Final Project Report

Student Name: BANG Jun Young **Student ID:** 15101119D

Design of task 1

The shape of the Task 1 data is (180, 20). Simply saying, there are 20 features and 180 data entries. Correspondingly, the label data includes 180 target variables. Before applying any kind of feature engineering or machine learning method, I tried training the crude logistic regression, Decision Tree, Random Forest, and Gradient Boosting models using the given raw data.

- The mean score of 5-fold Cross validation is 0.661
- Following on, the mean cv score of the Decision Tree, Random Forest, and Gradient Boosting Classifiers are 0.606, 0.733, 0.756, respectively.

These evaluation results are showing us quite a low credibility for tested classifiers except for the Gradient Boosting classifier, but we'll improve the mean score with various techniques and models. The important aspect of these two datasets is that all the dependent variables are continuous numerical variables. Moreover, there are no missing values.

The initial effort of task 1 was focused on implementing a traditional algorithm using a voting classifier, but after task 2, I changed the direction of implementing the 1-dimensional convolutional neural network. 1D CNN is like 2D CNN learned in the course, but the difference is that it is a kernel filter that only moves in one dimension. Since the given data is 1-dimensional data, we can use the 1D CNN kernel in the size of (1xn). To avoid overfitting problem due to the small dataset size, I implemented the dropout and batch normalization to the network. Furthermore, I tried several different network configurations.

Configuration and Result

```python
class Conv1d_net(nn.Module):
    def __init__(self):
        super(Conv1d_net, self).__init__() # 1*20
        self.conv1 = nn.Conv1d(1, 5, 2, padding=1) # 6@1*20
        self.pool = nn.MaxPool1d(2) # 6@10
        self.fc1 = nn.Linear(50, 24)
        self.fc2 = nn.Linear(24, 8)
        self.fc3 = nn.Linear(8, 3)

        self.drop_3 = nn.Dropout(0.3)
        self.drop_4 = nn.Dropout(0.4)
        self.conv1_bn = nn.BatchNorm1d(5)

    def forward(self, x):
        x = self.pool(F.elu(self.conv1_bn(self.conv1(x))))
        x = x.view(-1, 50)
        x = self.drop_4(F.elu(self.fc1(x)))
        x = self.drop_3(F.elu(self.fc2(x)))
        x = self.fc3(x)
        return x
```

Figure 1. Class of 1d CNN model in PyTorch

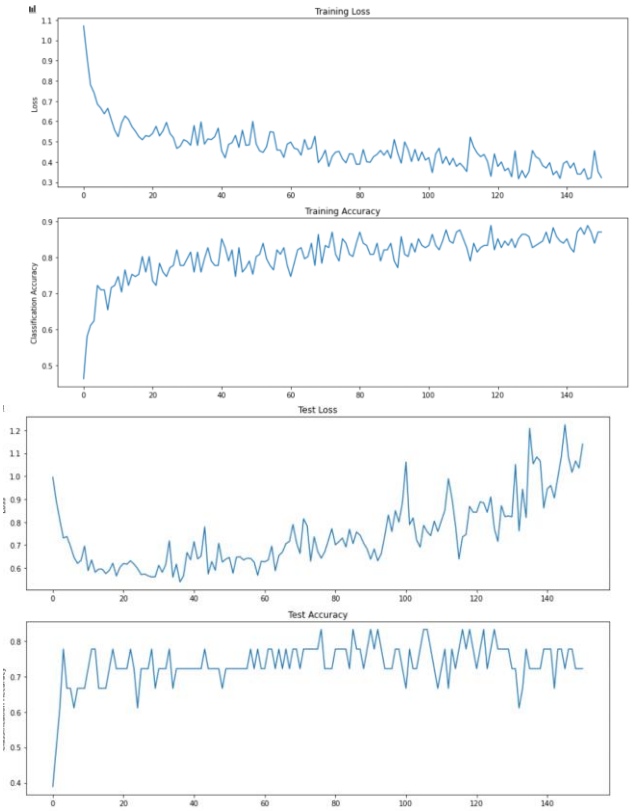

Figure 2. Train and Validation Result of the task 1 model

Design of task 2

Task 2 dataset is 3-dimensional data. The shape of the dataset is (180, 10, 10). What this means is that we are going to predict 180 target variables with each (1, 10, 10) data entry. I built a CNN model with PyTorch library.

Since the size of the given dataset is small compared to the majorities of deep learning application, the overfitting must be concerned. Overfitting refers to the phenomenon where the training model is too biased toward the given dataset. When the model is suffering from high variance and low bias, it means overfitting. The size of the training dataset indeed can hugely affect deep learning performance and overfitting problem, because the dataset may not sufficiently represent the problem the network wants to solve. There are several ways to regulate the neural network to avoid overfitting during the learning process so that the network can be generalized and handle unseen data fairly. The most typical and easily applicable method is Dropout and Batch Normalization.

