Lab2 : EEG classification

Name : 鄭謝廷揚 Student ID : A073501

Introduction :

In this lab, we will need to implement convolutional neural network which are **EEGNet** and **DeepConvNet** to classify simple EEG models with BCI competition dataset. Moreover, we need to try different kinds of activation function including ReLU, LeakyRelu, and ELU. Also, we will learn how to construct CNN model with PyTorch and CNN architecture graph.

Experiment set up :

A. The detail of my model

・EEGNet :

1. Architecture of EEGNet

```
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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25)
  )
  (separableConv): 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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

2. EEGNet model set up

```
class EEGNet_elu(nn.Module):
    def __init__(self):
        super(EEGNet_elu, self).__init__()
        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),
            nn.ELU(alpha=1.0),
            nn.AvgPool2d(kernel_size=(1,4), stride=(1, 4), padding=0),
            nn.Dropout(p=0.25)
        )
        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),
            nn.ELU(alpha=1.0),
            nn.AvgPool2d(kernel_size=(1,8), stride=(1, 8), padding=0),
            nn.Dropout(p=0.25)
        )
        self.classify = nn.Sequential(
            nn.Linear(in_features=736, out_features=2, bias=True)
            )

    def forward(self, x):
        x = self.firstconv(x)
        x = self.depthwiseconv(x)
        x = self.separableconv(x)
        x = x.view(x.size(0), -1)
        output =  self.classify(x)
        return output
```

3. EEGNet training state setup

```
torch.cuda.empty_cache()
eegnet = EEGNet_elu().cuda()
Optimizer = torch.optim.Adam(eegnet.parameters(), lr=Learning_rate)
Loss_func = nn.CrossEntropyLoss()

test_label_t = test_label_t.long().cuda()
#Optimizer
eegnet = eegnet.double()
test_result = []
train_result = []
for epoch in range(Epochs):
    for step, (b_x, b_y) in enumerate(train_dataloader):
        b_x = b_x.cuda()
        output = eegnet(b_x)
        b_y = b_y.long().cuda()
        loss = Loss_func(output, b_y)
        s1 =  torch.squeeze(b_y)
        s2 =  torch.squeeze(output)
        Optimizer.zero_grad()
        loss.backward()
        Optimizer.step()

    test_output = eegnet(test_data_t)
    pred_y = torch.max(test_output, 1)[1].cuda().data.squeeze()
    accuracy = float(torch.sum(pred_y == test_label_t)) /test_label_t.size(0)
    test result.append(accuracy)
```

▪ DeepConvNet：

1. Architecture of 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

2. DeepConvNet model set up

```
from functools import reduce
class DeepConvNet_elu(nn.Module):
    def __init__(self, deepconv=[25,50,100,200]):
        super(DeepConvNet_elu, self).__init__()
        self.conv1 = 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, eps=1e-05, momentum=0.1),
            nn.ELU(alpha=1.0),
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5)
            )
        self.conv2 = nn.Sequential(
            nn.Conv2d(25, 50, kernel_size=(1, 5), stride=(1,1), padding=(0,0), bias=True),
            nn.BatchNorm2d(50, eps=1e-05, momentum=0.1),
            nn.ELU(alpha=1.0),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size=(1, 5), stride=(1,1), padding=(0,0), bias=True),
            nn.BatchNorm2d(100, eps=1e-05, momentum=0.1),
            nn.ELU(alpha=1.0),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(100, 200, kernel_size=(1, 5), stride=(1,1), padding=(0,0), bias=True),
            nn.BatchNorm2d(200, eps=1e-05, momentum=0.1),
            nn.ELU(alpha=1.0),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
        self.classify = nn.Sequential(
            nn.Softmax()
            )
        flatten_size =  200 * reduce(
            lambda x,_: round((x-4)/2), deepconv, 750)
        self.classify2 = nn.Sequential(
            nn.Linear(flatten_size, 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(-1, self.classify2[0].in_features)
        #print(x.shape)
        output =  self.classify2(x)
        #print(output.shape)
        return output
```

3. DeepConvNet training step set up

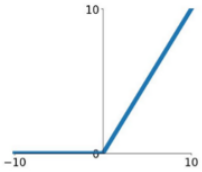
```
depconvnet = DeepConvNet_elu().cuda()
Optimizer = torch.optim.Adam(depconvnet.parameters(), lr=Learning_rate)
Loss_func = nn.CrossEntropyLoss()
test_label_t = test_label_t.long().cuda()
#Optimizer
Optimizer.zero_grad()
depconvnett = depconvnet.double()
test_result = [0]
train_result = [0]
for epoch in range(Epochs):
    for step, (b_x, b_y) in enumerate(train_dataloader):
        b_x = b_x.cuda()
        output = depconvnet(b_x)
        b_y = b_y.long().cuda()
        loss = Loss_func(output, b_y)
        s1 =  torch.squeeze(b_y)
        s2 =  torch.squeeze(output)
        Optimizer.zero_grad()
        loss.backward()
        Optimizer.step()
```

B. Explain the activation function

1. ReLU function and source code

ReLU

$$\max(0, x)$$

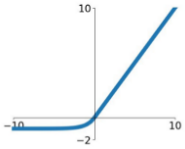



