In [244…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
from sklearn.model_selection import train_test_split
import tensorflow as tf
from matplotlib import pyplot as plt
import torch.nn as nn
from sklearn.preprocessing import StandardScaler
from collections import OrderedDict
import torch.optim as optim
import warnings
warnings.filterwarnings("ignore")
```

In [207…
```
def loss_fn(t_p,t_c):
    squared_diffs = (t_p-t_c)**2
    return squared_diffs.mean()
```

In [208…
```
def dloss_fn(t_p, t_c_):
    dsq_diffs = 2 * (t_p - t_c) /t_p.size(0)
    return dsq_diffs
```

In [209…
```
def dmodel_dw1(t_u, w1, w2, b):
    return t_u

def dmodel_dw2(t_u, w1, w2, b):
    return t_u
```

In [210…
```
def dmodel_db(t_u, w1, w2, b):
    return 1.0
```

In [211…
```
def grad_fn(t_u, t_c, t_p, w1, w2, b):
    dloss_dtp = dloss_fn(t_p, t_c)
    dloss_dw1 = dloss_dtp * dmodel_dw1(t_u, w1, w2, b)
    dloss_dw2 = dloss_dtp * dmodel_dw2(t_u, w1, w2, b)
    dloss_db = dloss_dtp * dmodel_db(t_u, w1, w2, b)
    return torch.stack([dloss_dw1.sum(), dloss_dw2.sum(), dloss_db.sum()])
```

In [212…
```
#problem 1
```

In [213…
```
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)
t_u = torch.tensor(t_u)
```

In [214…
```
def new_model(t_u, w1, w2, b):
    return w2 * t_u **2 + w1 *t_u + b
```

In [215…
```
w1 = torch.ones(()) #initial W is 1
w2 = torch.ones(()) #initial W is 1
b = torch.zeros(()) #initial b is 0
t_p = new_model(t_u, w1, w2, b)
t_p
```

Out[215]:
```
tensor([1310.1901, 3180.7100, 3445.4399, 6789.5103, 3225.9900, 2440.1101,
        1183.1101,  497.0399, 2390.9600, 3708.5601, 4746.9600])
```

In [216…
```
loss = loss_fn(t_p, t_c)
loss
```

Out[216]:
```
tensor(11709471.)
```

In [217…
```
t_un = 0.1 *t_u
```

In [218…
```
def training_loop(n_epochs, optimizer, params, t_u, t_c):
    for epoch in range(1, n_epochs+1):
        t_p = new_model(t_u, *params)
        loss=loss_fn(t_p,t_c)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

