第 1 章
深度学习快速入门

源于人工神经网络的深度学习技术随着可用于机器学习数据的积累而迅速发展。深度学习方法可以从大量的训练数据中自动学习出实例的特征，在语音识别、图像识别、自然语言处理等领域得到了广泛的应用。目前已有一些实现深度学习的框架，其中 TensorFlow 是一个流行的框架。

1.1 各种深度学习应用

目前，深度学习技术已应用于安防监控领域识别火灾警情和机器翻译多国语言文字等。

电话机器人首先做好几个相对固定的回答录音，然后识别客户回答的几个关键词，返回相应的应答路由。

JSON 格式的例子如下：

```
"keys": [
    "你是谁",
    "哪位",
    "你叫什么",
    "姓什么",
    "怎么称呼",
    "贵姓"
],
"response": [
    [
    "我姓王，你叫我小王就可以了",
    "是这样的，免贵姓王，您可以叫我小王"
    ]
```

可以采用模板引擎来编写应答。已有各种编程语言实现的模板引擎，例如用 Java 语言实现的 FreeMarker（https://freemarker.apache.org/）或者用 Python 语言实现的 Cheetah3 （http://cheetahtemplate.org/）。

另外，医疗系统可以根据语音识别患者主诉的结果，实现病历的自动填写。

1.2 准备开发环境

当前，很多语音识别应用都是在 Linux 操作系统下开发的。Linux 是围绕 Linux 内核构建的免费和开源软件操作系统系列。Linux 来源于 UNIX，是 UNIX 操作系统的开放源代码实现。Linux 发行版是一个基于 Linux 内核的由软件集合构成的操作系统。通常，Linux 用户通过下载一个 Linux 发行版获得操作系统。Linux 有一些常用的发行版，如 CentOS 和 Ubuntu 等版本。Ubuntu 是由 Canonical 公司开发的基于 Debian 的开源 Linux 操作系统；CentOS 是 Red Hat Enterprise Linux 的免费复制版。本节首先介绍在 Ubuntu 和 CentOS 下安装 Python，然后介绍在 Linux 下开发 Python 应用的编辑器 Micro。

在 Windows 下，可以使用 Sublime 这样的文本编辑器编写 Python 代码，也可以使用 PyDev 或者 PyCharm 集成开发环境。

1.2.1 Linux 基础

有些语音识别系统运行在 Linux 服务器中。为了远程登录 Linux 服务器，可以安装 KiTTY（https://www.fosshub.com/KiTTY.html）。在 KiTTY 的配置界面输入 IP 地址，用户名和密码后登录 Linux 服务器。如果是用 root 账户登录，则终端提示符是#，否则终端提示符是$。

查看 Ubuntu 操作系统版本号：

```
$cat /etc/issue
Ubuntu 18.04 LTS \n \l
```

或者：

```
$ lsb_release -r
Release:       18.04
```

获取 Ubuntu 的代号：

```
$ lsb_release -c
  Codename:      bionic
```

ls 命令用于列出当前目录下的文件。有的命令比较长，为了快速输入，可以用 Tab 键补全命令。History 是显示历史命令，用上箭头选择最近运行过的命令可再次执行。

可以使用支持 SSH 协议的终端仿真程序 SecureCRT 连接到远程 Linux 服务器。因为可以保存登录密码，所以比较方便。除了 SecureCRT，还可以使用开源软件 PuTTY（http://www.chiark.greenend.org.uk/~sgtatham/putty），以及可以保存登录密码的 PuTTY Connection Manager。

在终端启动的进程断开连接后进程会停止运行。为了让进程继续运行，可以使用 nohup 命令。

如果需要安装软件，可以下载对应的 RPM 安装包，然后使用 RPM 安装。但操作系统对应的 RPM 安装包找起来往往比较麻烦。一个软件包可能依赖其他的软件包。为了安装一个软件可能需要下载其他几个它所依赖的软件包。

