Assignment #4: 位操作、栈、链表、堆和 NN

Updated 1203 GMT+8 Mar 10, 2025

2025 spring, Complied by <mark>汤伟杰，信息管理系</mark>

> **说明：**
>
> 1. **解题与记录：**
>
>   对于每一个题目，请提供其解题思路（可选），并附上使用 Python 或 C++编写的源代码（确保已在 OpenJudge， Codeforces，LeetCode 等平台上获得 Accepted）。请将这些信息连同显示“Accepted”的截图一起填写到下方的作业模板中。（推荐使用 Typora https://typoraio.cn 进行编辑，当然你也可以选择 Word。）无论题目是否已通过，请标明每个题目大致花费的时间。
>
> 2. **提交安排：**提交时，请首先上传 PDF 格式的文件，并将.md 或.doc 格式的文件作为附件上传至右侧的“作业评论”区。确保你的 Canvas 账户有一个清晰可见的头像，提交的文件为 PDF 格式，并且“作业评论”区包含上传的.md 或.doc 附件。
>
> 3. **延迟提交：**如果你预计无法在截止日期前提交作业，请提前告知具体原因。这有助于我们了解情况并可能为你提供适当的延期或其他帮助。
>
> 请按照上述指导认真准备和提交作业，以保证顺利完成课程要求。

1. 题目

136.只出现一次的数字

bit manipulation, https://leetcode.cn/problems/single-number/

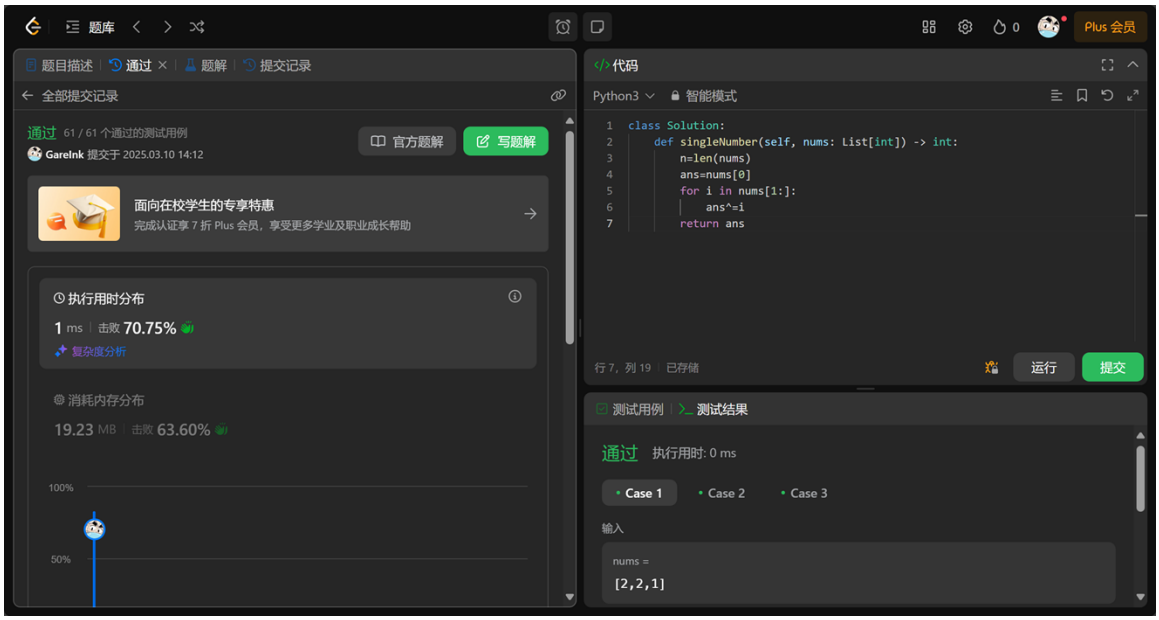
<mark>请用位操作来实现，并且只使用常量额外空间。</mark>

思路：这道题目感觉专门为了异或出的，只需要知道两个相同数字的异或结果是 0；0 与任何数字异或结果是这个数字本身即可。

代码：

```
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        n=len(nums)
        ans=nums[0]
        for i in nums[1:]:
            ans^=i
        return ans
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



20140:今日化学论文

stack, http://cs101.openjudge.cn/practice/20140/

[['一花', '一世界'], ['一叶', '一菩提']]

思路：感觉自己这道题目写的代码好丑，刚开始只看样例还以为所有的文字都是在[]里面，结果看了测试数据才发现只有要压缩的字母才会有括号，其他的都是直接给出的字符串，所以我在输入字符串后就在前后自行加上了一对括号（因为我是按照这个格式想出来的思路）。

大致思路是这样的：我想维护一个这样的栈：

