In [ ]:
```
import torch
import numpy as np
from torch.autograd import Variable
import time


#Call model of layers and its forward step
from Forward_with_Layer_Setting import Net

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

#Call training functions of Loss functions
from NSpde_loss import lossNSpde
from BoundaryLoss import lossBdry
from InitialConditionLoss import lossIC



def create_network(IC_Only_Train):

    net = Net()
    net = net.to(device)

    #Set final times for running training
    time_slices = np.array([.1,.2, .3, .4, .5, .6, .7, .8, .9, 1]) #, .25, .5, 1

    #Load Training Points
    x_domain, y_domain, t_zero, x_Bdry, y_Bdry, x_l_Bdry, x_u_Bdry, y_l_Bdry, y_u_Bdry = twoDimTrainPts(net, Domain

    start = time.time()

    #Start Training only on IC
    if IC_Only_Train == True:
        print('Training Only on the Initial Condition')
        Create_IC_Parameters(x_domain, y_domain, t_zero, 60000, 10**-3, 'IC_Only.pt', record_loss = 100, print_loss
        IC_Done = time.time()
        print('IC Time:\t', IC_Done-start)
        '''
        print('Training Only on the NSpde Condition')
```

```
    NSpde_Only_training(net, x_domain, y_domain, t_zero, x_Bdry, y_Bdry, x_l_Bdry, x_u_Bdry,
                  y_l_Bdry, y_u_Bdry, time_slices, 20000, 10**-3, record_loss = 100, print_loss = 1000)
    torch.save(net.state_dict(), f"NSpde_afterIC_.pt")
    NSpde_Done = time.time()
    print('MvBdry Time:\t', NSpde_Done-start)
    '''
    return 0

time_vec = [0, 0, 0, 0]


#attempt to load IC if it exists
try:
    net.load_state_dict(torch.load("IC_Only.pt"))
except:
    pass
'''
#attempt to load MvBdry if it exists
try:
    net.load_state_dict(torch.load("NSpde_Only.pt"))
except:
    pass
'''
global epsilon #used to track loss
epsilon = []

print('Training PDE')

for i in range(4):
    #Set loop to optimize in progressively smaller learning rates
    if i == 0:
        #First loop uses progressively increasing time intervals
        print('Executing Pass 1')
        iterations = 30000
        learning_rate = 10**-2
    elif i == 1:
        print('Executing Pass 2')
        #time_slices = time_slices[-1]
        iterations = 30000
        learning_rate = 10**-3
    elif i == 2:
```

```
            print('Executing Pass 3')
            iterations = 2 #0000
            learning_rate = 5*10**-6
        elif i ==3:
            print('Executing Pass 4')
            iterations = 2 #0000
            learning_rate = 10**-6

        training_loop(net, x_domain, y_domain, t_zero, x_Bdry, y_Bdry, x_l_Bdry, x_u_Bdry,
                      y_l_Bdry, y_u_Bdry, time_slices, iterations, learning_rate, IC_coefficient = 1, record_loss =
        torch.save(net.state_dict(), f"NNlayers_Bubble_{i}.pt")
        np.savetxt('epsilon.txt', epsilon)
        time_vec[i] = time.time()

    np.savetxt('epsilon.txt', epsilon)

    end = time.time()

    print("Total Time:\t", end-start, '\nPass 1 Time:\t', time_vec[0]-start, '\nPass 2 Time:\t', time_vec[1]-start,


def twoDimTrainPts(net, Domain_collocation, Bdry_collocation):
    #Set of all the recorded xy variables as base data for chasing during training

