



北京大学内部教材  
未经本校允许,不得翻印

# 智能硬件应用实验



北京大学信息科学技术学院

2023 年 9 月



# 前言

“智能硬件应用实验”是一门面向低年级本科生的基础课程，目的是让同学们可以在大学学习的早期阶段了解人工智能的基本概念和应用领域，同时了解计算机硬件设备的简单工作原理。课程内容分成三个主线：Python 语言编程、树莓派硬件控制、人工智能算法。

本讲义通过单元实验把相关的知识串联起来，希望学生通过实验可以对本课程三条主线的知识有充分了解。讲义仅对实验本身需要了解的内容进行了简要的介绍，通过实验步骤帮助同学们完成实验。同学们如果需要深入了解相关知识还需要参考讲义中提供的其它参考书籍和网络资料与在线论文。

每个实验内容具有一定的独立性，学生在实验之前必须做好预习，了解实验中所使用硬件的基本原理，掌握相关算法的实现方案，这样才能在实验的过程中充分利用实验室的资源，在有限的实验时间内完成实验，并在学有余力的情况下对相关算法进行深入研究。如果在做实验的过程中遇到困难，也要及时和老师或助教沟通，以免耽误太多时间，不能按时完成实验。

课程的最后还需要同学们完成一个综合项目，利用现有的硬件资源，实现一个有一定趣味性、创新性的智能应用。项目独立完成，可以参考网络上的现有资源，可以使用开源的代码和程序，但必须有自己设计的部分，最后的作品应可以演示。

# 目 录

|                              |           |
|------------------------------|-----------|
| <b>第一章 智能硬件简介</b>            | <b>1</b>  |
| 1.1 硬件平台 . . . . .           | 1         |
| 1.2 操作系统环境 . . . . .         | 3         |
| 1.3 Python 编程语言简介 . . . . .  | 5         |
| 1.4 人工智能简介 . . . . .         | 11        |
| <b>第二章 简单游戏策略</b>            | <b>14</b> |
| 2.1 电脑游戏中的智能 . . . . .       | 14        |
| 2.2 树莓派的硬件扩展 . . . . .       | 15        |
| 2.3 Python 语言控制硬件 . . . . .  | 15        |
| 2.4 动手实验:猜拳游戏 . . . . .      | 19        |
| 2.5 思考题 . . . . .            | 20        |
| <b>第三章 聚类算法</b>              | <b>21</b> |
| 3.1 聚类算法简介 . . . . .         | 21        |
| 3.2 1-Wire 总线与传感器 . . . . .  | 21        |
| 3.3 用到的 Python 库 . . . . .   | 23        |
| 3.4 动手实验:模式识别 . . . . .      | 25        |
| 3.5 思考题 . . . . .            | 26        |
| <b>第四章 OpenCV 基础</b>         | <b>27</b> |
| 4.1 OpenCV 简介 . . . . .      | 27        |
| 4.2 图片形状检测 . . . . .         | 29        |
| 4.3 动手实验:图片处理及图形识别 . . . . . | 32        |
| 4.4 思考题 . . . . .            | 33        |
| <b>第五章 支持向量机分类器</b>          | <b>34</b> |
| 5.1 支持向量机 . . . . .          | 34        |
| 5.2 SPI OLED 显示模块 . . . . .  | 36        |
| 5.3 用到的 Python 库 . . . . .   | 41        |
| 5.4 动手实验:手写数字识别 . . . . .    | 45        |
| 5.5 思考题 . . . . .            | 46        |
| <b>第六章 构建专家系统</b>            | <b>47</b> |
| 6.1 专家系统 . . . . .           | 47        |
| 6.2 树莓派的舵机控制 . . . . .       | 52        |
| 6.3 动手实验:动物识别专家 . . . . .    | 58        |

|                                      |            |
|--------------------------------------|------------|
| 6.4 思考题 . . . . .                    | 59         |
| <b>第七章 OpenCV 进阶</b>                 | <b>60</b>  |
| 7.1 图片的统计特性 . . . . .                | 60         |
| 7.2 人脸识别算法基本原理 . . . . .             | 61         |
| 7.3 树莓派对摄像头的处理 . . . . .             | 63         |
| 7.4 运动物体的追踪 . . . . .                | 64         |
| 7.5 动手实验:视频的处理及人脸识别 . . . . .        | 66         |
| 7.6 思考题 . . . . .                    | 67         |
| <b>第八章 人工神经网络</b>                    | <b>68</b>  |
| 8.1 神经网络 . . . . .                   | 68         |
| 8.2 从线性回归到 MLP . . . . .             | 68         |
| 8.3 PyTorch 简介 . . . . .             | 70         |
| 8.4 卷积神经网络(CNN)模型 . . . . .          | 72         |
| 8.5 模数转换(AD)与数模转换(DA) . . . . .      | 74         |
| 8.6 动手实验:通过手势识别实时控制发光二极管亮度 . . . . . | 77         |
| 8.7 思考题 . . . . .                    | 87         |
| <b>第九章 语音识别和处理入门</b>                 | <b>88</b>  |
| 9.1 树莓派音频输入输出方法 . . . . .            | 88         |
| 9.2 HMM-GMM 模型介绍 . . . . .           | 89         |
| 9.3 动手实验:语音控制系统 . . . . .            | 91         |
| 9.4 思考题 . . . . .                    | 92         |
| <b>第十章 自然语言处理入门</b>                  | <b>93</b>  |
| 10.1 词向量的编码与处理方法 . . . . .           | 93         |
| 10.2 Twitter 文本情感分析 . . . . .        | 97         |
| 10.3 未名教学二号集群使用说明 . . . . .          | 100        |
| 10.4 动手实验:文本的情感分析 . . . . .          | 100        |
| <b>第十一章 TensorFlow 应用</b>            | <b>102</b> |
| 11.1 TensorFlow 介绍 . . . . .         | 102        |
| 11.2 MNIST 入门 . . . . .              | 102        |
| 11.3 动手实验:物体识别 . . . . .             | 105        |
| <b>第十二章 大语言模型简介</b>                  | <b>106</b> |
| 12.1 Transformer 模型 . . . . .        | 106        |
| 12.2 大语言模型 . . . . .                 | 107        |
| 12.3 python API 接口 . . . . .         | 108        |
| 12.4 动手实验:调用大语言模型 . . . . .          | 112        |
| <b>第十三章 综合实验</b>                     | <b>114</b> |
| 13.1 实验目的 . . . . .                  | 114        |
| 13.2 实验内容 . . . . .                  | 114        |



# 第一章 智能硬件简介

随着人工智能(Artificial Intelligence, 缩写为 AI)相关技术的不断发展, AI 已经深入到了社会生活的方方面面。本书将 AI 技术与电子设备硬件相结合, 通过“智能硬件”这一概念, 把智能算法通过真实的硬件平台实现。读者在学习智能算法的同时还能了解微处理器系统的基本知识, 掌握使用不同的硬件模块实现真实的智能设备的方法。

## 1.1 硬件平台

我们探讨的 AI 技术大部分只是一种算法, 可以在通用计算机上进行研究和验证。如果要把相关技术实现为产品, 就必须使用更加经济的硬件平台。这些硬件平台必须具有一定的运算能力, 可以运行特定的算法, 而实现算法最高效的平台就是微处理器。本书将使用树莓派(Raspbian Pi)作为主要的硬件, 完成全部算法的实现。

### 1.1.1 树莓派简介



图 1.1: 树莓派

树莓派 (Raspberry Pi) 是一款基于 ARM 的微型电脑主板, 外观尺寸仅有信用卡大小, 却具有电脑的所有基本功能, 又称卡片式电脑。树莓派以 SD/MicroSD 卡为内存硬盘, 卡片主板周围有 1/2/4 个 USB 接口和一个 10/100 以太网接口 (A 型没有网口), 可连接键盘、鼠标和网线, 同时拥有视频模拟信号的电视输出接口和 HDMI 高清视频输出接口, 以上部件全部整合在一张仅比信用卡稍大的主板上, 具备所有 PC 的基本功能, 只需接通显示器和键盘, 就能执行如电子表格、文字处理、玩游戏、播放高清视频等诸多功能。树莓派与外设的接口和连接方式如图1.2所示。

树莓派问世于 2012 年, 2019 年推出了树莓派 4。树莓派第 4 代 B 型搭载了 1.5GHz 的 64 位四核处理器 (ARM Cortex-A72 1.5GHz 64-bit quad-core ARMv8 CPU), 2GB 内存, 支持 802.11 b/g/n/ac 无线网卡, 低功耗蓝牙 5.0, 两个 USB 2.0 和两个 USB 3.0 接口, 1 个千兆网接口, 两个支持 4k 分辨率的 hdmi 显示接口。树莓派 4B 型外观如图1.3所示。它通过 USB Type-C 接口供电, 至少可以提供 2A 的电流才可

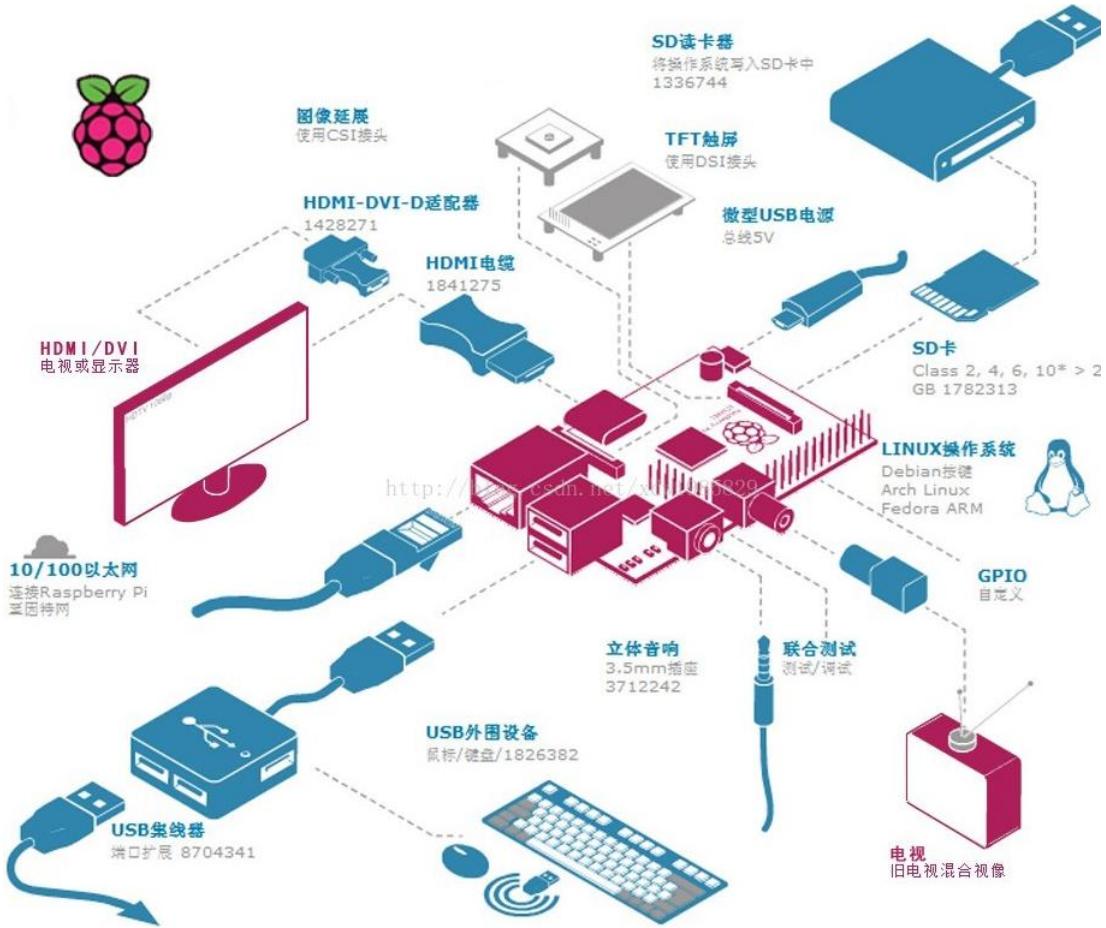


图 1.2: 树莓派的连接与扩展方式

以保证正常的工作。



图 1.3: 树莓派 4 代 B 型外观及接口配置

树莓派 4 代 B 型可直接外接显示器和键盘鼠标实现基本的输入输出。树莓派 4 代 B 型具有 HDMI 视频输出接口，可通过 HDMI 接口连接显示器输出树莓派的显示界面。键盘和鼠标可通过树莓派提供的 USB 接口连接到树莓派。

树莓派也可以通过网络连接进行访问，常用的方式是通过 SSH (Secure Shell) 访问树莓派，SSH 是专为远程登录会话和其他网络服务提供安全性的协议。SSH 客户端适用于多种平台，Windows 操作系统下

常用的 SSH 客户端软件包括 XShell、Putty 以及 SSH Secure Shell Client 等软件。

树莓派还可以通过远程登录的方式使用图形界面,所使用的协议为 VNC(Virtual Network Console)。使用前必须首先在树莓派的操作系统中安装 VNC 服务器,比如 tightvncserver,然后在其它计算机使用 VNC 客户端登录。Windows 操作系统下常见的客户端有 ReadVNC, VNC Viewer 等。

## 1.2 操作系统环境

树莓派中运行的操作系统是定制的 Raspbian,它基于 Debian 的 Linux 发行版,专门为树莓派的硬件所优化。Linux 操作系统内核是 1991 年由芬兰学生 Linux Torvalds 首先公开发布的,它的发行遵循 GPL(GNU general public license)协议,在 Internet 上不断被发展和完善。由于 Linux 的开放源代码的特性,很多优秀的程序员加入到 Linux 的开发行列,使得 Linux 的发展非常迅速,成为今天无论在服务器平台还是在嵌入式平台都非常具有竞争力的操作系统。

### 1.2.1 Linux 命令行简介

登录到树莓派操作系统中之后可以使用图形界面完成基本的文件操作和环境设置,但要更加灵活的使用 Linux 操作系统,或者需要通过字符协议远程登陆树莓派,就需要掌握 Linux 的常用字符命令。在图形界面下可以打开模拟终端获得命令行操作界面,这里列举了一些命令的功能(方括号中内容为命令参数),具体的使用方法还需要通过在线帮助或其它资料来熟悉。

#### 1. 文件操作

- 更改当前目录:cd [dir]
- 创建目录:mkdir [dir]
- 删除目录:rmdir [dir]
- 列出当前目录文件:ls
- 将 file1 复制到 file2:cp file1 file2
- 删除文件:rm [file]
- 显示文本文件内容:more [file]

#### 2. 网络操作

- 查看 IP 地址:ip addr
- 测试网络连通性:ping [IP]
- 通过 ssh 协议复制文件:scp

#### 3. 其它命令

- 显示磁盘占用情况:df
- 显示当前日期和时间:date
- 显示命令的说明信息:man [command]
- 安装软件 soft:apt-get install [soft]
- 退出当前会话:exit

想熟悉命令行的使用就必须多尝试,充分理解命令的精确含义,许多命令都具有“`--help`”参数用来显示简单的帮助信息,或者使用`man`命令来查找手册。

### 1.2.2 Bash 使用技巧

在 Linux 系统中,响应用户输入指令的程序被称为 shell。比较常用的 shell 有 csh、zsh 和 bash。由于多数发行版的默认 shell 都是 bash,这里就简单介绍一下它的操作特点。

首先是命令补全功能,它可以大大提高用户在命令行工作的效率。当输入一个比较长的命令的时候,用户可以只输入这个命令的开头几个字母,然后按 `<Tab>` 键。如果当前系统可以搜索到的命令中以这几

个字母开头的只有一个，系统就会自动把这个命令补全；否则再次输入 <Tab> 键可以显示出所有满足条件的命令，用户可以继续输入字母直到满足的命令只有一个的时候再按 <Tab> 键补全。

命令补全功能还适用于目录名和文件名的补全，在其他软件的辅助下还可以支持命令行参数的补全。例如当前目录中只有一个目录，用户可以如下输入将目录名补全。

```
1 $ cd <Tab>
```

另外一个功能是命令历史记录，bash 会自动记录用户输入的所有命令，用 history 命令可以查看输入命令的历史。默认情况下，bash 会记录最近输入的 500 条命令。在命令提示符下，按“上”方向键可以调出以前输入的命令，也可以用“下”方向键回到之后输入的命令。使用 Ctrl+r 按键组合还可以对历史进行查找，可以根据接下来输入的内容在历史记录中寻找最佳的匹配。

### 1.2.3 编辑器使用

树莓派自带的 Python 编辑和调试器是 Thonny，大多数情况可以满足编程的要求，这里另外还要介绍的是 Linux 系统自带的 vi 编辑器。

vi 是所有 Linux 发行版都提供的一个文本编辑软件。但 vi 和 Windows 下的任何其他编辑器都不类似，初次接触很可能完全无法了解它的用法。但在某些特殊环境，如嵌入式系统中，也许 vi 就是唯一的选择。另外，vi 是所有 Linux 和 UNIX 都包括的命令，掌握了它的用法就可以在所有 Linux 系统中进行编辑了。

vim 是 vi 的“超集”，它提供更加丰富的功能，更加适合编程的使用。扩展的功能包括语法高亮、多级撤销、可视选择等。大多数 Linux 发行版提供的都是 vim。vim 的另一个扩展是 gvim，它提供了图形化的工作前端，可以使用鼠标和操作菜单。执行 vi file\_name 进入 vi 的界面。vi 是一个有“模式”的编辑器，它具有“命令”、“命令行”和“编辑”三种主要模式。刚执行的 vi 处于命令状态。按“i”键进入插入状态就可以进行文字编辑了。编辑完成后按“Esc”键则可以返回命令状态，在命令状态按“:”键进入命令行状态，在命令行状态输入 wq<回车>（write and quit）就可以存盘退出。

vi 的命令状态主要是通过按键完成特定的功能，例如：

1. x: 删除光标位置的字符
2. dd(连续按两次 d 键): 是删除并保存当前行到缓冲区
3. p: 将缓冲区中内容输出到光标位置。
4. yy: 保存当前行到缓冲区
5. u: 取消上一次操作
6. h,l,j,k: 左、右、下、上移动光标

上面提到的命令都要注意区分大小写，它们前面还可以加数字前缀，例如 20j 就是下移 20 行光标。

在命令状态按“:”进入命令行状态，输入的“:”在窗口的最低端显示，如下面的例子所示。删除这个输入的冒号可以退出命令行状态。这个状态用来输入比较复杂的命令，例如查找替换：

```
1 :s/One/1/g
```

s 命令代表替换。第一个“/”号后面的是查找内容，第二个“/”后面是替换内容，第三个“/”后面是一个标志，g 代表替换本行的所有目标。

vi 的查找命令支持正则表达式，例如使用“\*”号进行模式匹配，熟练掌握正则表达式可以完成复杂的编辑功能。

在命令行状态的一个重要功能就是帮助，使用

```
1 :help :s
```

就可以查看上面介绍的 s 命令的说明。在查看帮助的时候,如果有其他条目的链接,则可以按“Ctrl+]”组合键跳转过去,然后按“Ctrl+T”组合键返回。

到目前为止已经介绍过好多“求助”的方法了。主要目的是告诉大家如何利用 Linux 来自己帮助自己,这是加快掌握 Linux 的捷径。因为没有任何一本书可以包揽 Linux 的所有功能和用法,只有在遇到难题的时候学会如何自己解决,才能真正掌握 Linux 的精髓。

## 1.3 Python 编程语言简介

Python 语言是一种面向对象的解释型计算机程序设计语言,由荷兰人 Guido van Rossum 于 1989 年发明,第一个公开发行版发行于 1991 年。Python 语言简单易学,代码简洁易懂,具有丰富和强大的库,封装了包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV 文件、密码系统、GUI(图形用户界面)和其他与系统有关的操作。Python 语言的主要缺点是相比于编译性语言其运行速度较慢。

### 1.3.1 Python 运行环境

Python 是一种脚本语言,需要在解释器的帮助下运行。树莓派中安装了两个版本的 Python 解释器,分别是 python2 和 python3,代表了两个比较流行的版本。如果采用 ssh 的方式登录到树莓派,直接运行 python 或者 python3 就可以打开这两个解释器的交互环境。如果是采用 VNC 登录或者直接操作树莓派,需要首先打开一个模拟终端环境。树莓派自带的 Thonny 或 Geany 也都是比较适合新手使用的集成开发工具。

如图1.4,在菜单中选择附件中的 LX 终端或者直接单击提示栏中的有“>\_”符号的图标就可以打开模拟终端,同样运行 python 即可。

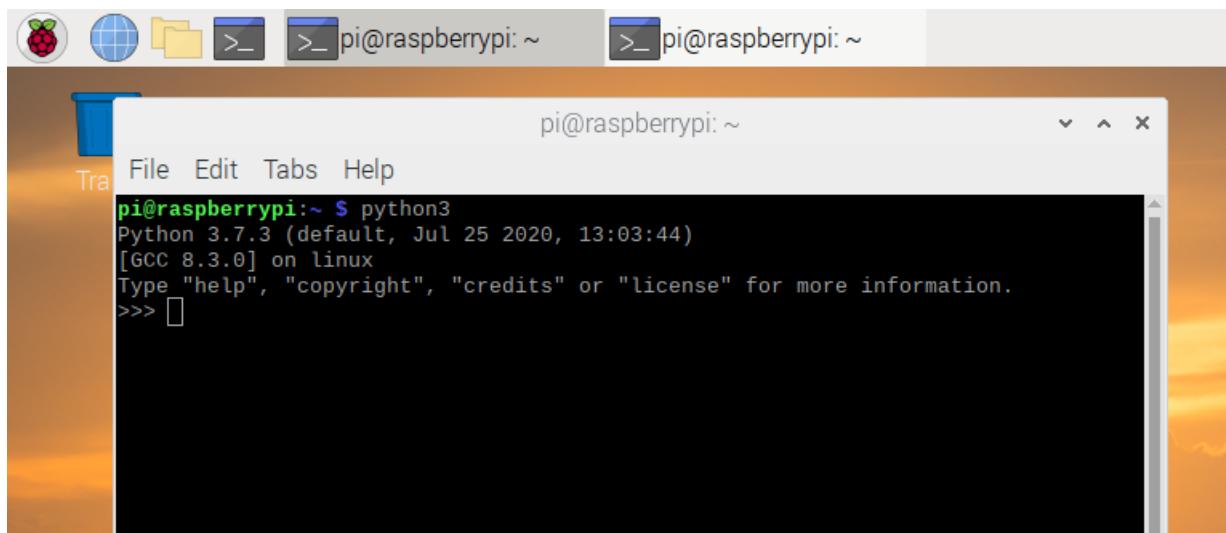


图 1.4: Python 运行环境

在交互模式的提示符 >>> 下,直接输入代码,按回车,就可以立刻得到代码执行结果,例如

```
1  >>> 100+200
2  300
```

如果要让 Python 打印出指定的文字,可以用 print() 函数,然后把希望打印的文字用单引号或者双引号括起来。

```

1 >>> print("hello,world!")
2 hello, world!

```

在 Python 交互模式下输入 `exit()` 并回车(或者按 `Ctrl+D`)，就退出了 Python 交互模式，并回到终端命令行。

Python 交互模式的代码是输入一行，执行一行。而命令行下直接运行.py 文件是一次性执行该文件内的所有代码。如

```

1 $ python3 hello.py
2 Hello, world!

```

Python 交互模式主要是为了调试 Python 代码，也便于初学者学习。在 Python 的交互式命令行写程序，优点是立刻就能得到结果，缺点是没法保存。因此实际编程的时候通常是采用文本编辑器编写 Python 代码，编写完成后保存为一个文件，程序就可以反复执行了。Python 程序的源代码是纯文本文件，通过缩进来标识不同的语法层次关系，而“#”号后面的内容为注释，可用来对程序进行解释，提高代码的可读性。

### 1.3.2 数据类型

Python 语言支持的数据类型包括整数、浮点数、字符串、布尔值以及列表等。Python 可以处理任意大小的整数，当然包括负整数，在程序中的表示方法和数学上的写法一模一样，例如：1, 100, -8080, 0 等。计算机由于使用二进制，所以，有时候用十六进制表示整数比较方便，十六进制用 0x 前缀和 0-9, a-f 表示，例如：0xff00, 0xa5b4c3d2 等。

浮点数可以用数学写法，如 1.23, 3.14, -9.01 等。但是对于很大或很小的浮点数，就必须用科学计数法表示，把 10 用 e 替代， $1.23 \times 10^9$  就是 1.23e9，或者 12.3e8, 0.000012 可以写成 1.2e-5 等。

字符串是以单引号或双引号括起来的任意文本，比如'abc', "xyz" 等等。需要注意的是，单引号或双引号本身只是一种表示方式，不是字符串的一部分，因此，字符串'abc' 只有 a, b, c 这 3 个字符。如果单引号本身也是一个字符，那就可以用双引括起来，比如"I'm OK" 包含的字符是 I, ' , m, 空格, O, K 这 6 个字符。

布尔值和布尔代数的表示完全一致，一个布尔值只有 `True`、`False` 两种值，要么是 `True`，要么是 `False`，在 Python 中，可以直接用 `True`、`False` 表示布尔值。

### 1.3.3 变量

变量是指向各种类型值的名字。在 Python 中，变量的使用环境非常宽松，没有明显的变量声明，等号 = 是赋值语句，可以把任意数据类型赋值给变量。同一个变量可以反复赋值，而且可以是不同类型的变量。这种变量本身类型不固定的语言称之为动态语言，与之对应的是静态语言。静态语言在定义变量时必须指定变量类型，如果赋值的时候类型不匹配，就会报错。

在 Python 中，如果你不能确定变量或数据的类型，就用解释器内置的函数 `type` 确认。例如：

```

1 >>> a = 'name'
2 type(a)
3 <class 'str'>
4 >>> classmates = ['Michael', 'Bob', 'Tracy']
5 >>> type(classmates)
6 <class 'list'>

```

### 1.3.4 元组、列表、字典和集合

在程序设计的时候经常需要把多个元素包装到一起来使用,例如纸张的规格、训练的多组数据等。这种可以容纳多个变量的数据结构叫做容器,Python 有多个容器类别,分别适合不同的应用场景。

元组(tuple)是最简单的容器,所有元组元素使用括号包围,例如:

```
1 BrickSize = (15, 23, 56)
2 Classmates = ('Bill', 'Carl')
```

元组一旦建立就不可以改变,里面的元素也可以是不同类型,常用的元组操作可以参考下面示例:

```
1 len(BrickSize)      # 获取元组长度
2 print(Classmates[1]) # 通过索引获取元素(第一个元素的索引为0)
3 BrickSize.index(15) # 查找元素的索引号
```

需要注意的是如果元组只有一个元素,那么定义的时候要在这个元素后面加上逗号,如 a = (1,)。

列表(list)是 Python 内置的另外一种容器类型,他和元组一样是一种有序的集合,但可以随时添加和删除其中的元素。列表中包含的元素可以具有相同的数据类型,也可以具有不同数据类型,例如:

```
1 Names = ['Michael', 'Sarah', 'Tracy']
2 Objects = ['Apple', 123, True]
3 Sizes = [[10, 20], [15, 25], [20, 35]] # 列表当然也可以嵌套
```

列表在程序中经常使用,用法也十分灵活,下面是一些应用的例子:

```
1 Names.append('Sarah')      # 在末尾增加元素
2 Objects.insert(2, 456)      # 在索引2的位置插入新元素
3 Objects.pop(2)             # 删除索引2的元素(不加参数缺省删除末尾元素)
4 Sizes.reverse()            # 元素顺序颠倒
5 Objects.clear()            # 删除全部元素
6 Sizes.sort()               # 对元素进行排序
7 Names.remove('Sarah')      # 根据内容删除元素(只删除第一个匹配)
```

注意列表在复制的时候只是新建一个引用,所代表的内容是相同的,如果需要完整复制(深复制)一个列表元素,要用 copy 方法,例如:

```
1 >>> Names = ['Michael', 'Sarah', 'Tracy']
2 >>> K=Names                  # 浅复制
3 >>> Q=Names.copy()          # 深复制
4 >>> Names.append('Violet') # 修改原始变量的值
5 >>> K                      # K 的值同时被改变
6 ['Michael', 'Sarah', 'Tracy', 'Violet']
7 >>> Q                      # Q 的值没有变化
8 ['Michael', 'Sarah', 'Tracy']
```

和列表类似,字典(dict)也是可以保存多个元素的变量类型,其每个元素都是一个冒号分隔的键值对,例如:

```
1 >>> dict = {'Alice': '1234', 'Bob': '5678', 'Charlie': '90'}
```

可以通过“键”对“值”进行查询,也可以对“键”进行赋值,例如:

```
1 >>> dict.keys()           # 返回全部的键列表
2 dict_keys(['Alice', 'Bob', 'Charlie'])
```

```

3      >>> dict.values()      # 返回全部的值列表
4      dict_values(['1234', '5678', '90'])
5      >>> print(dict['Alice'])
6      1234
7      >>> dict['John']='45'   # 赋值的同时新建了一个键值对

```

字典也包含一些内置的函数，如 `clear()` 清除字典内容；`key in dict` 查询键是否存在；`len(dict)` 查询字典元素个数等。

集合（set）也与列表类似，但是不能存储重复的数据，并且是没有存放顺序的区别，集合可以通过 `add()` 增加元素，也可以通过 `remove()` 删除元素。比较特别的，集合可以求交集和并集，例如：

```

1      >>> Names1 = {'Michael', 'Sarah', 'Tracy'}
2      >>> Names2 = {'Sarah', 'Violet', 'Alice'}
3      >>> Names1.intersection(Names2)          # 交集
4      {'Sarah'}
5      >>> Names1.union(Names2)                # 并集
6      {'Sarah', 'Alice', 'Tracy', 'Violet', 'Michael'}
7      >>> Names1.difference(Names2)           # 集合区别
8      {'Michael', 'Tracy'}

```

### 1.3.5 序列的下标访问

Python 中的字符串、元组和列表都是有序结构，都可以用下标进行访问，可以统称为序列。序列的下标访问主要有下面几种方式：

1. `seq[index]` 获得索引所在位置元素
2. `seq[index1:index2(:stride)]` 获得索引从 `index1` 到 `index2` 的全部元素，`stride` 是步长，缺省为 1，`index1` 空缺表示从头开始，`index2` 空缺表示到最后一个元素，`seq[:]` 表示全部元素。
3. `seq1 + seq2` 连接两个序列
4. `seq * expr` 序列元素重复 `expr` 遍

当索引值为负时，它表示从序列最后一个元素开始计数，例如，`seq[-1]` 可以获得序列的最后一个元素。

### 1.3.6 基本输入输出

在 Python 中，用 `print()` 函数在括号中加上字符串，就可以向屏幕上输出指定的文字。例如：

```

1      >>> print('hello,world')
2      hello, world
3      >>> name = 'Tom'
4      >>> print(name)
5      Tom

```

`print()` 函数也可以接受多个字符串，用逗号“,”隔开，就可以连成一串输出，`print()` 函数会依次打印每个字符串，遇到逗号“,”会输出一个空格，例如，

```

1      >>> print('The\brown\fox', 'jumps\bover', 'the\blazy\bdog.')
2      The brown fox jumps over the lazy dog.
3      >>> print('100\b+\b200\b=', 100 + 200)
4      100 + 200 = 300

```

在 Python 中,可以利用 input() 函数完成基本输入。利用 input() 函数可以让用户输入字符串,并存放到指定的变量里。例如,

```
1 >>> name = input()
2 Michael
```

当用户输入 name = input() 并按回车后,Python 交互式命令行就在等待输入了,用户可以输入任意字符,然后按回车后完成输入。另外,可以在 input() 函数加入提示信息。例如,

```
1 >>> name = input('please enter your name:')
2 please enter your name: Michael
3 >>> print('hello,', name)
4 hello, Michael
```

### 1.3.7 条件判断

在 Python 程序中,用 if 语句实现条件判断。例如,

```
1 total = query() # 查询库存
2 if total == 0:
3     print('Out of stock')
4     order(100)
```

根据 Python 的缩进规则,如果 if 语句判断是 True,就执行缩进的两行语句,否则,什么也不做。也可以给 if 语句添加一个 else 语句,如果 if 判断是 False,不执行 if 后的语句,而是执行 else 后的语句。需要注意的是 if 语句和 else 语句后面要添加冒号。例如:

```
1 food = input('What do you like to eat? ')
2 if food == 'Apple':
3     print('Please enjoy')
4 else:
5     print("Sorry, we don't have any")
```

另外,可利用 elif 语句添加多个条件判断,elif 是 else if 的缩写,可以有多个 elif 语句。例如:

```
1 age = int(input('Enter your age: '))
2 if age >= 18:
3     print('You are an adult!')
4 elif age >= 6:
5     print('You are a teenager')
6 else:
7     print('You are a kid')
```

### 1.3.8 循环

Python 的循环有两种,一种是 for...in 循环,另一种是 while 循环。for...in 循环可以依次将列表中的每个元素迭代出来。例如:

```
1 for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
2     print(i)
```

可以利用 Python 提供的 range() 函数控制 for 循环, range() 函数可以产生一个整数序列。上面利用 for 循环的求和代码可用 range() 函数进行简化。

```
1 for i in range(0,10): # 从 0 循环至 9
2     print(i)
```

另外,range() 函数产生的整数序列可以指定步进值,例如:

```
1 for i in range(0,10,2): # 从 0 循环至 9, 步进值为 2
2     print(i)
```

Python 的 while 循环只要条件满足就不断循环, 条件不满足时退出循环。例如:

```
1 n = 1
2 while n <= 100:
3     print(n)
4     n = n + 1
```

对于循环语句, 可以通过 break 语句退出循环或通过 continue 语句跳过当前循环。例如:

```
1 n = 1
2 while n <= 100:
3     if n > 10: # 当 n = 11 时, 条件满足, 执行 break 语句
4         break # 结束当前循环
5     print(n)
6     n = n + 1
7 print('END')
8 n = 0
9 while n < 10 :
10     n = n + 1
11     if n % 2 == 0 : # 如果 n 是偶数, 执行 continue 语句
12         continue # 跳过当前循环
13     print (n)
```

### 1.3.9 Python 的文件操作

在操作文件之前, 需要先把文件打开, 操作之后还需要把文件关闭, 与这两个操作对应的函数是 open 和 close。open 操作返回一个文件类, 后续对这个文件的操作都通过这个类来进行。open 函数的定义如下:

```
1 open(name[, mode[, buffering]])
```

其中 name 参数为文件的名称, mode 为文件的打开方式, buffering 表示缓冲区的大小。最常用的 mode 参数为”r” 和 ”w”, 分别代表读和写的方式, 当用 ”w” 的方式打开时, 如果文件已经存在, 则里面的内容会被清空。还有一种 ”a” 方式用来在文件后面添加新的内容。另外 ”r+”、”w+” 和 ”a+” 都以读写方式打开文件, 但 ”w+” 会清空文件, ”a+” 会将文件指针放到文件末尾。

文件操作包括文件的读和写, 对于文本文件, 可以认为其内容是一个字符串数组, 每个字符串代表文件的一行, 这样就可以利用下面的示例代码将整个文件读入。

```
1 f = open(filename, 'r')
2 lines = f.readlines()
3 f.close()
```

文件主要使用 write 函数将数据写入, 简单示例如下:

```

1 f = open(filename, 'a')
2 f.write('Last line\n')
3 f.close()

```

### 1.3.10 Python 库

Python 语言可以通过库来进行扩展，实现一些在特定领域比较常见的算法，供所有软件开发人员来使用，大大提高了 Python 语言的开发效率。本书后续介绍的人工智能算法很多都有相应的库来支持，复杂的运行过程可以通过简单几个 Python 语句来实现，降低了人工智能的应用门槛，这也使得 Python 成为人工智能开发中最常用的编程语言。

在 Python 的发行版中还包含了大量的标准库，可以在不额外安装软件的前提下使用。例如文件系统的操作、常用数据类型、常用算法等等。例如 `random` 库实现了常用随机数的产生，`random()` 函数在  $[0, 1)$  区间生成均匀分布的随机浮点数。`randint(a, b)` 函数返回在  $[a, b]$  区间的均匀分布整数。例如，下面代码生成了十个随机整数。

```

1 import random      # 导入 random 模块
2 data = []          # 定义一个空的列表 data
3 for i in range(0, 10):
4     data.append(random.randint(0, 99))

```

Python 标准库的资料可以参考官方的在线文档(<https://docs.python.org/zh-cn/3/library/>)，了解标准库所包含的内容并在程序中进行运用可以简化代码设计，保证代码的质量。

### 1.3.11 Python 程序的编写和调试

对于新手来说，直接使用 Vim 编辑源代码来开发 Python 程序确实有些难度，树莓派操作系统中安装的 Thonny 则是一个更好的选择。它是一个简单的 Python 集成开发环境，支持语法高亮、代码补全、单步调试等功能，可以方便的编写 Python 代码。图 1.5 是 Thonny 的运行界面，上半部分是编辑的程序代码，下半部分是程序的运行结果。

点击工具栏上的调试按钮或者在 Run 菜单中选择 Debug 命令就可以对程序进行单步调试。单步调试时可以选择 Step Over(快捷键 F6)一次执行一条语句，对于函数和类方法等可以一次跳过；也可以选择 Step Into(快捷键 F7)在遇到函数和类方法时可以进入模块内部单步执行。Step Out(快捷键 F8)是跳出当前模块，表示进入到函数和类方法时执行到这个模块的末尾，跳到上一层级的代码。

在编写程序时熟练使用调试功能可以帮助新手快速熟悉语言功能，找出程序中存在的 bug，加深对 Python 程序运行过程的理解。

## 1.4 人工智能简介

“智能”在新华字典中的定义是“智慧和能力”。这个过分简单的解释并不能让我们清楚的了解什么是智能。实际上，对于人类如何思考，如何解决问题，科学家们依然没有明确的答案。大自然经过千百万年的进化发展，使人类具备了智慧的大脑，可以对物理世界进行观察和分析，并创造出自然界不存在的工具来为人类服务。如今，人类的创造目标包括了智能，这也就是人工智能的主要研究方向。

人类对智能的探索其实古已有之，在《列子》中就记载了偃师设计的木人为周穆王跳舞的故事，这个木人不但可以跳舞说话，还可以抛媚眼。中国古书中记载的类似故事还有很多，不过都没有原理说明，更没有图纸，只能是当时人们的美好幻想。也从另外一个侧面说明人类对自身的思维和运动能力一直具有强烈的

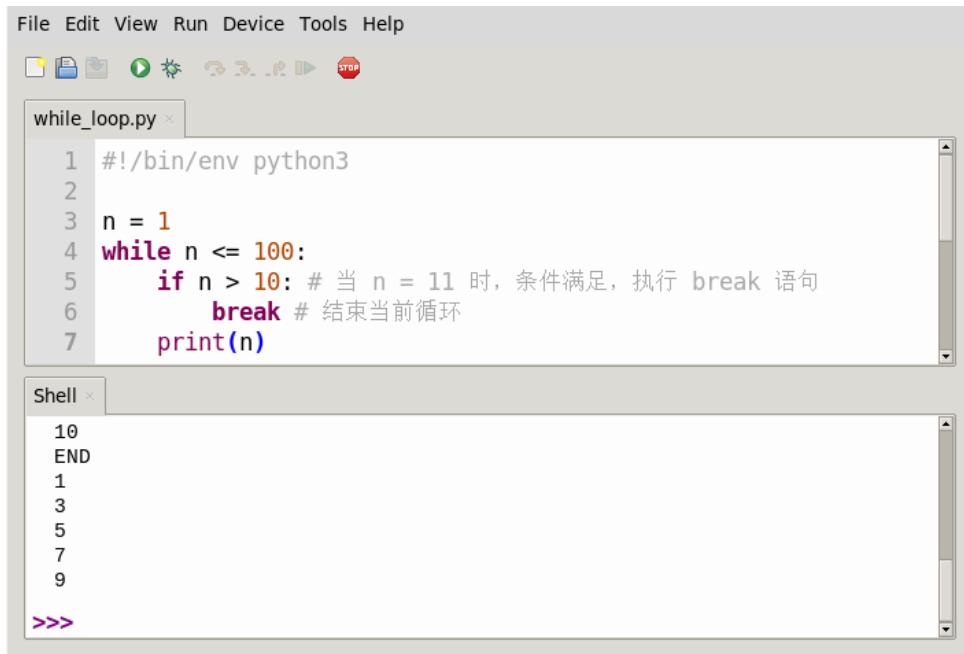


图 1.5: Thonny 的运行界面

好奇心,想了解其背后的原理,并制造出相应的工具。可以认为这种非天然的人类工具具有的智能就是人工智能,与之相关的科学研究组成了人工智能这一学科方向。

人类的智能有很多类别,比如运动能力、计算能力、语言能力、视觉识别能力、逻辑分析能力等等。其中一些能力需要与其它能力配合才可以算作具有智能。比如运动能力,人类很早就可以制造出远超人类力量与速度的工具,但这些还需要人的操作,并不具有智能。最近发展起来的类人机器人则需要复杂的计算模型,并配合视觉处理、逻辑推理等多个模块才可以获得差强人意的效果;智能汽车自动驾驶也是需要多种传感器和算法的支持才能在公路上自由行驶。

