Lab3: EEG Classification

Wei-Yun Hsu
Institute of Multimedia Engineering
National Yang Ming Chiao Tung University

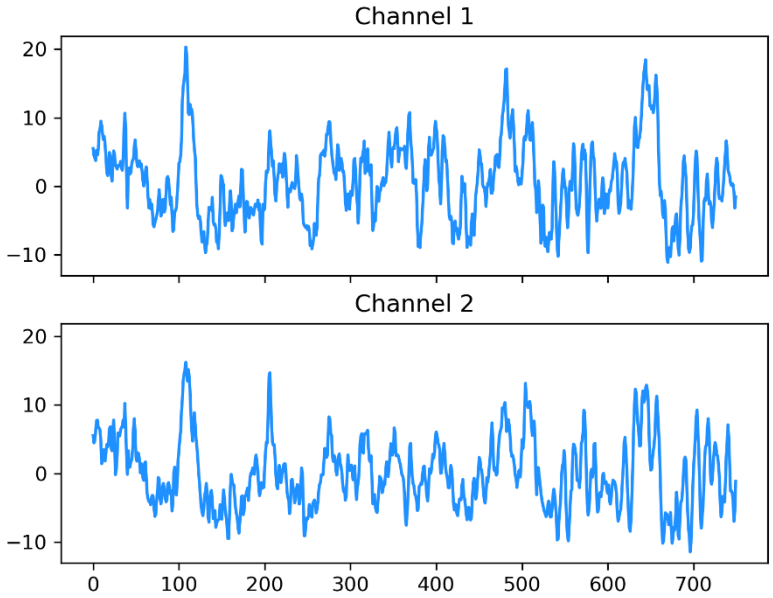
1. Introduction

In this lab, we will

- implement two neutral network structures, EEGNet and DeepConvNet
- compare the accuracy with three kinds of activation function including ReLU, LeakyReLU and ELU
- visualize the accuracy trend

A. Datasets

- EEG signal can refer to BCI Competition III – IIIb, each dataset has 2 channels and 750 sample points
- Training data: S4b_train.npz, X11b_train.npz
- Testing data: S4b_test.npz, X11b_test.npz

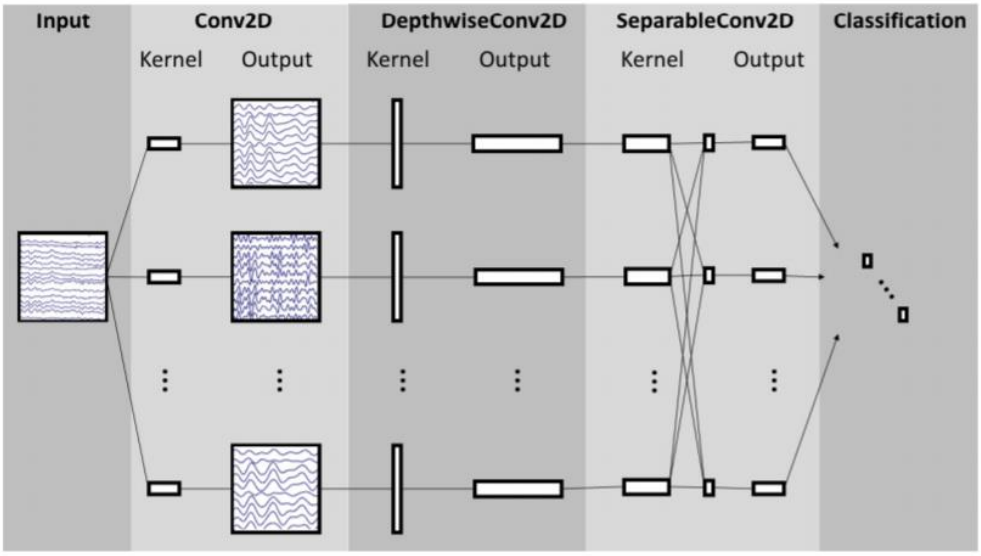


2. Experiment setup

A. The detail of your model

A-1. EEGNet

The following figure is the overview of EEGNet and one of the core design is depthwise convolution. it connect to each feature map individually, to learn frequency-specific spatial filters, and to reduce the number of trainable parameters. This mechanism is inspired by FBCSP algorithm.



