Lab3: EEG classification

Hsiang-Chun Yang
Institute of Multimedia Engineering
National Yang Ming Chiao Tung University
`yanghc.cs09g@nctu.edu.tw`

1 Introduction

In this lab, we need to implement two neural network structures, EEGNet and DeepConvNet, and compare the EEG classification accuracy between different model with three activation functions, ReLU, Leaky ReLU, and ELU.

1.1 Dataset

- Training data: `S4b_train.npz`, `X11b_train.npz`
- Testing data: `S4b_test.npz`, `X11b_test.npz`

There are 1080 data in both training and testing set. Each data consists of two channels of EEG signals. Figure 1 is the visualization of a single data.

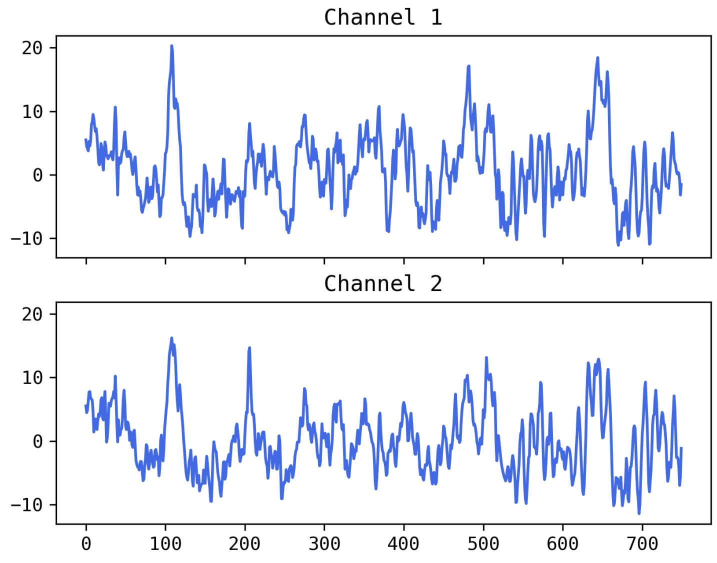

Figure 1: Visualization of EEG data

2 Experiment setup

2.1 EEGNet

Figure 2 is the overall visualization of the EEGNet. The design of EEGNet is to reduce the computational cost. It separates conventional convolution operation into a depth-wise convolution and a separable convolution. `DepthwiseConv2D` will learn the correlation between different signal channels, then `SeparableConv2D` will learn how to combine each feature map.

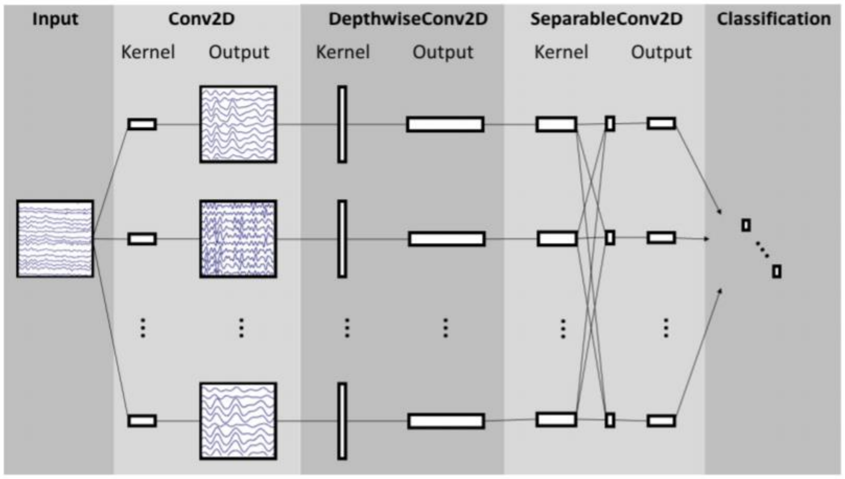

Figure 2: Architecture of EEGNet

The implementation of EEGNet is as follows.

```
class EEGNet(nn.Module):
    def __init__(self, activation):
        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)

        self.firstconv = nn.Sequential(
            nn.Conv2d(1, 16, (1, 51), (1, 1), (0, 25), bias=False),
            nn.BatchNorm2d(16))

        self.depthwiseconv = nn.Sequential(
            nn.Conv2d(16, 32, (2, 1), (1, 1), groups=16, bias=False),
            nn.BatchNorm2d(32),
            self.activation,
            nn.AvgPool2d((1, 4), (1, 4), (0, 0)),
            nn.Dropout(p=0.25))

        self.separableconv = nn.Sequential(
            nn.Conv2d(32, 32, (1, 15), (1, 1), (0, 7), bias=False),
            nn.BatchNorm2d(32),
            self.activation,
            nn.AvgPool2d((1, 8), (1, 8), (0, 0)),
            nn.Dropout(p=0.25))

        self.classify = nn.Sequential(nn.Linear(736, 2))

    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
```

2.2 DeepConvNet

DeepConvNet is just a CNN model with normal convolution operations. The implementation of DeepConvNet is as follows.

```
class DeepConvNet(nn.Module):
    def __init__(self, activation):
        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)

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

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

        self.conv2 = nn.Sequential(
            nn.Conv2d(50, 100, (1, 5), (1, 1), (0, 0)),
            nn.BatchNorm2d(100),
            self.activation,
            nn.MaxPool2d((1, 2)),
            nn.Dropout(p=0.5))

        self.conv3 = nn.Sequential(
            nn.Conv2d(100, 200, (1, 5), (1, 1), (0, 0)),
            nn.BatchNorm2d(200),
            self.activation,
            nn.MaxPool2d((1, 2)),
            nn.Dropout(p=0.5))

        self.classify = nn.Sequential(nn.Linear(43*200, 2))

    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
```

2.3 Activation functions

- ReLU$(x) = max(0, x)$
- Leaky ReLU(x) $\begin{cases} x, \text{ if } x \geq 0 \\ \text{negative\_slope} \times x, \text{ otherwise} \end{cases}$
- ELU$(x) = max(0, x) + min(0, \alpha * (e^x - 1))$

By default, α in ELU is set to 1.0 and negative slope in Leaky ReLU is set to 0.01.

