Label 2 : EEG classification

Outline

1. Introduction

2. Experiment setups

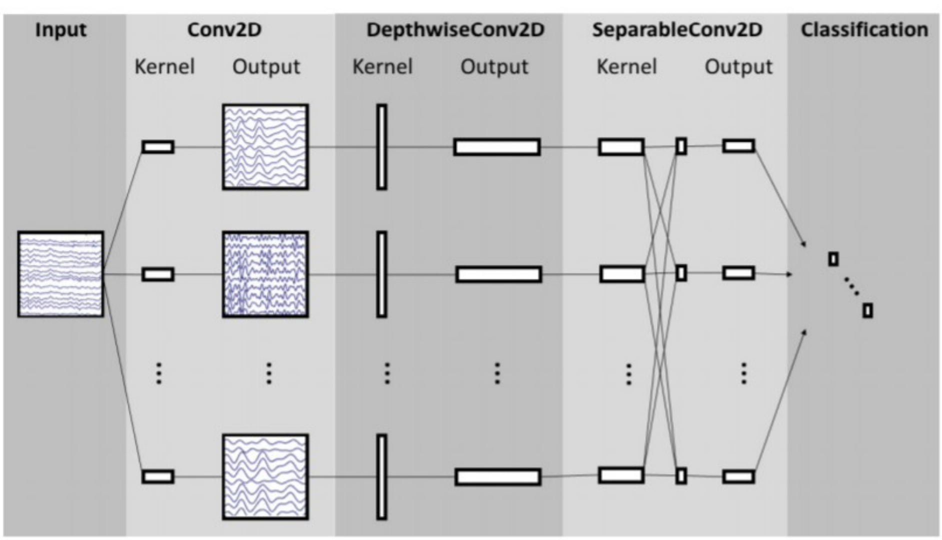
3. Experiment results

4. Discussion

1. Introduction

根據 BCI(Brain-Computer Interfaces)的 Dataset, 是針對人類在行為或是看到某見事物時所產生腦波訊號, 運用神經網路來學習這些訊號來運用到用腦波操作機器等可能性。實驗運用到兩個神經網路架構：EEG, DeepConvNet, 搭配三種激勵函數 ELU, ReLU, LeakyReLU, 總共六種組合來獲得最高的準確度。

EEG Net:



架構整體分成四層 Conv2D、 DepthwiseConv2D、 SeparableConv2D、 Classification, 下面分別介紹各層的功用:

1. Conv2D: 與一般的 Cnn 一樣是用來先將輸入做幾種特徵的分類已好後續的各層的分析。
2. DepthwiseConv2D: 將第一層的 Conv2D 後的各項特徵做頻率濾波, 使用的是深度捲積的方式, 已學習到特定的頻率的所產生的回饋。
3. SeparableConv2D: 也同樣是深度捲積, 但不同的是要逐點捲積, 並將學習如何將特徵做最佳的混合在一起。
4. Classification: 是使用線性的網路的方式將特徵做全盤的分類, 以此實驗來說輸出是 0 或 1 的可能性分類。

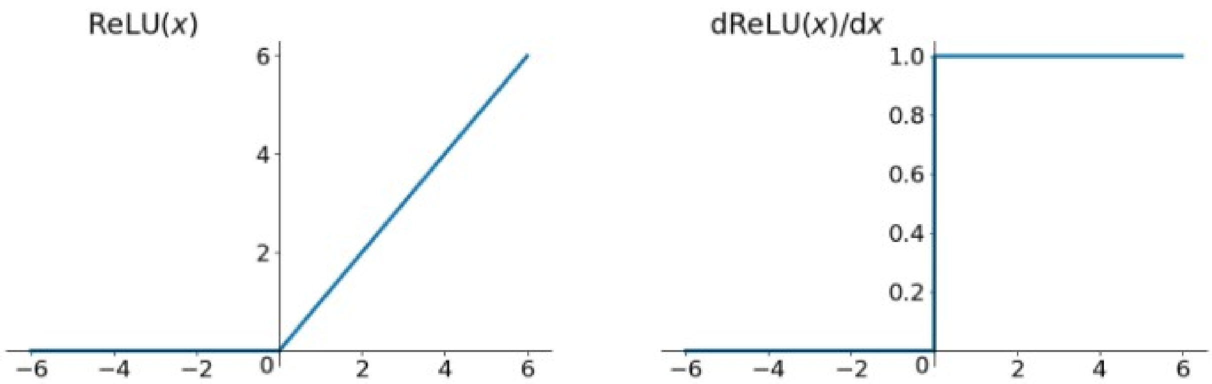
根據 Paper, Loss Function 和 Optimizer 分別是使用 cross-entropy 以及 Aadm。

DeepConvNet

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

整體架構分了五層，前面四層皆不段的特徵做跟深一層的分類，在最後一層用 Softmax 做最後的輸出，但因應此實驗最後還會添加一層 Classification 做輸出，與 EEG 一樣的設定。同樣的 Loss Function 和 Optimizer 分別是使用 cross-entropy 以及 Aadm。

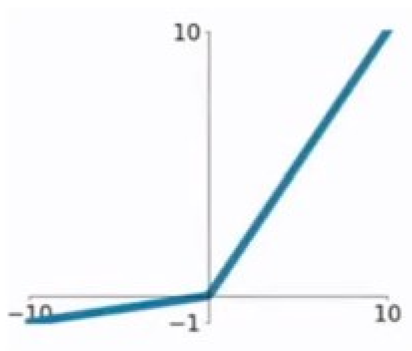
ReLU



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

是一個非飽和激勵函式且 zero-centere, 解決所謂的“梯度消失”問題, 且它能加快收斂速度, 對正數成線性輸出、負數直接為零, 所以在正數不飽和，在負數硬飽和, 也就會導致 ReLU 是完全不被啟用的，這就表明一旦輸入到了負數，ReLU 就沒有任何反饋。

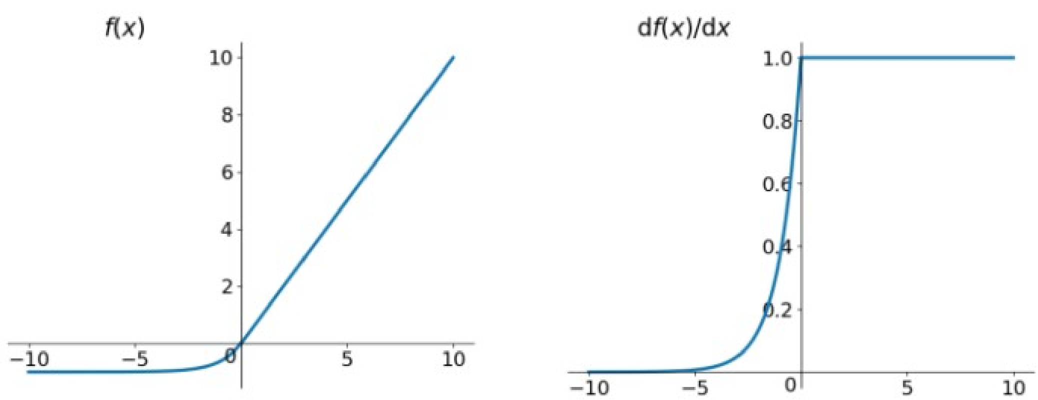
LeakyReLU



$$f(x) = max(ax, x)$$

為了解決 ReLU 在負數的不激勵問題，便在負數做一個可以自定義的斜度，但也謹此解決這問題，與其餘 ReLU 一樣。

ELU



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

同樣是要解決負數問題, 但同時也算是解決 zero-centered 的問題, 可是代價就是在負數時的要計算自然指數, 導致計算量龐大的問題。

2. Experiment setups

EEG Net 程式碼

```
# Layer 1: Conv2D
self.conv1 = nn.Conv2d(1, 16, kernel_size = (1, 51), stride = (1, 1), padding = (0, 25), bias = False)
self.batchNorm1 = nn.BatchNorm2d(16, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True)
# self.conv1.apply(self.weights_init)

# Layer 2: DepthwiseConv2D
self.conv2 = nn.Conv2d(16, 32, kernel_size = (2, 1), stride = (1, 1), groups = 16, bias = False)
self.batchNorm2 = nn.BatchNorm2d(32, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True)
self.activation2 = self.getActivation(eActivation)
self.avgPool2 = nn.AvgPool2d(kernel_size = (1, 4), stride = (1, 4), padding = 0)
self.dropout2 = nn.Dropout2d(p = 0.25)
# self.conv2.apply(self.weights_init)

# Layer 3: SeparableConv2D
self.conv3 = nn.Conv2d(32, 32, kernel_size = (1, 15), stride = (1, 1), padding = (0, 7), bias = False)
self.batchNorm3 = nn.BatchNorm2d(32, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True)
self.activation3 = self.getActivation(eActivation)
self.avgPool3 = nn.AvgPool2d(kernel_size = (1, 8), stride = (1, 8), padding = 0)
self.dropout3 = nn.Dropout2d(p = 0.25)
# self.conv3.apply(self.weights_init)

# Output: Classification
self.linear = nn.Linear(in_features = 736, out_features = 2, bias = True)
# self.weights_init_normal(self.linear)

# Loss:
self.loss = self.getLossFunc(eLossFunc)
self.optimizer = self.getOptimizer(eOptimizer)
```

DeepConvNet 程式碼

```
# Layer 1
self.conv1_1 = nn.Conv2d(1, 25, kernel_size = (1, 5), stride = (1, 1), bias = False)
self.conv1_2 = nn.Conv2d(25, 25, kernel_size = (2, 1), stride = (1, 1), bias = False)
self.batchNorm1 = nn.BatchNorm2d(25)
self.activation1 = self.getActivation(eActivation)
self.maxPool1 = nn.MaxPool2d(kernel_size = (1, 2))
self.dropout1 = nn.Dropout2d(p = 0.5)
# self.conv1_1.apply(self.weights_init)
# self.conv1_2.apply(self.weights_init)

# Layer 2:
self.conv2 = nn.Conv2d(25, 50, kernel_size = (1, 5), stride = (1, 1), bias = False)
self.batchNorm2 = nn.BatchNorm2d(50)
self.activation2 = self.getActivation(eActivation)
self.maxPool2 = nn.MaxPool2d(kernel_size = (1, 2))
self.dropout2 = nn.Dropout2d(p = 0.5)
# self.conv2.apply(self.weights_init)

# Layer 3:
self.conv3 = nn.Conv2d(50, 100, kernel_size = (1, 5), stride = (1, 1), bias = False)
self.batchNorm3 = nn.BatchNorm2d(100)
self.activation3 = self.getActivation(eActivation)
self.maxPool3 = nn.MaxPool2d(kernel_size = (1, 2))
self.dropout3 = nn.Dropout2d(p = 0.5)
# self.conv3.apply(self.weights_init)

# Layer 4:
self.conv4 = nn.Conv2d(100, 200, kernel_size = (1, 5), stride = (1, 1), bias = False)
self.batchNorm4 = nn.BatchNorm2d(200)
self.activation4 = self.getActivation(eActivation)
self.maxPool4 = nn.MaxPool2d(kernel_size = (1, 2))
self.dropout4 = nn.Dropout2d(p = 0.5)
# self.conv4.apply(self.weights_init)

# Output:
self.Softmax = nn.Softmax(dim = 0)
self.linear = nn.Linear(in_features = 8600, out_features = 2, bias = True)
# self.weights_init_normal(self.linear)

# Loss:
self.loss = nn.CrossEntropyLoss()
self.optimizer = optim.Adam(self.parameters())
```

動態調整 Activation, Optimizer, Loss (Only EEG)

```
def getActivation(self, eActivation):
    if eActivation == EActivation.ELU:
        return nn.ELU()

    elif eActivation == EActivation.ReLU:
        return nn.ReLU()

    else:
        return nn.LeakyReLU()

def getOptimizer(self, eOptimizer):
    if eOptimizer == EOptimizer.SGD:
        return optim.SGD(self.parameters(), lr = 1)

    elif eOptimizer == EOptimizer.Momentum:
        return optim.SGD(self.parameters(), lr = 1,  momentum = 0.8)

    elif eOptimizer == EOptimizer.RMSprop:
        return optim.RMSprop(self.parameters(), lr = 1)

    else:
        return optim.Adam(self.parameters(), lr = 1, weight_decay = 0.01)

def getLossFunc(self, eLossFunc):
    if eLossFunc == ELossFunc.NLLLoss:
        return nn.NLLLoss()

    elif eLossFunc == ELossFunc.CrossEntropyLoss:
        return nn.CrossEntropyLoss()

    else:
        return nn.PoissonNLLLoss()
```

Adam 的 weight_decay

從預設的 0 改為 0.01, 將 weight 做些許的衰減有助於訓練結果

```
optim.Adam(self.parameters(), lr = 1, weight_decay = 0.01)
```

