DLP Lab4 report

310553009 張力仁

1. Introduction

In this lab, we want to conduct a EEG signal classification, and implement 2 kinds of model architectures (EEGNet and DeepConvNet) for EEG classification with 3 different activation functions (ELU, LeakuReLU, and ReLU).

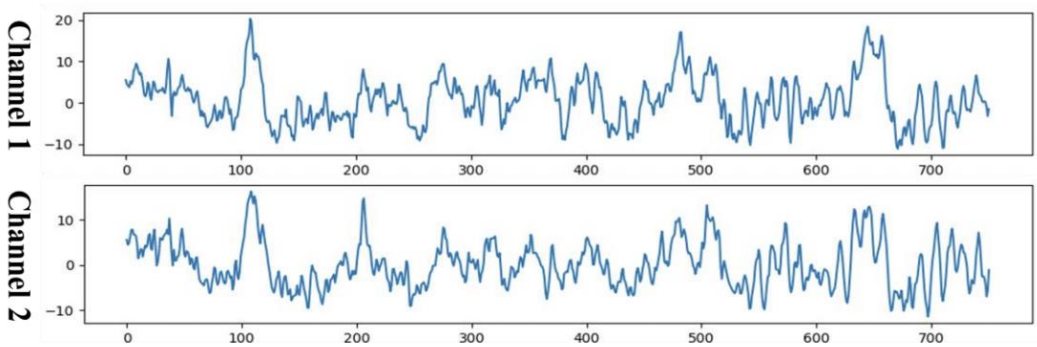
The objective of this Lab is to

- obtain highest accuracy of two models with three different activation function
- show the learning curve
- compare the result between the combination of 2 models and 3 activation function

2. Experiment set up

2.1. Dataset: BCI competition IIIB

This is a motor imagary dataset including 3 subjects and recoding by bipolar EEG channel in 125Hz. All trials are divided into 2 classes and each trial contain 750 data points (6 seconds). The signal shows below



Because I use pytorch framework in this Lab, we need to convert the original data type from numpy array to tensor. And to let pytorch handle the data more efficiently, we will use the dataloader provided by pytorch to wrap the dataset.

```
def get_dataloader(train_data, train_label, test_data, test_label):

    x_train = torch.Tensor(train_data)
    x_test = torch.Tensor(test_data)
    y_train = torch.Tensor(train_label).view(-1,1)
    y_test = torch.Tensor(test_label).view(-1,1)

    # 存成tensordataset
    train_dataset = TensorDataset(x_train, y_train)
    test_dataset = TensorDataset(x_test, y_test)
```

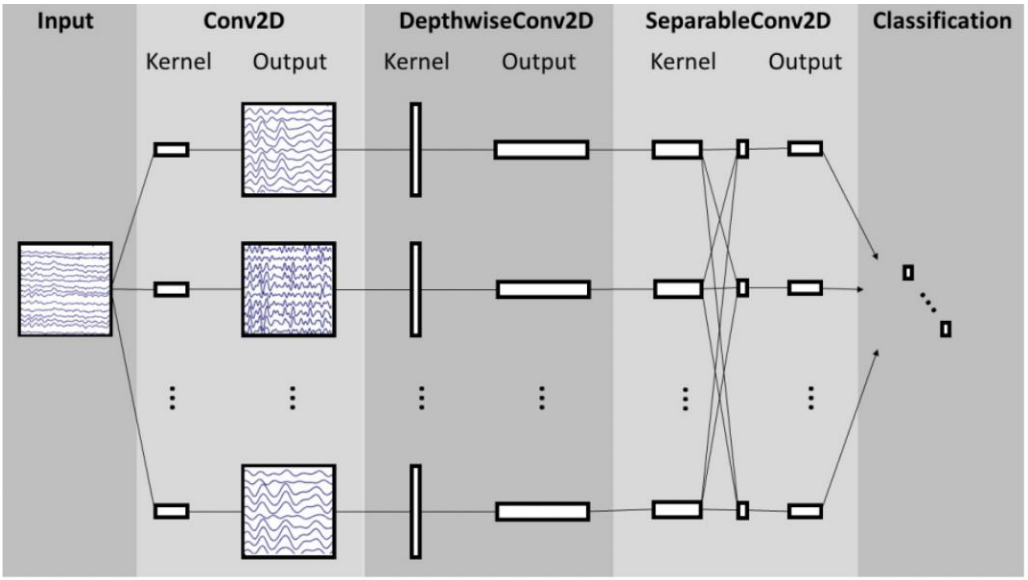
```
# 包成dataloader
train_dl = DataLoader(
    dataset = train_dataset,
    batch_size = BATCH_SIZE,
    shuffle = True,
    num_workers = 0
)

test_dl = DataLoader(
    dataset = test_dataset,
    batch_size = len(test_dataset),
    shuffle = False,
    num_workers = 0
)
return train_dl, test_dl
```

