4-class Classification Problem of ECG Signals by Four Network Models (final-project)

Name: WangYexiang
SID: 12012529

***Abstract*—In this project, a fully connected feedforward deep network, a CNN (could be any modern CNN network), a RNN (could be any RNN such as Pyramid RNN, LSTM, GRU, Grid LSTM), and an attention network to solve the above 4-class classification problem of ECG signals. By this project, the 4 nets are trained and students will get more familiar with them, having a deeper understanding. How to chose a better model with effective hyper parameters is very important.**

***Keywords*—Deep Learning Models, classification, fully connected deep network, CNN, RNN, attention network.**

I. Introduction

To deal with 4-class classification problem of ECG signals by a fully connected feedforward deep network, a CNN (could be any modern CNN network), a RNN (could be any RNN such as Pyramid RNN, LSTM, GRU, Grid LSTM), and an attention network, there a dataset for training models, "egc_data.csv" comprises a total of 8528 recordings with 188 features (the first 188 columns of ecg_data.csv) extracted from single ECG signals. There are supposed to be 4 categories labeled 1 through 4, corresponding to "Normal", "Atrial Fibrillation (AF)", "Non-AF related abnormal heart rhythms", and noisy recording". The distribution of normal, AF, other rhythms and noisy data is largely imbalanced in the dataset.

Consider the following performance metrics: F1-score for normal(Fnorm), AF(Faf), other rhythms(Foth) and the final accuracy Ft = (Fnorm + Faf + Foth)/3. Compare and analyze the results obtained from the four approaches. It is suggested a 5-fold cross validation is considered to observe the performance.

In this project, the **pytorch** is used to help training the prediction models and sklearn is uesd to calculate relative parameters.

II. Problem Formualtion

A. MLP

MLP, "Multilayer Perceptron", is a a fully connected feedforward deep network. In addition to the input and output layers, it can have multiple hidden layers in between. The most important part of MLP is active funciton, which makes the model different from linear model. The active function is nonlinear function, giving the model nonlinearity. Solve **nonlinear** separable problems, use multiple layers of functional neurons. Neurons are not connected to the same layer, nor are they connected across layers. Figure 1 is a simple MLP.

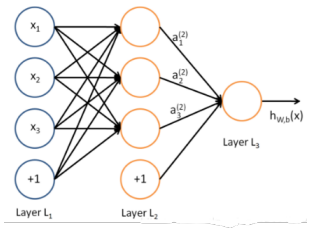

Fig. 1. A simple MLP

B. CNN

CNN, "Convolutional Neural Network", is a mathematical or computational model that mimics the structure and function of biological neural networks (the central nervous system of animals, especially the brain). A neural network consists of a large number of artificial neurons, which are built differently in different ways. CNN neural networks can have simple decision-making ability and simple judgment ability like people, and can give better results in image and speech recognition. In this project, the **LeNet** is selected, which will be described specifically in next section.

C. RNN(LSTM)

RNN, "Recurrent Neural Networks", has achieved great success and widespread application in many natural language processing (NLP). RNNs are called circulatory neural networks, that is, the current output of a sequence is also related to the previous output. The specific manifestation is that the network will remember the previous information and apply it to the calculation of the current output. In this project, the **LSTM** is selected, which will be described specifically in next section.

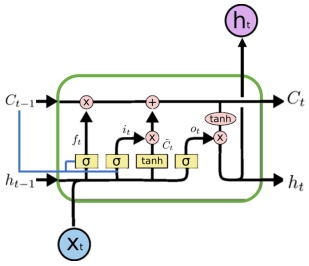

Fig. 2. LSTM

D. Attention Network

The mechanism of attention, if understood at a superficial level, matches his name perfectly. His core logic is ”from focusing on everything to focusing on the point”.There are three main reasons why the Attention mechanism is introduced: few parameters, fast speed, and good effect.

III. Method and Algorithms

In this project, MLP, LeNet, LSTM, attention are used to figure out the 4-class classification problem. Relative structres of models and algorithms will be introduced briefly below.

