DLP – LAB04: EEGNet Classification

學號：310611008 姓名：張祐誠

Introduction:

在這次 LAB 中，要達到以下事項：

1. 建立簡單的 EEGNet 和 DeepConvNet 作為 EEG classification model.
2. 嘗試在模型中使用不同的 Activation function，在本次 Lab 中使用的分別是"ReLU", "ELU" 和 "LeakyReLU"
3. 利用上述模型對一組 BCI dataset 進行訓練。

EEGNet 架構示意圖：

是一種利用分離卷積的方式提取特徵的模型，文獻中指出在有限的數據下能有比 CNN 更好的性能和泛化性。

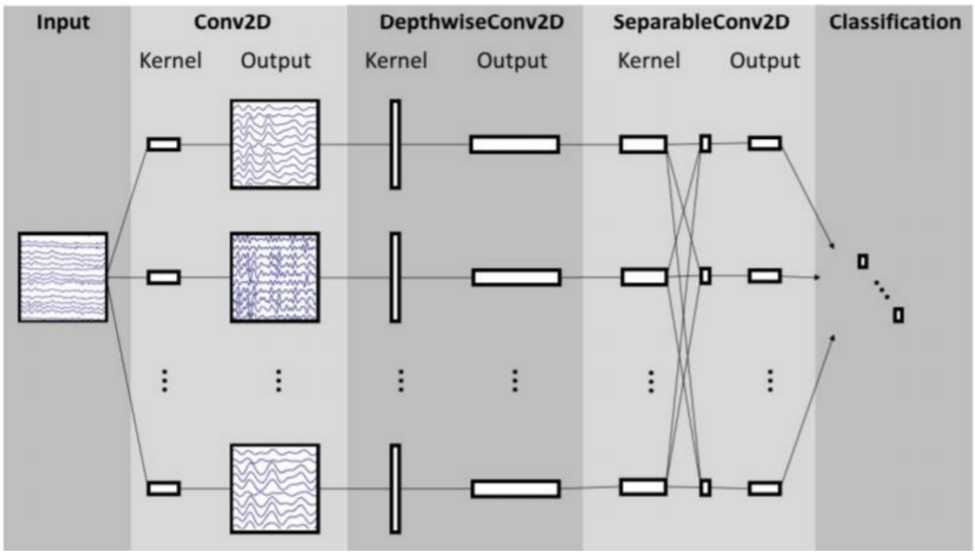


Figure credit to "Depthwise Separable Convolution"

BCI datasets:

是一種透過獲取頭腦的神經活動所獲得的電波訊號，其中示例於下圖：

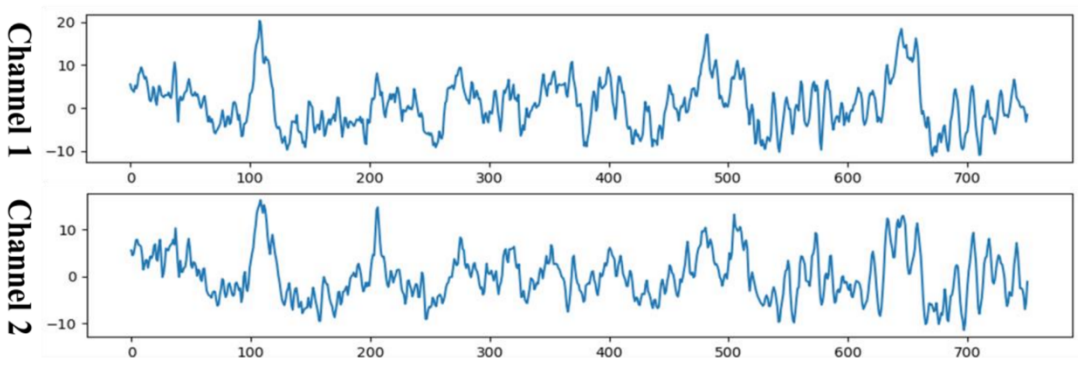


Figure credit to TA's slide

Experiment Setups:

A. Detail of model

如前一頁所提及，EEGNet 使用深度和可分離卷積來構建特定於 EEG 的模型。本次 LAB 使用的第一種 model 是 EEGNet，利用 pytorch 將所建立的模型 print 出來後的細節如下：

```
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)
  )
  (depthwiseCov): 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)
  )
)
```

