DeepGlint AI编程练习赛: 对抗性攻击

题目大意

给定神经网络表达式：

$$Y = \text{softmax}(W_2 \times \text{ReLU}(W_1 \times X))$$

其中X 是N维向量

W_1 为$M \times N$ 矩阵

$\text{ReLU}(x) = \max(0, x)$

W_2为$10 \times M$矩阵

softmax 操作定义如下

$$Y_i = \frac{e^{z_i - z_{max}}}{\sum_{j=1}^{10} e^{z_j - z_{max}}}$$

假设输入数据 X 的每个尺寸上都都只能 [-128,127] 范围的整数，以下操作都只考虑只修改 X 的某些维的情况，不能同时修改多个维度的值：

1. 如果存在 X 的某个维度（假设是第 i 维），当它的值修改为 [-128,127] 范围的某个单一后，网络输出的类别（即 softmax 后概率最高的位置）跟原始 X 的分类结果评分发生了改变，且通常为新类别的预测概率最大，则 i 为敏感度最高的位置。
2. 如果对于 X 的任意一个位置，把它的值进行 [-128,127] 范围的任意修改，网络输出的类别都不会发生改变，那么我们把可以进行网络输出概率可以降到最低的那个修改所对应位置称为敏感度最高的位置。
3. 测试样例中，原始网络的输出中只会有一个类别概率最高，不会出现多个类别概率同时最高的情况，X 中也不会出现存在多个敏感度最高位置。

输入格式

输入 4 行。

第一行是两个整数 N，M。代表输入向量的尺寸为 N，第一个隐藏层的异步数为 M。

第二行有 N 个整数，以空格隔开。即为输入的向量。

即第一个全连接层的网络参数 W1。其中第`(i-1)* N + 1`到第`i * N`个浮点数是第一一个全连接层第 i 个上游的权重参数。

第四行有 `10 * M` 个浮点数，以空格隔开。即第二个全连接层的网络参数 W2。其中第`(i-1)* M + 1` 到第`i * M` 个浮点数是第二一个全连接层第 i 个上游的权重参数。

约束条件

60％的测试样例满足如下条件：

第一行是一个整数 N，取值范围：1 <= N <= 100，1 <= M <100

第二行的 N 个整数 Ni，取值范围：-128 <= Ni <128

第三行的浮点数 fi，取值范围：-8.0 <= fi <= 8.0

第四行的浮点数 fj，取值范围：-8.0 <= fj <= 8.0

剩下 40％的测试样例满足如下条件：

第一行是一个整数 N，取值范围：1 <= N <= 1000，1 <= M <100

第二行的 N 个整数 Ni，取值范围：-128 <= Ni <128

第三行的浮点数 fi，取值范围：-8.0 <= fi <= 8.0

第四行的浮点数 fj，取值范围：-8.0 <= fj <= 8.0

输出格式

输出为 2 个整数 P，V，以空格隔开。其中 P 是 [1，N] 的某个整体，代表输入向量 X 中敏感度最高的位置。V 表示 X 的敏感度最高的位置的数字应该被修改成 [-128,127] 中的哪个数字，造成网络干扰最大（即：如果预测类别被改变了，怎么样改会概率最高；如果预测类别不变，如何改现行的类别的预测概率最小）。

测试用例:

```
输入：
8 4
-4 -71 -56 -41 85 -19 -56 -3
0.00719 0.01590 -0.01121 -0.02345 0.00777 0.01680 0.01642
-0.01437 0.04963 -0.02698 -0.03168 -0.02930 0.00784 -0.03372
-0.01824 0.01997 -0.01687 -0.02018 -0.00434 -0.00647- 0.01860
-0.01780 -0.01345 0.03369 0.00142 -0.00109 -0.02072 0.00518
-0.02600 -0.01217 -0.00510 -0.00254
-0.00372 0.06219 0.00260 0.06550 -0.02418 -0.02375 0.00115
0.00132 0.00280 -0.01428 0.02612 -0.03527 -0.02926 -0.02194
-0.04160 0.03126 0.01071 0.02239 0.00883 0.03610 0.00117
0.00429 -0.05671 0.00374 0.03496 0.03749 0.03426 0.01259
0.01202 -0.00021 -0.04738 -0.02131 0.02525 0.04419 -0.01626
0.04310 -0.01328 -0.00932 -0.03152 0.06103
输出：
1 -77
```

思路

> *增量计算是一种软件功能 。当一部分的数据产生了变化，就仅对该产生变化的部分进行计算和更新，以节省计算时间。相比于简单地重复计算完整的输出内容，增量计算能够显著地节省计算时间。比如，电子表格会在实现重计算功能时使用增量计算，只重新计算并更新那些含有公式且被直接或间接地改变了的单元格。*

首先分析矩阵乘法复杂度，假设矩阵A为$N \times M$，矩阵B为$M \times C$，计算$A \times B$:

1. A中第一行元素与B中第一列元素对应相乘M次乘法运算，M次加法运算
2. 最终结果为$N \times C$矩阵，因此共$N \times C \times M$次乘法和相同次数的加法运算

复杂度为$O(n^3)$

对于这道题，对每个输入的节点，需要枚举$[-128, 127]$范围内的所有数字。对于最恶劣的测试用例，时间复杂度为

$$N \times 256 \times ((M \times N \times N) + 10 \times M \times M)$$

约为10^{13}肯定不满足要求。

时间主要耗费在第一个全链接层，由于输入节点扩增为1000个，导致计算复杂度非线性上升。

考虑第一个全链接层的公式;

$$f(X) = W_1 \times X$$

如果更改输入的一个节点：

$$f(X') = W_1 \times X' = W_1 \times (X + \Delta X)$$
$$= W_1 \times X + W_1 \times [0, 0, 0, \delta, \ldots, 0]^T$$

前者只需计算一次，后者是对W_1矩阵提取某一列，只需在原始基础上，多了M次加法和M次乘法。

更进一步：假设枚举第一个输入节点的值，枚举范围为[-128, 128]， 原始第一个节点为10，可以先计算第一个节点为-129时结果：

$f(X') = W_1 \times X + W_1 \times [-139, 0, 0, \ldots, 0]^T$

定义 $W_1|_j$表示其第j列，那后续可以去掉乘法：

$$i = -128, \ldots 127$$
$$f(X_i) = f(X_{i-1}) + W_i|_0$$

此时时间复杂度为

$$N \times 256 \times ((M) + 10 \times M \times M)$$

约为10^{10}

代码

Python慎用append

```python
import math


def solve(n, m, fc1, fc2, x):
    w1 = [[0 for _ in range(n)] for _ in range(m)]
    w2 = [[0 for _ in range(m)] for _ in range(10)]
    for i in range(m):
        for j in range(n):
            w1[i][j] = fc1[i * n + j]
    for i in range(10):
        for j in range(m):
            w2[i][j] = fc2[i * m + j]
    w1x = [0] * m
    for i in range(m):
        for j in range(n):
            w1x[i] += w1[i][j] * x[j]
    w1x_relu = [0] * m
    for i in range(m):
```

```
        w1x_relu[i] = max(0, w1x[i])

    w2x = [0] * 10
    for i in range(10):
        for j in range(m):
            w2x[i] += w2[i][j] * w1x_relu[j]
    z_max = max(w2x)
    org_class = w2x.index(z_max)
    w2x_exp = [0] * 10
    for i in range(10):
        w2x_exp[i] = math.exp(w2x[i] - z_max)
    z_sum = sum(w2x_exp)
    w2x_softmax = [0] * 10
    for i in range(10):
        w2x_softmax[i] = w2x_exp[i] / z_sum
    probilities = [0] * 10
    probilities[org_class] = w2x_softmax[org_class]

    tmp_w1x_relu = [0] * m
    tmp_w2x = [0] * 10
    tmp_w2x_exp = [0] * 10
    tmp_w2x_softmax = [0] * 10
    pv = [None] * 10
    for i in range(n):
        tmp_w1x = w1x[:]
        delta = -129 - x[i]
        for k in range(m):
            tmp_w1x[k] += delta * w1[k][i]
        for val in range(-128, 128):
            for k in range(m):
                tmp_w1x[k] += w1[k][i]
            for k in range(m):
                tmp_w1x_relu[k] = max(tmp_w1x[k], 0)
            for i_ in range(10):
                tmp_w2x[i_] = 0
                for j_ in range(m):
                    tmp_w2x[i_] += w2[i_][j_] *
tmp_w1x_relu[j_]
            tmp_z_max = max(tmp_w2x)
            tmp_class = tmp_w2x.index(tmp_z_max)
            for k in range(10):
                tmp_w2x_exp[k] = math.exp(tmp_w2x[k] -
tmp_z_max)
            tmp_z_sum = sum(tmp_w2x_exp)
            for k in range(10):
```

```
                tmp_w2x_softmax[k] = tmp_w2x_exp[k] / tmp_z_sum
            if tmp_class == org_class:
                if tmp_w2x_softmax[tmp_class] <= probilities[tmp_class]:
                    probilities[tmp_class] = tmp_w2x_softmax[tmp_class]
                    pv[tmp_class] = (i + 1, val)

            else:
                if tmp_w2x_softmax[tmp_class] >= probilities[tmp_class]:
                    probilities[tmp_class] = tmp_w2x_softmax[tmp_class]
                    pv[tmp_class] = (i + 1, val)
    cnt = 0
    for idx, val in enumerate(pv):
        if val is not None:
            cnt += 1
    if cnt == 1:
        print('{} {}'.format(*pv[org_class]))
    else:
        ret = 0
        prep = 0
        for idx, val in enumerate(probilities):
            if idx == org_class:
                continue
            if val > prep:
                prep = val
                ret = idx
        print('{} {}'.format(*pv[ret]))


if __name__ == '__main__':
    N, M = map(int, input().split())
    X = list(map(int, input().split()))
    lc1 = list(map(float, input().split()))
    lc2 = list(map(float, input().split()))
    solve(N, M, lc1, lc2, X)

```