人工智能第一次作业

姓名

学号

1 Q1

由上图可知，

$$v_k = b_k + \sum_i^m x_i w_k i \tag{1}$$

$$y_k = \varphi(v_k) = \frac{1}{1+e^{-av_k}} \tag{2}$$

若输入向量 $x = [x_1, x_2, ..., x_m]^T$ 所对应输出 $y > \xi$,ξ 为阈值，$0 < \xi < 1$, 则 x 属于 C_1 类; 否则，x 属于 C_2 类。于是有:

$$v_k = -\frac{1}{a}\ln\left(\frac{1}{\xi} - 1\right) \tag{3}$$

$$b_k + \sum_i^m x_i w_{ki} = -\frac{1}{a}\ln\left(\frac{1}{\xi} - 1\right) \tag{4}$$

令 $B = b_k + \frac{1}{a}\ln\left(\frac{1}{\xi} - 1\right)$，于是有

$$\sum_i^m x_i w_{ki} + B = 0 \tag{5}$$

因此可得决策面是一个超平面。

2 Q2

如果 XOR 问题是线性可分的，那么必然存在一个平面 $w_1x_1 + w_2x_2 + b = 0$, 可以对真值表中所有的情况进行分类，我们仅考虑以下不等式:

$$0w_1 + 0w_2 + b < 0 \quad (1) \tag{6}$$

$$0w_1 + 1w_2 + b \geq 0 \quad (2) \tag{7}$$

$$1w_1 + 0w_2 + b \geq 0 \quad (3) \tag{8}$$

$$1w_1 + 1w_2 + b < 0 \quad (4) \tag{9}$$

得到

$(1) + (4) \longrightarrow w_1 + w_2 + 2b < 0$

$(2) + (3) \longrightarrow w_1 + w_2 + 2b \geq 0$

以上两式相互矛盾，因此，不存在符合这一平面的 w_1,w_2,b，XOR 问题线性不可分。

3 Q3

3.1 a

有如下结论：

AND:

$$w = \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}, \quad v = x_1 + x_2 - 1.5$$

OR:

$$w = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}, \quad v = x_1 + x_2 - 0.5$$

COMPLEMENT:

$$w = \begin{bmatrix} 0.5 \\ -1 \end{bmatrix}, \quad v = -x_1 + 0.5$$

当 $v \geq 0$, $y = 1$, 否则 $y = 0$。

3.2 b、c

对于 b 中三种情境： 初始权重是随机选择，学习率为 1.0，权重的变化轨迹从图 1 到图 3 展示。从图中可以看出，权重在初始阶段发生变化，然后保持稳定（收敛）。原因是权重只在发生误分类时才会改变，一旦决策边界能够适应所有点，权重就不会再变化。

离线计算和学习过程的比较如表 1 所示。每个权重的参数略有不同，因为学习过程是从随机选择的权重开始的。离线计算和学习过程的决策边界图从图 5 到图 7 展示。结果表明，两种决策边界之间存在差异，但它们都能正确地对情况分类。

对于 c 中 XOR： XOR 函数不是线性可分的，这意味着它不能通过单一的线性决策边界来分离。因此，使用只有一个权重向量和偏置的线性模型无法正确分类 XOR 函数的所有点，权重的轨迹不会收敛，而是在每个时期都保持波动。

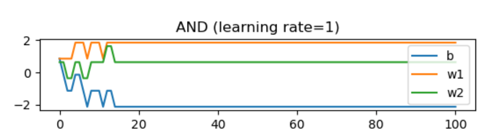

图 1: AND weights (learning rate=1)

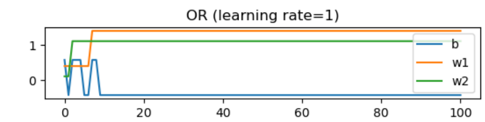

图 2: OR weights (learning rate=1)

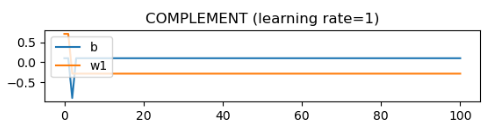

图 3: COMPLEMENT (learning rate=1)

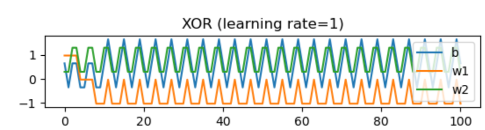

图 4: NAND weights (learning rate=1)

