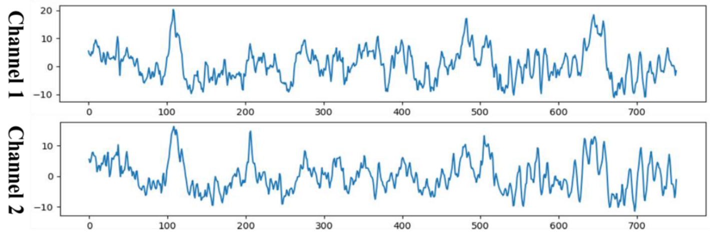
EEG classification

1. Introduction

在這個實驗中，需要實作EEG的分類模型，分別是EEGNet和 DeepConvNet，並嘗試三種不同的 activation function（ReLU, Leaky ReLU, ELU）。

(使用資料集:BCI Competition III - IIIb) 目標是將腦波訊號分類成兩種類別(左手、右手)。

Reference: http://www.bbci.de/competition/iii/desc_IIIb.pdf

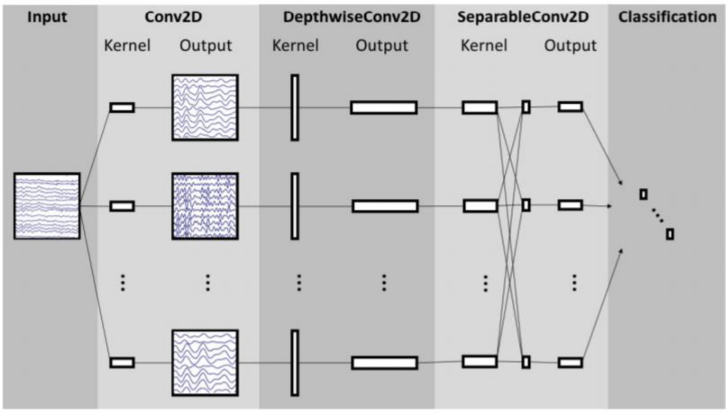


資料集中有兩個 channel，各有 750 個資料點，標記為左,右手兩種類別。

兩種**模型的比較:**

◆DeepConvNet:採用基本CNN架構

◆EEG:是為專門一般的腦電圖識別任務而設計的通用且緊湊的捲積神經網絡，設計思路則是借鑒了MobileNet。在訓練數據有限的情況下，EEGNet具有更強的泛化能力和更高的性能，用少量的訓練資料就可得到不錯的結果。



上圖是EEGnet的整體結構圖，只有三個卷積模塊，重點是depthwise conv (逐通道的捲積層操作)和separable conv這兩個卷積模塊。

其中separable conv由一個Depthwise Convolution(逐通道的捲積層操作)和一個Pointwise Convolution(逐點的捲積層操作)組成。

2. Experiment set up

A. The detail 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): 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), bias=False)
    (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)
  )
)
```

程式碼:

```
class EEG(nn.Module):
    def __init__(self, act_func):
        super(EEG, self).__init__()
        self.activationDict = {
            'ReLU': nn.ReLU(),
            'LeakyReLU': nn.LeakyReLU(),
            'ELU': nn.ELU(),
        }
        self.firstConv = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=16, kernel_size=(1,51), stride=(1,1),padding=(0,25), bias=False), #input=1x2x750 output=16x2x750
            nn.BatchNorm2d(16, eps=1e-5, momentum=0.1, affine=True, track_running_stats=True),
        )
        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=(2,1), stride=(1,1), groups=16, bias=False), #input=16x2x750 output=32x1x750
            nn.BatchNorm2d(32, eps=1e-5, momentum=0.1, affine=True, track_running_stats=True),
            self.activationDict[act_func],
            nn.AvgPool2d(kernel_size=(1,4), stride=(1,4), padding=0), #input=32x1x750 output=32x1x187
            nn.Dropout(p=0.25),
        )
        self.separableConv = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=(1,15),stride=(1,1),padding=(0,7), bias=False), #input=32x1x187 output=32x1x187
```