    # Domain boundary in the range [0, 1]x[0, 2] and time in [0, 1].
    x_l = net.x1_l
    x_u = net.x1_u
    y_l = net.x2_l
    y_u = net.x2_u

    #time starts at lower bound 0, ends at upper bouund updated in slices
    t_l = 0

    #Pick IC/NSpde Condition Training Random Points in Numpy
    x_domain = np.random.uniform(low= x_l, high=x_u, size=(Domain_collocation, 1))
    y_domain = np.random.uniform(low= y_l, high=y_u, size=(Domain_collocation, 1))

    #Move to pytorch tensors
    x_domain = Variable(torch.from_numpy(x_domain).float(), requires_grad=True).to(device)
    y_domain = Variable(torch.from_numpy(y_domain).float(), requires_grad=True).to(device)
```

```
    #Pick IC Training t starting points to make tensor
    t_zero = Variable(torch.zeros_like(x_domain), requires_grad=True).to(device)

    #Pick BC Training Random Points in Numpy
    x_Bdry = np.random.uniform(low=x_l, high=x_u, size=(Bdry_collocation,1))
    y_Bdry = np.random.uniform(low=y_l, high=y_u, size=(Bdry_collocation,1))

    #Move to pytorch tensors
    x_Bdry= Variable(torch.from_numpy(x_Bdry).float(), requires_grad=True).to(device)
    y_Bdry = Variable(torch.from_numpy(y_Bdry).float(), requires_grad=True).to(device)

    ##Pick pts to make tensor for No-Slip Boundary Condition
    x_l_Bdry = Variable(x_l * torch.ones_like(x_Bdry), requires_grad=True).to(device)
    x_u_Bdry = Variable(x_u * torch.ones_like(x_Bdry), requires_grad=True).to(device)
    y_l_Bdry = Variable(y_l * torch.ones_like(x_Bdry), requires_grad=True).to(device)
    y_u_Bdry = Variable(y_u * torch.ones_like(x_Bdry), requires_grad=True).to(device)


    return x_domain, y_domain, t_zero, x_Bdry, y_Bdry, x_l_Bdry, x_u_Bdry, y_l_Bdry, y_u_Bdry

def tsliceTrainPts(net, Domain_collocation, Bdry_collocation, final_time):
    #Set of all the recorded t variable as base data for chasing during training

    #time starts at lower bound 0, ends at upper bouund updated in slices
    t_l = net.t_l

    #Pick IC/NSpde Condition Training Random Points in Numpy
    t_domain = np.random.uniform(low=t_l, high=final_time, size=(Domain_collocation, 1))

    #Move to pytorch tensors
    t_domain = Variable(torch.from_numpy(t_domain).float(), requires_grad=True).to(device)

    #Pick IC Training t starting points to make tensor
    t_zero = Variable(torch.zeros_like(t_domain), requires_grad=True).to(device)

    #Pick BC Training Random Points in Numpy
    t_Bdry = np.random.uniform(low=t_l, high=final_time, size=(Bdry_collocation,1))

    #Move to pytorch tensors
    t_Bdry = Variable(torch.from_numpy(t_Bdry).float(), requires_grad=True).to(device)
```

```
    return t_domain, t_Bdry

def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']

def Create_IC_Parameters(x_domain, y_domain, t_zero, iterations, learning_rate, filename, record_loss, print_loss):
    ICnet = Net().to(device)

    IC_Only_training(ICnet, x_domain, y_domain, t_zero, iterations, learning_rate, record_loss, print_loss)

    torch.save(ICnet.state_dict(), filename)


def IC_Only_training(net, x_domain, y_domain, t_zero, iterations, learning_rate, record_loss, print_loss):
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    #learning rate update
    for g in net.optimizer.param_groups:
        g['lr'] = learning_rate

    #training loop
    epsilon_IC = [] #placeholder to track decreasing loss
    for epoch in range(1, iterations+1):


        # Resetting gradients to zero
        net.optimizer.zero_grad()

        #Loss based on Initial Condition
        loss = lossIC(net, x_domain, y_domain, t_zero)

        loss.backward()

        # Gradient Norm Clipping
        torch.nn.utils.clip_grad_norm_(net.parameters(), max_norm= 5*10**2, norm_type=2, error_if_nonfinite=True)

        #Gradient Value Clipping
        #nn.utils.clip_grad_value_(net.parameters(), clip_value=1.0)

        net.optimizer.step()
```

```
        #Print Loss every 1000 Epochs
        with torch.autograd.no_grad():

            if epoch%record_loss == 0:
                epsilon_IC = np.append(epsilon_IC, loss.cpu().detach().numpy())
            if epoch%print_loss == 0:
                print("Iteration:", epoch, "Initial Condition Loss:", loss.data)

    np.savetxt('epsilon_IC.txt', epsilon_IC)


def training_loop(net, x_domain, y_domain, t_zero, x_Bdry, y_Bdry, x_l_Bdry, x_u_Bdry, y_l_Bdry, y_u_Bdry, time_sli
    global epsilon
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    #learning rate update
    for g in net.optimizer.param_groups:
        g['lr'] = learning_rate

    for final_time in time_slices:

        with torch.autograd.no_grad():
            print("Current Final Time:", final_time, "Current Learning Rate: ", get_lr(net.optimizer))

        indicator = False
        reset_regularization = 1000

        #Iterate over these points

        t_domain, t_Bdry = tsliceTrainPts(net, Domain_collocation = int(1000), Bdry_collocation = int(100), final_t
        for epoch in range(1, iterations+1):
            # Loss calculation based on partial differential equation (PDE)

            if epoch%reset_regularization == 0:
                indicator = False

            if epoch%reset_regularization != 0: #To detect error on forward/Backward, add hashtag on this whole lin
            #with torch.autograd.detect_anomaly(): #use this line alternatively by deleting hashtag.

