LAB2. EEG classification

陽明 大一大二不分系二年級 10612012 張凱博

(一) Introduction

Convolutionary Neuron Network 簡稱 CNN，中文名稱為「卷積類神經網絡」，是受到人類大腦視覺皮層在識別物體時是如何工作的方法，所引發的方法，所引發的一系列模型。

CNN 這類的類神經網絡被視為，被視為特徵搜尋引擎，藉由輸入 hyperparameter: filter size、padding size 和 stride 後，進行卷積(convolution)，前面幾層擷取低階特徵，最後會層層結合形成高階特徵，來建立所謂的 feature hierarchy。 在每一次 convolution 完後就會接一個 pooling layer，對一塊塊數值化過後的 data 不重疊地取他們的最大或是平均運算，最後進行 dropout 避免 model 在 training 時過度依賴某一個特定的 neuron。

這次 lab 的主題是以 Electroencephalogram(腦電圖)為 data 建構 CNN model 進行 EEG 的分類，屬於影像識別的一種。

(二)Experiment set up

A. The detail of model

1. EEGNet

在我的 EEGNet model 裡，共有 3 層 convolutional layer，每層 model 利用 torch.nn.sequential()函數建置。

(1) 第一層 layer (firstconv)

用 nn.Conv2d()進行 convolution，input channel 為 1，output channel 為 16，kernel_size 長寬為 1, 51，填補模式為 same padding，接下來用 BatchNorm2d 進行 normalization。

(2) 第二層 layer (depthwiseConv)

用 nn.Conv2d()進行 convolution，input channel 為 16，output channel 為 32，kernel_size 長寬為 2, 1，填補模式為 same padding，接下來用 BatchNorm2d 進行 normalization，再用 activation function 引入非線性，再用 average pooling layer 做 subsampling，最後在用 0.25 的機率隨機 dropout neuron。

(3) 第三層 layer (separableConv)

用 nn.Conv2d()進行 convolution，input channel 和 output channel 都是 32，kernel_size 長寬為 1, 15，填補模式為 same padding，用 BatchNorm2d 進行 normalization，再用 activation function 引入非線性，再用 average pooling layer 做 subsampling，最後在用 0.25 的機率隨機 dropout neuron。

(4) 第四層 layer (classify)

將 736 個 input feature 用線性轉換成 2 種類別(out put feature)

```
class EEGNet(nn.Module):
    def __init__(self, activation):
        super(EEGNet, self).__init__()
        activations = nn.ModuleDict([['LeakyReLU', nn.LeakyReLU()],
                                     ['ReLU', nn.ReLU()],
                                     ['ELU', nn.ELU()]
                                    ])

        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),
            activations[activation],
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout(p=0.25)
        )
        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),
            activations[activation],
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout(p=0.25)
        )
        self.classify = nn.Sequential(
            nn.Linear(in_features=736, out_features=2, bias=True)
        )
    def forward(self, x):
        out = self.firstconv(x)
        out = self.depthwiseConv(out)
        out = self.separableConv(out)
        out = out.view(out.size(0), -1) # flatten the output of conv2 to (batch_size, 736)
        out = self.classify(out)
        return out
```

2. DeepConvNet

在我的 EEGNet model 裡，共有 3 層 convolutional layer，每層 model 利用 torch.nn.sequential()函數建置。

(1) 第一層 layer (firstconv)

用 2 層 nn.Conv2d()進行 convolution，input channel 為 1，output channel 為 25，kernel_size 長寬為 1, 51，填補模式為 valid padding，用 BatchNorm2d 進行 normalization，再用 activation function 引入非線性，再用 maxium pooling layer 做 subsampling，最後在用 0. 5 的機率隨機 dropout neuron。

(2)第二層 layer (secondlayer)

用 nn.Conv2d()進行 convolution，input channel 為 25，output channel 為 50，kernel_size 長寬為 1, 5，填補模式為 valid padding，接下來用 BatchNorm2d 進行 normalization，再用 activation function 引入非線性，再用 average pooling layer 做 subsampling，最後在用 0. 5 的機率隨機 dropout neuron。

(3)第三層 layer (thirdlayer)

用 nn.Conv2d()進行 convolution，input channel 為 50，output channel 為 100，kernel_size 長寬為 1, 5，填補模式為 valid padding，接下來用 BatchNorm2d 進行 normalization，再用 activation function 引入非線性，再用 average pooling layer 做 subsampling，最後在用 0.5 的機率隨機 dropout neuron。

(4) 第四層 layer (forthlayer)

用 nn.Conv2d()進行 convolution，input channel 為 100，output channel 為 200，kernel_size 長寬為 1, 5，填補模式為 valid padding，接下來用 BatchNorm2d 進行 normalization，再用 activation function 引入非線性，再用 average pooling layer 做 subsampling，最後在用 0.5 的機率隨機 dropout neuron。

