Lab02 - EEG classification

Jeff Cheng 410551012

GitHub - jeffchengtw/DeepLearning: DLP in NCTU

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/jeffchengtw/DeepLearning

jeffchengtw/
DeepLearning

DLP in NCTU

1 Contributor 0 Issues 0 Stars 0 Forks

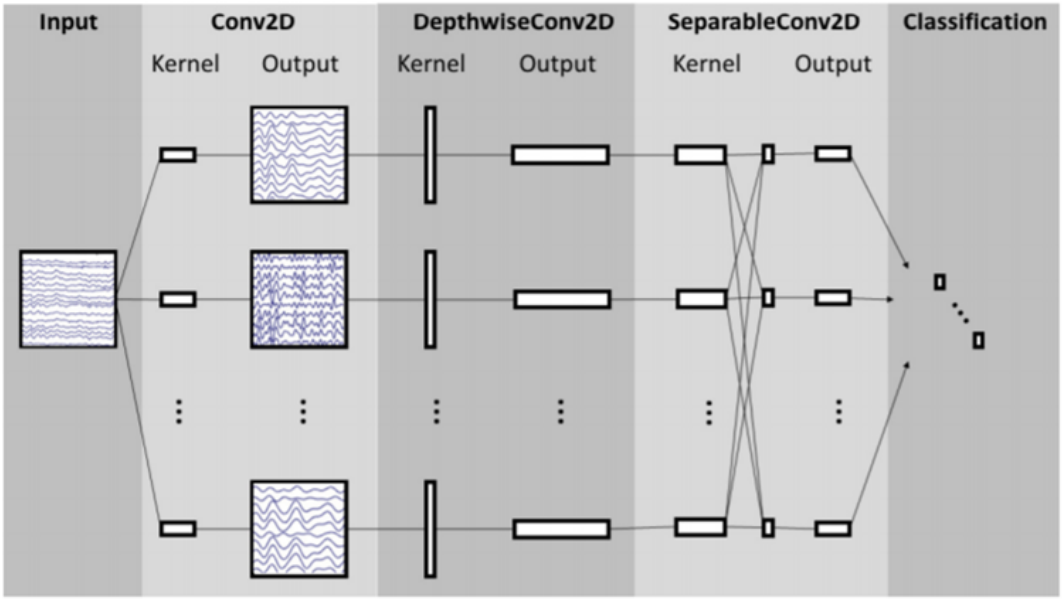
Introduction

Lab Objective

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』.

EEGNet

From the architecture of the model, it can be seen that the second and third layers use different convolution methods from the past, called Depthwise Convolution and Separable Convolution.



Depthwise separable convolution

The basic idea of separable convolution

Separate the kernel into two smaller kernels.

For example :

$$\begin{bmatrix} 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Instead of doing one convolution with 9 multiplications, we do two convolutions with 3 multiplications each (6 in total) to achieve the same effect. With less multiplications, computational complexity goes down, and the network is able to run faster.

The main issue with the spatial separable convolution is that not all kernels can be “separated” into two, smaller kernels. This becomes particularly bothersome during training, since of all the possible kernels the network could have adopted, it can only end up using one of the tiny portion that can be separated into two smaller kernels.

Depthwise convolution :

- Filter number = input channel number
- Each filter only considers one channel
- There is no interaction between channels

Pointwise convolution :

Pointwise convolution must consider the relationship across channels, scan the feature map from depthwise convolution, and get a new feature map.

DeepConvNet

The following figure shows the network architecture that the Lab02 needs to implement.

I define it as **3 blocks.**

Block

“Conv2D” , we can see that the number of parameters is 150. The reason for the “150” is $25 \times 1 \times 5 + 25$, there are 25 filters with each size $1 * 5$ and 25 bias here.
“BatchNormLayer” , the number of parameters is “2*25” as the table shown which means that there are 25 features with 2 channel.
‘mode = valid ’ : it means no padding and it assumes that all the dimensions are valid so that the input image gets fully covered by a filter and the stride specified by you.

