Lab2 : EEG classification

IOC/資科工碩

學號: 310551031

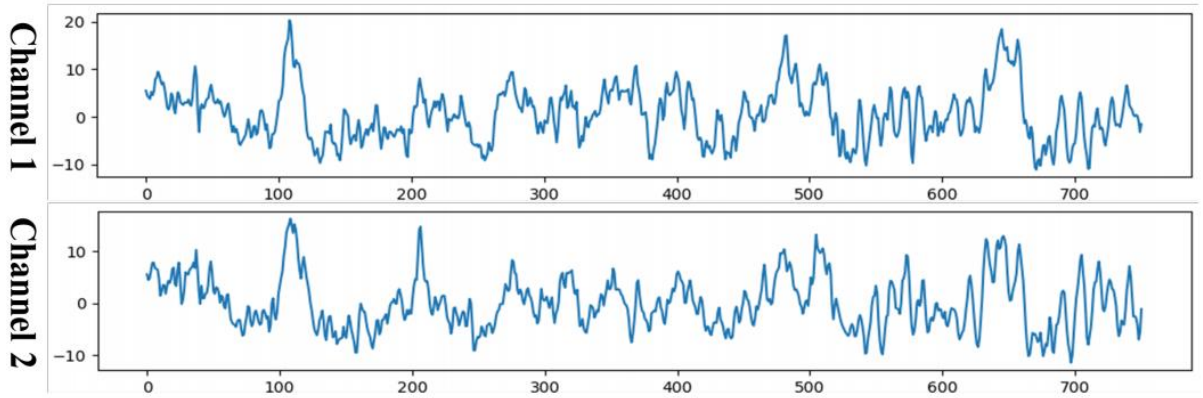
張皓雲

目錄

1. Introduction

A. 問題定義、達成目標與資料說明

本實驗注重在實作 EEGNet 與 DeepConvNet，並調整模型內的各項 Layer 內的參數或是超參數(hyper-parameters)，以進行 BCI Competition III 比賽的腦電圖訊號分類。除此之外，本實驗並無另外對資料進行額外的處理，不過卻希望嘗試將多個優化器(Optimizers)、Learning rate schedule 及不同的 Loss function 等方法一併加入實驗進行實驗的探討。此次需要分類的訊號總共有兩個，分別為左手(Left hand)與右手(Right hand)的腦電圖訊號資料。在**圖一**中可看到本次實驗的資料總共有兩個 Channel，分別為 Channel 1 與 Channel 2，其中每個 Channel 都有 750 個樣本點資料，因此單筆資料的維度為(1,2,750)。而本次實驗總共使用 1080 筆資料進行訓練，因此訓練資料的維度為(1080,1,2,750)。對應到本實驗的訓練資料數量，本實驗的訓練標籤同樣為 1080 筆。由於每一種 Loss Function 在計算 Loss 值時需要不同的資料維度，因此本實驗的訓練標籤維度可以分為(1080,1)與(1080,)兩種。本次實驗結果的 accuracy 均可以超過 83%，最高甚至可以到 88%。



圖一、腦電圖訊號資料示意圖

B. 實驗環境與檔案使用說明

B.1. 實驗環境

本次實驗使用的 python 套件有 os、Numpy、Pandas、Matplotlib、Collections 及 Pytorch 的各項子套件等。Numpy 的用途在儲存各項資料、數值的運算及統計等。Pandas 與 Collections 的用途則為進行數據結果的分析、儲存各項數據結果，例如: Confusion Matrix、儲存訓練的 accuracy 與 loss 數值等…。Matplotlib 的用途則是繪出 TrainingAccuracy 與 Testing Accuracy 隨著 epochs 變動的曲線圖、Training loss 與 Testing Accuracy 隨著 epochs 變動的曲線圖等….。套件的 Logo 如**圖二**、**圖三**、**圖四**與**圖五**所示。硬體設備、虛擬環境如**表一**所述。

表一、軟硬體實作環境

	實驗實作環境
處理器	**Intel(R) Core(TM) i7-6700 CPU@3.40GHz**

Python 版本	3.6.13
虛擬環境	Anaconda 2.0.4
實作套件	os、Numpy、Matplotlib、Collection、Pandas、Pytorch
作業系統	Microsoft Windows 10

圖二、Pandas 套件 Logo

圖三、Matplotlib 套件 Logo

圖四、Numpy 套件 Logo

圖五、Pytorch 套件 Logo

B.2. 檔案使用說明

Source code 資料夾中總共有 dataloader.py、EEGNet_training_ELU.py、EEGNet_training_ReLU.py、EEGNet_training_LeakyReLU.py、DeepConvNet_training_ELU.py、DeepConvNet_training_ReLU.py 及 DeepConvNet_training_LeakyReLU.py、Plot_History_Result 等八種檔案。

B.2.1. dataloader.py 檔案功能說明

dataloader.py 檔案的功能為製作實驗所需的訓練、測試資料，回傳的資料型態為 Numpy。其中回傳的訓練、測試資料與標籤維度為(1080,1,2,750)和(1080,)，(1080,)維度指的是 1x1080 的矩陣。

B.2.2. EEGNet_training_ELU.py 檔案功能說明

EEGNet_training_ELU.py 檔案的功能為訓練 Activation function 為 ELU 的 EEGNet 模型、產生視覺化訓練結果、產生測試結果、產生其他相關結果及示意圖、儲存最好的模行權重等...。使用此檔案時，需將 dataloader.py 放置於跟此檔案同個資料夾下。此檔案會將訓練與測試過程中的 loss 與 accuracy 記錄下來並存至檔名為 EEGNet_ELU.csv 的檔案內(此檔案會放在 history_csv 資料夾下)，並將此模型訓練的權重檔以 EEGNet_checkpoint_ELU.rar 的名稱儲存(此檔案會放在 checkpoint 資料夾下)。

B.2.3. EEGNet_training_ReLU.py 檔案功能說明

EEGNet_training_ReLU.py 檔案的功能為訓練 Activation function 為 ReLU 的 EEGNet 模型、產生視覺化訓練結果、產生測試結果、產生其他相關結果及示意圖、儲存最好的模行權重等...。使用此檔案時，需將 dataloader.py 放置於跟此檔案同個資料夾下。此檔案會將訓練與測試過程中的 loss 與 accuracy 記錄下來並存至檔名為 EEGNet_ReLU.csv 的檔案內(此檔案會放在 history_csv 資料夾下)，並將此模型訓練的權重檔以 EEGNet_checkpoint_ReLU.rar 的名稱儲存(此檔案會放在 checkpoint 資料夾下)。

B.2.4. EEGNet_training_LeakyReLU.py 檔案功能說明

EEGNet_training_LeakyReLU.py 檔案的功能為訓練 Activation function 為 LeakyReLU 的 EEGNet 模型、產生視覺化訓練結果、產生測試結果、產生其他相關結果及示意圖、儲存最好的模行權重等...。使用此檔案時，需將 dataloader.py 放置於跟此檔案同個資料夾下。此檔案會將訓練與測試過程中的 loss 與 accuracy 記錄下來並存至檔名為 EEGNet_LeakyReLU.csv 的檔案內(此檔案會放在 history_csv 資料夾下)，並將此模型訓練的權重檔以 EEGNet_checkpoint_ReLU.rar 的名稱儲存(此檔案會放在 checkpoint 資料夾下)。

B.2.5. DeepConvNet_training_ELU.py 檔案功能說明

DeepConvNet_training_ELU.py 檔案的功能為訓練 Activation function 為 ELU 的 DeepConvNet 模型、產生視覺化訓練結果、產生測試結果、產生其他相關結果及示意圖、儲存最好的模行權重等...。使用此檔案時，需將 dataloader.py 放置於跟此檔案同個資料夾下。此檔案會將訓練與測試過程中的 loss 與 accuracy 記錄下來並存至檔名為 DeepConvNet_ELU.csv 的檔案內(此檔案會放在 history_csv 資料夾下)，並將此模型訓練的權重檔以 DeepConvNet_checkpoint_ELU.rar 的名稱儲存(此檔案會放在 checkpoint 資料夾下)。

