Report for Final Project

*An attempt for four kinds of deep neural networks usage on ECG signal analysis

YiFeng Zhang
12012907
From Department of Electronic and Electricity Engineering

***Abstract*—ECG signal can be often seen in our real life. With the aid of ECG figures, doctors can easily divide patients into parts. In this project, we are trying to find out a proper neural network which can judge their situations best. In four kinds of networks used, all three complex structures all perform well, which is greatly improved compared with MLP.**

***Index Terms*—MLP, CNN, RNN, Attention, CrossEntropy**

I. Introduction

Heart disease is often seen in our daily life, which always results in severe consequences. Among all these problems, Atrial Fibrillation (AF) is the most common sustained cardiac arrhythmia, which usually occurs in 1-2% among the general population, and is associated with significant mortality and morbidity association of the risk of stroke, heart failure and even death. Regular detection method to this kind of disease by doctors, need the aid of ECG (Electrocardiogram) signals.

So definitely how to decide from the signal waves that whether a signal implies the patient has AF is quite importantSince it is a kind of classification problem, we can try to use a sort of deep learning models to find out main features in these wave data which can help to classify them.

II. Data analysis and Loss Choice

A. Data Research

1) Figure Confirm: Considering that we are not familiar with this kind of data, the first job I did is try to find out some ways to judge the data quality in case that the data is bad. Since the data is a kind of ECG signals, intuitively I though that this dataset is made up of sampling points from the real ECG signals, while the sampling object is voltage. So the most convenient way to judge its quality is to draw it out. The plot of one piece of data is shown in Fig. 1 and Fig. 2.

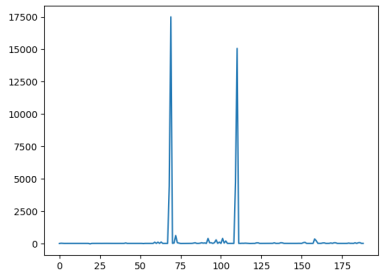

Fig. 1. Example of ECG signal

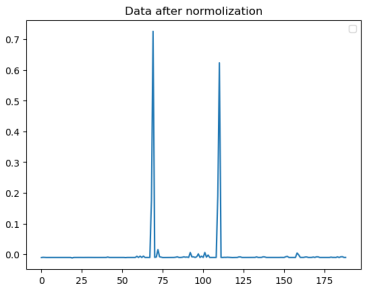

Fig. 2. Example of ECG signal (normalized)

So we can judge by our common sense that this is a quite normal ECG signal and we don't need to worry about that signals are occupied with too large noises since peaks in this plot can be found with no effort. By the way, the normalized data might also implies that normalization is useless since shape of two plots are the same from the naked eye.

2) K-fold cross validation: Another important trick to deal with this dataset is the **K-fold cross validation**. Since we do not have a large dataset, by using the cross validation we can avoid the model from overfitting if we can know the loss of both train and valid set at the same time. Using k-fold cross-validation allows us to get an estimate of the model's performance that is more reliable than when merely trained and tested the model on a single split of the data is used.

When doing K-fold, we just need to decide the begin and end for each valid set, then combine other parts to generate the train set. Inputs and outputs are shown in Fig. 3.

```
def get_k_fold(k, n, x_total, y_total):
    '''
        K-fold 分割函数，用于进行交叉验证训练
        传入参数:
            k: 总分割组数
            n: 本次所需的测试组组别
            x_total: 整体features,类型为dataframe,下同
            y_total: 整体labels

        传出参数:
            x_train, y_train: 对应训练集的输入输出
            x_valid, y_valid: 对应测试集的输入输出

        注意：此函数仅适合处理y为一维的情况
    '''
```

Fig. 3. Arguments for K-fold.

3) Futher research: However, I was still worried about that if this peaks themselves were created by some sorts of machine stoppage and etc., all the reasons I didn't know. So

I tried to find out some more professional related papers to ECG signal analysis. Luckily, the source of the dataset was found by me [1]. In this paper, the author showed how this dataset was generated and what they had done to optimize the dataset. It is said in this paper that all possible mechanical error are removed, and those signals with wrong labels have been corrected by themselves (in dataset version 2).

