README

Lab2 Report

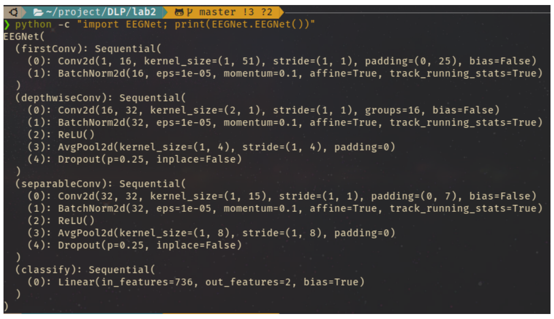
Lab2: EEG Classification
Student: 林聯政 ID: 0856154
Github link: https://github.com/summelon/DLP_Lab

Introduction

- In this lab, I implemented two kinds of neural network from scratch by PyTorch.
- In both networks, I compared the impact of different activation function according to their strategy and performance.
- For better test time accuracy, I have done some experiments to explore the relation of model performance and hyperparameter setting.

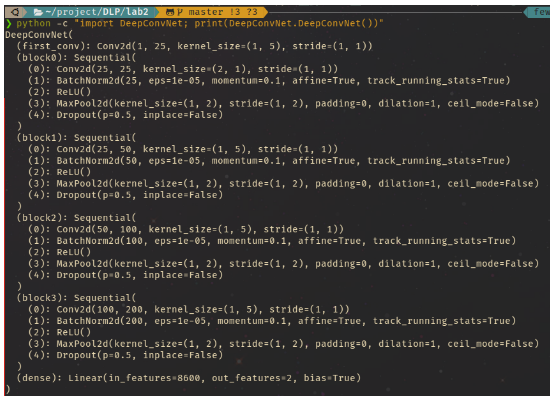
Experiment Setup

1. The detail of models
    - EEGNet



```
~/project/DLP/lab2  master !3 ?2
❯ python -c "import EEGNet; print(EEGNet.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): ReLU()
    (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): ReLU()
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

        - EEGNet consists of a `firstConv` block, a `depthwiseConv` block, a `separableConv` block and `flatten` layer before a `linear` layer.
    - DeepConvNet

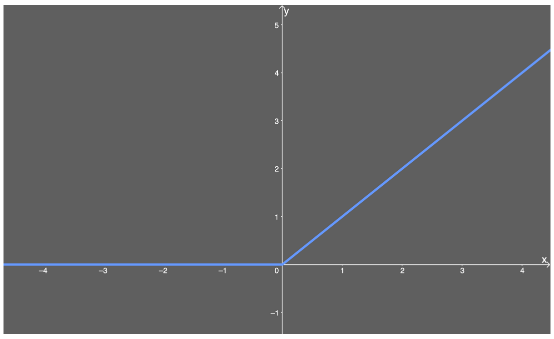