(5) 第四層 layer (classify)

將 200*1*358 個 input feature 用線性轉換成 2 種類別(out put feature)

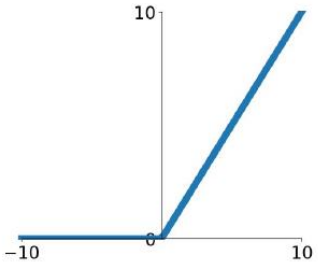
```
class DeepConvNet(nn.Module):
    def __init__(self, activation):
        super(DeepConvNet, self).__init__()
        activations = nn.ModuleDict([['LeakyReLU', nn.LeakyReLU()],
                                     ['ReLU', nn.ReLU()],
                                     ['ELU', nn.ELU()]
                                    ])

        self.firstlayer = nn.Sequential(# (1, 2, 750)
            nn.Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=False), # (25, 2, 746)
            nn.Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), padding=(0, 0), bias=False), # (25, 1, 746)
            nn.BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            activations[activation],
            nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0),  # (25, 1, 373)
            nn.Dropout(p=0.5)
        )
        self.secondlayer = nn.Sequential(# (25, 1, 373)
            nn.Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), bias=False), # (50, 1, 369)
            nn.BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            activations[activation],
            nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 1), padding=0), # (50, 1, 368)
            nn.Dropout(p=0.5)
        )
        self.thirdlayer = nn.Sequential( # (50, 1, 368)
            nn.Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1), bias=False), # (100, 1, 364)
            nn.BatchNorm2d(100, eps=1e-05, momentum=0.1),
            activations[activation],
            nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 1), padding=0), # (100, 1, 363)
            nn.Dropout(p=0.5)
```

```
        self.forthlayer = nn.Sequential( # (100, 1, 363)
            nn.Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1), bias=False), # (200, 1, 359)
            nn.BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            activations[activation],
            nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 1), padding=0), # (200, 1, 358)
            nn.Dropout(p=0.5)
        )
        self.classify = nn.Sequential(
            nn.Linear(in_features=200*1*358, out_features=2, bias=True)
        )
    def forward(self, x):
        out = self.firstlayer(x)
        out = self.secondlayer(out)
        out = self.thirdlayer(out)
        out = self.forthlayer(out) # (batch, 200, 1, 358)
        out = out.view(out.size(0), -1) # flatten the output of conv2 to (batch_size, 200*1*358)
        out = self.classify(out)
        return out
```

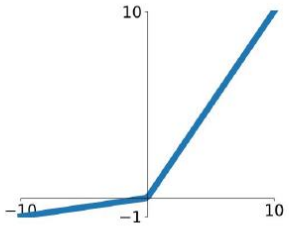
B. Explain the activation function (ReLU, Leaky ReLU, ELU)

1. ReLU



在 CNN 中常用的，對負數直接置為零，對正數利用 f(x) = x 原樣輸出，在初始化的時候通常會設稍大於零的數字以免用 random 的參數落在負數區域而不被活化，因不用 exponential 而 training 速度較快。

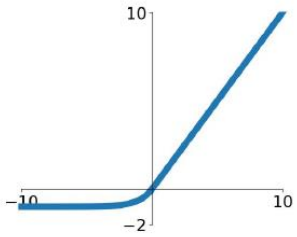
2. Leaky ReLU



F(x) = max(0.01, x)

為了解決一開始參數在初始化而無法被 activate 的過程，在負數區域使 y = 0.01*x，在正數區域使 y = x。

3. (Exponential Linear Units) ELU



若 x>0，f(x) = x，反之當 x 小於等於 0，f(x) = (alpha)*(exp(x)-1)。可以說是綜合 relu 和 LeakyReLU 兩個 activation function 的優點，既不會因為在初始化的時候陷入 f(x) = 0 的窘境，又因為在負數區域有漸進線，所以可以對 Noisy 有 tolerance。但缺點是因為有 exponential，所以計算量較大。

