人工智能基础课程设计指导书

《人工智能基础》课程组 编著

上海海洋大学海洋智能信息实验教学示范中心

实验一： Python 基础-1 基本语法与数据类型

一、实验目的

1、掌握 python3 中输入输出相关方法

2、掌握字符串处理函数

3、理解 python3 中列表、元组、字典、collection 类型及相关操作

二、实验内容

1、Python 入门之基础语法：观察测试输入输出的特点，利用 Python3 提供的 print()、format()、input()方法；完成行与缩进，标识符、保留字、注释基础语法知识部分代码

2、Python 入门之字符串处理：完成字符串拼接方式、字符转换、查找和替换等部分的代码。

3、Python 入门之玩转列表：完成列表元素的增删改、元素排序、切片等操作

4、Python 入门之元组与字典：完成元组转换以及元组计算、字典添加、查找、修改并输出。

5、Python 入门之 collections 模块：结合具体代码，实现命名元组、计数器、双向队列、有序字典、默认字典的处理

三、实验步骤

1、编写一个对用户输入，进行加减乘除四则运算的程序。根据提示，在右侧编辑器 Begin-End 区间补充代码，接收用户输入的两个数 a 和 b，对其进行加减乘除四则运算，通过 print 函数打印四次运算结果，使结果输出形式与预期输出保持一致。

2、给定一个字符串，要利用 Python 提供的字符串处理方法 find( )和 replace( )，从该字符串中查找特定的词汇，并将其替换为另外一个更合适的词。例如，给定一个字符串 Where there are a will, there are a way，发现这句话中存在语法错

误，其中 are 应该为 is，需要通过字符串替换将其转换为 Where there is a will, there is a way。

3、利用合适的方法快速创建数字列表，并能够对列表中的元素数值进行简单的统计运算。本部分的编程任务是补全 src/Step1/guests.py 文件的代码，实现相应的功能。具体要求如下：

step 1：将 guests 列表末尾的元素删除，并将这个被删除的元素值保存到 deleted_guest 变量；

step 2：将 deleted_guest 插入到 step 1 删除后的 guests 列表索引位置为 2 的地方；

step 3：将 step 2 处理后的 guests 列表索引位置为 1 的元素删除；

打印输出 step 1 的 deleted_guest 变量；

打印输出 step 3 改变后的 guests 列表。

本关涉及的代码文件 src/Step1/guests.py 的代码框架如下：

```
# coding=utf-8
# 创建并初始化 Guests 列表
guests = []

while True:
    try:
        guest = input()
        guests.append(guest)
    except:
        break
# 请在此添加代码，对 guests 列表进行插入、删除等操作
########## Begin ##########


########## End ##########
```

四、实验报告要求

参见实验报告模板

实验二：Python 基础-2 运算符与控制结构

一、实验目的

1、掌握 Python3 运算符及运算符的优先级

2、掌握 Python3 基本控制结构

二、实验内容

1、Python 入门之运算符的使用：算术、比较、赋值运算符、逻辑运算符、位运算符、成员运算符、身份运算符、运算符的优先级

2、Python 入门之控制结构：顺序与选择结构、三元操作符

3、Python 入门之控制结构：While 循环与 break 语句、for 循环与 continue 语句、循环嵌套、迭代器

三、实验步骤

1、要求对给定的苹果和梨的数量进行算术运算、比较、赋值运算，然后输出相应的结果

2、本任务是理解选择结构，学会使用最基本的选择语句 if-else。请根据要求实现以下某公司根据员工的工龄来决定员工工资的涨幅，如下所示：

工龄大于等于 5 年并小于 10 年时，涨幅是现工资的 5%；

工龄大于等于 10 年并小于 15 年时，涨幅是现工资的 10%；

工龄大于等于 15 年时，工资涨幅为 15%。

编程要求：将 src/step2/choose.py 中的代码补充完毕，然后点击评测。

```
1.     workYear = int(input())
2.     # 请在下面填入如果 workYear < 5 的判断语句
3.     ########## Begin ##########
4.
5.     ########## End ##########
6.         print("工资涨幅为 0")
7.     # 请在下面填入如果 workYear >= 5 and workYear < 10 的判断语句
8.     ########## Begin ##########
```

```
9.
10.     ########## End ##########
11.         print("工资涨幅为 5%")
12.     # 请在下面填入如果 workYear >= 10 and workYear < 15 的判断语句
13.     ########## Begin ##########
14.
15.     ########## End ##########
16.         print("工资涨幅为 10%")
17.     # 请在下面填入当上述条件判断都为假时的判断语句
18.     ########## Begin ##########
19.
20.     ########## End ##########
21.         print("工资涨幅为 15%")
```

3、本关的编程任务是补全 sumScore.py 文件中的部分代码，具体要求如下：

当输入学生人数后，填入在 for 循环遍历学生的代码；

当输入各科目的分数后的列表后，填入 for 循环遍历学生分数的代码。

本关涉及的代码文件 sumScore.py 的代码框架如下:

```
1.      studentnum = int(input())
2.      # 请在此添加代码，填入 for 循环遍历学生人数的代码
3.      ########## Begin ##########
4.
5.      ########## End ##########
6.          sum = 0
7.          subjectscore = []
8.          inputlist = input()
9.          for i in inputlist.split(','):
10.             result = i
11.             subjectscore.append(result)
12.         # 请在此添加代码，填入 for 循环遍历学生分数的代码
13.         ########## Begin ##########
14.
15.         ########## End ##########
16.             score = int(score)
17.             sum = sum + score
18.         print("第%d 位同学的总分为:%d" %(student,sum))
```

四、实验报告要求

参见实验报告模板

实验三：Python 基础-3 函数与模块

一、实验目的

1、掌握 Python3 函数结构和调用的相关知识。

2、了解并掌握 Python 模块和内置函数的相关知识。

二、实验内容

1、Python 入门之函数结构：函数的参数、函数的返回值、函数的作用域

2、Python 入门之函数调用：内置函数、函数正确调用

3、Python 入门之模块：模块的定义、内置模块中的内置函数

三、实验步骤

1、本关的编程任务是补全 src/step3/scope.py 文件的代码，实现相应的功能。

具体要求：编写程序，功能是求两个正整数的最小公倍数；

要求实现方法：先定义一个 private 函数 _gcd()求两个正整数的最大公约数，再定义 public 函数 lcm()调用 _gcd()函数求两个正整数的最小公倍数；

调用函数 lcm()，并将输入的两个正整数的最小公倍数输出。

本关涉及的代码文件 src/step3/scope.py 的代码框架如下：

```
1.      # coding=utf-8
2.      # 输入两个正整数 a,b
3.      a = int(input())
4.      b = int(input())
5.
6.      # 请在此添加代码，求两个正整数的最小公倍数
7.      ########## Begin ##########
8.
9.      ########## End ##########
10.
11.     # 调用函数，并输出 a,b 的最小公倍数
12.     print(lcm(a,b))
```

2、本关的编程任务是补全 src/Step2/func_call.py 文件的代码，实现相应的功能。

具体要求如下：定义一个函数，实现对输入的数值列表进行从小到大的顺序排序；输出排序后的数值列表。

本关涉及的代码文件 src/Step2/func_call.py 的代码框架如下：

```
1.      # coding=utf-8
2.      # 输入数字字符串，并转换为数值列表
3.      a = input()
4.      num1 = eval(a)
5.      numbers = list(num1)
6.
7.      # 请在此添加代码，对数值列表 numbers 实现从小到大排序
8.      ########## Begin ##########
9.
10.     ########## End ##########
```

四、实验报告要求

参见实验报告模板

实验四：知识表示与逻辑推理

一、实验目的

1、学习命题与逻辑相关知识，熟悉命题函数以及命题函数的等价

2、学习常用逻辑等值式及逻辑公式的等值演算，熟悉逻辑推理相关理论。

3、了解并学会使用 sympy 库推理相关方法

二、实验内容

1、学习命题与逻辑相关知识，编程得出$(P \to Q) \wedge R$的真值表。

2、熟悉命题函数以及命题函数的等价，完成练习。

3、学习常用逻辑等值式及逻辑公式的等值演算，编程验证等值演算结果。

4、熟悉逻辑推理的相关理论，完成练习。

三、实验步骤

阅读代码中注释部分要求，补充实现 begin-end 间代码，并运行测试结果。

```
#coding=utf-8
import sympy as sym
# 定义符号 p,q。
#***** Begin *****#

#***** End *****#
# 输出析取三段论，中间逗号使用析取符号。
#***** Begin *****#

#***** End *****#
# 将判断命题公式是否重言式写成函数。
#***** Begin *****#

#***** End *****#
# 按照例子验证析取三段论为重言式。
#***** Begin *****#

#***** End *****#
```

```
19. # 判断((p -> q) ^ q) -> p 是否为重言式。
20. #***** Begin *****#
21.
22. #***** End *****#
```

四、实验报告要求

参见实验报告模板

实验五：搜索问题与技术

一、实验目的

1、理解掌握深度优先、广度优先算法原理。

2、掌握A^*算法原理并理解算法实现。

二、实验内容与要求

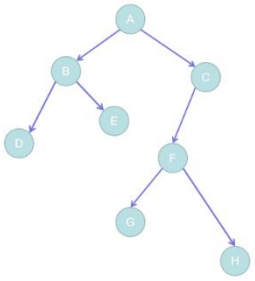
1、结合具体案例，理解掌握深度优先、广度优先算法原理。

2、学习使用A^*算法求解扫地机器人从起点到终点的最短路径搜索。

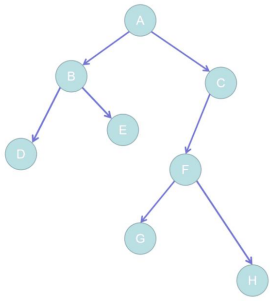
三、实验步骤

1、利用盲目搜索策略实现机器人路径规划

深度优先搜索算法是一种常用的盲目搜索策略，基本思想就是一路走到底，到底走不通了再回头。举个例子，现在身处这样一个迷宫的地点 A ，想要走到地点 H，应该怎么办呢？



首先先从地点 B 和地点 C 中选一个走，假设选择了地点 B，然后接着从地点 D 和地点 E 中二选一。走到地点 D 时发现，已经是死胡同了，而且并不是想要去的地点 H，所以就需要往后退一步，回到地点 B。然后再走到没试过的地点 E，接下来的过程以此类推。整个搜索过程如图所示(当某个结点在搜索过程中搜索过会标记成蓝色)：



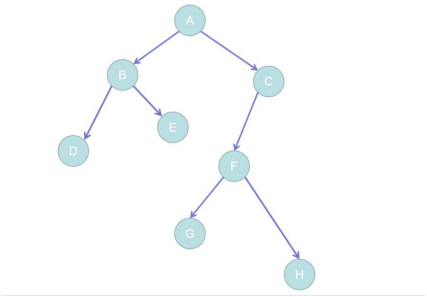
算法实现

想要实现深度优先搜索算法来实现刚刚的走迷宫问题比较简单，需要不断的递归调用自身来实现，Python 代码如下：

```
1.    '''
2.    迷宫的定义：
3.    A 能走到 B 和 C
4.    B 能走到 D 和 E
5.    C 能走到 F
6.    F 能走到 G 和 H
7.    D,E,G,H 是死胡同
8.    '''
9.    graph = {
10.               'A': ['B', 'C'],
11.               'B': ['D', 'E'],
12.               'C': ['F'],
13.               'F': ['G', 'H'],
14.               'D': [],
15.               'E': [],
16.               'G': [],
17.               'H': []
18.           }
19.
20.   def dfs(graph, start, visited=None):
21.       '''
22.       深度优先搜索，从 A 走到 H
23.       :param graph: 待搜索的迷宫
24.       :param start: 开始搜索的起点
25.       :param visited:  已经搜索过的地点集合
26.       '''
27.       if visited is None:
28.           visited = set()
29.       visited.add(start)
30.       print(start, end='')
31.       # 当前地点为 H 时结束搜索
32.       if start == 'H':
33.           return
34.       # 看看当前位置有哪些路可以走，如果能走并且之前没有走过就走
35.       for v in graph[start]:
36.           if v not in visited:
37.               dfs(graph, v, visited)
38.   # 从迷宫的 A 走到 H，并按搜索的先后顺序打印地点
39.   dfs(graph, 'A')
```

打印结果：ABDECFGH

广度优先搜索算法也是一种常用的盲目搜索策略，思想与深度优先搜索算法相反，广度优先搜索可以看成是“横着”搜索。想从地点 A 走到地点 H。那么首先会走到地点 B，然后发现不是我想要去的地点 H，所以退一步回到地点 B 再走到 C。结果发现又不是想要去的地方，所以暂且先走到地点 B，然后跟刚刚的思路一样走到地点 D 和地点 E。如果整个搜索过程都按照这个思路进行的话，就称为广度优先搜索。过程如图所示：



算法实现

想要实现深度优先搜索算法来实现刚刚的走迷宫问题比较简单，需要一个额外的数据结构（队列）。队列的特性是先进先出（可以想象成排队），队列在这里的作用是将与当前地点相邻的地点保存起来。当发现一个相邻地点就将其放入队列，当要往前走一步时从队列的头部抽出一个地点出来，当搜索到地点 H 或者队列为空时算法结束。Python 代码如下：

```
1. '''
2. 迷宫的定义：
3. A 能走到 B 和 C
4. B 能走到 D 和 E
5. C 能走到 F
6. F 能走到 G 和 H
7. D,E,G,H 是死胡同
8. '''
9. graph = {
10.           'A': ['B', 'C'],
11.           'B': ['D', 'E'],
12.           'C': ['F'],
13.           'F': ['G', 'H'],
14.           'D': [],
15.           'E': [],
16.           'G': [],
```

```
            'H': []
        }
def bfs(graph, start):
    '''
    广度优先搜索，从 A 走到 H
    :param graph: 待搜索的迷宫
    :param start: 开始搜索的起点
    '''
    # queue 为队列，当队列为空或者当前地点为 H 时搜索结束
    visited, queue = set(), [start]
    while queue:
        # 从队列中出队，即当前所处的地点
        vertex = queue.pop(0)
        if vertex not in visited:
            visited.add(vertex)
            print(vertex, end='')
            # 当前地点是`H`的话就结束搜索
            if vertex == 'H':
                return
            # 将当前所处地点所能走到的地点放入队列
            for v in graph[vertex]:
                if v not in visited:
                    queue.extend(v)
# 从迷宫的 A 走到 H，并按搜索的先后顺序打印地点
bfs(graph, 'A')
```

