AI VIET NAM – COURSE 2023

Module 2 - Exercise 4

Numpy và Vectorization

Ngày 18 tháng 6 năm 2023

Giới thiệu về bài tập: Ở phần bài tập này các bạn sẽ được ôn tập cách sử dụng thư viện numpy để thao tác trên 1D và 2D array. Cũng như cách cài đặt giải thuật linear regression cho bài toán advertising theo cách vectorization dùng stochastic gradient descent, m samples (mini-batch gradient descent), and N samples (batch gradient descent).

Bài tập 1 (Làm quen với numpy):

Question 1: Câu nào sau đây là đúng để tạo mảng 1 chiều từ 0 đến 9

```
a) import numpy as np
   arr = np.arange(0, 10, 1)

b) import numpy as np
   arr = np.arange(1, 10, 1)

c) import numpy as np
   arr = np.arange(0, 9, 1)

d) import numpy as np
   arr = np.arange(1, 11, 1)
```

Question 2: Cách tạo một mảng boolean 3x3 với tất cả giá trị là True

```
a) arr = np.ones((3,3)) > 0

b) arr = np.ones((3,3), dtype=bool)

c) arr = np.full((3,3), fill_value=True, dtype=bool)

d) a,b and c
```

Question 3: Kết quả của đoạn code sau đây:

```
import numpy as np
arr = np.arange(0,10)
print(arr[arr%2 == 1])
```

```
a) [1 3 5 7 9]

b) [1 2 3 4 5]

c) [2 4 6 8 9]

d) [1 7 5 7 9]
```

Question 4: Kết quả của đoạn code sau đây:

```
import numpy as np
arr = np.arange(0,10)
arr[arr%2 ==1] = -1
print(arr)
```

a) [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1]

b) [0 -1 2 -1 4 -1 6 -1 8 -1]

c) [-1 0 -1 -0 -1 0 -1 0 -1 0]

d) [-1 1 -1 1 -1 1 -1 1 -1 1]

Question 5: Kết quả của đoạn code sau đây:

```
import numpy as np
arr = np.arange(10)
arr_2d = arr.reshape(2,-1)
print(arr_2d)
```

a) [[0 1 2 3 4 5 6 7 8 9]]

b) [[0 1 2 3 4]
   [5 6 7 8 9]]

c) [[0 1 2 3 4]]

d) [[0 1 2 3]
   [5 6 7 8]]

Question 6: Kết quả của đoạn code sau đây:

```
import numpy as np
arr1 = np.arange(10).reshape(2,-1)
arr2 = np.repeat(1,10).reshape(2,-1)
c = np.concatenate([arr1,arr2],axis =0)
print("Result: \n", c)
```

a) [[0 1 2 3 4]
    [5 6 7 8 9]
    [1 1 1 1 1]
    [1 1 1 1 1]]

b) [[0 1 2 3 4]
   [5 6 7 8 9]]

c) [[0 1 2 3 4]
    [5 6 7 8 9]
    [1 1 1 1 1]
    [5 6 7 8 9]]

d) [[0 1 2 3 4]]

Question 7: Kết quả của đoạn code sau đây:

```
import numpy as np
arr1 = np.arange(10).reshape(2,-1)
arr2 = np.repeat(1,10).reshape(2,-1)
```

```
c = np.concatenate([arr1,arr2],axis =1)
print("C = ", c)
```

a)
```
[[0 2 2 2 2 1 1 1 1 1]
 [5 6 7 8 9 1 1 1 1 1]]
```

b)
```
[[1 1 1 1 1 1 1 1 1 1]
 [5 6 7 8 9 5 6 7 8 9]]
```

c)
```
[[0 1 2 3 4 1 1 1 1 1]
 [5 6 7 8 9 1 1 1 1 1]]
```

d)
```
[[0 2 2 2 2 1 1 1 1 1]
 [2 2 2 2 1 1 1 1 1 1]]
```

Question 8: Kết quả của đoạn code sau đây:

```
import numpy as np
arr = np.array([1,2,3])
print(np.repeat(arr,3))
print(np.tile(arr,3))
```