```
stack = ['num', 'letters']
```

就是严格保持一个数字+一个字母的格式，**与题目中说的[xs]保持一致**。对于输入，

(1)遇到左括号或者数字压入栈中；

(2)遇到字母，为了保持维护的栈的'num'+'letters'的格式，

1. 如果此时栈的最后一个元素是 letters，就把这个字母直接补到这个元素末尾，**延长了这个 letters**；
2. 如果不是，就直接压入。

(3)遇到右括号，整体思路是一直 pop 元素直到找到匹配的左括号，有两种情况：

1. 这个右括号是整个字符串最末尾的那个，**由于栈已经严格维护**，只需要先 pop 一次得到 letters，再 pop 若干次得到 num，然后这时栈中就只剩下最左侧的左括号了，把 num*letters 赋给 ans 就是答案；

2. 这个右括号是字符串中间的某一个，根据题意这个右括号所对应的括号结构与维护的栈的结构是一样的，所以接下来的操作与 1 是一样的，然后得到的 num*letters 就和**(2)**中处理字母的情况是一样的了。

我觉得这个思路在我代码写完之后叙述起来还是有点绕，所以觉得我的代码写的很丑陋。

代码：

```
s=input()
s='['+s+']'
stack=[]
ans=''

for i in s:
    if i=='[':
        stack.append(i)
    elif i==']':
        letters=stack.pop()
        num=''
        while stack[-1].isdigit(): # 防止出现 1
            num=stack.pop()+num
        num=int(num) if num else 1

        #把'['pop 掉
        stack.pop()
        if stack:
            if stack[-1][-1].isalpha():
                stack[-1]+=num*letters
            else:
                stack.append(num*letters)
        else:
            ans=num*letters

    # 保证字母连成一片 letters，['num', 'letters', ['num', 'letters']]
    elif i.isalpha():
        if stack[-1][-1].isalpha():
            stack[-1]+=i
        else:
            stack.append(i)
    else:
        stack.append(i)
print(ans)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

#48524463提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
s=input()
s='['+s+']'
stack=[]
ans=''
for i in s:
    if i=='[':
        stack.append(i)
    elif i==']':
        letters=stack.pop()
        num=''
        while stack[-1].isdigit(): # 防止出现1
            num=stack.pop()+num
        num=int(num) if num else 1
        #把[pop掉
        stack.pop()
        if stack:
            if stack[-1][-1].isalpha():
                stack[-1]+=num*letters
            else:
                stack.append(num*letters)
        else:
            ans=num*letters