打印结果：ABCDEFGH

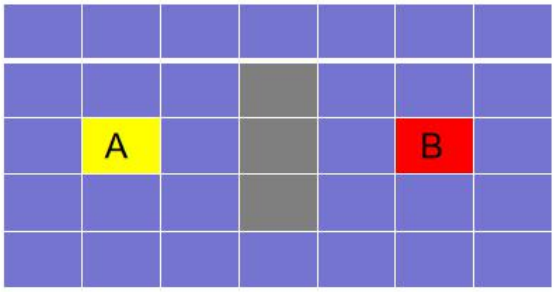
参考算法原理介绍，根据注释部分要求，补充实现 begin-end 间代码，并运行测试结果。

```
    def PlayMazz(mazz, start, end):
        '''''
        走迷宫，从 start 走到 end
        :param mazz: 迷宫
        :param start: 迷宫的起点
        :param end: 迷宫的出口
        '''
        # queue 为队列，当队列为空或者当前地点为 H 时搜索结束
        visited, queue = set(), [start]
        while queue:
            # 从队列中出队，即当前所处的地点
            vertex = queue.pop(0)
            if vertex not in visited:
```

```
                visited.add(vertex)
                print(vertex, end='')
                #********* Begin *********#
                #当走到出口时结束算法


                #********* End *********#
                # 将当前所处地点所能走到的地点放入队列
                for v in mazz[vertex]:
                    if v not in visited:
                        queue.extend(v)
```

2、学习使用A^*算法求解扫地机器人从起点到终点的最短路径搜索。

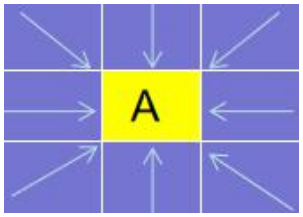
启发式搜索策略在搜索过程中引入启发信息，减少搜索范围，以便尽快的找到解，其中最为经典且最常用的算法是A^*算法，其最为典型的应用就是寻路。假设有一个扫地机器人，这个扫地机器人需要从 A 走到 B 去充电（其中灰色部分表示墙，扫地机器人不能穿墙）。怎样才能让扫地机器人更加智能地找到去充电的最短路径呢？可以使用A^*算法！



在了解A^*算法的算法流程之前，先要知道两个列表：开启列表和关闭列表。开启列表其实就是一个等待检查的方块的列表，关闭列表是不需要检查的方块的列表。来看看A^*算法的执行流程。

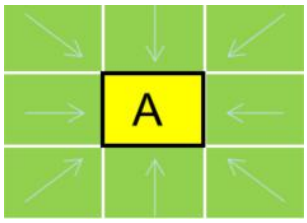
1 ：从起点 A 开始，把它作为待处理的方块，并存到开启列表中。

2 ：寻找起点 A 周围可以到达的方块，把这些地点存到开启列表中，并将它们的父方块设置成 A。如下图所示（箭头指向方块的“父亲”）：



3 ：从开启列表中删除 A ，并将 A 加到关闭列表。如下图所示（方块变

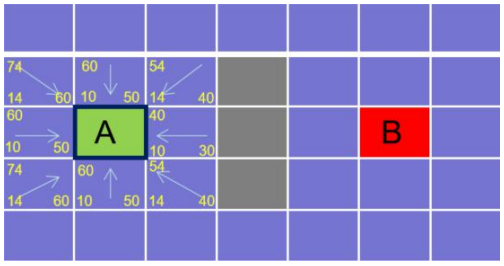
成绿色表示方块在开启列表中，黑色描边表示方块在关闭列表中）：



4 ：从开启列表中找出最好的方块作为下一步要走到的位置。

那么什么样的才是最好的呢？可以通过公式：$F = G + H$来计算。其中 G 表示从起点 A 移动到指定方块的代价（可以斜着移动）。H 表示从指定的方块移动到终点 B 的预计代价（在这里假定只可以上下左右四个方向移动）。

来看个例子。假设横竖移动的代价为 10 ，斜着移动的代价为 14 。可以计算出起点 A 周围方块的 G，H，F 的值。如下图所示（方块的左上角的数字表示 F ，左下角表示 G ，右下角表示 H ）：



有了这些值之后，就从开启列表中选出 F 值最低的方块，并走过去。

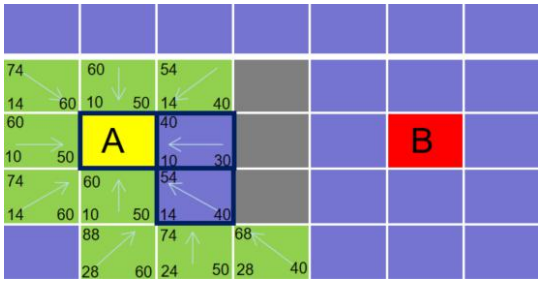
5、把它从开启列表中删除，并存到关闭列表中。

6、检查它所有可以到达的方块，若方块并不在开启列表中，则将其加入到开启列表，并计算它的 G,H,F 的值，并设置父方块。若方块已经存在于开启列表中(假设为方块 D)，就检查如果用新的路径到达 D 的话， G 值是否会更低一些，如果新的 G 值更低，那就把 D 的父方块改为它。如果新的 G 值更高，就什么都不做。

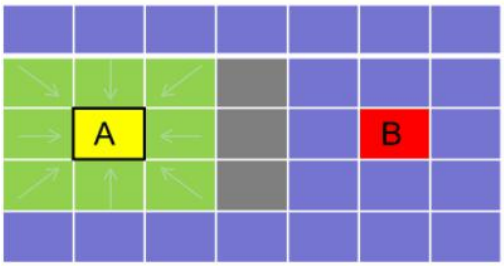
7、继续检查开启列表哪个方块的 F 值最小，我们发现有两个方块的 F 值

为 54 ，所以可以随便选一个，比如选下面那个方块走过去。

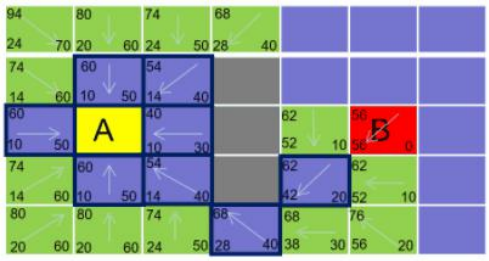
8、将下面的 3 个方块加入到开启列表，并更新其父方块和 G,H,F 的值。



就这样， A 算法就是从开启列表找出 F 值最小的，将它从开启列表中移除掉，并加到关闭列表中，再继续找出它周围可以到的的方块，如此循环下去。当开启列表里出现了终点的时候，说明最优路径已经找到了。整个过程如下图所示：



还差最后一步，就是获取路径。获取路径的思想很简答，由于方块都有父方快，所以从终点 B 开始，沿着父方块的方向走，能够回到起点 A ，而这一条路径就是从 A 到 B 的路径。



算法实现

map 地图节点构造如下：

```
    def GenerateMap(m, n):
        map = list()
        for j in range(m):
            nodeRow = list()
            map.append(nodeRow)
            for i in range(n):
                node = Node()
                node.y = j
                node.x = i
                node.unable = False
                node.distanceFromDes = -1  # 距离终点的距离
                node.distanceFromOri = -1  # 距离起点的距离
                node.allDistance = -1
                node.added = False
                node.closed = False
                node.parent = None
                nodeRow.append(node)
        return map
```

算法的伪代码如下：

```
# 构造开启列表，开启列表为 openedList
openedList = list()
# 将起点的 G 和 F 设置成 0，H 已经设置过了 map 表示地图节点，originIndex 表示出发地
node = map[oriIndex[0]][oriIndex[1]]
node.distanceFromOri = 0
node.allDistance = 0
# 将起点存到开启列表中
openedList.append(node)
node.added = True
# 循环检查开启列表
while len(openedList) != 0:
    # 将开启列表中第一个方块删除
    node = openedList.pop(0)
    # 方块的 closed 状态设置成 True，相当于加入到关闭列表
    node.closed = True
    # 如果走到了终点就获取路径
    if node.y == desIndex[0] and node.x == desIndex[1]:
        finalListNeedReverse = list()
        while node != None:
            finalListNeedReverse.append(node)
            node = node.parent
            finalListNeedReverse.reverse()
            return finalListNeedReverse
    # neighboursList 存放的是当前方块周围的方块
```

```
        neighboursList = list()
        y = node.y
        x = node.x
        parentDistanceFromOri = node.distanceFromOri
        # 检查当前方块周围的方块
        for needNodey in (y + 1, y, y - 1):
            if needNodey < 0 or needNodey >= mapSize[0]:
                continue
            for needNodex in (x + 1, x, x - 1):
                if needNodex < 0 or needNodex >= mapSize[1]:
                    continue
                needNode = map[needNodey][needNodex]
                # 不考虑不可达、在关闭列表中以及已经在开启列表中的方块
                if needNode.unable == True or needNode.closed == True or needNod
e.added == True:
                    continue
                yOffset = needNodey - y
                xOffset = needNodex - x
                allOffset = yOffset + xOffset
                # 计算可达并没有被添加到开启列表中的方块的 G 值
                if allOffset == 1 or allOffset == -1:
                    distanceFromOri = parentDistanceFromOri + 1
                else:
                    distanceFromOri = parentDistanceFromOri + 1.4
                # 更新最小的 G 值
                if needNode in neighboursList:
                    if distanceFromOri < needNode.distanceFromOri:
                        needNode.distanceFromOri = distanceFromOri
                else:
                    needNode.distanceFromOri = distanceFromOri
                    neighboursList.append(needNode)
        # 设置 neighboursList 中的方块的父方块，F 值等
        for needNode in neighboursList:
            needNode.parent = node
            needNode.allDistance = needNode.distanceFromOri + needNode.distanceF
romDes
            needNode.added = True
            openedList.append(needNode)
        # 将方块根据 F 值从小到大排序，这样每次只要获取列表中的一个方块就能得到 F 值最小的
方块
        openedList.sort(key=lambda x: x.allDistance)
```

参考算法原理介绍，根据注释部分要求，补充实现 begin-end 间代码，并运行测试结果。

```
from a_star_utils import Node
def A_star(map, mapSize, start, end):
    '''''
    A*算法，从 start 走到 end
    :param map:地图
    :param mapSize:地图大小，例如[10,10]表示地图长 10 宽 10
    :param start:表示出发地，类型为列表，如[1,2]表示出发地为地图中的第 1 行第 2 列的方块
    :param end:表示目的地，类型为列表，如[1,2]表示目的地为地图中的第 1 行第 2 列的方块
    :return:从出发地到目的地的路径
    '''
    openedList = []
    #********* Begin *********#

    #********* End *********#
    node.distanceFromOri = 0
    node.allDistance = 0
    #********* Begin *********#

    #********* End *********#
    while len(openedList) != 0:
        node = openedList.pop(0)
        node.closed = True
        if node.y == end[0] and node.x == end[1]:
            finalListNeedReverse = []
            while node != None:
                finalListNeedReverse.append(node)
                node = node.parent
            finalListNeedReverse.reverse()
            return finalListNeedReverse
        neighboursList = []
        y = node.y
        x = node.x
        parentDistanceFromOri = node.distanceFromOri
        for needNodey in (y + 1, y, y - 1):
            if needNodey < 0 or needNodey >= mapSize[0]:
                continue
            for needNodex in (x + 1, x, x - 1):
                if needNodex < 0 or needNodex >= mapSize[1]:
                    continue
                needNode = map[needNodey][needNodex]
```

```
                    if needNode.unable == True or needNode.closed == True or nee
dNode.added == True:
                        continue
                    yOffset = needNodey - y
                    xOffset = needNodex - x
                    allOffset = yOffset + xOffset
                    if allOffset == 1 or allOffset == -1:
                        distanceFromOri = parentDistanceFromOri + 1
                    else:
                        distanceFromOri = parentDistanceFromOri + 1.4
                    if needNode in neighboursList:
                        #********* Begin *********#

                        #********* End *********#
                    else:
                        needNode.distanceFromOri = distanceFromOri
                        neighboursList.append(needNode)
            for needNode in neighboursList:
                needNode.parent = node
                #********* Begin *********#

                # ********* End *********#
                needNode.added = True
                openedList.append(needNode)
            openedList.sort(key=lambda x: x.allDistance)
        return None
```

四、实验报告要求

参见实验报告模板

实验六：机器学习-线性回归

第一关：数据载入与分析

一、实验目的

1、理解掌握线性回归模型原理，并利用工具库或包实现线性回归中编写数据载入，损失函数,梯度下降函数三部分。

2、学会使用 sklearn 构建线性回归算法，并利用经典数据集掌握线性回归分析过程。

二、实验内容

1、利用 pandas、numpy 构建一个完整的线性回归模型，主要编写数据载入，损失函数,梯度下降函数三部分。

2、使用 sklearn 构建线性回归算法，并利用已知标签的波斯顿房价数据对模型进行训练，然后对未知标签的波斯顿房价数据进行预测。

三、实验步骤

1、数据载入与分析。利用 pandas 读入数据 data，并将数据属性分别命名为'Population'和'Profit'。

```
#encoding=utf8
import os
import pandas as pd

if __name__ == "__main__":
    path = os.getcwd() + '/ex1data1.txt'
    #利用 pandas 读入数据 data，并将数据属性分别命名为'Population'和'Profit'
    #********* begin *********#

    #********* end *********#
    print(data.shape)
```

2、计算损失函数:

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

根据以上公式，编写计算损失函数$computeCost(X, y, theta)$，最后返回 cost。

X：一元数据矩阵，即 Population 数据；

y：目标数据，即 Profit 数据；

theta：模型参数；

cost：损失函数值。

```
#encoding=utf8
import numpy as np

def computeCost(X, y, theta):
    #根据公式编写损失函数计算函数
    #********* begin *********#

    #********* end *********#
    return cost
```

3、进行梯度下降得到线性模型:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)}) x_j^{(i)}$$

根据以上公式，编写计算损失函数$gradientDescent(X,y,theta,alpha,iters)$，最后返回theta, cost。

x：一元数据矩阵，即 Population 数据；

y：目标数据，即 Profit 数据；

theta：模型参数；

m：数据规模；

α：学习率。

```
#encoding=utf8
import numpy as np

def computeCost(X, y, theta):
    inner = np.power(((X * theta.T) - y), 2)
    return np.sum(inner) / (2 * len(X))

def gradientDescent(X, y, theta, alpha, iters):
    temp = np.matrix(np.zeros(theta.shape))
    parameters = int(theta.ravel().shape[1])
    cost = np.zeros(iters)
```

```
12.
13.         for i in range(iters):
14.             error = (X * theta.T) - y
15.
16.             for j in range(parameters):
17.                 #********* begin *********#
18.
19.                 #********* end *********#
20.             theta = temp
21.             cost[i] = computeCost(X, y, theta)
22.
23.         return theta, cost
```

