第八章 构建长短时记忆网络模型进行中文新闻内容的分类

8.1 场景描述

最近，小卢公司接到某新闻网站的要求，该新闻网站每天都会收到数万条新闻投稿，但是，不同类别的新闻需要发送到各自类别的小组去进行处理，面对每天上万条的新闻内容，需要很大的人力成本来完成这项工作，因此，该公司希望能通过人工智能来代替新闻分类这一工作。

8.2 任务描述

针对 10 中新闻类别，建立一个长短时记忆学习模型，模型组成：一个长短时记忆神经网络层，一个隐藏层和一个输出层，首先对已有分类标签的新闻内容进行数据预处理，数据预处理包括中文分词、中文字典创建、词向量生成，然后将处理好的数据送入长短时记忆学习模型进行参数训练，训练好的模型可对测试数据进行分类预测。

8.3 任务分解

按照任务要求，我们卷积神经网络识别多目标的过程描述如图 8-1 所示。

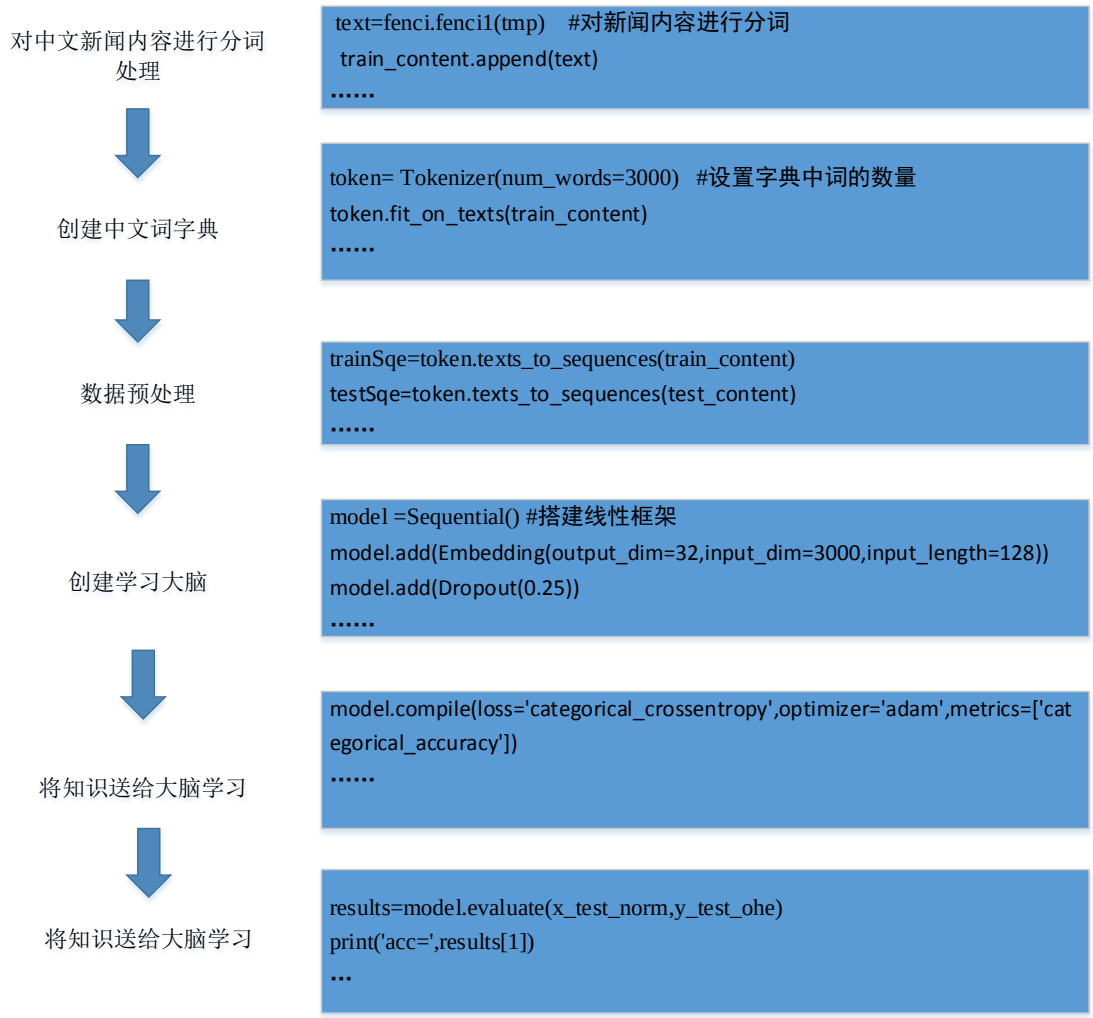


图 8-1 手写数字识别大脑的优化流程图

8.4 知识储备

8.4.1 自然语言处理

- 自然语言处理（NLP）简介

NLP 是数据科学里的一个分支，它的主要覆盖的内容是：以一种智能与高效的方式，对文本数据进行系统化分析、理解与信息提取的过程。通过使用 NLP 以及它的组件，我们可以管理非常大块的文本数据，或者执行大量的自动化任务，并且解决各式各样的问题，如自动摘要，机器翻译，命名实体识别，关系提取，情感分析，语音识别，以及主题分割等等。

- 文本预处理

因为文本数据在可用的数据中是非常无结构的，它内部会包含很多不同类型的噪点。所以在对文本进行预处理之前，它暂时是不适合被用于做直接分析的。

文本预处理过程主要是对 文本数据进行清洗与标准化。这个过程会让我们的数据没有噪声，并可以对它直接做分析。

数据预处理的过程主要包括以下三个部分：

（1）噪声移除

任何与数据上下文无关的文本片段以及 end-output 均可被认为是噪音。例如，语言停顿词（一般是在语言里常用的单词，如：is, am, the, of, in 等等），URL 或链接，社交媒体里的实体（如@符号，#标签等），标点符号，以及工业特有词汇等。这个步骤就是为了移除文本里所有类型的噪音实体。

