DLP2023 Lab2 Report

DLP

Introduction

In this lab, we need to implent EEGNet and a DeepConvNet using pytorch. And use these two model to make predictions on BCI competition dataset.

Experiment Setup

MODELS

1. In traning I use cross entropy as loss functions, and Adam as optimizer.

- EEGNet
    1. I add a flaaten function between Layer4 and classify layer to match the matrix multiplication.
    2. I use cross entropy as loss function while training. In Pytorch, the nn.functional.cross_entropy have already include softmax function inside it. So we don't need add a nn.Softmax in our model.
    3. softmax is a function that will convert input to multi-class probability distrbution, which have sum = 1.

```
EEGNet(
  (firstconv): Sequential(
    (0): Conv2d(1, 16, kernel_size=(1, 51), stri
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1
  )
  (depthwiseConv): Sequential(
    (0): Conv2d(16, 32, kernel_size=(2, 1), stri
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1
    (2): ActivationLayer(
      (activation): LeakyReLU(negative_slope=0.0
    )
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1
    (4): Dropout(p=0.25, inplace=False)
  )
  (seperableConv): Sequential(
    (0): Conv2d(32, 32, kernel_size=(1, 15), str
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1
    (2): ActivationLayer(
      (activation): LeakyReLU(negative_slope=0.0
    )
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1
    (4): Dropout(p=0.25, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2,
  )
)
```

The reason why the EEGNet is more efficient than DeepConvNet is that it select Depthwise Convolution and Sperable Convolution. They can reduce the time complexity caused by convolution computations. Also, acheive the similar performance.

- DeepConvNet
  1. I add a flaaten function between Layer4 and classify layer to match the matrix multiplication.
  2. I use cross entropy as loss function while training. In Pytorch, the nn.functional.cross_entropy have already include softmax function inside it. So we don’t need add a nn.Softmax in our model.
  3. softmax is a function that will convert input to multi-class probability distrbution, which have sum = 1.

```
DeepConvNet(
  (Layer1): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stric
    (1): Conv2d(25, 25, kernel_size=(2, 1), stri
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1
    (3): ActivationLayer(
      (activation): LeakyReLU(negative_slope=0.0
    )
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1
    (5): Dropout(p=0.5, inplace=False)
  )
  (Layer2): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stri
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1
    (2): ActivationLayer(
      (activation): LeakyReLU(negative_slope=0.0
    )
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1
    (4): Dropout(p=0.5, inplace=False)
  )
  (Layer3): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), str
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.
    (2): ActivationLayer(
      (activation): LeakyReLU(negative_slope=0.0
    )
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1
    (4): Dropout(p=0.5, inplace=False)
  )
  (Layer4): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), st
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.
    (2): ActivationLayer(
      (activation): LeakyReLU(negative_slope=0.0
    )
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1
    (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
  )
)
```

ACTIVATION FUNCTIONS

- ReLU

```
ReLU(x)=max(0,x)
```

- Leaky ReLU

```
LeakyReLU(x)=max(0,x)+negative_slope∗min(0,x)
# where negative_slpoe = 1e-2
```

- ELU

```
ELU(x)=
{ x,              if x > 0
  α∗(exp(x)−1),   if x≤0
}
# where α = 1.0
```

Experiment Results

For experiments, I change Activation Layer for different activation functions.

Highest Testing Accuracy

```python
model = EEGNet(torch.nn.ReLU())
model = model.to(device)
model.load_state_dict(torch.load('./weights/EEG_ReLU_874'))
test_loss, test_acc = test(model, test_loader)
print("EEG_ReLU",test_acc)
```
✓ 0.0s

