NCTU DLP Lab2-Report
BCI Classification using EEGNet and DeepConvNet

廖家鴻 0786009

2020/4/14

Outline

- **Introduction**
- **Experimental Setup**
  - A. The detail of your model
    - EEGNet
    - DeepConvNet
  - B. Explain the activation function (ReLU, Leaky ReLU, ELU)
- **Experimental Result**
  - A. The highest testing accuracy
    - Screenshot with two models
    - Anything you want to present
  - B. Comparison Figures
    - EEGNet
    - DeepConvNet
- **Discussion and extra experiments**

Introduction

- **Experimental Setup**
  - A. The detail of your model
    - ◆EEGNet
    - ◆DeepConvNet
  - B. Explain ReLU, Leaky ReLU, ELU

在Experimental Setup中，會將pytorch實作的EEGNet及DeepConvNet兩個model的architecture呈現出來，以及說明ReLU、LeakyReLU、ELU的內涵。

- **Experimental Result**
  - A. The highest testing accuracy
    - ◆Screenshot with two models
    - ◆Anything you want to present
  - B. Comparison Figures
    - ◆EEGNet
    - ◆DeepConvNet

在Experimental Result中，首先會呈現出兩個model搭配三種不同激活函數的訓練中，出現最好的test data accuracy。再來會將兩個model的training history圖以accuracy的變化呈現出來。

然後得到最好的結果是EEGNet_ReLU的效果最佳，test_acc達到87.13%。

- **Discussion and extra experiments**

最後，based on EEGNet_ReLU，實驗不同的batch_size、learning_rate、kernel_size的影響，再嘗試提高model效能。

最後面會說明我目前所有實驗中的最佳結果：88.15%

Experimental Setup

Experimental Setup-Detail of EEGnet

- EEGNet implementation details

```
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)
  )
  (depthwiseConv): 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): ReLU()
    (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): ReLU()
    (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)
  )
)
```

Feature map的維度演變如下：

Input → 1x2x750

firstConv → 16x2x750

Depthwise → 32x1x750

Avgpooling → 32x1x187

Separable → 32x1x187

Avgpooling → 32x1x23

=>所以Classify的

in_features =32x1x23 =736

Experimental Setup-Detail of DeepConvNet

```
DeepConvNet(
  (firstConv): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), bias=False)
  )
  (block1): Sequential(
    (0): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), groups=25, bias=False)
    (1): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (block2): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (block3): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (block4): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=8600, out_features=2, bias=True)
  )
)
```

