Week 7

Q1

1. 写出上面网络的公式：

$$y_1 = \sum_{j=1}^{3} (\cdots)$$

请适当使用下标，例如w_{ij}，使得节点命名相对易读。

$$y_1 = b_1^{(2)} + \sum_{j=1}^{3} w_j^{(2)} t_j$$

$$t_j = \sigma\left(b_j^{(1)} + \sum_{i=1}^{2} w_{ij}^{(1)} x_i\right)$$

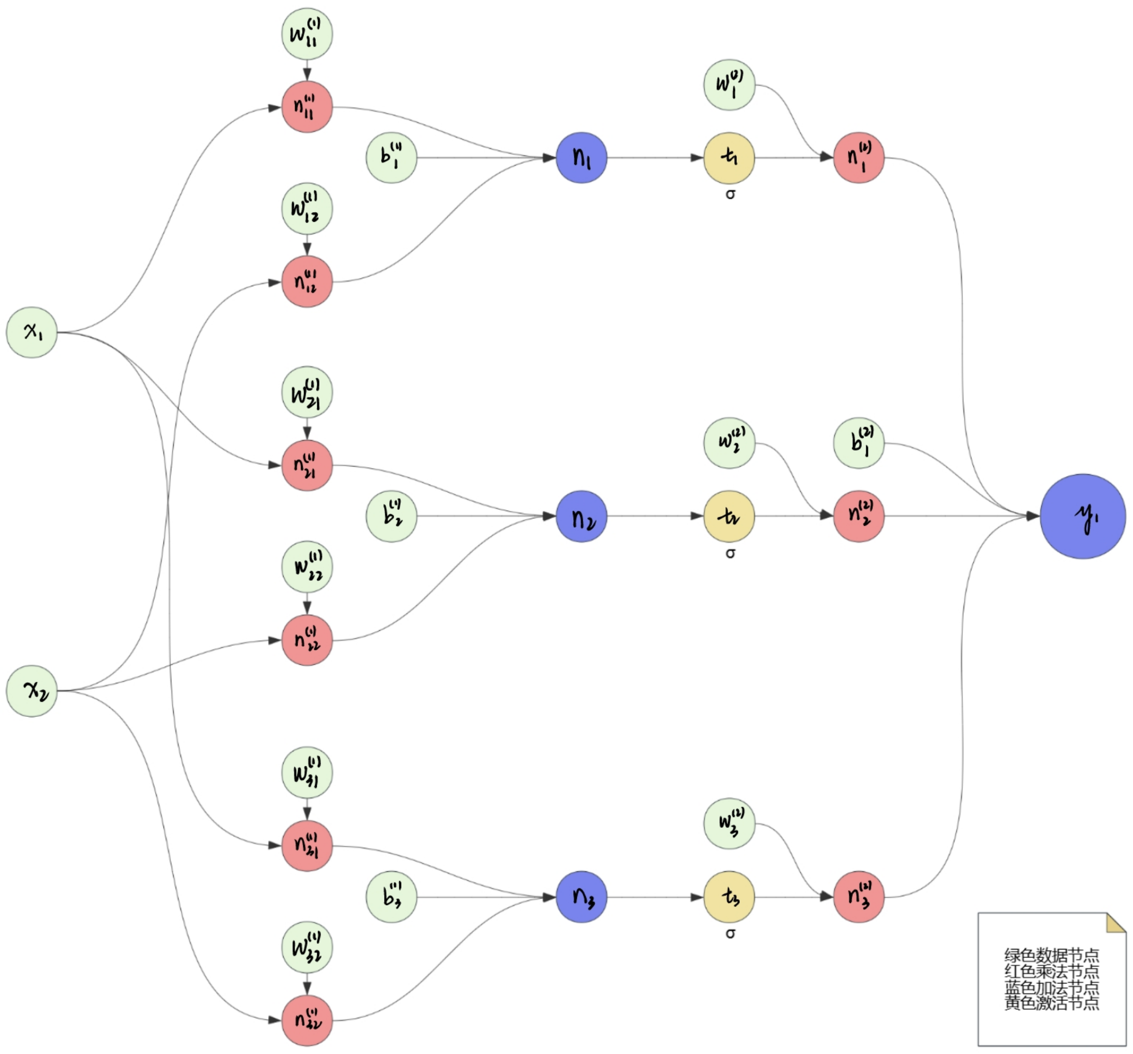
$$y_1 = b_1^{(2)} + \sum_{j=1}^{3} w_j^{(2)} \cdot \sigma\left(b_j^{(1)} + \sum_{i=1}^{2} w_{ij}^{(1)} x_i\right)$$

