4105 Hw 5, Owen Evans, 801206774

In [307]:
```
%matplotlib inline
import numpy as np
import torch
import torch.optim as optim

torch.set_printoptions(edgeitems=2, linewidth=75)
```

In [308]:
```
t_c = [0.5,  14.0, 15.0, 28.0, 11.0,  8.0,  3.0, -4.0,  6.0, 13.0, 21.0]
t_u = [35.7, 55.9, 58.2, 81.9, 56.3, 48.9, 33.9, 21.8, 48.4, 60.4, 68.4]
t_c = torch.tensor(t_c).unsqueeze(1) # <1>
t_u = torch.tensor(t_u).unsqueeze(1) # <1>


t_un = 0.1 * t_u
t_un.shape
```

Out[308]:
```
torch.Size([11, 1])
```

In [356]:
```
n_samples = t_u.shape[0]
n_val = int(0.2 * n_samples)

shuffled_indices = torch.randperm(n_samples)

train_indices = shuffled_indices[:-n_val]
val_indices = shuffled_indices[-n_val:]

train_indices, val_indices
```

Out[356]:
```
(tensor([ 7,  2,  9,  1,  4,  6,  0,  3, 10]), tensor([5, 8]))
```

In [357]:
```
t_u_train = t_u[train_indices]
t_c_train = t_c[train_indices]

t_u_val = t_u[val_indices]
t_c_val = t_c[val_indices]

t_un_train = 0.1 * t_u_train
t_un_val = 0.1 * t_u_val
```

In [358]:
```
def model(t_u, w2, w1, b):
    return w2 * t_u**2 + w1 * t_u + b
```

In [359]:
```
def lin_model(t_u, wlin, blin):
    return wlin * t_u + blin
```

In [360]:
```
def loss_fn(t_p, t_c):
    squared_diffs = (t_p - t_c)**2
    return squared_diffs.mean()
```

In [361]:
```
loss= loss_fn(t_p,t_c)
```

In [362]:
```
w1=torch.ones(())
w2=torch.ones(())
b =torch.zeros(())
wlin=torch.ones(())
blin= torch.zeros(())

t_p= model(t_u, w2, w1, b)
t_plin= lin_model(t_u,wlin,blin)
```

In [363]:
```
def dloss_fn(t_p, t_c):
    dsq_diffs = 2 * (t_p - t_c) / t_p.size(0)
    return dsq_diffs
```

In [364]:
```
def dmodel_w1(t_u, w2, w1, b):
    return t_u

def dmodel_w2(t_u, w2, w1, b):
    return t_u**2

def dmodel_b(t_u, w2, w1, b):
    return 1.0
```

In [365]:
```
def dmodel_wlin(t_u, wlin, blin):
    return t_u
def dmodel_blin(tu, wlin, blin):
    return 1.0
```

In [366]:
```
def grad_fn(t_u, t_c, t_p, w2, w1, b):
    dloss_tp = dloss_fn(t_p, t_c)
    dloss_w2 = dloss_tp * dmodel_w2(t_u, w2, w1, b)
    dloss_w1 = dloss_tp * dmodel_w1(t_u, w2, w1, b)
    dloss_b = dloss_tp * dmodel_b(t_u, w1 , w2, b)
    return torch.stack([dloss_w2.sum(), dloss_w1.sum(), dloss_b.sum()])
```

In [367]:
```
def lin_grad_fn(t_u, t_c, t_p, wlin, blin):
    dloss_tp = dloss_fn(t_p, t_c)
    dloss_w = dloss_tp * dmodel_wlin(t_u, wlin, blin)
    dloss_b = dloss_tp * dmodel_blin(t_u, wlin, blin)
    return torch.stack([dloss_w.sum(), dloss_b.sum()])
```

In [368]:
```
def training_loop(n_epochs, learning_rate, params, t_u, t_c,print_results):
    print('')
    print('Learning rate:',learning_rate)
    for epoch in range(1, n_epochs + 1):
        w1, w2, b = params
        t_p = model(t_u, w2, w1, b)
        grad = grad_fn(t_u, t_c, t_p, w1, w2, b)
        loss = loss_fn(t_p, t_c)
        params = params - learning_rate * grad
        if epoch % 500 == 0:
            print('Epoch %d, Loss %f' % (epoch, float(loss)))
    return params, loss
```

