

Experiment Report

311552024 詹偉翔

1. Introduction

In this lab, we will implement the EEG classification that is EEGNet and DeepConvNet. Moreover, we will change the inside activation function with ReLU, LeakyReLU, and ELU. To see which combination will be the best model with highest accuracy.

2. Experiment Setup

A. The detail of my model

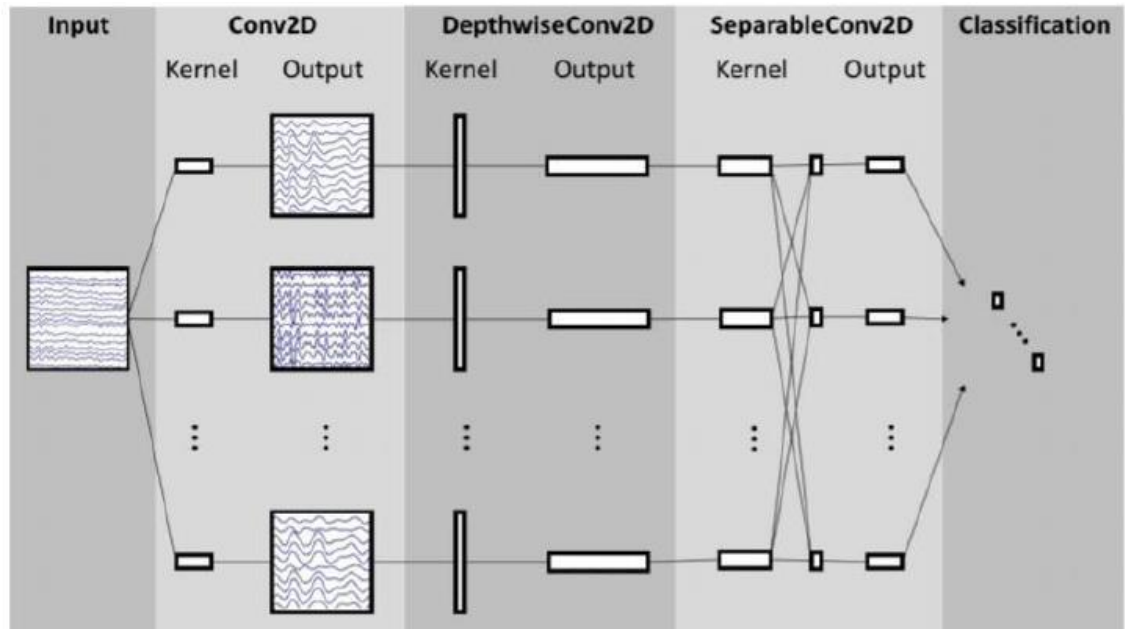
I set the parameters below:

- learning rate = 0.0007
- batch size = 1024
- epochs = 1000

i. EEGNet

This is one of popular models to solve EEG classification problem.

There are three convolution blocks : Firstconv , Depthwisedconv, and separableconv. The filter is more than previous blocks in convolution blocks. The whole architecture is the same as the architecture that TA provides.



This is architecture of EEGNet.

```
class EEGNet(nn.Module):
    def __init__(self, activation):
        super(EEGNet, self).__init__()
        self.device = torch.device("cuda")
        # Layer 1
        self.firstconv = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding = (0, 25), bias = False),
            nn.BatchNorm2d(16, eps=1e-05, momentum=0.1, affine = True, track_running_stats = True)
        )
        # Layer 2
        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias = False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine = True, track_running_stats = True),
            activation,
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding = 0),
            nn.Dropout(p=0.25)
        )
        # Layer 3
        self.separableConv = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding = (0, 7), bias = False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine = True, track_running_stats = True),
            activation,
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding = 0),
            nn.Dropout(p=0.25)
        )
        # fc Layer
        self.classify = nn.Sequential(nn.Linear(736, 2, bias = True))

    def forward(self, x):
        x = self.firstconv(x)
        x = self.depthwiseConv(x)
        x = self.separableConv(x)
        x = x.view(x.shape[0], -1)
        output = self.classify(x)
        return output
```

Here is my code.

ii. DeepConvNet

It is used to compare with EEGNet. There are more layers about 5 convolution blocks and more parameters in the DeepConvNet, so it need more time to training, and I believe that if we have enough time,

this model will be better than EEGNet. The architecture in my program is the same as the below architecture.

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

The architecture of DeepConvNet is this, where $C = 2$, $T = 750$, and $N = 2$.

```

class DepConvNet(nn.Module):
    def __init__(self, activation):
        super(DepConvNet, self).__init__()
        self.device = torch.device("cuda")
        self.ficrtconv = nn.Conv2d(1, 25, kernel_size=(1, 5))
        self.convblock1 = nn.Sequential(
            nn.Conv2d(25, 25, kernel_size=(2, 1)),
            nn.BatchNorm2d(25, eps=1e-05, momentum=0.1),
            activation,
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5)
        )
        self.convblock2 = nn.Sequential(
            nn.Conv2d(25, 50, kernel_size=(1, 5)),
            nn.BatchNorm2d(50, eps=1e-05, momentum=0.1),
            activation,
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5)
        )
        self.convblock3 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size=(1, 5)),
            nn.BatchNorm2d(100, eps=1e-05, momentum=0.1),
            activation,
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5)
        )
        self.convblock4 = nn.Sequential(
            nn.Conv2d(100, 200, kernel_size=(1, 5)),
            nn.BatchNorm2d(200, eps=1e-05, momentum=0.1),
            activation,
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5)
        )
        self.classify = nn.Sequential(nn.Linear(8600, 2))

    def forward(self, x):
        x = self.ficrtconv(x)
        x = self.convblock1(x)
        x = self.convblock2(x)
        x = self.convblock3(x)
        x = self.convblock4(x)
        x = x.view(x.shape[0], -1)
        output = self.classify(x)
        return output

```

Here is my code.

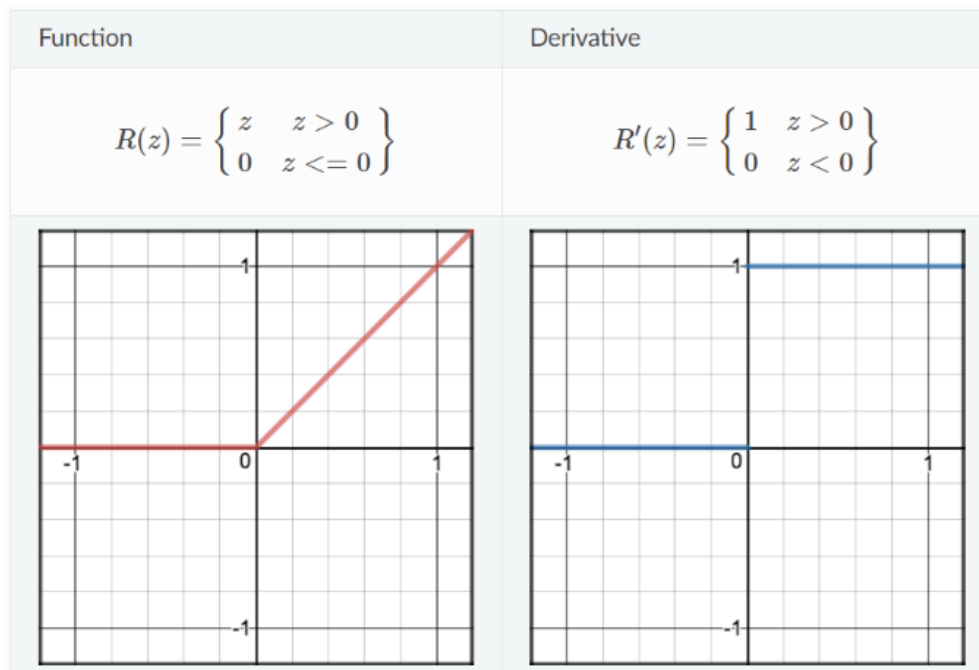
B. Explain the activation function

The activation function is used to make the linear function become non-linear function.

i. ReLU

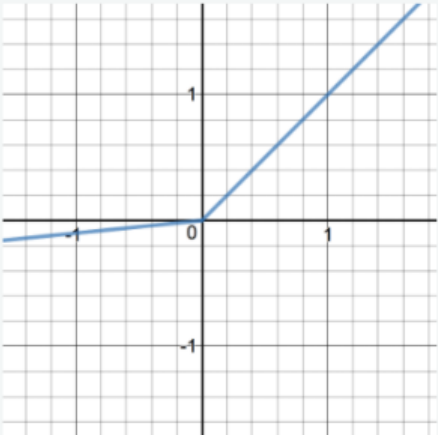
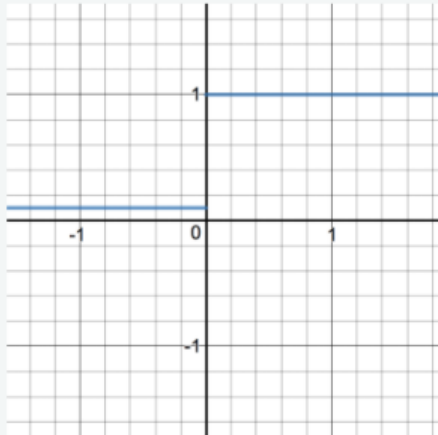
Relu : $\max(0, z)$, is one of the most used activation function. It has the advantage as the sigmoid, but it is easier to calculate.

There are some problem of it, that is when z is below than 0, the weights can't be update!



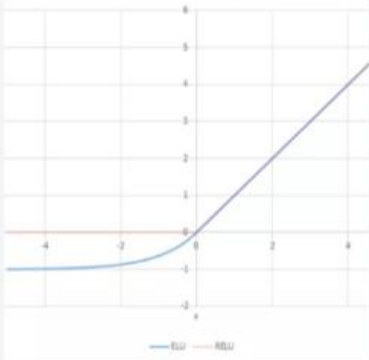
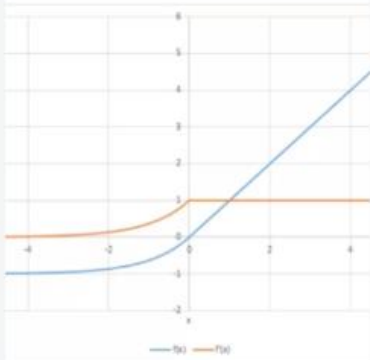
ii. LeakyReLU

It is the ReLU's brother. It solve the problem of ReLU, that is ReLU can't update weights when $z < 0$. Also, It can be calculated as negative number. But it is most like a linear function, it can't be used in complex classify problem.

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ \alpha z & z \leq 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ \alpha & z \leq 0 \end{cases}$
	

iii. ELU

It smooth more slowly than ReLU and it also has a alpha can decides its curve when $z \leq 0$. Moreover, it is the same as LeakyReLU, that is it can output negative number.

