Lab2 EEG Classification

311554046 林愉修

1. Introduction

利用 Pytorch implement EEGNet 和 DeepConvNet來做 Binary Classification，資料為 BCI Competition III - Dataset IIIb，資料的shape為(2, 750)。

2. Experiment set up

A. The details 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): ReLU()
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25, inplace=False)
  )
  (seperableConv): 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): ReLU()
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25, inplace=False)
  )
  (classify): Linear(in_features=736, out_features=2, bias=True)
)
```

  此 model 使用了 Depthwise seperable Convolution，目的是希望在不影響輸出結構的情況下去減少運算量，EEGnet 由1個普通 conv + 1個Depthwise conv + 1個 Separable conv 組成，其中，Separable Convolution 由一個 Depthwise Convolution 和一個 Pointwise Convolution 組成。

> Depthwise convolution:
> 和一般 convoution 不同，會建立與 input channel 數相同個數的filter，並且每個 filter 針對對應的 channel 分開去做 convolution。

> Pointwise convolution:
> 先對每個輸出 channel 建立一個大小為 1×1×M 的 filter 後 (M 為輸入層的 channel 數)，將輸入層的所有點進行 convolution 運算。假如輸出層有 N 個 channel，則會建立 N 個 1×1×M 的 filter。

- DeepConvNet

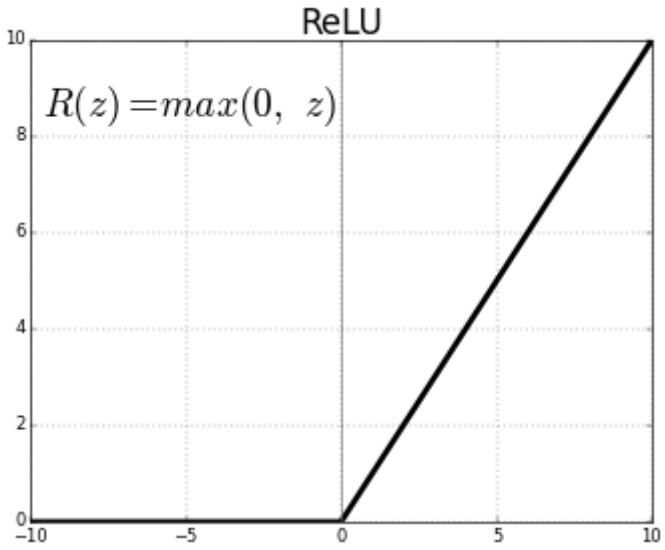
```
DeepConvNet(
