DLP Lab2 Report

0756138 黃伯凱

April 10, 2019

1 Introduction

這次要實作出兩個Convolution Neural Network，分別是EEGNet以及DeepConvNet，用的Dataset是BCI Competition III - IIIb。這一個dataset是EEG dataset，每一筆data都有兩個channel，target為判斷出是左手的訊號還是右手的訊號，所以只有兩個class。

在這篇報告中，我會比較EEGNet以及DeepConvNet，也用了幾種Dropout參數和optimizer試試看哪種對我的model比較好，最後會以圖片呈現比較的結果。

此題目是2016年出自於Cornell University的一篇paper，連結如下：

EEGNet: A Compact Convolutional Network for EEG-based Brain-Computer Interfaces

Dataset出自於BCI Competition III - IIIb，連結如下：

BCI Competition III - IIIb

2 Experiment

在這個章節中，會討論兩種Neural Network的架構，也會介紹3種不同的activation function。

2.1 Detail of Model

2.1.1 EEGNet

paper中EEGNet的架構圖如Figure 1.

Block	Layer	# filters	Size	# params	Output	Activation	Options
1	Input				(C, T)		
	Reshape				(1, C, T)		
	Conv2D	F_1	(1, 64)	$64 * F_1$	(F_1, C, T)	Linear	Mode = same
	BatchNorm			$2 * F_1$	(F_1, C, T)		
	DepthwiseConv2D	D $* F_1$	(C, 1)	$C * D * F_1$	(D $* F_1$, 1, T)	Linear	Mode = valid, depth = D, max norm = 1
	BatchNorm			$2 * D * F_1$	(D $* F_1$, 1, T)		
	Activation				(D $* F_1$, 1, T)	ELU	
	AveragePool2D		(1, 4)		(D $* F_1$, 1, T // 4)		
	Dropout*				(D $* F_1$, 1, T // 4)		$p = 0.25$ or $p = 0.5$
2	SeparableConv2D	F_2	(1, 16)	$16 * D * F_1 + F_2 * (D * F_1)$	(F_2, 1, T // 4)	Linear	Mode = same
	BatchNorm			$2 * F_2$	(F_2, 1, T // 4)		
	Activation				(F_2, 1, T // 4)	ELU	
	AveragePool2D		(1, 8)		(F_2, 1, T // 32)		
	Dropout*				(F_2, 1, T // 32)		$p = 0.25$ or $p = 0.5$
	Flatten				($F_2 *$ (T // 32))		
Classifier	Dense	N $*$ ($F_2 *$ T // 32)			N	Softmax	Max norm = 0.25

Figure 1: EEGNet Architecture

我的EEGNet的實作，前幾行的activations dictionary是為了找最佳Accuracy時可以彈性改變activation function而加入。

```
activations = {
    'ReLU': nn.ReLU(),
    'LeakyReLU': nn.LeakyReLU(),
    'ELU': nn.ELU()
}

class EEGNet(nn.Module):
    def __init__(self, activation_function, dropout):
        super(EEGNet, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,      # input height
                out_channels=16,    # n_filters
                kernel_size=(1,51),      # filter size
                stride=1,           # filter movement/step
                padding=(0, 25),
            ),
            nn.BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, (2,1), 1, groups=16, bias=False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            activations[activation_function],
            nn.AvgPool2d(kernel_size=(1,4), stride=(1,4), padding=0),
            nn.Dropout(p=dropout)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(32, 32, (1,15), 1, padding=(0,7), bias=False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            activations[activation_function],
            nn.AvgPool2d(kernel_size=(1,8), stride=(1,8), padding=0),
            nn.Dropout(p=dropout)
        )
        self.out = nn.Sequential(nn.Linear(in_features=736, out_features=2, bias=True))

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output
```

Figure 2: Code Snippet for EEGNet

2.1.2 DeepConvNet

paper中的DeepConvNet架構如Figure 3.