Q2

2. 画出上述网络的计算图

   计算图中应该包含上一问中命名的 w, b 节点需要指明类型（加法节点、乘法节点、激活函数节点）

你需要在计算图中给出节点的命名，例如n_1，并在后续代码中使用它们。



Q3

3. 将你的网络权重w初始化为1，偏置b初始化为0，并计算前向传播在$(2, -2)$ 处的值。

   答案是一个固定的数字，但是你需要使用代码计算它。

代入公式:

$$y_1 = b_1^{(2)} + \sum_{j=1}^{3} w_j^{(2)} \cdot \sigma\left(b_j^{(1)} + \sum_{i=1}^{2} w_{ij}^{(1)} x_i\right)$$

$$\text{初始化后：} y_1 = \sum_{j=1}^{3} \sigma\left(\sum_{i=1}^{2} x_i\right)$$

$x_1 = 2, x_2 = -2$:

$$y = 0$$

Q4

4. 保持上述权重和偏置的初始化不变，计算输出结果 y_1 关于输入x_1, x_2 在$(2, -2)$的梯度

$$[\frac{\partial y_1}{\partial x_1}(2,-2), \frac{\partial y_1}{\partial x_2}(2,-2)]$$

答案是固定的一个向量，但是你需要使用代码计算它。

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial x_2} = \sum_{j=1}^{3} \sigma' \left(\sum_{i=1}^{2} x_i \right) = 0$$

Q5

5. 以如下函数为目标函数：

$$f(x_1, x_2) = 2x_1 + x_1 x_2, -1 \le x_1, x_2 \le 1$$

利用第五/第六次作业的代码，将网络结构、前向传播过程、梯度反向传播更新替换为计算图的形式，实现对目标函数的拟合任务。数据采样点数、学习率、训练轮数任选。

关键的伪代码结构如下：

```
for epoch = 1 : epoches
    % 所有节点的累计梯度清零
    w_1_11.grad = 0;

    for [x1, x2] in Train_data
        % 前向传播
        n_1.value = n_1_11.value + n_1_21.value + b_1_1.value;

        % 反向传播计算当前梯度
        d_n_1 = double(n_1.value > 0) * d_t_1;

        % 累计梯度
        n_1.grad = n_1.grad + d_n_1;
    end

    % 更新参数
    w_1_11.value = w_1_11.value - lr * w_1_11.grad / N;
end
```

对比Python中的代码（Pytorch为例）：

```
for epoch in range(epoches):
    for data in train_loader:
        inputs, labels = data
        optimizer.zero_grad()                 # 清空梯度缓存
        outputs = model(inputs)               # 前向传播计算输出结果
        loss = criterion(outputs, labels)      # 计算损失值
        loss.backward()                       # 反向传播累计梯度
        optimizer.step()                      # 优化器更新网络参数
```