Feature map的維度演變如下：

Input → 1x2x750

firstConv → 25x2x746

Block1

Conv2d → 25x1x746

Maxpooling → 25x1x373

Block2

Conv2d → 50x1x369

Maxpooling → 50x1x184

Block3

Conv2d → 100x1x180

Maxpooling → 100x1x90

Block4

Conv2d → 200x1x86

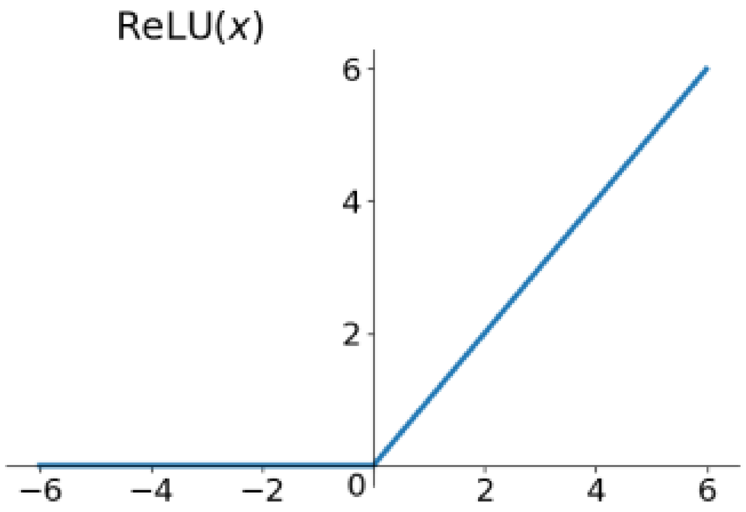
Maxpooling → 200x1x43

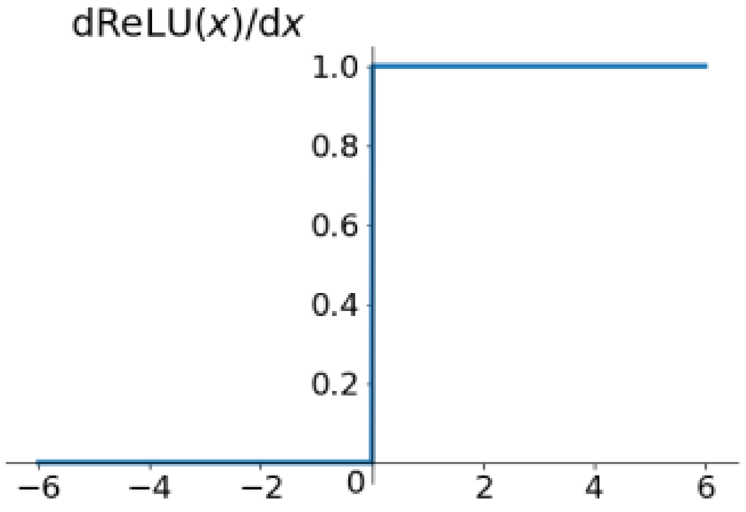
=>所以Classify的

in_features =200x1x43 =8600

Experimental Setup-Explanation of ReLU

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





ReLU函數從公式上來看，其實每個x值與0比然後取其大的函數，函數圖型如上圖，而其導數如下圖所示，而ReLU優點如下：

- 在正值區間明顯解決了gradient vanishing問題。
- 只需判斷輸入是否大於0，所以計算速度非常快。
- 綜合上述兩點，模型收斂速度會遠快於sigmoid和tanh

另外有所謂Dead ReLU Problem的情況，指的是某些neuron可能永遠不會被activate，導致相應的weights很可能永遠不能被更新。經文獻說明有兩個主要原因可能導致這種情況產生:

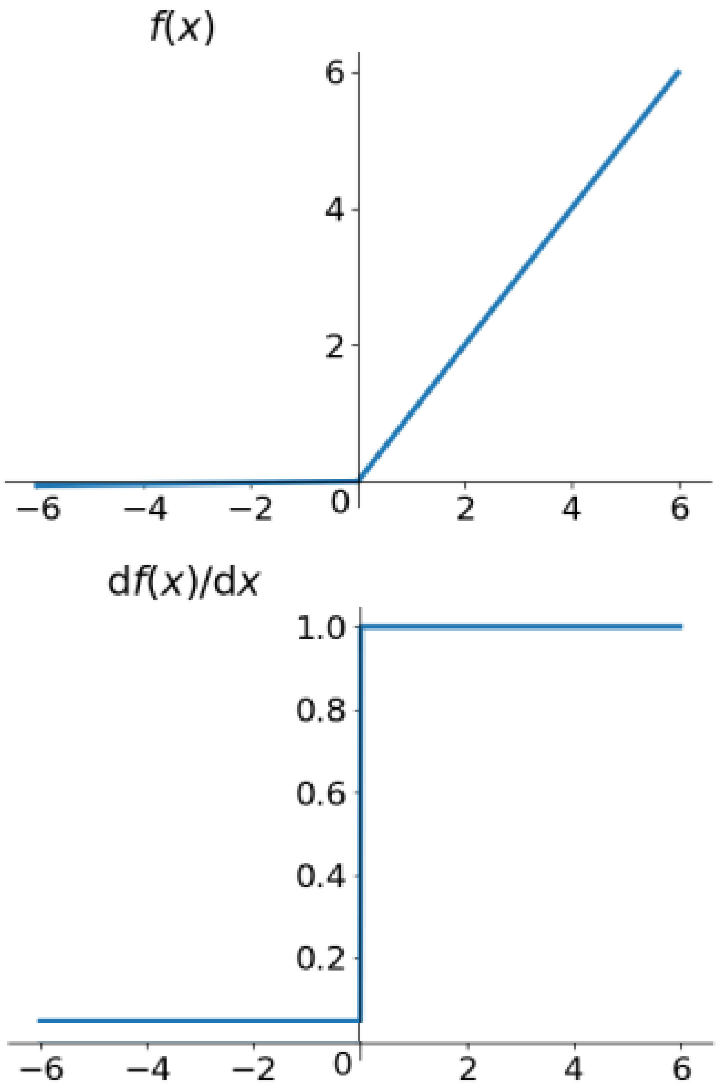
(1) 非常不幸的參數初始化所造成，但這種情況比較少見。

(2) learning rate太高導致在訓練過程中參數更新太大，提高神經網絡進入這種狀態的可能性。

解決方法是可以採用Xavier初始化方法，以及避免將learning rate設置太大或使用adagrad optimizer等隨epoch自動調節learning rate的參數更新法。

Experimental Setup-Explanation of leakyReLU

$f(x) = \max(0.01x, x)$



前人為了解決Dead ReLU Problem而提出了leakyReLU的激活函數。

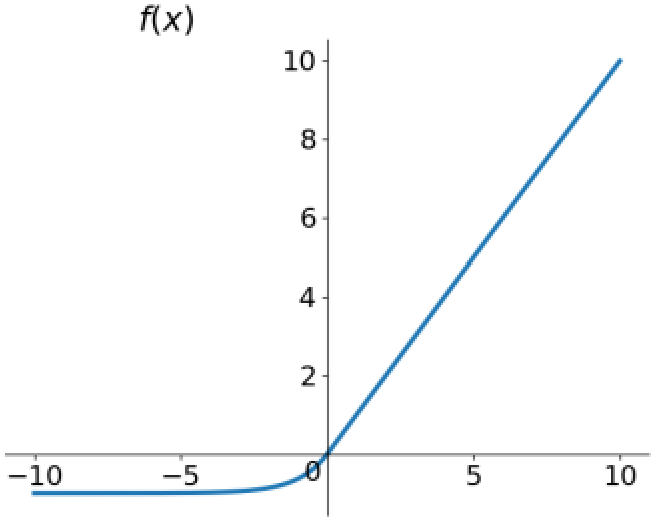
上圖即為leakyReLU的函數圖示例，不同於原始的ReLU，其在負值區間的直線斜率不是零，而是小小的0.01。下圖即為leakyReLU的函數圖。

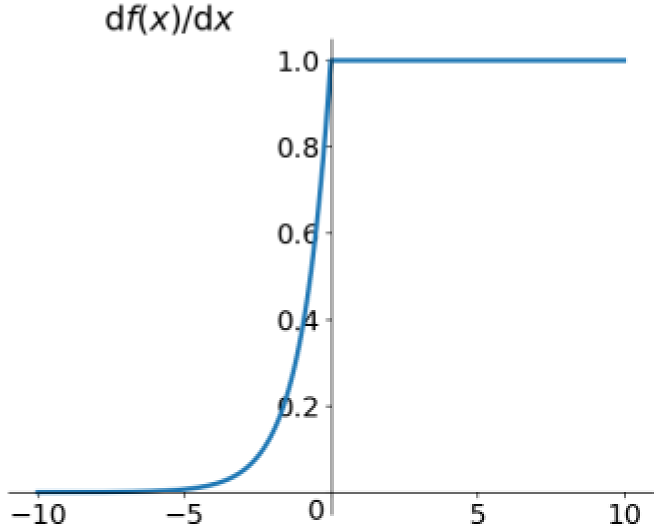
理論上來說，Leaky-ReLU有ReLU的所有優點，並且沒有Dead ReLU問題，然而在實際操作當中，並沒有Leaky-ReLU總是好於ReLU的證明。

補充：另外有種直觀的做法，是基於參數的方法，即Parametric ReLU: f(x) = max(ax, x)，其中a可以由backpropagation的過程中一起learn出來。

