Deep Learning and Practice Lab2 – EEG classification

310554009 杜葳葳

1. Introduction

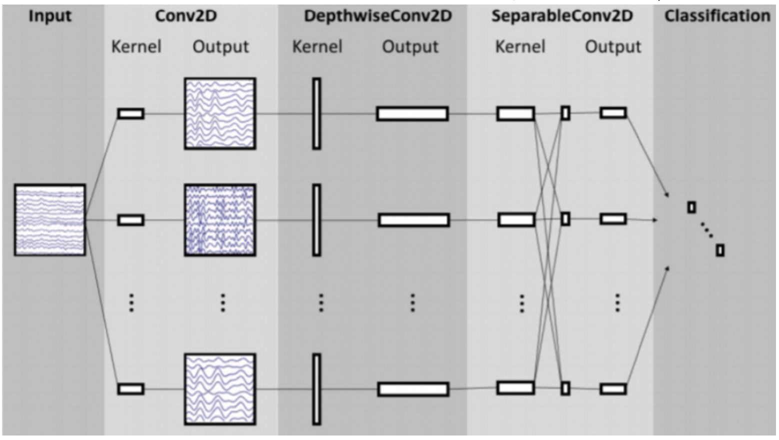
比較 EEGNet 和 DeepConvNet 對 BCI competition 資料集做大腦訊號分類器, EEGNet 和 DeepConvNet 皆為卷積神經網絡, 主要用來自動提取特徵和分類。實驗嘗試三種 Activation function(ReLU, Leaky ReLU, ELU) 、並調整超參數做比較，最終可以到達 87.87%的準確率。

2. Experiment set up

A. The detail of your model

- **EEGNet**

依照 spec 的規定建立模型，主要分為四個部分，第一個 conv2D 將訊號讀入並提取特徵，depthwiseconv2D 將訊號維度降低，separableconv2D 的輸入與輸出層沒有完全連接，最後是分類層。




```
EEGNet(
  (firstconv): Sequential(
    (0): Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25), bias=False)
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (deptwiseConv): Sequential(
    (0): Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25, inplace=False)
  )
  (separableConv): Sequential(
    (0): Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

- **DeepConvNet**

  與 EEGNet 相比，有較多個卷積層。

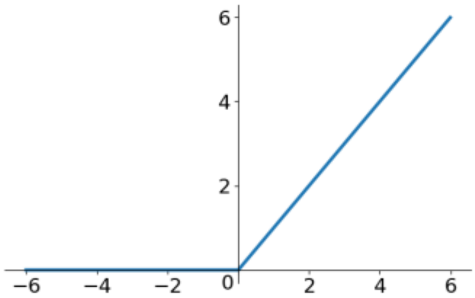
Layer	# filters	size	# params	Activation	Options
Input		(C, T)			
Reshape		(1, C, T)			
Conv2D	25	(1, 5)	150	Linear	mode = valid, max norm = 2
Conv2D	25	(C, 1)	25 * 25 * C + 25	Linear	mode = valid, max norm = 2
BatchNorm			2 * 25		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	50	(1, 5)	25 * 50 * C + 50	Linear	mode = valid, max norm = 2
BatchNorm			2 * 50		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	100	(1, 5)	50 * 100 * C + 100	Linear	mode = valid, max norm = 2
BatchNorm			2 * 100		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	200	(1, 5)	100 * 200 * C + 200	Linear	mode = valid, max norm = 2
BatchNorm			2 * 200		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Flatten					
Dense	N			softmax	max norm = 0.5

```
DeepConvNet(
  (conv0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1))
  (conv1): Sequential(
    (0): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (1): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (conv4): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (classify): Linear(in_features=8600, out_features=2, bias=True)
)
```

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

- **ReLU (Rectified Linear Unit)**

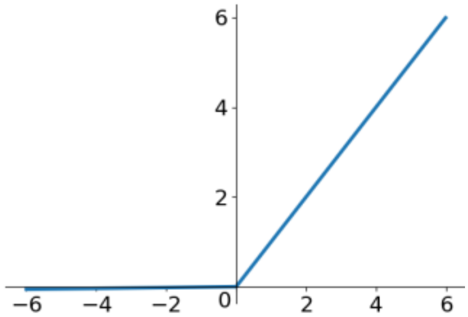
  若值為正數，則輸出該值大小，若值為負數，則輸出為 0。ReLU 是近年來最頻繁被使用的激勵函數，因其存在以下特點，包含:解決梯度爆炸問題、計算數度相當快、收斂速度快等特性。



$$f(x) = x^{+} = \max(0, x)$$

- **Leaky ReLU**

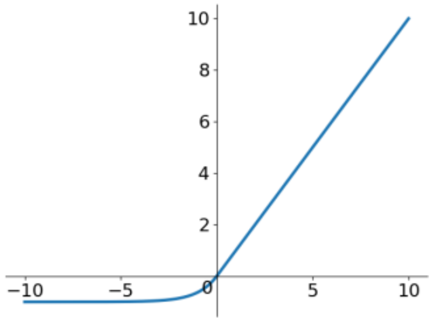
  為了解決 Dead ReLU Problem，將 ReLU 的前半段輸出設為 $0.01x$，故能防止負號時永遠無法被激活的問題。Leaky ReLU 擁有 ReLU 的所有優點，也成功避免 Dead ReLU Problem 的問題，但實際上沒有辦法證明 Leaky ReLU 永遠優於 ReLU。



$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$$

- **ELU (Exponential Linear Units)**

  ELU 也是為了解決 Dead ReLU 問題而被提出



$$f(x) = \begin{cases} x, & \mathit{if}\ x > 0 \\ \alpha\left(e^{x} - 1\right), & \mathit{otherwise} \end{cases}$$

3. Experimental results

A. The highest testing accuracy

- **Screenshot with two models**
  - **EEGNet**