My implementation of EEGNet is as follows.

```
class EEGNet(nn.Module):
    def __init__(self, activation='relu'):
        super(EEGNet, self).__init__()

        if activation == 'relu':
            self.activation = nn.ReLU()
        elif activation == 'lrelu':
            self.activation = nn.LeakyReLU()
        elif activation == 'elu':
            self.activation = nn.ELU(alpha=1.0)
        else:
            print('Unknown activation function')
            raise NotImplementedError

        self.firstconv = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25), bias=False),
            nn.BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True))

        self.depthwiseconv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias=False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            self.activation,
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout(p=0.5))

        self.separableconv = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            self.activation,
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout(p=0.5))

        self.classify = nn.Sequential(nn.Linear(in_features=736, out_features=2, bias=True))

    def forward(self, x):
        out = self.firstconv(x)
        out = self.depthwiseconv(out)
        out = self.separableconv(out)
        out = self.classify(out.flatten(start_dim=1))
        return out
```

A-2. DeepConvNet

DeepConvNet is designed for decoding imagined or executed movements from raw EEG. It provides a computationally efficient training strategy to train convolutional network on a larger number of input crops per EEG trial.

My implementation of DeepConvNet is as follows.

```
class DeepConvNet(nn.Module):
    def __init__(self, activation='relu'):
        super(DeepConvNet, self).__init__()

        if activation == 'relu':
            self.activation = nn.ReLU()
        elif activation == 'lrelu':
            self.activation = nn.LeakyReLU()
        elif activation == 'elu':
            self.activation = nn.ELU(alpha=1.0)
        else:
            print('Unknown activation function')
            raise NotImplementedError

        self.conv0 = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=True),
            nn.Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), padding=(0, 0), bias=True),
            nn.BatchNorm2d(25),
            self.activation,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )

        self.conv1 = nn.Sequential(
            nn.Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=True),
            nn.BatchNorm2d(50),
            self.activation,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5),
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=True),
            nn.BatchNorm2d(100),
            self.activation,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5),
        )
```

```
        self.conv3 = nn.Sequential(
            nn.Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=True),
            nn.BatchNorm2d(200),
            self.activation,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5),
        )

        self.classify = nn.Sequential(nn.Linear(in_features=8600, out_features=2, bias=True))

    def forward(self, x):
        out = self.conv0(x)
        out = self.conv1(out)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.classify(out.flatten(start_dim=1))
        return out
```

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

Try to use three different activation and compare their results.

- ReLU

$$ReLU(x) = max(0, x)$$

One of its limitations is when $x < 0$, gradient will be 0 because of the weights will not get adjusted during training. That means, ReLU can result in dead

neurons.

- Leaky ReLU

$$LeakyReLU(x) = \begin{cases} x, & if\ x > 0 \\ negative\_slope \times x, & otherwise \end{cases}$$

Leaky ReLU is one alternative to fix the "dying ReLU" problem by having a small negative slope. But as it possess linearity, it can't be used for the complex classification task.

- ELU

$$ELU(x) = max(0, x) + min(0, \alpha * (exp(x) - 1))$$

Similar to Leaky ReLU, ELU is a strong alternative to ReLU, it becomes smooth slowly until its output equal to $-\alpha$ whereas RELU sharply smoothes.

C. Experiment/parameter setup

The hyper-parameters for experiments as shown below.

- batch size: 32, 64
- learning rate: 0.01, 0.001
  - ✓ StepLR(optimizer, step_size=30, gamma=0.9)
- epochs: 300, 400, 500
- optimizer: Adam
- loss function: cross entropy
- activation function: ReLU, Leaky ReLU, ELU

