合肥工業大學

计算机与信息学院

人工智能原理实验报告

专　业　班　级　计科 20-4 班

学生姓名及学号　杨帆-2020216657

任　课　教　师　卜晨阳

2022 ~2023 学年第 一 学期 宣城校区

说　明

实验报告是关于实验教学内容、过程及效果的记录和总结，因此，应注意以下事项和要求：

1．实验报告要求：**格式规范，语言表达清楚，数据和程序真实。并能够理论联系实际，认真分析实验中出现的问题与现象，总结经验。**

2．每位同学应独立完成实验报告的撰写，严禁抄袭或拷贝，否则，一经查实，按作弊论取，并取消理论课考试资格。

3. 可根据实际需要调整每个单元格的篇幅。

4. 请按照要求填写实验报告。算法源代码请放置在附录中。

实验序号一　　作业名称：人工智能之启发式搜索算法

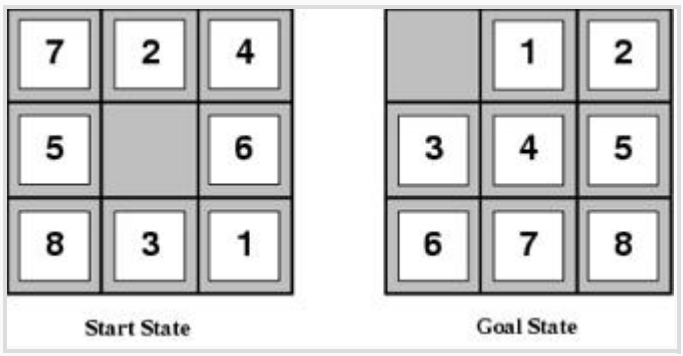
实验时间： 2022 年 10 月 30 日

实验内容

一、题目内容和要求：

实验任务

八数码问题是在一个 3×3 的棋盘上有 1-8 位数字随机分布，以及一个空格，与空格相连的棋子可以滑动到空格中，问题的解是通过空格滑动，使得棋盘转化为目标状态，如下图所示。



为了简化问题的输入，首先将空格用数字 0 表示，然后将 3×3 的棋盘用 9 位长的字符串表示，则上图的初始状态为 724506831，目标状态为 012345678，本关卡所有目标状态均为 012345678，也保证初始状态到目标状态有解。

对于上图的初始状态，将数字 2 移动到空格，称之为 *u* 操作（空格上移），将数字 3 移动到空格，称之为 *d* 操作（空格下移），将数字 5 移动到空格，称之为 *l* 操作（空格左移），将数字 6 移动到空格，称之为 *r* 操作（空格右移），则一个合法移动路径为 *lurdrdllurrdllurrulldrrull*。

```
724     724     024     204     254     ...     012
506     056     756     756     706     ...     345
831     831     831     831     831     ...     678
    l       u       r       d       ...     l
```

编程要求

本关的编程任务是补全右侧代码片段 salvePuzzle 、 calcDistH 和 moveMap 中 Begin 至 End 中间的代码，具体要求如下：

在 salvePuzzle 中，根据输入参数 *init*（初始状态，如 724506831）和 *targ*（目标状态，均为 012345678），实现 A* 搜索算法，返回八数码问题的移动路径，如上图的移动路径：*lurdrdllurrdllurrulldrrull*。

在 calcDistH 中，计算当前状态（参数 *srcmap*，如 724506831）到目标状态（参数 *destmap*，如 012345678）的启发式函数值 $h(n)$，并返回 $h(n)$。

在 moveMap 中，实现行动转换，并返回下一个状态，例如当前状态为参数 *curmap*=724506831，当前 8 数码状态 *curmap* 中空格 0 的位置索引 *i*=4，移动空格到位置 *j*=3，则返回的新状态为 *newmap*=724056831。

测试说明

平台将自动编译补全后的代码，并生成若干组测试数据，接着测试程序会调用上述函数，并判断函数返回的路径是否为合法解，若是则输出 Accepted 表示程序正确，否则程序错误。

以下是平台的测试样例：

测试输入： 724506831 预期输出： Accepted

二、问题背景和相关工作介绍

1.相关知识

为了完成本关任务，你需要掌握：1.评估函数，2.贪婪最佳优先搜索，3.A*搜索：缩小总评估代价，4.求解思路。

评估函数

在有信息搜索 Informed Search 策略中，常使用的是最佳优先搜索 Best First Search ，它的结点扩展是基于评估函数值 $f(n)$选择的。评估函数被看做是代价估计，因此代价最低的结点最先被选择扩展。

对 $f(n)$的选择决定了搜索策略，大部分的最佳优先搜索算法的 $f(n)$由启发式函数 $h(n)$构成：

$h(n)$=结点 n 到目标的最小代价路径的代价估计值

贪婪最佳优先搜索

贪婪最佳优先搜索 Greedy Best-First Search 试图扩展距离目标结点最近的结点，原因是这种策略可能可以非常快的找到解，因此，贪婪最佳优先搜索只使用启发式信息，即 $f(n)=h(n)$。

A*搜索：缩小总评估代价

A* 搜索（A 星搜索）是最广为人知的最佳优先搜索，它对结点 n 的代价评估结合了 $g(n)$，即到达此结点 n 已经花费的路径代价，和 $h(n)$，即从该结点 n 到目标结点所花代价。

$f(n)=g(n)+h(n)$

由于 $g(n)$ 是从开始结点到结点 n 的路径代价，而 $h(n)$ 是从结点 n 到目标结点的最小路径代价的估计值因此：

$f(n)$=经过结点 n 的最小代价解的估计代价

所以，要寻找最小代价的解，首先扩展的是 $g(n)+h(n)$ 值最小的结点。可以发现，A* 搜索算法与一致代价搜索算法类似，区别是 A* 搜索算法使用 $g(n)+h(n)$ 而不是 $g(n)$。

2. 已有求解算法 A 算法

对于求解过程中，启发式搜索与盲目搜索的区别就在于对于 OPEN 表的重排，优先选择最优希望的节点扩展。

重排的依据就是：从起始节点到达该节点的耗散 g(n)和从该节点到目标节点的消耗 h(n)结合起来对节点进行评价，即

f(n)=g(n)+h(n)

三、解题思路

该问题是将与空格相连的数字移动到空格的位置上，也就相当于将空格移动到与之相连的位置，因此，以空格为当前结点，扩展结点可能为上下左右四个相连的位置，若使用一般的搜索算法，可能陷入无限搜索中，永远搜不到目标解，而 A* 搜索算法则能非常好的将搜索过程导向求解目标。

问题给的是字符串数据 724506831，可以还原成如下形式：

```
对应 8 数码状态 8 数码状态下标 ID
字符串：724506831 7 2 4 0 1 2
下标 ID：012345678 5 0 6 3 4 5
8 3 1 6 7 8
```

那么空格的 *l* 移动操作即为下标 4 和下标 3 上所对应的数字的交换，分别为 0 和 5，交换后的新的状态为：

```
对应 8 数码状态 8 数码状态下标 ID
字符串：724056831 7 2 4 0 1 2
下标 ID：012345678 0 5 6 3 4 5
8 3 1 6 7 8
```

以此类推，空格的 *lrud* 各操作均可用以上的交换过程表达。

A* 算法的重中之重就是启发式函数 $h(n)$的设计，不同的设计方法可能产生不同的求解路径。在这里，可以选择欧氏距离作为评估函数值：除 0 之外，各个数字在当前状态的下标与目标状态的下标的绝对值之和。例如：当前状态为 123456780，目标状态为：012345678，数字 1 的下标分别为 0 和 1，数字 2 的下标分别为 1 和 2，...，数字 8 的下标分别为 7 和 8，则当前状态与目标状态的评估值为 $h(n)=abs(1-2)+abs(2-3)+\cdots+abs(7-8)=8$。

四、算法伪代码

```
input: open_list, close_list, g
/* 输入：开放列表、关闭列表、开放列表中每个元素对应的 g(x)值 */
output: open_list, close_list, g
/* 输出：上述输入的更新后的结果 */

def expandNode(open_list, close_list, g)
    /* 找到估价函数最小的节点 */
    for each_node in openlist
        f(each_node) = h(each_node) + g(each_node)
    node = 有最小 f(x)值的节点
    /* 拓展该节点 */
    expand(node)
    /* 将该节点移动到关闭列表 */
    close_list.append(node)
```

open_list.delete(node)

五、实验设置

1、画出 A*算法求解 N 数码问题的流程图。

根据 A*算法的实验原理，f=g(n) +h(n) ；这样估价函数 f(n)在 g(n)一定的情况下，会或多或少的受距离估计值 h(n)的制约，节点距目标点近，h 值小，f 值相对就小，能保证最短路的搜索向终点的方向进行, 因此 f 是根据需要找到一条最小代价路径的观点来估算节点的。设计 A*算法的流程图如图 1.1 所示，按照流程图编写伪代码，进而编写完整的程序。其中 OPEN 表保存所有已生成而未考察的节点，CLOSED 表中记录已访问过的节点。在扩展结点时，还需要考虑两个表即 OPEN 表和 CLOSED 表中是否存在了该节点的后继节点。具体编程思路参照算法:

将起始点加入 open 表

当 open 表不为空时：

寻找 open 表中 f 值最小的点 current

它是终止点，则找到结果，程序结束。

否则，Open 表移出 current，对 current 表中的每一个临近点

若它不可走或在 close 表中，略过

若它不在 open 表中，加入。

若它在 open 表中，计算 g 值，若 g 值更小，替换其父节点为 current，更新它的 g 值。

若 open 表为空，则路径不存在。

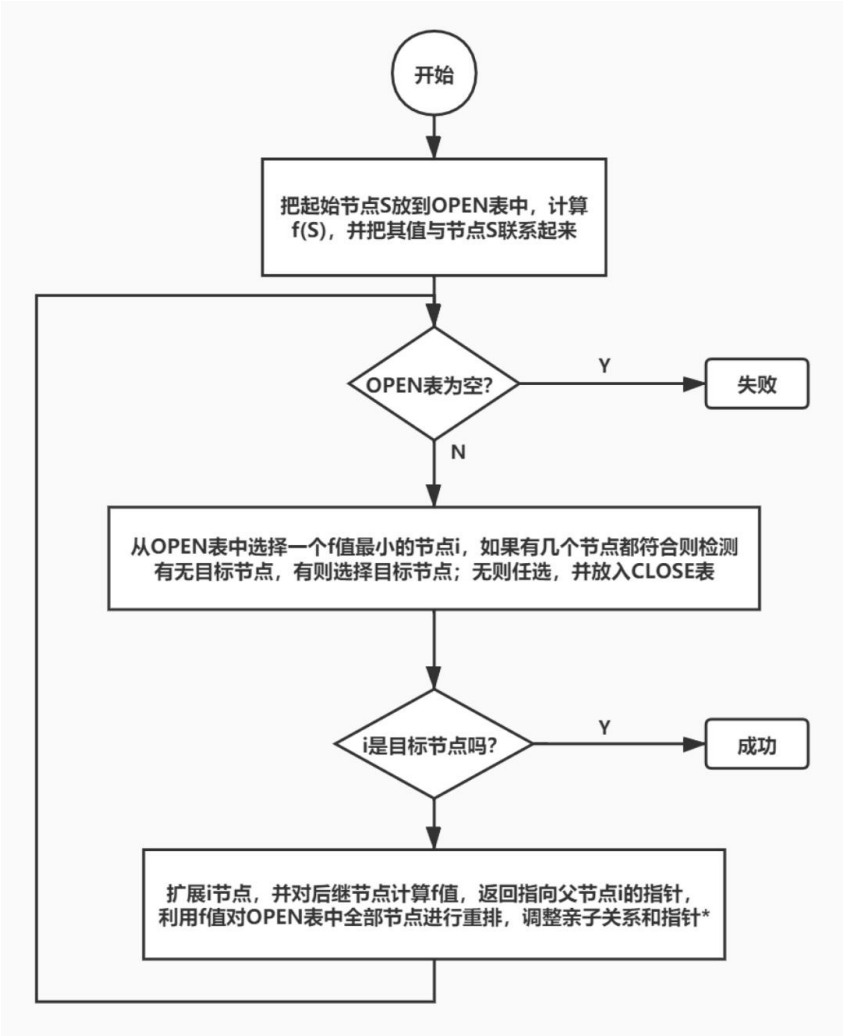


图 1.1 A*算法求解八数码问题的流程图

2、分析不同的估价函数对 A*算法性能的影响。

对于同一问题启发函数 h(n)可以有多种设计方法。在本次时实验中，通过选用“将牌不在位数”和“将牌‘不在位’的距离和”两种不同的启发函数，同时还编写了不考虑 h 值进行搜索，即不采用启发性搜索的算法（按照广度优先搜索的策略）。我们通过将第一种和第二种启发函数对比，发现第二种启发函数优于第一种启发函数，将采用启发函数与不采用启发性函数对比发现，采用启发性函数远远优于不采用启发性函数。

3、根据宽度优先搜索算法和 A*算法求解八数码问题的结果，分析启发式搜索的特点。

我们可以发现采用 A*算法求解八数码问题时间以及搜索的节点数目远远小于采用

宽度优先搜索算法，这说明对于八数码问题，选用的启发性信息有利于搜索效率的提高。但是理论上来讲，如果选用的启发性信息过强，则可能找不到最优解。

六、实验结果

	启发函数$h(n)$		
	不在位数	将牌“不在位”的距离和	0
初始状态	283604175	283604175	283604175
目标状态	123804765	123804765	123804765
最优解	最佳路径长度为：8 最佳路径为： 第1层的状态： 2 8 3 0 6 4 1 7 5 第2层的状态： 2 8 3 1 6 4 0 7 5 第3层的状态： 2 8 3 1 6 4 7 0 5 第4层的状态： 2 8 3 1 0 4	最佳路径长度为：8 最佳路径为： 第1层的状态： 2 8 3 0 6 4 1 7 5 第2层的状态： 2 8 3 1 6 4 0 7 5 第3层的状态： 2 8 3 1 6 4 7 0 5 第4层的状态： 2 8 3 1 0 4	最佳路径长度为：8 最佳路径为： 第1层的状态： 2 8 3 0 6 4 1 7 5 第2层的状态： 2 8 3 1 6 4 0 7 5 第3层的状态： 2 8 3 1 6 4 7 0 5

	7 6 5 第5层的状态: 2 0 3 1 8 4 7 6 5 第6层的状态: 0 2 3 1 8 4 7 6 5 第7层的状态: 1 2 3 0 8 4 7 6 5 第8层的状态: 1 2 3 8 0 4 7 6 5	7 6 5 第5层的状态: 2 0 3 1 8 4 7 6 5 第6层的状态: 0 2 3 1 8 4 7 6 5 第7层的状态: 1 2 3 0 8 4 7 6 5 第8层的状态: 1 2 3 8 0 4 7 6 5	第4层的状态: 2 8 3 1 0 4 7 6 5 第5层的状态: 2 0 3 1 8 4 7 6 5 第6层的状态: 0 2 3 1 8 4 7 6 5 第7层的状态: 1 2 3 0 8 4 7 6 5 第8层的状态: 1 2 3 8 0 4 7 6 5
扩展节点数 (不包括叶子节点)	17	8	294
生成节点数 (包含叶子节点)	33	17	470
运行时间 (迭代次数)	220.00ms	120.00ms	1469.00ms

七、实验结果分析

实验成功完成。

八、总结

启发式搜索特点和实际工程应用

1.对于启发性信息：设对同一个问题定义了两个 A* 算法 A1 和 A2，若 A2 比 A1 有较多的启发信息，即对所有非目标节点有 h2(n) > h1(n)，则在具有一条从 s 到 t 的路径的隐含图上，搜索结束时，由 A2 所扩展的每一个节点，也必定由 A1 所扩展，即 A1 扩展的节点数至少和 A2 一样多。

2.在实际应用中理论上是设计接近、又总是≤h*(n)的 h(n)，但是这种设计往往也要消耗大量的时间，如果 h(n)本身设计的复杂，每次计算时消耗在 h(n)上的复制度也就会增加，使得求解变慢。

3.对于搜索深度和启发性信息的权衡：

f(n) = g(n) + wh(n)

搜索图的浅层时，让 w 取较大值，突出启发式函数的作用，加速向纵深方向搜索。搜索到较深的层次时，让 w 取较小值，以使 g(n)所占权重大，并确保 wh(n)≤h* (n)，引导搜索向横广方向发展，寻找到较短的解答路径。

● 感想、体会、建议：

当时面对 300 行的代码时，不知从何处下手，通过查阅资料和请教老师，以及与同学深入探讨，仔细研究 A*算法之后，才明白程序如何编写，各部分的函数如何构成。同时，通过本次实验，发现选用不同的启发函数，对于实验的结果有较大的影响。选用第一或第二种（也就是采用 A*算法）远远优于普通的广度优先搜索，同时，明显的感觉到第二种启发函数效率更高，更快的找到最优解。但是，在实验过程中，也遇到了一些问题，比如初始值的八数码初始值的选择对于实验结果的影响很大，在选取一些样例时，比如 1,3,0,2,8,4,7,6,5，实验结果达到 20000 次依然没有停止，无法比较两种启发函数的优越性，鉴于时间原因，选取一些迭代次数较小就可以达到目标状态的样例进行验证，发现第二种结果优于第一种启发函数的结果。总的来说，实践出真知，只有把书上的理论知识运用到实践，才是真正地掌握。本次实验顺利完成，还是挺开心的。

上机报告成绩、评语： 指导教师签名： 年　　月　　日

实验序号二　作业名称：　群智能算法

实验时间： 2022 年 10 月 31 日

实验内容

一、题目内容和要求：

1.任务描述：遗传算法 - 函数最优解计算

本关任务：使用 python 实现遗传算法，并求目标函数最优解。

2.任务描述：粒子群算法 - 目标函数最优解计算

本关任务：使用 python 实现粒子群算法，并求解目标函数最优解。

3.任务描述：蚁群算法 - 商队旅行最短路径计算

本关任务：使用 python 实现蚁群算法，并寻找商队旅行最短路径。

4.任务描述：动手实现旅行商问题

本关任务：使用 python 实现遗传算法解决 TSP 问题。

二、问题背景和相关工作介绍

1.遗传算法 - 函数最优解计算

为了完成本关任务，你需要掌握：1.遗传算法，2.使用 python 实现遗传算法。

遗传算法

基因和染色体

在遗传算法中，我们首先需要将要解决的问题映射成一个数学问题，也就是所谓的**数学建模**，那么这个问题的一个可行解即被称为一条**染色体或个体**。如：

$3x+4y+5z<100$

[1,2,3]，[2,3,4]，[3,2,1]均为这个函数的可行解，这些可行解在遗传算法中均被称为染色体，每一个元素就被称为染色体上的一个基因。

染色体编码与解码

遗传算法的运算对象是表示染色体的符号串，所以必须把变量 x，y，z 编码为一种符号串。常见的编码方式如用无符号**二进制整数**来表示。解码即将二进制整数转换回最初的表现型。

编码： 5 --> 0101。 解码： 0101 --> 5。

初始群体的产生

遗传算法是对群体进行的进化操作，需要给其准备一些表示起始搜索点的初始群体数据。假如群体规模的大小取为 4，即群体由 4 个染色体组成，每个染色体可通过随机方法产生。