                ###Training steps
```

```
        # Resetting gradients to zero
        net.optimizer.zero_grad()

        #Loss based on Initial Condition
        mse_IC = lossIC(net, x_domain, y_domain, t_zero)
        # Gradient Norm Clipping
        torch.nn.utils.clip_grad_norm_(net.parameters(), max_norm= 5*10**2, norm_type=1, error_if_nonfinite

        #Loss based on Boundary Condition (Containing No-Slip and Free-slip)
        mse_BC = lossBdry(net, x_Bdry, y_Bdry, t_Bdry, x_l_Bdry, x_u_Bdry, y_l_Bdry, y_u_Bdry)
        # Gradient Norm Clipping
        torch.nn.utils.clip_grad_norm_(net.parameters(), max_norm= 5*10**2, norm_type=1, error_if_nonfinite

        #Loss based on PDE
        mse_NS = lossNSpde(net, x_domain, y_domain, t_domain)
        # Gradient Norm Clipping
        torch.nn.utils.clip_grad_norm_(net.parameters(), max_norm= 5*10**2, norm_type=1, error_if_nonfinite


        if indicator == False:
            indicator = True
            IC_regular = mse_IC.detach()
            BC_regular = mse_BC.detach()
            pde_regular = mse_NS.detach()

        raw_loss = IC_coefficient * mse_IC + mse_BC + mse_NS

        mse_IC = mse_IC #/IC_regular
        mse_BC = mse_BC #/BC_regular
        mse_NS = mse_NS #/pde_regular

        #Combine all Loss functions
        loss = mse_BC + 10**5 *mse_IC  + 10**5 * mse_NS #IC_coefficient *
        # Gradient Norm Clipping
        torch.nn.utils.clip_grad_norm_(net.parameters(), max_norm= 5*10**2, norm_type=1, error_if_nonfinite

        loss.backward()
    # Gradient Norm Clipping
    torch.nn.utils.clip_grad_norm_(net.parameters(), max_norm= 5*10**2, norm_type=1, error_if_nonfinite=Fal

    #Gradient Value Clipping
```

```
            #nn.utils.clip_grad_value_(net.parameters(), clip_value=1.0)
            net.optimizer.step()

            #Print Loss every 1000 Epochs
            with torch.autograd.no_grad():
                if epoch%record_loss == 0:
                    epsilon = np.append(epsilon, raw_loss.cpu().detach().numpy())
                if epoch%print_loss == 0:
                    print("Iteration:", epoch, "\tTotal Loss:", loss.data)
                    print("IC Loss: ", mse_IC.data, "\tBC Loss: ", mse_BC.data, "\tNS PDE Loss: ", mse_NS.data)



create_network(True)
create_network(False)
```

```
Training Only on the Initial Condition
Iteration: 1000 Initial Condition Loss: tensor(0.0472, device='cuda:0')
Iteration: 2000 Initial Condition Loss: tensor(0.0214, device='cuda:0')
Iteration: 3000 Initial Condition Loss: tensor(0.0139, device='cuda:0')
Iteration: 4000 Initial Condition Loss: tensor(0.0123, device='cuda:0')
Iteration: 5000 Initial Condition Loss: tensor(0.0113, device='cuda:0')
Iteration: 6000 Initial Condition Loss: tensor(0.0105, device='cuda:0')
Iteration: 7000 Initial Condition Loss: tensor(0.0098, device='cuda:0')
Iteration: 8000 Initial Condition Loss: tensor(0.0086, device='cuda:0')
Iteration: 9000 Initial Condition Loss: tensor(0.0077, device='cuda:0')
Iteration: 10000 Initial Condition Loss: tensor(0.0072, device='cuda:0')
Iteration: 11000 Initial Condition Loss: tensor(0.0068, device='cuda:0')
Iteration: 12000 Initial Condition Loss: tensor(0.0064, device='cuda:0')
Iteration: 13000 Initial Condition Loss: tensor(0.0060, device='cuda:0')
Iteration: 14000 Initial Condition Loss: tensor(0.0057, device='cuda:0')
Iteration: 15000 Initial Condition Loss: tensor(0.0054, device='cuda:0')
Iteration: 16000 Initial Condition Loss: tensor(0.0051, device='cuda:0')
Iteration: 17000 Initial Condition Loss: tensor(0.0049, device='cuda:0')
Iteration: 18000 Initial Condition Loss: tensor(0.0047, device='cuda:0')
Iteration: 19000 Initial Condition Loss: tensor(0.0045, device='cuda:0')