```
nn.ReLU(),
```

2. ELU function and source code

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

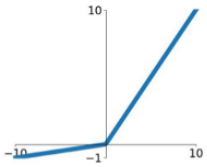



```
nn.ELU(alpha=1.0),
```

3. Leaky ReLU and source code

Leaky ReLU

$$\max(0.1x, x)$$




```
nn.LeakyReLU(0.1),
```

4. Disscusion

ReLU is the most commonly used activation function in neural networks, and have following advantages.

- It's easy to compute
- It doesn't have the vanishing gradient problem suffered by other activation functions like sigmoid or tanh.

・It converges faster.

Leaky ReLU has all the advantages of ReLU, besides there will be no Dead ReLU issues.

ELU has all the basic advantages of ReLU, and:

・There will be no Dead ReLU issues

・The mean of the output is close to 0, zero-centered

Experimental result :

A. The highest testing accuracy

- **Screenshot**

1. Screenshot for EEGNet

In [50]:
```
import numpy as np
import matplotlib.pyplot as plt

for i in Results:
    print(i,"       ",max(Results[i]))
# plot the data
fig = plt.figure()
x = np.arange(0, 351, 1)
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, Results['EEG_test_elu'], color='tab:blue',label='EEG_test_elu')
ax.plot(x, Results['EEG_train_elu'], color='tab:orange',label='EEG_train_elu')
ax.plot(x, Results['EEG_test_relu'], color='tab:green',label='EEG_test_relu')
ax.plot(x, Results['EEG_train_relu'], color='tab:red',label='EEG_train_relu')
ax.plot(x, Results['EEG_test_leakyrelu'], color='tab:brown',label='EEG_test_leakyrelu')
ax.plot(x, Results['EEG_train_leakyrelu'], color='tab:pink',label='EEG_train_leakyrelu')
# set the limits
ax.set_xlim([-50, 350])
ax.set_ylim([0.5, 1])

ax.set_title('Activation function(EEGNet)')
ax.set_xlabel('Epoch')
ax.set_ylabel('Accuracy')
ax.legend()
# display the plot
plt.show()
```