为了简化安装操作，可以使用黄狗升级管理器，一般简称 yum。yum 会自动计算出程序之间的相互关联性，并且计算出完成软件包的安装需要哪些步骤。这样在安装软件时，不会再被那些关联性问题所困扰。

yum 软件包管理器自动从网络下载并安装软件。yum 有点类似 360 软件管家，但是不会有商业倾向的推销软件。例如安装支持 wget 和 rzsz 命令的软件：

```
#yum install wget
#yum install lrzsz
```

Windows 格式文本文件的换行符为\r\n，而 Linux 文件的换行符为\n。

dos2unix 是将 Windows 格式文件转换为 Linux 格式的实用命令。dos2unix 命令其实就是将文件中的\r\n 转换为\n。

开发语音识别系统的过程中，可能会用到大量的数据文件。如需要在 Linux 操作系统上维护同一文件的两份或多份副本，除了保存多份单独的物理文件副本之外，还可以采用保存一份物理文件副本和多个虚拟副本的方法。这种虚拟的副本就称为链接。链接是目录中指向文件真实位置的占位符。在 Linux 中有两种不同类型的文件链接：符号链接和硬链接。

符号链接就是一个实实在在的文件，它指向存放在虚拟目录结构中某个地方的另一个文件。这两个通过符号链接在一起的文件，彼此的内容并不相同。

如果现有空间不够用，可以增加存储设备后扩容。首先用 lsblk 命令查看现有空间情况。在一个 Linux 账号中显示如下：

```
[root@localhost ~]#lsblk
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0      11:0    1 1024M  0 rom
xvda    202:0    0  100G  0 disk
├─xvda1 202:1    0    8G  0 part [SWAP]
```

```
└─xvda2 202:2    0   32G  0 part /mnt
xvde    202:64   0 1000G  0 disk
```

创建一个要扩展的目录：

```
#mkdir /ext
```

加载文件系统到这个目录下：

```
#mount /dev/xvde /ext
```

确认加载成功：

```
[root@localhost ~]#df -m
Filesystem        1M-blocks     Used     Available     Use%     Mounted on
/dev/xvda2        32752         32496    256           100%     /
devtmpfs          32108         0        32108         0%       /dev
tmpfs             32020         1        32020         1%       /dev/shm
tmpfs             32020         186      31835         1%       /run
tmpfs             32020         0        32020         0%       /sys/fs/cgroup
tmpfs             6374          1        6374          1%       /run/user/0
/dev/xvde         1007801       77       956509        1%       /ext
```

对于大的文件可以使用 wget 命令在后台下载。

```
#wget -bc <path>
```

这里的参数 b 表示在后台运行；参数 c 表示支持断点续传。

可以在 Windows 下编辑文本文件，然后使用 Perl 把 Windows 下的文本文件转换成 Linux 可识别的格式：

```
#perl -p -e 's/\r$//' < winfile.txt > unixfile.txt
```

在 Ubuntu 操作系统下安装 Python3：

```
#apt install python3
```

Ubuntu 系统上默认的 root 密码是随机生成的，而 root 权限是通过 sudo 命令授予的。可以在终端输入命令 sudo passwd，然后输入当前用户的密码，按回车键，终端会提示输入新的密码并确认，此时的密码就是 root 新密码。修改成功后，输入命令 su root，再输入新的密码即可。

可以使用 sed(Stream EDitor)命令查找和替换文件中的文本。例如：

```
#sed -i 's/original/new/g' file.txt
```

命令行参数说明如下：

-i：-i = in-place（保存回原始文件）。

命令字符串说明如下：

s：s =替换命令。

original：original =描述要替换的单词的正则表达式（或者只是单词本身）。

new：new =用来替换的目标文本。

g：g = global（即替换所有而不仅仅是第一次出现的）。

file.txt：file.txt =文件名。

例如把 cmd.sh 中的 queue.pl 替换成为 run.pl，替换结果输出到 cmd.local.sh 文件。

```
#sed 's/queue.pl/run.pl/g' cmd.sh > cmd.local.sh
```

1.2.2　Micro 编辑器

为了方便在服务器端开发 Python、Perl、Shell、C++相关应用，可以采用 Micro（https://github.com/zyedidia/micro）这样的终端文本编辑器。

