**Lab2: EEG classification**

Student id / name: A113599 / 楊淨富

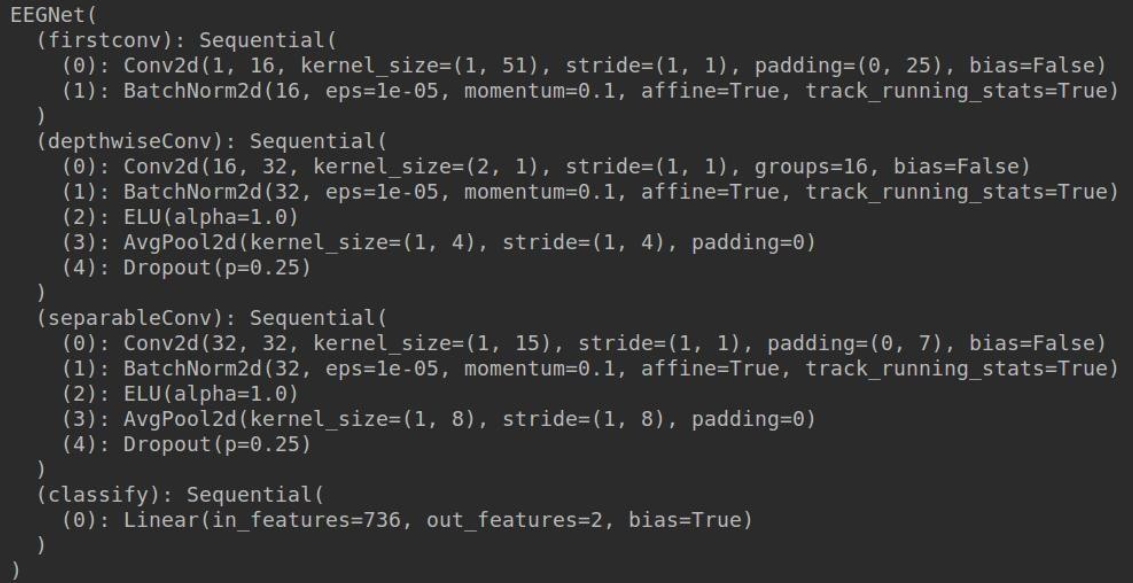
**I. Introduction:**

分別訓練兩個models，一個為EEG net，另一個為Deep convolutional net。並將BCI(brain-computer interface)腦波圖的dataset作為input輸入至兩個models，並進行2-class的classification。

**II. Experiment setups**

1. **The detail of your model**

* EEGNet



先對input data進行pre-processing，用torch.utils.data這個module的DataLoader，將train\_data, train\_label, test\_data, test\_label分別轉換成tensor後包在一起。

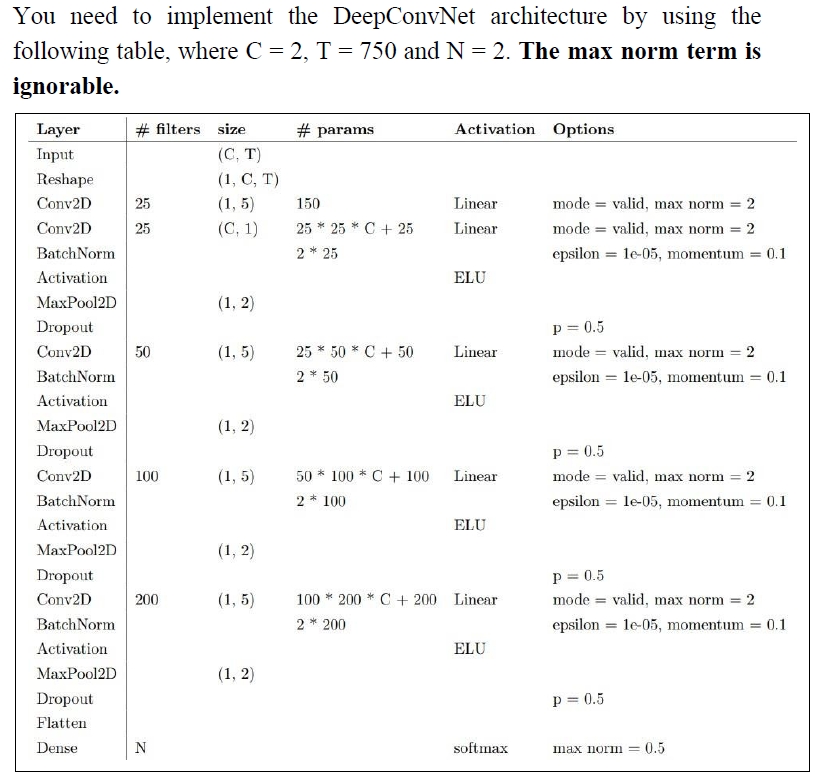
之後是network的architecture，除了input layer與output layer外，中間的hidden layer分為三層。