Other more advanced methods of data pre-process (such as R-peaks pick using the **Hamilton–Tompkins algorithm** and other time domain and frequency domain analysis [2]) are too difficult to be understood by me. So finally I decided no extra process needed to the dataset.

B. Loss function choice

A machine learning model can not be built up without a proper loss function, and so do a deep learning model. In our lecture we have learned lots of loss function for various problem. Among them, the top two most commonly used loss functions are **MSE** loss (1) and **CrossEntropy** loss (2).

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1}$$

$$L_{CE} = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_i^k log(\hat{y}_i{}^k) \tag{2}$$

In these two kinds of loss function, the MSE loss usually works when we need to build up a regression model, while CrossEntropy loss is especially designed for classifcation problems. Based on mathematical statistic and information theory, CrossEntropy performs quite well when we need to do estimation on the probabilities of the output with each input, which is pretty more appropriate with this project. So with all the models I built in this project, the loss functions I chose were all CrossEntropy loss.

III. MODEL DESIGN

This project required us to built four kinds of deep learning models, the MLP, CNN, RNN and attetion model. Here are my results.

A. Fully Connected model

Fully connected network, which is actually MLP in this project, is the most simple deep network to be used in classification problems.

MLP usually contains two parts. The first part only has some linear layer along with an activate function. In all the model in this project, I took the **ReLU** function as my activate function since its derivative is quite simple so my computer can work it out fast. At the same time, compared to **Sigmoid** or **tanh** function, this function can avoid gradient explosion and gradient disappearance to some extent. Formula of Relu is shown in (3).

$$f(x) = max(0, x) \tag{3}$$

The MLP for this project contains one input layer, one hidden layer and one output layer. Both input and hidden layer are followed by an Relu function. The size for each layer is 188*128, 128*64 and 64*4. Setting the number of training steps to 2000, using CrossEntropy Loss and the **SGD** as the optimizer and 0.00005 as the learning rate, the loss function graph is shown in Fig. 6

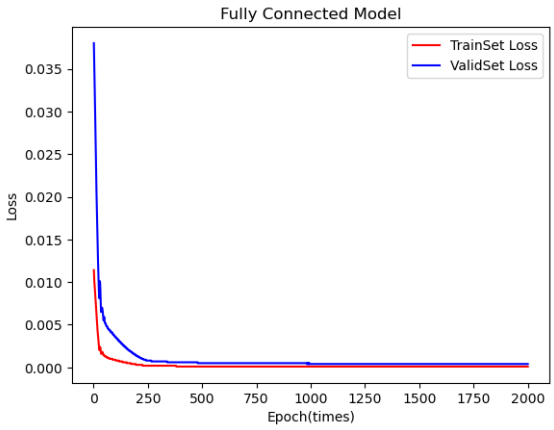

Fig. 4. Loss graph for MLP.

This figure clearly shows that the model fits both trainset and validset well without overfitting to just trainset. And its also clear that there are too many training steps.

Among all 5-fold sets, this model performs best to the second fold, while the f1 score is shown in TABLE I

TABLE I
F1 SCORES OF MLP

Normal F1	AF F1	Others F1	Average F1
0.8766	0.7372	0.6173	0.7437

So we can see that though MLP is quite a simple model, it can still perform well in these kind of hard in complex scenes.

By the way, the model is built without a **Softmax** function in the end because the CrossEntropyLoss function from ***Pytorch*** library will do itself, so there is no need to do so.

B. CNN model

CNN model is usually combined with several Convolution layers, some pool layers and a fully connected net for final output. Convolution layers usually works for feature capture, while the pool layer can compress data size so that there will not be too much neurons in the final fully connected net. In the CNN I designed, I came up with a structure as follows:

- 1D-convolution with a Relu
- 1D-MaxPooling
- 1D-convolution with a Relu
- 1D-MaxPooling
- N to 4 MLP output

For the first CNN model, the channel in each conv-layers are (1,6) and (6,16), while kernel-size and padding for both

two model are set as 3 and 1. In the two maxpooling layers, the size of model will be compressed to half of the origin.