Experimental Setup-Explanation of ELU

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





ELU也是為解決ReLU的問題而被提出，上圖為ELU的函數圖，下圖則為ELU的導數曲線圖。

顯然ELU有ReLU的所有基本優點，以及不會有Dead ReLU problem。

然而它的一個小問題在於計算量稍大。類似Leaky ReLU，理論上雖然好於ReLU，但在實際使用中，到目前並沒有好的證據來證明ELU總是優於ReLU。

Experimental Result

part1. basic results

Experimental Result – highest test_acc

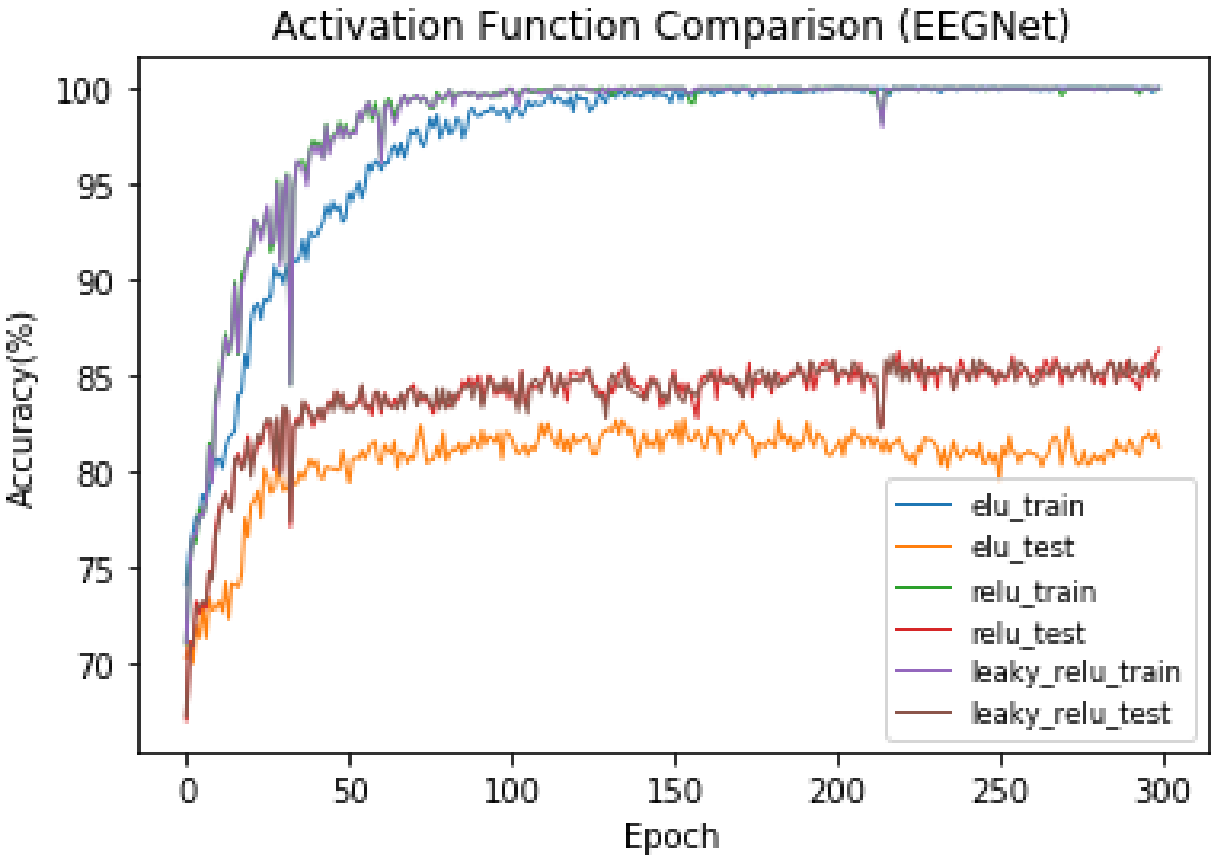
```
###### Training settings #######
parser = argparse.ArgumentParser(description='PyTorch DLP_Lab2_BCI_training')
parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                    help='input batch size for training (default: 64)')
parser.add_argument('--epochs', type=int, default=500, metavar='N',
                    help='number of epochs to train (default: 300)')
parser.add_argument('--lr', type=float, default=0.001, metavar='LR',
                    help='learning rate (but lr will decay from default 0.001)')
```