B.2.6. DeepConvNet_training_ReLU.py 檔案功能說明

DeepConvNet_training_ReLU.py 檔案的功能為訓練 Activation function 為 ReLU 的 DeepConvNet 模型、產生視覺化訓練結果、產生測試結果、產生其他相關結果及示意圖、儲存最好的模行權重等...。使用此檔案時，需將 dataloader.py 放置於跟此檔案同個資料夾下。此檔案會將訓練與測試過程中的 loss 與 accuracy 記錄下來並存至檔名為 DeepConvNet_ReLU.csv 的檔案內(此檔案會放在 history_csv 資料夾下)，並將此模型訓練的權重檔以 DeepConvNet_checkpoint_ReLU.rar 的名稱儲存(此檔案會放在 checkpoint 資料夾下)。

B.2.7. DeepConvNet_training_LeakyReLU.py 檔案功能說明

DeepConvNet_training_LeakyReLU.py 檔案的功能為訓練 Activation function 為 LeakyReLU 的 DeepConvNet 模型、產生視覺化訓練結果、產生測試結果、產生其他相關結果及示意圖、儲存最好的模行權重等...。使用此檔案時，需將 dataloader.py 放置於跟此檔案同個資料夾下。此檔案會將訓練與測試過程中的 loss 與 accuracy 記錄下來並存至檔名為 DeepConvNet_LeakyReLU.csv 的檔案內(此檔案會放在 history_csv 資料夾下)，並將此模型訓練的權重檔以 DeepConvNet_checkpoint_LeakyReLU.rar 的名稱儲存(此檔案會放在 checkpoint 資料夾下)。

B.2.8. Plot_History_Result.py 檔案功能說明

如**圖六**所示，在使用此檔案前，請確保前面 B.2.2.至 B.2.7 的檔案都有執行成功。並在同一個資料夾下創建 checkpoint 資料夾及 history_csv。此檔案的功能為將前面產生的 loss 及 accuracy 資料，繪至成各項曲線圖。其中包含全部模型訓練的 Loss 曲線圖、EEGNet Accuracy 的 Activation function 比較曲線圖及 DeepConvNet Accuracy 的 Activation function 比較曲線圖。

圖六、檔案執行先後順序

B.2.9. ALL_model.py 檔案功能說明

在這個檔案裡面，存放 B.2.2 至 B.2.7 的模型架構，因此在 model.testing 的檔案內就可以呼叫此檔案的模型架構進行測試。

B.2.10. model_testing.py 檔案功能說明

此檔案的功能為測試所有的模型結果，並透過 pandas 將所有的結果整合並顯示出來。

2. Experiment setups

A. 模型架構說明

A.1. EEGNet

本實驗使用的 EEGNet 模型架構圖與模型參數圖如**圖七**與**圖八**所示。在此模型架構中，本實驗總共使用到以下六種 Layer，分別為 Conv2d、BatchNorm2d、Avgpool2d、Dropout、Flatten、Linear。Activation function 的介紹則留到 B 部份解釋。在此架構中，本實驗專注在調整 Activation function、Dropout。在訓練 EEGNet 模型時，本實驗經常遇到 Overfitting 的問題，及模型陷入區域最佳解的狀況。因此在這部份中，Dropout、BatchNorm2d 及 Activation function 就成為本實驗必須要調整的兩個 Layer。EEGNet 模型的作用是會先使用 Convolution 抽取資料的特徵，再使用 BatchNormalization 將特徵資料進行 Normalize，因此特徵資料的分佈就會變成平均值為 0 的 Normal Distribution。當資料變成 Normal Distribution 之後，資料的分佈就會被限縮在 0 附近，因此資料在進入下一層 Activation function 時就不會有太多的特徵資料被去除，梯度消失的狀況也比較不容易發生。最後再使用 Dropout，讓某個神經元的啟用值以一定的機率停止工作，如此模型也不會學習到太多區域性的特徵。

```
(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=8, bias=False)
  (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ELU(alpha=0.1)
  (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
  (4): Dropout(p=0.35, 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): ELU(alpha=0.1)
  (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
  (4): Dropout(p=0.35, inplace=False)
)
(classify): Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=736, out_features=2, bias=True)
)
```

圖七、EEGNet 模型架構圖

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]	128
BatchNorm2d-4	[-1, 32, 1, 750]	64
ELU-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
ELU-10	[-1, 32, 1, 187]	0
AvgPool2d-11	[-1, 32, 1, 23]	0
Dropout-12	[-1, 32, 1, 23]	0
Flatten-13	[-1, 736]	0
Linear-14	[-1, 2]	1,474

Total params: 17,938
Trainable params: 17,938
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

圖八、EEGNet 模型 output shape 與參數量

A.2. DeepConvNet

本實驗使用的 DeepConvNet 模型架構圖與模型參數圖如**圖九**與**圖十**所示。在此模型架構中，本實驗總共使用到以下六種 Layer，分別為 Conv2d、BatchNorm2d、MaxPool2d、Dropout、Flatten、Linear。Activation function 的介紹同樣留到 B 部份解釋。在此架構中，本實驗專注在調整 Activation function、Dropout 及 BatchNorm2d 等 Layer。在訓練 DeepConvNet 模型時，本實驗經常遇到 Overfitting 的問題(測試資料集的 Accuracy 比訓練資料集的 Accuracy 還低)，甚至是模型陷入區域最佳解的狀況。因此在這部份中，Dropout、BatchNorm2d 及 Activation function 就成為本實驗必須要調整的 Layer。

```
(model): 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): LeakyReLU(negative_slope=0.04)
  (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
  (5): Dropout(p=0.44, inplace=False)
  (6): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), bias=False)
  (7): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): LeakyReLU(negative_slope=0.09)
  (9): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
  (10): Dropout(p=0.44, inplace=False)
  (11): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1), bias=False)
  (12): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (13): LeakyReLU(negative_slope=0.04)
  (14): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
  (15): Dropout(p=0.44, inplace=False)
  (16): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1), bias=False)
  (17): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (18): LeakyReLU(negative_slope=0.09)
  (19): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
  (20): Dropout(p=0.44, inplace=False)
  (21): Flatten(start_dim=1, end_dim=-1)
  (22): Linear(in_features=8600, out_features=2, bias=True)
)
```

圖九、DeepConvNet 模型架構圖

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 25, 2, 746]	125
Conv2d-2	[-1, 25, 1, 746]	1,250
BatchNorm2d-3	[-1, 25, 1, 746]	50
LeakyReLU-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]	6,250
BatchNorm2d-8	[-1, 50, 1, 369]	100
LeakyReLU-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]	25,000
BatchNorm2d-13	[-1, 100, 1, 180]	200
LeakyReLU-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]	100,000
BatchNorm2d-18	[-1, 200, 1, 86]	400
LeakyReLU-19	[-1, 200, 1, 86]	0
MaxPool2d-20	[-1, 200, 1, 43]	0
Dropout-21	[-1, 200, 1, 43]	0
Flatten-22	[-1, 8600]	0
Linear-23	[-1, 2]	17,202

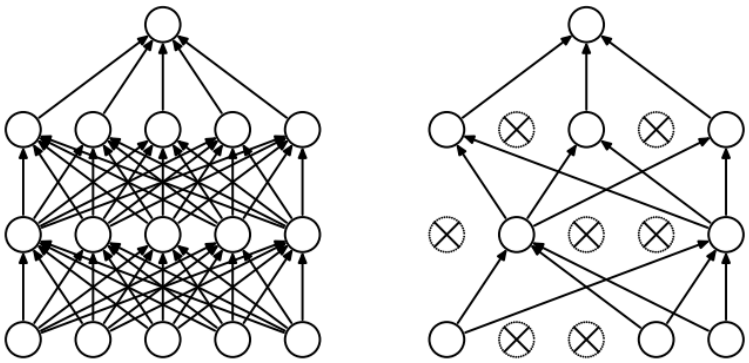
Total params: 150,577
Trainable params: 150,577
Non-trainable params: 0

Input size (MB): 0.01
Forward/backward pass size (MB): 2.56
Params size (MB): 0.57
Estimated Total Size (MB): 3.14