3. Experimental results

A. The highest testing accuracy

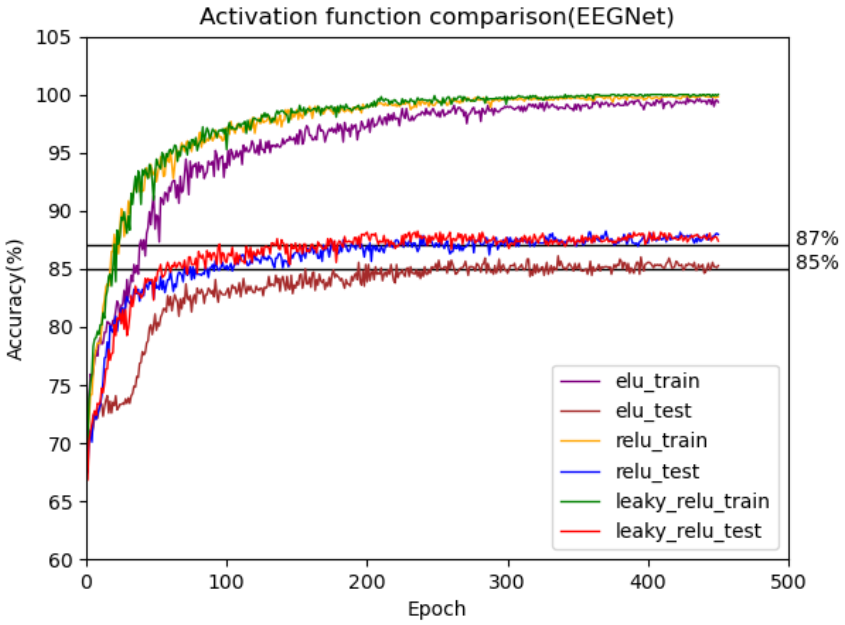
	ReLU	Leaky ReLU	ELU
EEGNet	**0.8796**	**0.8741**	**0.8528**
DeepConvNet	0.8241	0.8250	0.8185

The hyper-parameters of the above results are as follows.

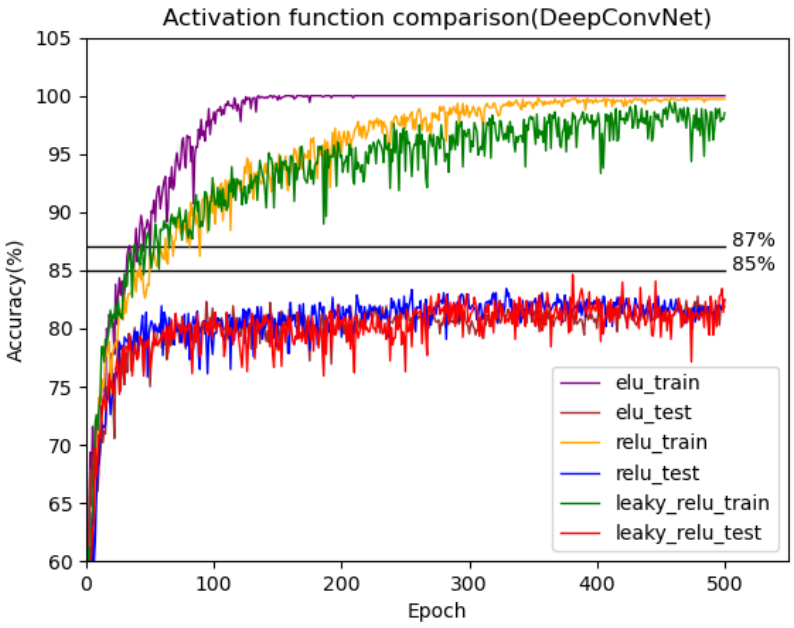
	ReLU	Leaky ReLU	ELU	
EEGNet	lr=0.001 bs=64	lr=0.001 bs=64	lr=0.001 bs=64	(epoch=450)
DeepConvNet	lr=0.01 bs=32	lr=0.01 bs=32	lr=0.01 bs=64	(epoch=500)

