hw1

Wenhua Bao 2512664
Yue Liu 2803140

May 31, 2020

1 Task1

HW 1

2020年5月4日 10:57

(a) associative: $A(BC)=(AB)C$

set $A_{n\times n}=(a_{ij})$ $B_{n\times n}=(b_{ij})$ $C_{n\times n}=(c_{ij})$

$AB=(d_{ij})$ $BC=(e_{ij})$

$ABC=(f_{ij})$ $A(BC)=(g_{ij})$

$\Rightarrow d_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{in}+b_{nj}$, $j,i=1,2,\cdots n$

$e_{ij}=b_{i1}c_{1j}+b_{i2}c_{2j}+\cdots+b_{in}c_{nj}$, $i,j=1,2,\cdots n$

$f_{ij}=d_{i1}c_{1j}+d_{i2}c_{2j}+\cdots+d_{in}c_{nj}$, $i,j=1,2,\cdots n$

$g_{ij}=a_{i1}e_{1j}+a_{i2}e_{2j}+\cdots+a_{in}e_{nj}$, $i,j=1,2,\cdots n$

$\Rightarrow \forall i,j=1,2,\cdots n$

$f_{ij}=d_{i1}c_{1j}+d_{i2}c_{2j}\cdots+d_{in}c_{nj}$

$=(a_{i1}b_{11}+a_{i2}b_{21}\cdots+a_{in}b_{n1})c_{1j}+$

$(a_{i1}b_{12}+a_{i2}b_{22}\cdots+a_{in}b_{n2})c_{2j}+$

$\cdots(a_{i1}b_{1n}+a_{i2}b_{2n}\cdots+a_{in}b_{nn})c_{nj}$

$=a_{i1}(b_{11}c_{1j}+b_{12}c_{2j}+\cdots+b_{1n}c_{nj})+$

$a_{i2}(b_{21}c_{1j}+\cdots+b_{2n}c_{nj})\cdots$

$+a_{in}(b_{n1}c_{1j}+\cdots b_{nn}c_{nj})$

$=a_{i1}e_{1j}+a_{i2}e_{2j}+\cdots a_{in}e_{nj}$

$=g_{ij}$

$\Rightarrow (AB)C=A(BC)$

distributive : $A(B+C) = AC+BC$

$(A+B)C = AC+BC$

$$[(A+B)C]_{ij} = \sum_k (A_{ik} + B_{ik}) C_{kj}$$

$$= \sum_k A_{ik} C_{kj} + \sum_k B_{ik} C_{kj}$$

$$= (AC)_{ij} + (BC)_{ij}$$

$\Rightarrow (A+B)C = AC+BC$

works similarly $A(B+C) = AB+AC$

Commtative : $AB = BA$

set $A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$ $B = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

$AB = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $BA = \begin{bmatrix} 1 & 4 & 9 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

$AB \neq BA$

commtative is incorrect .

1b) $A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 6 \\ 0 & 0 & 1 \end{bmatrix}$

$[A\ E] = \begin{bmatrix} 1 & 2 & 3 & 1 & 0 & 0 \\ 1 & 4 & 6 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$

$\xrightarrow{r_2 - r_1} \begin{bmatrix} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 2 & 3 & -1 & 1 & 0 \end{bmatrix} \xrightarrow{r_1 - r_2} \begin{bmatrix} 1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 2 & 3 & -1 & 1 & 0 \end{bmatrix}$

$$\xrightarrow{r_2-r_1}\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 2 & 3 & -1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]\xrightarrow{r_1-r_2}\left[\begin{array}{cccccc} 1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 2 & 3 & -1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{r_2-3r_3}\left[\begin{array}{cccccc} 1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 2 & 0 & -1 & 1 & -3 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]\xrightarrow{\frac{1}{2}r_2}\left[\begin{array}{cccccc} 1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 1 & 0 & -\frac{1}{2} & \frac{1}{2} & -\frac{3}{2} \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

$$A^{-1}=\begin{bmatrix} 2 & -1 & 0 \\ -\frac{1}{2} & \frac{1}{2} & -\frac{3}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A=\begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 6 \\ 1 & 0 & 0 \end{bmatrix} \quad \det(A)=\begin{vmatrix} 1 & 2 & 3 \\ 1 & 4 & 6 \\ 1 & 0 & 0 \end{vmatrix}$$

$$= 1\times4\times0 + 2\times6\times1 + 3\times1\times0$$
$$- 3\times4\times1 - 2\times1\times0 - 1\times6\times0$$
$$= 12-12=0$$