圖十、DeepConvNet 模型 output shape 與參數量

A.3. Dropout 的使用與說明

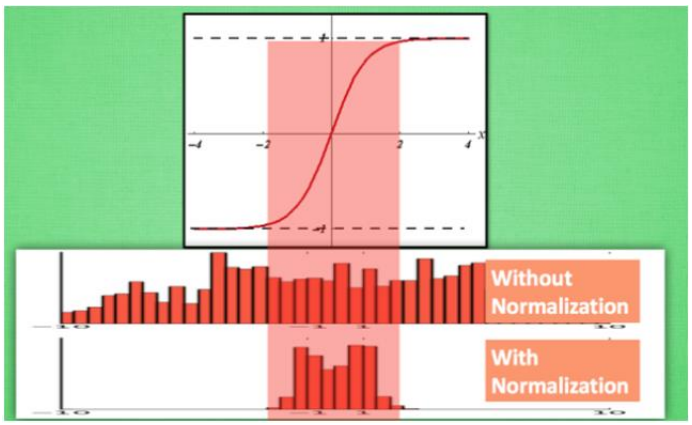
在機器學習與深度學習模型中，當模型的引數太多，樣本數量太少的情況下，就會使得訓練出來的模型容易產生過度擬合(Overfitting)的情況。Dropout 為適合解決此問題的幾個方法之一。如**圖十一**所示，Dropout 在向前傳播的時候，可以讓某個神經元的啟用值以一定的機率停止工作。如此就可以使得模型不會太依賴某些區域性的特徵，模型泛化性就會變得更強。因此對於 DeepConvNet 這種參數量較多的模型來說，加入 Dropout 之後，將會使得模型能夠不容易發生過度擬合(Overfitting)的現象。



圖十一、使用 Dropout 前(左)後(右)的網路狀況

A.4. BatchNorm2d 的使用與說明

BatchNormalization 的方法與先對輸入數據做 feature scaling 再對模型進行訓練的方法相當類似。一般在訓練模型之前，都會先將資料進行 normalize 的動作，如此就可以使得模型的收斂速度變得非常快。而 BatchNormalization 的做法就是在模型的每一層輸入之前，對資料做 normalize 的動作。因此模型層間的資料輸入分佈就會變成平均值為 0、標準差為 1 的 Normal Distribution。梯度消失的問題也較不容易發生。如**圖十二**所示，在模型層中，資料輸入的分佈可能會如圖中的 Without Normalization 一樣分佈較為廣泛。但受到 Activation function 的影響，輸入的資料值只有在 0 附近才可以使得資料值可以被傳遞至下一層，因此 BatchNormalization 在此時的用途就非常的重要，BatchNormalization 可以將資料變成 Normal Distribution，因此資料也能夠順利地傳至模型下一層。因此模型使用 Normalization 的話，可以使得模型的訓練狀況較佳。



圖十二、使用 BatchNormalization 的好處

A.5. Loss Function 的使用與說明

本次實驗分別使用到 MSE 與 Crossentropy 兩種 Loss Function 計算 Loss 值。由於這兩種 Loss function 計算 Loss 值方式的不同，因此他們要求的預測標籤與實際標籤的維度也大不相同。以本實驗的訓練資料為例，MSE 要求的預測標籤與實際標籤的維度必需為(1080,1)與(1080,1)，而 Crossentropy 要求的預測標籤維度必須為(1080,2)，實際標籤維度必須為(1080,)。

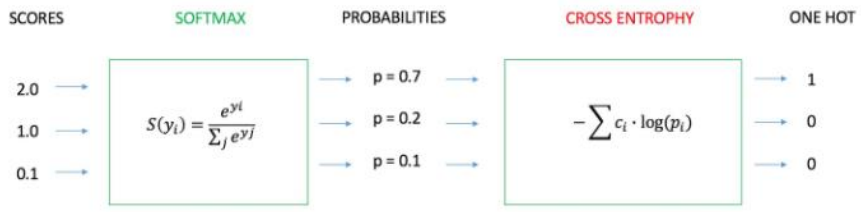
A.5.1. MSE

本實驗使用 MSE 計算 Loss 值的計算方法如**式(1)**所示，式(1)中表示出必須將預測值與實際值相減，因此這兩種標籤必須為同一個維度。本實驗引入的套件為 torch.nn.functional。因此本實驗就可以 torch.nn.functional.mse_loss(預測值,實際值)進行 Loss 值的計算。

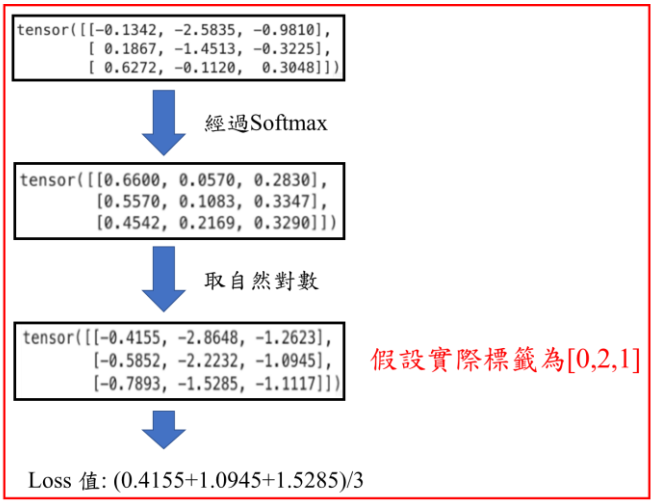
$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \qquad \textbf{式(1)}$$

A.5.2. Crossentropy

Crossentropy 計算 Loss 值的方法如**圖十三_一**所示。計算 Loss 值總共有三個步驟，第一個步驟就是將模型預測的結果經過 Softmax 行成各單列總和數值為一的情況，第二個步驟是再將各數值取自然對數並去掉負值，第三個步驟是將每列的運算結果與真實標籤對應的位置數值取出做平均。以**圖十三_二**的模型輸出結果為例，初始的模型輸出結果經過 Softmax 層之後，就會形成單列總和數值為一的情況，也就是 0.6600+0.570+0.2830=1。接著再對這些輸出結果取自然對數取絕對值。做完上述的事情之後，再依據標籤的位置(假設此處的真實標籤結果為[0,2,1])，進入各列取出該索引位置的數值，再將這些個數取平均，也就是要進行(0.4155+1.0945+1.5285)/3 的計算，計算完就能得到 Loss 值的計算結果。此處實作 Crossentropy 的套件也為 torch.nn.functional，因此以 torch.nn.functional.CrossEntropyLoss (預測值,實際值)的指令，就可進行以下三個步驟的運算。



圖十三_一、Crossentropy 的實作方法



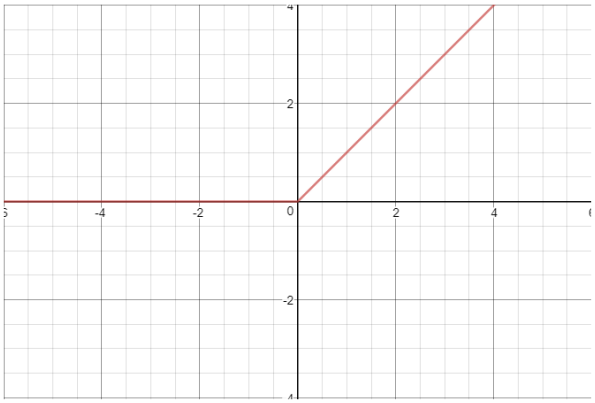
圖十三_二、Crossentropy 的算法

B. Activation function 的說明與使用

B.1. ReLU 的介紹與使用

本實驗分別在 EEGNet 與 DeepConvNet 裡面使用到 ReLU 這個 Activation function，此函數如**式(2)**所示。在**式(2)**中，他會將小於 0 的輸入變成 0，大於 0 的輸入則不變。因此模型如果在做反向傳播的話，大於 0 的輸入，其梯度就會等於 1。這樣就能減低梯度消失的發生機率，並能夠使得模型運算的速度變快，因為模型只需判斷其值是否大於 0 就好。ReLU 函數則可以對應至**圖十四**。使用時僅需呼叫 torch.nn 即可使用此 Activation function。

$$f(x)=\begin{cases}0 & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases} \qquad \textbf{式(2)}$$



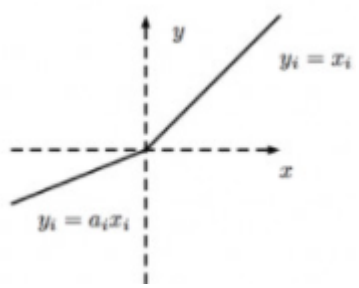
圖十四、ReLU function

B.2. LeakyReLU 的介紹與使用

本實驗分別在 EEGNet 與 DeepConvNet 裡面使用到 LeakyReLU 這個 Activation function，此函數如**式(3)**所示。可以看到**式(3)**中，原本的 ReLU function 是將所有的負數都變成 0，但此處的 LeakyReLU function 則是給予負值一個非 0 的斜率。因此模型如果在做反向傳播的話，大於 0 的輸入，其梯度就

