

Deep Learning and Practice

Lab4-1 : EEG classification

310605009

吳公耀

一、 Introduction (20%)

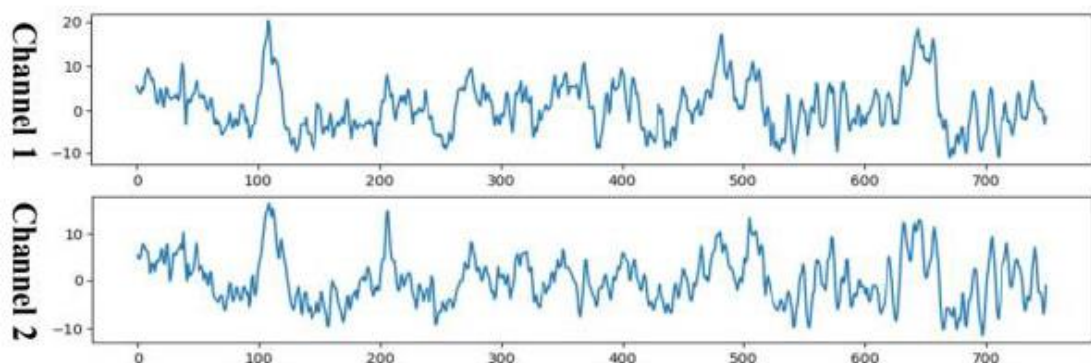
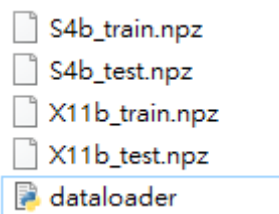
A. Lab Objective

這次的作業是要透過 pytorch 建構網路 EEGNet 以及 DeepConvNet 來實現 EEG 分類，且透過不同的 activation function 包括(ReLU, Leaky ReLu, ELU)來看看結果有甚麼影響。

B. Dataset

BCI Competition III - IIIb Cued motor imagery with online feedback (nonstationary classifier) with 2 classes (left hand, right hand) from 3 subjects [2 classes, 2 bipolar EEG channels]

The training data and testing data have been preprocessed and named [S4b_train.npz, X11b_train.npz] and [S4b_test.npz, X11b_test.npz] respectively



2 bipolar EEG signals

二、 Experiment set up(30%)

A. The detail of your model

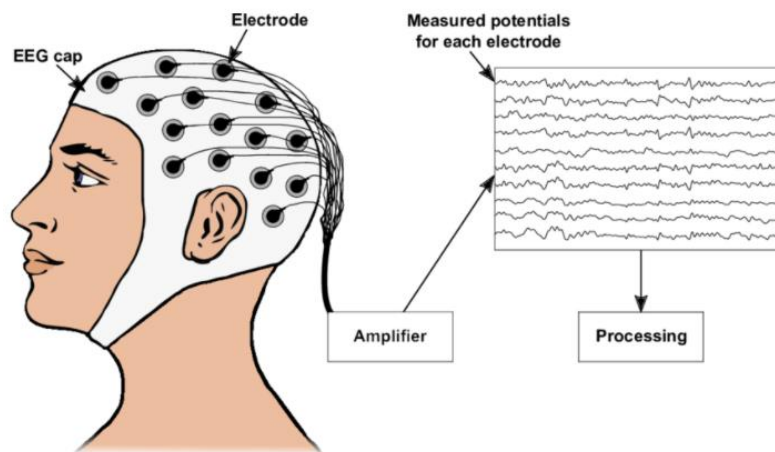
先轉成透過 GPU 運行，之後將助教給的資料做 dataloader，方便之後的訓練

```
device = torch.device("cuda")
train_data, train_label, test_data, test_label = dataloader.read_bci_data()

train_data = torch.tensor(train_data, dtype = torch.float).to(device)
train_label = torch.tensor(train_label, dtype = torch.long).to(device)
test_data = torch.tensor(test_data, dtype = torch.float).to(device)
test_label = torch.tensor(test_label, dtype = torch.long).to(device)

train_Dataset = Data.TensorDataset(train_data, train_label)
train_loader = Data.DataLoader(train_Dataset, batch_size = batch_size, shuffle = True)
```