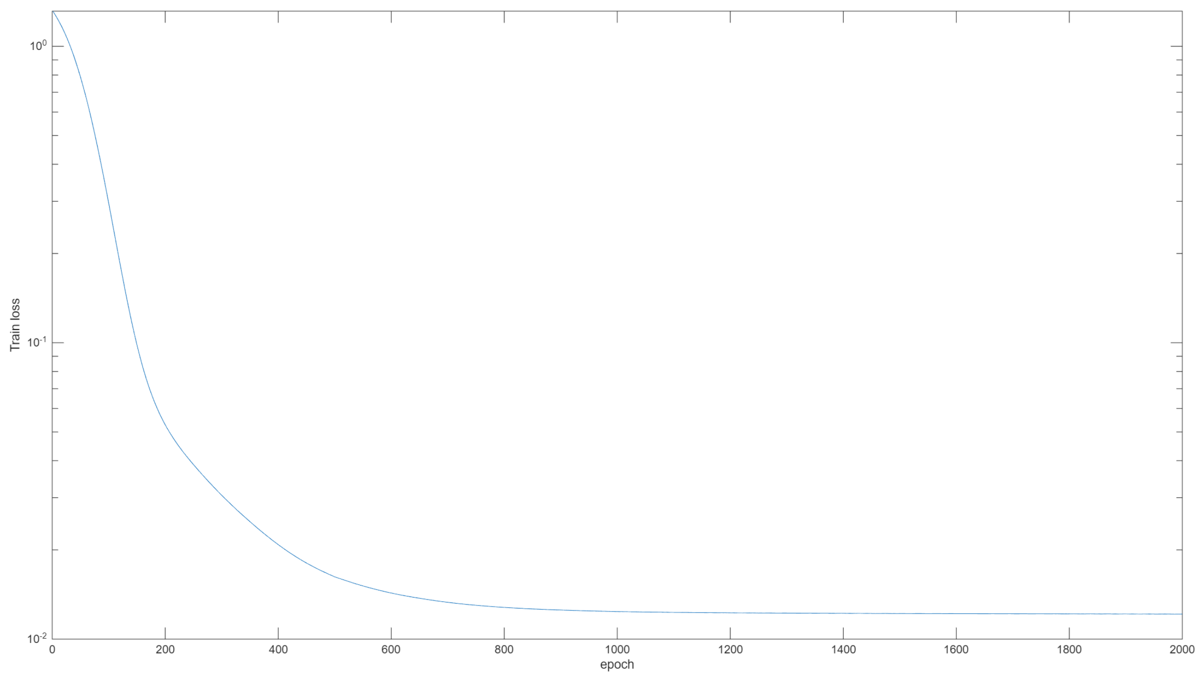
完整代码

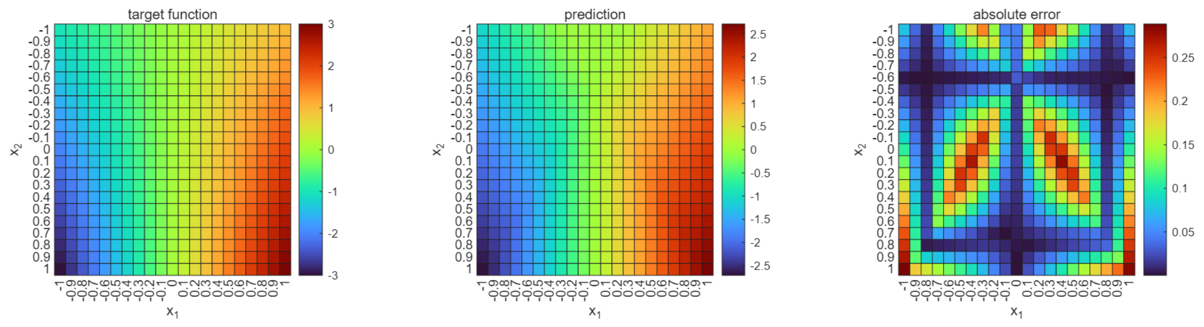
```matlab
clear
clc

rng(42);

f_target = @(x1, x2) 2 .* x1 + x1 .* x2;

N = 21 * 21;
% 构建训练集合
[X1, X2] = meshgrid(linspace(-1, 1, 21), linspace(-1, 1, 21));
X_train = [X1(:), X2(:)];
Y_train = f_target(X_train(:, 1), X_train(:, 2));


s = 0.5;
% 定义网络 第一个位置是上标 后面的是下标
w_1_11.value = s * randn; w_1_11.grad = 0;  % n1
w_1_21.value = s * randn; w_1_21.grad = 0;
b_1_1.value = 0; b_1_1.grad = 0;

w_1_12.value = s * randn; w_1_12.grad = 0;  % n2
w_1_22.value = s * randn; w_1_22.grad = 0;
b_1_2.value = 0; b_1_2.grad = 0;

w_1_13.value = s * randn; w_1_13.grad = 0;  % n3
w_1_23.value = s * randn; w_1_23.grad = 0;
b_1_3.value = 0; b_1_3.grad = 0;

w_2_1.value = s * randn; w_2_1.grad = 0; % layer2
w_2_2.value = s * randn; w_2_2.grad = 0;
w_2_3.value = s * randn; w_2_3.grad = 0;
b_2_1.value = 0; b_2_1.grad = 0;

% 定义中间节点
n_1_11.value = 0; n_1_11.grad = 0;  % n1
n_1_21.value = 0; n_1_21.grad = 0;

n_1_12.value = 0; n_1_12.grad = 0;  % n2
n_1_22.value = 0; n_1_22.grad = 0;

n_1_13.value = 0; n_1_13.grad = 0;  % n3
n_1_23.value = 0; n_1_23.grad = 0;

n_1.value = 0; n_1.grad = 0;
n_2.value = 0; n_2.grad = 0;
n_3.value = 0; n_3.grad = 0;

t_1.value = 0; t_1.grad = 0;
t_2.value = 0; t_2.grad = 0;
t_3.value = 0; t_3.grad = 0;

n_2_1.value = 0; n_2_1.grad = 0;
n_2_2.value = 0; n_2_2.grad = 0;
n_2_3.value = 0; n_2_3.grad = 0;
```

```
y1.value = 0; y1.grad = 0;



epoches = 2000; % 训练轮数
lr = 1e-2;


train_loss = zeros(1, epoches);

for epoch = 1:epoches
    % 当前轮次中计算前向传播的值

    % ------------ 梯度清零 --------------
    % layer1
    w_1_11.grad = 0; w_1_21.grad = 0; b_1_1.grad = 0; % n1
    w_1_12.grad = 0; w_1_22.grad = 0; b_1_2.grad = 0; % n2
    w_1_13.grad = 0; w_1_23.grad = 0; b_1_3.grad = 0; % n3

    % layer2
    w_2_1.grad = 0; w_2_2.grad = 0; w_2_3.grad = 0;
    b_2_1.grad = 0;

    % 中间节点
    n_1_11.grad = 0; n_1_21.grad = 0;
    n_1_12.grad = 0; n_1_22.grad = 0;
    n_1_13.grad = 0; n_1_23.grad = 0;
    n_1.grad = 0; n_2.grad = 0; n_3.grad = 0;
    t_1.grad = 0; t_2.grad = 0; t_3.grad = 0;
    n_2_1.grad = 0; n_2_2.grad = 0; n_2_3.grad = 0;
    y1.grad = 0;

    if mod(epoch, 500) == 0
        lr = lr * 0.8;
    end

    % 遍历数据点
    for i = 1 : N
        x1 = X_train(i, 1);
        x2 = X_train(i, 2);
        y_target = Y_train(i);

        % ------------ 前向传播 --------------
        % 乘法节点
        n_1_11.value = x1 * w_1_11.value;  % n1
        n_1_21.value = x2 * w_1_21.value;

