LAB2 : EEG Classification

0516069 翁英傑

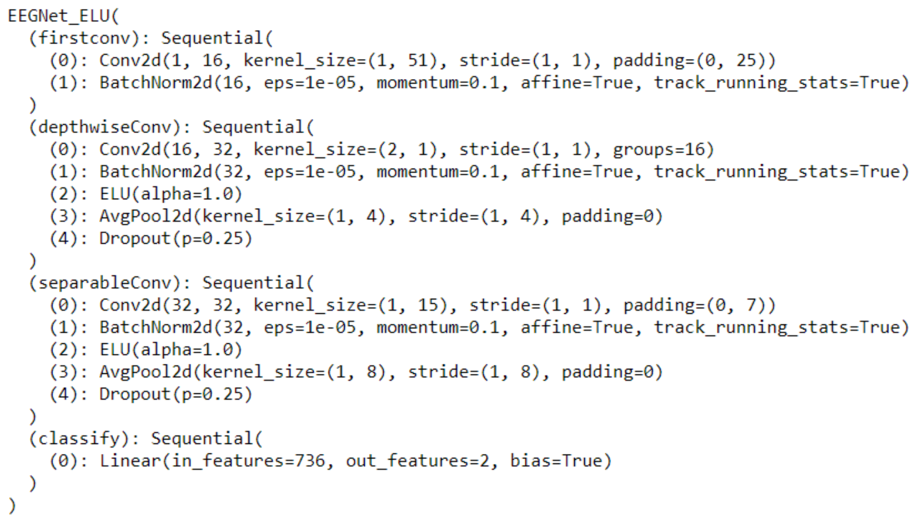
1. Introduction (20%)

Implement EEGNet and DeepConvNet from scratch with pytorch torch.nn, where both network are used to classify a binary classification from BCI competition dataset.

2. Experiment set up (30%)

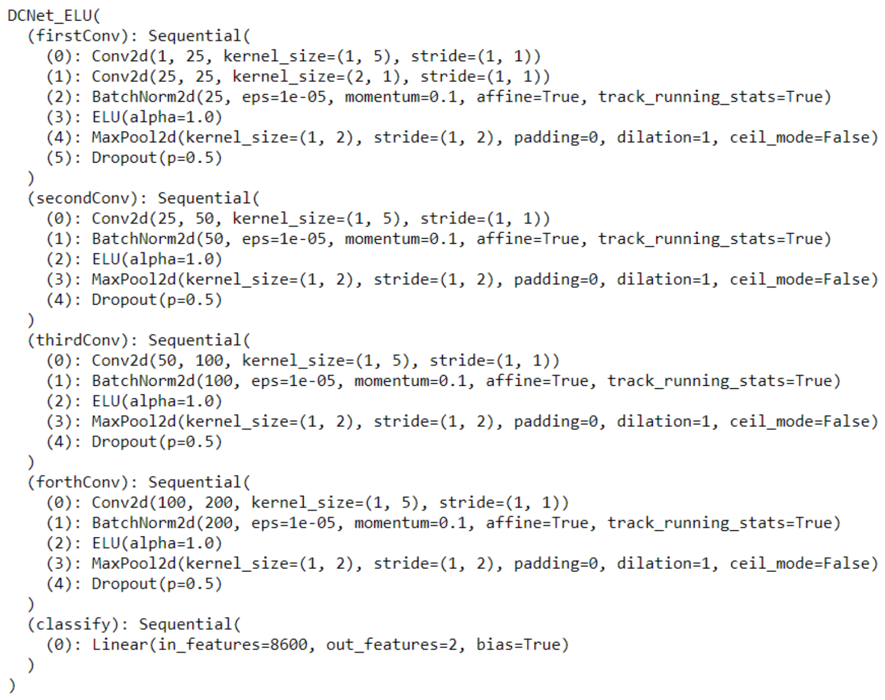
A. With both Network structure shown with the output of Net :

EEGNet (where activation function here is ELU)



```
EEGNet_ELU(
  (firstconv): Sequential(
    (0): Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25))
    (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)
    (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))
    (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 (where activation function here is ELU)