可以在/home/soft/micro 目录下运行：

```
#curl https://getmic.ro | bash
```

设置成在任意路径均可运行 Micro：

```
#cd /usr/bin
#sudo ln -s /home/soft/micro/micro micro
```

或者编辑/etc/profile 文件，增加 micro 所在的路径到 PATH 环境变量/home/soft/micro。

```
#./micro /etc/profile
```

增加如下行：

```
export PATH=/home/soft/micro:$PATH
```

可以使用它编辑配置文件：

```
#./micro  run.pl
```

输入：

```
die "run.pl: Hello Error";
```

这里的 die 表示终止脚本运行，并显示出 die 后面的双引号中的内容。

保存文件后，按 Ctrl+Q 组合键退出。

1.2.3　Shell 基础

Shell 是用户和 Linux 内核之间的接口程序。在命令行提示符下输入的每个命令都由 Shell 先解释然后传给 Linux 内核。

Shell 是一个命令语言解释器，拥有自己内建的 Shell 命令集。此外，Shell 也能被系统中其

他有效的 Linux 实用程序和应用程序所调用。

Shell 具有如下主要功能。

命令解释功能：将用户可读的命令转换成计算机可理解的命令，并控制命令执行。

输入/输出重定向：操作系统将键盘作为标准输入，将显示器作为标准输出。当这些定向不能满足用户需求时，用户可以在命令中用符号“>”或“<”重新定向。

管道处理：利用管道将一个命令的输出送入另一个命令，实现多个命令组合完成复杂命令的功能。

系统环境设置：用 Shell 命令设置环境变量，维护用户的工作环境。

程序设计语言：Shell 命令本身可以作为程序设计语言，将多个 Shell 命令组合起来，编写能实现系统或用户所需功能的程序。

Shell 有很多种，如 zshell 和 fish。这里介绍 Bash（Bourne Again Shell）。

可以使用 ps 命令查看当前使用的是哪种 shell。

```
#ps
   PID TTY          TIME CMD
19977 pts/1    00:00:00 bash
50063 pts/1    00:00:00 ps
```

在屏幕上打印“Hello”：

```
echo "Hello"
```

将 ABC 分配给 a：

```
a=ABC
```

输出 a 的值：

```
echo $a
```

在屏幕上打印 ABC。

将 ABC.log 分配给 b：

```
b=$a.log
```

输出 b 的值：

```
#echo $b
ABC.log
```

把文件“ABC.log”的内容写入到 testfile：

```
cat $b > testfile
```

“指令 --help”会输出帮助信息。

可以把重复执行的 Shell 脚本写入一个文本文件。在 Linux 中，文件扩展名不作为系统识别文件类型的依据，但是可以作为我们识别文件的依据，可以简单地将脚本文件名以.sh 结尾。

在 Linux 下，可以通过 vi 命令创建一个诸如 script.sh 的文件：vi script.sh。创建好脚本文件后就可以在文件内按脚本语言要求的格式编写脚本程序了。

在创建的脚本文件中输入以下代码并保存退出：

```
#! /bin/bash
echo "hello world!"
```

添加脚本文件的可执行运行权限 chmod 777 script.sh，之后运行文件./script.sh 得到结果：

```
hello world!
```

Shell 脚本中用#表示注释，相当于 C 语言的//注释。但如果#位于第一行开头，并且是#!（称为 Shebang）则例外，它表示该脚本使用后面指定的解释器/bin/sh 解释执行。每个脚本程序必须在开头包含这个语句。

使用参数 n 检查语法错误，例如：

```
#bash -n ./test.sh
```

如果 Shell 脚本有语法错误，则会提示错误所在行；否则，不输出任何信息。

if 语句的语法是：

```
if [ condition ] then
   command1
elif              #和 else if 等价
   then
      command2
   else
      default-command
fi
```

这里的 fi 就是 if 反过来写。

例如，为了判断某个命令是否存在，可以使用如下的格式：

```
if which programname >/dev/null; then
   echo exists
else
   echo does not exist
fi
```

