[Lab2] EEG Classification

0616014 楊政道

April 14, 2020

1 Introduction

1.1 Lab Objective

In this assignment, I will use pytorch to construct two models, EEGNet and DeepConvNet, with three different activation functions including ReLU, LeakyReLU, and ELU.

1.2 Dataset

This dataset has two channels. There are 750 data points for each channel.

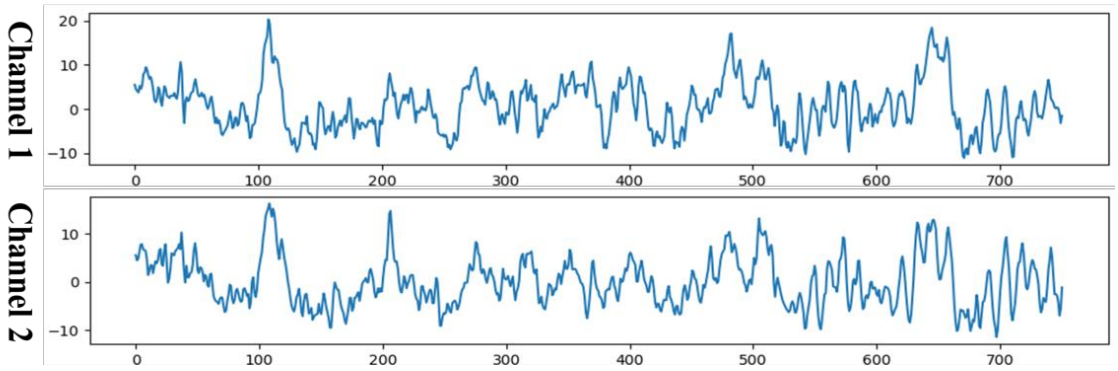

Figure 1: EEG signals

1.3 Project Structure

```
Lab2_0616014.zip
├── data/
│   └── dataloader.py
├── model/
│   ├── DeepConvNet.py
│   └── EEGNet.py
├── weight/
│   └── record.json
├── train.py
└── evaluate.py
```

1.3.1 data/dataloader.py

To generate the pytorch dataloader for our training and testing data.

1.3.2 model/EEGNet.py

To define the structure of EEGNet and its forward function.

1.3.3 model/DeepConvNet.py

To define the structure of DeepConvNet and its forward function.

1.3.4 weight/record.json

To mentain the maximum test accuracy. If we have higher test accuracy, we will replace the weight and update the record file. The model parameters will be stored in weight directory.

1.3.5 train.py

To define the Trainer class and train networks.

1.3.6 evaluate.py

It will load the network parameters we have stored and evaluate the network.

2 Experiment Setup

2.1 EEGNet

2.1.1 structure

```
EEGNet(
  (firstconv): Sequential(
    (0): Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25), bias=False)
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (depthwiseConv): Sequential(
    (0): Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): [Activation Function]()
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=[dropout], inplace=False)
  )
  (separableConv): Sequential(
    (0): Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): [Activation Function]()
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=[dropout], inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

2.2 DeepConvNet

2.2.1 structure

```
DeepConvNet(
  (conv0): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1))
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): [Activation Function]()
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=[dropout], inplace=False)
  )
  (conv1): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): [Activation Function]()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=[dropout], inplace=False)
  )
```

