Lab3-EEG-Classification

張千祐
Department of Computer Science
National Yang Ming Chiao Tung University
qianyou.cs11@nycu.edu.tw

1 Introduction

在這次 lab 中，目標是希望做 EEG 的分類任務，資料集來源是 BCI Competition III，希望能夠實作兩種不同的神經網路來做預測，分別是 EEGNet 和 DeepConvNet，並且需要嘗試三種不同的 activation function 並比較結果，過程中我們可以自行調整 hyper parameters 來取得更高的準確率，最後畫出準確率對 epoch 曲線來視覺化結果。

2 Experiment setups

2.1 The details of my models

關於兩種 models 的細節都是依照作業 spec 給予的細節實作，並無不同，Figure 1分別是 EEGNet 和 DeepConvNet 的截圖。

2.2 Explain the activation functions

在神經網路當中，activation functions 扮演提供非線性轉換的關鍵因子，在沒有 activation functions 的神經網路當中，每一層的神經元運算只會是線性轉換，而現實中大部分遭遇的問題都是非線性的問題，使得 activation functions 在神經網路當中成了不可或缺的存在。下面會分別介紹 ReLU, Leaky ReLU 和 ELU 三種 activation functions。

2.2.1 ReLU

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

如 Figure 2，Rectified Linear Unit(ReLU) 是常見的 activation function，使用上無論是函數值或是微分值的計算都很快速，只需要判斷是否大於 0，缺點是會有 dying ReLU problem，因爲在數值小於 0 的地方微分值也皆爲 0，因此一開始若 learning rate 過大，網路的初始權重又剛好使得輸出是負數，很可能導致神經元永遠不會更新數值。

2.2.2 Leaky ReLU

$$\text{Leaky ReLU}(x, \alpha) = \begin{cases} x, & \text{if x} \geq 0 \\ \alpha x, & \text{otherwise} \end{cases}$$

如 Figure 2，Leaky ReLU 修正了 dying ReLU problem，與 ReLU 不同的地方是在小於 0 的地方函數值不爲 0，而是 αx，這麼一來小於 0 的地方微分值也不會爲 0，使得神經元能夠透過訓練階段的 back propagation 被更新，通常 α 預設是 0.01。

2.2.3 ELU

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

```
network = EEGNet("relu")
print(network)
```

[10] ✓ 0.0s

```
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)
  )
  (seperableConv): 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)
  )
)
```

(a) EEGNet

```
network = DeepConvNet("relu")
print(network)
```

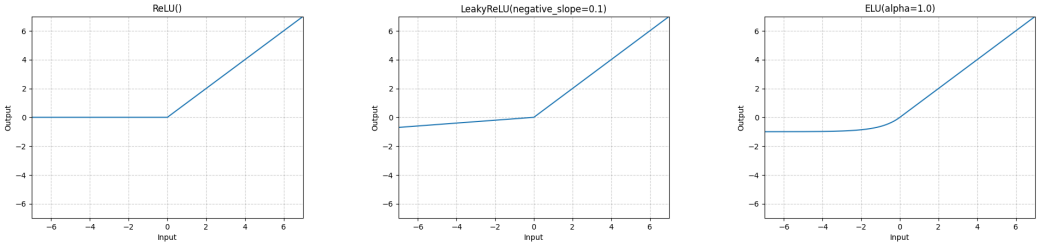
[11] ✓ 0.1s

```
DeepConvNet(
  (layers): ModuleList(
    (0): Sequential(
      (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), padding=valid)
      (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), padding=valid)
      (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (3): ReLU()
      (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
      (5): Dropout(p=0.5, inplace=False)
    )
    (1): Sequential(
      (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), padding=valid)
      (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
      (4): Dropout(p=0.5, inplace=False)
    )
    (2): Sequential(
      (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1), padding=valid)
      (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
      (4): Dropout(p=0.5, inplace=False)
    )
    (3): Sequential(
      (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1), padding=valid)
      (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
      (4): Dropout(p=0.5, inplace=False)
    )
    (4): Sequential(
      (0): Flatten(start_dim=1, end_dim=-1)
      (1): Linear(in_features=8600, out_features=2, bias=True)
    )
  )
)
```