判断 yum 是否存在的例子：

```
if which yum >/dev/null; then
   echo "exists"
else
```

```
    echo "does not exist"
fi
```

case 语句的语法是：

```
case 字符串 in
    模式 1)
        语句
        ;;
    模式 2)
         语句
         ;;
    *)
         默认执行的语句
         ;;
esac
```

这里的 esac 就是 case 反过来写。例如：

```
extension="png"
case "$extension" in
    "jpg"|"jpeg")
       echo "It's image with jpeg extension."
    ;;
    "png")
    echo "It's image with png extension."
;;
    "gif")
       echo "Oh, it's a giphy!"
    ;;
    *)
       echo "Woops! It's not image!"
    ;;
esac
```

这里使用“|”把“jpg”和“jpeg”这两个模式连接到了一起。

1.2.4 Linux 下安装 Python

首先检查 Python 3 是否已经正确安装，以及所使用的版本号：

```
#python3 -V
Python 3.4.5
```

检查 Python 3 所在的路径：

```
#which python3
```

```
/usr/bin/python3
```

如果使用 CentOS，可以使用 yum 安装 Python 3。首先查找可供安装的 Python 版本：

```
#yum search python3
```

然后安装想要的版本：

```
#yum install python36
```

如果使用 Ubuntu 操作系统，首先运行以下命令更新软件包列表并将所有系统软件升级到可用的最新版本：

```
#sudo apt-get update && sudo apt-get -y upgrade
```

然后安装 pip 包管理系统：

```
#sudo apt-get install python3-pip
```

1.2.5　选择 Python 版本

Linux 系统中有可能同时存在多个可用的 Python 版本，每个 Python 版本都对应一个可执行二进制文件。可以使用 ls 命令来查看系统中有哪些 Python 的二进制文件可供使用。

```
$ ls /usr/bin/python*
```

python 命令执行 Python 2。可以使用 python3 命令执行 Python 3。如何使用 python 命令执行 Python 3？

一种简单安全的方法是使用别名。将如下命令放入~/.bashrc 或~/.bash_aliases 文件中：

```
alias python=python3
```

最好在终端中使用'python3'命令，在 Python 3.x 文件中使用 shebang 行'#!/usr/bin/env python3'。

1.2.6　使用 AWK

典型的 AWK 程序充当过滤器。它从标准输入读取数据，并输出标准输出的过滤数据。它一次读取数据的一个记录。默认情况下，一次读取一行文本。每次读取记录时，AWK 自动将记录分隔到字段中。字段在默认情况下也是由空格分隔的。每个字段被分配给一个变量，该变量有一个数字名称。变量$ 1 是第一个字段，$ 2 是第二个字段，以此类推。$ 0 表示整个记录。此外，还设置了一个名为 NF 的变量，其中包含在记录中检测到的字段的数目。

来试试一个很简单的例子。过滤 ls 命令的输出：

```
#ls -l ./ | awk '{print $0}'
```

显示文本文件 nohup.out 匹配（含有）字符串“sun”的所有行：

```
#awk '/sun/{print}' nohup.out
```

由于显示整个记录（全行）是 awk 的默认动作，因此可以省略 action 项。

例如，得到 Python 的版本号：

```
#python 2>&1 --version | awk '{print $2}'
2.7.5
```

这里的 2>&1 意思是：把标准错误重定向到标准输出。

1.2.7 Windows 下安装 Python

在图形化用户界面出现之前，人们就是用命令行来操作计算机的。Windows 命令行是通过 Windows 系统目录下的 cmd.exe 执行的。执行这个程序最直接的方式是找到这个程序，然后双击。但 cmd.exe 并没有一个桌面的快捷方式，所以这样太麻烦。

可以在开始菜单的运行窗口直接输入程序名，回车后运行这个程序。打开“开始”→“运行”，这样就会打开资源管理器中的运行程序窗口。或者使用快捷键——窗口键+R，打开运行程序窗口。总之，输入要运行的程序名 cmd 后单击“确定”按钮，出现命令提示窗口。因为能够通过这个黑屏的窗口直接输入命令来控制计算机，所以也称为控制台窗口。

