

$lr=0.002$

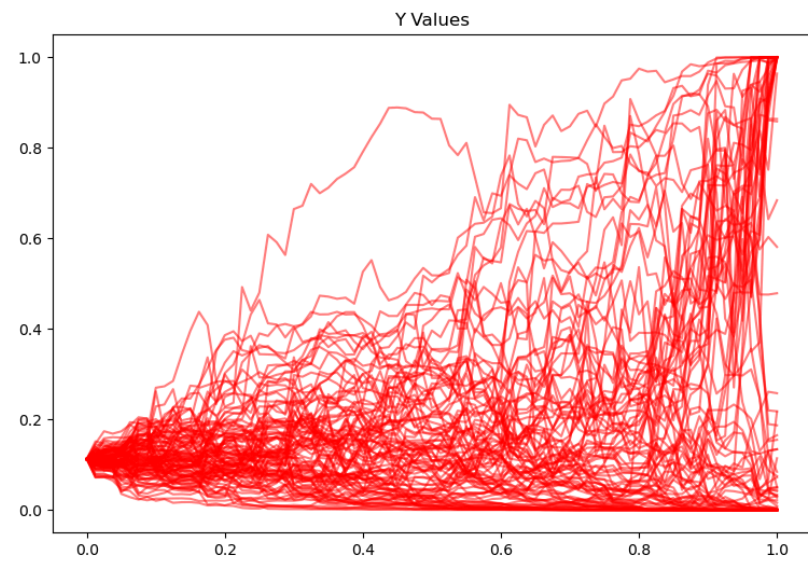
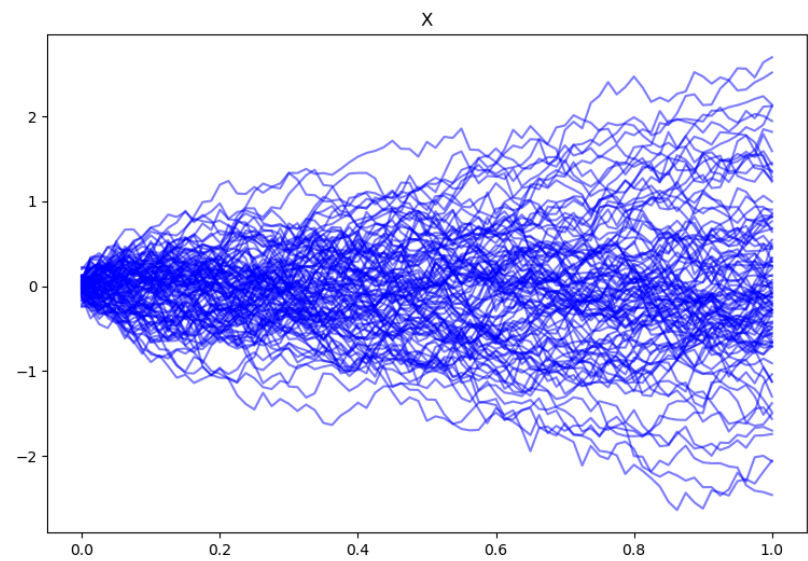
target: $\sigma=0.05$

optimizer: AdamW(weight_decay=5e-4)