Lab objective

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 * 5 + 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 * 5 + 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 * 5 + 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

My implementation

```
       Layer (type)               Output Shape         Param #
================================================================
           Conv2d-1          [-1, 25, 2, 746]             150
           Conv2d-2          [-1, 25, 1, 746]           1,275
      BatchNorm2d-3          [-1, 25, 1, 746]              50
             ELU-4          [-1, 25, 1, 746]               0
        MaxPool2d-5          [-1, 25, 1, 373]               0
        Dropout2d-6          [-1, 25, 1, 373]               0
           Conv2d-7          [-1, 50, 1, 369]           6,300
      BatchNorm2d-8          [-1, 50, 1, 369]             100
             ELU-9          [-1, 50, 1, 369]               0
       MaxPool2d-10          [-1, 50, 1, 184]               0
       Dropout2d-11          [-1, 50, 1, 184]               0
          Conv2d-12         [-1, 100, 1, 180]          25,100
     BatchNorm2d-13         [-1, 100, 1, 180]             200
            ELU-14         [-1, 100, 1, 180]               0
       MaxPool2d-15          [-1, 100, 1, 90]               0
       Dropout2d-16          [-1, 100, 1, 90]               0
          Conv2d-17          [-1, 200, 1, 86]         100,200
     BatchNorm2d-18          [-1, 200, 1, 86]             400
            ELU-19          [-1, 200, 1, 86]               0
       MaxPool2d-20          [-1, 200, 1, 43]               0
       Dropout2d-21          [-1, 200, 1, 43]               0
          Linear-22                 [-1, 2]          17,202
================================================================
Total params: 150,977
Trainable params: 150,977
Non-trainable params: 0
----------------------------------------------------------------
```

Experiment set up

A. The detail of model

- EEGNet

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 2, 750]             816
       BatchNorm2d-2          [-1, 16, 2, 750]              32
            Conv2d-3          [-1, 32, 1, 750]              64
       BatchNorm2d-4          [-1, 32, 1, 750]              64
              ReLU-5          [-1, 32, 1, 750]               0
              ReLU-6          [-1, 32, 1, 750]               0
              ReLU-7          [-1, 32, 1, 750]               0
         AvgPool2d-8          [-1, 32, 1, 187]               0
           Dropout-9          [-1, 32, 1, 187]               0
           Conv2d-10          [-1, 32, 1, 187]          15,360
      BatchNorm2d-11          [-1, 32, 1, 187]              64
             ReLU-12          [-1, 32, 1, 187]               0
             ReLU-13          [-1, 32, 1, 187]               0
             ReLU-14          [-1, 32, 1, 187]               0
        AvgPool2d-15           [-1, 32, 1, 23]               0
          Dropout-16           [-1, 32, 1, 23]               0
           Linear-17                 [-1, 2]           1,474
================================================================
Total params: 17,874
Trainable params: 17,874
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.61
Params size (MB): 0.07
Estimated Total Size (MB): 1.69
----------------------------------------------------------------
```

- DeepConvNet

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 25, 2, 746]             150
            Conv2d-2          [-1, 25, 1, 746]           1,275
       BatchNorm2d-3          [-1, 25, 1, 746]              50
              ELU-4          [-1, 25, 1, 746]               0
         MaxPool2d-5          [-1, 25, 1, 373]               0
         Dropout2d-6          [-1, 25, 1, 373]               0
            Conv2d-7          [-1, 50, 1, 369]           6,300
       BatchNorm2d-8          [-1, 50, 1, 369]             100
              ELU-9          [-1, 50, 1, 369]               0
        MaxPool2d-10          [-1, 50, 1, 184]               0
        Dropout2d-11          [-1, 50, 1, 184]               0
           Conv2d-12         [-1, 100, 1, 180]          25,100
      BatchNorm2d-13         [-1, 100, 1, 180]             200
             ELU-14         [-1, 100, 1, 180]               0
        MaxPool2d-15          [-1, 100, 1, 90]               0
        Dropout2d-16          [-1, 100, 1, 90]               0
           Conv2d-17          [-1, 200, 1, 86]         100,200
      BatchNorm2d-18          [-1, 200, 1, 86]             400
             ELU-19          [-1, 200, 1, 86]               0
        MaxPool2d-20          [-1, 200, 1, 43]               0
        Dropout2d-21          [-1, 200, 1, 43]               0
           Linear-22                 [-1, 2]          17,202
================================================================
Total params: 150,977
Trainable params: 150,977
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.49
Params size (MB): 0.58
Estimated Total Size (MB): 3.07
----------------------------------------------------------------
```

B. Explaination of activation function

ReLU (Rectified Linear Unit)

Formulas such as (1), which are often used for neuron outputs, can therefore be written as (2).

The ReLU is half rectified (from bottom). Output is zero when z is less than zero and output is equal to z when z is above or equal to zero.

