智能硬件应用实验

北京大学信息科学技术学院

2020 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。

与理论课程不同，本讲义并没有完整详细的对三个主线的内容进行介绍，而是通过八个单元实验把相关的知识串联起来，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其它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

实验一　Raspberry Pi 与 Python 编程语言

1.1　实验目的

1. 熟悉树莓派的连接访问。
2. 熟悉 Linux 常用命令。
3. 初步了解 Python 开发环境与编程语言。

1.2　实验原理

1.2.1　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，2016 年推出了树莓派 3。实验课程选取的是树莓派第 3 代 B 型。相对于之前的版本树莓派 3B 型搭载 1.2GHz 的 64 位四核处理器 (ARM Cortex-A53 1.2GHz 64-bit quad-core ARMv8 CPU)，增加了 802.11 b/g/n 无线网卡，增加了低功耗蓝牙 4.1 适配器，最大驱动电流增加至 2.5A。树莓派 3B 型外观如图 1.3所示。它通过 Micro USB 接口供电，至少可以提供 2A 的电流才可以保证正常的工作。

树莓派 3 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 3 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。

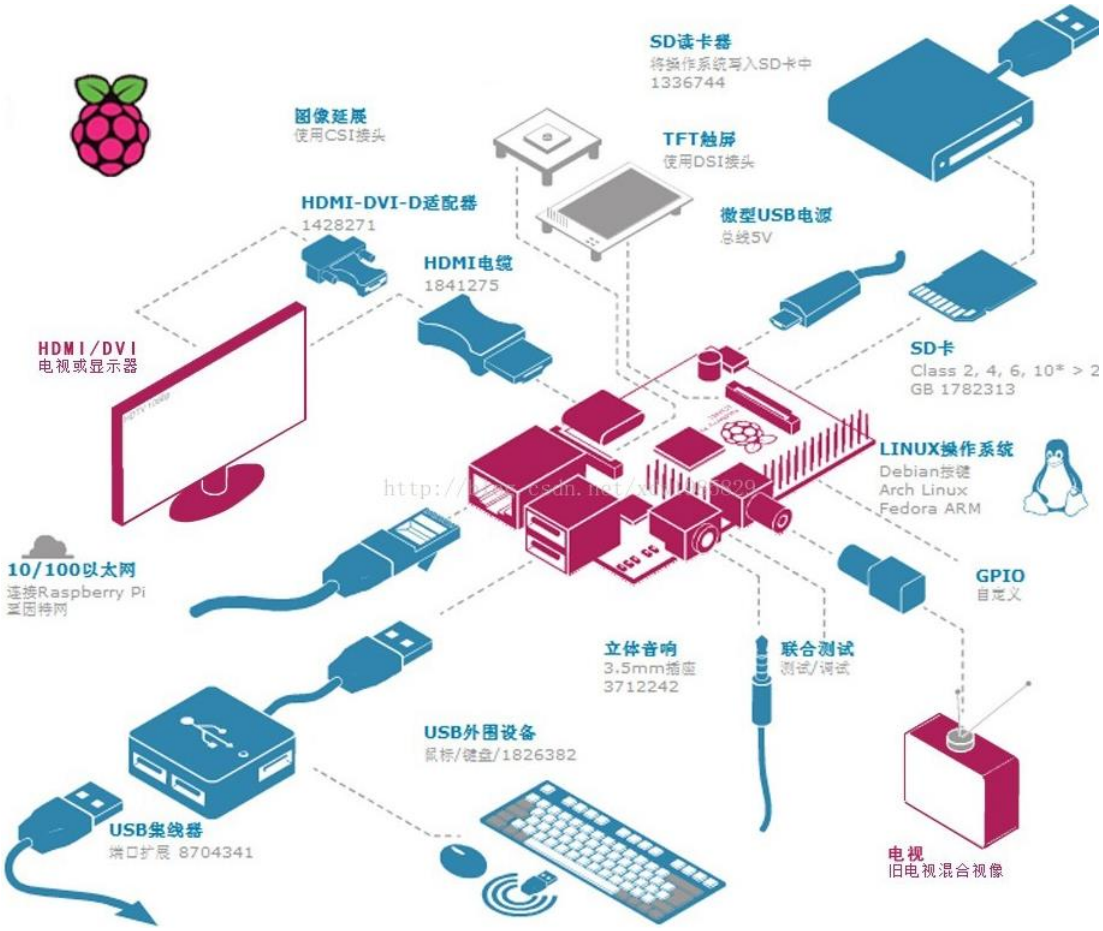


图 1.2: 树莓派的连接与扩展方式

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。

1.2.2 Linux 常用命令

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。

登录到树莓派操作系统中之后需要掌握基本的 Linux 操作，下面列举了常用的一些命令，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

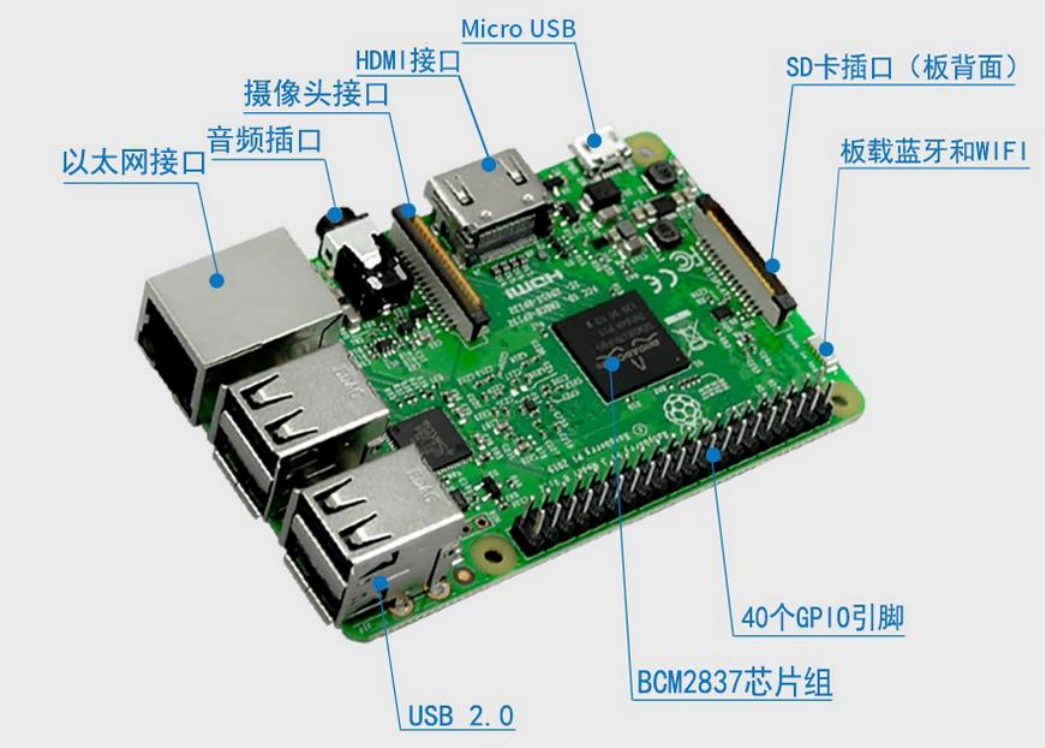


图 1.3: 树莓派 3 代 B 型外观及接口配置

- 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 或 less file 或 cat 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：apt-get install soft
- 退出当前会话：exit

1.2.3 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。

1.2.3.1 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。

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。

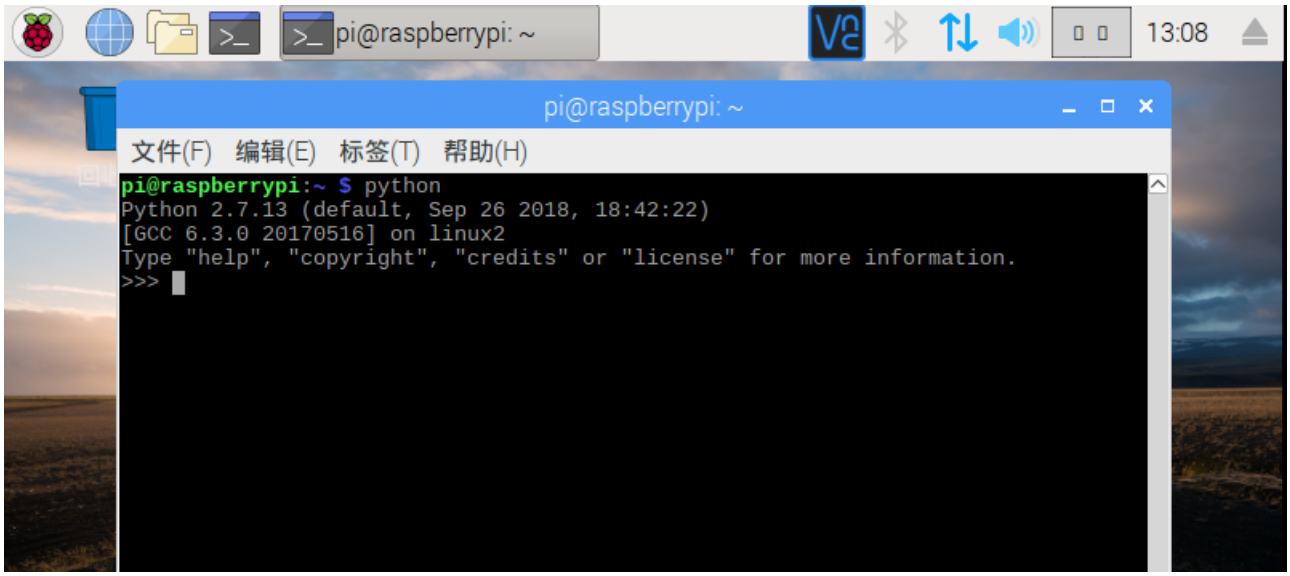


图 1.4: Python 运行环境

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

```
>>> 100+200
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。

```
>>> print("hello, world!")
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

```
$ python3 hello.py
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。

1.2.3.2 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，1.23×10^9 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。

列表（list）是 Python 内置的一种数据类型，是一种有序的集合，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 ['Michael', 'Sarah', 'Tracy']，['Apple' ，123 ，True] 等。

1.2.3.3 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：

```
>>>a = 'name'
type(a)
<class 'str'>
>>>classmates = ['Michael', 'Bob', 'Tracy']
>>>type(classmates)
<class 'list'>
```

1.2.3.4 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：

```
>>>print('hello, world')
hello, world
>>>name = 'Tom'
>>>print(name)
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，

```
>>>print('The brown fox', 'jumps over', 'the lazy dog.')
The brown fox jumps over the lazy dog.
>>>print('100 + 200 =', 100 + 200)
```

```
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，

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

当用户输入 name = input() 并按下回车后，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，

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

1.2.3.5 条件判断

在 Python 程序中，用 if 语句实现条件判断。例如，

```
age = 20
if age >= 18:
    print('your age is', age)
    print('adult')
```

根据 Python 的缩进规则，如果 if 语句判断是 True，就把缩进的两行 print 语句执行，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：

```
age = 3
if age >= 18:
    print('your age is', age)
    print('adult')
else:
    print('your age is', age)
    print('teenager')
```

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：

```
age = 3
if age >= 18:
    print( 'adult' )
elif age >= 6 :
    print( 'teenager' )
else :
    print( 'kid' )
```

1.2.3.6 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 lis 中的每个元素迭代出来。例如：

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

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

另外，range() 函数产生的整数序列可以指定步进值，例如：

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

```
n = 1
while n <= 100:
    print(n)
    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：

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

1.3 实验内容

1.3.1 示例代码验证

在树莓派上编辑运行示例程序，实验步骤：

1. 接通树莓派的电源，启动完全后打开一个终端。
2. 进入目录\home\pi，建立自己的工作目录，把实验中编写的程序都放在个人目录下，避免和其他同学程序混淆。
3. 编辑和运行下面的示例程序。

编辑程序可以使用 vim 编辑器，运行程序的方式是在命令行下输入：python3 example.py

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

注:int() 函数可以将字符串转换为整形数,float() 函数可以将字符串转换为浮点数。

1.3.2 实现冒泡排序

将列表中的数据利用冒泡排序算法实现从大到小的排列。下面代码用于产生待排序的列表数据

```
import random # 导入 random 模块
data = [] # 定义一个空的列表 data
for i in range(0,10): # 在列表 data 插入 10 个 0~99 之间的随机数
    data.append(random.randint(0,99))
print(data) # 打印列表

count = len(data) # 获取列表的长度
for i in range(0,count): # 遍历列表中的元素
    print(data[i], end=' ')
```

上面的代码中值得注意的部分有:

1. 在 Python 中,# 号用于定义单行注释语句。
2. 在 Python 中的 import 语句是用于导入模块。
3. 模块 random 为 Python 内建模块,用于产生随机数,randint 函数产生随机整数。
4. 可以利用索引访问列表中的元素,索引是从 0 开始的,例如 data[2] 可以访问列表 data 的第 3 个元素

1.3.3 实现猜拳游戏

猜拳是一种简单的游戏,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。

下面的示例代码采用键盘输入作为用户输入实现基于随机数的猜拳游戏。

```
import random
user = 0
while True:
    user_raw = input("请输入:剪刀(0) 石头(1) 布(2)退出(3):")
    user = int(user_raw)
    if user == 3:
        break
    computer = random.randint(0,2)
```

```
if (user == 0 and computer == 2) or (user == 1 and computer == 0)\
        or (user == 2 and computer == 1):
    print("你赢了")
elif  computer == user:
    print("平局")
else:
    print("你输了")
```