● EEGNet

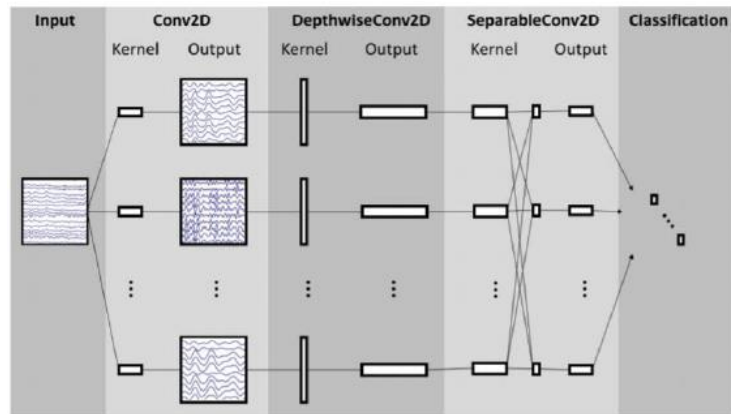


人類的大腦進行運作時，大腦的神經細胞會不斷放電，科學家運用腦波儀在人類頭皮上測得微弱的電波或磁變化，此即是腦量原理。

腦電圖、腦波圖（Electroencephalography, EEG）是透過醫學儀器腦電圖描記儀，將人體腦部自身產生的微弱生物電於頭皮處收集，並放大記錄而得到的曲線圖。

研究者希望藉由腦波和大意識的相對應關係，透過測量來推狀態。如在醫學領域，研究者會憂鬱症、阿茲海默患睡眠時的腦波以進一步研究病理和因。

下圖為 EGGNet 的架構



下圖為模型運行時的整個細節

```
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)
  )
  (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): ReLU()
    (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

根據助教所給的架構設計 DeepConvNet

You need to implement the DeepConvNet architecture by using the following table, where $C = 2$, $T = 750$ and $N = 2$. **The max norm term is ignorable.**

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

下圖為模型的整個細節

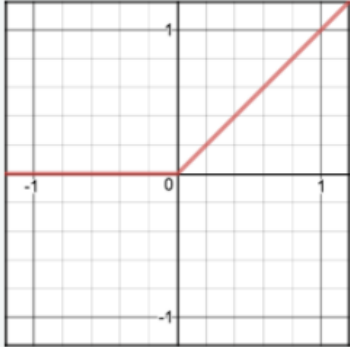
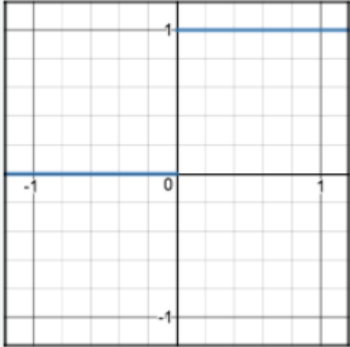
```