Pros:

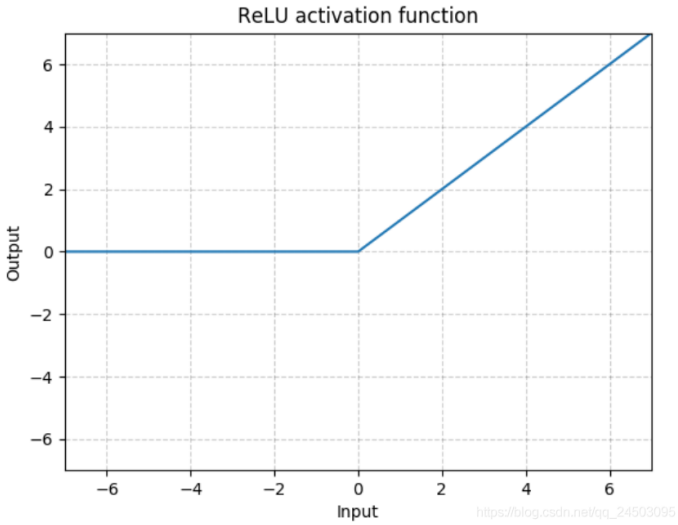
- It avoids and rectifies vanishing gradient problem.
- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.

Cons:

The negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

$$f(x) = max(0, x) \tag{1}$$

$$max(0, w^T x + b) \tag{2}$$

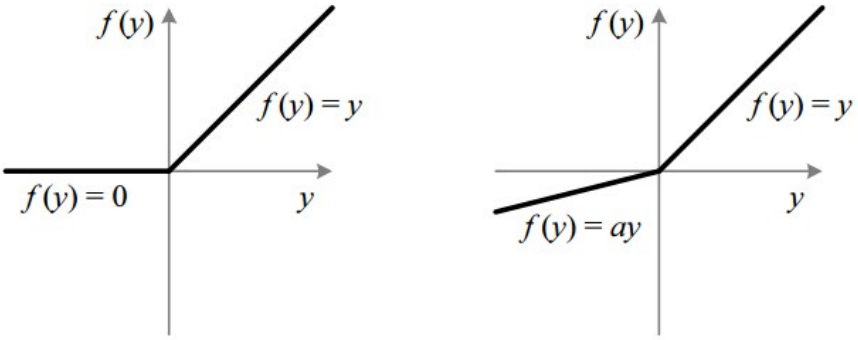


Leaky ReLU

When the input value x is negative, the gradient of the leaky linear rectification function (Leaky ReLU) is a constant $\lambda \in (0, 1)$.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \lambda x & \text{if } x \leq 0 \end{cases}$$

Comparison of ReLU and Leaky ReLU



Pros:

- Leaky ReLUs are one attempt to fix the “dying ReLU” problem by having a small negative slope (of 0.01, or so).

Cons:

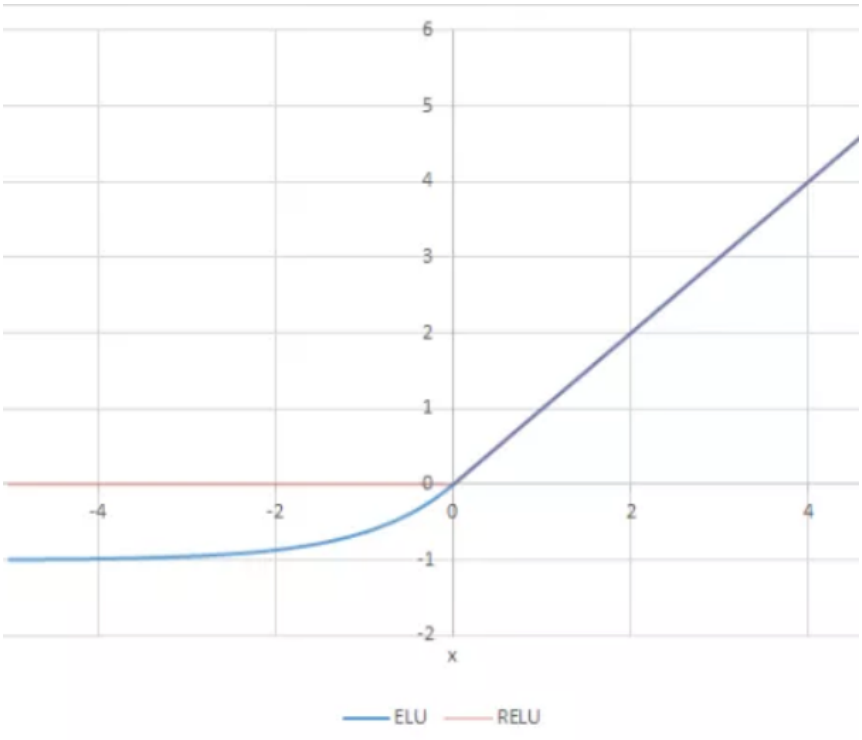
- As it possess linearity, it can’t be used for the complex Classification. It lags behind the Sigmoid and Tanh for some of the use cases.

