Report Spec (60%)

1. Introduction (20%)

EEG的全名是electroencephalogram，是將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。這次作業要透過Pytorch將助教給的EEGNet和DeepConv架構實做出來，透過這兩個神經網路去對這堆有2個channel的EEG data做分類，並且在這兩個Net上需要透過3種不同activation，分別是relu,leakyrelu,elu三種，共6種組合。每一種組合在計算時在每個epoch都會印出accuray來觀察training data及testing data 的accuracy是如何變化，並將結果畫成圖，且會統計每個組合中testing_data 準確率最高的以供助教評分，而讀入input資料的部份是透過助教寫好的dataloader.py讀入，再透過torch內建的DataLoader 將data跟label合併在一起，共分成train_data及test_data兩份資料 (內含label)。

2. Experiment set up (30%)

A. The detail of your model

在ptytorch上建構網路的架構可以分成下列兩個步驟:

(i)先建好整體網路架構(init設計每個網路層，forward設計如何傳輸資料):

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

先在init中建好網路的架構(包含channel, kernel size , srtide , padding...等)

再用foward實做出input資料透過在init建好的神經網路送到output的功能(包含flatten)

(ii)透過run將資料透過神經網路送到output，並計算loss function及透過optimizer更新weight:

在run中先建構criterion(用來計算loss function)及optimizer(用來更新weight)，建完後將資料送到神經網路中，透過該神經網路回傳的output來predict classification(左右手)，predict的方法是看idex[0]還是index[1]的值較大，較大的那方即是prediction。output計算出來後透過和label的比較計算出loss，再透過loss 來計算gradient，有了gradient後再透過optimizer來對weight做更新及完成網路的各項計算及更新。

**要注意的是在計算gadient之前必須加入"optimizer.zero_grad()"這行，原因是因為nn.module中預設的 gradient計算方式是accumulated的(為了給RNN計算時使用)，因此在實做一般的神經網路時必須先將gradient值清空才不會有計算上的錯誤。

照著這兩個步驟及參考助教給的spec，設計出EEGNet及DeepConv(忽略了maxnorm)。

EEGNet的神經網路架構是convolution -> depth-wise CNN -> separable CNN -> classifier。

DeepConv的神經網路較單純，前面都是convolution layer，最後再接到classifiers。

Hyper parameters: learning rate =1e-3 ,epochs過半後decay rate:0.9999 , epochs =2000 , batch_size = 64 , loss = CrossEntropy , optimizer = Adam

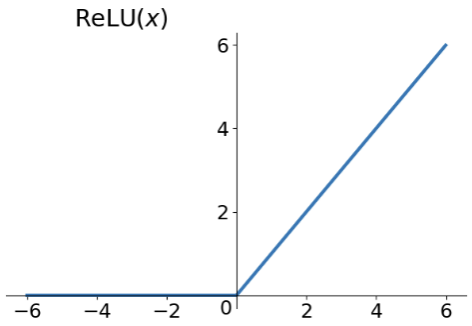
Note:main中的train 設1的時候是跑training data 並存weight，設0的時候是做validation。

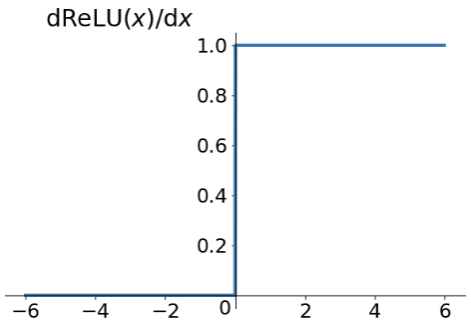
B. Explain the activation function (ReLU, Leaky ReLU, ELU)

下列是這三個activation function的數學式，及原式跟微分後的值圖。

ReLU:

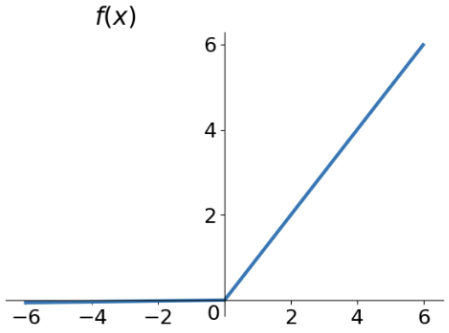
$$\mathrm{ReLU} = \max(0, x)$$

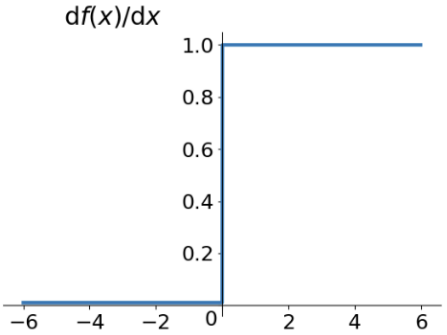




Leaky ReLU:

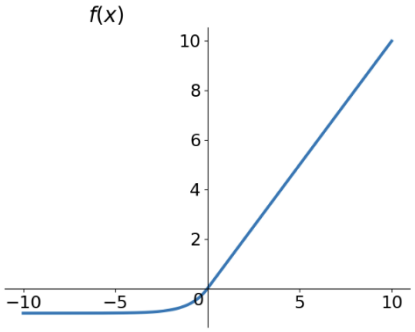
$$f(x) = \max(0.01x, x)$$

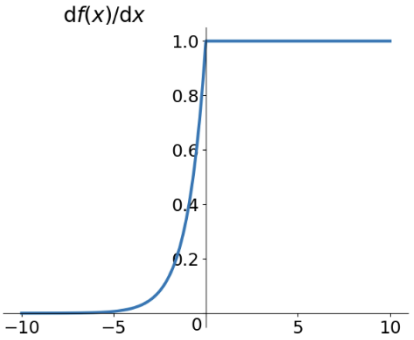




ELU:

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





可以看到在大於0的情況下三者都一樣，而小於0時三者皆不同

relu的好處是小於零的值全都當作零，這計算起來很簡便，且能讓網路更快converge

而壞處是若input全部都是負值的話那就會全部都被當作0，而無法順利學習。

leaky relu則是為了改善這個缺點，也就是讓它對負值也能夠有小比例的機會學習。

elu的話也是為了解決relu的問題而誕生的，看到微分的圖會發現relu的微分值會一瞬間從1降到0，而elu的微分值在下降時則有一段緩衝空間。

雖說看起來好像leaky 跟elu都解決了relu的問題，但實際上，這三種activation functio並沒有絕對誰比較好，會根據實做的不同，依據網路及資料而定。

3. Experimental results (30%)

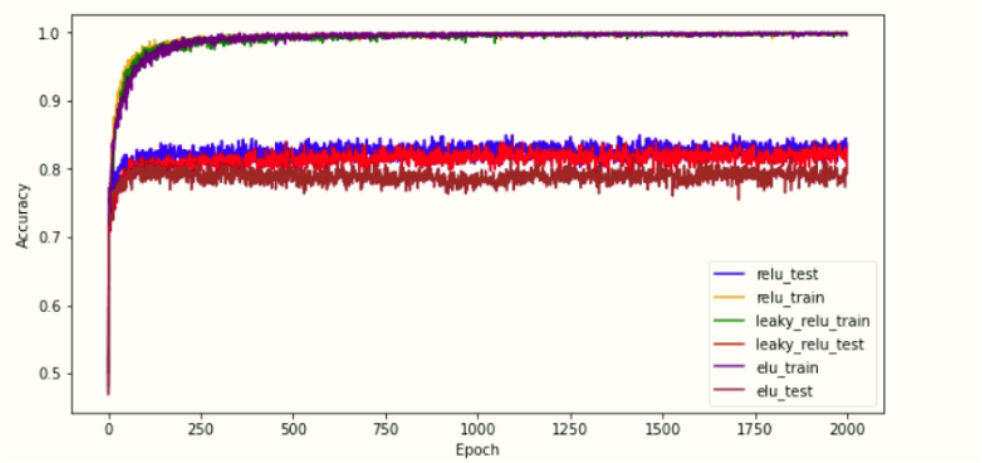
A. The highest testing accuracy

因為每次的shuffle及net computation都是具有隨機性的，因此無法保證每一次的結果都相同，正因為隨機性的緣故shuffle也有機會能在training的時候帶來比沒有shuffle過更好的結果，就算在參數都沒調整的情況下去執行同一份code也有可能產生不同的accuracy，所以在訓練的時候可以透過torch.save(net.state_dict(),path)在網路執行過程中將testing accuracy最高的參數儲存起來，如此一來可以在每次的訓練中保留最好的結果，在透過net.load_state_dict()就可以在validation時把load最好的參數，測出最高的accuracy，得到下圖的結果。

