Lab2 - EEG Classification

Introduction

In this lab, I implemented EEG classification using EEGNet and DeepConvNet with three activation functions, including ReLU, LeakyReLU, and ELU.

Implementation Details

EEGNet & DeepConvNet

Initialize the network based on the specs. The followings is the created network structures:

```
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)
  )
  (classifier): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

```
DeepConvNet(
  (firstConv): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1))
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ReLU()
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (secondConv): 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)
```

```
  )
  (thirdConv): 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)
  )
  (fourthConv): 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)
  )
  (classifier): Sequential(
    (0): Linear(in_features=8600, out_features=2, bias=True)
  )
)
```

Activation Functions

- ReLU

$$y = max(x, 0)$$

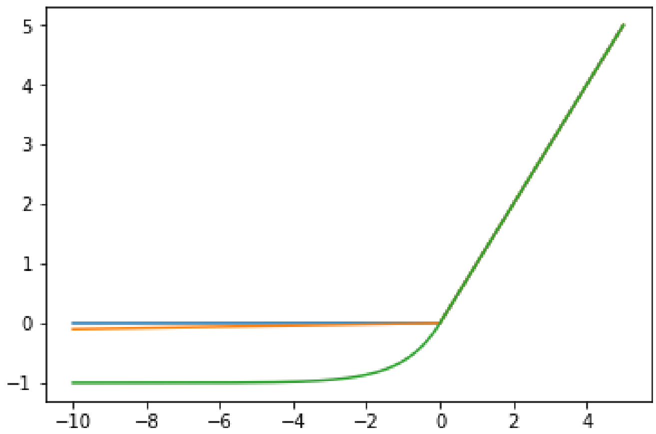
- Leaky ReLU

$$y = \begin{cases} x, & \text{if } x > 0 \\ 0.01 \times x, & \text{if } x \leq 0 \end{cases}$$

- ELU

$$y = \begin{cases} x, & \text{if } x > 0 \\ \alpha(\mathrm{e}^x - 1), & \text{if } x \leq 0 \end{cases}$$

- Comparison



Data Splitting

Defined a function to split data into batches.

```
def data_loader(x, bs=64):
    for i in range(len(x) // bs):
        yield x[i*bs:(i+1)*bs]
    if len(x)%bs != 0:
        yield x[(i+1)*bs:]
```

Experiment Reproducibility

To reproduce the experiments with best results, set the random seed before training.

```
seed = 95
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
np.random.seed(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

Training Steps

In each epoch:

- Split the data into batches

```
for x, y in zip(data_split(X[idx]), data_split(Y[idx])):
    ...
```

- Copy the data to device (GPU)

```
x = x.to(device)
y = y.to(device)
```

- Output the results (forward) & calculate the loss

```
outputs = model(x)
loss = criterion(outputs, y)
lossList.append(loss.item())
```