4、使用 sklearn 构建线性回归算法，并利用已知标签的波斯顿房价数据对模型进行训练，然后对未知标签的波斯顿房价数据进行预测。

波斯顿房价数据集共有 506 条波斯顿房价的数据，每条数据包括对指定房屋的 13 项数值型特征和目标房价组成。

在 sklearn 中通过 LinearRegression 方法实现线性回归。

LinearRegression 类中的 fit 函数用于训练模型，fit 函数有两个向量输入：

其中，X 大小为 [样本数量,特征数量] 的 ndarray，存放训练样本；Y 值为整型，大小为 [样本数量] 的 ndarray，存放训练样本的标签值。

LinearRegression 类中的 predict 函数用于预测，返回预测值，predict 函数有一个向量输入：

其中，X 大小为 [样本数量,特征数量] 的 ndarray，存放预测样本。

```
1.  #encoding=utf8
2.  from sklearn.linear_model import LinearRegression
3.  def lr(train_data,train_label,test_data):
4.  '''
5.  input:train_data 用来训练的数据
6.  train_label 用来训练的标签
7.  test_data 用来测试的数据
8.  '''
9.  #********* Begin *********#
10.
11. #********* End *********#
12. return predict
```

四、实验报告要求

参见实验报告模板

实验七、机器学习-逻辑回归

一、实验目的

理解并掌握逻辑回归核心思想。

二、实验内容

1、根据理论知识编程实现 sigmoid 函数。

2、用 Python 构建梯度下降算法，并求取目标函数最小值。

3、使用 sklearn 构建逻辑回归算法，并利用癌细胞数据对模型进行训练，然后对未知的癌细胞数据进行识别。

三、实验步骤

1、根据所学知识完成编程题，实现 sigmoid 函数。

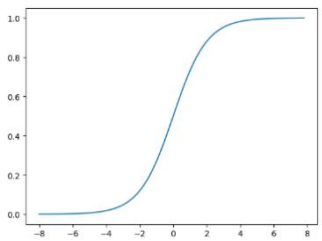
逻辑回归通过回归的思想来解决二分类问题的算法。逻辑回归将样本特征和样本所属类别的概率联系在一起，假设已经训练好了一个逻辑回归的模型为$f(x)$，模型输出是样本 x 的标签是 1 的概率，则该模型可以表示$\hat{p} = f(x)$。若得到了样本 x 属于标签 1 的概率后，能想到当$\hat{p} > 0.5$ 时 x 属于标签 1 ，否则属于标签 0 。所以有

$$\hat{y} = \begin{cases} 0 & \hat{p} < 0.5 \\ 1 & \hat{p} > 0.5 \end{cases}$$

逻辑回归中样本所属标签的概率和线性回归有关系，将线性回归的输出作为另一个函数的输入，该函数能够进行转换，假设函数为 σ ，转换后的概率为$\hat{p}$，则逻辑回归在预测时可以看成 $\hat{p} = \sigma(W^T x + b)$。σ 其实就是 sigmoid 函数：

$$\sigma(t) = 1/1 + e^{-t}$$

函数图像如下图所示：



从 sigmoid 函数图像可以看出当 t 趋近于 $-\infty$ 时函数值趋近于 0，当 t 趋

近于 $+\infty$ 时函数值趋近于 1 。可见 sigmoid 函数的值域是 (0,1) ，满足要将 $(-\infty,+\infty)$ 的实数转换成 (0,1) 的概率值的需求。因此逻辑回归在预测时可以看成:

$$\hat{p} = 1/(1 + e^{-W^T x+b})$$

```
#encoding=utf8
import numpy as np
#sigmoid 函数
def sigmoid(t):
    '''''
    完成 sigmoid 函数计算
    :param t: 负无穷到正无穷的实数
    :return: 转换后的概率值
    '''
    #********** Begin **********#

    #********** End **********#
```

2、用 Python 构建梯度下降算法，并求取目标函数最小值。

梯度下降的中心思想就是迭代地调整参数从而使损失函数最小化。通过测量参数向量 θ 相关的损失函数的局部梯度，并不断沿着降低梯度的方向调整，直到梯度降为 0，达到最小值。梯度下降公式如下：

$$\theta \coloneqq \theta - \eta \nabla loss$$

对应到每个权重公式为：

$$w_i \coloneqq w_i - \eta \frac{\partial loss}{\partial w_i}$$

其中 η 为学习率，是 0 到 1 之间的值，是个超参数，需要我们自己来确定大小。

梯度下降算法流程：

1. 随机初始参数；
2. 确定学习率；
3. 求出损失函数对参数梯度；
4. 按照公式更新参数；

重复 3、4 直到满足终止条件（如：损失函数或参数更新变化值小于某个阈值，或者训练次数达到设定阈值）。

```
# -*- coding: utf-8 -*-

import numpy as np
import warnings
warnings.filterwarnings("ignore")

def gradient_descent(initial_theta,eta=0.05,n_iters=1000,epslion=1e-8):
    '''''
    梯度下降
    :param initial_theta: 参数初始值，类型为 float
    :param eta: 学习率，类型为 float
    :param n_iters: 训练轮数，类型为 int
    :param epslion: 容忍误差范围，类型为 float
    :return: 训练后得到的参数
    '''
    #    请在此添加实现代码    #
    #********** Begin *********#

    #********** End **********#
```

3、使用 sklearn 构建逻辑回归算法，并利用癌细胞数据对模型进行训练，然后对未知的癌细胞数据进行识别。

乳腺癌数据集，其实例数量是 569 ，实例中包括诊断类和属性，帮助预测的属性一共 30 个，各属性包括为 radius 半径（从中心到边缘上点的距离的平均值）， texture 纹理（灰度值的标准偏差）等等，类包括： WDBC-Malignant 恶性和 WDBC-Benign 良性。用数据集的 80% 作为训练集，数据集的 20% 作为测试集，训练集和测试集中都包括特征和诊断类。

在 sklearn 中，使用 LogisticRegression 方法实现逻辑回归算法，LogisticRegression 的构造函数中有三个常用的参数可以设置：

- solver：{‘newton-cg’,’lbfgs’,’liblinear’,’sag’,’saga’}，分别为几种优化算法。默认为 liblinear 。
- C ：正则化系数的倒数，默认为 1.0，越小代表正则化越强。
- max_iter ：最大训练轮数，默认为 100。和 sklearn 中其他分类器一样，LogisticRegression 类中的 fit 函数用于训练模型，fit 函数有两个向量输入：
- X ：大小为 [样本数量,特征数量] 的 ndarray，存放训练样本。

- Y ：值为整型，大小为 [样本数量] 的 ndarray，存放训练样本的分类标签。LogisticRegression 类中的 predict 函数用于预测，返回预测标签，predict 函数有一个向量输入：
- X ：大小为[样本数量,特征数量]的 ndarray，存放预测样本。

LogisticRegression 的使用代码如下：

```
1.      logreg = LogisticRegression(C=100)
2.      logreg.fit(train_data, train_label)
3.      predict = logreg.predict(test_data)
```

其中，train_data 为训练数据，train_label 为训练标签，test_data 为测试数据。predict 为癌细胞诊断结果。

在 begin-end 间补充代码，使用 sklearn 实现逻辑回归算法。测试说明：程序会调用你实现的方法对癌细胞数据进行识别，正确率大于 0.95 则视为通关。

```
1.      #encoding=utf8
2.      from sklearn.linear_model import LogisticRegression
3.
4.      def cancer_clf(train_data,train_label,test_data):
5.          '''
6.          train_data(ndarray):训练数据
7.          train_label(ndarray):训练标签
8.          test_data(ndarray):ces 数据
9.          '''
10.         #********* Begin *********#
11.
12.         #********* End *********#
13.         return predict
```

四、实验报告要求

参见实验报告模板

实验八：机器学习-线性判别分析

一、实验目的

1、理解并掌握 LDA 模型思想。

2、分别利用 python 语言和 sklearn 构建 LDA 对数据进行降维。

二、实验内容

1、使用 python 实现 LDA 并对给定数据进行降维。

2、利用 sklearn 构建 LDA 对数据进行降维。

三、实验步骤

1、使用 python 实现 LDA 并对给定数据进行降维。

编程要求：实现 LDA 方法。算法流程如下：

```
1. 划分出第一类样本与第二类样本
2. 获取第一类样本与第二类样本中心点
3. 计算第一类样本与第二类样本协方差矩阵
4. 计算类内散度矩阵
5. 计算 w
6. 计算新样本集
```

测试说明：程序会调用你实现好的方法对随机生成的二维数据进行降维，若处理后数据与正确数据 l2 距离小于 10，则视为通关，否则输出你处理后的数据。

```
1. #encoding=utf8
2. import numpy as np
3. from numpy.linalg import inv
4. def lda(X, y):
5.     '''
6.     input:X(ndarray):待处理数据
7.           y(ndarray):待处理数据标签，标签分别为 0 和 1
8.     output:X_new(ndarray):处理后的数据
9.     '''
10.    #********* Begin *********#
11.
12.        #********* End *********#
13.    return X_new
```

2、利用 sklearn 构建 LDA 对数据进行降维。

编程要求：利用 sklearn 实现 LDA 方法。

```
#encoding=utf8
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
def lda(x,y):
    '''
    input:x(ndarray):待处理数据
          y(ndarray):待处理数据标签
    output:x_new(ndarray):降维后数据
    '''
    #********* Begin *********#

    #********* End *********#
    return x_new
```

四、实验报告要求

参见实验报告模板

实验九：机器学习- Adaboost

一、实验目的

1、理解并实现 Adaboost，并通过相关花数据集对 Adaboost 模型进行训练和测试。

2、调用 sklearn 中的 Adaboost 模型，并通过给定数据集对 Adaboost 模型进行训练和测试。

二、实验内容

1、用 Python 实现 Adaboost，并通过鸢尾花数据集中鸢尾花的 2 种属性与种类对 Adaboost 模型进行训练。我们会调用你训练好的 Adaboost 模型，来对未知的鸢尾花进行分类。

2、调用 sklearn 中的 Adaboost 模型，并通过癌细胞数据集对 Adaboost 模型进行训练。我们会调用你训练好的 Adaboost 模型，来对未知的癌细胞进行识别。

三、实验步骤

1、编程要求：根据提示，在编辑器的 begin-end 间补充 Python 代码，实现 Adaboost 算法，并利用训练好的模型对鸢尾花数据进行分类。

测试说明：只需返回分类结果即可，程序内部会检测您的代码，预测正确率高于 95% 视为过关。

```
#encoding=utf8
import numpy as np
#adaboost 算法
class AdaBoost:
    '''''
    input:n_estimators(int):迭代轮数
          learning_rate(float):弱分类器权重缩减系数
    '''
    def __init__(self, n_estimators=50, learning_rate=1.0):
        self.clf_num = n_estimators
        self.learning_rate = learning_rate
    def init_args(self, datasets, labels):
        self.X = datasets
```

```
            self.Y = labels
            self.M, self.N = datasets.shape
            # 弱分类器数目和集合
            self.clf_sets = []
            # 初始化 weights
            self.weights = [1.0/self.M]*self.M
            # G(x)系数 alpha
            self.alpha = []
        #********* Begin *********#
           #********* End *********#
```

2、利用 sklearn 中的 Adaboost 解决乳腺癌数据集中的预测分类问题

乳腺癌数据集的实例数量是 569 ，实例中包括诊断类和属性，帮助预测的属性一共 30 个，各属性包括为 radius 半径（从中心到边缘上点的距离的平均值），texture 纹理（灰度值的标准偏差）等等，类包括： WDBC-Malignant 恶性和 WDBC-Benign 良性。用数据集的 80% 作为训练集，数据集的 20% 作为测试集，训练集和测试集中都包括特征和诊断类。想要使用该数据集可以使用如下代码：

```
    from sklearn.datasets import load_breast_cancer
    #加载数据
    cancer = load_breast_cancer()
    #获取特征与标签
    x,y = cancer['data'],cancer['target']
    #划分训练集与测试集
    x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,random_state=666)
```

数据集中部分数据与标签如下图所示：

mean radiu	mean textu	mean perin	mean area	mean smoot	mean compa	mean conca	mean conca	mean symme	mean fract
13.85	15.18	88.99	587.4	0.09516	0.07688	0.04479	0.03711	0.211	0.05853
13.49	22.3	86.91	561	0.08752	0.07698	0.04751	0.03384	0.1809	0.05718
18.22	18.7	120.3	1033	0.1148	0.1485	0.1772	0.106	0.2092	0.0631
12.8	17.46	83.05	508.3	0.08044	0.08895	0.0739	0.04083	0.1574	0.0575
11.08	18.83	73.3	361.6	0.1216	0.2154	0.1689	0.06367	0.2196	0.0795
13.14	20.74	85.98	536.9	0.08675	0.1089	0.1085	0.0351	0.1562	0.0602
11.89	17.36	76.2	435.6	0.1225	0.0721	0.05929	0.07404	0.2015	0.05875
14.06	17.18	89.75	609.1	0.08045	0.05361	0.02681	0.03251	0.1641	0.05764
14.87	20.21	96.12	680.9	0.09587	0.08345	0.06824	0.04951	0.1487	0.05748
14.27	22.55	93.77	629.8	0.1038	0.1154	0.1463	0.06139	0.1926	0.05982

AdaBoostClassifier

AdaBoostClassifier 的构造函数中有四个常用的参数可以设置：

- algorithm ：这个参数只有 AdaBoostClassifier 有。主要原因是 scikit-learn

实现了两种 Adaboost 分类算法， SAMME 和 SAMME.R。两者的主要区别是弱学习器权重的度量， SAMME.R 使用了概率度量的连续值，迭代一般比 SAMME 快，因此 AdaBoostClassifier 的默认算法 algorithm 的值也是 SAMME.R；

- n_estimators ：弱学习器的最大迭代次数。一般来说 n_estimators 太小，容易欠拟合，n_estimators 太大，又容易过拟合，一般选择一个适中的数值。默认是 50；
- learning_rate ：AdaBoostClassifier 和 AdaBoostRegressor 都有，即每个弱学习器的权重缩减系数 ν，默认为 1.0；
- base_estimator ：弱分类学习器或者弱回归学习器。理论上可以选择任何一个分类或者回归学习器，不过需要支持样本权重。我们常用的一般是 CART 决策树或者神经网络 MLP。和 sklearn 中其他分类器一样，AdaBoostClassifier 类中的 fit 函数用于训练模型，fit 函数有两个向量输入：
- X ：大小为**[样本数量,特征数量]**的 ndarray，存放训练样本；
- Y ：值为整型，大小为**[样本数量]**的 ndarray，存放训练样本的分类标签。

AdaBoostClassifier 类中的 predict 函数用于预测，返回预测标签， predict 函数有一个向量输入：

X ：大小为**[样本数量,特征数量]**的 ndarray，存放预测样本

AdaBoostClassifier 的使用代码如下：

```
1.    ada=AdaBoostClassifier(n_estimators=5,learning_rate=1.0)
2.    ada.fit(train_data,train_label)
3.    predict = ada.predict(test_data)
```

编程要求：在 begin-end 区域内填写 ada_classifier(train_data,train_label,test_data)函数完成癌细胞识别任务，其中：

- train_data：训练样本；
- train_label：训练标签；
- test_data：测试样本。

测试说明：只需返回预测结果即可，程序内部会检测您的代码，预测正确率高于95% 视为过关。

```
#encoding=utf8
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
def ada_classifier(train_data,train_label,test_data):
    '''
    input:train_data(ndarray):训练数据
          train_label(ndarray):训练标签
          test_data(ndarray):测试标签
    output:predict(ndarray):预测结果
    '''
    #********* Begin *********#

    #********* End *********#
    return predict
```

四、实验报告要求

参见实验报告模板

实验十：机器学习-k 均值算法

一、实验目的

1、理解并掌握 k 均值无监督分类算法思想。

2、了解 Mini Batch KMeans 方法对 K-Means 的改进，并实现两种算法的可视化展示。

二、实验内容

1、补全一个实现 k 近邻分类方法的小程序。

2、实现 K-Means 与 Mini Batch KMeans 方法并进行可视化展示。

三、实验内容与步骤

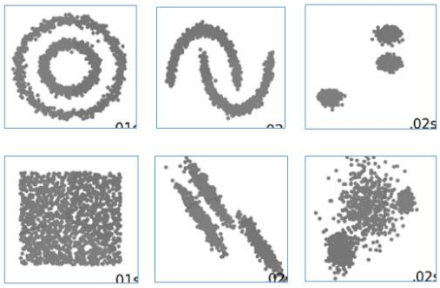
1、数据集准备：6 种不同的聚类数据集

准备好六种不同的聚类数据集，代码如下：

```
from sklearn.cluster import MiniBatchKMeans
from sklearn.cluster import KMeans
import numpy as np

X = np.array([[1,2],[1,4],[1,0],
              [4,2],[4,0],[4,4],
             [4,5],[0,1],[2,2],
             [3,2],[5,5],[1,-1]])
n = int(input())
if n== 0:
    #MiniBatchKMeans 模块
    #********** Begin **********#


    #********** End **********#

else:
#KMeans 模块
    #********** Begin **********#

    #********** End **********#

#输出所有点的类别、两类的中心点并预测[0,0],[4,4]的类别
```

```
    #********** Begin **********#

    #********** End **********#
```

将会得到如下图所示的六类数据集：



2、设置聚类参数。设置聚类参数代码如下：

```
    default_base = {'quantile': .3,
                    'eps': .3,
                    'damping': .9,
                    'preference': -200,
                    'n_neighbors': 10,
                    'n_clusters': 3}
    datasets = [
        (noisy_circles, {'damping': .77, 'preference': -240,
                         'quantile': .2, 'n_clusters': 2}),
        (noisy_moons, {'damping': .75, 'preference': -220, 'n_clusters': 2}),
        (varied, {'eps': .18, 'n_neighbors': 2}),
        (aniso, {'eps': .15, 'n_neighbors': 2}),
        (blobs, {}),
        (no_structure, {})]
```

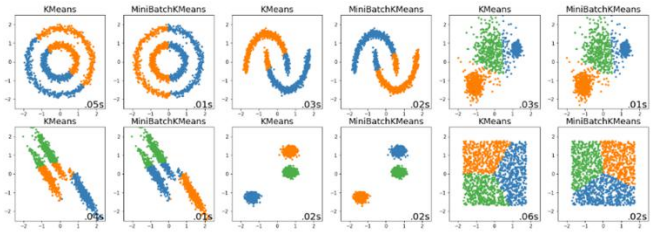
3、创建聚类对象 & 应用聚类方法。创建聚类对象代码如下：

```
kmeans = cluster.KMeans(n_clusters=params['n_clusters'])
two_means = cluster.MiniBatchKMeans(n_clusters=params['n_clusters'])
```

应用聚类方法代码如下：

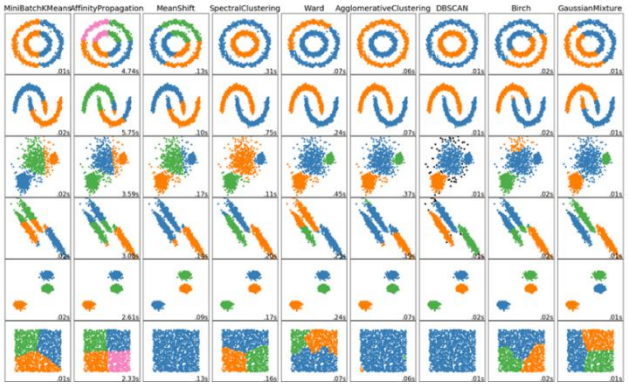
```
    for name, algorithm in clustering_algorithms:
            t0 = time.time() #start time
            algorithm.fit(X) #clustering
            t1 = time.time() #end time
            y_pred = algorithm.predict(X)
```

聚类结果展示



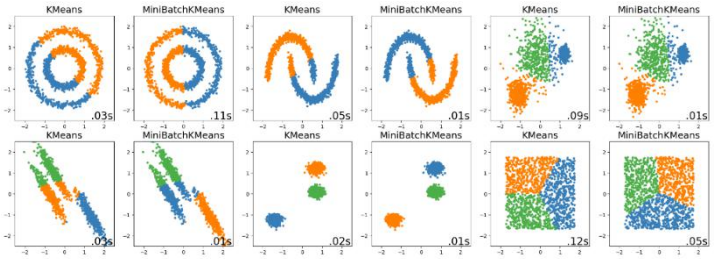
4、其他聚类方法

如谱聚类、DBSCAN、均值移动和自底向上的聚类等方法也需要了解。部分聚类实验结果可视化展示如下：



编程要求：请仔细阅读右侧代码，结合相关知识，在 Begin-End 区域内进行代码补充，实现 K-Means 与 Mini Batch KMeans 方法并进行可视化展示。除补充完整代码之外，还应逐行理解右边示例代码，学习各函数及参数的用法，并举一反三，运用到其他的分类问题之中。

测试说明：平台会对你编写的代码进行测试。图片预期输出结果为：



四、实验报告要求

参见实验报告模板

实验十一：机器学习- PCA

一、实验目的

1、理解并掌握 PCA 基本原理

2、了解 sklearn 中的 PCA 接口对数据进行降维的处理过程。

二、实验内容与步骤

1、补充 python 代码，完成 PCA 函数，实现降维功能。

2、调用 sklearn 中的 PCA 接口来对数据继续进行降维，并使用 sklearn 中提供的分类器接口（可任意挑选分类器）对癌细胞数据进行分类。

三、实验步骤

1、补充 python 代码，完成 PCA 函数，实现降维功能。

PCA 的算法流程：

PCA 在降维时，需要指定将维度降至多少维，假设降至 k 维，则 PCA 的算法流程如下：

1. demean；
2. 计算数据的协方差矩阵；
3. 计算协方差矩阵的特征值与特征向量；
4. 按照特征值，将特征向量从大到小进行排序；
5. 选取前 k 个特征向量作为转换矩阵；
6. demean 后的数据与转换矩阵做矩阵乘法获得降维后的数据。

因此，PCA 算法伪代码如下：

```
#假设数据集为 D，PCA 后的特征数量为 k
def pca(D, k):
    after_demean=demean(D)
    计算 after_demean 的协方差矩阵 cov
    value, vector = eig(cov)
    根据特征值 value 将特征向量 vector 降序排序
    筛选出前 k 个特征向量组成映射矩阵 P
    after_demean 和 P 做矩阵乘法得到 result
    return result
```

编程要求：

在 begin-end 之间填写 pca(data, k)函数，实现 PCA 算法，要求返回降维后的数据。

注意：为了顺利评测，计算协方差矩阵时请使用 NumPy 提供的 cov 函数。

```
import numpy as np

def pca(data, k):
    '''
    对 data 进行 PCA，并将结果返回
    :param data:
    :param k:
    :return: 降维后的数据
    '''

            #********* Begin *********#

            #********* End *********#
```

2、调用 sklearn 中的 PCA 接口来对数据继续进行降维，并使用 sklearn 中提供的分类器接口（可任意挑选分类器）对癌细胞数据进行分类。

乳腺癌数据集，其实例数量是 569，实例中包括诊断类和属性，帮助预测的属性一共 30 个，各属性包括 radius 半径（从中心到边缘上点的距离的平均值），texture 纹理（灰度值的标准偏差）等等，类包括：WDBC-Malignant 恶性和 WDBC-Benign 良性。用数据集的 80% 作为训练集，数据集的 20% 作为测试集，训练集和测试集中都包括特征和诊断类。

sklearn 中已经提供了乳腺癌数据集的相关接口，想要使用该数据集可以使用如下代码：

```
    from sklearn import datasets
    #加载乳腺癌数据集
    cancer = datasets.load_breast_cancer()
    #X 表示特征，y 表示标签
    X = cancer.data
    y = cancer.target
```

数据集中部分数据与标签如下图所示（其中 0 表示良性，1 表示恶性）：

	0	1	2	3	4	5
0	17.99	10.38	122.8	1001	0.1184	0.2776
1	20.57	17.77	132.9	1326	0.08474	0.07864
2	19.69	21.25	130	1203	0.1096	0.1599
3	11.42	20.38	77.58	386.1	0.1425	0.2839
4	20.29	14.34	135.1	1297	0.1003	0.1328
5	12.45	15.7	82.57	477.1	0.1278	0.17
6	18.25	19.98	119.6	1040	0.09463	0.109
7	13.71	20.83	90.2	577.9	0.1189	0.1645
8	13	21.82	87.5	519.8	0.1273	0.1932
9	12.46	24.04	83.97	475.9	0.1186	0.2396

54	0
55	1
56	0
57	0

PCA 的构造函数中有一个常用的参数可以设置：

- n_components ：表示想要将数据降维至 n_components 个维度。

  PCA 类中有三个常用的函数分别为： fit 函数用于训练 PCA 模型；transform 函数用于将数据转换成降维后的数据，当模型训练好后，对于新输入的数据，也可以用 transform 方法来降维；fit_transform 函数用于使用数据训练 PCA 模型，同时返回降维后的数据。

其中 fit 函数中的参数:

- X ：大小为[样本数量,特征数量]的 ndarray ，存放训练样本。

  transform 函数中的参数：

- X ：大小为[样本数量,特征数量]的 ndarray ，存放训练样本。

  fit_transform 函数中的参数：

- X ：大小为[样本数量,特征数量]的 ndarray ，存放训练样本。

PCA 的使用代码如下：

```
from sklearn.decomposition import PCA

#构造一个将维度降至 11 维的 PCA 对象
pca = PCA(n_components=11)
#对数据 X 进行降维，并将降维后的数据保存至 newX
newX = pca.fit_transform(X)
```

编程要求：

在 begin-end 之间填写 cancer_predict(train_sample, train_label, test_sample) 函数实现降维并对癌细胞进行分类的功能。

```
from sklearn.decomposition import PCA

def cancer_predict(train_sample, train_label, test_sample):
    '''
    使用 PCA 降维，并进行分类，最后将分类结果返回
    :param train_sample:训练样本，类型为 ndarray
    :param train_label:训练标签，类型为 ndarray
    :param test_sample:测试样本，类型为 ndarray
    :return: 分类结果
    '''
    #********* Begin *********#

    #********* End *********#
```

四、实验报告要求

参见实验报告模板

实验十二：神经网络

一、实验目的

1、理解并掌握神经网络基本要素：激活函数、反向传播、交叉熵

2、理解神经网络反向传播过程

3、了解利用 sklearn 构建神经网络过程。

4、学会使用 pytorch 搭建出卷积神经网络

二、实验内容

1、用 Python 实现常用激活函数。

2、用 sklearn 构建神经网络模型，并通过鸢尾花数据集中鸢尾花的 4 种属性与种类对神经网络模型进行训练。我们会调用你训练好的神经网络模型，来对未知的鸢尾花进行分类。

3、使用 pytorch 搭建出卷积神经网络并对手写数字进行识别

二、实验步骤

1、用 Python 实现常用激活函数。

神经网络是由一个个神经元也就是感知机组成，感知机数学模型如下：

$$f(x) = sign(w_1x_1 + w_2x_2 + ... + w_nx_n + b)$$

$$sign = egin\{cases\}, -1\&x < 0, +1\&xgeq0, end\{cases\}$$

其中的 sign 函数被称为阶跃函数，是用来引入非线性因素的。而在神经网络里，称其为激活函数。

如果不用激活函数，每一层输出都是上层输入的线性函数，无论神经网络有多少层，输出都是输入的线性组合，这种情况就是最原始的感知机。只能对线性数据进行分类；如果使用的话，激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到众多的非线性模型中，对非线性数据进行分类：

神经网络中经常使用的一个激活函数就是 sigmoid 函数，函数公式如下：

$$sign = egin\{cases\}, -1\&x < 0, +1\&xgeq0, end\{cases\}$$

现在常使用 relu 激活函数，函数公式如下：

$$relu(x) = egin\{cases\}, 0\&x < 0, x\&xgeq0, end\{cases\}$$

relu 函数在 x 大于等于 0 时，输出的是 x 本身，函数变化率恒为 1，这样就避免了梯度消失的情况。当 x 小于 0 时，输出为 0，即神经元不被激活，这样也符合人脑内并不是所有神经元同时被激活的情况。并且，relu 函数计算成本非常低，所以在神经网络过深时常使用 relu 函数作为激活函数。

```
#encoding=utf8

def relu(x):
    '''
    x:负无穷到正无穷的实数
    '''
    #********* Begin *********#

    #********* End *********#
```

2、用 sklearn 构建神经网络模型，并通过鸢尾花数据集中鸢尾花的 4 种属性与种类对神经网络模型进行训练。我们会调用你训练好的神经网络模型，来对未知的鸢尾花进行分类。

鸢尾花数据集是一类多重变量分析的数据集。通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于(Setosa，Versicolour，Virginica)三个种类中的哪一类。

想要使用该数据集可以使用如下代码：

```
    #获取训练数据
    train_data = pd.read_csv('./step2/train_data.csv')
    #获取训练标签
    train_label = pd.read_csv('./step2/train_label.csv')
    train_label = train_label['target']
    #获取测试数据
    test_data = pd.read_csv('./step2/test_data.csv')
```

