深度學習 HW3

機器人學程 李啟安 310605015

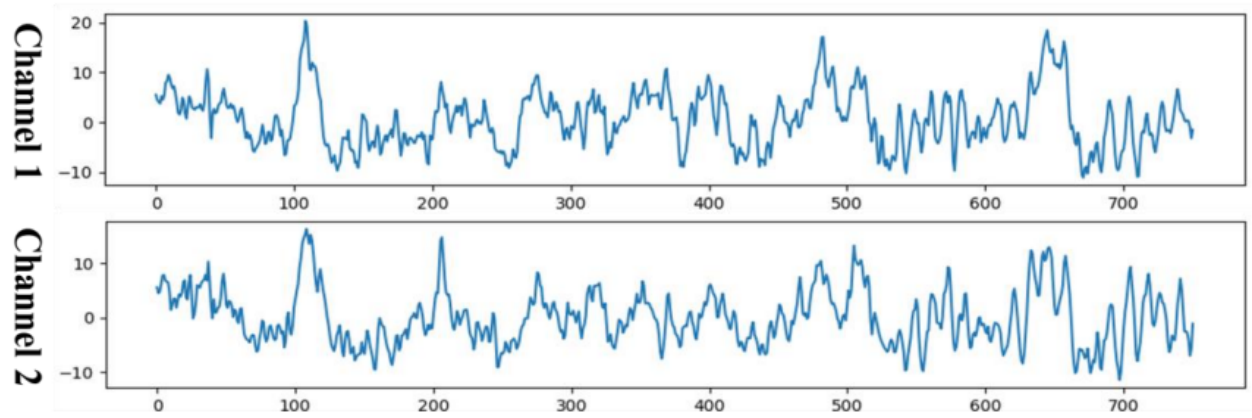
tags: 深度學習

[TOC]

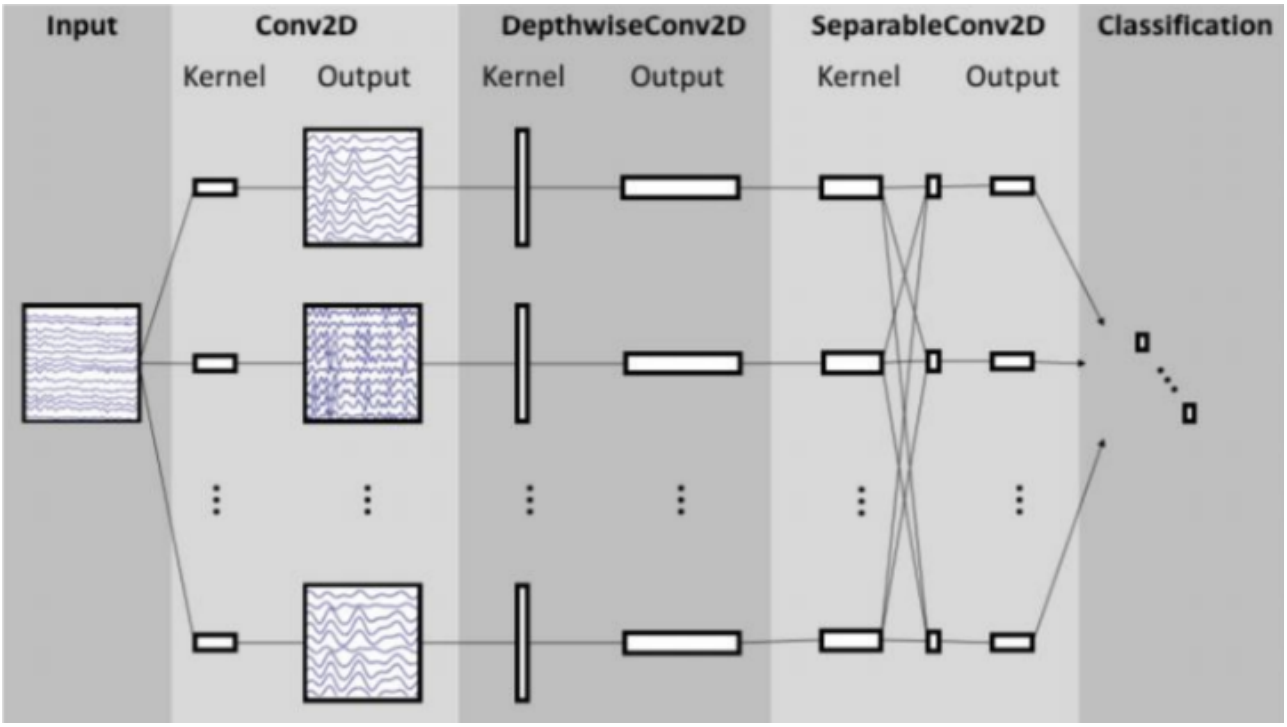
1. Introduction

In this lab, we will build a simple EEG classification models which are EEGNet and DeepConvNet with BCI competetion dataset. We will anaylze different activation function, such as ReLU, LeakyReLU and ELU.