```
            nn.BatchNorm2d(32, eps=1e-5, momentum=0.1, affine=True, track_running_stats=True),
            self.activationDict[act_func],
            nn.AvgPool2d(kernel_size=(1,8), stride=(1,8), padding=0), #input=32x1x187 output=32x1x23
            nn.Dropout(p=0.25)
        )
        self.classifyConv = nn.Sequential(
            nn.Flatten(), #input=32x1x23 output=736
            nn.Linear(in_features=736,out_features=2,bias=True)
        )
    def forward(self, x):
        x = self.firstConv(x)
        x = self.depthwiseConv(x)
        x = self.separableConv(x)
        x = self.classifyConv(x)

        return x
```

◆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

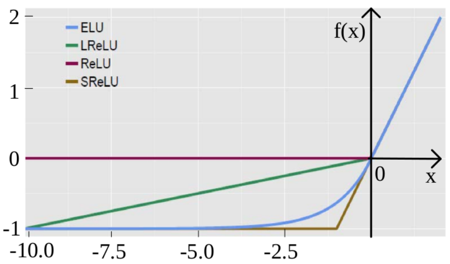
程式碼:

```
class DeepConvNet(nn.Module):
    def __init__(self, act_func):
        super(DeepConvNet, self).__init__()
        self.activationDict = {
            'ReLU': nn.ReLU(),
            'LeakyReLU': nn.LeakyReLU(),
            'ELU': nn.ELU(),
        }