A. MLP

As Figure 3 shows, the MLP in this project have 4 hidden layers and the numbers of neutrons of each layer are 128, 64, 32 and 10. The number of outputs is 4, because this is a 4-class classfication problem. Every layer's activate function is ReLU. By testing many times, the effect of ReLU is better than Sigmoid.

```
def __init__(self):
    super().__init__()
    self.h1 = nn.Sequential(nn.Linear(188, 128), nn.ReLU())
    self.h2 = nn.Sequential(nn.Linear(128, 64), nn.ReLU())
    self.h3 = nn.Sequential(nn.Linear(64, 30), nn.ReLU())
    self.h4 = nn.Sequential(nn.Linear(30, 10), nn.ReLU())
    self.o = nn.Sequential(nn.Linear(10, 4))
```

Fig. 3. The structure of MLP in this project

B. LeNet (CNN)

CNNs are designed to simulate how the human eye perceives. Traditional CNNs generally contain the following three layers: Convolution layers, Subsampling layers, or pooling layers and Fully connected layers.

```
def __init__(self):
    super().__init__()
    self.net = nn.Sequential(
        nn.Conv1d(1, 2, kernel_size=5, padding=2), nn.ReLU(),
        nn.AvgPool1d(kernel_size=2, stride=2),
        nn.Conv1d(2, 4, kernel_size=5), nn.ReLU(),
        nn.AvgPool1d(kernel_size=2, stride=2),
        nn.Flatten(),
        nn.Linear(180, 84), nn.ReLU(),
        nn.Linear(84, 32), nn.ReLU(),
        nn.Linear(32, 4)
    )
```

Fig. 4. The structure of LeNet in this project

LeNet is alse obeyed this structure. In this project, as Figure 4 shows, it is consisted of a 1d Convolution layers, a 1d AvgPool, a 1d Convolution layers, a 1d AvgPool, and 3 fully connneted layers. Because of the input is 1d, the Convolution layer is also 1d.

C. LSTM (RNN)

LSTM, Long Short Term Memory, is a special kind of recurrent neural network. Unlike general feedforward neural networks, LSTMs can analyze the inputs using time series; In short, when using a feedforward neural network, the neural

```
class LSTM(nn.Module):
    def __init__(self, input_size=1, hidden_size=5, num_layers=2):
        super(LSTM, self).__init__()
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, bidirectional=True, batch_first=True)
        self.linear = nn.Linear(hidden_size * 2, 4)
        self.hidden_size = hidden_size

    def forward(self, x):
        hidden = (torch.zeros(self.num_layers * 2, x.shape[0], self.hidden_size).cuda(),
                  torch.zeros(self.num_layers * 2, x.shape[0], self.hidden_size).cuda())
        out,_ = self.lstm(x, hidden)
        # print(out.shape)
        out = self.linear(out[:, -1, :])
        return out
```

Fig. 5. The structure of LSTM in this project

network thinks that what we input at time t is completely irrelevant to what is input at moment t+1. In this project, LSTM is completed by help of pytorch, which is shown as Figure 5.

D. Attention Network

In this project, the attention is along with RNN to deal with the 1d input with 188 features. As Figure 6 indicates, the whole network was constructed by a RNN layer, a fully connected layer, a RNN layer, a Attention layer and finally a fully connected layer. It takes a problem that the speed of training whole model is very slow.

```
class RNNWithAttention(nn.Module):

    def __init__(self, i_size=1, h_size=5):
        super(RNNWithAttention, self).__init__()
        self.h1 = nn.Sequential(RNN(input_size=i_size, hidden_size=h_size),
                                nn.BatchNorm1d(h_size - 1))
        self.linear = nn.Linear(h_size - 1, h_size)
        self.rnn2 = RNN(input_size=h_size, hidden_size=h_size)
        self.attention = Attention(h_size)
        self.linear2 = nn.Linear(h_size, 4)

    def forward(self, x):
        # x = self.rnn1(x)
        # x = self.batchnorm(x)
        o1 = self.h1(x)
        o2 = F.relu(self.linear(o1))
        o3, _ = self.attention(o2)
        o4 = o3.unsqueeze(1)
        o5 = self.rnn2(o4)
        return o5
```

Fig. 6. The structure of Attention Network in this project

E. Training Procss

No matter what network model, the whole process of training is similar. The only thing need to change is the format of input. Now the MLP example is piked, as Figure 7 shows below.

```
optimizer = torch.optim.Adam(net.parameters(),
                             lr=learning_rate,
                             weight_decay=weight_decay)
for epoch in range(num_epochs):
    for X, y in train_iter:
        optimizer.zero_grad()  # 将梯度清零
        y_hat = net(X)  # 将数据输入网络
        # 处理y, 使之可以使用交叉熵计算loss
        y = y-torch.ones(y.size())
        y = y.squeeze()
        # print(y_hat.size(), y.size())
        ls = loss(y_hat, y.to(torch.long))  # 计算loss值

        ls.backward()  # 误差反向传播
        optimizer.step()  # 参数更新
```

Fig. 7. Train Process

Firstly, the data iterator is made to take input data into train function. Then, the optimizer, which is very important,

is ready. when coming into the training loop, after make historical gradient zero, calculate the loss of now model and using the optimizer to improve whole model and push the training process.