- Data:



- EEGNet



- 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

2. Experiment Setup

A. The detail of your model

- Initialize

```
if args.network == "DeepConvNet":
    self.model = DeepConvNet(args.activation_func)

elif args.network == "EEGNet":
    self.model = EEGNet(args.activation_func)
```

- We can enter desire model (EEGNet or DeepConvNet) and also with desire acitvation function we want to use.

- EEGNet

```
self.firstconv = nn.Sequential(
    # because the kernal size is different, so we choose different padding in height and width
    # Height and Width is different!
    nn.Conv2d(1,16, kernel_size=(1,51), stride = (1, 1), padding=(0,25), bias=False),
    nn.BatchNorm2d(16, affine=True)
)

self.depthwiseConv = nn.Sequential(
    nn.Conv2d(16,32,kernel_size=(2,1),stride=(1,1),groups=16,bias=False),
    nn.BatchNorm2d(32, affine=True),
    activation[activation_func],
    nn.AvgPool2d(kernel_size=(1, 4), stride=(1 , 4), padding=0),
    nn.Dropout(p=0.25)
)

self.separableConv = nn.Sequential(
    nn.Conv2d(32, 32, kernel_size=(1, 15), stride=(1,1), padding=(0,7), bias=False),
    # learn gamma and beta
    nn.BatchNorm2d(32, affine=True),
    activation[activation_func],
    nn.AvgPool2d((1, 8), stride=(1, 8), padding = 0),
    nn.Dropout(p=0.25)
)

self.classify = nn.Sequential(
    nn.Linear(736, 2, bias=True)
)
```

There are four blocks in EEGNet. FirstConv, depthwiseConv, seperableConv and classification. Based on TA's instruction, we can create the network.

- DeepConvNet

```
self.featurelayer1 = nn.Sequential(
    # parameters = Cin*h*w*Cout + Cout(bias)
    # https://towardsdatascience.com/pytorch-conv2d-weights-explained-ff7f68f652eb
    nn.Conv2d(1,25, kernel_size=(1,5), stride = (1, 2), bias=True),
    nn.Conv2d(25,25,kernel_size=(2,1), bias=True),
    # update gamma and beta parameters = 2x25
    nn.BatchNorm2d(25, affine=True),
    activation[activation_func],
    # kernel size, stride
    nn.MaxPool2d(1,2),
    nn.Dropout(0.5)

)

self.featurelayer2 = nn.Sequential(
    nn.Conv2d(25, 50, kernel_size = (1,5), stride = (1,2), bias = True),
    nn.BatchNorm2d(50, affine = True),
    activation[activation_func],
    nn.MaxPool2d((1,2)),
    nn.Dropout(0.5)
)

self.featurelayer3 = nn.Sequential(
    nn.Conv2d(50,100,kernel_size=(1,5), bias = True),
    nn.BatchNorm2d(100, affine=True),
    activation[activation_func],
    nn.MaxPool2d((1,2)),
    nn.Dropout(0.5)
)

self.featurelayer4 = nn.Sequential(
    nn.Conv2d(100,200,kernel_size=(1,5), bias= True),
    nn.BatchNorm2d(200, affine=True),
    activation[activation_func],
    nn.MaxPool2d((1,2)),
    nn.Dropout(0.5)
)

self.classify = nn.Linear(1600, 2, bias = True)
```

The DeepConvNet is more complicated. It has five blocks, such as "featurelayer1", …."featurelayer4" and "classify". In the last layer, the weight has constraint of max_norm = 0.5, so I create the constraint class.

```
class weightConstraint(object):
    def __init__(self):
        pass

    def __call__(self, module):

        if hasattr(module,'weight'):
            print("Entered")
            w=module.weight.data
            w = torch.clamp(w, max = 0.5)
            module.weight.data=w
```

In the training.py, we can implement the constraint on the classify layer.

