```
import numpy as np
import matplotlib.pyplot as plt

import torch
from torch.utils.data import DataLoader

from src.utils import CustomDataset
from src.utils import Data
from src.utils import PINNLoss
from src.NN import FNN, LSTM

from sklearn.preprocessing import MinMaxScaler, StandardScaler

torch.manual_seed(101)


### data file
# file = "data/combinedDataRev3Labeled.csv"
# file = "data/combinedDataRev4SlimLabeled.csv"
# file = "data/combinedDataRev4LabeledSynthetic.csv" # file used for presentation results
# file = "data/combinedDataFirstLabeled.csv"

# fault conditions
# file = "data/combinedSimpleFaultLabeled.csv" # all fault data
# file = "data/combinedSimpleNormalLabeled.csv" # no fault data
# file = "data/combinedSimpleFaultComboLabeled.csv" # combined fault and no fault data (fake)
# file = "data/completeDataFaultSimpleV3Labeled.csv" # combined fault and no fault data (real)

# paper files
# file = "data/PAPER_normal.csv"
file = "data/PAPER_fault.csv"
# file = "data/PAPER_bypass.csv"

### create dataframe
DataObj = Data(file)
df = DataObj.ImportData()

### define input (X) and output (y) data
Vc_colnames = ["vc1","vc2","vc3","vc4","vc5","vc6","vc7","vc8"]
X_colnames = ["g1upper","g2upper","g3upper",
"g4upper","g5upper","g6upper","g7upper","g8upper","i1","i2"]
y_colnames = ["vout"]
X = np.array(df[X_colnames])#[5000:15000,:]
y = np.array(df[y_colnames])#[5000:15000,:]

Vc_all = np.array(df[Vc_colnames])#[5000:15000,:]

### split data into training and testing datasets
frac = 0.8
X_train, y_train, X_test, y_test = DataObj.TrainTestSplit(X, y, frac=frac)
X_train = X_train.reshape(-1, 1, 10)
y_train = y_train.reshape(-1, 1, 1)
X_test = X_test.reshape(-1, 1, 10)
y_test = y_test.reshape(-1, 1, 1)

### use the custom Pytorch Dataset generator to get train and test data
train_data = CustomDataset(X_train, y_train)
test_data = CustomDataset(X_test, y_test)

### use the generated train/test data and batch size to create DataLoader objects for the train
and test (validataion) sets
batch_size = 200
trainloader = DataLoader(train_data, batch_size=batch_size, shuffle=False)
valloader = DataLoader(test_data, batch_size=batch_size, shuffle=False)

# ### neural network parameters
in_dim = 10
hidden_dim = 201
out_dim = 9

### define the neural network
model = LSTM(in_dim, out_dim, hidden_dim)
```

```
### training parameters
num_epochs = 300
learn_rate = 0.008125855587066246

### Define the loss functions
# weight on nn output to true output loss
gamma1 = 1.513575511635682
# weight on nn state output to dynamic state output loss
gamma2 = 0.8440506511631696
# weight on nn output to dynamic output loss
gamma3 = 0.0005791257706961797
physics_loss_func = PINNLoss(gamma2, gamma3)
loss_func = torch.nn.MSELoss()

### define the optimizer
optimizer = torch.optim.Adam(model.parameters())

idx = np.arange(0,len(X_train),batch_size) # vector for manual batch training

### TRAIN THE PINN
loss_all = []
loss_NN = []
loss_physics = []
for j in range(num_epochs):
    l_tot = 0
    l1_tot = 0
    l2_tot = 0
    l3_tot = 0
    # for X_train, y_train in trainloader:
    for i in range(len(idx)-1):

        # initialize the capacitor voltages
        Vc_k_est = torch.tensor(Vc_all[idx[i]:idx[i+1], :],
dtype=torch.float32).reshape(batch_size, 1, 8)

        # define the inputs and outputs for the current batch
        X_train_torch = torch.tensor(X_train[idx[i]:idx[i+1], :, :], dtype=torch.float32) / 10