开始的路径往往是 C:\Users\Administrator。就像公园的地图上往往会标出游客的当前位置。Windows 命令行也有个当前目录的概念。这个 C:\Users\Administrator 就是当前路径。

可以用 cd 命令改变当前路径，例如改变到 C:\Python\Python27 路径。

```
C:\Users\Administrator>cd C:\Python\Python27
```

系统约定从指定的路径找可执行文件。这个路径通过 PATH 环境变量指定。环境变量是一个“变量名=变量值”的对应关系，每一个变量都有一个或者多个值与之对应。如果是多个值，则这些值之间用分号隔开。例如 PATH 环境变量可能对应这样的值：“C:\Windows\system32; C:\Windows”。表示 Windows 会从 C:\Windows\system32 和 C:\Windows 两个路径找可执行文件。

设置或者修改环境变量的具体操作步骤是：首先在 Windows 桌面右键“计算机”，在弹出的“属性”对话框中选择“高级”→“环境变量”，然后设置用户变量，或者系统变量，接着再设置环境变量 PATH 的值。

需要重新启动命令行才能让环境变量设置生效。为了检查环境变量是否设置正确，可以在命令行中显示指定环境变量的值。需要用到 echo 命令。echo 命令用来显示一段文字。

```
C:\Users\Administrator>echo Hello
```

执行上面的命令将在命令行输出：Hello。

如果要引用环境变量的值，可以用前后两个百分号把变量名包围起来：“%变量名%”。echo 命令用来显示一个环境变量中的值。

```
C:\Users\Administrator>echo %PATH%
```

假设把 Python 安装在 D:\Python\Python27 目录下，则可以在计算机属性中手工设置 PATH 环境变量，然后检查环境变量的值：

```
C:\Users\Administrator.PC-201509301458>echo %PATH%
C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;D:\apache-maven-3.5.2\bin;D:\Python\Python27
```

用如下命令检查 Python 是否正确安装，以及所使用的版本号：

```
>python -version
```

或者用 where 命令检查系统是否已经安装了 Python：

```
C:\Users\Administrator>where python
C:\Python27\python.exe
D:\cygwin64\bin\python
D:\Programs\Python\Python36\python.exe
```

如果没有安装 Python，则可以使用 Chocolatey（https://chocolatey.org）安装 Python 3：

```
>choco install python3
```

1.2.8　搭建 PyDev 集成开发环境

在 Windows 下可以使用 PyDev（https://github.com/fabioz/Pydev）开发 Python 应用。

首先检查 Windows 操作系统中是否已经安装了 JDK：

```
>where javac
C:\Program Files\Java\jdk1.8.0_181\bin\javac.exe
```

如果没有，则可以使用软件包管理器 Chocolatey（https://chocolatey.org/）安装 JDK：

```
>choco install jdk
```

从 Eclipse 官方网站下载 Eclipse。

解压 pydev 插件后，会发现有 features、plugins 两个文件夹，然后把 features 文件夹下的所有文件移到 D:\Program Files\eclipse\features 目录下，把 plugins 文件夹下的所有文件移到 D:\Program Files\eclipse\plugins 目录下。

注意：D:\Program Files\eclipse 为 Eclipse 安装目录。

重启 Eclipse，在 windows->preferences 出现 PyDev 配置项，表示 PyDev 插件安装成功。

创建新项目，项目属性改成 UTF-8 编码。创建 Python 模块：

```
'''
Created on 2019年3月6日

@author: Administrator
'''
print('hi')
```

单击工具栏的 Run 按钮运行。控制台视图显示输出结果：

```
hi
```

这里使用了 print 函数输出一个字符串到控制台。

Eclipse 默认是英文界面，如果习惯用中文界面，可以从 http://www.eclipse.org/babel/downloads.php 下载支持中文的语言包。