        n_1_12.value = x1 * w_1_12.value;  % n2
        n_1_22.value = x2 * w_1_22.value;

        n_1_13.value = x1 * w_1_13.value;  % n3
        n_1_23.value = x2 * w_1_23.value;

        % 加法节点
        n_1.value = n_1_11.value + n_1_21.value + b_1_1.value;
        n_2.value = n_1_12.value + n_1_22.value + b_1_2.value;
        n_3.value = n_1_13.value + n_1_23.value + b_1_3.value;
        % 激活函数节点
        t_1.value = max(n_1.value, 0);
        t_2.value = max(n_2.value, 0);
        t_3.value = max(n_3.value, 0);

        % 第二层乘法节点
        n_2_1.value = t_1.value * w_2_1.value;
        n_2_2.value = t_2.value * w_2_2.value;
        n_2_3.value = t_3.value * w_2_3.value;

        % 第二层加法节点，最终输出
        y1.value = n_2_1.value + n_2_2.value + n_2_3.value + b_2_1.value;

        % 累加到当前的训练误差中
        train_loss(epoch) = train_loss(epoch) + (y1.value - f_target(x1,
x2)).^2;


        % ------------ 反向传播 --------------

        % 输出层的梯度
        d_y1 = 2 .* (y1.value - f_target(x1, x2));


        % 第二层乘法节点梯度
        d_n_2_1 = d_y1;
        d_n_2_2 = d_y1;
        d_n_2_3 = d_y1;

        % 第二层偏置梯度
        d_b_2_1 = d_y1;

        % 第二层激活函数节点梯度
        d_t_1 = w_2_1.value * d_n_2_1;
        d_t_2 = w_2_2.value * d_n_2_2;
        d_t_3 = w_2_3.value * d_n_2_3;
        % 第二层权重梯度
        d_w_2_1 = t_1.value * d_n_2_1;
        d_w_2_2 = t_2.value * d_n_2_2;
        d_w_2_3 = t_3.value * d_n_2_3;

        % 第二层加法节点梯度
        d_n_1 = double(n_1.value > 0) * d_t_1;
        d_n_2 = double(n_2.value > 0) * d_t_2;
        d_n_3 = double(n_3.value > 0) * d_t_3;

        % 第一层偏置梯度
        d_b_1_1 = d_n_1;
        d_b_1_2 = d_n_2;
        d_b_1_3 = d_n_3;

        % 第一层乘法节点梯度
        d_n_1_11 = d_n_1;
        d_n_1_21 = d_n_1;
        d_n_1_12 = d_n_2;
        d_n_1_22 = d_n_2;
        d_n_1_13 = d_n_3;
        d_n_1_23 = d_n_3;
        % 第一层权重梯度
        d_w_1_11 = x1 * d_n_1_11;
        d_w_1_21 = x2 * d_n_1_21;

        d_w_1_12 = x1 * d_n_1_12;
        d_w_1_22 = x2 * d_n_1_22;

        d_w_1_13 = x1 * d_n_1_13;
        d_w_1_23 = x2 * d_n_1_23;


        % 累计梯度
        w_1_11.grad = w_1_11.grad + d_w_1_11;
        w_1_21.grad = w_1_21.grad + d_w_1_21;
        b_1_1.grad = b_1_1.grad + d_b_1_1;

        w_1_12.grad = w_1_12.grad + d_w_1_12;
        w_1_22.grad = w_1_22.grad + d_w_1_22;
        b_1_2.grad = b_1_2.grad + d_b_1_2;

        w_1_13.grad = w_1_13.grad + d_w_1_13;
        w_1_23.grad = w_1_23.grad + d_w_1_23;
        b_1_3.grad = b_1_3.grad + d_b_1_3;

