Lab 2 Report - EEG Classification

- Student Info
  1. Student ID: 310555024
  2. Student Name: 林廷翰

- Introduction (20%)

  此次實驗是基於PyTorch實作兩個neural network - EEGNet及DeepConvNet，而的dataset為BCI Competition III – IIIb，目的是腦波圖的分類。

  在dataset方面，dataset shape為(B, 1, 2, 750)，其中B代表的是batch size，是我們可以調整的參數之一，而(2, 750)則是因為`read_bci_data`已經將兩個channel的data合為一個channel，每個channel原本就具有750筆資料，代表的是一段時間內的腦波變化，而label的0, 1則代表左手還是右手。

  在實作方面TA提供`read_bci_data`用於load training及testing data，並提供EEGNet及DeepConvNet兩模型的參數，因此我們需要根據模型參數建立模型，再根據lab 0所教的PyTorch基本操作，實作training及testing，最後根據accuracy紀錄模型參數，以便在demo時可重現accuracy的結果。

- Experiments Setups (30%)
  1. The detail of your model
     - EEGNet

       根據spec上的定義，建立`EEGNet`，在`init` function內會建立每一個layer，而forward會將output做為下一層的input。

       1. `first_conv`

```python
        # nn.Sequential is sequential model container. The forward() method of Sequential accepts any input and
        # forwards it to the first module it contains. It then “chains” outputs to inputs sequentially for each
        # subsequent module, finally returning the output of the last module.
        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(
                # C from an expected input of size (N, C, H, W).
                num_features=16,
                # A value added to the denominator for numerical stability.
                eps=1e-05,
                # The value used for the running_mean and running_var computation.
                momentum=0.1,
                # A boolean value that when set to True, this module has learnable affine parameters.
                affine=True,
                # A boolean value that when set to True, this module tracks the running mean and variance,
                # and when set to False, this module does not track such statistics.
                track_running_stats=True,
            )
```

2. `depth_wise_conv`

```python
        self.depth_wise_conv = nn.Sequential(
            nn.Conv2d(
                in_channels=16,
                out_channels=32,
                kernel_size=(2, 1),
                stride=(1, 1),
                # Number of blocked connections from input channels to output channels.
                # Reference: https://blog.csdn.net/monsterhoho/article/details/80173400
                groups=16,
                bias=False,
            ),
            nn.BatchNorm2d(
                num_features=32,
                eps=1e-05,
                momentum=0.1,
                affine=True,
                track_running_stats=True,
            ),
            activation_function(),
            nn.AvgPool2d(
                kernel_size=(1, 4),
                stride=(1, 4),
                padding=0,
            ),
            # During training, randomly zeroes some of the elements of the input tensor with probability p using
            # samples from a Bernoulli distribution.
            # The objective is avoiding overfitting to some channel data.
            nn.Dropout(
                # probability of an element to be zeroed.
                p=0.25,
            )
        )
```

3. `separable_conv`

```
        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(
                num_features=32,
                eps=1e-05,
                momentum=0.1,
                affine=True,
                track_running_stats=True,
            ),
            activation_function(),
            nn.AvgPool2d(
                kernel_size=(1, 8),
                stride=(1, 8),
                padding=0
            ),
            nn.Dropout(p=0.25)
        )
```

4. `classify`

```
        self.classify = nn.Sequential(
            # Flattens a contiguous range of dims into a tensor.
            nn.Flatten(),
            # Applies a linear transformation to the incoming data.
            nn.Linear(
                in_features=736,
                out_features=2,
                bias=True,
            )
        )
```

5. `forward`

```
    def forward(self, inputs: TensorDataset):
        inputs = self.first_conv(inputs)
        inputs = self.depth_wise_conv(inputs)
        inputs = self.separable_conv(inputs)
        return self.classify(inputs)
```

- DeepConvNet

  和`EEGNet`相似，根據spec上的定義，建立`DeepConvNet`，在`init` function內會建立每一個layer，而forward會將output做為下一層的input。

  1. `conv0`

```
        self.conv0 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=self.deepconv[0],
                kernel_size=(1, 5),
                stride=(1, 1),
                padding=(0, 0),
                bias=True
            ),
            nn.Conv2d(
                in_channels=self.deepconv[0],
                out_channels=self.deepconv[0],
                kernel_size=(2, 1),
                stride=(1, 1),
                padding=(0, 0),
                bias=True
            ),
            nn.BatchNorm2d(
                num_features=self.deepconv[0]
            ),
            activation_function(),
            nn.MaxPool2d(
                kernel_size=(1, 2)
            ),
            nn.Dropout(
                p=dropout
            )
        )
```