在噪音移除里，一个常见的方法是：准备一个噪音实体的字典，然后对 text object 进行迭代（以 token 或单词），去除掉那些存在于噪音字典里的 tokens（单词或实体）。

（2）词汇规范化

另外一种文本型的噪音与一个词语的多种表达形式有关。例如，“play”，“player”，“played”，“plays”和“playing”都是单词“play”的变种。尽管它们有不同的意思，但是根据上下文来看，它们是意思是相似的。这个步骤是将一个单词的所有不同形式转换为它的规范形式（也被称为词条（lemma））。规范化在特征工程里，是对文本处理的一个关键步骤。因为它将高维的特征（N 个不同的特征）转换到了低维空间（1 个特征），这对于机器学习模型来说是非常完美的。

最常见的词汇规范化的实践有：

1）词干提取（Stemming）：词干提取是一个初级的、基于规则的脱去后缀（如“ing”，“ly”，“es”，“s”等等）的过程

2）词元化（Lemmatization）：另一方面，词元化，是一个组织好的、一步一步的获取词根的过程。并使用了词汇表（单词在字典里的重要性）和形态学分析（单词结构与语法关系）

（3）对象标准化

文本数据经常包含一些不存在于标准词汇字典里的单词或短语。这些部分是无法被搜索引擎和模型所识别的。

一些例子如：首字母缩略词，井字标签与它后面的词汇，以及口语俚语等。对此我们可以使用正则表达式和人工准备的数据字典，来修正这些噪音。

- 将文本转化为特征（在文本数据上使用特征工程）

为了分析一个已经做了预处理的数据，我们需要将它转化为特征。根据使用用途不同，文本特征可以根据各种技术建立而成。如：句法分析（Syntactical Parsing），实体（entities） / N 元语法（N-grams） / 基于单词（word-based）特征，统计学（Statistical）特征，以及词向量（word embeddings）。接下来我们会对进行介绍。

（1）词向量（Word Embedding）（text vectors）

Word embedding 是一种非常现代的用向量表示单词的方式。它的目标是为了将高维的词特征重新定义为低维的特征向量，主要通过保留语料库里的上下文相关性完成。它已经被广泛应用于如卷积神经网络，循环神经网络等深度学习模型中

Word2Vec 和 GloVe 是两个非常流行的为文本创建词向量的模型。这些模型使用文本语料库作为输入，并生成词向量作为输出。

Word2Vec 模型由预处理模块、被称为连续词袋（Continuous Bag of Words）的一个浅神经网络模块以及另一个名叫 skip-gram 的浅神经网络模型组成。这些模型已经被广泛的用于其他各种 NLP 问题。它首先从训练语料库建立一个词汇表，然后学习词向量的表现方式。

- NLP 的重要任务

（1）文本分类

文本分类经典的 NLP 问题之一。众所周知的例子包括：垃圾邮件识别，新闻主题分类，情感分析，以及搜索引擎的页面组织。

文本分类，简单来说，它就是一种将文本对象（文档或句子）分类到一个固定的类别的技术。当数据量非常大时，它在数据的组织、信息过滤，以及数据存储等方面起到非常大的作用。

一个典型的自然语言分类器包含两部分：1. 训练；2. 预测。如下图 8-2 所示：

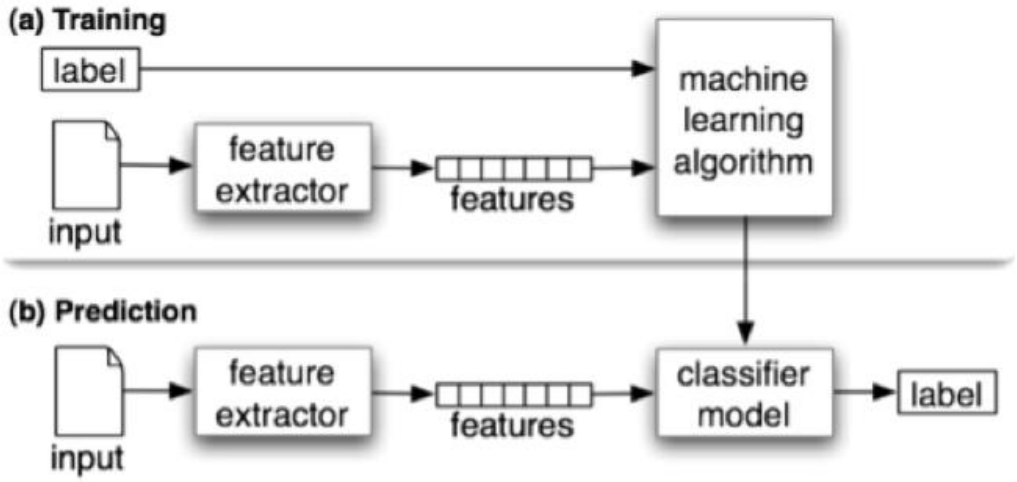


图 8-2 典型的自然语言分类器

首先，文本在输入后，它的特征会被创建。然后机器学习算法从这些特征学习一组参数。之后使用学习到的机器学习模型对新文本做预测。

文本分类很大程度上依赖于特征的质量与数量。当然，在使用任何机器学习训练模型时，一般来说，引入越多的训练数据总会是一个比较好的事。

（2）文本匹配 / 相似度（Text Matching / Similarity）

在 NLP 中，一个很重要的领域是通过匹配文本对象找到相似体。它的主要应用有：自动拼写修正，重复数据删除，以及基因组分析，等等。

根据需求，我们有若干个文本匹配技术可供选择。这个部分会详细的描述比较重要的技术：

1） 莱文斯坦距离（Levenshtein Distance）- 两个字符串之间的莱文斯坦距离可以被定义为：将一个字符串转换为另一个字符串时，所需的最小编辑次数。可允许的编辑操作有插入，删除，或者替换一个单字符。