DataLoader 的 shuffle

shuffle 將每次資料取出實做隨機處理有助於訓練結果

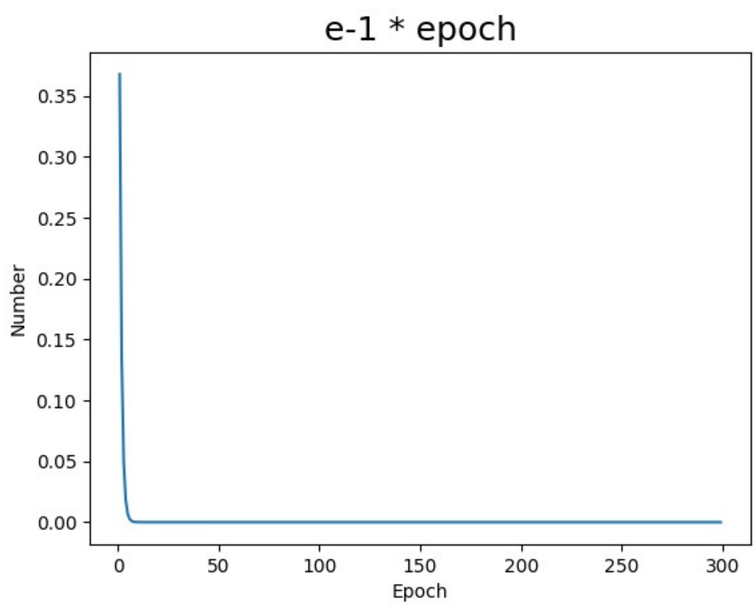
```
DataLoader(dataset, batch_size = 64, shuffle=True)
```

動態的調整 learning rate

用自然指數的曲線式下降, 但一旦小於最低的數值就鎖住在這

learning rate 以防過低的狀況

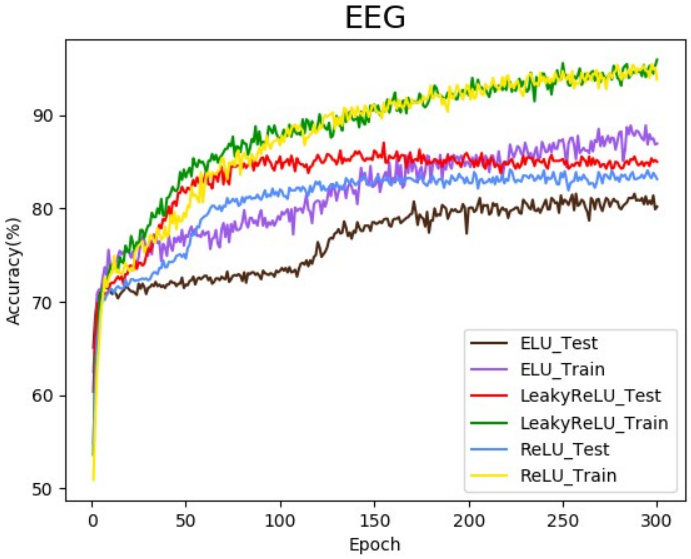
```
lr = math.exp(-1 * epoch)
if lr < learningRate:
    lr = learningRate
Net.setLearningRate(lr)
```

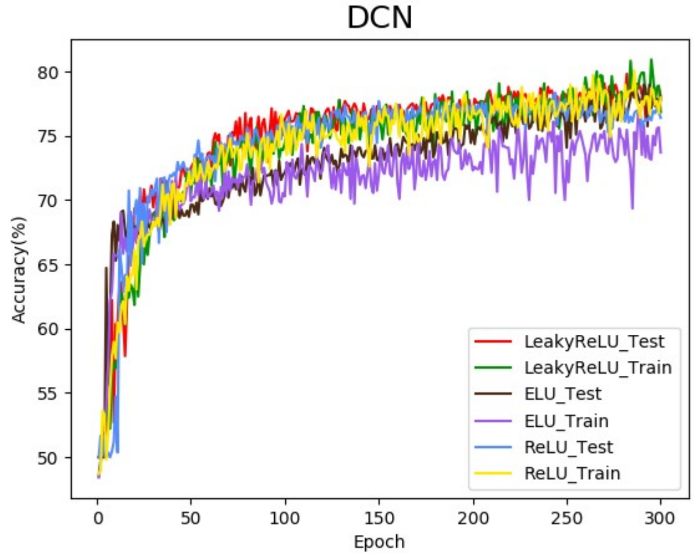


3. Experiment results

Max Accuracy(%):