a)
```
[1 1 1 2 2 2 3 3 3]
[1 2 3 1 2 3 1 2 3]
```

b)
```
[1 1 1 1 1 1 1 1 1]
[2 2 2 2 2 2 2 2 2]
```

c)
```
[1 1 1 1 1 1 1 1 1]
[3 3 3 3 3 3 3 3 3]
```

d)
```
[2 2 2 2 2 2 2 2 2]
[3 3 3 3 3 3 3 3 3]
```

Question 9: Kết quả của đoạn code sau đây:

```
import numpy as np
a = np.array([2,6,1,9,10,3,27])
index = np.where((a>=5)&(a<=10))
print("result", a[index])
```

a)
```
[ 3  9 10]
```

b)
```
[ 1  9 10]
```

c)
```
[ 6  9 10]
```

d)
```
[ 2  1 10]
```

Question 10: Kết quả của đoạn code sau đây:

```
import numpy as np

def maxx(x,y):
    if x >= y :
        return x
    else:
        return y

a = np.array([5,7,9,8,6,4,5])
```

```
b = np.array([6,3,4,8,9,7,1])

pair_max = np.vectorize(maxx,otypes=[float])
print(pair_max(a,b))
```

```
a)  [5, 7, 9, 8, 6, 4, 5]

b)  [6, 3, 4, 8, 9, 7, 1]

c)  [6. 8. 9. 8. 7. 7. 5.]

d)  [6. 7. 9. 8. 9. 7. 5.]
```

Question 11: Kết quả của đoạn code sau đây:

```
import numpy as np

a = np.array([5,7,9,8,6,4,5])
b = np.array([6,3,4,8,9,7,1])

print("Result",np.where(a<b, b, a))
```

```
a)  [6. 7. 9. 8. 9. 7. 5.]

b)  [5, 7, 9, 8, 6, 4, 5]

c)  [6, 3, 4, 8, 9, 7, 1]

d)  [6. 8. 9. 8. 7. 7. 5.]
```

Bài tập 2 (Làm quen với Ma trận):

Matrix Properties: Các bạn chứng minh các tính chất của ma trận sau:

1. $\mathbf{AB} \neq \mathbf{BA}$, ($\mathbf{A} \in \mathbf{R}^{m \times m}$ and $\mathbf{B} \in \mathbf{R}^{m \times m}$) ($\mathbf{A}$ và $\mathbf{B}$ là ma trận vuông)
2. $(\mathbf{A}+\mathbf{B})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}} + \mathbf{B}^{\mathrm{T}}$, ($\mathbf{A} \in \mathbf{R}^{m \times n}$ and $\mathbf{B} \in \mathbf{R}^{m \times n}$)
3. $(\mathbf{AB})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$, ($\mathbf{A} \in \mathbf{R}^{m \times n}$ and $\mathbf{B} \in \mathbf{R}^{n \times k}$)

Example:

- $\mathbf{A}+\mathbf{B} = \mathbf{B}+\mathbf{A}$, ($\mathbf{A} \in \mathbf{R}^{m \times n}$ and $\mathbf{B} \in \mathbf{R}^{m \times n}$)

$$\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \dots & \dots & \dots \\ b_{m1} & \dots & b_{mn} \end{bmatrix}$$

$$\mathbf{A}+\mathbf{B} = \begin{bmatrix} (a_{11}+b_{11}) & \dots & (a_{1n}+b_{1n}) \\ \dots & \dots & \dots \\ (a_{m1}+b_{m1}) & \dots & (a_{mn}+b_{mn}) \end{bmatrix} = \begin{bmatrix} (b_{11}+a_{11}) & \dots & (b_{1n}+a_{1n}) \\ \dots & \dots & \dots \\ (b_{m1}+a_{m1}) & \dots & (b_{mn}+a_{mn}) \end{bmatrix} = \mathbf{B}+\mathbf{A}$$

Bài tập 3 (Hiện thực giải thuật linear regression theo phương pháp vectorization):

Linear Regression: Các bạn thực hiện train linear regresion model trên tập data advertising.csv theo các yêu cầu sau. Các bạn sẽ dựa trên 3 thông tin đầu vào là TV, Radio, Newspaper để dự đoán Sale.

Giới thiệu về tập data: Data có 200 samples (rows), gồm 4 cột thông tin Tv, Radio, Newspaper, và Sales. Đề bài yêu cầu dùng thông tin ở 3 cột đầu tiên (Tv, Radio, Newspaper) để dự đoán được cột cuối cùng (Sale) dùng linear regression model.

Để chuẩn hoá data đầu vào, AIVN cung cấp trước cho các bạn function đọc dữ liệu và chuẩn hoá **mean_normalization(X)** như bên dưới:

```
# dataset
data = genfromtxt('advertising.csv', delimiter=',', skip_header=1)
N = data.shape[0]
X = data[:,:3]
y = data[:,3:]

# Normalize input data by using mean normalizaton
def mean_normalization(X):
  N = len(X)
  maxi = np.max(X)
  mini = np.min(X)
  avg = np.mean(X)
  X = (X-avg) / (maxi-mini)
  X_b = np.c_[np.ones((N, 1)), X]
  return X_b, maxi, mini, avg

X_b, maxi, mini, avg = mean_normalization(X)
```

Yêu cầu của bài tập này là các bạn lần lượt hiện thực lại giải thuật linear regression để dự doán Sales dựa vào các yêu cầu sau:

1. Hoàn thành fucntion **stochastic_gradient_descent()** để huấn luyện data sử dụng Stochastic Gradient Descent. Lưu ý các bạn cần tận dụng tối đa vectorization để hoàn thiện bài tập này.

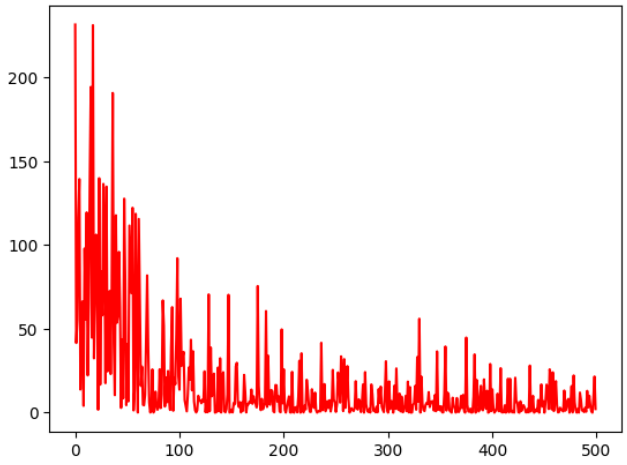
   - **input**: (4 inputs) X_b, y, n_epochs, learning_rate
   - **output**: thetas_path, losses

```
    def stochastic_gradient_descent(X_b, y, n_epochs=50, learning_rate=0.00001):

    # thetas = np.random.randn(4, 1) # uncomment this line for real application
    thetas = np.asarray([[1.16270837], [-0.81960489],  [1.39501033],
    [0.29763545]])

    thetas_path = [thetas]
    losses = []

    for epoch in range(n_epochs):
        for i in range(N):
            # select random number in N
            # random_index = np.random.randint(N) #In real application, you
    should use this code
            random_index = i # This code is used for this assignment only

            xi = X_b[random_index:random_index+1]
            yi = y[random_index:random_index+1]

            # Compute output
            *******Your code here ********

            # Compute loss li
            *******Your code here ********

            # Compute gradient for loss
```

```
            *******Your code here ********

            # Compute gradient
            *******Your code here ********

            # update theta
             *******Your code here ********

            # logging
             *******Your code here ********

    return thetas_path, losses

```

Hình 1 là kết quả sau khi thực thi đoạn code sau:

```
sgd_theta, losses = stochastic_gradient_descent(X_b, y, n_epochs=50,
   learning_rate=0.01 )

x_axis = list(range(500))
plt.plot(x_axis,losses[:500], color="r")
plt.show()
```



Hình 1: Kết quả loss values sử dụng Stochastic Gradient Descent

Question 12: Kết quả của đoạn code sau đây:

```
sgd_theta, losses = stochastic_gradient_descent(X_b, y, n_epochs=1, learning_rate
    =0.01 )
print(np.sum(losses))
```

```
a)  7754.64

b)  6754.64

c)  8754.64

d)  9754.64
```

