Assignment 8

# Part 1 – Results from DL14A.py

## Original Implementation of GRU

### Output

Chart, line chart

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated



### Comments

The above screenshots show the output for the original code. This is the base that will be used to compare and contrast the other implementations with.

### Code

# DL14A.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using GRU from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL14A.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 = 20  
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 GRUScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(GRUScratch, self).\_\_init\_\_()  
 self.resetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.updategate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = 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)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 R = torch.sigmoid(self.resetgate(input))  
 Z = torch.sigmoid(self.updategate(input))  
 input2 = torch.cat((X, R \* H), 1)  
 H\_tilda = torch.tanh(self.candidate(input2))  
 H = Z \* H + (1 - Z) \* H\_tilda  
 return self.linear(H)  
  
model = GRUScratch()  
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()

## Implementation of GRU-1

### Output

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated



### Comments

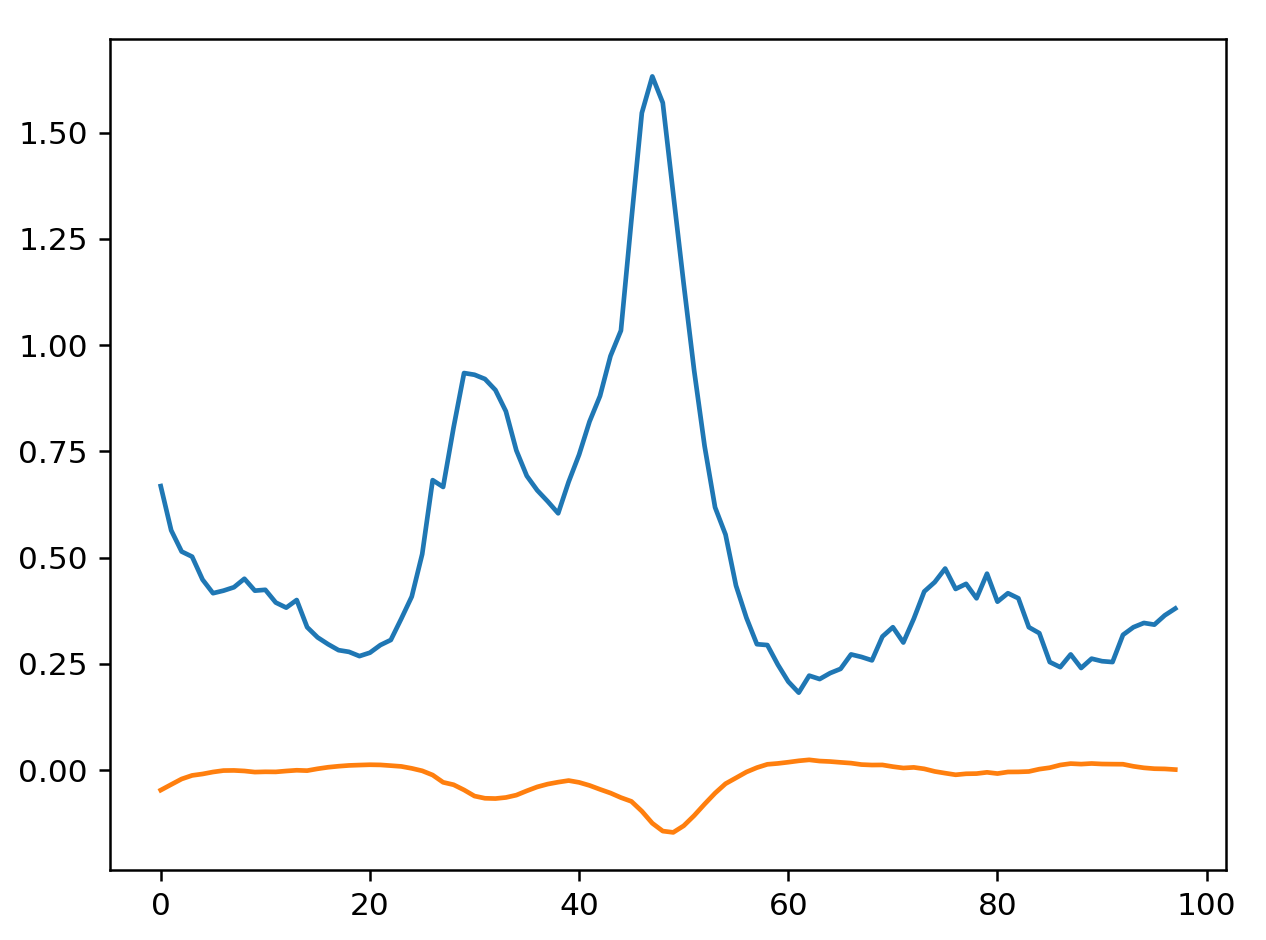
Variant 1 of GRU or GRU-1 computes each gate using only the previous hidden state and the bias, thus reducing the total number of parameters by 2 x nm. The above output shows that this variation gives better results compared to GRU-0 in terms of performance.

### Code

# DL14A.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using GRU from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL14A.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 = 20  
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 GRUScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(GRUScratch, self).\_\_init\_\_()  
 self.resetgate = torch.nn.Linear(hidden\_size, hidden\_size, bias=True)  
 self.updategate = torch.nn.Linear(hidden\_size, hidden\_size, bias=True)  
 self.candidate = torch.nn.Linear(input\_size+hidden\_size, hidden\_size, bias=True)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size, bias=True)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 R = torch.sigmoid(self.resetgate(H))  
 Z = torch.sigmoid(self.updategate(H))  
 input = torch.cat((X, R \* H), 1)  
 H\_tilda = torch.tanh(self.candidate(input))  
 H = Z \* H + (1 - Z) \* H\_tilda  
 return self.linear(H)  
  
  
model = GRUScratch()  
y2 = model(X)  
plt.show()  
  
loss\_fun = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters())  
rounds = 1000  
plt.plot(y)  
plt.plot(y2.detach().numpy())  
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()

## Implementation of GRU-2

### Output



Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Logo

Description automatically generated

### Comments

Variant 2 of GRU or GRU computes each gate using only the previous hidden state, thus reducing the total number of parameters by 2 x (nm+n). seems to slightly improve the performance compared to the original GNU RNN.

### Code

# DL14A.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using GRU from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL14A.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 = 20  
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 GRUScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(GRUScratch, self).\_\_init\_\_()  
 self.resetgate = torch.nn.Linear(hidden\_size, hidden\_size, bias=False)  
 self.updategate = torch.nn.Linear(hidden\_size, hidden\_size, bias=False)  
 self.candidate = torch.nn.Linear(input\_size+hidden\_size, hidden\_size, bias=False)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size, bias=False)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 R = torch.sigmoid(self.resetgate(H))  
 Z = torch.sigmoid(self.updategate(H))  
 input = torch.cat((X, R \* H), 1)  
 H\_tilda = torch.tanh(self.candidate(input))  
 H = Z \* H + (1 - Z) \* H\_tilda  
 return self.linear(H)  
  
  
model = GRUScratch()  
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()

## Implementation of GRU-3

### Output

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated



### Comments

This variation uses only the bias to perform computations, thereby reducing the total number of parameters, to 2 \*(nm+n2). The accuracy is slightly less than what GRU-0 achieves.

### Code

# DL14A.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using GRU from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL14A.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 = 20  
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 GRUScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(GRUScratch, self).\_\_init\_\_()  
 self.resetgate = torch.zeros((1, hidden\_size))  
 self.updategate = torch.zeros((1, hidden\_size))  
 self.candidate = torch.nn.Linear(input\_size+hidden\_size, hidden\_size, bias=False)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size, bias=False)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 R = torch.sigmoid(self.resetgate)  
 Z = torch.sigmoid(self.updategate)  
 input = torch.cat((X, R \* H), 1)  
 H\_tilda = torch.tanh(self.candidate(input))  
 H = Z \* H + (1 - Z) \* H\_tilda  
 return self.linear(H)  
  
  
model = GRUScratch()  
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()

## Overall Comments on GRU

Overall, GRU 1 gives the best results, followed by GRU-2, GRU-0 and finally GRU-3/

# Part 2 – Results from DL14B.py

## Original Implementation of MGU

### Output

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated



### Comments

The above screenshots show the output for the original code. This is the base that will be used to compare and contrast the other implementations with.

### Code

# DL14B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using MGU from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL14B.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 = 20  
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 MGUScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(MGUScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size)  
 self.candidate = 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)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 input = torch.cat((X, H), 1)  
 F = torch.sigmoid(self.forgetgate(input))  
 input2 = torch.cat((X, F \* H), 1)  
 H\_tilda = torch.tanh(self.candidate(input2))  
 H = (1 - F) \* H + F \* H\_tilda  
 return self.linear(H)  
  
model = MGUScratch()  
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()

## Implementation of MGU1

### Output

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated



### Comments

Variant 1 of MGU or MGU1 removes the input signal from the gate signal equation, which makes it dependent only on the unit history and bias. The above output shows that this variation gives slightly better results compared to MGU0 in terms of performance.

### Code

# DL14B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using MGU from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL14B.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 = 20  
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 MGUScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(MGUScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(hidden\_size, hidden\_size)  
 self.candidate = 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)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 F = torch.sigmoid(self.forgetgate(H))  
 input2 = torch.cat((X, F \* H), 1)  
 H\_tilda = torch.tanh(self.candidate(input2))  
 H = (1 - F) \* H + F \* H\_tilda  
 return self.linear(H)  
  
model = MGUScratch()  
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()

## Implementation of MGU-2

### Output

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated



### Comments

This variant removes both the input and the bias from the gate signal equation, making it dependent solely on the unit history. This variant seems to produce a slightly better result than the original.

### Code

# DL14B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using MGU from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL14B.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 = 20  
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 MGUScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(MGUScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.nn.Linear(hidden\_size, hidden\_size, bias=False)  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size, bias=False)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 F = torch.sigmoid(self.forgetgate(H))  
 input2 = torch.cat((X, F \* H), 1)  
 H\_tilda = torch.tanh(self.candidate(input2))  
 H = (1 - F) \* H + F \* H\_tilda  
 return self.linear(H)  
  
model = MGUScratch()  
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()

## Implementation of MGU-3

### Output

Chart, line chart

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Text

Description automatically generated with medium confidence

### Comments

This variation seems to produce results with similar performance of that of the original.

### Code

# DL14B.py CS5173/6073 cheng 2023  
# autoregression on hospitalization  
# using MGU from scratch  
# using Linear  
# using MSELoss and Adam  
# with random sample of training data  
# Usage: python DL14B.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 = 20  
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 MGUScratch(torch.nn.Module):   
 def \_\_init\_\_(self):  
 super(MGUScratch, self).\_\_init\_\_()  
 self.forgetgate = torch.zeros((1, hidden\_size))  
 self.candidate = torch.nn.Linear(input\_size + hidden\_size, hidden\_size, bias=False)  
 self.linear = torch.nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, x):  
 H = torch.randn(len(x), hidden\_size)  
 X2 = torch.reshape(x.T, (tau, len(x), input\_size))  
 for X in X2:  
 F = torch.sigmoid(self.forgetgate)  
 input = torch.cat((X, F \* H), 1)  
 H\_tilda = torch.tanh(self.candidate(input))  
 H = (1 - F) \* H + F \* H\_tilda  
 return self.linear(H)  
  
model = MGUScratch()  
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()

## Overall Comments on MGU

Overall MGU2 gives the best performance.