Student Names and IDs:

- Yiteng Lu (2488152)
- Wenge Xie (2466824)
- Zengtian deng(2207324)

Homework 4

Part 1: Exam-Style Questions

Problem 1.1

Suppose that the histogram of orientations $\theta \in [0, 180)$ has 18 equal bins, numbered 0 to 17, and with centers $5, 15, \ldots, 175$. Bilinear voting is used, as described in the class notes on Histograms of Oriented Gradients. Give the fraction of vote that each of the bins receives when a measurement $\theta = 42$ is observed. Fractions are values in $[0, 1]$.

Answer

$$v_3 = 0.3$$
$$v_4 = 0.7$$

Problem 1.2

The x and y components of ∇I at a particular pixel in an image are $10\sqrt{3}$ and 10, respectively. What are the magnitude and orientation of the gradient there? Express orientation in degrees relative to the x axis.

Answer

$$\mu = 20$$
$$\theta = 30^\circ$$

Problem 1.3

A HOG feature is computed from a 128×64 window divided into cells with 8×8 pixels each. The size of a block is 2×2 cells. Central differences are used to compute gradients. Each cell histogram has nine bins of equal size, and bin 0 is for orientations in $[0, 20)$ degrees.

Rather than using a constant ϵ in the denominator during normalization of a histogram $\mathbf{v}$, the following procedure is used at all stages to normalize a histogram $\mathbf{v}$ to $\mathbf{v}'$:

$$\mathbf{v}' = \begin{cases} \frac{\mathbf{v}}{\|\mathbf{v}\|} & \text{if } \|\mathbf{v}\| \neq 0 \\ \mathbf{v} & \text{otherwise.} \end{cases}$$

The threshold τ used to saturate the feature entries during contrast normalization is set to infinity (so that it has no effect).

For a given window, the pixel at position $(5, 5)$ has value 50, and all other pixels are zero.

Give an expression for the entries of the HOG feature $\mathbf{h}$ for that window.

Answer

$$\mathbf{h} = \left[\frac{1}{\sqrt{6}}, 0, 0, 0, \frac{2}{\sqrt{6}}, 0, 0, 0, \frac{1}{\sqrt{6}}, 0, 0....0\right]$$

$\mathbf{h}$ is a vector contains 3780 entries but everything after the 9th entry are zeros

Problem 1.4

The magnitude of a vector $\mathbf{x} = [x, y]^T \in \mathbb{R}^2$ is

$$f(x, y) = \sqrt{x^2 + y^2} \ .$$

Write algebraic expressions and exact decimal numerical values of the gradient ∇f and Hessian H_f of f at $x = 3$ and $y = 4$.

[Hint: $1/5^3 = 1/125 = 8/1000$.]

Answer

$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{x}{\sqrt{(x^2+y^2)}} \\ \frac{y}{\sqrt{(x^2+y^2)}} \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.8 \end{bmatrix}$$

$$H_f(\mathbf{x}) = \frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{x}^T} = \begin{bmatrix} -(x^2+y^2)^{-\frac{3}{2}}x^2 + (x^2+y^2)^{-\frac{1}{2}} & -(x^2+y^2)^{-\frac{3}{2}}yx \\ -(x^2+y^2)^{-\frac{3}{2}}yx & -(x^2+y^2)^{-\frac{3}{2}}y^2 + (x^2+y^2)^{-\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} 0.128 \\ -0.096 \end{bmatrix}$$

Problem 1.5

The second stage of line search repeatedly narrows a bracketing triple (a, b, c) established in the first stage. Suppose that at some point during the second stage the bracketing triple is:

$$a = 0 \quad , \quad b = 2 \quad , \quad c = 6$$

with function values

$$h(a) = 6 \quad , \quad h(b) = 3 \quad , \quad h(c) = 4$$

along the search line. Assume further that

$$h(1) = 2 \quad \text{and} \quad h(4) = 5 \ .$$

What is the bracketing triple after that? Standard line search is applied, that is, no golden ratio is used. No need to explain.

[Hint: Draw a sketch.]

Answer

$$a = 0 \quad , \quad b = 2 \quad , \quad c = 4$$

Problem 1.6

The gradient and Hessian at $\mathbf{z}_0 = [4, 1]^T$ for some function $f : \mathbb{R}^2 \to \mathbb{R}$ are as follows:

$$\nabla f(\mathbf{z}_0) = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \text{and} \quad H_f(\mathbf{z}_0) = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} .$$

Find the point $\mathbf{z}_1$ that results by taking a single Newton step from $\mathbf{z}_0$. Show your reasoning briefly.