F. K-Fold

The usual practice is to divide a part of the training data as validation data to evaluate the training effect of the model. The validation data is taken from the training data, but does not participate in the training, which can relatively objectively evaluate the degree of matching of the model to data outside the training set. The evaluation of models in validation data is commonly used in cross-validation, also known as circular validation.

It divides the original data into K(in this project, K=5) groups (K-Fold), makes a validation set for each subset of data, and uses the remaining K-1 subset data as the training set, so that K models will be obtained. These K models are evaluated in the validation set, and the final error is added and averaged to obtain the cross-validation error. Cross-validation makes effective use of limited data, and the evaluation results can be as close as possible to the performance of the model on the test set, which can be used as an indicator for model optimization.

IV. Experiment Results and Analysis

A. MLP

In once training process, the loss figure is shown as Figure 8 as below. The training epoch is 200, and it can be found that the loss is tend to converge though it looks like unstable.

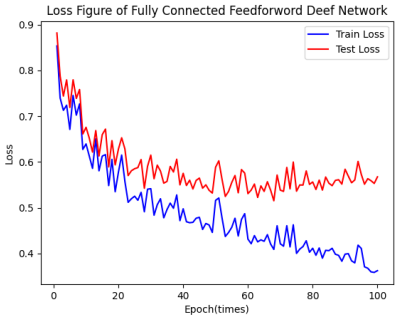

Fig. 8. Loss Figure of Fully Connected Feedforward Deep Network

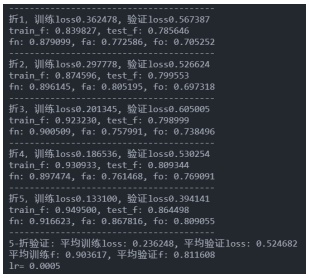

Fig. 9. Results of MLP

In other hand, the results of MLP is shown in Figure 9. By the tests, when the hyper parameters, learning rate, is 0.0005, the Ft(final accuracy, and the average value of 5 fold training) is up to 0.81, which is very good.

B. LeNet (CNN)

In once training process, the loss figure is shown as Figure 10 as below. The training epoch is 100, and it is obviously can be thought that the loss is tend to converge.

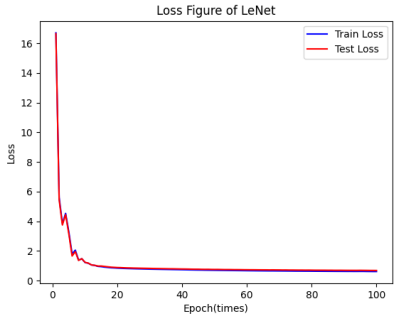

Fig. 10. Loss Figure of LeNet

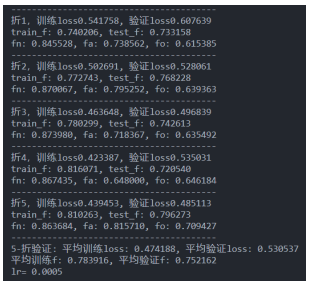

Fig. 11. Results of LeNet

In other hand, the results of MLP is shown in Figure 11. By the tests, when the hyper parameters, learning rate, is 0.0005, the Ft(final accuracy, and the average value of 5 fold training) is up to 0.75, which is not bad.

C. LSTM (RNN)

In once training process, the loss figure is shown as Figure 12 as below. The training epoch is 200, and it is can be considered that the loss is tend to converge. However, the test loss and train loss is all too small, but vary a lot.

In other hand, the results of MLP is shown in Figure 13. By the tests, when the hyper parameters, learning rate, is 0.005, the Ft(final accuracy, and the average value of 5 fold training) is up to almost best, but it is still very low accuracy. The final average value is only 0.4448.

The loss is very small, so I guess the data is not relying on time too much, or the data samples have many useless cases. In other hand, by searching on the Internet, I found the original dataset have 18000 features! Therefore our datasets is a little part of the original datasets, causing many information lost. Or it is just too easy to overfit.