2）语音匹配（Phonetic Matching）- 语音匹配的算法以一个关键词作为输入（如人名，地名等），然后产生出一个字符串，这个字符串与一组语音上（大致）相似的单词有关。这个技术在搜索超大文本语料库、修正拼写错误以及匹配相关名字时非常有帮助。Soundex 和 Metaphone 是其中两个组主要的语音算法。Python 里的 Fuzzy 模块可以用来为不同的单词计算 soundex 字符串，如：

3）灵活的字符串匹配（Flexible String Matching）- 一个完整的文本匹配系统里包括多种不同的算法，它们通过管道的方式组合起来，计算文本变化的种类（compute variety of text variations）。正则表达式对于这个任务也非常有用。其他

常见的技术包括：精准字符串匹配，lemmatized matching，以及紧凑匹配（处理空格，标点，俚语等）

4）余弦相似度 - 当文本以向量表示时，一个余弦相似度也可以用于衡量向量相似度。下面的代码将文本转化为向量（使用词频的方式），并且使用了余弦相似度来计算两个文本之间的相似度

（3）指代消解（Coreference Resolution）

指代消解是一个在句子里寻找单词（或短语）之间关系连接的过程。考虑这个句子“Donald went to John’s office to see the new table. He looked at it for an hour.”

人们可以很快的指出“he”指代的是 Donald（而不是 John），并且“it”指代的是 table（而不是 John’s office）。指代消解是 NLP 的一个组成部分，它会自动的完成这个工作。这个技术常被用于文件摘要，问答系统，以及信息提取。Stanford CoreNLP

（4）其他 NLP 应用

文本摘要：给出一个文本文章或段落，自动对它做总结，并根据重要性、相关性的程度，按次序输出句子（依次输出最重要并最相关的句子）。

机器翻译：通过处理语法、语义学以及真实世界的信息，自动将一个文本的语言翻译为另外一个语言的文本。

自然语言的生成与理解：将计算机数据库里的信息或语义意图转化为人类可读的语言叫做自然语言生成。为了更方便计算机程序处理，而将文本块转换为更逻辑化的结构的操作叫做自然语言理解

视觉字符识别：给出一打印后的文本图，识别与之对应的文本

文档信息化：对文档（网站，文件，pdf 和图片）里文本数据的进行语法分析，将它们处理为干净、可分析的格式

- NLP 相关的重要库

  scikit-learn：python 里的机器学习库

  Natural Language Toolkit（NLTK）：包含所有 NLP 技术的完整工具

  Pattern：一个 web mining 模块，用于 NLP 和机器学习

  TextBlob：操作简单的 nlp 工具 API，构建于 NLTK 和 Pattern

spaCy：Industrial strength NLP with Python and Cython

Gensim：主题建模

Stanford Core NLP：Stanford NLP group 提供的 NLP 服务包

8.4.2 中文分词

中文分词(Chinese Word Segmentation) 指的是将一个汉字序列切分成一个一个单独的词。分词就是将连续的字序列按照一定的规范重新组合成词序列的过程。现有的分词方法可分为三大类：基于字符串匹配的分词方法、基于理解的分词方法和基于统计的分词方法。

- 基于字符串匹配的分词方法

基于字符串匹配的分词方法又称机械分词方法，它是按照一定的策略将待分析的汉字串与一个“充分大的”机器词典中的词条进行配，若在词典中找到某个字符串，则匹配成功（识别出一个词）。

按照扫描方向的不同，字符串匹配分词方法可以分为正向匹配和逆向匹配；按照不同长度优先匹配的情况，可以分为最大（最长）匹配和最小（最短）匹配；按照是否与词性标注过程相结合，可以分为单纯分词方法和分词与词性标注相结合的一体化方法。常用的字符串匹配方法有如下几种：

（1）正向最大匹配法（从左到右的方向）；

（2）逆向最大匹配法（从右到左的方向）；

（3）最小切分（每一句中切出的词数最小）；

（4）双向最大匹配（进行从左到右、从右到左两次扫描）

这类算法的优点是速度快，时间复杂度可以保持在 O（n）,实现简单，效果尚可；但对歧义和未登录词处理效果不佳。

- 基于理解的分词方法

基于理解的分词方法是通过让计算机模拟人对句子的理解，达到识别词的效果。其基本思想就是在分词的同时进行句法、语义分析，利用句法信息和语义信息来处理歧义现象。它通常包括三个部分：分词子系统、句法语义子系统、总控部分。在总控部分的协调下，分词子系统可以获得有关词、句子等的句法和语义信息来对分词歧义进行判断，即它模拟了人对句子的理解过程。这种分词方法需要使用大量的语言知识和信息。由于汉语语言知识的笼统、复杂性，难以将各种

语言信息组织成机器可直接读取的形式，因此目前基于理解的分词系统还处在试验阶段。

- 基于统计的分词方法

基于统计的分词方法是在给定大量已经分词的文本的前提下，利用统计机器学习模型学习词语切分的规律（称为训练），从而实现对未知文本的切分。例如最大概率分词方法和最大熵分词方法等。随着大规模语料库的建立，统计机器学习方法的研究和发展，基于统计的中文分词方法渐渐成为了主流方法

主要的统计模型有：N 元文法模型（N-gram），隐马尔可夫模型（Hidden Markov Model ，HMM），最大熵模型（ME），条件随机场模型（Conditional Random Fields，CRF）等。

在实际的应用中，基于统计的分词系统都需要使用分词词典来进行字符串匹配分词，同时使用统计方法识别一些新词，即将字符串频率统计和字符串匹配结合起来，既发挥匹配分词切分速度快、效率高的特点，又利用了无词典分词结合上下文识别生词、自动消除歧义的优点。