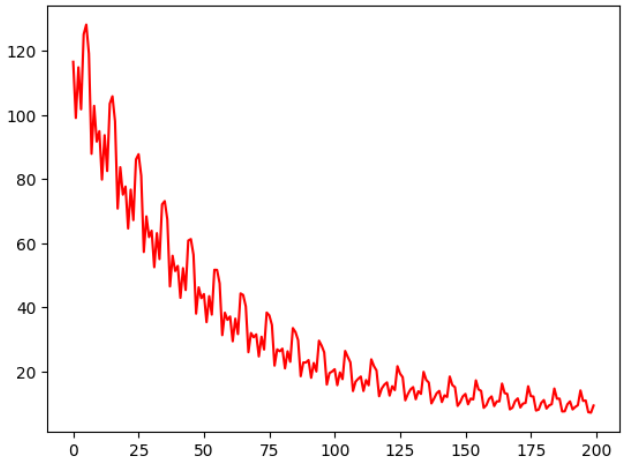
2. Hoàn thành fucntion **mini_batch_gradient_descent()** để huấn luyện data sử dụng Mini-batch Gradient Descent. Lưu ý các bạn cần tận dụng tối đa vectorization để hoàn thiện bài tập này.

- **input**: (5 inputs) X_b, y, n_epochs, minibatch_size, learning_rate
- **output**: thetas_path, losses

```
    def mini_batch_gradient_descent(X_b, y, n_epochs=50,  minibatch_size = 20, learning_rate=0.01):

    # thetas = np.random.randn(4, 1)
    thetas = np.asarray([[1.16270837], [-0.81960489],  [1.39501033], [0.29763545]])

    thetas_path = [thetas]
    losses = []

    for epoch in range(n_epochs):
        # shuffled_indices = np.random.permutation(N) # uncomment this code for real application

        shuffled_indices = np.asarray([21, 144, 17, 107, 37, 115, 167, 31, 3, 132, 179, 155, 36, 191, 182, 170, 27, 35, 162, 25, 28, 73, 172, 152, 102, 16, 185, 11, 1, 34, 177, 29, 96, 22, 76, 196, 6, 128, 114, 117, 111, 43, 57, 126, 165, 78, 151, 104, 110, 53, 181, 113, 173, 75, 23, 161, 85, 94, 18, 148, 190, 169, 149, 79, 138, 20, 108, 137, 93, 192, 198, 153, 4, 45, 164, 26, 8, 131, 77, 80, 130, 127, 125, 61, 10, 175, 143, 87, 33, 50, 54, 97, 9, 84, 188, 139, 195, 72, 64, 194, 44, 109, 112, 60, 86, 90, 140, 171, 59, 199, 105, 41, 147, 92, 52, 124, 71, 197, 163, 98, 189, 103, 51, 39, 180, 74, 145, 118, 38, 47, 174, 100, 184, 183, 160, 69, 91, 82, 42, 89, 81, 186, 136, 63, 157, 46, 67, 129, 120, 116, 32, 19, 187, 70, 141, 146, 15, 58, 119, 12, 95, 0, 40, 83, 24, 168, 150, 178, 49, 159, 7, 193, 48, 30, 14, 121, 5, 142, 65, 176, 101, 55, 133, 13, 106, 66, 99, 68, 135, 158, 88, 62, 166, 156, 2, 134, 56, 123, 122, 154])


        X_b_shuffled = X_b[shuffled_indices]
        y_shuffled = y[shuffled_indices]

        for i in range(0, N, minibatch_size):
            xi = X_b_shuffled[i:i+minibatch_size]
            yi = y_shuffled[i:i+minibatch_size]

            # compute output
            *******Your code here ********

            # compute loss
            *******Your code here ********

            # compute derivative of loss
            *******Your code here ********

            # compute derivative of parameters
            *******Your code here ********

            # update parameters
            thetas = thetas - learning_rate*gradients
            thetas_path.append(thetas)

            loss_mean = np.sum(loss)/minibatch_size
            losses.append(loss_mean)

    return thetas_path, losses

```

Hình 2 là kết quả sau khi thực thi đoạn code sau:

```
mbgd_thetas, losses = mini_batch_gradient_descent(X_b, y, n_epochs=50,
    minibatch_size = 20, learning_rate=0.01)

x_axis = list(range(200))
plt.plot(x_axis,losses[:200], color="r")
plt.show()
```



Hình 2: Kết quả loss values sử dụng Mini batch Gradient Descent

Question 13: Kết quả của đoạn code sau đây:

```
mbgd_thetas, losses = mini_batch_gradient_descent(X_b, y, n_epochs=50,
    minibatch_size = 20, learning_rate=0.01)
print(round(sum(losses),2))
```

```
a)  7865.65

b)  6865.65

c)  5865.65

d)  8865.65
```

3. Hoàn thành fucntion **batch_gradient_descent()** để huấn luyện data sử dụng batch Gradient Descent. Lưu ý các bạn cần tận dụng tối đa vectorization để hoàn thiện bài tập này.