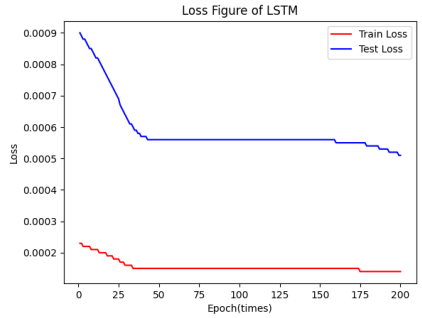

Fig. 12. Loss Figure of LSTM

```
f1score for normal (Fn):  0.7899
f1score for AF (Faf):  0.0
f1score for oth (Foth):  0.4447
final accuracy (Ft):  0.4115
f1score for normal (Fn):  0.8278
f1score for AF (Faf):  0.0
f1score for oth (Foth):  0.4917
final accuracy (Ft):  0.4398
f1score for normal (Fn):  0.8329
f1score for AF (Faf):  0.3846
f1score for oth (Foth):  0.5328
final accuracy (Ft):  0.5834
f1score for normal (Fn):  0.771
f1score for AF (Faf):  0.0
f1score for oth (Foth):  0.3262
final accuracy (Ft):  0.3658
f1score for normal (Fn):  0.795
f1score for AF (Faf):  0.0
f1score for oth (Foth):  0.475
final accuracy (Ft):  0.4234
```

Fig. 13. Results of LSTM

D. Attention Network

In once training process, the loss figure is shown as Figure 14 as below. The training epoch is 200, and it is can be considered that the loss is tend to converge. However, the test loss and train loss is all too small, but vary a lot.

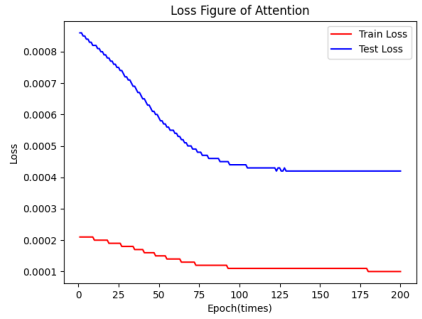

Fig. 14. Loss Figure of Attention Network

In other hand, the results of MLP is shown in Figure 15. By the tests, when the hyper parameters, learning rate, is 0.005, the Ft(final accuracy, and the average value of 5 fold training) is up to almost best, but it is still very low accuracy. The final average value is only 0.5071.

The reason of the unfavorable result maybe is as same as the LSTM. Loss of information and little relation about time cause the not good score. Maybe the loss of infomation take away many characteristics or features, making attention not

```
f1score for normal (Fn):  0.84
f1score for AF (Faf):  0.1918
f1score for oth (Foth):  0.5421
final accuracy (Ft):  0.5246
f1score for normal (Fn):  0.8691
f1score for AF (Faf):  0.0
f1score for oth (Foth):  0.5292
final accuracy (Ft):  0.4661
f1score for normal (Fn):  0.8676
f1score for AF (Faf):  0.3119
f1score for oth (Foth):  0.602
final accuracy (Ft):  0.5939
f1score for normal (Fn):  0.8352
f1score for AF (Faf):  0.0
f1score for oth (Foth):  0.6209
final accuracy (Ft):  0.4854
f1score for normal (Fn):  0.8563
f1score for AF (Faf):  0.0
f1score for oth (Foth):  0.5401
final accuracy (Ft):  0.4655
```

Fig. 15. Results of Attention Network

work. Maybe it is still the old problem, that the model is to easy to overfit.

V. Conclusion and Future Problems

For different problems and datasets, the selection of models is significant. In this project, the ecg data performs better on MLP and CNN, but not good on LSTM and Attention Network. The reason is from multiple aspects, and we need to analyse carefully. For me, the other thing matters is researching on internet. When the original datasets is found, one possibility of loss of information appear natural. Because our dataset is a little part of the origin(for our computer running the algorithms), some characteristics and features disappearing is normal. However it not only reminds me of the important of dataset and preoperation, but also the importance of investigation.

In future, the part of signal or how to deal with these situation is a direction to think.

References

[1] Python—CNN https://zhuanlan.zhihu.com/p/143092725
[2] http://t.csdn.cn/WuV4W
[3] http://t.csdn.cn/dTS0C
[4] https://www.knowledgedict.com/tutorial/torch-lr.html
[5] https://zhuanlan.zhihu.com/p/462192060
[6] https://developer.aliyun.com/article/692322: :text=Python
[7] http://t.csdn.cn/ANXzE

.