8.4.3 中文分词工具 jieba

jieba 分词是国内使用人数最多的中文分词工具。jieba 分词支持三种模式：

（1）精确模式：试图将句子最精确地切开，适合文本分析；

（2）全模式：把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义；

（3）搜索引擎模式：在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

jieba 分词过程中主要涉及如下几种算法：

（1）基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图 (DAG)；

（2）采用了动态规划查找最大概率路径，找出基于词频的最大切分组合；

（3）对于未登录词，采用了基于汉字成词能力的 HMM 模型，采用 Viterbi 算法进行计算；

（4）基于 Viterbi 算法做词性标注；

（5）基于 tf-idf 和 textrank 模型抽取关键词；

测试代码如下所示：

```
# -*- coding: utf-8 -*-
"""
jieba分词测试
"""
import jieba
#全模式
test1 = jieba.cut("杭州西湖风景很好,是旅游胜地!", cut_all=True)
print("全模式: " + "| ".join(test1))
#精确模式
test2 = jieba.cut("杭州西湖风景很好,是旅游胜地!", cut_all=False)
print("精确模式: " + "| ".join(test2))
#搜索引擎模式
test3= jieba.cut_for_search("杭州西湖风景很好,是旅游胜地,每年吸引大量前来游玩的游客!")
print("搜索引擎模式:" + "| ".join(test3))
```

图 8-3 jieba 分词代码显示

测试结果如下图 8-4 所示：

```
全模式: 杭州| 西湖| 风景| 很| 好| | | 是| 旅游| 旅游胜地| 胜地| |
精确模式: 杭州| 西湖| 风景| 很| 好| ,| 是| 旅游胜地| !
搜索引擎模式:杭州| 西湖| 风景| 很| 好| ,| 是| 旅游| 胜地| 旅游胜地| ,| 每年| 吸引| 大量| 前来| 游玩| 的| 游客| !
```

图 8-4 jieba 分词显示的结果

在本章中，创建了一个 fenci1 函数，用来直接返回一个用空格隔开的分词后的字符串。

代码如下图 8-5 所示：

```
#fenci.py
import jieba
def fenci1(s):
        cut=jieba.cut(s)
        text=' '.join(cut)
        return text
```

图 8-5　fenci1 函数代码

在运行本章代码时，请先创建本文件。

8.4.3 循环神经网络和长短时记忆网络

- 循环神经网络（RNN）

人类并不是每时每刻都从一片空白的大脑开始他们的思考。在你阅读这篇文章时候，你都是基于自己已经拥有的对先前所见词的理解来推断当前词的真实含义。我们不会将所有的东西都全部丢弃，然后用空白的大脑进行思考。我们的思想拥有持久性。

传统的神经网络并不能做到这点，看起来也像是一种巨大的弊端。例如，假设你希望对电影中的每个时间点的时间类型进行分类。传统的神经网络应该很难

来处理这个问题——使用电影中先前的事件推断后续的事件。RNN 解决了这个问题。RNN 是包含循环的网络，允许信息的持久化。

RNN（Recurrent Neural Network）是一类用于处理序列数据的神经网络。首先我们要明确什么是序列数据，摘取百度百科词条：时间序列数据是指在不同时间点上收集到的数据，这类数据反映了某一事物、现象等随时间的变化状态或程度。这是时间序列数据的定义，当然这里也可以不是时间，比如文字序列，但总归序列数据有一个特点——后面的数据跟前面的数据有关系。

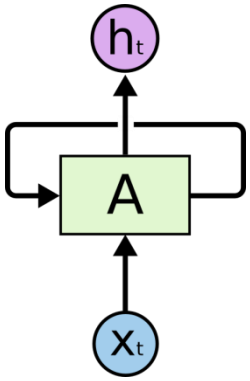


图 8-6 RNN 循环神经网络

在上面的示例图中，神经网络的模块 A，正在读取某个输入x_i，并输出一个值 h_i。循环可以使得信息可以从当前步传递到下一步。RNN 可以被看做是同一神经网络的多次赋值，每个神经网络模块会把消息传递给下一个。所以，如果我们将这个循环展开：

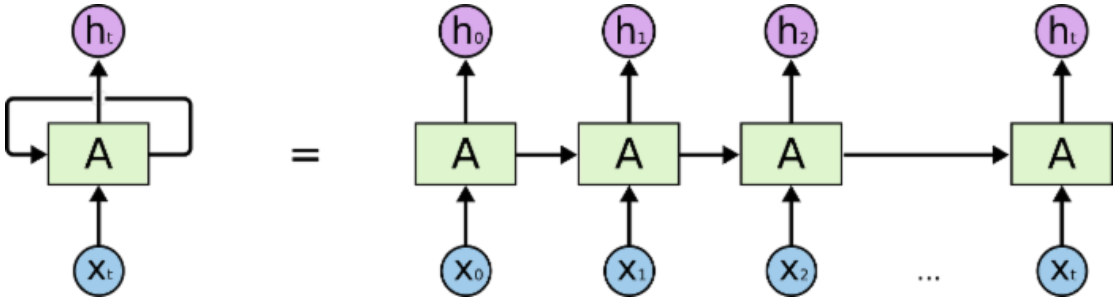


图 8-7　RNN 循环神经网络

图 8-揭示了 RNN 本质上是与序列和列表相关的。他们是对于这类数据的最自然的神经网络架构。

并且 RNN 也已经被人们应用了。在过去几年中，应用 RNN 在语音识别，语言建模，翻译，图片描述等问题上已经取得一定成功，并且这个列表还在增长。

RNN 的关键点之一就是他们可以用来连接先前的信息到当前的任务上，例如使用过去的视频段来推测对当前段的理解。

有时候，我们仅仅需要知道先前的信息来执行当前的任务。例如，我们有一个语言模型用来基于先前的词来预测下一个词。如果我们试着预测 “the clouds are in the sky” 最后的词，我们并不需要任何其他的上下文 —— 因此下一个词很显然就应该是 sky。在这样的场景中，相关的信息和预测的词位置之间的间隔是非常小的，RNN 可以学会使用先前的信息。