In [369]:
```
def training_loop_lin(n_epochs, learning_rate, params, t_u, t_c,print_results):
    for epoch in range(1, n_epochs + 1):
        w,b = params
        t_p = lin_model(t_u, w, b)
        grad = lin_grad_fn(t_u, t_c, t_p, w, b)
        loss = loss_fn(t_p, t_c)
        params = params - learning_rate * grad
        if epoch % 500 == 0:
            print('Epoch %d, Loss %f' % (epoch, float(loss)))
    return params, loss
```

In [370]:
```
epochs = 5000
print_eras = 500
delta = [0.1, 0.01, 0.001, 0.0001]
ini_params = torch.tensor([1.0, 1.0, 0.0])
ini_lin_params = torch.tensor([1.0, 0.0])
trained_params = torch.zeros(len(delta), 3)
trained_loss = torch.zeros(len(delta), 1)
trained_lin_params = torch.zeros(1, 2)
trained_lin_loss = torch.zeros(1, 1)
for i in delta:
    trained_params[delta.index(i)], trained_loss[delta.index(i)] = training_loop(
trained_lin_params, trained_lin_loss = training_loop_lin(epochs, 1e-4, ini_lin_pa
```

```
Learning rate: 0.1
Epoch 500, Loss nan
Epoch 1000, Loss nan
Epoch 1500, Loss nan
Epoch 2000, Loss nan
Epoch 2500, Loss nan
Epoch 3000, Loss nan
Epoch 3500, Loss nan
Epoch 4000, Loss nan
Epoch 4500, Loss nan
Epoch 5000, Loss nan

Learning rate: 0.01
Epoch 500, Loss nan
Epoch 1000, Loss nan
Epoch 1500, Loss nan
Epoch 2000, Loss nan
Epoch 2500, Loss nan
Epoch 3000, Loss nan
Epoch 3500, Loss nan
Epoch 4000, Loss nan
Epoch 4500, Loss nan
Epoch 5000, Loss nan

Learning rate: 0.001
Epoch 500, Loss 801.891052
Epoch 1000, Loss 8211912.500000
Epoch 1500, Loss 84494336000.000000
Epoch 2000, Loss 869382381109248.000000
Epoch 2500, Loss 8945286854492553216.000000
Epoch 3000, Loss 92040227714157412089856.000000
Epoch 3500, Loss 947023669622583830507421696.000000
Epoch 4000, Loss 9744150645797794671664415375360.000000
Epoch 4500, Loss 1002599516930571772546197370762690056.000000
Epoch 5000, Loss inf

Learning rate: 0.0001
Epoch 500, Loss 4.033324
Epoch 1000, Loss 4.332656
Epoch 1500, Loss 5.098123
Epoch 2000, Loss 7.042311
Epoch 2500, Loss 11.967194
```

```
Epoch 3000, Loss 24.429565
Epoch 3500, Loss 55.952553
Epoch 4000, Loss 135.675476
Epoch 4500, Loss 337.285278
Epoch 5000, Loss 847.119873
Epoch 500, Loss 29.505890
Epoch 1000, Loss 28.943773
Epoch 1500, Loss 28.505281
Epoch 2000, Loss 28.074451
Epoch 2500, Loss 27.650877
Epoch 3000, Loss 27.234444
Epoch 3500, Loss 26.825020
Epoch 4000, Loss 26.422497
Epoch 4500, Loss 26.026747
Epoch 5000, Loss 25.637672
```

In [371]:
```
print(trained_params)
```

```
tensor([[        nan,         nan,         nan],
        [        nan,         nan,         nan],
        [ 2.7191e+19, -4.2890e+18, -2.6094e+18],
        [ 2.2811e+01, -3.1993e+00, -2.5509e+00]])
```

In [372]:
```
best = trained_params[3]
print(best)
t_p = model(t_un, * best_test)
t_p_lin = lin_model(t_un, * trained_lin_params)
```

```
tensor([22.8110, -3.1993, -2.5509])
```

