Deep Learning and Practice

#Lab03 EEG classification　309505002 鄭紹文

1. Introduction：

腦電波圖是記錄頭上某兩端點的電位差，緣由於人在清醒、壓力環境、昏迷等不同狀況時，腦電波的振動頻率會有不同變化，由於人類的大腦裡有許多神經細胞，細胞活動會發出電磁波。此次lab時做兩種不同的網路，EEGNet以及Deep convolution network搭配三種activation function：ELU、ReLU、Leaky-ReLu來分析已經經過preprocessing的BCI competition的資料*(fig.1)*。

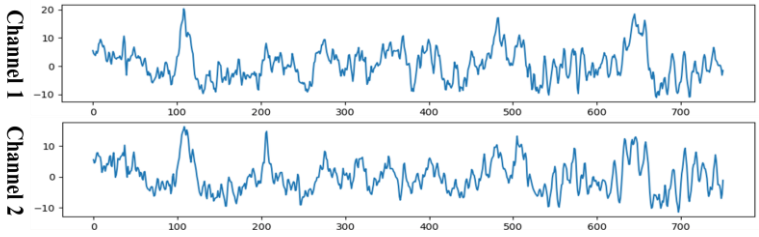
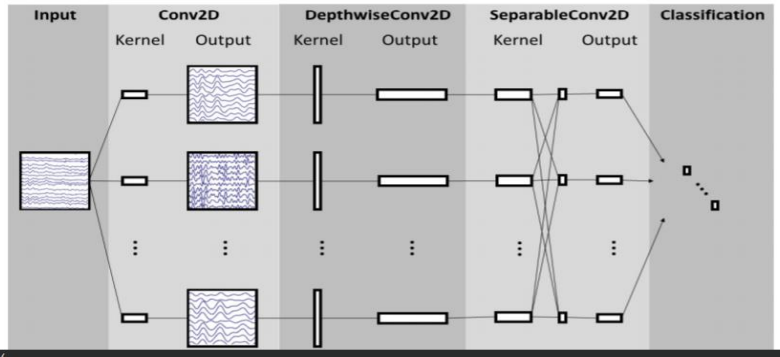


fig. 1

2. Experiment setups：

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)
  )
)
```

fig. 2

```
----------------------------------------------------------------
        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
         AvgPool2d-6          [-1, 32, 1, 187]               0
           Dropout-7          [-1, 32, 1, 187]               0
            Conv2d-8          [-1, 32, 1, 187]          15,360
       BatchNorm2d-9          [-1, 32, 1, 187]              64
             ReLU-10          [-1, 32, 1, 187]               0
        AvgPool2d-11           [-1, 32, 1, 23]               0
          Dropout-12           [-1, 32, 1, 23]               0
           Linear-13                  [-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): 1.16
Params size (MB): 0.07
Estimated Total Size (MB): 1.23
----------------------------------------------------------------
```

fig. 3

主要利用 Spec 上所附的架構，使用 PyTorch 建構 network，同時將靠近輸出的 dropout 設為 0.5 做優化調整。需要注意最後要有 flatten layer，將 output 將值傳入最後的 classification layer。

- **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
              ReLU-4          [-1, 25, 1, 746]               0
         MaxPool2d-5          [-1, 25, 1, 373]               0
           Dropout-6          [-1, 25, 1, 373]               0
            Conv2d-7          [-1, 50, 1, 369]           6,300
       BatchNorm2d-8          [-1, 50, 1, 369]             100
              ReLU-9          [-1, 50, 1, 369]               0
        MaxPool2d-10          [-1, 50, 1, 184]               0
          Dropout-11          [-1, 50, 1, 184]               0
           Conv2d-12         [-1, 100, 1, 180]          25,100
      BatchNorm2d-13         [-1, 100, 1, 180]             200
             ReLU-14         [-1, 100, 1, 180]               0
        MaxPool2d-15          [-1, 100, 1, 90]               0
          Dropout-16          [-1, 100, 1, 90]               0
           Conv2d-17          [-1, 200, 1, 86]         100,200
      BatchNorm2d-18          [-1, 200, 1, 86]             400
             ReLU-19          [-1, 200, 1, 86]               0
        MaxPool2d-20          [-1, 200, 1, 43]               0
          Dropout-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
```