* + 第一層為CNN
  + 第二層為Depthwise convolution
  + 第三層為Separable convolution

而Depthwise convolution及 Separable convolution的計算是希望在不影響輸出結構的狀況下減少運算量，具體參數皆按照spec。

此外，此model有一些Hyper parameters，以下是我tune的參數:

* Batch size = 108
* Learning rate = 0.001
* Epochs = 1201
* Optimizer: torch.optim.Adam()
* Loss function: torch.nn.CrossEntropyLoss()
* DeepConvNet



Input和EEGNet一樣，都是先用DataLoader包成一包。

Network的architecture，除了input layer與output layer外，中間的hidden layer共四層，每層皆為CNN。但除了第一層內部用到了2層的CNN外，其餘第二到第四層皆使用一層的CNN，並且再透過Batch normalization、Activation funcion、Max pooling 2D和Dropout來做為該層最後的output。

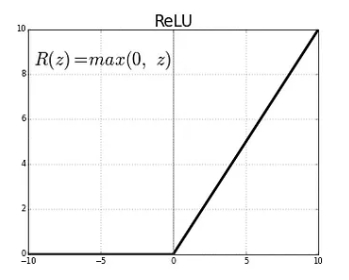
* Batch normalization:透過對每個batch進行標準化處理，有助於加速訓練過程(因為能更順利傳遞gradient從而加快收斂速度)，並提高模型的泛化性(generalization)。
* Activation function: Non-linearity
* Max pooling 2D:降維操作，透過保留最顯著的特徵並丟棄不重要的細節來減少計算量。
* Dropout: 在訓練過程中隨機地丟棄或關閉一部份neuron來防止overfitting，並且可以提高模型的泛化性(generalization)。

此外，此model有一些Hyper parameters，以下是我tune的參數:

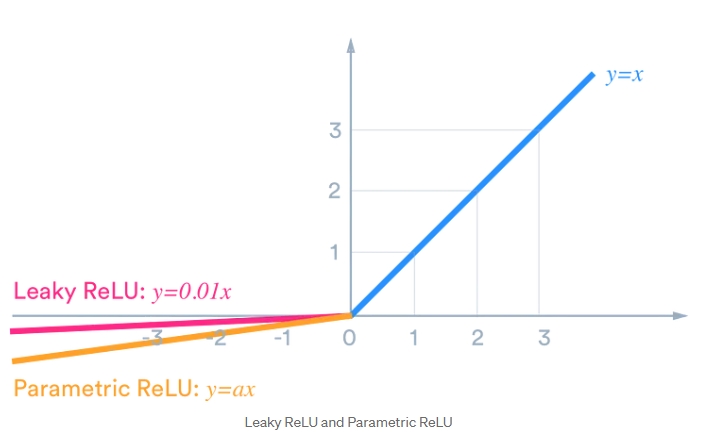
* Batch size = 1080
* Learning rate = 0.001
* Epochs = 1201
* Optimizer: torch.optim.Adam()
* Loss function: torch.nn.CrossEntropyLoss()