    # 保证['num', 'letters', ['num', 'letters']]
    elif i.isalpha():
        if stack[-1][-1].isalpha():
            stack[-1]+=i
        else:
            stack.append(i)
    else:
        stack.append(i)
print(ans)
```

基本信息

#: 48524463
题目: 20140
提交人: 24n2400016635
内存: 3824kB
时间: 28ms
语言: Python3
提交时间: 2025-03-11 18:51:42

English 帮助 关于

160.相交链表

linked list, https://leetcode.cn/problems/intersection-of-two-linked-lists/

思路：这道题的题目刚开始没读懂，看到了题目评论区才发现：要找的节点是指**地址相同**，而不是仅仅判断 node1.val == node2.val，这个在第一个样例里面也有显现。自己没想到怎么用双指针，用集合来存 A 的然后依次判断 B 的感觉很直观，看了题解感觉这种在两个链表**循环**的双指针真的太神奇了，A 的走完了就继续走 B，B 的走完了就继续走 A，即使二者没有相交节点，最后也会在遍历完成时返回 None==None。

同时结合了评论区的内容对空间复杂度为 O(1)的代码变量命名进行了修改。

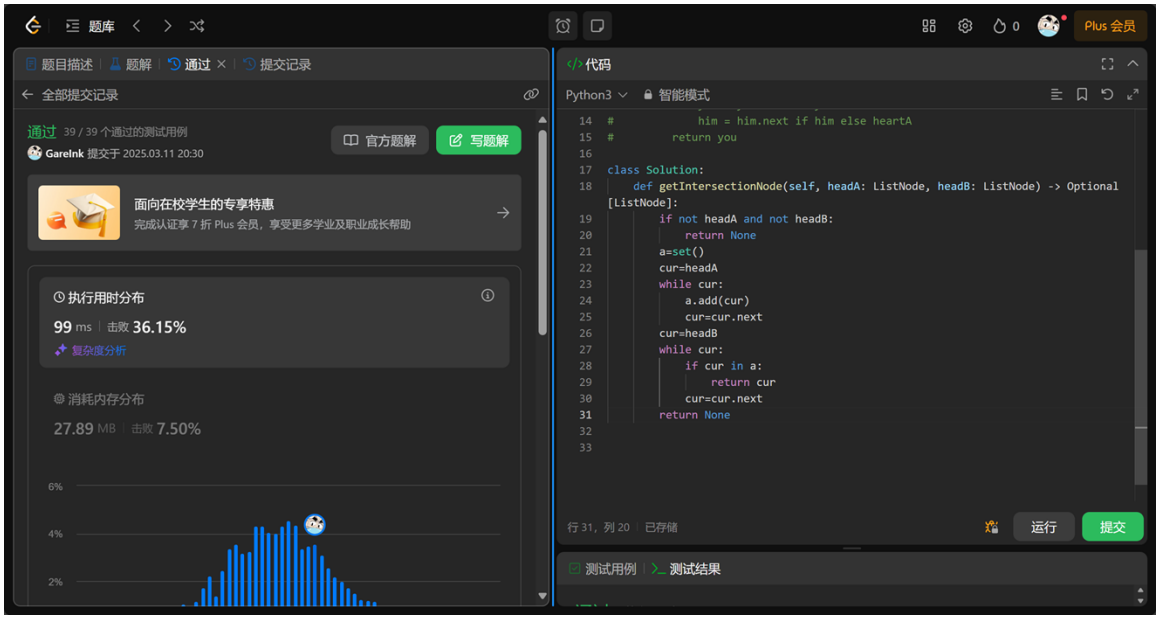
代码：

```
# 空间复杂度：O(n)
class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) ->
Optional[ListNode]:
        if not headA and not headB:
            return None
        a=set()
        cur=headA
        while cur:
            a.add(cur)
            cur=cur.next
```

```
        cur=headB
        while cur:
            if cur in a:
                return cur
            cur=cur.next
        return None

# 空间复杂度：O(1)
class Solution:
    def getIntersectionNode(self, heartA: ListNode, heartB: ListNode) ->
Optional[ListNode]:
        if not heartA or not heartB:
            return None
        you, him = heartA, heartB
        while you != him:
            you = you.next if you else heartB
            him = him.next if him else heartA
        return you
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



206.反转链表

linked list, https://leetcode.cn/problems/reverse-linked-list/

思路：这是在寒假期间做的，当时被递归的写法迷惑了一下午。递归的思路妙就妙在这个函数的返回值上：

在链表不为空或者长度不为 1 的时候，这个函数体的第一行就进行了函数调用，并将返回值赋给了 reversed_head；然后整个函数返回的也是这个 reversed_head。经过一阵思考才发现，哦，原来这一行函数调用，实际上是调用了 n 次（n 是链表长度-1），也就是说，一旦调用了这个函数，就会一直调用到这个链表的最后一个节点，才会达到退出条件，并返回这个最终节点。而实际上，它就是想要的最终返回结果，因此把这个节点直接赋值给 reversed_head，同时通过将整个函数的返回值也设置为这个变量以保证在回溯时该变量能够不断**原地赋值**（自己给自己赋值）。因此，在调用递归函数这一行的下面两行，实际上是在对当前递归到的节点的**下一个节点的 next 指针**调整到**自己身上**，并且把**自己的当前 next 指针**调整为 **None**，这样也保证了链表的单向性。总之，这个递归很烧脑，但是想清楚了很通透！

