n_samples)
np.random.shuffle(indices)

# Define the split ratio
split_ratio = 0.8

# Calculate the split index
split_index = int(n_samples * split_ratio)

# Split the data based on shuffled indices
train_indices = indices[:split_index]
test_indices = indices[split_index:]

x_train, x_test = x[train_indices], x[test_indices]
y_train, y_test = y[train_indices], y[test_indices]
```

Figure 5: Shuffling and Splitting input data

layer to reduce the MSE. After the training, we used the model's history to pass the testing data. We are then comparing our results with that of linear regression. We have also saved our model, which can be used to pass training data directly without training again.

5 Discussion

Earlier, we were implementing the neural network from scratch. We were able to implement forward propagation and computed mse. However, it was challenging to implement back-propagation. We were then allowed to use the API's built-in functions. It made the work easier, but it was still challenging to give up the first approach and adjust to the new one. A lot of time was required to adjust the parameters of the model, like epoch, learning rate, batch size and neurons, to get accurate results. Additional challenges were faced in saving the model and using it for testing in the new code file. Some issues were faced in adjusting plots later in the code but were easily dealt with. We used built-in linear regression for comparison but then decided to incorporate the one implemented in the earlier assignment. Adjusting this linear regression model in the current project was also challenging. We also faced some other issues that did not consume much time.

6 Results

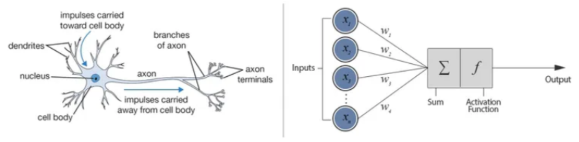


Figure 6: neural network

6.1 Training and Validation Loss

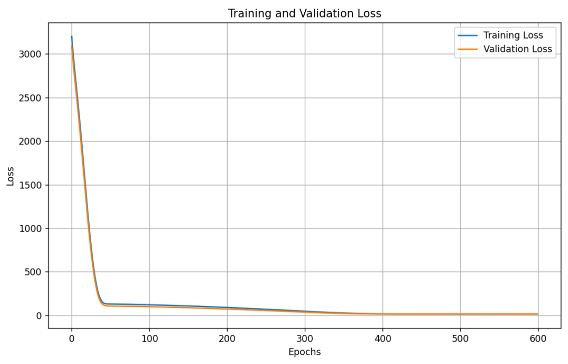


Figure 7: Training and Validation Loss

6.2 Training Error and Validation Error

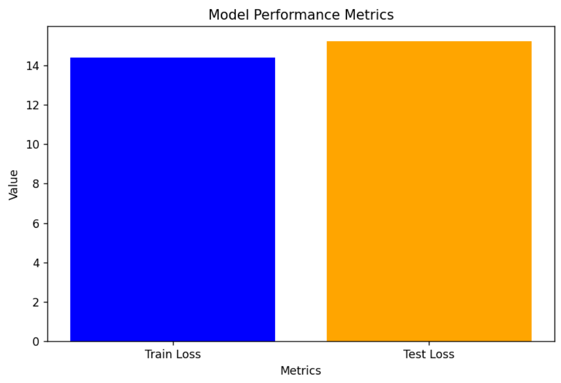


Figure 8: Training and Validation Error

6.3 Comparison with Linear Regression

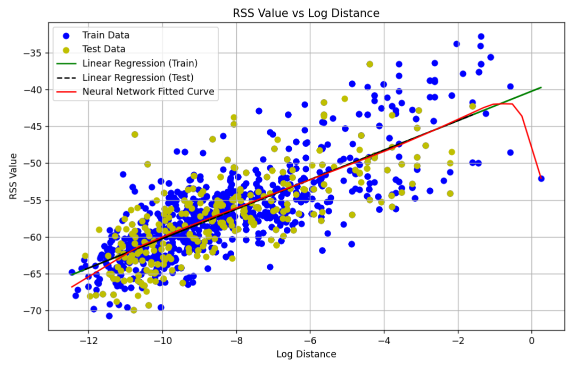


Figure 9: Comparision between linear regression and neural network

Neural Network Train Loss: 14.395055770874023
Neural Network Test Loss: 15.326796531677246
Linear Regression Train Loss: 14.688511956170288
Linear Regression Test Loss: 15.367668063014394

7 Conclusion

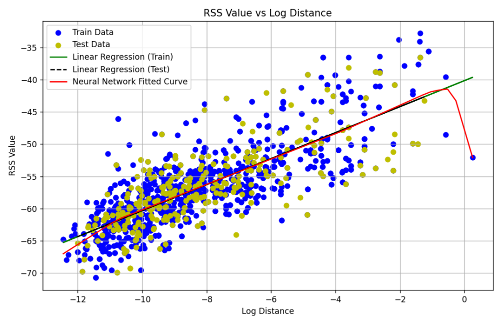


Figure 10: neural network fitting curve 1

Neural Network Train Loss: 14.33200454711914
Neural Network Test Loss: 15.64250659942627
Linear Regression Train Loss: 14.59385785416448
Linear Regression Test Loss: 15.620919834548708

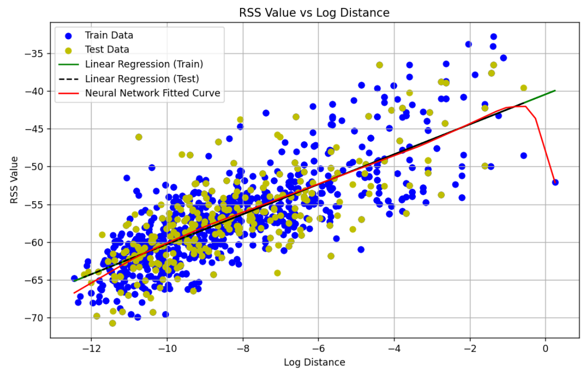


Figure 11: neural network fitting curve 2

Neural Network Train Loss: 14.468652725219727
Neural Network Test Loss: 14.851754188537598
Linear Regression Train Loss: 14.738257633709217
Linear Regression Test Loss: 15.00952616043783

We have trained and tested our code multiple times to conclude. We eventually concluded that it is preferred to use the neural network model for better accuracy. However, it was also noticed that for the given problem, MSE for linear regression and Neural network is close and linear regression is faster as it does not require training. So, it is a trade-off of speed and accuracy.

Citations

Hyperlinks for the citation is in ReadMe Here are the citations for the related works mentioned above:

- https://www.researchgate.net/publication/353438370$_{K}eras_{a}nd_{T}$
  $On_{E}xperienceKerasandTensorFlow : AHands - OnExperienc$
- https://towardsdatascience.com/building-our-first-neural-network-in-keras-bdc8abbc17f5Building Our