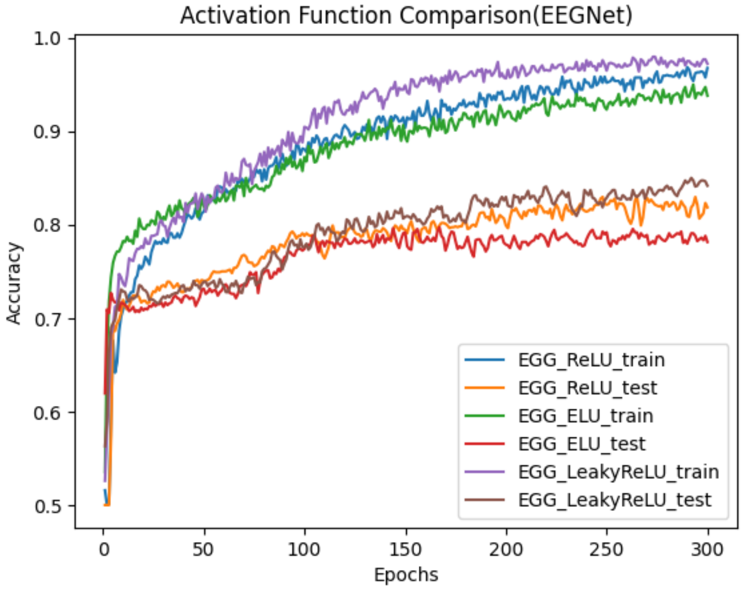
```
EEG_ReLU 0.8740740740740741
```

```python
model = DeepConvNet(torch.nn.ReLU())
model = model.to(device)
model.load_state_dict(torch.load('./weights/deep_ReLU'))
test_loss, test_acc = test(model, test_loader)
print("deep_ReLU", test_acc)
```
✓ 0.0s