如果想要切换回英文开发环境，则可以使用命令行进入 Eclipse 主目录后输入：

```
eclipse -nl en
```

切换回简体中文：

```
eclipse -nl zh_CN
```

然后安装插件，配置路径。

插件通过 Eclipse 的软件安装机制安装。从“帮助”菜单中选择“安装新软件”。输入网址 http://pydev.org/updates，并选择 PyDev 执行 next 开始安装，安装完需要重启 Eclipse。

必须配置 PyDev 才能与 Eclipse 和 Python 设置一起正常工作。从 Eclipse 主菜单中选择 Window→Preferences。这将打开“首选项”对话框。选择 Pydev→Interpreter - Python：

“Quick Auto-Config”按钮可以按照用户希望的方式设置 Eclipse，但前提是选择了某个版本的 Python 3。如果找到早期版本的 Python，请确保使用“删除”按钮将其删除，然后手动选择 python.exe 文件所在的路径。

1.3 体验 TensorFlow 文本分类

以文本分类这个经典问题来体验 TensorFlow。

1.3.1 安装 TensorFlow

这里介绍在 Linux 操作系统下安装 TensorFlow。当前推荐使用 Ubuntu 发行版本。

当用户最终让自己的 Linux 操作系统正常运行以后，请打开一个终端并安装一些必要的软件。

git：分布式版本控制系统。

wget：使用 HTTP、HTTPS 和 FTP 协议进行数据传输。

必须安装的软件包括：

awk：编程语言，用于搜索和处理文件和数据流中的模式。

bash：UNIX Shell 和脚本编程语言。

grep：逐行处理文本并打印与指定模式匹配的任何行。

make：从源代码自动构建可执行程序和库。

bazel：从源代码自动构建 TensorFlow 可执行程序和库。

perl：动态编程语言，非常适合文本文件处理。

例如，安装 git 可以用如下的命令。

```
#apt-get install git
```

要仅为在 CPU 上使用而安装当前版本：

```
#pip install tensorflow
```

如果要使用支持 CUDA 的 GPU 卡，则安装 TensorFlow 的 GPU 版本：

```
#pip install tensorflow-gpu
```

在交互式环境测试 TensorFlow：

```
>>> import tensorflow as tf
>>> tf.enable_eager_execution()
>>> tf.add(1, 2).numpy()
3
>>> hello = tf.constant('Hello, TensorFlow!')
>>> hello.numpy()
'Hello, TensorFlow!'
```

使用函数 tf.nn.softmax()测试 Tensorflow。例如，有一个 4 维的向量。-1 是最低值，3 是最高值。这些值都归一化为 0～1 的数值。

```
import tensorflow as tf
x = [0., -1., 2., 3.]
softmax_x = tf.nn.softmax(x)
session = tf.Session()
print('x:',x)
print('softmax_x:',session.run(softmax_x))
```

softmax 是逻辑函数的推广，softmax 将任意实数值的 K 维向量“压缩”到[0,1]区间的实数值的 K 维向量，向量中的元素值加起来为 1。

如果 TensorFlow 依赖的 numpy 出错，则可以考虑先卸载 numpy，然后重新安装 Tensorflow。

```
>pip uninstall numpy
>python -m pip install tensorflow
```

可以使用交互式会话测试 Tensorflow：

```
>>> import tensorflow as tf
>>> x = tf.placeholder("float", [2])
>>> a = tf.placeholder(tf.float32, name='a')
>>> b = tf.placeholder(tf.float32, name='b')
>>> c = tf.add(a, b, name='c')
>>> sess = tf.InteractiveSession()
>>> sess.run(c, feed_dict={a: 2.1, b: 1.9})
4.0
```

1.3.2 实现文本分类

在训练文本分类模型之前，必须先准备数据。可以创建一个简单的 JSON 文件来保存训练所需的数据。

```
    {
    "time" : ["what time is it?", "how long has it been since we started?", "that's a long
time ago", " I spoke to you last week", " I saw you yesterday"],
    "sorry" : ["I'm extremely sorry", "did he apologize to you?", "I shouldn't have been
rude"],
    "greeting": ["Hello there!", "Hey man! How are you?", "hi"],
    "farewell": ["It was a pleasure meeting you", "Good Bye.", "see you soon", "I gotta
go now."],
    "age": ["what's your age?", "How old are you?", "I'm a couple of years older than her",
"You look aged!"]
    }
```