ELU (Exponential Linear Unit)

Exponential Linear Unit or its widely known name ELU is a function that tend to converge cost to zero faster and produce more accurate results. Different to other activation functions, ELU has a extra alpha constant which should be positive number.

ELU is very similiar to RELU except negative inputs. They are both in identity function form for non-negative inputs. On the other hand, ELU becomes smooth slowly until its output equal to -α whereas RELU sharply smoothes.

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



Pros:

- ELU becomes smooth slowly until its output equal to -α whereas RELU sharply smoothes.
- ELU is a strong alternative to ReLU.
- Unlike to ReLU, ELU can produce negative outputs.

Cons:

- For x > 0, it can blow up the activation with the output range of [0, inf].

Experimental results

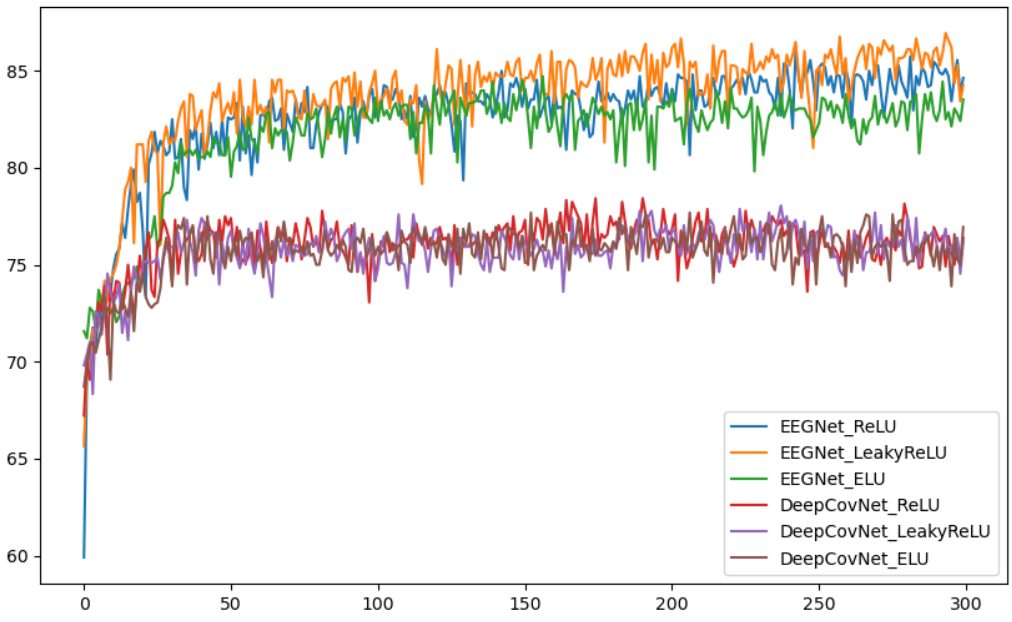
```
# hyperparameters
EPOCH = 300
batch_size = 128
LEARNING_RATE = 1e-3
```