DeepConvNet(
  (conv0): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), bias=False)
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ReLU()
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (conv1): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (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(50, 100, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (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(100, 200, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (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=8600, out_features=2, bias=True)
  )
)

```

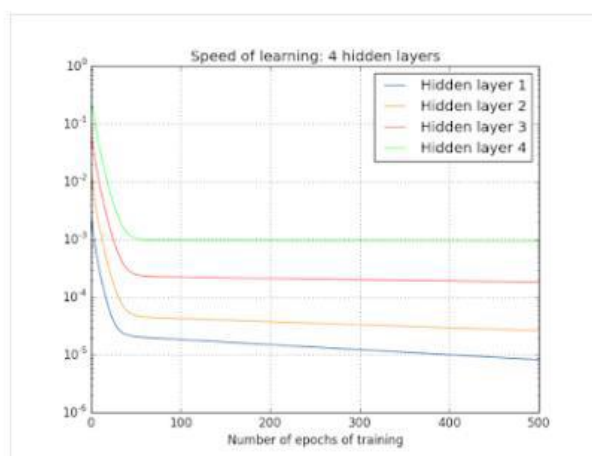
B. Explain the activation function (ReLU, Leaky ReLU, ELU)

● ReLU

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$
	
<pre>def relu(z): return max(0, z)</pre>	<pre>def relu_prime(z): return 1 if z > 0 else 0</pre>

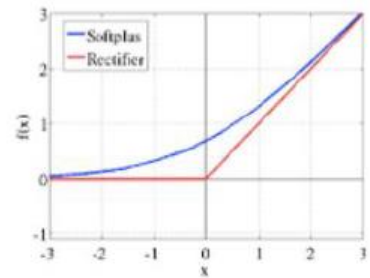
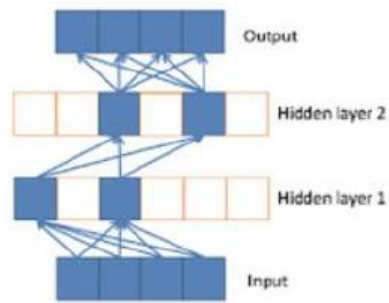
ReLU 是一個非線性單調、跟 0 取大值的函數。取大值的函數。ReLU 函數並不是全區間皆可微分，但是不可微分的部分可以使用 Sub-gradient 進行取代，ReLU 是近年來最頻繁被使用的激勵函數，其原因如下：

(1)解決的梯度消失問題



ReLU 的分段線性性質能有效的克服梯度消失的問題。

(2)類神經網路的稀疏性



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 (Overfitting) 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，當遇到以下兩種情形時容易導致 dead ReLU 發生。

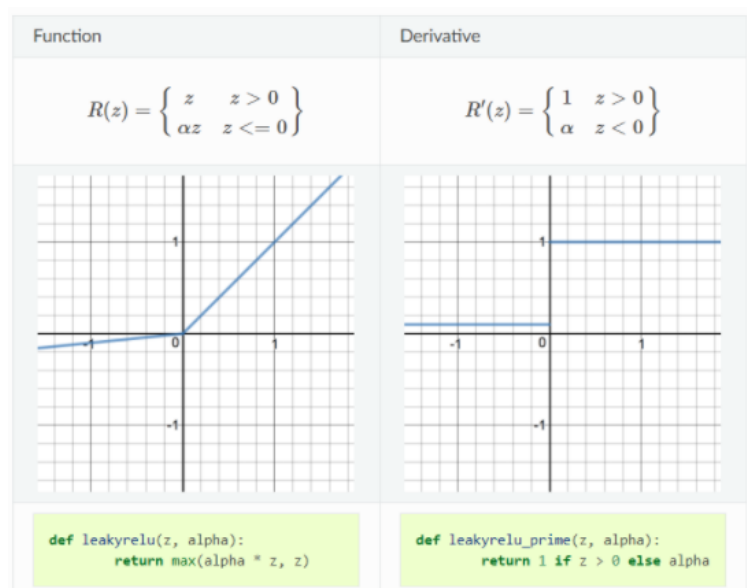
- 初始化權重設定為不能被激活的數值。
- 學習率設置過大，在剛開始進行誤差反向傳遞時，容易修正權重值過大，導致權重梯度為 0，神經元即再也無法被激活

(3)計算量大幅降低，只需要判斷是否大於 0

但相對地的缺點包括

- (1)只能在神經網路模型的隱藏層中使用。
- (2)可能會導致權重更新不了，因為輸入 $x < 0$ 時，梯度將為 0 (ReLU 的特性)，這樣會使權重調整不了，這稱為 dead ReLU problem。
- (3)當輸入大於 0，這個激勵函數可能會 blow up。

● Leaky ReLU



具有 ReLU 的特點，但 $x < 0$ 時梯度不為 0，可以解決 dead ReLU problem，但相對的缺點可能包括

- (1) 計算量稍大效能可能會低於 Sigmoid 與 Tanh。
- (2) $x < 0$ 時是線性的，因此不能在複雜的分類中使用。

基本上 Leaky ReLU 擁有 ReLU 的所有優點，也成功避免 dead ReLU problem 的問題產生，但是於實際使用上，並沒有辦法完全證明 Leaky ReLU 永遠優於 ReLU。

● ELU



ELU 是一種傾向於將數值更快地收斂到零並產生更準確結果的函數，且具有一個額外的 α 常數，且這個常數為正的。

而 ELU 也是為解決 ReLU 存在的問題而提出，ELU 有 ReLU 的基本優點，以及：

- (1) 較為緩慢的變平滑。
- (2) ELU 為 ReLU 的替代品，可以有負的輸出。
- (3) 不會有 Dead ReLU 問題
- (4) 輸出的均值接近 0，zero-centered

而存在的問題包括當輸入大於 0，這個激勵函數可能會 blow up。且理論上雖然好於 ReLU，但在實際使用中目前並沒有好的證據 ELU 總是優於 ReLU。

下圖可以看到這次作業中直接用 pytorch 裡面的函式，並沒有另外設計。