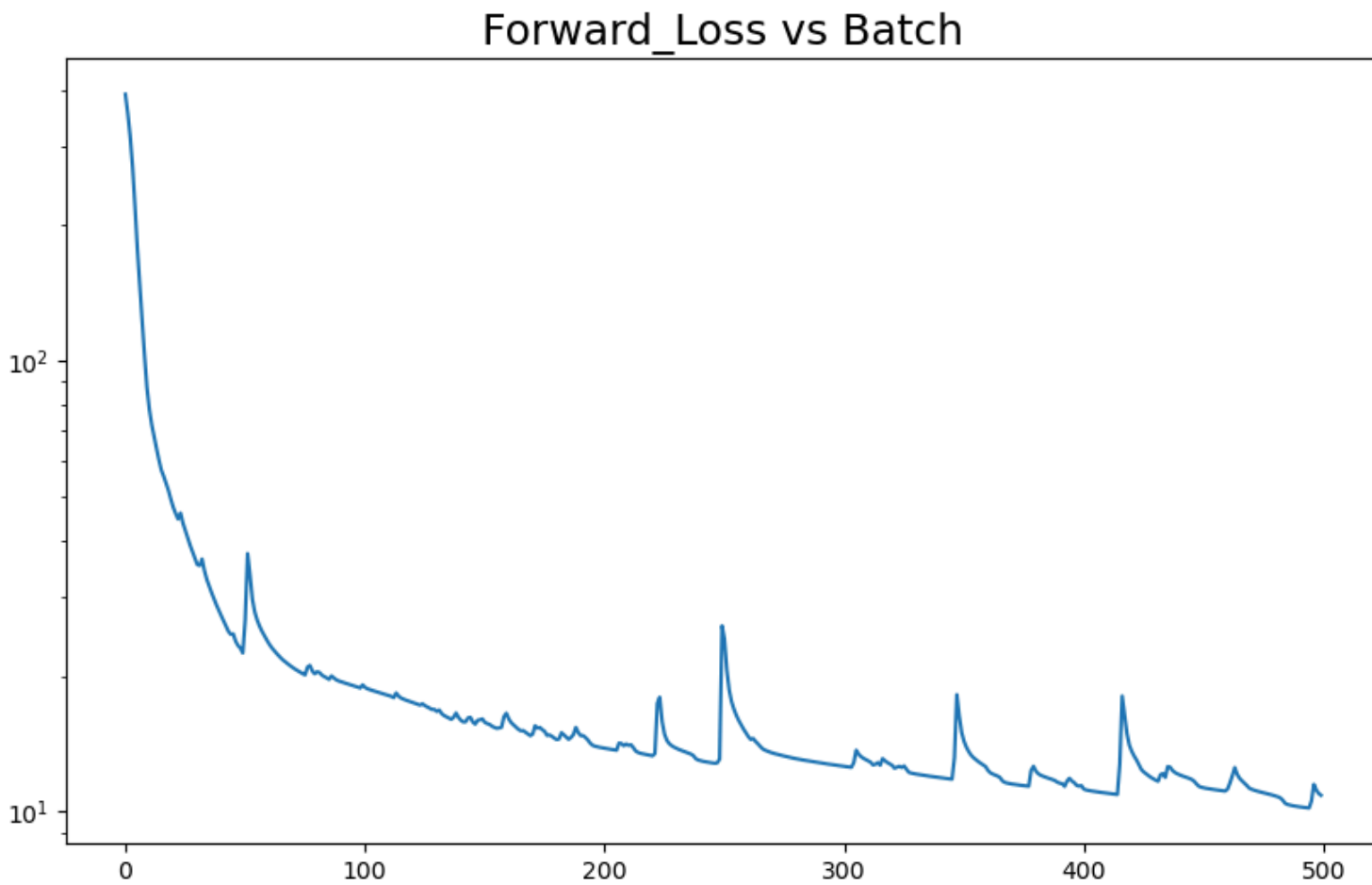
scheduler: StepLR(step=50, gamma=0.995)