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。

1.4 思考题

1. 在猜拳游戏中，用户如果输入了非数字按键，程序就会报错退出，如何避免？
2. 如果猜拳游戏的对手是电脑，什么样的策略更好？

实验二　GPIO 的使用与线性回归模型

2.1　实验目的

1. 了解树莓派的 GPIO 接口。
2. 了解树莓派 RPi.GPIO 模块的基本使用。
3. 掌握树莓派 GPIO 接口的编程方式和使用方法。
4. 了解线性回归和机器学习的基本概念。

2.2　实验原理

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 3 代 B 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 A+ / B+ / 2 / 3 代 B 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。

2.2.3　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：

```
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。

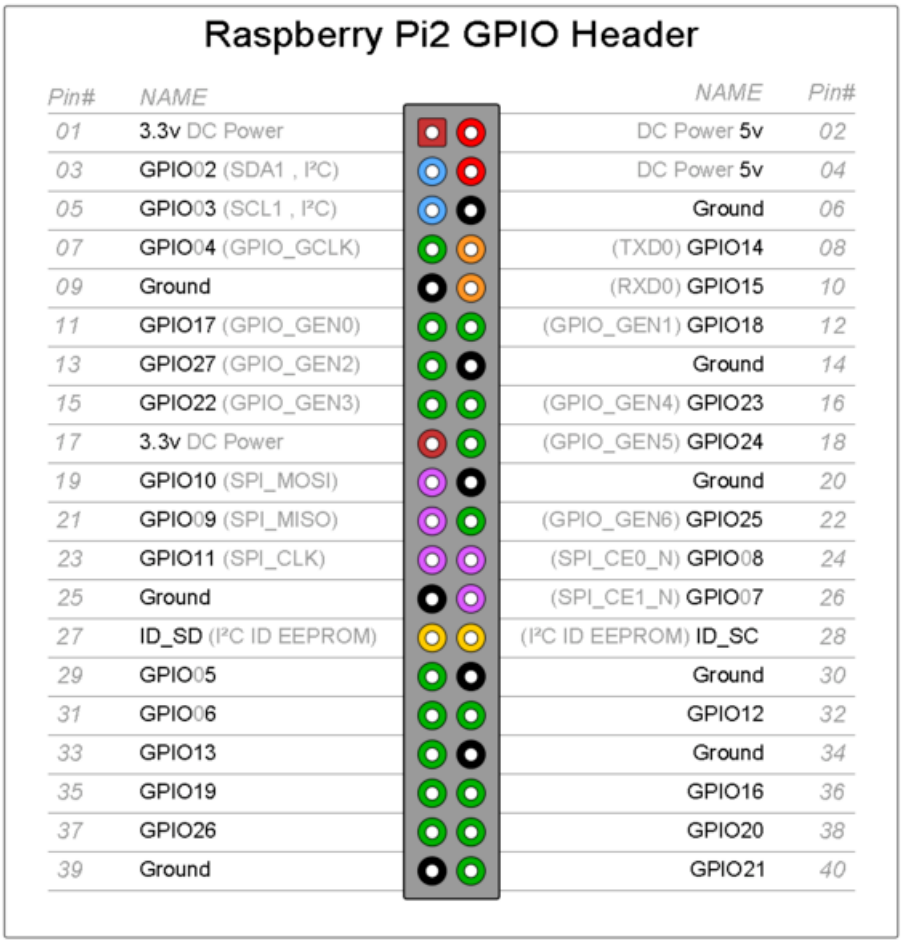


图 2.1: 树莓派 GPIO 引脚定义

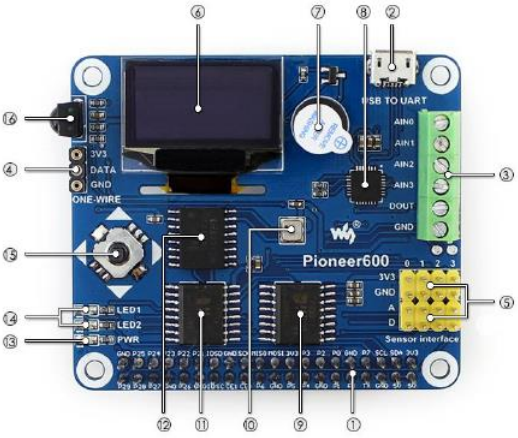
目前有两种方式可以通过 RPi.GPIO 对树莓派上的 IO 引脚进行编号。第一种方式是使用 BOARD 编号系统。该方式参考树莓派主板上 P1 接线柱的引脚编号。使用该方式的优点是无需考虑主板的修订版本，硬件始终都是可用的状态，无需从新连接线路和更改您的代码。

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。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：

```
GPIO.setmode(GPIO.BOARD) # 采用 BOARD 编号
# 或者
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：

```
# 配置 channel 指定的通道为输入通道
#channel 与使用的编号方式对应
GPIO.setup(channel, GPIO.IN, GPIO.PUD_UP)
# 配置 channel 指定的通道为输出通道
GPIO.setup(channel, GPIO.OUT)
# 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
GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)
```



[核心接口简介]

1. **Raspberry Pi GPIO 接口**
   方便接入 RPi
2. **USB TO UART 接口**
   方便通过串口终端控制 Raspberry Pi
3. **AD/DA 输入输出接口（接线端子）**
   方便在各种场合使用
4. **ONE-WIRE 接口**
   可接入 DS18B20 等单总线设备
5. **传感器接口**
   方便接入各类传感器

[器件简介]

6. **0.96 寸 OLED**
   驱动芯片为 SSD1306，128×64 分辨率，SPI 接口
7. **蜂鸣器**
8. **CP2102**
   USB 转串口芯片
9. **PCF8591**
   8 位精度 AD/DA 芯片，I2C 接口
10. **BMP180**
    压力传感器，I2C 接口
11. **PCF8574**
    I/O 扩展芯片，I2C 接口
12. **DS3231**
    高精度 RTC 芯片，I2C 接口
13. **电源 LED**
14. **用户 LED**
15. **五向摇杆**
16. **LFN0038K 红外接收头**

图 2.2: PIONEER600 树莓派扩展板

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 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。

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

```
# 创建一个 通道 channel 的 PWM 实例
pwm = GPIO.PWM(channel, frequency )
# 启动 PWM ，并指定占空比 dc ， dc 的范围 0.0~100.0
pwm.start(dc)
# 更改 PWM 脉冲重复的频率为 frequency
pwm.ChangeFrequency(freq)
# 更改 PWM 的占空比为 dc
```

```
    pwm.ChangeDutyCycle(dc)
    # 停止 PWM
    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：

```
    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。

2.2.4 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：

```
    try:
        # 业务实现代码
        ...
    except Error1:
        # 错误处理代码
        ...
    finally:
        # 不管是否发生异常，一定会执行的代码
        ...
```

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。

```
try:
    while True:   # 死循环
        time.sleep(1) # 休眠 1 秒钟
except KeyboardInterrupt:  # 键盘中断事件
    GPIO.cleanup()
```

2.2.5 感知器与线性回归模型

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 2.3 所示。

图 2.3 中各个参数的关系如下：

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

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。

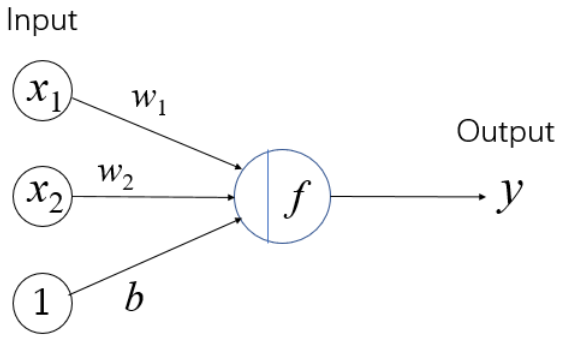


图 2.3: 神经元感知器模型

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

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

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 η，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。

2.3 实验内容

2.3.1 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 37 号引脚，BCM 编号为 20。试利用树莓派 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。

```
import RPi.GPIO as GPIO

KEY = 20
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(KEY, GPIO.IN,GPIO.PUD_UP)  # 上拉电阻
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：

```
def my_callback(ch):
    print("KEY PRESS")
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，就会调用指定的回调函数 my_callback（函数的参数是按键的编号），其中最后一个参数是软件消抖的延时毫秒数。请用这个函数实现本小节的按键检测程序。

2.3.2 LED 指示灯

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 LED 指示灯的闪烁，点亮 0.2 秒，熄灭 0.2 秒。
3. 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。

```
import RPi.GPIO as GPIO
import time
LED = 26
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED,GPIO.OUT)

p = GPIO.PWM(LED,50)
p.start(0)
try:
    while True:
        for dc in range(0,101,5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.05)
        for dc in range(100,-1,-5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.05)
except KeyboardInterrupt:
    pass # 空语句,不执行任何操作,只起到占位作用
    p.stop()
    GPIO.cleanup()
```

2.3.3 按键 LED 联调

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，每按下一次按键就切换 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()）。

2.3.4 线性回归模型的实现

假定线性回归模型参数个数为 2，即 $f(\mathbf{x}) = w_1x_1 + w_2x_2 + b$。$w_1, w_2, b$。训练数据通过随机的方式产生。

```
len = 2000
x1 = np.random.rand(len) * 10   # 产生均匀分布的 x1
x2 = np.random.rand(len) * 10
y = x1 * m1 + x2 * m2 + m3 + np.random.randn(len) # y 要加上随机噪声
```

使用梯度下降法估算 m_i 的值，并和预设值进行比较，在算法中选取合适的学习率。上面的算法相当于提供了 2000 组训练数据，每一组数据可以通过算法对估算的参数进行微调，当全部数据都使用过之后，称为进行了一个 epoch，往往需要进行多个 epoch 才可以获得满意的估算结果。

请用按键控制算法的进度，每按一次按键进行一个 epoch，并打印当前的梯度下降量，设定合适的算法结束条件，当条件满足时，点亮板上的 LED 灯。

2.4 思考题

1. 梯度下降法的学习率不宜设置过大，请分析如何在算法中检测到学习率过大的情况。

实验三　1-Wire 总线与聚类算法

3.1　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. 学习 K 均值聚类算法

3.2　实验原理

3.2.1　1-Wire 总线

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。

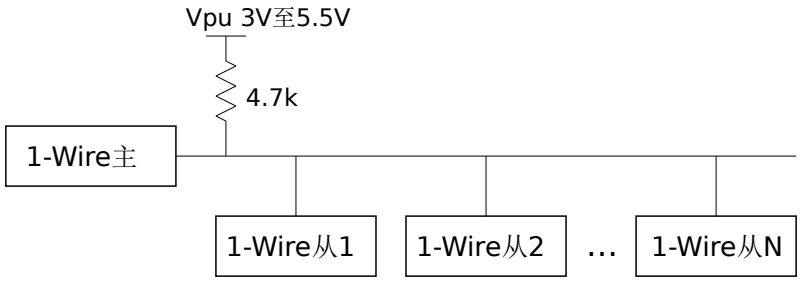


图 3.1: 1-Wire 总线系统组成

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。

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的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。

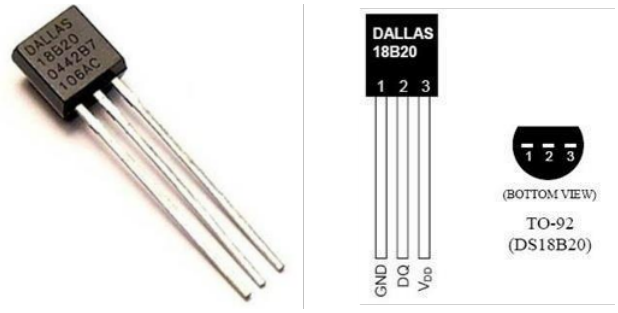


图 3.2: DS18B20 外观与引脚定义

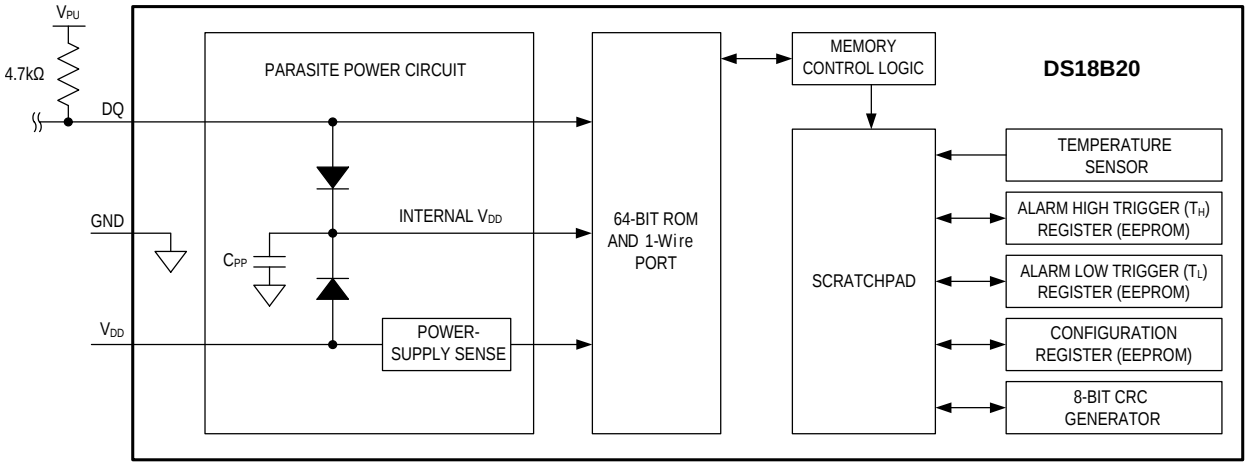


图 3.3: DS18B20 内部结构

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。

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：

```
pi@raspberrypi:~# sudo modprobe w1-gpio
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:

```
pi@raspberrypi:~# cd /sys/bus/w1/devices
pi@raspberrypi:~# ls
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:

```
pi@raspberrypi:~# cd 28-00000674869d
pi@raspberrypi:~# cat w1_slave
70 01 4b 46 7f ff 10 10 e1 : crc=e1 YES
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。

3.2.4 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：

```
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。

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

写入文件的简单示例如下所示：

```
f = open(filename, 'a')
f.write('Last line')
f.close()
```

如果要查看目录中的文件，可以使用 python 的标准库 glob，如下面的示例打印出 dir 目录下的全部文件：

```
import glob
for name in glob.glob('dir/*'):
    print(name)
```

在 glob 的参数中，可以使用通配符等正则表达式来匹配文件名称。

3.2.5 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。

3.2.6 部分 python 库的使用

NumPy 是 Python 中科学计算的基础软件包。它是一个提供多了维数组对象，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。

```
import numpy as np
mean = 5
sigma = 1.3
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。

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。

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

3.3 实验内容

3.3.1 读取温度传感器的值

1. 按照实验原理，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。
3. 利用 LED 指示灯实现温度告警的功能。

基于实验内容 2 实现温度告警功能: 检测当前环境温度值，当环境温度超过 30 摄氏度时，通过 Pioneer 600 扩展板上的 LED 指示灯闪烁进行告警；当环境温度低于 30 摄氏度时，熄灭 LED 指示灯，停止告警。

3.3.2 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 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。

实验四　SPI 总线与支持向量机

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. 编写 Python 程序，访问 SPI 连接的 OLED 设备。
4. 掌握支持向量机的基本原理

4.2　实验原理

4.2.1　SPI 总线传输原理

SPI（Serial Peripheral Interface）接口标准广泛用于 MCU 与外部设备直接的数据交互，如一些传感器芯片、控制芯片、LCD 等。最早由 Motorola 公司提出，目前由 Freescale 公司进行标准的维护。

4.2.1.1　SPI 功能框图

SPI 功能模块框图如图 4.1所示：

SPI 接口引脚定义如表 4.1所示。

表 4.1: SPI 引脚定义

名称	位宽	功能描述	备注
SCLK	1	串口时钟	由 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	

SPI 地址映射如表 4.2 所示。

4.2.1.2　SPI 传输方式配置

如图 4.2 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 4.3和 4.4 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：

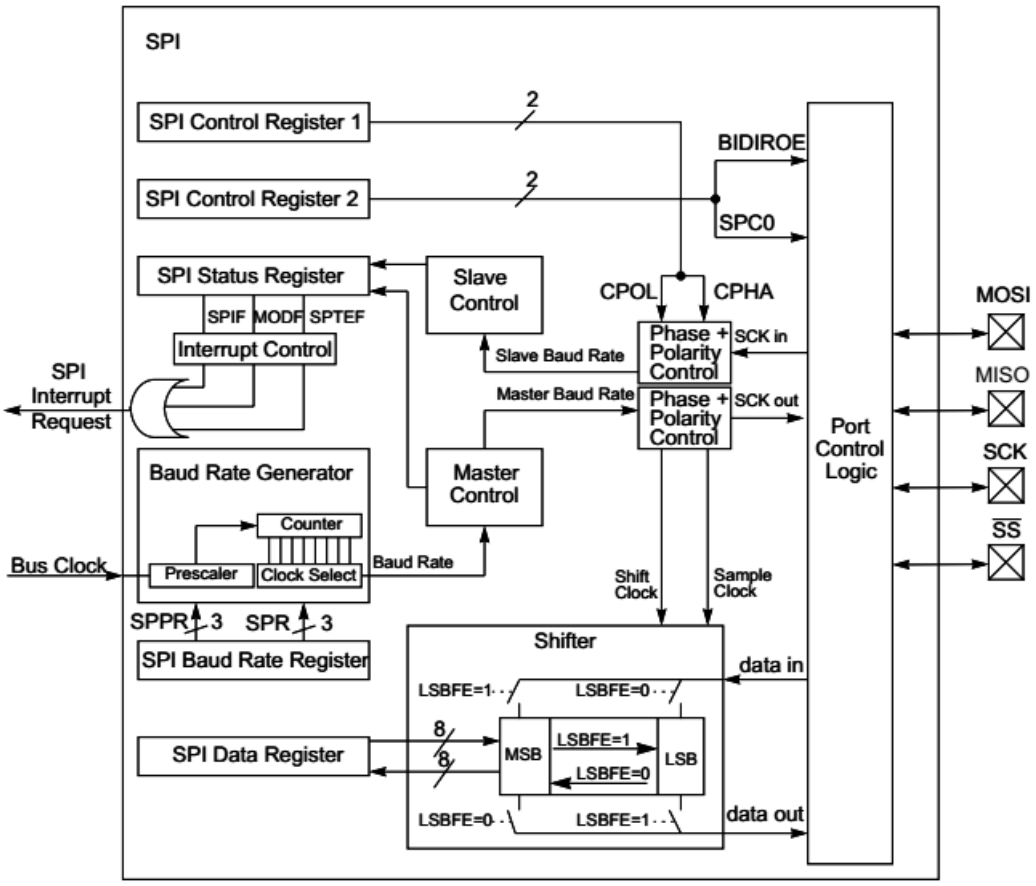


图 4.1: SPI 功能框图

表 4.2: SPI 寄存器定义

Address	功能	读写
0	SPI Control Register1 (SPICR1)	Read/Write
1	SPI Control Register2 (SPICR2)	Read/Write
2	SPI Baud Rate Register (SPIBR)	Read/Write
3	SPI Statue Register (SPISR)	Read
4	Reserved	—
5	SPI Data Register (SPIDR)	Read/Write
6	Reserved	—
7	Reserved	—

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

其中 BaudRateDivisor 由波特率控制寄存器中的 6 个 bit 决定：

$$BaudRateDivisor = (SPPR + 1) \times 2^{SPR+1}$$

在 BusClock=25MHz，波特速率从 12.21kbps～ 12.5Mbps，如表 4.3所示。

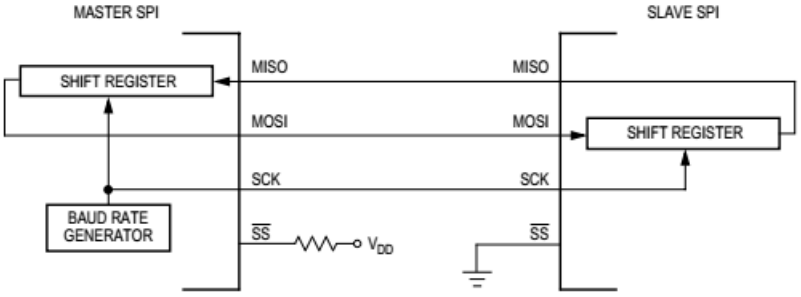


图 4.2: SPI 传输

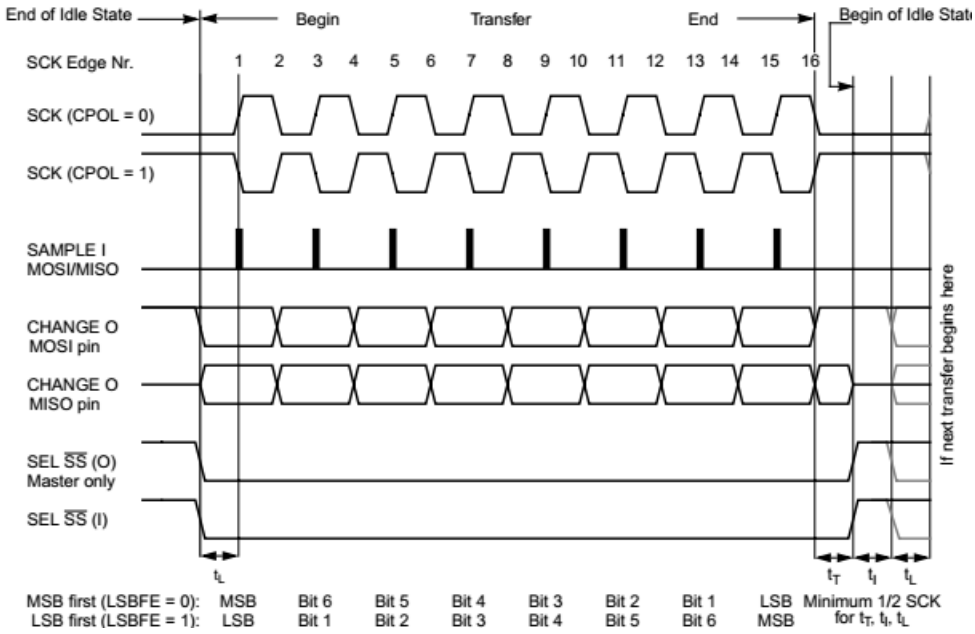


图 4.3: CPHA=0 SPI 传输时序图

表 4.3: SPI 波特率设置

SPPR2	SPPR1	SPPR0	SPR2	SPR1	SPR0	BaudRateDivisor	BuadRate
0	0	0	0	0	0	2	12.5MHz

4.2.1.3 SPI 连接模式

典型主从设备连接如图 4.5 所示。

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。

4.2.2 OLED 设备及访问

实验中采用 128*64 点阵的 OLED 显示屏，使用 SSD1306 驱动芯片驱动 OLED 屏幕。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、**I²C** 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。

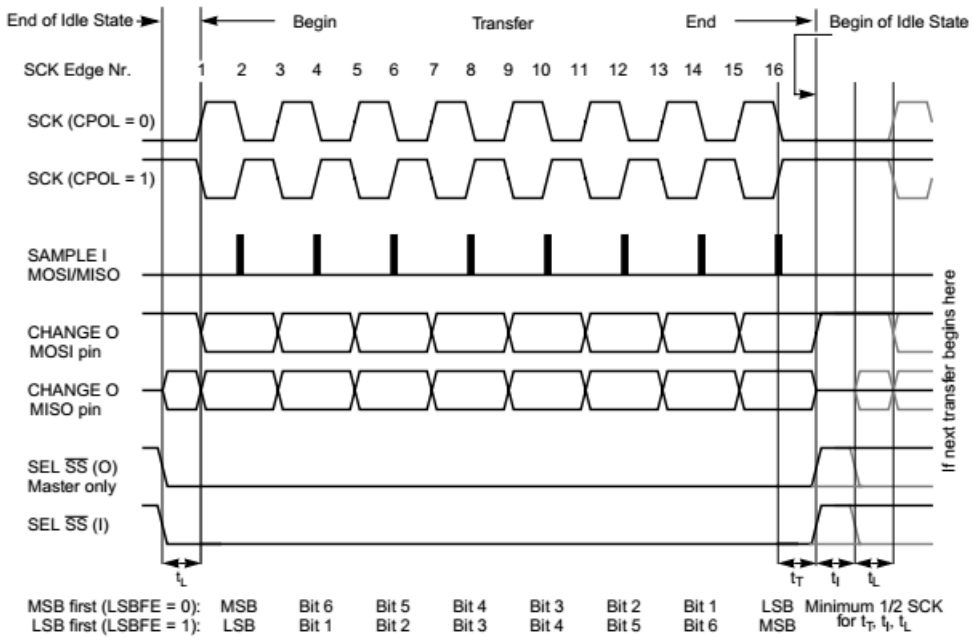


图 4.4: CPHA=1 SPI 传输时序图

图 4.5: SPI 典型主从设备连接方式

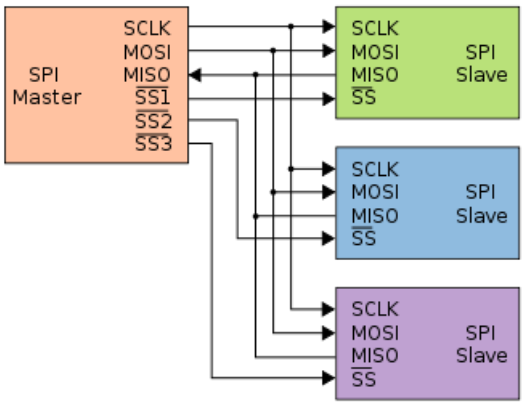


图 4.6: SPI 典型主从设备连接方式

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 4.8所示，4 线接口的时序如图 4.9所示。

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。在之前的环境配置试验中已经完成了 SPI-dev 的安装。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。

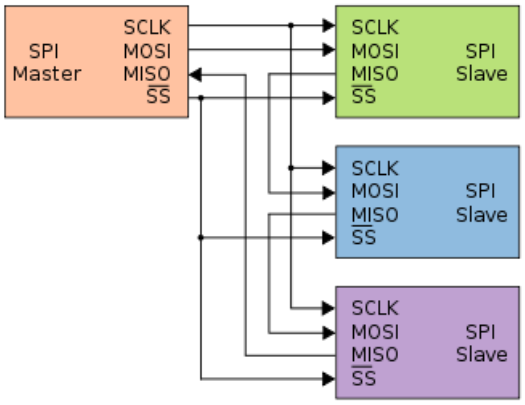


图 4.7: SPI 典型主从设备连接方式

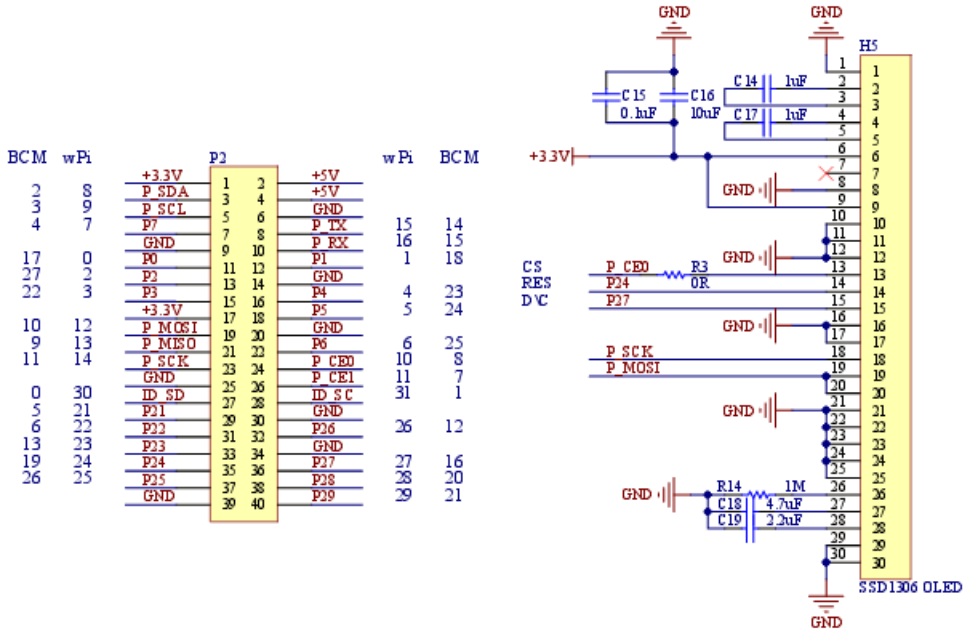


图 4.8: 树莓派与驱动芯片的连接

图像处理代码使用了 python 的 Pillow 库，如果系统中缺少相应的库文件可以采用下面的命令安装。