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

fig. 4

DeepConv EEGNet

```
class DeepConvNet(nn.Module):
    def __init__(self,activation):
        super(DeepConvNet, self).__init__()
        self.deep = [25,50,100,200]
        self.activation_func = activation


        self.layer_0 = nn.Sequential(
            nn.Conv2d(1, self.deep[0], kernel_size = (1, 5), stride = (1, 1), padding = (0, 0), bias = True),
            nn.Conv2d(self.deep[0], self.deep[0], kernel_size = (2, 1), stride = (1, 1), padding = (0, 0), bias = True),
            nn.BatchNorm2d(self.deep[0], eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True),
            self.activation_func(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(0.2))

        for i in range(1, 4):
            setattr(self, 'layer_'+str(i),nn.Sequential(
                nn.Conv2d(self.deep[i-1], self.deep[i], kernel_size = (1, 5), stride = (1, 1), padding = (0, 0), bias = True
                nn.BatchNorm2d(self.deep[i], eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True),
                self.activation_func(),
                nn.MaxPool2d(kernel_size=(1, 2)),
                nn.Dropout(0.5))
                )

        flatten_size =  self.deep[-1] * reduce(lambda x,_: round((x-4)/2), self.deep, 750)
        self.layer_4 = nn.Sequential(
            nn.Linear(in_features = flatten_size, out_features = 2, bias = True)
        )

    def forward(self, x):
        x = self.layer_0(x)
        x = self.layer_1(x)
        x = self.layer_2(x)
        x = self.layer_3(x)
        # flatten

        x = x.view(-1, self.layer_4[0].in_features)
        x = self.layer_4(x)
        return x
```

fig. 5

DeepConvNet同樣照著Spec建構network，並且在參數上把drop out做調整，靠近input端的drop out設成0.2。

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

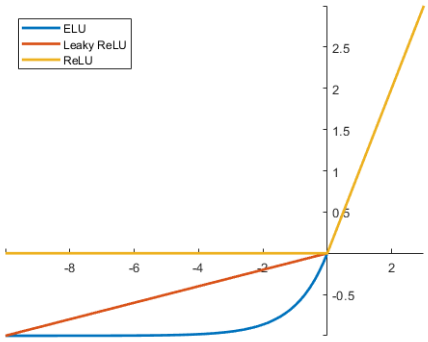


fig. 6

- **ReLU：**

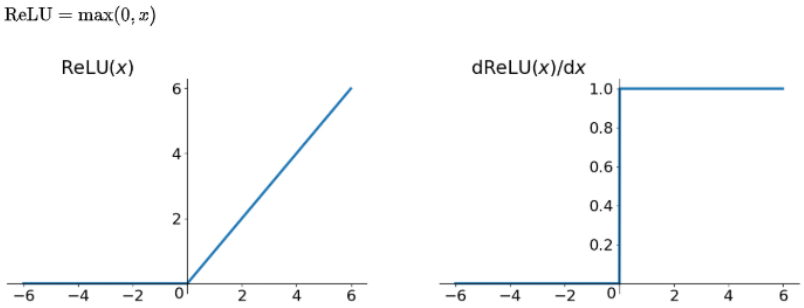


fig. 7

ReLU函數圖形如fig.7，若值為正數時，則輸出該值大小，反之若值為負數，則輸出為0。ReLU函數並非全區間皆可微分，但是不可微分的部分可使用Sub-gradient進行取代。

ReLU是近年來最頻繁被使用的activation function，原因在於其存在以下特點：解決gradient vanishing問題、計算速度相當快、收斂速度快等特性。