```
nn.ModuleDict([['ELU', nn.ELU(alpha = 1.0)], ['ReLU', nn.ReLU()], ['LeakyReLU', nn.LeakyReLU()]])
```

三、 Experimental results(30%)

A. The highest testing accuracy

這是目前最高的準確率，訓練的參數設定為 Batch size=256, lr=7e-4, epoch=1000, Optimizer = Adam

	ReLU	ELU	LeakyReLU
EEGNet	0.882407	0.851852	0.892593
DeepConvNet	0.834259	0.823148	0.843519

最高可到達 89.20%準確率

● Screenshot with two models

(1)EEGNet

```
class EEGNet(nn.Module):
    #根據spec架構創建 EEGNet 模型
    def __init__(self, activation = None, dropout=0.25):
        super(EEGNet, self).__init__()
        activation_function = nn.ModuleDict([['ELU', nn.ELU(alpha = 1.0)], ['ReLU', nn.ReLU()], ['LeakyReLU', nn.LeakyReLU()]])

        self.firstconv = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size = (1, 51), stride = (1,1), padding = (0,25), bias = False ),
            nn.BatchNorm2d(16, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True)
        )

        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size = (2,1), stride = (1,1), groups = 16, bias = False ),
            nn.BatchNorm2d(32, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True),
            activation_function[activation],
            nn.AvgPool2d(kernel_size = (1, 4), stride = (1, 4), padding = 0),
            nn.Dropout(p = dropout)
        )

        self.separableConv = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size = (1, 15), stride = (1,1), padding = (0, 7), bias = False
            ),
            nn.BatchNorm2d(32, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True),
            #選擇 activation function
            activation_function[activation],
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout(p = dropout)
        )

        self.classify = nn.Sequential(
            nn.Linear(736, out_features = 2, bias = True)
        )

    def forward(self, x):
        x = self.firstconv(x)
        x = self.depthwiseConv(x)
        x = self.separableConv(x)
        # flatten
        x = x.view(x.size(0), -1)
        output = self.classify(x)
        return output
```


(2)DeepConvNet

```
class DeepConvNet(nn.Module):
    def __init__(self, activation=None, deepconv=[25,50,100,200], dropout=0.5):
        super(DeepConvNet, self).__init__()
        activation_function = nn.ModuleDict([['ELU', nn.ELU(alpha = 1.0)], ['ReLU', nn.ReLU()], ['LeakyReLU', nn.LeakyReLU()]])

        self.deepconv = deepconv
        self.conv0 = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size = (1, 5), stride = (1, 1), bias = False),
            nn.Conv2d(25, 25, kernel_size = (2, 1), stride = (1, 1), bias = False),
            nn.BatchNorm2d(25, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True),
            activation_function[activation],
            nn.MaxPool2d(kernel_size = (1, 2)),
            nn.Dropout(p = 0.5)
        )

        for idx in range(1, len(deepconv)):
            setattr(self, 'conv'+str(idx), nn.Sequential(
                nn.Conv2d(deepconv[idx-1], deepconv[idx], kernel_size=(1,5),stride=(1,1), padding=(0,0), bias=False),
                nn.BatchNorm2d(deepconv[idx], eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True),
                activation_function[activation],
                nn.MaxPool2d(kernel_size=(1, 2)),
                nn.Dropout(p=dropout)
            ))

        flatten_size = deepconv[-1] * reduce(
            lambda x,_: round((x-4)/2), deepconv, 750)
        self.classify = nn.Sequential(
            nn.Linear(8600, 2, bias=True),
        )

