Deep Learning and Practice Lab 3

309552007

袁鈺勛

A. Introduction

這次的作業是要利用 pytorch 做出 EEGNet 和 DeepConvNet 來分類 EEG signals，在程式中有提供 arguments 來控制 model 的各個參數，利用’-m’可以控制當次要訓練哪個 model，而’-l’可以控制 DeepConvNet 的額外 linear layer 數量，預設 1 表示 DeepConvNet 會有 2 層 linear layers。

```
parser = ArgumentParser(description='EEGNet & DeepConvNet')
parser.add_argument('-m', '--model', default='EEG', type=check_model_type, help='EEGNet or DeepConvNet')
parser.add_argument('-e', '--epochs', default=150, type=int, help='Number of epochs')
parser.add_argument('-lr', '--learning_rate', default=1e-2, type=float, help='Learning rate')
parser.add_argument('-b', '--batch_size', default=64, type=int, help='Batch size')
parser.add_argument('-o', '--optimizer', default='adam', type=check_optimizer_type, help='Optimizer')
parser.add_argument('-lf', '--loss_function', default='cross_entropy', type=check_loss_type, help='Loss function')
parser.add_argument('-d', '--dropout', default=0.25, type=float, help='Dropout probability')
parser.add_argument('-l', '--linear', default=1, type=check_linear_type,
                    help='Extra linear layers in DeepConvNet (default is 1)')
parser.add_argument('-v', '--verbosity', default=0, type=check_verbosity_type, help='Whether to show info log')
```

B. Experiment set up

1. The detail of your model

a. EEGNet

下面 3 張圖為 EEGNet model 的 code，在 initial function 會建立每一層，並且在 forward function 會將 input 經過每一層得到 output，因為計算 loss 的時候就是用 cross entropy，所以在 model 內沒有再經過 softmax。

```python
class EEGNet(nn.Module):
    def __init__(self, activation: nn.modules.activation, dropout: float):
        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)
        )

        self.depth_wise_conv = nn.Sequential(
            nn.Conv2d(
                in_channels=16,
                out_channels=32,
                kernel_size=(2, 1),
```

```python
                stride=(1, 1),
                groups=16,
                bias=False
            ),
            nn.BatchNorm2d(32),
            activation(),
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout(p=dropout)
        )

        self.separable_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),
```

```python
            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)
        )

    def forward(self, inputs: TensorDataset) → Tensor:
        """
        Forward propagation
        :param inputs: input data
        :return: results
        """
        first_conv_results = self.first_conv(inputs)
        depth_wise_conv_results = self.depth_wise_conv(first_conv_results)
        separable_conv_results = self.separable_conv(depth_wise_conv_results)
        return self.classify(separable_conv_results)
```

b. DeepConvNet

下面 4 張圖為 DeepConvNet model 的 code，一樣在 initial function 會建立每一層，最後面的 linear layers 預設是有 2 層，一層是 flatten size 到 100，另一層是 100 到 2，可以透過參數增加層數，並且在 forward function 會將 input 走過每一層得到 output，同樣也沒有經過 softmax。

```python
class DeepConvNet(nn.Module):
    def __init__(self, activation: nn.modules.activation, dropout: float, num_of_linear: int,
                 filters: Tuple[int] = (25, 50, 100, 200)):
        super().__init__()

        self.filters = filters
        self.conv_0 = nn.Sequential(
            # an input = [1, 1, 2, 750]
            nn.Conv2d(
                in_channels=1,
                out_channels=filters[0],
                kernel_size=(1, 5),
                bias=False
            ),
            # an input = [1, 25, 2, 746]
            nn.Conv2d(
                in_channels=filters[0],
                out_channels=filters[0],
                kernel_size=(2, 1),
                bias=False
            ),
            # an input = [1, 25, 1, 746]
```

```python
            nn.BatchNorm2d(filters[0]),
            activation(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            # an input = [1, 25, 1, 373]
            nn.Dropout(p=dropout)
        )

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

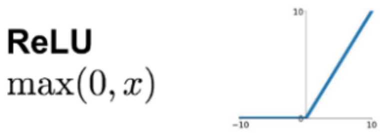
```python
        # If num_of_linear == 1, then there are 2 linear layers
        self.flatten_size = filters[-1] * reduce(lambda x, _: round((x - 4) / 2), filters[:-1], 373)
        interval = round((100.0 - 2.0) / num_of_linear)
        next_layer = 100
        features = [self.flatten_size]
        while next_layer > 2:
            features.append(next_layer)
            next_layer -= interval
        features.append(2)

        layers = [('flatten', nn.Flatten())]
        for idx, in_features in enumerate(features[:-1]):
            layers.append((f'linear_{idx}', nn.Linear(in_features=in_features,
                                                       out_features=features[idx + 1],
                                                       bias=True)))
            if idx != len(features) - 2:
                layers.append((f'activation_{idx}', activation()))
                layers.append((f'dropout_{idx}', nn.Dropout(p=dropout)))
        self.classify = nn.Sequential(OrderedDict(layers))
```