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

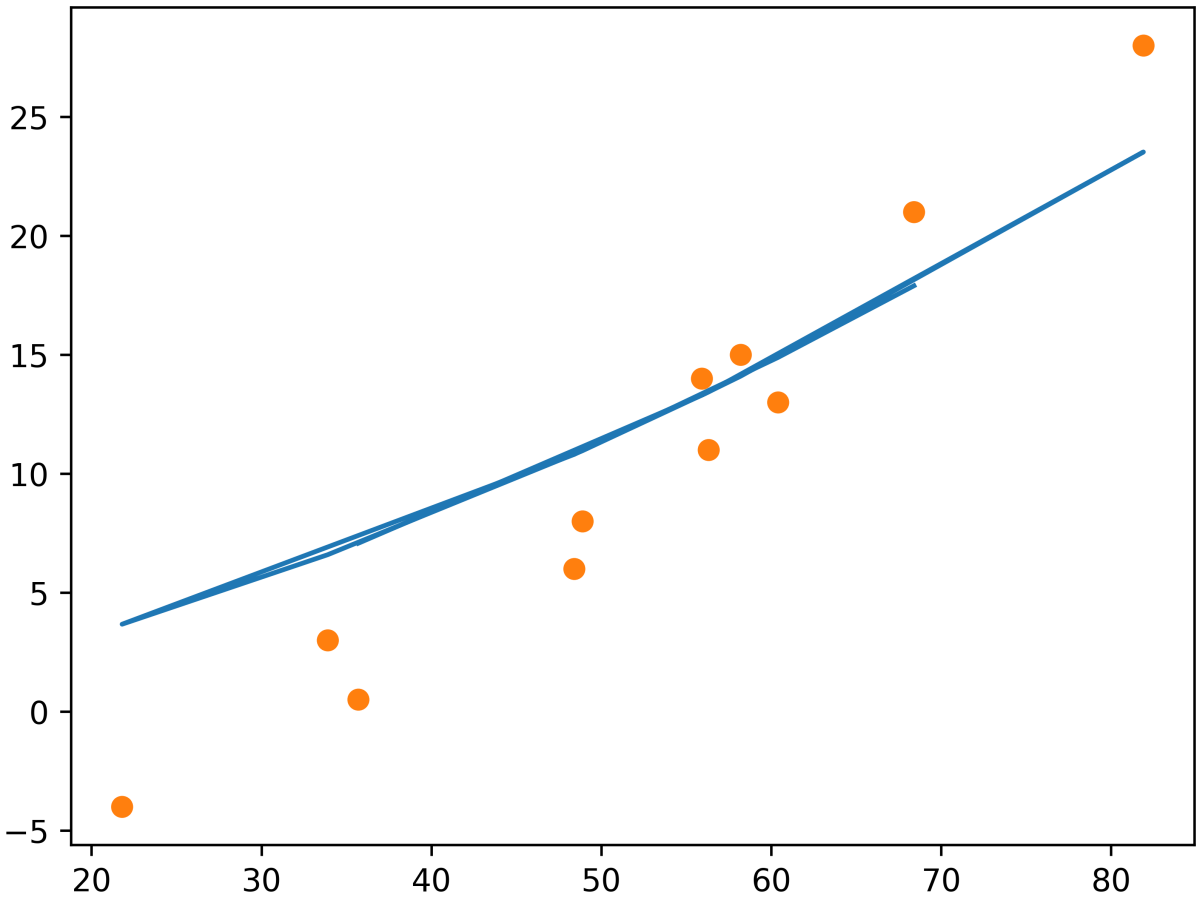
In [219…
```
params = torch.tensor([2.0,1.0,0.0])
params.requires_grad=True
learning_rate = 0.00001
optimizer = optim.SGD([params],lr=learning_rate)

training_loop(
n_epochs = 5000,
optimizer = optimizer,
params = params,
t_u = t_un,
t_c = t_c)

t_p = new_model(t_un, *params)
fig = plt.figure(dpi=600)
plt.xlabel=("Temperature (Fahrenheigh)")
plt.ylabel=("Temperature (Celsuis)")
plt.plot(t_u.numpy(), t_p.detach().numpy())
plt.plot(t_u.numpy(), t_c.numpy(),'o')
```

```
Epoch 500, Loss 21.894091
Epoch 1000, Loss 21.276659
Epoch 1500, Loss 20.679893
Epoch 2000, Loss 20.103096
Epoch 2500, Loss 19.545605
Epoch 3000, Loss 19.006769
Epoch 3500, Loss 18.485960
Epoch 4000, Loss 17.982580
Epoch 4500, Loss 17.496046
Epoch 5000, Loss 17.025795
```

Out[219]:
```
[<matplotlib.lines.Line2D at 0x1b0c3532880>]
```



1.c None linear line fits the data better, hence letting it have a more accurate loss and prediction.

Problem 2

In [220…
```
device = torch.device("cuda:0")
housing = pd.DataFrame(pd.read_csv('Housing.csv'))
housing.head()
```

Out[220]:

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

In [221…
```
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
def binary_map(x):
    return x.map({'yes' : 1, 'no' : 0, 'furnished' : 1, 'semi-furnished' : 0.5, 'unfur
housing[varlist] = housing[varlist].apply(binary_map)
housing.head()
```

Out[221]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheatin
0	13300000	7420	4	2	3	1.0	0.0	0.0	0.
1	12250000	8960	4	4	4	1.0	0.0	0.0	0.
2	12250000	9960	3	2	2	1.0	0.0	1.0	0.
3	12215000	7500	4	2	2	1.0	0.0	1.0	0.
4	11410000	7420	4	1	2	1.0	1.0	1.0	0.

In [222…
```
features = ['price','area', 'bedrooms', 'bathrooms', 'stories', 'parking']
dataset = housing[features]
dataset = StandardScaler().fit_transform(dataset)
dataset
```

Out[222]:
```
array([[ 4.56636513,  1.04672629,  1.40341936,  1.42181174,  1.37821692,
         1.51769249],
       [ 4.00448405,  1.75700953,  1.40341936,  5.40580863,  2.53202371,
         2.67940935],
       [ 4.00448405,  2.21823241,  0.04727831,  1.42181174,  0.22441013,
         1.51769249],
       ...,
       [-1.61432675, -0.70592066, -1.30886273, -0.57018671, -0.92939666,
        -0.80574124],
       [-1.61432675, -1.03338891,  0.04727831, -0.57018671, -0.92939666,
        -0.80574124],
       [-1.61432675, -0.5998394 ,  0.04727831, -0.57018671,  0.22441013,
        -0.80574124]])
```

In [223…
```
raw_y = dataset[:, 0]
raw_x=dataset[:,1:6]
x=torch.from_numpy(raw_x)
y=torch.from_numpy(raw_y)
```

In [224…
```
n_samples = x.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
```

In [225…
```
def housing_model(X,W1,W2,W3,W4,W5,B):
    U=W5*X[:,4] + W4*X[:,3] + W3*X[:,2] + W2*X[:,1] + W1*X[:,0] + B
    return U

def loss_fn(Yp, Y):
    squared_diffs = (Yp - Y)**2
    return squared_diffs.mean()
```

In [226…
```
train_t_u = x[train_indices]
train_t_c = y[train_indices]
val_t_u = x[train_indices]
val_t_c = y[train_indices]
```

```
train_t_un = 0.1 * train_t_u
val_t_un = 0.1*val_t_u
```

In [227…

```
def training_loop(n_epochs, optimizer, params, train_t_u, val_t_u,
                  train_t_c, val_t_c):
    for epoch in range(1,n_epochs+1):
        train_t_p = housing_model(train_t_u, *params)
        train_loss = loss_fn(train_t_p, train_t_c)

        val_t_p = housing_model(val_t_u, *params)
        val_loss = loss_fn(val_t_p, val_t_c)

        with torch.no_grad():
            val_t_p = housing_model(val_t_u, *params)
            val_loss = loss_fn(val_t_p, val_t_c)
            assert val_loss.requires_grad == False

        optimizer.zero_grad()
        train_loss.backward()
        optimizer.step()

        if epoch <= 3 or epoch % 500 == 0:
            print (f"epoch {epoch}, Training loss {train_loss.item():.4f}," f" Validat

    return params
```

In [228…

```
def training_SGD(lr):
    params=torch.tensor([1.0,1.0,1.0,1.0,1.0,0.0],requires_grad=True)
    learning_rate=lr
    optimizer=optim.SGD([params],lr=learning_rate)

    training_loop(
        n_epochs = 5000,
        optimizer = optimizer,
        params = params,
        train_t_u = train_t_un,
        val_t_u = val_t_un,
        train_t_c = train_t_c,
        val_t_c = val_t_c)
```

In [229…

```
training_SGD(0.0001)
```

```
epoch 1, Training loss 0.6332, Validation loss 0.6332
epoch 2, Training loss 0.6332, Validation loss 0.6332
epoch 3, Training loss 0.6332, Validation loss 0.6332
epoch 500, Training loss 0.6325, Validation loss 0.6325
epoch 1000, Training loss 0.6318, Validation loss 0.6318
epoch 1500, Training loss 0.6311, Validation loss 0.6311
epoch 2000, Training loss 0.6304, Validation loss 0.6304
epoch 2500, Training loss 0.6298, Validation loss 0.6298
epoch 3000, Training loss 0.6291, Validation loss 0.6291
epoch 3500, Training loss 0.6285, Validation loss 0.6285
epoch 4000, Training loss 0.6279, Validation loss 0.6279
epoch 4500, Training loss 0.6272, Validation loss 0.6272
epoch 5000, Training loss 0.6266, Validation loss 0.6266
```