```
$ pip3 install Pillow
```

```
import time
import spidev as SPI
import SSD1306
from PIL import Image  # 调用相关库文件

RST = 19
DC = 16
bus = 0
device = 0  # 树莓派管脚配置
disp = SSD1306.SSD1306(rst=RST,dc=DC,spi=SPI.SpiDev(bus,device))
```

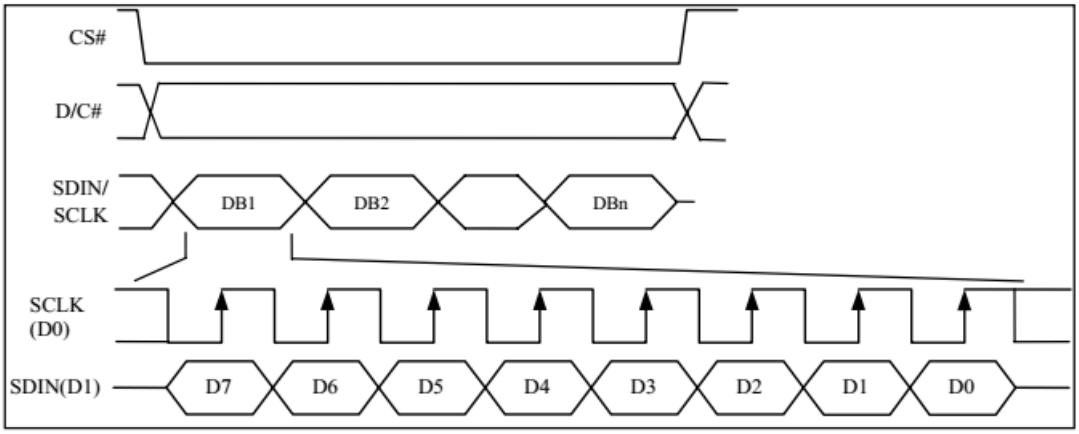


图 4.9: 4 线 SPI 传输时序

```
disp.begin()
disp.clear()
disp.display()  # 初始化屏幕相关参数及清屏

image = Image.open('happycat.png').resize((disp.width, disp.height),
    Image.ANTIALIAS).convert('1')
disp.image(image)
disp.display()  # 显示图片
```

在显示字符的时候，可以调用 ImageDraw 及 ImageFont，如下代码所示：

```
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont

# 初始化代码省略
font = ImageFont.load_default()
image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
draw = ImageDraw.Draw(image)
draw.bitmap((0,0), logo, fill=1)
draw.text((x,top), 'This is first line', font=font, fill=255)
disp.image(image)
disp.display()  # 显示图片
```

4.2.3 Python 的对象封装

Python 语言是一种面向对象的编程语言，上面代码中所使用的 SSD1306 就通过“类”的方式对相关的程序模块进行了封装。在 Python 中定义“类”的关键字是 `class`，其构造函数用 `__init__` 来定义。例如 SSD1306 中的代码：

```
class SSD1306(object):
    """class for SSD1306  128*64 0.96inch OLED displays."""
```

```
    def __init__(self,rst,dc,spi):
        self.width = 128
        self.height = 64
        ....
```

阅读 SSD1306.py 程序，了解对象封装的基本方法，在后续的编程过程中，可以尝试把各个模块封住起来，便于在其它的程序中调用。

4.2.4 支持向量机简介

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？支持向量机（Support Vector Machine）就是这样一种寻找最优超平面的算法，它可以保证距离超平面最近的几个训练样本（所谓支持向量）到这个超平面的距离之和最大，也就是获得最鲁棒的解。具体的数学原理这里不做讨论，感兴趣的同学可以阅读相关的书籍[1]。

对于有些样本集，并没有一个超平面可以完成类别的划分，这时候可以通过所谓核函数（Kernel）将数据集变换的高维空间，再来寻找合适的划分平面。常用的核函数有 linear、poly、rbf、sigmoid 等。

4.2.5 scikit-learn

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。

```
import matplotlib.pyplot as plt

# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, metrics

# The digits dataset
digits = datasets.load_digits()

# The data that we are interested in is made of 8x8 images of digits, let's
# have a look at the first 4 images, stored in the `images` attribute of the
# dataset.  If we were working from image files, we could load them using
# matplotlib.pyplot.imread.  Note that each image must have the same size. For these
# images, we know which digit they represent: it is given in the 'target' of
# the dataset.
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)
```

[1] 例如周志华的《机器学习》第六章

```
# To apply a classifier on this data, we need to flatten the image, to
# turn the data in a (samples, feature) matrix:
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a classifier: a support vector classifier
classifier = svm.SVC(gamma=0.001)

# We learn the digits on the first half of the digits
classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])

# Now predict the value of the digit on the second half:
expected = digits.target[n_samples // 2:]
predicted = classifier.predict(data[n_samples // 2:])

print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))

images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
for index, (image, prediction) in enumerate(images_and_predictions[:4]):
    plt.subplot(2, 4, index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)

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。

4.3 实验内容

4.3.1 OLED 设备测试

1. 通过 SPI 设备访问 OLED，在显示屏上显示“hello world!”。
2. 设计并显示一个 OLED 时钟，含有北大的 logo，时钟能显示当前时间，并显示某个纪念日的倒计时（如北大校庆日或期末考试）。

4.3.2 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。

4.3.3 使用 OLED 显示结果

将预测的图像显示在 OLED 上，并将预测的结果用 print 函数显示在屏幕上。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，因此需要利用 Image 库中的 resize 函数转换一下，示例如下：

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

4.4 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？

实验五　I²C 总线与专家系统简介

5.1　实验目的

1. 了解 **I²C** 总线传输的原理。
2. 编写 Python 程序，访问扩展板上的 **I²C** 设备。
3. 熟悉并使用 **I²C** 调试工具。
4. 了解专家系统的基本概念和设计方法

5.2　实验原理

5.2.1　I²C 总线传输原理

I²C 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 **I²C** 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 **I²C** 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 **I²C** 应用如图 5.1所示。

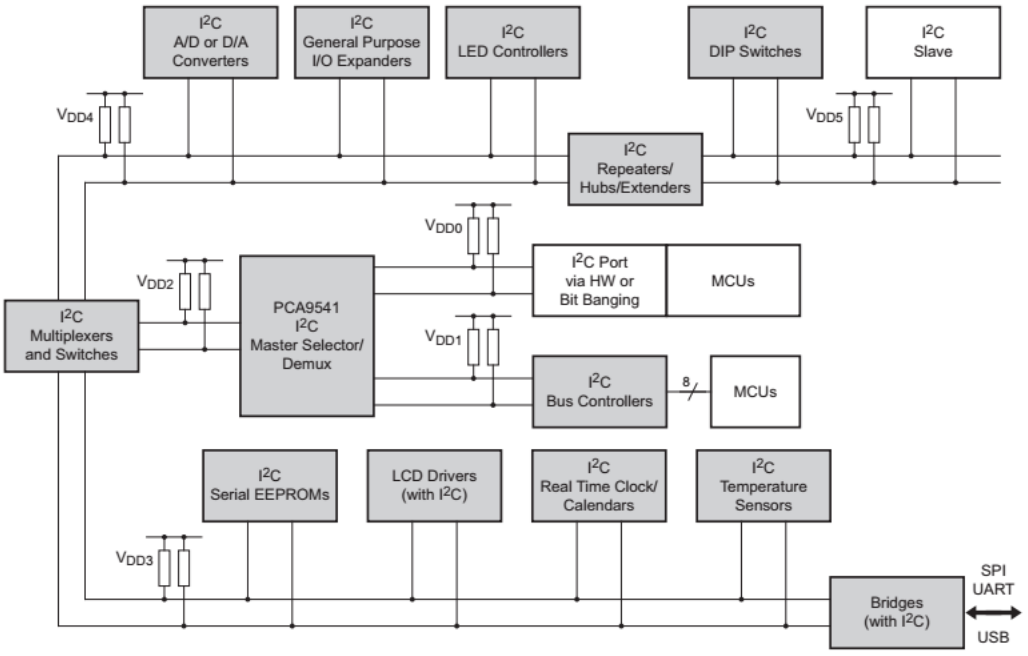


图 5.1: 典型 **I²C** 应用

一般的，使用 **I²C** 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。**I²C** 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。

I²C 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 5.2所示。

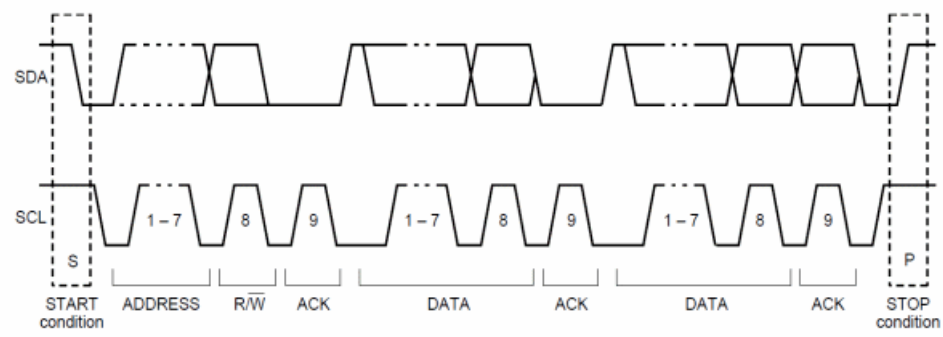


图 5.2: **I²C** 数据传输过程

I²C 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 **I²C** 数据与地址，都是先传递最高位，最后传递最低位。

5.2.2 扩展板上的 I²C 设备

在扩展板上连接了多个 **I²C** 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。其内部框图如图 5.3所示，内部寄存器如图 5.4所示。

PCF8574 是基于 **I²C** 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 5.5所示，外设连接如图 5.6 所示。从图 5.6 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（分别对应左上下上四个方向），P4 端口接了发光管 LED2，P7 端口接了蜂鸣器输出。

5.2.3 I²C 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 **I²C** 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: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