(b) DeepConvNet

Figure 1: Screenshot of architectures of models

如 Figure 2，ELU 是另外一種解決 dying ReLU problem 的 activation function，而且在 x=0 的位置也有微分值。



(a) ReLU (b) LeakyReLU (c) ELU

Figure 2: Function value of ReLU, Leaky ReLU and ELU

3 Experimental results

3.1 The highest testing accuracy

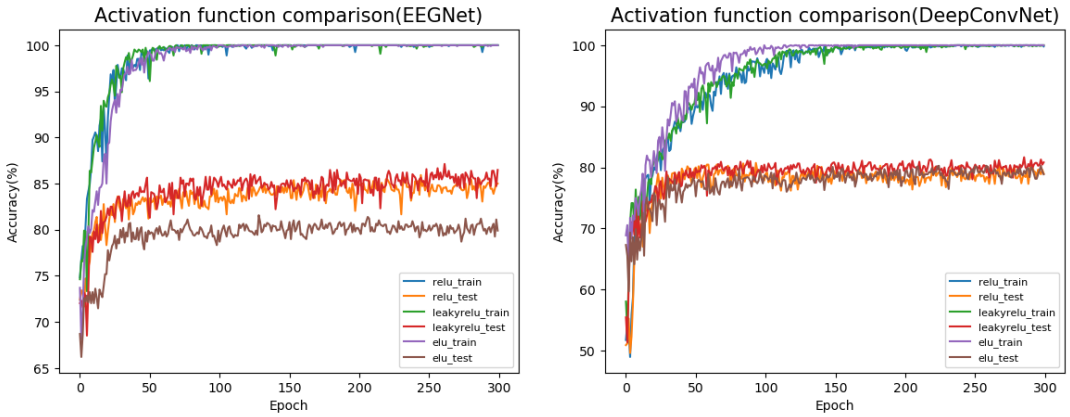
以最高的 testing accuracy 來看，當固定 activation function 時，可以看到 EEGNet 在三種不同的 activation functions 之中都是表現較佳的；而固定 network 後，則是 Leaky ReLU 提供了最好的表現，ReLU 次之，ELU 則是最差。

	ReLU	Leaky ReLU	ELU
EEGNet	85.93%	87.13%	81.57%
DeepConvNet	80.74%	81.67%	80.28%

Table 1: Table of the highest testing accuracy
∗ lr=3e-5, Epochs=300

3.2 Comparison figures

從訓練圖來看，可以看到兩張圖都是差不多在 Epoch=100 時，testing accuracy 就不會再上升。從 activation function 的角度作比較，可以看到 DeepConvNet 在三種不同 activation function 的表現較有一致性，然而 EEGNet 則是受 activation function 不同，影響較大。



(a) EEGNet (b) DeepConvNet

Figure 3: Screenshot of comparison result

4 Discussion

因爲在 activation function 那邊提到當 lr 設定較大時，使用 ReLU 當作 activation function 可能會無法更新神經元，因此我這邊分別用 lr=5e-3(original), 1e-1, 5e-1 來做比較，看當 lr 越大時是否表現就眞的越差，這邊將以最高的 testing accuracy 作呈現。

lr	EEG-ReLU	EEG-Leaky	EEG-ELU	Deep-ReLU	Deep-Leaky	Deep-ELU
5e-3(original)	85.93%	87.13%	81.57%	80.74%	81.67%	80.28%
1e-1	81.76%	81.94%	80.37%	50.28%	77.32%	80.83%
5e-1	73.24%	73.06%	73.61%	54.44%	74.54%	80.09%

Table 2: The highest testing accuracy of different lr

可以看到 Table 2，跟原始的 lr 比起來，很明顯地在 lr 越大時，大部分的表現都變差，但相對的來說，以 ReLU 作爲 activation function 的表現變差的幅度更多，尤其是在 Deep-ConvNet 版本的 ReLU，從一開始能夠到達約 80% 左右，在 lr 變大時只剩下約 50% 的最高準確率，也驗證了我前面的說法。