(1) 梯度消失問題 (vanishing gradient problem)：

對於使用誤差反向傳遞運算類的神經網絡來說，更新權重時，梯度計算的考量最為重要，使用Sigmoid以及tanh函數較容易發生梯度消失問題，當輸入值接近飽和區(sigmoid函數在小於-4和大於+4的時候)進行激發時，**一階微分值趨近於0**，就發生梯度消失的問題，使得誤差反向傳遞計算，無法有效地進行權重更新，而這個現象在神經網路層數加深時會更加明顯，而ReLU函數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：

ReLU函數相較於Sigmoid以及tan h來說，大幅下降計算量，因為在這裡我們不需要使用任何指數運算，**只需要判斷輸入值是否大於0，來進行輸出**。

- **Leaky ReLU：**

$$f(x) = \max(0.01x, x)$$

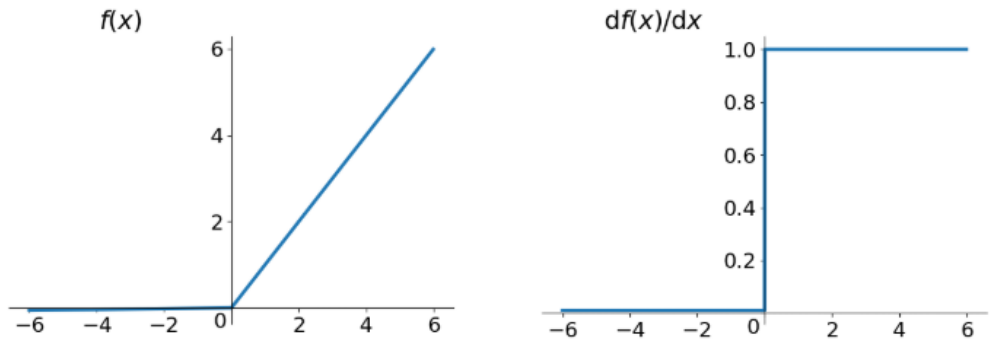


fig. 8

Leaky ReLU函數圖形如fig.8，為了解決Dead ReLU Problem *(ReLU在負數區域被kill的現象叫做dead relu。ReLU在訓練的時很脆弱，在x<0時，梯度為0。這個神經元及之後的神經元梯度會永遠為0，不再對任何資料有所影響，導致相應參數永遠不會被更新。通常兩個原因：(1). 參數初始化問題、(2)learning rate太高導致在訓練過程中參數更新太大。)*

Leaky ReLU將ReLU的前半段輸出設為0.01x，如此即能**防止值為負號時永遠無法被激活**之問題。理論上，Leaky ReLU擁有ReLU的所有優點，也成功避免Dead ReLU Problem的問題產生，但在實際使用上，並沒有辦法完全證明Leaky ReLU永遠優於ReLU。

- **ELU：**

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

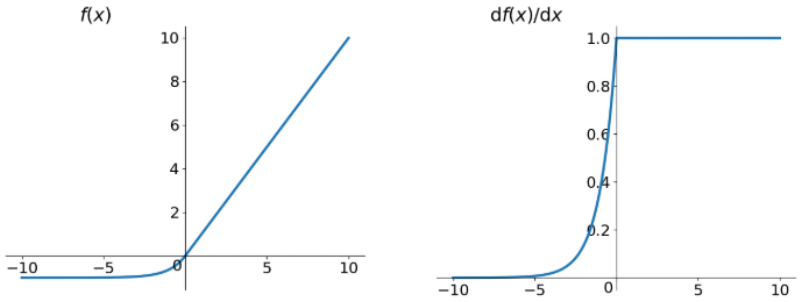


fig. 9

ELU函數圖形如fig.9，ELU也是為解決ReLU存在的問題而提出，同時也有ReLU的基本所有優點，以及(1)不會有Dead ReLU問題、(2)輸出的均值接近0，zero-centered。