        self.doubleConv = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=25, kernel_size=(1,5)), #input=1x2x750 output=25x2x746
            nn.Conv2d(in_channels=25, out_channels=25, kernel_size=(2,1)), #input=25x2x746 output=25x1x746
            nn.BatchNorm2d(25, eps=1e-5, momentum=0.1),
            self.activationDict[act_func],
            nn.MaxPool2d(kernel_size=(1,2)), #input=25x1x746 output=25x1x373
            nn.Dropout(p=0.5)
        )
        self.secondConv = nn.Sequential(
            nn.Conv2d(in_channels=25, out_channels=50, kernel_size=(1,5)), #input=25x1x378 output=50x1x369
            nn.BatchNorm2d(50, eps=1e-5, momentum=0.1),
            self.activationDict[act_func],
            nn.MaxPool2d(kernel_size=(1,2)), #input=50x1x369 output=50x1x184
            nn.Dropout(p=0.5)
        )
        self.thirdConv = nn.Sequential(
            nn.Conv2d(in_channels=50, out_channels=100, kernel_size=(1,5)), #input=50x1x184 output=100x1x180
            nn.BatchNorm2d(100, eps=1e-5, momentum=0.1),
            self.activationDict[act_func],
            nn.MaxPool2d(kernel_size=(1,2)), #input=100x1x180 output=100x1x90
            nn.Dropout(p=0.5)
        )
        self.fourthConv = nn.Sequential(
            nn.Conv2d(in_channels=100, out_channels=200, kernel_size=(1,5)), #input=100x1x90 output=200x1x86
            nn.BatchNorm2d(200, eps=1e-5, momentum=0.1),
            self.activationDict[act_func],
            nn.MaxPool2d(kernel_size=(1,2)), #input=200x1x86 output=200x1x43
            nn.Dropout(p=0.5)
        )
        self.flatten = nn.Flatten()
        self.dense = nn.Sequential(
            nn.Linear(in_features=8600,out_features=2)
        )
    def forward(self, x):
        x = self.doubleConv(x)
        x = self.secondConv(x)
        x = self.thirdConv(x)
        x = self.fourthConv(x)
```

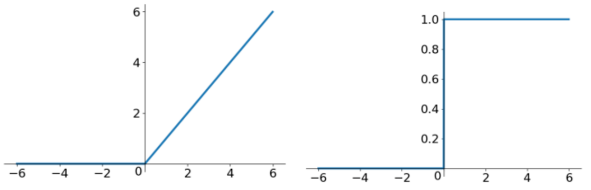
```
        x = self.flatten(x)
        x = self.dense(x)
        return x
```

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



◆ReLU

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

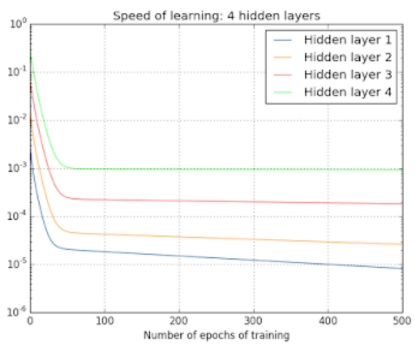


ReLU　　　　ReLU導數

若輸入為正數，則輸出該值大小，若值為負數，則輸出為0。ReLU函數並不是全區間皆可微分，但是不可微分的部分可以使用Sub-gradient進行取代。其有以下特點:

(1)解決梯度消失問題

ReLU的分段線性性質能有效地克服梯度消失之問題。



(2)計算量大幅降低

無需使用任何指數運算，只需要判斷輸入值是否大於0，來進行輸出。

(3)生物事實(細胞激活現象)

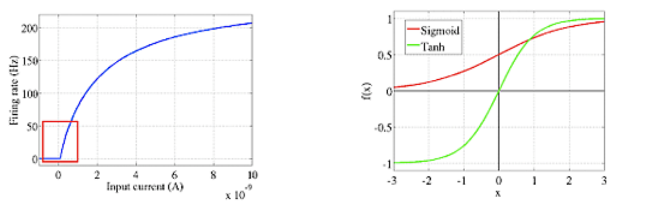


Figure 1: *Left:* **Common neural activation function motivated by biological data.** *Right:* **Commonly used activation functions** in neural networks literature: logistic sigmoid and hyperbolic tangent (*tanh*).

ReLU函數成功模擬了細胞對於刺激的反應現象:當對細胞的刺激未達到一定強度時，神經元不會進行訊息傳遞，但當超越啟動之強度時，會引起神經衝動，而進行訊息之傳遞。

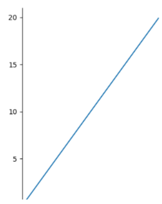
(4)類神經網路的稀疏性(奧卡姆剃刀原則)

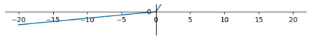
ReLU激勵函數會使負數部分的神經元輸出為0，可以讓網路變得更加多樣性，如同Dropout的概念，可以緩解過擬合(Over fitting)之問題，但會衍生Dead ReLU的問題(當某個神經元輸出為0後，就難以再度輸出)。

容易導致dead ReLU發生的原因:

1. 初始化權重設定為不能被激活的數值。
2. 學習率設置過大，在剛開始進行誤差反向傳遞時，容易修正權重值過大，導致權重梯度為0，神經元即再也無法被激活。

◆Leaky ReLU

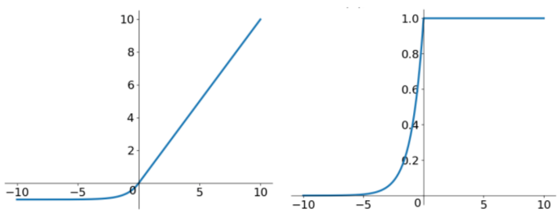




ReLU是將所有的負值都設為零，而為了解決Dead ReLU Problem，Leaky ReLU給所有負值賦予一個非零斜率，如此一來，即能防止值為負號時永遠無法被激活之問題。理論上來說，Leaky ReLU擁有ReLU的所有優點，也成功避免Dead ReLU Problem的問題產生，但是於實際使用上，還沒有辦法完全證明Leaky ReLU永遠優於ReLU。

◆ELU

$$f(x)=\begin{cases} x, & \text{if } x>0 \\ \alpha\left(e^{x}-1\right), & \text{otherwise} \end{cases}$$



$f(x)$　　　$f(x)$導數

ELU函數也是為了解決Dead ReLU問題而被提出，但需要計算指數,計算量較大。其平均激活均值趨近爲 0，並負飽和區的存在使得 ELU 比 Leaky ReLU 更加健壯，抗噪聲能力更強。理論上來說，ELU 擁有ReLU的所有優點，也成功避免Dead ReLU Problem的問題產生，但是於實際使用上，還沒有辦法完全證明Leaky ReLU永遠優於ReLU。

3. Experimental results

A. The highest testing accuracy

◆Screenshot with two models

Learning rate = 1e-3 (train/test) (單位：%)

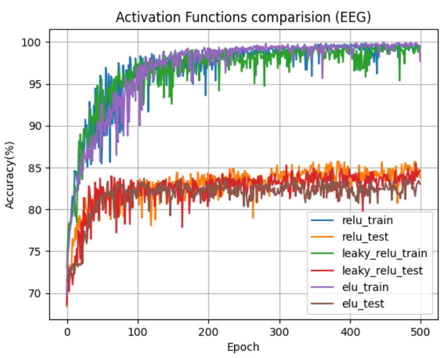
model \ activation	ReLU	LeakyReLU	ELU
EEGNet	99.4/83.79	99.35/84.62	97.68/83.05
DeepConvNet	96.01/76.85	96.66/77.07	99.90/77.59

