Deep Learning Lab3

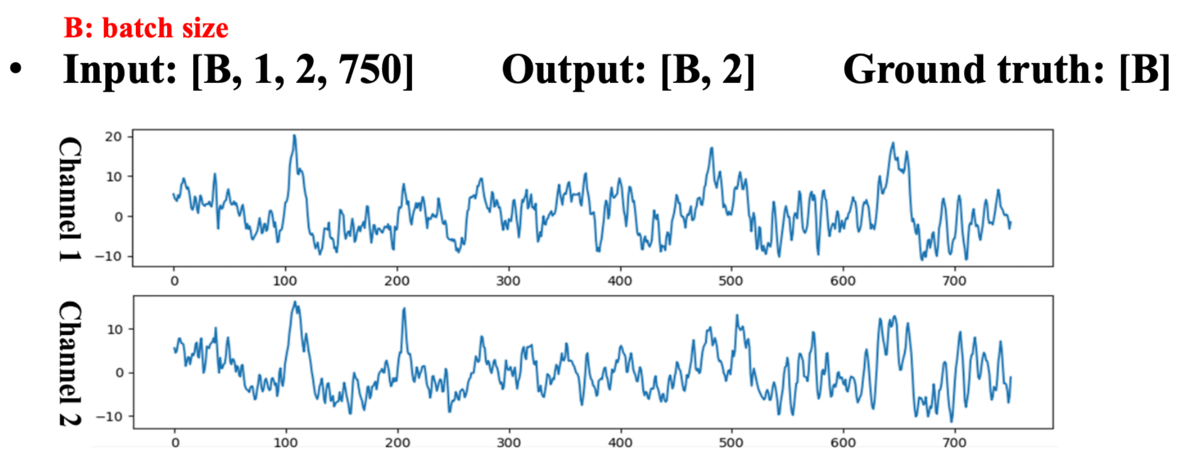
周睦鈞

311553060

1. Introduction

這次作業要利用 Pytorch 去 implement 兩種 Network , 分別是 EEGNet 跟 DeepConvNet , 用這兩種模型來做分類問題

其中訓練資料為 BCI competition (C = 1, H = 2, W = 750), 同時測試不同 activation function 對結果的影響程度



2. Experiment set up

a. The detail of your model

a-1. EEGNet

```python
import torch.nn as nn

class EEGNet(nn.Module):
    def __init__(self, activation, dropout):
        super().__init__()

        self.first_conv = nn.Sequential(
            nn.Conv2d(
                in_channels = 1,
                out_channels = 16,
                kernel_size = (1, 51),
                stride = (1, 1),
                padding = (0, 25),
                bias = False
            ),
            nn.BatchNorm2d(16, eps=1e-5, momentum=0.1, affine=True, track_running_stats = True)
        )
```

```python
        self.deep_wise_conv = nn.Sequential(
            nn.Conv2d(
                in_channels = 16,
                out_channels = 32,
                kernel_size = (2, 1),
                stride = (1, 1),
                groups = 16,
                bias = False
            ),
            nn.BatchNorm2d(32, eps = 1e-5, momentum = 0.1, affine = True, track_running_stats = True),
            activation,
            nn.AvgPool2d(kernel_size = (1, 4), stride = (1, 4), padding = 0),
            nn.Dropout(p = dropout)
        )
```

```python
        self.seperable_conv = nn.Sequential(
            nn.Conv2d(
                in_channels = 32,
                out_channels = 32,
                kernel_size = (1, 15),
                stride = (1, 1),
                padding = (0, 7),
                bias = False
            ),
            nn.BatchNorm2d(32, eps = 1e-5, momentum = 0.1, affine = True, track_running_stats = True),
            activation,
            nn.AvgPool2d(kernel_size = (1, 8), stride=(1, 8), padding=0),
            nn.Dropout(p = dropout)
        )

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

```python
    def forward(self, input):
        first_conv_out = self.first_conv(input)
        deep_wise_conv_out = self.deep_wise_conv(first_conv_out)
        separable_conv_out = self.seperable_conv(deep_wise_conv_out)
        out = self.classify(separable_conv_out)

        return out
```

1. 左邊的程式碼為 EEGNet 的 implementation， 主要由四個大 module 組成，分別是 first_conv ， deep_wise_conv ， seperable_conv ，以及 classify
2. forward 函數會將 input 分別經過上面提及過的 module 得到最終的 output , 由於我們在計算 loss 的時候使用 nn.CrossEntropyLoss() , 他在計算的時候會幫我們算 softmax 所以我們不需要再過一個 softmax 函數

a-2. DeepConvNet