如： 011101 ， 101011 ， 011100 ， 111001。

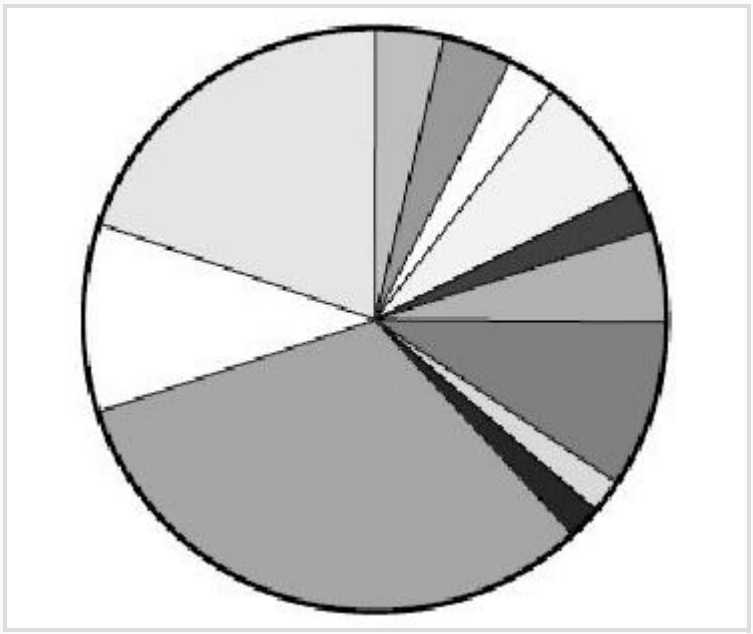
物竞天择

1. 适应度函数：遗传算法中以染色体适应度的大小来评定各个染色体的优劣程度，从而决定其遗传机会的大小。
2. 选择函数：自然界中，越适应的个体就越有可能繁殖后代。但是也不能说适应度越高的就肯定后代越多，只能是从概率上来说更多。常用的选择方法有轮盘赌选择法。

若 fi 表示每个染色体的适应度，则每个个体遗传下来的概率为：

$$p(i)=i\sum fi \quad fi$$

由公式可以看出，适应度越高，则遗传下来的概率就越大，好比赌轮盘，轮盘上所占面积越大，则被小球滚到的概率就越大。



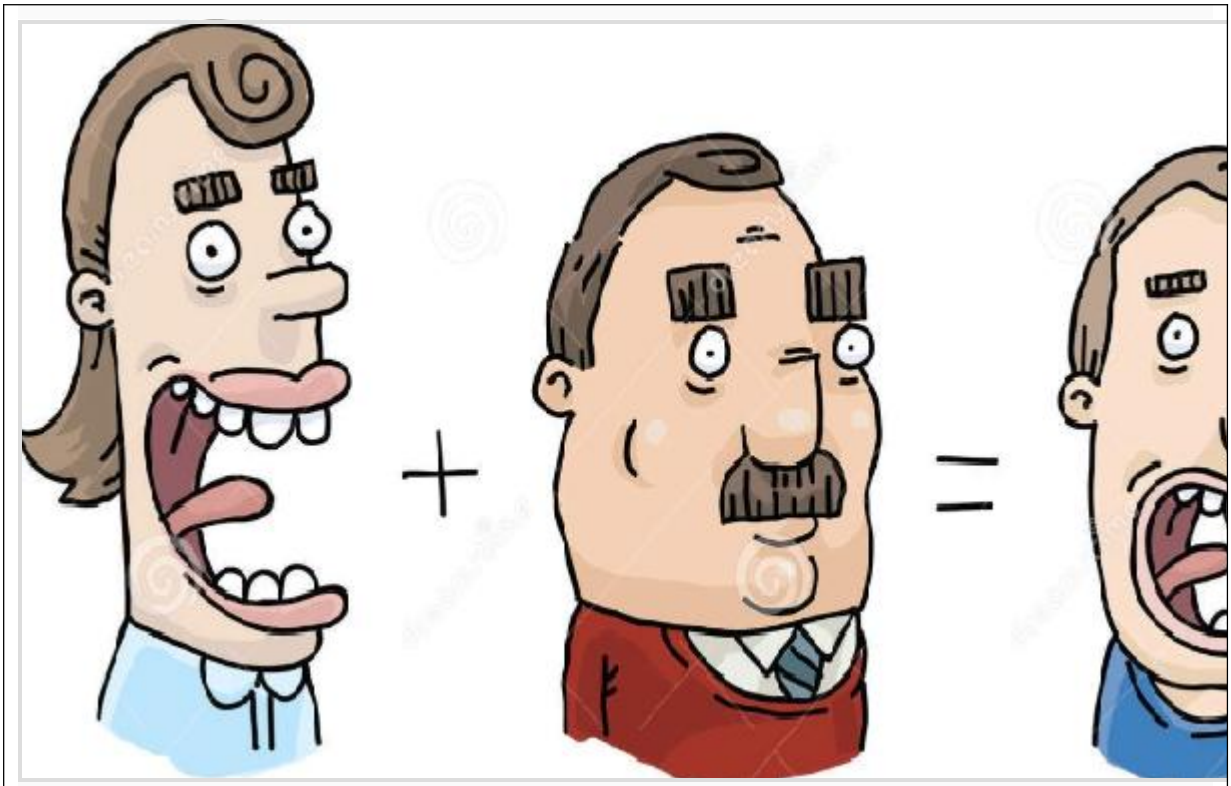
交叉与变异

交叉是遗传算法中产生新的个体的主要操作过程，以一定的概率决定个体间是否进行交叉操作。

如上图为父辈染色体进过交叉后产生新的染色体的过程。

变异为另一种产生新个体的操作，它可以为种群带来多样性：

也就是将**我**的染色体中的基因，随机的由 0 变成 1，或 1 变成 0。



2.粒子群算法 - 目标函数最优解计算

为了完成本关任务，你需要掌握：1.编码与适应度函数，2.粒子群算法原理，3.粒子群算法流程，4.使用 python 实现粒子群算法。

编码与适应度函数

在粒子群算法中也需要进行编码，不过相对于遗传算法粒子群算法编码非常简单。例如，函数：

$f(x1\ \ ,x2\ \ )=x12\ \ +x22$

可直接将函数解$(x1\ \ ,x2\ \ )$作为编码。而函数的值 $f(x1\ \ ,x2\ \ )$即可作为适应度，若求解函数最小值则适应度越小越好，若求解函数最大值则适应度越大越好。

粒子群算法原理

粒子群函数是根据鸟群寻找食物实现的优化算法，每一只鸟被称为粒子，即函数的一个解。我们已经知道，每一只鸟寻找食物是根据离食物最近的鸟的位置，与自己曾经离食物最近的位置来决定改变自己现在的位置。根据这个原理，粒子群算法核心公式如下：

$v=wv+c1\ \ r1\ \ (p-x)+c2\ \ r2\ \ (pg\ \ -x)...(1)$

$x=v+x...(2)$

其中，$x=(x1 \quad ,x2 \quad ,..,xn \quad)$为鸟群的位置，$v=(v1 \quad ,v2 \quad ,..,vn \quad)$为鸟飞行的速度，即鸟群更新位置的因素。而公式 2 就是决定速度的因素：

1. p:个体最佳位置
2. pg:全局最佳位置
3. w:惯性权重因子，用来控制速度的更新
4. c1,c2:加速度常数，通常设为 2
5. r1,r2:0 到 1 之间的随机数

3. 蚁群算法 - 商队旅行最短路径计算

为了完成本关任务，你需要掌握：1.蚁群算法原理，2.蚁群算法流程，3.使用蚁群算法解决商队旅行问题。

蚁群算法原理

大家已经知道，蚂蚁是根据信息素浓度来选择下一步行走的路线，信息素越浓，蚂蚁选择的概率越大，公式如下：

$pijk(t) \quad =s \in Jk \quad (i) \sum \quad [\tau is \quad (t)]\alpha[\eta is \quad (t)]\beta[\tau ij \quad (t)]\alpha[\eta ij \quad (t)]\beta$

$pijk(t)$ ：第 k 只蚂蚁 t 时刻，从城市 i 到 j 的概率。

$\tau ij \quad (t)$: t 时刻蚂蚁走完一周后，城市 i 到 j 的信息素浓度。

ηij :启发式因子。

$\eta ij \quad =dij \quad 1$

$\tau ij \quad (t+n)=(1-\rho)\tau ij \quad (t)+\Delta\tau ij$

$\Delta\tau ij \quad =k=1 \sum m \quad \Delta\tau ijk \quad (t+n)$

$\Delta\tau ijk \quad =Lk \quad Q$

α：信息素重要程度。 β：启发式因子重要程度。 ρ:信息素挥发速度。 Lk :蚂蚁 k 在本次周游中所走路径长度。 m:蚂蚁个数。 n:城市个数。 d:城市之间距离。

4. 动手实现旅行商问题

为了完成本关任务，你需要掌握：1.使用遗传算法解决 TSP 问题。

使用遗传算法解决 TSP 问题。

蚁群算法对于参数的敏感程度较高，参数设置的好，算法的结果也就好，参数设置的

不好则运行结果也就不好。所以上一关将通关所设置的阈值较高。本关，我们将使用**遗传算法**来解决 TSP 问题，试图寻找到更短的旅行路线。

三、解题思路

1.遗传算法流程

1. 初始化种群
2. 计算适应度
3. 选择适应度高的个体
4. 通过交叉变异选择新的染色体
5. 终止进化

使用 python 实现遗传算法。

本关任务是使用遗传算法求解目标函数最大值，首先需要随机产生很多个解，即初始化种群，代码如下：

```
#初始化种群
pop = np.random.randint(2, size=(POP_SIZE, DNA_SIZE)) # Size 函数需要传入两个参数 POP_SIZE 为解的个数，即染色体个数。DNA_SIZE 为染色体长度
```

其中 POP_SIZE 为解的个数，即染色体个数。DNA_SIZE 为染色体长度。

由于染色体为二进制编码，所以还需要将二进制转换为浮点数的解码方法:

```
#解码
def translateDNA(pop):
    return pop.dot(2 ** np.arange(DNA_SIZE)[::-1]) / float(2**DNA_SIZE-1) * X_BOUND[1]
```