下面表格列举了一些常见的人工智能研究方向,而本书也只是涉及了其中的一小部分。

表 1.1: 人工智能研究方向

| 研究方向   | 简单描述                                   |
|--------|----------------------------------------|
| 自然语言处理 | 研究通过文本或者音频进行语言交流的设备或者程序                |
| 逻辑推理   | 一种专门用于推理的编程语言,如 Prolog、Datalog 等       |
| 机器视觉   | 让计算机从图片或者视频中获取信息并进行处理                  |
| 机器学习   | 使计算机可以通过学习提高算法的性能                      |
| 机器人    | 通过算法使机器可以执行复杂的动作任务                     |
| 情感计算   | 通过视觉或文本等识别和处理或者模拟人类的情感                 |
| 认知理论   | 研究人类思维的结构,通过计算机对大脑进行建模                 |
| 游戏智能   | 在游戏中让非人类玩家具有类似真实人类的表现                  |
| 智能助理   | 类似目前手机中常见的语音助手,帮助人们完成一些简单任务            |
| 专家系统   | 将知识用计算机可以理解的方式进行表达,使计算机可以完成需要大量知识的复杂任务 |

虽然目前人类对自己大脑的了解还不能解释智能的产生,但人工智能的发展却丝毫不受影响。大部分的人工智能研究并不基于人类智能的原理,而主要依靠背后的数学原理和电子学技术。随着计算能力的提

升和人工智能研究的深入，很多从前看起来非常困难的任务已经被实现。机器图像识别已经在很多工业领域获得应用，自然语言处理也可以在电话和网络客服中很好的工作，围棋比赛也没有人类可以战胜电脑，人工智能正进入一个蓬勃发展的阶段，不断有新的算法出现，计算机的“智能”水平正不断获得提升。

在人工智能领域，python 语言的使用十分普遍，很多算法都通过 python 语言实现并以开源的方式发布在网络上。为了更方便的实现人工智能应用，python 语言还发展出很多程序框架，比较著名的有 scikit-learn、TensorFlow、PyTorch、Caffe、Theano 等。对于比较复杂的人工智能应用，由于需要强大的计算资源，很多 AI 公司也都公开了网络调用 API，可以利用云资源来实现特定的功能。

## 第二章 简单游戏策略

本章我们用树莓派制作一个猜拳机器人，玩家可以通过按键输入手势类型与树莓派对抗，树莓派则通过一定的算法提高自己的胜率。通过本章的学习，读者可以掌握树莓派 GPIO 的基本用法，并加深对 Python 语言的理解。

### 2.1 电脑游戏中的智能

相信大部分读者都玩过电脑游戏，而电脑游戏中又有很大一部分是对抗性的。在电脑游戏中，玩家的对手——电脑——必然具有一定的“智能”才能使得游戏比较“有趣”。对于不同的游戏类型，所需要的智能算法也不同，这里仅讨论有限状态、公开信息的简单对抗游戏，例如猜拳游戏。通过这个大家都很熟悉的游戏，探讨可能涉及到的算法。

#### 2.1.1 随机策略

就像人类有时会通过抛硬币来决定一些事情，计算机通过产生随机数的方法来进行决策是最简单而往往也很有效的方法。比如在猜拳游戏中，如果一方采用完全随机的出拳策略，理论上胜负也同样随机，因此会有 50% 左右的胜率，不算好也不算差。

但目前大部分随机数发生器都是伪随机序列，虽然看起来分布比较均匀，却是通过公式产生，可以通过一定的算法来预测。经典的随机数产生方法是线性同余法(Linear Congruence Generator, LCG)，其基本公式如下：

$$X_{n+1} = (aX_n + c) \bmod m \quad (2.1)$$

线性同余法定义了三个常数：乘量  $a$ 、增量  $c$  和模数  $m$ ,  $X_i$  为生成的随机数序列。为了保证产生的随机数可以获得最大周期(不重复长度)，这三个常数应该符合：

1.  $m$  与  $c$  互质
2.  $m$  的所有质因数都能整除  $a - 1$
3. 若  $m$  是 4 的倍数， $a - 1$  也是 4 的倍数
4.  $a, c$  都是正数，还要比  $m$  小

对于比较简单的情况，游戏一方仅初始化一次随机数( $X_0$  的取值，即随机数的种子)，游戏另一方就有可能猜到这个种子，并正确预测出后续产生的全部随机数，所谓的随机性也就完全丧失了。为了获得更好的随机数，可以采用更好的算法，或者采用可以产生真随机数的物理硬件来实现。

#### 2.1.2 根据历史数据进行决策

在多次重复游戏中，游戏历史数据可以作为分析对象，简单的历史模式匹配就可以成为很好的策略：当各方面条件都相同时，采用一个过去曾经成功的动作获胜的概率会比较大。而如果考虑到对方也会分析历史，那么曾经采用的动作有可能被针对，相对要降低选择的概率。

部分多次重复的游戏运行过程可以抽象成马尔可夫链：从一个游戏状态到另外一个状态的转换状态概率可以视为常数。这样经过几轮游戏之后，就可以积累一定的数据，并根据统计信息计算状态转移的概率。

假设模型的状态空间为  $S_i, i = 1, \dots, n$ , 动作空间为  $A_i, i = 1, \dots, m$ 。则在特定状态下采取某种动作的概率可以表示为  $P(A_i|S_j)$ 。对于多阶情况则可以表示为:  $P(A_i|S_{j_1}, S_{j_2}, \dots, S_{j_k})$ 。

对于猜拳游戏，根据双方的出拳状态，一共有 9 种可能，那下一次的出拳概率可以根据这 9 种情况采取不同的概率应对。而对方的出拳概率可以根据历史统计资料获得。上面只是考虑了一阶的情况，对于高阶情况则需要考虑更多的历史状态。

### 2.1.3 利用深度学习进行决策

前面的方法都是利用了人类的经验来设计策略，能不能通过人工学习的方式获得更好的策略呢？这也是目前人工智能研究中的重要分支，在很多领域也已经获得了很好的结果。例如著名的 Alpha-Zero 就可以完全通过学习，在围棋游戏中打败人类顶尖高手。

## 2.2 树莓派的硬件扩展

### 2.2.1 树莓派 GPIO 接口简介

在嵌入式系统中，经常需要控制许多结构简单的外部设备或者电路，这些设备有的需要由 CPU 提供输出信号，有的需要向 CPU 提供输入信号，并且许多设备或电路只要求有开/关两种状态就可以满足使用需要，例如比如 LED 的亮与灭。对此类设备的控制，使用传统的串口或者并口就显得比较复杂。

在嵌入式微处理器上通常提供了一种“通用可编程 I/O 端口”，也就是 GPIO(General Purpose Input Output)。一个 GPIO 端口至少需要两个寄存器，一个做控制用的“通用 IO 端口控制寄存器”，还有一个是存放数据的“通用 I/O 端口数据寄存器”。数据寄存器的每一位是和 GPIO 的硬件引脚对应的，而数据的传递方向是通过控制寄存器设置的，通过控制寄存器可以设置每一位引脚的数据流向。

GPIO 是树莓派最强大的特点之一，它也是树莓派与外部世界交互的物理接口之一。如图1.3 所示，树莓派型提供了 40 个 GPIO 引脚可用于对简单外设进行控制。树莓派 40 个 GPIO 引脚定义如图2.1所示。

### 2.2.2 PIONEER600 树莓派扩展板简介

Pionner600 扩展板是一款 Raspberry Pi 的外围扩展板，带有丰富的板载资源，扩展了多种常用接口，提供了一些简易的 I/O 设备，例如 LED 指示灯、五向遥杆等。Pionner600 扩展板的外观与配置如图 2.2 所示。

## 2.3 Python 语言控制硬件

### 2.3.1 RPi.GPIO 的基本使用

树莓派 RPi.GPIO Python 模块提供了 GPIO 相关的操作包括 GPIO 接口的配置、输入及输出等。

下面的语句实现 RPi.GPIO 模块的导入：

```
1 import RPi.GPIO as GPIO
```

通过该操作完成了 RPi.GPIO 模块的导入，并为其定义了一个别名 GPIO，在之后的程序中可通过别名使用 RPi.GPIO 模块。

| Raspberry Pi2 GPIO Header |                                    |                                    |      |
|---------------------------|------------------------------------|------------------------------------|------|
| Pin#                      | NAME                               | NAME                               | Pin# |
| 01                        | 3.3v DC Power                      | DC Power 5v                        | 02   |
| 03                        | GPIO02 (SDA1 , I <sup>2</sup> C)   | DC Power 5v                        | 04   |
| 05                        | GPIO03 (SCL1 , I <sup>2</sup> C)   | Ground                             | 06   |
| 07                        | GPIO04 (GPIO_GCLK)                 | (TXD0) GPIO14                      | 08   |
| 09                        | Ground                             | (RXD0) GPIO15                      | 10   |
| 11                        | GPIO17 (GPIO_GEN0)                 | (GPIO_GEN1) GPIO18                 | 12   |
| 13                        | GPIO27 (GPIO_GEN2)                 | Ground                             | 14   |
| 15                        | GPIO22 (GPIO_GEN3)                 | (GPIO_GEN4) GPIO23                 | 16   |
| 17                        | 3.3v DC Power                      | (GPIO_GEN5) GPIO24                 | 18   |
| 19                        | GPIO10 (SPI_MOSI)                  | Ground                             | 20   |
| 21                        | GPIO09 (SPI_MISO)                  | (GPIO_GEN6) GPIO25                 | 22   |
| 23                        | GPIO11 (SPI_CLK)                   | (SPI_CE0_N) GPIO08                 | 24   |
| 25                        | Ground                             | (SPI_CE1_N) GPIO07                 | 26   |
| 27                        | ID_SD (I <sup>2</sup> C ID EEPROM) | (I <sup>2</sup> C ID EEPROM) ID_SC | 28   |
| 29                        | GPIO05                             | Ground                             | 30   |
| 31                        | GPIO06                             | GPIO12                             | 32   |
| 33                        | GPIO13                             | Ground                             | 34   |
| 35                        | GPIO19                             | GPIO16                             | 36   |
| 37                        | GPIO26                             | GPIO20                             | 38   |
| 39                        | Ground                             | GPIO21                             | 40   |

图 2.1: 树莓派 GPIO 引脚定义

目前有两种方式可以通过 RPi.GPIO 对树莓派上的 IO 引脚进行编号。第一种方式是使用 BOARD 编号系统。该方式参考树莓派主板上 P1 接线柱的引脚编号。使用该方式的优点是无需考虑主板的修订版本，硬件始终都是可用的状态，无需从新连接线路和更改您的代码。

第二种方式是使用 BCM 编号。这是一种较低层的工作方式。该方式参考 Broadcom SOC 的通道编号，具体的对应关系可参考图 2.1，使用过程中，要保证主板上的引脚与图表上标注的通道编号相对应。在使用 GPIO 时必须指定使用哪种编号方式，指定方式如下：

```

1  GPIO.setmode(GPIO.BOARD) # 采用 BOARD 编号
2  # 或者
3  GPIO.setmode(GPIO.BCM) # 采用 BCM 编号

```

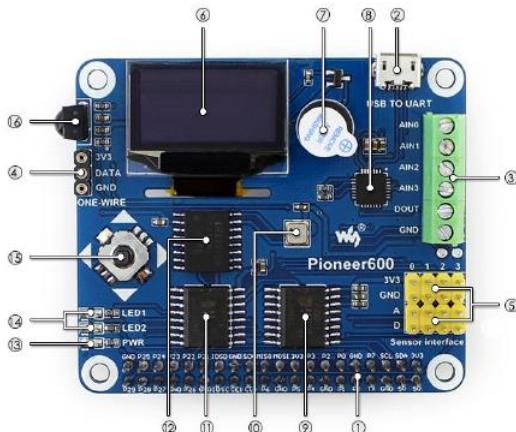
在使用 GPIO 输入输出前，需要对每个用于输入或输出的引脚配置通道。配置的方式如下：

```

1  # 配置 channel 指定的通道为输入通道
2  #channel 与使用的编号方式对应
3  GPIO.setup(channel, GPIO.IN, GPIO.PUD_UP)
4  # 配置 channel 指定的通道为输出通道
5  GPIO.setup(channel, GPIO.OUT)
6  # 配置 channel 指定的通道为输出通道，且规定通道初始输出高电平
7  GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)

```

在设置为输入状态的时候，如果外部电路没有连接上拉电阻，可以通过 GPIO.PUD\_UP 参数设置



- |                                |                           |
|--------------------------------|---------------------------|
| <b>[ 核心接口简介 ]</b>              |                           |
| <b>1. Raspberry Pi GPIO 接口</b> | <b>7. 蜂鸣器</b>             |
| 方便接入 RPi                       |                           |
| <b>2. USB TO UART 接口</b>       | <b>8. CP2102</b>          |
| 方便通过串口终端控制 Raspberry Pi        | USB 转串口芯片                 |
| <b>3. AD/DA 输入输出接口 (接线端子)</b>  | <b>9. PCF8591</b>         |
| 方便在各种场合使用                      | 8 位精度 AD/DA 芯片, I2C 接口    |
| <b>4. ONE-WIRE 接口</b>          | <b>10. BMP180</b>         |
| 可接入 DS18B20 等单总线设备             | 压力传感器, I2C 接口             |
| <b>5. 传感器接口</b>                | <b>11. PCF8574</b>        |
| 方便接入各类传感器                      | I/O 扩展芯片, I2C 接口          |
| <b>[ 器件简介 ]</b>                |                           |
| <b>6. 0.96 寸 OLED</b>          | <b>12. DS3231</b>         |
| 驱动芯片为 SSD1306, 128×64 分辨率,     | 高精度 RTC 芯片, I2C 接口        |
| SPI 接口                         |                           |
|                                | <b>13. 电源 LED</b>         |
|                                | <b>14. 用户 LED</b>         |
|                                | <b>15. 五向摇杆</b>           |
|                                | <b>16. LFN0038K 红外接收头</b> |

图 2.2: PIONEER600 树莓派扩展板

GPIO 的内部上拉电阻有效, 以保证输入端在没有接入信号的时候有确定的输入值(1)。完成通道的配置后, 就可以通过通道读取 GPIO 引脚的值或者设置 GPIO 引脚的输出状态。

```

1  GPIO.input(channel) # 读取 GPIO 引脚的值
2  # 引脚的值: 0/GPIO.LOW/False 或者 1/GPIO.HIGH/True.
3  GPIO.output(channel, state)
4  # 设置 GPIO 引脚的输出状态为 state
5  # State 的值: 0/GPIO.LOW/False 或者 1/GPIO.HIGH/True.

```

脉宽调制 (PWM) 是指用微处理器的数字输出来对模拟电路进行控制, 是一种对模拟信号电平进行数字编码的方法。在树莓派上, 可以通过对 GPIO 的编程来实现脉宽调制。RPI.GPIO 模块的脉宽调制 (PWM) 功能的基本使用方式如下:

```

1  # 创建一个 通道 channel 的 PWM 实例
2  pwm = GPIO.PWM(channel, frequency )
3  # 启动 PWM , 并指定占空比 dc , dc 的范围 0.0~100.0
4  pwm.start(dc)
5  # 更改 PWM 脉冲重复的频率为 frequency
6  pwm.ChangeFrequency(freq)
7  # 更改 PWM 的占空比为 dc
8  pwm.ChangeDutyCycle(dc)
9  # 停止 PWM

```

```
10    pwm.stop()
```

一般来说，程序到达最后都需要释放资源，这个好习惯可以避免偶然损坏树莓派。释放程序本中使用的引脚如下：

```
1    GPIO.cleanup()
```

注意，GPIO.cleanup() 只会释放掉脚本中使用的 GPIO 引脚，并会清除设置的引脚编号规则。

### 2.3.2 按键检测

利用树莓派上 GPIO 接口实现按键的输入检测。Pioneer600 扩展板上扩展了一个五向遥杆，遥杆可上下左右拨动，也可以按下。遥杆按下的输入接到了树莓派 GPIO 的第 38 号引脚，BCM 编号为 20（即图2.1 中 GPIO20）。可以利用树莓派 RPi.GPIO 模块实现对 Pioneer600 扩展板五向遥杆的按键输入检测，按下一次按键就在屏幕上打印“KEY PRESS”。下面代码实现了对按键输入 GPIO 通道的初始化。

```
1    import RPi.GPIO as GPIO
2
3    KEY = 20
4
5    GPIO.setmode(GPIO.BCM)
6    GPIO.setup(KEY, GPIO.IN, GPIO.PUD_UP)  # 上拉电阻
7    print("Key Test Program")
```

完成测试按键程序的过程中注意要保证每次按键只有一次输出信息（消抖），同时检测按键间隙要有延时，免得占用过多 CPU 时间。

另外一种实现检测按键的方式是利用软件中断和回调函数，GPIO 模块包含 add\_event\_detect 函数，用来设置发生指定事件时刻的回调函数：

```
1    from RPi.GPIO import add_event_detect
2    def my_callback(ch):
3        print("KEY PRESS")
4    add_event_detect(channel, GPIO.RISING, callback=my_callback, bouncetime=200)
```

当系统检测到指定 GPIO 的上升沿（还可以设置为 GPIO.FALLING 或 GPIO.BOTH）时，就会自动调用指定的回调函数 my\_callback（函数的参数是按键的编号），用户程序无需再调用这个函数。add\_event\_detect 函数的最后一个参数是软件消抖的延时毫秒数，用于防止由于按键的抖动引起的错误调用。

### 2.3.3 PWM 信号输出

数字端口的输出只能是高和低两种状态，如果用来控制 LED 就只有亮和灭两种状态。但如果这个连接了 LED 的端口输出一个方波，则这个 LED 就有一半的时间处于亮的状态，另外一半时间处于灭的状态。当方波的频率比较高的时候，人看起来这个 LED 就是一种半暗的状态。

更进一步，当控制输出的方波有不同的占空比的时候，LED 的亮度就会有不同的变化，从宏观上看就是端口的输出功率在不停的变化。这种输出信号的形式就叫做 PWM（Pulse Width Modulation，脉宽调制）。下面的程序代码实现了控制 LED 灯产生呼吸灯效果，请在树莓派上编辑运行程序，并观察效果。

```
1    import RPi.GPIO as GPIO
2    import time
3
4    LED = 26
5
6    GPIO.setmode(GPIO.BCM)
```

```

5     GPIO.setup(LED,GPIO.OUT)
6
7     p = GPIO.PWM(LED,50)
8     p.start(0)
9     try:
10        while True:
11            for dc in range(0,101,5):
12                p.ChangeDutyCycle(dc)
13                time.sleep(0.05)
14            for dc in range(100,-1,-5):
15                p.ChangeDutyCycle(dc)
16                time.sleep(0.05)
17        except KeyboardInterrupt:
18            pass # 空语句,不执行任何操作,只起到占位作用
19        p.stop()
20    GPIO.cleanup()

```

### 2.3.4 Python 的异常捕获机制

Python 的异常捕获机制可以让程序具有更好的容错性,当程序运行出现意外情况时,系统会自动生成一个 Error 对象来通知程序,从而实现将“业务实现代码”和“错误处理代码”分离,提供更好的可读性。异常捕获机制的代码结构如下:

```

1   try:
2       # 业务实现代码
3       ...
4   except Error1:
5       # 错误处理代码
6       ...
7   finally:
8       # 不管是否发生异常,一定会执行的代码
9       ...

```

例如希望程序等待用户按下 Ctrl+C 组合键再退出,可以用下面的代码来实现。

```

1   try:
2       while True:    # 死循环
3           time.sleep(1) # 休眠 1 秒钟
4   except KeyboardInterrupt: # 键盘中断事件
5       pass
6   GPIO.cleanup()

```

## 2.4 动手实验:猜拳游戏

### 2.4.1 实验目的

1. 熟悉树莓派的开发环境。
2. 熟悉 Python 语言的编程和调试。
3. 掌握树莓派 GPIO 接口的编程方式和使用方法。

#### 4. 设计简单的游戏策略

##### 2.4.2 实验内容

###### 2.4.2.1 GPIO 输出

利用树莓派上 GPIO 接口实现 LED 指示灯的输出控制。Pioneer600 扩展板 LED 指示灯的 BCM 编号为 26。请实现以下功能：

1. 利用树莓派 RPi.GPIO 模块的输出功能实现 LED 指示灯的闪烁，点亮 0.2 秒，熄灭 0.2 秒。
2. 利用树莓派 RPi.GPIO 模块的脉宽调制功能实现呼吸灯的功能。

###### 2.4.2.2 按键输入

利用按键对 LED 指示灯的输出进行控制。

1. 利用按键控制 LED 灯明灭，每按一次按键就切换 LED 灯的状态，由明至暗或由暗至明。
2. 单击按键使 LED 灯进入闪烁模式，再次单击按键闪烁频率加倍，双击按键停止闪烁（提示：双击可以定义为 0.5 秒钟内按键两次，计时可以使用 time.time()）。

###### 2.4.2.3 实现猜拳游戏

猜拳是一种简单的小游戏，共有剪刀、石头、布三个手势。二人同时用手做出相应形状，输赢判断规则为：剪刀赢布，布赢石头，石头赢剪刀。在树莓派上实现猜拳游戏要通过三个按键来实现三种输入，三个按键分别接入到 GPIO 的 29、31、33 号管脚（BCM 编号为 5、6、13）代表剪刀、石头、布三个手势。程序在用户按下按键前产生自己的手势编号，当用户按下按键时判断胜负。用 LED 灯作为按键输入成功的反馈，在屏幕上打印当前的比分（胜/局数）。

提升计算机的出拳策略，根据用户历史“出拳”信息，尽量提高树莓派的胜率。策略可以仅仅根据上一次的用户出拳结果，也可以根据多次出拳结果的统计信息。

## 2.5 思考题

1. 如何产生不同分布的随机数？
2. 如果猜拳游戏的对手是电脑，什么样的策略更好？

# 第三章 聚类算法

聚类是机器学习中非常重要的一类算法，它不需要对数据进行标记，属于非监督学习。本章将介绍最常用的 K 均值聚类算法，对从传感器获取的数据进行分类，进而实现预测的功能。本章所使用的传感器使用 1-Wire 总线进行通信，仅需要一根数据线就可以传输消息。

## 3.1 聚类算法简介

聚类算法试图将数据集中的样本分成若干个通常不相交的子集，每个子集称为一个簇（cluster）。通过这样的划分，每个簇就可能对应于一个潜在的概念，例如可以把生物分成植物和动物，把书籍分成文史类和科技类等。这种分类方式在人类认识世界的过程中不断的在进行着，但如果让计算机掌握这样的分类能力就不那么容易了。

基于不同的策略，可以设计出多种类型的聚类算法，这里介绍一种非常简单的 K 均值聚类算法（K-means clustering）。

K 均值聚类算法是一种迭代求解的聚类分析算法，其步骤是随机选取 K 个位置作为初始的聚类中心，然后计算每个样本与各个聚类中心之间的距离，把每个样本分配给距离它最近的聚类中心。聚类中心以及分配给它们的样本就代表一个聚类。每分配完成后，聚类的聚类中心会根据聚类中现有的样本被重新计算，获得新的聚类中心。

这个过程将不断重复直到满足某个终止条件。终止条件可以是没有（或最小数目）样本被重新分配给不同的聚类，或者是聚类中心不再发生变化。

## 3.2 1-Wire 总线与传感器

### 3.2.1 1-Wire 总线简介

1-Wire 总线（单总线）是 Maxim 全资子公司 Dallas 的一项专有技术。与目前多数标准串行数据通信方式 SPI/I2C/MICROWIRE 不同，它采用单根信号线，既传输时钟又传输数据，而且数据传输是双向的。1-Wire 总线接口具有节省 I/O 资源、结构简单、成本低廉、便于总线扩展和维护等诸多优点。

如图 3.1 所示，1-Wire 总线由一个总线主节点以及一个或多个从节点组成系统，主节点通过一根信号线对从节点进行数据的读取。1-Wire 总线的主节点一般是微控制器，从节点通常为单总线器件，例如温度传感器、身份识别器等。每一个符合 1-Wire 协议的从节点都有一个唯一的地址，包括 48 位的序列号、8 位的家族代码和 8 位的 CRC 代码。主节点对各从节点的寻址依据这 64 位的不同来进行。

1-Wire 总线利用一根信号线实现双向通信。因此其协议对时序的要求较严格。基本的时序包括复位及应答时序、写一位时序、读一位时序。在复位及应答时序中，主器件发出复位信号后，要求从器件在规定的时间内送回应答信号；在位读和位写时序中，主器件要在规定的时间内读回或写出数据。1-Wire 总线适用于单个主机系统，能够控制一个或多个从机设备。主机可以是微控制器，从机可以是单总线器件，它们之间的数据交换只通过一条信号线。当只有一个从机位于总线上时，系统可按照单节点系统操作；而当多个从机位于总线上时，则系统按照多节点系统操作。

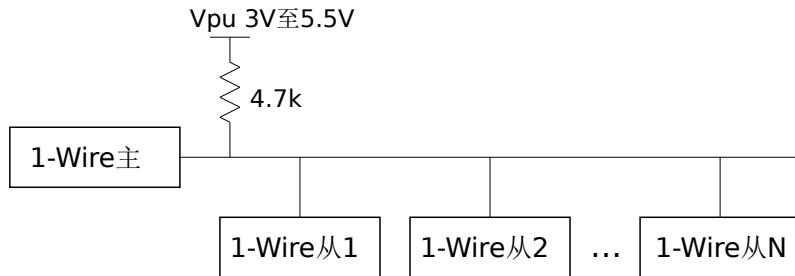


图 3.1: 1-Wire 总线系统组成

1-Wire 总线被定义为仅有一根数据线，数据线连接一个 4.7K 欧姆的上拉电阻，电阻再接到电源 (3V 到 5.5V)。每个设备 (主设备或从设备) 通过一个漏极开路或 3 态门引脚连接至数据线上。这就允许每个设备“释放”数据线，当设备没有传递数据时其他设备可以有效地使用数据线。

### 3.2.2 温度传感器 DS18B20

DS18B20 数字温度传感器提供 9-Bit 到 12-Bit 的摄氏温度测量精度和一个用户可编程的非易失性且具有过温和低温触发报警的报警功能。DS18B20 采用的 1-Wire 通信即仅采用一个数据线 (以及地) 与微控制器进行通信。该传感器的温度检测范围为 -55°C 至 +125°C，并且在温度范围超过 -10°C 至 85°C 之外时还具有 +0.5°C 的精度。此外，DS18B20 可以直接由数据线供电而不需要外部电源供电。

每片 DS18B20 都有一个独一无二的 64 位序列号，所以一个 1-Wire 总线上可连接多个 DS18B20 设备。因此，在一个分布式的大环境里用一个微控制器控制多个 DS18B20 是非常简单的。这些特征使得其在环境控制、在建筑、设备及机械的温度监控系统、以及温度过程控制系统中有着很大的优势。DS18B20 的外观与引脚定义如图 3.2 所示，内部结构图 3.3 所示。

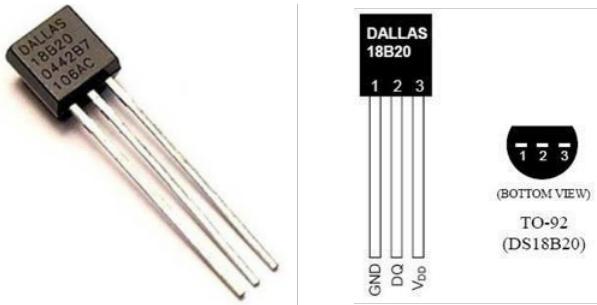


图 3.2: DS18B20 外观与引脚定义

### 3.2.3 树莓派温度传感器 DS18B20 的基本使用

实验器材中的 Pioneer 600 扩展板利用树莓派 GPIO 接口扩展了 1-Wire 总线接口，可以方便的接入各种 1-Wire 器件。实验中用到的温度传感器 DS18B20 通过 Pioneer 600 扩展板提供的 1-Wire 接口连接到树莓派，参考图 2.2 中的 ONE-WIRE 接口的位置。

树莓派提供了 2 个驱动模块 w1-gpio 与 w1-therm，分别用于通过 GPIO 扩展 1-Wire 总线接口及提供对温度传感器 DS18B20 的读写控制。其中，w1-gpio 提供了 1-Wire 总线的 I/O 操作方法，w1-therm 提供了对温度传感器 DS18B20 的内部操作方法。在通过 1-Wire 总线访问温度传感器之前需要确认系统已加载 w1-gpio 与 w1-therm 模块。在命令行输入 lsmod 命令，lsmod 命令用于显示已载入系统的模块。lsmod 命令的输出如包含 w1-gpio 与 w1-therm 模块则表示可以正常使用此接口。

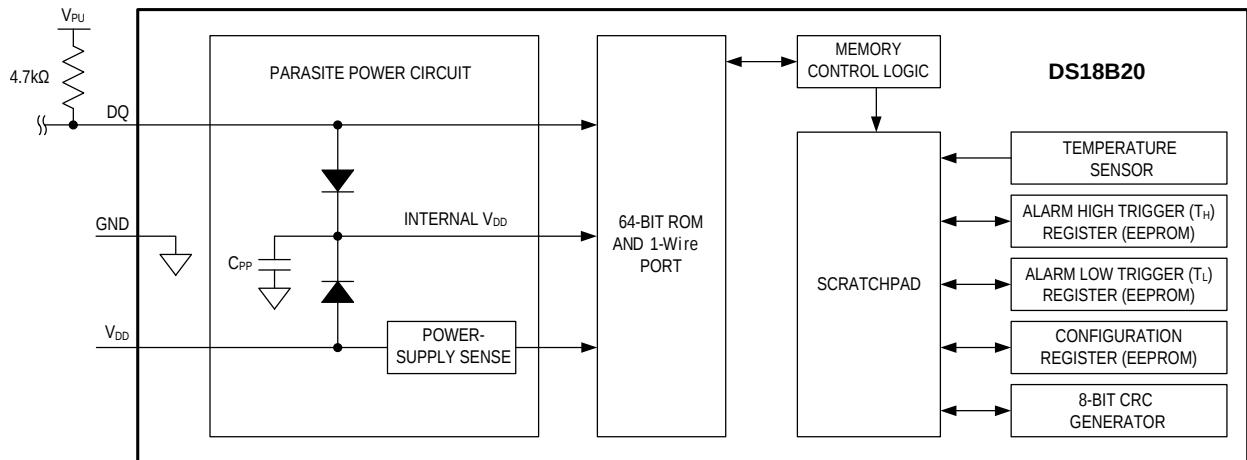


图 3.3: DS18B20 内部结构

如果通过 `lsmod` 命令发现 `w1-gpio` 与 `w1-therm` 模块尚未加载，可以通过 `modprobe` 命令加载这 2 个模块。加载方式如下：

```
1 pi@raspberrypi:~$ sudo modprobe w1-gpio
2 pi@raspberrypi:~$ sudo modprobe w1-therm
```

其中，`sudo` 是 linux 系统管理指令，允许系统管理员让普通用户执行一些或者全部的 root 权限命令。确认或完成 `w1-gpio` 与 `w1-therm` 模块的加载后，进入文件系统 `/sys/bus/w1/devices` 目录，并显示当前目录，操作方式和输出结果如下：

```
1 pi@raspberrypi:~$ cd /sys/bus/w1/devices
2 pi@raspberrypi:~$ ls
3 28-00000674869d w1_bus_master1
```

执行 `ls` 命令后在 `/sys/bus/w1/devices` 目录下会发现一个以 `28-XXXX` 开头的文件夹（如果接多个 DS18B20，将会看到多个 `28-xxxx` 的文件，分别对应各个 DS18B20）。进入以 `28-XXXX` 开头的文件夹，显示文件夹会发现一个名为 `w1_slave` 的文件，利用 `cat` 命令读取文件 `w1_slave` 的内容会返回温度传感器当前的温度值。操作方式与输出结果如下：

```
1 pi@raspberrypi:~$ cd 28-00000674869d
2 pi@raspberrypi:~$ sudo cat w1_slave
3 70 01 4b 46 7f ff 10 10 e1 : crc=e1 YES
4 70 01 4b 46 7f ff 10 10 e1 t=23000
```

执行 `cat w1_slave` 命令输出 2 行结果，第一行显示了 CRC 校验结果，最后的 YES 表示 CRC 校验成功，数据有效；第二行输出结果中的 `t=23000` 就是当前的温度值，换算成摄氏度需要除以 1000，即当前温度为  $23000/1000=23$  摄氏度。

## 3.3 用到的 Python 库

### 3.3.1 glob 库

在不知道文件具体名称的时候如果要查看目录中的文件或者目录，可以使用 python 的标准库 `glob`，它的参数是要查看的文件名，一般通过使用通配符来查找多个文件。例如下面代码列出 `dir` 目录下的全部文件：

```

1 import glob
2 for name in glob.glob('dir/*'):
3     print(name)

```

常用的通配符有：

- 星号(\*)匹配 0 个或多个字符
- 问号(?)匹配 1 个字符
- 中括号([])表示范围,例如 [a-k] 表示匹配这个范围的全部字符

如果需要递归的查找可以使用 recursive=True 参数,例如:

```

1 import glob
2 files = glob.glob('**/*.txt'):

```

此时 files 是当前目录下全部扩展名为 txt 的文件列表。这里使用两个星号用来匹配 0 个或多个目录,仅在递归模式下使用。

### 3.3.2 NumPy 库

NumPy 是 Python 中科学计算的基础软件包。它是一个提供多维数组对象,多种派生对象(如:掩码数组、矩阵)以及用于快速操作数组的函数及 API,它包括数学、逻辑、数组形状变换、排序、选择、I/O、离散傅立叶变换、基本线性代数、基本统计运算、随机模拟等等。

我们这里可以用它生成特定分布的随机数。例如下面代码生成了 10 个方差为 sigma, 均值为 mean 的正态分布随机数。

```

1 import numpy as np
2 mean = 5
3 sigma = 1.3
4 x=mean+sigma*np.random.randn(10)

```

NumPy 包的常用数据类型是 ndarray 对象。它与标准 Python Array(数组)之间有几个重要的区别:

1. NumPy 数组在创建时具有固定的大小,与 Python 的原生数组对象(可以动态增长)不同。更改 ndarray 的大小将创建一个新数组并删除原来的数组。
2. NumPy 数组中的元素都需要具有相同的数据类型,因此在内存中的大小相同。例外情况:Python 的原生数组里包含了 NumPy 的对象的时候,这种情况下就允许不同大小元素的数组。
3. NumPy 数组有助于对大量数据进行高级数学和其他类型的操作。通常,这些操作的执行效率更高,比使用 Python 原生数组的代码更少。

### 3.3.3 Matplotlib 库

Matplotlib 是一个 Python 2D 绘图库,可以生成各种格式和跨平台交互式环境的出版物质量图片数据。它的语法格式类似于 matlab,例如下面代码画出 x 数据的直方图。

```

1 import matplotlib.pyplot as plt
2 plt.hist(x,80,histtype='bar',facecolor='yellowgreen',alpha=0.75)
3 plt.show()

```

## 3.4 动手实验:模式识别

### 3.4.1 实验目的

1. 了解 1-Wire 单总线基本原理。
2. 了解温度传感器 DS18B20 的基本使用方式。
3. 掌握树莓派 1-Wire 总线接口的编程方式和使用方法。
4. 使用 K 均值聚类算法对数据进行分类。

### 3.4.2 读取温度传感器的值

1. 在终端下通过 cat 命令读出当前的温度值。
2. 使用 python 代码获取当前温度值并显示到屏幕上。

在使用 python 代码读取 DS18B20 的温度时,不能假定相关的设备文件名已知,即为了更好的可移植性,不能将 DS18B20 的序列号认为是常量。正确的方法是使用 glob 库,利用通配符查找到对应的设备文件,然后再打开文件获取温度数据,这样可以保证程序在不同的开发版上都可以正常工作。

### 3.4.3 聚类算法实现

假定某共用办公室有两个人使用,他们使用办公室的时候都会用空调遥控器设置房间的温度。但空调遥控器显示面板坏掉了,只能通过加减温度的方式盲调整。已知他们习惯的温度不同,办公室的温度计有记录功能,每天记录 12 小时的温度,每 30 分钟记录一次,根据一个月所记录的数据,通过聚类算法,算出这两个人的喜好温度,并估算他们各使用办公室多少时间。

可以认为这两个人使用办公室的时候室内温度是不同方差和不同均值的正态分布随机数,采用 randn 函数生成模拟数据,并用 matplotlib 画出数据的直方图。

使用 K 均值聚类算法将模拟数据分成两个簇,解答前面的问题。

使用温度传感器读入温度,根据温度值,判断是谁在使用办公室。

### 3.4.4 二维数据的聚类

使用随机算法生成二维的模拟数据,实现聚类算法。考虑下面两种情况:

- 模拟数据围绕两个中心点正态分布,方差相同
- 模拟数据采用如下代码生成,分布如图 3.4 所示

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 r1 = 2
5 r2 = 4
6 p1 = []
7 p2 = []
8
9 for i in range(10000):
10     x = np.random.rand() * r2 * 2 - r2;
11     y = np.random.rand() * r2 * 2 - r2;
12     if abs(x * x + y * y - r1) < 0.5:
13         p1.append([x, y])
14     if abs(x * x + y * y - r2) < 0.5:

```

```
15     p2.append([x, y])
16 plt.scatter(np.array(p1)[:,0], np.array(p1)[:,1], c='r', s=3)
17 plt.scatter(np.array(p2)[:,0], np.array(p2)[:,1], c='b', s=3)
18 plt.show()
```

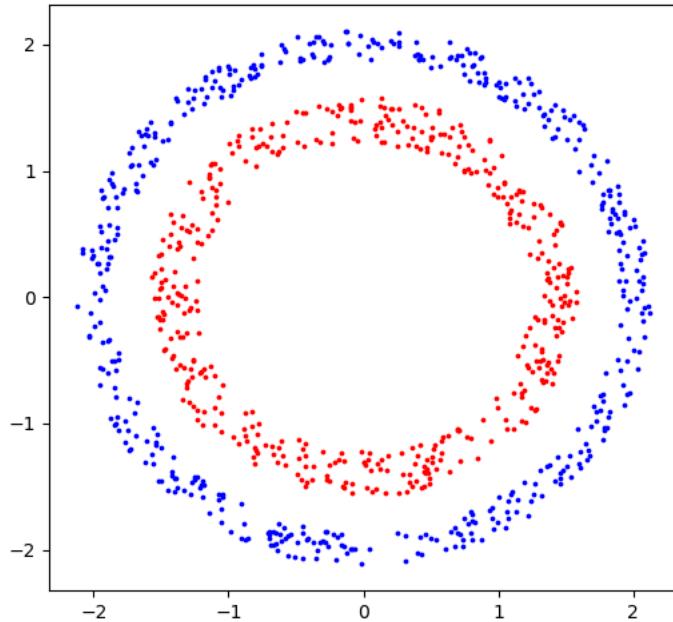


图 3.4: 环状数据

### 3.5 思考题

1. 分析实验结果, 得到的聚类中心是否与模拟数据的均值相等, 不等的话请解释偏差的原因。
2. 讨论 k 均值算法的局限性

# 第四章 OpenCV 基础

机器视觉一直是人工智能中研究的热点，也是一些上层应用的基石。如何对图片、视频流数据进行处理，进而对目标进行提取、识别是该领域中的重要工作。本章内容以 OpenCV 程序库为基础，首先介绍图片处理的基本原理，再引入一些经典的图像识别算法，并在树莓派平台上完成图片的处理及简单图形识别实践，进一步深入学习 Python 中对矩阵的处理及 Numpy 库的使用。

## 4.1 OpenCV 简介

OpenCV (Open Source Computer Vision Library) 是一个开源的机器视觉程序库，源自于 1999 年 Intel 的一个研究项目。OpenCV 包含多种常见的机器视觉算法，目前仍然处于积极开发中，不断有新的功能和算法被整合进来。OpenCV 支持多用开发语言如 C++, Java, Python 等，本试验主要采用 Python 语言。

OpenCV 具有模块化结构，包含不同层次的功能为用户使用。下面是常用的模块列表：

核心功能 (Core functionality): 定义了基本的数据结构，包括多维矩阵数组和被其他模块使用的基本功能。

图像处理 (Image processing): 一个图像处理模块，它包括线性和非线性图像滤波，几何图形转化（重置大小，放射和透视变形，通用基本表格重置映射），色彩空间转换，直方图等。

视频处理 (video): 一个视频处理模块，它包括动作判断，背景弱化和目标跟踪算法。

3D 校准 (calib3d): 基于多视图的几何算法，平面和立体摄像机校准，对象姿势判断，立体匹配算法，和 3D 元素的重建。

平面特征 (features2d): 突出的特征判断，特征描述和对特征描述的对比。

对象检测 (objdetect): 目标和预定义类别实例化的检测（例如：脸、眼睛、杯子、人、汽车等等）。

图形接口 (highgui): 一个容易使用的用户功能界面。

视频输入输出 (videoio): 一个容易使用的视频采集和视频解码器。

OpenCV 采用 C++ 语言开发，它还包含一个 Python 语言的绑定，在树莓派上可以直接使用 Python 代码使用 OpenCV 所提供的功能。

OpenCV 包含的功能模块非常多，这章主要介绍图片、物体检测与识别中常用的功能，其它的模块与函数的用法请参考其它书籍或者在线文档。

### 4.1.1 图片的存储

在计算机中，图片由二维的数据矩阵（高度，宽度）来表示及存储，如图 4.1 所示。矩阵中的每个元素表示一个像素点 (Pixel)，在彩色图片中每个像素点由三种基本颜色组成。

OpenCV 中提供了图片的读、写函数 `imread()`、`imshow()`、`imwrite()` 来读取、显示、保存图片，`imread(filename, flags=None)` 函数能够读取大多数类型的图片并以 Numpy 数组的形式保存图片像素点的信息，函数中的 `flags` 指示图片读取的模式，常用的读取灰度图片 `flags=0`，彩色图片 `flags=1`。例如下

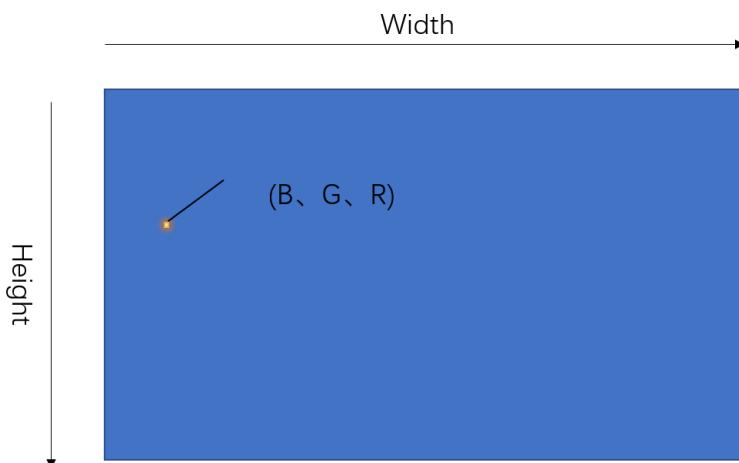


图 4.1: 图片及像素点示意图

面的代码就用来打开一副保存在电脑上的图像。需要注意的是为正常显示图片，调用 `imshow()` 函数后需要采用 `waitKey()` 函数设定显示等待时间。

```

1 #!/bin/python
2 import cv2 as cv # 加载 OpenCV 库
3 img = cv.imread('test_set/lena_color_512.tif',1)    # 打开图像文件
4 cv.imshow('Lena',img)      # 显示图像
5 cv.waitKey(0)            # 等待用户按键
6 cv.destroyAllWindows() # 关闭显示窗口

```

### 4.1.2 图片的基本处理

对图片的基本处理即对二维图片数据的处理，主要包括对图片色彩的空间的处理以及对图片尺寸、滤波等处理，OpenCV 中也提供了丰富的处理函数实现对图片的基本操作。

- 图片的颜色空间分解

颜色空间是采用数学的方式表示颜色的方法，常见的有 RGB、HSV 等。RGB 就是采用颜色的红色(R)、绿色(G)和蓝色(B)分量来表示颜色；HSV 色调(H)、饱和度(S)和明度(V)三个分量来表示颜色，如图 4.2 表示。由于 HSV 颜色空间更符合人的主观感受，因此在图像处理软件中有很多应用，在机器视觉领域也经常用来对色彩进行分析。但 OpenCV 中缺省的颜色空间采用 BGR 表示，常常需要进行图片的颜色空间转化。OpenCV 中提供 `cvtColor()` 函数来实现不同颜色空间的转化。

在图片的颜色处理中常常需要对图片的进行颜色过滤，提取图片中感兴趣的色彩。OpenCV 采用 `inRange()` 函数对图像中的色彩进行过滤。但是需要注意的是 `inRange()` 函数中需要指定色彩的范围，为更加方便的对设置范围进行调整可以使用 `createTrackbar()`、`getTrackbarPos()` 函数在图形界面中增加滑动条<sup>1</sup>，参考代码如下(这里没有使用滑动条)：

```

1 import numpy as np
2 import cv2 as cv
3 from picamera2 import Picamera2
4 import time
5 cam = Picamera2() # 初始化摄像头
6 cam.still_configuration.main.size = (640,480)
7 cam.still_configuration.main.format = 'RGB888'

```

```

8 cam.configure("still")
9 cam.start()
10 time.sleep(1)
11 while ( True ):
12     frame = cam.capture_array("main") # 读取摄像头数据
13     hsv=cv.cvtColor(frame, cv.COLOR_BGR2HSV) # 转换颜色空间
14     # 通过颜色设计模板
15     image_mask=cv.inRange(hsv,np.array([40,50,50]), np.array([80,255,255]))
16     # 计算输出图像
17     output=cv.bitwise_and(frame,frame,mask=image_mask)
18     cv.imshow('Original',frame) # 显示原始图像
19     cv.imshow('Output',output) # 显示输出图像
20     if cv.waitKey(1) == ord("q"): # 等待按键
21         break
22 cv.destroyAllWindows()
23 cam.release()

```

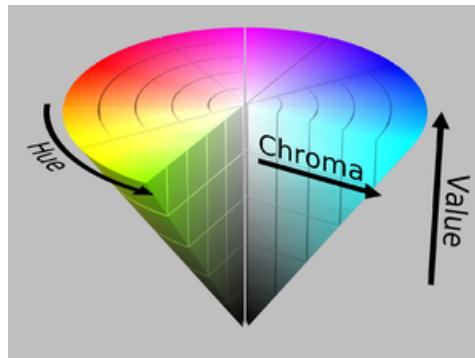


图 4.2: HSV 色彩空间

- 图片的基本处理

图片的基本处理包括旋转、拉伸、透视、分割、混合等，其本质就是对图像的矩阵进行一些变换，例如当对图片进行拉伸时通过插值函数来补全其中的像素点，将多幅图片合成在一起形成一幅图片。OpenCV 提供了丰富的图片处理函数如 `cv_bitwise_and()`, `cv_add()`, `cv_resize()`, 具体可以参考 OpenCV 中提供的例子<sup>2</sup>。

## 4.2 图片形状检测

图片中的不同的形状通常依赖于色彩、强度上的变化来区分，而计算机要识别不同的形状依赖于色彩、强度的变化信息的提取，在图片处理中该类信息采用梯度 (gradient) 来描述。梯度信息是图片中重要信息，关于梯度的计算及应用是图形检测的基础。在进行图片形状检测之前，通常需要对图片做颜色变化、二值化、滤波等预处理。

### 4.2.1 图片滤波及梯度运算

- 图片滤波运算

在计算图片的梯度前，采用滤波方法降低噪声带来的影响。对图像的滤波处理类似信号处理中的卷积运算，主要区别在于关于图片的滤波是对二维数据的运算，图像的滤波处理能够有效消除图片中

的噪声, 式 4.1 为一个 3X3 的归一化平滑滤波器, OpenCV 中提供了 `filter2D()`, `Blur()` 等滤波函数, 深入理解图像处理中的滤波运算也是图像处理的基础。OpenCV 提供了许多轮廓检测函数, 如 `Canny()`, `findContours()` 等。

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (4.1)$$

- 图片的梯度(gradient)计算

在 OpenCV 中对于图片梯度的计算也可以看做是一种特殊的滤波运算, OpenCV 中提供多种梯度运算函数, 如 `Sobel()`、`Scharr()`、`Laplacian()` 等, `Sobel()` 函数能够获得指定方向上的图像的梯度值, `Scharr()` 是对 `Sobel()` 的进一步优化。`Laplacian()` 方法计算两个方向上的梯度  $\Delta src = \frac{\delta^2 src}{\delta x^2} + \frac{\delta^2 src}{\delta y^2}$ , 其核函数<sup>1</sup>如式 4.2 所示。通过计算图像的梯度来实现对图像的边检测。

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (4.2)$$

## 4.2.2 图片形状检测

对于图片形状检测, 根据实际应用场景及检测需求, 主要可以基于色彩、形状来进行识别及检测。Hough 变换支持对规则的形状如直线、圆形检测, 对于复杂形状的检测 OpenCV 提供了 `findContours()` 来完成不规则形状的识别。下面分别介绍其基本原理及用法。

- 特殊形状检测

对于特殊形状如直线、圆形, 也可以采用特殊的检测方法进行检测。如 OpenCV 中采用 Hough 变换对图形中的特殊形状(直线、圆形等)进行检测。Hough 变换的基本原理就是将图片中点变换到另外一个坐标系, 如图 4.3 所示, 对于图中点依次进行 Hough 变换  $(r, \theta)$ , 每条经过点  $(x, y)$  的直线都有唯一的  $(r, \theta)$  与之对应, 随着经过该点的直线的斜率的变化  $(r, \theta)$  满足 `sin()` 函数。对图片中的点均进行 Hough 变换, 可以得出同在一条直线上点穿过  $r_0, \theta_0$  的次数最多, 由此可以判断同在一条直线上的点。

类似的对于圆形的识别, 根据圆形满足式  $(x - x_0)^2 + (y - y_0)^2 = r^2$ , 其中  $(x_0, y_0)$  为圆心,  $r$  为圆的半径。对于圆形的识别, 首先是在已知半径  $r_0$  下通过对圆的交点进行计次, 根据最大的计次数来确定圆心点, 如图 4.4 ;根据确定的圆心的像素点, 再确定半径。

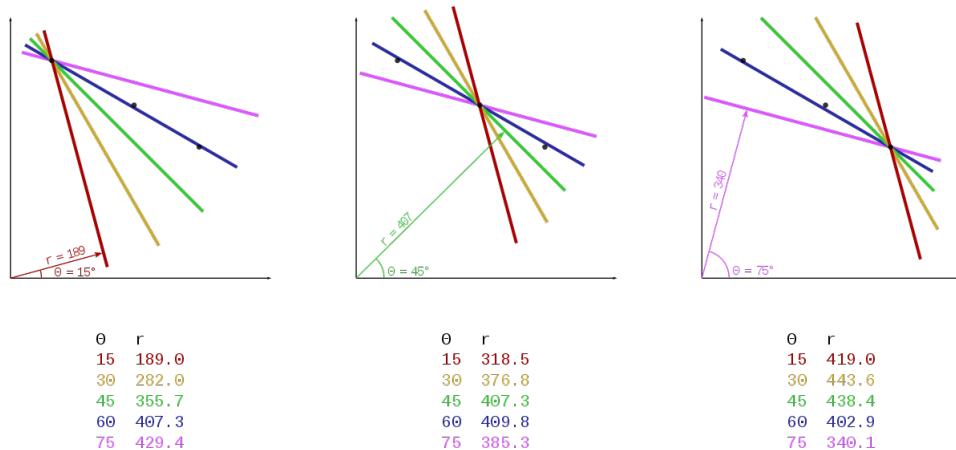
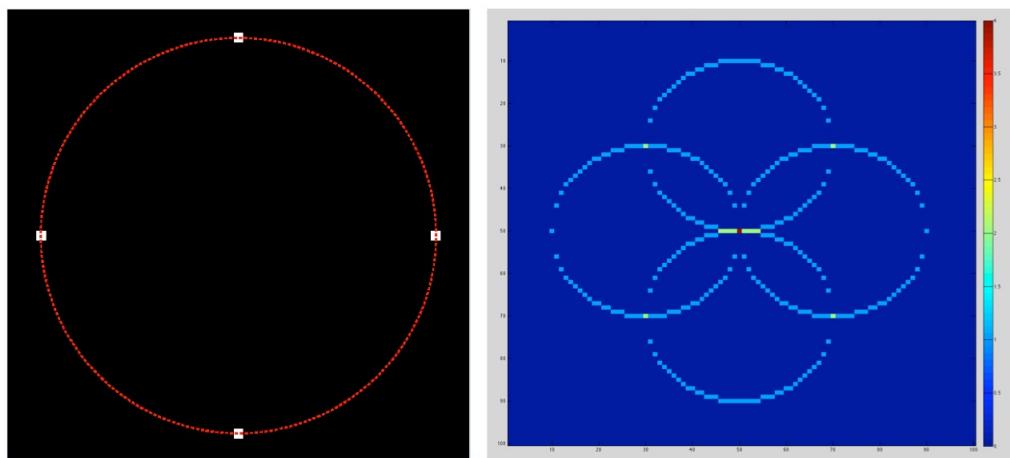
从 Hough 变换的基本原理看出, Hough 变换采用不同空间的映射完成特殊形状的检测, 但是带来了巨大的运算量, 为进一步降低运算量首先对图片进行二值化处理并完成边检测。对处理后的结果再进行 Hough 变换。

- 轮廓检测(contours)

对于一些更为复杂的形状检测, OpenCV 提供的轮廓检测函数 `findContours()` 及轮廓处理函数 `drawContours()` 来进行检测和识别。该检测函数主要针对二值图像, 因此, 在函数调用之前需要采用 `threshold()` 函数对图片进行二值化处理, 下面是一个轮廓检测的例子。

---

<sup>1</sup>核函数: 核函数是机器学习中将二维特征映射到高维空间中时采用的手段, 使低维上不可分的特征在高维空间展开后可区分。可以结合代数的知识以及 4.2 将其理解为一个映射或运算即可。具体的数学推导比较复杂, 对于应用实践来说是不必要的, 感兴趣的同学可以了解支持向量机相关理论。

图 4.3: 直线的 Hough 变换<sup>3</sup>图 4.4: 圆形 Hough 变换<sup>3</sup>

```

1 contours,hierarchy = cv.findContours(thresh, 1, 2)
2 cnt = contours[0]
3
4 #the moment return a dic
5 M = cv.moments(cnt)
6 print( M )
7
8 # the centric of the contour
9 cx = int(M['m10']/M['m00'])
10 cy = int(M['m01']/M['m00'])
11
12 # the area of the contour
13 area = cv.contourArea(cnt)
14
15 # the Perimeter of the contour
16 perimeter = cv.arcLength(cnt,True)

```

在图像的轮廓被检测出来后，轮廓的面积、周长、中心点等是描述轮廓的重要参量，另外在轮廓的边

界被识别出来后通常用一个标准的形状(矩形、圆形、椭圆等)来标识轮廓。OpenCV 中也提供了相应的`moments()`、`contourArea()`、`arcLength()`函数来计算轮廓的相关参量。

## 4.3 动手实验:图片处理及图形识别

### 4.3.1 图片的简单的处理

可以使用树莓派自带的命令用摄像头拍摄一张自己校园卡或身份证件的照片:

```
1 $ libcamera-jpeg -o pic.jpeg
```

利用 OpenCV 对该照片进行一些简单的操作如颜色变换、矫正处理。Perspective Transformation 是 OpenCV 中的一个功能,它允许你将图像从一个透视投影转换到另一个透视投影,常用于校正图像的透视畸变或者提取特定区域的图像<sup>4</sup>)。

使用透视变化需要先定义原始图像中的四个顶点坐标以及目标图像中的对应顶点坐标:

```
1 import cv2 as cv
2 import numpy as np
3 original_points = np.float32([[x1, y1], [x2, y2], [x3, y3], [x4, y4]])
4 target_points = np.float32([[x1_prime, y1_prime], [x2_prime, y2_prime],
5     [x3_prime, y3_prime], [x4_prime, y4_prime]])
6 # 计算透视变换矩阵:
7 perspective_matrix = cv.getPerspectiveTransform(original_points, target_points)
8 # 进行透视变换:
9 output_image = cv.warpPerspective(image, perspective_matrix, (width, height))
```

这里的 width 和 height 是输出图像的尺寸。

### 4.3.2 图片形状的识别

- 使用 Hough 变换来识别照片中的圆形使用 Hough 变换来识别圆形主要有以下几个步骤:
  1. 图像滤波
  2. 转化为灰度图片
  3. 使用 `HoughCircles()` 来寻找圆形
  4. 在图片中将圆形标记出来

参考代码如下所示:

```
1 import numpy as np
2 import cv2 as cv
3 img = cv.imread('opencv-logo-white.png',0)
4 img = cv.medianBlur(img,5)
5 cimg = cv.cvtColor(img,cv.COLOR_GRAY2BGR)
6 circles = cv.HoughCircles(img,cv.HOUGH_GRADIENT,1,20,
7                             param1=50,param2=30,
8                             minRadius=0,
9                             maxRadius=0)
10 circles = np.uint16(np.around(circles))
11 for i in circles[0,:]:
```

```

10     # draw the outer circle
11     cv.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
12     # draw the center of the circle
13     cv.circle(cimg,(i[0],i[1]),2,(0,0,255),3)
14 cv.imshow('detected circles',cimg)
15 cv.waitKey(0)
16 cv.destroyAllWindows()

```

- 使用`findContours()`函数来识别轮廓

使用`findContours()`函数来识别图片中的轮廓，并在原图片中将识别出的结果标记出来，另外计算轮廓的中心、周长、面积等参数。需要注意的是`findContours()`需要对图片进行二值化处理。参考代码如下所示：

```

1 import numpy as np
2 import cv2 as cv
3
4 # find the contours
5 im = cv.imread('test.jpg')
6 imggray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
7 ret, thresh = cv.threshold(imggray, 127, 255, 0)
8 contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.
    CHAIN_APPROX_SIMPLE)
9
10 # draw the contours, in green color
11 cnt = contours[4]
12 cv.drawContours(img, [cnt], 0, (0,255,0), 3)

```

### 4.3.3 注意事项

#### 1. Numpy 库的使用

Numpy 库主要用于 Python 数组的运算，也是 OpenCV 关于图片处理的基础。Numpy 数组的主要属性有 `ndim`,`shape`,`size`,`dtype`,`itemsize` 等，Numpy 提供了丰富的处理函数及矩阵运算函数，如`arange()`,`zeros()`,`rev()`,`reshape()`,`sort()` 等。Numpy 库的使用可以参考 [5] 中获得帮助。

#### 2. 数据位数扩展

由于图片中各像素点采用 `uint8` 表示，但是当做滤波等运算的时候为保持计算精度，通常要对相应的像素点进行数据位的扩展，如将其转化成 64 位的浮点数等，在计算完成后再转化为 `uint8`。

#### 3. OpenCV 中性能的评估

为评估算法的运算性能，OpenCV 中也提供了对于的函数如`getTickCount()`,`getTickFrequency()`等通过获取计算的时间来评估算法的性能。

## 4.4 思考题

- 对图片进行滤波处理与信号处理中的滤波有何异同？其核函数如何设计？
- Hough 变换的基本原理是什么？其他领域中有没有类似的处理方法？

# 第五章 支持向量机分类器

本章我们将使用支持向量机实现手写数字的识别，识别的图像和最终的结果将使用树莓派的 OLED 显示屏来显示。图像识别所使用的数据集采用公开数据，而通过摄像头识别数字的任务将留给读者完成。

## 5.1 支持向量机

对于给定的样本集，分类算法的目的就是要找到一个划分超平面，将不同类别的样本分开，但能实现这个目的的超平面可能有很多，要如何确定最好的那一个呢？图5.1是一个二维的例子，可以有不同的直线将两类样点分开。直观的看粗实线的分类效果最好，样点到直线的最短距离是最大，而且对于不同的样点相等。

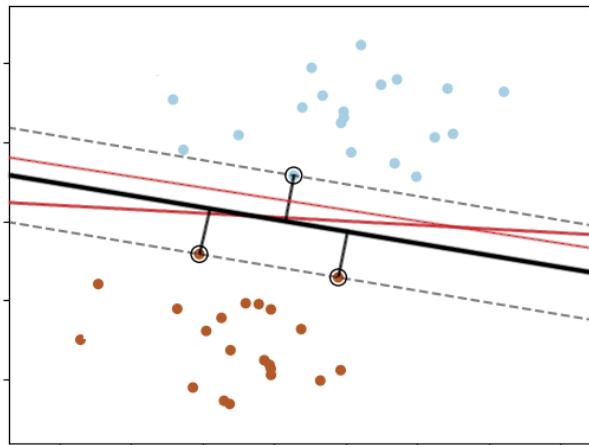


图 5.1: 支持向量示例

支持向量机(Support Vector Machine)就是利用这一数学特征寻找最优超平面的算法，距离超平面最近的几个训练样本(图5.1 中圈出的三个点)被称为支持向量。最后找到的超平面是可只根据支持向量来确定，因此 SVM 方法往往在样点数比较少的时候表现也很好，训练速度也比较快。

### 5.1.1 支持向量机的基本型

对于样本数据集  $(\mathbf{x}_i, y_i), y_i \in +1, -1$ ，划分超平面可以表示为：

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0 \quad (5.1)$$

样本空间中任意点到该超平面的距离可以写为：

$$r = \frac{\mathbf{w}^T \mathbf{x} + b}{\|\mathbf{w}\|} \quad (5.2)$$

如果这个超平面可以正确的对样本进行分类, 可以令:

$$\begin{cases} \mathbf{w}^T \mathbf{x} + b \geq +1, & y_i = +1; \\ \mathbf{w}^T \mathbf{x} + b \leq -1, & y_i = -1. \end{cases} \quad (5.3)$$

其中使等号成立的样点就是支持向量, 两个异类支持向量到超平面的距离和为:

$$\gamma = \frac{2}{\|\mathbf{w}\|} \quad (5.4)$$

寻求最佳分割超平面的过程就是最大化  $\gamma$  的优化问题, 即:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2, \quad \text{s.t. } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, m \quad (5.5)$$

利用拉格朗日乘子法可以得到这个问题的等价问题, 其拉格朗日函数可以写为:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)), \quad (\alpha_i \geq 0) \quad (5.6)$$

令  $L(\mathbf{w}, b, \alpha)$  对  $\mathbf{w}$  和  $b$  的偏导为零可以解出  $\alpha_i$  的值, 同时由于约束条件为不等式,  $\alpha_i$  还必须满足 KKT (Karush-Kuhn-Tucker) 条件, 即:

$$\begin{cases} \alpha_i \geq 0; \\ y_i f(x_i) - 1 \geq 0; \\ \alpha_i (y_i f(x_i) - 1) = 0. \end{cases} \quad (5.7)$$

于是对于任意样本  $(x_i, y_i)$ , 若  $\alpha_i = 0$  则该样本对  $f(x_i)$  没有影响, 若  $\alpha_i > 0$  则必有  $y_i f(x_i) = 1$ , 即样点在最大间隔边界上, 是一个支持向量。

### 5.1.2 软间隔

如果训练样本并不是线性可分, 前面介绍的算法就会失效, 而缓解这一问题的一个方法就是采用“软间隔”, 允许部分样本不满足约束条件  $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$ , 如图 5.2 所示, 虽然在部分样本上会分类错误, 但能获得一个可行的分类方案。

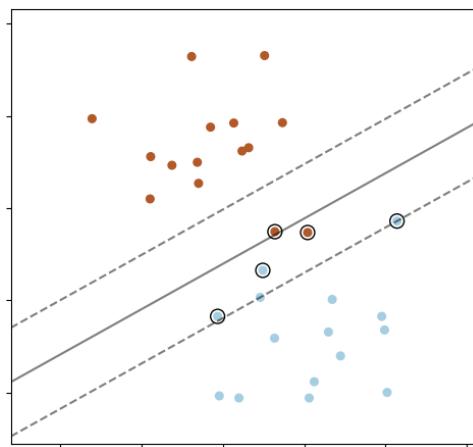


图 5.2: 软间隔示意图

此时的优化目标为

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m l_{0/1}(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1) \quad (5.8)$$

其中  $l_{0/1}$  定义如下, 被称为“0/1 损失函数”:

$$l_{0/1}(z) = \begin{cases} 1, & \text{if } z < 0; \\ 0, & \text{otherwise.} \end{cases} \quad (5.9)$$

其中  $C > 0$  是一个常数, 其取值越小, 约束对优化目标的贡献越小, 可以有更多的样点不满足约束条件, 而当  $C$  取值为无穷大时, 优化目标等效于“硬间隔”, 所有样点都要满足约束条件。

然而  $l_{0/1}$  函数非连续、非凸, 不易进行数学分析。于是通常会用一些其他函数进行替代, 常用的替代损失函数有:

- hinge 损失:  $l_{hinge}(z) = \max(0, 1 - z)$
- 指数损失:  $l_{exp}(z) = \exp(-z)$
- 对数损失:  $l_{log}(z) = \log(1 + \exp(-z))$

### 5.1.3 核函数

对于训练样本线性不可分的情况, 还可以将样本从原始空间映射到一个更高维的特征空间, 使得这些样本在新的特征空间内线性可分。数学上可以证明如果原始空间是有限维度, 那么一定存在一个高维特性空间使得样本线性可分。令  $\phi(\mathbf{x})$  为映射函数, 则变换后的模型可表示为:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \quad (5.10)$$

在通过拉格朗日方程求解的过程中, 会涉及到高维空间的内积计算, 但由于特征空间的维数可能很高, 甚至是无穷维, 直接计算内积通常很困难, 因此定义如下的核函数:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \quad (5.11)$$

有了核函数就可以不必设计  $\phi()$  函数, 因为仅靠核函数就可以完成计算并且根据下面的定理可以更容易的设计核函数。

令  $\chi$  为输入空间,  $\kappa()$  为定义在  $\chi \times \chi$  上的对称函数, 则  $\kappa$  是核函数当且仅当对于任意数据集  $D = \mathbf{x}_1, \dots, \mathbf{x}_m$ , “核矩阵” $\mathbf{K}$  总是半正定的。

$$\mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_j) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_i, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_i, \mathbf{x}_j) & \dots & \kappa(\mathbf{x}_i, \mathbf{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_m, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_m, \mathbf{x}_j) & \dots & \kappa(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix} \quad (5.12)$$

常用的核函数有如下几个:

- 线性核  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- 多项式核  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^d$ ,  $d \geq 1$  为多项式的次数
- 高斯核  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$ ,  $\sigma > 0$  为高斯核的带宽
- 拉普拉斯核  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma}\right)$ ,  $\sigma > 0$
- Sigmoid 核  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta \mathbf{x}_i^T \mathbf{x}_j + \theta)$

## 5.2 SPI OLED 显示模块

### 5.2.1 SPI 总线传输原理

SPI (Serial Peripheral Interface) 接口标准广泛用于微处理器与外部设备之间的数据交互, 如一些传感器芯片、控制芯片、LCD 等。该标准最早由 Motorola 公司提出, 目前由 NXP 公司进行标准的维护。

SPI 接口采用主从方式进行全双工的数据传输, 其引脚包含四条信号线, 具体定义如表 5.1 所示。

表 5.1: SPI 引脚定义

| 名称   | 位宽 | 功能描述    | 备注                     |
|------|----|---------|------------------------|
| SCLK | 1  | 串口时钟    | 由主设备(Master)提供         |
| MOSI | 1  | 主设备数据输出 | Master Output Slave In |
| MISO | 1  | 从设备数据输出 | Master In Slave Output |
| SS   | 1  | 从设备片选信号 | Slave Select           |

### 5.2.2 SPI 连接模式

典型主从设备连接如图5.3 所示。

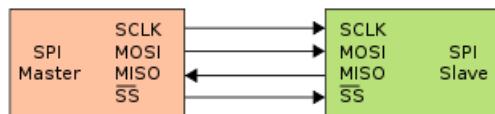


图 5.3: SPI 典型主从设备连接方式

SPI 主设备支持多个 SPI 从设备连接,主要有两种方式:并行连接及链式连接。并行连接方式 SPI 需要多个片选信号线区别多个从设备;链式链接仅需要一根片选线,从设备的输入连接到下一个从设备的输入。

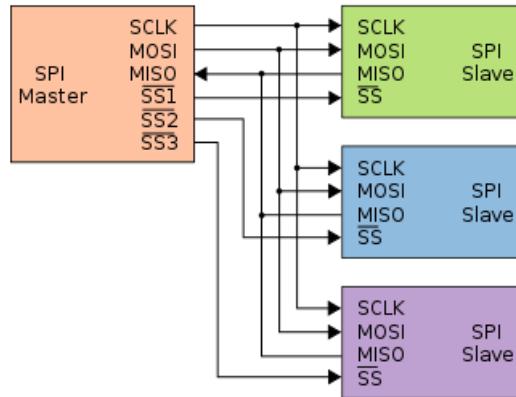


图 5.4: SPI 主从设备并行连接方式

### 5.2.3 SPI 传输方式配置

如图5.6 所示,当有数据传输时,SPI 主设备产生串行时钟,通过移位寄存器将数据逐 bit 进行传输,根据所配置的传输模式,从设备进行数据采样,完成数据接收。

时钟极性(CPOL)和时钟相位(CPHA)用于设定从设备何时采样数据,图 5.7 和 5.8 给出了不同配置下的数据采样位置。两种传输模式中, SCK 与数据的相对位置不同,因此,在两种模式下对于数据采样位置有不同要求,CPHA=1 模式下,采样位置比 SCK 晚半个时钟周期。

SPI 波特率时钟由波特率控制寄存器决定,波特率由如下公式决定,具体算法需要参考处理器的芯片手册:

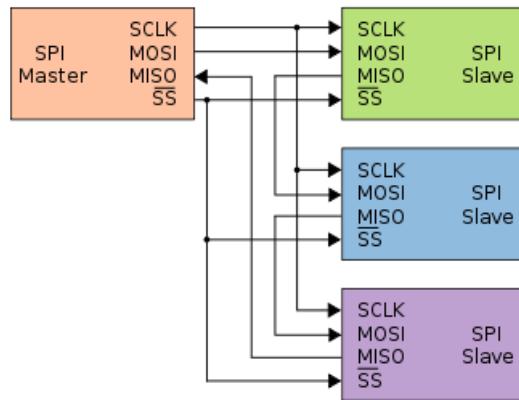


图 5.5: SPI 主从设备链式连接方式

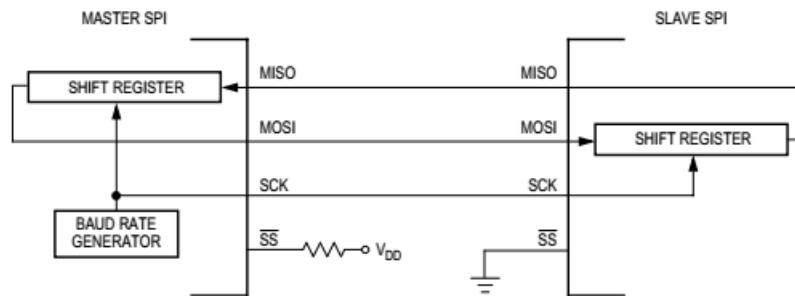


图 5.6: SPI 传输

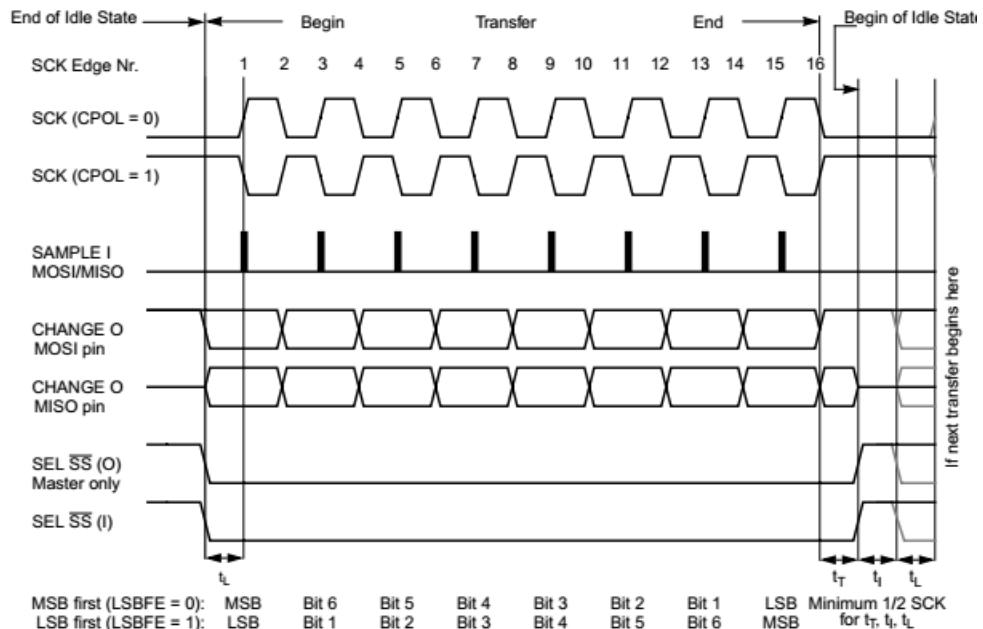


图 5.7: CPHA=0 SPI 传输时序图

$$\text{BaudRate} = \frac{\text{BusClock}}{\text{BaudRateDivisor}}$$

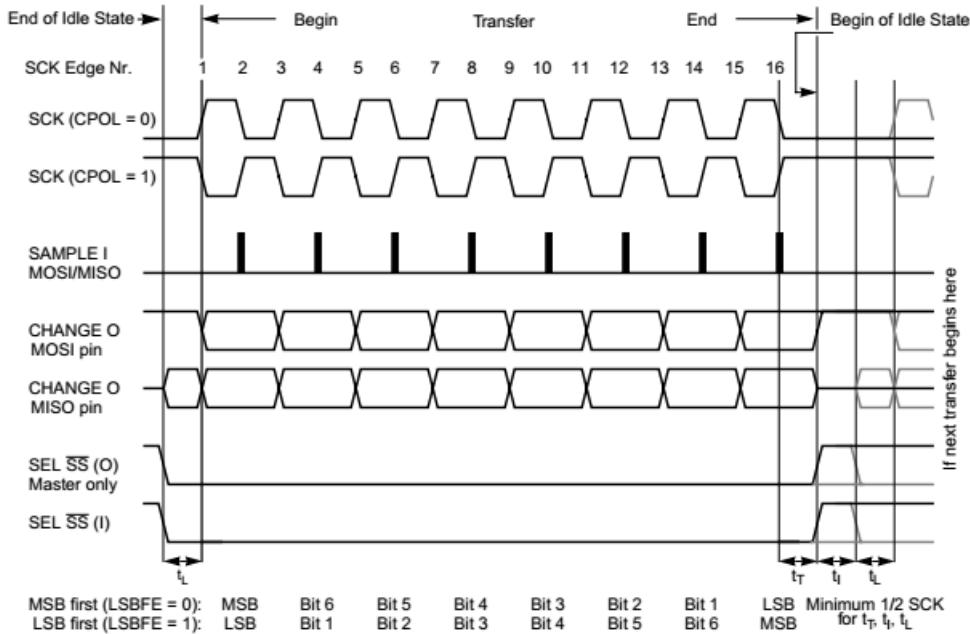


图 5.8: CPHA=1 SPI 传输时序图

#### 5.2.4 树莓派的 SPI 接口

不同的树莓派版本可能包含不同数量的 SPI 接口，其中所有树莓派都支持 SPI0 接口，其具体定义如表 5.2 所示，注意这里的 SS 官脚被称为 CE(Chip Enable)，因此 SPI0 接口包含两个片选信号，可以连接两个 SPI 从设备。

表 5.2: 树莓派 SPI0 管脚定义

| 管脚定义 | 接口编号 | 管脚名称   | 管脚功能       |
|------|------|--------|------------|
| MOSI | 19   | GPIO10 | SPI0_MOSI  |
| MISO | 21   | GPIO09 | SPI0_MISO  |
| SCLK | 23   | GPIO11 | SPI0_SCLK  |
| CE0  | 24   | GPIO08 | SPI0_CE0_N |
| CE1  | 26   | GPIO07 | SPI0_CE1_N |

除了第一代的树莓派，其它版本还支持 SPI1 接口，具体定义如表 5.3 所示，它支持三个片选信号。

表 5.3: 树莓派 SPI1 管脚定义

| 管脚定义 | 接口编号 | 管脚名称   | 管脚功能       |
|------|------|--------|------------|
| MOSI | 38   | GPIO20 | SPI1_MOSI  |
| MISO | 35   | GPIO19 | SPI1_MISO  |
| SCLK | 40   | GPIO21 | SPI1_SCLK  |
| CE0  | 12   | GPIO18 | SPI1_CE0_N |
| CE1  | 11   | GPIO17 | SPI1_CE1_N |
| CE3  | 36   | GPIO16 | SPI1_CE2_N |

而对于树莓派 4 来说,它还支持 SPI3、SPI4、SPI5 和 SPI6,具体的管脚定义不在这里列出,感兴趣的读者可以参考数据手册<sup>6</sup>。

### 5.2.5 OLED 设备及访问

动手实验中采用  $128 \times 64$  点阵的 OLED 显示屏,使用 SSD1306 驱动芯片驱动。SSD1306 支持多种数据接口类型如 8bit 68xx/80xx 并行数据接口,SPI 数据接口、I<sup>2</sup>C 数据接口。具体支持接口类型由相关寄存器配置确定,在实验中我们采用 SPI 数据接口类型。

SPI 支持 3 线、4 线接口,其主要区别在于 4 线接口中多余的一根线用来做数据及命令指示,本实验中采用 4 线接口类型。SSD1306 与树莓派的主要连接如图 5.9 所示,4 线接口的时序如图 5.10 所示。其中 CS# 为片选,低电平有效,D/C# 为数据命令指示信号,低电平表示当前传输的内容为命令,高电平表示当前传输的内容为数据。

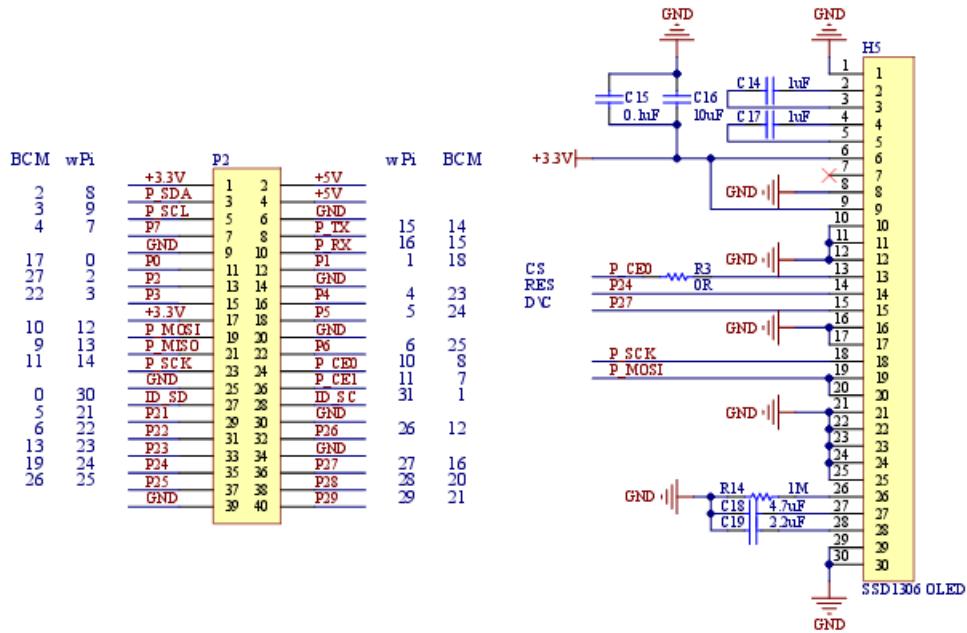


图 5.9: 树莓派与驱动芯片的连接

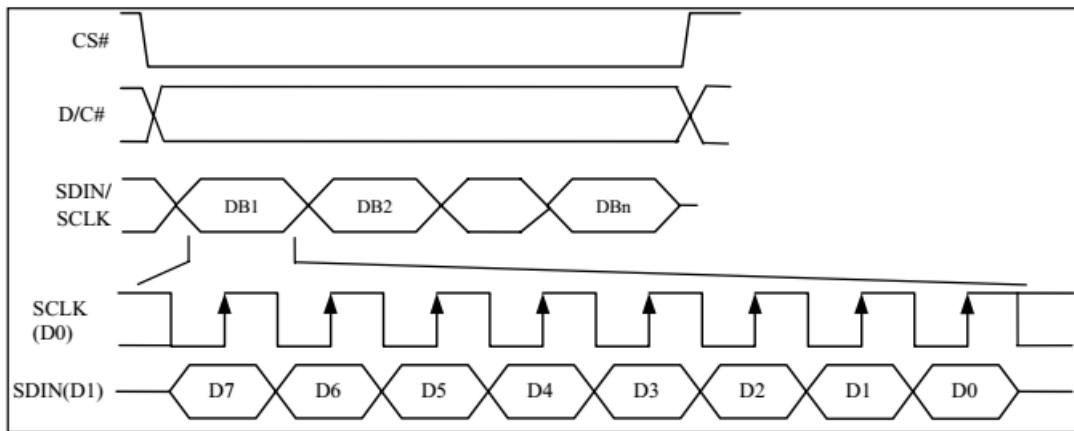


图 5.10: 4 线 SPI 传输时序

显示屏由驱动芯片的列驱动(SEG0–SEG127)和行驱动(COM0–COM63)管脚进行控制,当行驱动有

效时,128 个列驱动管脚用来决定当前行的显示,因此显示屏的显示内容会一行一行的依次更新,直到整个屏幕内容更新完毕。控制器内部通过一个存储器保存当前的显示内容,由于 SSD1306 驱动的是单色显示模块,因此每个点只需要一个比特信息进行保存,一共需要  $128 \times 8$  个字节的存储空间,这些存储空间分成了 8 个页(从 PAGE0 到 PAGE7),每页存储器保存 8 个行的显示内容。具体如图 5.11 所示。

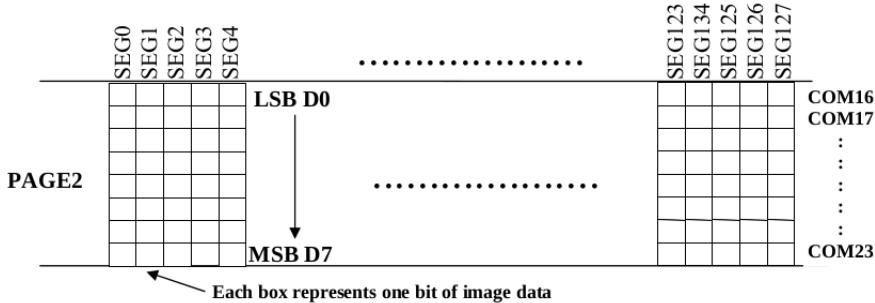


图 5.11: SSD1306 的 PAGE2 内存布局

图 5.11 展示的是显示模块第 16 到第 23 行的显示内容,其中每个小方格代表一个像素点。在 SSD1306 中有三种地址模式: 页地址模式, 水平地址模式和垂直地址模式, 模块可以通过 SPI 接口写入控制命令, 其中控制地址模式的命令为 0x20, 具体方式是通过 SPI 接口发送命令字 0x20, 然后发送命令参数, 0x00 代表水平地址模式, 0x01 代表垂直地址模式, 0x10 代表页地址模式(复位缺省值)。

- 在水平地址模式下,每次写入点阵数据后“列地址”自动加 1,准备好下一个存储地址,从而实现连续更改显示内容的功能。在此模式下写完一个页会自动换到下一个页的首地址,写完全部的页会自动换到第一页首地址。
- 在垂直地址模式下,每次写入点阵数据后“页地址”自动加 1,写完全部页后切换到第一页,同时“列地址”自动加 1,写完最后一列会自动换到第一列地址。此模式下可以按列更新显示内容,而水平模式是按行更新内容。
- 在页地址模式下,写入模式与水平地址模式类似,只是页地址不再更新。

完整的 SSD1306 命令列表可以参考其数据手册,这里不再举例。

## 5.3 用到的 Python 库

### 5.3.1 spidev

spidev 库利用 Linux 内核的 spidev 驱动来操作 SPI 总线。下面的示例通过 SPI 总线发送三个字节的数据。