缺點在於**計算量稍大**，類似於Leaky ReLU，理論上雖然好於ReLU，但在實際使用中目前並沒有好的證據ELU總是優於ReLU。

3. Experiment results：

A. The highest testing accuracy：

Plot Comparison Result

```
print("EEGNet v.s DeepConvNet")
data=[{"Net":"EEGNet", "ReLU":acc_EGG_ReLU*100, "Leaky ReLU":acc_EGG_LeakyReLU*100, "ELU":acc_EGG_ELU*100},
      {"Net":"DeepConvNet", "ReLU":acc_DeepConv_ReLU*100, "Leaky ReLU":acc_DeepConv_LeakyReLU*100, "ELU":acc_DeepConv_ELU*1
df=pd.DataFrame(data, columns=['Net', 'ReLU', 'Leaky ReLU', 'ELU'])
# print(df)
tb=Texttable()
tb.set_cols_align(['l', 'r', 'r', 'r'])
tb.set_cols_dtype(['t', 'f', 'f', 'f'])
tb.add_rows(df.values,header=False)

print(tb.draw())
```

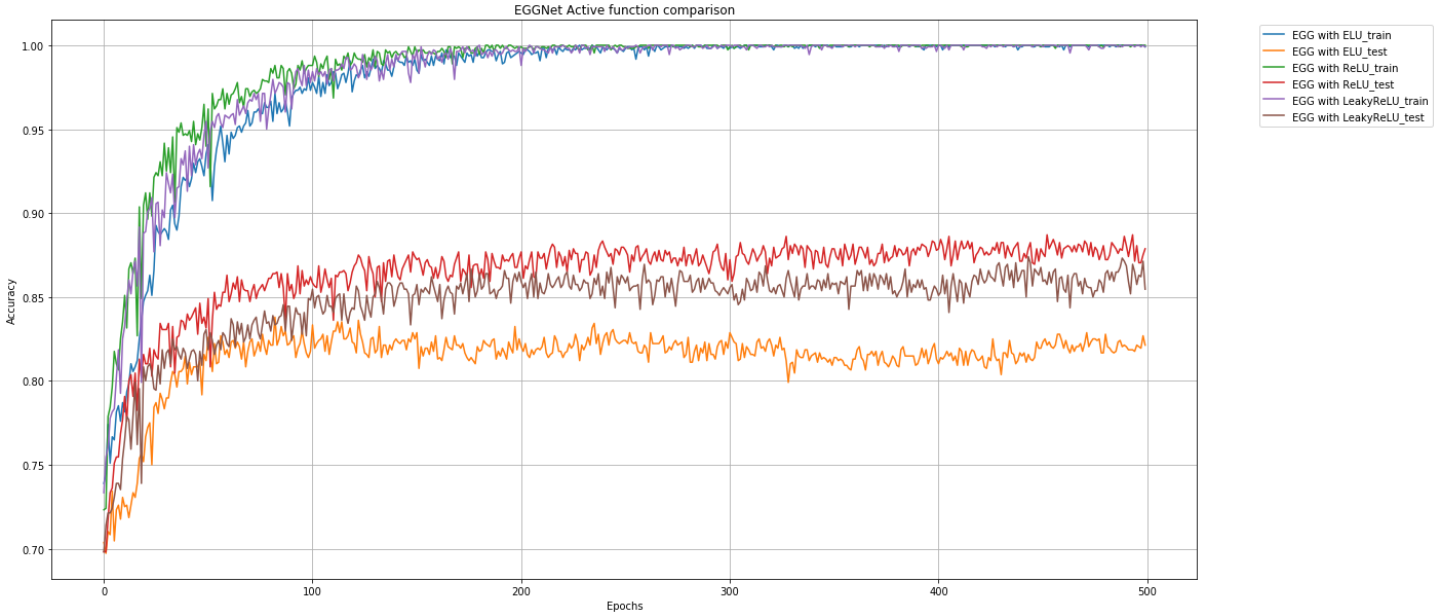
```
EEGNet v.s DeepConvNet
+-------------+--------+------------+--------+
|     Net     |  ReLU  | Leaky ReLU |  ELU   |
+=============+========+============+========+
| EEGNet      | 88.704 |   87.407   | 83.796 |
+-------------+--------+------------+--------+
| DeepConvNet | 87.685 |   87.130   | 83.889 |
+-------------+--------+------------+--------+
```