1. **Explain the activation function(ReLU, LeakyReLU, ELU)**

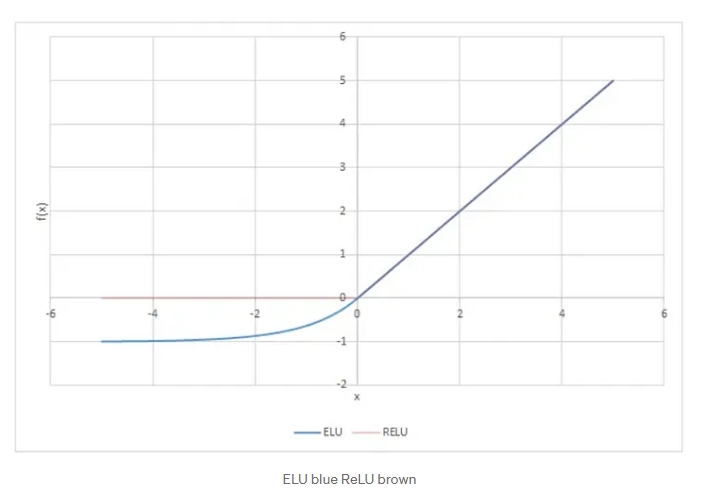
* ReLU:

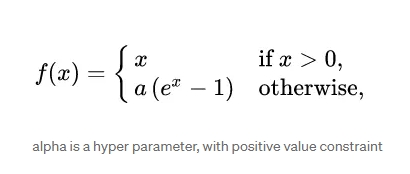


* + 如果輸入小於0，則它的輸出為0，否則輸出原始輸入。
  + 優點:
    - 避免並修正梯度消失的問題。
    - 計算的cost比tanh和sigmoid更低。
  + 缺點:
    - ReLU應該用於hidden layer，不適用於output layer。
    - 在訓練過程中，某些梯度可能會變得脆弱並消失。這可能導致權重更新，使得某些神經元永遠不再對任何數據點產生活性，即ReLu可能導致神經元失效。
    - 對於ReLu的負區域（x < 0），梯度將為0，這意味著在梯度下降過程中權重不會進行調整。這意味著處於該狀態的神經元將不再對誤差/輸入的變化做出響應（因為梯度為0，什麼都不會變化）。這稱為「dying ReLu problem」。
    - ReLU的輸出範圍是[0,∞)，這意味著它可能使activation的值爆炸。
* LeakyReLU:



* LeakyReLU是ReLU的一種變形。不同於ReLU在y < 0時輸出為0，LeakyReLU允許一個小的、非零的常數斜率α（通常為α=0.01）。然而，目前這種優勢在所有任務中是否都能一致取得更好的結果仍然不確定。
* 優點:
  + 透過具有一個小的負斜率(通常為0.01左右)來解決dying ReLU problem。
* 缺點:
  + 由於它具有線性特性，它不能用於複雜的分類任務。對於某些例子，LeakyReLU的表現不如Sigmoid和Tanh。
* ELU(Exponential LU):





* + ELU產生的結果比ReLU更準確，並且收斂速度更快。對於>0的正輸入，ELU和ReLU相同，但對於<0的負輸入，ELU緩慢地平滑到-α而ReLU則急遽平滑。
  + 優點:
    - ELU的輸出在平滑到-α時變化較慢，而ReLU的平滑程度較快。
    - 與ReLU不同，ELU可以產生負輸出。
  + 缺點:
    - 與ReLU相同，因為輸出範圍是[0,∞)，這意味著它可能使activation的值爆炸。