```

1 import spidev
2 spi = spidev.SpiDev()      # 新建 SPI 设备类
3 spi.open(bus, device)     # 打开特定的 SPI 设备
4 to_send = [0x01, 0x02, 0x03]
5 spi.writebytes(to_send)    # 发送上面定义的三个字节

```

示例中的 bus 是总线号,对于树莓派就是 SPI 的编号,device 是设备号,对应总线上的片选号,因此 `spi.open(0,0)` 就代表 SPI0 的 CE0 片选对应的 SPI 设备。发送数据采用 `writebytes` 函数,而接收数据采用 `readbytes` 函数(函数参数为读取的字节数)。由于 SPI 协议采用全双工方式通信,发送数据的同时也可以进行接收数据,此时可以用 `xfer` 函数。`readbytes` 函数和 `xfer` 函数的用法示例如下:

```
1 ...
2     dat = spi.readbytes(n)
3     buf = spi.xfer([0x06, 0x00, 0x03])
```

### 5.3.2 scikit-learn 中的 SVM

scikit-learn 是一套包含许多机器学习算法的 Python 库，包括前面介绍 KMeans 还有这节课介绍的 SVM。这里先看它的一个 SVM 示例代码。

```
1 import matplotlib.pyplot as plt
2 from sklearn import datasets, svm, metrics
3
4 digits = datasets.load_digits() # 手写数字数据集为 8x8 的图片，16 级灰度
5
6 # digits.images 是图片数据，digits.target 是标签
7 images_and_labels = list(zip(digits.images, digits.target))
8 # 使用 matplotlib 绘制前4个样本
9 for index, (image, label) in enumerate(images_and_labels[:4]):
10     plt.subplot(2, 4, index + 1)
11     plt.axis('off')
12     plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
13     plt.title('Training: %i' % label)
14
15 n_samples = len(digits.images)
16 # 将二维数据变成一维
17 data = digits.images.reshape((n_samples, -1))
18
19 # 建立分类器
20 classifier = svm.SVC(gamma=0.001)
21
22 # 用前一半数据进行训练
23 classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])
24
25 expected = digits.target[n_samples // 2:]
26 # 预测
27 predicted = classifier.predict(data[n_samples // 2:])
28
29 print("Classification report for classifier %s:\n%s\n"
30       % (classifier, metrics.classification_report(expected, predicted)))
31 print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))
32
33 # 使用 matplotlib 绘制前4个预测样本
34 images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
35 for index, (image, prediction) in enumerate(images_and_predictions[:4]):
36     plt.subplot(2, 4, index + 5)
37     plt.axis('off')
38     plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
39     plt.title('Prediction: %i' % prediction)
```

```
40
41     plt.show()
```

这段代码的功能是将手写数字的图像进行识别。dataset 类中包含手写数字的图像和标签，程序使用前一半数据进行训练，然后用后一半对训练结果进行验证，并使用 matplotlib 显示了部分训练数据和训练数据。程序的功能可以通过所带的注释大致了解，部分函数的说明可以参考网上 scikit-learn 的文档。

### 5.3.3 Python 图像支持库

Python 有很多可以进行图像处理的支持库，其中 PIL (Python Image Library) 是历史比较悠久的一个，使用的人很多。但 PIL 仅支持 Python2，一个它的分支 Pillow 完成了对 Python3 的支持。PIL 中最重要的类就是 Image，使用它可以打开已有的图片或创建新的图片进行后续处理。下面的代码可以获取已有图像的基本信息。

```
1   from PIL import Image # 注意依然使用 PIL
2
3   image = Image.open("example.jpg")
4   print('width:', image.width)
5   print('height:', image.height)
6   print('size:', image.size)
7   print('mode:', image.mode)
8   print('format:', image.format)
```

width 表示图片的像素宽度，height 表示图片的像素高度，width 和 height 组成了 size 属性，size 是包含两个元素的元组。mode 属性表示图片的模式，表 5.4 是常见的几种图像模式。

表 5.4: PIL 颜色模式

| 模式   | 描述                 |
|------|--------------------|
| 1    | 1 位像素，黑白图片         |
| L    | 8 位像素，灰度图片         |
| P    | 8 位像素，使用调色板定义颜色    |
| RGB  | 24 位像素，真彩色         |
| RGBA | 32 位像素，真彩色加透明度模板   |
| CMYK | 32 位像素，色彩分离(常用于印刷) |
| HSV  | 24 位像素，色相、饱和度、颜色空间 |

不同的模式之间可以用 convert 函数转换，例如下面的代码将图像转为灰度图片。

```
1   from PIL import Image
2
3   image = Image.open("example.jpg")
4   image1 = image.convert('L')
```

图片的裁剪和缩放可以用 crop 和 resize 函数，例如：

```
1   from PIL import Image
2
3   image = Image.open("example.jpg")
```

```

4     image_crop = image.crop(box=(100,100,800,600))
5     image_resize = image.resize((400, 300))

```

其中 `crop` 函数通过 `box` 参数表示裁剪的区域, 元组的四个参数分别是左上和右下的像素坐标(图片的左上角坐标为 $(0,0)$ )。

新建一个图片可以用 `new` 函数, 利用 `ImageDraw` 类提供的函数可以进行简单图形的绘制。例如:

```

1  from PIL import Image, ImageDraw
2
3  image = Image.new('RGB', (400, 300), 'black')
4  draw = ImageDraw.Draw(image)
5  draw.line((0, 0, 400, 300), fill='red') # 画一条红色对角线
6  image.show() # 调用外部程序显示图片

```

`new` 函数的三个参数分别是图像的模式、尺寸和颜色, 上面程序的执行结果是在黑色的背景下绘制了一条红色直线。

如果要在图片上生成字符串, 可以使用 `ImageFont` 类选择字体, 使用 `text` 函数绘制, 如下代码所示:

```

1  from PIL import Image, ImageFont, ImageFont
2
3  image = Image.new('RGB', (400, 300), 'black')
4  font = ImageFont.load_default()
5  draw = ImageDraw.Draw(image)
6  draw.text((0,0), 'This is first line', font=font, fill=255)

```

### 5.3.4 SPI 与 OLED 的支持

树莓派与驱动芯片之间采用了 SPI 接口, 同时需要对树莓派的相关管脚属性进行配置。下面给出了访问 OLED 的 Python 示例, 例子中对 SSD1306 做了类封装, 其中定义了一些 OLED 基本屏幕操作, 如 `__init__()`, `command(self, cmd)`, `data(self, val)` 等函数, 实验中通过调用函数来实现 OLED 的显示。

例如 `command` 命令用来向 SSD1306 发送命令, 具体代码如下:

```

1  def command(self, cmd):
2      GPIO.output(self._dc,GPIO.LOW)      # _dc 低电平表示输出命令
3      self._spi.writebytes([cmd])        # 通过 spi 输出具体命令内容

```

下面的示例代码使用 SSD1306 类显示了一个图片, 可以作为实验的参考。

```

1  import time
2  import spidev as SPI
3  import SSD1306
4  from PIL import Image # 调用相关库文件
5
6  RST = 19
7  DC = 16
8  bus = 0
9  device = 0 # 树莓派管脚配置
10 disp = SSD1306.SSD1306(rst=RST,dc=DC,spi=SPI.SpiDev(bus,device))
11
12 disp.begin()
13 disp.clear()

```

```
14 disp.display() # 初始化屏幕相关参数及清屏  
15  
16 image = Image.open('pku.png').resize((disp.width, disp.height),  
17     Image.ANTIALIAS).convert('1')  
18 disp.image(image)  
19 disp.display() # 显示图片
```

## 5.4 动手实验:手写数字识别

### 5.4.1 实验目的

1. 了解 SPI 总线传输的原理。
2. 了解 OLED 设备基本原理。
3. 掌握支持向量机的基本原理

### 5.4.2 实验步骤

#### 5.4.2.1 OLED 设备测试

1. 通过 SPI 设备访问 OLED, 在显示屏上显示 “hello world!”。
2. 使用 OLED 显示模块显示一幅图片

#### 5.4.2.2 验证支持向量机示例代码

将示例代码在树莓派上运行, 查看运行结果, 理解程序输出的评价指标。试着修改使用不同的核函数, 查看运行结果。

#### 5.4.2.3 使用 OLED 显示结果

将预测的图像和预测结果都显示在 OLED 上, 每次显示一副图像, 通过按键控制显示下一副图像。

由于 OLED 显示屏是单色显示, 分辨率也比提供的图像高, 因此需要利用 Image 库中的 resize 函数转换一下, 示例如下:

```
1 # kk 中保存了图像的原始数据  
2 digit = Image.fromarray((kk*8).astype(np.uint8), mode='L').resize((48,48)).  
3     convert('1')  
4 img = Image.new('1',(disp.width,disp.height),'black')  
5 img.paste(digit, (0, 16, digit.size[0], digit.size[1]+16))  
6 disp.clear()  
7 disp.image(img)  
8 disp.display()
```

#### 5.4.2.4 使用摄像头读入手写数字

在白纸上写一位数字, 通过摄像头读入, 并处理成符合算法输入的格式, 通过训练好的模型进行识别。

## 5.5 思考题

1. 为什么不把全部的数据都用来做训练呢？然后在训练数据集上进行验证效果一定会更好！
2. 将具有 16 级灰度的图像显示在单色屏上，能保证显示效果的原理是什么？

# 第六章 构建专家系统

本章我们将使用树莓派实现一个简单的专家系统，这个系统可以通过动物所具有的特征识别出动物的名称。在这个过程中，我们将学习专家系统的基本知识，学会使用 Prolog 语言编写简单的动物识别逻辑。最后我们使用 Python 语言编写用户的交互界面，并通过舵机将识别的结果输出。

## 6.1 专家系统

### 6.1.1 专家系统简介

所谓的专家系统(Expert System)是一种利用知识进行推理的程序。专家系统具有某些领域的知识的规则库(rules)，例如鸟类识别专家系统关于几种鸟类的规则库可能是

喜鹊是留鸟、颜色黑白、体型中等  
白鹭是旅鸟、颜色白、体型大  
翠鸟是留鸟、颜色蓝、体型小

当给出一些事实(facts)，专家系统可以利用它们进行推理，得到特定的结论。例如给出下面的查询条件：

居留类型：留鸟  
颜色：蓝  
体型：小

专家系统就可以得出查询的鸟是翠鸟的答案。

当然前面的示例过于简单，并不能体现专家系统的优点。一个复杂的规则库和完备的推理程序可以完成只有人类专家才可以做到的任务，这类任务往往规则库非常复杂，人类需要多年的学习和积累经验才能完成，这也是为什么这类软件被称为专家系统的原因。常用的应用方向有工程、医药、军事等。

编写专家系统一般需要特殊的编程语言，这类语言具有逻辑推理机制，可以提高编程的效率，减少程序出错的概率。其中比较有代表性的就是 Prolog。

### 6.1.2 Prolog 编程语言

Prolog 是一种逻辑型程序设计语言，它的英文就是 Programming in Logic 的缩写。它具有自动搜索、模式匹配、回溯等性能，主要特点是以谓词逻辑为理论基础。用 Prolog 编程时，人们注重和关心对问题的描述，而不是问题的求解过程。例如前面小节中的规则库和事实库可以用 Prolog 描述如下：

```
bird(magpie) :- color(black_white), season(all_year), size(medium).  
bird(egret) :- color(white), season(spring_autumn), size(large).  
bird(kingfisher) :- color(blue), season(all_year), size(small).  
season(all_year).  
color(blue).  
size(small).
```

Prolog 的这种编程设计思想被称为声明式编程语言(Declarative language)，其它如 C 语言是命令式编程语言(Imperative language)。命令式编程语言通过具体的流程控制来描述程序的执行过程，而声明

式语言采用更高层的方式对问题进行描述，只表达出要达到的目标而由运行环境完成对目标问题的计算过程。

在 Prolog 的程序中，首字母小写的单词表示函数或者常量（Prolog 术语叫做 predicate，中文一般翻译为“谓词”，本文后续都用函数来指代），不用声明就可以直接使用。Prolog 每行语句由句点结束，包含 `:=` 的表示规则，代表蕴含的关系，例如 `bird(magpie)` 为真的条件是 `:=` 后面的三个函数都为真。`:=` 后面的内容用逗号分隔表示逻辑的与关系；如果采用分号分隔则表示逻辑或的关系；逻辑或的关系也可以写成独立的两个语句。不包含 `:=` 的语句为事实，例如 `color(blue)` 这个函数永远为真。

首字母大写的单词在 Prolog 中是变量，如果出现在程序中就组成了一个查询语句。例如对于上面的程序，`bird(X)` 就是一条查询，而根据前面定义的规则和事实，唯一符合条件的查询就是 `X = kingfisher`。

下划线 `_` 是一个特殊的变量，被称为匿名变量，它的作用域仅限于当前语句，也就是说程序中不同语句所使用的匿名变量没有关系。匿名变量在使用的时候往往只关注是否这个变量可以取到值，而不关心这个值到底是什么。因此对于上面的程序，`bird(_)` 给出的结果是 `true`，即表示这个变量可以取到值。

### 6.1.3 Prolog 的回溯算法

从算法的角度来看，Prolog 的解释器或编译器需要在设定的事实和规则的前提下，在定义域内搜索特定问题全部的解。这是一个被广泛研究的计算问题，一般通过回溯算法（深度优先搜索）的算法来获得答案。了解回溯的基本过程可以为编写高效率代码提供指导。Prolog 也提供了一些语法来优化搜索的过程。

例如定义一个求最大值的函数 `max(X,Y,Z)`，其中 `Z` 是前两个变量的最大值。可以这样写这个程序：

```
max(X,Y,Y) :- X <= Y.
max(X,Y,X) :- X > Y.
```

假定查询语句是 `max(2,3,Y)`，那么通过第一条规则就可以得出 `Y=3`，可 Prolog 还会去检查第二条规则试图找到其它的答案，而这显然是徒劳的。因为程序设计者了解这个函数只能有一个答案，如果第一条规则满足就不必检查第二条规则，此时可以使用 Prolog 的 cut 功能，用 `!` 符号阻止后续的回溯搜索。具体写法如下：

```
max(X,Y,Y) :- X <= Y, !.
max(X,Y,X) :- X > Y.
```

Prolog 在遇到 `!` 之后，前面搜索过程中的可回溯节点全部不再回溯，包括最开始检查第一条规则的节点，不再会去检查之后的规则了。

对于 `max` 函数，通过一次判断就应该可以得到结论，第二条规则的判断有些浪费。在 Prolog 中，可以用一种特殊的语法表达 `if-then-else` 的结构：`(A -> B ; C)` 表示对 `A` 进行检测，如果正确那么检测 `B`，如果不正确则检测 `C`。此时 `max` 函数可以写为：

```
max(X,Y,Z) :- (X <= Y -> Z = Y ; Z = X).
```

### 6.1.4 Prolog 内置函数

在程序中如果要在屏幕上输出内容，可以使用 Prolog 的 `write()` 函数，例如可以用 `write('Hello World!')` 打印一条信息。`nl` 可以用来打印一个回车符，`tab(X)` 用来打印 `X` 个制表符。`read(X)` 用来从键盘输入，并把输入的内容赋值给 `X` 变量：

```
hello :-
    write('What is your name?'), read(X),
    write('Hello'), tab(1), write(X), nl.
```

使用 `read` 输入时要以句点结束，并注意不要大写首字母（表示一个变量）。

在 Prolog 中，`true` 永远返回正确，而 `fail` 永远返回错误。

大部分的自定义函数具有确定的状态，要么返回真，要么返回假。有时候程序需要一些函数的返回值可以改变，此时需要用 `dynamic` 函数来进行声明，声明的方式有两种：

```
:  
:- dynamic my_predicate/2.  
dynamic(my_predicate/2).
```

上面的声明定义了 `my_predicate` 函数为动态类型，并包含 2 个参数。Prolog 中都在函数名后面加斜线和数字的方式表明函数中的参数个数。

使用 `assert(X)` 或者 `assertz(X)` 函数可以在规则库的尾部中添加一条新的规则或者事实，而 `asserta(X)` 将新的规则添加到规则库的开头位置。`retract(X)` 函数将 X 从规则库中去除。但要注意，使用 `assert` 增加新的规则只适合动态函数，而从文件中加载的规则缺省都是静态函数。对于之前示例中包含了 `color(blue)` 的规则，就无法用 `assert(color(red))` 进行修改，但可以增加新定义的函数，例如 `assert(bill(long))`，这个 `bill` 函数就是动态类型。

### 6.1.5 swipl 简介

swipl 是 Prolog 的一个实现，下面的交互示例假定前面的鸟类识别代码保存在 `bird.pl` 中。查询结束后按 `Ctrl+D` 或者 `halt.` 命令退出程序。

```
$ swipl bird.pl  
Welcome to SWI-Prolog (threaded, 32 bits, version 8.0.2)  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.  
Please run ?- license. for legal details.  
  
For online help and background, visit http://www.swi-prolog.org  
For built-in help, use ?- help(Topic). or ?- apropos(Word).  
  
?- bird(kingfisher).  
true.  
  
?- bird(magpie).  
false.  
  
?- bird(X).  
X = kingfisher.  
  
?- halt.
```

swipl 包含一个 `trace` 命令，可以对代码的每一步搜索进行跟踪，不但便于了解 Prolog 语句执行的原理，也可以帮助分析程序中可能的错误。下面是 `bird(X)` 查询的跟踪结果。在跟踪结束后，可以使用 `notrace` 命令停止后续查询的跟踪。

```
?- trace.  
true.  
  
[trace] ?- bird(X).  
Call: (10) bird(_7862) ? creep  
Call: (11) color(black_white) ? creep  
Fail: (11) color(black_white) ? creep  
Redo: (10) bird(_7862) ? creep  
Call: (11) color(white) ? creep  
Fail: (11) color(white) ? creep  
Redo: (10) bird(_7862) ? creep  
Call: (11) color(blue) ? creep  
Exit: (11) color(blue) ? creep  
Call: (11) season(all_year) ? creep  
Exit: (11) season(all_year) ? creep  
Call: (11) size(small) ? creep  
Exit: (11) size(small) ? creep
```

```
Exit: (10) bird(kingfisher) ? creep
X = kingfisher.
```

其中 Call 事件表示 Prolog 在试图满足一个条件的要求; Fail 事件表示条件不能满足; Redo 事件表示回溯到前一个节点寻找下一个满足条件的值; EXIT 事件表示当前条件已经满足。上面示例的跟踪过程可以描述为:首先试图为 `bird(X)` 找到一个答案,根据第一个规则,检查 `X` 是不是 `magpie`;根据这条规则还要检查 `color` 是否满足;`color` 条件无法通过;回溯到 `bird(X)` 寻找第二个答案,此时根据第二个规则,选择答案为 `egret`;……;经过一系列的搜索过程,最终获得答案为 `kingfisher`。

### 6.1.6 Prolog 代码示例

```
/* animal identification game.
   start with ?- go.      */

go :- hypothesize(Animal),
      write('I guess that the animal is: '),
      write(Animal),
      nl,
      undo.

/* hypotheses to be tested */
hypothesize(cheetah)    :- cheetah, !.
hypothesize(tiger)      :- tiger, !.
hypothesize(giraffe)    :- giraffe, !.
hypothesize(zebra)      :- zebra, !.
hypothesize(ostrich)    :- ostrich, !.
hypothesize(penguin)    :- penguin, !.
hypothesize(albatross)  :- albatross, !.
hypothesize(unknown).           /* no diagnosis */

/* animal identification rules */
cheetah :- mammal,
          carnivore,
          verify(has_tawny_color),
          verify(has_dark_spots).
tiger :- mammal,
        carnivore,
        verify(has_tawny_color),
        verify(has_black_stripes).
giraffe :- ungulate,
          verify(has_long_neck),
          verify(has_long_legs).
zebra :- ungulate,
        verify(has_black_stripes).

ostrich :- bird,
          verify(does_not_fly),
          verify(has_long_neck).
penguin :- bird,
          verify(does_not_fly),
          verify(swims),
          verify(is_black_and_white).
albatross :- bird,
            verify(appears_in_story_Ancient_Mariner),
            verify(flys_well).

/* classification rules */
```

```

mammal      :- verify(has_hair), !.
mammal      :- verify(gives_milk).
bird        :- verify(has_feathers), !.
bird        :- verify(flys),
            verify(lays_eggs).
carnivore   :- verify(eats_meat), !.
carnivore   :- verify(has_pointed_teeth),
            verify(has_claws),
            verify(has_forward_eyes).
ungulate    :- mammal,
            verify(has_hooves), !.
ungulate    :- mammal,
            verify(chews_cud).

/* how to ask questions */
ask(Question) :-
    write('Does the animal have the following attribute: '),
    write(Question),
    write('? '),
    read(Response),
    ( (Response == yes ; Response == y)
    ->
        assert(yes(Question)) ;
        assert(no(Question)), fail).

:- dynamic yes/1,no/1.

/* How to verify something */
verify(S) :-
    (yes(S)
    ->
        true ;
    (no(S)
    ->
        fail ;
        ask(S))). 

/* undo all yes/no assertions */
undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.
```

这个程序就是本章开始提到的动物识别程序，在 swipl 下运行 go。可以通过回答一系列问题来判断动物的名称。下面是一个运行的例子：

```

?- go.
Does the animal have the following attribute: has_hair? yes.
Does the animal have the following attribute: eats_meat? |: no.
Does the animal have the following attribute: has_pointed_teeth? |: no.
Does the animal have the following attribute: has_hooves? |: y.
Does the animal have the following attribute: has_long_neck? |: n.
Does the animal have the following attribute: has_black_stripes? |: y.
I guess that the animal is: zebra
true.
```

上面代码中的 verify 函数实现的功能就是对每个需要验证的事实进行判断，如果 yes(S) 成立，就返回 true，如果 no(S) 成立，就返回 false，否则通过 ask(S) 函数询问用户，得到判断的答案，把这个答案通过 yes() 和 no() 添加到知识库中。

### 6.1.7 Python 语言的 Prolog 接口

pyswip 是 Python 语言与 swipl 的接口, 通过这个接口可以实现与 Prolog 程序的联合编程。使用库中的 `consult` 函数可以加载保存在文件中的规则, 通过函数 `assertz` 可以添加需要查询的事实条件, 通过函数 `query` 可以运行 Prolog 的查询语句。现在将 `bird.pl` 中的事实部分删除, 保存为 `birdrule.pl` 运行如下 Python 程序, 可以获得对 `bird(X)` 的查询结果。

```
from pyswip import Prolog      # 导入模块
prolog = Prolog()
prolog.consult('birdrule.pl')  # 导入 Prolog 的代码（规则库）
prolog.assertz("color(blue)") # 添加事实
prolog.assertz('season(all_year)')
prolog.assertz('size(small)')
for result in prolog.query('bird(X)'): # query 代表查询
    print(result["X"])
```

程序的查询结果是一个字典的列表, 对于上面的示例则只有一个元素: `{'X': 'kingfisher'}`。

pyswip 还支持设计可以在 Prolog 中调用的外部函数, 例如:

```
from pyswip import Prolog, registerForeign

def hello(t):          # 包含一个参数, 返回值为布尔类型
    print("Hello, ", t)
hello.arity = 1         # 这个属性是必须的

registerForeign(hello) # 注册函数, 使 Prolog 代码可以使用

prolog = Prolog()
prolog.assertz("father(michael,john)") # 事实1: michael 是 john 的父亲
prolog.assertz("father(michael,gina)") # 事实2: michael 是 gina 的父亲
print(list(prolog.query("father(michael,X), hello(X)"))) # 查询
```

程序运行结果为:

```
Hello, john
Hello, gina
[{'X': 'john'}, {'X': 'gina'}]
```

对于前面的动物识别程序, 可以将 Prolog 代码中的 `verify` 等函数去掉, 而改为用 Python 代码实现。也就是用 Python 编写 `verify` 函数, 实现用户交互接口, 显示需要验证的事实, 输入用户的选择并将用户的选择作为布尔类型返回。函数可以通过一个字典记录每个回答的结果(相当于 Prolog 代码中的 `yes` 和 `no`, 这样面对重复的问题就可以不必每次都询问用户。Python 代码的编写将在动手实验部分完成。

## 6.2 树莓派的舵机控制

### 6.2.1 舵机简介

电机也称作马达, 是电子设备中常用的动力原件。电机从使用用途上可以分为驱动电机和控制电机, 其中驱动电机常通过持续的旋转为设备提供动力, 例如电转、电动车、洗衣机等电子产品的动力大多由驱动电机提供。控制电机主要分为步进电机和伺服电机。步进电机是一种将电脉冲转化为角位移的执行机构, 可以驱动电路控制转向固定的角度, 经常用于精密控制, 例如数控机床、硬盘驱动器等。伺服电机的可以通过电压控制电机的转速或力矩, 要求电机的反应要快, 经常用于各种运动控制系统中。

舵机是一种通过电信号控制转动角度的设备, 可以看作是简单的伺服电机, 由于经常应用于航模的舵面控制, 因而被称为舵机。虽然和步进电机都是用来控制转动角度, 但步进电机多用于负载不大, 精度要求高的场合, 而舵机往往需要输出较大的力矩, 用于大负载的场景。



图 6.1: SG90 舵机

图 6.1 是一种比较常用的 9g 舵机(由于重量只有 9 克,所以一般称为 9 克舵机),它的主要指标如下:

- 工作电压: 直流 4.8V 到 6V
- 工作电流: 80mA 到 100mA
- 待机电流: 5mA
- 扭力: 1.3 到 1.7kg/cm
- 转速: 0.09 到 0.10 秒每 60 度(4.8V)
- 信号周期: 20 毫秒

舵机的连接线为三根,其中红色线为电源,棕色线为地线,黄色线为信号线。信号线的数据格式为 PWM,这与第 2.3.3 节中介绍的信号格式一致,只不过这里并不是用来输出功率,而是通过舵机内部的控制逻辑,将 PWM 信号转换为角度。由于 PWM 信号比较简单,很多处理器都有硬件的 PWM 控制器,可以通过一个 GPIO 管脚实现对舵机的控制。

控制舵机时,PWM 信号的周期为 20ms,当高电平时间为 0.5ms 时舵机输出的角度为 0 度,当高电平时间为 2.5 ms 时输出角度为 180 度(对于 90 度舵机则输出 90 度),占空比与输出角度为线性关系,通过计算可以得到占空比与角度的关系,下面示例程序展示了舵机在树莓派上测试的代码。

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.OUT) # 使用 12 号管脚来控制舵机
pwm = GPIO.PWM(12,50)
pwm.start(0)
pwm.ChangeDutyCycle(2.5) # 输出 0 度
sleep(1)
pwm.ChangeDutyCycle(7.5) # 输出 90 度
sleep(1)
pwm.stop()
GPIO.cleanup()
```

### 6.2.2 多舵机联合控制

对于机械臂、机器人等应用,舵机的数目一般较多,需要的电流比较大,无法通过树莓派进行供电,此时一般选择使用专门的舵机驱动电路来驱动和控制舵机。一种常用的多舵机控制模块基于 PCA9685 芯片,可以同时支持 16 路舵机的控制。图 6.2 为该模块的外形。

PCA9685 模块的 V+ 接口可以用来接入外部电源驱动舵机,SDA, SCL 是树莓派的 I<sup>2</sup>C 接口。支持 PCA9685 模块的开源代码很多,下面的示例使用 `adafruit_servokit` 来控制接入的最多 16 路舵机。

```
from time import sleep
from adafruit_servokit import ServoKit

kit = ServoKit(channels=16)
```

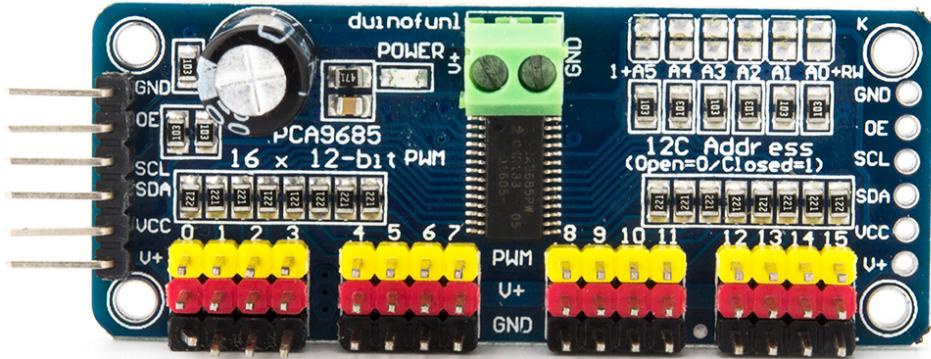


图 6.2: 16 路舵机驱动控制版

```
kit.servo[0].angle = 175
sleep(1)
kit.servo[0].angle = 45
sleep(1)
```

### 6.2.3 I<sup>2</sup>C 总线传输原理

I<sup>2</sup>C 总线最初由 Philips Semiconductor(NXP Semiconductors)提出,用于处理微处理器与芯片之间的串行低速数据交互。相比于 SPI 连接信号更少,不需要额外的地址译码器及片选信号,多个器件可以简单的连接到同一条 I<sup>2</sup>C 总线,在与传感器、低速 AD/DA、低速 OLED 显示屏等低速数据交互方面获得了较为广泛的使用。SMBus(System Management Bus)是 I<sup>2</sup>C 的一个子集,最早在 1995 年由 Intel 公司定义,最广泛的应用在电脑主板及外部电源模块、温度传感器、电压传感器等数据交互。典型的 I<sup>2</sup>C 应用如图6.3所示。

一般的,使用 I<sup>2</sup>C 设备连接中由主(master)、从(slave)设备组成。主设备发起/结束一次传输,并维护时钟信号,从设备根据主设备的寻址来响应数据传输。在同一时刻允许一个设备发送数据至总线,其余设备接收总线的信号。I<sup>2</sup>C 总线由两根线 SDA 及 SCL 两根线连接主从设备,两根线都采用上拉方式连接到 VDD。

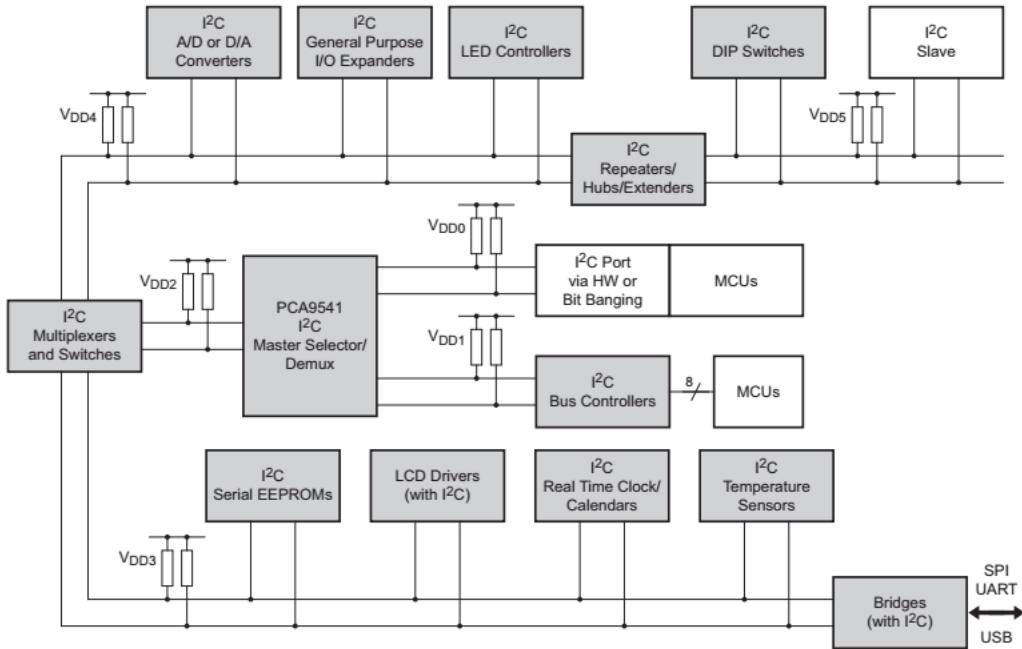
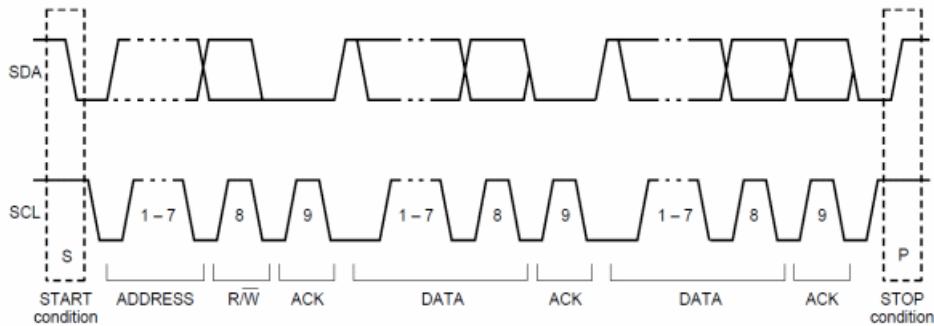
I<sup>2</sup>C 支持多种模式下速率传输,分别是标准模式、快速模式、快速模式 +、高速模式。传输速率从 100kbit/s(标准模式)至 3.4Mbit/s(高速模式)。一次典型的数据传输过程如图6.4所示。

I<sup>2</sup>C 数据传输由主设备产生一个起始位开始,然后传递 7 个地址位(对于 7 位地址模式),指定通信的从设备。接下来传递的一位是读写位,0 表示写入,1 表示读出。再接下来的一位由从设备驱动,如果从设备的地址与主设备发出的地址相同,从设备将把 SDA 信号拉低,表示确认这次数据传输(ACK)。当从设备地址被确认,真正的数据传输就开始了。如果是写入数据,则数据线仍由主设备驱动,从设备在每个字节传输之后也要驱动一位的 ACK 信号。如果是读出数据,则数据线由从设备驱动,主设备仅在应答时输出 ACK 位。最后由主设备产生一个停止位表示数据传输结束。所有的 I<sup>2</sup>C 数据与地址,都是先传递最高位,最后传递最低位。

### 6.2.4 扩展板上的 I<sup>2</sup>C 设备

在扩展板上连接了多个 I<sup>2</sup>C 设备,有 DS3231、BMP280 和 PCF8574。

DS3231 是一款高度集成的 RTC(Real Time Clock)时钟芯片,自动维护时分秒、年月日时间信息。其

图 6.3: 典型 I<sup>2</sup>C 应用图 6.4: I<sup>2</sup>C 数据传输过程

内部框图如图 6.5 所示，内部寄存器如图 6.6 所示。

PCF8574 是基于 I<sup>2</sup>C 接口的 8bit IO 接口扩展芯片，通过其接收方向杆的输入及控制蜂鸣器的输出，其内部框图如图 6.7 所示，外设连接如图 6.8 所示。从图 6.8 中可以看出：五方向摇杆按钮的四个方向接到了 PCF8574 的 P0 到 P3 端口（分别对应“左”、“上”、“下”、“右”四个方向），P4 端口接了发光管 LED2，P7 端口接了蜂鸣器输出。

### 6.2.5 I<sup>2</sup>C 设备访问

在树莓派中开发环境中已经安装了安装 smbus 库和 i2c-tools 工具。i2c-tools 中 i2cdetect 完成树莓派 I<sup>2</sup>C 设备的扫描，如下所示：

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- --
10:          -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- --
30:          -- -- -- -- -- -- -- -- --
40:          -- -- -- -- 48 -- -- -- -- --
```

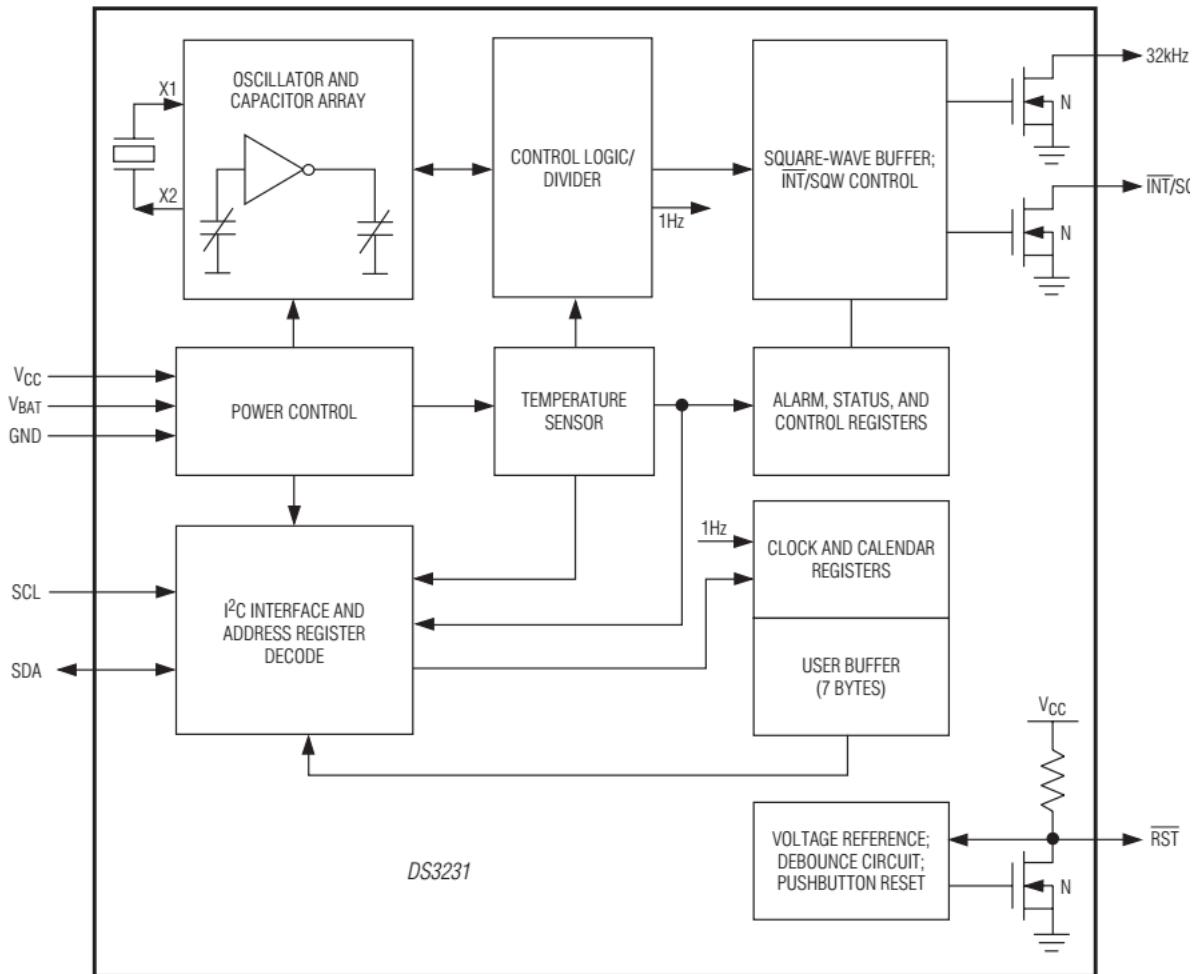


图 6.5: DS3231 内部框图

```

50: -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
60: -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
70: -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

```

根据扫描的结果,树莓派共接入了四个 I<sup>2</sup>C 设备,其中地址 20 为 PCF8574 IO 扩展芯片(参考图6.7 和图 6.8,它连接了五方向按键的其中四个方向、LED2 发光管和 BUZ 蜂鸣器);地址 48 为 PCF8591 AD/DA 扩展芯片(参考图8.13);地址 68 为 DS3231 实时钟芯片(参考图6.5);地址 76 为 BMP280 芯片(为气压传感器,本实验暂未使用)。

Smbus 完成了一些基本 I<sup>2</sup>C 函数的封装,可以使用 pydoc smbus 获取库帮助信息。常用的函数有:

1. `read_byte(address)` 从地址 address 中读取数据
2. `write_byte(address, data)` 向地址 address 中写入 data

下面的示例程序给出了五向摇杆输入的读取。

```

import smbus

address = 0x20
bus = smbus.SMBus(1)      # 初始化i2c Bus

# 关闭蜂鸣器、初始化准双向 IO 口为输入

```

| ADDRESS | BIT 7<br>MSB | BIT 6      | BIT 5            | BIT 4    | BIT 3   | BIT 2   | BIT 1   | BIT 0<br>LSB | FUNCTION          | RANGE                 |  |  |
|---------|--------------|------------|------------------|----------|---------|---------|---------|--------------|-------------------|-----------------------|--|--|
| 00h     | 0            |            | 10 Seconds       |          |         |         | Seconds |              |                   | Seconds 00-59         |  |  |
| 01h     | 0            |            | 10 Minutes       |          |         |         | Minutes |              |                   | Minutes 00-59         |  |  |
| 02h     | 0            | 12/24      | AM/PM<br>20 Hour | 10 Hour  | Hour    |         |         |              | Hours             | 1-12 + AM/PM<br>00-23 |  |  |
| 03h     | 0            | 0          |                  |          | 0       | 0       | Day     |              |                   | Day 1-7               |  |  |
| 04h     | 0            | 0          | 10 Date          |          | Date    |         |         |              | Date              | 01-31                 |  |  |
| 05h     | Century      | 0          | 0                | 10 Month | Month   |         |         |              | Month/<br>Century | 01-12 +<br>Century    |  |  |
| 06h     |              | 10 Year    |                  |          |         | Year    |         |              | Year              | 00-99                 |  |  |
| 07h     | A1M1         | 10 Seconds |                  |          |         | Seconds |         |              | Alarm 1 Seconds   | 00-59                 |  |  |
| 08h     | A1M2         | 10 Minutes |                  |          |         | Minutes |         |              | Alarm 1 Minutes   | 00-59                 |  |  |
| 09h     | A1M3         | 12/24      | AM/PM<br>20 Hour | 10 Hour  | Hour    |         |         |              | Alarm 1 Hours     | 1-12 + AM/PM<br>00-23 |  |  |
| 0Ah     | A1M4         | DY/DDT     |                  |          | 10 Date |         | Day     |              | Alarm 1 Day       | 1-7                   |  |  |
| 0Bh     | A2M2         | 10 Minutes |                  |          |         | Minutes |         |              | Alarm 2 Minutes   | 00-59                 |  |  |
| 0Ch     | A2M3         | 12/24      | AM/PM<br>20 Hour | 10 Hour  | Hour    |         |         |              | Alarm 2 Hours     | 1-12 + AM/PM<br>00-23 |  |  |
| 0Dh     | A2M4         | DY/DDT     |                  |          | 10 Date |         | Day     |              | Alarm 2 Day       | 1-7                   |  |  |
| 0Eh     | EOSC         | BBSQW      | CONV             | RS2      | RS1     | INTCN   | A2IE    | A1IE         | Control           | —                     |  |  |
| 0Fh     | OSF          | 0          | 0                | 0        | EN32kHz | BSY     | A2F     | A1F          | Control/Status    | —                     |  |  |
| 10h     | SIGN         | DATA       | DATA             | DATA     | DATA    | DATA    | DATA    | DATA         | Aging Offset      | —                     |  |  |
| 11h     | SIGN         | DATA       | DATA             | DATA     | DATA    | DATA    | DATA    | DATA         | MSB of Temp       | —                     |  |  |
| 12h     |              | DATA       | DATA             | 0        | 0       | 0       | 0       | 0            | LSB of Temp       | —                     |  |  |

图 6.6: DS3231 内部寄存器

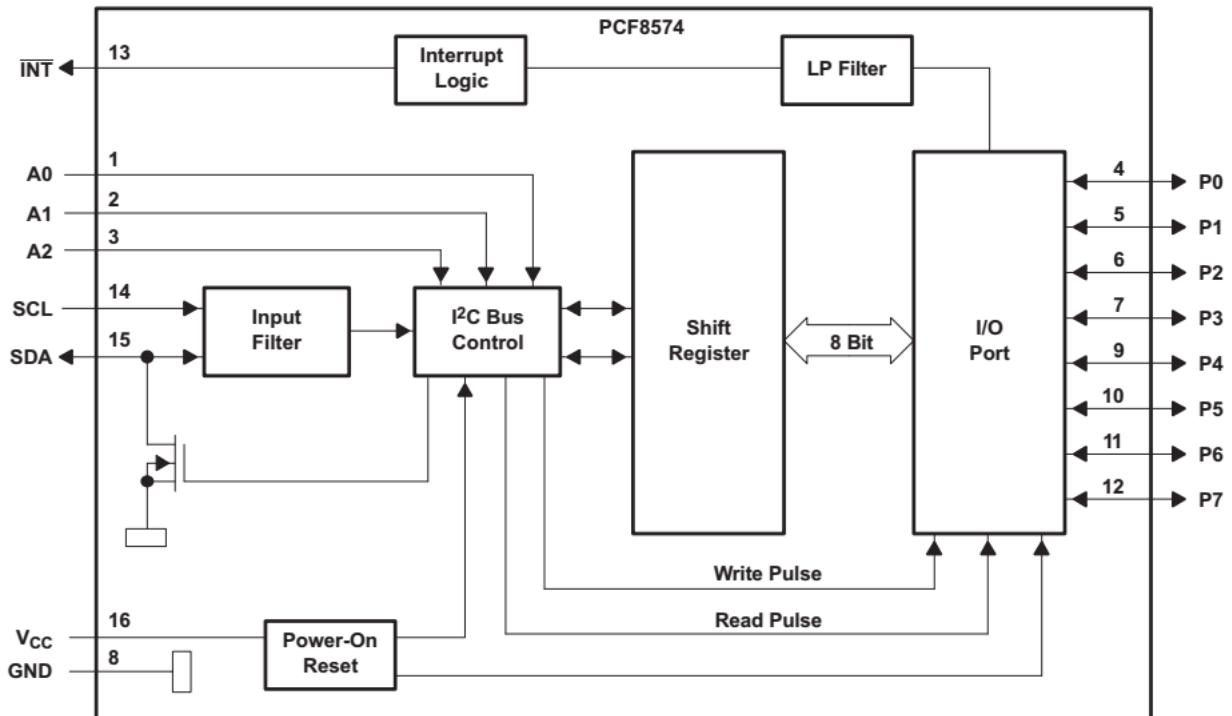


图 6.7: PCF8574 内部框图

```

bus.write_byte(address, 0xff)

# 摆杆输入
joystick_val = bus.read_byte(address)

```

RTC DS3231 的地址为 0x68, 挂载在 i2cbus-1 上, 下面示例给出了 RTC 的访问流程

```
import smbus
```

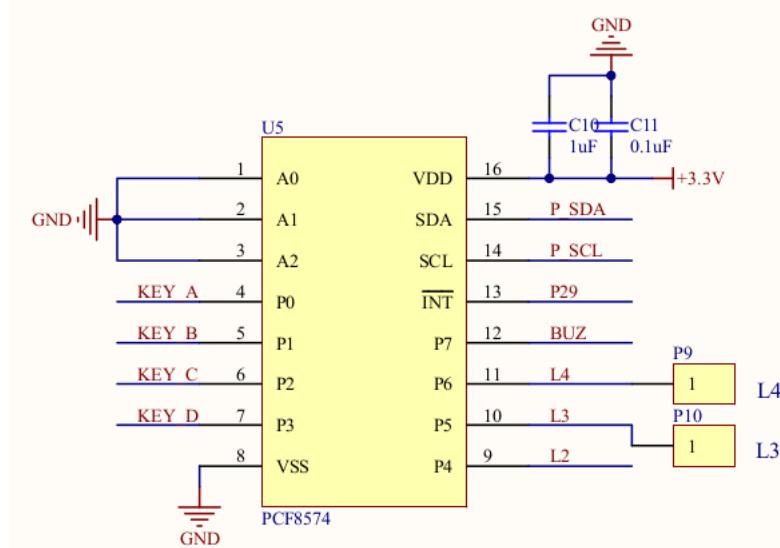


图 6.8: PCF8574 外设连接

```

import time # 包含相关库文件

address = 0x68
register = 0x00
bus = smbus.SMBus(1)      # 初始化I2C Bus
# FixTime 定义为 2019 年 6 月 12 日 18 时
FixTime = [0x00,0x00,0x18,0x03,0x12,0x06,0x19]

# 定义时钟操作函数
def ds3231SetTime():
    bus.write_i2c_block_data(address,register,FixTime)

def ds3231ReadTime():
    return bus.read_i2c_block_data(address,register,7);

ds3231SetTime() # 设置时间
ds3231ReadTime() # 读出时间

```

`write_i2c_block_data` 中的第一个参数是 I<sup>2</sup>C 的地址, 第二个参数是传递给从设备的命令, DS3231 的 I<sup>2</sup>C 协议规定这个命令就是后续传递数据的起始地址。根据图 6.6 中寄存器的描述。`ds3231SetTime` 写入了前 7 个地址, 分别设置了年月日时分秒和星期的内容。`ds3231ReadTime` 读出前 7 个地址, 也包含了全部的时间信息。

注意这里 DS3231 中保存的数据都是 BCD 编码类型, 也就是 0x19 代表十进制数 19, 在操作和应用的时候要注意转换。

## 6.3 动手实验:动物识别专家

### 6.3.1 实验目的

1. 了解专家系统的基本概念和设计方法
2. 掌握 Prolog 的基本语法
3. 学习使用 Python 语言编写 Prolog 接口
4. 学习使用树莓派控制舵机

### 6.3.2 实验内容

#### 6.3.2.1 测试示例 Prolog 代码

使用 swipl 运行本章介绍的 Prolog 代码, 利用 trace 功能观察程序进行推理的过程, 加深对 Prolog 实现的理解。特别对于动物识别代码, 要充分理解 verify 函数的实现, 便于接下来用 Python 进行实现。

#### 6.3.2.2 使用 Python 完成专家系统

去掉 animal.pl 中 go, ask, verify, undo 的定义, 利用 Python 代码实现 verify 函数, 完成专家系统的实现。这里我们虽然使用命令行界面完成实验, 与原来的 Prolog 代码运行效果区别不大。但 Python 的使用带来了更多的可能性, 可以设计更友好的输入输出界面, 使知识结构和用户接口分离开来, 便于系统的升级。

#### 6.3.2.3 使用舵机进行输出

这里的舵机作为一个转盘指针使用, 可以将专家系统的输出结果直观的显示出来。采用扩展板上 5 向遥杆来替代键盘输入(如左键表示 n, 右键表示 y)完成上述试验内容。后续可以将输入升级成语音, 那这个系统就可以做成一个小盒子, 成为一个很好玩的小玩具。

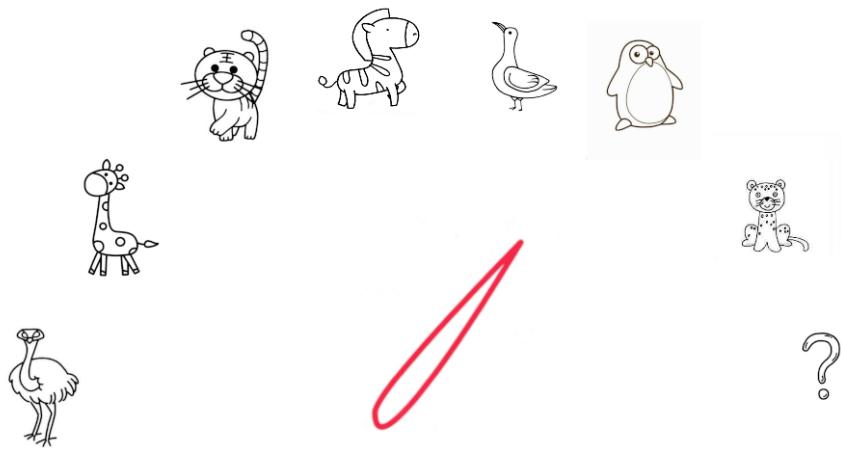


图 6.9: 动物识别的输出

## 6.4 思考题

1. Prolog 的使用场景并不仅限于专家系统, 也可以应用在其他领域, 通过查找资料说明。
2. 知识结构和用户接口分离的设计模式, 你还可以举出类似例子吗?

# 第七章 OpenCV 进阶

在 OpenCV 基础中，介绍了 OpenCV 对图片的基本处理，并采用 Hough 变换实现了对规则形状的识别。本章在已有图片识别的基础上，进一步介绍人脸识别的基本原理，并采用树莓派摄像头完成人脸的识别及图片的跟踪。

## 7.1 图片的统计特性

在 OpenCV 基础中介绍了图片基本概念及图片的基本操作，如图片的颜色尺寸、颜色空间等。本章进一步介绍图片统计特性，如图片的直方图、反射投影直方图等基本概念，通过直方图均衡处理来实现对图片的处理，并为后续运动图像的处理奠定基础。

- 图片直方图

直方图是对图片强度分布的描述，从统计学的角度来分析图片。直方图由 XY 二维表示，其中 X 值为像素值、Y 轴为落入对应 X 轴坐标区间内的图像中的对应像素数。OpenCV 中提供了直方图分析函数`calcHist()`，典型的用法如下所示。图 7.1 给出了直方图的示意。通过对直方图的均衡的处理，来实现对图片的处理，例如对图片的强度的均衡处理，OpenCV 提供了均衡函数`cv.equalizeHist()`。为进一步提升均衡性能，在全局均衡的基础上引入了局部自适应直方图均衡函数`cv.createCLAHE()`, `cv.clahe.apply()`。

```
1 img = cv.imread(filename,0)
2 hist = cv.calcHist([img],[0],None,[256],[0,256])
```

- 直方图反射投影

Swain and Ballard 最早在论文 [7] 中提出了反向投影的概念，图片的直方图反应了图片在强度的概率分布。直方图反向投影用来实现图像分割，背景与对象分离，对已知对象位置进行定位。反向投影在模式匹配、对象识别、视频跟踪中均有应用，OpenCV 中经典算法之一 CAMeanShift 就是基于反向投影实现对已知对象的位置查找与标记、从而达到连续跟踪。OpenCV 提供了一个内置函数`cv.calcBackProject()`来实现反射投影，需要注意的是在传递给`calcBackProject()`函数之前，应该对象直方图采用函数`cv.normalize()`进行规范化。反射投影的 python 例子如下所示：

```
1 roi = cv.imread('pic_roi.png')
2 hsv = cv.cvtColor(roi, cv.COLOR_BGR2HSV)
3 target = cv.imread('pic.png')
4 hsvt = cv.cvtColor(target, cv.COLOR_BGR2HSV)
5 # calculating object histogram
6 roihist = cv.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )
7 # normalize histogram and apply backprojection
8 cv.normalize(roihist,roihist,0,255,cv.NORM_MINMAX)
9 dst = cv.calcBackProject([hsvt],[0,1],roihist,[0,180,0,256],1)
```

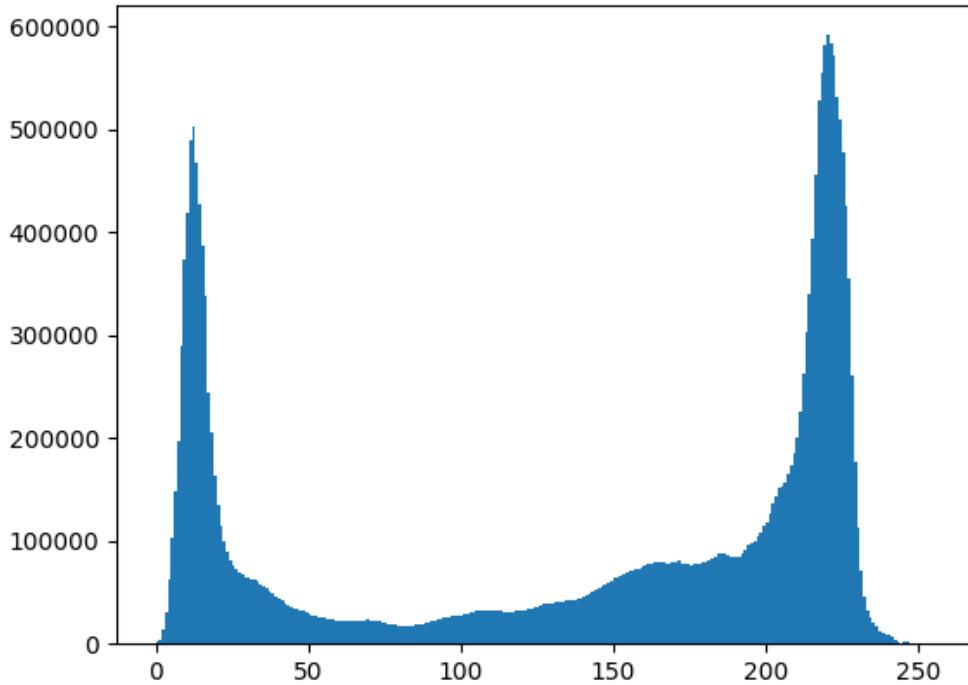


图 7.1: 图片的几种特征值示意

## 7.2 人脸识别算法基本原理

人脸识别是机器分类中的重要领域之一,对于给出的一张图片如何让机器识别出其中的人脸,许多研究者发展了多种人脸识别的方法,如基于颜色的人脸识别、基于运动的人脸识别等<sup>8</sup>。这些识别方法或多或少均有一些限制同时无法取得满意的识别效果。直至 2001 年 Viola 与 Jones 提出了基于图片特征的弱分类器级联的人脸识别算法,人脸识别算法才取得了突破性进展。随着算力的不断增强,基于多种神经网络复杂的人脸识别算法也被提出,进一步提升了识别成功率。

虽然 Viola 与 Jones 提出的人脸识别算法也存在一些限制,如需要正脸、固定尺寸的照片等,但是相比于后来提出的基于神经网络的复杂算法,降低了运行平台的处理能力要求,OpenCV 中的人脸识别算法也是基于该算法,因此,本章着重介绍该算法的基本原理。该算法主要有两个核心,主要是 Haar 图片的图片特征的弱分类器的构建及多个弱分类器的级联。

- Haar 特征计算

Haar-feature 思想源自匈牙利数学家 Alfréd Haar 于 1904 年提出的 Haar 序列,Haar 序列与 Fourier 序列类似构成了空间的一序列正交基,通过一系列的正交基将对象分解出来。OpenCV 通过计算 Haar-feature 来实现对人脸的识别,论文 [9] 给出了算法的基本原理。图片  $\mathbf{I}(\mathbf{x}, \mathbf{y})$  的特征定义式如式 7.1 所示,其中  $\mathbf{I}$  为图片, $(x, y)$  为像素点,  $w(x, y) \in (-1, 0, 1)$  为权值。

$$f(\mathbf{I}) = \begin{cases} 1, & \text{if } \sum_{x,y} w(x, y) \mathbf{I}(\mathbf{x}, \mathbf{y}) > 0 \\ 0 & \text{else.} \end{cases} \quad (7.1)$$

如图 7.2 所示,某一像素框中的特性定义为在矩形框内灰色部分的像素点的和与白色像素点和之差,通过特性的计算来实现对特性检测。图 7.2c 对应于人脸中眼睛的特征计算,眼睛在鼻子的两侧,眼

睛的颜色比鼻子的颜色暗。

需要注意的是，根据上述的定义图片的特征的计算量是非常巨大的，为实现对图像特征的快速计算，OpenCV 通过对图像的积分运算快速实现了对图像特性的计算，像素点  $(x, y)$  积分的定义为  $i(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$ ，图 7.3 给出了面积 D 的图片积分的计算方式为  $D = I(4) - I(2) - I(3) + I(1)$ 。

- 简单分类器级联

在解决图片特征值快速计算后，如何从众多的特征值中选择相关的特征值进行来进行图片的分类，AdaBoost 方法被用来构建简单的分类器实现了分类速度的快速提升。更进一步的，如何众多简单分类器的基础上进一步提升检测的性能，核心思想就是将简单的分类器级联，将上一级分类器的判决的结果作为下一个分类器的输入，通过逐级检测以提升检测的概率。其基本原理如图 7.4 所示。

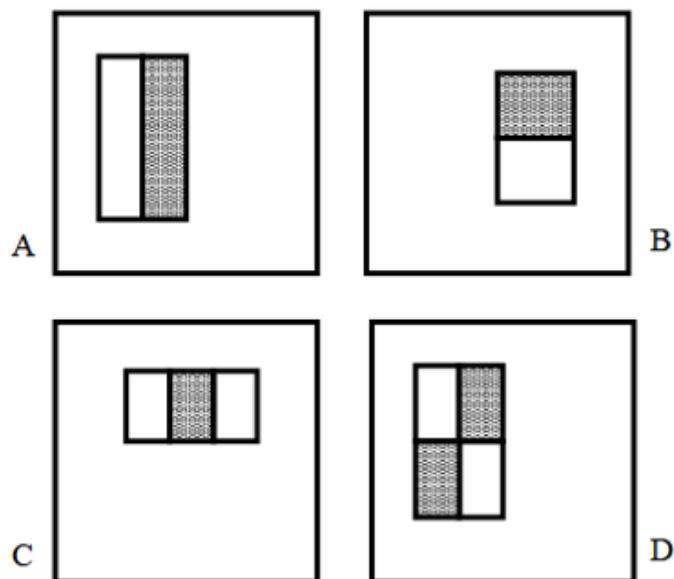


图 7.2: 图片的几种特征值示意

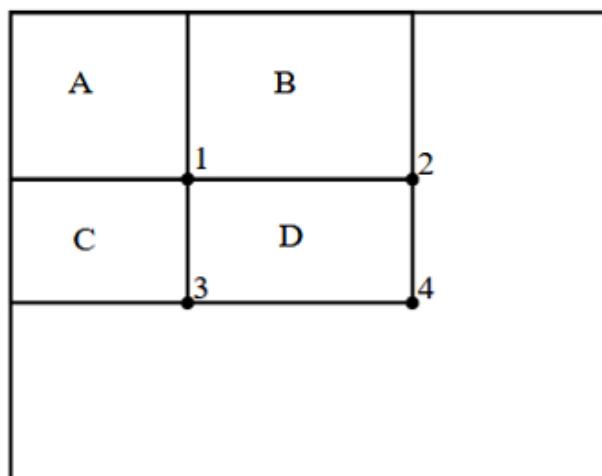


图 7.3: 图积分值快速计算示意图

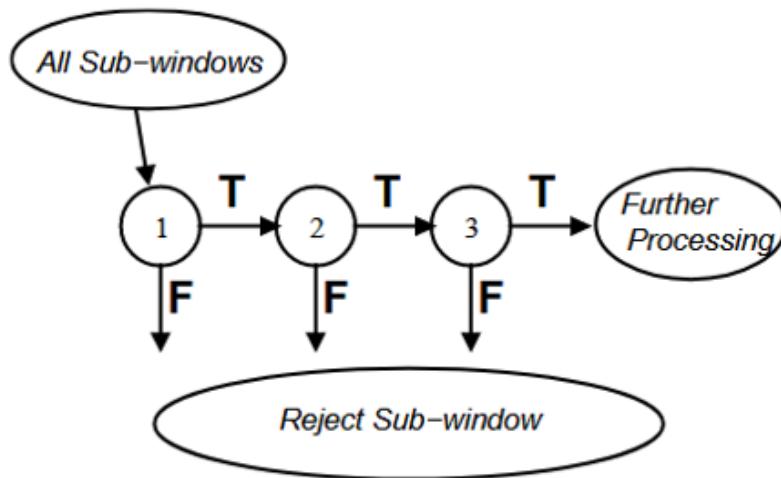


图 7.4: 级联检测器

下面代码给出了 OpenCV 中对人脸识别的代码：

```

1 import cv2 as cv
2
3 filename = '/path/to/my/pic.jpg' # 图片的位置
4
5 def detect(filename):
6     face_cascade = cv.CascadeClassifier(cv.data.haarcascades +
7         'haarcascade_frontalface_default.xml')
8     img = cv.imread(filename)
9     gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
10    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
11
12    # 将检测到的人脸标记出来
13    for (x,y,w,h) in faces:
14        img = cv.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
15
16    cv.namedWindow('Face Detected!!')
17    cv.imshow('Face Detected!!', img)
18    cv.waitKey(0)
19    cv.destroyAllWindows()

```

在完成对人脸的识别后，进一步完成眼睛的识别。眼睛识别的原理与人脸识别类似，其主要代码如下。

```

1 eye_cascade = cv2.CascadeClassifier(cv.data.haarcascades +
2     'haarcascade_eye.xml')

```

## 7.3 树莓派对摄像头的处理

- OpenCV 中摄像头使用

OpenCV 有直接对摄像头操作的库，例如下面的代码就可以将系统中的摄像头所拍摄的内容显示在窗口中。

```

1 #!/bin/python
2 import cv2 # 加载 OpenCV 库
3 cam = cv2.VideoCapture(0)      # 打开摄像头
4 cam.set(3, 1024)    # 设置图像宽度
5 cam.set(4, 768)     # 设置图像高度
6 while(True):
7     ret, frame = cam.read() # 读入一帧图像
8     cv2.imshow('VideoTest',frame) # 显示图像
9     if cv2.waitKey(1) == ord("q") :      # 等待按键
10        break
11 cam.release()      # 释放摄像头硬件
12 cv2.destroyAllWindows() # 关闭全部窗口

```

代码中 `cv2.VideoCapture(0)` 用来打开系统中的摄像头,一般具有设备文件`/dev/video0`,如果系统中有多个摄像头,可以改变该函数的参数来选择不同设备。

- 树莓派摄像头使用

在树莓派中,上面的代码只适用于支持 v4l 驱动的摄像头,对于树莓派 CSI 接口的摄像头,还可以使用不同的方法。树莓派系统包含一个 `python-picamera2` 的软件包用来简化 CSI 接口摄像头的使用。下面是一个示例代码:

```

1 from picamera2 import PiCamera2
2 import time
3 import cv2           # 导入需要的库
4
5 cam = PiCamera2()
6 cam.still_configuration.main.size = (640,480)
7 cam.still_configuration.main.format = 'RGB888'
8 cam.configure("still")
9 cam.start()
10 time.sleep(1)          # 等待摄像头模块初始化
11 while(True):
12     image = cam.capture_array("main") # 读入一帧图像
13     cv2.imshow("Frame", image) # 显示图像
14     key = cv2.waitKey(1) & 0xFF # 等待按键
15     if key == ord("q"):
16         break
17 cam.stop()            # 释放摄像头硬件
18 cv2.destroyAllWindows() # 关闭全部窗口

```

## 7.4 运动物体的追踪

运动物体的追踪主要有两种方式,分别为背景差分法(Background subtractor)和均值平移(Mean-shift and CAMShift)。

- 背景差分法

背景差分法利用当前帧与背景帧的差别实现对运动物体的定位,如图 7.5。

背景差分法将每个当前帧均与背景帧进行差分,从而获取运动到图像中的物体或图像中运动的物体。因此,理论上背景帧不存在运动物体是最理想的状态。

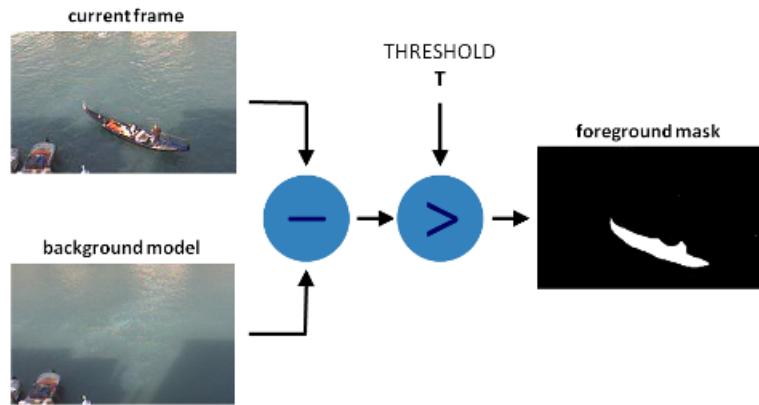


图 7.5: 背景差分法

- 均值平移法

均值平移主要是根据概率密度的梯度爬升来寻找局部最优。通过直方图反投影将图像的像素值根据所要追踪物体的归一化直方图反投影为概率值,并通过迭代逐步寻找到概率密度最大的区域,作为该物体的最有可能出现的区域,从而追踪到物体,如图 7.6。

CamShift 算法的全称是”Continuously Adaptive Mean-SHIFT”, 即: 连续自适应的 MeanShift 算法。CamShift 是对视频序列的所有图像帧都作 MeanShift 运算, 并将上一帧的结果(即搜索窗口的中心位置和窗口大小)作为下一帧 MeanShift 算法的搜索窗口的初始值, 如此迭代下去。简单点说, meanShift 是针对单张图片寻找最优迭代结果, 而 camShift 则是针对视频序列来处理, 并对该序列中的每一帧图片都调用 meanShift 来寻找最优迭代结果。需要注意的是, 在使用均值便平移时, 非常依赖于参数设置(即初始位置、各个维度的带宽等), 因此效果还需具体而定。CAMShift 的例子如下所示:

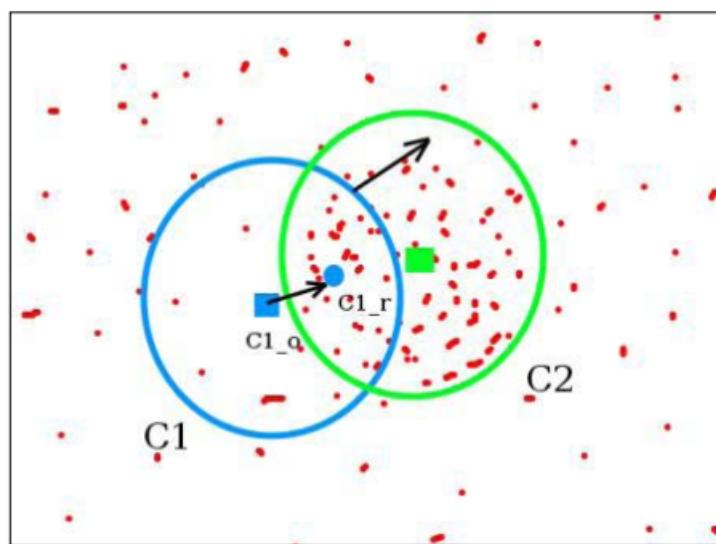


图 7.6: 均值平移法

```

2
3 while(1):
4     ret ,frame = cap.read()
5     if ret == True:
6         hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
7         dst = cv.calcBackProject([hsv],[0],roi_hist,[0,180],1)
8
9         # apply meanshift to get the new location
10        ret, track_window = cv.CamShift(dst, track_window, term_crit)
11
12        # Draw it on image
13        pts = cv.boxPoints(ret)
14        pts = np.int0(pts)
15        img2 = cv.polylines(frame,[pts],True, 255,2)
16        cv.imshow('img2',img2)
17        k = cv.waitKey(60) & 0xff
18        if k == 27:
19            break
20        else:
21            cv.imwrite(chr(k)+".jpg",img2)
22
23    else:
24        break
25
26 cv.destroyAllWindows()
27 cap.release()

```

## 7.5 动手实验:视频的处理及人脸识别

### 7.5.1 人脸识别

1. 读取树莓派的摄像头,完成自己人脸、眼睛的识别

采用摄像头对人脸的识别与在照片中完后人脸的识别类似,主要从摄像头的实时流中获取数据,对每一帧图片完成人脸的识别并进行标记,主要核心代码参考实验原理部分。部分参考代码如下。

```

1 while (True):
2     # 从摄像头中读取数据并转化成灰度
3     ret, frame = camera.read()
4     gray = cv.cvtColor(frame, cv2.COLOR_BGR2GRAY)

```

2. 完成是否佩戴口罩的简单识别(选做)

不依赖识别模型,主要原理可以根据眼睛的位置定位到口罩区域,并判别该区域是否存在口罩来确定该人是否佩戴口罩。(完成摄像头中一人的口罩识别即可,多人识别可能存在运行速度的限制)

其中在口罩区域中判别是否存在口罩的部分参考代码如下:

```

1
2 hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
3 # 提取皮肤区域

```

```

4 skin = cv2.inRange(hsv, np.array([0, 25, 0]), np.array([50, 255, 255]))
5 # 在皮肤图层提取口罩区域
6 skin_mask = np.zeros((frame.shape[0], frame.shape[1]), dtype=np.uint8)
7 skin_mask[mask_y_begin:mask_y_end, mask_x_begin:mask_x_end] =
8     skin[mask_y_begin:mask_y_end, mask_x_begin:mask_x_end]
9 out = cv2.bitwise_and(frame, frame, mask = skin_mask)
10 # 计算非皮肤的占比
11 mask = np.where((out == 0), 0, 1).astype('uint8')
12 mask_area = mask.sum()
13 if mask_area / ((mask_x_end - mask_x_begin) * (mask_y_end - mask_y_begin)) >
14     0.8:
15     ...
16 else:
17     ...

```

### 3. 注意事项

- (a) Haar 人脸识别的限制 Haar 算法对于人脸的识别有一些限制, 如图片的大小、并需要正面的人脸图片。

## 7.5.2 物体跟踪

### 1. 跟踪移动的乒乓球

使用 BackgroundSubtractor 识别移动的乒乓球, 并用方框将其圈出来。需要注意的是, 在处理的时候可能出现噪声, 并且处理不是完美的, 因此在画出移动物体的边框时, 可以使用 contourArea() 判断边框的大小, 从而确定是否为物体, 并将物体标出。部分参考代码如下:

```

1
2 bs = cv2.createBackgroundSubtractorKNN(detectShadows = True)
3 ...
4 fgmask = bs.apply(frame)
5 th = cv2.threshold(fgmask.copy(), 244, 255,
6     cv2.THRESH_BINARY)[1]
7 dilated = cv2.dilate(th,
8     cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3)),
9     iterations = 2)
10 contours, hier = cv2.findContours(dilated,
11     cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) # 提取出移动物体的轮廓

```

### 2. 利用 CAMShift 方法来跟踪乒乓球(选做)

## 7.6 思考题

1. 直方图反射投影的基本原理是什么?
2. CAMShift 算法的局限性有哪些?

# 第八章 人工神经网络

本章将介绍神经网络的概念，从最简单的神经元模型——线性回归，到如何堆叠多层已创建深度神经网络，并介绍如何在 PyTorch 开源神经网络框架下建立和训练神经网络，最后介绍在视觉领域应用最多的卷积神经网络。在这个单元，我们将动手实验，实现通过手势识别实时控制发光二极管亮度的功能。实验中也用到了 AD/DA 的原理，以及前边章节中所介绍过的 OpenCV 等知识。

## 8.1 神经网络

人工神经网络 (Artificial Neural Network) 是机器学习的研究热点，是深度学习的基础。它从信息处理角度对人脑神经元网络进行抽象，其中最常用的是用感知器模型来模拟单个神经元，如图8.1 所示。

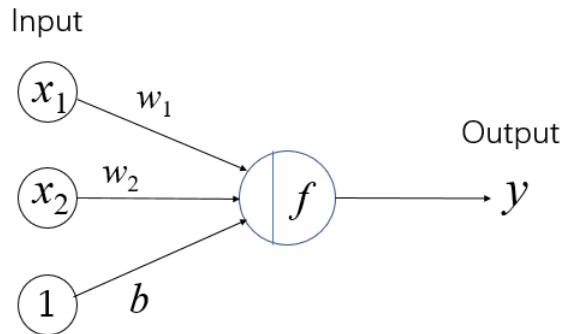


图 8.1: 神经元感知器模型

图8.1 中各个参数的关系如下：

$$y = f\left(\sum_{i=0}^n w_i x_i + b\right)$$

在这个模型中，神经元接收来自  $n$  个其它神经元传递过来的输入信号，通过不同权重叠加后并与阈值  $b$  比较，最后通过“激活函数” $f$  处理以产生神经元的输出。理想的激活函数是阶跃函数，输出“0”对应神经元抑制，输出“1”对应神经元兴奋。但由于阶跃函数不连续、不光滑，实际常用 sigmoid 等函数作为激活函数，它把输出限制在  $(0,1)$  范围内。复杂的神经网络由多个不同参数的神经元组成网络，通过训练的方式确定参数的值，实现解决问题的目的。

## 8.2 从线性回归到 MLP

本节通过简化激活函数为恒等函数，将神经元模型简化为线性模型，通过学习的方式获得线性模型的参数，进而了解机器学习的基本方法。这样的模型一般被称为线性回归模型。

对于给定数据集  $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)$ , 其中  $\mathbf{x}_i = (x_1, x_2, \dots, x_n)$  为输入,  $y_i$  为输出。线性回归模型试图获得一个线性模型

$$f(\mathbf{x}) = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

来尽可能的根据输入值预测实际输出的值, 即  $f(\mathbf{x}_i) \simeq y_i$ 。

利用欧式距离定义一个误差函数:

$$E = (f(\mathbf{x}_i) - y_i)^2$$

通过数据学习的目的就是最小化这个误差。从数学上的角度来看, 梯度的方向是函数增长速度最快的方向, 那么梯度的反方向就是函数减少最快的方向。

$$\frac{\partial E}{\partial w_i} = 2(f(\mathbf{x}_i) - y_i)x_i = 2\Delta_i x_i$$

$$\frac{\partial E}{\partial b} = 2(f(\mathbf{x}_i) - y_i) = 2\Delta_i$$

定义学习率为  $\eta$ , 更新参数  $w_i$  的方式就是用  $w_i - \eta\Delta_i x_i$  替代  $w_i$ , 这里将常数项 2 省去。这样根据数据对  $w_i$  不断更新以获得最优的线性模型参数的方法就是梯度下降法。

前边简单介绍了以线性回归模型为主的人工神经网络, 下面进一步了解由神经元组成的网络。图8.1 中的感知器模型可以看成是两个输入, 一个输出的单层人工神经网络(输入层不涉及计算, 不记入网络的层数)。

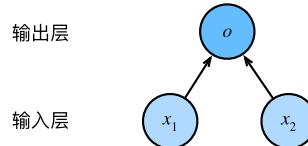


图 8.2: 线性回归模型

更常见的人工神经网络会有更多的层, 中间的层称为隐藏层。如图8.3是含一个隐藏层的网络。由于线性变换的组合仍然是线性变换, 为了体现多层模型的价值, 在每个感知器模型中都引入了非线性的激活函数, 在前文我们提及了 sigmoid 函数, 目前更常用的是一个更简单的 ReLU (Rectified Linear Unit) 函数 (如图 8.4):

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

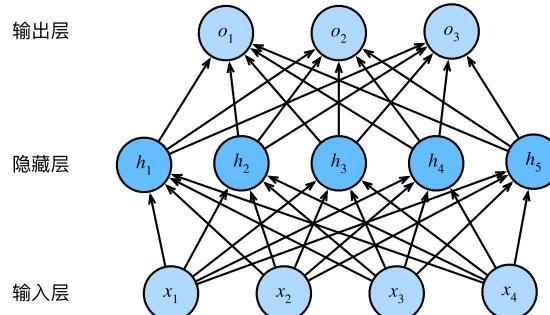


图 8.3: 多层感知器模型人工神经网络

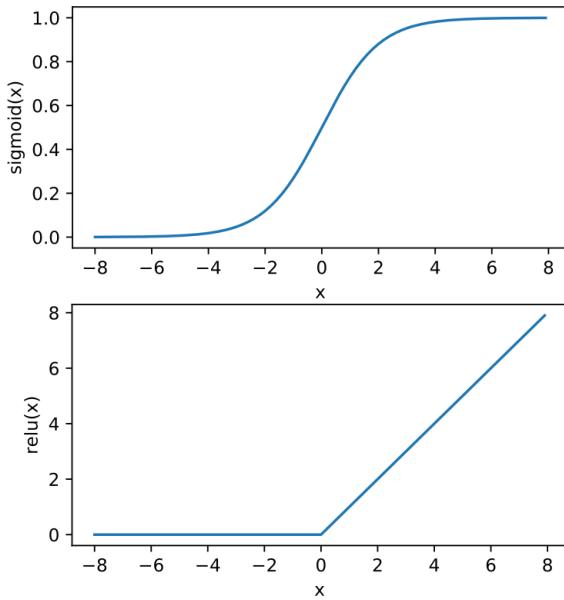


图 8.4: sigmoid 和 ReLU 函数

### 8.3 PyTorch 简介

PyTorch 是由 Facebook 的人工智能研究小组在 2016 年开发的基于 Torch 的 Python 机器学习库。Pytorch 是 torch 的 python 版本，是由 Facebook 开源的神经网络框架。

PyTorch 的程序可以立即执行计算，这正好符合 Python 的编程方法，不需要完成全部代码才能运行，可以轻松的运行部分代码并实时检查。PyTorch 支持互动式的调试，使得调试和可视化变得非常容易。与 Tensorflow 的静态计算图不同，pytorch 的计算图是动态的，以便可以在运行时构建计算图，甚至在运行时更改它们，在不知道创建神经网络需要多少内存的情况下这非常有价值。PyTorch 可以顺利地与 Python 数据科学栈集成。它非常类似于 numpy，甚至注意不到它们的差别。

对 PyTorch 的最基本理解包括如下三方面：

- Numpy 风格的 Tensor(张量)操作。Tensor 是神经网络框架中重要的基础数据类型，可以简单理解为 N 维数组的容器对象。tensor 之间通过运算进行连接，从而形成计算图。PyTorch 中 tensor 提供的 API 参考了 Numpy 的设计，因此熟悉 Numpy 的用户基本上可以无缝理解并创建和操作 tensor，同时 torch 中的数组和 Numpy 数组对象可以无缝的对接。Torch 定义了七种 CPU tensor 类型和八种 GPU tensor 类型，torch 模块内提供了操作 tensor 的接口，而 Tensor 类型的对象上也设计了对应了接口，例如 torch.add() 与 tensor.add() 等价。
- 变量自动求导。tensor 对象通过一系列的运算可以组成动态图，每个 tensor 对象可以方便的计算自己对目标函数的梯度。这样就可以方便的实现神经网络的后向传播过程。
- 神经网络层与损失函数优化等高层封装。网络层的封装存在于 torch.nn 模块，损失函数由 torch.nn.functional 模块提供，优化函数由 torch.optim 模块提供。

torch.nn 模块提供了创建神经网络的基础构件，这些层都继承自 Module 类。在 nn.functional 模块中，提供多种激活函数的实现。通常对于可训练参数的层使用 module，而对于不需要训练参数的层如 softmax 这些，可以使用 functional 中的函数。用 PyTorch 定义如图8.3所示 MLP 神经网络的代码如下：

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
```

```

super(Net, self).__init__()
self.hidden = nn.Linear(4, 5)
self.out    = nn.Linear(5, 3)

def forward(self, x):
    x = F.relu(self.hidden(x))
    x = self.out(x)
    return x
net=Net()
print(net)

```

神经网络实现的基本步骤:

- 准备数据: 带标签的训练集和测试集
- 定义网络结构 model
- 定义损失函数
- 定义优化算法 optimizer
- 训练
  - 前向传播计算网络输出 output 和计算损失函数 loss
  - 反向传播更新参数:
    - \* 将上次迭代计算的梯度值清 0: optimizer.zero\_grad()
    - \* 反向传播, 计算梯度值: loss.backward()
    - \* 更新权值参数: optimizer.step()
  - 保存训练集上的 loss 和验证集上的 loss 以及准确率以及打印训练信息(可选)
- 图示训练过程中 loss 和 accuracy 的变化情况(可选)
- 在测试集上测试

深度学习框架中涉及很多参数,下面介绍一下 batch、iterations 和 epochs 的概念:

深度学习的优化算法,即梯度下降,每次的参数更新有两种方式:

1. 遍历全部数据集算一次损失函数,然后算函数对各个参数的梯度,更新梯度。这种方法每更新一次参数都要把数据集里的所有样本都看一遍,计算量开销大,计算速度慢,不支持在线学习,这称为 Batch gradient descent,批梯度下降。
2. 每看一个数据就算一下损失函数,然后求梯度更新参数,这个称为随机梯度下降,stochastic gradient descent。这个方法速度比较快,但是收敛性能不太好,可能在最优点附近晃来晃去,得不到最优点。两次参数的更新也有可能互相抵消掉,造成目标函数震荡的比较剧烈。

为了克服两种方法的缺点,现在一般采用的是一种折中手段,mini-batch gradient decent,小批的梯度下降,这种方法把数据分为若干个批,按批来更新参数,这样,一个批中的一组数据共同决定了本次梯度的方向,下降起来就不容易跑偏,减少了随机性。另一方面因为批的样本数与整个数据集相比小了很多,计算量也不是很大。基本上现在的梯度下降都是基于 mini-batch 的,所以深度学习框架的函数中经常会出现 batch\_size,就是指这个。

每一次迭代都是一次权重更新,每一次权重更新需要 batch\_size 个数据进行 Forward 运算得到损失函数,再 BP 算法更新参数。1 个 iteration 等于使用 batch\_size 个样本训练一次。

epochs 被定义为向前和向后传播中所有批次的单次训练迭代。这意味着 1 个周期是整个输入数据的单次向前和向后传递。简单说,epochs 指的就是训练过程中数据将被“轮”多少次。

例如,训练集有 1000 个样本,batchsize=10,那么训练完整个样本集需要:100 次 iteration,1 次 epoch。

## 8.4 卷积神经网络(CNN)模型

CNN 相比多层感知机等简单神经网络的特点是：一，连接仅与图像相邻像素相对应的神经元，让神经元处理来自彼此接近的像素的信息，比如为了识别字母或数字，需要分析像素的相关性，它们决定了元素的形状；二，使用共享参数，一层中所有神经元之间共享有限数量的权重，由此减少了权重数量，有助于防止过拟合。

### 8.4.1 卷积层

卷积层是 CNN 的重要组成部分，由一组过滤器(Filters, 也称为卷积核，或特征检测器)组成，每个过滤器都应用于输入数据的所有区域。过滤器由一组可学习的权重定义。如图8.5所示，每个输入神经元可以代表一个像素的灰度值，首先，在图像的右上角应用  $3 \times 3$  过滤器(通常大小是任意的，如  $2 \times 2, 4 \times 4, 5 \times 5$  等)，由于有九个输入神经元相连，就有 9 个权重，过滤器的输出是其输入的加权和。过滤器的输出表示下一层神经元的激活值，如果对应于该过滤器的特征出现在该处，则下一层相应的神经元将激活，神经元之后使用激活函数，最常见的激活函数是 ReLU。对于下一层的每个新的神经元，过滤器将在输入图像上移动，然后用每组新的输入神经元计算其输出。对于移动，过滤器的权重在整个图像中不会改变(权重共享)，即使用所有相同的 9 个过滤器权重计算所有输出神经元的激活，每次使用一组不同的输入神经元。每个过滤器对应于特定的一个特征，通过共享权重，可保证过滤器能够在整个图像中找到特征。

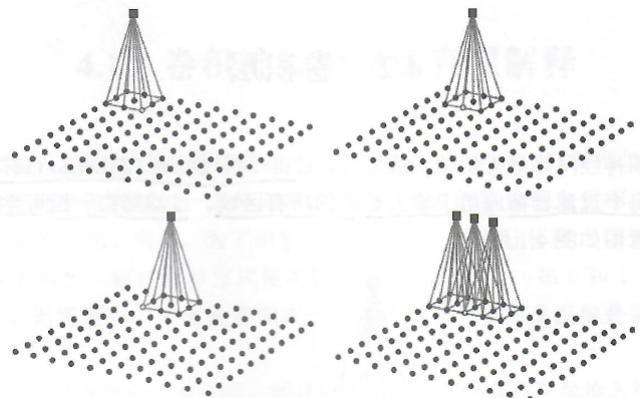


图 8.5: 过滤器及其在图像上的移动

在空间上排列的神经元称为深度切片(Depth Slices)或特征图(Feature Map)，切片可以用作网络中其他层的输入。在卷积层，偏置权重也在所有神经元之间共享，对整个切片使用单个偏置权重。对于彩色图片，可以将图像分成多个颜色通道，若为 RGB 图像，则为 3，将每个颜色通道视为一个深度切片，切片的组合称为深度为 3 的输入体积(Input Volume)。如图8.6所示，唯一的  $3 \times 3$  过滤器应用于每个切片，将这三个过滤器组合成一个  $3 \times 3 \times 3 + 1$ (含一个偏置)的大过滤器，有 28 个权重。

一个过滤器对应于一个特定特征，如边或线，而许多特征都很重要，故一组输入切片上可以应用多个过滤器，每个过滤器将生成一个唯一的输出切片。

输入和输出特征映射具有不同的维度，假设有一个大小为 (width, height) 的输入层和一个维度为 (filter\_w, filter\_h) 的过滤器，应用卷积后，输出层的维度为 (width-filter\_w+1, height-filter\_h+1)，如图8.7所示。

设计一个 CNN，需要做的是定义网络架构，如卷积层的数量、输出体积的深度以及过滤器的大小，经训练后网络将找出特征。

过滤器不仅可以每次移动一个像素，也可以移动多个像素，卷积层的这个参数称为步幅(步长)，而通常输入的所有维度的步幅都是相同的。把步幅包括进去，输出层的维度为  $((\text{width}-\text{filter}_w)/\text{stride}_w+1, (\text{height}-\text{filter}_h)/\text{stride}_h+1)$ 。例如，一个  $28 \times 28$  输入图像，使用步幅为 1 的  $3 \times 3$  过滤器卷积生成的正方

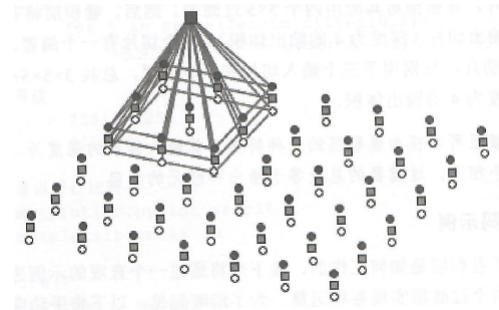


图 8.6: 深度为 3 的输入切片示例

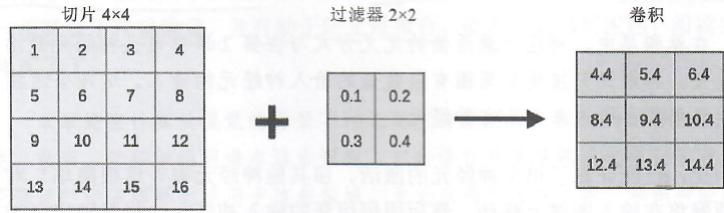


图 8.7: 在 4x4 切片上使用 2x2 过滤器进行卷积的示例

形切片输出大小为  $28-3+1=26$ , 但是当步长为 2 时, 得到  $(28-3)/2+1=13$ 。较大步幅的主要作用是增加输出神经元的感受野, 使网络能够检测到输入的更大、更复杂的特征。

如果需要使输出的幅度与输入相同, 可以在卷积运算之前用 0 的行和列填充输入片的边缘, 如图8.8所示。加入填充这个因素, 假设输入切片为  $I = (I_w, I_h)$ , 过滤器为  $F = (F_w, F_h)$ , 步幅为  $S = (S_w, S_h)$ , 填充为  $P = (P_w, P_h)$ , 则输出切片  $O = (O_w, O_h)$  由以下公式给出:

$$O_w = \frac{I_w + 2P_w - F_w}{S_w} + 1$$

$$O_h = \frac{I_h + 2P_h - F_h}{S_h} + 1$$

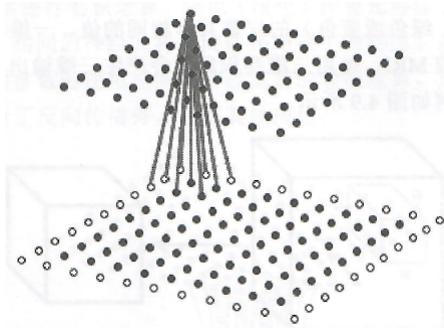


图 8.8: 对卷积层进行填充以使得输出与输入幅度相同

#### 8.4.2 池化层

借助池化层(Pooling Layers)也可以增加神经元的感受野。池化层将输入切片分割成一个网格, 每个网格单元代表一个由多个神经元组成感受野。然后在网格的每个单元上应用池化操作。池化层不会改变体积深度, 因为池化操作是在每个切片上独立执行的。

最常用的最大池化会在每个局部感受野(网格单元)中获取具有最高激活值的神经元。感受野为 2x2 的最大池化如图8.9所示。平均池化则输出每个感受野内所有激活的平均值。池化层涉及两个参数, 步幅

(与卷积层相同)及感受野大小,实际上只使用两种组合,第一种是 $2\times 2$ 的感受野,步幅为2;第二种是 $3\times 3$ 的感受野,步幅为2(重叠)。

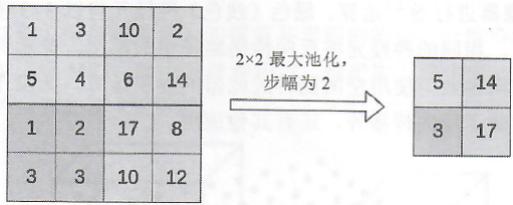


图 8.9: 最大池化示例

用 $I$ 表示输入切片的大小,用 $F$ 表示感受野的大小,用 $S$ 表示步幅的大小,用 $O$ 表示输出的大小,池化层通常没有填充,其输出大小计算公式如下:

$$O_w = \frac{I_w + 2P_w - F_w}{S_w} + 1$$

$$O_h = \frac{I_h + 2P_h - F_h}{S_h} + 1$$

#### 8.4.3 卷积神经网络的结构

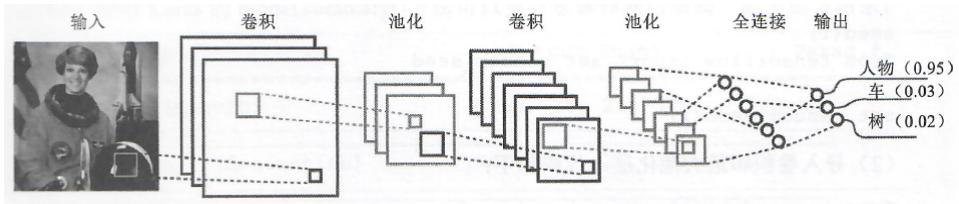


图 8.10: 卷积神经网络的结构

一个基本 CNN 的结构如图8.10所示。大多数 CNN 有如下基本属性:

1. 通常将一个或多个卷积层与一个池化层交替使用,由此卷积层可以在感受野大小的每一级检测特征。较深层的感受野大小大于网络起始处的感受野大小,由此可以从较大的输入区域捕获更复杂的特征。
2. 最深层检测到的特征是高度抽象的,无法直接阅读,通常在最后一个卷积/池化层之后添加一个或多个全连接层。可以将全连接层视为网络语言(人类无法理解)和人类语言之间的翻译器。
3. 较深的卷积层通常比初始层存在更多的过滤器(因此体积深度更高)。网络开始时的特征检测器工作在一个小的感受野上,只能检测在所有类之间共享的有限数量的特征,如边或线。而一个较深的层将检测到更复杂和众多的特征,比如汽车、树或人,每个类都有自己的一组特征,如轮胎、车门、树叶和脸等,因此需要更多的特征检测器。
- 4.

## 8.5 模数转换(AD)与数模转换(DA)

把模拟量转换为数字量的设备称为模数(A/D)转换器。在单端输入情况下,A/D 转化启动时,对 A/D 输入的模拟信号与地之间的差值与 A/D 的最小分辨率比较,根据比较的值量化输出得到最终的 AD 值(如图8.11)。A/D 的最小分辨率与 A/D 的位数有关。如在 8bit 条件下,A/D 的最小分辨电压为:

$$V_{lsb} = \frac{V_{REF} - V_{AGND}}{256}$$

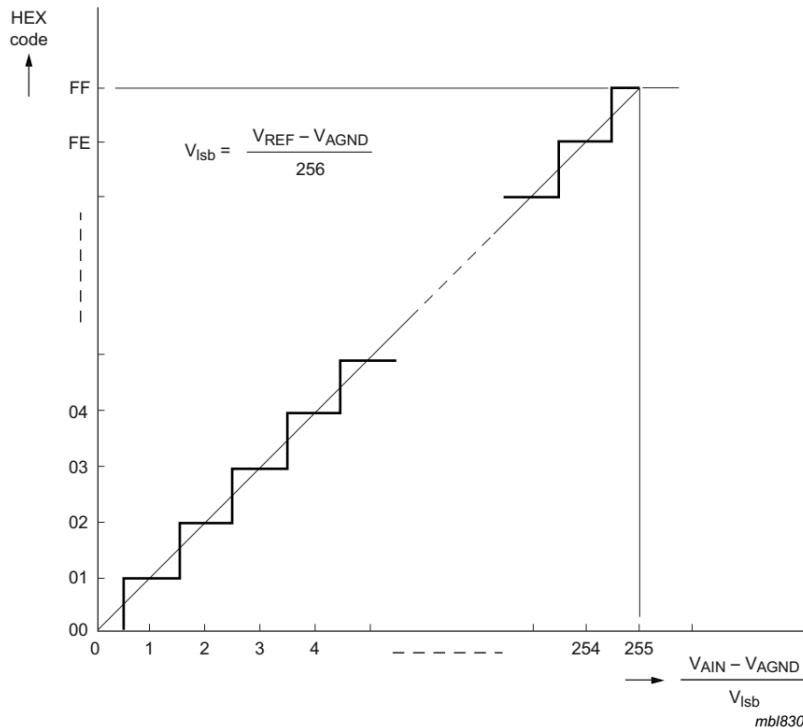


图 8.11: 单端 A/D 原理

把数字量转换为模拟量的设备称为数模(D/A)转换器。如图 8.12, 数模转换根据输入(8bit 数据)进行 8bit->256 译码, 将译码的输出值驱动分档开关, 最后输出不同的电压值, 完成 D/A 转换。

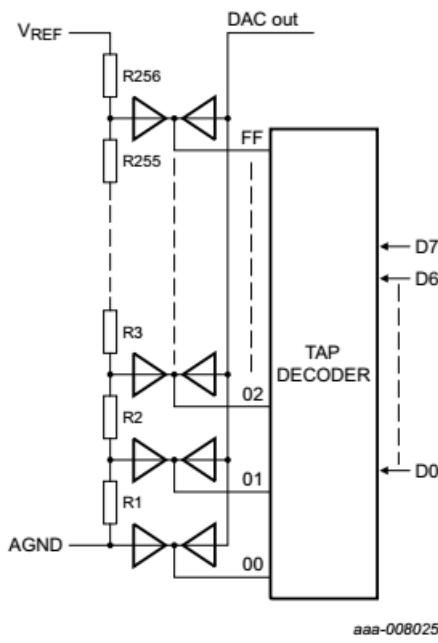


图 8.12: 数模转换原理

PCF8591 芯片也挂载在树莓派的 I<sup>2</sup>C 总线上, 其内部框图如图8.13所示。

扩展板上 PCF8591 的 A0/A1/A2 均接低电平, 根据 PCF8591 地址配置规则(表8.1), PCF8591 的地址为 0x48。PCF8591 集成了多路 AD/DA 功能, 在使用其功能时, 需要对其控制寄存器进行控制, 控制寄存器的定义如图8.14所示:

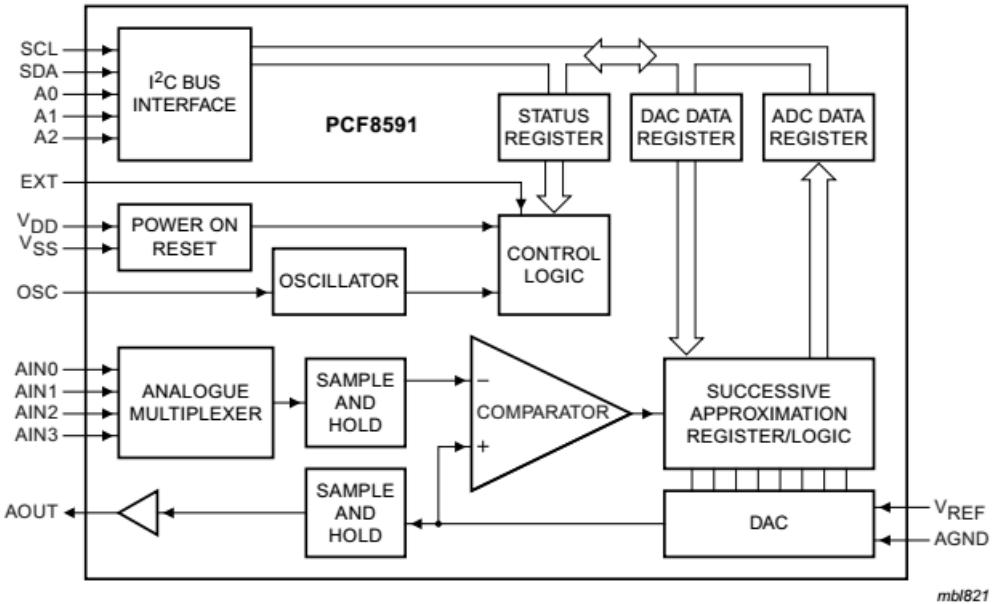


图 8.13: PCF8591 内部框图

表 8.1: PCF8591 地址配置

| 位  | 7 | 6 | 5 | 4 | 3  | 2  | 1  |
|----|---|---|---|---|----|----|----|
| 地址 | 1 | 0 | 0 | 1 | A2 | A1 | A0 |

控制寄存器共有 8bit, 通过配置不同 bit, 来实现 A/D 工作模式、通道选择及 D/A 输出功能的定义。A/D 的参考代码如下：

```
import smbus
import time      # 包含相关库文件

address = 0x48
A0 = 0x40
bus = smbus.SMBus(1) # 初始化 i2c Bus
while True:
    bus.write_byte(address,A0)
    value = bus.read_byte(address) # 循环读出
    time.sleep(1)
```

D/A 的参考代码如下：

```
import smbus
import time      # 包含相关库文件

address = 0x48
A0 = 0x40
bus = smbus.SMBus(1) # 初始化 i2c Bus
while True:
    bus.write_byte_data(address, cmd, value) # 循环写入
    .... # 省略其它代码
```

smbus 的使用可以使用 pydoc smbus 命令来查看。

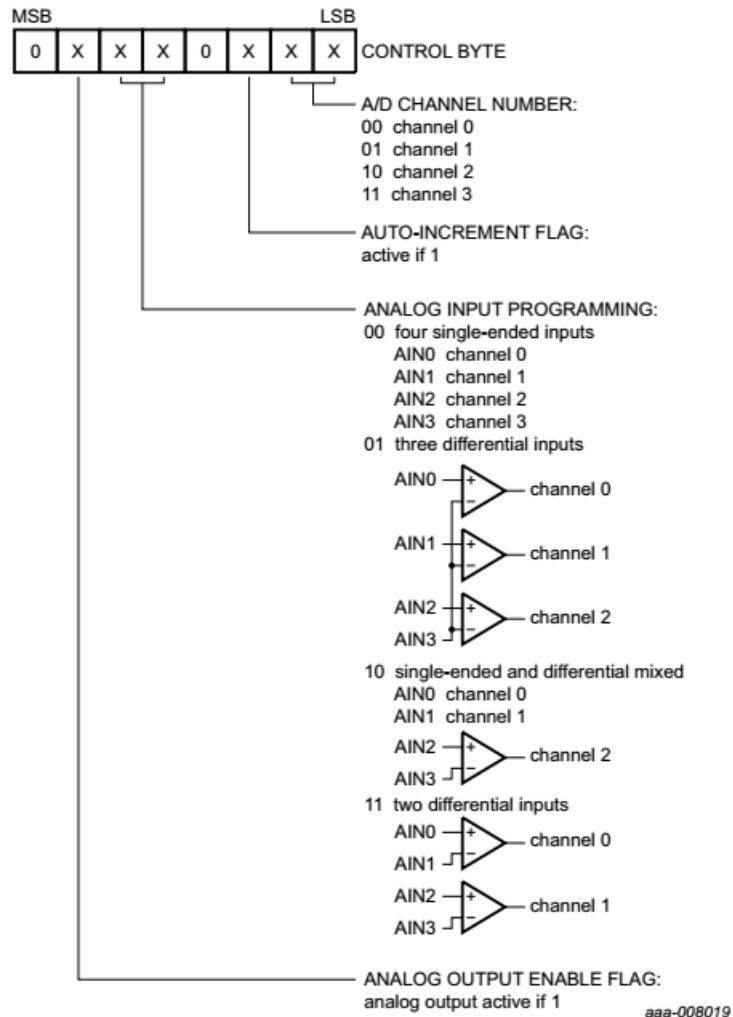


图 8.14: PCF8591 控制寄存器定义

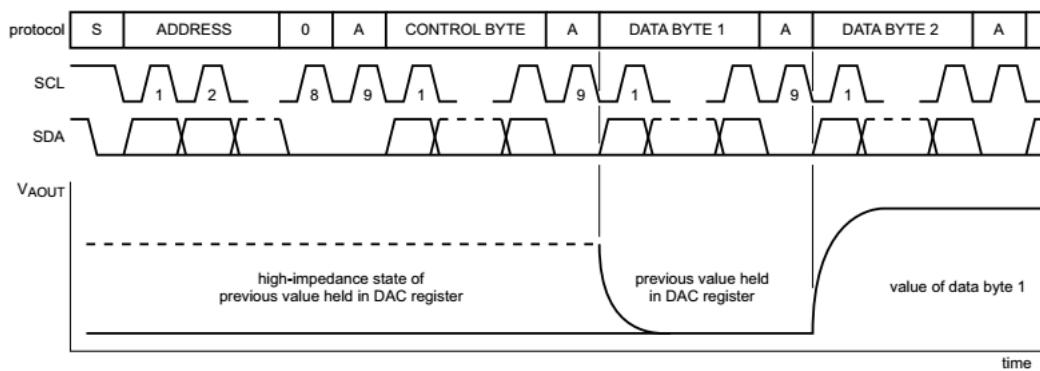


图 8.15: PCF8591 D/A 接口时序

## 8.6 动手实验:通过手势识别实时控制发光二极管亮度

### 8.6.1 实验目的

- 熟悉 AD/DA 原理,实现控制发光二极管开、关、调亮、调暗的功能。
- 学习通过 PyTorch 搭建人工神经网络模型的方法。
- 学习通过卷积神经网络(CNN)实现简单手势识别的方法,掌握调整 CNN 网络参数的方法。

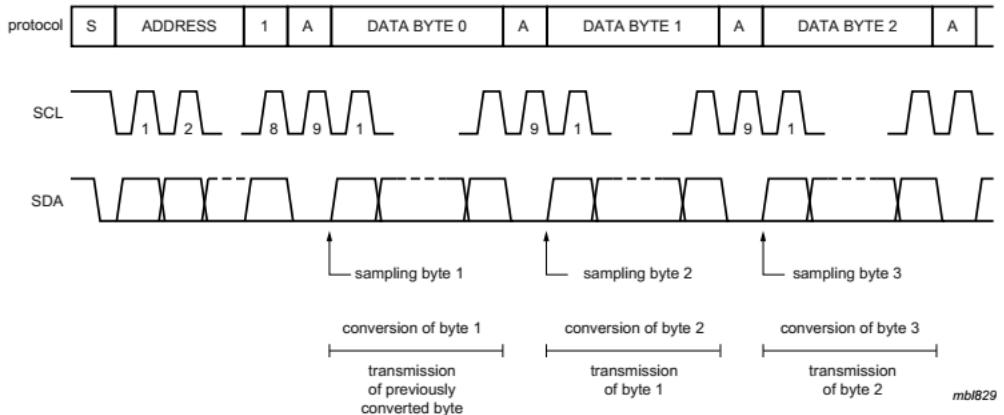


图 8.16: PCF8591 A/D 接口时序

- 实现通过手势识别实时控制发光二极管亮度的功能。

### 8.6.2 实验内容

#### 8.6.2.1 AD/DA 模块的检验及发光二极管的控制

- 使用面包板,对电位器的电压信号进行采样并存储,输出信号的电压信息。参考电路如图 8.17。
- 调整 DA 输出,点亮发光二极管,设计二极管的发光模式,如开、关、调亮、调暗等功能。参考电路如图 8.17。
- 用电位器的输出控制发光二极管的明暗。

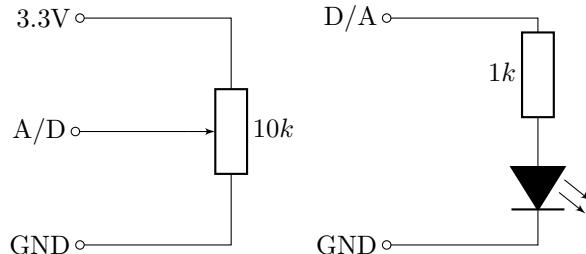


图 8.17: 滑动变阻器电路和发光二极管电路

#### 8.6.2.2 通过 OpenCV 获取训练样本并建立训练集和测试集

这部分已提供相对完整的代码及已经建立的数据集,可以初步使用提供的训练集和测试集进行后续实验,如效果不好,可尝试重新建立自己手势的数据集,需时 30-50 分钟。

- 读取树莓派的摄像头,完成手势样本的捕捉

利用摄像头每 0.5 秒捕捉一次图片,手在摄像头前做出一种手势,在摄像头捕捉期间不断改变手势的角度方位,一共捕捉 4 类每类 500 张图片。将捕捉到的图片经过灰度化、调整大小、高斯模糊,用编号命名保存到指定手势的文件夹下。

```

1 # 此代码用于获取训练集与测试集样本
2 # 使用摄像头取样,按颜色空间提取肤色,经灰度化调整大小模糊化后保存
3 # 运行代码,将出现两个cv2输出窗口,分别显示摄像头捕捉的原图及处理后的图像的实时
   预览

```



图 8.18: 捕获样本后进行的处理

```

4 # 待准备好后按q键，将开始以0.5秒为间隔捕捉样本，进度将在终端显示，可按w键停止
5
6 # 设置样本保存的地址、样本命名起始编号、需要获得的样本数
7 # 可以提前建好相关文件夹
8 FOLDER_NAME = './dataset/train/Five' # train, test / Five, Palm, Left, Right
9 START_INDEX = 1
10 CAPTURE_CNT = 1000
11
12 import numpy as np
13 import cv2
14 import time
15
16 cam = cv2.VideoCapture(0)
17
18 # 开始捕捉前对效果进行预览
19 while True:
20     ret, frame = cam.read()
21     HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) # HSV颜色空间的图像
22     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # 灰度化的图像
23     image_mask = cv2.inRange(HSV, np.array([0, 0, 0]), np.array([50, 200, 160]))
24         # 实验室相机的肤色
25     output = cv2.bitwise_and(gray, gray, mask=image_mask) # 按颜色空间提取手

```

```

25     output = cv2.resize(output, (80, 60)) # 缩小图片
26     output = cv2.blur(output, (2, 2)) # 模糊处理
27     cv2.imshow('orig', frame) # 显示预览
28     cv2.imshow('gray', output)
29     if cv2.waitKey(1) == ord('q'): # 按q退出预览
30         break
31
32 # 开始捕捉样本
33 index = START_INDEX # 样本命名的编号, 从START_INDEX以1为步距往上递增
34 while index < START_INDEX + CAPTURE_CNT: # 取样CAPTURE_CNT个
35     print(f'process:{index}/{CAPTURE_CNT}')
36     ret, frame = cam.read()
37     HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
38     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
39     # image_mask = cv2.inRange(HSV, np.array([0, 48, 80]), np.array([20, 255,
40         255])) # 自己电脑相机
41     image_mask = cv2.inRange(HSV, np.array([0, 0, 0]), np.array([50, 200, 160]))
42         ) # 实验室相机
43     output = cv2.bitwise_and(gray, gray, mask=image_mask)
44     output = cv2.resize(output, (80, 60))
45     output = cv2.blur(output, (2, 2))
46     cv2.imshow('orig', frame)
47     cv2.imshow('gray', output)
48     cv2.imwrite(f'{FOLDER_NAME}/{index}.png', output) # 保存到FOLDER_NAME下
49     index += 1
50     time.sleep(0.5) # 取样间隔
51     if cv2.waitKey(1) == ord('w'): # 可按w中断
52         break
53 cv2.destroyAllWindows()
54 cam.release()

```

## 2. 将捕捉到的样本平分为训练集与测试集

建立文件夹“dataset”，含子文件夹“train”与“test”，两个文件夹分别各含四个子文件夹“Palm”、“Five”、“Left”、“Right”，每个子文件夹下有 250 份样本。下图为部分样本的展示。

### 8.6.2.3 搭建神经网络模型并调整网络参数

- 搭建神经网络，采用“卷积-池化-卷积-池化-一维化-全连接-输出”的结构，训练采用小批梯度下降及 Adam 优化器，损失函数使用交叉熵。

这里给出一个示例代码，对应的网络结构是“卷积 1-池化 1-卷积 2-池化 2-一维化-全连接 1-全连接 2”：

```

1 # 此代码用于训练及测试神经网络
2 # 需要在此代码所在的目录下存放一个数据集文件夹
3 # 数据集文件夹包含 "train"、"test" 两个子文件夹
4 # 两个子文件夹下均有四个以手势类型命名的文件夹，存放着样本
5 # 训练部分使用小批量梯度下降、Adam 优化器
6 # 使用 tensorboard 的 SummaryWriter 记录每轮训练后在训练集、测试集上的损失、准确率
    等情况

```



图 8.19: 四种手势的样本示例

```

7 # SummaryWriter、训练好的模型将保存在数据集文件夹下
8 # 测试部分将给出模型在测试集上的混淆矩阵
9 # 混淆矩阵图片将保存在数据集文件夹下
10
11 DATA_FOLDER = 'dataset' # 数据集文件夹名
12 DO_TRAIN = True # 是否进行训练，True需要有数据集
13 DO_TEST = True # 是否进行测试，True需要DO_TRAIN为True或数据集文件夹下已存在一个网络模型
14 SAVE_MODEL = True # 是否保存训练好的模型，True需要DO_TRAIN为True
15 SAVE_CMFIG = True # 是否保存混淆矩阵，True需要DO_TEST为True
16 LEARNING_RATE = 0.01 # Adam优化器学习率
17 MAX_EPOCH = 10 # 进行epoch数
18 BATCH_SIZE = 10 # 每批含样本数
19
20 import numpy as np
21 import matplotlib.pyplot as plt
22 import torch
23 import torchvision
24 import torchvision.transforms as transforms
25 import torch.nn as nn
26 import torch.optim as optim
27 import torch.nn.functional as F
28 import time
29 import itertools
30 from torch.utils.tensorboard import SummaryWriter
31
32 writer = SummaryWriter() # 用于记录训练
33 torch.set_grad_enabled(True)
34
35 train_set = torchvision.datasets.ImageFolder(f'./{DATA_FOLDER}/train',
36     transform=transforms.Compose(
37         [transforms.Grayscale(1), transforms.ToTensor()]))) # 从数据集文件夹导入训练样本，灰度化并转为张量
38 train_loader = torch.utils.data.DataLoader(train_set, batch_size=BATCH_SIZE,
39

```

```
38                                         shuffle=True) # 训练集加载器，随机  
39                                         打乱训练集并以BATCH_SIZE个为一批  
40  
41 # 神经 网络  
42 # 采用“输入-卷积-池化-卷积-池化-一维化-全连接1-全连接2-输出”的结构  
43 # 激活函数为ReLU  
44 # 图片原始大小为80*60  
45 class Network(nn.Module):  
46     def __init__(self):  
47         super().__init__()  
48         self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5,  
49                             stride=1)  
50         self.conv2 = nn.Conv2d(in_channels=6, out_channels=12, kernel_size=5,  
51                             stride=1)  
52         self.fc1 = nn.Linear(in_features=12 * 3 * 2, out_features=60)  
53         self.fc2 = nn.Linear(in_features=60, out_features=30)  
54         self.out = nn.Linear(in_features=30, out_features=4)  
55  
56     def forward(self, t):  
57         t = F.relu(self.conv1(t)) # ->76*56  
58         t = F.max_pool2d(t, kernel_size=4, stride=4) # ->19*14  
59         t = F.relu(self.conv2(t)) # ->15*10  
60         t = F.max_pool2d(t, kernel_size=4, stride=4) # ->3*2  
61         t = t.reshape(-1, 12 * 3 * 2)  
62         t = F.relu(self.fc1(t))  
63         t = F.relu(self.fc2(t))  
64         return self.out(t)  
65  
66 # 训练部分  
67 def train():  
68     network = Network()  
69     optimizer = optim.Adam(network.parameters(), lr=LEARNING_RATE)  
70     images, labels = next(iter(train_loader)) # 从前面的加载器中按批获得样本  
71  
72     # 在tensorboard中记录一批样本的图像，添加网络，以监视样本或网络可能的异常  
73     grid = torchvision.utils.make_grid(images) # 将一批样本做成网格  
74     tb = SummaryWriter(f'./{DATA_FOLDER}') # SummaryWriter将保存在数据集文件夹  
75     下  
76     tb.add_image('images', grid) # 添加样本网格  
77     tb.add_graph(network, images) # 添加网络  
78  
79     # 比较网络对一批样本的预测preds与样本的真实标签labels，返回预测正确的个数  
80     def get_num_correct(preds, labels):  
81         return preds.argmax(dim=1).eq(labels).sum().item()  
82  
83     # 训练MAX_EPOCH轮  
84     for epoch in range(MAX_EPOCH):
```

```
83     t0 = time.time() # 记录用时
84     total_loss = 0 # 记录训练集上的损失
85     total_correct = 0 # 记录训练集上的正确个数
86     for batch in train_loader: # 按批进行
87         images, labels = batch
88         preds = network(images) # 获得目前网络对这一批的预测
89         loss = F.cross_entropy(preds, labels) # 计算交叉熵
90         optimizer.zero_grad() # 梯度清零，避免循环时backward()累加梯度
91         loss.backward() # 反向传播求解梯度
92         optimizer.step() # 更新参数
93         total_loss += loss.item() # 更新训练集上的损失
94         total_correct += get_num_correct(preds, labels) # 更新训练集上正确
95         # 一个epoch完成，开始在测试集上看效果
96         pred_set = torchvision.datasets.ImageFolder(f'./{DATA_FOLDER}/test',
97             transform=transforms.Compose(
98                 [transforms.Grayscale(1), transforms.ToTensor()])) # 按和前面
99             train_set一样的方法获得测试集pred_set
100            prediction_loader = torch.utils.data.DataLoader(pred_set, batch_size=
101                BATCH_SIZE) # 以及对应加载器
102            p_total_loss = 0 # 记录在训练集上的损失
103            p_total_correct = 0 # 记录训练集上总正确数
104            for batch in prediction_loader:
105                images, labels = batch
106                preds = network(images)
107                loss = F.cross_entropy(preds, labels)
108                p_total_loss += loss.item()
109                p_total_correct += get_num_correct(preds, labels)
110
111            # 在终端显示epoch数、准确率、用时，监视训练进程
112            print('epoch', epoch + 1, 'total_correct:',
113                total_correct, 'loss:', total_loss)
114            print('train_set_accuracy:', total_correct / len(train_set))
115            print('test_set_accuracy:', p_total_correct / len(pred_set))
116            print('time_spent:', time.time() - t0)
117
118            # 在SummaryWriter中分别记录训练集和测试集的损失、准确率、正确数信息
119            tb.add_scalar('Loss', total_loss, epoch)
120            tb.add_scalar('Prediction_Loss', p_total_loss, epoch)
121            tb.add_scalar('Number_Correct', total_correct, epoch)
122            tb.add_scalar('Prediction_Number_Correct', p_total_correct, epoch)
123            tb.add_scalar('Accuracy', total_correct / len(train_set), epoch)
124            tb.add_scalar('Prediction_Accuracy', p_total_correct / len(pred_set),
125                epoch)
126            tb.close()
127            if SAVE_MODEL: # 保存模型在数据集文件夹下
128                torch.save(network, F'./{DATA_FOLDER}/network.pkl')
```

```
127 if DO_TRAIN:
128     train()
129
130
131 # 测试模型，绘制混淆矩阵
132 def test(network):
133     def get_all_preds(model, loader):
134         # 传入网络模型和加载器，获得模型对加载器中样本的全部预测
135         all_preds = torch.tensor([])
136         for batch in loader:
137             images, labels = batch
138             preds = model(images) # 按批获得预测
139             all_preds = torch.cat((all_preds, preds), dim=0) # 再拼接起来
140         return all_preds
141
142     with torch.no_grad(): # 测试时不需要计算梯度等数据，节省资源
143         pred_set = torchvision.datasets.ImageFolder(f'./{DATA_FOLDER}/test',
144                                                       transform=transforms.Compose(
145                                                       [transforms.Grayscale(1), transforms.ToTensor()])) # 导入测试集
146         prediction_loader = torch.utils.data.DataLoader(pred_set, batch_size=BATCH_SIZE) # 对应加载器，由于是测试，无需shuffle
147         train_preds = get_all_preds(network, prediction_loader) # 获取网络对测试集数据的预测
148         total_types = len(pred_set.classes) # 总分类数
149         stacked = torch.stack((torch.tensor(pred_set.targets), train_preds.argmax(dim=1)), dim=1) # 拼接测试样本的真实类型与预测类型
150         cmt = torch.zeros(total_types, total_types, dtype=torch.int32) # 初始化混淆矩阵
151         for p in stacked: # 根据测试样本的真实类型与预测类型，在混淆矩阵的对应位置计数+1
152             train_label, predicted_label = p.tolist()
153             cmt[train_label, predicted_label] = cmt[train_label, predicted_label] +
154             1
155
156         def plot_confusion_matrix(cm, classes, title='Confusion_matrix', cmap=plt.cm.Blues):
157             # 传入ndarray型的混淆矩阵，分类名，标题，配色；绘制上文得到的混淆矩阵
158             plt.imshow(cm, interpolation='nearest', cmap=cmap)
159             plt.title(title)
160             plt.colorbar() # 设置颜色渐变条
161             tick_marks = np.arange(len(classes)) # 图像有分类数个刻度
162             plt.xticks(tick_marks, classes, rotation=45) # 用分类名标签xy刻度
163             plt.yticks(tick_marks, classes)
164             thresh = cm.max() / 2.
165             # 在表格上对应位置显示数字，为可视化效果，以其中最大数据的一半为界确定
166             # 文字颜色的黑白
167             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
168                 plt.text(j, i, format(cm[i, j], 'd'), horizontalalignment="center",
169                           color="white" if cm[i, j] > thresh else "black")
```

```

167     plt.tight_layout() # 填充图像区域
168     plt.ylabel('True_label') # 命名坐标
169     plt.xlabel('Predicted_label')
170
171     # 绘图
172     plt.figure(figsize=(4, 4))
173     plot_confusion_matrix(cmt.numpy(), pred_set.classes)
174     if SAVE_CMFIG: # 将混淆矩阵图片保存在数据集文件夹下
175         plt.savefig(f'./{DATA_FOLDER}/confusion_matrix.png')
176     plt.show()
177
178
179 if DO_TEST:
180     test(torch.load(f'./{DATA_FOLDER}/network.pkl'))

```

这个网络从第三轮开始在测试集上就出现了过拟合的情况:

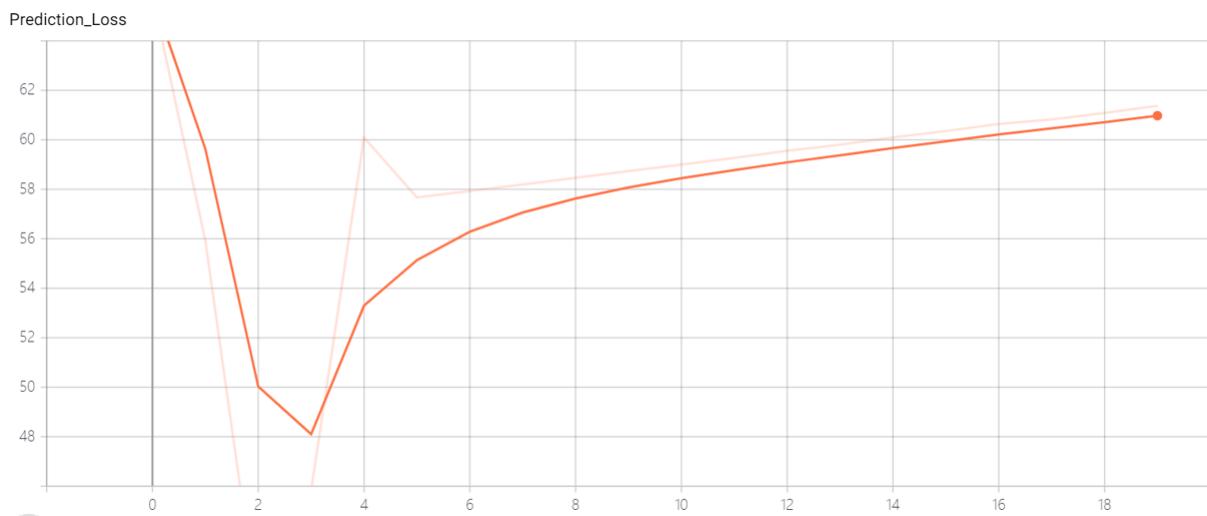


图 8.20: 过拟合情况示例,从第三轮开始

请根据示例代码,调整网络结构和参数,尝试删除一个全连接层,以实现“卷积-池化-卷积-池化-一维化-全连接-输出”的结构。

## 2. 测试与调整网络

如示例程序所示,在训练集上每训练一轮后用模型过一遍测试集,得到测试集的 loss 随 epoch 变化,利用 tensorboard 记录相关数据。训练完成后计算准确率并绘制混淆矩阵(如图所示)。用这些记录判断训练的效果,例如是否出现了过拟合,再依此调参或更改网络结构。