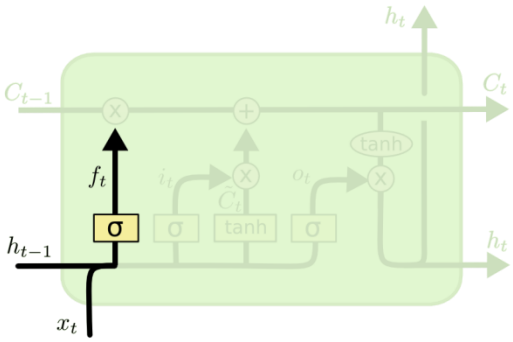
但是同样会有一些更加复杂的场景。假设我们试着去预测“我出生在广州...我能说流利的本地话”最后的词。当前的信息建议下一个词可能是一种语言的名字，但是如果我们需要弄清楚是什么语言，我们是需要先前提到的离当前位置很远的“广州”的上下文的。这说明相关信息和当前预测位置之间的间隔就肯定变得相当的大。不幸的是，在这个间隔不断增大时，RNN 会丧失学习到连接如此远的信息的能力。

- 长短时记忆网络（LSTM）

长短期记忆网络是 RNN 的一种变体，RNN 由于梯度消失的原因只能有短期记忆，LSTM 网络通过精妙的门控制将短期记忆与长期记忆结合起来，并且一定程度上解决了梯度消失的问题。

Long Short Term（LSTM）网络，是一种 RNN 特殊的类型，可以学习长期依赖信息。在很多问题，LSTM 都取得相当巨大的成功，并得到了广泛的使用。

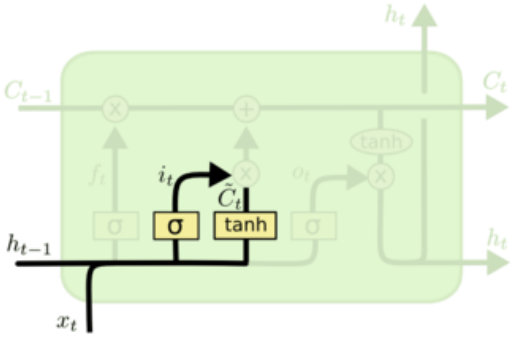
LSTM 通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是 LSTM 的默认行为。LSTM 将 RNN 中隐藏层中加入了一个对信息处理的单元——这个单元除了 RNN 原先简单的 tanh 函数以外还有几个 sigmoid 函数共同作用来完成 LSTM 的功能。对于 RNN 来说，每个隐藏层单元之间的状态是连续的，LSTM 也不例外；而为了让这些插入的 sigmoid 函数作用于状态，将生成的权值与状态之间进行 pointwise 乘法运算。第一个 sigmoid 函数的作用是遗忘，用来舍弃部分不需要的信息（即通过h_{t-1}与一个 0 到 1 的数值给隐藏层单元状态f_t，0 是完全舍弃，1 是完全保留）。



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

图 8-8　LSTM 循环神经网络遗忘门

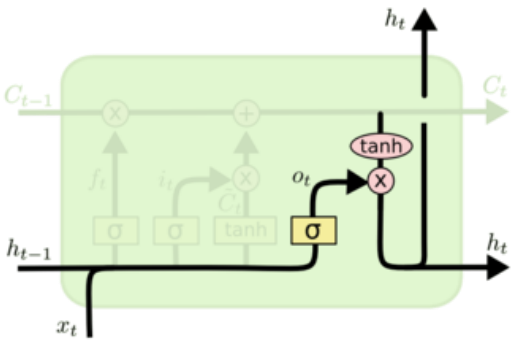
接着是更新门，这里由两部分组成，一个是 *sigmoid* 函数用来决定什么致我们要更新，另一个则是 *tanh* 函数创建一个新的候选值向量C_t。



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

图 8-9　LSTM 循环神经网络更新门

这时我们进行更新，我们将两者进行乘法后，再将从遗忘门出来的状态与我们新的候选值向量相加来完成对C_{t-1}更新至C_t这一过程；最后的时候，我们需要确定输出什么值，这个相当于我们对这个隐藏层单元的状态做一个处理之后进行复制一份传给下一个隐藏层单元，具体操作就是将状态通过 tanh 进行处理，并将处理后的数据和先前h_{t-1}与x_t两项进行 sigmoid 处理所生成的o_t相乘以获得我们想要输出的那部分，即h_t并输出给下一个隐藏层单元。



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

图 8-10　LSTM 循环神经网络传递下一个隐藏单元

这样就算是一个 *LSTM* 中一个隐藏层单元所需要进行的操作了。当然宏观上来讲，其实说白了就是单元状态与上一个输出的同时传导，有选择的（避免梯度爆炸）将我们需要传递的信息传达到后面的隐藏层单元（避免梯度消失），可以说这样算是解决了 *RNN* 上两个令人头疼的问题了。

