DL Lab2 EEG classification

0756079 陳冠聞

April 9, 2019

1 Introduction

EEG (Electroencephalography) is an electrophysiological monitoring method to record electrical activity of the brain, and it is used in various applications such as diagnose epilepsy or activation recognition. In this lab, I implement two CNN architectures with three activation functions to do the EEG classification tasks, and compare the performance between them.

2 Experimental Setup

2.1 EEGNet

I first implement the EEGNet with PyTorch package. To make more flexible, I set the activation function and dropout ratio as parameters for model initialization. All setting of the EEGNet are the same as the one provided by TA, except for the default dropout ratio is set to 0.5, while the dropout ratio for TA is 0.25. The reason of change dropout ratio is because of model performance, and will be mention in discussion part. The detail implementation is shown in Figure 1.

2.2 DeepConvNet

I also implement the DeepConvNet with PyTorch package. Same as EEGNet, I set the activation function and dropout ratio as model initialization parameters, and all the other setting are the same as the one provided by TA. The detail implementation is shown in Figure 2.

```python
class EEGNet(nn.Module):
    def __init__(self, activation_name='ELU', dropout_ratio=0.5):
        super(EEGNet, self).__init__()
        self.conv_1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=(1, 51), stride=(1,1), padding=(0,25), bias=False),
            nn.BatchNorm2d(16),
        )
        self.depthwise_conv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(2, 1), stride=(1,1), groups=16, bias=False),
            get_activation_function(activation_name),
            nn.AvgPool2d(kernel_size=(1,4), stride=(1,4), padding=0),
            nn.Dropout(p=dropout_ratio),
        )
        self.separable_conv = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=(1, 15), stride=(1,1), padding=(0,7),bias=False),
            nn.BatchNorm2d(32),
            get_activation_function(activation_name),
            nn.AvgPool2d(kernel_size=(1,8), stride=(1,8), padding=0),
            nn.Dropout(p=dropout_ratio),
        )
        self.classifier = nn.Sequential(
            nn.Linear(in_features=736, out_features=2, bias=True)
        )

    def forward(self, x):
        out = self.conv_1(x)
        out = self.depthwise_conv(out)
        out = self.separable_conv(out)
        out = out.view(-1, 736) # flattern
        out = self.classifier(out)
        return out
```

Figure 1: Code Snippet for EEGNet

```
class DeepConvNet(nn.Module):
    def __init__(self, activation_name='ELU',dropout_ratio=0.5):
        super(DeepConvNet, self).__init__()
        self.conv_1 = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size=(1, 5), bias=True),
            nn.Conv2d(25, 25, kernel_size=(2, 1), bias=True),
            nn.BatchNorm2d(25),
            get_activation_function(activation_name),
            nn.MaxPool2d(kernel_size=(1,2), stride=(1,2), padding=0),
            nn.Dropout(p=dropout_ratio),
        )
        self.conv_2 = nn.Sequential(
            nn.Conv2d(25, 50, kernel_size=(1, 5), bias=True),
            nn.BatchNorm2d(50),
            get_activation_function(activation_name),
            nn.MaxPool2d(kernel_size=(1,2), stride=(1,2), padding=0),
            nn.Dropout(p=dropout_ratio),
        )
        self.conv_3 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size=(1, 5), bias=True),
            nn.BatchNorm2d(100),
            get_activation_function(activation_name),
            nn.MaxPool2d(kernel_size=(1,2), stride=(1,2), padding=0),
            nn.Dropout(p=dropout_ratio),
        )
        self.conv_4 = nn.Sequential(
            nn.Conv2d(100, 200, kernel_size=(1, 5), bias=True),
            nn.BatchNorm2d(200),
            get_activation_function(activation_name),
            nn.MaxPool2d(kernel_size=(1,2), stride=(1,2), padding=0),
            nn.Dropout(p=dropout_ratio),
        )
        self.fc = nn.Linear(8600, 2)
    def forward(self, x):
        out = self.conv_1(x)
        out = self.conv_2(out)
        out = self.conv_3(out)
        out = self.conv_4(out)
        out = out.view(x.shape[0], -1)
        out = self.fc(out)
        return out
```

Figure 2: Code Snippet for DeepConvNet

2.3 Activation Functions

ReLU (Rectified Linear Unit) is the most popular choice of hidden unit, because the gradient of it will not saturate in positive area, and it is computationally efficient, which leads to faster converge than traditional sigmoid unit. However, ReLU also has downsides. The most significant one is that the gradient will be zero at the negative area, resulting in the corresponding pararmeters will not be updated.

To solve the problem of dead ReLU, there are a few choices. One choice is to use the LeakyReLU, which makes the negative part have a small slope instead of zero. ELU (Exponential Linear Units) is another choice, which also makes the negative area have small gradient, but will saturate when input is very negative.

Although both LeakyReLU and ELU have the advantage that model can update parameters even in negative area, but in practice, there are no significant evidence to support they are better than ReLU, and the problem of ELU and LeakyReLU is that the computation is more expensive than ReLU.