    def forward(self, x):
        for i in range(len(self.deepconv)):
            x = getattr(self, 'conv'+str(i))(x)
        # flatten
        x = x.view(x.size(0), -1)
        output = self.classify(x)
        return output
```

● anything you want to present

(1) epoch

以下未針對不同 epoch 進行實驗，其他參數設定為 learning rate=7e-4,
batch size =256, Optimizer = Adam

epoch	accuracy			
700		ReLU	ELU	LeakyReLU
	EEGNet	0.881481	0.847222	0.875926
	DeepConvNet	0.814815	0.816667	0.840741
2000		ReLU	ELU	LeakyReLU
	EEGNet	0.887037	0.837037	0.883333
	DeepConvNet	0.832407	0.814815	0.836111
5000		ReLU	ELU	LeakyReLU
	EEGNet	0.882407	0.851852	0.892593
	DeepConvNet	0.834259	0.823148	0.843519

(2) batch size

以下未針對不同 batch size 進行實驗，其他參數設定為 learning rate=7e-4, epoch=700, Optimizer = Adam

batch size	accuracy			
32		ReLU	ELU	LeakyReLU
	EEGNet	0.876852	0.844444	0.872222
	DeepConvNet	0.817593	0.808333	0.817593
64(default)		ReLU	ELU	LeakyReLU
	EEGNet	0.872222	0.834259	0.873148
	DeepConvNet	0.820370	0.818519	0.832407
256		ReLU	ELU	LeakyReLU
	EEGNet	0.881481	0.847222	0.875926
	DeepConvNet	0.814815	0.816667	0.840741
1024		ReLU	ELU	LeakyReLU
	EEGNet	0.858333	0.833333	0.860185
	DeepConvNet	0.819444	0.809259	0.805556

(3) learning rate

以下未針對不同 learning rate 進行實驗，其他參數設定為 batch size=256, epoch=700, Optimizer = Adam

learning rate	accuracy			
7e-4		ReLU	ELU	LeakyReLU
	EEGNet	0.881481	0.847222	0.875926
	DeepConvNet	0.814815	0.816667	0.840741
1e-2		ReLU	ELU	LeakyReLU