上述代码中： pop.dot(2 ** np.arange(DNA_SIZE)[::-1])已经转换成十进制但是需要归一化到 0~5 ，如有 1111 这么长的 DNA,要产生的十进制数范围是 [0, 15] ，而所需范围是 [-1, 1] ，就将 [0,15] 缩放到 [-1,1] 这个范围。

a[::-1]相当于 a[-1:-len(a)-1:-1]，也就是从最后一个元素到第一个元素复制一遍。所以你看到一个倒序，np.arange(DNA_SIZE)[::-1]得到 10,9,8,...,0 。

如将 10101 转换到 0 到 5 之间：

x=25−124+22+2+0 ×5

然后计算每个染色体的适应度，由于是求解最大值，函数值越大则越应该被选择，所以，将每个染色体对应的函数值减去最小值作为适应度：

```
#获取染色体适应度
def get_fitness(pred):
    return pred + 1e-3 - np.min(pred)
```

再选择适应度高的染色体

再对染色体进行交叉变异操作

总流程如下：

```
def F(x): return np.sin(10*x)*x + np.cos(2*x)*x

def ga(F):
    '''
    F:需要求解的函数
    '''
    #初始化种群
    pop = np.random.randint(2, size=(POP_SIZE, DNA_SIZE))
    #开始进化
    for _ in range(N_GENERATIONS):
        #计算函数值
        F_values = F(translateDNA(pop))
        #计算适应度
        fitness = get_fitness(F_values)
        #选择适应度高的个体
        pop = select(pop, fitness)
        pop_copy = pop.copy()
        #通过交叉变异选择新的染色体
        for parent in pop:
            #交叉产生子代
            child = crossover(parent, pop_copy)
            #变异产生子代
            child = mutate(child)
            #子代代替父代
            parent[:] = child
    #获取最优解
    x = translateDNA(pop)[-1]
    return x
```

其中：

1. N_GENERATIONS 为进化轮数。
2. CROSS_RATE 为交叉概率。
3. MUTATION_RATE 为变异概率。
4. X_BOUND 为函数定义域。

2. 粒子群算法流程

1. 随机初始粒子群位置与速度
2. 计算粒子群适应度
3. 根据公式更新粒子群位置与速度
4. 重复 2,3 直到满足停止条件

使用 python 实现粒子群算法

首先我们需要对粒子群位置与速度进行随机初始：

```
import numpy as np
#初始化粒子群位置
x = np.random.uniform(x_bound[0], x_bound[1],(pop_size, dim))
#初始化粒子群速度
v = np.random.rand(pop_size,dim)
```

其中，x_bound 为 x 取值范围。pop_size 为粒子群大小，即鸟的数量。dim 为搜索空间维度。

再根据 x 计算适应度：

```
#f(x1,x2)=(x1-4)**2+(x2-5)**2,函数值即适应度
def f(x):
    return np.sum(np.square(x-np.array([4,5])), axis=1)
#计算适应度
fitness = f(x)
```

同时计算出全局最优位置与个体最优适应度、全局最优适应度：

```
#全局最优位置
pg = x[np.argmin(fitness)]
#个体最优适应度
individual_best_fitness = fitness
#全局最优适应度
```

```
global_best_fitness = np.min(individual_best_fitness)
```

最后开始进化，不断更新粒子群位置。

3. 蚁群算法流程

1. 初始化各个参数
2. 随机放置蚂蚁到各个城市
3. 对每一只蚂蚁选择下一个城市
4. 走遍所有城市时，计算当前最佳路径并更新信息素浓度
5. 重复 2,3,4 直到达到最大迭代次数
6. 输出最短路径

使用蚁群算法解决商队旅行问题

旅行商（TSP）问题：

1. 有一队商客，他们想要去 n 个城市进行旅游。
2. 每个城市只能去一次。
3. 从某城市出发，最终得回到这个城市。
4. 选择出最短的路线。

按照算法流程，首先得初始化参数：

然后再选择每个蚂蚁的出发城市：

```
#随机产生各个蚂蚁的起点城市
pathtable[:numcity, 0] = np.random.permutation(range(0, numcity))[:]
pathtable[numcity:, 0] = np.random.permutation(range(0, numcity))[:numant-numcity]
```

再选择下一步蚂蚁要行走的城市：

同时更新信息素

最后不断重复上述步骤，选择出最短路径。

4. 群智能算法

同求解目标函数最值问题类似，我们首先得初始化种群：

```
#初始化种群
pop = np.vstack([np.random.permutation(DNA_size) for _ in range(POP_size)])
```

其中，DNA_size 为染色体长度，POP_size 为种群大小，即染色体个数。每一个染色

体为一种旅行的路线，如从城市 1 -->城市 2 -->城市 3 编码为： [1,2,3] 。

然后需要对旅行路线进行解码：

DNA 为初始的种群，即所有的初始的旅行路线，city_position 为所有城市的坐标位置。最后返回所有城市的横坐标与纵坐标，即 line_x 与 line_y。

再通过横坐标与纵坐标计算旅行路线总距离与其对应的适应度

适应度计算公式如下：

$fitness=etotaldistance \quad DNAsize2$

由公式可以看出，距离越大，适应度越低，且通过指数函数放大每个路线的适应度的差距。使最佳的路线有更大的概率被选择。

再使用轮盘赌算法保留适应度高的染色体：

```
#选择适应度高的染色体
def select(pop,fitness):
    idx = np.random.choice(np.arange(POP_size), size=POP_size, replace=True, p=fitness / fitness.sum())
    return pop[idx]
```

同样的，对保留下来的染色体，进行交叉变异操作，达到进化的目的，产生子代：

其中：

1. CROSS_RATE 为交叉概率
2. MUTATE_RATE 为变异概率
3. N_GENERATIONS 为进化次数

四、算法伪代码

1. 任务描述：遗传算法 - 函数最优解计算

```
/*  P(t)表示某一代的群体，t 为当前进化代数
Best 表示目前已找到的最优解  */
Procedure GA
begin
  t←0;
  initialize(P(t));
  evaluate(P(t));
  keep_best(P(t));
  while(不满足终止条件) do
  begin
    P(t)←selection(P(t));
```

```
    P(t)←crossover(P(t));
    P(t)←mutation(P(t));
    t←t+1;
    P(t)←P(t-1);
    evaluate(P(t));
    if (P(t)的最优适应值大于 Best 的适应值)
        //以 P(t)的最优染色体替代 Best
        replace(Best);
        end if
    end
 end
```

2. 任务描述：粒子群算法 - 目标函数最优解计算

```
procedure PSO
   for each particle i
    Initialize velocity Vi and position Xi for particle i
     Evaluate particle i and set pBesti= Xi
   end for
   gBest = min {pBesti}
   while not stop
    for i=l to N
      Update the velocity and position of particle i
        Evaluate particle i
    if fit (Xi) < fit (pBesti)
        pBesti= Xi;
   if fit(pBesti) < fit (gBest)
        gBest = pBesti;
   end for
   end while
   print gBest
end procedure
```

3. 任务描述：蚁群算法 - 商队旅行最短路径计算

```
for 每一只蚂蚁 k∈[1.N]
   为每只蚂蚁 k∈[1.N]初始化开始城市 Ck1
   为蚂蚁 k∈[1.N]初始化已访问城市集合: Ck← -{Ck1}
   for  每一个城市 j∈[1,n]且 j∈Ck
      计算概率
Next j
   蚂蚁 k 以概率去到城市 j
   使用 ck q+1 表示上一行中选择的城市
   Ck←-Ck∪{ck q+1} .
next  蚂蚁
```

next q

4. 任务描述：动手实现旅行商问题

private void genetic(); //一次繁殖操作

private void hybridization(); //一次杂交操作

private Entity getEntity(); //从父代中选取一个个体（轮盘赌策略 - 选择操作）

private void swapAndHandleConflict(); //交换基因并解决冲突（交叉操作）

private void variable(); //变异子代（变异操作）

private void updateEntity(); //精英保留，更新父代信息

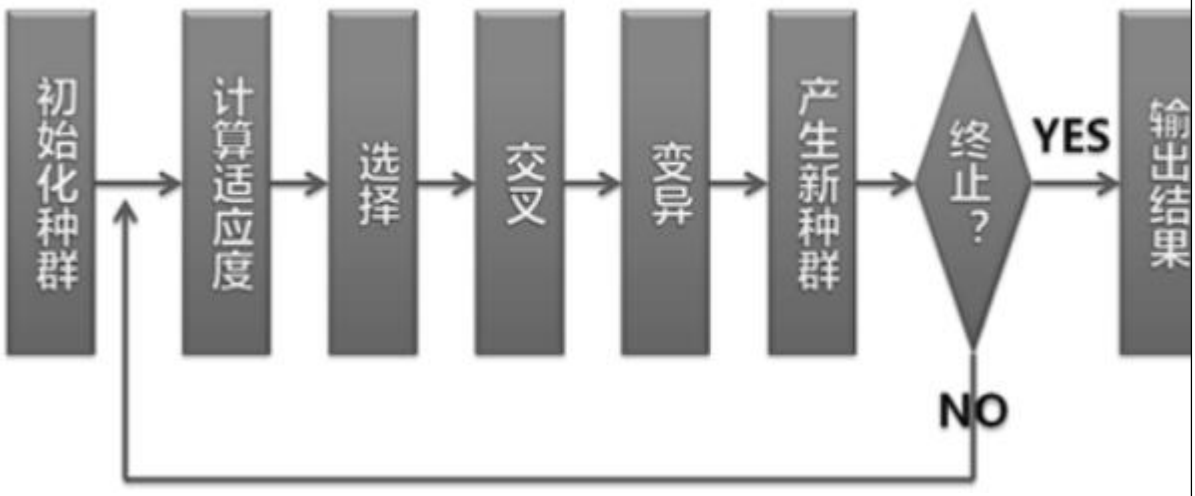
this.setReproducePercent(); //对新一代父代重新计算其个体参与轮盘赌策略的概率

private void choose(); //选取记录最优个体

五、实验设置

要求：请给出实验的设置，包括实验工具、实验配置、算法参数设置、比较的算法等

1. 任务描述：遗传算法 - 函数最优解计算



1. 确定编码方案——二进制编码，长度 33 位。

2. 初始化种群：使用计算机在 0～1 之间产生随机数 K，并按照数 K 的 值初始化基因位：0≤K＜0.5，基因为置为 1，0.5≤K≤1，基因为置为 0。得到 33 位的 01 序列字符串。

3. 计算个体适应度，以 maxf（x1,x2)为适应度函数，因所需求的值为最大值，适应度函数值越大表示适应度越高，个体越优良。

4. 轮盘赌算法

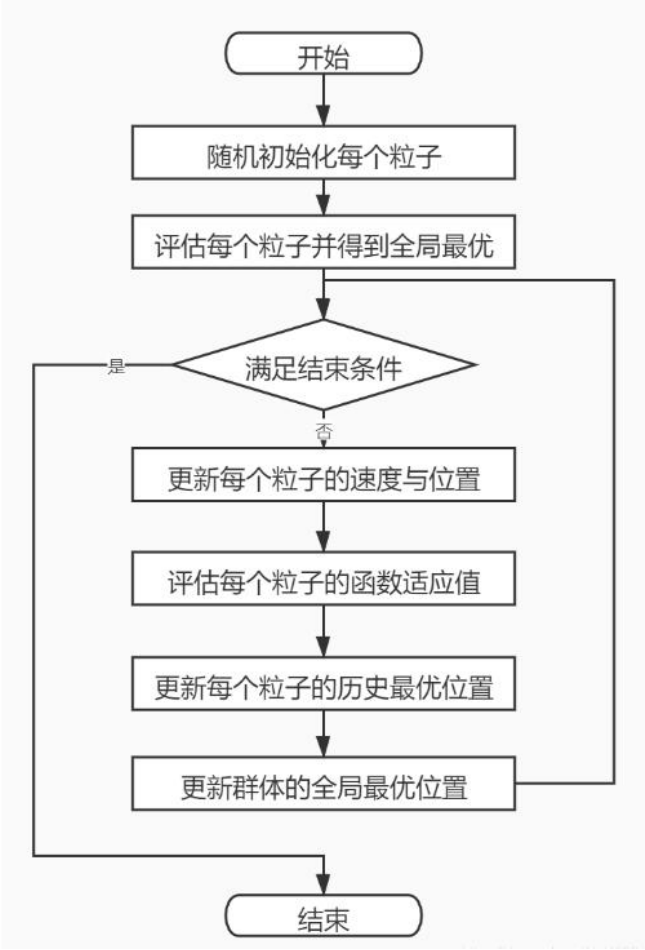
1)计算适应度比例，即每个个体的选择概率。

(2)计算每个个体的累积概率（适应度越大累积概率越高），相当于转盘上的“跨度”，“跨度”越大越容易选到。即每个个体之前所有个体的选择概率之和，相当于概率论中的概率分布函数

(3)生成 0-1 随机数，挑选在跨度之间的个体（累积概率越高被选中的概率越大）。
5.根据交叉概率 pc=0.6 对选出的个体进行交 叉，将两点交叉作为交叉算子，记产生的后代个体的集合为 01。
5.设定变异概率 Pm=0.01（变异概率 0.01 表示 01 中平均有 1%个基因发生变异）对交叉产生的个体 进行变异，将单点变异作为变异算子，记产生 的后代个体的集合为 02。
6.计算所有后代个体的适应度，选出最好的 E=2 个个体直接保留到下一代种群。然后，根据轮盘赌选择 N-E=8 个个体进入下一 代种群。
7.若进化代数 t=1000，则终止算法，输出适应度最大的个体作为问题的最优解。否则，令 t=t+1，转入第三步。

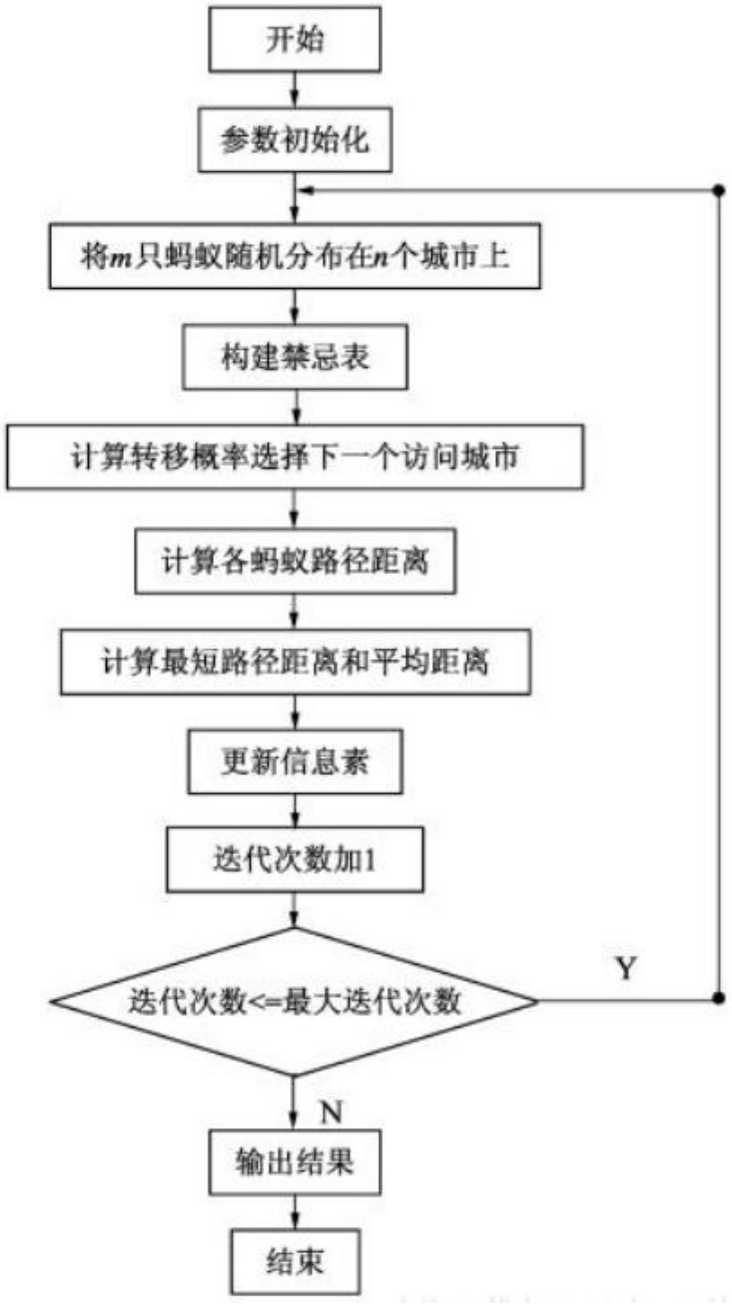
遗传算法根据对个体的优良选培及适当的交叉变异，逐步逼近最优解。
显然进化代数越高，解的精度和可信度越高，但是计算时间和算力要求更高。

2.任务描述：粒子群算法 - 目标函数最优解计算

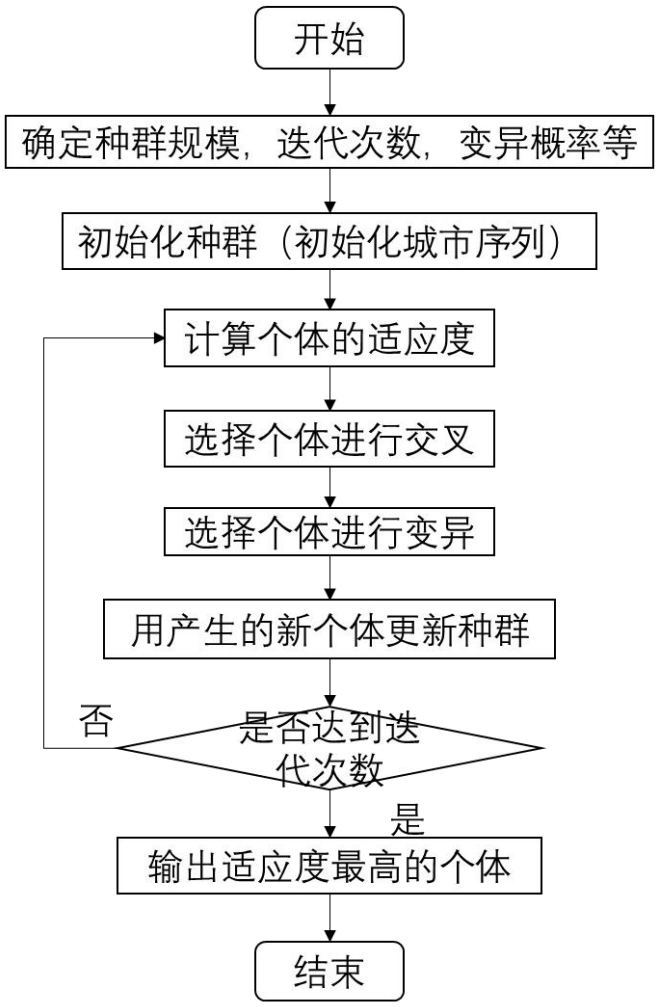


3.任务描述：蚁群算法 - 商队旅行最短路径计算
1）设整个蚂蚁群体中蚂蚁的数量为 m，城市的数量为 n，城市 i 与城市 j 之间的距离为 dij（i,j=1,2,3,…,n），t 时刻城市 i 与城市 j 链接路径上的信息素浓度为 xij。初始时刻，各个城市间连接路径上的信息素浓度相同且为 0.
2）蚂蚁 k（k=1,2,3,…,n）根据各个城市间连接路径上的信息素浓度决定其下一个访问