2. `conv1` - `conv3`

   由於layer操作相似，因此可以基於for loop來建立`conv1` - `conv3`。

```
        for index in range(1, len(self.deepconv)):
            setattr(self, 'conv' + str(index), nn.Sequential(
                nn.Conv2d(
                    in_channels=self.deepconv[index - 1],
                    out_channels=self.deepconv[index],
                    kernel_size=(1, 5),
                    stride=(1, 1),
                    padding=(0, 0),
                    bias=True
                ),
                nn.BatchNorm2d(
                    num_features=self.deepconv[index]
                ),
                activation_function(),
                nn.MaxPool2d(
                    kernel_size=(1, 2)
                ),
                nn.Dropout(
                    p=dropout
                )
            ))
```

3. `classify`

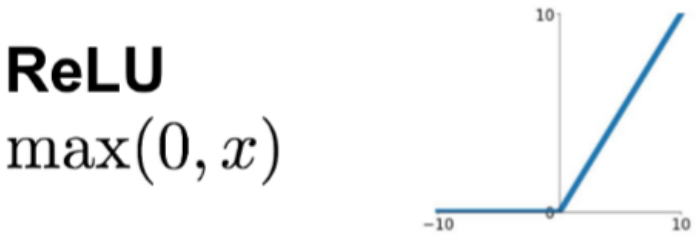
```
        flatten_size = self.deepconv[-1] * reduce(lambda x, _: round((x - 4) / 2), self.deepconv, 750)
        self.classify = nn.Sequential(
            nn.Linear(
                in_features=flatten_size,
                out_features=2,
                bias=True,
            ),
        )
```

4. `forward`

```python
def forward(self, inputs: TensorDataset):
    for index in range(len(self.deepconv)):
        inputs = getattr(self, 'conv' + str(index))(inputs)

    # Flatten
    inputs = inputs.view(-1, self.classify[0].in_features)
    inputs = self.classify(inputs)
    return inputs
```

2. Explain the activation function
    - ReLU
        1. 介紹

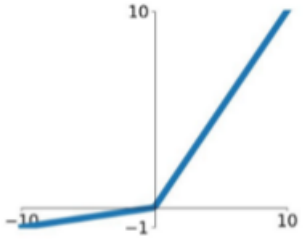


為最常見的activation function。

        2. 優點
            - back-propagation過程運算簡單
            - training過程不需要很多的computation。
            - 相對Sigmoid不會saturate。
            - 收斂數度快。
        3. 缺點
            - 當hidden layer output為0時，gradient為0，產生dead ReLU。
    - Leaky ReLU
        1. 介紹

Leaky ReLU

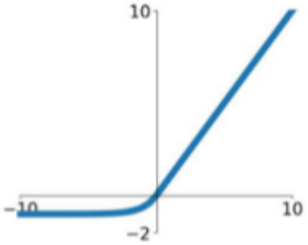
$$\max(0.1x, x)$$



2. 優點
    - 具有ReLU的所有特性。
    - 改善ReLU的dead ReLU問題。
- ELU
    1. 介紹

ELU

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



    2. 優點
        - 不具ReLU的dead ReLU問題。
        - 比Leaky ReLU更接近0平均輸出。
    3. 缺點
        - 需更多的computation，原因是exponential function。

- Experiments Results (30%)
    1. The highest testing accuracy
        - EEG
            1. Parameters

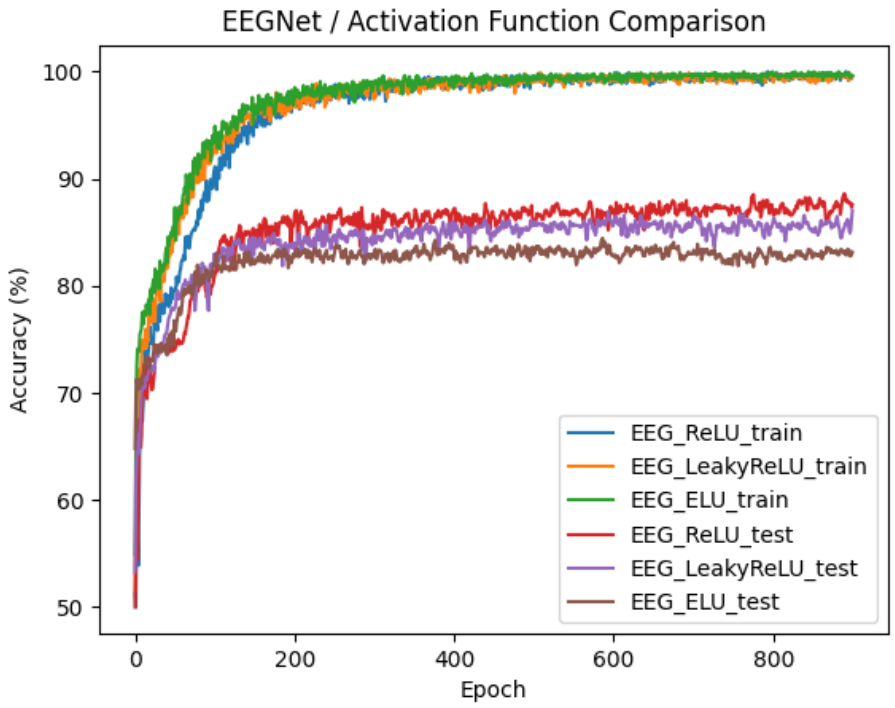
- batch_size=1080
- learning_rate=0.005
- epoch=900
- optimizer: Adam
- loss_function: `torch.nn.CrossEntropyLoss()`