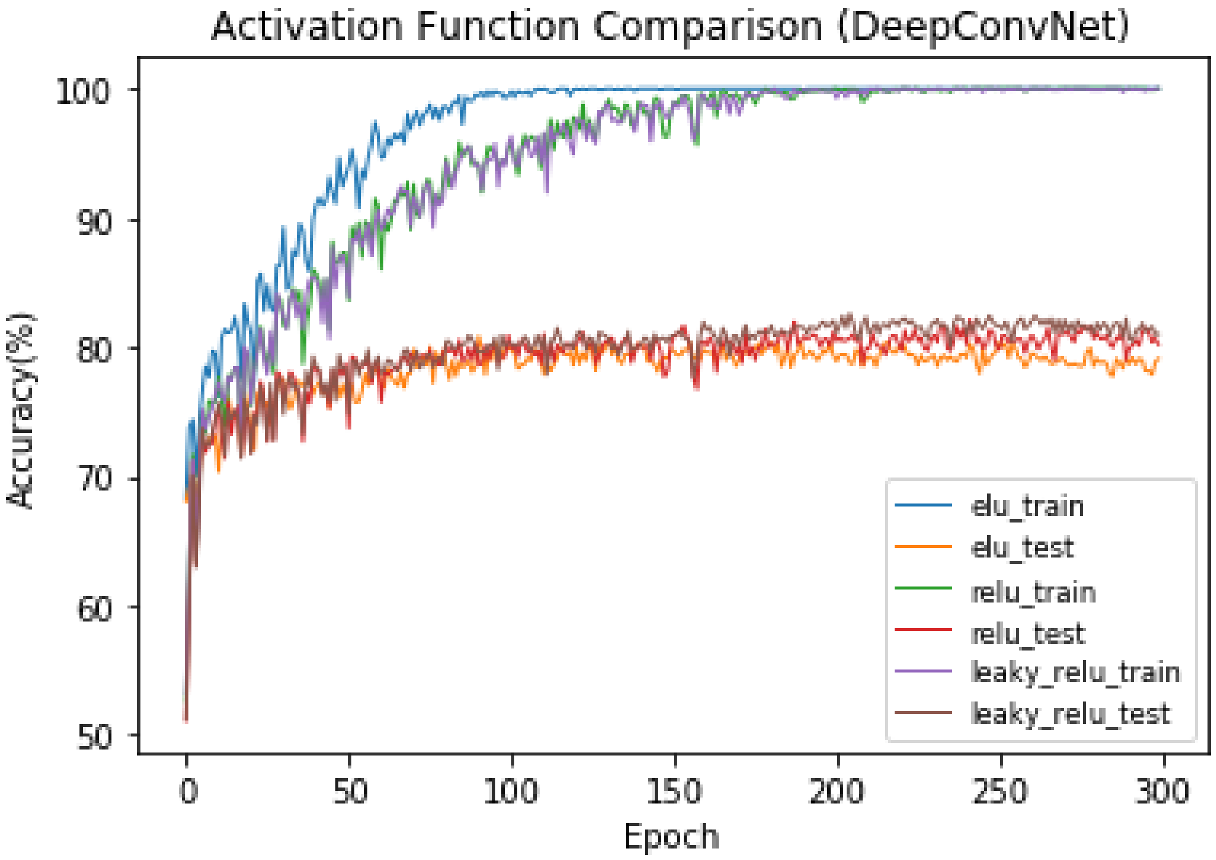
- Batch size= 64
- Learning rate = 1e-3
- Epochs = 500
- Optimizer: Adam

	EEGNet	DeepConvNet
ELU	Result of EEGnet_ELU: The best test accuracy is 82.78% at epoch=153 Training time taken: 1.0 minutes 9.4 seconds	Result of DeepConvNet_ELU: The best test accuracy is 81.67% at epoch=353 Training time taken: 1.0 minutes 44.6 seconds
ReLU	Result of EEGNet_ReLU: The best test accuracy is 87.13% at epoch=413 Training time taken: 1.0 minutes 9.0 seconds	Result of DeepConvNet_ReLU: The best test accuracy is 82.31% at epoch=303 Training time taken: 1.0 minutes 44.5 seconds
Leacky-ReLU	Result of EEGnet_LeakyReLU: The best test accuracy is 87.13% at epoch=483 Training time taken: 1.0 minutes 9.3 seconds	Result of DeepConvNet_LeakyReLU: The best test accuracy is 82.69% at epoch=332 Training time taken: 1.0 minutes 43.7 seconds

