# DLP HW3

# 309553008

- Introduction

  本次作業是實作 EEGNet 與 DeepConvNet 兩種架構，並分別測試三種不同的 activation function(ELU、ReLU、Leaky ReLU)結果有何差異。
  本次 dataset 是來自 BCI Competition III – IIIb 的腦波圖，有兩個 class(左右腦)，目的是利用上述兩種架構預測輸入的腦波圖是左腦還是右腦產出的。

- Experiment set up

  前處理:
  先使用助教提供的 dataloader 拿出 train/test data 與 label，然後將 type 轉乘 tensor，再把維度拉好，這樣就可以使用 pytorch 的 DataLoader 了

```
# data
train_dataset, test_dataset = gen_dataset(*dataloader.read_bci_data())
```

```
def gen_dataset(train_x, train_y, test_x, test_y):
    datasets = []
    for x, y in [(train_x, train_y), (test_x, test_y)]:
        x = torch.stack(
            # convert np.ndarray to tensor
            [torch.Tensor(x[i]) for i in range(x.shape[0])]
        )
        y = torch.stack(
            # convert np.ndarray to tensor
            [torch.Tensor(y[i:i+1]) for i in range(y.shape[0])]
        )
        datasets += [TensorDataset(x, y)]

    return datasets
```

  EEGNet:
  根據 ppt 上的參數設定網路架構，activation function 預設為 ELU，其中在 EEGNet 架構中 depthwiseConv 與 separableConv 這兩層都是為了減少參數量而設計的。

```python
class EEGNet(nn.Module):
    def __init__(self, activation=None, dropout=0.25):
        super(EEGNet, self).__init__()

        # set activation function
        if not activation:
            activation = nn.ELU(alpha=1.0)

        # Layer 1 : firstconv
        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)
        )

        # Layer 2 : depthwiseConv
        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),
            activation(),
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout(p=dropout)
        )

        # Layer 3 : separableConv
        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),
            activation(),
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout(p=dropout)
        )

        # Layer 4 : classify
        self.classify = nn.Sequential(
            nn.Linear(in_features=736, out_features=2, bias=True)
        )


    def forward(self, x):
        x = self.firstconv(x)
        x = self.depthwiseConv(x)
        x = self.separableConv(x)
        # flatten
        x = x.view(-1, 736)
        x = self.classify(x)

        return x
```

DeepConvNet:

參考網路上有一個人把每一層的 in/output channel 用一個 list 存起來，方便計算，因為除了第一層長得比較不一樣，其他層的架構都一樣(只有 channel 數差異)，這樣存起來後可以用一個 for 來快速地架好需要的層數

```python
class DeepConvNet(nn.Module):
    def __init__(self, activation=None, deepconv=[25,50,100,200], dropout=0.5):
        super(DeepConvNet, self).__init__()

        if not activation:
            activation = nn.ELU

        self.deepconv = deepconv

        # Layer 0
        self.conv0 = nn.Sequential(
            nn.Conv2d(1, deepconv[0], kernel_size=(1, 5)),
            nn.Conv2d(deepconv[0], deepconv[0], kernel_size=(2,1)),
            nn.BatchNorm2d(deepconv[0], eps=1e-05, momentum=0.1),
            activation(),
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=dropout)
        )

        for idx in range(1, len(deepconv)):
            setattr(self, 'conv'+str(idx), nn.Sequential(
                nn.Conv2d(deepconv[idx-1], deepconv[idx], kernel_size=(1,5), stride=(1,1), padding=(0,0), bias=True),
                nn.BatchNorm2d(deepconv[idx], eps=1e-05, momentum=0.1),
                activation(),
                nn.MaxPool2d(kernel_size=(1, 2)),
                nn.Dropout(p=dropout)
            ))


        flatten_size =  deepconv[-1] * reduce(lambda x,_: round((x-4)/2), deepconv, 750)
        self.classify = nn.Sequential(
            nn.Linear(flatten_size, 2, bias=True),
        )

    def forward(self, x):
        for i in range(len(self.deepconv)):
            x = getattr(self, 'conv'+str(i))(x)
        # flatten
        x = x.view(-1, self.classify[0].in_features)
        x = self.classify(x)
        return x
```