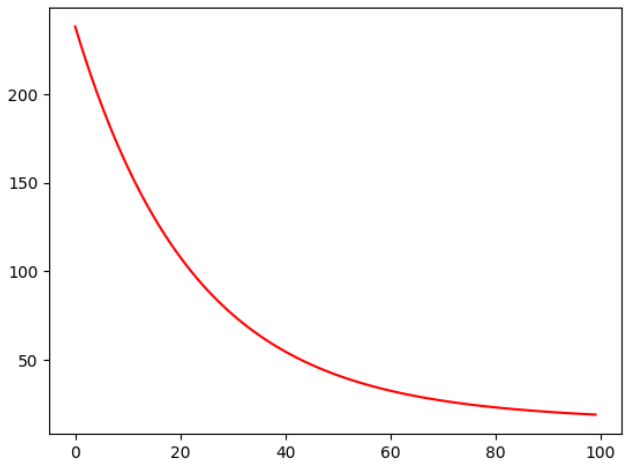
   - **input**: (4 inputs) X_b, y, n_epochs, learning_rate
   - **output**: thetas_path, losses

```
  def batch_gradient_descent(X_b, y, n_epochs=100, learning_rate=0.01):

    # thetas = np.random.randn(4, 1) # uncomment this line for real application
    thetas = np.asarray([[1.16270837], [-0.81960489],  [1.39501033],
    [0.29763545]])

    thetas_path = [thetas]
    losses = []

    for i in range(n_epochs):
```

```
        # compute output
        *******Your code here ********


        # Compute loss
        *******Your code here ********


        # Compute losss's derivative
        *******Your code here ********


        # computer parameters' derivative
        *******Your code here ********

        # Update parameters
        thetas = thetas - learning_rate*gradients
        thetas_path.append(thetas)

        mean_loss = np.sum(loss)/N
        losses.append(mean_loss)

    return thetas_path, losses
```

Hình 3 là kết quả sau khi thực thi đoạn code sau:

```
bgd_thetas, losses = batch_gradient_descent(X_b, y, n_epochs=100, learning_rate
    =0.01)

x_axis = list(range(100))
plt.plot(x_axis,losses[:100], color="r")
plt.show()
```



Hình 3: Kết quả loss values sử dụng batch Gradient Descent

Question 14: Kết quả của đoạn code sau đây:

```
bgd_thetas, losses = batch_gradient_descent(X_b, y, n_epochs=100, learning_rate
    =0.01)
print(round(sum(losses),2))
```

a) 7716.46

```
b)  8716.46

c)  6716.46

d)  5716.46
```

NOTE: YÊU CẦU CỦA ĐỀ BÀI LÀ PHẢI THỰC HIỆN THEO VECTORIZATION

- **X_b**: là thông tin Tv, Radio, Newspaper (thông tin model nhận vào và sử dụng để predict Sale) (đã được normalize)
- **y**: là thông tin Sale (thông tin momg muốn model dự đoán đúng)
- **n_epochs**: Số lần train toàn bộ sample trong data
- **minibatch_size**: Số lượng sample sẽ được train trong 1 step (Chỉ sử dụng ở câu b)
- **learning_rate**: Tốc độ học
- **thetas_path**: List weights của model từ lúc khởi tạo cho đến sau mỗi lần cập nhật weights
- **losses**: List loss của mỗi step sau khi cập nhật

Bài tập 4 (Ưu nhược điểm của giải thuật linear regression): (Thảo luận và trao đổi trực tiếp trên lớp) Linear gression có thể ứng dụng cho bài toán phân loại không? néu có gì như thê nào. Tìm hiểu về "Normal Equation in Linear Regression"