城市，设 Pkij(t)表示 t 时刻蚂蚁 k 从城市 i 转移到城市 j 的概率。
其中 nij(t)为启发函数，nij(t)=1/dij，表示蚂蚁从城市 i 转移到城市 j 的期望程度；allowk（k=1,2,…m）为蚂蚁 k 待访问城市的集合，开始时，allowk 中有(n-1)哥元素，即包括除了蚂蚁 k 出发城市的其他所有城市，随着时间的推进，allowk 中的元素不断减少，直至为空，即表示所有的城市均访问完毕；α为信息素重要程度因子，其值越大，表示信息素的浓度在转移中起的作用越大；β为启发函数重要程度因子，其值越大，表示启发函数在转移中起的作用越大，即蚂蚁会以较大的概率转移到距离短的城市。
3）在蚂蚁释放信息素的同时，各个城市间连接路径上的信息素逐渐消失，设参数 ρ(o<ρ<1)表示信息素的挥发程度。因此，当所有蚂蚁完成一次循环后，各个城市间连接路径上的信息素浓度需要进行实时更新。

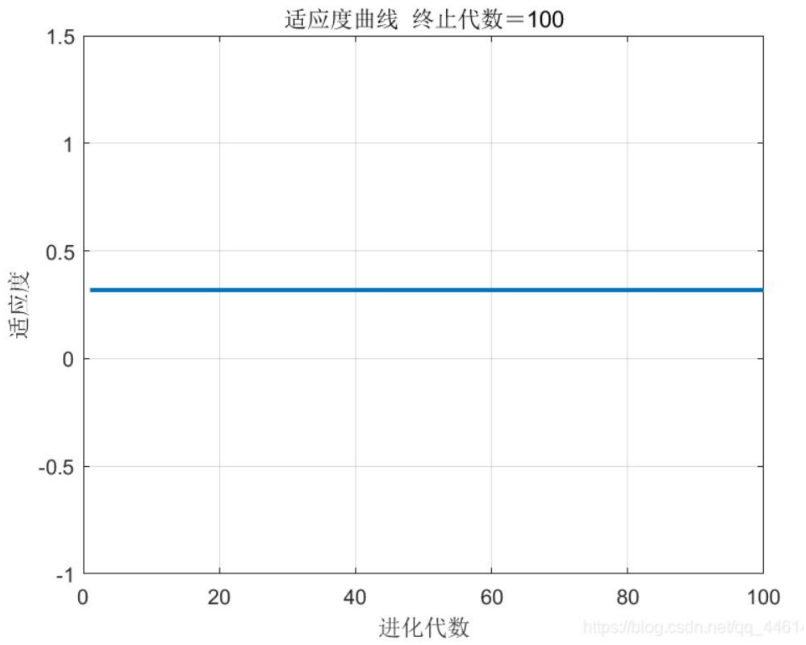


4. 任务描述：动手实现旅行商问题



六、实验结果

1. 任务描述：遗传算法 - 函数最优解计算



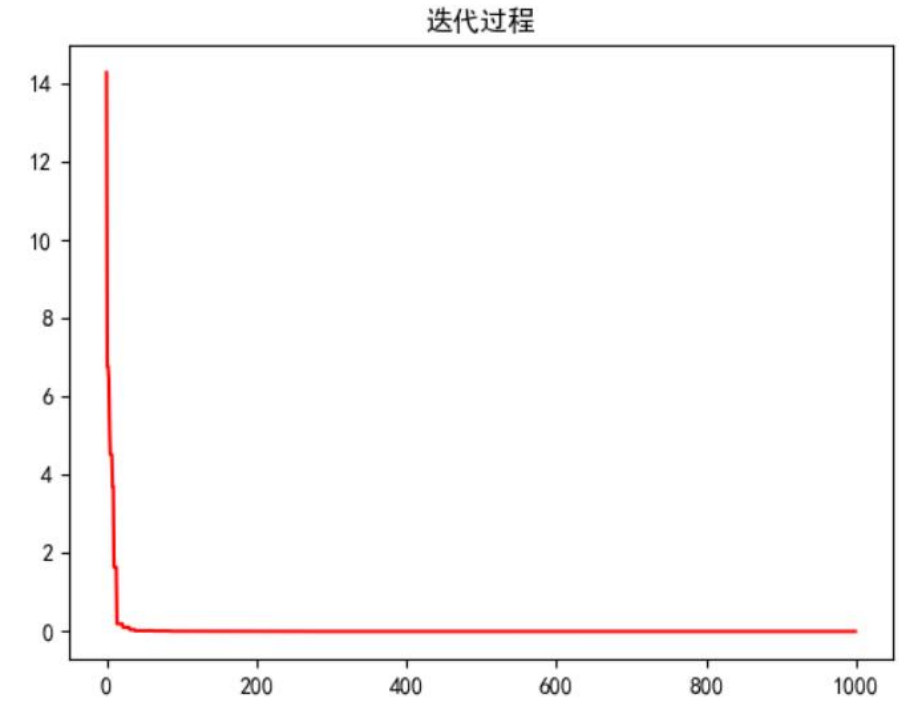

```
zbest =

   28.2689    0.2750   15.8455

最值为
0.3187
实际最值为 0
```

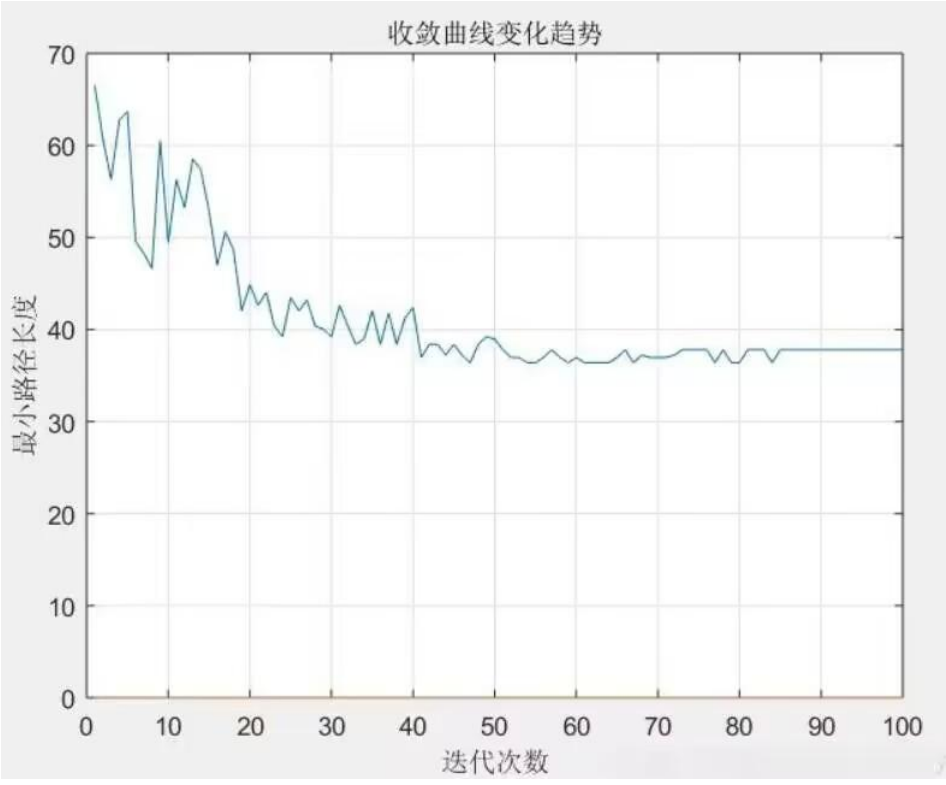
2. 任务描述：粒子群算法 - 目标函数最优解计算

总体来说，粒子群算法是一种较大概率收敛于全局最优解的，适合在动态、多目标优化环境中寻优的一种高效率的群体智能算法。



3.任务描述：蚁群算法 - 商队旅行最短路径计算

蚁群算法是一种很好解决问题的方法。比较容易理解，并且操作简单，适应于大多数场景。而缺点就是启发式算法，很难有得到保证参数。蚁群算法可以作为很多启发式寻路算法的改进版本，核心是设定合理的信息素更新方式，当然蚁群算法涉及多次迭代和多只蚂蚁，计算的成本自然也会上升。



4. 任务描述：动手实现旅行商问题

旅行商问题属于 NP 难问题，无法在线性的复杂度中求解。利用遗传算法这种启发式算法可以在相对短的时间内找到一个相对优化的解。而且在实验中我们发现，一些超参数的设置，比较变异率，种群大小，迭代次数对于最终的结果都是至关重要的。另外对于一些策略，比如交叉策略，精英保留策略的运用对于产生一个好的解显得尤为重要。

七、实验结果分析

实验均成功完成。

1. 其实在遗传算法中，他就是类似与进化论一样，优胜劣汰，根据你给定的条件，他就会朝着给定的方向走。因为我们给定的是最大值和最小值的条件，那么他就会往最大值和最小值的方向走，但是每一次都是随机的，所以就会取每一代中最大的和最小的再进行下一轮的迭代，因为我们给定的是 20 次迭代，如果说给的迭代次数越多，那么他的效果就会更加接近真值，就会离正真的最值更近。但是在这个问题中，我们也不用求得那么准确，大概接近就可以达到我们的要求。

2.粒子群算法是一种随机搜索算法 。粒子的下一个位置受到自身历史经验和全局历史经验的双重影响，全局历史经验时刻左右着粒子的更新，群体中一旦出现新的全局最优，则后面的粒子立马应用这个新的全局最优来更新自己，大大提高了效率，相比与一般的算法(如遗传算法的交叉)，这个更新过程具有了潜在的指导，而并非盲目的随机 。自身历史经验和全局历史经验的比例尤其重要，这能左右粒子的下一个位置的大体方向，所以，粒子群算法的改进也多种多样，尤其是针对参数和混合其他算法的改进 。通过对实验结果和已知最小值比较可知，该算法能有效解决这类问题，需要特别指出的是 xm 的取值只是近似值，另外，本文的粒子群算法程序只是实现粒子群算法的基本功能，该算法还有待进一步改进。

3.蚁群算法根据模拟蚂蚁寻找食物的最短路径行为来设计的仿生算法，因此一般而言，蚁群算法用来解决最短路径问题，并真的在旅行商问题（TSP，一个寻找最短路径的问题）上取得了比较好的成效。目前，也已渐渐应用到其他领域中去，在图着色问题、车辆调度问题、集成电路设计、通讯网络、数据聚类分析等方面都有所应用。