```
  (conv2): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): [Activation Function]()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=[dropout], inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): [Activation Function]()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=[dropout], inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=8600, out_features=2, bias=True)
  )
)
```

2.3 Activation Function

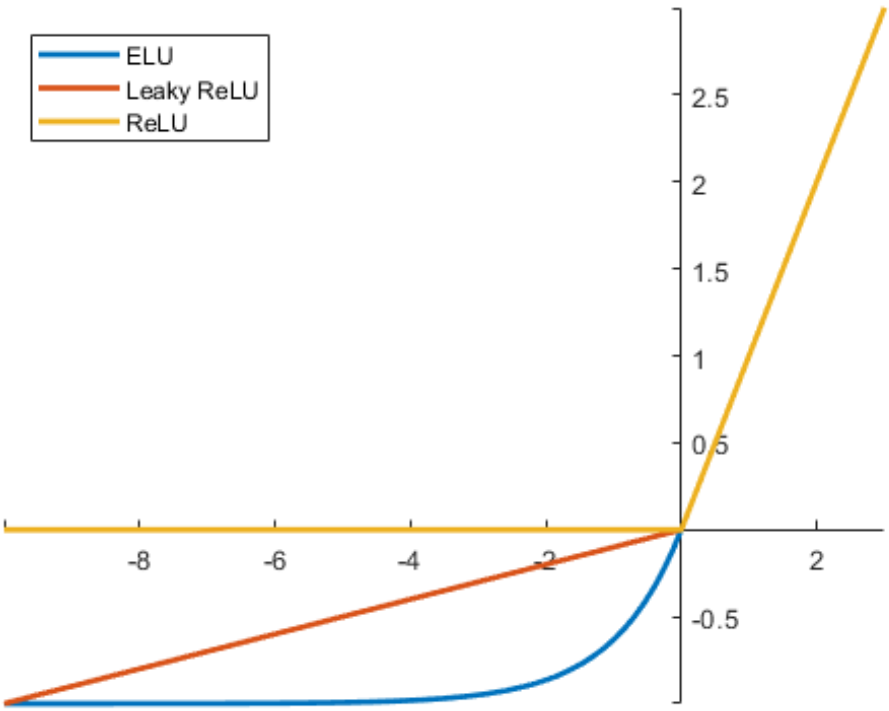

Figure 2: Activation Function

2.3.1 ELU

$$\texttt{ELU}(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

2.3.2 ReLU

$$\texttt{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

2.3.3 LeakyReLU

$$\texttt{LeakyReLU}(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$$

The differences of these three activation functions are the part smaller than zero in x. We can find the gradient in ReLU and ELU wiil shrink into zero as the value of the x smaller. It will make gradient shrink when we initialize the network weight too small.

3 Experimental Result

3.1 Comparison Plot

3.1.1 EEGNet

Hyper parameters:

- optimizer: Adam
- criterion: CrossEntropy
- epoch size: 1000
- batch size: 1080
- learning rate: 0.001

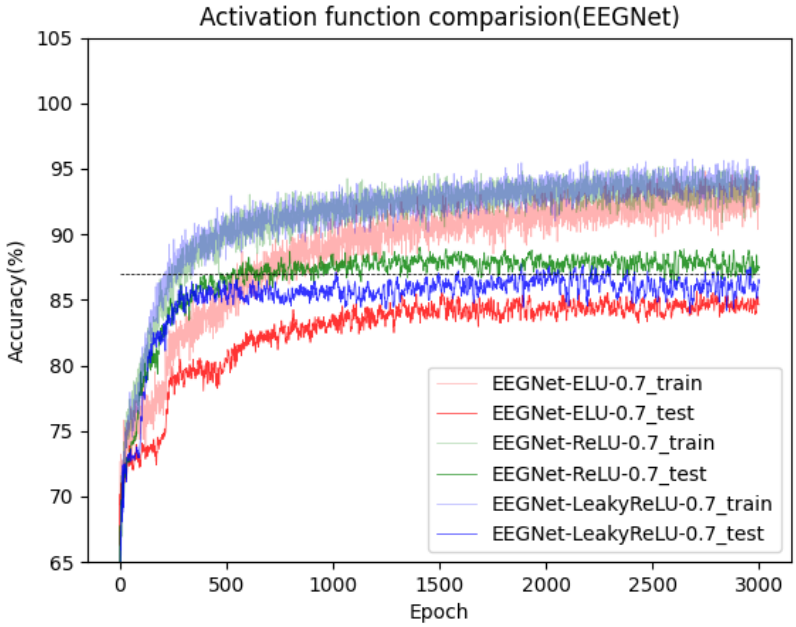

Figure 3: EEGNet Comparison

3.1.2 DeepConvNet

- optimizer: Adam
- criterion: CrossEntropy
- epoch size: 3000
- batch size: 1080
- learning rate: 0.001

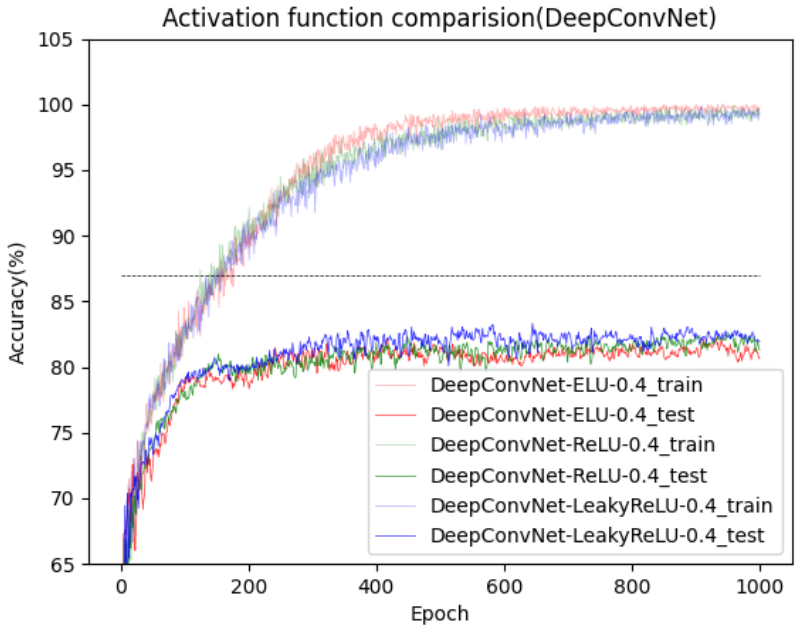