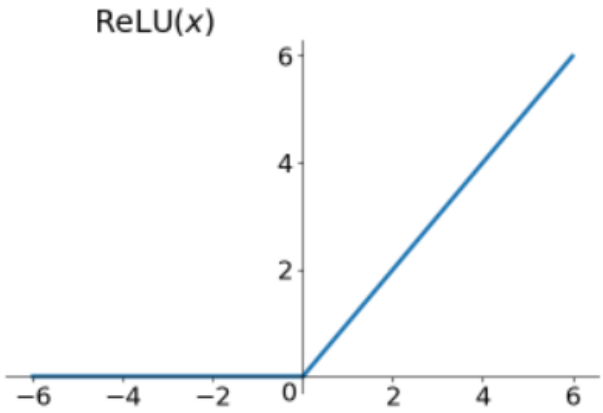
```
self.constraint = weightConstraint()

# setup max_norm constraint
if args.constraint:
    self.model._modules['classify'].apply(self.constraint)
```

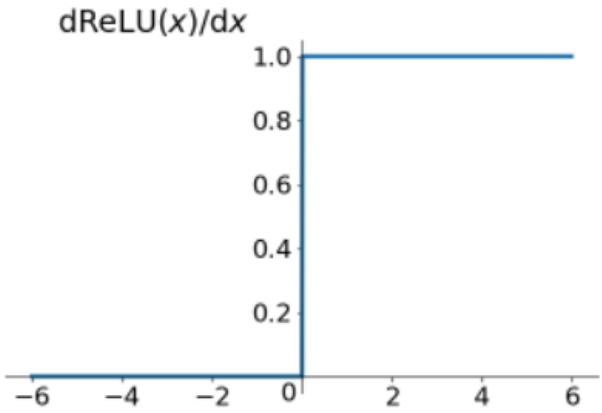
args.constraint is a bool variable to determine whether the constraint is able.

B. Explanation of the activation function

參考自 https://zhuanlan.zhihu.com/p/25110450

ReLU





- Formula:

$$ReLU(x) = max(0, x)$$

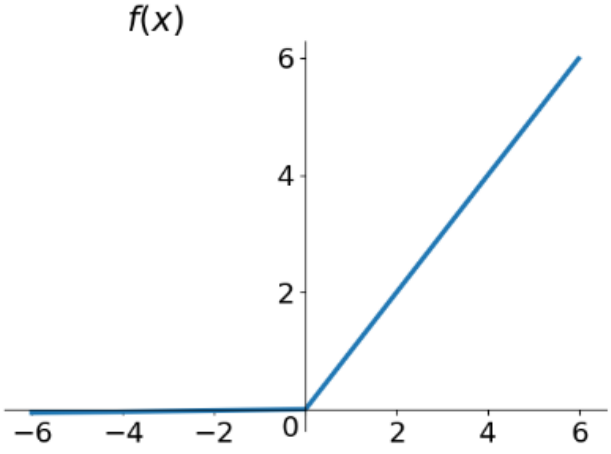
- Advantage:

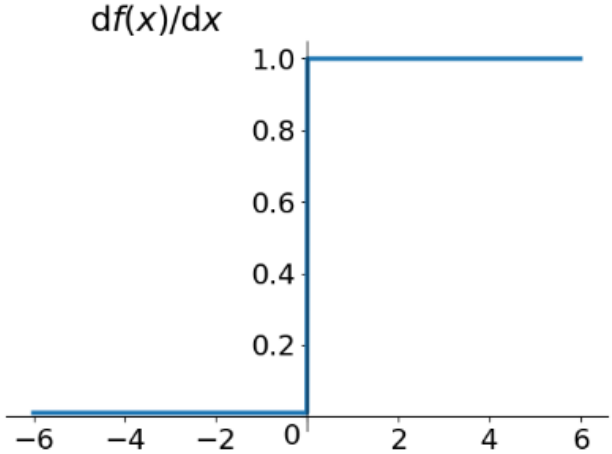
1. Solve gradient vanishing (in Positive side)
2. caculate quickly
3. converger more quickly than tanh, sigmoid

- Disadvantage:

1. Dead ReLU Problem : ReLU neurons become inactive and only ouput 0 for any input

Leaky ReLU





- Formula

$$f(x) = max(0.01x, x)$$

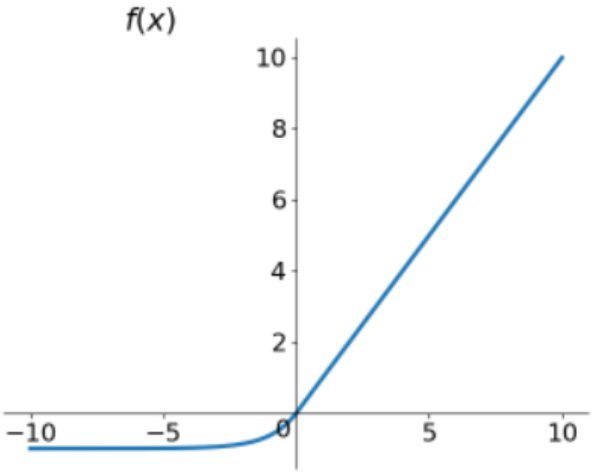
- Advantage:

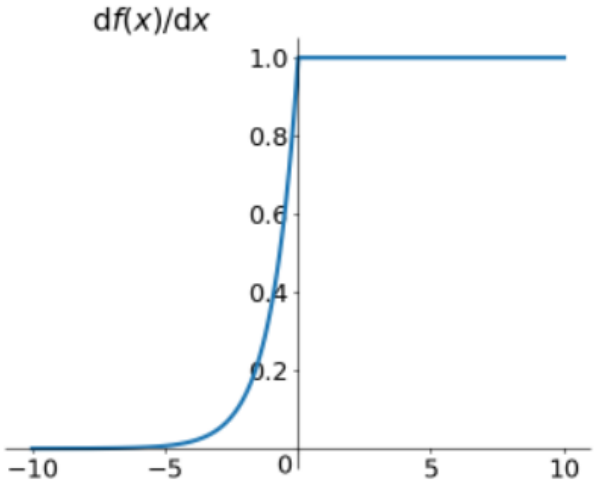
1. All the advantages ReLU has
2. Avoid Dead ReLU problem

- Disadvantage:

1. 沒有完全證明任何情況下總是比ReLU好

ELU Network (Exponential Layer Unit)





- Formula:

$$x < 0 : ELU\left(x\right) = e^{x} - 1 \; x > 0 : ELU\left(x\right) = x$$

- Advantage：
    - Avoid Dead ReLU Problem
    - Zero-centered
- Disadvantage
    - 沒有被完全證明任何情況都好於 ReLU
    - 計算量稍微大

3. Experimental Result

A. The highest testing accuracy

- DeepConvNet

```
DeepConvNet(
  (featurelayer1): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 2))
    (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): LeakyReLU(negative_slope=0.01)
    (4): MaxPool2d(kernel_size=1, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (featurelayer2): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 2))
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (featurelayer3): 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): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (featurelayer4): 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): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (classify): Linear(in_features=1600, out_features=2, bias=True)
)
```

- learning_rate: 0.002
- epochs: 300
- batch: 12

Activation Function	ReLU	ELU	LeakyReLU
Test Accuracy	81.296	78.241	80.648

- 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): LeakyReLU(negative_slope=0.01)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25, 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): LeakyReLU(negative_slope=0.01)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