A. The highest testing accuracy

```
Epoch : 284 Loss : 0.77 Accuracy : 76.39 % Best accuracy : 77.78 %
Epoch : 285 Loss : 0.80 Accuracy : 76.39 % Best accuracy : 77.78 %
Epoch : 286 Loss : 0.80 Accuracy : 76.20 % Best accuracy : 77.78 %
Epoch : 287 Loss : 0.84 Accuracy : 77.96 % Best accuracy : 77.96 %
Epoch : 288 Loss : 0.85 Accuracy : 75.56 % Best accuracy : 77.96 %
Epoch : 289 Loss : 0.81 Accuracy : 77.31 % Best accuracy : 77.96 %
Epoch : 290 Loss : 0.69 Accuracy : 75.37 % Best accuracy : 77.96 %
Epoch : 291 Loss : 0.79 Accuracy : 75.74 % Best accuracy : 77.96 %
Epoch : 292 Loss : 0.87 Accuracy : 76.94 % Best accuracy : 77.96 %
Epoch : 293 Loss : 0.79 Accuracy : 75.93 % Best accuracy : 77.96 %
Epoch : 294 Loss : 1.01 Accuracy : 76.76 % Best accuracy : 77.96 %
Epoch : 295 Loss : 0.84 Accuracy : 77.50 % Best accuracy : 77.96 %
Epoch : 296 Loss : 0.90 Accuracy : 76.11 % Best accuracy : 77.96 %
Epoch : 297 Loss : 0.86 Accuracy : 76.67 % Best accuracy : 77.96 %
Epoch : 298 Loss : 0.92 Accuracy : 77.41 % Best accuracy : 77.96 %
Epoch : 299 Loss : 0.94 Accuracy : 74.91 % Best accuracy : 77.96 %
                ReLU LeakyReLU      ELU
EEGNet      87.6852   87.7778  85.0926
DeepCovNet  78.6111   78.6111   77.963

(pytorch) D:\NCTU\2022_02\DeepLearning\Lab02>
```

```
             ReLU     LeakyReLU   ELU
EEGNet      87.6852   87.7778   85.0926
DeepCovNet  78.6111   78.6111   77.963
```

B. Comparison figures



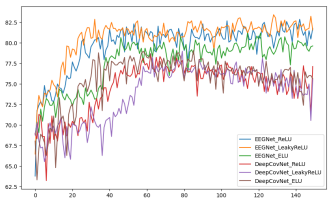
Discussion

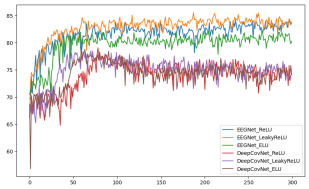
A. Effect of epoch

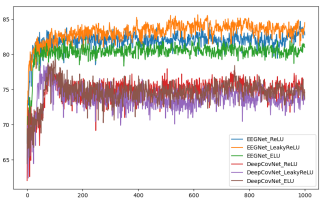
At the beginning of training using the hyperparameters provided on the briefing paper, but the test accuracy cannot meet the full score requirements (87%). So first confirm whether the training times are not enough, so I increased the number of epochs. However, as the number of epochs increased, **Even if more epochs are trained, the accuracy of the model is not improved.**

	EEG_ReLU	EEG_LeakyReLU	EEG_ELU	DeepConvReLU	DeepConvLeakyReLU	DeepConv_ELU
epoch = 150	82.87	83.42	81.20	78.51	78.98	79.07
epoch = 300	84.16	85.55	82.40	79.07	78.51	79.25
epoch = 1000	84.72	85.64	82.40	78.14	78.51	79.25

```
##################################################################
# hyperparameters
EPOCH = 150
batch_size = 64
LEARNING_RATE = 1e-2

Result :

               ReLU   LeakyReLU   ELU
EEGNet      82.8704   83.4259  81.2037
DeepCovNet  78.5185   78.9815  79.0741
##################################################################
# hyperparameters
EPOCH = 300
batch_size = 64
LEARNING_RATE = 1e-2

Result :

                ReLU  LeakyReLU  ELU
EEGNet      84.1667   85.5556  82.4074
DeepCovNet  79.0741   78.5185  79.2593
##################################################################
# hyperparameters
EPOCH = 1000
batch_size = 64
LEARNING_RATE = 1e-2

Result :
              ReLU    LeakyReLU  ELU
EEGNet      84.7222   85.6481  82.4074
DeepCovNet  78.1481   78.5185  79.2593
##################################################################
```







B. Effect of Regularization

Since over-fitting & under-fitting are the two major phenomena of the model, and through more epoch training, I think the possibility of under-fitting is relatively low, so I want to confirm whether there is an over-fitting problem in the model, so I compare the training loss and testing loss. The curve is drawn and observed. After observation, it was found that the loss curve of the preset hyperparameters, the testing loss continued to soar significantly in the second half of the training, but the

training loss continued to be at a low point, which means that the current model is more suitable for the training data set. , that is, the phenomenon of **over-fitting**.