- Explain the activation function

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

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

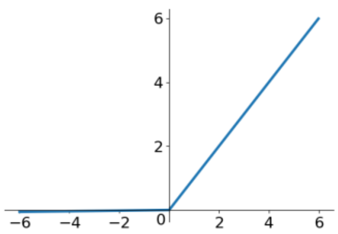
By default, the negative slope = 0.01

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

The α value for the ELU formulation.Default: 1.0

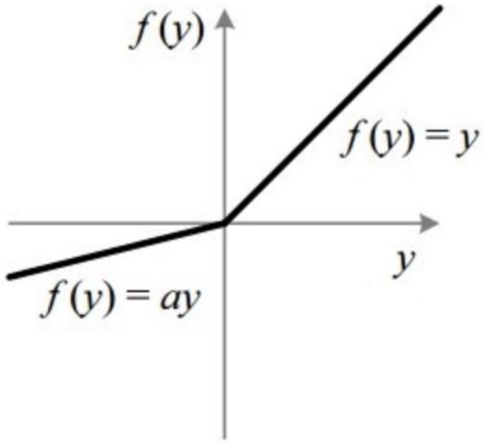
1. ReLU:

   將負值攤平為 0



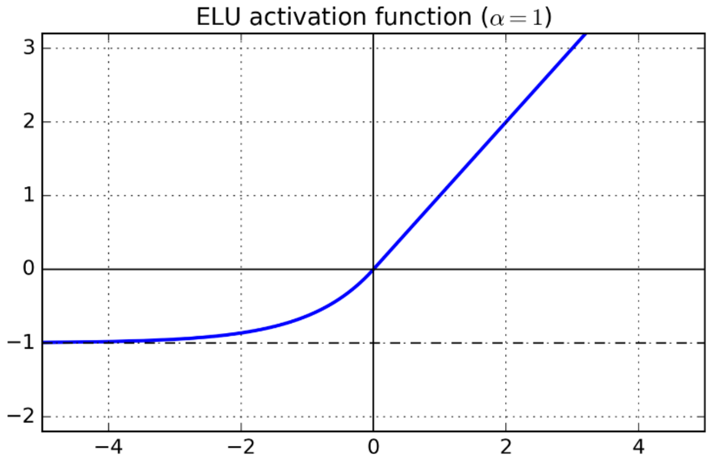
2. Leaky ReLU:

與 ReLU 長得差不多，差別在值為負的時候 Leaky ReLU 不會是一條 y = 0 的直線，會是一條斜率較小的斜線




3. ELU:

在正負交界處(y = 0)時是一條連續函數




- Experimental results:

The highest testing accuracy:

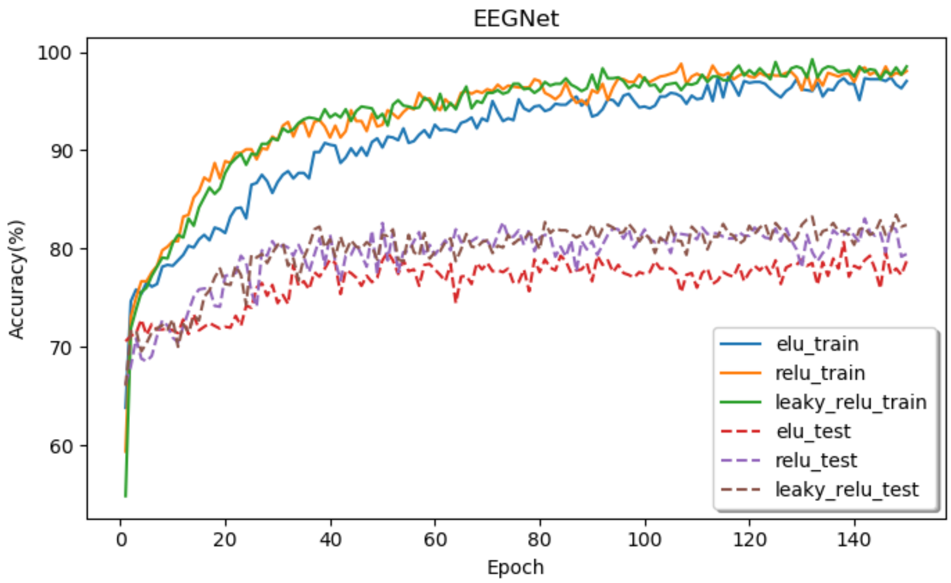
- Batch size= 128
- Learning rate = 1e-3
- Epochs = 300
- Optimizer: Adam
- Loss function: torch.nn.CrossEntropyLoss()