實驗結果：EEGNet with ReLU和leakyReLU都達到87.13%，但ReLU在較低的epoch數達到87.13%

Result Comparison





實驗結果：以上實驗除了激活函數外，其它參數都一樣，並且我也將GPU的隨機參數seed設成1，也就是讓初始化都一樣。而會發現在左圖EEG的結果中，ReLU和leakyReLU的在training過程的趨勢非常接近。而ELU在實驗中的表現，雖然在DeepConvNet的收斂速度最快，但在這兩個model過程的accuracy均表現最差。

Discussion and
Extra experiments

Discussion and extra experiments-1

- Learning rate = 1e-3
- Epochs = 500
- Optimizer: Adam

比較Batch size= 64, 32, 16

Batch 64
```
Result of EEGnet_ReLU:
The best test accuracy is 87.13% at epoch=413

Training time taken: 1.0 minutes 9.0 seconds
```

Batch 32
```
Result of EEGnet_ReLU:
The best test accuracy is 86.30% at epoch=499

Training time taken: 2.0 minutes 9.1 seconds
```

Batch 16
```
Result of EEGnet_ReLU:
The best test accuracy is 86.85% at epoch=299

Training time taken: 3.0 minutes 46.9 seconds
```

結果發現batch64的效果最佳：

1. Test accuracy超過87%
2. 因為iteration較少，所以training速度快。

- Batch size= 64
- Epochs = 500
- Optimizer: Adam

比較Learning rate = 1e-4, 1e-3, 1e-2

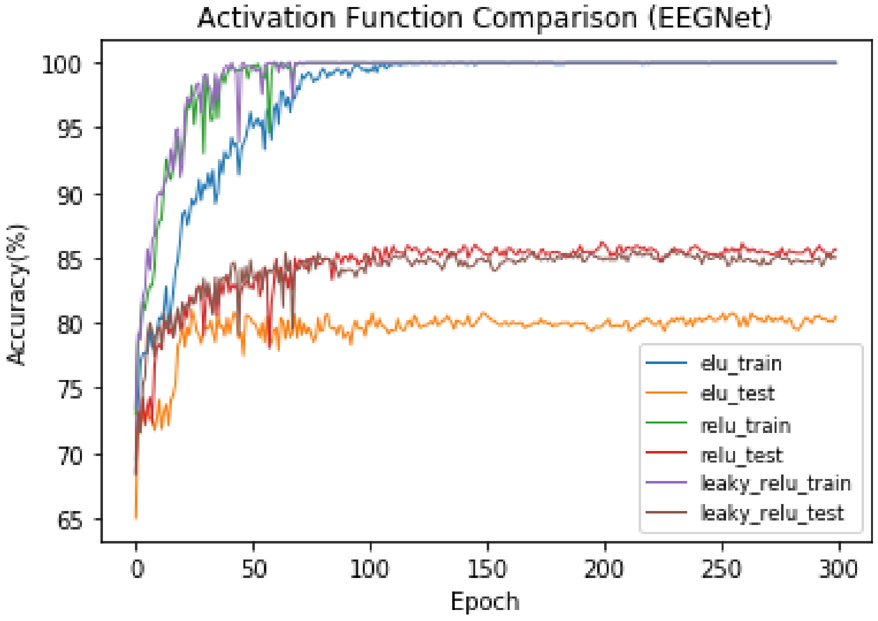
```
Result of EEGnet_ReLU:
The best test accuracy is 86.94% at epoch=416

Training time taken: 1.0 minutes 8.0 seconds
```
Lr =1e-4