8.5 任务实现步骤

按照任务流程，我们可以将该任务分解成如下几个子任务，依次完成：

第一步：对中文新闻内容进行分词处理，从本地读取训练和测试新闻内容，调用 jieba 模块，实现分词处理，并将新闻内容和标签分离出来。

第二步：创建中文词字典，从分词的新闻内容中，获得中文词字典。

第三步：数据预处理，对输入新闻内容进行词向量化处理，输出标签进行 one-hot-encoding 处理。

第四步：创建学习大脑，完成基于 RNN 的训练模型创建，填充模型内容。

第五步：将知识送给大脑学习，设置模型的训练参数，启动模型进行训练，并动态查看模型的训练状态。

第六步：使用训练好的“大脑”模型，对中文新闻的测试数据进行分类。

8.5.1 创建 jupyter notebook 项目

（1）打开 jupyter notebook，如图 8-11 所示。

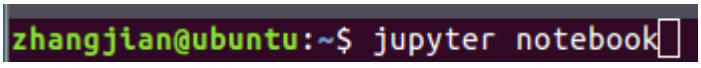


图 8-11　启动 jupyter notebook

（2）在 python3 下新建一个 notebook 项目，命名为 rask8-1，如图 8-12 所示。

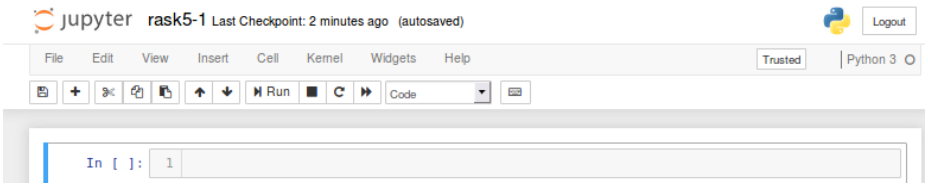


图 8-12　新建 notebook 项目示意图

8.5.2 对中文新闻内容进行分词处理

（1）在 jupyter notebook 中输入图 8-13 中显示的代码，并确认代码无错误。

```
from keras.utils import np_utils   #导入keras中的utils工具模块命名为np_utils
import fenci    #导入fenci模块
print('**************start to fenci***********') #开始处理
#新闻数据的10个分类
newstype={"体育":0, "娱乐":1, "家居":2,"房产":3,"教育":4,
          "时尚":5,"时政":6, "游戏":7,"科技":8, "财经":9}
fp=open('cnews.train.txt')   #读取训练新闻数据
train_label=[]
train_content=[]
test_label=[]
test_content=[]
n=0
while True:
    cur_text=fp.readline() #循环读取一条新闻数据
    if cur_text=="":    #如果读取完所有新闻,则结束循环
        break
    item=cur_text.split()    #通过空格分割新闻类别和内容
    train_label.append(newstype[item[0]])   #将新闻类别保存到train_label数组
    tmp=''.join(item[1:]) #将字符串数组item除了第一项后面所有的新闻内容拼接成一个字符串
    text=fenci.fencil(tmp)     #对新闻内容进行分词,分词后的结果返回text字符串中
    train_content.append(text)   #将分词后的新闻内容保存到train_content数组
    n=n+1
```

```
print("训练数据一共有"+str(n)+"条新闻")

n=0
file_tl=open('train_label.txt','w')  #创建train_label.txt文件
file_tl.write(str(train_label))   #将train_label数组中的内容写入train_label.txt文件
file_tl.close() #关闭文件
file_tc=open('train_content.txt','w')  #创建train_content.txt文件
file_tc.write(str(train_content));   #将train_content数组中的内容写入train_content.txt文件
file_tc.close() #关闭文件

fp=open('cnews.test.txt')#读取测试新闻数据
while True:
    cur_text=fp.readline() #循环读取一条新闻数据
    if cur_text=="":  #如果读取完所有新闻,则结束循环
        break
    item=cur_text.split()  #通过空格分割新闻类别和内容
    test_label.append(newstype[item[0]]) #将新闻类别保存到test_label数组
    tmp=''.join(item[1:]) #将字符串数组item除了第一项后面所有的新闻内容拼接成一个字符串
    text=fenci.fencil(tmp)  #对新闻内容进行分词,分词后的结果返回text字符串中
    test_content.append(text) #将分词后的新闻内容保存到train_content数组
    n=n+1
file_testc=open('test_content.txt','w')    #创建test_label.txt文件
file_testc.write(str(test_content));   #将test_label数组中的内容写入test_label.txt文件
file testc.close()  #关闭文件
```

```
file_testl=open('test_label.txt','w')    #创建test_content.txt文件
file_testl.write(str(test_label));  #将test_content数组中的内容写入test_content.txt文件
file_testl.close()  #关闭文件
print("测试数据一共有"+str(n)+"条新闻")

print(train_label[0])  #打印训练数据中第一条新闻的类别
print(train_content[0]) #打印训练数据中第一条新闻进行分词后的结果
print('**************success to finish fenci readNewsData***********')
```