2. Accuracy

```
EEG_ReLU_train: 100.00 %
EEG_LeakyReLU_train: 99.91 %
EEG_ELU_train: 100.00 %
EEG_ReLU_test: 88.61 %
EEG_LeakyReLU_test: 87.04 %
EEG_ELU_test: 84.44 %

Process finished with exit code 0
```

3. Comparison figures



4. Discussion

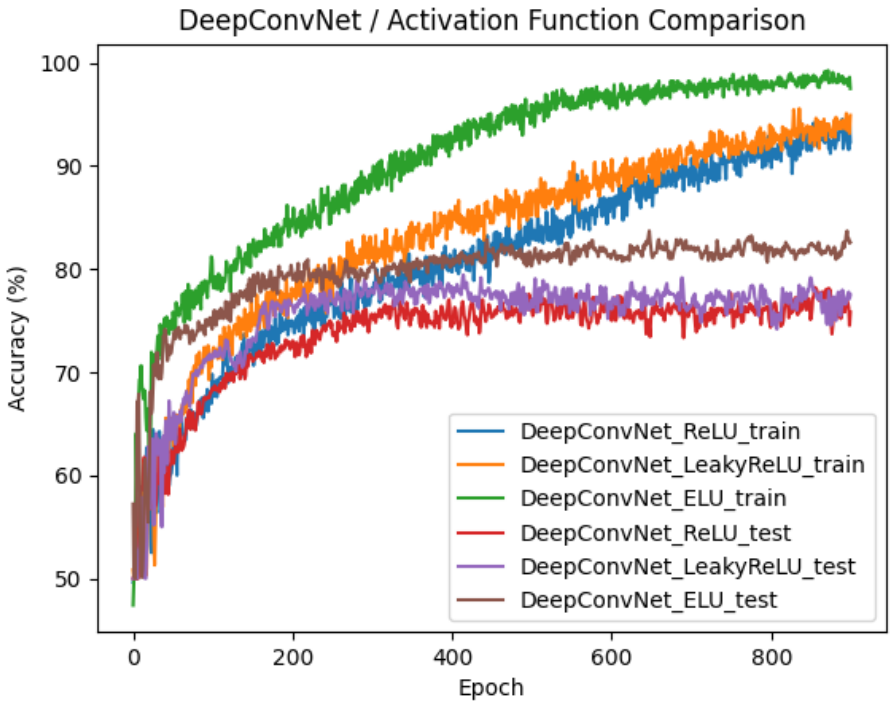
在嘗試使用不同的`batch_size`後，發現當`batch_size`變大時，accuracy會變大，因此直接把`batch_size`調到最大的1080，得到87.78%的accuracy。原因我認為是當`batch_size`變大時，所算的weight gradient是基於所有dataset，因此結果不會偏向每次batch的dataset。

- DeepConvNet
  1. Parameters
     - batch_size=1080
     - learning_rate=0.005
     - epoch=900
     - optimizer: Adam
     - loss_function: `torch.nn.CrossEntropyLoss()`
  2. Accuracy

```
DeepConvNet_ReLU_train: 94.54 %
DeepConvNet_LeakyReLU_train: 95.56 %
DeepConvNet_ELU_train: 99.26 %
DeepConvNet_ReLU_test: 78.24 %
DeepConvNet_LeakyReLU_test: 79.44 %
DeepConvNet_ELU_test: 83.70 %

Process finished with exit code 0
```