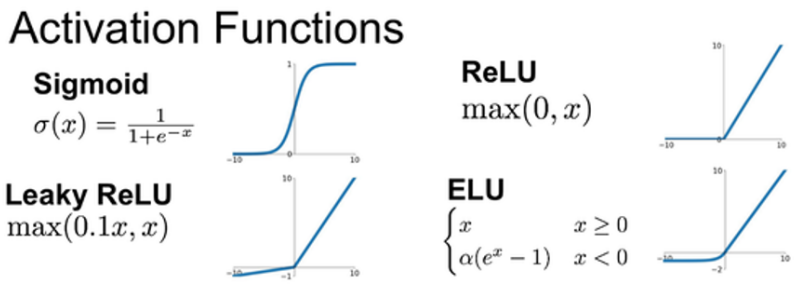

Figure 3: Different Activation Functions

3 Experimental Result

3.1 Training setting

I train the EEGNet with batch size 128 for 800 epochs. For optimizer, I use SGD with momentum 0.5. Initial learning rate is set to 0.1, then multiply by 0.5 at epoch 300, 500 and 700.

For the DeepConvNet, I use batch size 128 for 800 epochs, and SGD with momentum 0.9 as optimizer. Initial learning rate is set to 0.05, then multiply by 0.1 at epoch 300, 500 and 700.

3.2 Best Accuracy

The best accuracy is achieved by EEGNet using LeakyReLU as activation function. We can see that EEGNet outperform DeepConvNet for all activation functions. The reason might be that DeepConvNet have more number of parameters ($\approx 151k$) comparing to EEGNet ($\approx 18k$), and the training data size is relative small (only 1080 instances), so DeepConvNet is easier to overfitting. Another reason might be that I haven't find the best hyperparameters for DeepConvNet, so the comparison is just not fair.

	EEGNet	DeepConvNet
ReLU	87.87%	82.68%
LeakyReLU	88.61%	82.40%
ELU	84.35%	80.46%

Figure 4: Accuracy Comparison

3.3 Accuracy Trend

From Figure 5 we can see that ReLU and LeakyReLU can converge faster and reach higher accuracy both in training and test accuracy than ELU. In addition, Figure 6 shows that the ELU converges faster, but the test accuracy is the lowest. So although ELU is superior than ReLU in theory, in practice it's not guarantee to perform better.