  (conv0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1))
  (conv1): Sequential(
    (0): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (1): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (conv2): 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): 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)
  )
  (conv3): 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)
  )
  (conv4): 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)
  )
  (dense): Linear(in_features=8600, out_features=2, bias=True)
)
```

  傳統的CNN架構，經過多層(Conv, BatchNorm, Activation, MaxPool, Dropout)，並且 Conv 及 MaxPool 沒有做 padding。

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

- ReLU

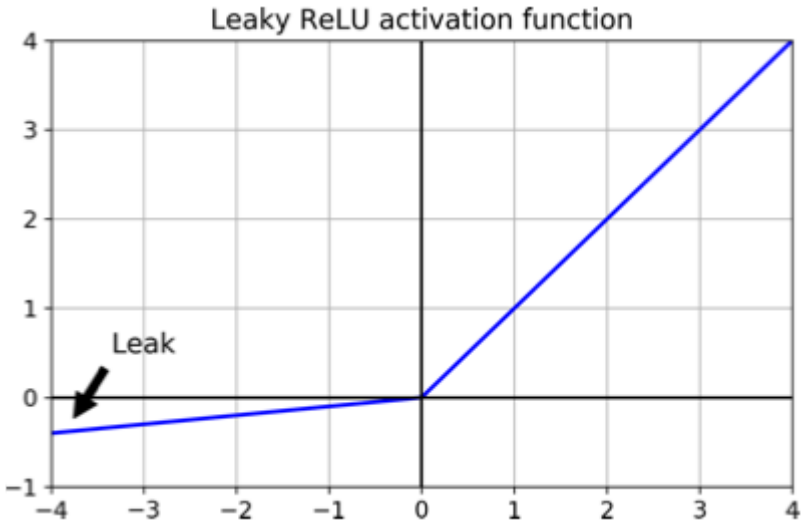
$$\mathrm{ReLU}(x) = (x)^+ = \max(0, x)$$



由於 Sigmoid 的 Gradient Vanishing 問題，因此衍生出Recified Linear Unit (ReLU)這個activation function，其優點除了減少Gradient Vanishing 問題外，運算也十分簡單，使得計算成本下降，不過它可能會導致權重更新不了，因 input < 0 時，gradient為0，這樣會沒辦法更新weight，稱為dying ReLu problem，也因此後來提出 Leaky ReLU 來改善這個問題。

- Leaky ReLU

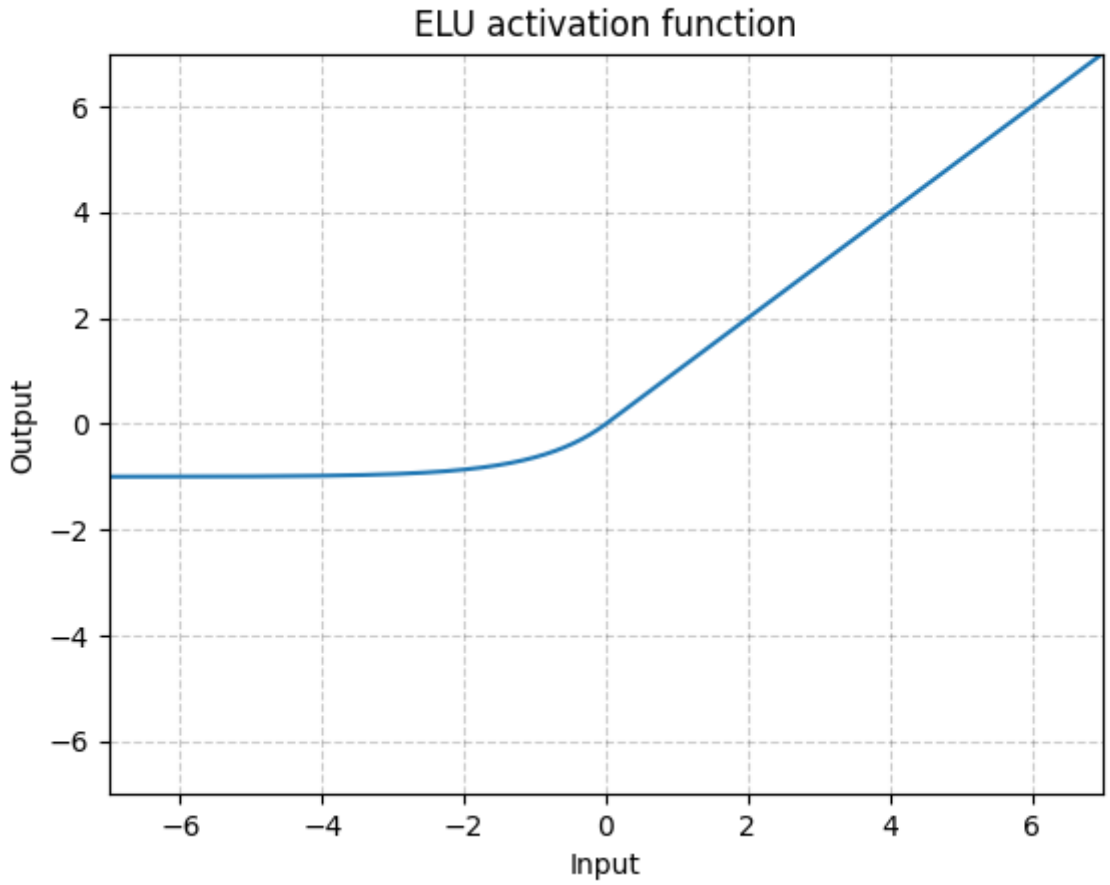
$$\text{LeakyRELU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$