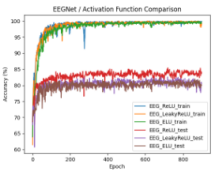
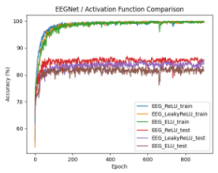
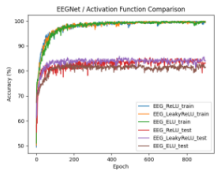
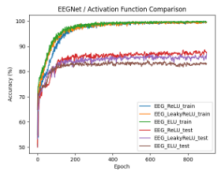
  3. Comparison figures



4. Discussion

   嘗試利用與`EEGNet`相同的parameters，但結果並沒有`EEGNet`來的好。但從comparison figure的趨勢可看出，可能在給更多的epoch表現會更佳。

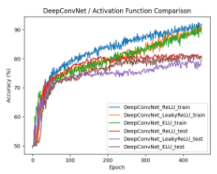
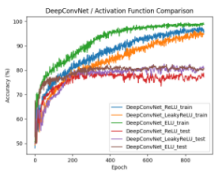
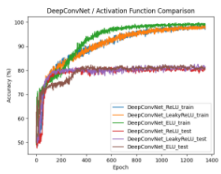
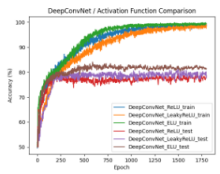
- Discussion (20%)
  1. Try different batch size in EEGNet
     - Parameters
       1. learning_rate=0.005
       2. epoch=900
       3. optimizer: Adam
       4. loss_function: `torch.nn.CrossEntropyLoss()`
     - Results

	batch_size=64	batch_size=256	batch_size=512	batch_size=1080
Graph				
Accuracy	EEG_ReLU_train: 100.00 % EEG_LeakyReLU_train: 100.00 % EEG_ELU_train: 100.00 % EEG_ReLU_test: 85.46 % EEG_LeakyReLU_test: 82.59 % EEG_ELU_test: 82.31 %	EEG_ReLU_train: 100.00 % EEG_LeakyReLU_train: 100.00 % EEG_ELU_train: 100.00 % EEG_ReLU_test: 86.85 % EEG_LeakyReLU_test: 85.65 % EEG_ELU_test: 83.52 %	EEG_ReLU_train: 100.00 % EEG_LeakyReLU_train: 100.00 % EEG_ELU_train: 100.00 % EEG_ReLU_test: 85.37 % EEG_LeakyReLU_test: 85.93 % EEG_ELU_test: 83.33 %	EEG_ReLU_train: 100.00 % EEG_LeakyReLU_train: 99.91 % EEG_ELU_train: 100.00 % EEG_ReLU_test: 88.61 % EEG_LeakyReLU_test: 87.04 % EEG_ELU_test: 84.44 %

- Discussion

  當`batch_size`上升時，會產生更好的accuracy，且收斂時間點差異不大，但實際上在`batch_size`256及512之間，差距並不大。

2. Try more epoch in DeepConvNet
   - Parameters
     1. batch_size=1080
     2. learning_rate=0.005
     3. optimizer: Adam
     4. loss_function: `torch.nn.CrossEntropyLoss()`
   - Results

	epoch=450	epoch=900	epoch=1350	epoch=1800
Graph				
Accuracy	DeepConvNet_ReLU_train: 92.41 % DeepConvNet_LeakyReLU_train: 91.20 % DeepConvNet_ELU_train: 91.02 % DeepConvNet_ReLU_test: 81.76 % DeepConvNet_LeakyReLU_test: 79.54 % DeepConvNet_ELU_test: 81.48 %	DeepConvNet_ReLU_train: 97.78 % DeepConvNet_LeakyReLU_train: 96.20 % DeepConvNet_ELU_train: 99.54 % DeepConvNet_ReLU_test: 79.63 % DeepConvNet_LeakyReLU_test: 82.50 % DeepConvNet_ELU_test: 82.31 %	DeepConvNet_ReLU_train: 98.89 % DeepConvNet_LeakyReLU_train: 98.80 % DeepConvNet_ELU_train: 99.81 % DeepConvNet_ReLU_test: 81.57 % DeepConvNet_LeakyReLU_test: 82.50 % DeepConvNet_ELU_test: 82.41 %	DeepConvNet_ReLU_train: 99.44 % DeepConvNet_LeakyReLU_train: 99.35 % DeepConvNet_ELU_train: 99.81 % DeepConvNet_ReLU_test: 79.35 % DeepConvNet_LeakyReLU_test: 81.11 % DeepConvNet_ELU_test: 84.07 %

- Discussion

  當`epoch`上升時，對training data會產生更好的accuracy，但對testing data就不一定，可能變高也可能變低。

- Reference

1. https://www.youtube.com/watch?v=GMSjDTU8Zlc&ab_channel=CloudCasts-AlanSmith
2. https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092