其中 Activation function 會依據實驗更改，上圖是以 Leaky ReLU 作為 Activation function 為例。

第二個使用的是 DeepConvNet，所使用的架構表如下圖所示：

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 =
BatchNorm			2 * 200		epsilon = 1e-05, momentu
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Flatten					
Dense	N			softmax	max norm = 0.5

Table credit to TA's slide

其中 C = 2, T = 750, N = 2

利用 pytorch print 出來的模型細節如下：

```
DeepConvNet(
  (conv1): 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, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (conv2): 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)
  )
  (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)
  )
  (classify): Sequential(
    (0): Linear(in_features=1600, out_features=2, bias=True)
    (1): Softmax(dim=None)
  )
)
```

同樣其中 Activation function 會依據實驗更改，上圖是以 Leaky ReLU 作為 Activation function 為例。

同 Introduction 中敘述，在本次 Lab 中使用了三種不同的 Activation function，分別是 “ReLU”, “LeakyReLU”, “ELU”, 會在接下來進行介紹。

Other hyper parameters:
Learning rate: 1e-2
Batch size: 128
Epoch: 500

B. Activation Function

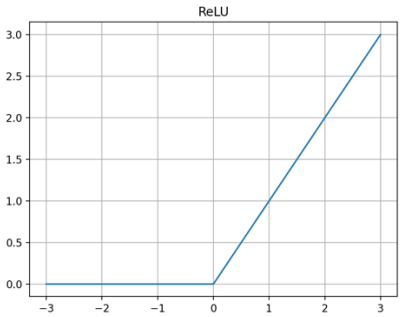
在深度模型中，Activation function 扮演的角色是在每一層之間形成非線性項，而這項特質相當重要，是深度神經網路架構的基礎，如果 Activation function 不存在的話，那深度網路就僅僅只是線性疊加而已。除此之外，有些 Activation function 如 softmax 在特定條件下可以擴大領先優勢。

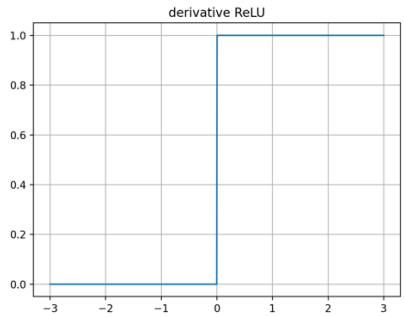
ReLU:

其中目前流行使用的是 ReLU function。

```python
def relu(self, x):
    x2 = np.copy(x)
    x2[x2<0] = 0
    return x2
```

```python
def de_relu(self, x):
    x2 = np.copy(x)
    x2[x2>=0] = 1
    x2[x2<0] = 0
    return x2
```



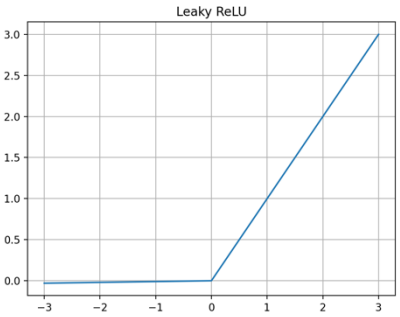


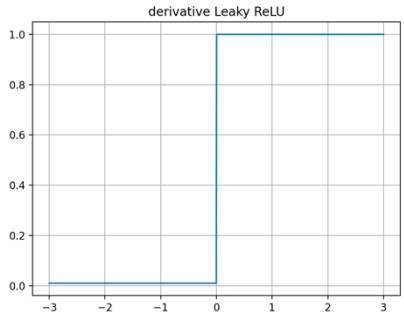
直覺上來看 ReLU 的計算上成本相對較低，但同時也保有了學習非線性函數的特性。但也可以看到一個明顯的問題，就是當 x<0 時，所有資料都會被消去，所以當更新權重時，x<0 的資料就無法進行學習；而這項特質也意味著 ReLU 只能在 hidden layer 中使用，無法使用在輸出上。因此有了下面兩個 Activation function 的產生。