2.2. Model architecture

2.2.1. EEGNet

EEGNet is a CNN model designed for the BCI task (mainly for EEG data). It adapts the concept from depthwise separable convolution. In their model, they want to extract the the 2 different kind of features from EEG: temporal and spatial feature by 2 different designed convolution layers.

The first convolution block aims to extract different temporal pattern from the input signal. The following Depthwise convolution is used to extract the the spatial information from different EEG channels. and last separable convolution learn all spatial and temporal features and then go through the classifier to produce output



```
self.conv_block1 = nn.Sequential(
    nn.Conv2d(in_channels=1, out_channels=8, kernel_size=(1, 51), stride=(1,1), padding=(0,25), bias=False),
    nn.BatchNorm2d(8)
)
self. depthwiseConv = nn.Sequential(
    nn.Conv2d(in_channels=8, out_channels=16, kernel_size=(2, 1), stride=(1,1), groups=8, bias=False),
    nn.BatchNorm2d(16),
    self.activation,
    nn.AvgPool2d((1,4), stride=(1,4), padding=0),
    nn.Dropout(0.15)
)
self.separableConv = nn.Sequential(
    nn.Conv2d(in_channels=16, out_channels=16, kernel_size=(1, 15), stride=(1,1), padding = (0,7), bias=False),
    nn.BatchNorm2d(16),
    self.activation,
    nn.AvgPool2d((1,8), stride=(1,8), padding=0),
    nn.Dropout(0.15)
)
self.classifier = nn.Sequential(
    nn.Linear(368, 1, bias=True),
    nn.Sigmoid()
)
```

```
def forward(self, x):
    x = self.conv_block1(x)
    x = self.depthwiseConv(x)
    x = self.separableConv(x)
    # print(f'shape before flatten: {x.size()}')
    x = x.view(-1, self.classifier[0].in_features)
    #print(f'reshape: {x.size()}')
    out = self.classifier(x)

    return out
```

In the EEGNet, mainly, we can adjust the number of the output kernel of the first convolution block and the multiple of the output of depth-wise convolution. In my implementation, I choose 8 and 2 respectively.

2.2.2. DeepConvNet

The DeepConvNet consists of multiple convolution block, just like a normal CNN. This model is used to be a comparison in the research of EEGNet.

The main architecture of DeepConvNet is **4 convolution blocks** and a output layer. The convolution block consists of 5 layers: **2D convolution, Batchnormalize, Activation function, Maxpooling 2D, and Dropout**. And, there is a extra convolutional layer in the first convolution block. So, in this architecture, there are [25, 50, 100, 200] convolutional kernel on each block.

```
self.kernel_num = kernel_num #[25, 50, 100, 200]

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

for idx in range(1, len(kernel_num)):
    setattr(self,'conv'+str(idx), nn.Sequential(
        nn.Conv2d(kernel_num[idx-1], kernel_num[idx], kernel_size=(1,5), stride=(1,1), padding=(0,0), bias=True),
        nn.BatchNorm2d(kernel_num[idx]),
        self.activation,
        nn.MaxPool2d(kernel_size=(1,2)),
        nn.Dropout(0.5)
    ))

flatten_size = kernel_num[-1]*reduce(lambda x,_: round((x-4)/2), kernel_num, 750)
self.classifier = nn.Sequential(
        nn.Linear(flatten_size, 2, bias=True)
    )
def forward(self, x):

    for i in range(len(self.kernel_num)):
        x = getattr(self, 'conv' + str(i))(x)

    x = x.view(-1, self.classifier[0].in_features)
    output = self.classifier(x)

    return output
```

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

2.3. Activation function

There are 3 different activation functions we will use in this experiment.