以下是示例 JSON 训练的数据文件，其中包含 5 个类别。

```
    {
    "time": ["what time is it?", "how long has it been since we started?", "that's a long
time ago", " I spoke to you last week", " I saw you yesterday"],
    "sorry": ["I'm extremely sorry", "did he apologize to you?", "I shouldn't have been
rude"],
    "greeting": ["Hello there!", "Hey man! How are you?", "hi"],
    "farewell": ["It was a pleasure meeting you", "Good Bye.", "see you soon", "I gotta
go now."],
    "age": ["what's your age?", "How old are you?", "I'm a couple of years older than her",
"You look aged!"]
    }
```

数据加载和预处理：

```
#用于保存标点符号的表结构
tbl = dict.fromkeys(i for i in range(sys.maxunicode)
              if unicodedata.category(chr(i)).startswith('P'))
#从句子中删除标点符号的方法
def remove_punctuation(text):
    return text.translate(tbl)
#初始化词干分析器
stemmer = LancasterStemmer()
#用于保存从文件中读取的 JSON 数据的变量
data = None
#读取 JSON 文件并加载训练数据
with open('data.json') as json_data:
    data = json.load(json_data)
    print(data)
#获取要训练的所有类别的列表
categories = list(data.keys())
words = []
#包含句子中单词和类别名称的元组列表
docs = []
for each_category in data.keys():
    for each_sentence in data[each_category]:
        #从句子中删除标点符号
        each_sentence = remove_punctuation(each_sentence)
        print(each_sentence)
        #从每个句子中提取单词并附加到单词列表中
        w = nltk.word_tokenize(each_sentence)
        print("tokenized words: ", w)
        words.extend(w)
        docs.append((w, each_category))
#词干化并小写化每个单词，然后删除重复项
words = [stemmer.stem(w.lower()) for w in words]
words = sorted(list(set(words)))
print(words)
print(docs)
```

构建一个简单的深度神经网络，并用它来训练模型。

```
#重置底层图数据
tf.reset_default_graph()
#构建神经网络
net = tflearn.input_data(shape=[None, len(train_x[0])])
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, len(train_y[0]), activation='softmax')
```

```
net = tflearn.regression(net)
#定义模型并设置 tensorboard
model = tflearn.DNN(net, tensorboard_dir='tflearn_logs')
#开始训练（应用梯度下降算法）
model.fit(train_x, train_y, n_epoch=1000, batch_size=8, show_metric=True)
model.save('model.tflearn')
```

使用下面的代码测试神经网络文本分类 Python 模型。

```
#用几句话测试一下模型：
#前两个句子在训练集中出现过，最后两个句子没有在训练集中出现过
sent_1 = "what time is it?"
sent_2 = "I gotta go now"
sent_3 = "do you know the time now?"
sent_4 = "you must be a couple of years older then her!"
#一个接收句子的方法，并以可以馈送到 tensorflow 的形式返回数据
def get_tf_record(sentence):
    global words
    #切分出句子中的词
    sentence_words = nltk.word_tokenize(sentence)
    #返回每个词的词干
    sentence_words = [stemmer.stem(word.lower()) for word in sentence_words]
    #词袋
    bow = [0]*len(words)
    for s in sentence_words:
        for i, w in enumerate(words):
            if w == s:
                bow[i] = 1
    return(np.array(bow))
#开始预测 4 个句子中每个句子的分类结果
print(categories[np.argmax(model.predict([get_tf_record(sent_1)]))])
print(categories[np.argmax(model.predict([get_tf_record(sent_2)]))])
print(categories[np.argmax(model.predict([get_tf_record(sent_3)]))])
print(categories[np.argmax(model.predict([get_tf_record(sent_4)]))])
```

1.4 本章小结

由 Google 大脑团队开发，供 Google 内部使用的一个深度学习框架于 2015 年命名为 TensorFlow。TensorFlow 可运行于 Linux 或者 Windows、MacOS 等操作系统。