會等於 1。但小於 0 的輸入，其梯度就會等於 alpha。LeakyReLU 函數則可以對應至**圖十五**。使用時僅同樣需呼叫 torch.nn 即可使用此 Activation function。但此處與 ReLU 不同的地方在於，需要設定 alpha。

$$f(x)=\begin{cases}\alpha x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$$ **式(3)**

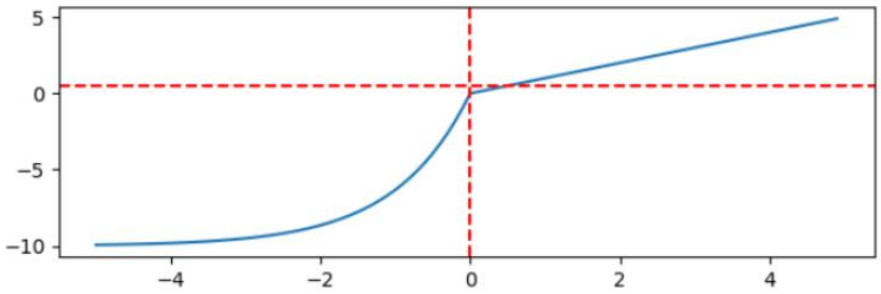


圖十五、LeakyReLU function

B.3. ELU 的介紹與使用

本實驗分別在 EEGNet 與 DeepConvNet 裡面使用到 ELU 這個 Activation function，此函數如**式(4)**所示。可以看到**式(4)**中，原本的 ReLU 是將所有的負數都變成 0，但此處的 ELU 則是將負值取自然對數再乘上一個 alpha 值。因此模型如果在做反向傳播的話，大於 0 的輸入，其梯度就會等於 1。但小於 0 的輸入，其梯度就會等於 f(x)+alpha。ELU 函數則可以對應至**圖十六**。使用時僅同樣需呼叫 torch.nn 即可使用此 Activation function。但此處與 ReLU 不同的地方在於，同樣需要設定 alpha。

$$f(x)=\begin{cases}\alpha(e^{x}-1) & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$$ **式(4)**



圖十六、ELU function

3. Results of testing

A. 最好的 testing accuracy

在兩個模型的實驗中，本實驗僅有針對 Activation function、Dropout 等兩種模型層進行調整，另外在模型外部的部份如優化器(Optimizers)、Loss function 與 Learning rate schedule 等，本實驗也有替換嘗試。**表二**為本實驗經過嘗試得出的最好的參數結果。**表三**為三種激活函數的兩種架構的最高準確率。在本次實驗中，結果最好的為 EEGNet，其測試結果可到 88.24%，使用的 Activation function 為 LeakyReLU。而 DeepConvNet 中最好的測試結果為 85.46%，其使用的 Activation function 為 ReLU。

表二、最好的模型實驗參數與方法

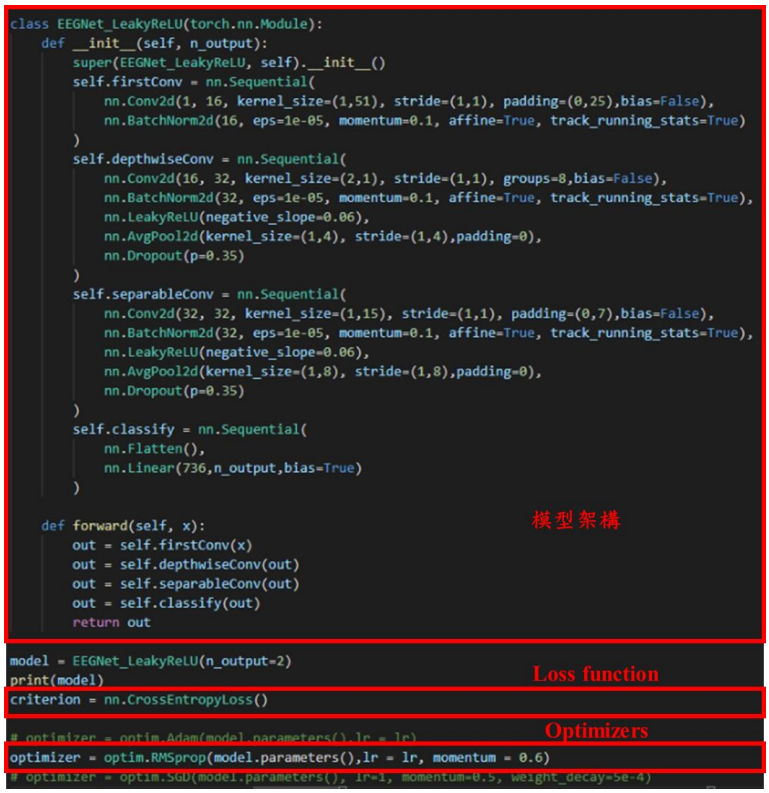
	Activation function	Dropout	Loss function	Optimizers	Training Accuracy	Testing Accuracy
EEGNet	LeakyReLU (alpha=0.06)	0.35	Crossentropy	RMSprop (lr = 1e-3, Momentum=0.6)	98.98%	88.24%
DeepConvNet	ReLU	0.40	Crossentropy	RMSprop (lr = 1e-3, Momentum=0.9, weight_decay=1e-3)	98.98%	85.46%

表三、三種激活函數的兩種架構的最高準確率

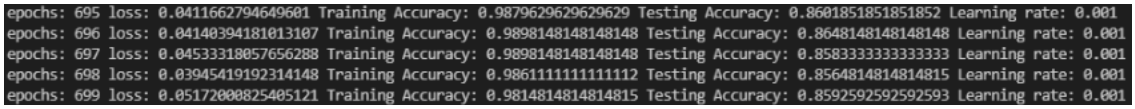
	ReLU	LeakyReLU	ELU
EEGNet	**87.12%**	**88.24%**	**87.222%**
DeepConvNet	85.46%	84.07%	83.79%

A.1. EEGNet

在此模型架構中，本實驗有設置一個機制為取得模型訓練中 Loss 值最小的模型結果，並將當下的模型權重及參數儲存起來以做測試之用。**圖十七**為模型參數配置與最好的調適方法。可以看到圖中模型參數的配置與初始模型的差異在於 Activation function 裡面的數值及 Dropout。為了使得模型能夠有多種非線性的變化，因此本實驗逐一嘗試多個 LeakyReLU 中的 Alpha 值，並搭配 Dropout 隨機關閉神經元。最終發現到 LeakyReLU 中的 alpha 數值為 0.06 並搭配 Dropout 為 0.35 的時後，將會是模型表現最佳的參數。此外本實驗也有嘗試使用 Adam 及 RMSprop，經過本實驗多次實驗的比較後發現，RMSprop 對於本實驗來說會是一個較有優勢的 Optimizer，因為 RMSprop 具有 momentum，因此當模型陷入區域最佳解的時候，模型就具有物理學動量的特性使得模型可以在多個區域最佳解中穿梭，直至找到一個全域最佳解。**圖十八**為 EEGNet 中最好方法的訓練過程(受到篇幅影響，因此只放最後幾個)。