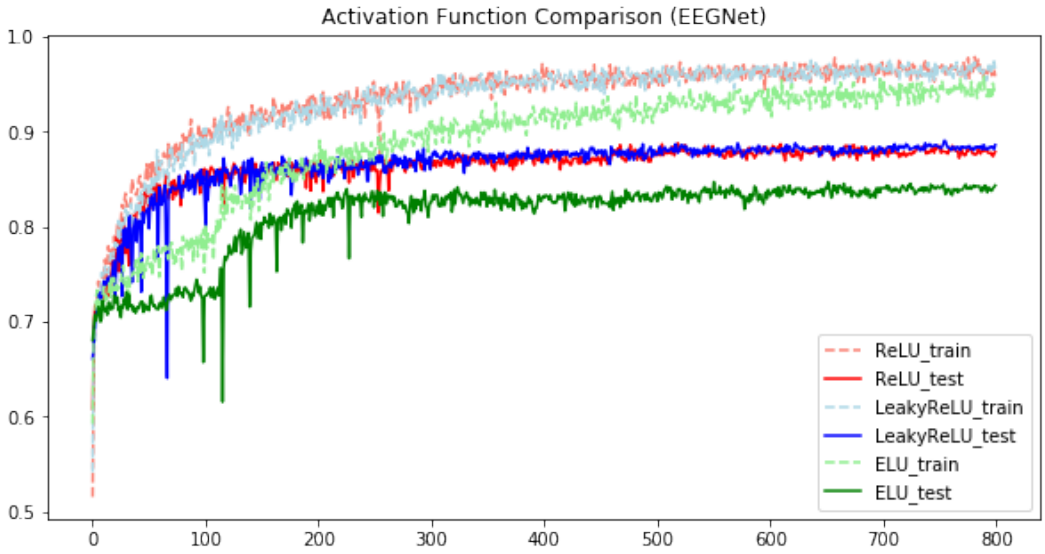

Figure 5: Activation Function Comparison for EEGNet

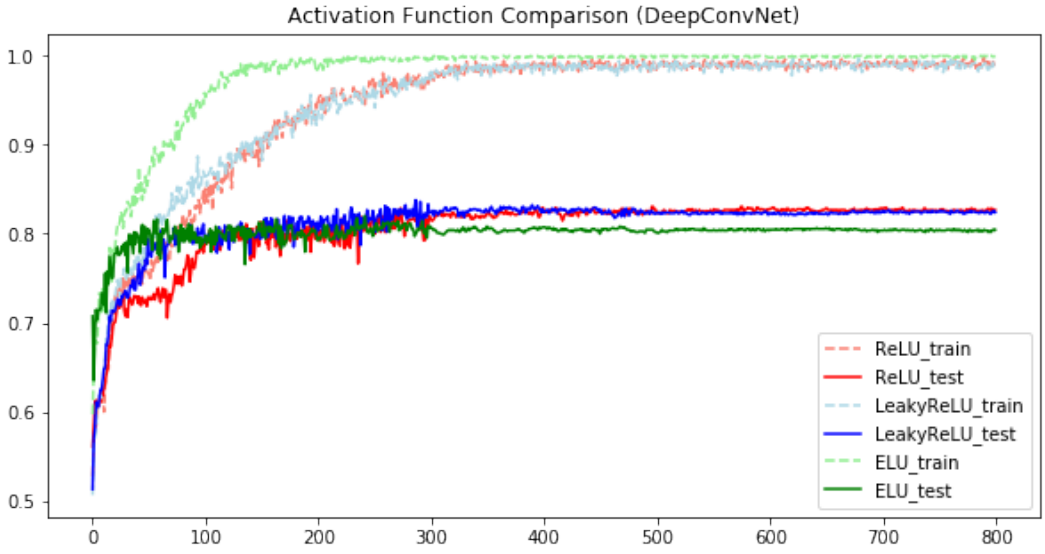

Figure 6: Activation Function Comparison for DeepConvNet

4 Discussion

4.1 Dropout ratio

In this lab, I choose dropout ratio $p = 0.5$ for EEGNet instead of $p = 0.25$ (provided by TA). The reason is that I observed that when dropout ratio is 0.25, the training accuracy converge to 100% very quickly, but test accuracy stuck at about 80%, which imply that the model is too strong, and it may fit the noise in the training data, resulting in bad generalization.

To make the model more generalizable, I raise the dropout ratio to 0.5. As shown in the Figure 7[1], $p = 0.5$ (light blue line) converge slower than $p = 0.25$ (light red line), but the test accuracy is higher (88.05% v.s. 85.27%). I also do experiment on $p = 0.75$, but it's seem that the model become too weak. The training accuracy of $p = 0.75$ can only reach about 88%, and the test error is 84.72%, which is even worse than $p = 0.25$ despite that the generalization gap of $p = 0.75$ is the smallest among all.

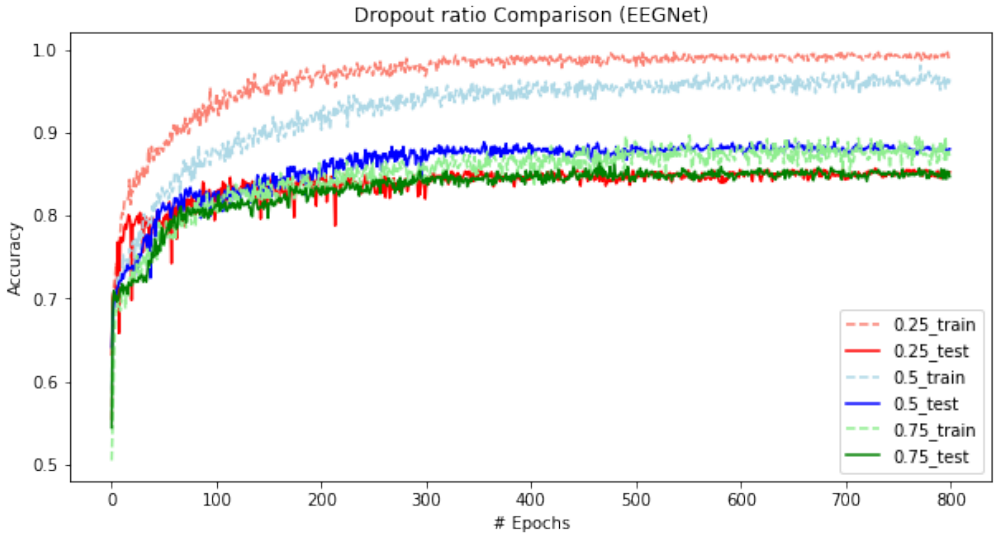

Figure 7: Dropout ratio Comparison for EEGNet

[1]Use ReLU and the same hyperparameters as in the section 3.1

4.2 EEGNet vs DeepConvNet

In this lab, the EEGNet outperforms the DeepConvNet for all activation functions despite that the number of parameters of EEGNet is much less. Furthermore, training EEGNet for 800 epochs take about 140 seconds on my 1080Ti machine, and it takes about 210 seconds to train DeepConvNet for 800 epochs. What I have learned from this lab is that instead of just stacking up many layers without thinking to build the deep neural network, understanding of the data and careful design of the model architecture can really help the model to run faster, reduce memory overhead, and even perform better.

	EEGNet	DeepConvNet
Best Accuracy	88.61%	82.68%
Training Time	~140s	~210s
# parameters	~18K	~150K

Figure 8: Comparison between EEGNet and DeepConvNet