$\Rightarrow$ A is a singular matrix $\Rightarrow$ A is not invertible

1c) $L=(A^TA)^{-1}A^T$

$R=A^T(AA^T)^{-1}$

$Ax=I$
$A^TAx=A^TI$
$x=(A^TA)^{-1}A^T$

$xA=I$
$xAA^T=IA^T$
$x=A^T(AA^T)^{-1}$

$A^T\in R^{2\times3}$, $(A^TA)^{-1}\in R^{3\times3}$ hance A^T can mutiply $(A^TA)^{-1}$

$(A^TA)^{-1}\in R^{3\times3}$, $A^T\in R^{2\times3}$, hance the size of $(A^TA)^{-1}$ can't mutiply A^T

$\therefore$ A has only Right-Pseudo Inverse

$\Rightarrow$

1.0.1 1d

Eigenvectors represent various linear transformations of the matrix,their corresponding eigenvalues represent the degree of the linear transformation.A square matrix in a high-dimensional space represents a set of linear transformations in a high-dimensional space.These transformation include many transformations of different degree and in different directions. Therefore, in order to simplify the problem and retain as much transformation information as possible, we can decompose the high dimensional matrix by use of eigenvalue and eigenvectors, so as to approximately describe the square matrix. In machine learning ,complex square matrix can be described with help of eigenvalues and eigenvectors. SVD,PCA and Linear Discriminant Analysis involve eigenvalues and eigenvectors.

2 Task2

2.1 2a

2.1.1 1

Exception

$$E_{\omega\sim p(\omega)}[f] = \sum_{w\in\Omega} P(\omega)f(\omega)$$

Variance

$$var = E[f^2] - E[f]^2$$

they are unlinear operator

2.1.2 2

$$E(A) = 1\times\frac{4}{18} + 2\times\frac{1}{18} + 3\times\frac{6}{18} + 4\times\frac{2}{18} + 5\times\frac{1}{18} + 6\times\frac{4}{18} = 3.39$$

$$var(A) = \frac{(1-3.39)^2\times 4 + (2-3.39)^2\times 1 + (3-3.39)^2\times 6 + (4-3.39)^2\times 2 + (5-3.39)^2\times 1 + (6-3.39)^2}{17}$$

$$= 3.31$$

$$E(B) = 2.78$$

$$var(B) = 3.01$$

$$E(C) = 3.44$$

$$var(C) = 3.08$$

2.1.3 3

$$A : KL(p||q) = \sum P(x) log \frac{P(x)}{Q(x)} = 0.19$$

$$B : KL(p||q) = 0.25$$

$$C : KL(p||q) = 0.02$$

C is closest to a fair, uniform die.

2.2 2b

suffer from cold is event x, suffer from back-pain is event y

$$p(y|x) = 0.3$$

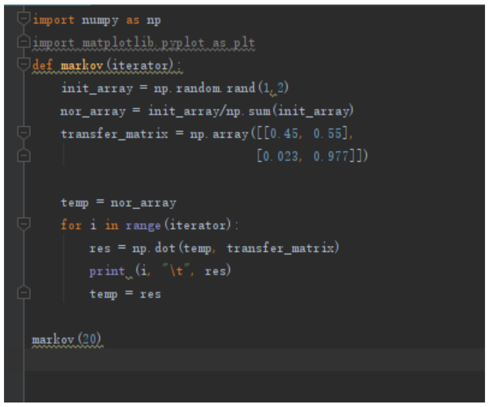
$$p(x) = 0.03$$

$$p(y|!x) = 0.1$$

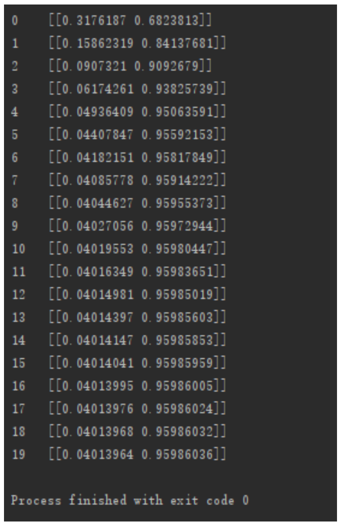
$$p(y) = p(x, y) + p(!x, y) = p(y|x) \times p(x) + p(y|!x) \times p(!x) = 0.106$$