- Dropout regularization is to randomly turn off (or ignore) the output of the selected number of nodes from a layer. This gives the effect of training a different number of nodes and connectivity to the prior layer during deep learning. As a result, a network with a large size can be trained with less tendency of overfitting, because the view of the network layers on training mode are different from, or smaller than, the configurated network setting.
- Batch Normalization is one way to increase the efficiency of the training. By applying it, we can normalize the output value from the network node or activation function. This normalized output can be also viewed as noise because each batch will have a different standard deviation and mean value. The benefit of Batch Normalization is that it reduces the risk of overfitting and Gradient vanishing problem. Also, the learning rate can be set higher so that the learning process can be faster than with a low learning rate value.
- Furthermore, according to the suggestion from several articles, I reduced the size of the network compared to the normal practices of CNN.

Configuration and Result

Here is the configuration of the model:

```python
class Conv2d_net(nn.Module):
    def __init__(self):
        super(Conv2d_net, self).__init__() # 10*10
        self.conv1 = nn.Conv2d(1, 6, 5, padding=2)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 10, 2, padding=1)
        self.fc1 = nn.Linear(10 * 6 * 6, 84)
        self.fc2 = nn.Linear(84, 36)
        self.fc3 = nn.Linear(36, 3)

        self.drop_3 = nn.Dropout(0.3)
        self.drop_5 = nn.Dropout(0.5)
        self.conv1_bn = nn.BatchNorm2d(6)
        self.conv2_bn = nn.BatchNorm2d(10)

    def forward(self, x):
        x = self.pool(F.elu(self.conv1_bn(self.conv1(x))))
        x = F.elu(self.conv2_bn(self.conv2(x)))
        x = x.view(-1, 10 * 6 * 6)
        x = F.elu((self.fc1(x)))
        x = F.elu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Figure 3. Class of 2d CNN model in PyTorch

Here is the result of training and validation of the CNN for task2.

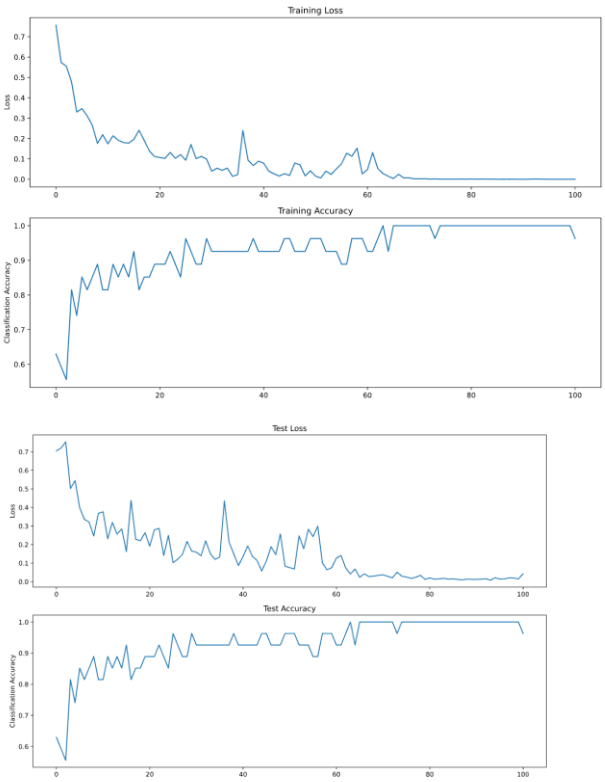

Figure 4. Train and Validation Result of the task 2 model

How to run the source code

1. Put the "data_test_task1.npy" and "data_test_task2.npy" file under the deliverable folder

2. Check if there are two PyTorch model file named ‘state_dict_task1_model.pt’ and ‘state_dict_task2_model.pt’

3. Run “python test_task1.py” under the deliverable folder

4. Run “python test_task2.py” under the deliverable folder

	1	2
libraries	torch numpy	torch numpy

Table 1. Library needed to be installed in prior of execution.

Submitted documents and files

- test_task1.py
  test_task2.py
  JunyoungBang_final_report.pdf
  task1_DL.ipynb
  task2_DL.ipynb

Reference

- Feature Engineering for Numerical Data
- Data Exploration and Analysis Using Python
- How to Prevent Overfitting
- Impact of Dataset Size on Deep Learning Model Skill And Performance Estimates
- Machine Learning: Bias VS. Variance | by Alex Guanga
- Don’t Use Dropout in Convolutional Networks
- Batch Normalization and Dropout in Neural Networks with Pytorch
- Batch Normalization of Linear Layers
- Neural Network Programming - Deep Learning with PyTorch