```
EEG_ReLU_train
epoch=  250  loss=  3.402955761110341e-05  correct=  0.9925925925925926
epoch=  500  loss=  1.7752998543006404e-05  correct=  0.9944444444444445
EEG_ReLU_test
epoch=  250  loss=  0.000572562438470346  correct=  0.8416666666666667
epoch=  500  loss=  0.0007224686719753124  correct=  0.8379629629629629
EEG_LeakyReLU_train
epoch=  250  loss=  6.823038889302148e-05  correct=  0.9694444444444444
epoch=  500  loss=  1.899072796934181e-05  correct=  0.9935185185185185
EEG_LeakyReLU_test
epoch=  250  loss=  0.0005801748898294237  correct=  0.837037037037037
epoch=  500  loss=  0.0007137201450489186  correct=  0.8462962962962963
EEG_ELU_train
epoch=  250  loss=  3.399812030019583e-05  correct=  0.9925925925925926
epoch=  500  loss=  5.2501130159254425e-05  correct=  0.9768518518518519
EEG_ELU_test
epoch=  250  loss=  0.0005791956075915584  correct=  0.8175925925925925
epoch=  500  loss=  0.0007926864756478204  correct=  0.8305555555555556
```

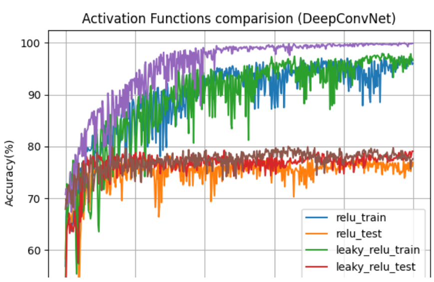
```
DeepConvNet_ReLU_train
epoch=  250  loss=  0.00014599338173866273  correct=  0.9435185185185185
epoch=  500  loss=  0.00014489585602724993  correct=  0.9601851851851851
DeepConvNet_ReLU_test
epoch=  250  loss=  0.0007435650737197311  correct=  0.7462962962962963
epoch=  500  loss=  0.0008095357705045629  correct=  0.7685185185185185
DeepConvNet_LeakyReLU_train
epoch=  250  loss=  0.00016172417887934932  correct=  0.937962962962963
epoch=  500  loss=  0.00011148922559287813  correct=  0.9666666666666667
DeepConvNet_LeakyReLU_test
epoch=  250  loss=  0.0006718383895026313  correct=  0.7555555555555555
epoch=  500  loss=  0.000962156057357788  correct=  0.7907407407407407
DeepConvNet_ELU_train
epoch=  250  loss=  3.118925831384129e-05  correct=  0.9898148148148148
epoch=  500  loss=  3.40719621076628e-06  correct=  0.9990740740740741
DeepConvNet_ELU_test
epoch=  250  loss=  0.000963965720600552  correct=  0.7768518518518519
epoch=  500  loss=  0.0014684530319990935  correct=  0.7759259259259259
```

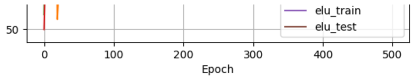
B. Comparison figures

◆ EEGNet



◆ DeepConvNet





4. Discussion

(1)發現調低學習率能訓練得更好。

註:本實驗使用的學習率為:1-e3，其結果在報告中可查訊。以下附上學習率1e-2與1-e3在test的準確率差異及習率為:1-e2的結果報告。

◆學習率1e-2與1-e3在test的準確率差異(1-e3準確率- 1-e2準確率)

model \ activation	ReLU	LeakyReLU	ELU
EEGNet	5	4.62	4.72
DeepConvNet	1.02	0.59	-0.55

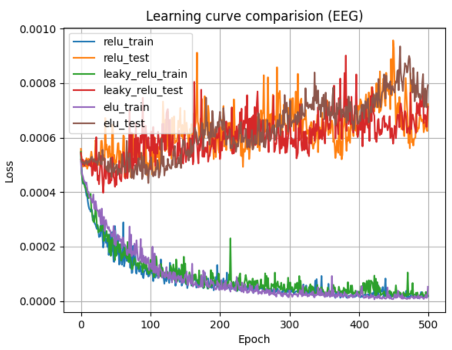
◆Learning rate = 1e-2 (train/test) (單位：%)

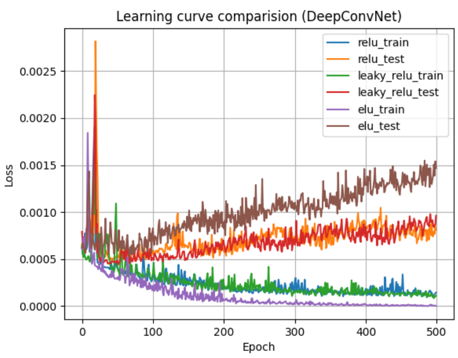
model \ activation	ReLU	LeakyReLU	ELU
EEGNet	98.24/78.79	98.98/80	98.05/78.33
DeepConvNet	95.37/75.83	95.55/76.48	99.62/78.14

