Assignment 7

# Part 1 – Running Code without any changes

## Output

Chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated



## Code

# DL13B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using LSTM from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL13B.py  
  
import numpy as np  
import random  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt'), dtype=torch.float32) / 500.0  
  
T = len(x)  
num\_train = T // 2  
tau = 4  
input\_size = 1  
hidden\_size = 10  
output\_size = 1  
batch\_size = 32  
sigma = 0.01  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X[:num\_train]  
ytrain = y[:num\_train]  
  
class LSTMScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(LSTMScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.inputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.outputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 C = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 I = torch.sigmoid(self.inputgate(input))  
 F = torch.sigmoid(self.forgetgate(input))  
 O = torch.sigmoid(self.outputgate(input))  
 C\_tilda = torch.tanh(self.candidate(input))  
 C = F \* C + I \* C\_tilda  
 H = O \* torch.tanh(C)  
 return self.linear(H)  
  
model = LSTMScratch()  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
loss\_fun = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters())  
rounds = 1000  
losses = np.zeros(rounds)  
indices = list(range(num\_train))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 y\_pred = model(X[batch\_indices])  
 loss = loss\_fun(y\_pred, y[batch\_indices])  
 losses[i] = loss.item()  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
print(losses[rounds - 1])  
plt.plot(losses)  
plt.show()

# Part 2 – Replace all sigmoid function with relu

## Output

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Text, logo

Description automatically generated

## Comments

Compared to part 1 (no changes), relu gives a slightly lower MSE, indicating that it produces better results.

## Code

# DL13B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using LSTM from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL13B.py  
  
import numpy as np  
import random  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt'), dtype=torch.float32) / 500.0  
  
T = len(x)  
num\_train = T // 2  
tau = 4  
input\_size = 1  
hidden\_size = 10  
output\_size = 1  
batch\_size = 32  
sigma = 0.01  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X[:num\_train]  
ytrain = y[:num\_train]  
  
class LSTMScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(LSTMScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.inputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.outputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 C = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 I = torch.relu(self.inputgate(input))  
 F = torch.relu(self.forgetgate(input))  
 O = torch.relu(self.outputgate(input))  
 C\_tilda = torch.tanh(self.candidate(input))  
 C = F \* C + I \* C\_tilda  
 H = O \* torch.tanh(C)  
 return self.linear(H)  
  
model = LSTMScratch()  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
loss\_fun = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters())  
rounds = 1000  
losses = np.zeros(rounds)  
indices = list(range(num\_train))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 y\_pred = model(X[batch\_indices])  
 loss = loss\_fun(y\_pred, y[batch\_indices])  
 losses[i] = loss.item()  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
print(losses[rounds - 1])  
plt.plot(losses)  
plt.show()

# Part 3 – Remove second application of tanh

## Using original code (with sigmoid)

### Output

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated



### Comments

This change gave similar (slightly worsened) results to the original code as shown above. No significant change in performance.

### Code

# DL13B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using LSTM from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL13B.py  
  
import numpy as np  
import random  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt'), dtype=torch.float32) / 500.0  
  
T = len(x)  
num\_train = T // 2  
tau = 4  
input\_size = 1  
hidden\_size = 10  
output\_size = 1  
batch\_size = 32  
sigma = 0.01  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X[:num\_train]  
ytrain = y[:num\_train]  
  
class LSTMScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(LSTMScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.inputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.outputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 C = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 I = torch.sigmoid(self.inputgate(input))  
 F = torch.sigmoid(self.forgetgate(input))  
 O = torch.sigmoid(self.outputgate(input))  
 C\_tilda = torch.tanh(self.candidate(input))  
 C = F \* C + I \* C\_tilda  
 H = O \* C  
 return self.linear(H)  
  
model = LSTMScratch()  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
loss\_fun = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters())  
rounds = 1000  
losses = np.zeros(rounds)  
indices = list(range(num\_train))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 y\_pred = model(X[batch\_indices])  
 loss = loss\_fun(y\_pred, y[batch\_indices])  
 losses[i] = loss.item()  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
print(losses[rounds - 1])  
plt.plot(losses)  
plt.show()

## Using modified code from part 2 (with relu)

### Output

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Logo

Description automatically generated

### Comments

This has by far produced the worst MSE (performance). This suggests that switching all sigmoid functions to relu and removing the second tanh function, worsens the performance.

### Code

# DL13B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using LSTM from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL13B.py  
  
import numpy as np  
import random  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt'), dtype=torch.float32) / 500.0  
  
T = len(x)  
num\_train = T // 2  
tau = 4  
input\_size = 1  
hidden\_size = 10  
output\_size = 1  
batch\_size = 32  
sigma = 0.01  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X[:num\_train]  
ytrain = y[:num\_train]  
  
class LSTMScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(LSTMScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.inputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.outputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 C = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 I = torch.relu(self.inputgate(input))  
 F = torch.relu(self.forgetgate(input))  
 O = torch.relu(self.outputgate(input))  
 C\_tilda = torch.tanh(self.candidate(input))  
 C = F \* C + I \* C\_tilda  
 H = O \* C  
 return self.linear(H)  
  
model = LSTMScratch()  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
loss\_fun = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters())  
rounds = 1000  
losses = np.zeros(rounds)  
indices = list(range(num\_train))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 y\_pred = model(X[batch\_indices])  
 loss = loss\_fun(y\_pred, y[batch\_indices])  
 losses[i] = loss.item()  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
print(losses[rounds - 1])  
plt.plot(losses)  
plt.show()

# Part 4 – Replace the first tanh with sigmoid/relu (with and without second one)

## Using original code (with sigmoid)

### With the second one

#### Output

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated



#### Comments

The MSE is comparatively larger here than the original code. The suggested change had a negative impact on the results.

#### Code

# DL13B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using LSTM from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL13B.py  
  
import numpy as np  
import random  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt'), dtype=torch.float32) / 500.0  
  
T = len(x)  
num\_train = T // 2  
tau = 4  
input\_size = 1  
hidden\_size = 10  
output\_size = 1  
batch\_size = 32  
sigma = 0.01  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X[:num\_train]  
ytrain = y[:num\_train]  
  
class LSTMScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(LSTMScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.inputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.outputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 C = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 I = torch.sigmoid(self.inputgate(input))  
 F = torch.sigmoid(self.forgetgate(input))  
 O = torch.sigmoid(self.outputgate(input))  
 C\_tilda = torch.sigmoid(self.candidate(input))  
 C = F \* C + I \* C\_tilda  
 H = O \* torch.tanh(C)  
 return self.linear(H)  
  
model = LSTMScratch()  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
loss\_fun = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters())  
rounds = 1000  
losses = np.zeros(rounds)  
indices = list(range(num\_train))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 y\_pred = model(X[batch\_indices])  
 loss = loss\_fun(y\_pred, y[batch\_indices])  
 losses[i] = loss.item()  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
print(losses[rounds - 1])  
plt.plot(losses)  
plt.show()

### Without the second one

#### Output

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Logo

Description automatically generated

#### Comments

Removing tanh worsened the performance further.

#### Code

# DL13B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using LSTM from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL13B.py  
  
import numpy as np  
import random  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt'), dtype=torch.float32) / 500.0  
  
T = len(x)  
num\_train = T // 2  
tau = 4  
input\_size = 1  
hidden\_size = 10  
output\_size = 1  
batch\_size = 32  
sigma = 0.01  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X[:num\_train]  
ytrain = y[:num\_train]  
  
class LSTMScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(LSTMScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.inputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.outputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 C = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 I = torch.sigmoid(self.inputgate(input))  
 F = torch.sigmoid(self.forgetgate(input))  
 O = torch.sigmoid(self.outputgate(input))  
 C\_tilda = torch.sigmoid(self.candidate(input))  
 C = F \* C + I \* C\_tilda  
 H = O \* C  
 return self.linear(H)  
  
model = LSTMScratch()  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
loss\_fun = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters())  
rounds = 1000  
losses = np.zeros(rounds)  
indices = list(range(num\_train))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 y\_pred = model(X[batch\_indices])  
 loss = loss\_fun(y\_pred, y[batch\_indices])  
 losses[i] = loss.item()  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
print(losses[rounds - 1])  
plt.plot(losses)  
plt.show()