数据集中部分数据与标签如下图所示：

sepal leng	sepal widt	petal leng	petal width
5.1	3.5	1.4	0.2
4.9	3	1.4	0.2
5.7	2.8	4.1	1.3
6.2	3.4	5.4	2.3
5.1	2.5	3	1.1
7	3.2	4.7	1.4
6.1	2.6	5.6	1.4
7.6	3	6.6	2.1
5.2	4.1	1.5	0.1
6.2	2.2	4.5	1.5
7.3	2.9	6.3	1.8

target
0
0
1
2
1
1
2
2

神经网络的训练方法跟逻辑回归相似，也是使用梯度下降算法来更新模型的参数，既然要使用梯度下降算法，就要知道损失函数对参数的梯度。反向传播算法能够快速的计算这些梯度。反向传播算法一共分为两部分：前向传播与反向传播。

前向传播指的是数据 x 从神经网络输入层，与当层的权重相乘，再加上当层的偏置，所得到的值经过激活函数激活后，再输入到下一层。最后，在输出层所得到的值经过 softmax 函数转化为网络对数据的预测。其中，输出层的 Z 值要经过 softmax 函数，转化为预测值。前向传播的主要目的就是得到预测值，再算出交叉熵损失函数。

交叉熵（cross entropy）是深度学习中常用的一个概念，一般用来求目标与预测值之间的差距。机器学习是用网络训练出来的分布 q 来表示真实分布 p，此时当两者的交叉熵越小时，模型训练的结果越接近样本的真实分布，这也是交叉熵被用来作为损失函数的原因。

通过前向传播能够求出损失函数，而反向传播就是利用前向传播得到的损失函数对参数求梯度，即每个参数的偏导。

之所以称为反向传播，是因为我们在利用链式法则的时候对各个参数求偏导的传播顺序跟前向传播相反，如我们要求 loss 对w_1的偏导，则要先求出 loss 对 a

的偏导，再求出 a 对z^2的偏导，再求出z^2对a^1的偏导，再求出a^1对z^1的偏导，再求出z^1对w^1的偏导，然后全部相乘就得到 loss 对w^1的偏导了。公式如下：

$$\frac{partialloss}{paartialw[1]}=\frac{partialloss}{partiala[2]}\cdot\frac{partiala[2]}{partialz[2]}\cdot\frac{partialz[2}{partiala[1]}\cdot\frac{partiala[1]}{partialz[1]}\cdot\frac{partialz[1]}{paartialw[1]}$$

所以反向传播的目的就是求出损失函数对各个参数的梯度。最后我们就可以用梯度下降算法来训练我们的神经网络模型了。

sklearn 中的神经网络

MLPClassifier 的构造函数中有四个常用的参数可以设置：

- solver：MLP 的求解方法 lbfs 在小数据上表现较好，adam 较为鲁棒，sgd 在参数调整较优时会有最佳表现（分类效果与迭代次数）；sgd 标识随机梯度下降；
- alpha：正则项系数，默认为 L2 正则化，具体参数需要调整；
- hidden_layer_sizes：hidden_layer_sizes=(3, 2)设置隐藏层 size 为 2 层隐藏层，第一层 3 个神经元，第二层 2 个神经元。
- max_iter：最大训练轮数。

  和 sklearn 中其他分类器一样，MLPClassifier 类中的 fit 函数用于训练模型，fit 函数有两个向量输入：
- X：大小为**[样本数量,特征数量]**的 ndarray，存放训练样本；
- Y：值为整型，大小为**[样本数量]**的 ndarray，存放训练样本的分类标签。

  LPClassifier 类中的 predict 函数用于预测，返回预测标签，predict 函数有一个向量输入：
- X：大小为**[样本数量,特征数量]**的 ndarray，存放预测样本。

MLPClassifier 的使用代码如下：

```
1.    from sklearn.neural_network import MLPClassifier
2.    mlp = MLPClassifier(solver='lbfgs',max_iter =10,
3.              alpha=1e-5,hidden_layer_sizes=(3,2))
4.    mlp.fit(X_train, Y_train)
5.    result = mlp.predict(X_test)
```

编程要求：

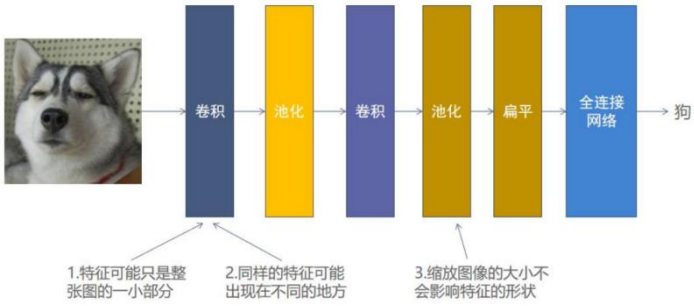
使用 sklearn 构建神经网络模型，利用训练集数据与训练标签对模型进行训练，然后使用训练好的模型对测试集数据进行预测，并将预测结果保存到./step2/result.csv 中。保存格式如下：

result
0
1
1
1
0

```
#encoding=utf8
import os
if os.path.exists('./step2/result.csv'):
    os.remove('./step2/result.csv')
#********* Begin *********#

#********* End *********#
```

3、使用 pytorch 搭建出卷积神经网络并对手写数字进行识别。

卷积神经网络是一种具有局部连接、权重共享等特性的深层前馈神经网络。将卷积，池化，全连接网络进行合理的组合，就能构建出属于自己的神经网络来识别图像中是猫还是狗。通常来说卷积，池化可以多叠加几层用来提取特征，然后接上一个全连接网络来进行分类。大致结构如下：



pytorch 构建卷积神经网络项目流程

数据集介绍与加载数据

本次使用数据集为 mnist 手写数字数据集，简单来讲就是如下的东西：

数据集分为训练集与测试集，训练集中一共有 60000 张图片，测试集中一共有 10000 张图片，每张图片大小为 28X28x1。图片标签为对应的数字，如 8 对应的 label 为 8，若使用 onehot 编码则对应的 label 为：[0,0,0,0,0,0,0,0,1,0]。

为节约计算时间，我们取训练集中的 6000 张图片用来训练，测试集中的 600 张进行测试。使用 pytorch 加载数据集方法如下：

```
#加载数据
import torchvision
train_data = torchvision.datasets.MNIST(
    root='./step3/mnist/',
    train=True,                                     # this is training data
    transform=torchvision.transforms.ToTensor(),    # Converts a
PIL.Image or numpy.ndarray to
    download=False,
)
#取 6000 个样本为训练集
train_data_tiny = []
for i in range(6000):
    train_data_tiny.append(train_data[i])
train_data = train_data_tiny
```

构建模型

加载好数据集，就需要构建卷积神经网络模型：

```
#构建卷积神经网络模型
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(         # input shape (1, 28, 28)
            nn.Conv2d(
                in_channels=1,              # input height
                out_channels=16,            # n_filters
                kernel_size=5,              # filter size
                stride=1,                   # filter movement/step
```

```
                padding=2,                  # if want same width and length of this image after con2d, padding=(kernel_size-1)/2 if stride=1
            ),                              # output shape (16, 28, 28)
            nn.ReLU(),                      # activation
            nn.MaxPool2d(kernel_size=2),    # choose max value in 2x2 area, output shape (16, 14, 14)
        )
        self.conv2 = nn.Sequential(         # input shape (16, 14, 14)
            nn.Conv2d(16, 32, 5, 1, 2),     # output shape (32, 14, 14)
            nn.ReLU(),                      # activation
            nn.MaxPool2d(2),                # output shape (32, 7, 7)
        )
        self.out = nn.Linear(32 * 7 * 7, 10)   # fully connected layer, output 10 classes
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)           # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        output = self.out(x)
        return output
cnn = CNN()
```

需要指出的几个地方：

```
1.  class CNN 需要继承 Module
2.  需要调用父类的构造方法：super(CNN, self).__init__()
3.  在 Pytorch 中激活函数 Relu 也算是一层 layer
4.  需要实现 forward()方法，用于网络的前向传播，而反向传播只需要调用 Variable.backward()即可。
```

定义好模型后还要构建优化器与损失函数：

torch.optim 是一个实现了各种优化算法的库。使用方法如下：

```
#SGD 表示使用随机梯度下降方法，lr 为学习率，momentum 为动量项系数
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01, momentum=0.9)
#交叉熵损失函数
loss_func = nn.CrossEntropyLoss()
```

训练模型

在定义好模型后，就可以根据反向传播计算出来的梯度，对模型参数进行更新，在 pytorch 中实现部分代码如下：

```
#将梯度清零
optimizer.zero_grad()
#对损失函数进行反向传播
loss.backward()
#训练
optimizer.step()
```

保存模型

在 pytorch 中使用 torch.save 保存模型，有两种方法，第一种：

保存整个模型和参数，方法如下：

torch.save(model, PATH)

第二种为官方推荐，只保存模型的参数，方法如下：

torch.save(model.state_dict(), PATH)

加载模型

对应两种保存模型的方法，加载模型也有两种方法，第一种如下：

model = torch.load(PATH)

第二种：

```
#CNN()为你搭建的模型
model = CNN()
model.load_state_dict(torch.load(PATH))
```

如果要对加载的模型进行测试，需将模型切换为验证模式

model.eval()

```
#mini_batch
train_loader = Data.DataLoader(dataset=train_data, batch_size=64, shuffle=True)
```

阅读代码中注释部分要求，补充实现 begin-end 间代码，并运行测试结果

```
#encoding=utf8
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.utils.data as Data
import torchvision
import os
```

```
    if os.path.exists('./step3/cnn.pkl'):
        os.remove('./step3/cnn.pkl')

    #加载数据
    train_data = torchvision.datasets.MNIST(
        root='./step3/mnist/',
        train=True,                                     # this is training data
        transform=torchvision.transforms.ToTensor(),    # Converts a
PIL.Image or numpy.ndarray to
        download=False,
    )
    #取 6000 个样本为训练集
    train_data_tiny = []

    for i in range(6000):
        train_data_tiny.append(train_data[i])

    train_data = train_data_tiny

    #********* Begin *********#


    #********* End *********#
    #保存模型
    torch.save(cnn.state_dict(), './step3/cnn.pkl')
```

四、实验报告要求

参见实验报告模板

实验十三：深度学习-CNN

一、实验目的

1、理解经典的卷积神经网络模型： LeNet 模型和 AlexNet 模型。

2、掌握基于 Pytorch 的经典的卷积神经网络模型的构建方法。

二、实验内容

1、编写基于 Pytorch 的 LeNet 模型。

2、编写基于 Pytorch 的 AlexNet 模型。

二、实验步骤

1、简单的卷积网络的搭建-LeNet 模型

LeNet 模型是一个早期用来识别手写数字图像的卷积神经网络。LeNet 的网络结构如下图所示：

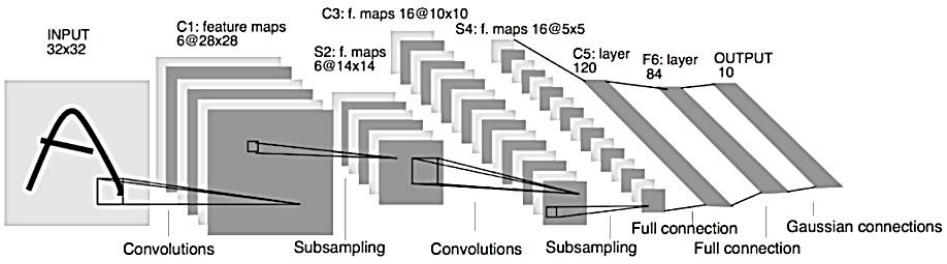


图 1 LeNet 模型结构图

在图中，INPUT 表示输入层，定义的输入图像大小为 32*32。

在 LeNet 的第一层中定义了 6 通道的卷积核，这里没有涉及填充和改变默认步幅，所以已知输入为 32*32，输出为 28*28，根据公式输入大小−卷积核大小+1=输出大小，可以得出卷积核的形状 5*5，为对于每一个卷积核可以训练的参数为 5*5+1(其中 1 为偏置项)，因此这一层一共可以训练的参数为(5*5+1)*6。第二层为池化层，输入为 28*28 的矩阵，池化层形状为 2*2，步幅为 2，原论文采样的方式为四个输入值相加，然后乘以一个可训练的参数，再加上偏置项，通过 Sigmoid 函数进行激活。第三层属于卷积层，输入为 S2 层中的 6 个通道的输出，卷积核大小为 5*5，但是输出通道数增加到 16，所以可以训练的参数为(5*5+1)*16。第四层是池化层，形状为 2*2，步幅为 2，原论文采样的方式为四个输入值相加，然后乘以一个可训练的参数，再加上偏置项，通过 Sigmoid 函数

进行激活。接下来三层是全连接层（ Fully Connected，简称 FC ），首先需要计算出上层的输出为 16*5*5=400，所以接下来三层的参数矩阵应为 400*120，120*84，84*10。层之间的激活函数也采用的是 Sigmoid 函数。

Pytorch 中模型的搭建

Module 类是 nn 模块里提供的一个模型构造类，是所有神经网络模块的基类，可以继承它来定义我们想要的模型。下面展示一个简单的卷积网络的实例，可以在待会的实战中采用相同的方法来搭建：

```
1.    import torch
2.    from torch import nn     # 导入 nn 模块
3.    class Sample(nn.Module):
4.        def __init__(self):
5.            super(Sample, self).__init__()
6.            self.conv = nn.Sequential(
7.                nn.Conv2d(1, 1, 5),     # 卷积层
8.                nn.Sigmoid(),     # 激活函数
9.                nn.MaxPool2d(2, 2),     # 最大池化层
10.           )
11.           self.fc = nn.Sequential(
12.               nn.Linear(14*14, 10),     # 全连接层
13.               nn.Sigmoid(),     # 激活函数
14.           )
15.
16.       def forward(self, img):     # 定义前向计算
17.           feature = self.conv(img)     # 卷积层
18.           output = self.fc(feature.view(img.shape[0], -1))     # 全连接层
19.           return output
```

在样例中，我们定义了一层卷积，一层池化，一层全连接层，也使用了激活函数，可以说在最简单的 LeNet 中需要用到的的组件都有了使用。

将 Sample 的实例对象打印出来就可以清楚地看到上面代码中搭建出的模型的整体结构：

```
1.    Sample(
2.      (conv): Sequential(
3.        (0): Conv2d(1, 1, kernel_size=(5, 5), stride=(1, 1))
4.        (1): Sigmoid()
5.        (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```
    )
    (fc): Sequential(
      (0): Linear(in_features=196, out_features=10, bias=True)
      (1): Sigmoid()
    )
  )
```