4.根据实验结果可以看出，当变异率从 0.001 逐渐增大的时候，城市之间的路径连线在得到不断的优化，即最短路径长度逐渐变短。当变异率逐渐增大到 0.008 以后，进一步增大到 0.1 甚至 0.2 的时候发现路径反而变差了，即最短路径长度再次出现增大的

情况，而且根据迭代曲线可以看出，在迭代的过程中出现了明显的震荡，证明了过高的变异率会导致遗传算法的不稳定性，从而使得算法陷入到了一个局部最优解中。
所以可以看出变异率对于遗传算法的最终效果显得尤为重要，过小或者过大都不是一种好的选择，根据实验结果 0.008 左右会是一个不错的参考值。

八、总结

1）遗传算法的难点在于：如何将复杂问题转化成可用遗传算法求解的问题寻找合适的编码方式；建立恰当的适应度值评价方法。这些都是在实验中亟待解决的问题。遗传算法的理论非常简单，而实际在使用中，每一个步骤和参数都是要仔细考虑的。以求解 mTSP 问题为模板，可以用遗传算法求解其它许多复杂的 VRP 问题，具体问题需要自己去探索如何解决。
如果自己动手实现过，会发现当城市数量较少时，遗传算法还可以得到较为理想的结果，当城市数量增加时，结果可能就不那么理想了，这个时候你就需要优化你的算法细节；你也可能遇到遗传算法过早收敛，即"早熟"的问题。凡事入门简单，精通却很难，想要真正熟练使用遗传算法还需要不断学习和实践。
2）本文利用粒子群算法思想，通过编写 matlab 程序，对一个三维连续函数进行优化得到了较好的仿真结果，证明粒子群算法在解决这类问题的可行性。另外，在编写本文的工作过程中，提高了自己对粒子群算法的理解和应用能力。为今后利用智能算法撰写论文或进行科学研究打下了很好地基础。

● 感想、体会、建议：

凡事入门简单，精通却很难，想要真正熟练使用实验中的各种算法还需要不断学习和实践。

上机报告成绩、评语：

指导教师签名：

年　　月　　日

实验序号三　　作业名称：　神经网络之网络基础

实验时间：　2022 年 10 月 31 日

实验内容

一、题目内容和要求：

1. 任务描述：神经元

本关任务：通过学习神经元的相关知识，编写一个线性神经元模型。

2. 任务描述：优化方法：梯度下降

本关任务：通过学习梯度下降法的相关知识，编写一个计算指定函数取得极值时对应的 x 的值的程序。

二、问题背景和相关工作介绍

1. 神经元

为了完成本关任务，你需要掌握：

1. 神经元模型；
2. pytorch 中 torch.mm(a,b)函数的使用。

神经元模型

神经元是神经网络操作的基本信息处理单位，是神经网络实现的基础。神经元模型有三个基本元素：　1.突触或连接链集。每一个都由其权值或者强度作为特征。具体来说，在连到神经元 *k* 的突触 *j* 上的输入信号 *xj*　乘以 *k* 的突触权值 *wkj*　；　2.加法器。用于求输入信号被神经元的相应突触加权的和；　3.激活函数。用来限制神经元输入振幅。　其基本结构如下图 1：

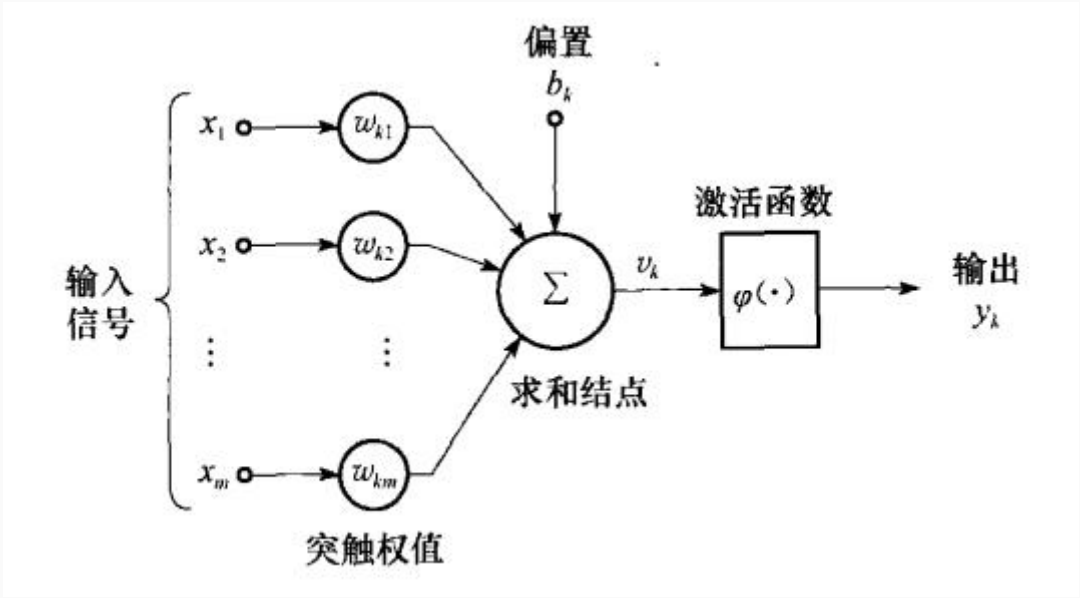


图 1 神经元模型

图中的神经元模型还包括一个外部偏置（bias），记为 *bk* 。偏置 *bk* 的作用是根据其为正或是为负，相应地增加或降低激活函数的网络输入。其数学公式如下：

$$uk = j=1\sum m \; wkj \; xj \quad yk = \varphi(uk + bk)$$

其中 *x*1 ,*x*2 ,...,*xm* 是输入信号，*wk*1 ,*wk*2 ,...,*wkm* 为神经元 *k* 的突触权值，*uk* 是输入信号的线性组合器的输出，*bk* 为偏置，激活函数为*φ*(·)，*yk* 神经元输出信号。

pytorch 中 torch.mm(a,b) 函数的使用

pytorch 深度学习框架为我们提供了一个进行矩阵乘法的方法——torch.mm(a,b)。举个例子，假设矩阵 a 的维度为 (3,4)，矩阵 b 的维度为(4,2)，则 torch.mm(a,b) 返回的矩阵的维度则为(3,2)。

2. 优化方法：梯度下降

为了完成本关任务，你需要掌握：

1. 优化算法概述；
2. 梯度下降法；
3. 梯度消失。

优化算法概述

优化算法的作用是通过不断改进模型中的参数使得模型的损失最小或准确度更高。在神经网络中，训练的模型参数主要是内部参数，包括权值（W）和偏置（B） 。 模型的内部参数在有效训练模型和产生准确结果方面起着非常重要的作用。常见的优化算法分为两类。 1.一阶优化算法。 该算法使用参数的梯度值来最小化损失值。最常用的一阶优化算法是梯度下降。函数梯度可以采用导数 *dxdy* 的多变量表达式进行表

达，用于表示 y 相对于 x 的瞬时变化率 。 通常为了计算多变量函数的导数，用梯度代替导数，并使用导数来计算梯度 。 2.二阶优化算法。 二阶优化算法使用二阶导数（也称为 Hessian 方法）来最小化损失值，以 Newton 法和 Quasi-Newton 法为代表的二阶优化算法目前最大的困难在于计算复杂度 ， 正是由于二阶导数的计算成本较高，因此该方法未得到广泛应用。

梯度下降法

梯度下降法（ Gradient Descent）是机器学习中最常用的优化方法，常常用于求解目标函数的极值。梯度是一个向量，表示函数在该点处的方向导数沿着该方向取得最大值，即函数在该点处沿着该方向变化最快、变化率最大，这个方向即为此梯度的方向，变化率即为该梯度的模 。 梯度下降是一种基于不断迭代的运算方法，每一步都是在求解目标函数的梯度向量，在当前位置的负梯度方向作为新的搜索方向，从而不断迭代。之所以采用在当前位置上的梯度反方向，是由于在该方向上的目标函数下降最快，可以找到局部最小值：同理， 要是沿着梯度的正方向作为新的搜索方向，则找到的是局部最大值。 例如，对于输入$(x1 \ ,x2 \ )$分别表示人的身高和体重，通过$f(x1 \ ,x2 \ )$输出的是偏胖或不偏胖的结果。现给定一堆的$(x1 \ ,x2 \ )$集合，通过训练使得$f(x1 \ ,x2 \ )$对于输入的值都能够判定其属于偏胖或不偏胖。拟合函数可以采用如下公式 ：

$$f(x)=\theta 1 \ x1 \ +\theta 2 \ x2 \ +x3$$

训练的过程则是找到有效的$f(x)$函数，使得训练中的样本均满足$f(x)$的输出结果 。 因此可以定义一个误差函数如下所示，误差函数表达的是预测值与真实值差的平方和的一半：

$$Loss(\theta)=21 \ i=1\sum n \ [(f(xi \ )-yi \ ]2$$

其 中 xi 表示第 i 个输入样本，yi 表示训练样本的期望输出，$f(xi \ )$是实际训练结果的输出 。 对于整个系统而言 ，误差函数越小，则表示对训练样本的拟合程度越高，因此可以将问题转换为对 $Loss(\theta)$求解极小值（极小值时误差最小），换言之，则为当$\theta 1 \ ,\theta 2 \ ,\theta 3$ 取何值时，整个系统的误差最小 。 因此需要求解 $Loss(\Theta)$的梯度，即依次对$\theta 1 \ ,\theta 2 \ ,\theta 3$ 进行求偏导数

在求得偏导数之后，θ需要在当前梯度位置的反方向调整

其中η为学习速率。 经过多次迭代即可得到最优的$\theta 1 \ ,\theta 2 \ ,\theta 3$ ，即在这些参数之下，$f(x)$训练的样本整体误差值最小 。

