Assignment 1

# Part 1

## Tau = 2

Chart

Description automatically generated

Chart, histogram

Description automatically generated

Text

Description automatically generated

## Tau = 4

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Text

Description automatically generated

## Tau = 8

Chart

Description automatically generated

Chart, histogram

Description automatically generated

Text

Description automatically generated

## Explanation

As Tau increases, the number of past conditions considered increases which provides a more accurate prediction (reduced MSE)

## Code

# DL3A.py CS5173/6073 cheng 2023  
# autoregression with linear regression  
# following D2l 9.1  
# using the closed form solution as in DL1A.py  
# Usage: python DL3A.py  
  
import torch  
import matplotlib.pyplot as plt  
  
T = 1000  
time = torch.arange(1, T + 1, dtype=torch.float32)  
x = torch.sin(0.01 \* time) + torch.randn(T) \* 0.2  
plt.plot(time, x)  
plt.show()  
  
num\_train = 600  
tau = 8  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
X\_b = torch.cat((torch.ones(len(X), 1), X), 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X\_b[:num\_train]  
ytrain = y[:num\_train]  
wb\_best = torch.linalg.inv(Xtrain.T.matmul(Xtrain)).matmul(Xtrain.T).matmul(ytrain)   
print(wb\_best)  
  
y\_pred = X\_b.matmul(wb\_best)  
plt.plot(time[tau:], y)  
plt.plot(time[tau:], y\_pred)  
plt.show()  
  
diff = y\_pred - y  
train\_loss = torch.mean(diff[:num\_train] \* diff[:num\_train])  
test\_loss = torch.mean(diff[num\_train:] \* diff[num\_train:])  
print('training MSE', train\_loss.item())  
print('test MSE', test\_loss.item())

# Part 2

## DL3A.py

### Output

Chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Text

Description automatically generated

### Code

# DL3A.py CS5173/6073 cheng 2023  
# autoregression with linear regression  
# following D2l 9.1  
# using the closed form solution as in DL1A.py  
# Usage: python DL3A.py  
  
import numpy as np  
import torch  
import matplotlib.pyplot as plt  
  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt'))  
plt.plot(x)  
plt.show()  
  
T = len(x)  
num\_train = T // 2 # using the first half of data for training  
tau = 4  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
X\_b = torch.cat((torch.ones(len(X), 1), X), 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X\_b[:num\_train]  
ytrain = y[:num\_train]  
wb\_best = torch.linalg.inv(Xtrain.T.matmul(Xtrain)).matmul(Xtrain.T).matmul(ytrain)   
print(wb\_best)  
  
y\_pred = X\_b.matmul(wb\_best)  
plt.plot(y)  
plt.plot(y\_pred)  
plt.show()  
  
diff = y\_pred - y  
train\_loss = torch.mean(diff[:num\_train] \* diff[:num\_train])  
test\_loss = torch.mean(diff[num\_train:] \* diff[num\_train:])  
print('training MSE', train\_loss.item())  
print('test MSE', test\_loss.item())

## DL3B.py

### Output

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

A picture containing text, sign

Description automatically generated

### Code

# DL3B.py CS5173/6073 cheng 2023  
# autoregression with linear regression  
# following D2l 9.1  
# gradient descent without backward()  
# Usage: python DL3B.py  
  
import numpy as np  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt')).float()  
T = len(x)  
num\_train = T // 2 # using the first half of data for training  
tau = 4  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
X\_b = torch.cat((torch.ones(len(X), 1), X), 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X\_b[:num\_train]  
ytrain = y[:num\_train]  
wb = torch.randn(tau+1, 1)  
eta = 0.001  
for i in range(4):  
 y\_pred = X\_b.matmul(wb)  
 plt.plot(y)  
 plt.plot(y\_pred)  
 plt.show()  
 gradient = Xtrain.T.matmul(y\_pred[:num\_train] - ytrain)  
 wb -= eta \* gradient  
  
 diff = y\_pred[num\_train:] - y[num\_train:]  
 test\_loss = torch.mean(diff \* diff)  
 print('test MSE', test\_loss.item())

## DL3C.py

### Output

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Text

Description automatically generated

### Code

# DL3C.py CS5173/6073 cheng 2023  
# autoregression with linear regression  
# following D2l 9.1  
# gradient descent with backward()  
# Usage: python DL3C.py  
  
import numpy as np  
import torch  
import matplotlib.pyplot as plt  
  
x = torch.tensor(np.genfromtxt('hamiltonCountyHospitalization.txt')).float()  
T = len(x)  
num\_train = T // 2 # using the first half of data for training  
tau = 4  
  
features = [x[i: T-tau+i] for i in range(tau)]  
X = torch.stack(features, 1)  
X\_b = torch.cat((torch.ones(len(X), 1), X), 1)  
y = x[tau:].reshape((-1, 1))  
Xtrain = X\_b[:num\_train]  
ytrain = y[:num\_train]  
wb = torch.randn(tau+1, 1)  
wb.requires\_grad\_(True)  
wb.grad = torch.zeros(tau+1, 1)  
eta = 0.2  
for i in range(4):  
 y\_pred = X\_b.matmul(wb)  
 plt.plot(y)  
 plt.plot(y\_pred.detach().numpy())  
 plt.show()  
 diff = (y\_pred - y)[:num\_train]  
 loss = torch.mean(diff \* diff)  
 print('loss:', loss.item())  
 wb.grad.zero\_()  
 loss.backward()  
 wb.data = wb.data - eta \* wb.grad  
  
 diff = y\_pred[num\_train:] - y[num\_train:]  
 test\_loss = torch.mean(diff \* diff)  
 print('test MSE', test\_loss.item())