MaxBatch=500

OptimStep=20



Average Error Est: 10.207014131546021
Batch Number: 495
Average Error Est: 10.19406180381775
Batch Number: 496
Average Error Est: 10.568049144744872
Batch Number: 497
Average Error Est: 11.500824975967408
Batch Number: 498
Average Error Est: 11.18021764755249
Batch Number: 499
Average Error Est: 10.981325435638428
Batch Number: 500
Average Error Est: 10.881841087341309



```
In [ ]: import numpy as np
import torch as torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import time
import random
from scipy.stats import norm
```

```
In [ ]: #Model and Params
#Numbers
NumTrain=500
NT=80
dt=1/NT
sigma=0.05

#Forward Loss
forward_losses = []
#Network Class for FBSDE
class Network(nn.Module):
    def __init__(self, lr, input_dims, fc1_dims, fc2_dims, n_outputs):
        """
        lr: learning rate
        """
        super(Network, self).__init__()

        #Pass input parameters
        self.input_dims = input_dims
        self.fc1_dims = fc1_dims
        self.fc2_dims = fc2_dims
        self.n_out = n_outputs

        #Construct network
        self.fc1 = nn.Linear(*self.input_dims, self.fc1_dims)
        nn.init.xavier_uniform_(self.fc1.weight)
        self.fc2 = nn.Linear(self.fc1_dims, self.fc2_dims)
        nn.init.xavier_uniform_(self.fc2.weight)
        self.fc3 = nn.Linear(self.fc2_dims, self.n_out)
        nn.init.xavier_uniform_(self.fc3.weight)

        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

```

        self.to(self.device)

    def forward(self, input):
        x = F.relu(self.fc1(input))
        x = F.relu(self.fc2(x))
        output = self.fc3(x)
        return output

## Functions
def Sample_Init(N, mean=0, sd=0.1):
    """
    Generate N samples of x0
    """
    xi = np.random.normal(mean, sd, size=N)

    return torch.FloatTensor(xi).view(-1, 1)

def SampleBMIncr(T, Npaths, Nsteps):
    # Returns Matrix of Dimension Npaths x Nsteps With Sample Increments of of BM
    # Here an increment is of the form dB
    dt = T / Nsteps
    dB = np.sqrt(dt) * np.random.randn(Npaths, Nsteps)
    return torch.FloatTensor(dB)

def target(x, sigma=sigma):
    x = x.detach().numpy()
    return torch.FloatTensor(-x/sigma)

# Forward Loss
def get_foward_loss_coupled(dB, init_x, NT, target, y0_model, z_models):
    x = init_x
    # y = torch.rand_like(x)
    y_tilde = y0_model(x)
    y = torch.sigmoid(y_tilde)
    for j in range(1, NT+1):

        z = z_models[j-1](x)
        x = x + y*dt + dB[:, j].view(-1, 1)
        y_tilde = (y_tilde + (z**2)*(1-2/(1+torch.exp(y_tilde)))/2*dt + z * dB[:, j].view(-1, 1)).clamp(min=-1, max=1)
        y = torch.sigmoid(y_tilde)

    loss = torch.mean((y_tilde - target(x))**2)
    return loss

def get_target_path_coupled(dB, init_x, NumBM, NT, y0_model, z_models):
    x_path = torch.ones(NumBM, NT+1)

```

```

y_path = torch.ones(NumBM,NT+1)
x = init_x
# y = torch.rand_like(x)
y_tilde=y0_model(x)
y=torch.sigmoid(y_tilde)
x_path[:,0] = x.squeeze()
y_path[:,0] = y.squeeze()
for j in range(1, NT+1):
    z = z_models[j-1](x)
    x += y*dt+ dB[:,j].view(-1,1)
    y_tilde = (y_tilde +(z**2)*(1-2/(1+torch.exp(y_tilde)))/2 *dt + z * dB[:,j].view(-1,1))#.clamp(min=-1,max=1)
    y=torch.sigmoid(y_tilde)
    x_path[:,j] = x.squeeze()
    y_path[:,j] = y.squeeze()
return x_path.detach(), y_path.detach()

class plot_results():
    def __init__(self,loss=forward_losses,sigma=sigma,Npaths=100,NumTrain=NumTrain,NT=NT):
        self.loss=loss
        self.x_path,self.y_path=get_target_path_coupled(dB, init_x, y0_model=y0_model_main, z_models=z_models_main, NumB
        self.number_of_paths=np.minimum(Npaths,NumTrain)
        self.sigma=sigma

