Neural Network and Deep Learning in Dataset Analysis

Bowen Tan

Research School of Computer Science, Australian National University
u6822656@anu.edu.au

Abstract. Neural network is a complex network system formed by a large number of simple neurons widely connected to each other [1]. Deep learning is to learn the internal laws and representation levels of sample data through the analysis of data sets. This project pre-process the provided dataset vehicle-x by deleting its missing patterns and normalize the value, constructed a full-connected and a bidirectional neural network model using simple neural network technique, and a recurrent neural network model based on deep leaning technique. After training these models, we got the conclusion that the losses decreased in each model and the test accuracy of the models is greater than 99%. It means that the column vehicleID is a proper classification target.

Keywords: Neural network, Deep learning, Iteration, Dataset

1 Introduction

This project selects the dataset vehicle-x, which contains 272 backbone models, with different colored textures, and creates 1,362 different vehicles [2]. There is an xml file in the dataset, involving all the information of the vehicles. It also has training, testing, and validating sets, containing various feature vectors, standing for the image of each vehicle. The reason of choosing this dataset is the feature vectors has all been processed and given, which means we can save time for working on dividing the raw data into training and testing set during the pre-processing work. Then I pre-process the dataset by deleting the missing patterns and normalizing the value of the dataset, constructing three neuron network classification model, based on deep learning technique to investigate whether divide the dataset into different classes according to the type of the vehicles is a good choice when doing classifying tasks. After training and validating, the result shows that vehicle ID is an appropriate target for classifying.

2 Methods

2.1 Data Pre-process

The dataset vehicle-x we got is in 600 mb, containing more than 75000 lines of data in total and 10 variables in the xml file of it. The variables are CamDis, CameraID, ImageName, …etc and all of them are in type of integer, except cameraID, ImageName, and vehicleID, which are in type of string. Because of this xml file provided most information of this dataset, we decided to transfer the given xml file named ‘finegrained_label.xml’ into a csv file. It is because that xml is hard to read by PyTorch and csv file is much easier to be read.

2.1.1 Data reading

Initially, we planned to read the xml file by using the attached package BeautifulSoup used alongside the lxml xml parser or Elementtree library [3]. However, we found that the column names of the dataset in the xml file was given in various items format instead of the xml roots, which is hard to be recognized by lxml. In order to solve this problem, we have to implement a much more tiring method. Firstly, we read the original xml line by line and write each line into a new txt file called ‘finegrained_label_raw’, and then get each column’s value according to the horizontal position of it, storing each set of values into a list and finally turn the bunch of lists together into a new csv-form dataset named ‘vehicle.csv’. As the size of the dataset is too large, the whole csv file generating code takes more than 5 minutes to finish running.

2.1.2 Deleting missing patterns

Considering that there may be some missing patterns in the original dataset, we read the dataset into a data frame form and then used the python build in function named apply to delete the missing value. The piece of code is as below.

```
df = pd.read_csv('vehicle.csv')
df.apply(lambda x: sum(x.isnull()),axis=0)
```

2.1.3 Data normalizing

Then, because we decided to set the last column “vehicleID” as the classification target, we normalized all of the values in this column into the range of 0 to 1 as it will help me do the follow up classification task much more convenient.

2.1.4 Training and testing set dividing

Next, we split the just-generated dataset 'vehicle.csv' into training and testing set, based on the feature vectors' names given in train and test file. We used the build-in function os.walk in the PyTorch package named OS to go through the features' names in each file and add the corresponding lines into the training and testing data frame in 'vehicle.csv'. Finally, we divided all the feature vectors into training and testing features. We created two lists called train_feature_matrix_dir and test_feature_matrix_dir, adding the given feature vectors into each list and turn them into two data frames for follow up data inputting process.

2.2 Neuron Network Model Building and Training

After data pre-processing, we used deep learning technique to construct a recurrent neuron network model, and a full connected and a bidirectional model based on simple neuron network technique for the classifying task. The process of model constructing is as figure 1, and the models are listed below.