```python
import torch.nn as nn

class DeepConvNet(nn.Module):
    def __init__(self, activation, dropout, channels = [25, 50, 100, 200]):
        super().__init__()
        self.channels = channels

        self.conv_0 = nn.Sequential(
            nn.Conv2d(
                in_channels = 1,
                out_channels = self.channels[0],
                kernel_size = (1, 5),
                bias = False
            ),

            nn.Conv2d(
                in_channels = self.channels[0],
                out_channels = self.channels[0],
                kernel_size = (2, 1),
                bias = False
            ),

            nn.BatchNorm2d(self.channels[0]),
            activation,
            nn.MaxPool2d(kernel_size = (1, 2)),
            nn.Dropout(p = dropout)
```

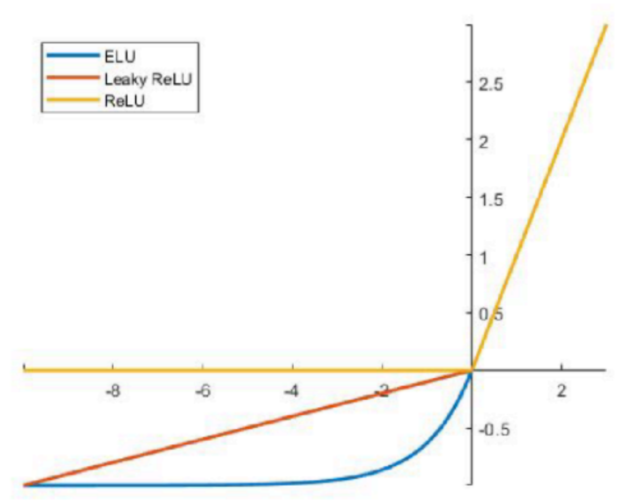
```python
        )

        for idx, channel in enumerate(self.channels[:-1], start = 1):
            setattr(self, f'conv_{idx}',
                    nn.Sequential(
                        nn.Conv2d(
                            in_channels = channel,
                            out_channels = self.channels[idx],
                            kernel_size = (1, 5),
                            bias = False
                        ),
                        nn.BatchNorm2d(self.channels[idx], eps = 1e-5, momentum = 0.1),
                        activation,
                        nn.MaxPool2d(kernel_size = (1, 2)),
                        nn.Dropout(p = dropout)
                    )
            )

        self.classify = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features = 8600, out_features = 2)
        )
```

```python
    def forward(self, input):
        out = input
        for idx in range(len(self.channels)):
            out = getattr(self, f'conv_{idx}')(out)

        return self.classify(out)
```

左邊為 DeepConvNet 的 implementation

由各種的 Conv2D, BatchNorm, Activation, MaxPooling, Dropout 組成 , 也因為後半部分的網路架構大致上一樣，因此我們這裡使用 for 迴圈搭配 setattr 來避免減短程式碼

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



b-1. ReLU:

會將負數的結果都變成 0 , 正數則保持不變 , 強制結果最後都要大於等於 0 , 這種 activation 的缺點是當 value 小於 0 的後在 backpropagation 就不會更新，因為 gradient 是 0

$$ReLU(x) = max(0, x)$$

b-2. Leaky_ReLU:

希望改善 ReLU 的缺點，因此在負數的時候還是會給予少部分的值，因此做法是在負數的地方乘以一個很小的值，讓該值會接近 0 但是又不會等於 0

$$LeakyReLU(x) = \begin{cases} x, & \text{if x > 0} \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$

b-3. ELU:

在負數的地方使用 exponential 讓 0 附近更為平滑，而且也同樣有負數的值

$$ELU(x) = max(0, x) + min(0, \alpha * (exp(x) - 1))$$

3. Experimental results

a. The highest testing accuracy

a-1. EEGNet