In [230…

```
training_SGD(0.001)
```

```
epoch 1, Training loss 0.6332, Validation loss 0.6332
epoch 2, Training loss 0.6332, Validation loss 0.6332
epoch 3, Training loss 0.6332, Validation loss 0.6332
epoch 500, Training loss 0.6266, Validation loss 0.6266
epoch 1000, Training loss 0.6205, Validation loss 0.6205
epoch 1500, Training loss 0.6147, Validation loss 0.6147
epoch 2000, Training loss 0.6090, Validation loss 0.6090
epoch 2500, Training loss 0.6036, Validation loss 0.6036
epoch 3000, Training loss 0.5983, Validation loss 0.5983
epoch 3500, Training loss 0.5932, Validation loss 0.5932
epoch 4000, Training loss 0.5883, Validation loss 0.5883
epoch 4500, Training loss 0.5835, Validation loss 0.5835
epoch 5000, Training loss 0.5789, Validation loss 0.5789
```

In [231…

```
training_SGD(0.01)
```

```
epoch 1, Training loss 0.6332, Validation loss 0.6332
epoch 2, Training loss 0.6331, Validation loss 0.6331
epoch 3, Training loss 0.6329, Validation loss 0.6329
epoch 500, Training loss 0.5789, Validation loss 0.5789
epoch 1000, Training loss 0.5402, Validation loss 0.5402
epoch 1500, Training loss 0.5123, Validation loss 0.5123
epoch 2000, Training loss 0.4920, Validation loss 0.4920
epoch 2500, Training loss 0.4771, Validation loss 0.4771
epoch 3000, Training loss 0.4661, Validation loss 0.4661
epoch 3500, Training loss 0.4580, Validation loss 0.4580
epoch 4000, Training loss 0.4518, Validation loss 0.4518
epoch 4500, Training loss 0.4471, Validation loss 0.4471
epoch 5000, Training loss 0.4435, Validation loss 0.4435
```

In [232…

```
training_SGD(0.1)
```

```
epoch 1, Training loss 0.6332, Validation loss 0.6332
epoch 2, Training loss 0.6318, Validation loss 0.6318
epoch 3, Training loss 0.6304, Validation loss 0.6304
epoch 500, Training loss 0.4435, Validation loss 0.4435
epoch 1000, Training loss 0.4306, Validation loss 0.4306
epoch 1500, Training loss 0.4284, Validation loss 0.4284
epoch 2000, Training loss 0.4278, Validation loss 0.4278
epoch 2500, Training loss 0.4276, Validation loss 0.4276
epoch 3000, Training loss 0.4275, Validation loss 0.4275
epoch 3500, Training loss 0.4275, Validation loss 0.4275
epoch 4000, Training loss 0.4275, Validation loss 0.4275
epoch 4500, Training loss 0.4275, Validation loss 0.4275
epoch 5000, Training loss 0.4275, Validation loss 0.4275
```

In [233…

```
def training_Adam(lr):
    params=torch.tensor([1.0,1.0,1.0,1.0,1.0,0.0],requires_grad=True)
    learning_rate=lr
    optimizer=optim.Adam([params],lr=learning_rate)

    training_loop(
        n_epochs = 5000,
        optimizer = optimizer,
        params = params,
        train_t_u = train_t_un,
        val_t_u = val_t_un,
        train_t_c = train_t_c,
        val_t_c = val_t_c)
```

In [234…
```
training_Adam(0.1)
```

```
epoch 1, Training loss 0.6332, Validation loss 0.6332
epoch 2, Training loss 0.6161, Validation loss 0.6161
epoch 3, Training loss 0.5890, Validation loss 0.5890
epoch 500, Training loss 0.4275, Validation loss 0.4275
epoch 1000, Training loss 0.4275, Validation loss 0.4275
epoch 1500, Training loss 0.4275, Validation loss 0.4275
epoch 2000, Training loss 0.4275, Validation loss 0.4275
epoch 2500, Training loss 0.4275, Validation loss 0.4275
epoch 3000, Training loss 0.4275, Validation loss 0.4275
epoch 3500, Training loss 0.4275, Validation loss 0.4275
epoch 4000, Training loss 0.4275, Validation loss 0.4275
epoch 4500, Training loss 0.4275, Validation loss 0.4275
epoch 5000, Training loss 0.4275, Validation loss 0.4275
```

The best linear model is adam, due to its lower loss and it reaches this loss faster.