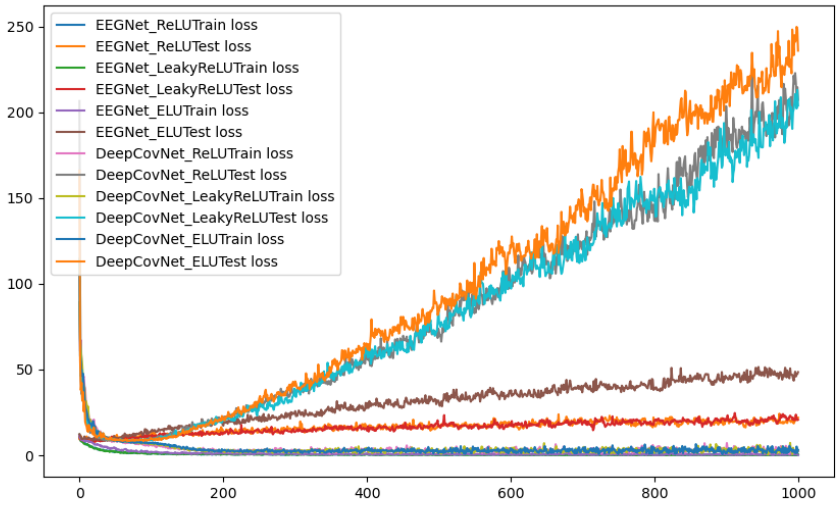
From the two comparison pictures below, it can be observed that the loss curve is obviously different. **Even if the loss with weight decay is trained in the later stage, it will not continue to increase.** It can be seen that weight decay does have a great effect on over-fitting. big improvement. The final model accuracy also successfully reached 87%.

Weight decay in pytorch (L2 Regularization)

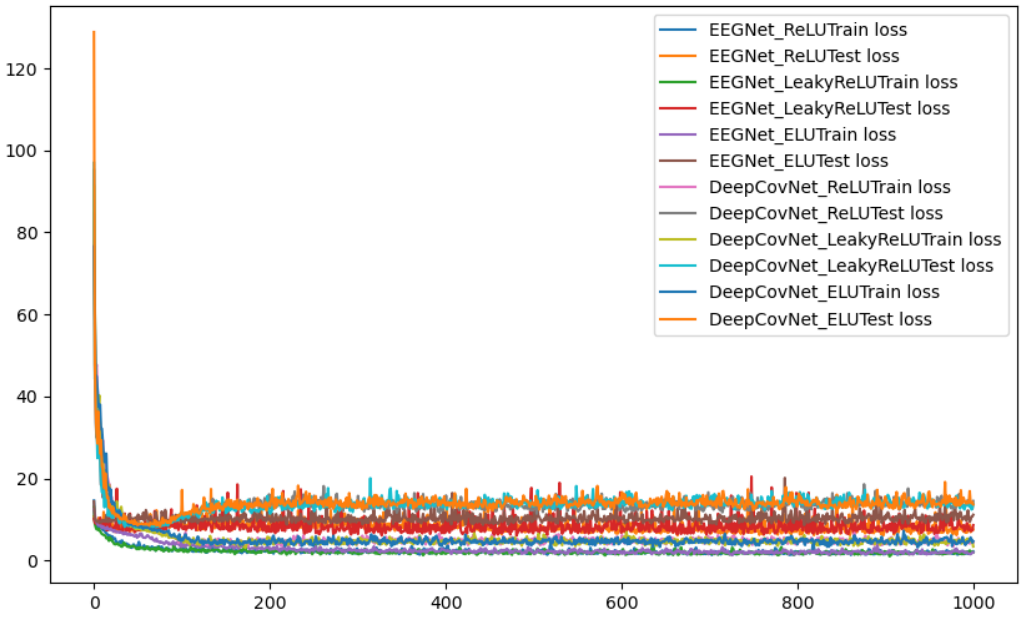
$$Obj = Loss + \frac{\lambda}{2} * \sum_{N}^{i} w_i^2$$

```
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=0.001)
              ReLU    LeakyReLU   ELU
EEGNet      87.5926   88.1481  81.2037
DeepCovNet  79.7222   79.2593  78.9815
```

w/o weight decay



w/ weight decay


EEGNet_ReLUTrain loss
EEGNet_ReLUTest loss
EEGNet_LeakyReLUTrain loss
EEGNet_LeakyReLUTest loss
EEGNet_ELUTrain loss
EEGNet_ELUTest loss
DeepCovNet_ReLUTrain loss
DeepCovNet_ReLUTest loss
DeepCovNet_LeakyReLUTrain loss
DeepCovNet_LeakyReLUTest loss
DeepCovNet_ELUTrain loss
DeepCovNet_ELUTest loss
120
100
80
60
40
20
0
0
200
400
600
800
1000