Leaky ReLU:

```python
def leaky_relu(self,x):
    x2 = np.copy(x)
    x2[x2<0] = 0.01*x2[x2<0]
    return x2
```

```python
def de_leaky_relu(self,x):
    x2 = np.copy(x)
    x2[x2>=0] = 1
    x2[x2<0] = 0.01
    return x2
```



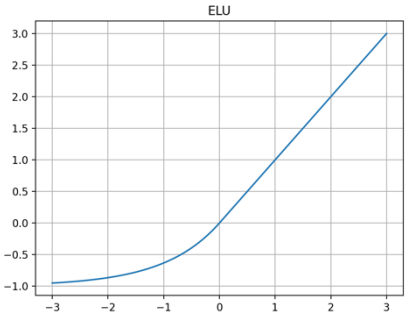


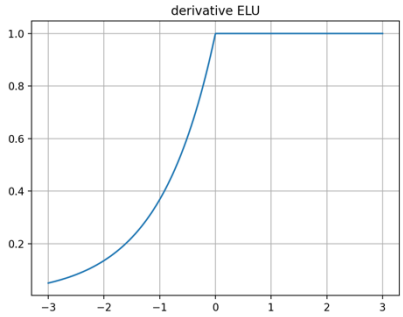
在這邊對 x<0 的部分允許了一個微小的梯度存在，在本次 lab 中使用 0.01 的斜率，嘗試解決前面 ReLU 的缺陷。

ELU:

```python
def elu(self, x, alpha=1.0):
    x2 = np.copy(x)
    x2[x2<0] = alpha*(np.exp(x2[x2<0]) - 1)
    return x2
```

```python
def de_elu(self, x, alpha=1.0):
    x2 = np.copy(x)
    x2[x2>=0] = 1
    x2[x2<0] = alpha*np.exp(x2[x2<0])
    return x2
```





ELU 比起 ReLU 在 x<0 有明顯的函數對應。在權重更新時也能有好的輸出，但因為有 exponential 項，所以在使用時要注意參數設定以避免飽和問題。

Experiment Results:

A. The highest testing accuracy:

Screen shot of with two models

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)
  )
  (depthwiseCov): 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)
  )
)
```

DeepConvNet:

```
DeepConvNet(
  (conv1): 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, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (conv2): 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)
  )
  (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)
  )
  (classify): Sequential(
    (0): Linear(in_features=1600, out_features=2, bias=True)
    (1): Softmax(dim=None)
  )
)
```

Highest testing accuracy(%):

	ReLU	Leaky ReLU	ELU
EEGNet	85.09	**87.41**	83.24
DeepConvNet	81.39	77.96	**81.76**