```
EEG_test_elu         0.8
EEG_train_elu         0.9953703703703703
EEG_test_relu         0.8379629629629629
EEG_train_relu         0.9981481481481481
EEG_test_leakyrelu         0.8212962962962963
EEG_train_leakyrelu         0.9953703703703703
```

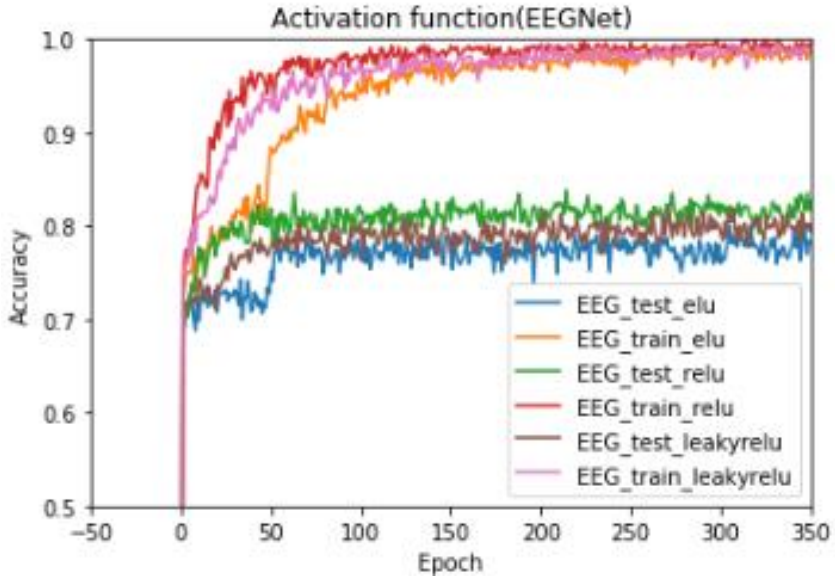
2. Screenshot for DeepConvNet

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt

for i in Results:
    print(i,"       ",max(Results[i]))
# plot the data
fig = plt.figure()
x = np.arange(0, 351, 1)
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, Results['DeepConvNet_test_elu'], color='tab:blue',label='DeepConvNet_test_elu')
ax.plot(x, Results['DeepConvNet_train_elu'], color='tab:orange',label='DeepConvNet_train_elu')
ax.plot(x, Results['DeepConvNet_test_relu'], color='tab:green',label='DeepConvNet_test_relu')
ax.plot(x, Results['DeepConvNet_train_relu'], color='tab:red',label='DeepConvNet_train_relu')
ax.plot(x, Results['DeepConvNet_test_leakyrelu'], color='tab:brown',label='DeepConvNet_test_leakyrelu')
ax.plot(x, Results['DeepConvNet_train_leakyrelu'], color='tab:pink',label='DeepConvNet_train_leakyrelu')
# set the limits
ax.set_xlim([-50, 400])
ax.set_ylim([0.4, 1])

ax.set_title('Activation function(DeepConvNet)')
ax.set_xlabel('Epoch')
ax.set_ylabel('Accuracy')
ax.legend()
# display the plot
plt.show()
```

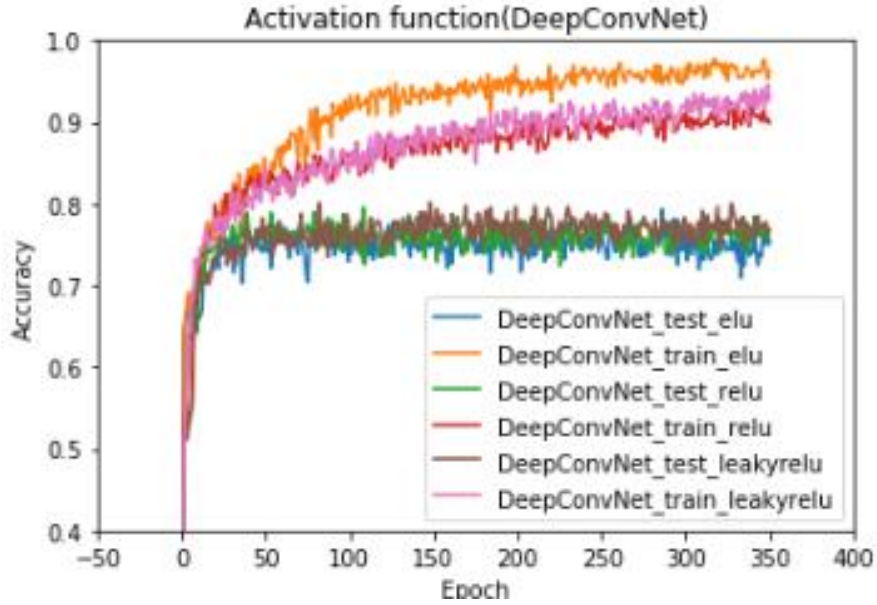
```
DeepConvNet_train_elu         0.9777777777777777
DeepConvNet_test_elu         0.7925925925925926
DeepConvNet_train_relu         0.924074074074074
DeepConvNet_test_relu         0.7953703703703704
DeepConvNet_train_leakyrelu         0.9462962962962963
DeepConvNet_test_leakyrelu         0.8018518518518518
```