```python
    def forward(self, inputs: TensorDataset) -> Tensor:
        """
        Forward propagation
        :param inputs: input data
        :return: results
        """
        partial_results = inputs
        for idx in range(len(self.filters)):
            partial_results = getattr(self, f'conv_{idx}')(partial_results)
        return self.classify(partial_results)
```

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

a. ReLU

ReLU 全名是 Recitified Linear Units，他會將所有為負數的結果改成 0，正數則保持不變，限制結果都會大於等於 0，缺點是正值可能會無限大，而且將負數改為 0 會使產生負數的 neuron 不會再對 error 產生反應，因為 gradient 是 0。

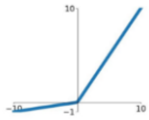


b. Leaky ReLU

Leaky ReLU 則是不希望放棄負數，所以會在負數上乘以一個較小

的值，使負數接近 0，和 ReLU 一樣會有正值可能無限大。

Leaky ReLU
$\max(0.1x, x)$

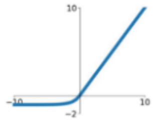


c. **ELU**

ELU 全名是 Exponential Linear Unit，和 ReLU 不同，他可以有負值，而且在在 0 那的轉變較為平滑，但同樣會有正值無限大的問題。

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



C. Experimental results

1. The highest testing accuracy

a. **EEGNet**

下圖的結果是將 learning rate 調成 0.005，batch size 調成 512，optimizer 用 adamax 時得到的，這時得到的 accuracy 會比較大應該是因為 batch size 調很大的關係。

```
EEG_ELU_train:       98.70 %
EEG_ReLU_train:      98.80 %
EEG_LeakyReLU_train: 98.89 %
EEG_ELU_test:        82.78 %
EEG_ReLU_test:       81.20 %
EEG_LeakyReLU_test:  85.74 %
```

b. **DeepConvNet**

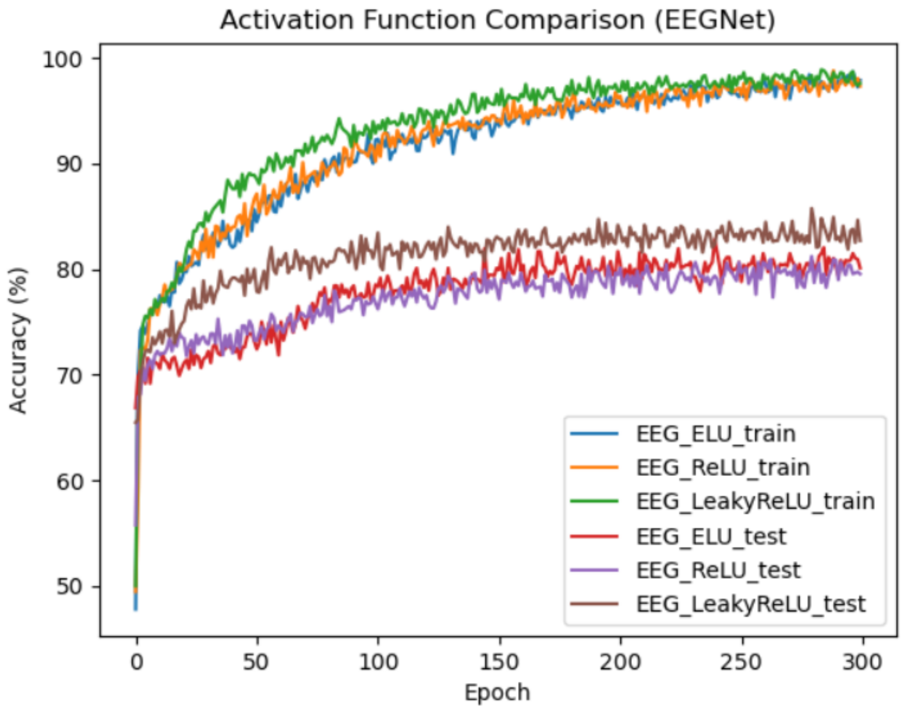
下圖的結果是將 learning rate 調成 0.001，batch size 調成 32，optimizer 用 adamax 以及 dropout 用 0.25 的結果。