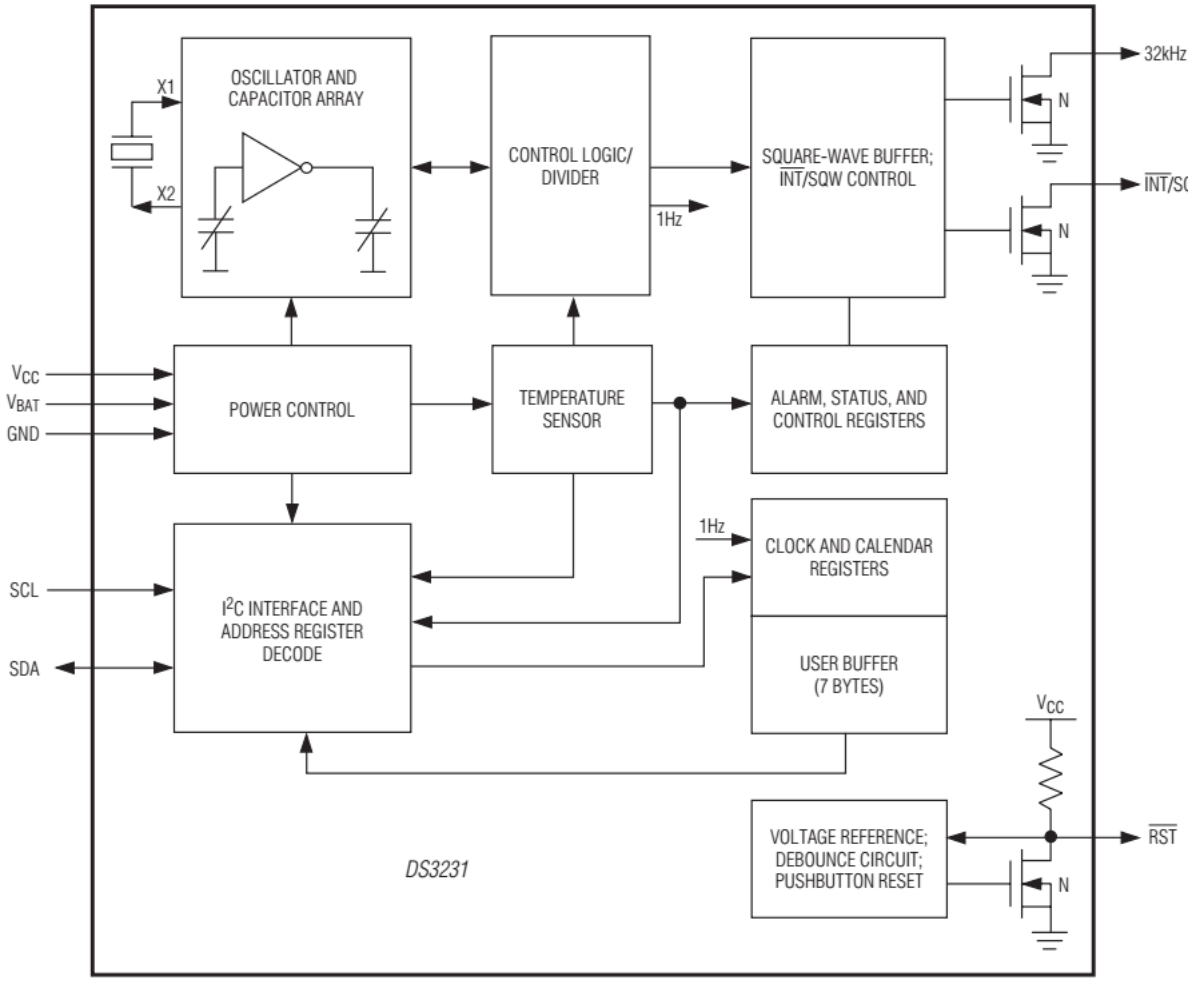


图 5.3: DS3231 内部框图

```
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

根据扫描的结果，树莓派共接入了四个 **I²C** 设备，其中地址 20 为 PCF8574 IO 扩展芯片（参考图 5.5 和图 5.6，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 6.3）；地址 68 为 DS3231 实时钟芯片（参考图 5.3）；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 **I²C** 函数的封装，可以使用 pydoc smbus 获取库帮助信息。

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
import time   # 包含相关库文件

address = 0x68
register = 0x00
bus = smbus.SMBus(1)    # 初始化i2c Bus
```

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	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/ Century	01–12 + 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/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	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	0	LSB of Temp	—

图 5.4: DS3231 内部寄存器

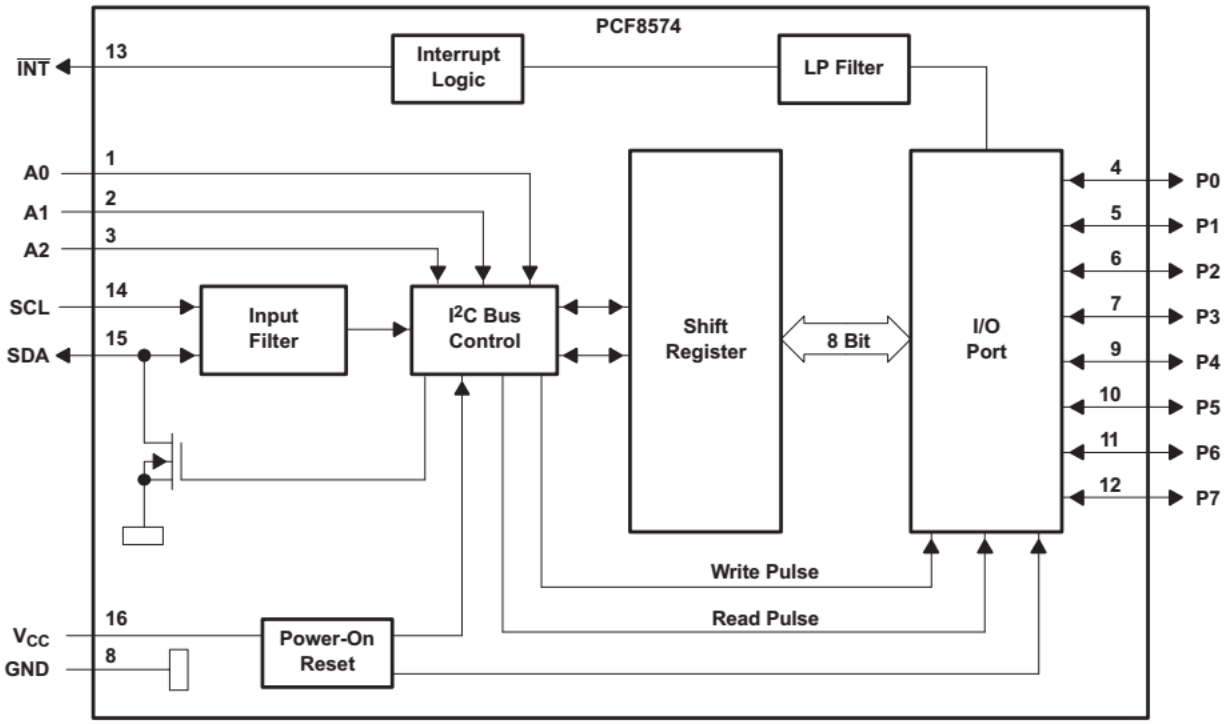


图 5.5: PCF8574 内部框图

```
# 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():
```

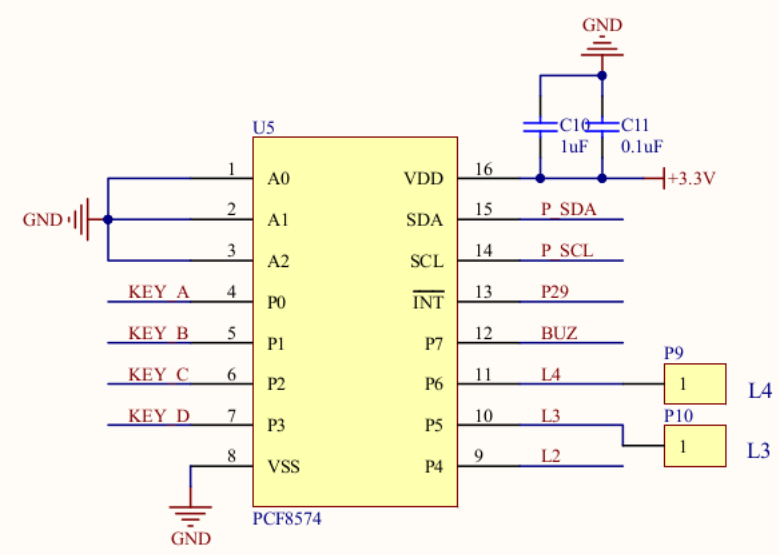


图 5.6: PCF8574 外设连接

```
    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 **I²C** 的地址，第二个参数是传递给从设备的命令，DS3231 的 **I²C** 协议规定这个命令就是后续传递数据的起始地址。根据图 5.4 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。

5.2.4 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

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势。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。

5.2.5 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 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。不包含 `:-` 的具有为事实，例如 `color(blue)` 这个函数永远为真。

swipl 是 Prolog 的一个实现，下面的交互示例假定前面的代码列表保存在 bird.pl 中。

```
$ 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.
```

大写开始的单词表示变量，因此 `bird(X)` 的查询结果给出了变量 `X` 的值。

5.2.6 pyswip 模块

pyswip 是 Python 语言与 swipl 的接口，现在将 `bird.pl` 中的事实部分删除，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('bird.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"])
```

pyswip 还支持设计 Prolog 中的外部函数，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)

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'}]
```

这个返回结果列表中是两个字典（dict）类型的元素，下面简单对字典进行一下说明。

5.2.7 Python 字典

和列表类似，字典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：

```
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：

```
print(dict['Alice'])
dict['John'] = '45'
```

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

5.3 实验内容

5.3.1 I^2C 设备基本用法

1. 访问板子的 **I^2C** 设备（RTC DS3231），对 RTC 进行初始时间配置，之后每秒钟读取一次 RTC 时间，并打印在屏幕上。

5.3.2 专家系统示例代码测试

按顺序完成示例代码，检查运行结果，加深对 Prolog 代码的理解。

5.3.3 较复杂 Prolog 程序

下面的代码是一个动物识别程序，先在 swipl 下运行查看效果，然后使用 python 替换里面的 `verify` 函数，利用 pyswip 实现类似的功能（在使用 python 调用这个 Prolog 模块的时候，先删除里面不再需要的 `go, ask, verify, undo` 定义）。这样相当于为这个 Prolog 程序实现了一个前端代码，可以提供给用户更好的使用界面。

```
/* animal.pro
  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),
    nl,
    ( (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.
```

实现 verify() 函数的时候，输入是需要验证的事实，返回值为布尔类型，表示需要确认是否具有这个事实。函数可以通过一个字典（dict）记录每个回答的结果，这样面对重复的问题就可以不必每次都回答了。

实验六　AD/DA 与人工神经网络模型

6.1　实验目的

1. 熟悉 AD/DA 原理。
2. 通过 I2C 总线访问扩展板的 PCF8591 AD/DA 芯片。
3. 了解 keras 框架和简单的人工神经网络模型搭建。

6.2　实验原理

6.2.1　模数转换原理

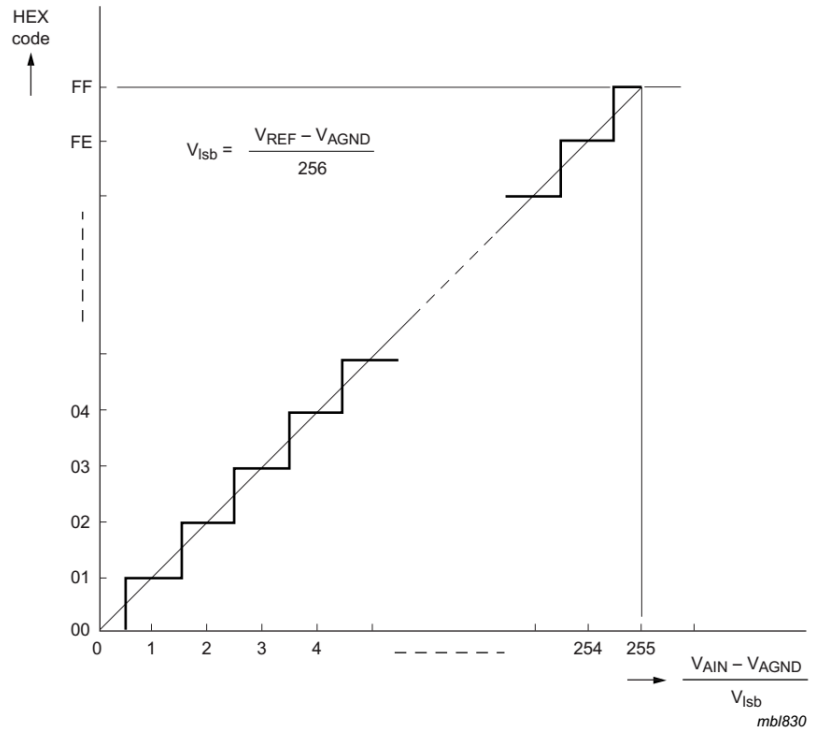


图 6.1: 单端 A/D 原理

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 6.1）。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：

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

6.2.2 数模转换原理

把数字量转换为模拟量的设备称为数模(D/A)转换器。如图 6.2,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。