梯度消失

梯度消失问题（ Gradient Vanishing Problem ）在神经网络层数相对较多时可能会遇到， 且随着神经网络层数的不断增加，发生的概率越明显。 对于梯度消失问题，可以通过图 1 进行说明。

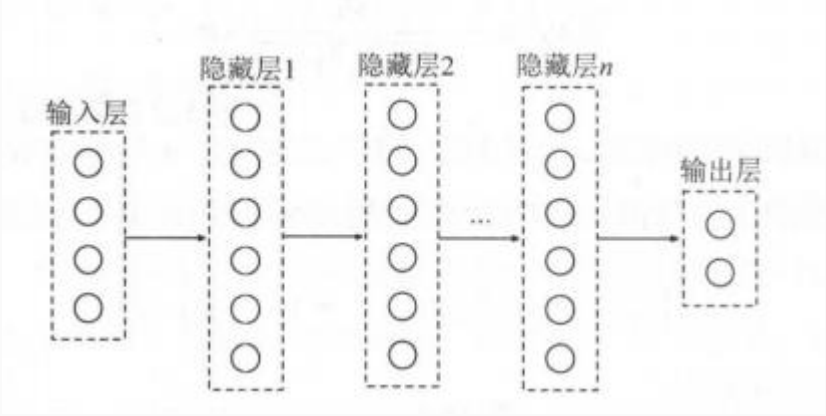


图 1 多层神经网络结构图

针对包含 n 层的隐藏层的神经网络， 一般情况下第 n 个隐藏层的权值还可以正常更新，但是相对于第 1 层和第 2 层的隐藏层，可能存在层的权值更新几乎不变，基本和初始的权值相差较小，这个时候则是发生了梯度消失问题 。 尤其在η较大时，发生梯度消失的概率更高，此刻第 1 层和第 2 层隐藏层只是被当作了 一个映射层，并没有发挥其层次结构的价值。 归纳而言，由于在反向传播算法过程中 ，使用的是矩阵求导的链式法则，即通过一系列的连乘，且连乘的数字在每层都小于 1 ，因此梯度越来越小，最终导致梯度消失 。 从梯度消失的角度而言，可以说明并不是网络结构设计的深度越深越好，但是理论上网络结构的深度越深，表达能力和抽象能力就越强，因而对于梯度消失问题，需要从激活函数以及网络层次结构设计上着手解决，例如激活函数可以考虑用 ReLU 函数代替 Sigmoid 函数 。 与梯度消失问题相反的则是梯度爆炸（ Exploding Gradient ），如果在反向传播算法过程中，连乘的数字在每层都大于 1 ，则梯度会越来越大，从而导致梯度爆炸问题的发生。梯度爆炸问题相对容易解决，可以通过简单地设置阀值来避免梯度爆炸。

三、解题思路

1.编写神经元流程

1. 准备数据：

数据集读入，

数据集乱序，

生成训练集和测试集(即 x train /y train)，

配成(输入特征， 标签)对，每次读入一小撮(batch)；

2.搭建网络：

定义神经网路中所有可训练参数

3.参数优化：

嵌套循环迭代，with 结构更 新参数，显示当前 loss

4.测试效果：

计算当前参数前向传播后的准确率，显示当前 acc

5.acc/loss 可视化

2.优化方法：梯度下降

（1）确定当前位置的损失函数的梯度，对于，其梯度表达式为：

$$\frac{\partial J(\theta_0,\theta_1,\cdots,\theta_n)}{\partial \theta_i}$$

（2）用步长乘以损失函数的梯度，得到当前位置下降的距离，即

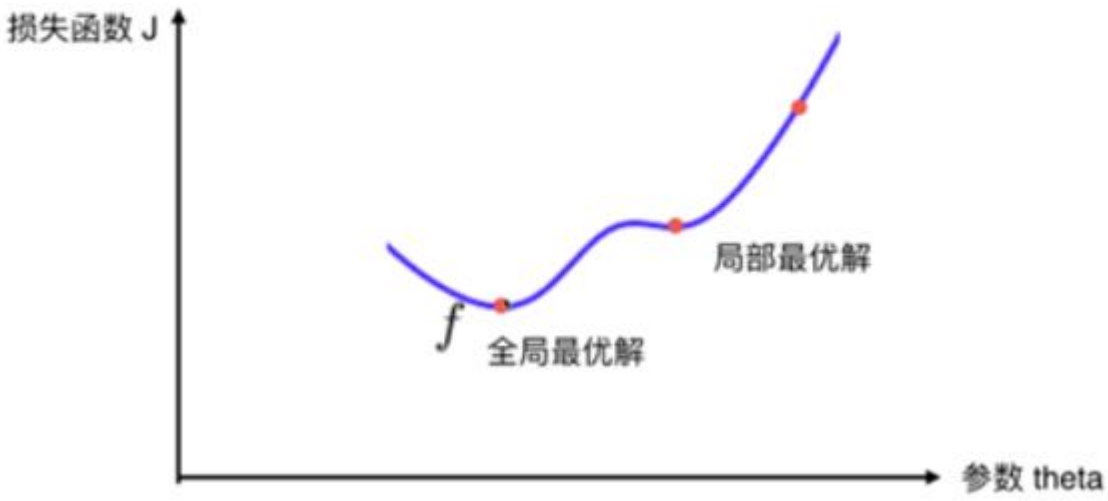
$$\alpha\frac{\partial J(\theta_0,\theta_1,\cdots,\theta_n)}{\partial \theta_i}$$

。

（3）确定是否所有的梯度下降的距离都小于，如果小于则算法终止，当前所有的即为最终结果。否则进入第（4）步。

（4）更新所有的，更新表达式如下，更新完成后转入步骤（1）：

$$\theta_i = \theta_i - \alpha\frac{\partial J(\theta_0,\theta_1,\cdots,\theta_n)}{\partial \theta_i}$$



四、算法伪代码

1. 神经元

Z = np.dot(w.t,X) + b

A = a(Z) = 1/(1+np.exp(-Z))

J = np.sum(-(Y*np.log(A) + (1-Y)*np.log(1-A))/m

dZ = A -Y

dw = np.dot(X,dZ.t)/m

db=np.sum(dZ)/m

2.优化方法：梯度下降

while 循环

梯度=gradient(Θ_0, Θ_1)

$\Theta_0, \Theta_1 = \Theta_0, \Theta_1$-梯度 x 学习率（这里的学习率理解为步伐的大小，这个学习率是需要在学习的过程中不断去调试的，如果步子太大一下子迈过了最低点，太小的话走得太慢）（式子中减去梯度乘以学习率才是下坡的；如果是加号那你是上坡的，因为梯度是带方向的）

其中 while 循环结束的话，可以使用损失函数去计算，（如果说上一次的损失减去这一次的损失小于 0.000001，也就是说损失已经是没有办法再往下边降了，这时候 while 循环就可以结束了）

五、实验设置

1. 神经元

在这个神经元模型中，神经元接收到来自 n 个其他神经元传递过来的输入信号，这些输入信号通过带权重的连接进行传递，神经元接收到的总输入值将与神经元的阈值进行比较，然后通过“激活函数”处理以产生神经元的输出。

激活函数有两种，一种跃阶函数是理想中的激活函数，它将输入值映射为输出值“0”或“1”，显然“1”对应神经元兴奋，“0”对应神经元抑制。然而跃阶函数具有不连续、不光滑等不太好的性质。

实际常用 sigmoid 函数作为激活函数。它把可能在较大范围内变化的输入值挤压到（0，1）输出值范围内，因此也称为“挤压函数”。

2.优化方法：梯度下降

（1）全梯度下降算法（FG）

（2）随机梯度下降算法（SG）

（3）小批量梯度下降法

六、实验结果

1. 神经元

神经网络的基本单位是神经元。

神经元是一种处理单元，是对人脑组织的神经元的某种抽象、简化和模拟。通过神经元，人工神经网络可以以数学模型模拟人脑神经元活动，继而进行高效的计算以及其他处理。

2.优化方法：梯度下降

迭代次数	x 所在位置	调整方向	新的 x 值位置
1	-3	-f'(-3)×0.4	-3+(-f'(-3)×0.4)=-6.00000000e-01
2	-6.00000000e-01	-f'(-6.00000000e-01)×0.4	-6.00000000e-01+(-f'(-6.00000000e-01)×0.4)=-1.20000000e-01
3	-1.20000000e-01	-f'(-1.20000000e-01)×0.4	-1.20000000e-01+(-f'(-1.20000000e-01)×0.4)=-2.40000000e-02
4	-2.40000000e-02	-f'(-2.40000000e-02)×0.4	-2.40000000e-02+(-f'(-2.40000000e-02)×0.4)=--4.80000000e-03
...	...	...	...

七、实验结果分析

两次实验均符合预期输出结果：ture

八、总结

1. 神经网络是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应。神经网络中最基本的成分是神经元（neuron）模型。即上述定义中的“简单单元”。在生物神经网络中，每个神经元与其他申请元相连，当它“兴奋”时，就会向相连的神经元发送化学物质，从而改变

这些神经元内的电位；如果某神经元的电位超过一个“阈值”，那么它就会被激活，即“兴奋”起来，向其他神经元发送化学物质。把许多个这样的神经元按一定的层次结构连接起来，就得到了神经网络。

事实上，我们可以先不考虑神经网络是否真的模拟了生物神经网络，只需将一个神经网络视为包含了许多参数的数学模型，这个模型是若干个函数。有效的神经网络学习算法大多以数学证明为支持。

2.优化方法是有多种的，如果输入数据是稀疏的，选择任一自适应学习率算法可能会得到最好的结果。选用这类算法的另一个好处是无需调整学习率，选用默认值就可能达到最好的结果。

- **感想、体会、建议：**

通过对两个实验的完成，进一步加深了对神经网络之网络基础这部分内容的认识。对于梯度下降、损失函数等掌握和使用的也更加娴熟。在实验过程中碰到了诸多问题，与同学沟通交流后得以解决。三个大实验的完成也让我更好的认识了人工智能原理这门课程，受益颇多。

上机报告成绩、评语：

指导教师签名：