```
Deep_ELU_train:       100.00 %
Deep_ReLU_train:      100.00 %
Deep_LeakyReLU_train: 100.00 %
Deep_ELU_test:        80.00 %
Deep_ReLU_test:       79.63 %
Deep_LeakyReLU_test:  80.28 %
```

2. Comparison figures

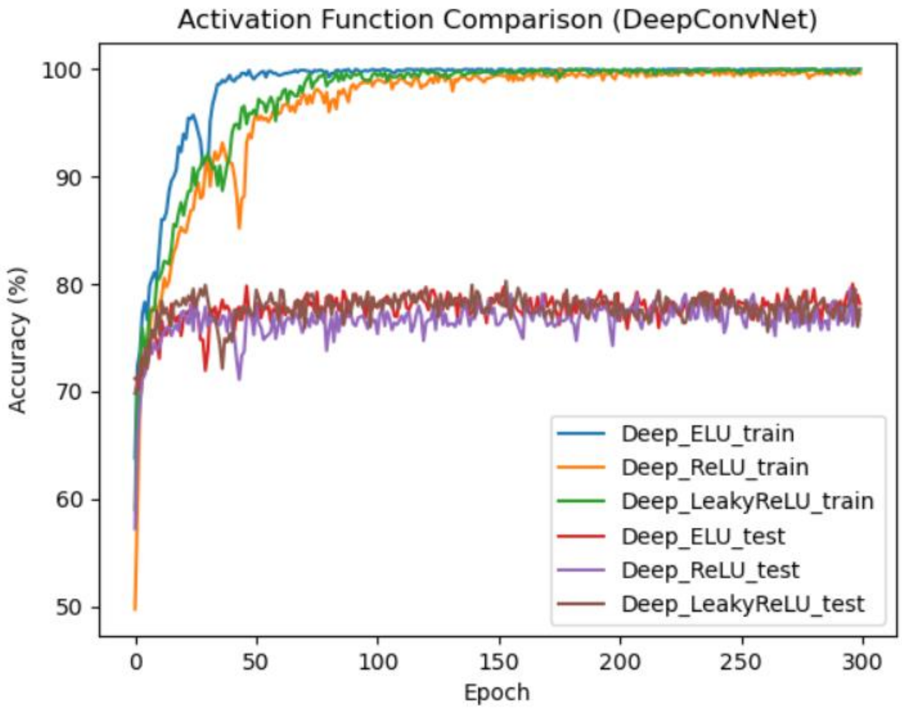
a. **EEGNet**

下圖為得到 EEGNet 最高 accuracy 時的比較圖。



b. DeepConvNet

下圖為得到 DeepConvNet 最高 accuracy 時的比較圖。

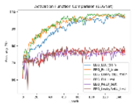
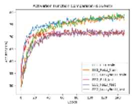
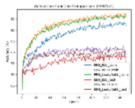
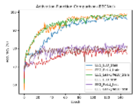
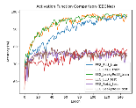
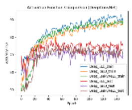
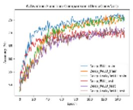
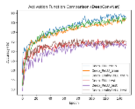
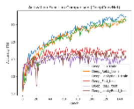
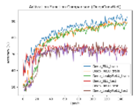


D. Discussion

以下實驗都是將 epoch 設定為 150。

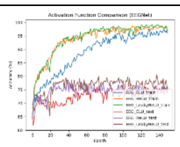
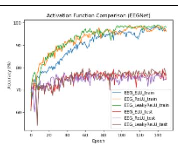
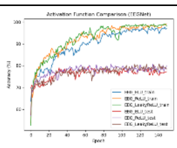
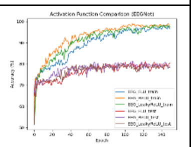
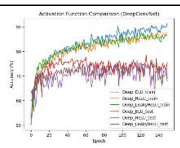
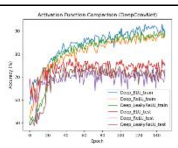
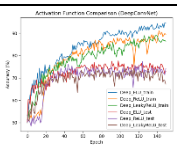
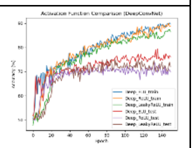
1. Optimizer

由下表比較出 Adamax 的效果比其他 optimizer 稍微好一些，在 EEGNet 比較明顯。

EEGNet					
	Adam (default)	Adadelta	Adagrad	Adamax	Adamw
Graph					
Accuracy	EEG_ELU_train: 98.33 % EEG_ReLU_train: 98.98 % EEG_LeakyReLU_train: 99.07 % EEG_ELU_test: 79.26 % EEG_ReLU_test: 78.43 % EEG_LeakyReLU_test: 79.63 %	EEG_ELU_train: 79.17 % EEG_ReLU_train: 79.17 % EEG_LeakyReLU_train: 79.72 % EEG_ELU_test: 73.15 % EEG_ReLU_test: 73.06 % EEG_LeakyReLU_test: 73.61 %	EEG_ELU_train: 93.70 % EEG_ReLU_train: 97.59 % EEG_LeakyReLU_train: 97.22 % EEG_ELU_test: 79.44 % EEG_ReLU_test: 82.87 % EEG_LeakyReLU_test: 81.30 %	EEG_ELU_train: 97.87 % EEG_ReLU_train: 99.17 % EEG_LeakyReLU_train: 98.70 % EEG_ELU_test: 80.83 % EEG_ReLU_test: 83.15 % EEG_LeakyReLU_test: 81.39 %	EEG_ELU_train: 97.87 % EEG_ReLU_train: 98.98 % EEG_LeakyReLU_train: 98.89 % EEG_ELU_test: 79.35 % EEG_ReLU_test: 78.15 % EEG_LeakyReLU_test: 79.26 %
DeepConvNet					
	Adam (default)	Adadelta	Adagrad	Adamax	Adamw
Graph					