阅读代码中注释部分要求，补充实现 begin-end 间代码，并运行测试结果。

```
mport torch
from torch import nn
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        '''
        这里搭建卷积层，需要按顺序定义卷积层、
        激活函数、最大池化层、卷积层、激活函数、最大池化层，
        具体形状见测试说明
        '''
        self.conv = nn.Sequential(
            ########## Begin ##########


            ########## End ##########
        )
        '''
        这里搭建全连接层，需要按顺序定义全连接层、
        激活函数、全连接层、激活函数、全连接层，
        具体形状见测试说明
        '''
        self.fc = nn.Sequential(
            ########## Begin ##########

            ########## End ##########
        )

    def forward(self, img):
        '''
        这里需要定义前向计算
        '''
        ########## Begin ##########

        ########## End ##########
```

2、编写基于 Pytorch 的 AlexNet 模型。

AlexNet 模型的网络结构如下图所示：

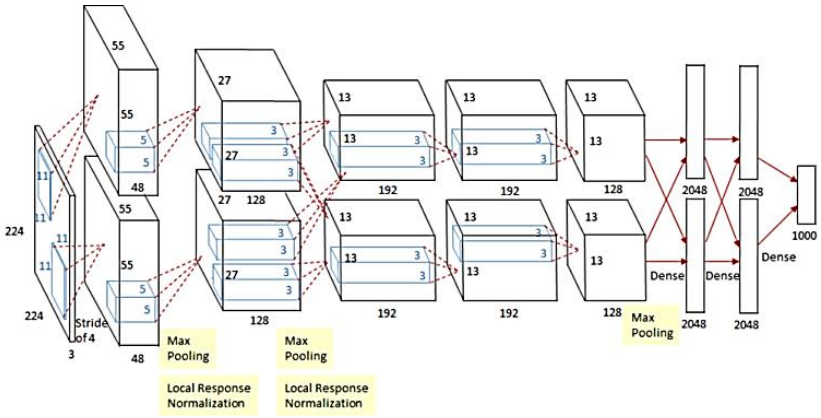


图 1 AlexNet 模型结构示意图

AlexNet 模型一共有八层，包含五个卷积层和三个全连接层，对于每一个卷积层，均包含了 ReLU 激活函数和局部响应归一化处理，接着进行了池化操作。

在模型设计时，通道数需要将图中上下两个部分的通道数相加来得到整个模型单层的通道数，例如第一层的通道数就是 96=48+48。从图中可以发现，输入的图像大小为 224*224，但是考虑到图像是由 RGB 组成的三通道，因此输入图像的形状是 3*224*224。(在 AlexNet 模型的实际处理过程中会通过预处理，图像的输入是 3*227*227)。

AlexNet 第一层中的卷积窗口形状是 96*11*11。第二层中的卷积窗口形状减小到 5*5 但是通道数从 96 增加值 256，之后采用 3*3 的卷积核，但是通道进一步增加至 384，然后继续一层 384*384*3*3 的卷积层，之后是 384*256*3*3 的最后一层卷积层。此外，第一、第二和第五个卷积层之后都使用了窗口形状为 3*3、步幅为 2 的最大池化层。接下来是两个输出个数为 4096 的全连接层。

同时 AlexNet 通过丢弃法来控制全连接层的模型复杂度，同时提高了模型的泛化能力。nn.Dropout()就是 Pytorch 中用于丢弃法的模块，使用方法为 nn.Dropout(0.5)。

阅读代码中注释部分要求，补充实现 begin-end 间代码，并运行测试结果。

```
import torch
from torch import nn
class AlexNet(nn.Module):
    def __init__(self):
        super(AlexNet, self).__init__()
```

```
            '''
            这里搭建卷积层，需要按顺序定义卷积层、
            激活函数、最大池化层、卷积层、激活函数、
            最大池化层、卷积层、激活函数、卷积层、
            激活函数、卷积层、激活函数、最大池化层，
            具体形状见测试说明
            '''
            self.conv = nn.Sequential(
                ########## Begin ##########

                ########## End ##########
            )
            '''
            这里搭建全连接层，需要按顺序定义
            全连接层、激活函数、丢弃法、
            全连接层、激活函数、丢弃法、全连接层，
            具体形状见测试说明
            '''
            self.fc = nn.Sequential(
                ########## Begin ##########

                ########## End ##########
            )

        def forward(self, img):
            '''
            这里需要定义前向计算
            '''
            ########## Begin ##########

            ########## End ##########
```

四、实验报告要求

参见实验报告模板

实验十四：深度学习-RNN

一、实验目的

1、学习 RNN 循环神经网络的基本概念并构建单个 RNNCell。

2、了解 LSTM 的核心思想，创建 LSTM 网络的一个 LSTMCell 。

3、学习如何一次执行得到 RNNCell 多步调用 call 方法得到的结果，掌握构建堆叠 RNNCell 的方法。

二、实验内容

1、学习 RNN 循环神经网络的基本概念并构建单个 RNNCell。

2、了解 LSTM 的核心思想，根据教程指导创建 LSTM 网络的一个 LSTMCell 。

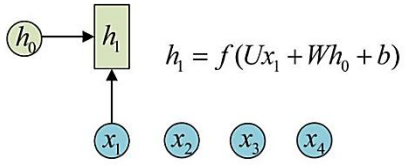
3、学习如何一次执行得到 RNNCell 多步调用 call 方法得到的结果，其次掌握构建堆叠 RNNCell 的方法。

三、实验步骤

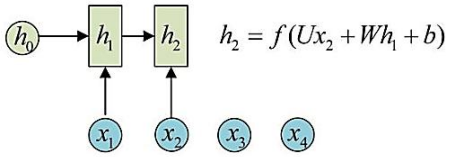
1、学习 RNN 循环神经网络的基本概念并构建单个 RNNCell。

如果要学习 TensorFlow 中的 RNN ，首先需要了解 RNNCell ，它是 TensorFlow 中实现 RNN 的基本单元，每个 RNNCell 都有一个 call 方法，使用方式是： (output, next_state) = call(input, state) 。

借助图片来说可能更容易理解。假设我们有一个初始状态 h0 ，还有输入 x1 ，调用 call(x1, h0) 后就可以得到 (output1, h1) ：

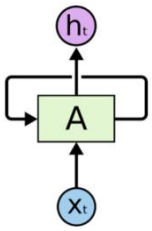


再调用一次 call(x2, h1) 就可以得到 (output2, h2) ：



也就是说，每调用一次 RNNCell 的 call 方法，就相当于在时间上“推进

了一步”,隐含层的 h1 代表了第一个 RNNCell 的状态,这个状态记录了 X1 输入后 RNNCell 积攒的有关先前输入序列的知识，当把 h1 传给第二个 RNNCell ,网络就完成了“经验”的传递工作,当我们把许许多多这样的 RNNCell 联结，就可以聚合成为下图所示的网络结构：



于是有关时间序列的经验的传递再结合当前的输入就可以构成具有“循环”结构的 RNN (循环神经网络)。

单个 RNNCell 的构建

方法 tf.nn.rnn_cell.BasicRNNCell() 可以构造一个 BasicRNNCell 类的实例即一个最基本的 RNNCell ，这个方法有一个参数 num_units 需要设置一下

有了 RNNCell 后，需要 call 方法看看这个网络能给什么样的输出，不过 BasicRNNCell 类里的这一调用方法叫做 call()。完整的过程如下：

cell=tf.nn.rnn_cell.BasicRNNCell(num_units=5)

新建一个输入值 x1。

x1=tf.placeholder(tf.float32,[batch_size,n_inputs])

batch_size 代表批量输入值的大小, n_inputs 代表单个输入值的维度

构建 BasicRNNCell，含有 n_units 个神经元

cell=tf.nn.rnn_cell.BasicRNNCell(num_units=n_units)

将初始状态初始化为全零

h0=cell.zero_state(batch_size=batch_size,dtype=tf.float32)

output ， h1 分别代表了这个 RNNCell 的输出和当前状态。

output,h1=cell.__call__(x1,h0)

打印当前状态

print(h1)

编程要求：根据提示，在右侧编辑器的 begin-end 间补充代码，创建一个包含 a 个神经元，可以接受一个 shape=[b , c] ,类型为 float32 的张量作为输入的

BasicRNNCell，并打印一次输入后该 BasicRNNCell 的输出。

测试说明：平台会对你编写的代码进行测试：

测试输入： 5 ， 4 ， 3 ；

预期输出：

```
5
 Tensor("basic_rnn_cell/Tanh:0", shape=(4, 5), dtype=float32)
```

测试输入： 151 ， 12 ， 122 ；

预期输出：

```
151
 Tensor("basic_rnn_cell/Tanh:0", shape=(12, 151), dtype=float32)
```

2、了解 LSTM 的核心思想，根据教程指导创建 LSTM 网络的一个 LSTMCell 。

Long Short Term 网络，一般叫做 LSTM，是一种特殊的 RNN 类型，可以学习长期依赖信息。LSTM 通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是 LSTM 的默认行为。

对于 BasicLSTMCell 类，LSTM 网络单元的情况有些许不同，因 为 LSTM 可以看做有两个隐含状态 h 层和 c 层，对应的隐含层就是一个 Tuple，每个都是 (batch_size, state_size) 的形状。下面介绍如何创建一个 LSTMCell：

```
#首先创建一个输入张量，batch_size,n_inputs 自行指定
inputs = tf.placeholder(np.float32, [batch_size,n_inputs])
```

```
#接下来正式创建一个包含 n_units 个神经元的 BasicLSTMCell 实例
lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(num_units=n_units)
```

```
# 通过 zero_state 得到一个全 0 的初始状态
h0 = lstm_cell.zero_state(batch_size=batch_size,dtype=np.float32)
```

```
#调用一次__call__()方法，得到输出和新的隐含状态
output, h1 = lstm_cell.__call__(inputs, h0)
```

```
print(h1.h)  #打印新的 h 层

print(h1.c)  #打印新的 c 层
```

编程要求：

创建一个包含 a 个神经元，可以接受一个 shape=[b , c] ,类型为 float32 的张量作为输入的 BasicLSTMCell ，并打印一次输入后该 BasicRNNCell 的隐含状态的 h ， c 层。

```
# -*- coding: utf-8 -*-
import tensorflow as tf

# 参数 a 是 BasicRNNCell 所含的神经元数，参数 b 是 batch_size，参数 c 是单个 input 的维数，shape = [ b , c ]
def creatRNNCell(a,b,c):
# 请在此添加代码 完成本关任务
# ********** Begin *********#

# ********** End **********#
```

3、学习如何一次执行得到 RNNCell 多步调用 call 方法得到的结果，其次掌握构建堆叠 RNNCell 的方法。

基础的 RNNCell 有一个很明显的问题：对于单个的 RNNCell ，我们使用它的 call 函数进行运算时，只是在序列时间上前进了一步。比如使用 x1 、 h0 得到 h1 ，通过 x2 、 h1 得到 h2 等。这样的话，如果我们的序列长度为 10 ，就要调用 10 次 call 函数，比较麻烦。对此， TensorFlow 提供了一个 tf.nn.dynamic_rnn 函数，使用该函数就相当于调用了 n 次 call 函数。即通过 {h0,x1, x2, …., xn} 直接得 {h1,h2…,hn} 。

具体来说，设我们输入数据的格式为 (batch_size, time_steps, input_size) ，其中 time_steps 表示序列本身的长度，如在 Char RNN 这一开源项目中，长度为 10 的句子对应的 time_steps 就等于 10 。最后的 input_size 就表示输入数据单个序列单个时间维度上固有的长度。另外我们已经定义好了一个 RNNCell ，调用该 RNNCell 的 call 函数 time_steps 次，对应的代码就是：

```
# inputs: shape = (batch_size, time_steps, input_size)

# cell: RNNCell

```

```
5.    # initial_state: shape = (batch_size, cell.state_size)。初始状态。一般可
以取零矩阵
6.
7.    outputs, state = tf.nn.dynamic_rnn(cell, inputs,
initial_state=initial_state)
```

此时，得到的 outputs 就是 time_steps 步里所有的输出。它的形状为 [batch_size, time_steps, cell_state_size] 。 state 是最后一步的隐状态，它的形状为 (batch_size, cell.state_size) 。

学习如何堆叠 RNNCell：MultiRNNCell

很多时候，单层 RNN 的能力有限，我们需要多层的 RNN 。将 x 输入第一层 RNN 的后得到隐层状态 h ，这个隐层状态就相当于第二层 RNN 的输入，第二层 RNN 的隐层状态又相当于第三层 RNN 的输入，以此类推。在 TensorFlow 中，可以使用 tf.nn.rnn_cell.MultiRNNCell 函数对 RNNCell 进行堆叠，相应的示例程序如下：

```
1.    import tensorflow as tf
2.
3.    import numpy as np
4.
5.
6.
7.    # 每调用一次这个函数就返回一个 BasicRNNCell
8.
9.    def get_a_cell():
10.      return tf.nn.rnn_cell.BasicRNNCell(num_units=128)
11.
12.   # 用 tf.nn.rnn_cell MultiRNNCell 创建 3 层 RNN
13.
14.   cell = tf.nn.rnn_cell.MultiRNNCell([get_a_cell() for _ in range(3)]) # 3
层 RNN
15.
16.   # 得到的 cell 实际也是 RNNCell 的子类
17.
18.   # 它的 state_size 是(128, 128, 128)
19.
20.   # (128, 128, 128)并不是 128x128x128 的意思
21.
22.   # 而是表示共有 3 个隐层状态，每个隐层状态的大小为 128
23.
```

```
    print(cell.state_size) # (128, 128, 128)

    # 使用对应的 call 函数

    inputs = tf.placeholder(np.float32, shape=(32, 100)) # 32 是 batch_size, 100 是 input_size

    h0 = cell.zero_state(32, np.float32) # 通过 zero_state 得到一个全 0 的初始状态

    output, h1 = cell.call(inputs, h0)

    print(h1) # 新的隐层状态中含有 3 个 32x128 的向量
```

通过 MultiRNNCell 得到的 cell 并不是什么新鲜事物，它实际也是 RNNCell 的子类，因此也有 call 方法、 state_size 和 output_size 属性。同样可以通过 tf.nn.dynamic_rnn 来一次运行多步。