$$p(x|y) = \frac{p(x, y)}{p(y)} = 0.085$$

2.3 2c



```
import numpy as np
import matplotlib.pyplot as plt
def markov(iterator):
    init_array = np.random.rand(1,2)
    nor_array = init_array/np.sum(init_array)
    transfer_matrix = np.array([[0.45, 0.55],
                                [0.023, 0.977]])

    temp = nor_array
    for i in range(iterator):
        res = np.dot(temp, transfer_matrix)
        print (i, "\t", res)
        temp = res

markov(20)
```



```
0    [[0.3176187 0.6823813]]
1    [[0.15862319 0.84137681]]
2    [[0.0907321 0.9092679]]
3    [[0.06174261 0.93825739]]
4    [[0.04936409 0.95063591]]
5    [[0.04407847 0.95592153]]
6    [[0.04182151 0.95817849]]
7    [[0.04085778 0.95914222]]
8    [[0.04044627 0.95955373]]
9    [[0.04027056 0.95972944]]
10   [[0.04019553 0.95980447]]
11   [[0.04016349 0.95983651]]
12   [[0.04014981 0.95985019]]
13   [[0.04014397 0.95985603]]
14   [[0.04014147 0.95985853]]
15   [[0.04014041 0.95985959]]
16   [[0.04013995 0.95986005]]
17   [[0.04013976 0.95986024]]
18   [[0.04013968 0.95986032]]
19   [[0.04013964 0.95986036]]

Process finished with exit code 0
```

The third generation has not changed significantly,the 21st has not changed.

My plan is failed.

3 Task3

3.1 3a

$$H(p) = \sum P(x) log \frac{1}{P(x)} = 1.28$$

$$max H(p) = \sum 0.25 log \frac{1}{0.25} = 2$$

The maximal entropy follows uniform distribution.

3.2 3b

3.2.1 1

$$min \ \sum_{i=1}^{4} p_i \log(p_i)$$

$$s.t. 6 = \sum_{i=1}^{4} 2p_i i, (p_i - 1) \leq 0$$

3.2.2 2

$$max_{\alpha,\beta} L(p_i, \lambda, \beta) = \sum_{i=1}^{4} p_i \log(p_i) + \alpha \sum_{i=1}^{4} (p_i - 1) + \beta (\sum_{i=1}^{4} 2p_i i - 6)$$

3.2.3 3

$$\frac{\partial L}{\partial p_i} = \log(p_i) + \frac{1}{ln2} + \alpha + 2\beta i, \ i = 1, 2, 3, 4$$

$$\frac{\partial L}{\partial \alpha} = \sum_{i=1}^{4} (p_i - 1)$$

$$\frac{\partial L}{\partial \beta} = (\sum_{i=1}^{4} 2p_i i - 6)$$

we need to solve the minimal value, but the $p_i - 1$is not 0, we can adjust α_i to make L become $-\infty$.