- Back propagation & optimize the network with the gradients

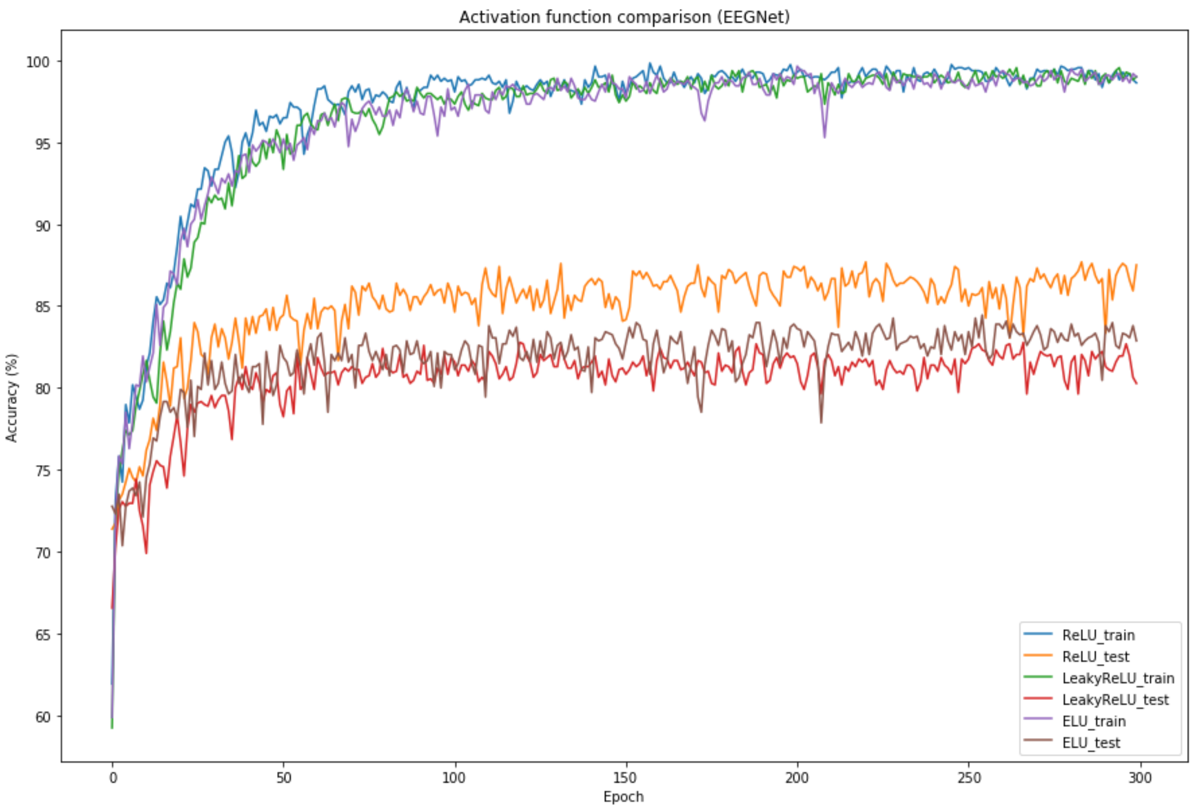
```
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Experiment Results

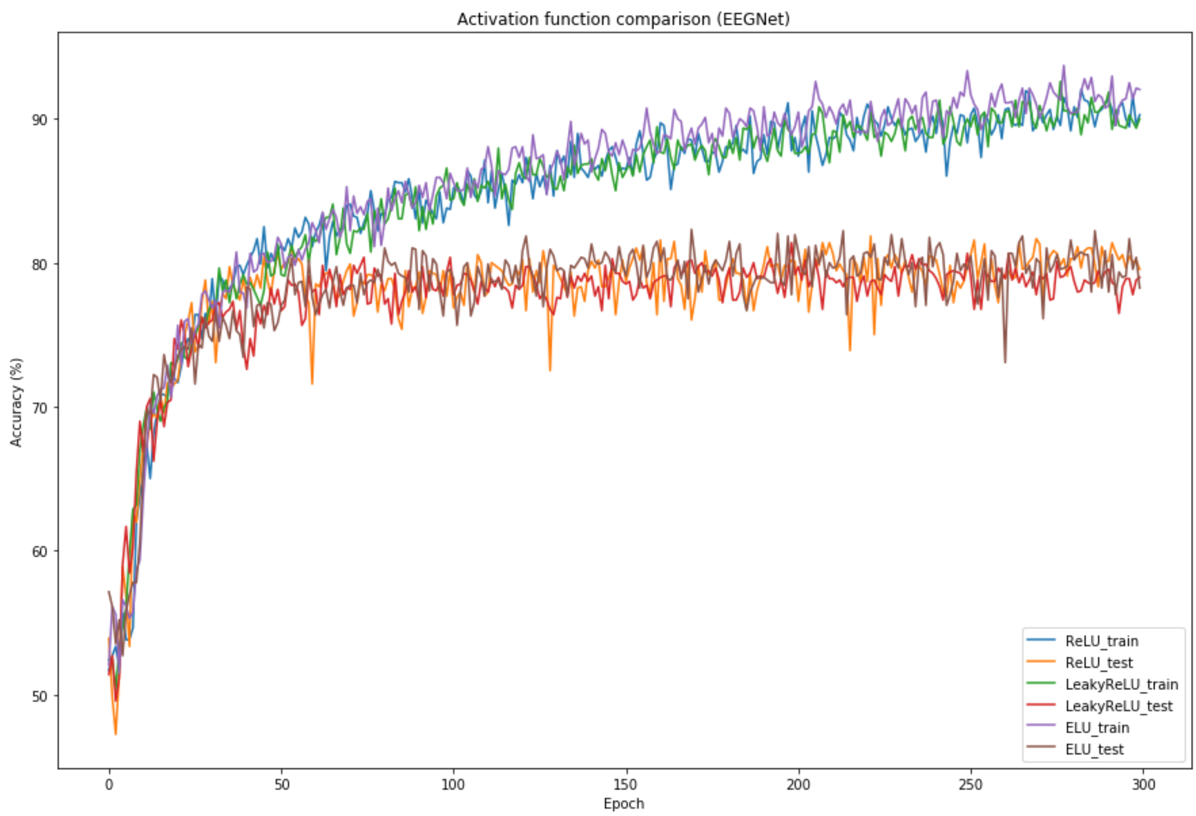
Hyper Parameters

- Batch size: 64
- Learning rate: 1e-02
- Epoch: 300
- Optimizer: Adam
- Loss function: cross entropy

EEGNet



DeepConvNet



Discussion - Dropout

Introduction to Dropout

Dropout is method that ignores neurons randomly at training with probability *p*. By making the traiining process a little noisy, it prevent the network from overfitting.

Result

- Activation function: ReLU
- Batch size: 64
- Learning rate: 1e-02
- Epoch: 300
- Optimizer: Adam
- Loss function: cross entropy

Results

Dropout	Training Accuracy	Testing Accuracy
0	100	82.037
0.2	99.6296	83.2407

Dropout	Training Accuracy	Testing Accuracy
0.4	97.5926	87.2222
0.6	92.7778	85.1851
0.8	82.1296	81.0185

According to the results above, with the dropout value increasing from 0.0 to 0.4, the training accuracy gets worse; however, the testing accuracy gets higher. Furthermore, when the dropout value is too high, the network cannot be trained well. Therefore, we can see that adding a appropriate dropout value can deal with overfitting and make the network stronger.