### Additional Comments

Instead of replacing the first tanh with sigmoid, if we replaced it with relu, we get MSEs of 0.008427893742918968 (with second tanh) and 0.00421510636806488 (without tanh) respectively. It can be inferred that using relu improves performance in this case.

## Using modified code from part 2 (with relu)

### With the second one

#### Output

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Text, logo

Description automatically generated

#### Comments

Compared to part 2 (relu instead of sigmoid in the first three instances) and sigmoid instead of the first tanh, we get better performance.

#### Code

# DL13B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using LSTM from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL13B.py  
  
import numpy as np  
import random  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt'), dtype=torch.float32) / 500.0  
  
T = len(x)  
num\_train = T // 2  
tau = 4  
input\_size = 1  
hidden\_size = 10  
output\_size = 1  
batch\_size = 32  
sigma = 0.01  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X[:num\_train]  
ytrain = y[:num\_train]  
  
class LSTMScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(LSTMScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.inputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.outputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 C = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 I = torch.relu(self.inputgate(input))  
 F = torch.relu(self.forgetgate(input))  
 O = torch.relu(self.outputgate(input))  
 C\_tilda = torch.sigmoid(self.candidate(input))  
 C = F \* C + I \* C\_tilda  
 H = O \* torch.tanh(C)  
 return self.linear(H)  
  
model = LSTMScratch()  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
loss\_fun = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters())  
rounds = 1000  
losses = np.zeros(rounds)  
indices = list(range(num\_train))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 y\_pred = model(X[batch\_indices])  
 loss = loss\_fun(y\_pred, y[batch\_indices])  
 losses[i] = loss.item()  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
print(losses[rounds - 1])  
plt.plot(losses)  
plt.show()

### Without the second one

#### Output

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart

Description automatically generated

Text, logo

Description automatically generated

#### Comments

Removing tanh worsened the results significantly.

#### Code

# DL13B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using LSTM from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL13B.py  
  
import numpy as np  
import random  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt'), dtype=torch.float32) / 500.0  
  
T = len(x)  
num\_train = T // 2  
tau = 4  
input\_size = 1  
hidden\_size = 10  
output\_size = 1  
batch\_size = 32  
sigma = 0.01  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X[:num\_train]  
ytrain = y[:num\_train]  
  
class LSTMScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(LSTMScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.inputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.outputgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 C = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 I = torch.relu(self.inputgate(input))  
 F = torch.relu(self.forgetgate(input))  
 O = torch.relu(self.outputgate(input))  
 C\_tilda = torch.sigmoid(self.candidate(input))  
 C = F \* C + I \* C\_tilda  
 H = O \* C # torch.tanh(C)  
 return self.linear(H)  
  
model = LSTMScratch()  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
loss\_fun = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters())  
rounds = 1000  
losses = np.zeros(rounds)  
indices = list(range(num\_train))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 y\_pred = model(X[batch\_indices])  
 loss = loss\_fun(y\_pred, y[batch\_indices])  
 losses[i] = loss.item()  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
y2 = model(X)  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
plt.show()  
  
print(losses[rounds - 1])  
plt.plot(losses)  
plt.show()

### Additional Comments

Instead of replacing the first tanh with sigmoid, if we replaced it with relu, we get MSEs of 0.004794727545231581 (with second tanh) and 0.3094731867313385 (without tanh) respectively. It can be inferred that using relu improves performance.

# Final Comments

Based on the above results, replacing the first tanh with relu and keeping the second tanh produces the best results.