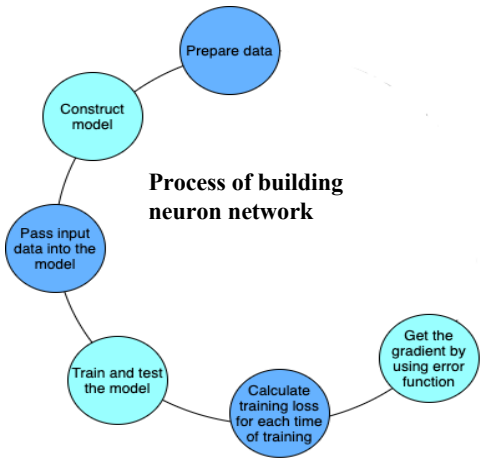

Figure 1 Process of constructing neuron network model

2.2.1 Full-connected Model

Full-connected neuron network connects every neuron in one layer to every neuron in another layer, which is a one direction transforming network that only convey data forward [4]. For the hyper parameters, as the classification is based on 1362 different vehicleID, we set the output neuron number as 1362. Additionally, we set the input size as 2048, representing the number of features, hidden size as 50, number of training as 50, and learning rate as 0.01. The sigmoid activation function we selected is Adam, and the code structure of this full-connected neuron network is as below,

```
class Net(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.sigmoid = nn.Sigmoid()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.sigmoid(out)
        out = self.fc2(out)
        return out
```

During each training process of the full-connected neuron network, we passed the input feature matrix into the neuron network model and then use the library function CrossEntropyLoss () to calculate loss between each feature vector and the training target. Lastly, we stored all losses into a list and drew a plot to visualize it. For easier comparing, we put each training result figure in figure 2 under the bullet point 3 result and discussion.

2.2.2 Bidirectional Model

Bidirectional neuron network model transfers input data in both directions of forward and backward and can be trained as either associative memories or cluster centroid finders, which provides the neuron network new abilities and enables us to design powerful data representation techniques which are a key factor in reducing network generalization error [5]. As it can convey the data in both directions, the number of hidden neurons size and the number of layers should be twice as before. In this case, for the setting of hyper parameters, we set the hidden size as 100, input size as 2048, output neuron number as 1362, number of training as 50, and learning rate as 0.01.

As it's a bidirectional neuron network model, the input data should be a 3-dimensional feature matrix. We reshaped the training feature matrix into [45438, 1, 2048] because the length of feature vector is 2048, and there are 45438 features in the training set in total. The other training process is similar to what we have done in the previous full-connected model's training.

```
class Net(nn.Module):
def __init__(self, input_size, hidden_size, num_layers,
num_classes):
  super(Net, self).__init__()
   self.input_size = input_size
   self.hidden_size = hidden_size
   self.num_layers = num_layers
  self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
batch_first=True, bidirectional=True)
  self.fc = nn.Linear(hidden_size * 2, num_classes)

def forward(self, x):
  x_label = torch.zeros(self.num_layers * 2, x.size(0),
self.hidden_size).to(device)
   y_label = torch.zeros(self.num_layers * 2, x.size(0),
self.hidden_size).to(device)
   out, _ = self.lstm(x, (x_label, y_label))
   out = self.fc(out[:, -1, :])
   return out
```

2.2.3 Recurrent Neuron Network model

Recurrent neural network is a kind of neural network that takes sequence data as input, recursively in the evolution direction of the sequence, and all nodes are connected in a chain. [6] It has memory, parameter sharing and Turing completeness, so it has advantages when learning the nonlinear characteristics of the sequence. Recurrent neural networks are also used in natural language processing for various time series forecasts.

Because the recurrent neural network has the property that the data is transferred between a node recursively, the feed forward process should be in the form of _, hidden = func((data, …, …), hidden), so that the hidden layer node could be recursively fed weight. For the hyper parameters, it's not much different from previous neuron networks, so does the training process.

```
class RNN(nn.Module):
def __init__(self, input_size, hidden_size, output_size):
          super().__init__()
          self.hidden_size = hidden_size
          self.lstm = nn.LSTM(input_size, hidden_size)
          self.linear = nn.Linear(hidden_size, output_size)
          self.hidden = (torch.zeros(1, 1, self.hidden_size),
torch.zeros(1, 1, self.hidden_size))

     def forward(self, seq):
          lstm_out, self.hidden = self.lstm(seq.view(len(seq),
1, -1), self.hidden)
          pred = self.linear(lstm_out.view(len(seq), -1))
          return pred[-1]
```

2.3 Model Testing

The method of testing the model accuracy is based on this accuracy calculating function,

$$(correct\_number/total\_number) * 100\%$$

We calculate the predictive value by using the build in function torch.max, and the predictive value is torch.max(pred_test, 1), where pred_test is the pre-loaded feature from testing dataset. The total number is the length of the predictive value, and the correct predicted number is the count of data that is same in predictive value and test value.

