Lab2

0856108 謝宗祐

1. Introduction

   In this lab, I have to build up two models (EEGNet and DeepConveNet) to do simple EEG classification with BCI competition dataset. In addition, I also need to replace the original activation function (ELU) to other two activation functions (ReLU and Leaky ReLU) to compare the accuracy.

2. Experimental set up

   A. The detail of two models

      (1). EEGNet

      I follow the structure provided by TA to build up the EEGNet, and below is my implementation. By the way, before the final classify layer, the input need to be reshaped into (B, 736). I think it's the only one that the spec didn't say.

```python
class EEGNet(nn.Module):
    def __init__(self, activation):
        super(EEGNet, self).__init__()
        if activation == 'relu':
            self.activate = nn.ReLU()
        elif activation == 'leaky':
            self.activate = nn.LeakyReLU()
        else:
            self.activate = nn.ELU()

        # firstconv
        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)

        # depthwiseconv
        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.avgpool1 = nn.AvgPool2d(kernel_size=(1,4), stride=(1,4), padding=0)

        # separableconv
        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.avgpool2 = nn.AvgPool2d(kernel_size=(1,8), stride=(1,8), padding=0)

        # classify
        self.linear1 = nn.Linear(736, 2, bias=True)
```

```python
def forward(self, x):
    # firstconv
    out = self.conv1(x)
    out = self.batchnorm1(out)

    # depthwiseconv
    out = self.conv2(out)
    out = self.activate(self.batchnorm2(out))
    out = F.dropout(self.avgpool1(out), p=0.25)

    # separableconv
    out = self.conv3(out)
    out = self.activate(self.batchnorm3(out))
    out = F.dropout(self.avgpool2(out), p=0.25)

    # classify
    out = out.view(-1,736)
    out = self.linear1(out)

    return out
```

      (2). DeepConvNet

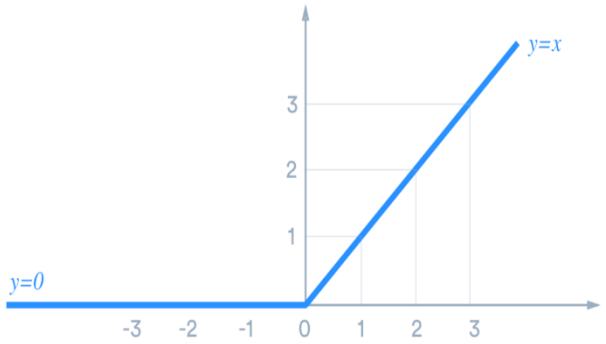
      I follow the structure provided by TA to build up DeepConvNet, and below is my implementation. Nothing different compare to the spec.

```python
class DeepConvNet(nn.Module):
    def __init__(self, activation):
        super(DeepConvNet, self).__init__()
        if activation == 'relu':
            self.activate = nn.ReLU()
        elif activation == 'leaky':
            self.activate = nn.LeakyReLU()
        else:
            self.activate = nn.ELU()

        self.C = 2
        self.T = 750
        self.N = 2

        self.conv1 = nn.Conv2d(1, 25, kernel_size=(1,5))
        self.conv2 = nn.Conv2d(25, 25, kernel_size=(self.C,1))
        self.batchnorm1 = nn.BatchNorm2d(25, eps=1e-05, momentum=0.1)
        self.maxpool = nn.MaxPool2d(kernel_size=(1,2))
        self.conv3 = nn.Conv2d(25, 50, kernel_size=(1,5))
        self.batchnorm2 = nn.BatchNorm2d(50, eps=1e-05, momentum=0.1)
        self.conv4 = nn.Conv2d(50, 100, kernel_size=(1,5))
        self.batchnorm3 = nn.BatchNorm2d(100, eps=1e-05, momentum=0.1)
        self.conv5 = nn.Conv2d(100, 200, kernel_size=(1,5))
        self.batchnorm4 = nn.BatchNorm2d(200, eps=1e-05, momentum=0.1)
        self.linear = nn.Linear(8600, self.N)
```

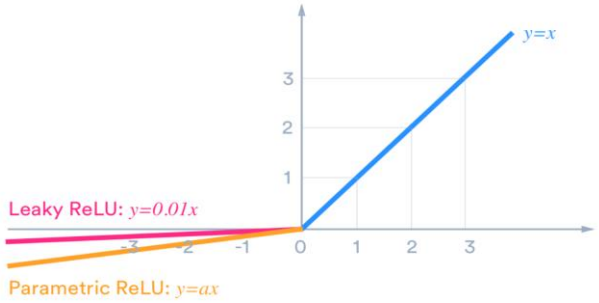
```python
def forward(self, x):
    out = self.conv1(x)
    out = self.conv2(out)
    out = self.activate(self.batchnorm1(out))
    out = F.dropout(self.maxpool(out), p=0.5)
    out = self.conv3(out)
    out = self.activate(self.batchnorm2(out))
    out = F.dropout(self.maxpool(out), p=0.5)
    out = self.conv4(out)
    out = self.activate(self.batchnorm3(out))
    out = F.dropout(self.maxpool(out))
    out = self.conv5(out)
    out = self.activate(self.batchnorm4(out))
    out = F.dropout(self.maxpool(out), p=0.5)
    out = out.view(-1, 8600) # flatten
    out = self.linear(out) # dense layer

    return out
```

B. Explain the activation function

(1). ReLU: This activation function is easy, it returns the input itself when the input is positive, returns zero when the input is negative or zero.