Answer

set derivatives of $f(\mathbf{z_k} + \nabla \mathbf{z}) = 0$:

$$H_k \nabla \mathbf{z} = -\nabla f(\mathbf{z}_0)$$

$$\begin{aligned} \nabla \mathbf{z} &= -H_k^{-1} \nabla f(\mathbf{z}_0) \\ &= -\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \\ &= -\begin{bmatrix} 0.6 & -0.2 \\ -0.2 & 0.4 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} -1 \\ 0 \end{bmatrix} \end{aligned}$$

$$\mathbf{z}_1 = \mathbf{z}_0 + \nabla \mathbf{z} = \begin{bmatrix} 4 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

Problem 1.7

A classification problem asks to find a classifier $h \in \mathcal{H}$ whose domain X and co-domain Y of h are defined as follows:

$$X = \{0, 2, 4, 6, 8, 10\} \quad \text{and} \quad Y = \{0, 1\}$$

and the hypothesis space $\mathcal{H}$ is the set of the following five functions

$$h_k(x) = \begin{cases} 0 & \text{if } x < k \\ 1 & \text{otherwise} \end{cases} \quad \text{for} \quad k = 1, 3, 5, 7, 9 \ .$$

The following tiny training set is given:

$$T = \{(0, 0), (2, 0), (4, 1), (6, 1)\} \ .$$

Make a table of the training risk $L_T(h_k)$ for $k = 1, 3, 5, 7, 9$. Express values in percent. Then use the table to determine the optimal classifier $\hat{h}$ for this training set (thus, we only care about empirical risk here).

Answer

k	1	3	5	7	9
$L_T(h_k)(\%)$	25	0	25	50	50

The optimal classifier on T is $\hat{k} = 3$.

Problem 1.8

Assume that the probability model $p(x, y)$ for the data in Problem 1.7 is as specified in the table below. The table contains the joint probability $p(x, y)$ with values expressed in percent, rather than as numbers in $[0, 1]$. It has one column per value of x and one row per value of y.

	0	2	4	6	8	10
0	12	11	10	7	6	4
1	2	4	6	10	12	16

Under the same circumstances as in Problem 1.7, and with the probability model above, make a table of the statistical risk $L_p(h)$ and determine the classifier that generalizes best. Express values in percent.

Answer

(Remove this line and complete the expressions below).

k	1	3	5	7	9
$L_p(h_k)$	40	33	29	32	38

The classifier that generalizes best is $\hat{k} = 5$.

Part 2: Projections

Problem 2.1

What is the shape (number of rows and columns) of U? Explain briefly.

Answer

The shape of U is $m \times (m - dim(u))$. First, since $\mathcal{O}(u)$ and $\mathbf{u}$ are corthogoanl complement to each other, then $dim(u) + dim(\mathcal{O}(u)) = m$, so $dim(\mathcal{O}(u)) = m - dim(u)$. The columns in U can span the space $\mathcal{O}(u)$, so the number of the columns in U is the same as $dim(\mathcal{O}(u))$, which means the matrix U has $m - dim(u)$ columns. Second, since the projection $\mathbf{p}$ is still in the space $\mathbb{R}^m$, and $\mathbf{p}$ is the linear combination of all the column vectors in U, then all the column vectors in U are also in the space $\mathbb{R}^m$, which means each of them has m elements. So, the matrix U has m rows.

Problem 2.2

Explain clearly how, given $\mathbf{u}$, you can use the Singular Value Decomposition (SVD) to compute a matrix whose columns form an orthonormal basis for $\mathcal{O}(\mathbf{u})$. Also explain why.

Then show a basis for $\mathcal{O}([1, 1, 0]^T)$ using your technique.

Give the projection matrix P for this space, and the projection of the vector $\mathbf{b} = [0, 2, -4]^T$ onto $\mathcal{O}([1, 1, 0]^T)$.

Use whatever code you'd like, but do **not** submit your code. Give your numerical results either exactly (if you can guess what they are) or with four decimal digits after the period.

Do **not** use any other techniques, such as Gram-Schmidt.