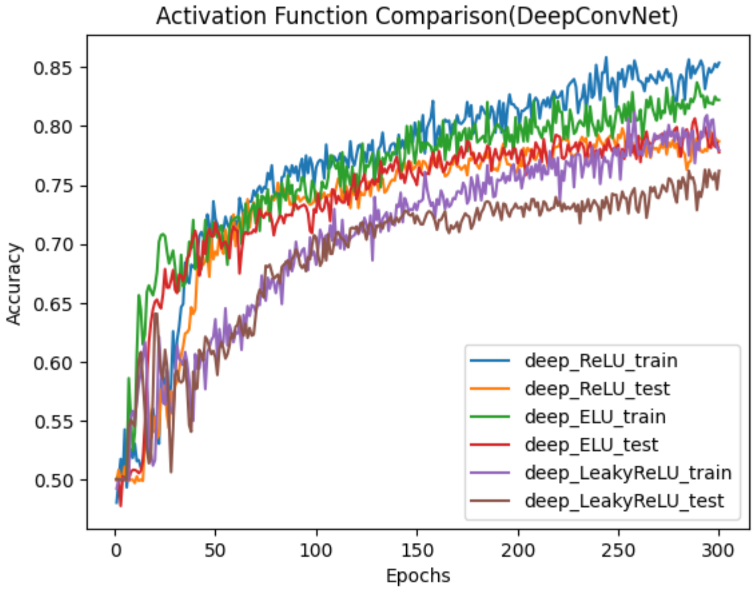
```
deep_ReLU 0.8009259259259259
```

Comparison Figures

- EEGNet



- DeppConvNet



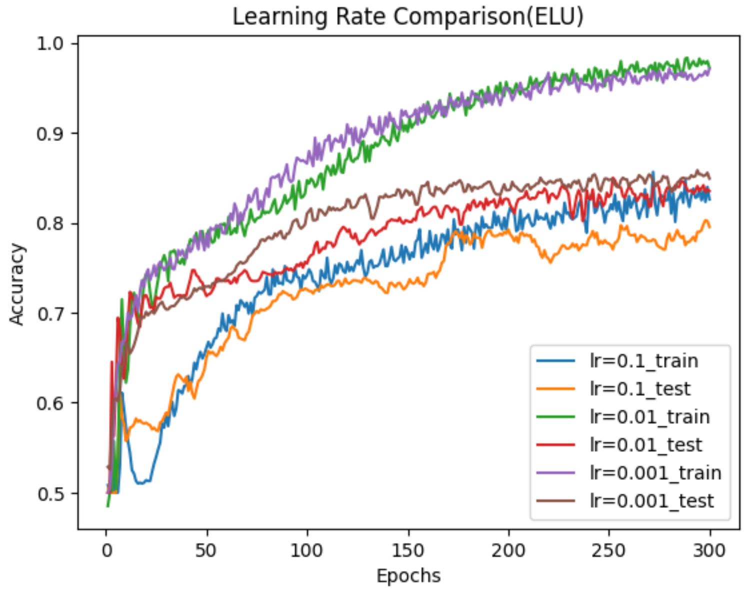
Discussion

TIME COMPARISON

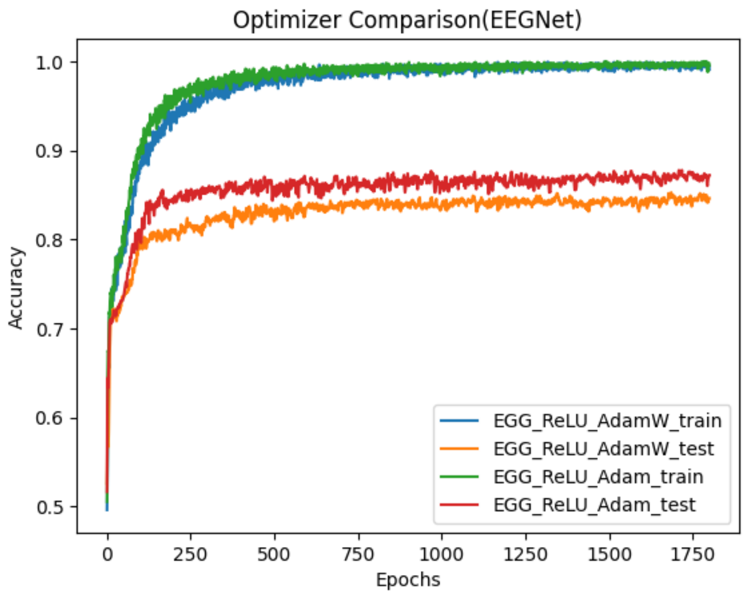
1. As we can see, in 300 epochs, the EEGNet have better accuracy than DeepConvNet. Moreover, EEG is more effective than DeppConvNet.
   - EEG_ReLU: 28.1s
   - EEG_ELU: 29.0s
   - EEG_LeakyReLU: 28.2s
   - DeepConvNet_ReLU: 1m 14.4s
   - DeepConvNet_ELU: 1m 16.7s
   - DeepConvNet_ELU: 1m 14.6s

2. And we can see that LeakyReLU or ReLU are more fit EEGNet, while Leaky ReLU is not that fit DeepConvNet.

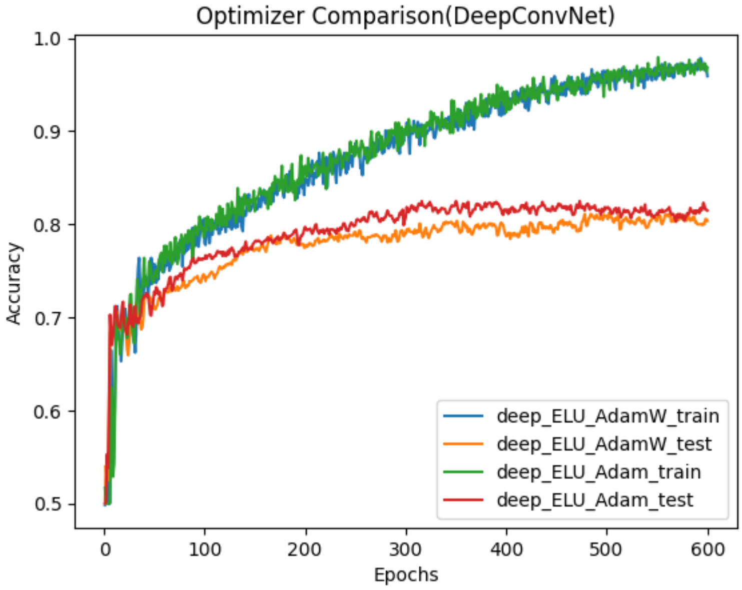
3. I also comapre the accuracy based on different learning rate on EEGNet when I fine tune the model.



4. Also, to produce highest accuracy, I try another optimzer - AdamW when training. AdmW is weight decay version of Adam optimizer. And weight decay is a regulariztion technique which scales down the weights in every step. This allows the model to be more generalizable, and solve the overffiting problem.
   Howerver, When I set the lr=0.001, actually the optimizer Adam perform better than AdamW significantly.



   This situation is the same on DeepConNet.



Reference

https://iclr-blogposts.github.io/2023/blog/2023/adamw/ (https://iclr-blogposts.github.io/2023/blog/2023/adamw/)

https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html (https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html)