B. Comparison figures

B-1. EEGNet



B-2. DeepConvNet

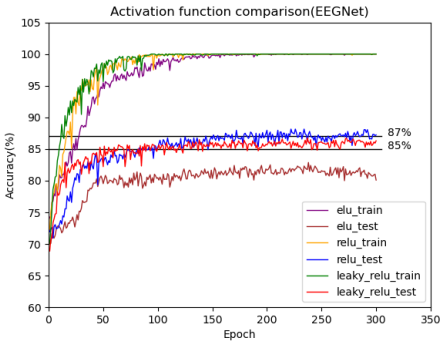
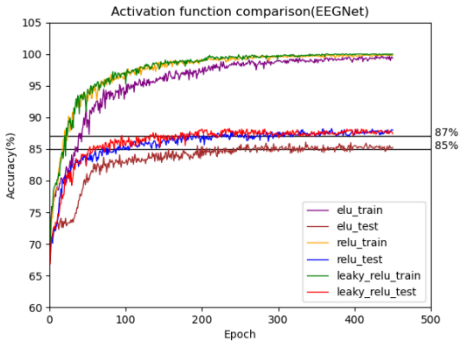


4. Discussion

A. The choice of dropout rate

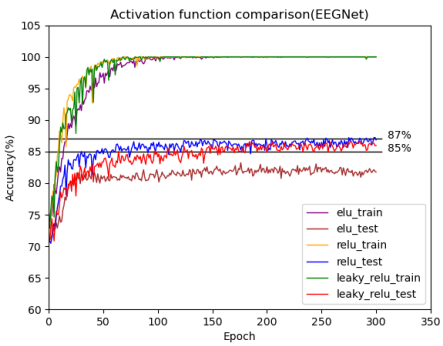
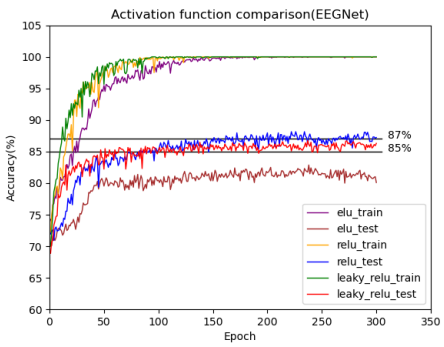
I observed that it would cause overfitting when the dropout rate of EEGNet is set to 0.25, so I tried to modify the value to 0.5, the result seemed to be

better, and no overfitting occurred. Because the mechanism of dropout layer is randomly sets input units to 0 with a frequency of rate at each step during training phase, which helps prevent overfitting.

EEGNet with dropout rate 0.25	EEGNet with dropout rate 0.5
	
Accuracy of EEGNet using relu: 87.22% Accuracy of EEGNet using leaky relu: 86.30% Accuracy of EEGNet using elu: 83.06%	Accuracy of EEGNet using relu: 87.96% Accuracy of EEGNet using leaky relu: 87.41% Accuracy of EEGNet using elu: 85.28%

B. The choice of batch size

When the same setting of learning rate, epoch and dropout rate on EEGNet, I found that the performance of the larger batch size usually outperform than that of the smaller batch size.

EEGNet with batch size 32	EEGNet with batch size 64
	
Accuracy of EEGNet using relu: 86.85% Accuracy of EEGNet using leaky relu: 85.93% Accuracy of EEGNet using elu: 81.76%	Accuracy of EEGNet using relu: 87.22% Accuracy of EEGNet using leaky relu: 86.30% Accuracy of EEGNet using elu: 80.09%

Therefore, I mostly try a larger batch size first to train the model, the training is more efficient, on the other hand, a higher batch size also helps the batch normalization layer to learn.

C. EEGNet v.s. DeepConvNet

As we can know that the number of parameters of DeepConvNet is larger

than EEGNet, but from the results of previous sections, EEGNet always outperform than DeepConvNet. It may mean that EEGNet has the better feature extraction ability based on its depthwise separable convolution structure. That means, EEGNet is more computationally efficient than DeepConvNet.