编程要求

根据前面所学知识，在 begin-end 间尝试构建一个 a 层的 RNN ，每一层由 state_size=b 的 RNNBasicCell 组成，初始化隐层状态为全零，输入一个 batch_size=c,time_steps=d,input_size=e 的输入序列，调用 tf.nn.dynamic_rnn 一次得到多步输入后的结果，并打印出最终的输出 output 的函数 MultiRNNCell_dynamic_call(a,b,c,d,e) 。

```
# -*- coding: utf-8 -*-
import tensorflow as tf
import numpy as np

# 参数 a 是 RNN 的层数，参数 b 是每个 BasicRNNCell 包含的神经元数即 state_size
# 参数 c 是输入序列的批量大小即 batch_size，参数 d 是时间序列的步长即 time_steps，参数 e 是单个输入 input 的维数即 input_size
def MultiRNNCell_dynamic_call(a,b,c,d,e):
# 用 tf.nn.rnn_cell MultiRNNCell 创建 a 层 RNN，并调用 tf.nn.dynamic_rnn
# 请在此添加代码 完成本关任务
    # ********** Begin *********#

    # ********** End **********#
```

四、实验报告要求

参见实验报告模板

实验十五：计算机视觉初窥及 GAN 应用

一、实验目的

1、了解计算机图像的基本原理，并利用开发库 openCV 对图片进行相应的处理。

2、了解生成对抗网络原理，并使用 keras 实现生成对抗网络的图像应用。

二、实验内容

1、利用 OpenCV 对图片进行相应的处理。

2、使用 OpenCV 实现图片边缘检测方法，并对图片进行边缘检测。

3、使用 keras 调用已训练好模型对图片中人物性别进行预测。

4、利用 openCV 对图片进行相应的处理。

5、使用 keras 实现生成对抗网络自动生成二次元头像。

三、实验步骤

1、利用 OpenCV 对图片进行相应的处理。

图片在计算机中以像素矩阵的形式储存，每个像素值在 0 到 255 之间，对于灰度图片，它只有一个通道。相比灰度图片，彩色图片稍微复杂一点，它有 RGB 三个通道，每一个通道对应着一个如上图中的像素矩阵。

OpenCV 是一款功能非常强大的计算机视觉库要对一张图片进行处理，首先得获取它，具体代码如下：

```
1.    import cv2
2.    #加载图片
3.    img = cv2.imread(img_path)
```

其中，cv2.imread() 即为获取图片的方法，其中需要填写的参数 img_path 即图片在计算机中所储存的路径。加载图片后，我们可以改变图片的尺寸，即宽高，使它变成我们需要的大小：

```
1.    import cv2
2.    #将图片尺寸转换成 HxWx3
3.    img = cv2.resize(img,(H,W))
```

其中，cv2.resize() 为改变图片尺寸的方法，参数 img 为需要改变的图片，H 为你需要设定成多高，W 为你需要设定成多宽。

将图片处理后，我们还需要保存它：

```
import cv2
#保存图片图片
cv2.imwrite(save_path, img)
```

其中，cv2.imwrite() 为保存图片的方法，参数 save_path 为你将图片保存的位置，img 为你所需保存的图片。

编程要求

实现改变图片尺寸与保存图片方法。

```
#encoding=utf8
import cv2
def get_img(img_path,save_path):
    '''
    img_path:图片储存路径
    save_path:图片保存路径
    img:处理后的图片
    '''
    #********* Begin *********#


    #********* End *********#
    return img
```

2、使用 OpenCV 实现图片边缘检测方法，并对图片进行边缘检测。

边缘检测是图像处理和计算机视觉中的基本问题，边缘检测的目的是标识数字图像中亮度变化明显的点。图像边缘检测大幅度地减少了数据量，并且剔除了可以认为不相关的信息，保留了图像重要的结构属性。

常用的边缘检测模板有 Laplacian 算子、Roberts 算子、Sobel 算子、 Kirsch 算子和 Prewitt 算子等。在这里简单介绍下 canny 边缘检测，大概分为以下五个步骤：

```
使用高斯滤波器，以平滑图像，滤除噪声。
计算图像中每个像素点的梯度强度和方向。
应用非极大值（Non-Maximum Suppression）抑制，以消除边缘检测带来的杂散响应。
```

```
4.    应用双阈值（Double-Threshold）检测来确定真实的和潜在的边缘。
5.    通过抑制孤立的弱边缘最终完成边缘检测。
```

在 OpenCV 中，已经实现好 canny 边缘检测方法，使用如下：

```
1.    import cv2
2.    #加载灰度图片
3.    img = cv2.imread(img_path,0)
4.    #进行 canny 边缘检测
5.    canny_edg=cv2.Canny(img,50,150)
6.    #保存处理后的图片
7.    cv2.imwrite(save_path,canny_edg)
```

其中 cv2.imread() 为加载图片方法，0 表示加载灰度图片。cv2.Canny() 为 canny 边缘检测方法，img 为所检测图片。最后将处理后的图片保存，save_path 为保存路径。

编程要求

实现边缘检测方法，并对图片进行边缘检测。

```
1.    #encoding=utf8
2.    import cv2
3.    def canny_edg(img_path,save_path):
4.        '''
5.        img_path:图片储存路径
6.        save_path:图片保存路径
7.        img:处理后的图片
8.        '''
9.        #********* Begin *********#
10.
11.
12.       #********* End *********#
13.       return img
```

3、使用 keras 调用已训练好模型对图片中人物性别进行预测。

对于计算机来说，为了识别出图中人物性别，首先得构造一个类似于函数一样的模型。函数它的输入跟输出都应该是数字。对于图片来说，它本身就是以像素矩阵的形式储存在计算机中，输出我们则可以用 0 来表示女性， 1 来表示男性。当然这只是一种选择，在深度学习中我们通常使用另外一种形式 [0,1] 表示女性, [1,0] 表示男性。不管哪种方式，我们都把问题转换成了去求函数 f(.)的表

达式。只要求出正确的表达式，就能对人物进行正确的性别识别。

使用 keras 识别人物性别流程

keras 是一个高层神经网络 API ，使用 keras 实现人物性别识别的第一部就是得获取数据，有了数据才能对模型进行训练。可以直接调用获取数据的方法代码如下：

```
#导入获取数据的方法
from load_face_dataset import get_img_and_label
#加载测试数据
data,label = get_img_and_label(data_path)
```

其中，get_img_and_label() 是获取图像和标签的方法，data_path 是数据存储的路径，data 与 label 是获取的图像与图像相对应的标签。

由于训练模型需要一定的时间，所以已经将模型提前训练好，只需加载模型，就能使用模型来进行预测。代码如下：

```
#导入加载模型方法
from keras.models import load_model
#加载模型
model = load_model(model_path)
```

其中，load_model() 是加载模型的方法，model_path 为模型存储的路径。model 为我们所加载的模型。最后就可以对加载的数据进行预测了：

#计算正确率

_,acc = model.evaluate(data,label,verbose=0)

其中，model.evaluate 为测试模型的方法，data , label 为输入的图像与对应的标签。acc 为模型预测的正确率。

编程要求

实现对人物性别进行预测，并返回正确率的方法。

```
#encoding=utf8
import cv2
def get_img(img_path,save_path):
'''
img_path:图片储存路径
save_path:图片保存路径
```

```
    img:处理后的图片
    '''
    #********* Begin *********#


    #********* End *********#
    return img
```

4、利用 openCV 对图片进行相应的处理。

人脸检测

在真实项目中，有时候还需要将图片中人脸的位置检测出来。人脸检测就是需要找出这个方框的正确位置。跟其它任务一样，要进行人脸检测首先得加载待检测图片：

```
import cv2
#加载图片
img = cv2.imread(img_path,1)
#转换为灰度图
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

其中，cv2.imread() 为加载图片方法，参数 img_path 为待识别图片路径， 1 表示加载 BGR 图片,接下来再使用 cv2.cvtColor() 方法将图片转换为灰度图。img 为待处理图片，cv2.COLOR_BGR2GRAY 表示将 BGR 图转换为灰度图。

然后再加载已经训练好的检测模型：

```
#加载模型
face_cascade = cv2.CascadeClassifier(model_path)
```

cv2.CascadeClassifier()为加载模型方法，model_path 为模型存放路径。

利用训练好的模型，就能检测出人脸的位置，代码如下：

```
#获取识别框坐标
face_rects = face_cascade.detectMultiScale(gray, 1.1, 10)
x, y, w, h = face_rects[0]
```

face_cascade.detectMultiScale() 为人脸检测方法， gray 为待检测灰度图, 1.1 表示检测框按 1.1 的比例放大, 10 表示一个目标至少要被检测到 10 次才算真正的目标。(x,y) 为检测框左上角坐标，w,h 为检测框宽高长度值。

最后，将检测框添加进图像中并保存图像：

```
#将识别框加入图片中
cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 3)
#保存图片
cv2.imwrite(img_save_path,img)
```

img 为目标图片，(x,y) 为检测框左上角坐标，(x+w,y+h) 为右下角坐标。(255,0,0) 表示蓝色，(0,255,0) 表示绿色，(0,0,255) 表示红色，3 表示 BGR 三个通道。

编程要求

实现人脸检测方法。

```
# -*- coding: utf-8 -*-
import cv2
def face_detection(img_path,img_save_path,model_path):
    '''
    img_path:待识别图片路径
    img_save_path:图片保存路径
    model_path:模型所在路径
    '''
    #********* Begin *********#

    #********* End *********#
    return face_rects[0]
```

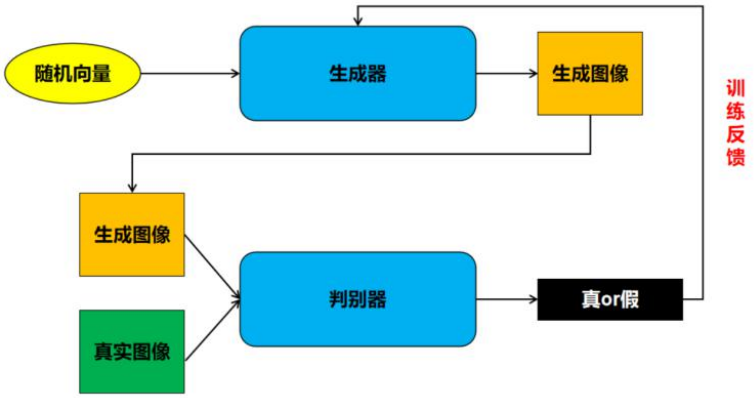
5、使用 keras 实现生成对抗网络自动生成二次元头像。

生成式对抗网络 （GAN，generative adversarial network） 由 Goodfellow 等人于 2014 年提出，它能够迫使生成图像与真实图像在统计上几乎无法区分，从而生成相当逼真的合成图像。

生成对抗网络的基本工作原理，一个生成网络（伪造者），一个判别网络（古董商人），二者训练的目的就是为了打败彼此。

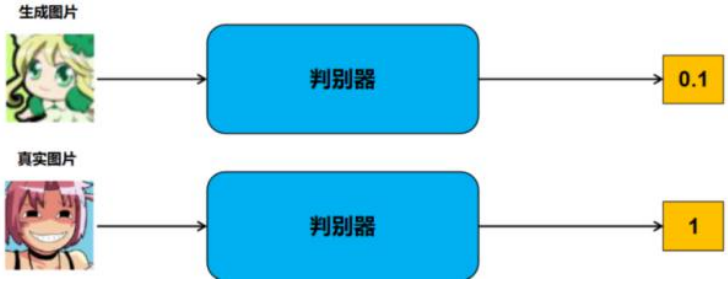
1. 生成网络（generator network）：它以一个随机向量作为输入，并将其解码为一张合成图像。
2. 判别网络（discriminator network）：以一张图像（真实的或合成的均可）作为输入，并预测该图像是来自训练集还是由生成器网络创建。

训练生成器网络的目的是使其能够欺骗判别器网络，因此随着训练的进行，它能够逐渐生成越来越逼真的图像，即看起来与真实图像无法区分的人造图像，以至于判别器网络无法区分二者。与此同时，判别器也在不断适应生成器逐渐提高的能力，为生成图像的真实性设置了很高的标准。流程如下：



利用生成对抗网络生成二次元头像

想要利用生成对抗网络我们首先要获取大量二次元头像，如上图。然后将二次元图像与生成网络生成的图像对判别器进行训练，得到一个能够准确识别真实图片与生成图片的网络。



判别器的输入为图片，输出为数值，数值越大代表图片为真实图片的概率越大。

当判别器训练好之后，我们将冻结判别器的参数，然后再对生成器进行训练，这样的目的是为了使生成器生成的图片越来越像真实图片。然后不断来回的对判别器与生成器进行训练，最后，生成器就可以生成一堆非常逼真的二次元头像了。

这里我们已经训练好模型，你只需要直接调用就可以用来生成二次元头像，代码如下：

```
#加载已训练好的生成模型,./step7/model/g.h5 为模型所在路径
generator = load_model('./step7/model/g.h5')
#将输入向量转换为[1,100]形状
vec = vec.reshape(1,100)
#获取输出图片
img = generator.predict(vec)
#将图片转换为三维
img = img[0]
#将像素转换为 0-255 之间的 int 类型
img = (img+1)*255/2
img = img.astype('uint8')
#rgb 转 bgr
img = cv2.cvtColor(img,cv2.COLOR_RGB2BGR)
#保存图片，./step7/img/generator.jpg 为图片保存路径
cv2.imwrite('./step7/img/generator.jpg',img)
```

编程要求：利用已训练好的生成对抗网络模型生成二次元头像。

测试说明：程序会调用你的方法生成二次元头像，若图片与正确图片差异小于阈值则视为通关。

```
#encoding=utf8
import cv2
from keras.models import load_model


def generator_img(vec):
    '''
```

```
        vec(int):输入向量
        '''
        #********* Begin *********#

        #********* End *********#
        return img
```

四、实验报告要求

参见实验报告模板

实验十六：自然语言理解

一、实验目的

1、编写 Python 代码实现原始数据加载的功能。

2、编写 Python 代码实现 Tokenize 功能。

3、编写 Python 代码实现 padding 功能。

4、编写 Python 代码搭建一个双向 RNN 神经网络

5、编写 Python 代码，实现机器翻译功能。

二、实验内容

1、编写 Python 代码实现原始数据加载的功能。

2、编写 Python 代码实现 Tokenize 功能。

3、编写 Python 代码实现 padding 功能。

4、编写 Python 代码搭建一个双向 RNN 神经网络

5、编写 Python 代码，实现机器翻译功能。

三、实验步骤

1、编写 Python 代码实现原始数据加载的功能。

原始数据非常简单，分为两个文件，一个是存放英语文本的 small_vocab_en.txt，另一个是存放对应的法语翻译的文本文件 small_vocab_en.txt。

每个文件中的每一行都是对应的翻译文本。如下图所示(第一张图是英文语料的部分截图，第二张图是法语语料的部分截图)：