表 1: Comparison of off-line calculation and learning procedure

	b	w_1	w_2
AND (off-line)	-1.5	1	1
AND (learning)	-14.69217366	10.17524972	5.43629322
OR (off-line)	-0.5	1	1
OR (learning)	-4.486140472	5.48337127	5.38394283
COMPLEMENT (off-line)	0.5	-1	NA
COMPLEMENT (learning)	0.18236827	-4.96731703	NA
XOR (learning)	0.85196828	-4.31005925	0.77310962

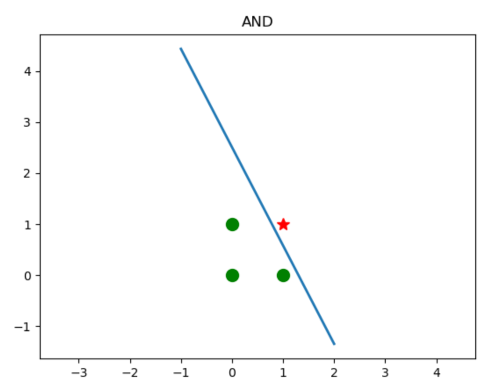

图 5: AND decision boundary

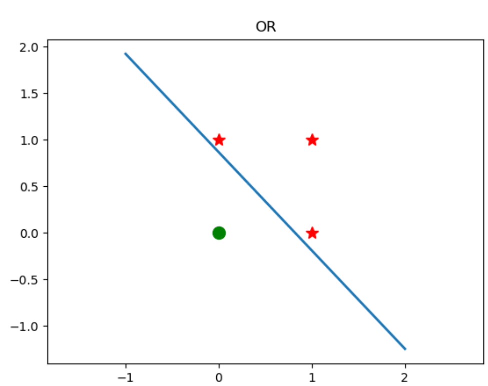

图 6: OR decision boundary

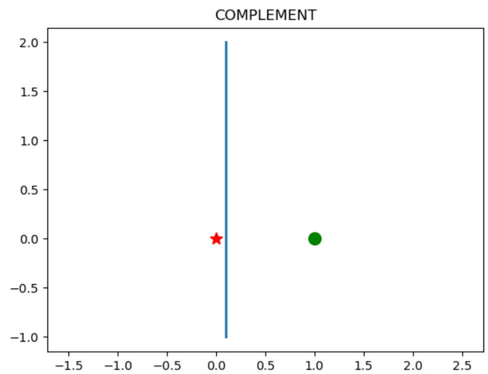

图 7: COMPLEMENT decision boundary