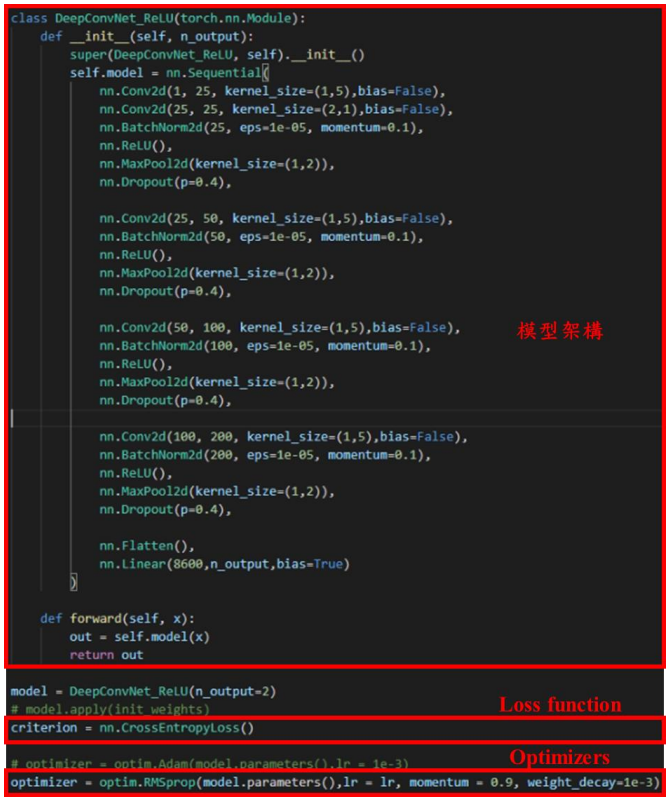
圖十七、EEGNet 中最好的模行參數配置與使用的調適方法



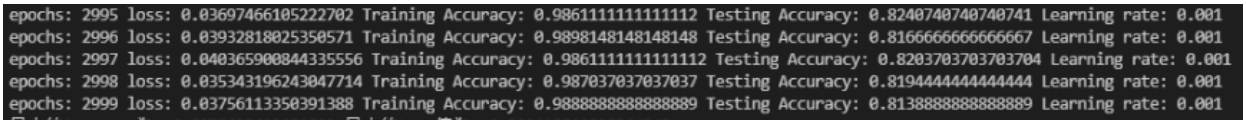
圖十八、EEGNet 中最好方法的訓練過程

A.2. DeepConvNet

在此模型架構中，本實驗同樣有設置一個機制為取得模型訓練中 Loss 值最小的模型結果，並將當下的模型權重及參數儲存起來以做測試之用。**圖十九**為模型參數配置與最好的調適方法。可以看到圖中模型參數的配置與初始模型的差異，同樣在於 Activation function 裡面的數值及 Dropout。為了使得模型能夠有多種非線性的變化，因此本實驗逐一嘗試多個 Activation function，並搭配 Dropout 隨機關閉神經元。最終發現到 ReLU 搭配 Dropout 為 0.4 的時後，將會是模型表現最佳的參數。**圖二十**為 DeepConvNet 中最好方法的訓練過程(受到篇幅影響，因此只放最後幾個)。



圖十九、DeepConvNet 中最好的模行參數配置與使用的調適方法

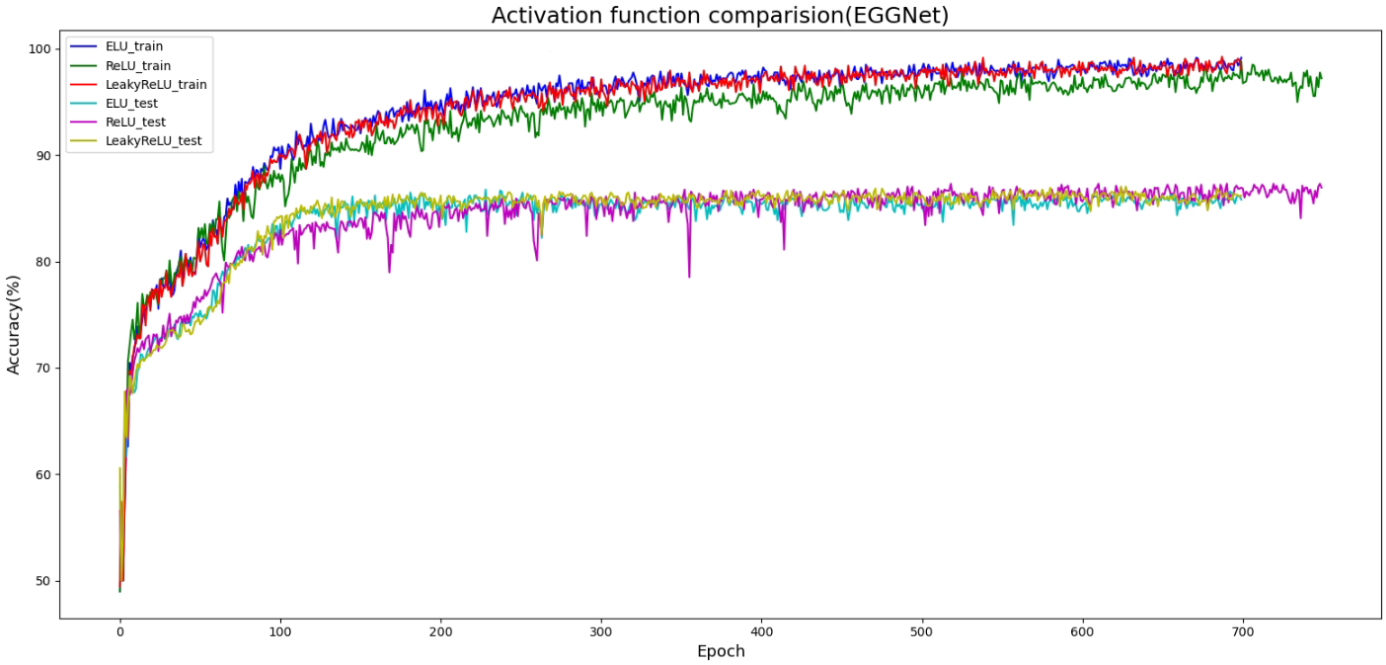


圖二十、DeepConvNet 中最好方法的訓練過程

B. 圖表比較

B.1. EEGNet 不同 Activation function 間的 Accuracy 比較

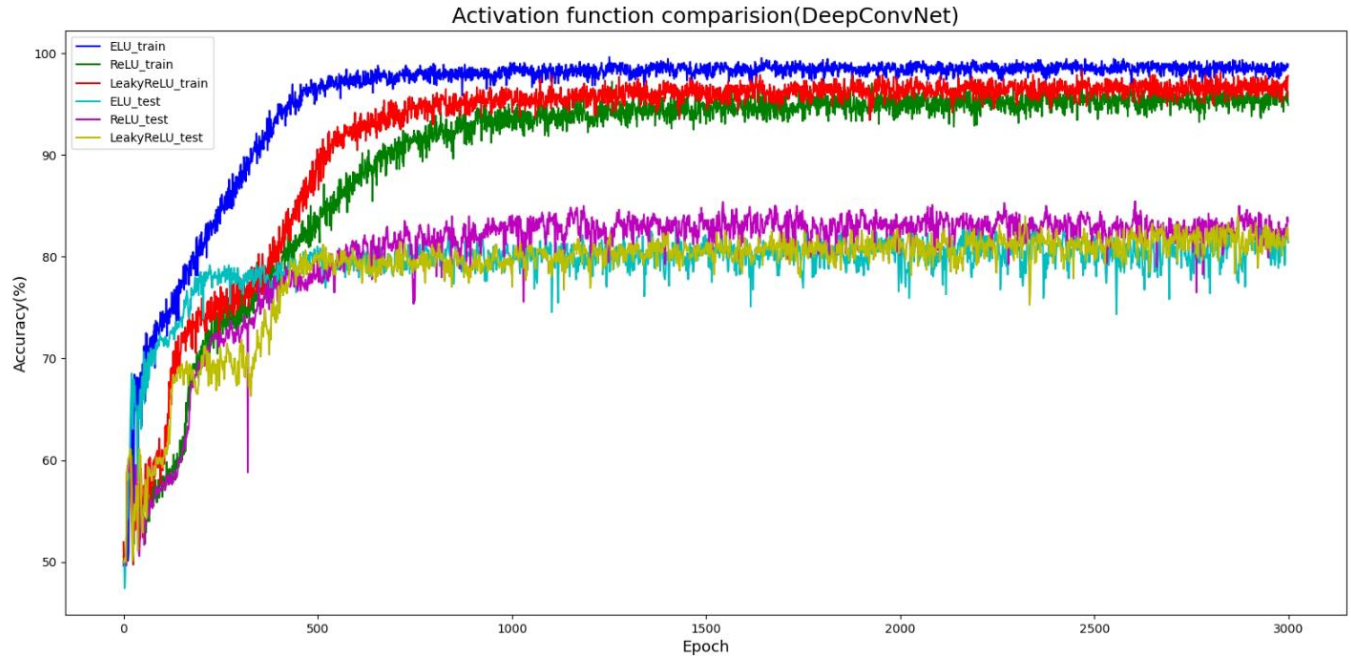
可以看到在**圖二十一**中，訓練的 epochs 數量普遍需要在大於 300 以後，Accuracy 才能趨於穩定，並且可以從圖中得知 LeakyReLU 的訓練 Accuracy 為三種 Activation function 中表現最好的。但在測試部份中，ReLU 的表現卻是三種 Activation function 中表現最好的。