- **ReLU**

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

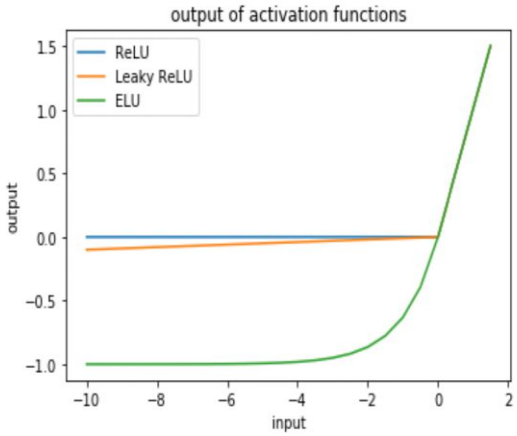
- **LeakyReLU**

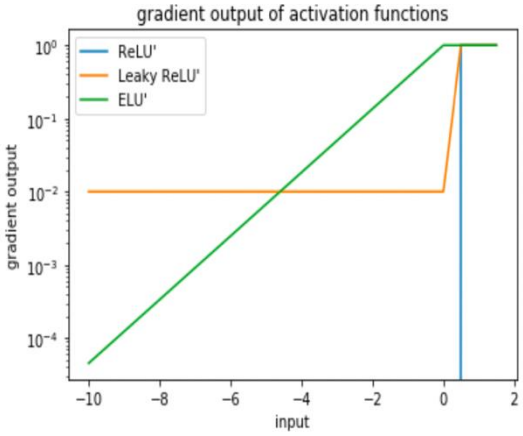
$$LeakyReUL(x) = \begin{cases} x, & x > 0 \\ negative\ slope * x, & otherwise \end{cases}$$

- **ELU**

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

ReLU is a well-known activation function in deep learning because it can prevent the BP process from gradient vanish. However, it also has some downsides. In ReLU, it always produces **zero-gradient on negative value**. So to speak, it discards the information from negative input. Therefore, the LeakyReLU and ELU, these kind of activation functions are developed. These two activation functions have the gradient from the negative input, and it may be useful for transmit some gradient backward.





2.4. Hyper-parameter setup

Optimizer: Adam
Loss function: Cross entropy loss
Epoch: 450
Batch size: 64
Learning rate: 1e-2(EEGNet) / 1e-3(DeepConvNet)
Weight decay: 0.0001