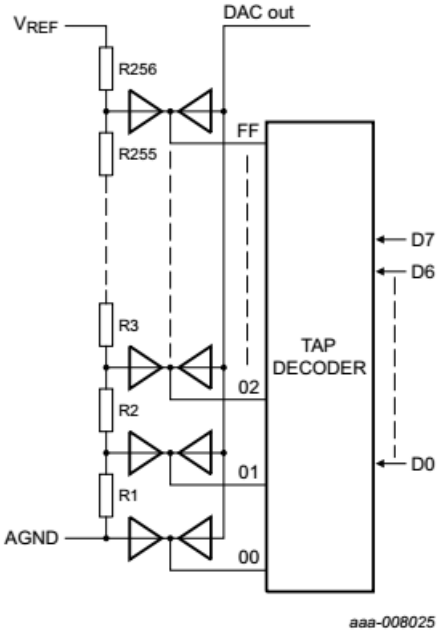


图 6.2: 数模转换原理

6.2.3 PCF8591 芯片介绍

PCF8591 芯片也挂载在树莓派的 **I²C** 总线上,其内部框图如图 6.3所示。

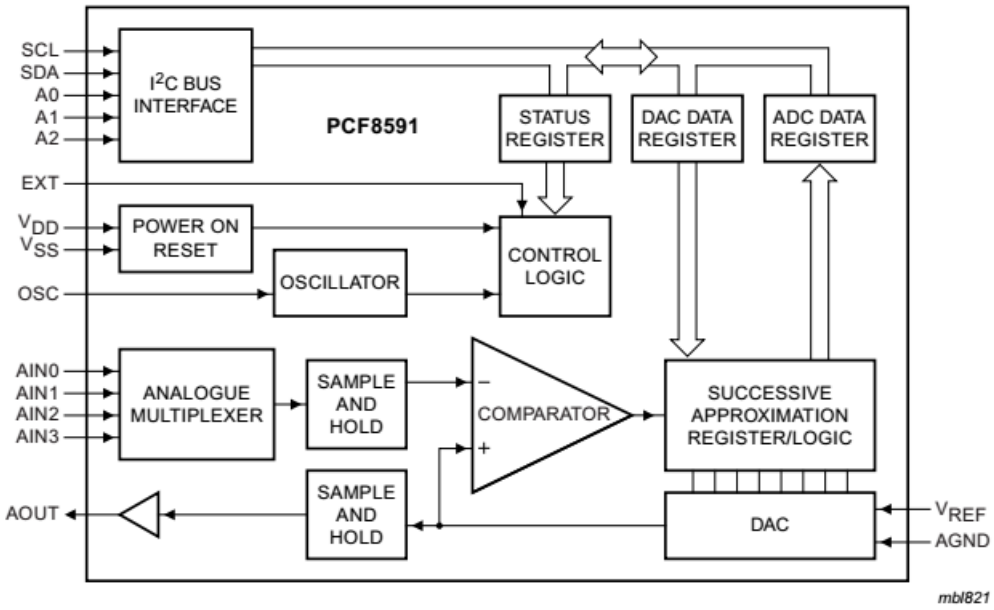


图 6.3: PCF8591 内部框图

扩展板上 PCF8591 的 A0/A1/A2 均接低电平,根据 PCF8591 地址配置规则(表 6.1),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 6.4所示：

表 6.1: PCF8591 地址配置

位	7	6	5	4	3	2	1
地址	1	0	0	1	A2	A1	A0

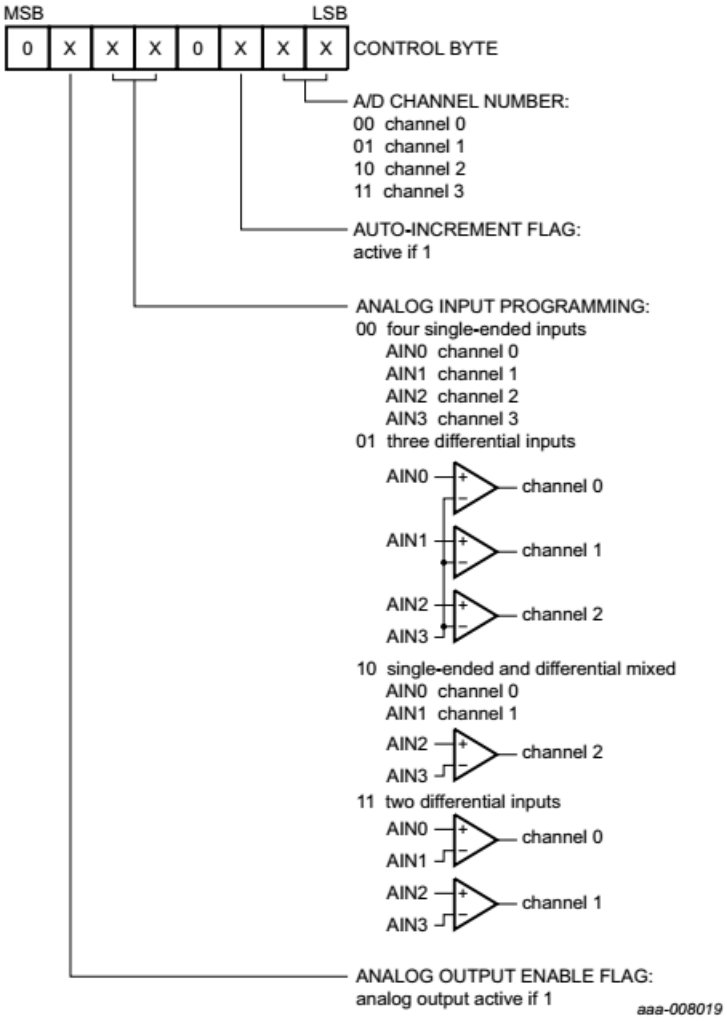


图 6.4: PCF8591 控制寄存器定义

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。

6.2.4 AD/DA 数据输出

A/D 的参考代码如下：

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

address = 0x48
A0 = 0x40
```

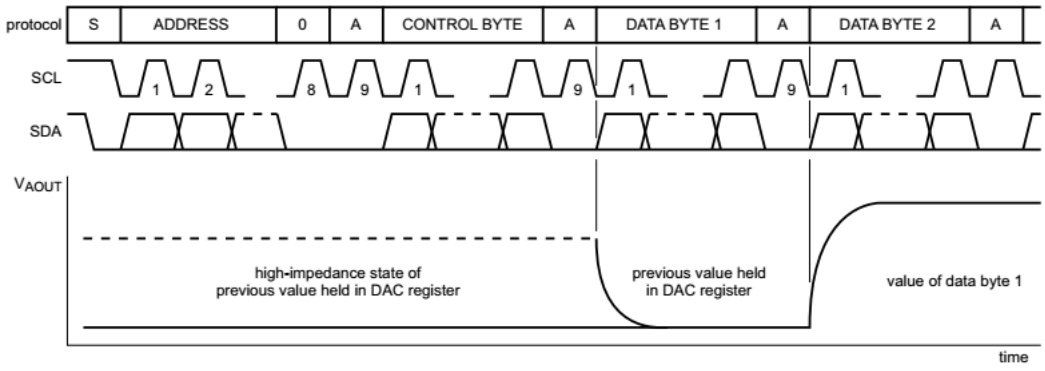


图 6.5: PCF8591 D/A 接口时序

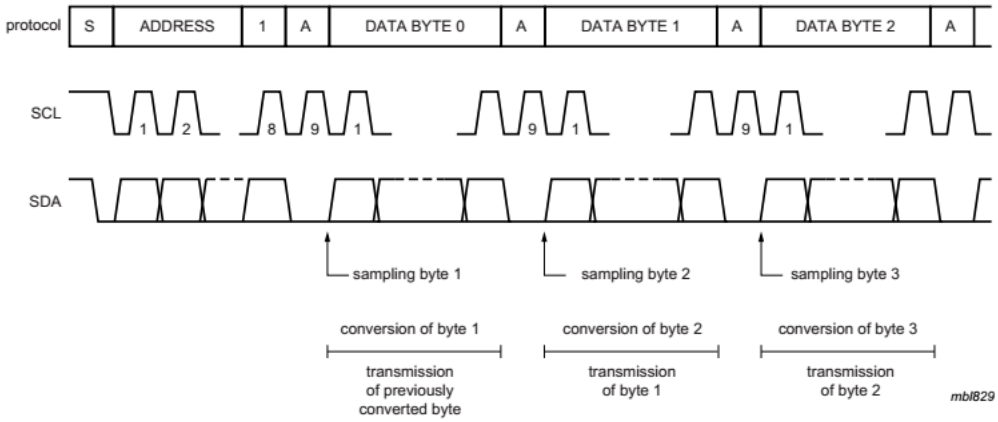


图 6.6: PCF8591 A/D 接口时序

```
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。

6.2.5 人工神经网络简介

在 2.2.5 小节中曾经简单的介绍了人工神经网络，但那时主要以线性回归模型为主，这里将进一步了解由神经元组成的网络。图 2.3 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）。

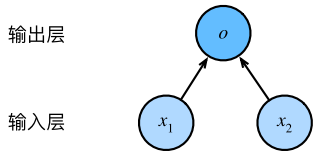


图 6.7: 线性回归模型

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 6.8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 2.2.5 小节中提到了 sigmoid 函数，目前更常用的是一个更简单的 ReLU（Rectified Linear Unit）函数：

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

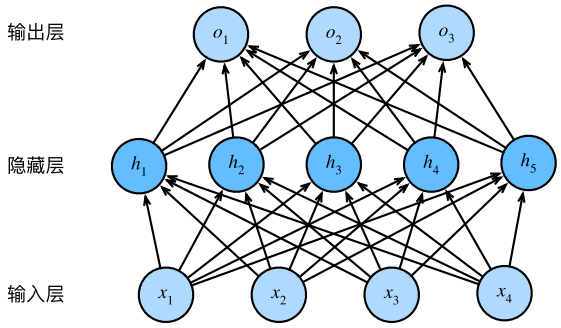


图 6.8: 多层感知器模型人工神经网络

6.2.6 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 之间的通过运算进行连接，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。

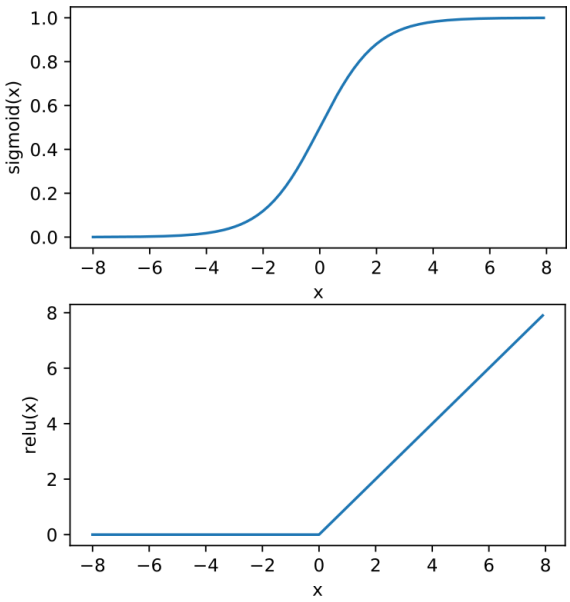


图 6.9: sigmoid 和 ReLU 函数

- 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 6.8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(3, 5)
        self.out   = nn.Linear(5, 4)

    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
- 定义网络结构 model
- 定义损失函数

- 定义优化算法 optimizer
- 训练
- 准备好 tensor 形式的输入数据和标签 (可选)
- 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。

6.2.7 Python 多线程

多线程是一种并行执行技术，在特定的应用环境下，采用并行执行的方式可以提高程序的执行效率，或者可以使程序编写简便，增加程序的可读性。实现并行执行的方法有很多种，这里介绍 threading 模块。

使用 threading 模块建立线程类的简单方法是调用 Thread 构造函数：

```
from threading import Thread
mythread = Thread(target=callable, args=())
```

其中 callable 是新建线程所运行的函数名，在主程序中可以通过 `mythread.start()` 启动线程，调用 `mythread.join()` 等待线程结束。如果希望主程序退出时，线程可以自动退出，可以通过 `mythread.setDaemon(True)` 来设置。

需要注意的是这里的并行并没有利用多核处理器同时执行，而是采用分时的方式使得两部分的程序看起来都在运行。而在所有的并行运行技术中，都要注意共享资源的使用，避免竞争和死锁。

6.3 实验内容

6.3.1 AD/DA 模块的检验

1. 使用面包板，对电位器的电压信号进行采样并存储，在 OLED 屏幕上展示信号的电压信息。参考电路如图 6.10。
2. 调整 DA 输出，点亮发光二极管，设计二极管的发光模式（如呼吸灯）。参考电路如图 6.10。
3. 利用多线程技术，用电位器的输出控制发光二极管的呼吸灯频率。

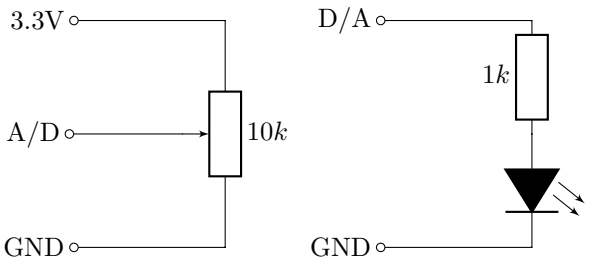


图 6.10: 滑动变阻器电路和发光二极管电路

6.3.2 使用 Pytorch 建立简单模型

下面是 PyTorch 的一个多层感知机示例程序，用来对 MNIST 数据进行分类：

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms

# 设备配置, 部署GPU加速
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Hyper-parameters, 定义各层规模及其他超参数
input_size = 784
hidden_size = 500
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001

# MNIST dataset, 准备数据集, 转换为Tensor数据格式
train_dataset = torchvision.datasets.MNIST(root='data',
                            train=True,
                            transform=transforms.ToTensor(),
                            download=True)

test_dataset = torchvision.datasets.MNIST(root='data',
                            train=False,
```