```
~/project/DLP/lab2  master !3 ?3                                        few
❯ python -c "import DeepConvNet; print(DeepConvNet.DeepConvNet())"
DeepConvNet(
  (first_conv): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1))
  (block0): Sequential(
    (0): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (1): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (block1): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (block2): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (block3): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (dense): Linear(in_features=8600, out_features=2, bias=True)
)
```

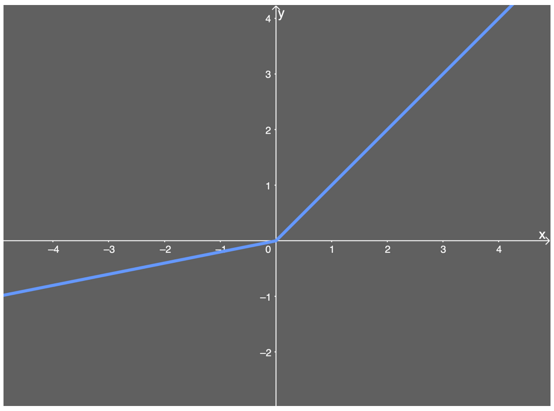
        - There is a single convolution layer in DeepConvNet. It is followed by four basic block which consists of conv2d, bn, activation function, max pooling and a dropout layer. At last, be similar to EEGNet, there is a `flatten` layer followed by a `linear` layer.

2. Explain the activation function

    - ReLU

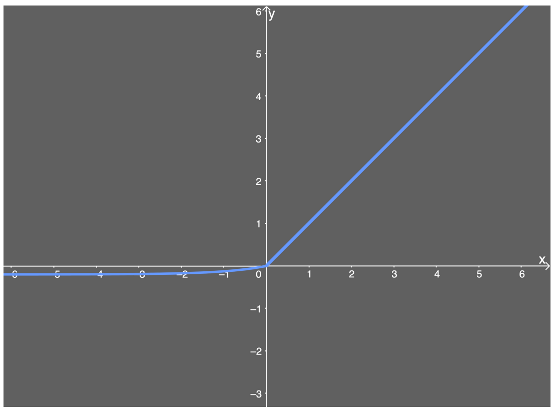
      

        - Pros
            - Easy for implement, low computational cost
            - Part of linear, but non-linearity
        - Cons
            - Neurons less than zero will die. The gradient of these parts become zero. These neuron won't be updated anymore.
    - Leaky ReLU

        - Pros
            - Small positive slope in the negative part, enable back propagation for negative input.
        - Cons
            - Inconsistent gradient descent may lead to unexpected results

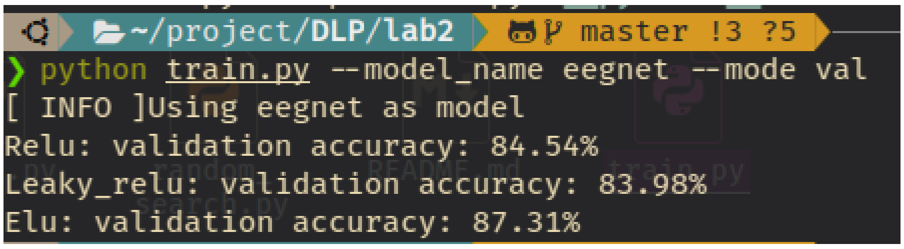
- ELU

  - Pros
    - It has not inconsistency problem compare to Leaky ReLU and dying neurons problem compare to ReLu.
    - Normally it should converge faster and achieve higher accuracy because of its merits.
  - Cons
    - It is worth using when training for faster convergence. On the other hand, it takes much more coputational cost in test time compare to ReLU and its variants.

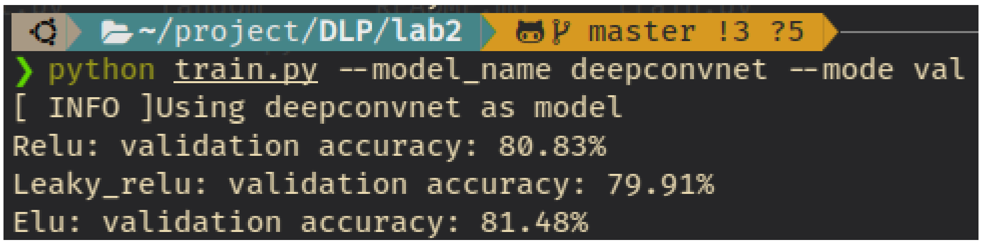
Experimental Results

1. The highest testing accuracy
   - Screenshot with two models
     - EEGNet