```
EEGNet_ReLU: 79.63%
EEGNet_Leaky_ReLU: 74.44%
EEGNet_ELU: 75.00%
```

在右方的參數下可以達到最高的 Testing Accuracy

```python
def parse_argument():
    parser = ArgumentParser(description = 'EEGNet and DeepConvNet')
    parser.add_argument('--epochs', default = 300, help = 'number of epoch')
    parser.add_argument('--lr', default = 0.005, help = 'learning rate')
    parser.add_argument('--dropout', default = 0.5, help = 'Dropout')
    parser.add_argument('--batch', default = 1024, help = 'Batch size')
    parser.add_argument('--savepath', default = './', help = 'Save path')
    parser.add_argument('--optimizer', default = torch.optim.Adam, help = 'optimizer function')
    parser.add_argument('--weight_decay', default = 0.01, help = 'Weight decay')

    return parser.parse_args()
```

a-2. DeepConvNet

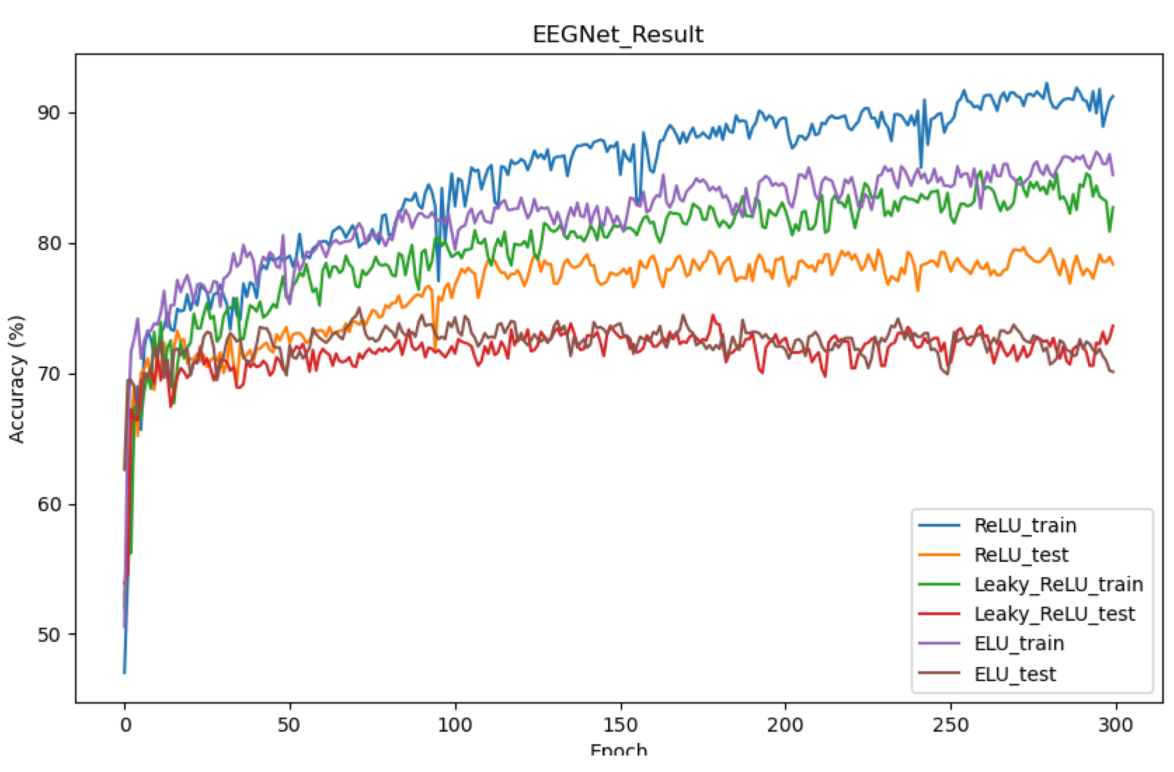
```
DeepConvNet_ReLU: 81.11%
DeepConvNet_Leaky_ReLU: 80.56%
DeepConvNet_ELU: 83.52%
```

在右方的參數下可以達到最高的 Testing Accuracy

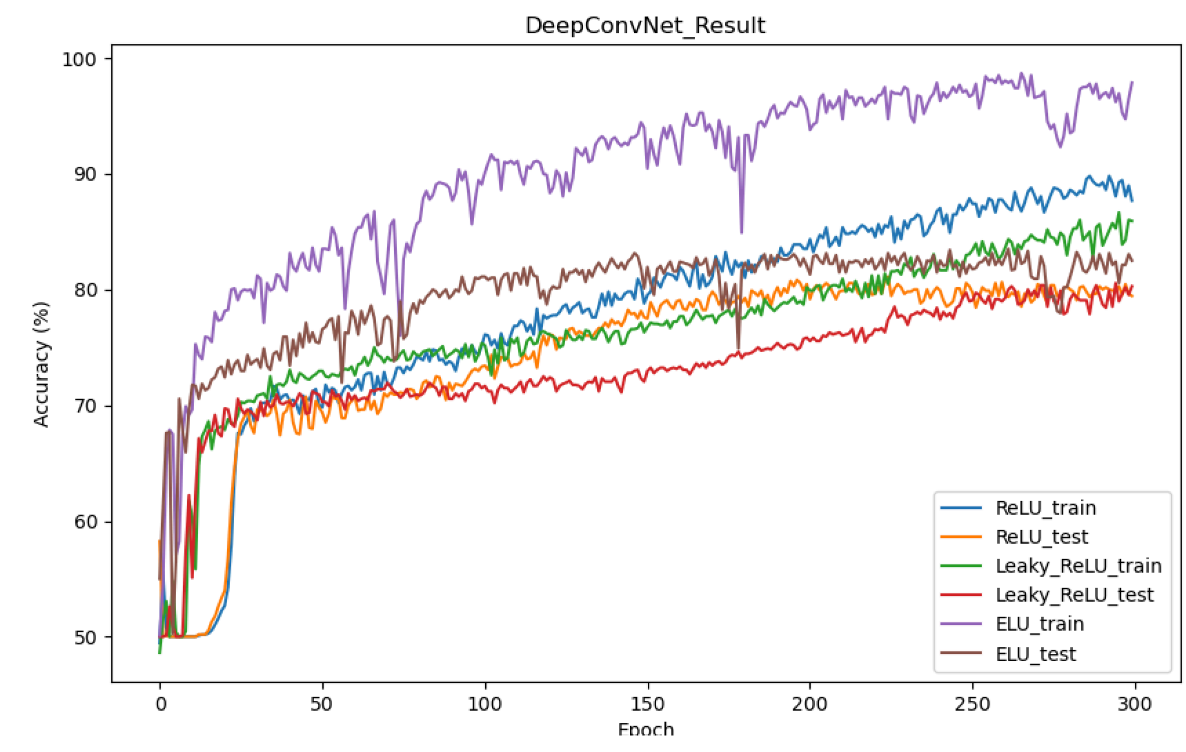
```python
def parse_argument():
    parser = ArgumentParser(description = 'EEGNet and DeepConvNet')
