Exercise: Weighting

Using the sample code (gradientDescent.py, linear Regression.py), find a set of test data (from https://data.gov.tw), define an equation with at least two parameters, utilize gradient descent to find the best parameters.

Submission requirements:

1. source code (weighting.py)
2. PDF documents
   Explaining your data and strategies.
   Show the outputs
3. Upload to e-learning before 3/22 14:10
4. Schedule a demo with the course TA (賴佳嬪)

Data:

Resource: 台灣電力公司_各年度再生能源別裝置容量

https://data.gov.tw/dataset/29933

	A	B	C	D	E
1	年度	風力(萬瓩)	太陽電(萬瓩	其他(含水力	總計(萬瓩)
2	97	24.6	0.25	193.79	218.63
3	98	34.69	0.55	193.69	228.93
4	99	47.15	0.7	197.74	245.59
5	100	51.83	4.44	204.07	260.33
6	101	58.5	13.25	208.13	279.87
7	102	60.98	28.3	208.13	297.4
8	103	63.28	43.97	208.13	315.38
9	104	64.23	66.6	208.23	339.06
10	105	67.77	93.07	209.04	369.88
11	106	69.03	138.33	209.07	416.43
12	107	69.96	234.22	209.3	513.48
13	108	72.08	355.87	209.44	637.39
14	109	91.58	556.51	210.15	858.25
15	110	109.21	720.61	211.06	1,040.88
16	111	160.13	924.47	211.03	1,295.63
17	112	266.15	1,167.24	212.46	1,645.86

The number of rows: 17

The number of columns: 5

First row: unit

Other rows: Each kind of energy volume

The data I use: Column B, Column C, Column D and Column E

Import CSV

```python
import csv

def remove_comma(n):
    while "," in n:
        n = n.replace(",", "")

    return float(n)

# import data
y, x1, x2, x3 = [], [], [], []
with open("data.csv", encoding = "utf-8") as f:
    reader = list(csv.reader(f))
    print(reader[0])

    for row in reader[1: ]:
        y.append(remove_comma(row[4]))
        x1.append(remove_comma(row[3]))
        x2.append(remove_comma(row[2]))
        x3.append(remove_comma(row[1]))
```

Import data

Define a function to remove comma from input string to avoid error when converting string into float. Use csv package to import data from csv file row by row.

Gradient Descent

1. Preprocess

```
def remove_comma(n):
    while "," in n:
        n = n.replace(",", "")

    return float(n)

# import data
y, x1, x2, x3 = [], [], [], []
with open("data.csv", encoding = "utf-8") as f:
    reader = list(csv.reader(f))
    print(reader[0])

    for row in reader[1: ]:
        y.append(remove_comma(row[4]))
        x1.append(remove_comma(row[3]))
        x2.append(remove_comma(row[2]))
        x3.append(remove_comma(row[1]))

# convert data into tensor
y = torch.tensor(y, dtype = torch.float32)
x1 = torch.tensor(x1, dtype = torch.float32)
x2 = torch.tensor(x2, dtype = torch.float32)
x3 = torch.tensor(x3, dtype = torch.float32)

# initialize the weights
w1 = torch.randn(1, requires_grad = True)
w2 = torch.randn(1, requires_grad = True)
w3 = torch.randn(1, requires_grad = True)
w4 = torch.randn(1, requires_grad = True)

# set learning rate and number of epochs
learning_rate = 1e-6
epochs = 10000
```

Import data from csv file then convert the data type from string to float. After getting each column of data, convert them into tensor. Then, initialize the weights, learning rate and number of epochs.

2. Train

```
# train
for epoch in range(epochs):
    # predict the housing price
    predictions = w1 * x1 + w2 * x2 + w3 * x3 + w4

    # calculate the loss (MSE)
    loss = torch.mean((predictions - y)**2)

    # back propagation
    loss.backward()

    # update the weights
    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w3 -= learning_rate * w1.grad
        w4 -= learning_rate * w1.grad

        # return the gradients into zero
        w1.grad.zero_()
        w2.grad.zero_()
        w3.grad.zero_()
        w4.grad.zero_()

    if epoch == 0 or epoch % 1000 == 999:
        print(f"Epoch {epoch + 1}, Loss: {loss.item()}")

print(f"weighting: w1 = {w1.item()}, w2 = {w2.item()}, w3 = {w3.item()}, w4 = {w4.item()}")
```

During each time of training, predict the result first. Use MSE to calculate the loss. Then, update the weights and their gradients by learning rate. Finally, display the result of the weights of each input.