在前边已搭建的神经网络结构上,尝试调小池化大小( $4*4 \rightarrow 2*2$ ),调大卷积核步长( $1 \rightarrow 2$ ),并可对 batch size、learning rate 还有每层网络神经元个数等相关参数进行一些调整,以取得正确率大于 93% 的网络。

### 8.6.2.4 通过手势识别实时控制发光二极管亮度

导入调至较好效果的神经网络。与通过 OpenCV 获取训练样本时类似,用摄像头捕捉图片,同样经过灰度化、调整大小、高斯模糊处理,将实时样本送进神经网络,可得到神经网络对各类手势的评分及最后判定结果,根据判定结果分别控制发光二极管的开、关、调亮、调暗。

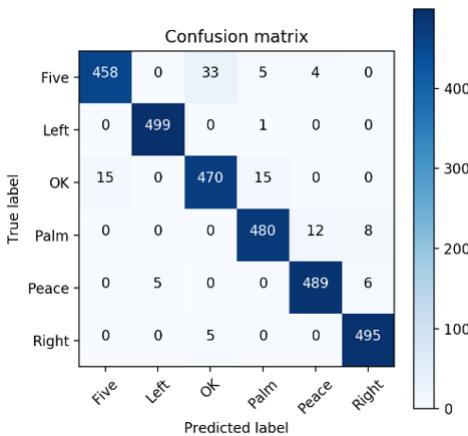


图 8.21: 在测试集上实测的混淆矩阵示例

神经网络导入的部分参考代码如下:

```

1 # 运行时需要在同目录下放置已训练好的“network.pkl”
2 import cv2
3 import numpy as np
4 import torch
5 import torch.nn as nn
6 import torch.nn.functional as F
7
8 # 神经网络同训练时的网络结构
9 class Network(nn.Module):
10     def __init__(self):
11         super().__init__()
12         self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride
13             =1)
14         self.conv2 = nn.Conv2d(in_channels=6, out_channels=12, kernel_size=5,
15             stride=1)
16         self.fc1 = nn.Linear(in_features=12 * 3 * 2, out_features=60)
17         self.fc2 = nn.Linear(in_features=60, out_features=30)
18         self.out = nn.Linear(in_features=30, out_features=4)
19
20     def forward(self, t):
21         t = F.relu(self.conv1(t)) # ->76*56
22         t = F.max_pool2d(t, kernel_size=4, stride=4) # ->19*14
23         t = F.relu(self.conv2(t)) # ->15*10
24         t = F.max_pool2d(t, kernel_size=4, stride=4) # ->3*2
25         t = t.reshape(-1, 12 * 3 * 2)
26         t = F.relu(self.fc1(t))
27         t = F.relu(self.fc2(t))
28         return self.out(t)
29
30 cam = cv2.VideoCapture(0)
31 network = torch.load('network.pkl')

```

通过神经网络对实时样本进行分类和判定的部分参考代码如下:

```
1 data = torch.tensor([[output]], dtype=torch.float) # 将获取的实时样本转为可传入网  
络的tensor  
2 pred_scores = network(data) # 获取各类型的分数  
3 prediction = pred_scores.argmax(dim=1).item() # 取最大者为结果
```

### 8.6.3 注意事项

#### 1. 图像色调与光照的影响

因为是按照颜色空间提取手，所以具体的效果和相片的色调有很大关系。面向强光时由于是按颜色空间提取肤色，可能出现较亮的部分无法提取到的情况，进而影响识别。

#### 2. 针对性添加样本和构建数据集

如果能够用自己的手重新构建数据集，识别率可能会进一步提高，不过需要花更多的时间。

## 8.7 思考题

#### 1. 如何改进容易出现误判或得分较低的姿势的正确率？

# 第九章 语音识别和处理入门

除了占 70% 的视觉信息外, 听觉信息是智能体的第二大信息来源。为此我们在本单元介绍树莓派音频的输入输出方法, 以及语音识别方面的一个经典模型——HMM-GMM 模型。最后动手实验实现用按键控制录音, 并在此基础上, 基于语音识别方法完成一个简单的语音控制系统。

## 9.1 树莓派音频输入输出方法

树莓派提供了音频输入和输出接口, 其中音频输出使用 3.5mm 耳机接口, 音频输入可以使用 USB 2.0 接口接入 USB 麦克风。

实验采用免驱 USB 麦克风, 直接插入树莓派上的 USB 2.0 即可, 不需要再安装驱动, 可以录制一段音频, 验证麦克风是否正常工作。首先, 使用 arecord -l 可以列出所有录音设备, 一般输出如下:

```
**** List of CAPTURE Hardware Devices ****
card 2: Device [USB PnP Sound Device], device 0: USB Audio [USB Audio]
  Subdevices:1/1
    Subdevice #0: subdevice #0
```

同样地, aplay -l 可以列出所有播放设备。

可执行 Linux 自带的录音/播放命令, 测试硬件是否正常:

```
sudo arecord -D "plughw:1,0" -d 5 test.wav
aplay -D hw:1,0 test.wav
```

arecord 是录音命令, 其中 hw:2,0 表示 arecord -l 命令中列出的 card 2、device 0, 如果你的 USB 声卡录音设备与这里显示不同, 还要进行相应修改, aplay 命令也是如此。

另外树莓派还带有一个具有硬件加速的视频播放程序—omxplayer—这里也可以用它进行播放:

```
omxplayer -o local test.wav
```

如果播放音频没有声音, 还可以通过 sudo raspi-config 设置当前音频播放设备为 Headphones 看是否可以解决。具体设置路径在 Advanced Options 下面的 Audio 菜单中。

在 Python 中执行录音命令需要 pyaudio 模块, 要使用 PyAudio, 首先使用 pyaudio.PyAudio() 实例化 PyAudio; 要录制或播放音频, 使用 pyaudio.PyAudio.open() 在设备上打开所需音频参数的流, 即设置了 pyaudio.Stream:

```
1  CHUNK = 1024
2  FORMAT = pyaudio.paInt16
3  CHANNELS = 2
4  RATE = 44100
5  p = pyaudio.PyAudio()
6  stream = p.open(format=FORMAT,
7                   channels=CHANNELS,
8                   rate=RATE,
```

```

9     input=True,
10    frames_per_buffer=CHUNK)

```

使用 pyaudio.Stream.write() 或 pyaudio.Stream.read() 录制或播放音频。需要注意的是，在“阻塞模式”，每个 pyaudio.Stream.write() 或 pyaudio.Stream.read() 阻塞直到操作完成；要动态生成音频数据或立即处理正在录制的音频数据，需要使用“回调”模式。使用 pyaudio.Stream.stop\_stream() 暂停播放/录制，并 pyaudio.Stream.close() 终止流。最后，使用 pyaudio.PyAudio.terminate() 终止 portaudio 会话。录制的数据以.wav 格式保存，需要调用 wave 库：

```

1   wf = wave.open(filename, 'wb')
2   wf.setnchannels(CHANNELES)
3   wf.setsampwidth(p.get_sample_size(FORMAT))
4   wf.setframerate(RATE)
5   wf.writeframes(data)

```

## 9.2 HMM-GMM 模型介绍

语音识别任务通常需要将一段音频转化成相应的文字，从数学角度上看，也就是求一段音频属于哪段文字的概率最大。传统语音识别框架中，所谓声学模型就是把语音的声学特征分类对应到（解码）音素或字词这样的单元；语言模型接着把字词解码成一个完整的句子。定义声学模型输入为  $O = (o_1, o_2, \dots, o_T)$ ，对应的句子  $W = (w_1, w_2, \dots, o_N)$ ，语音识别的任务就是求概率  $P(W|O)$  最大时对应的字序列  $W^*$ 。要计算  $P(W|O)$ ，可以利用贝叶斯公式：

$$P(W|O) = \frac{P(O|W)P(W)}{P(O)}$$

其中  $P(O)$  在计算时可以忽略，这样就得到了两部分  $P(O|W)P(W)$ ，分别对应于传统语音识别框架中的声学模型和语言模型，前者的任务是计算给定文字之后发出这段语音的概率；后者表示某一字词序列发生的概率。

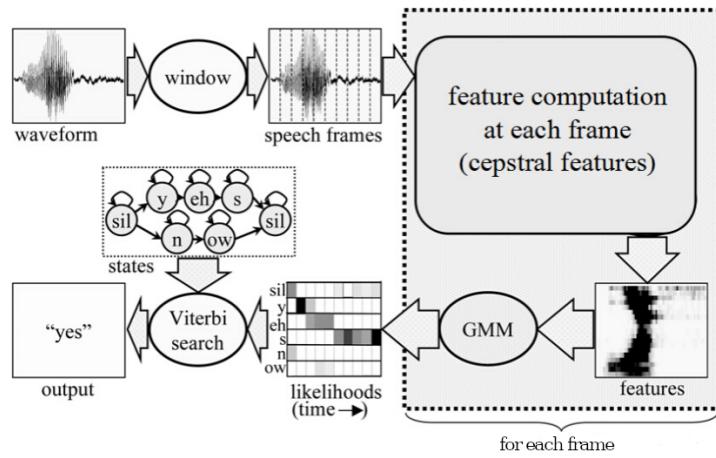


图 9.1: HMM-GMM 模型语音识别示意图

在命令词识别任务中，只需要使用到声学模型，本实验采用传统的 HMM-GMM (hidden Markov model and Gauss mixture model 隐马尔可夫与高斯混合) 模型进行识别。一个 HMM-GMM 模型对应一个孤立词，首先对输入的语音进行分帧，对每帧计算 MFCC (Mel Frequency Cepstral Coefficients 梅尔频率倒谱系数) 特征，得到一组描述语言信号能量在不同频率范围的分布的特征向量，之后对 HMM-GMM 模型进行训练。解码过程一般采用 Viterbi 算法(用于解决多步骤每步多选择模型的最优选择问题或篱笆

型图的最短路径问题), 输入一组特征向量, 对每帧用 GMM 计算隐藏状态的概率值, 结合 HMM 的转移概率, 用 Viterbi 算法进行路径搜索, 得到最大概率值, 这样就获得了最后的识别结果。

本实验只涉及最简单的案例——10 个独立词的识别。程序上, 以测试集为例, 音频的类别为文件名下划线后面的数字, 例如 1\_1.wav。

首先, 进行数据预处理, 输入为训练集路径或者测试集路径, 预处理程序 gen\_wavlist 分别得到音频文件字典 wavdict 及相应的标注字典 labeldict:

```

1 wavdict:
2 {'1_1': 'test_data\\1_1.wav', '1_10': 'test_data\\1_10.wav', '1_2': 'test_data\\1_2.
3 .wav',
4 '1_3': 'test_data\\1_3.wav', '1_4': 'test_data\\1_4.wav', '1_5': 'test_data\\1_5.
5 .wav',
6 '1_6': 'test_data\\1_6.wav', '1_7': 'test_data\\1_7.wav', '1_8': 'test_data\\1_8.
7 .wav',
8 '1_9': 'test_data\\1_9.wav'}
labeldict:
9 {'1_1': '1', '1_10': '10', '1_2': '2', '1_3': '3', '1_4': '4', '1_5': '5', '1_6': '6',
10 '1_7': '7', '1_8': '8', '1_9': '9'}
```

之后的特征提取直接调用 python\_speech\_features 实现 MFCC 算法:

```

1 from python_speech_features import mfcc
2 from scipy.io import wavfile
3 # 特征提取, feat = compute_mfcc(wadict[wavid])
4 def compute_mfcc(file):
5     fs, audio = wavfile.read(file)
6     mfcc_feat = mfcc(audio)
7     return mfcc_feat
```

我们利用 hmmlearn 工具包搭建 HMM-GMM, 因为需要识别 10 个独立词, 需要初始化 10 个独立的 HMM-GMM 模型, 即一个 HMM-GMM 模型的集合 self.models:

```

1 from hmmlearn import hmm
2 class Model():
3     def __init__(self, CATEGORY=None, n_comp=3, n_mix = 3, cov_type='diag',
4                  n_iter=1000):
5         super(Model, self).__init__()
6         self.CATEGORY = CATEGORY
7         self.category = len(CATEGORY)
8         self.n_comp = n_comp
9         self.n_mix = n_mix
10        self.cov_type = cov_type
11        self.n_iter = n_iter
12        # 关键步骤, 初始化models, 返回特定参数的模型的列表
13        self.models = []
14        for k in range(self.category):
15            model = hmm.GMMHMM(n_components=self.n_comp, n_mix = self.n_mix,
16                                covariance_type=self.cov_type, n_iter=self.n_iter)
17            self.models.append(model)
```

各个参数的意义:

- CATEGORY: 所有标签的列表
- n\_comp: 每个孤立词中的状态数
- n\_mix: 每个状态包含的混合高斯数量
- cov\_type: 协方差矩阵的类型
- n\_iter: 训练迭代次数

在语音处理中,每个 HMM 对应于一个词 (word),一个词由若干音素 (phoneme) 组成。一个词 (word) 表示成若干状态 (states), 每个状态 (state) 表示为一个音素; 汉语的词一般由 5 个状态组成, 英语的为 3 个。用混合高斯密度函数去表示每个状态的出现概率, 只需要求出其均值和协方差就可以了。在 HMM-GMM 模型中, 如果观测序列是一维的, 则观测状态的概率密度函数是一维的普通高斯分布。如果观测序列是 N 维的, 则隐藏状态对应的观测状态的概率密度函数是 N 维高斯分布。高斯分布的概率密度函数参数可以用  $\mu$  表示高斯分布的期望向量,  $\Sigma$  表示高斯分布的协方差矩阵。在 GaussianHMM 类中, “means” 用来表示各个隐藏状态对应的高斯分布期望向量 形成的矩阵, 而“covars” 用来表示各个隐藏状态对应的高斯分布协方差矩阵  $\Sigma$  形成的三维张量。参数 covariance\_type, 取值为“full” 意味所有的  $\mu, \Sigma$  都需要指定。取值为“spherical”则  $\Sigma$  的非对角线元素为 0, 对角线元素相同。取值为“diag”则  $\Sigma$  的非对角线元素为 0, 对角线元素可以不同, “tied” 指所有的隐藏状态对应的观测状态分布使用相同的协方差矩阵  $\Sigma$ 。

然后,用同一种类的数据训练特定的模型:

```

1 # 提取声学特征
2 mfcc_feat = compute_mfcc(wavdict[x])
3 # hmm-gmm 模型训练
4 model.fit(mfcc_feat)

```

模型训练完毕后,对待测试的数据分别用十个模型打分,选出得分最高的为识别结果。

```

1 # 提取声学特征
2 mfcc_feat = compute_mfcc(wavdict[x])
3 # 用模型打分
4 re = model.score(mfcc_feat)

```

为了让训练好的模型可以在后续的代码中使用, 可以使用 joblib 模块对模型参数进行保存, 例如保存和加载的代码如下所示:

```

1 def save(self, path="models.pkl"):
2     # 利用 joblib 保存生成的 hmm 模型
3     joblib.dump(self.models, path)
4
5 def load(self, path="models.pkl"):
6     # 导入 hmm 模型
7     self.models = joblib.load(path)

```

## 9.3 动手实验:语音控制系统

### 9.3.1 实验目的

1. 了解树莓派音频接口的使用方法。
2. 掌握语音识别基本算法。

### 9.3.2 实验内容

#### 9.3.2.1 测试麦克风

根据提供的代码, 测试麦克风录音效果。

#### 9.3.2.2 用按键控制录音

与对讲机的 PTT(Press to Talk)功能类似, 实现按键录音效果, 按下按键开始录音, 当按键抬起时, 录音结束。

#### 9.3.2.3 语音命令的训练与识别

生成用于训练和测试的语音命令(可以编写程序辅助文件的命名和整理), 每个命令至少 5 个音频用于训练, 命令个数不少于 5 个。使用 HMM-GMM 模型对语音命令进行识别。

#### 9.3.2.4 完成简单的语音控制系统

通过树莓派实现对控制命令的识别, 完成简单的语音控制系统。例如用来控制 LED 灯的开关状态, 增加/减少 LED 的亮度, 用来输入数字等。

【选做】在识别命令时, 取消使用按键, 自动识别是否有命令, 并对命令做出反应。

【选做】尝试修改模型超参数, 并重新进行训练, 以提高识别率。

## 9.4 思考题

1. 如何改进语音识别的正确率?

# 第十章 自然语言处理入门

## 10.1 词向量的编码与处理方法

### 10.1.1 WordNet 简介

WordNet 是由一个由普林斯顿大学认知科学实验室在心理学教授乔治·A·米勒的指导下建立和维护的英语字典。WordNet 根据词条的意义将它们分组，每一个具有相同意义的词条组称为一个 synset（同义词集合）。WordNet 为每个 synset 提供了简短，概要的定义，并记录不同 synset 之间的语义关系。在 WordNet 中，名词、动词、形容词和副词各自被组织成一个同义词网络，其中名词网络的主干是蕴含关系的层次（上位/下位关系），占据了关系中的将近 80%，层次中的最顶层是 11 个抽象概念，称为基本类别始点（unique beginners），例如实体（entity）等，层次越深，对应的 synset 定义越具体。如图10.1上位（Hypernym）：一个词比给定单词更具有一般意义；下位（Hyponym）：一个词比给定单词有更具体的意义；部分（meronym）：一个词是给定单词的一部分；整体（holonym）：一个词是给定单词的全部或整体。

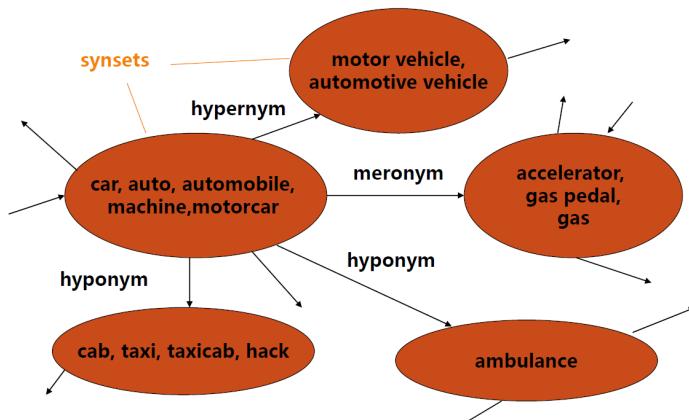


图 10.1: A WordNet Snapshot

### 10.1.2 基于语义词典的词汇相似度计算

词汇相似度（word similarity）指的是词汇间基于词义的关系，两个词汇之间具有越多的共性越相似。注意区分词汇相似度（word similarity）与词汇相关性（word relatedness），例如 car 与 bicycle 相似，而 car 与 gasoline 相关但不相似。使用语义词典计算词汇相似度时主要有两种方法，分别是 path based similarity 和 information content similarity，下面将会进行简单的介绍。

顾名思义，path based similarity 主要考虑两个词在词典层次结构中的相对位置，两个词在词典层次结构中越相邻，这两个词越相似。

图10.2 为 path based similarity 的一个示意，词汇间虚线连线上的数字表示它们之间路径的长度，路径越短代表相似度越高。在此基础上，有多种改进后的词汇相似度计算方法，例如 Wu-Palmer 相似度考虑

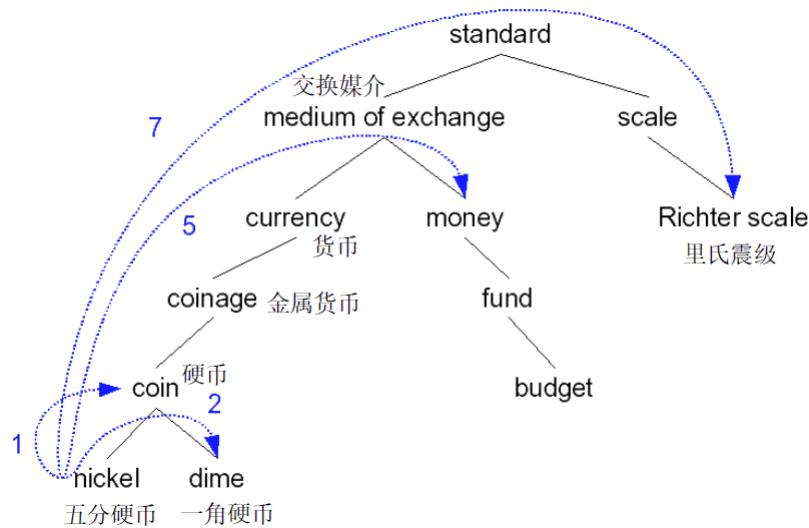


图 10.2: Path Based Similarity 示意

两个词汇和最低公共祖先(LCS, 又称最近公共父节点)的路径深度, 计算方法如下所示。

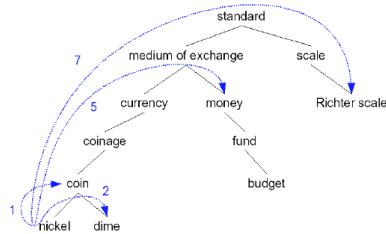
$$Sim_{wup} = \frac{2 * depth(LCS)}{depth(concept_1) + depth(concept_2)}$$

information content similarity 与上述方法稍有不同, 首先定义  $P(\text{concept})$  表示从一个语料库中随机选择一个词, 这个词属于概念 concept 的概率。

**words(c): 概念c所包容的词集 (包含子孙后代节点)**

**N: 词语总数**

$$P(c) = \frac{\sum_{w \in \text{words}(c)} count(w)}{N}$$

图 10.3:  $P(\text{concept})$  含义

$C(\text{concept}) = -\log P(\text{concept})$  表示概念 concept 包含的信息内容。Lin 相似度考虑词汇和它们的最低公共祖先所包含的信息内容来计算词汇相似度。

$$Sim_{lin} = \frac{2 * IC(LCS)}{IC(concept_1) + IC(concept_2)}$$

NLTK 包含众多一系列的语料库, 这些语料库可以通过 `nltk.package` 导入使用。每一个语料库可以通过一个叫做“语料库读取器”的工具读取语料库, 例如: `nltk.corpus`。前述 WordNet 词典也是 NLTK 语料库的一部分。

### 10.1.3 基于语料统计的词汇相似度计算

Word2Vec 是 google 在 2013 年推出的一个 NLP 工具, 它的特点是能够将单词转化为向量来表示, 这样词与词之间就可以定量的去度量他们之间的关系, 挖掘词之间的联系。Word2Vec 的思路是通过训练, 将原来 One-Hot 编码的每个词都映射到一个较短的词向量上来, 而这个较短的词向量的维度可以在训练时根据任务需要来指定, 用 Distributed Representation 表示的较短的词向量, 就可以较容易的分析词之

间的关系。(one-hot representation: 用一个很长的向量来表示一个词, 向量的长度为词典的大小, 向量的分量只有一个 1, 其他全为 0, 1 的位置对应该词在词典中的位置; distributed Representation: 通过训练将某种语言中的每一个词映射成一个固定长度的短向量, 将所有这些向量放在一起形成一个词向量空间, 而每一向量则为该空间中的一个点, 在这个空间上引入“距离”, 则可以根据词之间的距离来判断它们之间的(词法、语义上的)相似性。)

Word2Vec 的训练模型本质上是只具有一个隐含层的神经元网络。

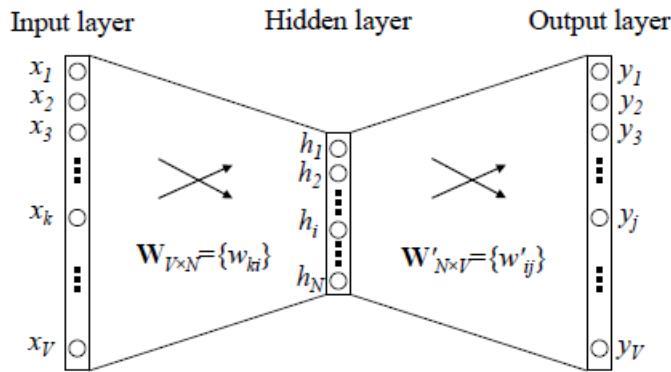


图 10.4: Word2Vec 模型结构

它的输入是采用 One-Hot 编码的词汇表向量, 它的输出也是 One-Hot 编码的词汇表向量。使用所有的样本, 训练这个神经元网络, 等到收敛之后, 从输入层到隐含层的那些权重, 便是每一个词的采用 Distributed Representation 的词向量。这样就把原本维数为 V 的词向量变成了维数为 N 的词向量(N 远小于 V), 并且词向量间保留了一定的相关关系。

Mikolov 在关于 Word2Vec 的论文中提出了 CBOW 和 Skip-gram 两种模型, CBOW 适合于数据集较小的情况, 而 Skip-Gram 在大型语料中表现更好。其中 CBOW 如图10.5左部分所示, 使用围绕目标单词的其他单词(语境)作为输入, 在映射层做加权处理后输出目标单词。与 CBOW 根据语境预测目标单词不同, Skip-gram 根据当前单词预测语境, 如图10.5右部分所示。假如我们有一个句子 “There is an apple on the table” 作为训练数据, CBOW 的输入为(is,an,on,the), 输出为 apple。而 Skip-gram 的输入为 apple, 输出为(is,an,on,the)。

利用 Word2Vec 模型, 得到词语的词向量表示后, 采用余弦相似度计算词语相似度。

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

gensim 是一款开源的第三方 Python NLP 工具包, 用于从原始的非结构化的文本中, 无监督地学习到文本隐层的主题向量表达。它支持包括 TF-IDF, LSA, LDA, 和 word2vec 在内的多种主题模型算法, 支持流式训练, 并提供了诸如相似度计算, 信息检索等一些常用任务的 API 接口。在 gensim 中, word2vec 相关的 API 都在包 gensim.models.word2vec 中, 和算法有关的参数都在类 gensim.models.word2vec.Word2Vec 中, 需要注意的参数有:

- 1) sentences: 我们要分析的语料, 可以是一个列表, 或者从文件中遍历读出;
- 2) size: 词向量的维度, 默认值是 100。这个维度的取值一般与我们的语料的大小相关, 如果是不大的语料, 比如小于 100M 的文本语料, 则使用默认值一般就可以了。如果是超大的语料, 建议增大维度。
- 3) window: 即词向量上下文最大距离, 这个参数在我们的算法原理篇中标记为 c, window 越大, 则和某一词较远的词也会产生上下文关系。默认值为 5。在实际使用中, 可以根据实际的需求来动态调整这个 window 的大小。如果是小语料则这个值可以设的更小。对于一般的语料这个值推荐在 [5,10] 之间。

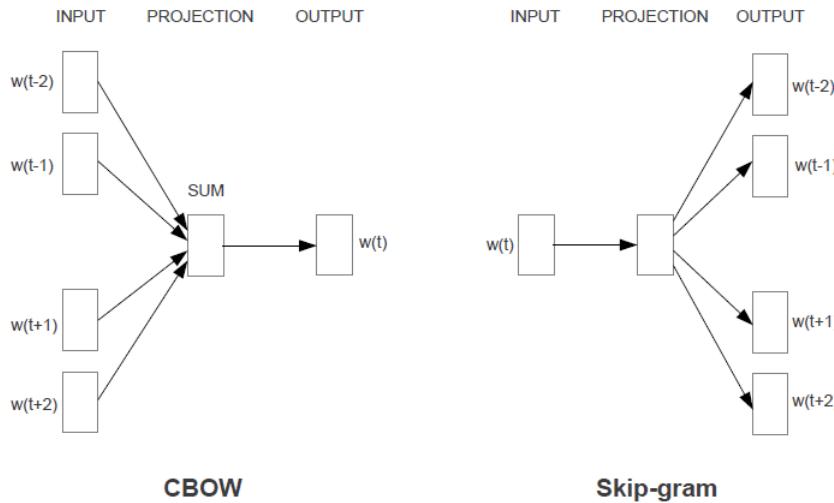


图 10.5: CBOW 模型与 Skip-gram 模型

4) sg: 即 word2vec 两个模型的选择。如果是 0，则是 CBOW 模型，是 1 则是 Skip-Gram 模型，默认是 0 即 CBOW 模型。

5) hs: 即 word2vec 两个解法的选择，如果是 0，则是 Negative Sampling，是 1 的话并且负采样个数 negative 大于 0，则是 Hierarchical Softmax。默认是 0 即 Negative Sampling。

6) negative: 即使用 Negative Sampling 时负采样的个数，默认是 5。推荐在 [3,10] 之间。

7) min\_count: 需要计算词向量的最小词频。这个值可以去掉一些很生僻的低频词，默认是 5。如果是小语料，可以调低这个值。

#### 10.1.4 结果评估方法——Spearman's rank correlation coefficient

Mturk-771 数据集提供了每对词语的相似度向量，需要与我们得到的词语相似度向量进行比较，评估各种方法效果优劣。pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的，提供了大量能使我们快速便捷地处理数据的函数和方法。pandas 中 DataFrame 对象的 corr() 方法用来计算 DataFrame 对象中所有列之间的相关系数 (df.corr())，包括 'pearson' 相关系数、'kendall' 相关系数和 'spearman' 秩相关)。相关性是两个变量之间关联的度量，用以检查两个变量之间变化趋势的方向以及程度，值范围 -1 到 +1，0 表示两个变量不相关，正值表示正相关，负值表示负相关，值越大相关性越强。当两个变量都有正态分布时，很容易计算和解释。而当我们不知道变量的分布时，我们必须使用非参数的秩相关 (Rank Correlation，或称为等级相关) 方法。秩相关是指使用变量之间序数的关联(而不是特定值)来量化变量之间的关联的方法。有序数据是具有标签值并具有顺序或秩相关的数据，例如：‘低’，‘中’ 和 ‘高’。Spearman 秩相关以 Charles Spearman 命名，这个统计方法量化了等级变量与单调函数相关联的程度，即递增或递减的关系。在没有重复数据的情况下，如果一个变量是另外一个变量的严格单调函数，那么二者之间的 spearman 秩相关系数就是 1 或 +1，称为完全 spearman 相关。如果 Y 随 X 的增加而增加，则 spearman 秩相关系数是正的，否则是负的。spearman 秩相关系数为 0 表示随着 X 的增加 Y 没有增加或减小的趋势，随着 X 和 Y 的越来越接近严格单调函数管系，spearman 秩相关系数在数值上越来越大。

由于基于词典和基于语料的方法中，都会有异常值的出现（信息内容文件没有条目，词典收入词条有限，语料中词汇有限），会对 Pearson correlation coefficient 的结果造成较大干扰，而 Spearman's rank correlation coefficient 对异常值不敏感。

## 10.2 Twitter 文本情感分析

### 10.2.1 情感分析介绍

情感分析是文本分类的一个分支,对带有情感色彩(正向/中立/负向)的主观性文本进行分析,以确定该文本的观点、喜好和情感倾向。主流方法有基于情感词典的情感分析和基于机器学习的情感分析两种方式。

基于情感词典的情感分析是指根据已构建的情感词典,对待分析文本进行文本处理,抽取情感词计算文本的情感倾向,最终的分类效果主要取决于情感词典的完善性。

基于机器学习的情感分析需要对文本进行一系列预处理(去除停用词、Stem、Tokenization 等),将文本转化为词向量表示后,可以利用逻辑回归、朴素贝叶斯及支持向量机等方法进行分类。

SemEval 数据集完成基本任务是推特的情感分析(Sentiment Analysis in Twitter)。对于推特的文本情感分析基于 SemEval 数据集始于 2013 年,之后任务和数据都在不断发展为更复杂。在 13 年到 15 年,任务是简单给一个推特文本,然后进行文本情感分类,分为 3 类(积极、消极、中立),称为任务 A;于 2015 年,在任务和任务中引入了 Topic 的概念,任务升级为给一个推特,并给一个 topic;推断推特内容关于这个 topic 的情感倾向,积极或消极(任务 B);于 2016 年,引入了两个分支,一是加入了 tweet quantification,也就是推特的量化分析;二是 five-point ordinal classification,也就是之前是推特的三分类,16 年拓展为五分类。

SemEval-2017 任务 4 由五个子任务组成,每个都提供阿拉伯语和英语:

- 1.Subtask A: 分析一个推特的情感,可以分为积极、消极、中立
- 2.Subtask B: 给一个推特,并给一个 topic;推断推特内容关于这个 topic 的情感倾向,积极或消极。
- 3.Subtask C: 在 B 任务的基础上,更加精细地分类,分为非常积极、弱倾向积极、中立、弱倾向于消极、非常消极(五个程度)
- 4.Subtask D: 关于一个 topic,给出一组的推特,估计这些推特在积极和消极的分布
- 5.Subtask E: 关于一个 topic,给出一组的推特,估计这些推特在五个情感程度的分布。

本单元基于 SemEval2017 Task 4 Subtask A: Message Polarity Classification,对给定的英文 twitter 进行情感分析,将文本划分为 positive, neutral 和 negative 三种情感。

### 10.2.2 双向 LSTM 介绍

LSTM 的全称是 Long Short-Term Memory,它是 RNN(Recurrent Neural Network)的一种。由于其设计的特点,LSTM 非常适合用于对时序数据(如文本数据)的建模。BiLSTM 是 Bi-directional Long Short-Term Memory 的缩写,是由前向 LSTM 与后向 LSTM 组合而成。两者在自然语言处理任务中都常被用来建模上下文信息。

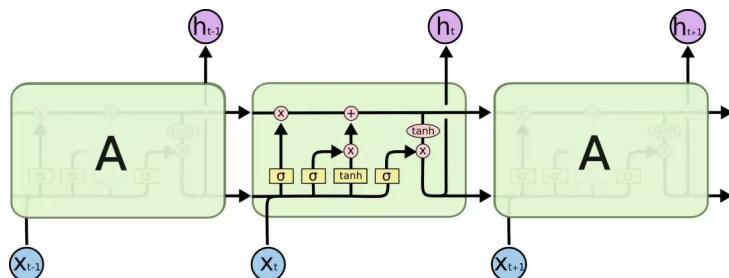


图 10.6: LSTM 总体框架

LSTM 的总体框架如图10.6 所示,由  $t$  时刻的输入词  $X_t$ ,细胞状态  $C_t$ ,临时细胞状态  $\tilde{C}_t$  隐层状态  $h_{t+1}$ ,遗忘门  $f_t$ ,记忆门  $i_t$  和输出门  $o_t$  组成。其计算过程可以概括为,通过对细胞状态中信息遗忘和记忆新的信息使得对后续时刻计算有用的信息得以传递,而无用信息被抛弃。

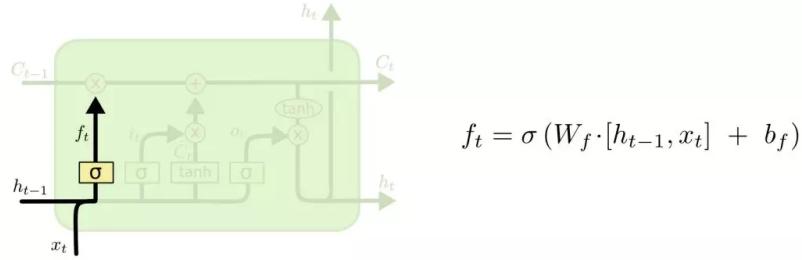


图 10.7: 遗忘门计算

首先计算遗忘门, 选择要遗忘的信息, 输入为前一层的隐层状态  $h_{t-1}$ , 当前时刻的输入词  $X_t$ , 输出为遗忘门的值  $f_t$ , 如图10.7所示。其中  $W_f$  为计算遗忘门对应的权重,  $b_f$  为偏置,  $\sigma$  为激活函数。

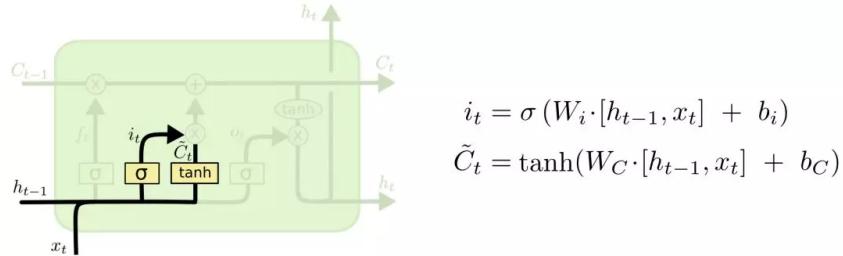


图 10.8: 记忆门和临时细胞状态计算

计算记忆门, 选择要记忆的信息, 输入为前一层的隐层状态  $h_{t-1}$ , 当前时刻的输入词  $X_t$ , 输出为记忆门的值  $i_t$  和临时细胞状态  $\tilde{C}_t$ , 如图10.8 所示。

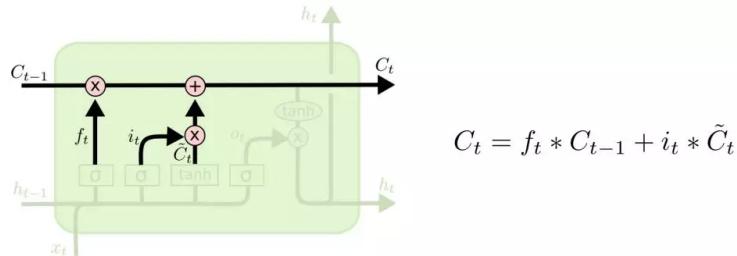


图 10.9: 当前时刻细胞状态计算

计算当前时刻细胞状态, 输入为记忆门的值  $i_t$ , 遗忘门的值  $f_t$ , 临时细胞状态  $\tilde{C}_t$  和上一时刻细胞状态  $C_{t-1}$ , 输出为当前时刻细胞状态  $C_t$ , 如图10.9 所示。

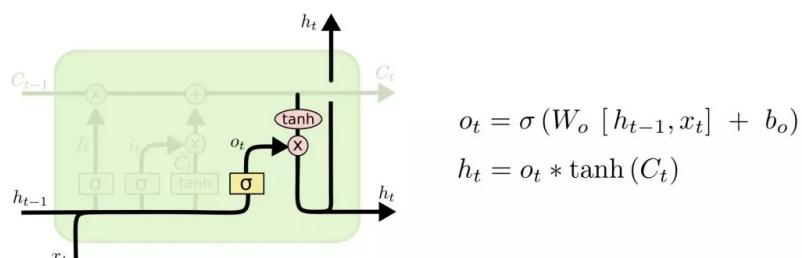


图 10.10: 计算输出门和当前时刻隐层状态

计算输出门和当前时刻隐层状态, 输入为前一时刻的隐层状态  $h_{t-1}$ , 当前时刻的输入词  $X_t$  和当前时刻细胞状态  $C_t$ , 输出为输出门的值  $o_t$  和隐层状态  $h_t$ , 如图 10.10 所示。

最终可以获得与句子长度相同的隐层状态序列  $h_0, h_1, \dots, h_{(n-1)}$ 。

但是 LSTM 存在一个问题，即只能编码从前到后的信息，而无法编码从后到前的信息。双向 LSTM 由一个前向 LSTM 和一个后向 LSTM 组成，因此具备了编码从后到前信息的能力。以编码“我爱中国”为例，如图10.11 所示，分词操作将句子划分为 [“我”，“爱”，“中国”] 三个词，前向 LSTM 得到  $h_{L0}, h_{L1}, h_{L2}$ ，后向 LSTM 得到  $h_{R0}, h_{R1}, h_{R2}$ ，将前向和后向的隐向量拼接得到  $[h_{L0}, h_{R0}], [h_{L1}, h_{R1}], [h_{L2}, h_{R2}]$ ，即  $h_0, h_1, h_2$ 。

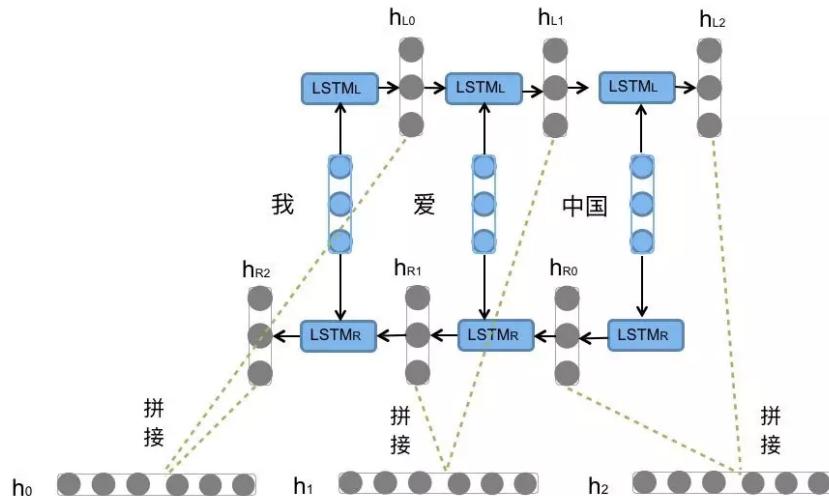


图 10.11: 双向 LSTM 编码句子示意

```

1 # PyTorch 双向 LSTM 模型定义
2 # bidirectional 控制是否是双向 LSTM
3 self.lstm = nn.LSTM(embedding_dim, hidden_dim, layer_num, dropout=drop_prob,
4 batch_first=True, bidirectional=self.bidirectional)