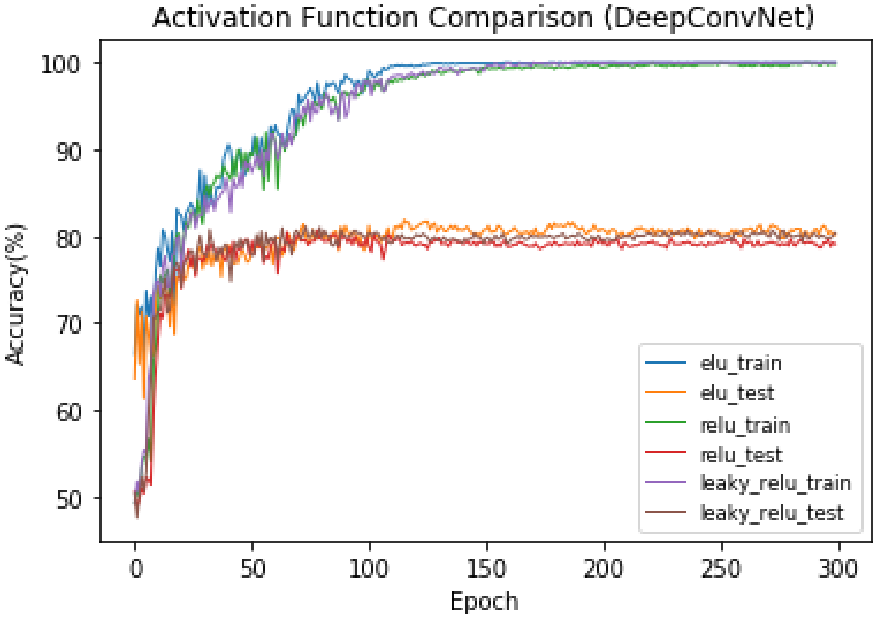
```
Result of EEGnet_ReLU:
The best test accuracy is 87.13% at epoch=413

Training time taken: 1.0 minutes 9.0 seconds
```
Lr =1e-3

```
Result of EEGnet_ReLU:
The best test accuracy is 87.22% at epoch=445

Training time taken: 1.0 minutes 8.0 seconds
```
Lr =1e-2

雖然Lr =1e-2達到最高的87.22%，但整個training過程中的fluctuation較Lr =1e-3來的大。

Discussion and extra experiments-2






```python
#learning rate scheduling
def adjust_learning_rate(optimizer, epoch):
    if epoch < 70:
        lr = 0.01
    elif epoch < 110:
        lr = 0.005
    elif epoch < 160:
         lr = 0.001
    else:
        lr = 0.0005

    for param_group in optimizer.param_groups:
        param_group['lr'] = lr
```

其實這兩個model我都有嘗試隨著epoch數增加而不同階段調低learning rate，如左圖示例。而這方面的實驗結果是：雖然training過程中model表現穩定(可能因lr變小)，但accuracy最高都沒有超過86.5%。

Discussion and extra experiments-3

```python
'''############# Define EEGnet here ###############'''
class EEGNet(nn.Module):
    def __init__(self, Activate):
        super(EEGNet, self).__init__()
        self.alpha = 1
        self.cvks1 = 61
        self.cvks2 = (2,1)
        self.cvks3 = 25
        self.out_ch1 = int(16*self.alpha)
        self.out_ch2 = int(32*self.alpha)
        self.out_ch3 = int(32*self.alpha)
```

(61,25)

```
Result of EEGnet_ReLU:
The best test accuracy is 88.15% at epoch=286

Training time taken: 1.0 minutes 7.9 seconds
```

(51,15)

```
Result of EEGnet_ReLU:
The best test accuracy is 87.13% at epoch=413

Training time taken: 1.0 minutes 9.0 seconds
```

(41,7)

```
Result of EEGnet_ReLU:
The best test accuracy is 86.11% at epoch=360

Training time taken: 1.0 minutes 8.1 seconds
```

最後再嘗試改變kernel_size的長度來實驗不同receptive field的影響：

發現同時firstConv的51提高到61、SeparableConv的15提高到25，可以得到目前所有實驗中最佳的結果88.15%

補充：這裡發現receptive field對BCI這個time series的影響頗大，例如左圖中當隨著kernel_size的變小，accuracy也隨之下降。

The End