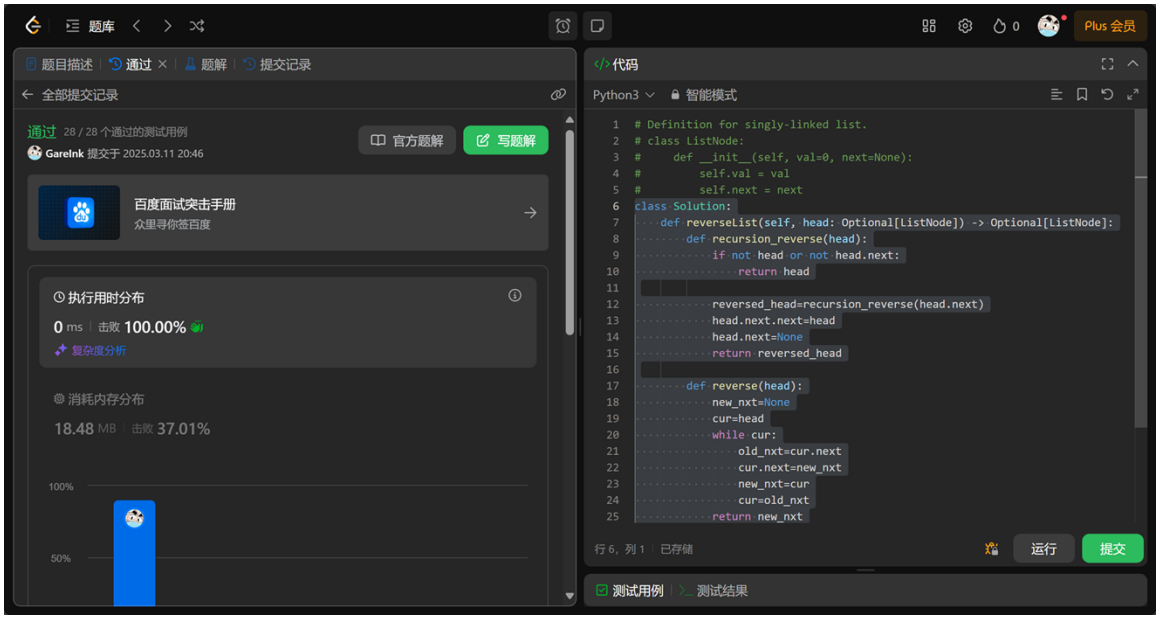
代码：

```
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        #回溯写法
        def recursion_reverse(head):
            if not head or not head.next:
                return head
                    #这一行的递归调用直接递归到了尾节点并返回最终结果
            reversed_head=recursion_reverse(head.next)
            #这两行在回溯时从后往前调整各个节点的 next 指针
            head.next.next=head
            head.next=None
            return reversed_head

        #迭代写法
        def reverse(head):
            new_nxt=None
            cur=head
            while cur:
                old_nxt=cur.next
                cur.next=new_nxt
                new_nxt=cur
                cur=old_nxt
            return new_nxt

        return recursion_reverse(head)
        # return reverse(head)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



3478.选出和最大的 K 个元素

heap, https://leetcode.cn/problems/choose-k-elements-with-maximum-sum/

思路：当时比赛时候遇到这个第二题直接放弃了，力扣出题太狠了。。

首先将 nums1 中的元素排序，同时保留各自的原始索引方便取 nums2 中的值，原因是 1 中大的元素显然是可以使用比它小的元素所对应的 nums2 中的值的，相当于不断复用之前的值结束后把自己对应的值再填进去。

然后维护一个最小堆，用来不断存 nums2 中的元素；维护一个 tota_sum 来维护当前最多 k 个元素的和；这两个变量结合起来的作用是：一旦 heap 长度大于 k 了，就弹出一个最小值，tota_sum 减去它；同时每次遍历一个值就用 tota_sum 加上。

同时为了处理样例 2 中有多个相同的值在 nums1 中的情况，维护 prev_val 和 prev_ans 这两个变量，一旦发现当前 nums1 中的 value 与 prev_value 相同就直接把 prev_ans 赋给当前遍历的索引。

好难啊啊啊啊啊。。。

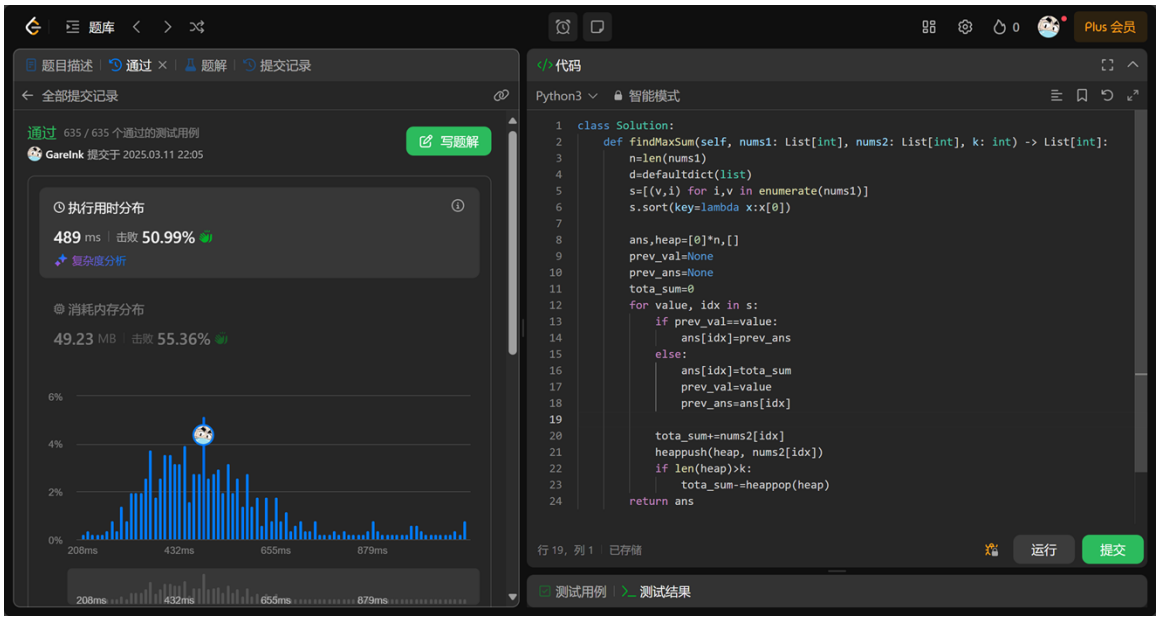
代码：