Training data有沒有經過shuffle後再進行training，會對結果有所影響(提升)，lab做完認為還是要對data做適當程度的shuffle。

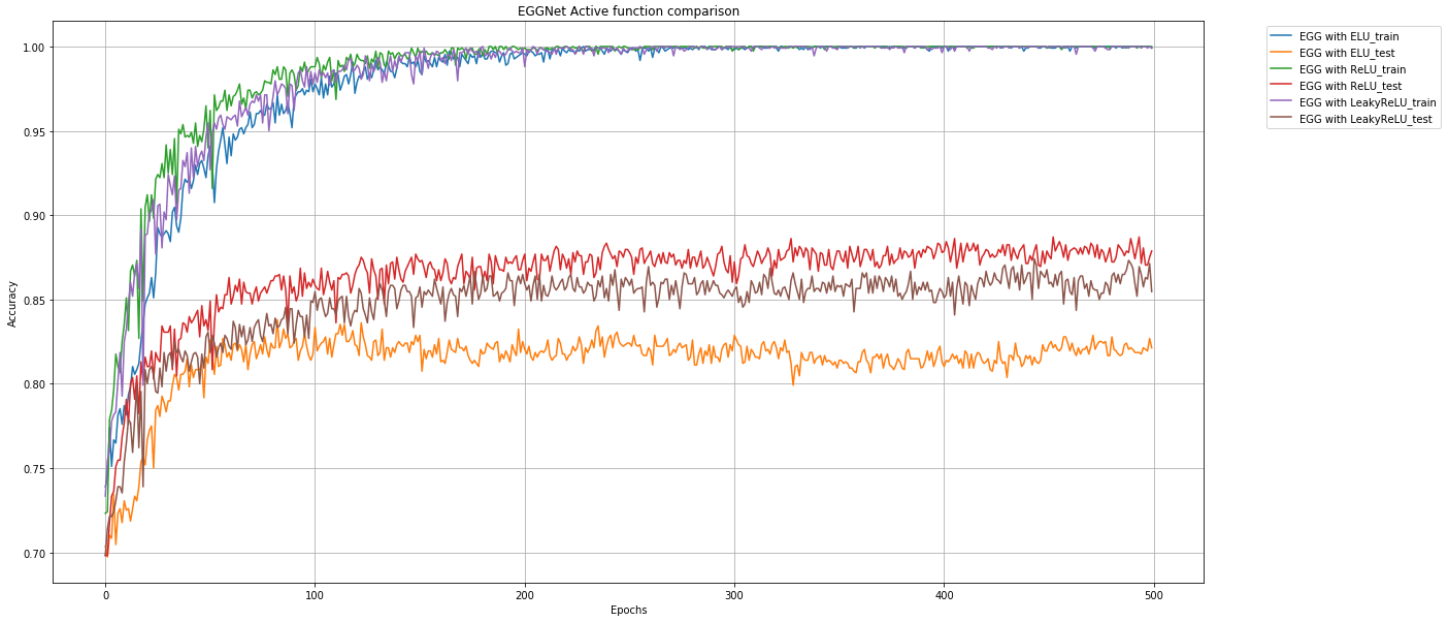
```
def shuffle_data(X, Y):
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)
    X = X[indices]
    Y = Y[indices]
    return X, Y
```

B. Comparison figures：

- **EEGNet：**



- **DeepConvNet：**



4. Discussion：

- **Batch size**

Batch size 決定一次訓練的樣本數目，其影響到模型的優化程度和速度，batch size 是否選擇的好，影響了記憶體效率和記憶體容量之間是否能尋找最佳平衡，其取值方法約略幾種:

1. Full batch：

若**資料集較小**，可採用全資料集。全資料集確定的方向能夠更好的代表樣本總體，從而更準確的朝向極值所在的方向，但是這個方法在比較大的資料中是不可行的。

2. mini batch：

選擇一個適中的 Batch Size 值，換言之我們選定一個 batch 的大小後，將會以 batch 的大小將資料輸入深度學習的網路中，然後計算這個 batch 的所有樣本的平均損失，即代價函式是所有樣本的平均。

3. Batch Size=1：

每次修正方向以各自樣本的梯度方向修正，難以達到收斂。

以一個正常資料集來說，當 Batch Size 太小，訓練資料就會非常難以收斂，從而導致 underfitting 的情況發生。

若增大 Batch Size，相對處理速度加快，但是增大 Batch Size，所需要的記憶體需求就會增大，同時 Epoch 的次數也需要增加才能達到最好的結果。

這裡就出現了 trade off 的情況發生，因為雖然因為 Batch Size 變大，處理效率相對提高，但 Epoch 同時也增加，導致耗時增加。

由 fig.10 可發現，這次 lab 當 batch size 較小時，準確率會來得比較高。

batch size	lr	Accur	NET	A_Func	epochs	shuffle
32	0.001	82.315	EEGNet	ELU	400	Yes
32	**0.001**	**87.037**	**EEGNet**	**ReLU**	**400**	**Yes**
32	0.001	87.407	EEGNet	LeakyReLU	400	Yes
64	0.001	84.722	EEGNet	ELU	400	Yes
64	**0.001**	**88.519**	**EEGNet**	**ReLU**	**400**	**yes**
64	0.001	85.833	EEGNet	LeakyReLU	400	Yes
128	0.001	84.444	EEGNet	ELU	400	Yes
128	**0.001**	**87.315**	**EEGNet**	**ReLU**	**400**	**Yes**
128	0.001	86.667	EEGNet	LeakyReLU	400	no
512	0.001	82.593	EEGNet	ELU	400	Yes
512	**0.001**	**86.019**	**EEGNet**	**ReLU**	**400**	**Yes**
512	0.001	86.019	EEGNet	LeakyReLU	400	Yes
1080	0.001	79.537	EEGNet	ELU	400	Yes
1080	**0.001**	**83.148**	**EEGNet**	**ReLU**	**400**	**Yes**
1080	0.001	82.13	EEGNet	LeakyReLU	400	yes

fig. 10

- **Learning rate issue：**

batch size	lr	Accur	NET	A_Func	epochs	shuffle
32	0.0001	80.741	EEGNet	ELU	400	Yes
32	0.0001	**85.093**	**EEGNet**	**ReLU**	**400**	**Yes**
32	0.0001	84.167	EEGNet	LeakyReLU	400	Yes
64	0.0001	77.778	EEGNet	ELU	400	Yes
64	0.0001	**85**	**EEGNet**	**ReLU**	**400**	**yes**
64	0.0001	83.519	EEGNet	LeakyReLU	400	Yes
128	0.0001	80.278	EEGNet	ELU	400	Yes
128	0.0001	**82.87**	**EEGNet**	**ReLU**	**400**	**Yes**
128	0.0001	81.852	EEGNet	LeakyReLU	400	Yes
512	0.0001	73.519	EEGNet	ELU	400	Yes
512	0.0001	**78.148**	**EEGNet**	**ReLU**	**400**	**Yes**
512	0.0001	80.185	EEGNet	LeakyReLU	400	Yes
1080	0.0001	74.074	EEGNet	ELU	400	Yes
1080	0.0001	**78.704**	**EEGNet**	**ReLU**	**400**	**Yes**
1080	0.0001	76.204	EEGNet	LeakyReLU	400	Yes