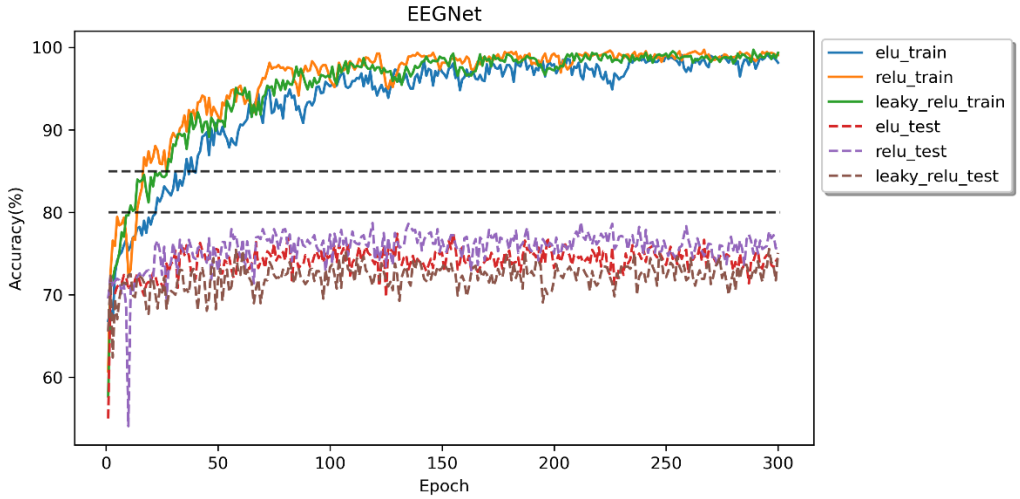
3. Experimental result

3.1. The highest testing accuracy

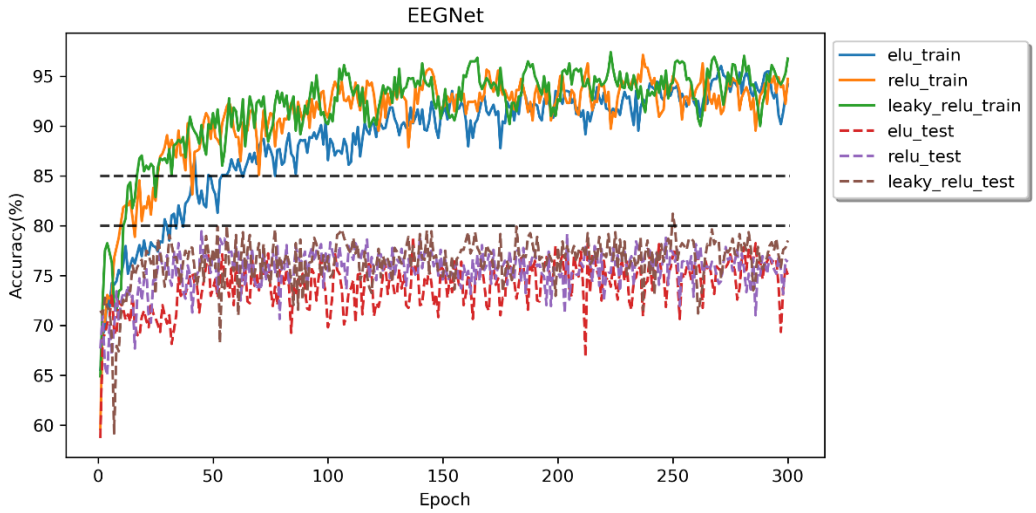
	ReLU	**LeakyReLU**	**ELU**
EEGNet	86.2	85.55	83.24
DeepConvNet	77.03	76.85	79.25