Answer

By the property of SVD, we have:

$$A = U\Sigma V^T, \text{where } A \in \mathbb{R}^{m\times n}, U \in \mathbb{R}^{m\times m}, \Sigma \in \mathbb{R}^{m\times n}, V \in \mathbb{R}^{n\times n}$$

So, if we do SVD on the given vector $\mathbf{u}$, we have:

$$\mathbf{u} = U_u \Sigma_u V_u^T, \text{where } u \in \mathbb{R}^{m\times 1}, U_u \in \mathbb{R}^{m\times m}, \Sigma_u \in \mathbb{R}^{m\times 1}, V_u \in \mathbb{R}^{1\times 1}$$

So,

$$\mathbf{u} = [\mathbf{u_1}, \mathbf{u_2}, \cdots, \mathbf{u_m}] \begin{bmatrix} \sigma_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} [1] = \sigma_1 \mathbf{u_1}$$

From the form of SVD, we can see that the vectors $\{\mathbf{u_2}, \cdots, \mathbf{u_m}\}$ are all orthogonal to $\mathbf{u_1}$, and they are orthonormal to each other. So, we can use them to span a space which is orthogonal to $\mathbf{u_1}$. Since $\mathbf{u} = \sigma \mathbf{u_1}$, the space we spanned by $\{\mathbf{u_2}, \cdots, \mathbf{u_m}\}$ is also orthogonal to $\mathbf{u}$, which means, we can build a matrix $\begin{bmatrix} \mathbf{u_2} & \cdots & \mathbf{u_m} \end{bmatrix}$ whose columns form an orthonormal basis for $\mathcal{O}(\mathbf{u})$.

For $\mathbf{u} = [1, 1, 0]^T$, first, compute its SVD decomposition:

$$\mathbf{u} = \begin{bmatrix} 0.7071 & -0.7071 & 0 \\ 0.7071 & 0.7071 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1.4142 \\ 0 \\ 0 \end{bmatrix} [1]$$

So, the basis for $\mathcal{O}(\mathbf{u})$ is

$$\left\{ \begin{bmatrix} -0.7071 \\ 0.7071 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}$$

The matrix U whose columns form the orthonormal basis of $\mathcal{O}(\mathbf{u})$ is then

$$\begin{bmatrix} -0.7071 & 0 \\ 0.7071 & 0 \\ 0 & 1 \end{bmatrix}.$$

The projection matrix P for this space is then UU^T, which is

$$\begin{bmatrix} 0.5 & -0.5 & 0 \\ -0.5 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The projection of the vector b onto $\mathcal{O}([1, 1, 0]^T)$ is then $P\mathbf{b}$, which is

$$\begin{bmatrix} -1 \\ 1 \\ -4 \end{bmatrix}.$$

Problem 2.3

For each of the two matrices U and P and the vector $\mathbf{p}$ you found in the previous problem, state whether the object is unique, and explain briefly why or why not.

Answer

U is not unique, but P and $\mathbf{p}$ are unique. The reason why U is not unique is because we can flip the vectors in the matrix $\begin{bmatrix} u_1 & u_2 & \cdots & u_m \end{bmatrix}$ of the SVD and make them point to the opposite direction. This operation will maintain a valid SVD decompositon if we flip the direction of the corresponding vectors in the matrix $\begin{bmatrix} v_1^T & v_2^T & \cdots & v_n^T \end{bmatrix}^T$. Since the matrix U is built by combining the vectors $u_2, \cdots, u_m$ together as the columns of U, the matrix U will be changed if we flipped the vectors $u_2, \cdots, u_m$. So, it is not unique. However, no matter which vectors we chose to be flipped, the P matrix will stay the same. Denote U as $\begin{bmatrix} u_2, \cdots, u_n \end{bmatrix}$, then $UU^T = u_2 u_2^T + \cdots u_m u_m^T$. No matter which vectors we chose to be pointing at the opposite direction of itself, UU^T will remain $u_2 u_2^T + \cdots u_m u_m^T$ (The minus sign will be canceled out). So, matrix P will remain the same, which means it is unique. Since $\mathbf{p} = P\mathbf{b}$, the vector $\mathbf{p}$ will remain unique if we choose an unique vector $\mathbf{b}$, because P is an unique matrix.

Problem 2.4

Compute the matrix

$$Q = I - \mathbf{n}\mathbf{n}^T$$

for the example given in Problem 2.2.

Show Q if it is different from the matrix P you obtained through SVD. Otherwise, just state that the two matrices are the same.