Accuracy	Deep_ELU_train: 92.96 % Deep_ReLU_train: 89.35 % Deep_LeakyReLU_train: 90.00 % Deep_ELU_test: 78.15 % Deep_ReLU_test: 74.54 % Deep_LeakyReLU_test: 76.11 %	Deep_ELU_train: 77.58 % Deep_ReLU_train: 77.04 % Deep_LeakyReLU_train: 76.30 % Deep_ELU_test: 71.94 % Deep_ReLU_test: 72.69 % Deep_LeakyReLU_test: 72.41 %	Deep_ELU_train: 89.54 % Deep_ReLU_train: 87.41 % Deep_LeakyReLU_train: 87.96 % Deep_ELU_test: 76.20 % Deep_ReLU_test: 75.28 % Deep_LeakyReLU_test: 76.67 %	Deep_ELU_train: 95.93 % Deep_ReLU_train: 93.70 % Deep_LeakyReLU_train: 93.33 % Deep_ELU_test: 77.13 % Deep_ReLU_test: 73.24 % Deep_LeakyReLU_test: 75.46 %	Deep_ELU_train: 94.35 % Deep_ReLU_train: 90.19 % Deep_LeakyReLU_train: 89.63 % Deep_ELU_test: 76.85 % Deep_ReLU_test: 76.39 % Deep_LeakyReLU_test: 77.31 %

2. Batch size

由下表可以看出當 batch size 上升的時候 accuracy 會些微上升，同時不同 activation 的結果之間差距也較小，畢竟每次 training 用較多的 data，可以降低 overfitting，但同時也會消耗較多的 memory。

EEGNet				
	Batch size 32	Batch size 64 (default)	Batch size 128	Batch size 256
Graph				
Accuracy	EEG_ELU_train: 97.50 % EEG_ReLU_train: 98.80 % EEG_LeakyReLU_train: 99.26 % EEG_ELU_test: 76.67 % EEG_ReLU_test: 78.61 % EEG_LeakyReLU_test: 80.65 %	EEG_ELU_train: 98.33 % EEG_ReLU_train: 98.98 % EEG_LeakyReLU_train: 99.07 % EEG_ELU_test: 79.26 % EEG_ReLU_test: 78.43 % EEG_LeakyReLU_test: 79.63 %	EEG_ELU_train: 97.69 % EEG_ReLU_train: 99.17 % EEG_LeakyReLU_train: 99.07 % EEG_ELU_test: 79.44 % EEG_ReLU_test: 80.56 % EEG_LeakyReLU_test: 80.83 %	EEG_ELU_train: 98.24 % EEG_ReLU_train: 99.07 % EEG_LeakyReLU_train: 98.52 % EEG_ELU_test: 81.11 % EEG_ReLU_test: 81.39 % EEG_LeakyReLU_test: 80.09 %
DeepConvNet				
	Batch size 32	Batch size 64 (default)	Batch size 128	Batch size 256
Graph				
Accuracy	Deep_ELU_train: 91.48 % Deep_ReLU_train: 87.59 % Deep_LeakyReLU_train: 87.78 % Deep_ELU_test: 78.43 % Deep_ReLU_test: 76.48 % Deep_LeakyReLU_test: 77.13 %	Deep_ELU_train: 92.96 % Deep_ReLU_train: 89.35 % Deep_LeakyReLU_train: 90.00 % Deep_ELU_test: 78.15 % Deep_ReLU_test: 74.54 % Deep_LeakyReLU_test: 76.11 %	Deep_ELU_train: 94.63 % Deep_ReLU_train: 91.02 % Deep_LeakyReLU_train: 88.98 % Deep_ELU_test: 77.59 % Deep_ReLU_test: 77.31 % Deep_LeakyReLU_test: 74.35 %	Deep_ELU_train: 90.19 % Deep_ReLU_train: 90.09 % Deep_LeakyReLU_train: 87.41 % Deep_ELU_test: 79.35 % Deep_ReLU_test: 73.70 % Deep_LeakyReLU_test: 75.19 %