	EEGNet	0.872222	0.840741	0.850000
	DeepConvNet	0.800926	0.828704	0.818519
0.01		ReLU	ELU	LeakyReLU
	EEGNet	0.837963	0.823148	0.873148
	DeepConvNet	0.812963	0.817593	0.811111

(4)Optimizer

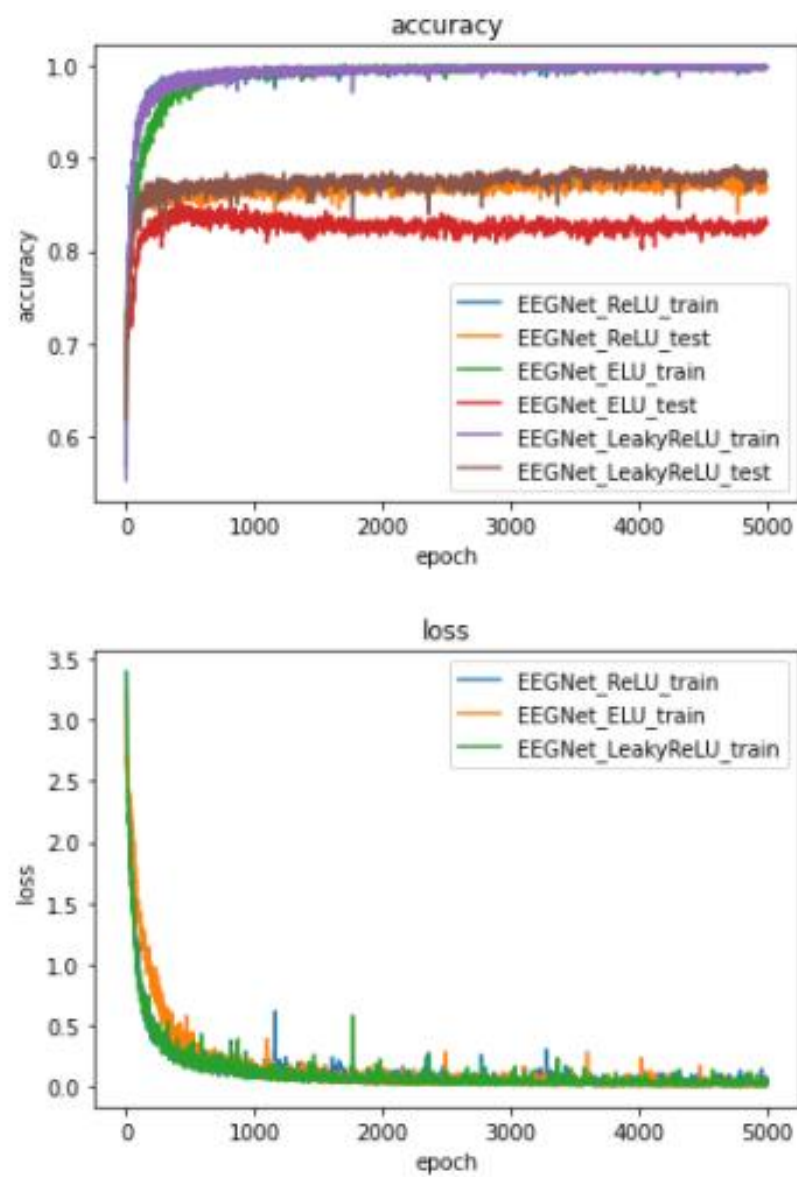
以下未針對不同 Optimizer 進行實驗，其他參數設定為 batch size=256, epoch=1000, learning rate 根據不同優化器進行調整，取最高的準確率。Epoch 也有加多一點因為有些優化器收斂得比較慢。

Optimizer/ Learning rate	accuracy			
Adam (default)/ 7e-4		ReLU	ELU	LeakyReLU
	EEGNet	0.892593	0.835185	0.876852
	DeepConvNet	0.829630	0.823148	0.835185
SGD/ 0.01		ReLU	ELU	LeakyReLU
	EEGNet	0.865741	0.826852	0.850926
	DeepConvNet	0.800926	0.792593	0.790741
Momentum/ 0.01		ReLU	ELU	LeakyReLU
	EEGNet	0.867593	0.824074	0.854630
	DeepConvNet	0.803704	0.789815	0.794444
AdaGrad/ 0.01		ReLU	ELU	LeakyReLU
	EEGNet	0.861111	0.837037	0.854630
	DeepConvNet	0.823148	0.817593	0.809259
RMSprop/ 7e-4		ReLU	ELU	LeakyReLU
	EEGNet	0.874074	0.839815	0.865741
	DeepConvNet	0.799074	0.815741	0.818519
	END			

B. Comparison figures

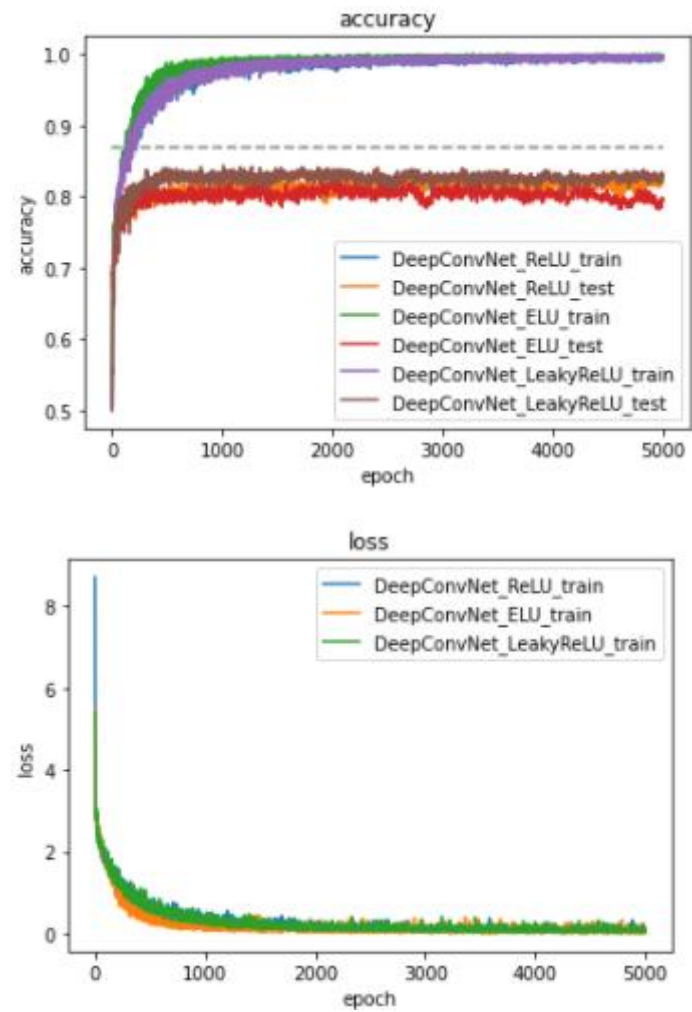
(1) EEGNet

下圖為得到 EEGNet 最高 accuracy 時的比較圖。 的



(2)DeepConvNet

下圖為得到 DeepConvNet 最高 accuracy 時的比較圖

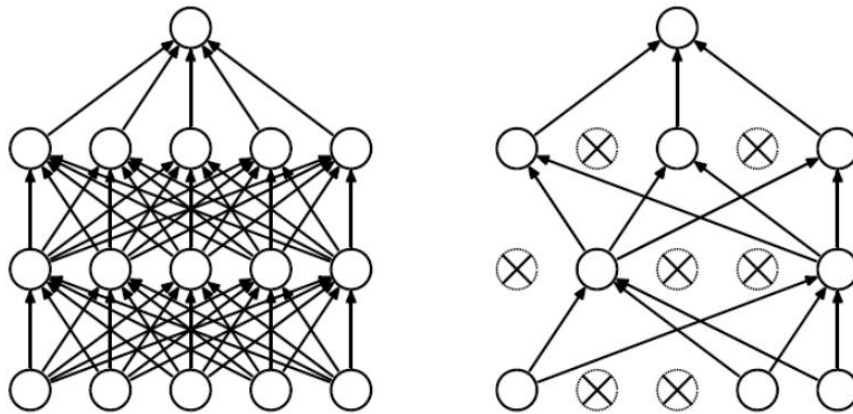


四、 Discussion (20%)

(一) Droup out

1. 介紹

在訓練神經網路的時候經常會遇到過 overfitting 的問題，過 overfitting 具體表現在：模型在訓練資料上 loss 較小，預測準確率較高；但是在測試資料上損失函式比較大，預測準確率較低。為防止網路的 overfitting，透過丟棄網路中的某些 neuron 來防止 feature 之間的合作 (co-adaption)，當某些 feature 被丟棄後，網路必須學會利用剩下的 feature 進行分類。Dropout 可以比較有效的緩解過擬合的發生，在一定程度上達到正則化的效果。由於每次都是隨機的丟棄，網路必須學會利用剩餘的 feature 本身來進行分類，而非透過某幾個 feature 所形成特定行為來進行分類。