Problem 3

In [235…
```
t_u_train = train_t_u
t_c_train = train_t_c

t_u_val = val_t_u
t_c_val = val_t_c

t_un_train = train_t_un
t_un_val = val_t_un
```

In [236…
```
linear_model = nn.Linear(1 , 1)

optimizer = optim.SGD(
    linear_model.parameters(), # <2>
    lr=1e-2)

seq_model = nn.Sequential(
            nn.Linear(5, 8), # <1>
            nn.Tanh(),
            nn.Linear(8, 1)) # <2>

seq_model = seq_model.double()
```

In [237…
```
[param.shape for param in seq_model.parameters()]

for name, param in seq_model.named_parameters():
    print(name, param.shape)
```

```
0.weight torch.Size([8, 5])
0.bias torch.Size([8])
2.weight torch.Size([1, 8])
2.bias torch.Size([1])
```

In [238…
```
def training_loop(n_epochs, optimizer, model, loss_fn, t_u_train, t_u_val,
                  t_c_train, t_c_val):
    for epoch in range(1, n_epochs + 1):
        t_p_train = model(t_u_train) # <1>
        loss_train = loss_fn(t_p_train, t_c_train)

        t_p_val = model(t_u_val) # <1>
```

```
        loss_val = loss_fn(t_p_val, t_c_val)

        optimizer.zero_grad()
        loss_train.backward() # <2>
        optimizer.step()

        if epoch == 1 or epoch % 50 == 0:
            print(f"Epoch {epoch}, Training loss {loss_train.item():.4f},"
                  f" Validation loss {loss_val.item():.4f}")

def loss_fn(Yp, Y):
    squared_diffs = (Yp - Y)**2
    return squared_diffs.mean()
```

In [245…

```
optimizer = optim.SGD(seq_model.parameters(), lr=1e-3) # <1>

training_loop(
    n_epochs = 200,
    optimizer = optimizer,
    model = seq_model,
    loss_fn = nn.MSELoss(),
    t_u_train = t_un_train,
    t_u_val = t_un_val,
    t_c_train = t_c_train,
    t_c_val = t_c_val)
```

```
Epoch 1, Training loss 1.0005, Validation loss 1.0005
Epoch 50, Training loss 0.9898, Validation loss 0.9898
Epoch 100, Training loss 0.9821, Validation loss 0.9821
Epoch 150, Training loss 0.9767, Validation loss 0.9767
Epoch 200, Training loss 0.9729, Validation loss 0.9729
```

As epochs increase te loss gets less. This is due to a longer time

Problem 3b

In [240…

```
t_u_train = train_t_u
t_c_train = train_t_c

t_u_val = val_t_u
t_c_val = val_t_c

t_un_train = train_t_un
t_un_val = val_t_un
```

In [241…

```
linear_model = nn.Linear(1 , 1)

optimizer = optim.SGD(
    linear_model.parameters(), # <2>
    lr=1e-2)

seq_model = nn.Sequential(
            nn.Linear(5, 8), # <1>
            nn.Tanh(),
            nn.Linear(8, 4), # <2>
            nn.Tanh(),
            nn.Linear(4, 2), # <3>
            nn.Tanh(),
            nn.Linear(2, 1))
```

```
seq_model = seq_model.double()
```

In [242…
```
[param.shape for param in seq_model.parameters()]

for name, param in seq_model.named_parameters():
    print(name, param.shape)
```

```
0.weight torch.Size([8, 5])
0.bias torch.Size([8])
2.weight torch.Size([4, 8])
2.bias torch.Size([4])
4.weight torch.Size([2, 4])
4.bias torch.Size([2])
6.weight torch.Size([1, 2])
6.bias torch.Size([1])
```

In [243…
```
optimizer = optim.SGD(seq_model.parameters(), lr=1e-3) # <1>

training_loop(
    n_epochs = 200,
    optimizer = optimizer,
    model = seq_model,
    loss_fn = nn.MSELoss(),
    t_u_train = t_un_train,
    t_u_val = t_un_val,
    t_c_train = t_c_train,
    t_c_val = t_c_val)
```

```
Epoch 1, Training loss 1.1143, Validation loss 1.1143
Epoch 50, Training loss 1.0704, Validation loss 1.0704
Epoch 100, Training loss 1.0388, Validation loss 1.0388
Epoch 150, Training loss 1.0165, Validation loss 1.0165
Epoch 200, Training loss 1.0008, Validation loss 1.0008
```

Adding more layers to the network allowed for a better loss values, this is due to how to network processes the inputs.

In [ ]:
```
```