Deep Learning Lab2 **Student ID: 312554014 Name: 羅翊軒**

1. Introduction

In this lab, we are asked to implement EEGNet and DeepConvNet. With some hyper parameters tuning and trying different activation function, we need to make accuracy the higher the better.

A. Environment

The environment and Python packages of this lab are as follows:

Processor	Intel® Core™ i5-13400F / GeForce RTX 3060 Ti
Python version	3.9.17
Python packages	random, Numpy, Pytorch, Matplotlib

B. Files

All the code is in one file, which file type is .ipynb. In this file, it will first make dataset with the previously given file dataloder.py, then creates two types of models and three activation function individually.

2. Experiment Set Up

A. The detail of your model

◆ EEGNet

```
EEGNet_ELU(
  (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, inplace=False)
  )
  (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, inplace=False)
  )
  (classify): Linear(in_features=736, out_features=2, bias=True)
)
```

As shown in the figure above, it's an example structure of EEGNet model with ELU activation function. Including four blocks, three leads by

Conv2d layer and one uses linear layer as the final output layer. In each of the first three blocks, every Conv2d layer is followed by a BatchNorm2d layer, which is used to normalize the features extracts from previous Conv2d layer before entering ELU activation layer. Then, applying AvgPool2d layer to create a down-sampling feature map and using Dropout layer to randomly sets some neurons to zero. It should be noticed that before model returns the final outputs, it will first go through a Softmax layer, so that the sum of all output values is 1.

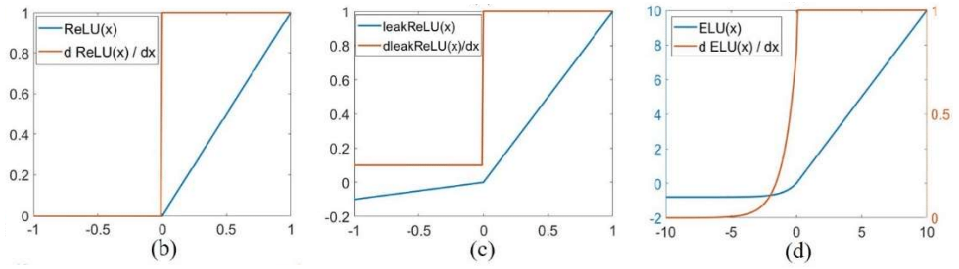
- DeepConvNet

```
DeepConvNet_ELU(
  (Conv2d_1): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), bias=False)
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ELU(alpha=0.5)
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (Conv2d_2): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=0.5)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (Conv2d_3): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=0.5)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (Conv2d_4): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=0.5)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (classify): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=8600, out_features=2, bias=True)
  )
)
```

As shown in the figure above, it's an example structure of DeepConvNet model with ELU activation function. Similar to EEGNet, it has BatchNorm2d layer right after Conv2d layer for normalizing the features extracting from Conv2d layer before entering the activation function layer. Then, it uses MaxPool2d layer as the down-sampling method that different from EEGNet and Dropout layer to randomly set neurons to zero

that same as EEGNet. After returning final outputs, it will go through Flatten layer, Linear layer and Softmax layer as well.

B. Explain the activation function (ReLU, Leaky ReLU, ELU)



- ReLU

  ReLU, rectified linear unit, is an activation which return 0 if it receives negative input else return input value directly, that is, it will turn off the neurons with negative activation. It is computationally efficient activation function widely used in neural network cause it help relaxing gradient vanish problem. However, it still suffers from dying neuron problem if updating too fast or learning is too large.

- Leaky ReLU

  Leaky ReLU is similar to ReLU. It is a modification of ReLU function, and it has a small gradient when the input is negative. With the characteristic above, it allows some information to pass through even when the input is negative and appeases the dying neuron problem.

- ELU

  ELU, exponential linear unit, is an activation function which is designed to appease the vanishing gradient problem and dying neuron problem. Similar to leaky ReLU, ELU has a small slope for negative input as well, but instead of a straight line, it uses a log curve. It combines both pros in sigmoid and ReLU, that has saturability on the left side of y-axis and without saturability on the right side. Although it has so many advantages,

it’s computation is a little bit complex.