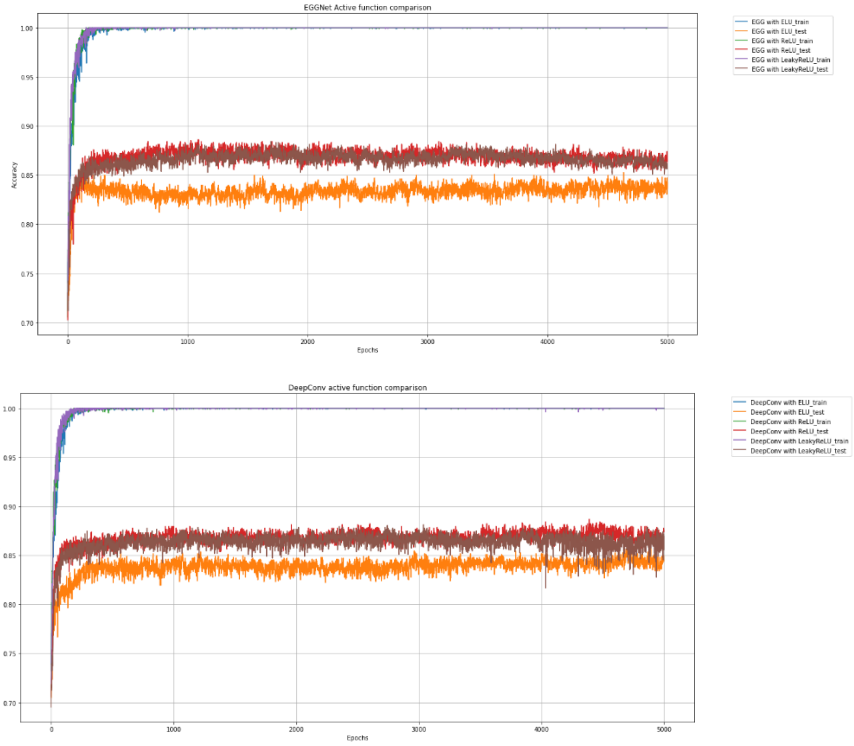
fig. 11

由 fig.10 跟 fig.11 比較可發現，在固定 epochs 的時候，learning rate 越低不依定會學習有更好的成效，反之可能會造成準確率降低，所以當 learning rate 調低時，epochs 要適時的調整。

- **epochs issue：**

EEGNet v.s DeepConvNet

Net	ReLU	Leaky ReLU	ELU
EEGNet	88.611	88.426	85.463
DeepConvNet	88.704	88.148	85.833



上圖為跑 5000epochs 的結果，可以看到準確率確實有提升，但沒有很顯著，我個人認為就是同樣的題目一直看一直看，沒有做變化去思考，自然在遇到新題目時不會變得更棒棒，所以 epochs 的影響到一定的數量後就不再這麼的顯著。

- **Dropout issue：**

Dropout = 0.1

EEGNet v.s DeepConvNet

Net	ReLU	Leaky ReLU	ELU
EEGNet	87.315	85.926	81.852
DeepConvNet	86.111	86.759	82.593

Dropout = 0.5

EEGNet v.s DeepConvNet

Net	ReLU	Leaky ReLU	ELU
EEGNet	88.704	87.407	83.796
DeepConvNet	87.685	87.130	83.889

Dropout = 0.9

EEGNet v.s DeepConvNet

Net	ReLU	Leaky ReLU	ELU
EEGNet	86.389	85.926	84.722
DeepConvNet	86.389	85.741	85.185

drop out 較低的時候正確率在訓練時很高，在測試時就表現差很大，意味著訓練是過擬合(overfitting)的，提高 drop out 後理論上就會改善，所以 drop out 是一種有效降低過擬合(overfitting)的方法，與權重衰減的精神有相似的效果，主要精神都是在訓練模型的時候，不要去過度依賴某些權重，進而達到正則化的效果，雖然實驗結果不明顯，但這個想法可以在之後繼續嘗試。