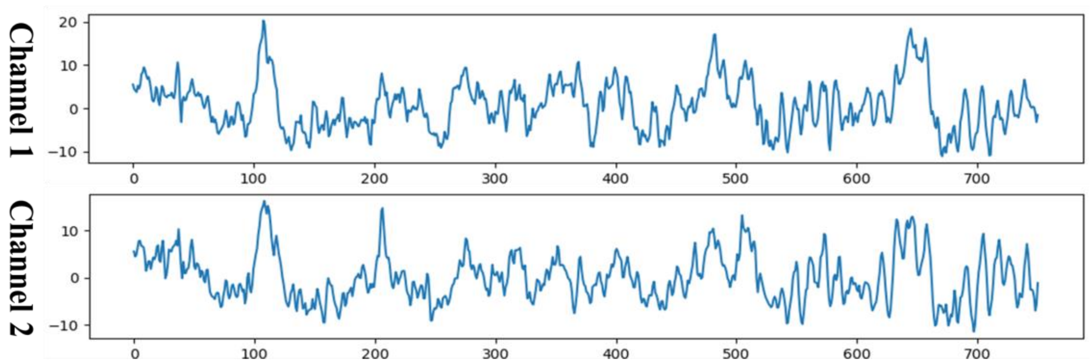
Lab4

1. Introduction (20%)

In the Lab we need to train the dataset of BCI Competition III (like the following figure from slide). And we try to train and test two different mode EEGNet and DeepConvNet with three different activate function ReLU, leaky ReLU, ELU



2. Experiment set up (30%)

A. The detail of your model

Using the structure from the slide. The only different is the activate function could change in the Lab.

EEGNet

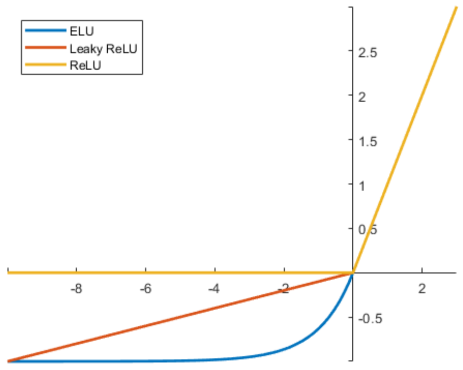
```
#Firstconv
self.conv1 = nn.Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25), bias=False)
self.batchnorm1 = nn.BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

#DepthwiseConv
self.conv2 = nn.Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias=False)
self.batchnorm2 = nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
self.act2 = activate #nn.ELU(alpha=1.0)
self.avgpool2 = nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
self.dropout2 = nn.Dropout(p=0.25)

#SperableConv
self.conv3 = nn.Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False)
self.batchnorm3 = nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
self.act3 = activate #nn.ELU(alpha=1.0)
self.avgpool3 = nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
self.dropout3 = nn.Dropout(p=0.25)

#Classify
self.linear4 = nn.Linear(in_features=736, out_features=2, bias=True)
```

DeepConvNet

```
self.conv1 = nn.Conv2d(1, 25, kernel_size=(1, 5))
self.conv2 = nn.Conv2d(25, 25, kernel_size=(self.C, 1))
self.batchnorm3 = nn.BatchNorm2d(25, eps=1e-05, momentum=0.1)
self.elu4 = activate
self.maxpool5 = nn.MaxPool2d(kernel_size=(1, 2))
self.dropout6 = nn.Dropout(p=0.5)
self.conv7 = nn.Conv2d(25, 50, kernel_size=(1, 5))
self.batchnorm8 = nn.BatchNorm2d(50, eps=1e-05, momentum=0.1)
self.elu9 = activate
self.maxpool10 = nn.MaxPool2d(kernel_size=(1, 2))
self.dropout11 = nn.Dropout(p=0.5)
self.conv12 = nn.Conv2d(50, 100, kernel_size=(1, 5))
self.batchnorm13 = nn.BatchNorm2d(100, eps=1e-05, momentum=0.1)
self.elu14 = activate
self.maxpool15 = nn.MaxPool2d(kernel_size=(1, 2))
self.dropout16 = nn.Dropout(p=0.5)
self.conv17 = nn.Conv2d(100, 200, kernel_size=(1, 5))
self.batchnorm18 = nn.BatchNorm2d(200, eps=1e-05, momentum=0.1)
self.elu19 = activate
self.maxpool20 = nn.MaxPool2d(kernel_size=(1, 2))
self.dropout21 = nn.Dropout(p=0.5)
self.linear22 = nn.Linear(in_features=8600, out_features=2)
```

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

This part is also build-in function in pytorch. All of them provide the non-linear part of the model



From: https://www.researchgate.net/figure/Illustration-of-output-of-ELU-vs-ReLU-vs-Leaky-ReLU-function-with-varying-input-values_fig8_334389306

ReLU,

When the input > 0 the output equal to input

Otherwise, output 0.