5. Extra

A. Implement any other classification model

A-1. ShallowConvNet

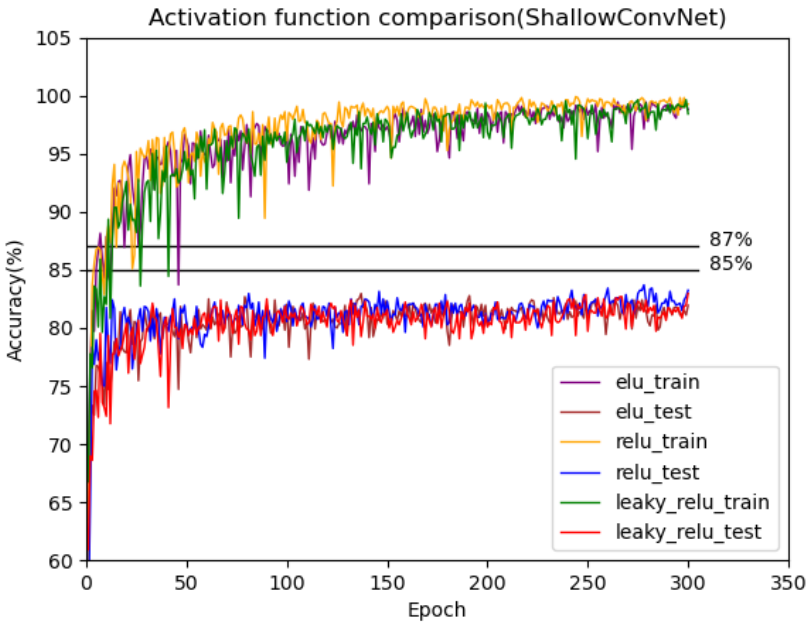
ShallowConvNet was proposed simultaneously with DeepConvNet for decoding motor-imagery EEG data, both of them are inspired by FBCSP. However, ShallowConvNet has been more popular than DeepConvNet because of its smaller model size.

My implementation of ShallowConvNet is shown below.

```
class ShallowConvNet(nn.Module):
    # Reference: https://github.com/CECNL/ExBrainable/blob/main/ExBrainable/models.py
    def __init__(self, activation='relu'):
        super(ShallowConvNet, self).__init__()

        if activation == 'relu':
            self.activation = nn.ReLU()
        elif activation == 'lrelu':
            self.activation = nn.LeakyReLU()
        elif activation == 'elu':
            self.activation = nn.ELU(alpha=1.0)
        else:
            print('Unknown activation function')
            raise NotImplementedError

        self.tp= 750
        self.ch= 2
        self.sf= 125
        self.n_class=2
        self.octsf= int(math.ceil(self.sf*0.1))
        self.tpconv1= int(self.tp- self.octsf+1)
        self.apstride= int(math.ceil(self.octsf/2))

        # kernel size=(ceil(sf*0.1)
        self.conv1 = nn.Conv2d(1, 40, (1, self.octsf), bias=False)
        #(n_ch,1)
        self.conv2 = nn.Conv2d(40, 40, (self.ch, 1), bias=False)
        self.Bn1   = nn.BatchNorm2d(40)
        self.acti = self.activation
        self.AvgPool1 = nn.AvgPool2d((1, int(self.apstride*5)), stride=(1,self.apstride ))
```

```
        self.Drop1 = nn.Dropout(0.25)
        self.classifier = nn.Linear(int(40*math.ceil((self.tpconv1-self.apstride*5) / self.apstride)),
                                    self.n_class, bias=True)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.Bn1(x)
        x = x**2
        x = self.acti(x)
        x = self.AvgPool1(x)
        x = torch.log(x)
        x = self.Drop1(x)
        x = x.view(-1, int(40*math.ceil((self.tpconv1-self.apstride*5 )/ self.apstride))) #40*74
        x = self.classifier(x)
        return x
```

The following figures are the comparison of ShallowConvNet using three kind of activation functions and the setting of hyper-parameters..




```
Accuracy of ShallowConvNet using relu: 83.24%
Accuracy of ShallowConvNet using leaky relu: 82.96%
Accuracy of ShallowConvNet using elu: 81.94%
```

	ReLU	Leaky ReLU	ELU	
ShallowConvNet	lr=0.01 bs=64	lr=0.01 bs=32	lr=0.01 bs=32	(epoch=300)

According to the section 3, as we can see that the performance of ShallowConvNet is better than DeepConvNet, but worse than EEGNet. Although the main difference between ShallowConvNet and DeepConvNet is the number of parameters or model size, I think this assignment is an easy classification task, the model complexity of DeepConvNet is large for it, this may be one of the reasons for this situation.