Using the same optimizer, setting learning rate as 0.0001, the loss graph for both trainset and validset is shown in Fig. 5.

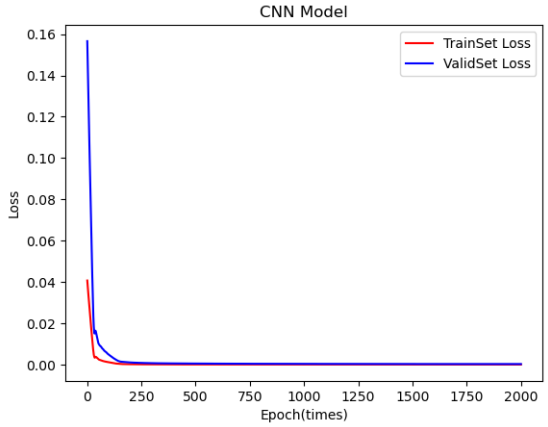

Fig. 5. Loss graph for CNN.

At the same time, the F1 score for this CNN model is shown in TABLE II.

TABLE II
F1 SCORES OF CNN

Normal F1	AF F1	Others F1	Average F1
0.8667	0.8135	0.6667	0.7823

It can be seen that with the aid of convolution and pooling, CNN can perform better than a simple MLP.

If the arguments of the model is changed as still two half-cut pooling, and the convolution channel pairs are (1,3) and (3,6), when the kernel size are 5 and 7 separately, the result of the model is shown in TABLE III.

TABLE III
F1 SCORES OF CNN

Normal F1	AF F1	Others F1	Average F1
0.8810	0.7964	0.6487	0.7753

By comparing the two tables, we can find that different CNN parameters will have different effects on various predictions. In this adjustment, the second group of parameters estimated AF better than the first group, but their estimates of other categories were relatively poor.

After this, we also tried to increase the number of layers of CNN to compare the performance difference of CNN with a large difference in the number of layers.

- 1D-(1 In, 6 Out, k-size 3) convolution with a Relu
- 1D-(6 In, 16 Out, k-size 3) convolution with a Relu
- 1D-MaxPooling
- 1D-(16 In, 32 Out, k-size 3) convolution with a Relu
- 1D-(32 In, 64 Out, k-size 3) convolution with a Relu
- 1D-MaxPooling
- N to 4 MLP output

Compared with the first two models, the number of convolution layers in this CNN has doubled, while the number of pooling layers and the structure of the final MLP output layer remain unchanged.

Keep the optimizer and other parameters unchanged, and the training result of the model is shown in TABLE IV.

TABLE IV
F1 SCORES OF CNN

Normal F1	AF F1	Others F1	Average F1
0.8700	0.8146	0.6749	0.7865

It is not difficult to find that although the number of convolution layers of CNN has been greatly increased, it has not played a better role in optimizing the prediction ability of the model. This may be because when there are two convolution layers, CNN has already extracted the features of the model to the limit. Therefore, the addition of additional convolution layers will not greatly help feature extraction.

At the same time, so many layers will greatly slow down the calculation of the model, so it is obviously not cost-effective to establish so many layers for this task.

C. RNN model

Recurrent neural networks are a type of neural network that allow previous outputs to be used as inputs while generating the next output. This kind of network can often play an important role in various unexpected situations, so this project will also try to apply RNN to prediction.

The RNN network selected in this model is LSTM network. The basic structure of the network is shown below.

- 5-hidden size, 2-hidden layer LSTM
- N to 4 MLP output

The 2-hidden layers means for both hidden state and cell state there will be two layers.