Leaky ReLU,

Similar to ReLU, but the slope isn’t 0 and not equal between positive part and negative part. (positive part > negative part)

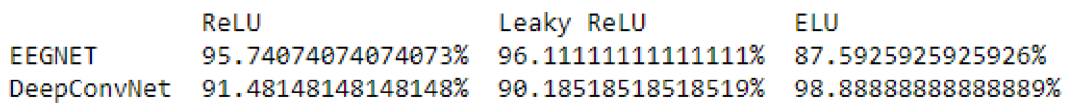
ELU

Similar to ReLU, but the slope isn’t 0 in negative part. But there is log function in negative part.

3. Experimental results (30%)

A. The highest testing accuracy

Screenshot with two models



```
              ReLU                Leaky ReLU          ELU
EEGNET        95.74074074074073%  96.11111111111111%  87.5925925925926%
DeepConvNet   91.48148148148148%  90.18518518518519%  98.88888888888889%
```

anything you want to present

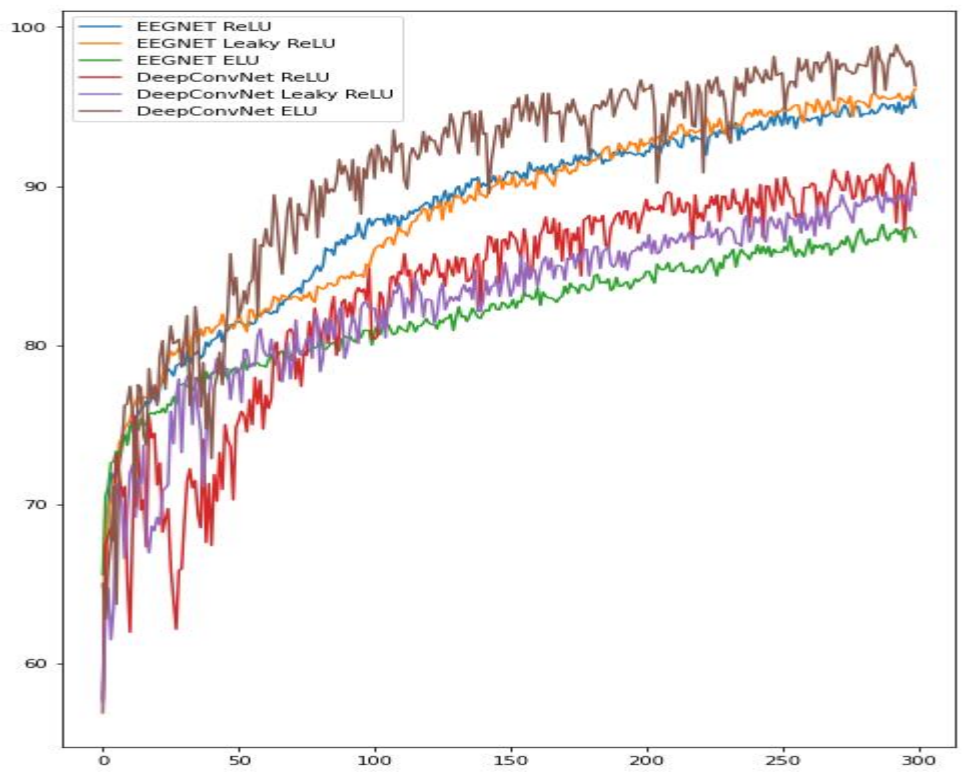
my optimizer using SGD(stochastic gradient decent):

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

And the momentum can make the function not learning so hard when the side change. If we remove the part it should only around 80% accuracy.

B. Comparison figures

EEGNet / DeepConvNet (only test part of dataset)



We can find the DeepConvNet with ELU show best performance.

4. Discussion (20%)

A. Anything you want to share

Python should copy the model when the function call that as the parameter, so I think my work copy the model too much time to slow down the whole work (using Nvidia 3090 but cost 10+ min), but I have no idea to write good Python code style without too much duplicate code. Also like the view() and flatten() do similar work, there are still many function I don't know in python's toolkit.

Without knowing about how the loader work make me confuse to use, but now a know that the original data is only 3-dim and via those function it expand to 4-dim and use transpose() to reorder the dataset and the last output is the [B 1 2 750]. And to use the data, we need to transfer to pytorch's tensor format (from numpy) with function in torch.utils.data.