```
new jersey is sometimes quiet during autumn , and it is snowy in april .
the united states is usually chilly during july , and it is usually freezing in november .
california is usually quiet during march , and it is usually hot in june .
the united states is sometimes mild during june , and it is cold in september .
your least liked fruit is the grape , but my least liked is the apple .
his favorite fruit is the orange , but my favorite is the grape .
paris is relaxing during december , but it is usually chilly in july .
new jersey is busy during spring , and it is never hot in march .
our least liked fruit is the lemon , but my least liked is the grape .
the united states is sometimes busy during january , and it is sometimes warm in november .
the lime is her least liked fruit , but the banana is my least liked .
he saw a old yellow truck .
india is rainy during june , and it is sometimes warm in november .
that cat was my most loved animal .
he dislikes grapefruit , limes , and lemons .
her least liked fruit is the lemon , but his least liked is the grapefruit .
california is never cold during february , but it is sometimes freezing in june .
china is usually pleasant during autumn , and it is usually quiet in october .
```

```
new jersey est parfois calme pendant l' automne , et il est neigeux en avril .
les états-unis est généralement froid en juillet , et il gèle habituellement en novembre .
california est généralement calme en mars , et il est généralement chaud en juin .
les états-unis est parfois légère en juin , et il fait froid en septembre .
votre moins aimé fruit est le raisin , mais mon moins aimé est la pomme .
son fruit préféré est l'orange , mais mon préféré est le raisin .
paris est relaxant en décembre , mais il est généralement froid en juillet .
new jersey est occupé au printemps , et il est jamais chaude en mars .
notre fruit est moins aimé le citron , mais mon moins aimé est le raisin .
les états-unis est parfois occupé en janvier , et il est parfois chaud en novembre .
la chaux est son moins aimé des fruits , mais la banane est mon moins aimé.
il a vu un vieux camion jaune .
inde est pluvieux en juin , et il est parfois chaud en novembre .
ce chat était mon animal le plus aimé .
il n'aime pamplemousse , citrons verts et les citrons .
son fruit est moins aimé le citron , mais son moins aimé est le pamplemousse .
californie ne fait jamais froid en février , mais il est parfois le gel en juin .
chine est généralement agréable en automne , et il est généralement calme en octobre .
paris est jamais le gel en novembre , mais il est merveilleux en octobre .
les états-unis est jamais pluvieux en janvier , mais il est parfois doux en octobre .
chine est généralement agréable en novembre , et il est jamais tranquille en octobre .
```

加载原始数据

既然想要实现将英语翻译成法语的功能，那么第一步肯定是加载原始数据。把硬盘中的数据加载到内存当中。使用 Python 读取文件中的数据非常简单，只需使用如下示例代码：

```
with open('data.txt', 'r') as f:
    raw_data = f.read()
```

编程要求

实现 load_data 函数，该函数需要加载 path 所代表的文件中的数据，并将文件中所有的内容按\n 分割，转换成一个列表后返回。

测试说明

平台会对你编写的代码进行测试：

测试输入：

无

预期输出：

加载原始语料成功

```
#coding:utf8
import os


def load_data(path):
'''
```

```
    读取原始语料数据
    :param path: 文件路径
    :return: 句子列表，如['he is a boy.', 'she is a girl']
    '''
    #*********Begin*********#

    #*********End*********#
```

2、编写 Python 代码实现 Tokenize 功能。

对于深度学习和机器学习程序来说，除了数字之外，什么都不认识。也就是说假设现在有一个机器翻译的程序，能将英语翻译成法语，然后我们把英语句子作为输入，输入到程序中。此时程序其实会将句子中的单词、符号等信息转换成数字，再丢给模型去计算。所以将单词转换成数字是实现机器翻译的第一步，而 Tokenize 就是最为常用的一种将单词转换成数字的方式。

Tokenize 看起来高大上，其实就是将每个单词都映射成一个数字而已。例如现在语料库中的句子如下：

```
['The quick brown fox jumps over the lazy dog .', 'By Jove , my quick
study of lexicography won a prize .', 'This is a short sentence .']
```

那么 Tokenize 就是统计所有句子中出现的不同的词，然后将每个单词与一个数字对应起来。如：

```
{'the': 1, 'quick': 2, 'a': 3, 'brown': 4, 'fox': 5, 'jumps': 6,
'over': 7, 'lazy': 8, 'dog': 9, 'by': 10, 'jove': 11, 'my': 12,
'study': 13, 'of': 14, 'lexicography': 15, 'won': 16, 'prize': 17,
'this': 18, 'is': 19, 'short': 20, 'sentence': 21}
```

所以经过 Tokenize 后，语料库中的句子变成了只有数字的列表，如下所示：

```
[[1, 2, 4, 5, 6, 7, 1, 8, 9], [10, 11, 12, 2, 13, 14, 15, 16, 3, 17],
[18, 19, 3, 20, 21]]
```

怎样实现 Tokenize

keras 为我们已经实现好了 Tokenize 功能，我们只需调用接口即可实现 Tokenize 。示例代码如下：

```
from keras.preprocessing.text import Tokenizer
```

```
    # 语料
    x = ['The quick brown fox jumps over the lazy dog .', 'By Jove , my quick study of lexicography won a prize .', 'This is a short sentence .']
    # 实例化 Tokenizer 对象，使用单词级别的 Tokenize
    x_tk = Tokenizer(char_level=False)
    # 对语料进行 Tokenize
    x_tk.fit_on_texts(x)
    # 打印 Tokenize 的语料
    print(x_tk.texts_to_sequences(x))
```

编程要求

实现 tokenize 函数。

测试说明

平台会对你编写的代码进行测试：

测试输入：

```
['The quick brown fox jumps over the lazy dog .', 'By Jove , my quick
study of lexicography won a prize .', 'This is a short sentence .']
```

预期输出：

```
[[1, 2, 4, 5, 6, 7, 1, 8, 9], [10, 11, 12, 2, 13, 14, 15, 16, 3, 17],
[18, 19, 3, 20, 21]]
```

```
#coding:utf8
from keras.preprocessing.text import Tokenizer


def tokenize(data):
'''
tokenize
:param data: 语料，类型为 list
:return: tokenize 后的语料，类型为 list
'''
#*********Begin*********#

#*********End*********#
```

3、编写 Python 代码实现 padding 功能。

通常情况下语料库中的每个句子中的单词数量不可能会是完全一致的。就比如上一关中用来举例的语料中 3 条句子的单次数量是不同的。但是在搭建神经网

络时，神经网络中每层的神经元的个数其实就已经确定下来了。特别是输入层的神经元数量，每一个神经元代表着一个词。一边是单次数量不确定，另一边是需要确定好单词数量，怎么办呢？这个时候可以使用 padding 方法。

padding 其实就是设定一个最大的单词数量，当 tokenize 后的句子中单词数量小于最大单词数量时，就在后面补 0。

假设 tokenize 后的句子为:[18 19 3 20 21]，最大单词数量为 10，那么 padding 后的句子为:[18 19 3 20 21 0 0 0 0 0]。

keras 同样为我们实现好了 padding 功能，省得我们重新造轮子。示例代码如下:

```
from keras.preprocessing.sequence import pad_sequences
# tokenize 后的语料
x = [[[18 19  3 20 21]]]
# 打印 padding 的结果，最大单词数量为 10
print(pad_sequences(x, maxlen=10, padding='post'))
```

编程要求

实现 padding 函数。注意：最大单次数量为 data 中所有句子的单次数量的最大值。

测试说明

平台会对你编写的代码进行测试：

测试输入：

```
[[1, 2, 4, 5, 6, 7, 1, 8, 9], [10, 11, 12, 2, 13, 14, 15, 16, 3, 17],
[18, 19, 3, 20, 21]]
```

预期输出：

```
Using TensorFlow backend.
[[ 1  2  4  5  6  7  1  8  9  0]
 [10 11 12  2 13 14 15 16  3 17]
 [18 19  3 20 21  0  0  0  0  0]]
```

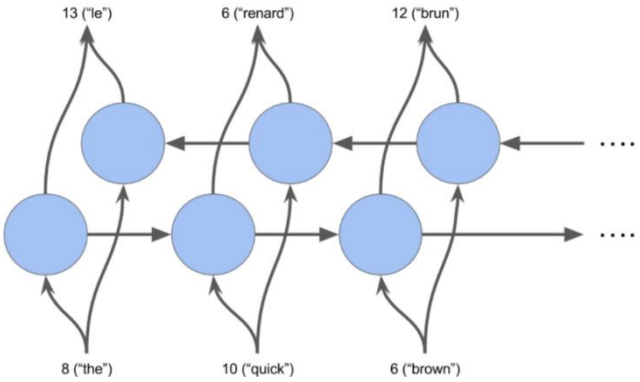
```
#coding:utf8
from keras.preprocessing.sequence import pad_sequences


```

```
def padding(data):
    '''
    padding，最大单词数量为 data 中所有句子的单词数量的最大值
    :param data: 语料，类型为 list
    :return: padding 后的语料，类型为 list
    '''
    #*********Begin*********#

    #*********End*********#
```

4、编写 Python 代码搭建一个双向 RNN 神经网络

双向 RNN

为什么需要既能从前往后又能从后往前的 RNN 呢，因为如果仅仅是从前往后的方式来理解句子的语义的话，会有可能产生歧义。例如：“He said, Teddy bears are on sale” 和 “He said, Teddy Roosevelt was a great President。在上面的两句话中，当我们看到“Teddy”和前两个词“He said”的时候，我们有可能无法理解这个句子是指 President 还是 Teddy bears。因此，为了解决这种歧义性，我们需要从后往前看。这就是双向 RNN 所能实现的。



怎样搭建双向 RNN 神经网络

以将英语翻译成法语的功能为例。当我们把英语和法语的语料进行 tokenize 和 padding 处理后，就可以将经过处理后的英语语料作为伸进网络的输入，将经过处理后的法语语料作为输出。所以这个网络应该是一个处理分类问题的神经网络。

因此，网络的输出层其实就是一个全连接层，神经元的数量就是法语 tokenize 后词的数量，激活函数是 softmax。

确定了输出层后，就要确定处理输入层数据的层了。很明显，就是用双向 RNN ！ keras 中有相应的接口来帮助我们实现双向 RNN 。示例代码如下:

```
# SimpleRNN 表示 RNN，Bidirectional 表示双向的意思。128 表示 RNN 有 128 个神经元，由于我们需要 RNN 处理后的序列，所以 return_sequences 为 True，dropout 是随机丢弃的概率，能够防止过拟合
Bidirectional(SimpleRNN(128, return_sequences = True, dropout=0.1), input_shape=input_shape)
```

知道怎样构建双向 RNN 层之后，相信你能够很轻松的构建出整个神经网络了。

编程要求

实现 build_model 函数，该函数的功能是构建一个含有 128 个神经元的双向 RNN 层和含有 french_vocab_size 个神经元的全连接层的神经网络。（怎样将层与层之间连接起来，请查阅 keras 官方文档。）

测试说明

平台会对你编写的代码进行测试：

测试输入：

无

预期输出：

```
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_1 (Bidirection (None, 52, 256)           35072
_________________________________________________________________
dense_1 (Dense)              (None, 52, 22)            5654
=================================================================
Total params: 40,726
Trainable params: 40,726
Non-trainable params: 0
_________________________________________________________________
```

阅读代码中注释部分要求，补充实现 begin-end 间代码，并运行测试结果。

```
#coding:utf8
from keras.models import Sequential, Model
from keras.layers import Input, Dense, SimpleRNN, Bidirectional


def build_model(input_shape, french_vocab_size):
```

```
    '''
    tokenize
    :param input_shape: 英语语料的 shape，类型为 list
    :param french_vocab_size: 法语语料词语的数量，类型为 int
    :return: 构建好的模型
    '''
    #*********Begin*********#
      #*********End*********#
```

5、编写 Python 代码，实现机器翻译功能。

编程要求

实现机器翻译功能。由于平台无法训练太久，所以这里只训练 10 个 epoch，若想达到比较好的翻译效果，可以在自己电脑上训练久一点。

```
#coding:utf8
import os
import warnings
warnings.filterwarnings('ignore')
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
from keras.models import Sequential, Model
from keras.layers import Input, Dense, SimpleRNN, Bidirectional,
TimeDistributed
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
    from keras.losses import sparse_categorical_crossentropy
    from keras.optimizers import Adam
    import numpy as np

    def load_data(path):
        '''
        读取原始语料数据，只读取前 3000 条文本
        :param path: 文件路径
        :return: 句子列表，如['he is a boy.', 'she is a girl']
        '''
        with open(path, 'r') as f:
            return f.readlines()[:3000]


    def tokenize(data):
        '''
        tokenize
        :param data: 语料，类型为 list
        :return: (tokenize 后的语料,tokenizer 对象)
```

```
        '''
        x_tk = Tokenizer(char_level = False)
        x_tk.fit_on_texts(data)
        return x_tk.texts_to_sequences(data), x_tk


    def padding(data, length=None):
        '''
        padding，最大单词数量为 dlength
        :param data: 语料，类型为 list
        :param data: 词数量，类型为 int
        :return: padding 后的语料，类型为 list
        '''
        if length is None:
            length = max([len(sentence) for sentence in data])
        return pad_sequences(data, maxlen = length, padding = 'post')


    def build_model(input_shape, french_vocab_size):
        '''
        tokenize
        :param input_shape: 英语语料的 shape，类型为 list
        :param french_vocab_size: 法语语料词语的数量，类型为 int
        :return: 构建好的模型
        '''
        model = Sequential()
        model.add(Bidirectional(SimpleRNN(128, return_sequences=True,
dropout=0.1), input_shape=input_shape[1:]))
        model.add(Dense(french_vocab_size, activation='softmax'))
        return model


    def logits_to_text(logits, tokenizer):
        """
        将神经网络的输出转换成句子
        :param logits: 神经网络的输出
        :param tokenizer: 语料的 tokenizer
        :return: 神经网络的输出所代表的字符串
        """
        index_to_words = {id: word for word, id in
tokenizer.word_index.items()}
        index_to_words[0] = '<PAD>'

```

```
        return ' '.join([index_to_words[prediction] for prediction in
np.argmax(logits, 1)])


    #*********Begin*********#
    #*********End*********#

    # 编译神经网络
    model.compile(loss=sparse_categorical_crossentropy, optimizer =
Adam(1e-3))
    # 训练（由于平台原因，无法训练太久，这里只训练 10 个 epoch）
    model.fit(tmp_x, preproc_french_sentences, batch_size=512, epochs=10,
validation_split=0.2, verbose=0)

    # 保存翻译结果
    with open('result.txt', 'w') as f:
        for i in range(10):
            result = english_sentences[i]+' -> ' +
logits_to_text(model.predict(np.expand_dims(tmp_x[i], 0))[0], french_tokenizer)
            f.write(result)
```

四、实验报告要求

参见实验报告模板