圖源: https://blog.csdn.net/edogawachia/article/details/80043673

(三) Experimental results

A. The highest testing accuracy

1. Screenshot with two models

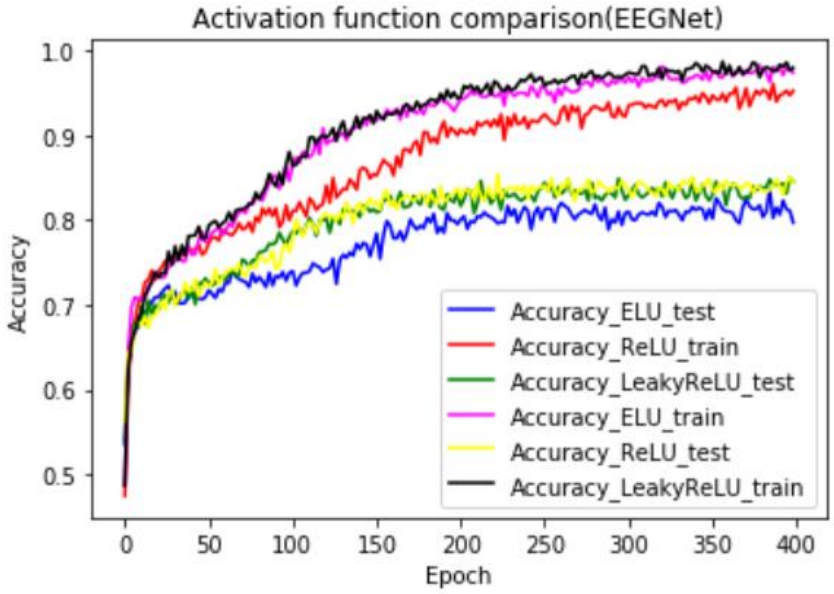
```
(1) EEGNet_ELU_MAX_accuracy_test_result: 0.837037037037037
(2) EEGNet_ReLU_MAX_accuracy_test_result: 0.85
(3) EEGNet_LeakyReLU_MAX_accuracy_test_result: 0.8546296296296296
================================================================================
(4) DeepConvNet_ELU_MAX_accuracy_test_result: 0.7703703703703704
(5) DeepConvNet_ReLU_MAX_accuracy_test_result: 0.7694444444444445
(6) DeepConvNet_LeakyReLU_MAX_accuracy_test_result: 0.7722222222222223
```

2. Anything I want to present

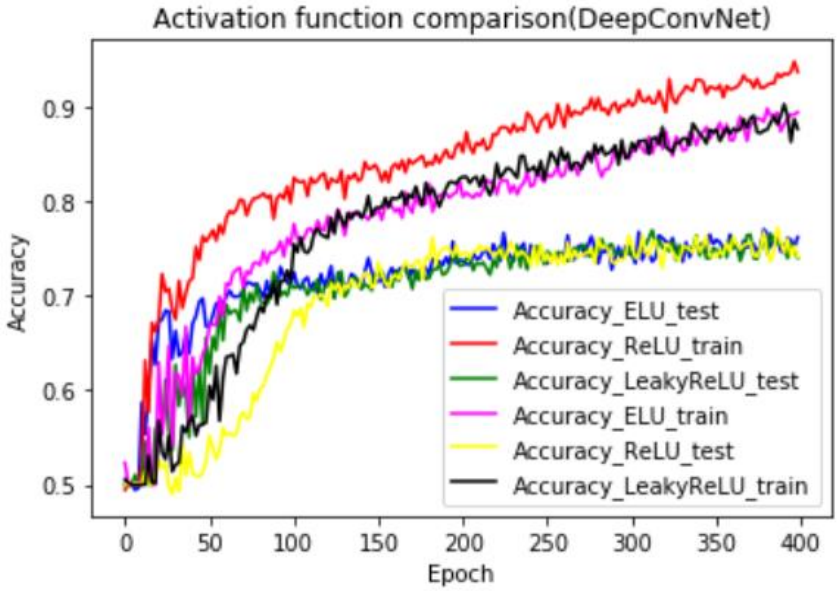
就預期結果而言，在同一種 model 底下不同的 activation function，如剛剛描素不同的 activation function 而言，LeakyRelu 的 accuracy 的 test result 的結果是最好的，而 ELU 和 Relu 的不一定可以比較得出來，有可能是 model 的 variance 或者是外部假設的 hyperparameter 不同而有所差別。

B. Comparison figures

1. EEGNet



2. DeepConvNet



(四) Discussion

這次 Lab，最初令我難以理解的是 CNN 的架構，因為是第一次實作，所以也和之前一樣做了很久，一開始是從 Pytorch 實作 CNN 的 tutorial 開始一步步看 sample 做的，到後來了解 tutorial 開始實作後，發現我對助教給的 data 完全是一無所知，在打不開 npz 檔的窘境之下，不得已我又參考了另一篇的 sample code，把 CNN 依次送入的 data batch 和 channel 搞清楚，最後在調一下 hyperparameter 才實做出來。

一開始我按照助教給定的 table 去實作，發現 6 個 accuracy 中，最高的大約是 83%，但是我把送進去的 batch size 改成 16、learning rate 改成 10^(-3)、num epochs 改成 400 之後，發現最高的 accuracy 有到 85.43，但是若將 batch size 改成 8，最高的 accuracy 卻只有約 80%，研判可能是 overfitting 的問題。