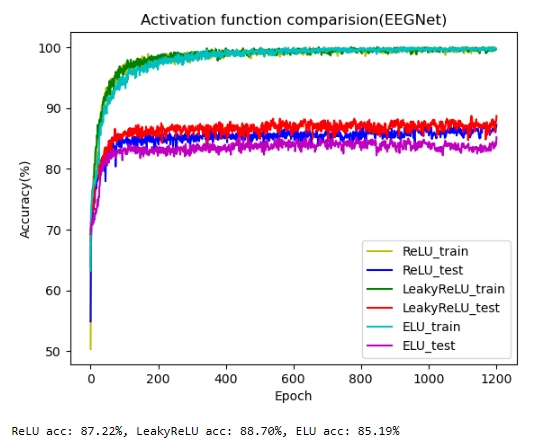
**III. Experimental results**

**A. The highest testing accuracy**

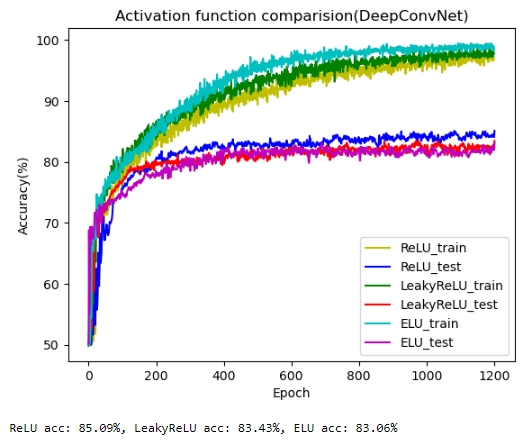
|  |  |  |
| --- | --- | --- |
|  | EEGNet | DeepConvNet |
| ReLU | 87.22% | 85.09% |
| LeakyReLU | 88.70% | 83.43% |
| ELU | 85.19% | 83.06% |

**B. Comparison figures**

* EEGNet



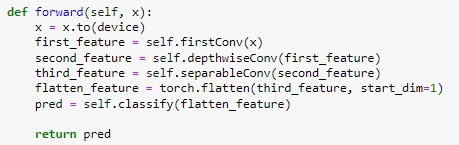
* DeepConvNet



**IV. Discussion**

**A. Anything you want to share**

1. 一開始不太會使用DataLoader()，查詢資料後才知道要把data放至TensorDataset()，然後才能夠再設定DataLoader()的dataset這個參數。此外，一開始我想說DataLoader()的batch\_size這個參數設越大越好(在GPU VRAM足夠的情況下)，因為可以整批放進去model訓練，加快訓練速度，但後來發現，如果設小一點，因為一次epoch中，更新參數的次數更多，反而讓訓練過程更穩定。除此之外，DataLoader()還有一個參數叫num\_workers，我一開始是設為我的CPU個數，即14，但發現訓練速度反而變成超慢，後來查詢資料才發現num\_workers設太高可能會導致資源競爭和調度開銷，從而導致訓練速度變慢。對於較小的dataset，設0或1就好。
2. 在feed forward的過程中，data和model都必須使用.to(device)才能放到GPU上運行。而且label必須是torch.long的type。



1. 一開始還沒存model，所以我把train跟test寫在同一個function，但後來要存model時發現這樣如果load回model，再跑一次這個function的話，等於又再經過一次train也就是參數會被多更新一次，所以把train跟test拆分出來。換言之，存好model後，重新load並再跑一次test就會是train出來的最好結果。
2. 畫圖的時候發現我存的acc都是single-element tensors，必須利用.item()這個function轉成Python scalar values，所以放進acc list時，都必須先進行轉換。舉例: 原本算出來的acc可能是tensor(5.)但經過.item()後會變成5.0。
3. 我一開始覺得epoch越大，理論上loss會越低，acc會越高，但實則不然，因為有時候可能會在最低點來回震盪。所以稍微測了epoch=1000-100000，最好實驗結果是大概設在1200左右會比較理想。
4. 我一開始是用筆電train的，費了一番工夫把cuda，anaconda，pytorch裝好後，發現train model實在太慢了，所以我就買了一台桌機，搞了一張4070(原本筆電是1060)，後來發現真的快很多XD(廢話)，以下是我去原價屋的菜單。然後買了主機，沒有螢幕好像也不行，碰巧看到原價屋有特價一台Acer的螢幕，2k 240Hz才7k，也手刀下單，用了60Hz好幾年的我一看到這畫面細膩度真的是超爽的。