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 = 1×10^{-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 = 1×10^{-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 = 1×10^{-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 = 1×10^{-05}, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					$p = 0.5$
Flatten					
Dense	N			Softmax	Max norm = 0.5

Figure 3: DeepConvNet Architecture

```
activations = {
    'ReLU': nn.ReLU(),
    'LeakyReLU': nn.LeakyReLU(),
    'ELU': nn.ELU()
}

class DeepConvNet(nn.Module):
    def __init__(self, activation_funtion, dropout):
        super(DeepConvNet, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,      # input height
                out_channels=25,    # n_filters
                kernel_size=(1,5),      # filter size
                stride=1,           # filter movement/step
            ),
            nn.Conv2d(25, 25, (2,1)),
            nn.BatchNorm2d(25, eps=1e-05, momentum=0.1),
            activations[activation_funtion],
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=dropout)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(25, 50, (1,5)),
            nn.BatchNorm2d(50, eps=1e-05, momentum=0.1),
            activations[activation_funtion],
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=dropout)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(50, 100, (1,5)),
            nn.BatchNorm2d(100, eps=1e-05, momentum=0.1),
            activations[activation_funtion],
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=dropout)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(100, 200, (1,5)),
            nn.BatchNorm2d(200, eps=1e-05, momentum=0.1),
            activations[activation_funtion],
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=dropout)
        )
        self.out = nn.Sequential(nn.Linear(in_features=8600, out_features=2, bias=True))

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output
```

Figure 4: Code Snippet for DeepConvNet

2.2 Activation Function

上一次Lab並沒有指定我們要用什麼activation function，所以在上次我都是使用Sigmoid Function當作Neural Network的activation function，但其實Sigmoid Function有蠻多缺點。

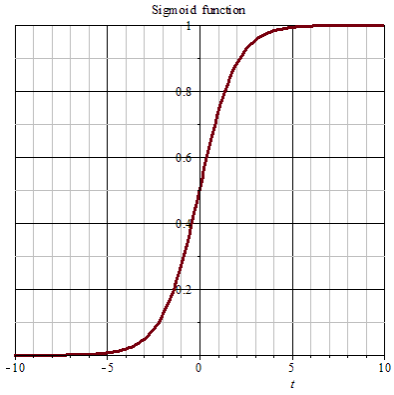

Figure 5: Sigmoid Function

第一個問題就是他的output並不是zero-centered，而是全部介於0跟1之間。這導致了gradient全部都是正的或者全部都是負的，而最後會使得optimization更困難。

第二個問題也是最嚴重的問題**Vanishing Gradient**。Sigmoid兩側的gradient差異太大，如果weight初始化得太大，activation value基本上都會出現在sigmoid的兩側。而兩側的gradient又幾乎為0，使得backpropagation時幾乎就都沒有gradient了。即使使用很好的初始化方法把activation value控制在某個合理範圍內，幾個epoch後又會有幾個neuron跑到兩側了。而一旦到兩側，因為gradient過小，就再也無法通過更新gradient來使其恢復。這個問題在提出ReLU後可以有效解決。接下來就討論ReLu以及其改良版的LeakyReLU和ELU。

2.2.1 ReLU

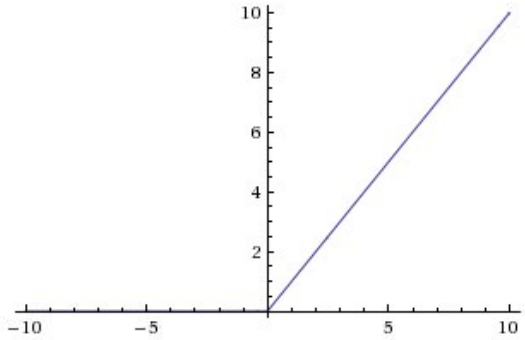

Figure 6: ReLU

ReLU的其中一個優點就是他的右側是線性的，所以他的gradient是常數。相較於sigmoid function的gradient隨著input值而改變，使用ReLU的neural network在學習的時候可以學得比較快。

此外，ReLU的左側都是零，所以當input value小於0的時候，output通通都會變成0。這樣可以讓backpropagation的計算更快速，因為從原本的dense matrix變成sparse matrix。

但是sparse matrix的優點同時也是ReLU的缺點，要是input value通通都小於0的時候，所有的activation value會全部變成0，因此gradient也跟著消失了，這樣的現象稱為*dead ReLU*。此外，ReLU和Sigmoid一樣也不是zero-centered。為了解決這些的問題，於是有了LeakyReLU。

2.2.2 LeakyReLU

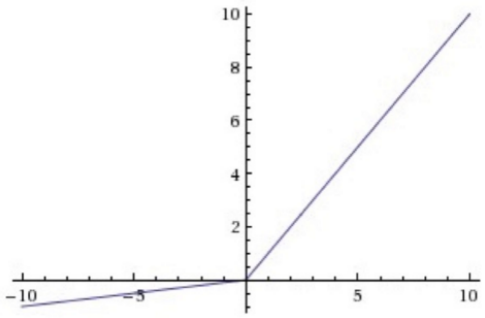

Figure 7: LeakyReLU

LeakyReLU的左右兩側都是線性的，它與ReLU一樣gradient都是常數，所以network可以學得很快。此外，他的左側不是0而是斜率很小的直線，所以他也沒有dead ReLU的問題。只是它就失去了ReLU的sparse matrix的優點。

2.2.3 ELU

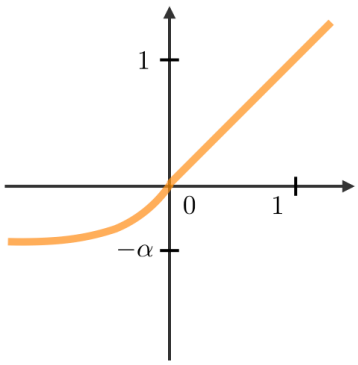

Figure 8: ELU

ELU的左側和ReLU一樣較平緩，不一樣的是他是一個指數函數。和LeakyReLU一樣可以避免dead ReLU的問題，但是由於他的左側是指數函數，所以他的計算量會稍微大一些。

3 Result

3.1 Best Accuracy

為了找到最好的參數選擇，除了題目要求的activation function之外，我調整了neural network中的dropout rate，也試著找最適合model的optimizer。每一次training總共有600個epoch，batch size = 64，learning rate = 0.01。

我嘗試了以下數種組合，並用grid search尋找能夠達到最高accuracy的參數。所以總共嘗試了36個組合的參數。

```
dropout=[0.2, 0.3, 0.4, 0.5]
activation_function = ['ReLU', 'ELU', 'LeakyReLU']
opts = ['Adam', 'SGD', 'RMSprop']
```

Figure 9: Parameter Choice

3.1.1 EEGNet Highest Accuracy

Figure 10. 是EEGNet有最高Accuracy的參數組合。Activation function為ReLU，dropout rate都是0.5，optimizer則是Adam。

Figure 10: Best EEGNet Parameter

大約在第70 epoch以後他的Testing Accuracy就已經達到0.85左右，在第472 epoch的時候達到最高的Accuracy 0.8777。在500 epoch以後也一直保持在86和87之間。

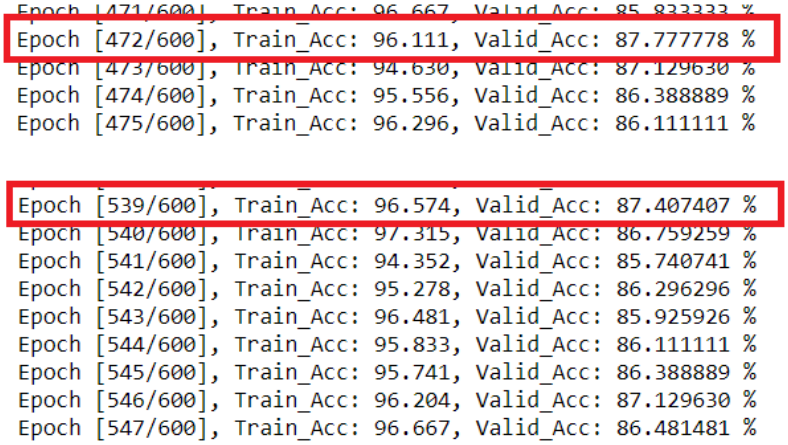

Figure 11: EEGNet Best Accuracy

3.1.2 DeepConvNet Highest Accuracy

Figure 12. 是DeepConvNet有最高Accuracy的參數組合。Activation function為LeakyReLU，dropout rate都是0.3，optimizer則是SGD。

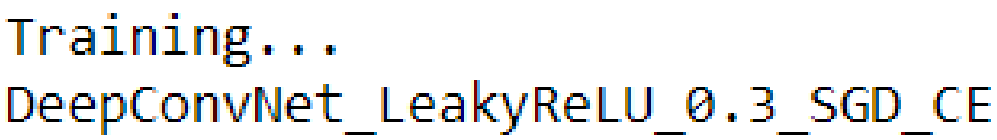

Figure 12: Best EEGNet Parameter

```
Epoch [476/600], Train_Acc: 96.019, Valid_Acc: 83.611111 %
Epoch [477/600], Train_Acc: 96.204, Valid_Acc: 83.240741 %
Epoch [478/600], Train_Acc: 96.019, Valid_Acc: 83.888889 %
Epoch [479/600], Train_Acc: 96.759, Valid_Acc: 84.537037 %
Epoch [480/600], Train_Acc: 96.667, Valid_Acc: 84.166667 %
Epoch [481/600], Train_Acc: 96.574, Valid_Acc: 83.703704 %
Epoch [482/600], Train_Acc: 95.463, Valid_Acc: 83.425926 %
```

Figure 13: Best DeepConvNet Accuracy

DeepConvNet的最高accuracy就沒有辦法像EEGNet那麼高，只有兩個model可以超過0.84，另一組是LeakyReLU，dropout rate = 0.5, optimizer為RMSprop的model。

3.2 Comparison Figures

由於我嘗試的參數非常多，也都有把數據存成csv檔並輸出成圖表，所以下方先放兩個最高accuracy的圖表，接下來兩頁都是我有測試的縮圖。

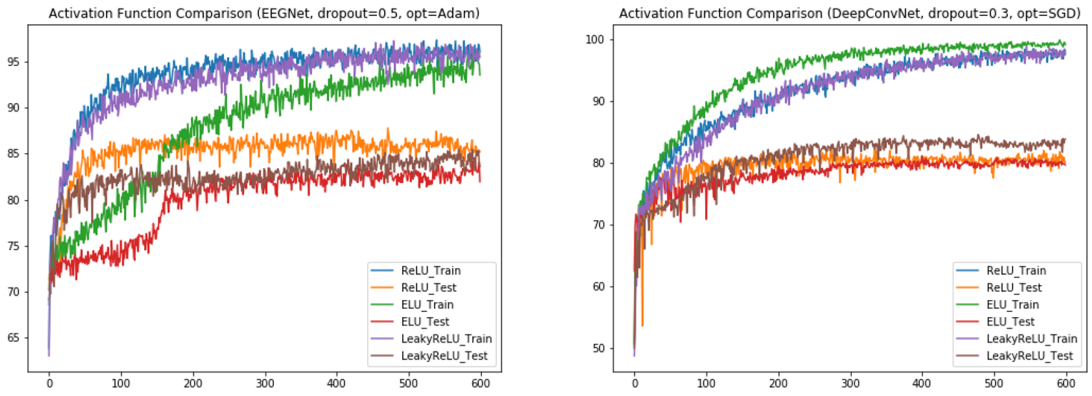

Figure 14: Highest Accuracy Comparison

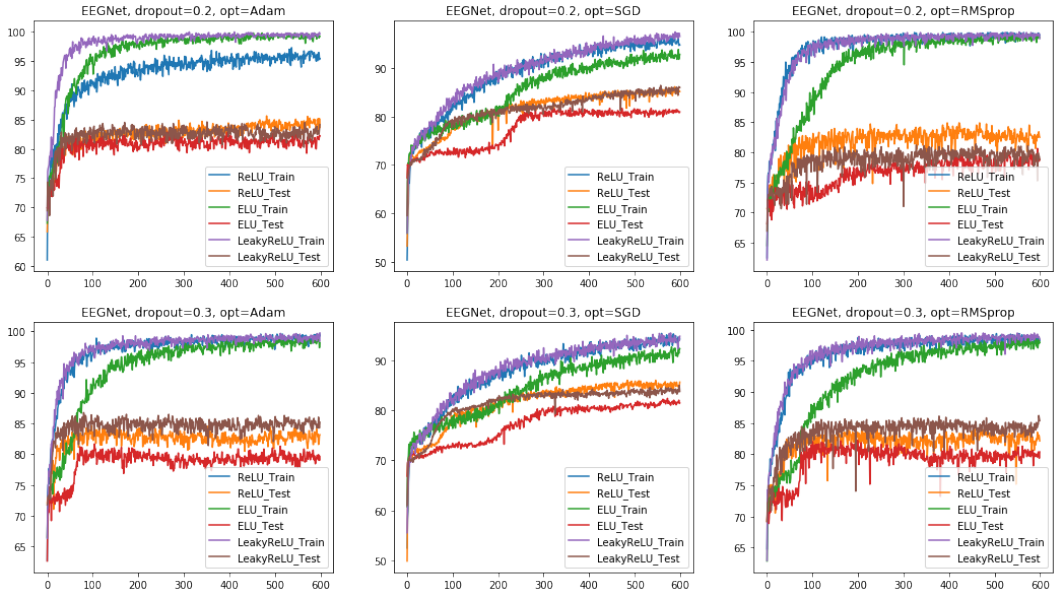

Figure 15: EEGNet Comparison-1

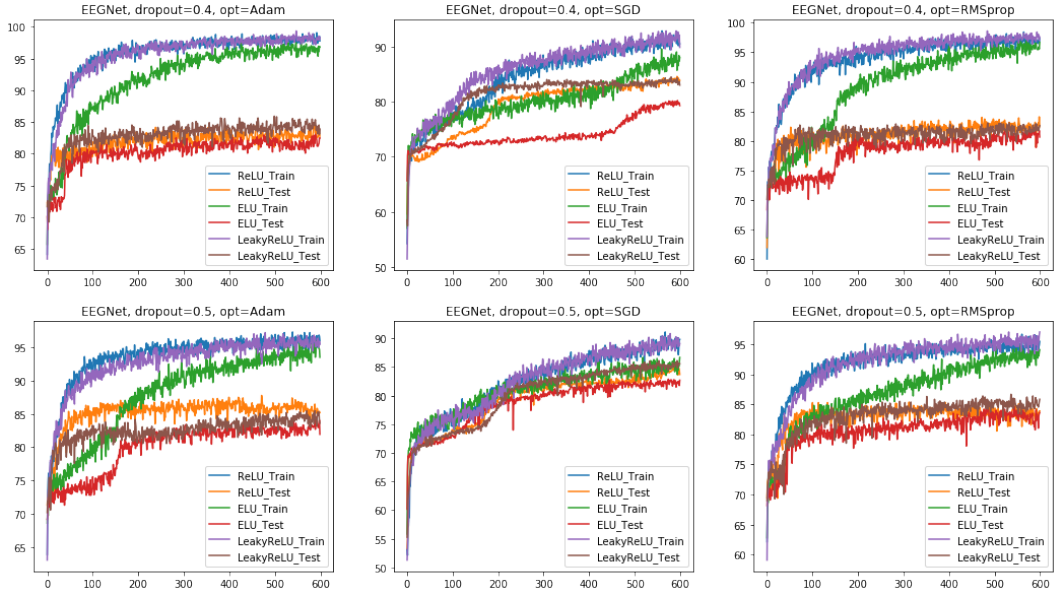

Figure 16: EEGNet Comparison-2