```
DCNet_ELU(
  (firstConv): 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)
  )
  (secondConv): 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): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5)
  )
  (thirdConv): 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): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5)
  )
  (forthConv): 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): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5)
  )
  (classify): Sequential(
    (0): Linear(in_features=8600, out_features=2, bias=True)
  )
)
```

B. Activation Function :

(1) ELU :

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

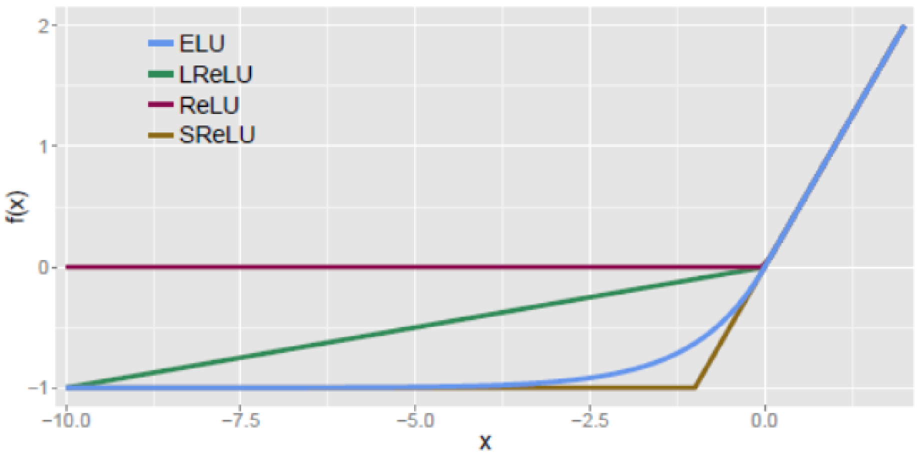
(2) ReLU :

$$f(x) = \max(0, x)$$

(3) LeakyReLU :

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

The comparison of the three activation function :

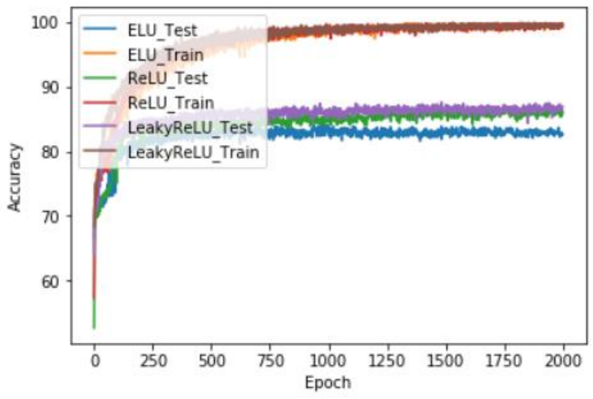


We can see the difference is the behavior when x < 0, which gives different value while calculating gradient in back propagation. This causes converge behavior to be difference. Each takes advantages in different dataset or model.

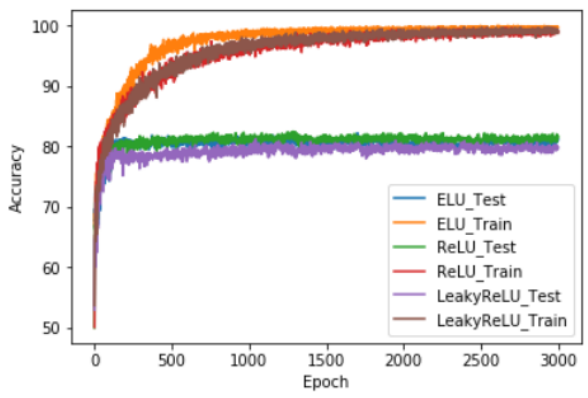
3. Experimental results (30%)

A. Accuracy graph :

(1) EEGNet :



(2) DeepConvNet :



Highest Accuracy for both Network with three activation functions :

	ELU	ReLU	LeakyReLU
EEGNet	84.17%	87.04%	**87.59%**
DeepConvNet	82.22%	**82.5%**	81.3%

B. Hyper Parameters :

Batch size = 64

Learning rate = 0.01

Epochs = 2000 (EEGNet) / 3000 (DeepConvNet)

Optimizer = SGD (lr=0.01, momentum=0.5, nesterov=True)

Loss function = Cross Entropy Loss

4. Discussion (20%)

While DeepConvNet is more deeper than EEGNet, it take more epochs for the loss to converge when training with same hyper parameters, which is pretty common in training a neural network. Also, I found that even with learning set to 0.01, it’s still too large for loss to converge while set learning rate smaller makes it longer to converge. So I took in the Nesterov momentum method to the solution.

In this LAB, training accuracy and testing accuracy does not converge to the same rate, which might be cause by lack of data. Lack of data causes the training set and the testing set might not form the same distribution. Common solution is to take the advantage of Law of Large Number, where the observed average should eventually converge to true mean.