        w_2_1.grad = w_2_1.grad + d_w_2_1;
        w_2_2.grad = w_2_2.grad + d_w_2_2;
        w_2_3.grad = w_2_3.grad + d_w_2_3;

        b_2_1.grad = b_2_1.grad + d_b_2_1;
    end

    train_loss(epoch) = train_loss(epoch) / N;



    % ------------ 参数更新 --------------
    % 梯度平均放在更新部分
    w_1_11.value = w_1_11.value - lr * w_1_11.grad / N;
    w_1_21.value = w_1_21.value - lr * w_1_21.grad / N;
    b_1_1.value = b_1_1.value - lr * b_1_1.grad / N;

    w_1_12.value = w_1_12.value - lr * w_1_12.grad / N;
    w_1_22.value = w_1_22.value - lr * w_1_22.grad / N;
    b_1_2.value = b_1_2.value - lr * b_1_2.grad / N;

    w_1_13.value = w_1_13.value - lr * w_1_13.grad / N;
    w_1_23.value = w_1_23.value - lr * w_1_23.grad / N;
    b_1_3.value = b_1_3.value - lr * b_1_3.grad / N;

    w_2_1.value = w_2_1.value - lr * w_2_1.grad / N;
    w_2_2.value = w_2_2.value - lr * w_2_2.grad / N;
    w_2_3.value = w_2_3.value - lr * w_2_3.grad / N;

    b_2_1.value = b_2_1.value - lr * b_2_1.grad / N;
end
```

```
% 计算结果并绘图

net_pre = zeros(21, 21);
for x1 = -1:0.1:1
    for x2 = -1:0.1:1
        % 乘法节点
        n_1_11.value = x1 * w_1_11.value;  % n1
        n_1_21.value = x2 * w_1_21.value;

        n_1_12.value = x1 * w_1_12.value;  % n2
        n_1_22.value = x2 * w_1_22.value;

        n_1_13.value = x1 * w_1_13.value;  % n3
        n_1_23.value = x2 * w_1_23.value;

        % 加法节点
        n_1.value = n_1_11.value + n_1_21.value + b_1_1.value;
        n_2.value = n_1_12.value + n_1_22.value + b_1_2.value;
        n_3.value = n_1_13.value + n_1_23.value + b_1_3.value;

        % 激活函数节点
        t_1.value = max(n_1.value, 0);
        t_2.value = max(n_2.value, 0);
        t_3.value = max(n_3.value, 0);

        % 第二层乘法节点
        n_2_1.value = t_1.value * w_2_1.value;
        n_2_2.value = t_2.value * w_2_2.value;
        n_2_3.value = t_3.value * w_2_3.value;

        % 第二层加法节点，最终输出
        y1.value = n_2_1.value + n_2_2.value + n_2_3.value + b_2_1.value;

        % 注意行列
        net_pre(round((x2 + 1.1) / 0.1), round((x1 + 1.1) / 0.1)) = y1.value;
    end
end



x = -1:0.1:1;
y = -1:0.1:1;

[X, Y] = meshgrid(x, y);

f_true = f_target(X, Y);



figure(1)
semilogy(train_loss)


figure('Position', [100, 300, 1400, 300])

subplot(1, 3, 1)
```

```
heatmap(f_true, "Colormap", turbo)

subplot(1, 3, 2)
heatmap(net_pre, "Colormap", turbo)

subplot(1, 3, 3)
heatmap(abs(net_pre - f_true), "Colormap", turbo)
```


Train loss
epoch
10^0
10^{-1}
10^{-2}


target function
prediction
absolute error
x_1
x_2