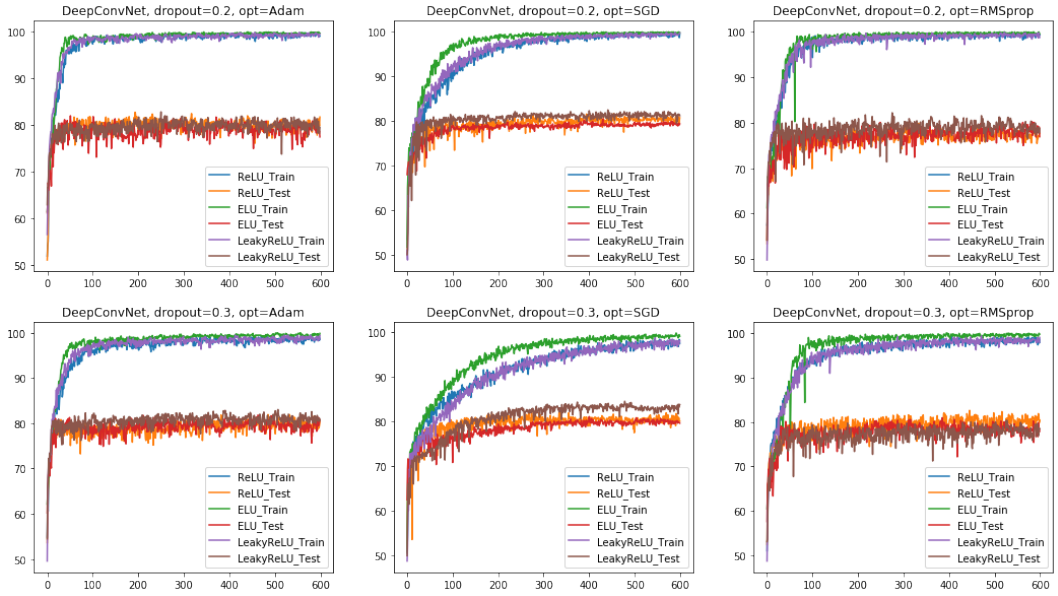

Figure 17: DeepConvNet Comparison-1

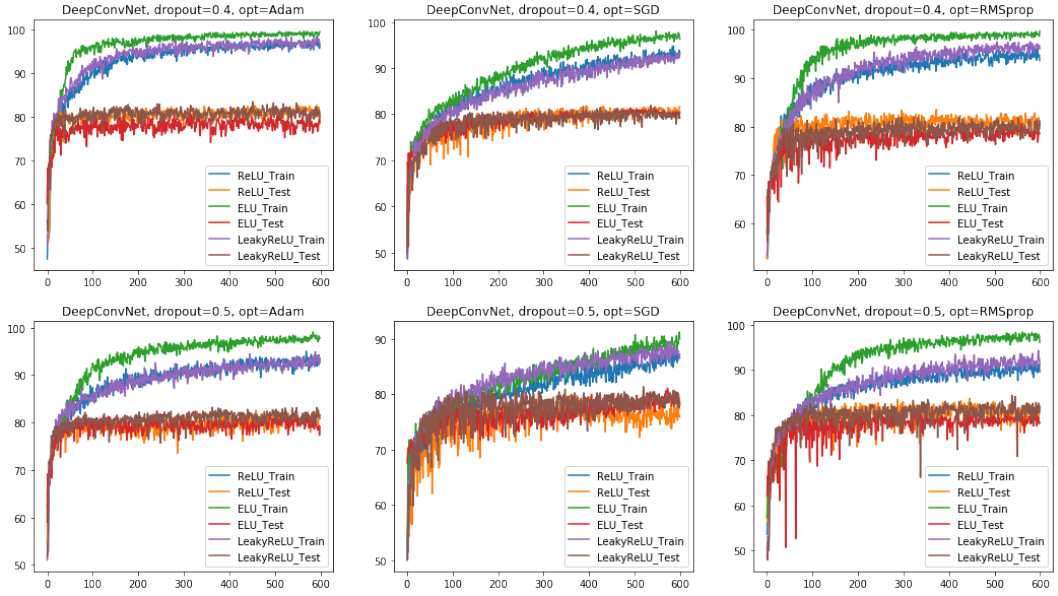

Figure 18: DeepConvNet Comparison-2

4 Discussion

從我的實驗當中可以發現，不論是用哪一個optimizer或者activation function，當我的dropout rate調越高的時候，testing accuracy會跟著越高，但是training rate會相對的降低。Figure 19. 可以明顯的看出來。