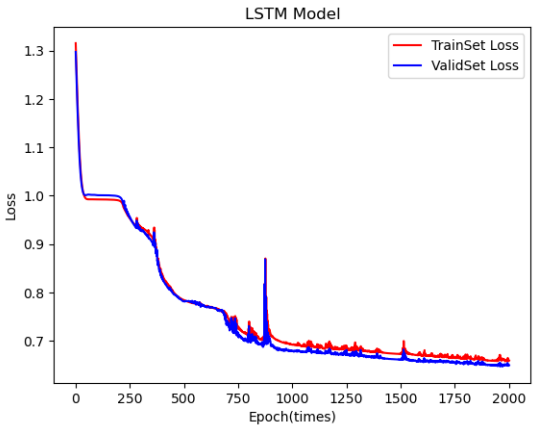

Fig. 6. Loss graph for LSTM.

It can be seen that LSTM network converges to the loss of both types of data during training. This shows that there is no overfitting in this training. The final prediction result of this model is shown in TABLE V.

TABLE V
F1 SCORES OF LSTM

Normal F1	AF F1	Others F1	Average F1
0.8532	0.7045	0.5754	0.7110

It can be seen that the LSTM model, as a relatively complex neural network, has good performance. However, compared with the general MLP, the prediction ability of this network has not been significantly improved. The possible reason is that the RNN model is not very suitable for the task scenario of this project.

For other tests of RNN model, the first thought is to convert LSTM model from single to two-way (i.e. BiLSTM). After modifying the model, the prediction result of the network is shown in TABLE VI.

TABLE VI
F1 SCORES OF BILSTM

Normal F1	AF F1	Others F1	Average F1
0.8909	0.7323	0.6720	0.7651

Compared with single LSTM, BiLSTM has significantly improved performance. The new model has significant improvement in various types of prediction, which shows that it is more effective to extract data features for RNN to analyze series from two directions at the same time.

However, since the model is bidirectional, the training and prediction time of this model is directly doubled compared to the previous model. Especially in view of the complexity and time-consuming nature of the RNN model calculation, we need to carefully consider whether it is worth using such a large amount of overhead to gain such an improvement when selecting the model.

D. RNN with Attention

Attention mechanisms are a type of neural network layer that allow the model to focus on certain parts of the input when processing it. This can be useful for tasks where the input has a long sequence of elements, and the model needs to selectively pay attention to certain elements in order to make a prediction.

Because the attention mechanism has various complex forms, which are very difficult to implement, the model used in this project is only to add an attention pooling layer to the two-layer RNN structure. The detailed structure of the model is as follows.

- (1 In-size, 5 hidden-size, 2 layers) LSTM
- Batch-Norm layer with Relu
- Point-wise attention
- (5 In-size, 5 hidden-size, 2 layers) LSTM
- N to 4 MLP output

Due to the unfamiliar attention mechanism, the attention mechanism used in this model is Point-wise attention. Compared with the previous RNN model, the RNN model with attention mechanism is more complex. The loss convergence curve of the model is shown in Fig. 7.

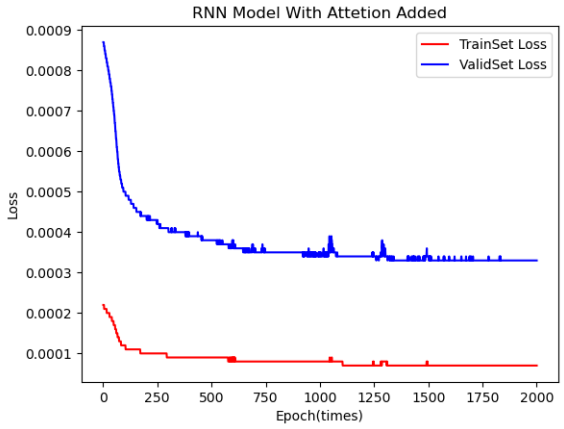

Fig. 7. Loss graph for RNN with Attention.