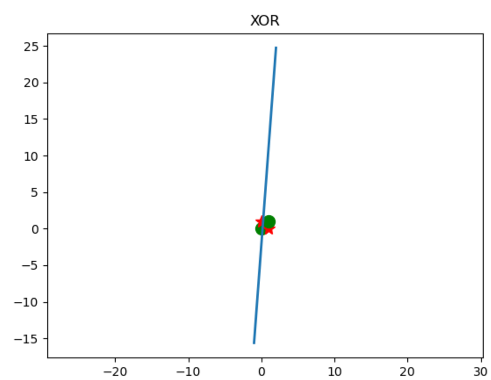

图 8: XOR decision boundary

4 Q4

4.1 a

结果如下，代码见附录：

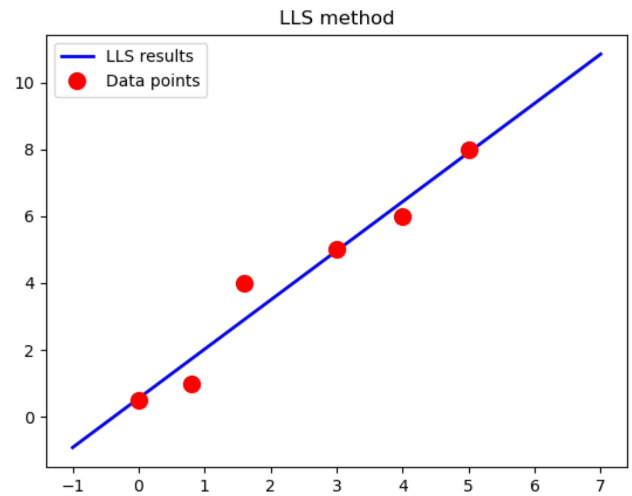

图 9: LLS 方法结果

4.2 b

LMS 方法结果如下：

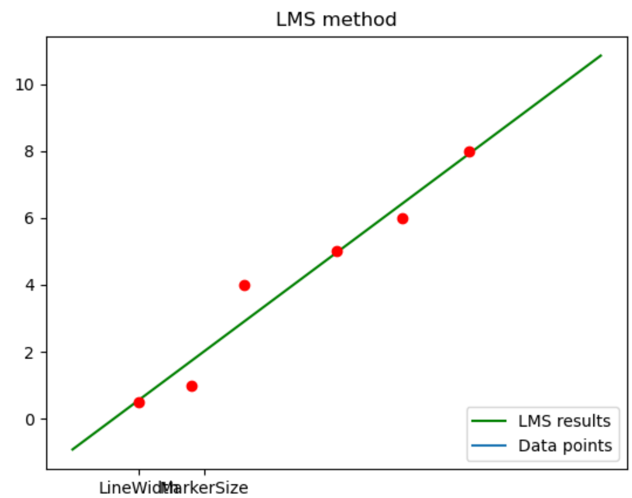

图 10: LMS 方法结果

LMS 权重轨迹如图 11 所示，展示了学习率 0.02, 迭代次数 200 下权重的变化。

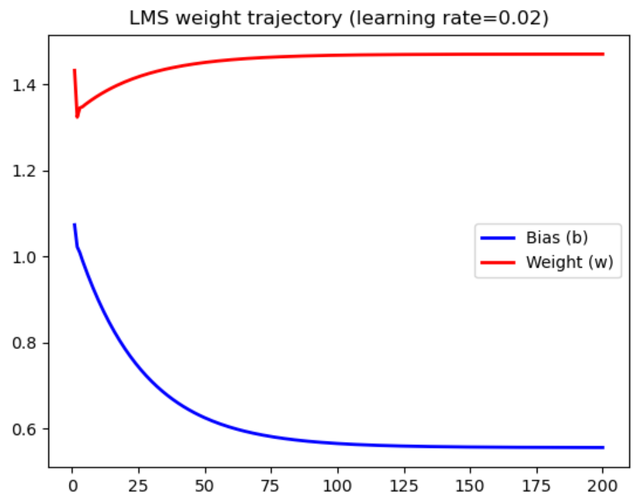

图 11: LMS 权重轨迹 (学习率 =0.01)

4.3 c

由下表可看到两种方法的权重在小数点后三位完全一样，差的不多。

表 2: 离线计算与学习过程的比较

	b	w
LLS	0.55543634	1.46995708
LMS	0.55534423	1.46998296

5 Q5

令

$$R = \begin{bmatrix} r_1^2 & & & \\ & r_2^2 & & \\ & & \ddots & \\ & & & r_n^2 \end{bmatrix} \tag{10}$$