    def FwdLoss(self,log=True):
        plt.figure(figsize=(10,6))
        plt.title("Forward_Loss vs Batch",fontsize=18)
        plt.plot(self.loss)

        if log==True:
            plt.yscale('log')

    def results(self,seed=0):
        random.seed(seed)
        idx_list = np.random.choice(NumTrain, self.number_of_paths, replace = False)
        x_plot = self.x_path.detach().numpy()[idx_list]
        y_plot = self.y_path.detach().numpy()[idx_list]
        t = np.array([i for i in range(NT+1)]) * 1/(NT)
        plt.figure(figsize=(20,6))
        plt.subplot(121)
        for i in range(self.number_of_paths):
            plt.plot(t,x_plot[i], color="blue", alpha=0.5)
        plt.title("X")

        plt.subplot(122)
        for i in range(self.number_of_paths):
            plt.plot(t,y_plot[i], color="red", alpha=0.5)
        plt.title("Y Values")

```



```

#Define optimization parameters
# params = list(y0_model_main.parameters())
params=[]
for i in range(NT):
    params += list(z_models_main[i].parameters())

#Set up optimizer and scheduler
optimizer = optim.AdamW(params, lr=learning_rate,weight_decay=5e-4)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.995)

for k in range(0,MaxBatch):

    print("Batch Number: ", k+1)
    sloss=0
    #optimize main network wrt the foward loss
    for l in range(0,OptimSteps):
        optimizer.zero_grad()

        loss = get_foward_loss_coupled(dB, init_x,NT=NT,target=target, y0_model=y0_model_main, z_models=z_models_main)
        # print(loss)

        loss.backward()
        # print(params)
        # torch.nn.utils.clip_grad_norm_(parameters=params,max_norm=5,norm_type=1)
        optimizer.step()
        scheduler.step()
        nloss = loss.detach().numpy()
        sloss += nloss
        # print('OptimStep: ' + str(l+1))
        # print('forward_loss: ' + str(nloss))
    avgloss = sloss/OptimSteps
    print("Average Error Est: ", avgloss)
    forward_losses.append(avgloss)

    #Generate a new batch if using multiple batches
    if(not single_batch):
        dB = SampleBMIncr(1, Npaths=NumTrain, Nsteps=NT+1)
        init_x = Sample_Init(N=NumTrain)

plot=plot_results(loss=forward_losses)
plot.FwdLoss()

```


Batch Number: 1
Average Error Est: 390.4913024902344
Batch Number: 2
Average Error Est: 354.5587677001953
Batch Number: 3
Average Error Est: 315.19701995849607
Batch Number: 4
Average Error Est: 269.4810157775879
Batch Number: 5
Average Error Est: 220.71451644897462
Batch Number: 6
Average Error Est: 177.60297622680665
Batch Number: 7
Average Error Est: 146.60689392089844
Batch Number: 8
Average Error Est: 122.84807815551758
Batch Number: 9
Average Error Est: 102.76398506164551
Batch Number: 10
Average Error Est: 87.26106834411621
Batch Number: 11
Average Error Est: 78.15623817443847
Batch Number: 12
Average Error Est: 72.19101486206054
Batch Number: 13
Average Error Est: 67.85396995544434
Batch Number: 14
Average Error Est: 63.964501762390135
Batch Number: 15
Average Error Est: 60.350241088867186
Batch Number: 16
Average Error Est: 57.29459190368652
Batch Number: 17
Average Error Est: 55.48044261932373
Batch Number: 18
Average Error Est: 53.51588134765625
Batch Number: 19
Average Error Est: 51.56993732452393
Batch Number: 20
Average Error Est: 49.29841613769531
Batch Number: 21
Average Error Est: 47.332977867126466
Batch Number: 22
Average Error Est: 45.85554504394531
Batch Number: 23
Average Error Est: 44.60461864471436
Batch Number: 24