In [375]:
```
from matplotlib import pyplot as plt

t_p = model(t_un, * params)
t_range = torch.arange(20., 90.).unsqueeze(1)

fig = plt.figure(dpi=600)
plt.xlabel("Fahrenheit")
plt.ylabel("Celsius")
plt.plot(t_u.numpy(), t_c.numpy(), 'o')
plt.plot(t_u.numpy(), t_p.detach().numpy())
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [375], in <cell line: 3>()
      1 from matplotlib import pyplot as plt
----> 3 t_p = model(t_un, * params)
      4 t_range = torch.arange(20., 90.).unsqueeze(1)
      6 fig = plt.figure(dpi=600)

TypeError: model() takes 4 positional arguments but 7 were given
```

Problem 2

In [327]:
```
#get data
import pandas as pd

house = pd.read_csv ('https://raw.githubusercontent.com/Owen3vans/4105_HW1/main/H
house.head()
```

Out[327]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheati
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [328]:
```
varlist = ['price','area', 'bedrooms', 'bathrooms', 'stories', 'parking']
housing = house[varlist]
housing.head()
```

Out[328]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

In [329]:
```
price= torch.tensor(housing['price'])
area= torch.tensor(housing['area'])
beds= torch.tensor(housing['bedrooms'])
baths= torch.tensor(housing['bathrooms'])
stories= torch.tensor(housing['stories'])
parking= torch.tensor(housing['parking'])

narea= area/ max(housing['area'])
nbeds= beds/ max(housing['bedrooms'])
nbaths= baths/ max(housing['bathrooms'])
nstories= stories/ max(housing['stories'])
nparking= parking/ max(housing['parking'])
```

In [330]:
```
def model(area, beds, baths, stories, parking, w1, w2, w3, w4, w5, b):
    return w1*area + w2*beds+ w3*baths + w4*stories + w5*parking + b
```

In [331]:
```
def loss_fn(t_p,price):
    squared_diffs= (t_p - price)**2
    return squared_diffs.mean()
```

In [332]:
```
params = torch.tensor([1.0,1.0,1.0,1.0,1.0,0.0], requires_grad=True)
params.grad is None
```

Out[332]:
```
True
```

In [333]:
```
loss = loss_fn(model(area, beds, baths, stories, parking, *params), price)
loss.backward()
params.grad
```

Out[333]:
```
tensor([-5.3383e+10, -2.9247e+07, -1.3220e+07, -1.8557e+07, -7.8404e+06,
        -9.5231e+06])
```