Answer

The two matrices are the same.

Part 3: Optimization and the SVD

In [4]:
```
import numpy as np
from scipy.optimize import minimize_scalar

def normalize(v):
    n = np.linalg.norm(v)
    if n > 0:
        v = v / n
    return v, n

# sphere_search returns a new point z1, not a search step size
def sphere_step(f, g, z0, epsilon=1.e-6, args=()):
    assert epsilon > 0 and epsilon < 1, 'epsilon must be in (0, 1)'
    assert z0.size > 1, 'sphere must be at least 2-dimensional'

    n0, delta = 0, 0
    while n0 < epsilon:
        z0 += delta * np.sqrt(epsilon) * np.random.random(z0.shape)
        z0, _ = normalize(z0)
        p0 = -g(z0, *args)  # Starting direction
        # Project p0 onto the tangent hyperplane to the sphere at z0 a
nd normalize
        s0, n0 = normalize(p0 - np.dot(z0, p0) * z0)
        delta = 1

    def h(theta, args):
        return f(z0 * np.cos(theta) + s0 * np.sin(theta), args)

    res = minimize_scalar(h, bracket=(0, 1), args=args)
    theta = res.x
    return z0 * np.cos(theta) + s0 * np.sin(theta)
```

Problem 3.1

Write a function with the following header and `assert` statements:

```
def first(A, epsilon=1.e-6, maxiter=10):
    assert A.size > 0, 'array cannot be empty'
    assert np.max(np.abs(A)) >= epsilon, 'array cannot be zero'
```

that uses `sphere_step` iteratively to return the tuple `u, sigma, v` of the first left singular vector, first singular value, and first right singular vector of `A`.

Show your code and the results of running the given tests. You may want to compare your results with those obtained with an off-the-shelf implementation of the SVD. However, do **not** submit your comparison, and do not use SVDs anywhere in your code.

Answer

In [15]:
```
import numpy as np
def first(A, epsilon=1.e-6, maxiter=10):
    assert A.size > 0, 'array cannot be empty'
    assert np.max(np.abs(A)) >= epsilon, 'array cannot be zero'
    if A.shape[1]  == 1:
        v  = np.array([1])
        sigma =  np.linalg.norm(np.dot(A,v))
        u = np.dot(A,v)/sigma
        return u,sigma , v
    if A.shape[0]  == 1:
        u  = np.array([1])
        sigma =  np.linalg.norm(A)
        v = A.T/sigma
        return u,sigma , v
    def f(u, A):
        return (-u.T@A.T@A@u)
    def g(u,A):
        c = A.shape[1]
        r = A.shape[0]
        gradient = []
        for i in range(c):
            temp = np.zeros((c,1))
            for j in range(c):
                temp[j] = np.dot(A[:,i].T,A[:,j])
            gradient.append(-((np.dot(u,temp)+np.dot(A[:,i].T, np.dot(
A,u))))[0])
        return np.asarray(gradient)


    v_k = np.ones(A.shape[1])
    v_k_1 = np.zeros(A.shape[1])
    T = 0
    while np.linalg.norm(v_k) - np.linalg.norm(v_k_1) > epsilon and T
< maxiter:
        if T > 0:
            v_k_1 = v_k
        v_k = sphere_step(f, g, v_k, epsilon=epsilon, args=(A,))
        T +=1
    sigma = np.linalg.norm(np.dot(A,v_k))
    u = np.dot(A,v_k)/sigma
    return u,sigma , v_k
```