图 8-13 对中文分词处理代码

（2）按 Ctrl+Enter 组合键执行代码，代码没有错误提示后按 Shift+Enter 组合键新建下一个单元格，结果显示如图 8-14 所示：

```
**************start to fenci***********
Dumping model to file cache /tmp/jieba.cache
Loading model cost 1.048 seconds.
Prefix dict has been built succesfully.
0
马晓旭 意外 受伤 让 国奥 警惕 无奈 大雨 格外 青睐 殷家 军 记者 傅亚雨 沈阳 报道 来到 沈阳 ， 国奥队 依然 没有 摆脱 雨水 的 困扰 。 7 月 31 日 下午 6 点 ， 国奥队 的 日常 训练 再度 受到 大雨 的 干扰 ， 无奈 之下 队员们 只 慢跑 了 25 分钟 就 草草收场 。 31 日 上午 10 点 ， 国奥队 在 奥体中心 外场 训练 的 时候 ， 天 就是 阴沉沉 的 ， 气象预报 显示 当天 下午 沈阳 就 有 大雨 ， 但 幸好 队伍 上午 的 训练 并 没有 受到 任何 干扰 。 下午 6 点 ， 当 球队 抵达 训练场 时 ， 大雨 已经 下 了 几个 小时 ， 而且 丝毫 没有 停下来 的 意思 。 抱 着 试一试 的 态度 ， 球队 开始 了 当天 下午 的 例行 训练 ， 25 分钟 过去 了 ， 天气 没有 任何 转好 的 迹象 ， 为了 保护 球员 们 ， 国奥队 决定 中止 当天 的 训练 ， 全队 立即 返回 酒店 。 在 雨 中 训练 对 足球队 来说 并 不是什么 稀罕 事 ， 但 在 奥运会 即将 开始 之前 ， 全队 变得 “ 娇贵 ” 了 。 在 沈阳 最后 一周 的 训练 ， 国奥队 首先 要 保证 现有 的 球员 不再 出现意外 的 伤病 情况 以免 影响 正式 比赛 ， 因此 这一 阶段 控制 训练 受伤 、 控制 感冒 等 疾病 的 出现 被 队伍 放在 了 相当 重要 的 位置 。 而 抵达 沈阳 之后 ， 中 后卫 冯萧霆 就 一直 没有 训练 ， 冯萧霆 是 7 月 27 日 在 长春 患上 了 感冒 ， 因此 也 没有 参加 29 日 跟 塞尔维亚 的 热身赛 。 队伍 介绍 说 ， 冯萧霆 并 没有 出现 发烧 症状 ， 但 为了 安全 起 见 ， 这 两天 还是 让 他 静养 休息 ， 等 感冒 彻底 好 了 之后 再 恢复 训练 。 由于 有 了 冯萧霆 这个 例子 ， 因此 国奥队 对雨中 训练 就 显得 特别 谨慎 ，主要 是 担心 球员 们 受凉 而 引发 感冒 ， 造成 非战斗 减员 。 而 女足 队员 马晓旭 在 热身赛 中 受伤 导致 无缘 奥运 的 前科 ， 也 让 在 沈阳 的 国奥队 现在 格外 警惕 ， “ 训练 中 不断 嘱咐 队员 们 要 注意 动作 ， 我们 可不能 再出 这样 的 事情 了 。 ” 一位 工作人员 表示 。 从 长春 到 沈阳 ， 雨水 一路 伴随 着 国奥队 ， “ 也 邪了 ， 我们 走 到 哪儿 雨 就 下 到 哪儿 ， 在 长春 几次 训练 都 被 大雨 给 搅和 了 ， 没想到 来 沈阳 又 碰到这种 事情 。 ” 一位 国奥 球员 也 对 雨水 的 “ 青睐 ” 有些 不解 。
**************success to finish fenci readNewsData***********
```

图 8-14 中文分词结果显示图

（3）下面对中文分词的代码进行解析，以便更加了解该图像集。

- from keras.utils import np_utils

  导入 keras 中的 utils 工具模块命名为 np_utils

- import fenci

  导入 fenci 模块

- print('**************start to fenci***********')
- newstype={"体育":0, "娱乐":1, "家居":2,"房产":3,"教育":4,
- "时尚":5,"时政":6, "游戏":7,"科技":8, "财经":9}

  已键值对的格式将 10 种类别的新闻进行定义。

表 8-1　新闻类别与对应的数值

新闻类别	体育	娱乐	家居	房产	教育	时尚	时政	游戏	科技	财经
值	0	1	2	3	4	5	6	7	8	9

- fp=open('cnews.train.txt')

读取训练新闻数据，本教材提供了用于训练的新闻数据，每种类别的新闻有5000 条，10 类新闻一共 50000 条数据，新闻数据的格式为[新闻类别]+[新闻内容]，新闻类别和新闻内容之间用空格隔开，每条新闻之间用回车键隔开。

- while True:

- cur_text=fp.readline()

循环读取每一条新闻数据，并将读取后的新闻数据保存在 cur_text 变量中。例如，读取第一条新闻后，显示的数据为：

体育　　马晓旭意外受伤让国奥警惕 无奈大雨格外青睐殷家军记者傅亚雨沈阳报道 来到沈阳，国奥队依然没有摆脱雨水的困扰。7月31日下午6点，国奥队的日常训练再度受到大雨的干扰，无奈之下队员们只慢跑了25分钟就草草收场。31日上午10点，国奥队在奥体中心外场训练的时候，天就是阴沉沉的，气象预报显示当天下午沈阳就有大雨，但幸好队伍上午的训练并没有受到任何干扰。下午6点，当球队抵达训练场时，大雨已经下了几个小时，而且丝毫没有停下来的意思。抱着试一试的态度，球队开始了当天下午的例行训练，25分钟过去了，天气没有任何转好的迹象，为了保护球员们，国奥队决定中止当天的训练，全队立即返回酒店。在雨中训练对足球队来说并不是什么稀罕事，但在奥运会即将开始之前，全队变得“娇贵”了。在沈阳最后一周的训练，国奥队首先要保证现有的球员不再出现意外的伤病情况以免影响正式比赛，因此这一阶段控制训练受伤、控制感冒等疾病的出现被队伍放在了相当重要的位置。而抵达沈阳之后，中后卫冯萧霆就一直没有训练，冯萧霆是7月27日在长春患上了感冒，因此也没有参加29日跟塞尔维亚的热身赛。队伍介绍说，冯萧霆并没有出现发烧症状，但为了安全起见，这两天还是让他静养休息，等感冒彻底好了之后再恢复训练。由于有了冯萧霆这个例子，因此国奥队对雨中训练就显得特别谨慎，主要是担心球员们受凉而引发感冒，造成非战斗减员。而女足队员马晓旭在热身赛中受伤导致无缘奥运的前科，也让在沈阳的国奥队现在格外警惕，“训练中不断嘱咐队员们要注意动作，我们可不能再出这样的事情了。”一位工作人员表示。从长春到沈阳，雨水一路伴随着国奥队，“也邪了，我们走到哪儿雨就下到哪儿，在长春几次训练都被大雨给搅和了，没想到来沈阳又碰到这种事情。”一位国奥球员也对雨水的“青睐”有些不解。

图 8-15 显示第一条新闻的原始格式和内容

- if cur_text=="":
- break

如果读取完所有新闻，则结束循环，当读取到的内容为空后，说明所有新闻的内容已经读取，则停止循环。

- item=cur_text.split()