具有 ReLU 的特點，但是在 input < 0 時，gradient不為0，而是一個很小的數，可以解決dying ReLU problem。

- ELU

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



同樣是為了解決dying ReLU problem，不過計算上較Leaky ReLU複雜。

3. Experiment results

A. The highest testing accuracy

- Screenshot with two models
  兩種 model使用同樣的 Hyperparameters，batch size 設為 540，optimizer 使用 Adam，並且 learning rate 設為 0.001，weight decay 設為 0.01，activation function 參數皆使用預設值，跑 300 epochs。

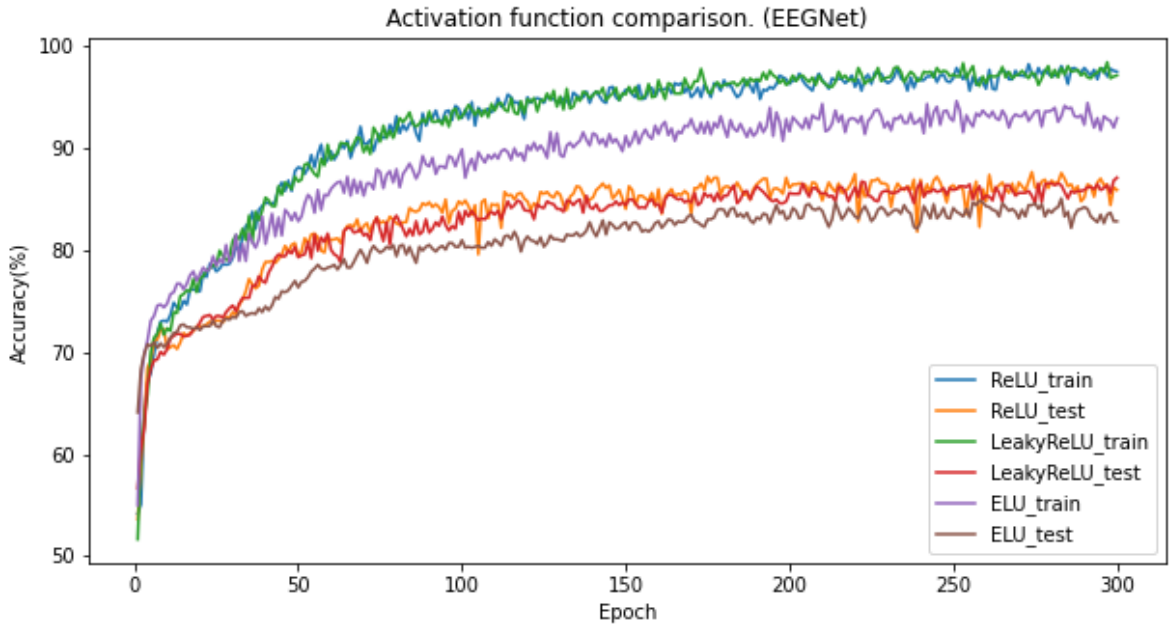
	ReLU_test	**LeakyReLU_test**	**ELU_test**
EEGNet	87.59%	85.27%	84.35%

	ReLU_test	LeakyReLU_test	ELU_test
DeepConvNet	81.94%	83.70%	80.00%

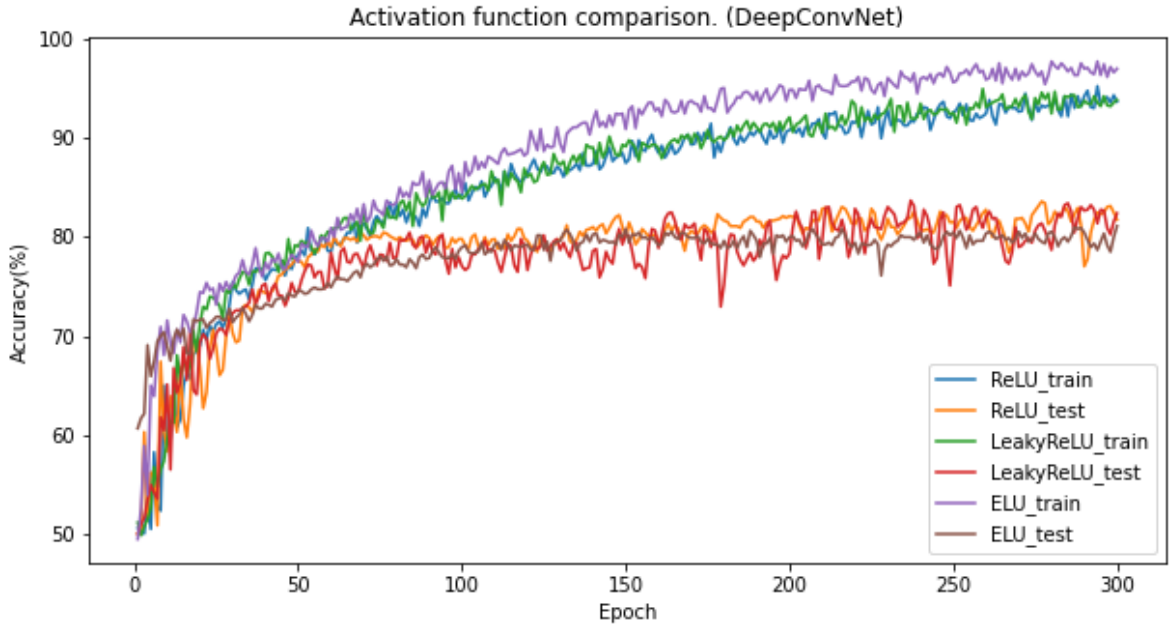
- Anything you want to present
  一開始 batch size 設為 64，跑出來最好的 test accuracy 大約只有 84% 左右，之後加大 batch size 為 540 後， test accuracy 也跟著上升至 87% 左右，不過再加大至 1080 (full batch) 時，test accuracy 反而會下降至 85% 左右，可能跟大的 batch size 容易陷入 sharp minima 有關。

B. Comparison figures

- EEGNet



- DeepConvNet



4. Discussion

A. Anything you want to shares

torch.nn.CrossEntropy() 要求 label 的 datatype 必須為 torch.long。

要將存好的 model weights load 進來時，出現錯誤，原因是我使用gpu 進行訓練，而 load 的裝置只有 cpu，因此須將 torch.load() 中的 map_location 參數設為 'cpu' 即可解決。