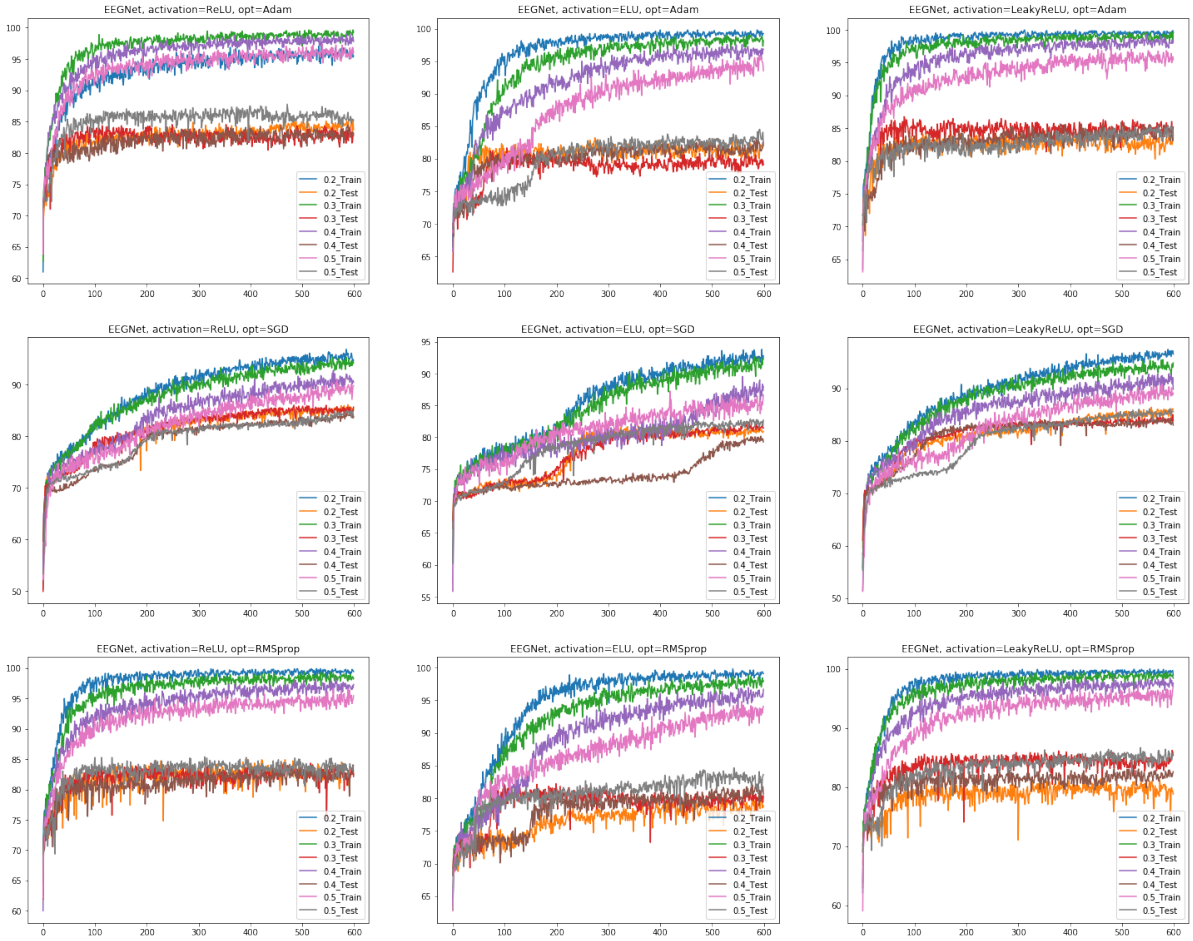

Figure 19: EEGNet Dropout Comparison

Deep neural network的兩大缺點就是：**慢**、**容易overfit**。而dropout的目的就是在解決overfit的問題，他會强迫neuron與隨機挑選出來的其他neuron共同合作，但是neural network要學會的parameter總數量是不變的。

因為加入了隨機的neuron這個變數，所以整個網路架構學起來的model會比較不適用在training data上，導致dropout rate越高的training accuracy會比較低。也因為學出來的model比較general，所以套用在testing data上也會比較好，testing accuracy比較高。

	ReLU	LeakyReLU	ELU
EEGNet	87.77 %	86.75 %	84.62 %
DeepConvNet	83.98 %	84.53 %	82.03 %

Figure 20: Best Accuracy of each Activation Function

根據前面Activation Function的介紹，LeakyReLU和ELU應該要比ReLU好一些，但是從這一次的實驗結果(Figure 20) 看來並不百分之百是這樣。這次的實驗反而是ELU的結果是最差的，而ReLU和LeakyReLU的表現差不多。所以就算從理論上來看或許ReLU比較差，但是實際上還是會因為是不同model而有不同的結果。