这里，X 是一个大小为 (n,m) 的矩阵，代表输入数据；w 是一个大小为 m 的向量, 代表模型的权重；则 $e = d - Xw$。

于是可以得到

$$J = \frac{1}{2}e^\top Re + \frac{1}{2}\lambda w^\top w \tag{11}$$

$$\frac{\partial J}{\partial w} = e^\top R(-X) + \lambda w^\top \tag{12}$$

$$(d - Xw)^\top R(-X) + \lambda w^\top = 0 \tag{13}$$

$$w^\top X^\top RX - d^\top RX + \lambda w^\top = 0 \tag{14}$$

$$w^\top (X^\top RX + \lambda I) = d^\top RX \tag{15}$$

$$w^\top = d^\top RX(X^\top RX + \lambda I)^{-1} \tag{16}$$

A Q3 代码

```
import numpy as np
import matplotlib.pyplot as plt

# 定义训练函数
def training(x, y, rates, epochs, name):
    dim = x.shape[0]
    num = x.shape[1]
    omegas = np.random.rand(epochs + 1, dim + 1)

    for k, r in enumerate(rates):
        for t in range(epochs):
            pos = (t % num) + 1
            now = np.concatenate(([1], x[:, pos - 1]))
            d = np.sum(now * omegas[t, :])
            d = d > 0
            e = d - y[pos - 1]
            omegas[t + 1, :] = omegas[t, :] - e * r * now

        plt.figure()
        plt.subplot(len(rates), 1, k + 1)
        t = np.arange(0, epochs + 1)
        if dim == 1:
            plt.plot(t, omegas[:, 0], label='b')
            plt.plot(t, omegas[:, 1], label='w1')
        elif dim == 2:
            plt.plot(t, omegas[:, 0], label='b')
            plt.plot(t, omegas[:, 1], label='w1')
            plt.plot(t, omegas[:, 2], label='w2')
        plt.legend()
        plt.title(f'{name} (learning rate={r})')

    return omegas[-1, :]


# 定义绘制决策边界的函数
# 定义绘制决策边界的函数
def drawDecBou(x, y, omega, name):
    dim = x.shape[0]
    num = x.shape[1]
```

```
    plt.figure()
    if dim == 2:
        t = np.linspace(-1, 2, 100)
        for i in range(num):
            if y[i]:
                plt.plot(x[0, i], x[1, i], 'r*', markersize=10)
            else:
                plt.plot(x[0, i], x[1, i], 'go', markersize=10)
        o1, o2, o3 = omega[0], omega[1], omega[2]
        ft = -(t * o2 + o1) / o3
        plt.plot(t, ft, linewidth=2)  # 这里修改了 'LineWidth' 为 'linewidth'

    elif dim == 1:
        t = np.linspace(-1, 2, 100)
        for i in range(num):
            if y[i]:
                plt.plot(x[0][i], 0, 'r*', markersize=10)
            else:
                plt.plot(x[0][i], 0, 'go', markersize=10)
        ft = np.ones(len(t)) * omega[0]
        plt.plot(ft, t, linewidth=2)  # 同样修改了 'LineWidth' 为 'linewidth'

    plt.xlim([-0.5, 1.5])
    plt.ylim([-0.5, 1.5])
    plt.axis('equal')
    plt.title(name)
    plt.show()


# 主程序
rates = [0.01, 0.1, 1, 5]
epochs = 100
x1 = np.array([[0, 0, 1, 1], [0, 1, 0, 1]])

# AND
y1 = np.array([0, 0, 0, 1])
omega1 = training(x1, y1, rates, epochs, 'AND')
drawDecBou(x1, y1, omega1, 'AND')

# OR
y2 = np.array([0, 1, 1, 1])
```

```
omega2 = training(x1, y2, rates, epochs, 'OR')
drawDecBou(x1, y2, omega2, 'OR')

# COMPLEMENT
x2 = np.array([[0, 1]])
y3 = np.array([1, 0])
omega3 = training(x2, y3, rates, epochs, 'COMPLEMENT')
drawDecBou(x2, y3, omega3, 'COMPLEMENT')

# EXCLUSIVE OR
y5 = np.array([0, 1, 1, 0])
omega5 = training(x1, y5, rates, epochs, 'XOR')
drawDecBou(x1, y5, omega5, 'XOR')
print(omega1,omega2,omega3,omega5)
```

B Q4 代码

```
import numpy as np
import matplotlib.pyplot as plt

# 数据
x = np.array([0, 0.8, 1.6, 3, 4.0, 5.0]).reshape(-1, 1)
y = np.array([0.5, 1, 4, 5, 6, 8])

# 添加偏置项
X_b = np.hstack((np.ones((x.shape[0], 1)), x))

# LLS
rate = 0.02  # 这个rate在LLS中未使用，但保留以与LMS对比
epochs = 200  # 这个epochs在LLS中未使用，但保留以与LMS对比
omega1 = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y

# 绘制 LLS 结果
t = np.linspace(-1, 7, 300).reshape(-1, 1)
ft1 = omega1[0] + omega1[1] * t
plt.figure()
plt.plot(t, ft1, 'b', linewidth=2)
plt.plot(x, y, 'ro', markersize=10)
plt.legend(['LLS results', 'Data points'])
plt.title('LLS method')
plt.show()
```

```
# LMS
omega2 = np.random.rand(2)
bias_history = []
weight_history = []

for i in range(epochs):
    predictions = omega2 @ X_b.T
    error = y - predictions
    omega2 += rate * error @ X_b  # 注意：这里X_b.T是因为我们需要将误差与输入特征（包括偏置
    bias_history.append(omega2[0])
    weight_history.append(omega2[1])

# 绘制 LMS 结果
ft2 = omega2[0] + omega2[1] * t
plt.figure()
plt.plot(t, ft2, 'g', linewidth=2)
plt.plot(x, y, 'ro', markersize=10)
plt.legend(['LMS results', 'Data points'])
plt.title('LMS method')
plt.show()

# 绘制 LMS 权重轨迹
plt.figure()
plt.plot(range(1, epochs + 1), bias_history, 'b', linewidth=2)
plt.plot(range(1, epochs + 1), weight_history, 'r', linewidth=2)
plt.title('LMS weight trajectory (learning rate=0.02)')
plt.legend(['Bias (b)', 'Weight (w)'])
plt.show()
```