Figure 4: DeepConvNet Comparison

3.2 Highest Test Accuracy

3.2.1 EEGNet

EEGNet			
Model	Acti-Func	Dropout	Max Test Acc
EEGNet	ELU	0.5	85.83%
EEGNet	ELU	0.6	86.76%
EEGNet	ELU	0.7	87.50%
EEGNet	ELU	0.8	82.04%
EEGNet	LeakyReLU	0.5	87.96%
EEGNet	LeakyReLU	0.6	89.91%
EEGNet	LeakyReLU	0.7	89.62%
EEGNet	LeakyReLU	0.8	86.94%
EEGNet	ReLU	0.5	88.06%
EEGNet	ReLU	0.6	89.72%
EEGNet	ReLU	0.7	**90.19%**
EEGNet	ReLU	0.8	90.09%

3.2.2 DeepConvNet

EEGNet			
Model	Acti-Func	Dropout	Max Test Acc
DeepConvNet	ELU	0.4	83.15%
DeepConvNet	ELU	0.5	83.80%
DeepConvNet	ELU	0.6	82.96%
DeepConvNet	LeakyReLU	0.4	**86.02%**
DeepConvNet	LeakyReLU	0.5	85.56%
DeepConvNet	LeakyReLU	0.6	84.26%
DeepConvNet	ReLU	0.4	85.83%
DeepConvNet	ReLU	0.5	84.35%
DeepConvNet	ReLU	0.6	82.50%

4 Discussion

4.1 Dropout

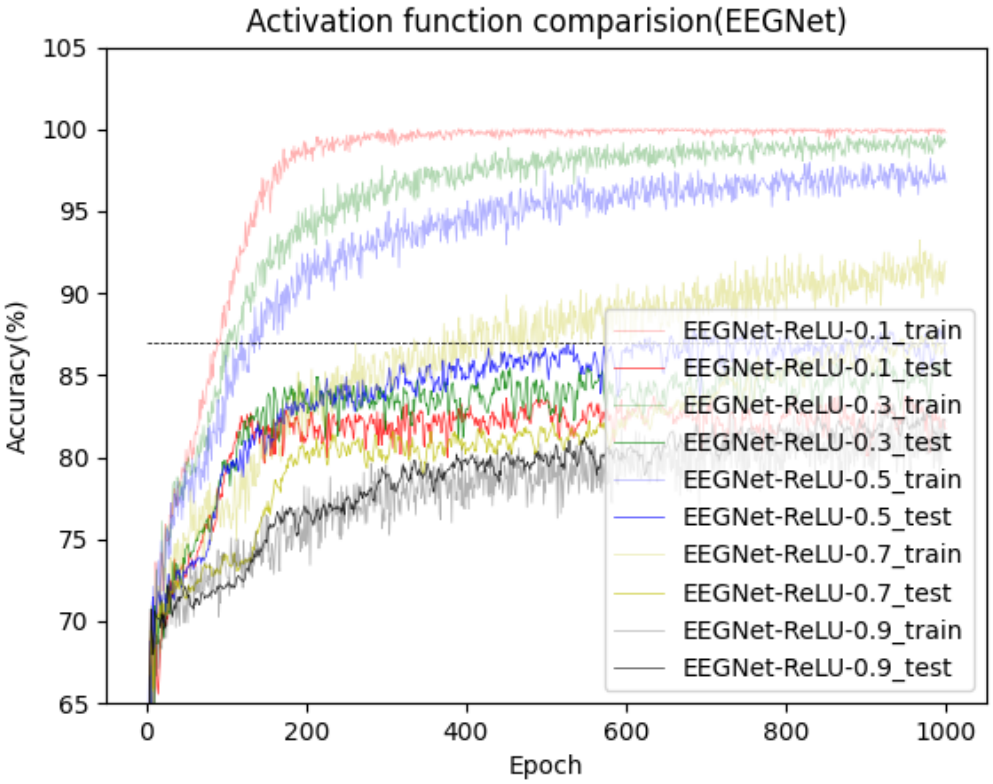

Figure 5: EEGNet vs Dropout

If we set the dropout rate too low(red line), the training accuracy will increase fast but testing accuracy still remains low. The model will be overfitting and perform bad on the test dataset.

On the other hand, if we set the dropout rate too high(black line), the training accuracy can't increase and the network will learning nothing. The testing accuracy, of course, will still remain low. The model will be underfitting and perform bad on the test dataset too.

Therefore, it's important to tune the dropout rate in order to prevent the model from overfitting and underfitting.