3.2. Figure comparison

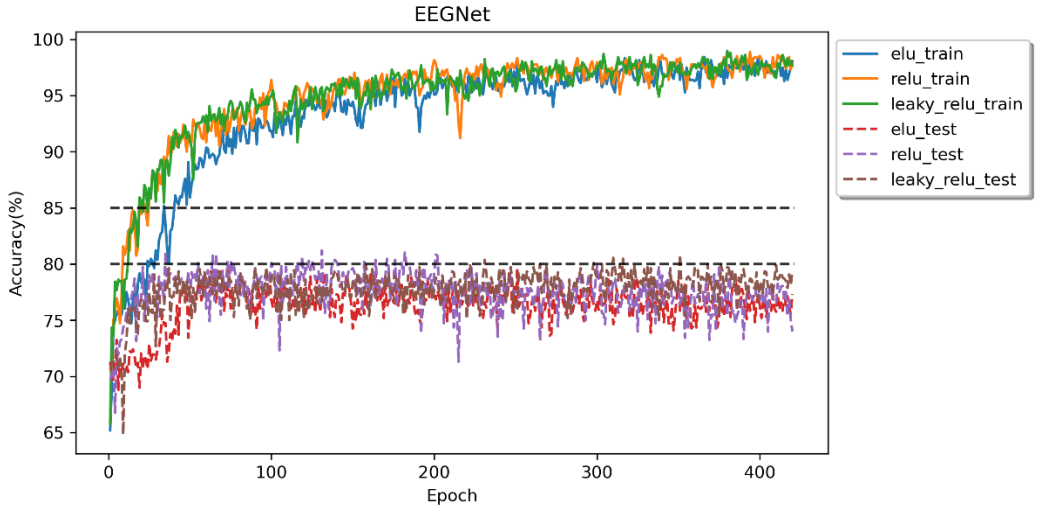
- **Original (EEGNet 16-2)**



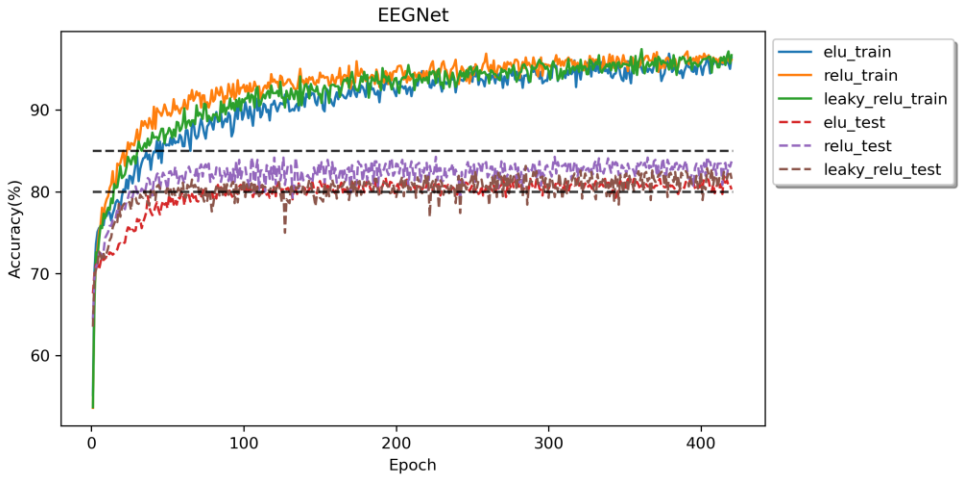
- **Weight decay 0.001**



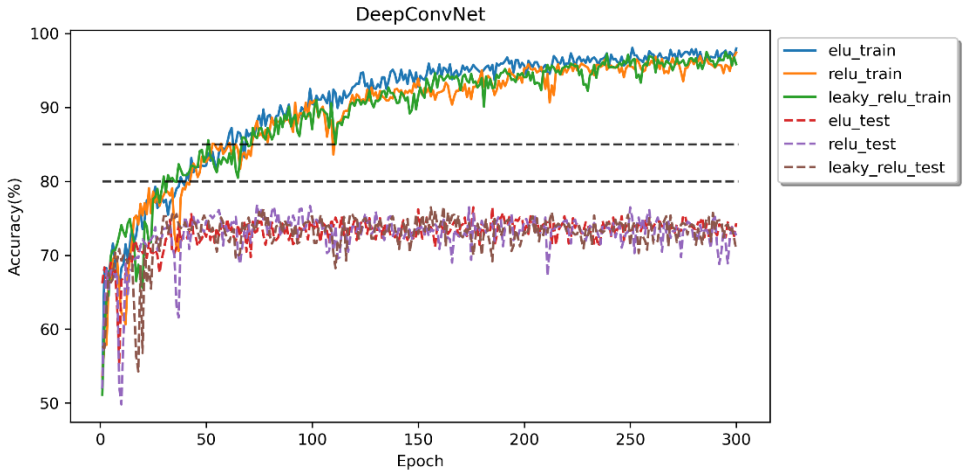
- **A compact model structure (EEGNet 8-2)**



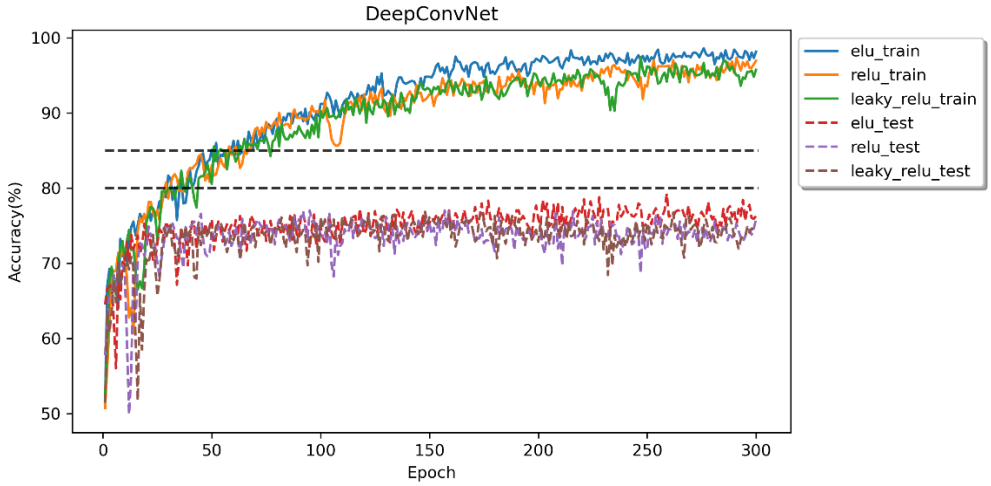
- **EEGNet8-2 (weight decay 0.0009)**



- **DeepConvNet (original)**



- **DeepConvNet (weight decay 0.001)**



4. Discussion

From the experimental result, we can find that EEGNet will sightly outperform the DeepConvNet. I think the reason maybe come from the structure of these 2 models. EEGNet design for the EEG classification task and it is compact. On the other hand, although

DeepConvNet has deeper structure, the data of EEG normally less than other fields (such as CV). Therefore, a compact model may perform well on this BCI Task.

As for the difference between the activation function, generally, LeakyReLU and ELU actually better than ReLU. And from the learning curve, we can observe that ELU converge slightly slower than other two activation function.

From the training result (loss curve), I notice that overfitting occurred on almost all situations. And there is a gap between training and testing result. It maybe come from the difference between the distribution of these two datasets. So, to increase the generalizability, I would like to **add L2-regularization (weight decay) to the training process and shrink the model structure (EEGNet 8-2)**. And from the result, it actually improves the accuracy.

Finally, in testing phase, we should activate model evaluation mode to disable dropout and batch normalization. In this way, model can fulfill all its ability to achieve the highest performance.