In [334]:
```
if params.grad is not None:
    params.grad.zero_()
```

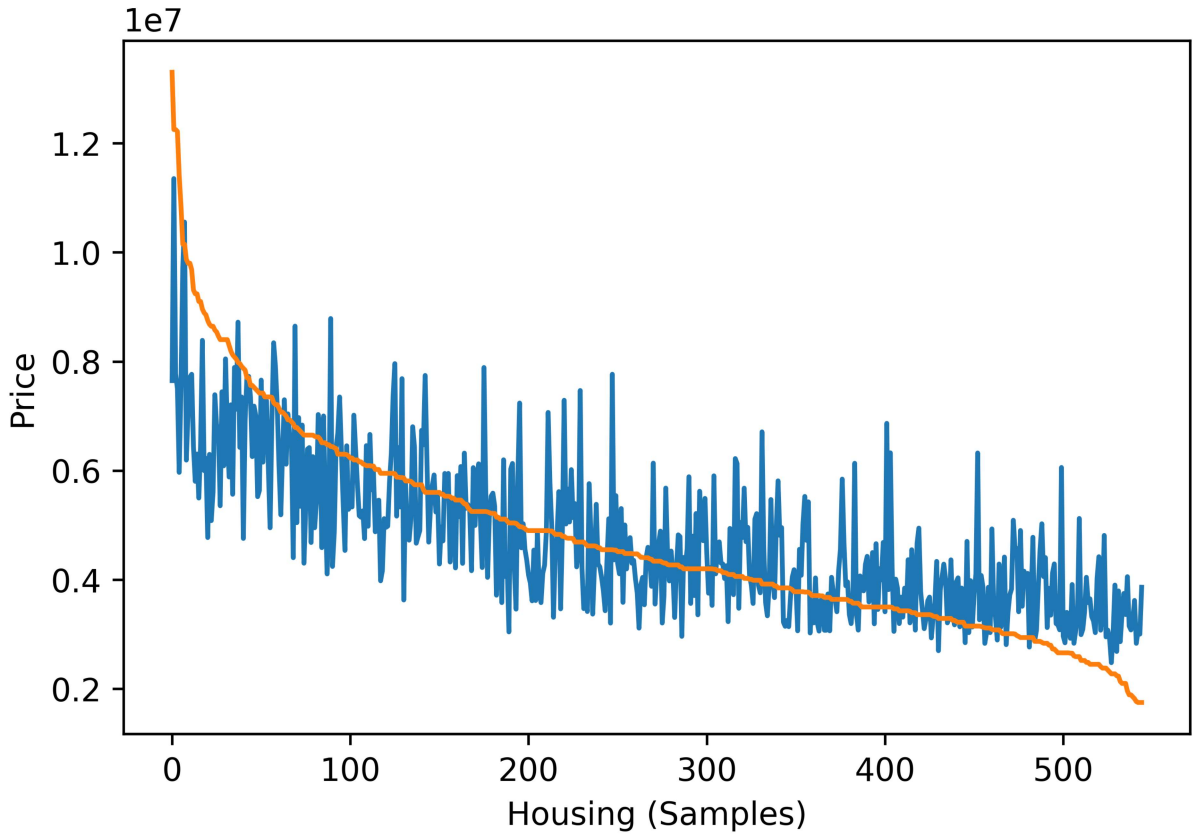
In [335]:
```
def training_loop(n_epochs, learning_rate, params, area, beds, baths, stories, pa
    for epoch in range(1, n_epochs + 1):
        if params.grad is not None:
            params.grad.zero_()

        t_p = model(area, beds, baths, stories, parking, *params)
        loss = loss_fn(t_p,price)
        loss.backward()

        with torch.no_grad():
            params -= learning_rate * params.grad

        if epoch % 500 == 0:
            print('Epoch %d, Loss %f' % (epoch, float(loss)))

    return params
```

In [336]:
```
params = training_loop(
    n_epochs = 5000,
    learning_rate = 1e-1,
    params = torch.tensor([1.0,1.0,1.0,1.0,1.0,0.0], requires_grad=True),
    area = narea,
    beds = nbeds,
    baths = nbaths,
    stories = nstories,
    parking = nparking,
    price = price)
```

```
Epoch 500, Loss 1562038632448.000000
Epoch 1000, Loss 1533011296256.000000
Epoch 1500, Loss 1531162132480.000000
Epoch 2000, Loss 1531023589376.000000
Epoch 2500, Loss 1531010482176.000000
Epoch 3000, Loss 1531008778240.000000
Epoch 3500, Loss 1531008516096.000000
Epoch 4000, Loss 1531008647168.000000
Epoch 4500, Loss 1531008647168.000000
Epoch 5000, Loss 1531008516096.000000
```

In [337]:
```
%matplotlib inline
t_p = model(narea, nbeds, nbaths, nstories, nparking, *params)
fig = plt.figure(dpi=600)
plt.rcParams["figure.figsize"] = (10,6)
plt.xlabel("Housing (Samples)")
plt.ylabel("Price")
plt.plot(t_p.detach().numpy())
plt.plot(price)
```

Out[337]:
```
[<matplotlib.lines.Line2D at 0x1e1015f2f40>]
```



Problem 3

In [338]:
```
housing.head()
```

Out[338]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

In [339]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(housing, test_size = 0.2)
test.shape
```

Out[339]: (109, 6)

In [340]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import warnings
warnings.filterwarnings('ignore')

sc=  MinMaxScaler()
train[varlist] = sc.fit_transform(train[varlist])
test[varlist] = sc.fit_transform(test[varlist])
test.head()
```