透過減少神經網路的層數、神經元個數等方式，可以限制神經網路的擬合能力，而 Dropout 的處理方式是隨機關閉一些神經元

2. 為什麼 Dropout 可以解決 overfitting?

(1) 取平均的作用

如果用相同的訓練資料去訓練 5 個不同的神經網路，一般會得到 5 個不同的結果，此時我們可以採用“5 個結果取均值”或者“多數取勝的投票策略”去決定最終結果。這種“綜合起來取平均”的策略通常可以有效防止過擬合問題。不同的網路可能產生不同的過擬合，取平均則有可能讓一些“相反的”擬合互相抵消。dropout 掉不同的隱藏神經元就類似在訓練不同的網路，隨機刪掉一半隱藏神經元導致網路結構已經不同，整個 dropout 過程就相當於對很多個不同的神經網路取平均。而不同的網路產生不同的過 overfitting，一些互為“反向”的擬合相互抵消就可以達到整體上減少過 overfitting。

(2)減少神經元之間複雜的共適應關係

因為 dropout 程式導致兩個 neuron 不一定每次都在一個 dropout 網路中出現。這樣 weight 的更新不再依賴於有固定關係的隱含節點的共同作用，阻止了某些 feature 僅僅在其它特定 feature 下才有效果的情況。迫使網路去學習更加 robustness 的 feature，這些 feature 在其它的 neuron 的隨機子集中也存在。

(二) Batch size

除了梯度本身，批量大小 (batchsize) 以及學習率 (learning rate) 直接決定了模型的權重更新，從優化本身來看是影響性能收斂最重要的參數

1. 大的 batchsize 減少訓練時間，提高穩定性但須的內存容量增加

同樣的 epoch 數量，大的數量，大的批量大小需要的數減少了，所以可訓練時間，目前已經有多篇文章公開論在 1 小時內訓練完 ImageNet 數據集。穩定，因為模型訓練曲線會更加平滑。在微調的時候大批量可能取得好的結果。

2. 學習率和 batchsize 的關係

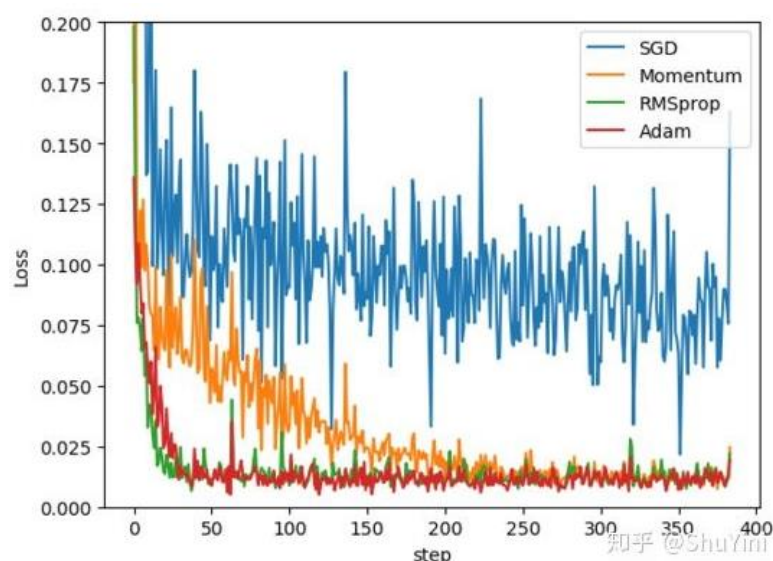
通常當我們增加 batchsize 為原來的 N 倍時，要保證經過同樣的本後更新權重替代，按照線性縮放規則學習率應該增加為原來的 N 倍。但是如果保證權重的方差不變，則學習率應該增加為原來的 \sqrt{N} 倍，目前這兩種策略都被研究過，使用前者的明顯居多。

結論：

- (1) 如果增加了學習率，那麼批量大小最好也跟著這樣收斂更穩定。
- (2) 如果真的要衰減，可以嘗試其他方法從而增加批量大小學習率對模型收斂影響真的很容易大。
- (3) 如果 batchsize 過小，訓練將會很難收斂，而導致 underfitting

可以從上面的實驗結果看到隨這增加 batchsize 大小，準確率也有些微的提升，但是太大的時候因為跑完一次 epoch 所需的迭代次數減少，要想達到相同的精度，其所花費的時間大大增加了，從而對參數的修正也就顯得更加緩慢跑完一次 epoch 所需的迭代次數減少，要想達到相同的精度，其所花費的時間大大增加了，從而對參數的修正也就顯得更加緩慢，反而沒有為訓練帶來比較好的效益。

(三) Optimizer



Optimizer	特點