Learning Rate: 2.5e-4, Batch Size: 64, Epoch: 300 Optimizer: Adam, Loss: CrossEntropyLoss			
	ReLU	Leaky ReLU	ELU
EEG	84.17	87.04	81.57
DeepConvNet	78.33	79.81	78.98





4. Discussion

這次嘗試了很多種調整 learning rate 和 batch size 的方式, 首先針對不同大小的 learning rate 對不同大小的 batch size 實測如下:

Leanring Rate	0.001									0.005								
Batch Size	2	4	8	18	36	72	120	270	540	2	4	8	18	36	72	120	270	540
100 epoch	72.31	75.09	78.43	77.5	78.33	77.59	78.98	77.87	73.89	68.33	70.65	75.28	74.07	73.43	74.44	80.09	74.91	79.81
200 epoch	74.07	74.72	75.93	77.5	78.7	78.89	79.44	77.59	76.76	72.87	69.63	73.89	75.65	73.15	74.44	78.24	75.56	77.13
300 epoch	73.98	74.54	76.2	76.57	76.85	79.35	77.87	78.06	77.59	71.48	66.39	73.61	75.09	73.43	74.17	78.06	75.37	77.13
Leanring Rate	0.01									0.05								
Batch Size	2	4	8	18	36	72	120	270	540	2	4	8	18	36	72	120	270	540
100 epoch	67.69	67.5	75.19	74.91	69.17	72.69	75.56	76.76	78.7	62.13	63.8	65.56	75.19	71.2	70	73.15	70.74	73.06
200 epoch	65.28	66.57	73.89	73.61	71.48	73.7	73.98	75	75.74	63.24	69.81	62.96	71.67	72.13	71.3	69.81	71.76	74.44
300 epoch	66.76	69.07	72.31	71.2	70.09	73.24	74.63	75.65	76.3	68.15	68.89	67.69	72.69	69.81	67.59	70	72.22	74.63
Leanring Rate	0.1									0.5								
Batch Size	2	4	8	18	36	72	120	270	540	2	4	8	18	36	72	120	270	540
100 epoch	61.85	64.81	57.69	64.44	69.35	70	70.46	63.43	72.69	51.02	49.35	50.65	51.02	63.24	63.33	66.3	67.04	70.56
200 epoch	67.59	60.56	63.8	76.85	73.8	68.24	66.48	71.85	71.48	53.33	49.26	48.7	47.96	70.93	64.17	59.07	71.02	70.65
300 epoch	62.96	68.52	64.91	77.04	72.59	68.06	68.98	71.39	72.78	48.89	48.89	50.28	50.28	67.69	73.8	69.91	71.11	68.8
Leanring Rate	0.9																	
Batch Size	2	4	8	18	36	72	120	270	540									
100 epoch	50.19	49.63	50.56	50.37	50.46	68.52	66.94	60.65	69.72									
200 epoch	50.09	49.07	52.96	50.65	49.35	65.65	67.78	65.83	72.69									
300 epoch	49.63	51.57	49.63	50.09	50	62.59	66.02	71.02	71.02									
	49.97	50.09	51.05	50.37	49.94	65.59	66.91	65.83	71.14									

發生現象有

1. learning rate 越大, batch size 越小せ,準確度下降

2. batch size 越大準確度因 learning rate 的影響不是很明顯

但 batch size 越大, 記憶體吃的量就越重, 且 epoch 次數要越多才會有比較好的結果, batch size 也不像 learning rate 影響準確度的幅度還大, 所以後面採取較小的 learning rate 適當的 batch size, 分別是 2.5e-4 和 64 。

嘗試使用初始化 weight 如下

```python
def weights_init_normal(self, m):
    m.weight.data.normal_(0.0, 1 / np.sqrt(m.in_features))
    m.bias.data.fill_(0.001)

def weights_init(self, m):
    nn.init.xavier_uniform_(m.weight)
    if m.bias:
        torch.nn.init.xavier_uniform_(m.bias)
```

使訓練的準確度比較穩定, 但並沒有增加準確度。

也有嘗試幾種 optimizer 與 loss function 也沒有在這實驗上有

更好的訓練準確度, 甚至有些 loss function 要特別調整輸出的維度。

```python
class EOptimizer(IntEnum):
    SGD = 0
    Momentum = 1
    RMSprop = 2
    Adam = 3
```

```python
class ELossFunc(IntEnum):
    NLLLoss = 0
    CrossEntropyLoss =1
    PoissonNLLLoss = 2
```