```
class Solution:
    def findMaxSum(self, nums1: List[int], nums2: List[int], k: int) ->
List[int]:
        n=len(nums1)
        d=defaultdict(list)
        s=[(v,i) for i,v in enumerate(nums1)]
        s.sort(key=lambda x:x[0])

        ans,heap=[0]*n,[]
        prev_val=None
        prev_ans=None
        tota_sum=0
        for value, idx in s:
            if prev_val==value:
                ans[idx]=prev_ans
            else:
                ans[idx]=tota_sum
                prev_val=value
                prev_ans=ans[idx]

            tota_sum+=nums2[idx]
            heappush(heap, nums2[idx])
            if len(heap)>k:
                tota_sum-=heappop(heap)
        return ans
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



Q6.交互可视化 neural network

https://developers.google.com/machine-learning/crash-course/neural-networks/interactive-exercises

Your task: configure a neural network that can separate the orange dots from the blue dots in the diagram, achieving a loss of less than 0.2 on both the training and test data.

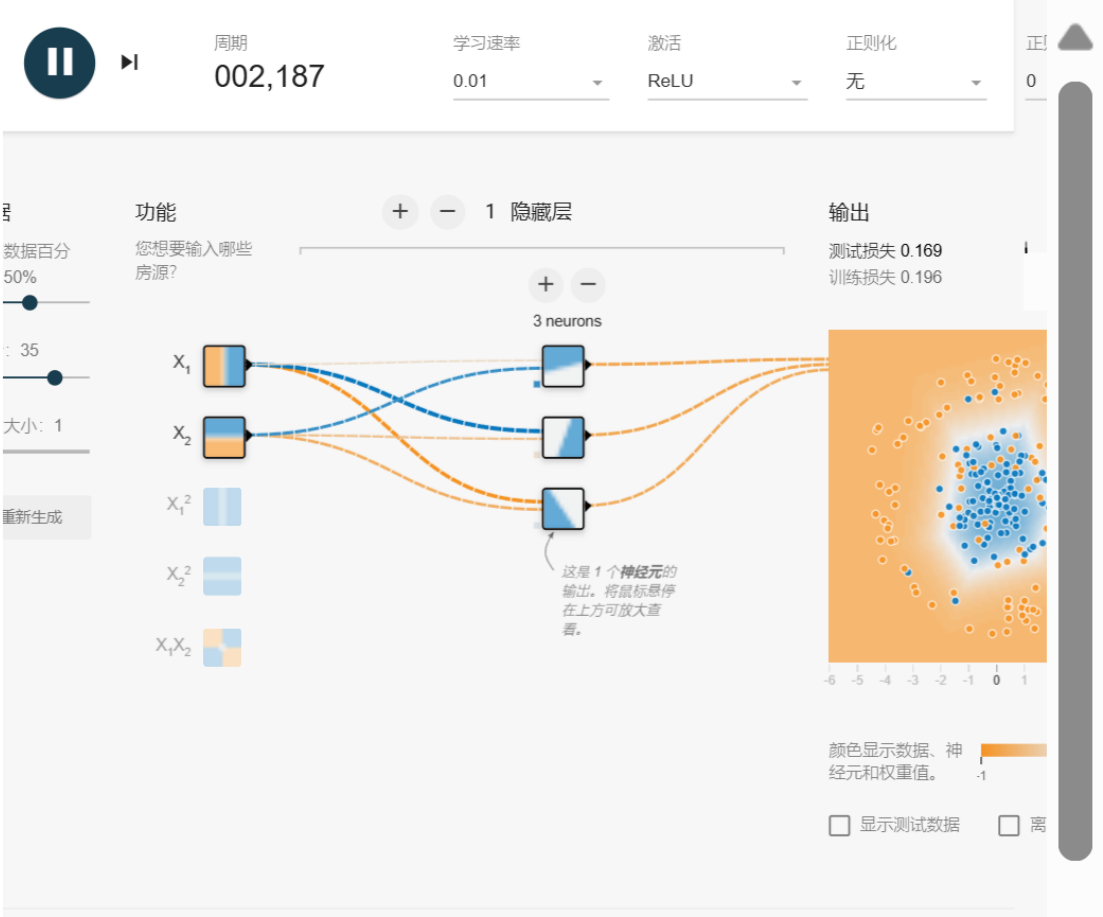
Instructions:

In the interactive widget:

1. Modify the neural network hyperparameters by experimenting with some of the following config settings:
    - Add or remove hidden layers by clicking the **+** and **-** buttons to the left of the **HIDDEN LAYERS** heading in the network diagram.
    - Add or remove neurons from a hidden layer by clicking the **+** and **-** buttons above a hidden-layer column.
    - Change the learning rate by choosing a new value from the **Learning rate** drop-down above the diagram.
    - Change the activation function by choosing a new value from the **Activation** drop-down above the diagram.
2. Click the Play button above the diagram to train the neural network model using the specified parameters.
3. Observe the visualization of the model fitting the data as training progresses, as well as the **Test loss** and **Training loss** values in the **Output** section.

4. If the model does not achieve loss below 0.2 on the test and training data, click reset, and repeat steps 1–3 with a different set of configuration settings. Repeat this process until you achieve the preferred results.

给出满足约束条件的<mark>截图</mark>，并说明学习到的概念和原理。



①仅仅通过增加隐藏层和神经元是不足以让这个网络进行非线性学习的，需要通过激活函数的作用来对每一次的输出进行非线性的映射来得到输出，最直观的地方就在这个计算过程中：

CALCULATIONS

$= \text{Sigmoid}(w_{11} x_1 + w_{21} x_2 + w_{31} x_3 + w_{41} x_4 + b)$

$= \text{Sigmoid}(0.04 * 0.17 + -0.45 * 0.46 + 0.41 * 0.73 + 0.09 * 0.20 + 0.69)$

$= \text{Sigmoid}(0.01 + -0.21 + 0.30 + 0.02 + 0.69)$

$= \text{Sigmoid}(0.81)$

$= 0.69$

在 s 型激活函数内部仍然是 weight 和 bias 两个参数与输入的线性计算，但是最后的 0.81 会经过非线性的激活函数来得到一个非线性的结果。

②常见的激活函数：sigmoid function，tanh function，ReLU function（这个名字听起来很高级结果公式这么简单，F(x)=max(0,x)）

2. 学习总结和收获

<mark>如果发现作业题目相对简单，有否寻找额外的练习题目，如“数算 2025spring 每日选做”、LeetCode、Codeforces、洛谷等网站上的题目。</mark>

这次作业的几个题目如果都是第一次独立做，我感觉难度相当大，特别是栈和堆的两道题，不过对于精神的洗礼是相当爽的（

每日选做要抓紧跟进了！今年的程设课程修改了教学内容，要把 python 纳入教学，感觉 python 太强大了！尝试了一下用 latex 敲代码和公式，感觉手工敲起来好麻烦但是最后转成的 pdf 字体好好看，如下图：

2 Neural networks

2.1 Common Activation Function

2.1.1 Sigmoid Function

$$F(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

2.1.2 Tanh Function

$$F(x) = tanh(x) \tag{6}$$

2.1.3 ReLU(Rectified Linear Unit)

$$F(x) = max(0, x) \tag{7}$$