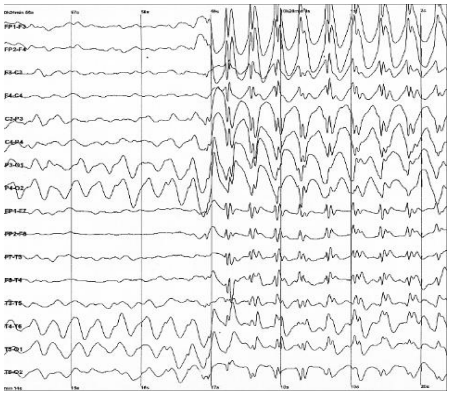
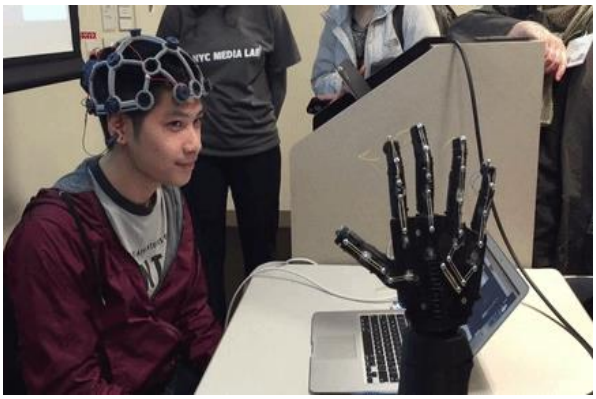
Lab2 : EEG classification

0756614 多媒所 林榆軒

1.Introduction

In this lab, you will need to implement simple EEG classification models which are **EEGNet, DeepConvNet [1]** with **BCI competition dataset**. Additionally, you need to try different kinds of activation function including ReLU, Leaky ReLU, ELU.





The architectures of two model has alread provided in the spec, so this lab is just like a simple pytorch tutorial. Complete all the necessary code and trying to tune the model with different hyper parameters in order to get the better result.

2. Experiment set up

The overall architectures of EEGNet and DeepConvNet are in the spec. I am not sure whether we can modify the modify the architecture in this lab or not, so I just created the exact same models as the images shown in the spec(Lab2-EEG_classfication.pptx p.9 and p.10).

A. The detail of your model

【EEGNet】

```
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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25)
  )
```

```
  (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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

```python
class EEGNET(nn.Module):
    def __init__(self,act=nn.ELU(alpha=1.)):
        super(EEGNET,self).__init__()
        self.firstconv=nn.Sequential(
            nn.Conv2d(1,16,(1,51),(1,1),padding=(0,25),bias=False),
            nn.BatchNorm2d(16,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)
        )
        self.depthwiseConv=nn.Sequential(
            nn.Conv2d(16,32,(2,1),(1,1),groups=16,bias=False),
            nn.BatchNorm2d(32,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True),
            act,
            nn.AvgPool2d((1,4),stride=(1,4),padding=0),
            nn.Dropout(p=0.25)
        )
        self.separableConv=nn.Sequential(
            nn.Conv2d(32,32,(1,15),(1,1),padding=(0,7),bias=False),
            nn.BatchNorm2d(32,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True),
            act,
            nn.AvgPool2d((1,8),stride=(1,8),padding=0),
            nn.Dropout(p=0.25)
        )
        self.classify=nn.Sequential(
            nn.Linear(in_features=736,out_features=2,bias=True)
        )
    def forward(self,x):
        x=self.firstconv(x)
        x=self.depthwiseConv(x)
        x=self.separableConv(x)
        x=x.view(-1,736)
        x=self.classify(x)
        return x
```

【DeepConvNet】

Layer	# filters	size	# params	Activation	Options
Input		(C, T)			
Reshape		(1, C, T)			
Conv2D	25	(1, 5)	150	Linear	mode = valid, max norm = 2
Conv2D	25	(C, 1)	25 * 25 * C + 25	Linear	mode = valid, max norm = 2
BatchNorm			2 * 25		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	50	(1, 5)	25 * 50 * C + 50	Linear	mode = valid, max norm = 2
BatchNorm			2 * 50		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	100	(1, 5)	50 * 100 * C + 100	Linear	mode = valid, max norm = 2
BatchNorm			2 * 100		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	200	(1, 5)	100 * 200 * C + 200	Linear	mode = valid, max norm = 2
BatchNorm			2 * 200		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5

Flatten			
Dense	N	softmax	max norm = 0.5