3.2.4 4

$$\theta_D = min_{p_i} L$$

$$\frac{\partial L}{\partial p_i} = 0, \ i = 1, 2, 3, 4$$

$$\alpha \sum_{i=1}^{4} (p_i - 1) = 0$$

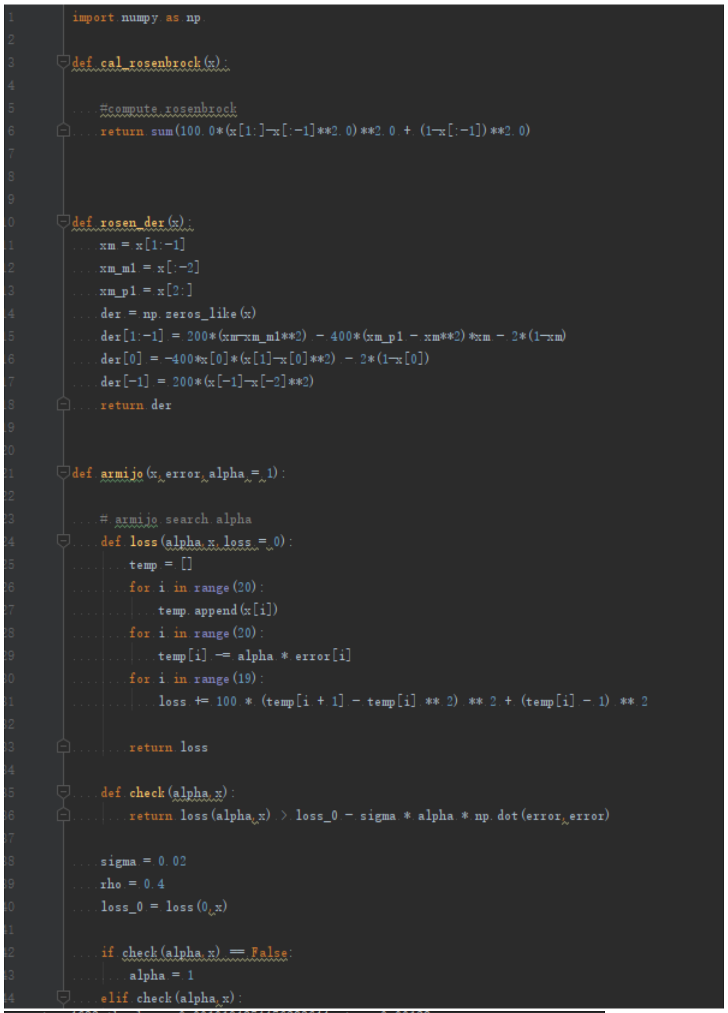
$$6 = \sum_{i=1}^{4} 2p_i i$$

we can compute $\alpha = 0$

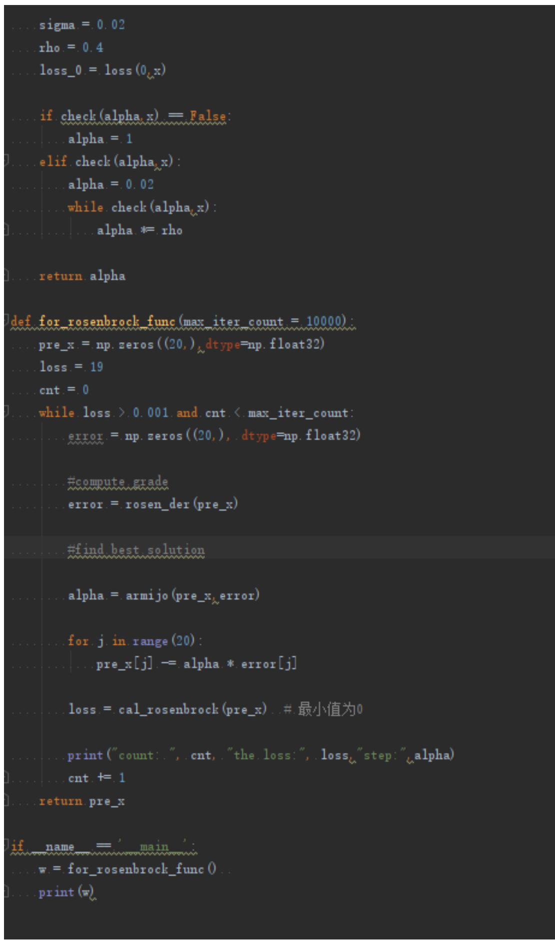
3.2.5 5

steepest descent

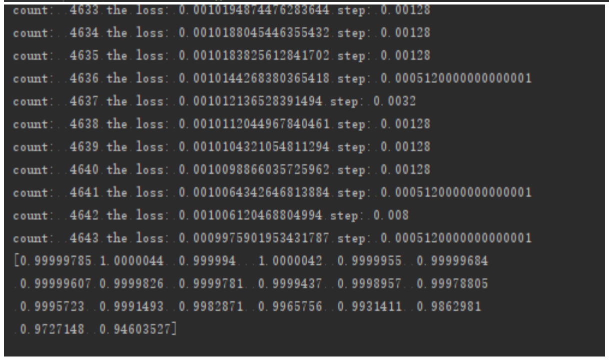
3.3 3c



```python
import numpy as np

def cal_rosenbrock(x):

    #compute rosenbrock
    return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)



def rosen_der(x):
    xm = x[1:-1]
    xm_m1 = x[:-2]
    xm_p1 = x[2:]
    der = np.zeros_like(x)
    der[1:-1] = 200*(xm-xm_m1**2) - 400*(xm_p1 - xm**2)*xm - 2*(1-xm)
    der[0] = -400*x[0]*(x[1]-x[0]**2) - 2*(1-x[0])
    der[-1] = 200*(x[-1]-x[-2]**2)
    return der


def armijo(x, error, alpha = 1):

    # armijo search alpha
    def loss(alpha, x, loss = 0):
        temp = []
        for i in range(20):
            temp.append(x[i])
        for i in range(20):
            temp[i] -= alpha * error[i]
        for i in range(19):
            loss += 100 * (temp[i + 1] - temp[i] ** 2) ** 2 + (temp[i] - 1) ** 2

        return loss

    def check(alpha, x):
        return loss(alpha, x) > loss_0 - sigma * alpha * np.dot(error, error)

    sigma = 0.02
    rho = 0.4
    loss_0 = loss(0, x)

    if check(alpha, x) == False:
        alpha = 1
    elif check(alpha, x):
```



```python
    sigma = 0.02
    rho = 0.4
    loss_0 = loss(0, x)

    if check(alpha, x) == False:
        alpha = 1
    elif check(alpha, x):
        alpha = 0.02
        while check(alpha, x):
            alpha *= rho

    return alpha

def for_rosenbrock_func(max_iter_count = 10000):
    pre_x = np.zeros((20,), dtype=np.float32)
    loss = 19
    cnt = 0
    while loss > 0.001 and cnt < max_iter_count:
        error = np.zeros((20,), dtype=np.float32)

        #compute grade
        error = rosen_der(pre_x)

        #find best solution

        alpha = armijo(pre_x, error)

        for j in range(20):
            pre_x[j] -= alpha * error[j]

        loss = cal_rosenbrock(pre_x)  # 最小值为0

        print("count: ", cnt, "the loss:", loss, "step:", alpha)