The disadvantage as for ReLu is that the gradient will be 0 when the input value is negative. Which means that the neurons cannot update their weights in this circumstance. This is so-called dying ReLu problem. While ELU and Leaky ReLU can avoid this problem because of the design at negative part of input values.

2.4 Experiment Setup

The hyper-parameters for experiments are as follows.

- optimizer: RAdam [1]
- loss function : cross entropy
- number of training epochs: 300
- batch size: 32, 64
- learning rate: 0.001
- activation function: ReLu, Leaky ReLU, ELU
- weight decay: 0, 0.001, 0.1

There will be 18 trials for both EEGNet and DeepConvNet.

3 Experimental results

3.1 The highest testing accuracy

	ReLU	Leaky ReLU	ELU
EEGNet	**87.59%**	**87.78%**	**85.74%**
DeepConvNet	83.61%	83.70%	81.76%

The hyper-parameters of each of the highest accuracy model are as follows.

	ReLU	Leaky ReLU	ELU
EEGNet	weight decay: 0.1 batch size: 32	weight decay: 0.1 batch size: 32	weight decay: 0.1 batch size: 64
DeepConvNet	weight decay: 0 batch size: 32	weight decay: 0.001 batch size: 64	weight decay: 0.1 batch size: 64

3.2 Comparison figures

3.2.1 EEGNet

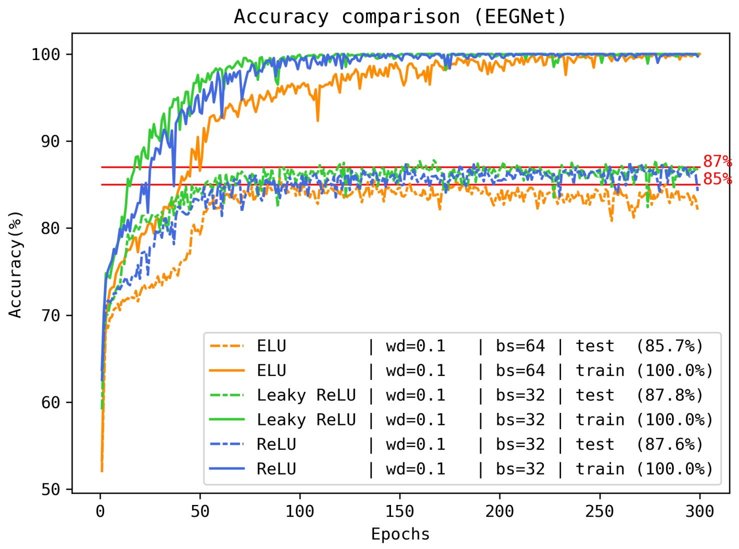

Figure 3: Comparison between different activation functions using EEGNet

3.2.2 DeepConvNet

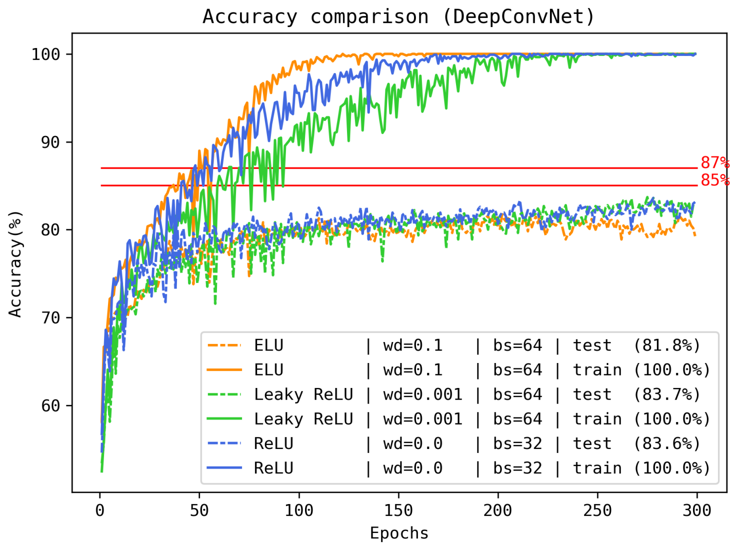

Figure 4: Comparison between different activation functions using DeepConvNet

4 Discussion

4.1 Weight decay

The purpose of weight decay is to prevent overfitting, and our experimental results just match this statement. Looking at the table in the section 3.1, we can observer that using weight decay mechanism can increase the performance of the model in most cases.

4.2 EEGNet v.s. DeepConvNet

The following table shows the number of parameters of EEGNet and DeepConvNet.

	number of parameters
EEGNet	17,874
DeepConvNet	150,977

The number of parameters of DeepConvNet is about 9 times larger than EEGNet. But according to the table and figures in section 3.1 and section 3.2, EEGNet always outperform DeepConvNet. It seems that EEGNet is better for extracing EEG signal features with depth-wise and separable convolution design. Also, as we mentioned in section 2.1, the design of EEGNet is more computational efficiency that DeepConvNet.

5 References

[1] Liu, Liyuan and Jiang, Haoming and He, Pengcheng and Chen, Weizhu and Liu, Xiaodong and Gao, Jianfeng and Han, Jiawei (2020) On the Variance of the Adaptive Learning Rate and Beyond *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*