3 Results and Discussion

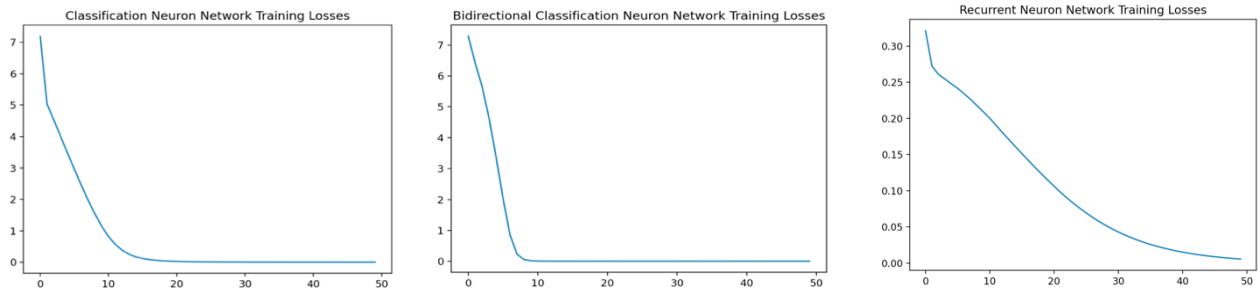

Figure 2 losses visualization

As the figure 2 shows, after training each neuron network model for 50 times, the losses of each model decrease obviously from the initial value to less than 0.008. It's a little bit different that in the deep learning based recurrent neuron network model, the loss decreased significantly just as soon as the training process start, while the losses start decrease obviously after training for 6 times in simple neuron network technique based model. Also, the loss in deep learning based model decreased smoother than others. The specific values are shown in table 1.

Table 1. Training losses of each mode

Names / Training Time	Full-connected Neuron Network	Bidirectional Neuron Network	Recurrent Neuron Network
0	7.1865	7.2802	6.5391
1	7.1865	7.2802	0.3210
6	2.9618	2.0260	0.2415
11	0.8201	0.0089	0.2000
16	0.1192	0.0052	0.1512
21	0.0278	0.0052	0.1063
26	0.0128	0.0051	0.0693
31	0.0089	0.0050	0.0430
36	0.0075	0.0050	0.0258
41	0.0069	0.0050	0.0152
46	0.0066	0.0050	0.0089

During the model testing process, we found that the number of correctly predicted is 15132, and the amount of data in total is 15142. According to the formula of accuracy, testing accuracy of the neuron network model is (15132/15142) * 100%, which shows that the accuracy is greater than 99%.

4 Conclusion and Future Work

In the experiment of this project, we got a vehicle dataset containing variable names such as camera ID, orientation, and vehicle ID. We pre-processed this dataset, and constructed two models by using simple neuron network technique and a recurrent model based on deep learning technique. After training these models, we found that it's a proper choice to regard the column vehicleID as the classification target.

Although the losses have decreased and the test accuracy is high enough, the training time of the neuron network models takes a long time, especially the deep learning technique, so we should try to decrease the complexity of my code and get the training process done in less time. Also, when normalize the target data, directly normalize it to 0~1 may not be the best choice, as it may misrepresent the original meaning of vehicle ID. Maybe normalize it by using hot-encode technique

is a better way. Additionally, we tried to read the given xml file 'finegrained_label.xml' and turn it into a csv file by using the build in package of PyTorch called lxml, but failed, what we have done involves various redundant and tiring process. We'll try to read xml file with the build-in packages in the future.

After completing this project, I've got much more advanced comprehension of machine learning and deep learning. Deep learning is a new field of machine learning. Its mobility lies in the establishment of a neural network that simulates the human brain for analysis and learning. It mimics the mechanism of the human brain to interpret data, such as images, sounds, and texts. I hold the view that deep learning is a main developing direction of machine learning and artificial intelligence, which is one of my research choices in the future.

References

[1] Larry Hardesty, Explained: Neural networks, 2017.

[2] A.F. Nejad, T.D. Gedeon, BiDirectional Neural Networks and Class Prototypes, 2002.

[3] Reading and Writing XML Files in Python, 2020

[4] https://en.wikipedia.org/wiki/Convolutional_neural_network

[5] Yue Yao, Liang Zheng, Xiaodong Yang, Milind Naphade, Tom Gedeon, Simulating Content Consistent Vehicle Datasets with Attribute Descent, 2020.

[6] Shen Mei-Li; Lee Cheng-Feng; Liu Hsiou-Hsiang; Chang Po-Yin; Yang Cheng-Hong, Effective multinational trade forecasting using LSTM recurrent neural network, 2021