```

利用双向 LSTM 编码文本信息后，使用全连接层进行文本的情感分类。

```

1 # 全连接层做情感分类
2 self.fc = nn.Linear(self.layer_num * hidden_dim, output_size)

```

PyTorch 提供的 LSTM 函数包括 7 个参数，含义如下：

- input\_size: 输入特征维数，与示例中 embedding\_dim 对应，代表每个单词用多少维向量表示，通常选择数百量级的向量维度
- hidden\_size: 隐藏层的特征维度
- num\_layers: LSTM 隐层的层数，默认为 1
- bias: 偏移值，表示隐层状态是否带 bias，默认为 True
- batch\_first: 为 True 时，表示输入输出的数据格式为 (batch, seq, feature)
- dropout: 除最后一层，每一层以特定的概率进行 dropout，默认为 0
- bidirectional: 表示是否是双向 LSTM

其他参数：

- vocab\_size: 表示根据当前语料构建的词典大小
- output\_size: 分类判别的取值数量，此处为 positive/neutral/negative 三种
- epochs: 表示模型训练迭代次数，需要根据数据集大小和模型拟合能力确定。设置太小会欠拟合，太大会过拟合

- clip: 为梯度裁剪阈值, 防止梯度爆炸/梯度消失
- learning\_rate: 为模型学习率, 设置过小容易使得训练速度过慢或陷入局部最优点, 过大可能导致震荡而无法收敛

### 10.3 未名教学二号集群使用说明

请至少提前一周申请未名教学二号集群账号: <http://hpc.pku.edu.cn/guide.html>, 申请成功后可以使用 ssh 或 Putty 远程登陆集群。

集群使用教程参见: [http://hpc.pku.edu.cn/\\_book/guide/soft\\_env/python.html](http://hpc.pku.edu.cn/_book/guide/soft_env/python.html), 重点参考“5. 使用软件“部分的”5.5.python”中的内容。

在集群上运行 Python 程序时, 每个人需要在 conda 里创建一个自己的虚拟环境, 再在自己的虚拟环境里运行, 所用到的库都需要自己安装, 但在集群上安装库比较方便。

```

1 module avial anaconda      # 查看可用的 conda 环境
2 module load anaconda/3.7.1   # 载入可用的 conda 环境
3 conda create -n myEnvPython python=<version>      # 创建虚拟环境
4 source activate myEnvPython # 进入已创建的虚拟环境

```

进入虚拟环境后通过 pip install 或 conda install 安装需要的库

任务提交方式: 参考上述链接中的“6. 作业调度系统”, 注意: 不能直接在集群上运行任务, 需要按教程中的方式编写脚本后, 通过提交任务的方式执行。

```

1 sinfo 查看可用的分区和节点
2 sbatch 提交任务
3 squeue 查看任务运行状态

```

脚本 job.sh 的示例如下:

```

1 #!/bin/bash
2 #SBATCH -o nohup.out
3 #SBATCH -p compute
4 #SBATCH -q normal
5 #SBATCH -J word2vec
6 #SBATCH --time=24:00:00
7
8 python3 main.py

```

通过 sbatch job.sh 提交任务, 通过 squeue 查看任务状态。

### 10.4 动手实验: 文本的情感分析

#### 10.4.1 实验目的

1. 学习词向量的编码与处理方法
2. 初步了解自然语言处理的基本方法
3. 学习计算机集群的远程登陆与作业管理

## 10.4.2 实验内容

### 10.4.2.1 词汇相似度计算实验

基于 Mturk-771 数据集进行实验,计算英文词汇的相似度,对比 3 种不同的词汇相似度计算方法的效果。Mturk-771 数据集存放在 data 文件夹中,数据格式为(word1, word2, sim),表示 word1 与 word2 的相似度为 sim。

Word2Vec 模型基于 text8(wikipedia)语料进行训练,该语料存放在 data 文件夹中。

utils.py 提供了读写.xlsx 文件、计算 Spearman's rank correlation coefficient 和绘图函数。

ContextBased.py 提供了基于 Word2Vec 模型的词汇相似度计算方法。

DictionaryBased.py 提供了基于语义词典的词汇相似度计算方法。

实验时,运行 main.py 即可。

```

1 # 返回基于语义词典的相似度
2 wupAns, linAns = DicSimilarity(rawData)
3 # 返回基于 Word2Vec 的相似度
4 ConAns = ConSimilarity(rawData, vecLength=150, isTrain=True)

```

评估方法采用 Spearman's rank correlation coefficient,理由是 Mturk-771 数据集提供了每对词语的相似度向量,需要与我们得到的词语相似度向量进行比较,评估各种方法效果优劣。

1)在树莓派上运行已训练好的模型,运行 main.py 即可,比较不同方法的优劣,同学们也可以尝试其它的基于语义词典的方法;

2)在未名二号教学集群上对 Word2Vec 模型进行重新训练,须调整 Word2Vec 模型的超参数,重新训练后,将模型下载到树莓派观察效果。

### 10.4.2.2 Twitter 文本情感分析实验

数据集存放在 data 文件夹中,data\_processor.py 文件中提供了文本预处理相关的函数。model.py 中定义了模型结构。

```

1 # 模型超参数设置
2 batch_size = 25
3 vocab_size = len(vocab_to_int) + 1
4 output_size = 3
5 embedding_dim = 400
6 hidden_dim = 256
7 n_layers = 2
8 epochs = 5
9 clip = 5
10 print_every = 10
11 learning_rate = 0.0005

```

1)在树莓派上运行已训练好的模型,运行 main.py 即可;2)感兴趣的同学可以定义自己的网络结构,或修改模型的超参数,在未名二号教学集群上进行重新训练,之后将模型下载到树莓派观察效果。

# 第十一章 TensorFlow 应用

## 11.1 TensorFlow 介绍

TensorFlow™ 是一个基于数据流编程(Dataflow Programming)的符号数学系统, 被广泛应用于各类机器学习(Machine learning)算法的编程实现, 其前身是谷歌的神经网络算法库 DistBelief。TensorFlow 的主要特点有:

- 使用图(graph) 来表示计算任务.
- 在被称之为会话(Session) 的上下文(context) 中执行图.
- 使用 tensor 表示数据.
- 通过变量(Variable) 维护状态.
- 使用 feed 和 fetch 可以为任意的操作(Arbitrary Operation) 赋值或者从其中获取数据

TensorFlow 代码结构如下:

```
1 import tensorflow as tf
2 b = tf.Variable(tf.zeros([100]))                      # 100-d vector, init to
3     zeroes
4 W = tf.Variable(tf.random_uniform([784,100], -1,1)) # 784x100 matrix w/rnd vals
5 x = tf.placeholder(name="x")                          # Placeholder for input
6 relu = tf.nn.relu(tf.matmul(W, x) + b)               # Relu(Wx+b)
7 C = [...]                                              # Cost computed as a
8     function of Relu
9 s = tf.Session()
10 for step in xrange(0, 10):
11     input = ...construct 100-D input array ...        # Create 100-d vector for
12         input
13     result = s.run(C, feed_dict={x: input})           # Fetch cost, feeding x=
14         input
15     print step, result
```

## 11.2 MNIST 入门

MNIST 是一个入门级的计算机视觉数据集, 它包含各种手写数字图片, 它也包含每一张图片对应的标签, 告诉我们这个是数字几。比如, 这四张图片的标签分别是 5, 0, 4, 1。

从一个很简单的数学模型开始(Softmax Regression), 了解如何使用 TensorFlow。

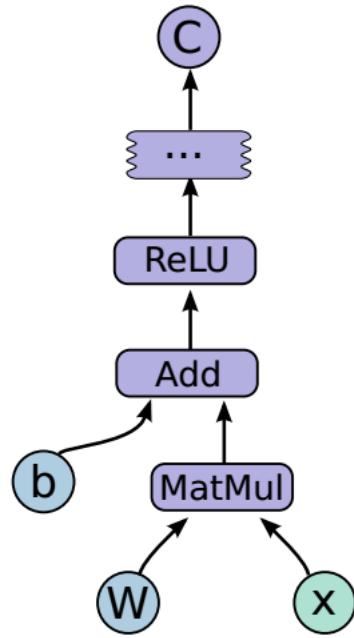


图 11.1: TensorFlow 计算图



图 11.2: MNIST 手写数字图片

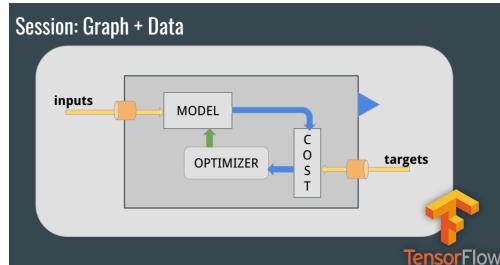


图 11.3: TensorFlow 工作流程

### 11.2.1 Softmax 回归介绍

MNIST 的每一张图片都表示一个从 0 到 9 的数字，希望得到给定图片代表每个数字的概率。比如说，我们的模型可能推测一张包含 9 的图片代表数字 9 的概率是 80% 但是判断它是 8 的概率是 5%，然后给予它代表其他数字的概率更小的值。softmax 模型可以用来给不同的对象分配概率。

softmax 回归(softmax regression)分两步：

第一步：得到一张给定图片属于某个特定数字类的证据(evidence)

$$evidence_i = \sum_j W_{i,j} x_j + b_i$$

其中  $W_{i,j}$  代表权重,  $b_i$  代表数字  $i$  类的偏置量,  $j$  代表给定图片  $x$  的像素索引用于像素求和。

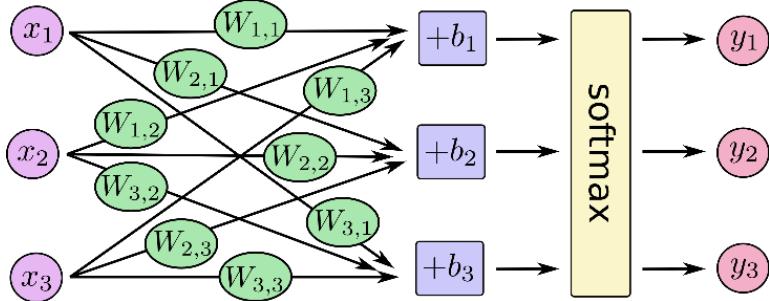


图 11.4: Softmax 回归

第二步: 用 softmax 函数可以把这些证据转换成概率  $y$

$$y = \text{softmax}(evidence)$$

其中:

$$\text{softmax}(y) = \text{normalize}(\exp(x)) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

### 11.2.2 实现回归模型

根据上述模型分析, 使用 NumPy 实现回归模型。

```

1 import tensorflow as tf
2 x = tf.placeholder("float", [None, 784])
3 W = tf.Variable(tf.zeros([784, 10]))
4 b = tf.Variable(tf.zeros([10]))
5 y = tf.nn.softmax(tf.matmul(x,W) + b)

```

### 11.2.3 训练回归模型

为了训练我们的模型, 首先需要定义一个指标来评估这个模型的好坏。在机器学习, 通常定义指标来表示一个模型是坏的, 这个指标称为成本(cost)或损失(loss), 然后尽量最小化这个指标。常用的成本函数是“交叉熵”(cross-entropy)。其定义如下:

$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

其中  $y$  是我们预测的概率分布,  $y'$  是实际的分布。

```

1 y_ = tf.placeholder("float", [None, 10])
2 cross_entropy = -tf.reduce_sum(y_*tf.log(y))

```

使用 TensorFlow 来训练建立的模型, TensorFlow 拥有一张描述你各个计算单元的图, 它可以自动地使用反向传播算法(Backpropagation algorithm)来有效地确定你的变量是如何影响你想要最小化的那个成本值的。然后, TensorFlow 会用你选择的优化算法来不断地修改变量以降低成本。

```
1 train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

TensorFlow 在这里实际上所做的是, 用梯度下降算法训练你的模型, 微调你的变量, 不断减少成本。

目前已经完成了模型的所有设置, 开始训练模型。该循环的每个步骤中, 我们都会随机抓取训练数据中的 100 个批处理数据点, 然后我们用这些数据点作为参数替换之前的占位符来运行 train\_step。

```
1     init = tf.initialize_all_variables()
2     sess = tf.Session()
3     sess.run(init)
4     for i in range(1000):
5         batch_xs, batch_ys = mnist.train.next_batch(100)
6         sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

## 11.3 动手实验:物体识别

### 11.3.1 实验目的

1. 了解 TensorFlow 的基本原理及用法;
2. 使用 TensorFlow 识别 MNIST 并识别自己手写数字;
3. 使用 TensorFlow 及摄像头对物体进行识别。

### 11.3.2 实验内容

1. 搭建 softmax 回归模型,对 MNIST 数据集进行训练,保存训练模型,并对自己的手写图片进行识别。
2. 使用 google 已建好的模型,打开树莓派摄像头,完成物体的识别。

# 第十二章 大语言模型简介

## 12.1 Transformer 模型

Google 在 2017 的论文: Attention is All You Need 中, 提出了一个新的神经网络模型: Transformer。这个模型完全采用注意力机制(Attention)处理序列数据, 在性能上相对之前的 RNN 模型有了很大的提高。目前几乎所有的自然语言模型都采用 Transformer 结构作为基本的组成单元, 在翻译、问答、知识处理等多个方面都取得了长足的进步。进而在 2013 年由 OpenAI 推出 ChatGPT 开始, 推动了一波大语言模型广泛应用的热潮。

### 12.1.1 注意力机制

注意力机制(Attention Mechanism)是一种用于深度学习模型中的重要技术, 旨在使模型能够在处理序列数据时更加关注输入序列中的不同部分, 从而提高模型的性能。注意力机制最初被广泛用于序列到序列的翻译任务, 但后来被扩展到各种其他任务中, 如语音识别、图像处理和自然语言处理等。

注意力机制的核心思想是在生成输出的过程中, 根据输入序列的不同部分来分配不同的权重。这使得模型可以根据输入序列的重要性动态地调整关注的位置, 从而更好地捕捉输入序列之间的关系。在序列到序列的任务中, 例如翻译, 模型需要根据输入句子的每个单词来生成输出句子的每个单词, 这就需要一种方法来确定输入句子中每个单词对于生成当前输出单词的重要性。

在注意力机制中, 通常会计算一个注意力权重向量, 该向量表示输入序列中每个位置的权重。这个权重向量可以基于不同的方法计算, 但最常见的方法是使用点积、加性(一层神经网络)或缩放点积等。注意力机制允许模型根据输入的不同部分分配不同的关注权重, 以便更好地处理序列数据。这对于捕捉长距离依赖关系、处理变长输入序列以及改善模型性能都非常有帮助。

### 12.1.2 Transformer 网络结构

传统的序列模型, 如循环神经网络(RNN)和长短时记忆网络(LSTM), 在处理长序列数据时存在一些问题, 例如难以捕捉长距离依赖关系和计算效率较低。Transformer 模型的设计就是为了解决这些问题。Transformer 模型的核心思想包括自注意力机制(Self-Attention)和多头注意力(Multi-Head Attention)。它将输入序列分别映射为查询(Q)、键(K)、值(V)向量, 然后通过计算注意力权重, 将值向量与权重相乘并求和, 从而获得上下文信息。通过多个注意力头的组合, 模型可以在不同的表示子空间中学习不同的关系。

Transformer 模型结构如图12.1 所示, 其主要的组成部分包括:

**编码器(Encoder):** 编码器由多个相同结构的层组成, 每一层都包含了多头自注意力机制和前馈神经网络。编码器将输入序列映射为一系列高维表示, 其中包含了输入序列的上下文信息。

**解码器(Decoder):** 解码器也由多个相同结构的层组成, 除了编码器的结构外, 解码器还包含一个多头注意力层, 用于对编码器输出进行自注意力计算, 以便更好地聚焦在输入序列的不同部分。

**自注意力机制(Self-Attention):** 这是 Transformer 模型的关键机制, 用于计算输入序列中每个位置的权重, 从而捕捉序列内不同位置的依赖关系。

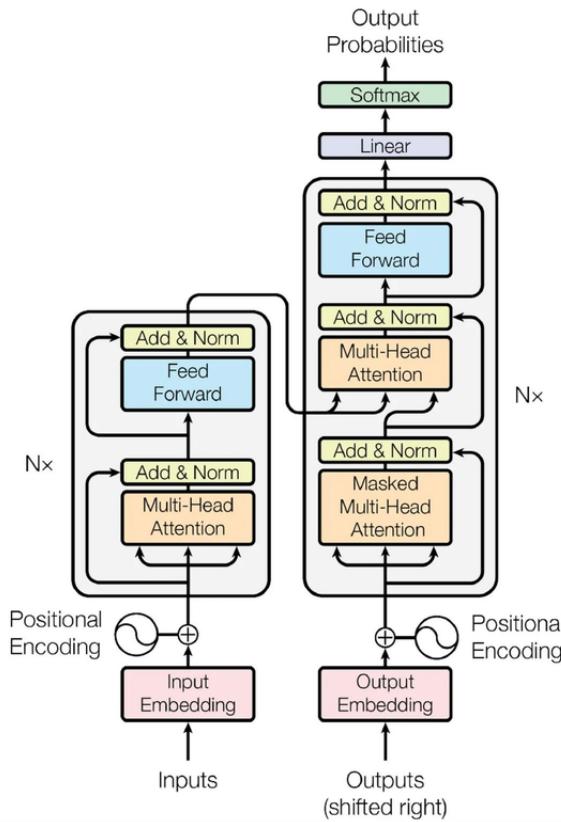


图 12.1: Transformer 网络结构

**多头注意力 (Multi-Head Attention):** 通过多个注意力头的并行计算, 模型可以学习不同类型的关系, 从而更好地表示输入序列。

**前馈神经网络 (Feedforward Neural Network):** 每个编码器和解码器层都包含一个前馈神经网络, 用于在注意力计算后对表示进行进一步的非线性变换。

**残差连接和层归一化:** 每个子层内都有残差连接和层归一化, 有助于防止训练过程中的梯度消失问题。

Transformer 模型的优势在于能够并行计算, 因此在训练过程中可以更高效地利用硬件资源。此外, Transformer 模型在预训练和微调的框架下, 在许多自然语言处理任务中取得了显著的性能提升。例如, BERT、GPT 系列和 T5 等模型都是基于 Transformer 架构的重要变体, 它们在各自的领域内都取得了领先的地位。

## 12.2 大语言模型

2023 年可以说是大模型爆发的一年, 许多科技公司和研究机构都推出了各自的大语言模型, 包括开源的 LLaMA 系列、OpenAI 的 GPT、清华大学的 GLM 等。

### 12.2.1 GPT 模型

GPT(Generative Pre-trained Transformer)是由 OpenAI 开发的一种极其强大的自然语言处理模型, 目前<sup>1</sup>已经推出了 4.0 版本, 其中提供给网页端免费使用的是 3.5 版本。GPT 模型在训练过程中使用了大量的文本数据进行预训练。模型在训练的过程中, 学会了语法结构、词义关系、常见知识等, 使其具备了一

<sup>1</sup>2023 年 9 月

定的常识和语言理解能力。GPT-3.5 是迄今为止最大的预训练语言模型之一，具有 1750 亿的参数。这意味着它有更大的容量来捕获复杂的语言模式和关系。

GPT 可以进行零样本学习 (zero-shot learning)。这意味着即使没有针对特定任务进行微调，它也能够在某种程度上执行各种任务，如回答问题、翻译文本等。只需通过提供简单的指示或示例，模型就可以进行推理和生成。GPT 可以用于各种自然语言处理任务，包括生成文章、写作、对话系统、问答系统、代码生成等。

### 12.2.2 ChatGPT

ChatGPT 是基于 GPT-3.5(或其他 GPT 变体)开发的一个特定应用，专注于进行自然语言对话和生成连续的对话文本。它被训练用于与用户进行交互，模拟人类对话的方式生成文本响应。ChatGPT 能够理解之前对话中的上下文，根据之前的对话内容进行响应。这使得它可以产生更准确和连贯的回复。类似于其他 GPT 模型，ChatGPT 的回复通常具有多样性，这意味着它可以以不同的方式回答相同的问题，从而增加了对话的趣味性和灵活性。

为了控制生成的输出，用户可以通过给出指导性提示或约束来引导 ChatGPT 的回复。例如，您可以明确指示模型以特定的角色回答、遵循某种语气或风格等。ChatGPT 可以在多种应用领域中使用，包括在线客服、虚拟助手、教育、娱乐、创意写作等。它可以与用户进行实时对话，为用户提供有关问题、信息、建议等方面的支持。

### 12.2.3 ChatGLM

GLM(Generative Language Model)是清华大学团队开发的大语言模型。GLM-130B 包含 1300 亿的参数，语言理解和 GPT-3 相当，在 MMLU 评测基准上，性能优于 GPT-3。ChatGLM 基于 GLM 模型进行微调，主要应用与对话场景，应用领域与 ChatGPT 基本相同。

ChatGLM 的 60 亿参数版本 (ChatGLM-6B) 已经开源，由于参数量少，可以在消费级的 GPU 上运行，方便本地使用和定制化。

ChatGLM 模型的接口可通过如下的 Python 代码进行调用：

```

1 response, history = model.chat(tokenizer,
2                               prompt,
3                               history=history,
4                               max_length=max_length if max_length else 2048,
5                               top_p=top_p if top_p else 0.7,
6                               temperature=temperature if temperature else 0.95)

```

其中 tokenizer 是大模型的分词器；prompt 是此次调用的提示词；history 是已经完成的对话历史；max\_length 表示回复的最大长度；top\_p 控制了生成文本时选择下一个词或标记的概率分布的截断点，较低的 top\_p 值会导致生成文本更加多样化，而较高的 top\_p 值则会导致生成文本更加可靠；temperature 这个参数影响了模型生成文本时的随机性，较高的温度值会增加生成文本的随机性，使其更具多样性。

## 12.3 python API 接口

虽然如同 ChatGLM-6B 这样的模型可以在本地运行，但对于大多数终端硬件来说，还是过于复杂了。使用这些复杂模型往往通过一定的编程接口远程进行调用。在与远程服务器进行交换数据的过程中，经常会使用一种被称为 JSON 的简单数据格式。

### 12.3.1 JSON 数据结构

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它使用人性化的文本来存储和表示数据对象。下面是一个简单的 JSON 数据包示例：

```
1 {
2     "name": "John",
3     "age": 30,
4     "interests": ["reading", "hiking", "coding"],
5     "address": {
6         "street": "123 Main St",
7         "city": "Anytown",
8     }
9 }
```

从上面的例子可以看出，JSON 格式比较易于理解。与 XML 相比，JSON 格式更小、更快，更易解析。JSON 使用 Javascript 语法来描述数据对象，但它仍可被其他编程语言读取和解析，几乎所有主流语言如 Python、Java、C# 等都有解析 JSON 的库。JSON 对象是由键值对组成的。键必须是字符串，值可以是数字、字符串、逻辑值、数组、对象等。JSON 是理想的网络数据交换格式，在异步浏览器和服务器通信中广泛使用。

### 12.3.2 FastAPI 简介

FastAPI 是一个现代化的 Python Web 框架，专门设计用于构建高性能、易于维护和快速开发的 Web 应用程序和 API。它在 Python 生态系统中备受欢迎，因为它具有以下特点和优势：

- 快速：FastAPI 非常快速，因为它借助 Python 3.6+ 中的异步特性，利用了异步编程的优势。这使得它能够处理高并发请求，并且具有出色的性能。
- 简单易用：FastAPI 的 API 设计非常直观，使用 Python 类型提示（Type Hints）来定义请求和响应模型，这样可以自动生成文档和验证请求数据。这使得开发人员能够更轻松地编写和维护代码。
- 自动文档生成：FastAPI 通过使用 Swagger UI 和 ReDoc 自动生成 API 文档，包括请求参数、响应模型、路由信息等。这样，开发人员和团队成员可以方便地查看和理解 API 的功能和用法。
- 强大的验证和序列化：FastAPI 支持请求参数验证、请求体验证、响应模型验证等，保证了数据的完整性和安全性。它还内置了 Pydantic 库，用于处理数据的验证和序列化。
- 内置支持 CORS 和安全性：FastAPI 提供了内置的 CORS（跨源资源共享）支持和安全性特性，可以帮助开发人员轻松地处理跨域请求和保障应用程序的安全性。
- 生态系统丰富：FastAPI 可以轻松与各种数据库、认证系统和第三方库集成，使开发更加灵活。它还支持异步请求处理、WebSockets 和后台任务等特性。
- 社区活跃：FastAPI 的社区非常活跃，有大量的文档、教程和第三方插件可供使用。这使得学习和使用 FastAPI 变得更加容易。

### 12.3.3 GET 接口

FastAPI 的一个典型示例如下：

```

1 from fastapi import FastAPI
2
3 app = FastAPI()
4
5 @app.get("/")
6 def read_root():
7     return {"message": "Hello, World"}
8
9 @app.get("/items/{item_id}")
10 def read_item(item_id: int, query_param: str = None):
11     return {"item_id": item_id, "query_param": query_param}

```

上述示例是一个简单的 FastAPI 应用，其中包含两个调用接口，分别用于处理根路径和带有参数的路径。FastAPI 会自动生成文档和进行请求参数验证。

客户端程序可以使用 requests 库来调用这个接口。以下是一个简单的示例，演示如何调用前面示例中的 / 和 /items/{item\_id} 接口：

```

1 import requests
2
3 # 定义 FastAPI 服务器的地址
4 server_url = "http://localhost:8000" # 将地址替换为你的 FastAPI 服务器地址
5
6 # 调用根路径
7 response = requests.get(f"{server_url}/")
8 if response.status_code == 200:
9     data = response.json()
10    print("根路径响应：", data)
11 else:
12    print("根路径请求失败")
13
14 # 调用带有参数的路径
15 item_id = 1 # 替换为你想要的 item_id
16 query_param = "example" # 替换为你想要的查询参数
17 response = requests.get(f"{server_url}/items/{item_id}?query_param={query_param}")
18 if response.status_code == 200:
19     data = response.json()
20     print("带有参数的路径响应：", data)
21 else:
22     print("带有参数的路径请求失败")

```

#### 12.3.4 POST 接口

在前面的示例中，接口调用的参数比较简单，可以直接编码到 URL 中进行调用，也就是所谓的 GET 方法。对于需要传递复杂参数的应用，就需要使用 POST 接口，此时往往会使用前面提到的 JSON 结构。

服务器端实现 POST 接口的代码示例如下：

```

1 # 创建接收 POST 请求的路由函数
2 @app.post("/items/")
3 async def create_item(item: Item):

```

```

4     # 在这里可以将接收到的 item 数据保存到数据库或进行其他操作
5     return {"item": item}

```

客户端代码的示例如下：

```

1 # 定义要发送的 JSON 数据
2 data = {
3     "name": "ExampleItem",
4     "description": "This is an example item."
5 }
6
7 # 发送 POST 请求
8 response = requests.post(f"{server_url}/items/", json=data)
9
10 if response.status_code == 200:
11     result = response.json()
12     print("POST 请求成功, 响应数据: ", result)
13 else:
14     print("POST 请求失败")

```

### 12.3.5 流式接口

大模型在生成文本数据的时候，并不是一次性把所有的文字内容都输出，而是随着模型的运算，不断的输出当前已经生成的文本。这也可以说成是大模型的生成文本速度比较慢，如果等到所有文字都准备好了再一次性输出，会让用户等待过多的时间，用户体验较差。毕竟用户的阅读速度有限，尽快的把生成的文字发送到用户端可以让用户尽早开始阅读。

为应对这一要求，很多大模型都提供流式的输出接口，保持服务器和客户端之间的链接，直到所有的数据发送完毕。FastAPI 可以使用下面的方式实现这个接口：

```

1 async def process(prompt, history):
2     try:
3         for response, history in model.stream_chat(tokenizer, prompt, history,
4                                                       max_length=max_length if max_length else 2048,
5                                                       top_p=top_p if top_p else 0.7,
6                                                       temperature=temperature if temperature else
7                                                       0.95):
8             await asyncio.sleep(0) # 这里必须加上这句才能捕获用户中断
9             yield "<>" + response
10    except asyncio.CancelledError:
11        # Force stop
12        pass
13
14 @app.post("/")
15 async def create_item(request: Request):
16     json_post_raw = await request.json() # 等待客户端数据
17     json_post = json.dumps(json_post_raw)
18     json_post_list = json.loads(json_post)
19     prompt = json_post_list.get('prompt')

```

```

20 max_length = json_post_list.get('max_length')
21 top_p = json_post_list.get('top_p')
22 temperature = json_post_list.get('temperature')
23 return StreamingResponse(process(prompt, history))

```

这里使用了 ChatGLM 的 `stream_chat` 接口，并在模型的返回数据上加入特殊字符串用于区分数据的边界<sup>2</sup>。对于客户端，可以利用下面的参考代码调用服务器的 API。

```

1 # 构造请求体并发送 POST 请求
2 response = requests.post(api_url, json=payload, headers=headers, stream=True)
3 # response.encoding = 'utf-8'
4 # 检查响应状态码
5 last_line = ''
6 try:
7     if response.status_code == 200:
8         # 逐行迭代接收数据
9         for line in response.iter_content(chunk_size=None):
10            if line:
11                # 找到回答的起始标记
12                idx = line.rfind(b'<^>')
13                if (idx < 0):
14                    continue
15                # 解码为 UTF-8 字符串
16                utf8_line = line[idx+3:].decode('utf-8')
17                # 在这里处理每行接收到的数据
18                last_line = utf8_line
19                os.system('clear')
20                print(last_line, flush=True)
21            else:
22                print("请求失败:", response.status_code)
23 except:
24     err = traceback.format_exc()
25     print(err)

```

## 12.4 动手实验：调用大语言模型

### 12.4.1 实验目的

1. 了解大语言模型的基本原理。
2. 利用大语言模型实现简单的文本分析。
3. 学习使用网络调用接口实现复杂机器学习任务。

### 12.4.2 实验内容

本次实验将利用大语言模型实现对在线考试中主观题的评分。具体题目和参考答案可以自主选择，但要求评分为 0~5 分。参考题目与答案如下：

题目：“第一次工业革命有哪些意义？” 答案应该包含如下几点，每回答对一点加 1 分，最多 5 分：

---

<sup>2</sup>这里为了简化采用这种方法，如果模型返回的数据包含这个字符串就会造成混乱

- 机器的大量使用, 提高了生产效率。
- 城市化进程加快, 人口大幅增加。
- 增加了商品的供应, 提高人民生活品质。
- 社会结构变化, 催生了社会改革运动。
- 工业废料的产生, 对环境造成不利影响。

#### 12.4.2.1 测试大语言模型

通过网页版接口与大语言模型进行对话, 了解大语言模型的基本能力和对话方式。通过多次尝试, 找到较好的提问方式可以解决本实验要完成的目标。

#### 12.4.2.2 利用服务器提供的 API 测试程序

编写程序从文本文件获取用户答案, 通过服务器 API 调用大模型进行交互, 输出用户的最终评分。代码内容参考前面介绍的流式接口的客户端程序。

# 第十三章 综合实验

## 13.1 实验目的

1. 使用树莓派完成一款实用产品的设计
2. 深入了解一种智能算法, 将其应用到系统中

## 13.2 实验内容

### 13.2.1 实验题目要求

综合实验要求每位同学利用实验平台的现有资源, 完成一款智能设备的原型设计。这款设备可以是任何应用场景的一个功能组件, 也可以是完整的解决方案。但必须充分考虑现有平台的计算能力, 既不能仅处理简单数据而浪费计算资源, 也不要因为项目过于复杂而无法胜任。

这里所谓的“智能”, 指的是完成的项目原型具有如下一种或者多种功能:

- 具有一定的“机器视觉”, 可以利用摄像头进行数据输入
- 具有一定的“机器听觉”, 能根据环境中的声音或者语音命令完成任务
- 具有一定的“机器智能”, 可以在尽可能少的人工干预情况下完成工作
- 具有一定的学习能力, 可以根据历史的经验改善处理问题的方式
- 具有一定的推理能力, 可以在复杂输入条件中做出最正确的判断

### 13.2.2 实验报告要求

实验报告要求完整、系统的对自己的作品进行介绍, 主要内容包括:

- 作品简介, 简要描述作品的功能和指标
- 系统方案, 介绍作品的基本设计思路及软硬件结构
- 系统实现, 包括完整的系统搭建流程、具体的实现细节, 以便读者可以重复同一工作
- 系统测试, 作品的展示内容和指标等数据
- 附录, 包含的程序及参考的开源项目或网站

### 13.2.3 参考实验题目

#### 13.2.3.1 对弈机器人

使用摄像头识别棋盘状态, 控制机械臂移动棋子, 完成对弈机器人的设计。对弈所使用的棋类可以是围棋、五子棋、象棋、国际象棋等。棋盘面积不大于 15cm × 15cm; 棋子采用塑料或者纸质, 可以通过吸盘吸取并移动。

### 13.2.3.2 智能移动机器人

采用树莓派控制移动底盘，控制方式可以是基于视觉的自主控制，也可以通过声音命令等给与辅助控制，还可以通过激光雷达进行环境及障碍物的识别。底盘可以使用实验室的 Turtlebot，也可以使用采用 Arduino 控制的智能小车。

Turtlebot 的串口通信协议可以参考官方的文档，Arduino 智能小车的通信协议如下：串口波特率 38400，控制命令如下：

- A 前进
- B 右转前进
- C 仅右转
- D 右转后退
- E 后退
- F 左转后退
- G 仅左转
- H 左转前进
- Z 停止
- I 关闭/打开电机

还支持发送命令更新部分内部执行参数，格式为{}中的命令：

- {P}，通过串口打印 PID 参数
- {0:xx}，更新基础运行速度
- {1:xx}，更新 PID 的 P 参数
- {2:xx}，更新 PID 的 I 参数

串口会即时反馈小车运行数据，格式为：{A:左轮数据:右轮数据:电池电压:舵机角度1} 和 {B:舵机角度2:电池电压:左轮数据:右轮数据}

### 13.2.3.3 猜拳机器人

通过摄像头识别手势，并控制机械手掌与用户玩猜拳游戏。机械手掌通过 5 个舵机进行控制，可以实现不同的手势动作。

### 13.2.3.4 绘画机器人

通过控制机械臂在纸上书写或绘画，实现创意应用产品。例如给用户画像，解答数学题等。

## 13.2.4 Atlas 200 DK 简介

Atlas 200 DK，是华为研发的一款高性能 AI 应用开发板，它集成了华为自研的 GPU 型 AI 芯片——昇腾 310 AI 处理器。昇腾(HUAWEI Ascend) 310 是一款高能效、灵活可编程的人工智能处理器，该处理器采用了华为的达芬奇架构，集成了丰富的计算单元，能够大大提高 AI 计算的效率。该芯片的适用性广泛，可以在 AI 全部的业务流程中起到提速的作用，不仅如此，该芯片还能够降低 AI 应用开发和部署的成本，提高应用效能。在综合实验阶段，同学们也可以选用此平台。

昇腾 AI 处理器的架构组成如图 13.1 所示，该芯片采用了经典的冯诺依曼式计算机架构，它的主要系统控制是 CPU 模块，但为了提供人工智能所需的计算能力，芯片集成了两个 AI 计算引擎，主要包含 AI Core 和 AI CPU。同时，为了提高 AI 模块的效率，芯片为其单独配置了特殊的专用模块任务调度器，该任务调度器 CPU 不处理其他事务，仅仅为 AI 模块服务调度。除此之外，芯片还加入了数字视觉预处理模块(DVPP)，该模块主要处理来自机器的图片和视频格式文件，对这些格式的文件进行前期预处理工作，例如

编码解码、裁剪等等。主存和 L2 缓冲区的设置也为 AI 计算提供了便利,使得算法能够高频、快速地访问数据,尤其是缓冲区,大大提高了计算所需的复用数据的使用效率。同时,芯片也提供了常见的各种硬件接口,如 USB、HBM、网卡、GPIO 等,为芯片的搭配和扩展使用提供了基础。根据芯片架构可以看出,昇腾 AI 处理器主要应用在需要大量计算的 AI 算法应用场景,尤其是语音、图像处理之类的人工智能领域。

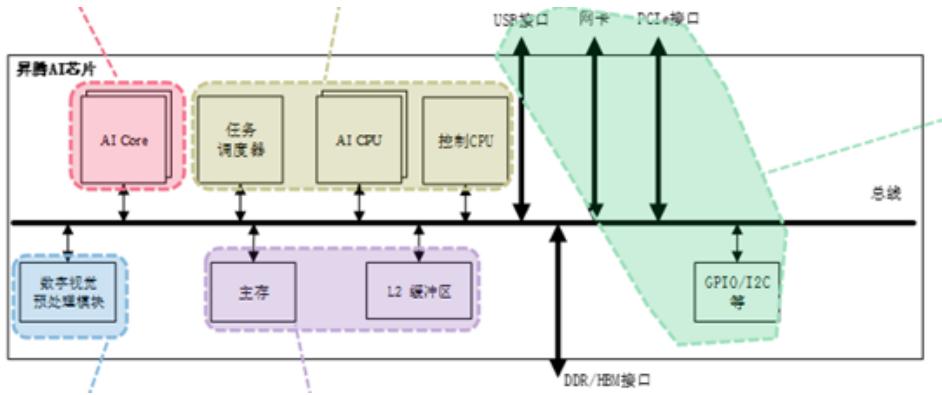


图 13.1: 昇腾 AI 处理器架构

作为搭载昇腾 310 AI 处理器的高性能开发板,Atlas 200 DK 可以实现对图像、视频等多种数据的分析和处理计算工作,与之相配套的开发者套件则必不可少。开发者套件是华为提供的一系列开发软件和硬件工具,通过安装开发者套件,用户可以使用 Atlas 200 DK 对外开放的 AI 加速模块接口,方便快捷地研发出适用于各种场景的产品,例如智能家电、机器人、无人机等。根据官网提供的数据,Atlas 200 DK 套件可提供 22TOPS (INT8) 的峰值计算能力,即在 8 位运算下,最高可提供每秒二十二万亿次的操作。不仅如此,开发板还支持两路摄像机输入和两路 ISP 图像处理,支持高动态范围 HDR10 技术标准。网络方面,开发板提供对外 1000M 高速网络连接,匹配强劲计算能力,预留有通用的 40-pin 扩展接口,极大地方便用户产品原型设计。图13.2 是 Atlas 200 DK 的外观。



图 13.2: Atlas 200 DK 开发板

CANN(Compute Architecture for Neural Networks)是华为公司针对人工智能应用场景推出的软件架构,为用户提供了多种编程接口。通过使用底层软件包,用户可便捷迅速地进行基于昇腾设备的产品业务开发。除此之外,用户还可以基于该软件包实现自定义算子、模型转换等功能。随着昇腾设备的更新迭代,CANN 软件包也在不断进行版本升级,但包内的主要工具作用大同小异。在实际使用时要根据开发板的型号选择合适的版本。

ACL(Ascend Computing Language)是指 CANN 中提供的一系列设备管理、内存管理、模型加载与执行、媒体数据处理等 C 语言的 API 库。pyACL(Python Ascend Computing Language)就是在 ACL 的

基础上使用 CPython 封装得到的 python API 库, 用户可以通过使用 python API 库对昇腾 AI 处理器进行资源和设备管理工作, 在这些封装好的接口库上开发深度神经网络应用, 用于实现目标识别、图像分类等功能。

# 参考文献

- [1] *OpenCV: Trackbar as the Color Palette*. URL: [https://docs.opencv.org/4.x/d9/dc8/tutorial\\_py\\_trackbar.html](https://docs.opencv.org/4.x/d9/dc8/tutorial_py_trackbar.html) (visited on 06/23/2022).
- [2] *OpenCV: Arithmetic Operations on Images*. URL: [https://docs.opencv.org/4.x/d0/d86/tutorial\\_py\\_image\\_arithmetics.html](https://docs.opencv.org/4.x/d0/d86/tutorial_py_image_arithmetics.html).
- [3] *Hough Transform*. In: *Wikipedia*. Feb. 20, 2022. URL: [https://en.wikipedia.org/w/index.php?title=Hough\\_transform&oldid=1073024927](https://en.wikipedia.org/w/index.php?title=Hough_transform&oldid=1073024927).
- [4] *OpenCV: Geometric Transformations of Images*. URL: [https://docs.opencv.org/4.x/da/d6e/tutorial\\_py\\_geometric\\_transformations.html](https://docs.opencv.org/4.x/da/d6e/tutorial_py_geometric_transformations.html).
- [5] *NumPy Quickstart — NumPy v1.24. Dev0 Manual*. URL: <https://numpy.org/devdocs/user/quickstart.html>.
- [6] *BCM2711 ARM Peripherals*. URL: <https://datasheets.raspberrypi.com/bcm2711/bcm2711-peripherals.pdf>.
- [7] M.J. Swain and D.H. Ballard. “Indexing via Color Histograms”. In: *[1990] Proceedings Third International Conference on Computer Vision*. [1990] Proceedings Third International Conference on Computer Vision. 1990, pp. 390–393. DOI: [10.1109/ICCV.1990.139558](https://doi.org/10.1109/ICCV.1990.139558).
- [8] *Face Detection: Facial Recognition and Finding Homepage*. Mar. 13, 2015. URL: <https://facedetection.com/> (visited on 07/27/2022).
- [9] P. Viola and M. Jones. “Rapid Object Detection Using a Boosted Cascade of Simple Features”. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001. Vol. 1. Kauai, HI, USA: IEEE Comput. Soc, 2001, pp. I-511-I-518. ISBN: 978-0-7695-1272-3. DOI: [10.1109/CVPR.2001.990517](https://doi.org/10.1109/CVPR.2001.990517).