Final Assignment Report

2016112083 김연웅

About My Model Design

```
class modelFinal(nn.Module):
    def __init__(self, input_size=35, hidden_size=10, num_layers=1, output_size=1):
        super(modelFinal, self).__init__()
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)
        self.fc1 =  nn.Linear(hidden_size, 32,bias=True)
        self.fc2 = nn.Linear(7, 32,bias=True)


        self.fc3 = nn.Linear(32 + 32, 32,bias=True)
        self.fc4 = nn.Linear(32,1,bias=True)



        self.Sigmoid = torch.nn.Sigmoid()
        self.relu = nn.ReLU()


        self.dropout = torch.nn.Dropout(p=0.2)
    def forward(self,x_time, x_static):
        h_0 = Variable(torch.zeros(self.num_layers, x_time.size(0), self.hidden_size)) #hidden state
        c_0 = Variable(torch.zeros(self.num_layers, x_time.size(0), self.hidden_size)) #internal state

        output, (hn, cn) = self.lstm(x_time, (h_0, c_0))
        hn = hn.view(-1, self.hidden_size)
        out_time = self.relu(hn)
        out_time = self.fc1(out_time)
        out_time = self.relu(out_time)

        out_static = self.fc2(x_static)
        out_static = self.relu(out_static)

        out = torch.cat((out_time, out_static), dim=1)

        out = (self.fc3(out))
        out = (self.relu(out))
        out = (self.fc4(out))

        y_pred = self.Sigmoid(out)
        return y_pred
```

Model Layers

time dataset - LSTM

- Input size: 35
- hidden size: 10
- num_layers: 1
- output size: 32

static dataset - FCN

- input layer : 7
- output layer : 32

Activation function : ReLU, Sigmoid(for binary classification)

converged - FCN

1st - input : 32 + 32 = 64, output: 32

2nd (output) - input: 32 , output: 1

I used fully-connected network for time-series features and LSTM network for static dataset. Then I converged into a fully network layer again, for classification task.

Number of Parameters

```
params = list(model.parameters())
print("The number of parameters:", sum([p.numel() for p in model.parameters() if p.requires_grad]), "elements")
```

```
The number of parameters: 4688 elements
```

weighted F1(up to three decimal points)

In [38]:
```
# Load best performance model
checkpoint = torch.load('./bestModel.pt')
trained_model = modelFinal()


trained_model.load_state_dict(checkpoint['state_dict'])

# evaluate
trained_model.eval()
with torch.no_grad():

    # load test dataset

    time_test_x = torch.load('time_test_X.pt',map_location=torch.device('cpu'))
    static_test_x = torch.load('static_test_X.pt',map_location=torch.device('cpu'))
    test_y = torch.load('test_y.pt', map_location=torch.device('cpu'))
    # evaluate using weighted f1 score.
    y_pred = trained_model(time_test_x, static_test_x)
    yp = y_pred.detach().numpy()
    yp = [1.0 if x > 0.5 else 0.0 for x in yp]

    print("weighted F1:", f1_score(test_y, yp, average='weighted'))
```

```
weighted F1: 0.9421980250140295
```

WEIGHTED F1: 0.942

Random Seeds Configuration

```
# Configure Random seeds
torch.manual_seed(777)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(777)
```