    parser.add_argument('--epochs', default = 300, help = 'number of epoch')
    parser.add_argument('--lr', default = 0.005, help = 'learning rate')
    parser.add_argument('--dropout', default = 0.5, help = 'Dropout')
    parser.add_argument('--batch', default = 256, help = 'Batch size')
    parser.add_argument('--savepath', default = './', help = 'Save path')
    parser.add_argument('--optimizer', default = torch.optim.Adam, help = 'optimizer function')
    parser.add_argument('--weight_decay', default = 0.005, help = 'Weight decay')

    return parser.parse_args()
```

b. Comparison figures

b-1. EEGNet



b-2. DeepConvNet



4. Discussion

1. 這次實驗中發現 training data 相較於 testing data 還要好訓練很多，因此如果不加以限制的話很容易讓訓練造成 overfitting , 因此我在 optimizer 中有加入 weight decay 來達到 regularization 的效果 , 其中又以 DeepConvNet 的影響較為大
2. 當 batch size 越大時，accuracy 有時候會有些的上升，可能使用更多的 training data 去訓練可以減少 overfitting 的程度，或者是讓 batch normalization 可以更好的在訓練中收斂，但缺點是會消耗比較多的 memory
3. 在做 accuracy 的運算時，要將 label data 的 type 改成 long , 否則 model predict 出來的 output 經過 torch.max 取得 one hot vector 的 index 沒辦法和 label data 做相等的判斷