```
                            transform=transforms.ToTensor())

# DataLoader类，进行batch_size(每个batch的大小),
# shuffle(是否进行shuffle操作),
# num_workers(加载数据的时候使用几个子进程)等操作
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                            batch_size=batch_size,
                            shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                            batch_size=batch_size,
                            shuffle=False)

# 定义网络结构，Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

def forward(self, x):
    # 向前传播，模型的计算流程
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out


model = NeuralNet(input_size, hidden_size, num_classes)

# Loss and optimizer
# 定义损失函数，使用的是交叉熵函数
criterion = nn.CrossEntropyLoss()
# 定义迭代优化算法，使用的是Adam (Adaptive Moment Estimation)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model 迭代训练
def train_model(model, train_loader):
    total_step = len(train_loader)
    for epoch in range(num_epochs):
        for i, (images, labels) in enumerate(train_loader):
            # Move tensors to the configured device
            images = images.reshape(-1, 28 * 28).to(device)
            labels = labels.to(device)
```

```
            # Forward pass, 前向传播计算网络结构的输出结果
            outputs = model(images)
            # 计算损失函数
            loss = criterion(outputs, labels)

            # Backward and optimize 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()

            # 打印训练信息
            if (i + 1) % 100 == 0:
                print('Epoch [{}/{}], Step [{}/{}], loss: {:.4f}'.format
                      (epoch + 1, num_epochs, i + 1, total_step, loss.item()))
    # 保存训练好的模型
    torch.save(model.state_dict(), 'model.ckpt')

# Test the model 测试模型
def test_model(model, test_loader, device):
    model.load_state_dict(torch.load('model.ckpt', map_location=device))
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in test_loader:
            images = images.reshape(-1, 28 * 28).to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        print('Accuracy is: {}%'.format(100 * correct / total))

train_model(model, train_loader)
test_model(model, test_loader, device)
```

由于官方的 MNIST 数据是 28×28 的图像，数据量比较大，使用树莓派进行训练很慢。请修改代码使用 scikit-learn 中的 8×8 的数据，并设置合适的参数完成分类，比较模型的分类准确性。

实验七　摄像头的使用

7.1　实验目的

1. 了解树莓派 CSI 接口摄像头的使用方法
2. 初步了解 opencv 中对图像的处理方法
3. 学习使用 opencv 对图像中物体进行跟踪

7.2　实验原理

机器视觉一直是人工智能中研究的热点，本实验将使用 OpenCV 对视频信号进行采集，并对采集到的数据进行初步的处理。

7.2.1　OpenCV 简介

OpenCV（Open Source Computer Vision Library）是一个开源的机器视觉程序库，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 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。

例如下面的代码就用来打开一副保存在电脑上的图像

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

OpenCV 包含的功能模块非常多，这里介绍两个最常用的功能，其它的模块与函数的用法请参考其它书籍或者在线文档。

7.2.1.1 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，在 OpenCV 中，缺省的颜色空间是 BGR，也就是 RGB 的另外一种排序方式。HSV 是根据颜色的直观感受来进行分解的表示法，包括色调（H）、饱和度（S）和明度（V）三个分量。

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。在 OpenCV 中，可以用 `cvtColor` 函数对颜色空间进行转换，然后使用 `inRange` 对图像中的色彩进行过滤。参考代码如下：

```
import numpy as np
import cv2
cam = cv2.VideoCapture(0)  # 初始化摄像头
while ( True ):
    ret, frame = cam.read() # 读取摄像头数据
    hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)  # 转换颜色空间
    # 通过颜色设计模板
    image_mask=cv2.inRange(hsv,np.array([40,50,50]), np.array([80,255,255]))
    # 计算输出图像
    output=cv2.bitwise_and(frame,frame,mask=image_mask)
    cv2.imshow('Original',frame) # 显示原始图像
    cv2.imshow('Output',output)  # 显示输出图像
    if cv2.waitKey(1) == ord("q"):     # 等待按键
        break
cv2.destroyAllWindows()
cam.release()
```

7.2.1.2 形状识别

从图像中识别特定的形状是一种常见的任务，而识别形状的基础是边界的识别。OpenCV 中包含一个 Canny 边界探测器，它的工作原理如下：

1. 通过高斯算法过滤图像噪声
2. 计算图像的梯度（基于 L1 范数或者 L2 范数）
3. 压缩高梯度线的宽度，获得比较细的边界线
4. 两个阈值参数用来选出图像的最终边界：低于 threshold1 的梯度会被排出边界，高于 threshold2 的梯度将包含在最终输出的边界中。而两个阈值之间的点必须与高于 threshold2 的点相连才被包含在输出中。

在边界探测的基础上就可以识别形状了，例如 cv2.HoughCircles() 用来识别图像中的圆形。它包含如下几个参数：

- 输入图像
- 识别算法，目前只能是 `HOUGH_GRADIENT`

- dp, 图像分辨率与算法分辨率的倒数

$$dp = \frac{\text{accumulator resolution}}{\text{image resolution}}$$

- minDist, 识别圆心的最小距离
- param1, Canny 算法中的 threshold2
- param2, 圆心识别算法的阈值
- 圆形半径的最小和最大值

下面是一个识别圆形的例子：

```
#!/bin/python
import cv2
cam = cv2.VideoCapture(0)   # 打开摄像头
while (True):
    ret , frame = cam.read()    # 获取摄像头数据
    grey = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY) # 色彩变换
    blur = cv2.blur(grey,(5,5)) # 过滤噪声
    circles = cv2.HoughCircles(blur,   # 识别圆形
    method=cv2.HOUGH_GRADIENT,dp=1,minDist=200,
    param1=100,param2=33,minRadius=30,maxRadius=175)
    if circles is not None:     # 识别到圆形
        for i in circles [0,:]: # 画出识别的结果
            cv2.circle(frame,(i[0],i[1]),i[2],(0,255,0),2)
            cv2.circle(frame,(i[0],i[1]),2,(0,0,255),3)
    cv2.imshow('Detected',frame)    # 显示识别图像
    if cv2.waitKey(1) == ord("q"):        # 等待按键
        break
cv2.destroyAllWindows()         # 关闭窗口
cam.release()   # 释放摄像头
```

图 7.1 是运行上面的例子的效果。

7.2.2 摄像头的使用

OpenCV 有直接对摄像头的支持，前面的示例代码也有对摄像头的使用。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。

```
#!/bin/python
import cv2  # 加载 OpenCV 库
cam = cv2.VideoCapture(0)       # 打开摄像头
cam.set(3, 1024)    # 设置图像宽度
cam.set(4, 768)     # 设置图像高度
while(True):
    ret, frame = cam.read() # 读入一帧图像
    cv2.imshow('Video Test',frame)  # 显示图像
    if cv2.waitKey(1) == ord("q") :       # 等待按键
        break
```

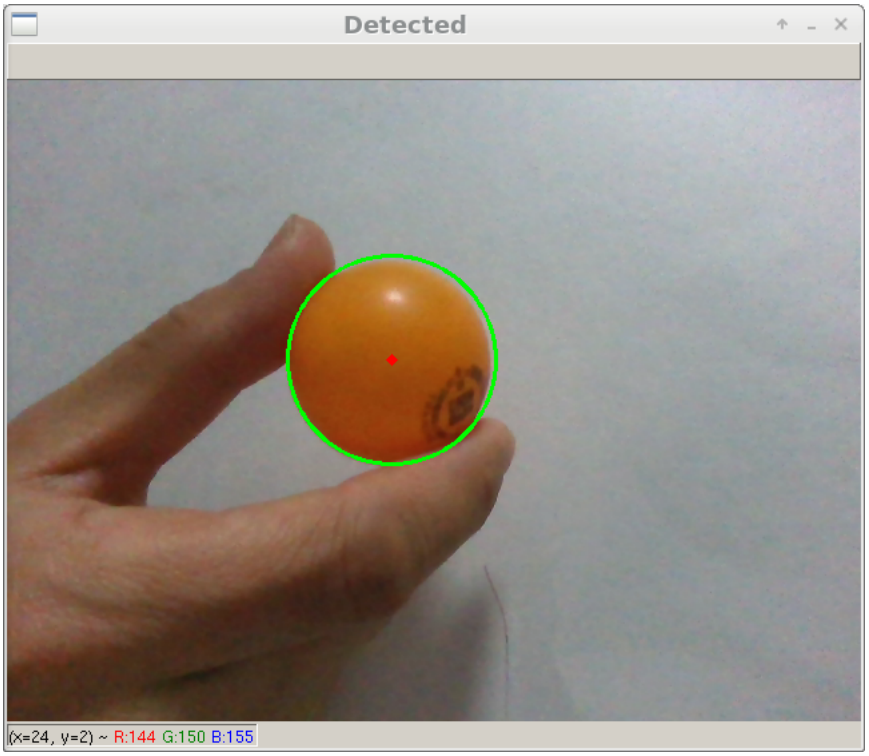


图 7.1: 从摄像头识别的乒乓球

```
cam.release()        # 释放摄像头硬件
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。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（图 7.2），就必须使用不同的方法[1]。

树莓派系统包含一个 `python-picamera` 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：

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2                          # 导入需要的库

camera = PiCamera()
camera.resolution = (640, 480)    # 设置分辨率
camera.framerate = 32             # 设置帧率
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)                   # 等待摄像头模块初始化

for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
    image = frame.array
    cv2.imshow("Frame", image)    # 显示图像
    key = cv2.waitKey(1) & 0xFF   # 等待按键
```

[1]通过加载 bcm2835-v4l2 驱动（modprobe bcm2835-v4l2），树莓派也可以采用标准的 v4l 接口使用摄像头。

图 7.2: CSI 接口摄像头

```
rawCapture.truncate(0)        # 准备下一副图像
if key == ord("q"):
    break
```

7.3 实验内容

7.3.1 摄像头输入测试

采用第 7.2.2 节中的代码对摄像头进行测试，如果摄像头不正常请及时与代课老师联系。

7.3.2 色彩识别

在摄像头拍摄范围内将皮肤的颜色过滤出来。通过采用模板的方式，将摄像头拍摄的内容除了皮肤的部分过滤掉。在测试的时候可以用手来做测试，屏幕输出的内容应该只有手的图像。

选做：将手部图像的中心点找出来，在屏幕上加强显示。

7.3.3 形状识别

识别如下的一个黑色圆盘，在输出图像中对这个圆盘进行标记。

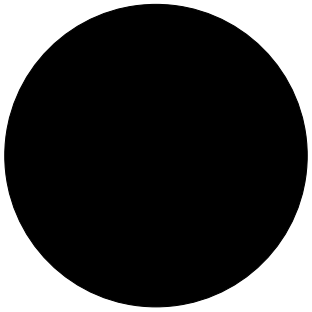


图 7.3: 检测目标

实验八　语音识别

8.1　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

8.2　实验原理

8.2.1　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。

树莓派中的 CPU 性能较差，而 GPU 较强大，omxplayer 是专门针对树莓派的 GPU 的播放器，支持硬件解码。这里也可以用 omxplayer 进行播放：

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

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：

```
CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 2
RATE = 44100
```

```
p = pyaudio.PyAudio()
stream = p.open(format=FORMAT,
        channels=CHANNELS,
        rate=RATE,
        input=True,
        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：

```
wf = wave.open(filename, 'wb')
wf.setnchannels(CHANNELS)
wf.setsampwidth(p.get_sample_size(FORMAT))
wf.setframerate(RATE)
wf.writeframes(data)
```

8.2.2 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 $O = (o_1, o_2, ..., o_T)$，对应的句子 $W = (w_1, w_2, ..., o_N)$，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。

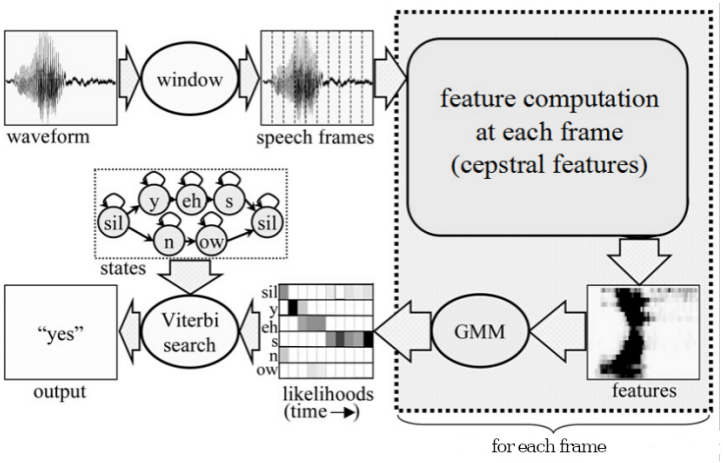


图 8.1: HMM-GMM 模型语音识别示意图

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 Coefficents 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：

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

```
from python_speech_features import mfcc
# 特征提取, feat = compute_mfcc(wadict[wavid])
def compute_mfcc(file):
    fs, audio = wavfile.read(file)
    mfcc_feat = mfcc(audio)
    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：