Out[340]:

	price	area	bedrooms	bathrooms	stories	parking
425	0.166667	0.089542	0.00	0.0	0.000000	0.666667
515	0.072464	0.091291	0.25	0.0	0.333333	0.000000
321	0.235507	0.120672	0.25	0.5	0.333333	0.666667
86	0.492754	0.333333	0.25	0.0	0.666667	0.000000
208	0.326087	0.074502	0.25	0.0	0.666667	0.000000

In [341]:
```
x_test = test
x_train = train
y_test = test.pop('price')
y_train = train.pop('price')

x_test = torch.tensor(x_test.values).float()
x_train = torch.tensor(x_train.values).float()
y_test = torch.tensor(y_test.values).float()
y_test = y_test[:,None]
y_train = torch.tensor(y_train.values).float()
y_train = y_train[:,None]
```

In [354]:
```
import torch.nn as nn
import torch.optim as optim

seq1 = nn.Sequential(
                nn.Linear(5,8),
                nn.Tanh(),
                nn.Linear(8,16),
                nn.Tanh(),
                nn.Linear(16,1))
seq1
```

Out[354]:
```
Sequential(
  (0): Linear(in_features=5, out_features=8, bias=True)
  (1): Tanh()
  (2): Linear(in_features=8, out_features=16, bias=True)
  (3): Tanh()
  (4): Linear(in_features=16, out_features=1, bias=True)
)
```

In [344]:
```
seq2 = nn.Sequential(
                nn.Linear(5,8),
                nn.Tanh(),
                nn.Linear(8,16),
                nn.Tanh(),
                nn.Linear(16, 32),
                nn.Tanh(),
                nn.Linear(32,64),
                nn.Tanh(),
                nn.Linear(64,1))
seq2
```

Out[344]:
```
Sequential(
  (0): Linear(in_features=5, out_features=8, bias=True)
  (1): Tanh()
  (2): Linear(in_features=8, out_features=16, bias=True)
  (3): Tanh()
  (4): Linear(in_features=16, out_features=32, bias=True)
  (5): Tanh()
  (6): Linear(in_features=32, out_features=64, bias=True)
  (7): Tanh()
  (8): Linear(in_features=64, out_features=1, bias=True)
)
```

In [345]:
```
optmiz = optim.SGD(seq_model.parameters(), lr = 1e-3)
```

In [346]:
```
def training_loop(n_epochs, optimizer, model, loss_fn, t_u_train, t_u_val, t_c_tr
    for epoch in range(1, n_epochs + 1):
        t_p_train = model(t_u_train)
        loss_train = loss_fn(t_p_train, t_c_train)

        t_p_val = model(t_u_val)
        loss_val = loss_fn(t_p_val, t_c_val)

        optimizer.zero_grad()
        loss_train.backward()
        optimizer.step()

        if epoch == 1 or epoch % 100 == 0:
            print(f"Epoch {epoch}, Training loss {loss_train.item():0.4f},"f" Val
```

In [347]:
```
training_loop(
    n_epochs = 200,
    optimizer = optmiz,
    model = seq_model,
    loss_fn = nn.MSELoss(),
    t_u_train = x_train,
    t_u_val = x_test,
    t_c_train = y_train,
    t_c_val = y_test)
```

```
Epoch 1, Training loss 0.0201, Validation loss 0.0374
Epoch 100, Training loss 0.0198, Validation loss 0.0359
Epoch 200, Training loss 0.0196, Validation loss 0.0350
```

In [348]:
```
training_loop(200, optimizer, seq1, nn.MSELoss(), x_train, x_test, y_train, y_tes
```

```
Epoch 1, Training loss 0.1681, Validation loss 0.2316
Epoch 100, Training loss 0.1681, Validation loss 0.2316
Epoch 200, Training loss 0.1681, Validation loss 0.2316
```

In [349]:
```
training_loop(200, optimizer, seq2, nn.MSELoss(), x_train, x_test, y_train, y_tes
```

```
Epoch 1, Training loss 0.0512, Validation loss 0.0881
Epoch 100, Training loss 0.0512, Validation loss 0.0881
Epoch 200, Training loss 0.0512, Validation loss 0.0881
```