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ \alpha \cdot (e^z - 1) & z \leq 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ \alpha \cdot e^z & z \leq 0 \end{cases}$
	

3. Experiment Result

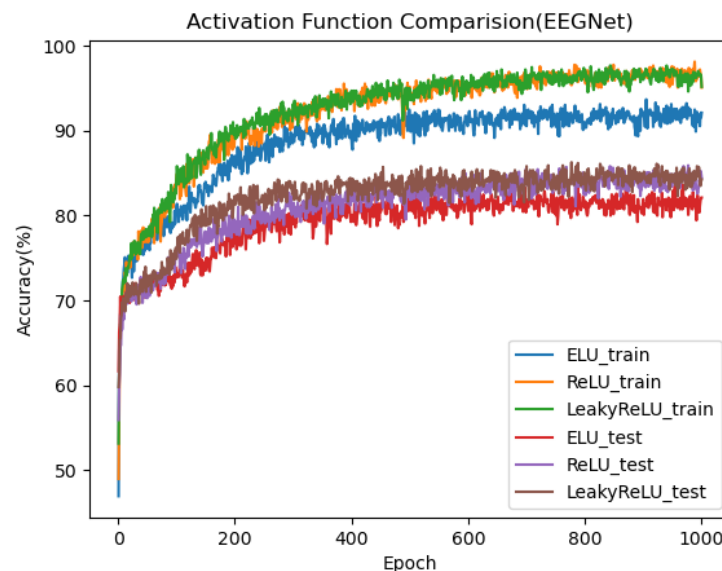
A. The highest testing models

	ELU	ReLU	LeakyReLU
EEGNet	82.12%	87.59%	86.30%
DeepConNet	80.46%	82.5%	82.59%

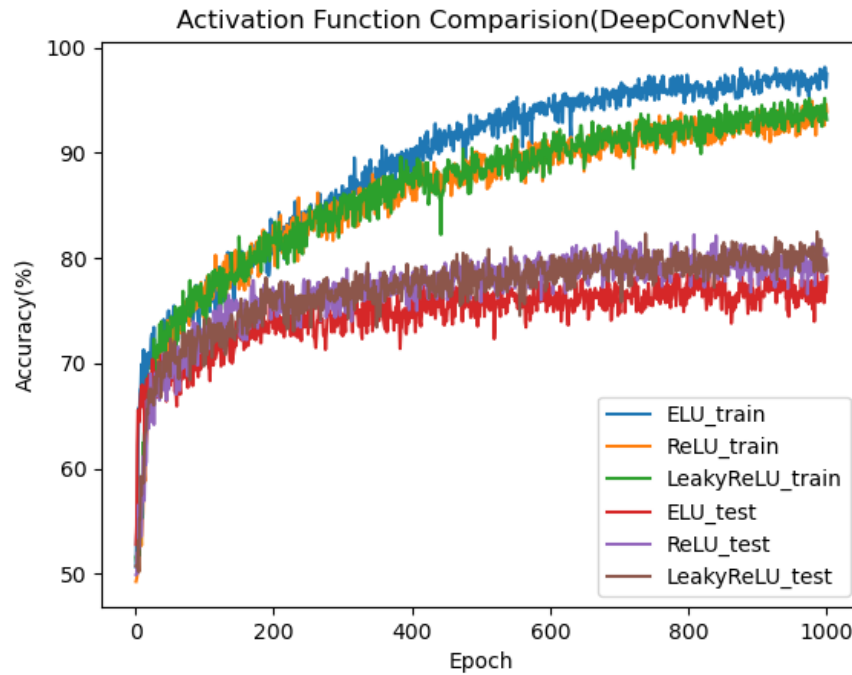
```
EEGNet_ELU test accuracy is : 82.12962962962963
DepConvNet_ELU test accuracy is : 80.46296296296296
EEGNet_ReLU test accuracy is : 87.5925925925926
DepConvNet_ReLU test accuracy is : 82.5
EEGNet_LeakyReLU test accuracy is : 86.29629629629629
DepConvNet_LeakyReLU test accuracy is : 82.5925925925926
End!
```

B. Comparison figures

i. EEGNet



ii. DeepConvNet



4. Discussion

There is unexpected that the performance of DeepConvNet is worse than the performance of EEGNet. I thought maybe the epochs is not enough to make DeepConvNet converge or this model isn't fit to this dataset.

But still the learning curve is expected, the more layers model will cost more time to converge.

5. Source

A. https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html