```
max of CNNnet, ELU: 81.76, ReLU: 81.39, LeakyReLU: 77.96
max of EGGnet, ELU: 83.24, ReLU: 85.09, LeakyReLU: 87.41
```

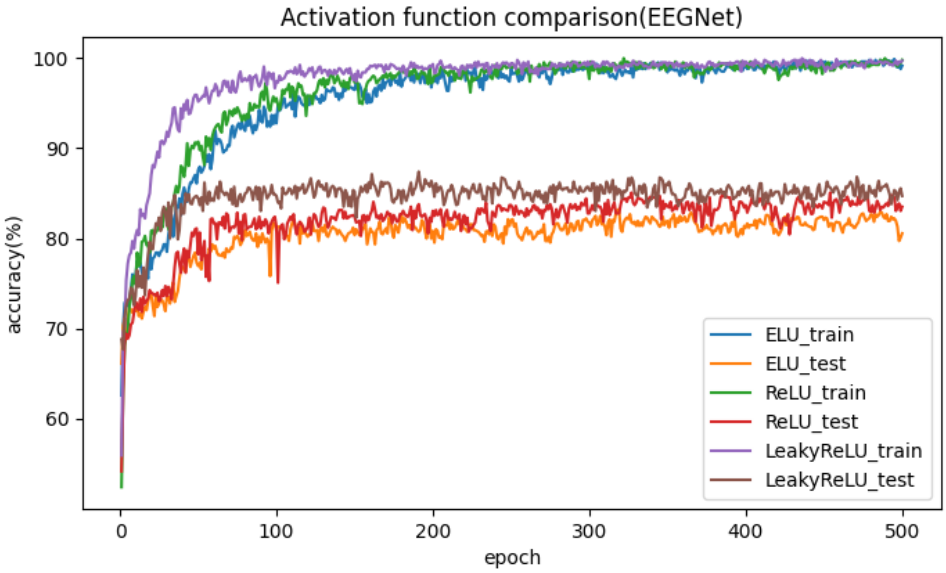
Extra thing to present:

在一開始我嘗試減少 learning rate, 期望得到更好的 accuracy，但除了掉到 local minimum 的狀況，所需要使用的 epoch 不僅越來越多，而且對模型的正確率的提升並沒有太好的提升；所以我在一開始的 learning rate 設為 0.01，然後當 testing data 的正確率到一個門檻後再降低 learning rate。

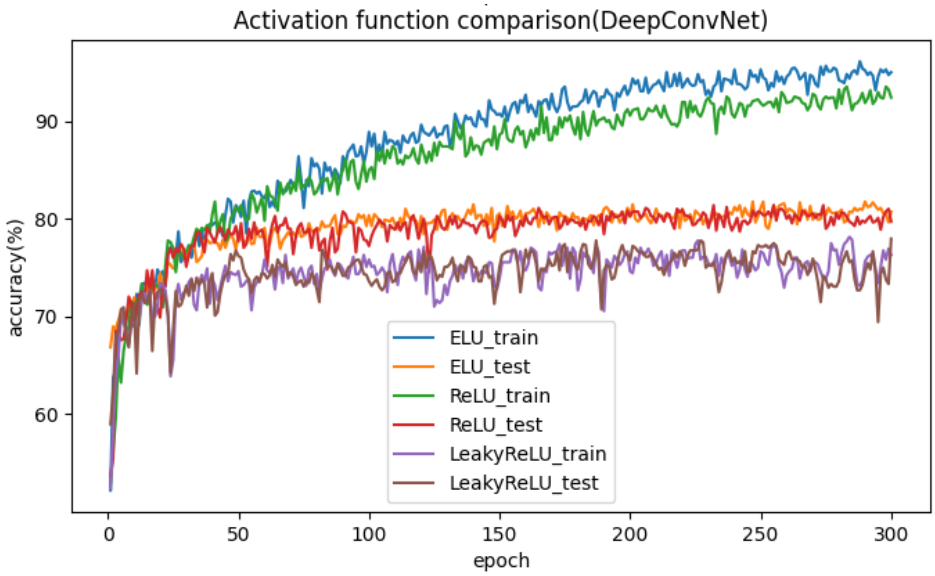
```python
if eval("{:.2f}".format(self.test_acc)) >=85:
    self.lr_rate = 1e-5

elif eval("{:.2f}".format(self.test_acc)) >=83:
    self.lr_rate = 1e-4

elif eval("{:.2f}".format(self.test_acc)) >=78:
    self.lr_rate = 1e-3
```

B. Comparison Figure:

EEGNet:



DeepConvNet:



可以看到在這次 LAB 中，不論在收斂速度或是模型準確率來比較，EEGNet 都比 DeepConvNet 來得好。

不過蠻值得探討的是，Leaky ReLU 再 EEGNet 裡表現得最出色，但在 DeepConvNet 裡面卻是最差的。

Discussion:

1. 我在 training 中有寫把 weight 存取與讀取的 function，不過根據 pytorch 官方的說明，只有存 dictionary 型態的 weight 而不是整個 model（因為存整個模型容易造成模型損失）。但如果之後需要比較多的 epoch 的 training 的話，就可以設定 check point，繼承之前的 weight 繼續。

```
def saveWeight(self):
    FILE = 'weight.pt'
    torch.save(self.best_model, FILE)

def loadWeight(self, file):
    self.model.load_state_dict(torch.load(file))
```

2. 雖然對於 BCI 這方面的 data 不熟悉，但根據上面的 accuracy-epoch 的圖，training accuracy 的準確率都來到 99%以上，懷疑有沒有 over fitting 的問題；另外在 testing accuracy 每次的準確度最後收斂的數值都有一點點落差，所以如果有時間的話會考慮進行 cross validation 來評估模型準確度。

3. 另外在 batch size 上，我發現在這次 LAB 中 batch size = 128 的表現比 64 (max at 85%) 和 256(max at 83%)都好，因此我使用 batch size = 128 作為我的 hyper parameters.。