        y_train_torch = torch.tensor(y_train[idx[i]:idx[i+1], :, :], dtype=torch.float32) / 1000

        # zero out the gradient of the optimizer
        optimizer.zero_grad()

        # predict Vth and Vc using the LSTM
        y_pred = model(X_train_torch)

        # calculate the mse loss
        mse_loss = gamma1*loss_func(y_train_torch, y_pred[:, :, -1].reshape(-1, 1, 1))

        # calculate the physics loss
        Vc_k_est, physics_loss, l2, l3 = physics_loss_func(y_pred, y_train_torch,
X_train_torch*10, Vc_k_est)

        # total loss
        loss = mse_loss + physics_loss

        # backpropogate through the augmented loss function
        loss.backward()

        # advance the optimizer
        optimizer.step()

        # track the loss for monitoring and plotting
        l_tot = l_tot + loss.detach()
        l1_tot = l1_tot + mse_loss.detach()
        l2_tot = l2_tot + l2.detach()
        l3_tot = l3_tot + l3.detach()

    print(f"Epoch {j} loss: {l_tot} ({l1_tot}, {l2_tot}, {l3_tot})")
    loss_all.append(l_tot)
    loss_NN.append(l1_tot)
    loss_physics.append(l2_tot + l3_tot)

### PLOT THE LOSS
```

```
fig, ax = plt.subplots(3,1)

ax[0].plot(loss_all)
ax[0].set_ylabel("Total loss")
ax[0].set_xticklabels([])

ax[1].plot(loss_NN)
ax[1].set_ylabel("NN MSE loss")
ax[1].set_xticklabels([])

ax[2].plot(loss_physics)
ax[2].set_ylabel("Physics loss")
ax[2].set_xlabel("Epochs")

plt.tight_layout()
plt.show()



### PLOT THE PERFORMANCE OF THE PINN ON TRAINING DATA
### use the trained PINN to estimate Vth and Vc using trainng inputs
Vc_hat_train = []
Vth_hat_train = []
Vth_true_train = []
for i in range(len(idx)-1):

    # define the inputs
    X_train_torch = torch.tensor(X_train[idx[i]:idx[i+1], :, :], dtype=torch.float32) / 10

    # define the true outputs
    y_train_torch = torch.tensor(y_train[idx[i]:idx[i+1], :, :], dtype=torch.float32)

    # define the true states
    Vc_k_est = torch.tensor(Vc_all[idx[i]:idx[i+1], :], dtype=torch.float32).reshape(batch_size,
8)

    # predict Vth and Vci using the trainged PINN
    y_pred_train = model(X_train_torch).detach().numpy()*1000

    # save the results for plotting
    Vc_hat_train.append(y_pred_train[:,:,:-1].reshape(-1,8))
    Vth_hat_train.append(y_pred_train[:,:,-1].flatten())
    Vth_true_train.append(y_train_torch.detach().numpy().flatten())

### put results into correct format for plotting
Vc_hat_train = np.array(Vc_hat_train).reshape(-1,8)
Vc_true_train = np.array(df[Vc_colnames])[:int(len(X)*frac),:]
Vth_hat_train = np.array(Vth_hat_train).flatten()
Vth_true_train =np.array(Vth_true_train).flatten()

### plot the results
fig, ax = plt.subplots(3,1,figsize=(10,8))
for i in range(8):
    # if i == 3 or i == 5:
    #     ax[0].plot(Vc_hat_train[:,i], label=f"$Vc_{i}$")
    #     ax[0].plot(Vc_true_train[:,i], 'k--')
    # elif i == 1 or i == 7:
    #     ax[2].plot(Vc_hat_train[:,i], label=f"$Vc_{i}$")
    #     ax[2].plot(Vc_true_train[:,i], 'k--')
    # else:
    #     ax[1].plot(Vc_hat_train[:,i], label=f"$Vc_{i}$")
    #     ax[1].plot(Vc_true_train[:,i], 'k--')

    if i < 4:
        ax[0].plot(Vc_hat_train[:,i], label=f"$Vc_{i}$")
        ax[0].plot(Vc_true_train[:,i], 'k--')
    else:
        ax[1].plot(Vc_hat_train[:,i], label=f"$Vc_{i}$")
        ax[1].plot(Vc_true_train[:,i], 'k--')

x_vec = np.arange(10000, 14000, 1)
ax[0].fill_between(x_vec, np.ones(len(x_vec))*np.min(Vc_true_train),
np.ones(len(x_vec))*np.max(Vc_true_train),alpha=0.5,label="fault")
```

```