In [16]:
```
def test(f, A_list):
    def printArray(name, A):
        print(name, A, sep='\n', end='\n\n')

    if f.__name__ is 'first':
        un, sn, vn = 'u', 'sigma', 'v'
        product = False
    else:
        un, sn, vn = 'U', 'Sigma', 'V'
        product = True

    np.set_printoptions(precision=4, suppress=True)

    for A in A_list:
        U, Sigma, V = f(A)
        print('=' * 10, 'testing', f.__name__, '=' * 10)
        printArray('A', A)
        if product:
            printArray("U * Sigma * V'", np.dot(U, np.dot(Sigma, V.T))
)
        printArray(un, U)
        printArray(sn, Sigma)
        printArray(vn, V)

s3 = np.sqrt(3)
A_list = (np.array([[s3, s3], [-3., 3.], [1., 1.]]) / np.sqrt(2.),
          np.outer([1., 2., 3.], [2., 0., -1., 3.]),
          np.array([[3.], [0.], [4.]]), np.array([[4., -3.]]))

# try:
test(first, A_list)
# except NameError:
#     pass
```

```
========== testing first ==========
A
[[ 1.2247  1.2247]
 [-2.1213  2.1213]
 [ 0.7071  0.7071]]

u
[-0. -1. -0.]

sigma
2.999999999999999

v
[ 0.7071 -0.7071]
```

```
========== testing first ==========
A
[[ 2.  0. -1.  3.]
 [ 4.  0. -2.  6.]
 [ 6.  0. -3.  9.]]

u
[0.2673 0.5345 0.8018]

sigma
14.000000000000002

v
[ 0.5345 -0.     -0.2673  0.8018]

========== testing first ==========
A
[[3.]
 [0.]
 [4.]]

u
[0.6 0.  0.8]

sigma
5.0

v
[1]

========== testing first ==========
A
[[ 4. -3.]]

u
[1]

sigma
5.0

v
[ 0.8 -0.6]
```

Problem 3.2

Use your function `first` to write a function with the following header and `assert` statement that computes the "tiny" SVD of a non-empty matrix A, with the given specifications:

```
def SVD(A, epsilon=1.e-3, maxiter=10):
    assert A.size > 0, 'array cannot be empty'
```

The function should return the tuple `U, Sigma, V` of the matrices in the "tiny" SVD of `A`, and should comply with the programming notes.

Show your code and the results of running the given tests. You may want to compare your results with those obtained with an off-the-shelf implementation of the SVD. However, do **not** submit your comparison. Do not use library SVDs anywhere in your code.

Answer

In [17]:
```
from numpy.linalg import matrix_rank
def SVD(A, epsilon=1.e-3, maxiter=10):
    m = A.shape[0]
    n = A.shape[1]
    r = matrix_rank(A)
    U = np.zeros((m,r))
    Sigma= np.zeros((r,r))
    V= np.zeros((n,r))
    assert A.size > 0, 'array cannot be empty'
    i = 0
    while np.max(np.abs(A)) > epsilon:
        u,sigma,v = first(A, epsilon=1.e-6, maxiter=maxiter)
        U[:,i] = u
        Sigma[i,i]=sigma
        V.T[i] = v
        v = np.reshape(v,(len(v),1))
        A = A-((A@v)@v.T)
        i+=1
    return U,Sigma,V
```

In [18]:
```
s3 = np.sqrt(3)
A_list = (np.array([[s3, s3], [-3., 3.], [1., 1.]]) / np.sqrt(2.),
          np.outer([1., 2., 3.], [2., 0., -1., 3.]),
          np.array([[3.], [0.], [4.]]), np.array([[4., -3.]]))

try:
    test(SVD, A_list)
except NameError:
    pass
```

```
========== testing SVD ==========
A
[[ 1.2247  1.2247]
 [-2.1213  2.1213]
 [ 0.7071  0.7071]]

U * Sigma * V'
[[ 1.2247  1.2247]
 [-2.1213  2.1213]
 [ 0.7071  0.7071]]

U
[[-0.     0.866]
 [ 1.     0.   ]
 [-0.     0.5  ]]

Sigma
[[3. 0.]
 [0. 2.]]

V
[[-0.7071  0.7071]
 [ 0.7071  0.7071]]

========== testing SVD ==========
A
[[ 2.  0. -1.  3.]
 [ 4.  0. -2.  6.]
 [ 6.  0. -3.  9.]]

U * Sigma * V'
[[ 2.  0. -1.  3.]
 [ 4.  0. -2.  6.]
 [ 6.  0. -3.  9.]]

U
[[0.2673]
 [0.5345]
 [0.8018]]
```

```
Sigma
[[14.]]

V
[[ 0.5345]
 [ 0.    ]
 [-0.2673]
 [ 0.8018]]

========== testing SVD ==========
A
[[3.]
 [0.]
 [4.]]

U * Sigma * V'
[[3.]
 [0.]
 [4.]]

U
[[0.6]
 [0. ]
 [0.8]]

Sigma
[[5.]]

V
[[1.]]

========== testing SVD ==========
A
[[ 4. -3.]]

U * Sigma * V'
[[ 4. -3.]]

U
[[1.]]

Sigma
[[5.]]

V
[[ 0.8]
 [-0.6]]
```