Adam	<ul style="list-style-type: none"> ◆ 結合了AdaGrad與Momentum的優點 ◆ 適用於大數據集和高維空間的資料 ◆ 目前最常使用的Optimizer
SGD	<ul style="list-style-type: none"> ◆ 應用大型數據集時，訓練速度很快 ◆ 只要噪聲不是特別大，SGD 都能很好地收斂 ◆ 需要自行設定learning rate，較難選擇到合適的learning rate ◆ 會造成loss function 有嚴重的震盪 ◆ 需要較長時間收斂至最小值
Momentum	<ul style="list-style-type: none"> ◆ 能夠在相關方向加速SGD，抑制SGD的嚴重震盪，進而加快收斂 ◆ 需自行設定 learning rate 和 β，有可能會使參數的移動方向偏移梯度下分的方向，進而導致沒有那麼快速的收斂
AdaGrad	<ul style="list-style-type: none"> ◆ 能夠自動調整learning rate，進而調整收斂 ◆ 適合處理稀疏梯度 ◆ 依然需要人工設置一個全局的learning rate ◆ 後期，分母梯度平方的累加會越來越大，會使梯度趨近於0，使得訓練結束
RMSprop	<ul style="list-style-type: none"> ◆ 每一次更新 learning rate 時，分母所除的 σ 都與前一次的有關係 ◆ 調整上面多了一個參數 α，可以自由調整新舊 gradient 的比重

雖然上面每個不同優化器都有做一些不同 learning rate 的調整，但是因為時間不夠，沒辦法好好一個一個去試看看每個 learning rate、batch size、epoch…找到最適合這兩個模型的參數，像是 SGD 雖然準確率沒有很好，但是印出來的結果 loss 也還沒有完全收斂結束，說不定再繼續訓練下去也會有更好的結果。每個不同的 model 配上不同得優化器也會有不一樣的結果。

(四) activation function

可以發現不同的 activation function 在相同的模型架構配合不一樣的參數都會有不同的結果。以這次實驗來說原本預期 ReLU 的模型會比使用其他兩個 function 的結果來的差一點，因為後來看網路上資料發現另外兩個 function 是為了處理 ReLU 的缺點而設計的，結果 ReLU 的結果不論是 EEGNet 以及 DeepConvNet 大部分都比其他兩個準確率稍微高了一點，所以以後再 activation function 的選擇上，可能還是得都進行嘗 才能確定使用哪個函數才會有最好的結果。

(五) EEGNet 以及 DeepConvNet

EEGNet僅有17874個參數，而DeepConvNet則有150577個參數，主要差在CNN的層數以及filter的數量。而DeepConvNet有著這麼多參數，但在這個實驗中沒有比較好的表現，反而準確率還比較低，訓練所需要的時間也比較長，不過這也可能是因為EEGNet是特別針對這個任務所設計的模型，所以才有更好的表現，可以了解到不段的增加模型的深度與廣度不一定會給實驗結果帶來好的影響，不同的數據種的特徵點都需要不同模型來配合才會有不一樣的結果，之後可以透過更多的練習來了解不同類型搭配怎麼樣的模型會有比較好的表現。