The comparison of prediction parameters of three RNN models is shown in TABLE VII

TABLE VII
COMPARISON BETWEEN THREE RNN MODEL

Model	Normal F1	AF F1	Others F1	Average F1
Normal LSTM	0.8532	0.7045	0.5754	0.7110
BiLSTM	0.8909	0.7323	0.6720	0.7651
Attention	0.8820	0.6887	0.6884	0.7530

It can be found that compared with the simplest LSTM, the optimization strategies of the two LSTMs are significantly improving. Adding attention mechanism can redistribute weight in the process of model training, so the model can select the characteristics of data better. This is somewhat similar to pooling strategy in CNN model or the linear layer in MLP, and the effect it has achieved is to further extract the features in the data. At the same after using the attention mechanism, the training of the model becomes more smooth. The more intuitive effect is reflected in the model's prediction of the accuracy of the training set. This is shown in Fig. 8.

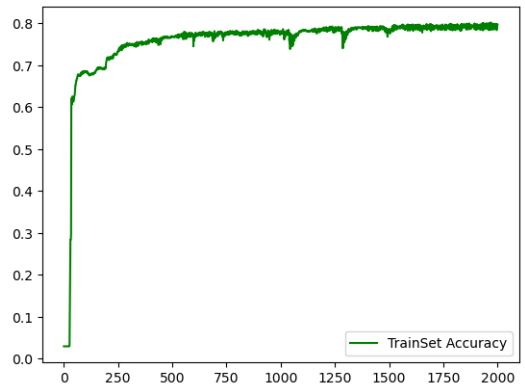

Fig. 8. Accuracy for RNN with Attention.

In the training process of two LSTMs without attention, the decline of loss is not smooth. In fact, loss encountered many bottlenecks in the process of decline, that is, it reached the local minimum and could not jump out. If there is no Adam optimizer, LSTM will eventually converge to this point and will not change. But after the attention mechanism is added, this problem does not appear again in the training of the model. There is no bottleneck in model training, and the model accuracy is relatively high. This should also be one of the advantages of attention mechanism.

IV. Trial and error

In this project, I encountered many problems. For example, at the beginning, the gradient descent process of my model seemed very strange, and the accuracy of prediction was always low (about 60%). I was puzzled by this problem for nearly two days. Finally, other students found that I did not add function **zero_grad()** in the process of model training. This fully shows that I am still very unskilled in the construction of model code. (so I **really really really** hope that there will be lab classes in this class ! Without wasting so much time on this I believe I can achieve better results.)

In addition, I also consulted other related papers on ECG signals [3]. However, even if I spent almost whole day to understand all these papers, due to limited capabilities, I was unable to reproduce the model mentioned in the paper. However, these papers still give me a lot of inspiration. For example, LSTM may have better performance if it is bidirectional. The actual results are also shown in the paper.

What is more, another important finding is that for a model, the starting point of its parameters has the greatest impact on the model performance. This is an obvious result, because the starting point of the model will determine the local optimal solution to which it converges, and different local optimal solutions will greatly limit the prediction ability of the model. Therefore, a good choice is more important than later efforts.

V. Conclusion

Firstly, thanks my lecturer and TA very much for your help in learning this course. You have given me a deeper understanding of AI and machine learning.

At the same time, special thanks to chat gpt [4]. The function of this network learning model is really powerful, which makes me greatly appreciate the role of machine learning and deep learning. I hope I can make something as powerful and practical as chat gpt in the future study and research. This project is still a big challenge for me. As a person who is only familiar with the principles, the task of writing the model for the first time is very difficult. But luckily, it took me nearly two weeks. Although the final effect was not ideal, it was still acceptable to me.

References

[1] Pyakillya B, Kazachenko N, Mikhailovsky N. Deep learning for ECG classifcation. J Phys Conf Ser. 2017;913:012004.

[2] Goodfellow SD, Goodwin A, Greer R, Laussen PC, Mazwi M, Eytan D. Towards understanding ecg rhythm classifcation using convolutional neural networks and attention mappings. In: Machine learning for healthcare conference, 2018; pp. 83–101.

[3] Cheng J, Zou Q, Zhao Y. ECG signal classification based on deep CNN and BiLSTM. BMC Med Inform Decis Mak. 2021 Dec 28;21(1):365. doi: 10.1186/s12911-021-01736-y. PMID: 34963455; PMCID: PMC8715576.

[4] https://chat.openai.com/