3. Output

```
Epoch 1, Loss: 2401775.25
Epoch 1000, Loss: 490.47406005859375
Epoch 2000, Loss: 490.47406005859375
Epoch 3000, Loss: 490.47406005859375
Epoch 4000, Loss: 490.47406005859375
Epoch 5000, Loss: 490.47406005859375
Epoch 6000, Loss: 490.47406005859375
Epoch 7000, Loss: 490.47406005859375
Epoch 8000, Loss: 490.47406005859375
Epoch 9000, Loss: 490.47406005859375
Epoch 10000, Loss: 490.47406005859375
weighting: w1 = 0.7809123396873474, w2 = 0.854692816734314, w3 = 2.0238184928894043, w4 = 0.752937376499176
```

The loss of each epoch and the weights of each input. Thus, the equation will be like below.

$$總計 = 風力 * 0.781 + 太陽電 * 0.855 + 其他 * 0.753$$

Linear Regression

1. Preprocess

```python
# import data
def remove_comma(n):
    while "," in n:
        n = n.replace(",", "")

    return float(n)

# import data
y, x1, x2, x3 = [], [], [], []
with open("data.csv", encoding = "utf-8") as f:
    reader = list(csv.reader(f))
    print(reader[0])

    for row in reader[1: ]:
        y.append([remove_comma(row[4])])
        x1.append([remove_comma(row[3])])
        x2.append([remove_comma(row[2])])
        x3.append([remove_comma(row[1])])

# convert data into tensor
y = torch.tensor(y, dtype = torch.float32)
x1 = torch.tensor(x1, dtype = torch.float32)
x2 = torch.tensor(x2, dtype = torch.float32)
x3 = torch.tensor(x3, dtype = torch.float32)

# initialize the weights
w1 = torch.randn(1, requires_grad = True)
w2 = torch.randn(1, requires_grad = True)
w3 = torch.randn(1, requires_grad = True)
w4 = torch.randn(1, requires_grad = True)

# set learning rate and number of epochs
learning_rate = 1e-8
epochs = 100
losses = []
```

Import data from csv file then convert the data type from string to float. After getting each column of data, convert them into tensor. Then, initialize the weights, learning rate and number of epochs.

2. Train

```python
# train
for epoch in range(epochs):
    y_pred = torch.matmul(x1, w1) + torch.matmul(x2, w2) + torch.matmul(x3, w3) + w4
    loss = torch.mean((y_pred - y)**2)

    loss.backward()
    losses.append(loss.item())

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w3 -= learning_rate * w2.grad
        w4 -= learning_rate * w2.grad

        w1.grad.zero_()
        w2.grad.zero_()
        w3.grad.zero_()
        w4.grad.zero_()

    if epoch in range(10) or epoch % 10 == 9:
        print(f"Epoch {epoch + 1}, Loss: {loss.item()}")

print(f"Trained weights: w1 = {w1.item()}, w2 = {w2.item()}, w3 = {w3.item()}, w4 = {w4.item()}")
```

During each time of training, predict the result by tensor multiplication first. Use MSE to calculate the loss. Then, update the weights and their gradients by learning rate. Finally, display the result of the weights of each input.

3. Output

```
Epoch 1, Loss: 612099.125
Epoch 2, Loss: 610289.125
Epoch 3, Loss: 608496.25
Epoch 4, Loss: 606720.375
Epoch 5, Loss: 604961.25
Epoch 6, Loss: 603218.6875
Epoch 7, Loss: 601492.5625
Epoch 8, Loss: 599782.5625
Epoch 9, Loss: 598088.625
Epoch 10, Loss: 596410.75
Epoch 20, Loss: 580468.3125
Epoch 30, Loss: 565939.125
Epoch 40, Loss: 552679.5
Epoch 50, Loss: 540560.625
Epoch 60, Loss: 529466.625
Epoch 70, Loss: 519293.46875
Epoch 80, Loss: 509947.875
Epoch 90, Loss: 501345.875
Epoch 100, Loss: 493412.21875
Trained weights: w1 = -0.47534555196762085, w2 = 1.302370309829712, w3 = 0.9568878412246704, w4 = 1.0675451755523682
```

The loss of each epoch and the weights of each input. Thus, the equation will be like below.

$$總計 = 風力 * (-0.475) + 太陽電 * 1.302 + 其他 * 0.957$$