        cnt += 1
    return pre_x

if __name__ == '__main__':
    w = for_rosenbrock_func()
    print(w)
```



```
count:  4633 the loss: 0.0010194874476283644 step: 0.00128
count:  4634 the loss: 0.0010188045446355432 step: 0.00128
count:  4635 the loss: 0.0010183825612841702 step: 0.00128
count:  4636 the loss: 0.0010144268380365418 step: 0.0005120000000000001
count:  4637 the loss: 0.001012136528391494 step: 0.0032
count:  4638 the loss: 0.0010112044967840461 step: 0.00128
count:  4639 the loss: 0.0010104321054811294 step: 0.00128
count:  4640 the loss: 0.0010098866035725962 step: 0.00128
count:  4641 the loss: 0.0010064342646813884 step: 0.0005120000000000001
count:  4642 the loss: 0.001006120468804994 step: 0.008
count:  4643 the loss: 0.0009975901953431787 step: 0.0005120000000000001
[0.99999785 1.0000044  0.999994   1.0000042  0.9999955  0.99999684
 0.99999607 0.9999826  0.9999781  0.9999437  0.9998957  0.99978805
 0.9995723  0.9991493  0.9982871  0.9965756  0.9931411  0.9862981
 0.9727148  0.94603527]
```

in step 4642 we achieve x=[0.99999785 1.0000044 0.999994 1.0000042 0.9999955

0.99999684 0.99999607 0.9999826 0.9999781 0.9999437 0.9998957 0.99978805 0.9995723 0.9991493 0.9982871 0.9965756 0.9931411 0.9862981 0.9727148 0.94603527]
if the delta is too long, the plot will concussion.

3.4 3d

3.4.1 1

Batch:

$$\theta_{j+1} = \theta - \alpha \delta_\theta J(\theta)$$

all the datas are used in the computing, so the result is exact but the calculation time is large and occupy much resource. stochastic:

$$\theta_{j+1} = \theta - \alpha \delta_\theta J(\theta, x^i, y^i)$$

every iteration only uses oone point in SGD, so the need of calculation time is less . But not all iterations can reach to the global optimum. Mini-Batch:

$$\theta_{j+1} = \theta - \alpha \delta_\theta J(\theta, x^{i:i+n}, j^{i:i+n})$$

This allows a balance between calculated amount and computing time,but the convergence is not very ideal.

3.4.2 2

When we use new data to iterate an inertia, we gives the old direction aweight too. The computing is stabler but not always useful when we need to run faster.