- learning_rate: 0.0002
- epochs: 300
- batch: 12

Activation Function	ReLU	ELU	LeakyReLU
Test Accuracy	85	82.96	**86.296**

Get the model > 87%

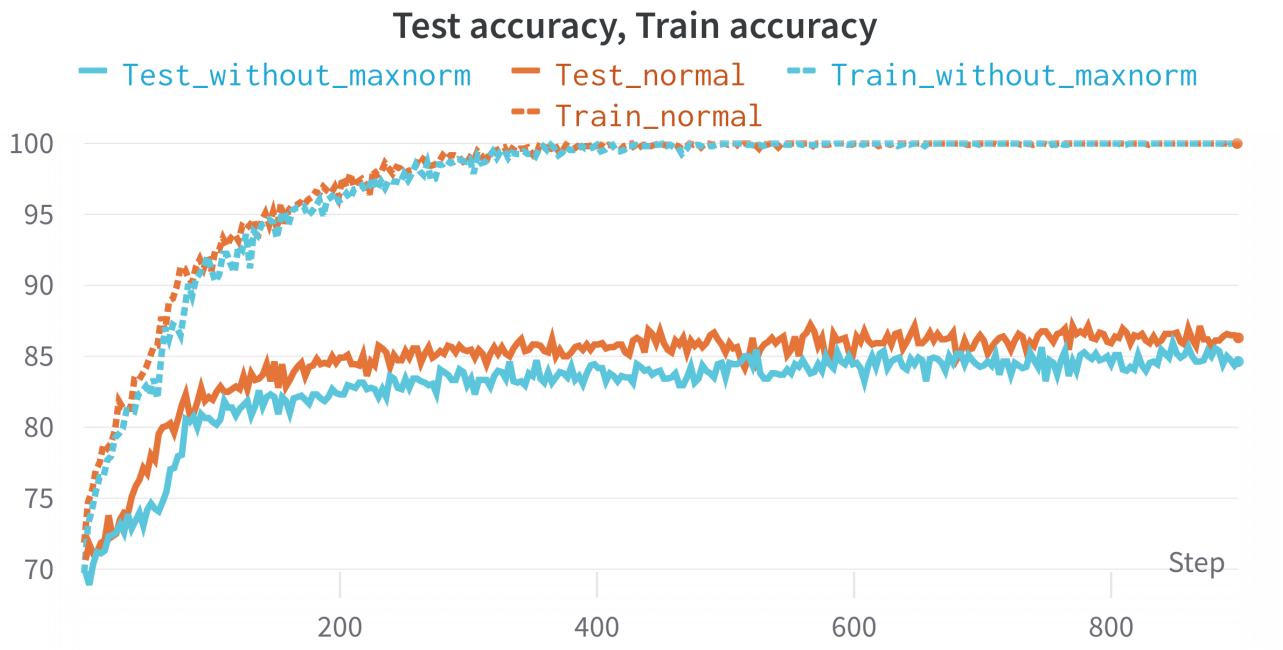
- I use the EEG_LeakyReLU as a **pretrained model** and turn the learning_rate to 0.00001 to continue training
- Then once the test accuracy > 87%, terminate the training process and save the model!

- **EEGNet with LeakyReLU is the best**

B. Anything you want to present

1. Without max_norm constraint

Try to delete the max_norm weight constatint. And we can see that the performance is almost the same in training_data, but huge different in testing_data.