```
EEG_ReLU_train
epoch=  250  loss=  9.842774382343999e-05  correct=  0.975
epoch=  500  loss=  5.96982737382253e-05  correct=  0.9824074074074074
EEG_ReLU_test
epoch=  250  loss=  0.00099451188687925  correct=  0.7657407407407407
epoch=  500  loss=  0.0011201995390432852  correct=  0.787962962962963
EEG_LeakyReLU_train
epoch=  250  loss=  4.254696131856353e-05  correct=  0.9879629629629629
epoch=  500  loss=  4.2715040897881544e-05  correct=  0.9898148148148148
EEG_LeakyReLU_test
epoch=  250  loss=  0.0010714112608521073  correct=  0.7888888888888889
epoch=  500  loss=  0.0013050772525646068  correct=  0.8
EEG_ELU_train
epoch=  250  loss=  4.1765164307974e-05  correct=  0.9888888888888889
epoch=  500  loss=  5.26627491193789e-05  correct=  0.9805555555555555
EEG_ELU_test
epoch=  250  loss=  0.001314703071558917  correct=  0.7898148148148149
epoch=  500  loss=  0.002448132082268044  correct=  0.7833333333333333
```

```
DeepConvNet_ReLU_train
epoch=  250  loss=  0.0001974168199080008  correct=  0.9314814814814815
epoch=  500  loss=  0.00018889683264273184  correct=  0.9537037037037037
DeepConvNet_ReLU_test
epoch=  250  loss=  0.0006119198821209095  correct=  0.7833333333333333
epoch=  500  loss=  0.0006306861837704976  correct=  0.7583333333333333
DeepConvNet_LeakyReLU_train
epoch=  250  loss=  0.00019607342503688953  correct=  0.9324074074074075
epoch=  500  loss=  0.00018065833935031184  correct=  0.9555555555555556
DeepConvNet_LeakyReLU_test
epoch=  250  loss=  0.0005242860979504055  correct=  0.7694444444444445
epoch=  500  loss=  0.000792608603283211  correct=  0.7648148148148148
DeepConvNet_ELU_train
epoch=  250  loss=  4.701902055078083e-05  correct=  0.9907407407407407
epoch=  500  loss=  2.0252031929515027e-05  correct=  0.9962962962962963
DeepConvNet_ELU_test
epoch=  250  loss=  0.001380135284529791 9 correct=  0.7657407407407407
epoch=  500  loss=  0.003982188966539171  correct=  0.7814814814814814
```

(2)發現不論在 EEGNet或是**DeepConvNet**，大概訓練 100 個 epoch 後，在 train data 正確率基本都可以達到九成，不過在 test data 上繼續訓練也沒有太多進步。 Loss 的學 習曲線中，推測為 overfit現象。





(3)weight decay的使用

weight decay（權值衰減）的使用其最終目的是防止過擬合。在損失函數中，weight decay是放在正則項（regularization）前面的一個係數，正則項一般指示模型的複雜度，所以weight decay的作用是調節模型複雜度對損失函數的影響，若weight decay很大，則復雜的模型損失函數的值也就大。

```
optimizer = optim.Adam(model.parameters(),Learning_Rate, weight_decay = 0.001)
```

在本次實驗中，加入weight decay，在DeepConvNet的表現上準確率皆提升2%-3%，但在EEG上就較沒有明顯的變化。