ax[0].set_title("Upper Arm", fontsize=18)
ax[0].set_ylabel("Cap Volt [V]", fontsize=14)
ax[0].set_xticklabels([])
ax[0].legend(loc='upper left', fontsize=14)

ax[1].fill_between(x_vec, np.ones(len(x_vec))*np.min(Vc_true_train),
np.ones(len(x_vec))*np.max(Vc_true_train),alpha=0.5,label="fault")
ax[1].set_title("Lower Arm", fontsize=18)
ax[1].set_xticklabels([])
ax[1].set_ylabel("Cap Volt [V]", fontsize=14)
ax[1].legend(loc='upper left', fontsize=14)

ax[2].fill_between(x_vec, np.ones(len(x_vec))*np.min(Vth_true_train),
np.ones(len(x_vec))*np.max(Vth_true_train),alpha=0.5,label="fault")
ax[2].plot(Vth_true_train, label="True")
ax[2].plot(Vth_hat_train, label="Est")
ax[2].set_ylabel("Thevenin Volt [V]", fontsize=14)
ax[2].set_xlabel("Time [sec]", fontsize=14)
xticks = ax[2].get_xticks()
ax[2].set_xticklabels(['{:.2f}'.format(val*1e-5) for val in xticks])
ax[2].legend(loc='upper left', fontsize=14)

plt.tight_layout()
plt.show()

### calculate the mean squared error between estimated and true data
Vc_mse = np.mean(np.sum((Vc_true_train[:Vc_hat_train.shape[0],:] - Vc_hat_train)**2, axis=1))
Vth_mse = np.mean((Vth_true_train - Vth_hat_train)**2)
print("Training data error:")
print("     Vc:", Vc_mse)
print("     Vth:", Vth_mse)
print("     Total:", Vc_mse+Vth_mse)



### PLOT THE PERFORMANCE OF THE PINN ON TRAINING DATA
### use the trained PINN to estimate Vth and Vc using trainng inputs
idx = np.linspace(0,len(X_test)-1,int(len(X_test)/batch_size)+1).astype(int)
Vc_hat = np.array([[],[],[],[],[],[],[],[]]).T
Vth_hat = np.array([])
Vth_true = np.array([])
for i in range(len(idx)-1):

    # define the inputs
    X_test_torch = torch.tensor(X_test[idx[i]:idx[i+1], :, :], dtype=torch.float32) / 10

    # define the true outputs
    y_test_torch = torch.tensor(y_test[idx[i]:idx[i+1], :, :], dtype=torch.float32)

    # predict Vth and Vci using the trainged PINN
    y_pred_test = model(X_test_torch).detach().numpy()*1000

    # save the results for plotting
    Vc_hat = np.vstack((Vc_hat, y_pred_test[:,:,:-1].reshape(-1,8)))
    Vth_hat = np.concatenate((Vth_hat, y_pred_test[:,:,-1].flatten()))
    Vth_true = np.concatenate((Vth_true, y_test_torch.detach().numpy().flatten()))

### put results into correct format for plotting
Vc_hat = np.array(Vc_hat).reshape(-1,8)
Vc_true = np.array(df[Vc_colnames])[-int(len(X)*(1-frac)):,:]
Vth_hat = np.array(Vth_hat).flatten()
Vth_true =np.array(Vth_true).flatten()

### plot the results
fig, ax = plt.subplots(3,1,figsize=(10,8))
for i in range(8):
    if i < 4:
        ax[0].plot(Vc_hat[:,i], label=f"$Vc_{i}$")
        ax[0].plot(Vc_true[:,i], 'k--')
    else:
        ax[1].plot(Vc_hat[:,i], label=f"$Vc_{i}$")
        ax[1].plot(Vc_true[:,i], 'k--')
```

```
ax[0].set_title("Upper Arm")
ax[0].set_xticklabels([])
ax[0].set_ylabel("Cap Volt [V]")
ax[0].legend()

ax[1].set_title("Lower Arm")
ax[1].set_xticklabels([])
ax[1].set_ylabel("Cap Volt [V]")
ax[1].legend()

ax[2].plot(Vth_true, label="True")
ax[2].plot(Vth_hat, label="Est")
ax[2].set_ylabel("Thevenin Volt [V]")
ax[2].set_xlabel("Time [sec]")
xticks = ax[2].get_xticks()
ax[2].set_xticklabels(['{:.2f}'.format(val*1e-5) for val in xticks])
ax[2].legend()
plt.show()

### calculate the mean squared error between estimated and true data
Vc_mse = np.mean(np.sum((Vc_true - Vc_hat)**2, axis=1))
Vth_mse = np.mean((Vth_true - Vth_hat)**2)
print("Training data error:")
print("     Vc:", Vc_mse)
print("     Vth:", Vth_mse)
print("     Total:", Vc_mse+Vth_mse)
```