B. Comparison figures

1. EEGNet



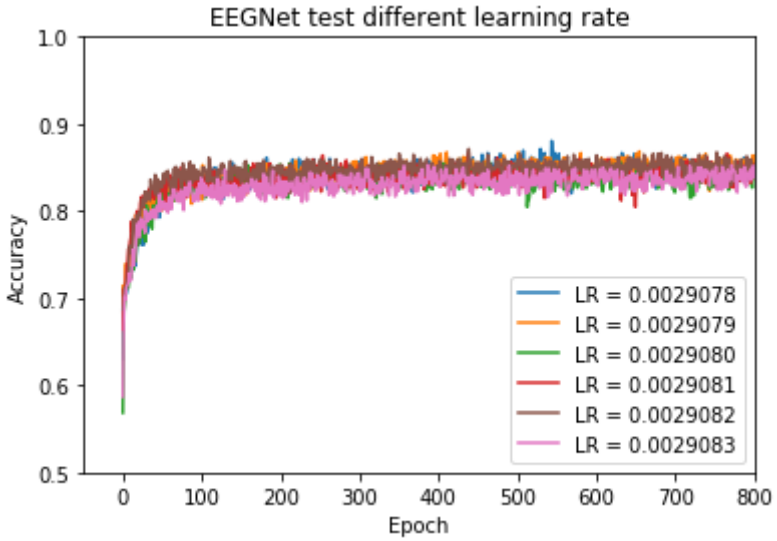
2. DeepConvNet



D. Testing different LR to get better result

```
Detail_result = {}

for i in range(6):
    Learning_rate = 0.002907800000001 + i*0.0000001
    torch.cuda.empty_cache()
    eegnet = EEGNet_relu().cuda()
    Optimizer = torch.optim.Adam(eegnet.parameters(), lr=Learning_rate, betas=(0.9, 0.999), eps=1e-08, weight_decay=0)
    Loss_func = nn.CrossEntropyLoss()

    test_label_t = test_label_t.long().cuda()
    #Optimizer
    eegnet = eegnet.double()
    test_result = []
    train_result = []
    for epoch in range(Epochs):
        for step, (b_x, b_y) in enumerate(train_dataloader):
            b_x = b_x.cuda()
            output = eegnet(b_x)
            b_y = b_y.long().cuda()
            loss = Loss_func(output, b_y).cuda()
            s1 =  torch.squeeze(b_y)
            s2 =  torch.squeeze(output)
            Optimizer.zero_grad()
            loss.backward()
            Optimizer.step()

        test_output = eegnet(test_data_t)
        pred_y = torch.max(test_output, 1)[1].cuda().data.squeeze()
        accuracy = float(torch.sum(pred_y == test_label_t)) /test_label_t.size(0)
        test_result.append(accuracy)

    print(Learning_rate,max(test_result))
    Detail_result[str(i)] =  test_result
```

```
0.002907800000001 0.8805555555555555
0.002907900000001 0.8685185185185185
0.002908000000001 0.8601851851851852
0.002908100000001 0.8666666666666667
0.002908200000009997 0.8712962962962963
0.002908300000001 0.8527777777777777
```



Disscusion :

In the lab, I have learned how to construct CNN model by PyTorch. To classify simple EEG we use EEGNet model and DeepConvNet model. The EEGNet model is better than DeepConvNet model in this lab. I think that deeper CNN network is not always better. Changing model from different situation is more important. Moreover, we need to try different initial weight and learning rate to get better result.