```
class Flatten(nn.Module):
    def forward(self, input):
        return input.view(input.size(0), -1)

class DeepConvNet(nn.Module):
    def __init__(self,act=nn.ELU(alpha=1.),C=2,T=750,N=2):
        super(DeepConvNet,self).__init__()
        self.net=nn.Sequential(
            nn.Conv2d(1,25,(1,5)),
            nn.Conv2d(25,25,(C,1)),
            nn.BatchNorm2d(25,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True),
            act,
            nn.MaxPool2d((1,2),stride=(1,2),padding=0),
            nn.Dropout(p=0.5),
            nn.Conv2d(25,50,(1,5)),
            nn.BatchNorm2d(50,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True),
            act,
            nn.MaxPool2d((1,2),stride=(1,2),padding=0),
            nn.Dropout(p=0.5),
            nn.Conv2d(50,100,(1,5)),
            nn.BatchNorm2d(100,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True),
            act,
            nn.MaxPool2d((1,2),stride=(1,2),padding=0),
            nn.Dropout(p=0.5),
            nn.Conv2d(100,200,(1,5)),
            nn.BatchNorm2d(200,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True),
            act,
            nn.MaxPool2d((1,2),stride=(1,2),padding=0),
            nn.Dropout(p=0.5),
            Flatten(),
            nn.Linear(8600,2)
        )
    def forward(self,x):
        x=self.net(x)
        return x
```

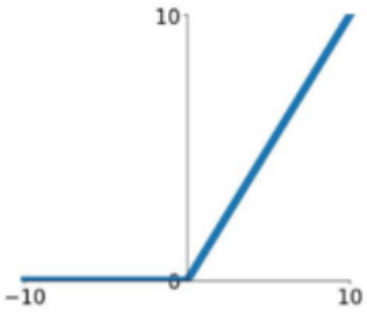
B. Explain the activation function (ReLU, Leaky ReLU, ELU)

The main idea of the activation function is turning the linear process into the non-linear process. Without the activation funtion, all the layers in the model can be simplified as an easy matrix multiplication, the input data and the ouput still has a linear relationship with each other, problem like XOR can not be sovled by such linear transform. So the activation function is essential in the deep learning model.

【ReLU】

ReLU

$$\max(0, x)$$



The two key points of useing ReLU as activation is that:

1. Solve the vanishing gradient problem

When using the sigmoid or tanh as the activation function, often encounter the vanishing gradient problem, which causes the gradient

approaches 0, and the backpropagation process can not be succefully done.

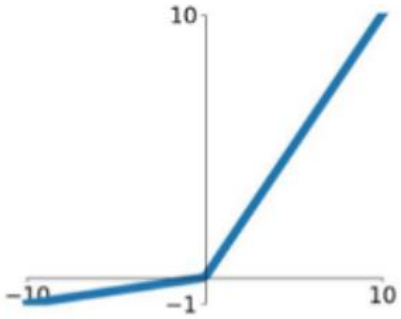
2.Occam's Razor rule

Relu can cause some units' ouput become zero, in this waym, it can relieve the overfitting problem during training process.

【Leaky ReLU】

Leaky ReLU

$$\max(0.1x, x)$$

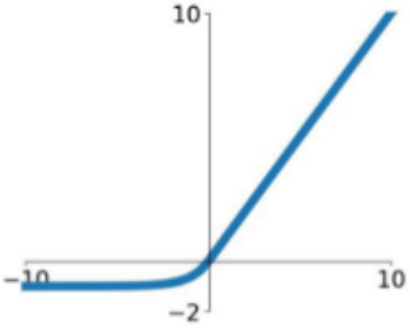


LeakyRelu is a special case of relu function. In the relu function, when all the input of this neuron is negative, the gradient pass through this unit will be zero, in other word, this neuron die. Once the neuron die, it is impossible back to life, the unit will complete loss its function. Therefore, the LeakyRelu is introduce to solve this problem. The neuron won't die and only cause the extremely small gradient.

【ELU】

ELU

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



Same as LeakyRelu function, the purpose of ELU is to solve ‘zero dying’ problem in Relu, but it combine some good features of both Relu and LeakyReLU. Theoretically, it should be better than LeakyReLU, but there has no direct evidence.

3. Experimental results

A. The highest testing accuracy

	ReLU	Leaky ReLU	ELU
EEGNet	**87.5%**	**87.8%**	82.5%
DeepConvNet	82%	81.5%	81.9%

· EEGNet

```
LeakyReLU(negative_slope=0.01) Epoch: [300], Loss: 0.00125
LeakyReLU(negative_slope=0.01)  Acc: 0.8787037037037037
```

```
ReLU() Epoch: [300], Loss: 0.00578
ReLU()  Acc: 0.875
```

```
ELU(alpha=1.0) Epoch: [300], Loss: 0.05872
ELU(alpha=1.0)  Acc: 0.825925925925926
```

· DeepConvNet

```
LeakyReLU(negative_slope=0.01) Epoch: [300], Loss: 0.19296
LeakyReLU(negative_slope=0.01)  Acc: 0.8157407407407408
```

```
ReLU() Epoch: [300], Loss: 0.17716
ReLU()  Acc: 0.8203703703703704
```

```
ELU(alpha=1.0) Epoch: [300], Loss: 0.09084
ELU(alpha=1.0)  Acc: 0.8194444444444444
```

【Screenshot with two models】

```
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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25)
  )
  (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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

```
DeepConvNet(
  (net): 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): ELU(alpha=1.0)
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5)
    (6): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1))
    (7): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ELU(alpha=1.0)
    (9): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (10): Dropout(p=0.5)
    (11): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1))
    (12): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ELU(alpha=1.0)
    (14): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (15): Dropout(p=0.5)
    (16): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1))
    (17): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): ELU(alpha=1.0)
    (19): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (20): Dropout(p=0.5)
    (21): Flatten()
    (22): Linear(in_features=8600, out_features=2, bias=True)
  )
)
```

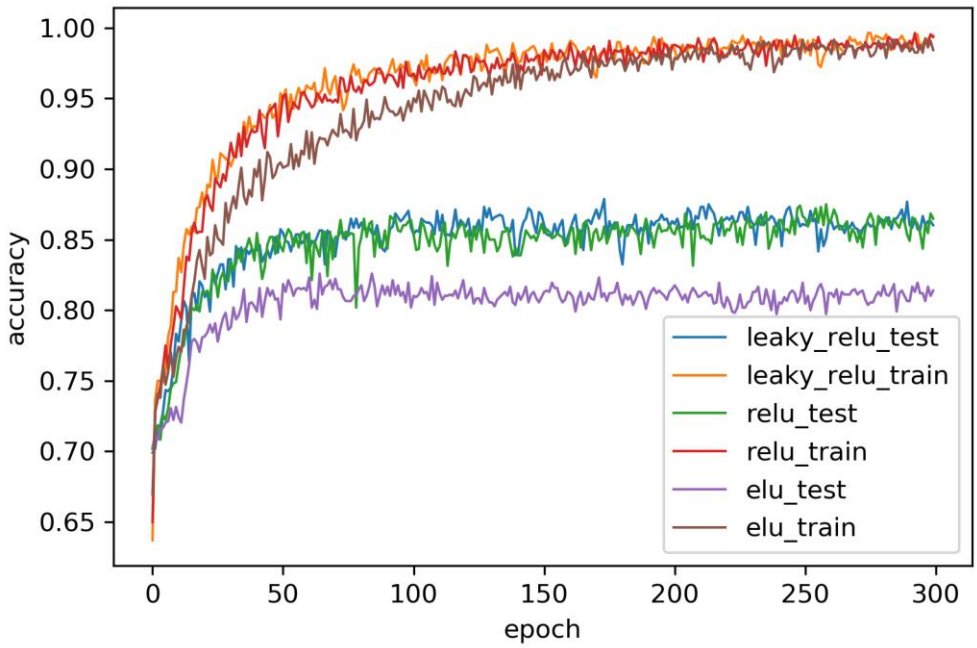
The result from the above table shows that EEGNet is outperformance, it is because that the lab only requires two of the results to beyond 87% accuracy in order to get 100 score. I only spent the time on tuning EEGNet.After I reach the criterion on EEGNet, I just decided to give up to find the better hyper parameters for DeepConvNet. That is the reason why the accuaracy is quite low on the DeepConNet.

【anything you want to present】
The hyper parameters for EEGNet and DeepConvNet are as follow:

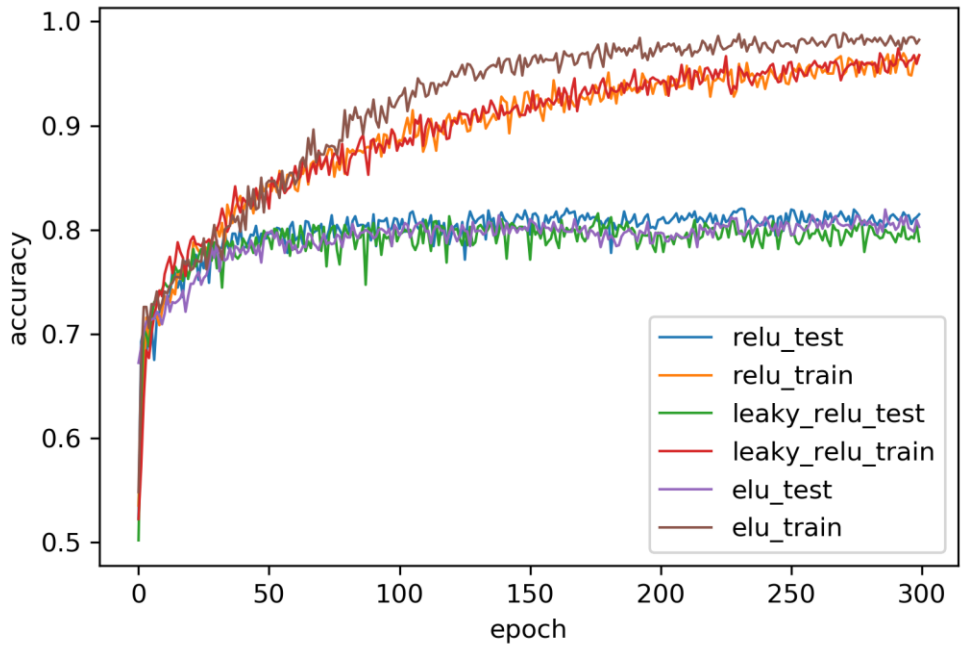
EEGNet
>Batch size= 32
>Learning rate = 1e-3
>Epochs = 300
>Optimizer: Adam
>Loss function: torch.nn.CrossEntropyLoss()

DeepConvNet
>Batch size= 32
>Learning rate = 5e-4
>Epochs = 300
>Optimizer: Adam
>Loss function: torch.nn.CrossEntropyLoss()

B. Comparison figures
【EEGNet】



【DeepConvNet】



4. Discussion

Theoretically, the DeepConvNet should have better accuracy than EEGNet, but the result in my experiment is in contrast. I think the main reason is that : I did not spent time on tuning the DeepConvNet, I just randomly pick some hyper parameter that make the result looks not too bad. Also the DeepConvNet is absolutely larger and deeper , so it is obviously hard to train than the EEGNet.

In the EEGNet, although different activation function can reach almost the same high accuracy ,but the testing result shows that the one with ELU as activation function is significantly worse than the others. The result of the ReLU and the LeakyReLU are really similar with each other.

It didn’t take too much time on finish all the code for the lab2. But it definitely need a while to find the better hyper parameters for the model, it was such a tedious work. Anyway, this Lab2 is a good tutorial for pytorch, and can learn a lot from it.