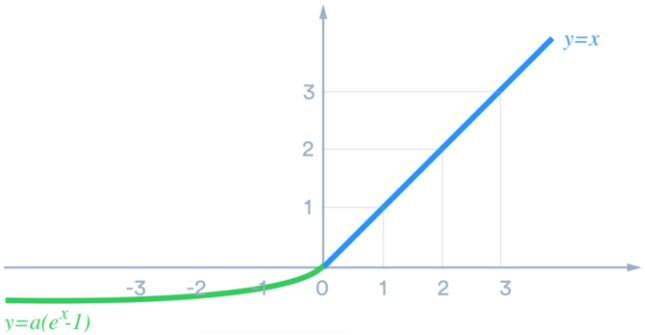


(2). Leaky ReLU: It's like ReLU function but with little difference. Leaky ReLU has a small slope for negative values, instead of altogether zero. For example, leaky ReLU may have y = 0.01x when x < 0.



(3). ELU: It is designed to combine the good parts of ReLU and leaky

ReLU.



3. Experimental results

A. The highest test accuracy

(1). Screenshot with two models

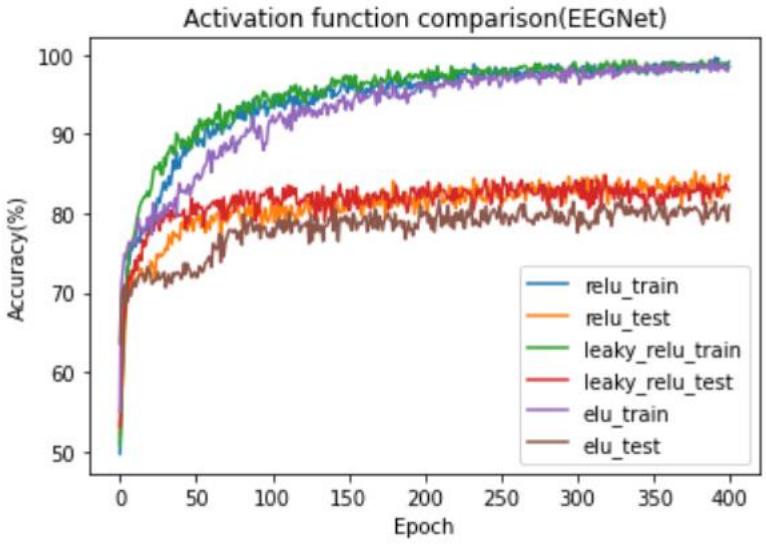
	ReLU	Leaky ReLU	ELU
EEGNet	**84.72%**	**82.87%**	**81.02%**
DeepConvNet	72.59%	73.15%	77.31%

(2). Anything you want to present

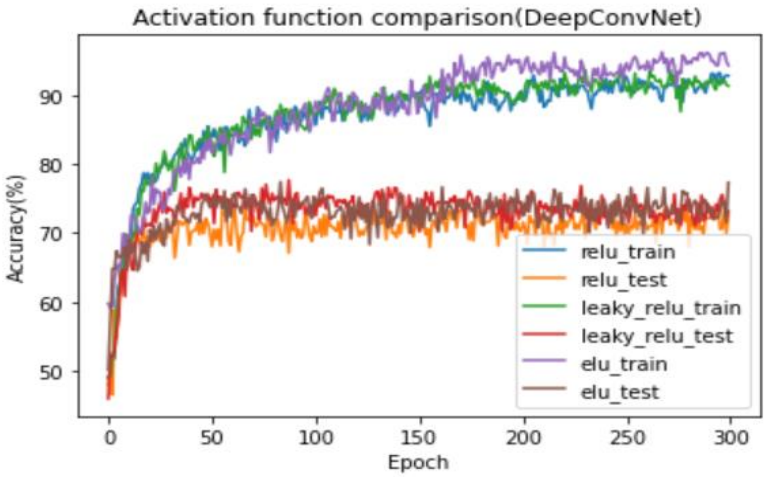
No.

B. Comparison figures

(1). EEGNet



(2). DeepConvNet



4. Discussion
   A. Since it takes less time for EEGNet, I only modify the hyperparameters in EEGNet and don't change the hyperparameters in DeepConvNet.
   B. Final hyperparameters setting for EEGNet:
   batch_size=400 , lr=0.02 , epochs=400 optimizer=SGD with momentum=0.9
   C. By the result shown above, we can see that EEGNet can get good accuracy in the training dataset, however, in the test dataset, the greatest accuracy is about 84%, so I think the model is overfitting, maybe I have to do some manipulations on the input data rather than blindly tuning the hyperparameters.