3. Experimental results

A. The highest testing accuracy

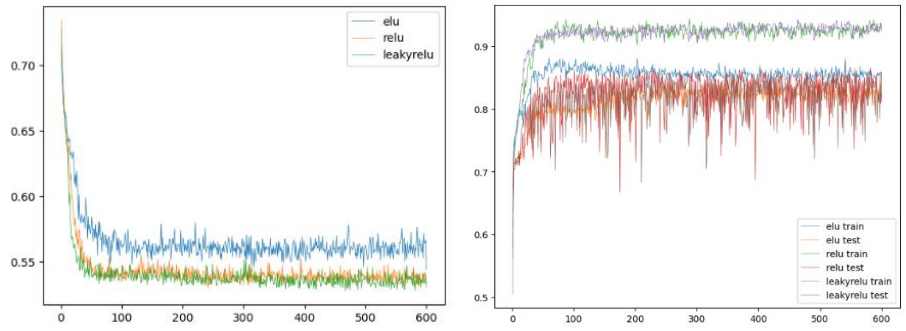
	EEGNet	DeepConvNet
ELU	84.25%	82.31%
ReLU	86.57%	84.25%
Leaky ReLU	**86.75%**	82.87%

EEGNet `Test : elu 0.8425925925925926, relu 0.8657407407407407, leakyrelu 0.8675925925925926`

DeepConvNet `Test : elu 0.8231481481481482, relu 0.8425925925925926, leakyrelu 0.8287037037037037`

B. Comparison figures

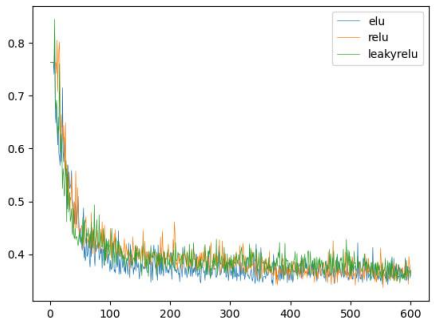
◆ EEGNet

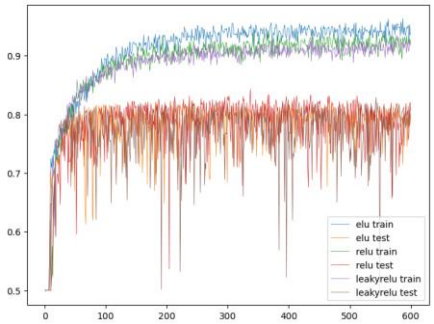


	Batch size	Loss	Optimizer	Learning rate	Train accuracy	Test accuracy
ELU	120	Cross Entropy Loss	Adam	1e-3	88.14%	84.25%
ReLU					94.35%	86.57%
Leaky ReLU					94.25%	86.75%

With batch size=120, learning rate=1e-3, and Adam optimizer, although the train accuracy of EEGNet model with Leaky ReLU activation can only up to 94.25%, the test accuracy with Leaky ReLU can reach 86% in 600 epochs.

◆ DeepConvNet





	Batch size	Loss	Optimizer	Learning rate	Train accuracy	Test accuracy
ELU	120	Cross Entropy Loss	RMSprop	1e-3	93.79%	82.31%
ReLU					92.59%	84.25%
Leaky ReLU					91.75%	82.87%

Different with EEGNet, DeepConvNet with RMSprop optimizer which momentum=0.5 performs better than Adam optimizer. Although ELU has the highest train accuracy, RELU and Leaky ReLU have better test accuracy, and ReLU can reach 84%.

4. Discussion

A. Batch size

After several experiments, I found that batch size between 60 to 120 are the most suitable choice, and the accuracy can up to 85%. When batch size close to 60, ReLU performs better than Leaky ReLU, however, when batch size close to 120, Leaky ReLU performs much better.

B. Activation

For EEGNet and DeepConvNet, ReLU and Leaky ReLU perform much better than ELU. ReLU and Leaky ReLU perform nearly the same, and other parameters will affect ReLU and Leaky ReLU's performance. It needs to see what's other parameters' value to tell which one performs the best.

C. Loss function

For two models, I test some loss function, like CrossEntropyLoss, BCEWithLogitsLoss, MSE. CrossEntropyLoss and BCEWithLogitsLoss both perform well, and I choose CrossEntropyLoss as the final loss function.

D. Optimizer

For EEGNet, optimizer Adam performs better than other optimizers like SGD, RMSprop; for DeepConvNet, optimizer RMSprop performs slightly better than optimizer Adam. The parameters, likes learning rate or weight decay, are really important, they can affect the result greatly.

E. Learning rate

There's a case that when the learning rate's too big, for example 1e-1, the loss will stop decreasing, that is, the loss curve will be a horizontal line, and the model can't fit well. This problem occurs more seriously and frequently when training DeepConvNet than training EEGNet.

F. Others

After trying several times, found that Dropout layer has p = 0.2 performs better than other values in EEGNet and p = 0.35 in DeepConvNet. The most important thing is that the initial weights of the model affect the result greatly.