圖二十一、EEGNet 中最好方法的訓練過程

B.2. DeepConvNet 不同 Activation function 間的 Accuracy 比較

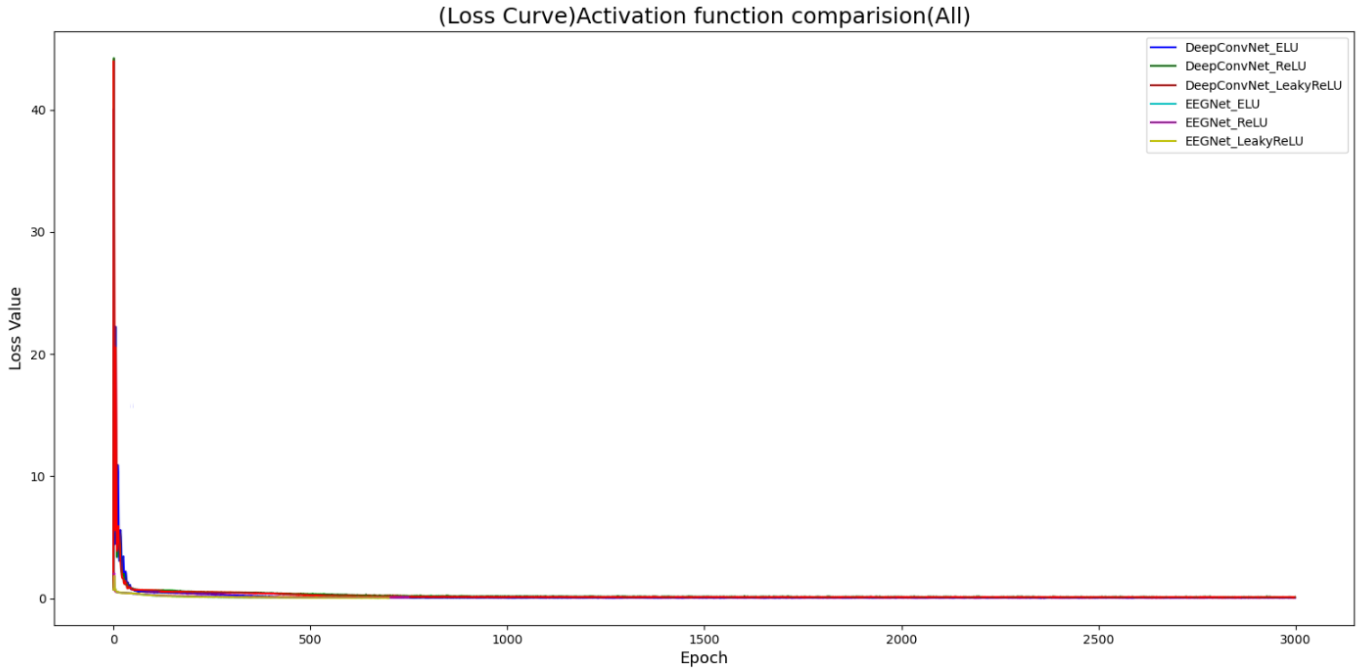
可以看到在**圖二十二**中，訓練的 epochs 數量普遍需要在大於 1000 以後，Accuracy 才能趨於穩定，並且可以從圖中得知 ELU 的訓練 Accuracy 為三種 Activation function 中表現最好的。但在測試部份中，ReLU 的表現卻是三種 Activation function 中表現最好的。相較於 EEGNet，因為 DeepConvNet 的參數較多，因此他可能會需要更多的時間去擬合模型。除此之外，由於 DeepConvNet 參數較多的特性，因此比起 EEGNet，DeepConvNet 較容易發生過度擬合的問題。



圖二十二、DeepConvNet 中最好方法的訓練過程

B.3. 全部模型方法不同 Activation function 間的訓練 Loss 比較

圖二十三為訓練過程中，訓練的 Loss 值變化曲線圖。並且在圖中，每一種模型+方法在第 500 個 epochs 之後就趨於穩定，甚至最後的模型訓練 Loss 值都可以到 0.1 以下。



圖二十三、全部模型 Loss 的訓練過程

4. Discussion

A. Loss function 之間的討論

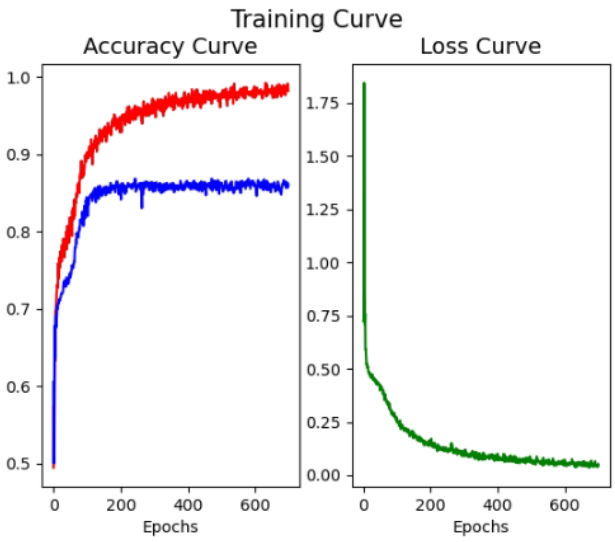
A.1. MSE 與 Crossentropy 之間的比較

在此部份的討論中，本實驗專注在討論 MSE 與 Crossentropy 之間的結果比較。因此在此部份，本實驗以最好的實驗結果，也就是透過 LeakyReLU+EEGNet 進行使用 MSE 與 Crossentropy 之間的訓練與測試 accuracy 與 loss 值討論。受到兩個 Loss function 計算 Loss 值方式的影響，目標標籤的維度必須要不一樣。**圖二十四**對應到的是使用 Crossentropy 的訓練及測試結果，圖中的紅線是訓練 Accuracy 曲線，藍線則是測試 Accuracy 曲線。**圖二十五**對應到的是使用 MSE 的訓練及測試結果，圖中的紅線是訓練 Accuracy 曲線，藍線則是測試 Accuracy 曲線。**表四**對應到的是使用 Crossentropy 或 MSE 的最好結果。結合三著的資料可以發現到，如果使用 MSE 的話，那可能模型需要擬合的時間會更多。相對的如果是使用 Crossentropy 的話，反而在將近 200 個 epochs 的時候模型就已經開始穩定了。而且以**表四**的結果來看，也可以發現使用 Crossentropy 進行這種二分類的方法評估會是比較有效的。

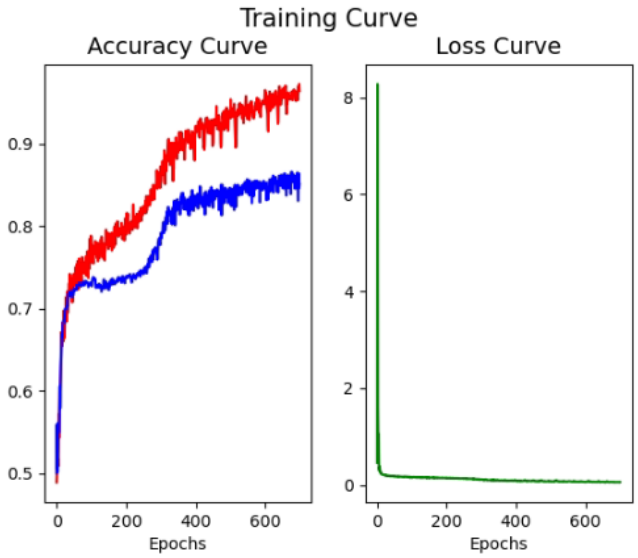
表四、使用 Crossentropy 或 MSE 的最好結果(Training Accuracy、Testing Accuracy 及 Loss)