```
Epoch  4842   Accuracy:  0.8481481481481481
[EEGNet] Activation function:  elu => Best Accuracy:  0.8481481481481481
Epoch  2942   Accuracy:  0.8787037037037037
[EEGNet] Activation function:  relu => Best Accuracy:  0.8787037037037037
Epoch  1623   Accuracy:  0.8675925925925926
[EEGNet] Activation function:  leakyrelu => Best Accuracy:  0.8675925925925926
```

■ DeepConvNet

```
Epoch  780   Accuracy:  0.812962962962963
[DeepConvNet] Activation function:  elu => Best Accuracy:  0.812962962962963
```

```
Epoch  1012   Accuracy:  0.8222222222222222
[DeepConvNet] Activation function:  relu => Best Accuracy:  0.8222222222222222
```

```
Epoch  636   Accuracy:  0.8212962962962963
[DeepConvNet] Activation function:  leakyrelu => Best Accuracy:  0.8212962962962963
```

■ Best model

Accuracy	Architecture	Activation Function	Epoch	LR	Batch Size	Optimizer
87.87%	EEGNet	ReLU	2942	1e-1	64	Adam

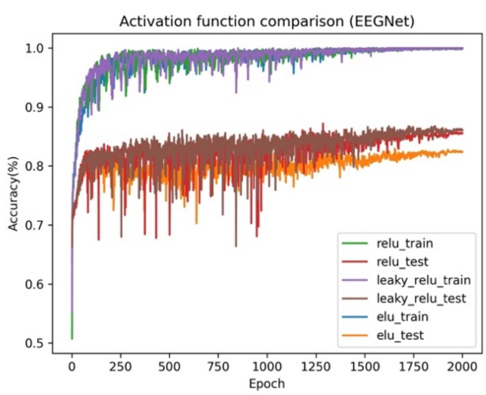
◆ Anything you want to present

為方便 demo、分為兩支程式，train.py 會將每個 activation function 最好（準確率最高）的模型存起來，test.py 則可以直接用 testing data 做測試，不用每次重新訓練，下面為其中一次執行 test.py 的截圖。

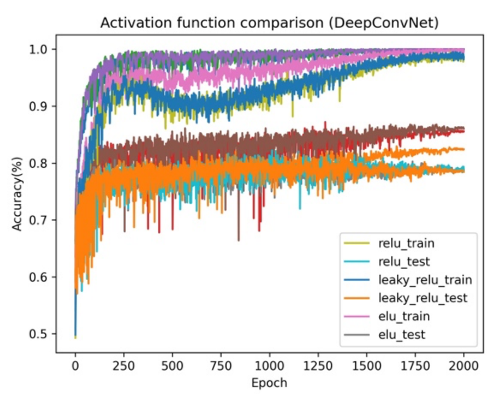
```
(DL) ubuntu@ec037-103:~/lab2$ python test.py
(1080, 1, 2, 750) (1080,) (1080, 1, 2, 750) (1080,)
train_loader: <torch.utils.data.dataloader.DataLoader object at 0x7f46026d4fd0>
test_loader: <torch.utils.data.dataloader.DataLoader object at 0x7f46026d4d30>
device:  cuda
[EEGNet] Activation function:  elu => Accuracy:  0.8296296296296296
[EEGNet] Activation function:  relu => Accuracy:  0.8722222222222222
[EEGNet] Activation function:  leakyrelu => Accuracy:  0.8675925925925926
--------------------------
/home/ubuntu/anaconda3/envs/DL/lib/python3.8/site-packages/torch/nn/functional.py:71
are an experimental feature and subject to change. Please do not use them for anythi
nternally at  /pytorch/c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
[DeepConvNet] Activation function:  elu => Accuracy:  0.812962962962963
[DeepConvNet] Activation function:  relu => Accuracy:  0.8222222222222222
[DeepConvNet] Activation function:  leakyrelu => Accuracy:  0.8212962962962963
```

B. Comparison figures

EEGNet



DeepConvNet



4. Discussion

在訓練與實驗過程中，有嘗試一次調整一個參數、固定其他參數，觀察準確率的變化。

A. Batch size

EEGNet 的表現皆比 DeepConvNet 來的好，其中 Batch size 為 64 與 128 時，模型會有比較好的表現。

- **EEGNet**

Batch size		8	16	32	64	128	256
Accuracy	ELU	0.81	0.82	0.81	0.82	0.82	0.82
	ReLU	0.83	0.84	0.84	0.83	0.84	0.83
	LeakyReLU	0.83	0.83	0.84	**0.85**	**0.85**	**0.85**

- **DeepConvNet**

Batch size		8	16	32	64	128	256
Accuracy	ELU	0.80	0.81	0.81	0.80	0.80	0.80
	ReLU	0.81	0.79	0.81	0.80	0.80	0.80
	LeakyReLU	0.80	0.80	0.80	0.79	0.80	0.80

B. Learning rate

Learning rate 在 1 和 1e-1 時模型會有比較好的表現，在訓練時有使用 Learning rate scheduler 來調整學習率，避免遇到無法收斂的問題。

- **EEGNet**

Learning rate		1	1e−1	1e−2	1e−3	1e−4
Accuracy	ELU	0.82	0.82	0.80	0.82	0.81
	ReLU	0.84	0.84	0.80	0.83	0.84
	LeakyReLU	**0.86**	**0.85**	0.79	0.85	0.85

- **DeepConvNet**

Learning rate		1	1e−1	1e−2	1e−3	1e−4
Accuracy	ELU	0.80	0.80	0.80	0.80	0.81
	ReLU	0.80	0.80	0.80	0.80	0.80
	LeakyReLU	0.80	0.80	0.79	0.80	0.79