Iteration: 20000 Initial Condition Loss: tensor(0.0043, device='cuda:0')
Iteration: 21000 Initial Condition Loss: tensor(0.0043, device='cuda:0')
Iteration: 22000 Initial Condition Loss: tensor(0.0042, device='cuda:0')
Iteration: 23000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 24000 Initial Condition Loss: tensor(0.0042, device='cuda:0')
Iteration: 25000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 26000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 27000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 28000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 29000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 30000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 31000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 32000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 33000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 34000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 35000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 36000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 37000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 38000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 39000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 40000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
```

```
Iteration: 41000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 42000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 43000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 44000 Initial Condition Loss: tensor(0.0041, device='cuda:0')
Iteration: 45000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 46000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 47000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 48000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 49000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 50000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 51000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 52000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 53000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 54000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 55000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 56000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 57000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 58000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 59000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
Iteration: 60000 Initial Condition Loss: tensor(0.0040, device='cuda:0')
IC Time:         760.195558309555
Training PDE
Executing Pass 1
Current Final Time: 0.1 Current Learning Rate:  0.01
Iteration: 200  Total Loss: tensor(5.0539e+16, device='cuda:0')
IC Loss:  tensor(1826.3171, device='cuda:0')    BC Loss:  tensor(5874.1938, device='cuda:0')    NS PDE Loss:  tensor
(5.0539e+11, device='cuda:0')
Iteration: 400  Total Loss: tensor(9.8734e+12, device='cuda:0')
IC Loss:  tensor(2528.6506, device='cuda:0')    BC Loss:  tensor(267583.0625, device='cuda:0')  NS PDE Loss:  tensor
(98731312., device='cuda:0')
Iteration: 600  Total Loss: tensor(8.4614e+12, device='cuda:0')
IC Loss:  tensor(3326.4094, device='cuda:0')    BC Loss:  tensor(80664.9219, device='cuda:0')   NS PDE Loss:  tensor
(84610280., device='cuda:0')
Iteration: 800  Total Loss: tensor(5.3073e+12, device='cuda:0')
IC Loss:  tensor(4679.4761, device='cuda:0')    BC Loss:  tensor(13956.8057, device='cuda:0')   NS PDE Loss:  tensor
(53068020., device='cuda:0')
Iteration: 1000         Total Loss: tensor(4.7828e+12, device='cuda:0')
IC Loss:  tensor(4013.5725, device='cuda:0')    BC Loss:  tensor(12148.8633, device='cuda:0')   NS PDE Loss:  tensor
(47824040., device='cuda:0')
```

In [ ]:

In [ ]: