In [3]:

```
import torch
import torch.optim as optim
from torch.nn.functional import normalize
from torch import nn as nn
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import os
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split
import time
from torchvision import datasets, transforms
import torch.nn.functional as F
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
```

Final Project

In [31]:

```
data_path = r"C:\Users\ccm51\Documents\ECGR_4105\Final_Project\dataset_csv_file.csv"
league_numpy = np.loadtxt(data_path, dtype = np.float32, delimiter = ",", skiprows=1)
print(league_numpy.shape)
```

```
(4028, 11)
```

In [16]:

```
League = torch.from_numpy(league_numpy)
data = League[:, :-1]
torch.reshape(data, (-1, 10, 1, 1))
data, data.shape, data.dtype
```

Out[16]:

```
(tensor([[ 95.,  79.,  31.,  ...,  17.,  27.,   3.],
         [118.,  79.,  65.,  ...,  17.,  47.,  66.],
         [ 68.,  82., 115.,  ...,  65., 100.,  85.],
         ...,
         [103.,  56.,  83.,  ..., 112.,  15.,  74.],
         [ 16.,  28.,  88.,  ..., 148.,   7.,  87.],
         [122.,  63.,  88.,  ...,  83.,  27.,  85.]]),
 torch.Size([4028, 10]),
 torch.float32)
```

In [74]:

```
target = League[:,-1].long()
target = torch.unsqueeze(target, dim=-1)
target, target.shape, target.dtype
```

Out[74]:

```
(tensor([[1],
         [0],
         [0],
         ...,
         [1],
         [0],
         [0]]),
 torch.Size([4028, 1]),
 torch.int64)
```

In [99]:

```
data_mean = torch.mean(data, dim=1).unsqueeze(-1)
data_std = torch.std(data, dim=1).unsqueeze(-1)
data_var = torch.var(data, dim=1).unsqueeze(-1)
data_normalized = (data - data_mean) / torch.sqrt(data_var)
data_normalized = torch.cat((data_normalized, target), dim=1)

train, test = train_test_split(data_normalized, train_size = 0.9, test_size = 0.1)

print(train.shape)
```

```
torch.Size([3625, 11])
```

In [121]:

```
train_loader = torch.utils.data.DataLoader(train, batch_size = 1, shuffle = True)

model = nn.Sequential(nn.Linear(10, 512),
                      nn.Tanh(),
                      nn.Linear(512, 256),
                      nn.Tanh(),
                      nn.Linear(256, 64),
                      nn.Tanh(),
                      nn.Linear(64, 2),
                      nn.LogSoftmax(dim=1))

loss_fn = nn.NLLLoss()
learning_rate = 1e-2
optimizer = optim.SGD(model.parameters(), lr = learning_rate)
n_epochs = 500
```

In [122]:

```
main_tic = time.perf_counter()
for epoch in range(1, n_epochs + 1):
    tic = time.perf_counter()
    loss_train = 0
    for imgs in train_loader:
        labels = imgs[:,-1].long()
        imgs = imgs[:,:-1]
        batch_size = imgs.shape[0]
        outputs = model(imgs.view(batch_size, -1))
        loss = loss_fn(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_train += loss.item()

    toc = time.perf_counter()
    print(f"Epoch {epoch}, Duration = {round(toc - tic, 3)} seconds, Loss: {round(loss_train / len(train_loader), 5)}")

main_toc = time.perf_counter()
print(f"Total Training Time = {round(main_toc - main_tic, 3)} seconds")
print(f"Average Training Time per Epoch = {round((main_toc - main_tic) / n_epochs , 3)} seconds")
```

```
Epoch 484, Duration = 3.975 seconds, Loss: 0.24409
Epoch 485, Duration = 4.121 seconds, Loss: 0.24311
Epoch 486, Duration = 4.229 seconds, Loss: 0.24251
Epoch 487, Duration = 3.909 seconds, Loss: 0.24329
Epoch 488, Duration = 3.96 seconds, Loss: 0.24284

Epoch 489, Duration = 4.348 seconds, Loss: 0.24206
Epoch 490, Duration = 4.205 seconds, Loss: 0.24256
Epoch 491, Duration = 3.965 seconds, Loss: 0.24239
Epoch 492, Duration = 4.65 seconds, Loss: 0.24258
Epoch 493, Duration = 4.435 seconds, Loss: 0.24158
Epoch 494, Duration = 4.784 seconds, Loss: 0.24179
Epoch 495, Duration = 4.25 seconds, Loss: 0.24228
Epoch 496, Duration = 4.688 seconds, Loss: 0.24161
Epoch 497, Duration = 4.671 seconds, Loss: 0.24065
Epoch 498, Duration = 4.131 seconds, Loss: 0.24127
Epoch 499, Duration = 4.192 seconds, Loss: 0.24105
Epoch 500, Duration = 5.14 seconds, Loss: 0.24129
Total Training Time = 2310.457 seconds
Average Training Time per Epoch = 4.621 seconds
```

In [123]:

```
val_loader = torch.utils.data.DataLoader(test, batch_size = 100, shuffle = False)

correct = 0
total = 0

val_tic = time.perf_counter()
with torch.no_grad():
    for imgs in val_loader:
        labels = imgs[:,-1].long()
        imgs = imgs[:,:-1]
        batch_size = imgs.shape[0]
        outputs = model(imgs.view(batch_size, -1))
        _, predicted = torch.max(outputs, dim = 1)
        total += labels.shape[0]
        correct += int((predicted == labels).sum())

val_toc = time.perf_counter()
print(f"Accuracy: {round(correct/total, 3)}, Duration = {round(val_toc - val_tic, 3)} seconds")
```

```
Accuracy: 0.677, Duration = 0.007 seconds
```

In [124]:

```
eg_vs_100t = [44, 43, 148, 70, 140, 96, 89, 88, 5, 121]
eg_vs_100t = torch.FloatTensor(eg_vs_100t)
outputs = model(eg_vs_100t.view(1, -1))
outputs
```

Out[124]:

```
tensor([[-4.4187, -0.0121]], grad_fn=<LogSoftmaxBackward0>)
```