Max_norm	Yes	No
Test Accuracy	84.019	86.296

2. Different batch_size

- learning_rate: 0.0002
- epochs: 300

Batch_size	10	12	14
Test Accuracy	84.019	86.296	86.019

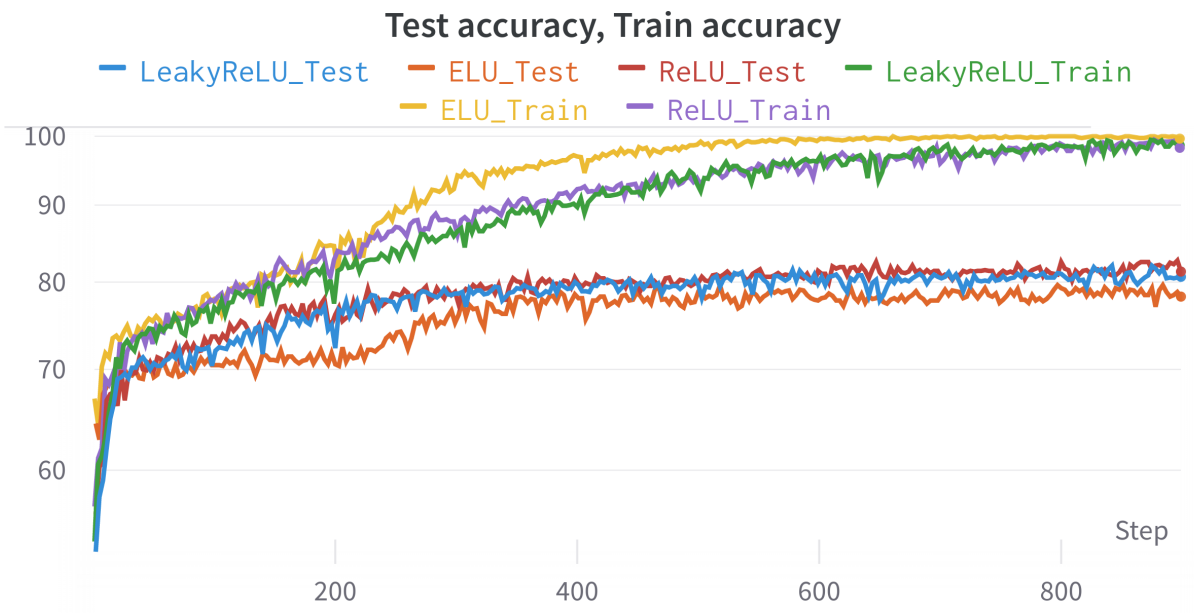
3. Different learning_rate

- batch_size : 12
- epochs: 300

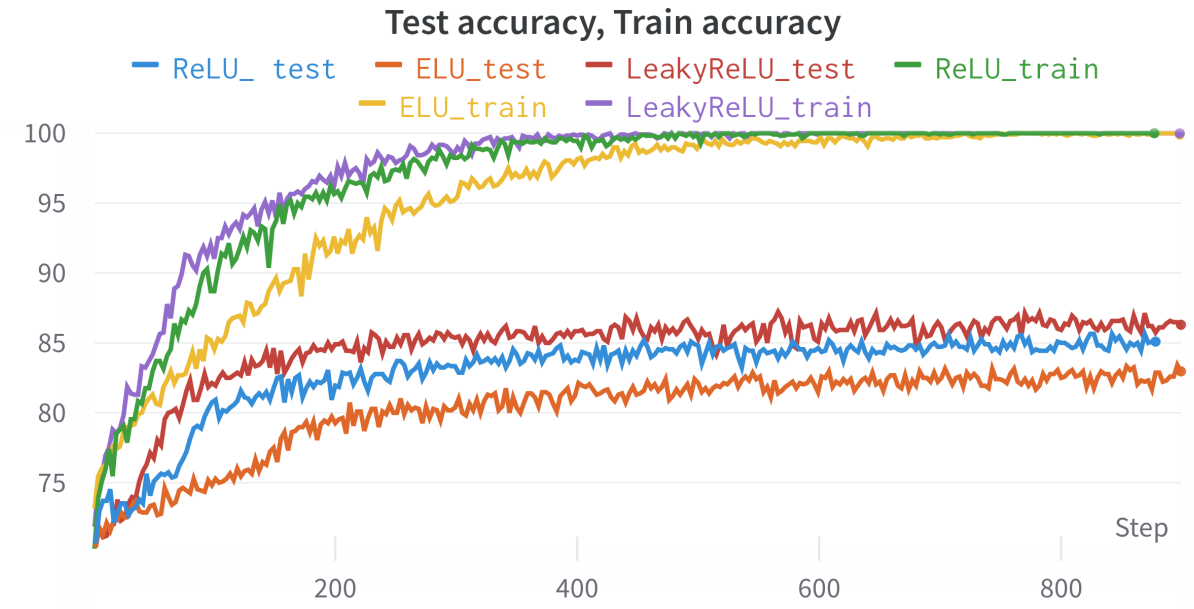
Learning_rate	0.0002	0.002	0.02
Test Accuracy	86.296	85.185	79.259

C. Comparison figures

- DeepConvNet



- EEGNet



We can see that EEGNet is better than DeepConvNet!

4. Discussion

1. It is very interesting that although DeepConvNet has deeper NN but the performace is not as good as EEG

2. Wandb is a really good tools ! I can deal with all my model online and won't lose them

3. Use arg.praser to define all argument!

```
parser.add_argument("network", nargs='?',default = "EEGNet",help="DeepConvNet or EEGNet")

# activation function
parser.add_argument("activation_func",  nargs='?',default = "LeakyReLU", help = "LeakyReLU, ReLU, ELU")

# batch size
parser.add_argument("batch_size", nargs='?',default= 32, type=int ,help= "batch size")

# cuda enable
parser.add_argument("cuda_enable", nargs='?',default= True ,help= "")

# learning rate
parser.add_argument("learning_rate", nargs='?',default=0.00001, type = float,help= "learning_rate, default is 0.003 ")

# epochs
parser.add_argument("epochs", nargs='?',default = 300, type = int,help= "epochs, default is 300 ")

# constraint
parser.add_argument("constraint", nargs='?',default = True , help= "Last layer constraint")

# test
parser.add_argument("test", nargs='?',default = 0 , help= "Testing Model ?")

# test
parser.add_argument("modelname", nargs='?',default = weight_folder + "bestgogogo" , help= "Testing Model ?")

# load model
parser.add_argument("load", nargs='?',default = 0 )

args = parser.parse_args()
```

4. Code Architecture