	MSE	Crossentropy
Training Accuracy	97.22%	98.98%
Testing Accuracy	86.57%	88.24%

Loss	0.055	0.03



圖二十四、Crossentropy 的訓練測試結果



圖二十五、MSE 的訓練測試結果

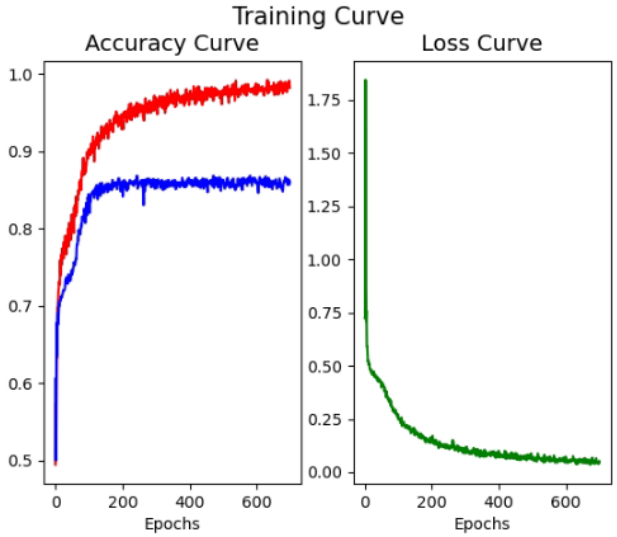
B. Optimizer 之間的討論

B.1. Adam 與 RMSprop 之間的比較

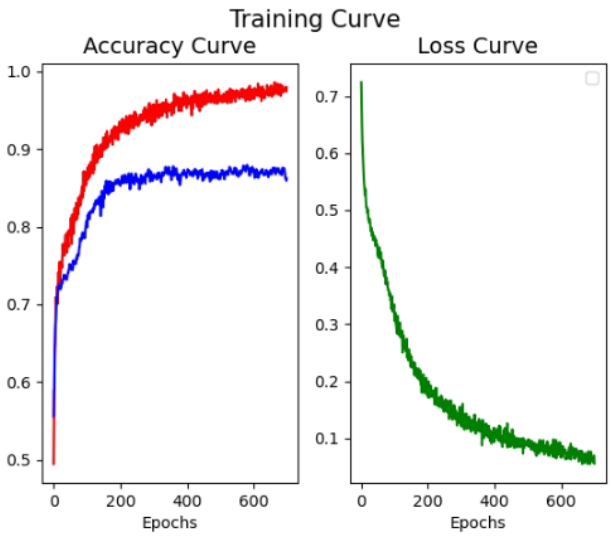
在此部份的討論中，本實驗專注在討論 Adam 與 RMSprop 之間的結果比較。因此在此部份，本實驗以最好的實驗結果，也就是透過 LeakyReLU+EEGNet 分別進行 Adam 與 RMSprop 之間的訓練與測試 Accuracy 與 Loss 值討論。**圖二十六**對應到的是使用 RMSprop 的訓練及測試結果，圖中的紅線是訓練 Accuracy 曲線，藍線則是測試 Accuracy 曲線。**圖二十七**對應到的是使用 Adam 的訓練及測試結果，圖中的紅線是訓練 Accuracy 曲線，藍線則是測試 Accuracy 曲線。**表五**對應到的是使用 Crossentropy 或 MSE 的最好結果。可以發現到 Adam 的 Testing Accuracy 與 RMSprop 的 Testing Accuracy 差不多，但 Adam 的 Testing Accuracy 卻比 RMSprop 還要少一點。其發生的原因可能是因為 RMSprop 的 momentum 是可調整的，因此 RMSprop 的泛化程度應該會比 Adam 還要高一些。

表五、使用 Adam 或 RMSprop 的最好結果(Training Accuracy、Testing Accuracy 及 Loss)

	Adam	RMSprop
Training Accuracy	98.61%	98.98%
Testing Accuracy	87.96%	88.24%
Loss	0.055	0.03



圖二十六、RMSprop 的訓練測試結果



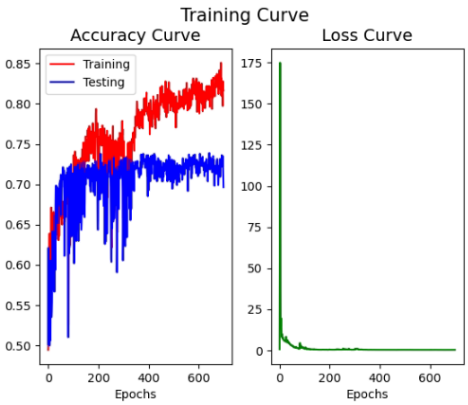
圖二十七、Adam 的訓練測試結果

B.2. RMSprop 內的 Learning rate 討論

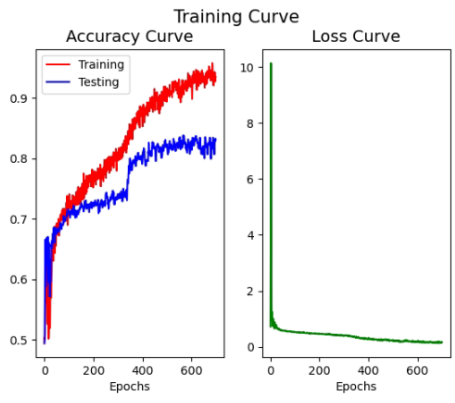
在此部份的討論中，本實驗專注在討論 Learning rate 設定之間的結果比較。因此在此部份，本實驗以最好的實驗結果，也就是透過 LeakyReLU+EEGNet 分別進行各 Learning rate 之間的訓練與測試 Accuracy 與 Loss 值討論。**圖二十八至圖三十二**對應到的是 Learning rate 數值為 1 至 1e-4 的 Accuracy 與 Loss 曲線，其中紅色的線為訓練 Accuracy 曲線，藍線則是測試 Accuracy 曲線。**表六**對應到的是個 Learning rate 最好的結果。看到這三者的資料可以發現，1e-3 為最好的 Learning rate。無論 Learning rate 最大或是最小都會使得模型的訓練效果不好。

表六、使用不同 Learning rate 的最好結果(Training Accuracy、Testing Accuracy 及 Loss)

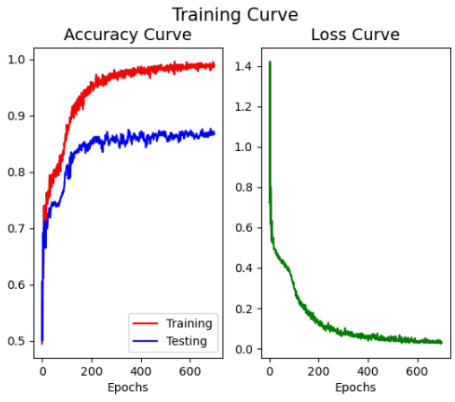
	1	0.1	0.01	0.001	0.0001
Training Accuracy	85.09%	95.74%	99.62%	98.98%	87.87%
Testing Accuracy	73.88%	83.79%	87.68%	88.24%	80.92%
Loss	0.34	0.128	0.023	0.03	0.305