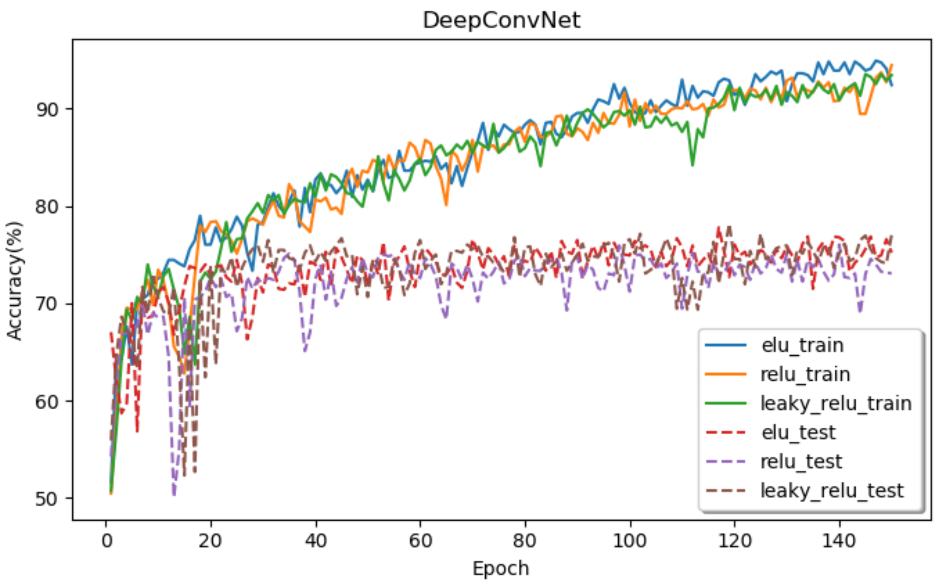
| | ReLU | Leaky ReLU | ELU |
|---|---|---|---|
| EEGNet | 85.27777777777777 | 85.37037037037037 | 84.25925925925925 |
| DeepConvNet | 80.0 | 80.0 | 78.98148148148148 |

Comparison figures:

助教給的參數:

- Batch size= 64
- Learning rate = 1e-2
- Epochs = 150
- Optimizer: Adam
- Loss function: torch.nn.CrossEntropyLoss()




| | ReLU | Leaky ReLU | ELU |
|---|---|---|---|
| GNet | 83.05555555555556 | 83.42592592592592 | 80.64814814814815 |
| epConvNet | 76.01851851851852 | 78.14814814814815 | 77.77777777777777 |

- 改動 learning rate:
  - Batch size= 128

- Learning rate = 1e-1
- Epochs = 300
- Optimizer: Adam
- Loss function: torch.nn.CrossEntropyLoss()

| | ReLU | Leaky ReLU | ELU |
|---|---|---|---|
| EGNet | 76.66666666666667 | 75.92592592592592 | 74.62962962962963 |
| eepConvNet | 53.333333333333336 | 70.64814814814815 | 77.68518518518519 |

- Learning rate = 1e-2

| | ReLU | Leaky ReLU | ELU |
|---|---|---|---|
| GNet | 82.31481481481481 | 81.94444444444444 | 78.51851851851852 |
| epConvNet | 75.55555555555556 | 75.18518518518519 | 76.94444444444444 |

- Learning rate = 1e-3

| | ReLU | Leaky ReLU | ELU |
|---|---|---|---|
| GNet | 83.79629629629629 | 82.87037037037037 | 80.83333333333333 |
| epConvNet | 77.12962962962963 | 78.79629629629629 | 77.5925925925926 |

- Learning rate = 1e-4

| | ReLU | Leaky ReLU | ELU |
|---|---|---|---|
| GNet | 79.72222222222223 | 78.88888888888889 | 80.0 |
| epConvNet | 77.5925925925926 | 77.5925925925926 | 77.12962962962963 |

- 改動 Optimizer:
  - Batch size= 128
  - Learning rate = 1e-3
  - Epochs = 300
  - Optimizer: Adagrad
  - Loss function: torch.nn.CrossEntropyLoss()

| | ReLU | Leaky ReLU | ELU |
|---|---|---|---|
| GNet | 75.83333333333333 | 74.81481481481481 | 74.9074074074074 |
| epConvNet | 75.46296296296296 | 74.16666666666667 | 75.83333333333333 |

  - Optimizer: RMSprop

| | ReLU | Leaky ReLU | ELU |
|---|---|---|---|
| GNet | 83.98148148148148 | 82.87037037037037 | 79.72222222222223 |
| epConvNet | 74.9074074074074 | 76.85185185185185 | 78.05555555555556 |

  - Optimizer: SGD

| | ReLU | Leaky ReLU | ELU |
|---|---|---|---|
| GNet | 73.61111111111111 | 72.68518518518519 | 73.51851851851852 |
| epConvNet | 71.85185185185185 | 72.22222222222223 | 72.87037037037037 |

Discussion:

嘗試了一下如果用 CPU 跑而不是用 GPU，發現真的很有感(其實是因為

train 到後面不知道為什麼電腦顯卡壞了，只要跑 cuda 就直接黑屏==)，如果用 GPU 跑這張 1070 大概都是 3 分鐘以後，而用 CPU 就跑了將近兩小時。

參考:

https://github.com/aliasvishnu/EEGNet/blob/master/EEGNet-PyTorch.ipynb
https://github.com/csielee/2019DL/blob/master/lab2/lab2.ipynb