       ```
       ~/project/DLP/lab2  master !3 ?5
       ❯ python train.py --model_name eegnet --mode val
       [ INFO ]Using eegnet as model
       Relu: validation accuracy: 84.54%
       Leaky_relu: validation accuracy: 83.98%
       Elu: validation accuracy: 87.31%
       ```

     - DeepConvNet

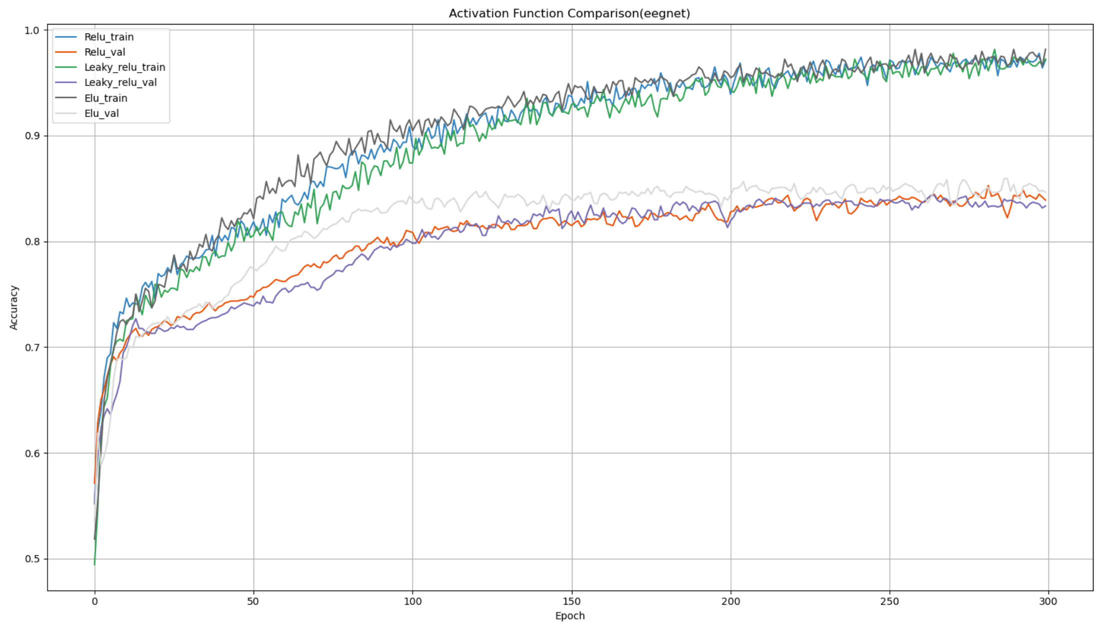
       

       ```
       ~/project/DLP/lab2  master !3 ?5
       ❯ python train.py --model_name deepconvnet --mode val
       [ INFO ]Using deepconvnet as model
       Relu: validation accuracy: 80.83%
       Leaky_relu: validation accuracy: 79.91%
       Elu: validation accuracy: 81.48%
       ```

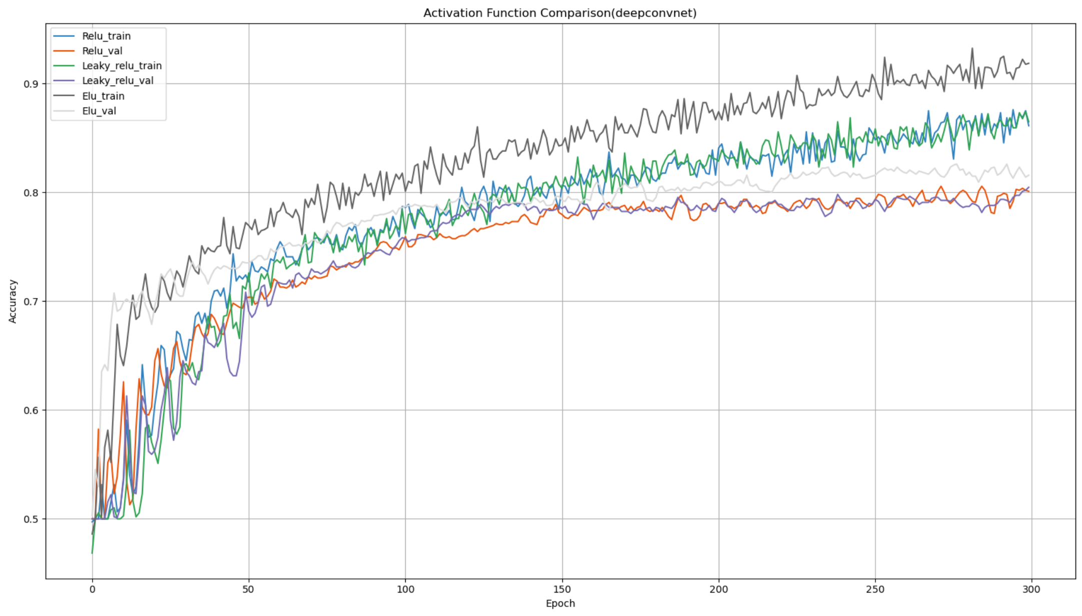
   - From the result, we can found that EEGNet is easier to train than DeepConvNet. In my opinion, because of the simple data in this time, shallow model convergences faster than complex model.
   - The experimental results also proves the advantage of activations I introduced above. The performance of ELU is always the best, which is as

our expected. However, its shortcoming is hidden here, due to we do not measure the test time latency here.

2. Comparison figures
    - EEGNet



    - DeepConvNet



Discussion

1. Different initialization will also affect the convergence speed and highest accuracy of the model.
2. In my experiments, use batch size as large as possible, takes advantage of better overall gradient updating. It means that we are able to update model to the most correct direction.
3. I found that suitable learning rate is important in this lab. If I use large learning rate, such as 1e-1 or 1e-2, later in the training phase, model loss may have fluctuation problem. On the other hand, too small learning rate will slow down the

convergence time. Consequently, a learning rate in a suitable value is benifit for both rapid convergence time and smooth convergence curve

Reference

https://mlfromscratch.com/activation-functions-explained/#/
https://deeplearninguniversity.com/elu-as-an-activation-function-in-neural-networks/
https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/