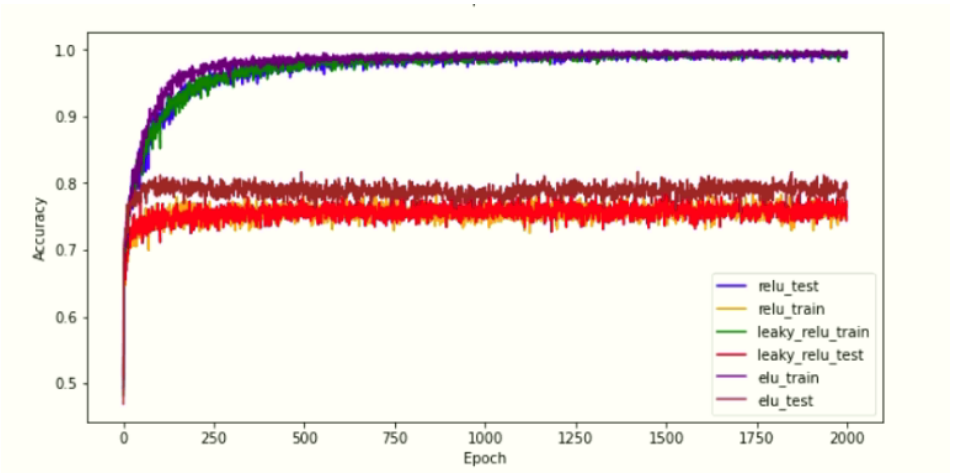
```
pytorch device:  cuda
load CKPT/EEG_ReLU_final_ckpt
Max_accuracy =  0.8787037037037037
load CKPT/EEG_LeakyReLU_final_ckpt
Max_accuracy =  0.8703703703703703
load CKPT/EEG_ELU_final_ckpt
Max_accuracy =  0.8138888888888889
load CKPT/DeepConv_ReLU_final_ckpt
Max_accuracy =  0.8111111111111111
load CKPT/DeepConv_LeakyReLU_final_ckpt
Max_accuracy =  0.7935185185185185
load CKPT/DeepConv_ReLU_final_ckpt
Max_accuracy =  0.7472222222222222
```

B. Comparison figures

EEGNet:



DeepConv:



上方的圖為EEGNet的Acuuracy，下方的圖是DeepConv的，可以發現EEG網路整體的表現比DeepConv網路的表現來的好一些，我推估是因為data正好是EEG類型的data的緣故，造成EEGNet的表現較好。從圖上可以看出EEG使用ELU作為activation的效果是比較差的，而DeepConv則相反，使用ELU的時候效果比其他兩個來的好。

4. Discussion (20%)

在這次的作業有發現幾點有趣的地方:

1.透過shuffle亂序後，能夠有機會產生比沒有shuffle過的data更好的結果，且大部分的結 果都是shuffle過後會較好。

2.在神經網路中需要透過optimizer.zero_grad()清除gradient，原因是因為在nn.module中預設gradient值是可以累加的，這個功能主要是給RNN使用，若非RNN的network需要加入這行才不會誤算gradient。

3.透過learning rate decay可以避免learning rate大小固定帶來的缺點，在多次的執行程式中有發現確實有時候會在epochs過半後(也就是開始decay時)的情況下達到更好的收斂結果。

4.在調整epochs的時候發現，由於這次的資料量不大的緣故，基本上超過2000個epochs就會讓training accuracy達到100%，但可以發現通常training到100%的神經網路在預測testing data上就不會再改進了，這是因為overfitting的關係，可以透過調整其他的參數，例如learning rate decay, dropout rate等等去做改善。