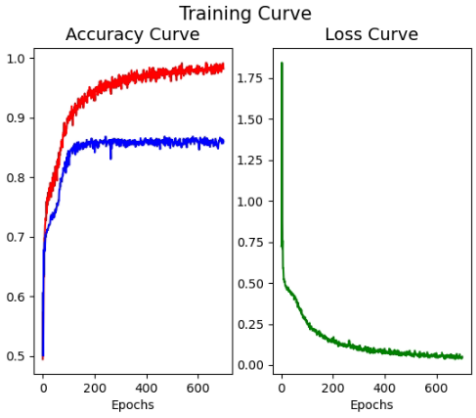
圖二十八、Learning rate=1



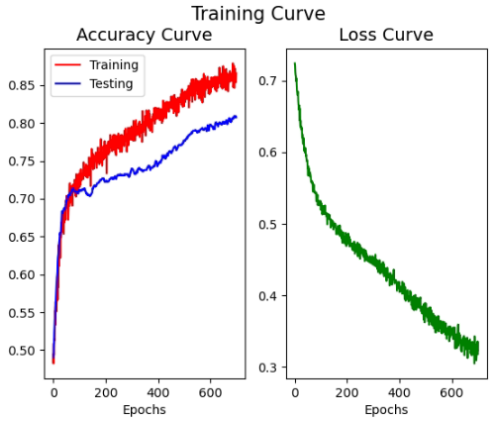
圖二十九、Learning rate=0.1



圖三十、Learning rate=1e-2



圖三十一、Learning rate=1e-3



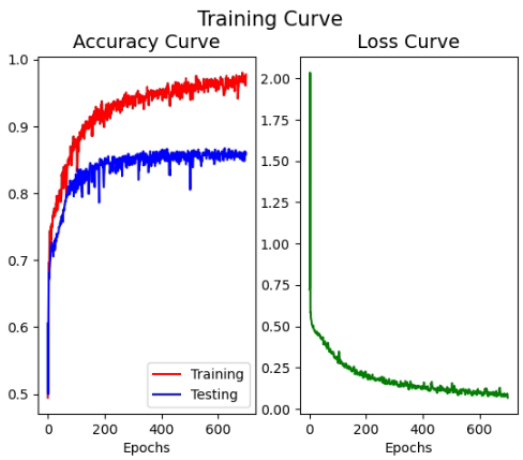
圖三十二、Learning rate=1e-4

B.3. RMSprop 內的 momentum 討論

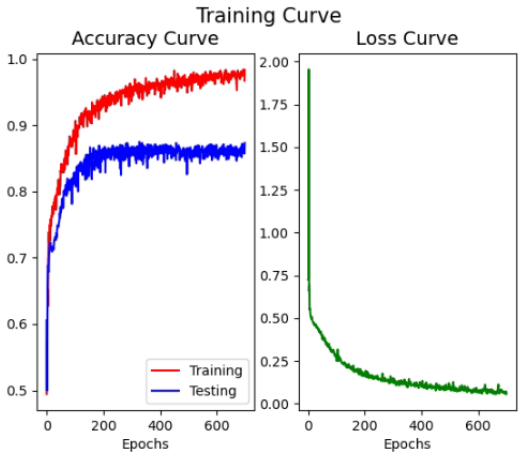
在此部份的討論中，本實驗專注在討論 momentum 設定之間的結果比較。因此在此部份，本實驗以最好的實驗結果，也就是透過 LeakyReLU+EEGNet 分別進行各 momentum 數值之間的訓練與測試 Accuracy 與 Loss 值討論。**圖二十三**至**圖三十六**對應到的是 momentum 為 0.1、0.3、0.6 及 0.9 的 Accuracy 與 Loss 曲線，其中紅色的線為訓練 Accuracy 曲線，藍線則是測試 Accuracy 曲線。**表七**對應到的是個 momentum 最好的結果。看到這三者的資料可以發現，0.6 為最好的 Learning rate。無論 momentum 最大或是最小都會使得模型的訓練效果不好。太大的 momentum 雖然可以使得模型跳脫區域最佳解，但其也有可能使得模型也跳脫全域最佳解。因此 momentum 一樣需要抓到最適合的數值。

表七、使用不同 momentum 的最好結果(Training Accuracy、Testing Accuracy 及 Loss)

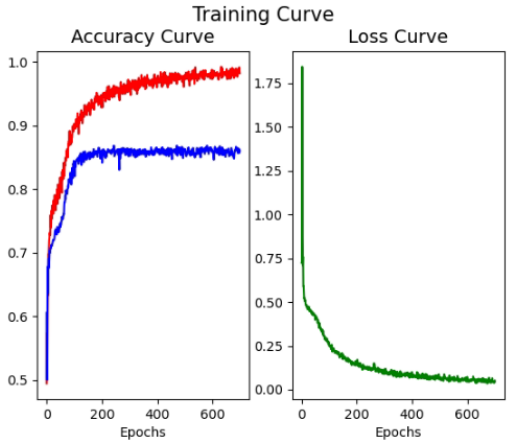
	0.1	0.3	0.6	0.9
Training Accuracy	98.05%	98.42%	98.98%	99.25%
Testing Accuracy	86.75%	87.5%	88.24%	85.92%
Loss	0.068	0.055	0.03	0.031



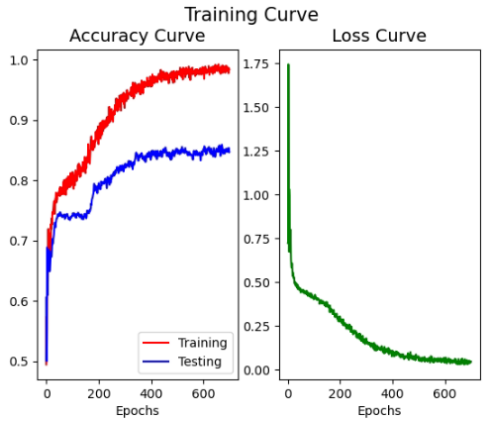
圖三十三、momentum=0.1



圖三十四、momentum=0.3



圖三十五、momentum=0.6



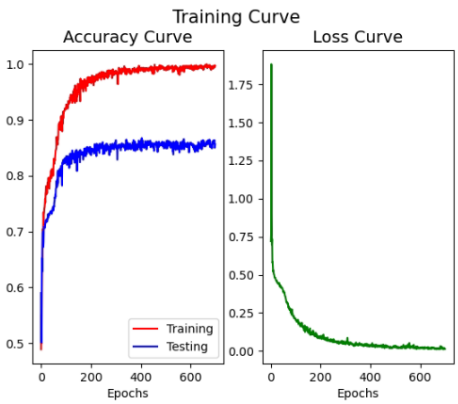
圖三十六、momentum=0.9

C. Dropout 參數設定之間的討論

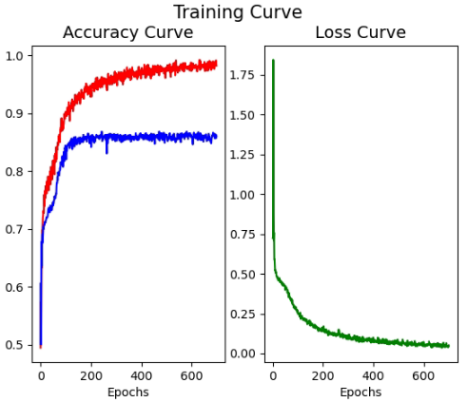
在此部份的討論中，本實驗專注在討論 Dropout 設定之間的結果比較。因此在此部份，本實驗以最好的實驗結果，也就是透過 LeakyReLU+EEGNet 分別進行各 Dropout 數值之間的訓練與測試 Accuracy 與 Loss 值討論。在此部份中，本實驗對每一層的 Dropout 都設定同樣的數值。**圖三十七**至**圖三十九**對應到的是 Learning rate 為 0.2、0.35 及 0.5 的 Accuracy 與 Loss 曲線，其中紅色的線為訓練 Accuracy 曲線，藍線則是測試 Accuracy 曲線。**表八**對應到的是個 momentum 最好的結果。看到這三者的資料可以發現，0.35 為最好的 Dropout 數值。結合三者的資料可以發現，Dropout 的數值如果設定的太低的話，模型就有可能會陷入 Overfitting 的情況。而 Dropout 的數值如果設定的太大的話，模型就有可能會使得模型的訓練 Accuracy 不好，連帶測試 Accuracy 變得不佳。

表八、使用不同 Dropout 數值的最好結果(Training Accuracy、Testing Accuracy 及 Loss)

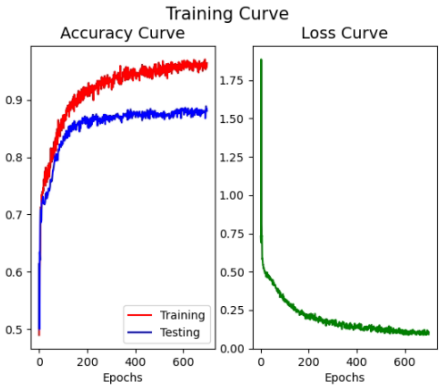
	0.2	0.35	0.5
Training Accuracy	100%	98.98%	95.46%
Testing Accuracy	86.75%	88.24%	84.53%
Loss	0.0104	0.03	0.18



圖三十七、Dropout=0.2



圖三十八、Dropout=0.35



圖三十九、Dropout=0.5

5. Reference

1. Dropout
   https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

2. BatchNormalization
   https://reurl.cc/GmvmWp
   https://mofanpy.com/tutorials/machine-learning/torch/intro-batch-normalization/

3. Crossentropy
   https://zhuanlan.zhihu.com/p/35709485