```
class Model():
    def __init__(self, CATEGORY=None, n_comp=3, n_mix = 3, cov_type='diag', n_iter=1000):
        super(Model, self).__init__()
        self.CATEGORY = CATEGORY
        self.category = len(CATEGORY)
        self.n_comp = n_comp
        self.n_mix = n_mix
        self.cov_type = cov_type
        self.n_iter = n_iter
        # 关键步骤, 初始化models, 返回特定参数的模型的列表
        self.models = []
        for k in range(self.category):
            model = hmm.GMMHMM(n_components=self.n_comp, n_mix = self.n_mix,
                  covariance_type=self.cov_type, n_iter=self.n_iter)
            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 μ 表示高斯分布的期望向量，Σ 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 Σ 形成的三维张量。参数 covariance_type，取值为”full” 意味所有的 μ, Σ 都需要指定。取值为“spherical”则 Σ 的非对角线元素为 0，对角线元素相同。取值为“diag”则 Σ 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 Σ。

然后，用同一种类的数据训练特定的模型：

```
# 提取声学特征
mfcc_feat = compute_mfcc(wavdict[x])
# hmm-gmm 模型训练
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。

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

8.3 实验内容

8.3.1 测试麦克风

根据提供的代码，测试麦克风录音效果。

8.3.2 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。

8.3.3 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。使用 HMM-GMM 模型对语音命令进行识别。

8.3.4 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。

实验九　自然语言处理入门

9.1　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

9.2　实验原理

9.2.1　词向量的编码与处理方法

9.2.1.1　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 9.1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。

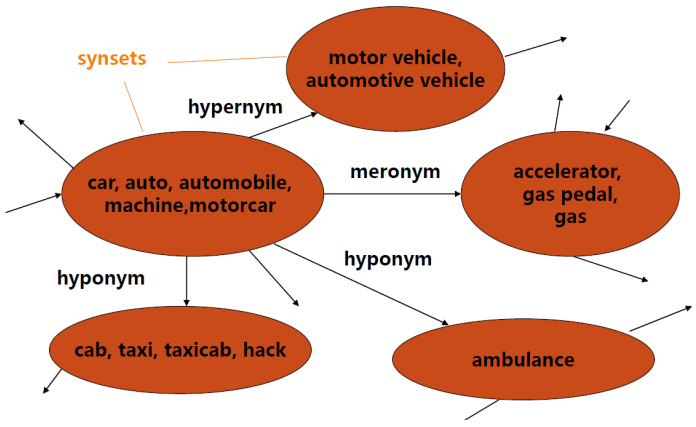


图 9.1: A WordNet Snapshot

9.2.1.2　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。

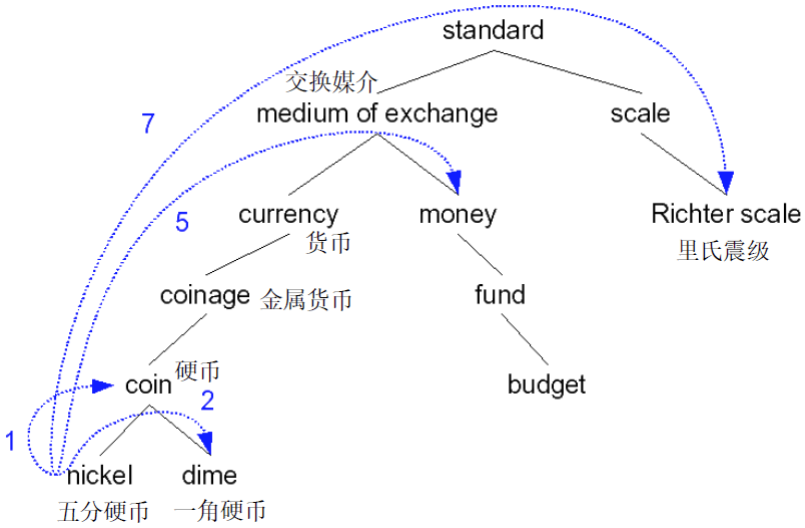


图 9.2: Path Based Similarity 示意

图 9.2 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(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

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

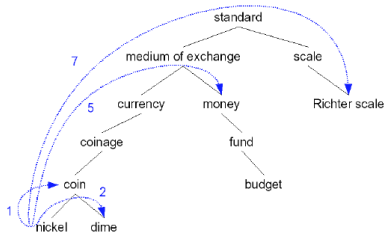


图 9.3: P(concept) 含义

C(concept) = -log P(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。

9.2.1.3 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。

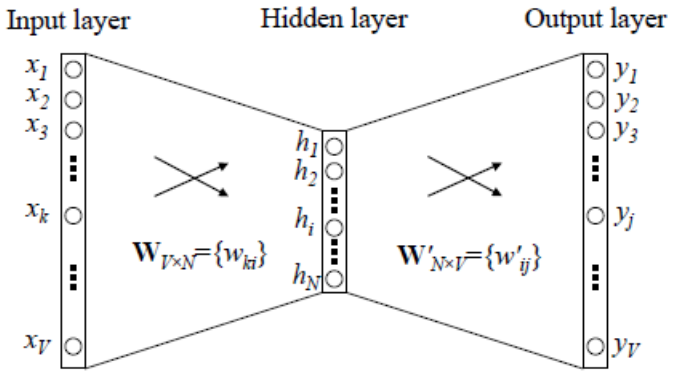


图 9.4: Word2Vec 模型结构

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 9.5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 9.5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。

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

ensim 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；

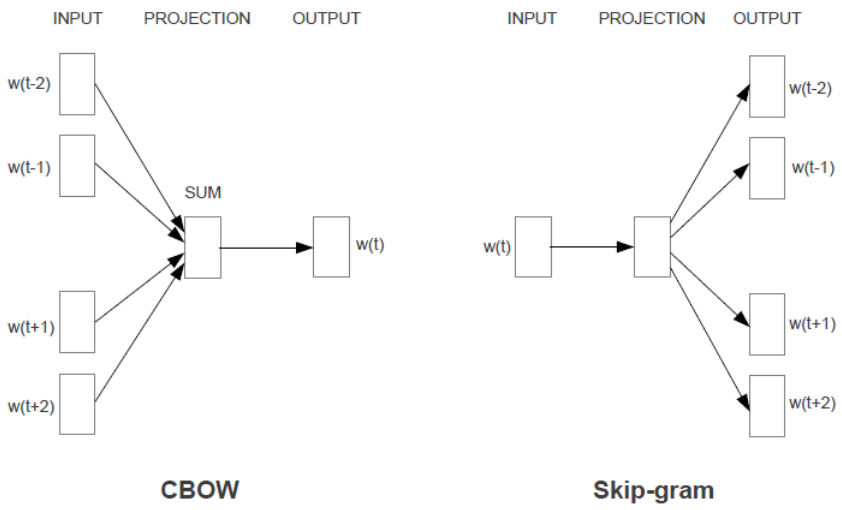


图 9.5: CBOW 模型与 Skip-gram 模型

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。

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。

9.2.1.4 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。

9.2.2 Twitter 文本情感分析

9.2.2.1 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。

9.2.2.2 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。

LSTM 的总体框架如图 9.6 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 hh_t，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。

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 9.7所示。其中 W_f 为计算遗忘门对应的权重，b_f 为偏置，σ 为激活函数。

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 9.8 所示。

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 9.9 所示。

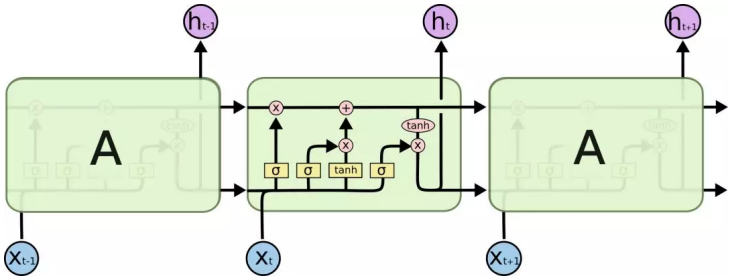
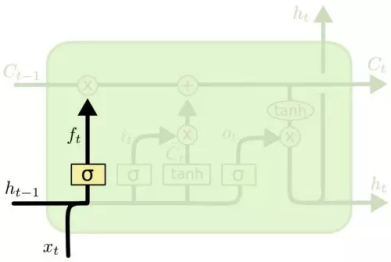
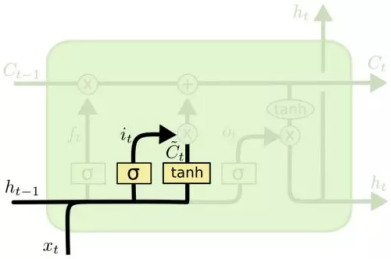


图 9.6: LSTM 总体框架



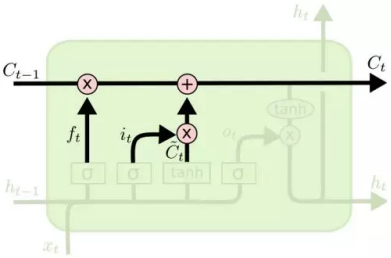
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

图 9.7: 遗忘门计算



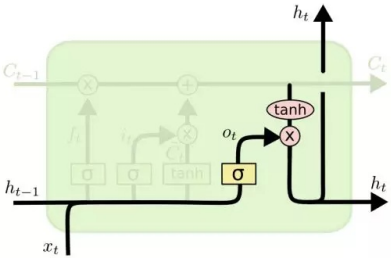
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

图 9.8: 记忆门和临时细胞状态计算



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

图 9.9: 当前时刻细胞状态计算



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

图 9.10: 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 9.10 所示。

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

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?? 所示，分词操作将句子划分为 [“我”, ”爱”, ”中国”] 三个词，前向 LSTML 得到 h_L0, h_L1, h_L2，后向 LSTMR 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。

```
# PyTorch 双向 LSTM 模型定义
# bidirectional 控制是否是双向 LSTM
self.lstm = nn.LSTM(embedding_dim, hidden_dim, layer_num, dropout=drop_prob,
    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。

```
# 全连接层做情感分类
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
- ouput_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

9.2.3 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，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。

```
module avial anaconda        # 查看可用的 conda 环境
module load anaconda/3.7.1   # 载入可用的 conda 环境
conda create -n myEnvPython python=<version>       # 创建虚拟环境
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。

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

脚本 job.sh 的示例如下：

```
#!/bin/bash
#SBATCH -o nohup.out
#SBATCH -p compute
#SBATCH -q normal
#SBATCH -J word2vec
#SBATCH --time=24:00:00

python3 main.py
```

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

9.3 实验内容

9.3.1 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。

```
# 返回基于语义词典的相似度
wupAns, linAns = DicSimilarity(rawData)
#返回基于 Word2Vec 的相似度
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。

9.3.2 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。

```
# 模型超参数设置
batch_size = 25
vocab_size = len(vocab_to_int) + 1
output_size = 3
embedding_dim = 400
hidden_dim = 256
n_layers = 2
epochs = 5
clip = 5
print_every = 10
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。

实验十　TensorFlow 应用

10.1　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。

10.2　实验原理

10.2.1　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：

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

10.2.2　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。

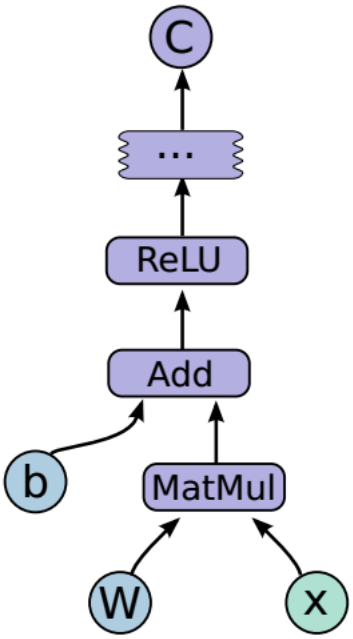


图 10.1: TensorFlow 计算图

图 10.2: MNIST 手写数字图片

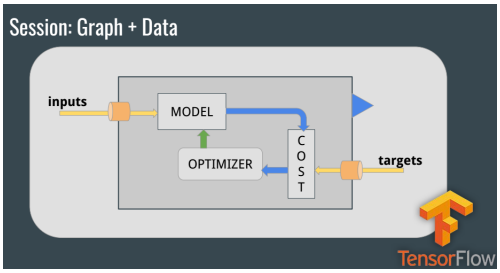


图 10.3: TensorFlow 工作流程

10.2.2.1 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。

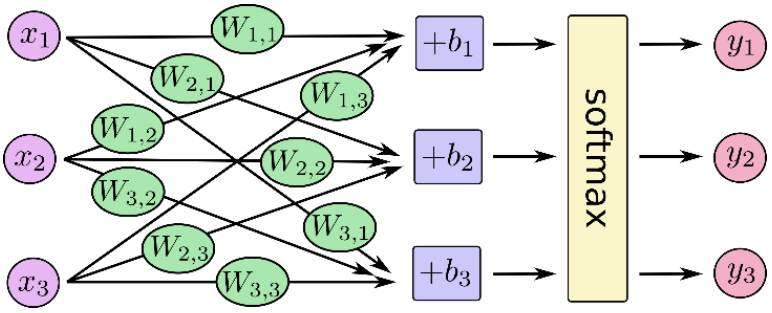


图 10.4: Softmax 回归

第二步:用 softmax 函数可以把这些证据转换成概率 y

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

其中:

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

10.2.2.2 实现回归模型

根据上述模型分析,使用 NumPy 实现回归模型。

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

10.2.2.3 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:

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

其中 y 是我们预测的概率分布, y' 是实际的分布。

```
y_ = tf.placeholder("float", [None,10])
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。

```
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。

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

10.3 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。