通过空格分割新闻类别和内容，由于新闻类型和新闻内容之间是用空格分隔的，因此，我们使用 split()函数进行分隔，分隔后的结果保存在 item 变量中，我们查看对第一条新闻进行 split 后 item 变量的值，如下图 8-16 所示：

['体育', '马晓旭意外受伤让国奥警惕', '无奈大雨格外青睐殷家军记者傅亚雨沈阳报道', '来到沈阳，国奥队依然没有摆脱雨水的困扰。7月31日下午6点，国奥队的日常训练再度受到大雨的干扰，无奈之下队员们只慢跑了25分钟就草草收场。31日上午10点，国奥队在奥体中心外场训练的时候，天就是阴沉沉的，气象预报显示当天下午沈阳就有大雨，但幸好队伍上午的训练并没有受到任何干扰。下午6点，当球队抵达训练场时，大雨已经下了几个小时，而且丝毫没有停下来的意思。抱着试一试的态度，球队开始了当天下午的例行训练，25分钟过去了，天气没有任何转好的迹象，为了保护球员们，国奥队决定中止当天的训练，全队立即返回酒店。在雨中训练对足球队来说并不是什么稀罕事，但在奥运会即将开始之前，全队变得“娇贵”了。在沈阳最后一周的训练，国奥队首先要保证现有的球员不再出现意外的伤病情况以免影响正式比赛，因此这一阶段控制训练受伤、控制感冒等疾病的出现被队伍放在了相当重要的位置。而抵达沈阳之后，中后卫冯萧霆就一直没有训练，冯萧霆是7月27日在长春患上了感冒，因此也没有参加29日跟塞尔维亚的热身赛。队伍介绍说，冯萧霆并没有出现发烧症状，但为了安全起见，这两天还是让他静养休息，等感冒彻底好了之后再恢复训练。由于有了冯萧霆这个例子，因此国奥队对雨中训练就显得特别谨慎，主要是担心球员们受凉而引发感冒，造成非战斗减员。而女足队员马晓旭在热身赛中受伤导致无缘奥运的前科，也让在沈阳的国奥队现在格外警惕，“训练中不断嘱咐队员们要注意动作，我们可不能再出这样的事情了。”一位工作人员表示。从长春到沈阳，雨水一路伴随着国奥队，“也邪了，我们走到哪儿雨就下到哪儿，在长春几次训练都被大雨给搅和了，没想到来沈阳又碰到这种事情。”一位国奥球员也对雨水的“青睐”有些不解。']

图 8-16 第一条新闻内容进行分隔后的结果

由图中结果可以看到，第一条数据被保存在 item 数组中，其中，item 数组的第一数据为新闻类别。

- train_label.append(newstype[item[0]])

将新闻类别保存到 train_label 数组。有上图可知，item 数组的第一个数据为新闻类别，因此，然后通过 newstype 数据将文本数据转换成数值，然后保存在 train_label 数组中，当我们循环完所有训练新闻数据后，train_label 按顺序依次保

存所有新闻类别数值，如下图所示，数值 0 表示该新闻是体育新闻。

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

图 8-17　新闻类别部分值显示

- tmp="".join(item[1:])

将字符串数组 item 除了第一项后面所有的新闻内容拼接成一个字符串，输出结果为图 8-18 ，此时的字符串中删除了所有的空格。

```
马晓旭意外受伤让国奥警惕无奈大雨格外青睐殷家军记者傅亚雨沈阳报道来到沈阳，国奥队依然没有摆脱雨水的困扰。7月31日下
午6点，国奥队的日常训练再度受到大雨的干扰，无奈之下队员们只慢跑了25分钟就草草收场。31日上午10点，国奥队在奥体中心外
场训练的时候，天就是阴沉沉的，气象预报显示当天下午沈阳就有大雨，但幸好队伍上午的训练并没有受到任何干扰。下午6点，当球
队抵达训练场时，大雨已经下了几个小时，而且丝毫没有停下来的意思。抱着试一试的态度，球队开始了当天下午的例行训练，25分
钟过去了，天气没有任何转好的迹象，为了保护球员们，国奥队决定中止当天的训练，全队立即返回酒店。在雨中训练对足球队来说并
不是什么稀罕事，但在奥运会即将开始之前，全队变得“娇贵”了。在沈阳最后一周的训练，国奥队首先要保证现有的球员不再出现意
外的伤病情况以免影响正式比赛，因此这一阶段控制训练受伤、控制感冒等疾病的出现被队伍放在了相当重要的位置。而抵达沈阳之
后，中后卫冯萧霆就一直没有训练，冯萧霆是7月27日在长春患上了感冒，因此也没有参加29日跟塞尔维亚的热身赛。队伍介绍说，
冯萧霆并没有出现发烧症状，但为了安全起见，这两天还是让他静养休息，等感冒彻底好了之后再恢复训练。由于有了冯萧霆这个例
子，因此国奥队对雨中训练就显得特别谨慎，主要是担心球员们受凉而引发感冒，造成非战斗减员。而女足队员马晓旭在热身赛中受
伤导致无缘奥运的前科，也让在沈阳的国奥队现在格外警惕，“训练中不断嘱咐队员们要注意动作，我们可不能再出这样的事情了。”
一位工作人员表示。从长春到沈阳，雨水一路伴随着国奥队，“也邪了，我们走到哪儿雨就下到哪儿，在长春几次训练都被大雨给搅和
了，没想到来沈阳又碰到这种事情。”一位国奥球员也对雨水的“青睐”有些不解。
```

图 8-18　对 split 分割后的新闻内容重新拼接成一个字符串

- text=fenci.fenci1(tmp)

对新闻内容进行分词，结果返回一个分词后进行拼接的字符串，分开的词之间用空格隔开。如图 8-19 中的显示第一条新闻分词后进行拼接的字符串。

```
马晓旭 意外 受伤 让 国奥 警惕 无奈 大雨 格外 青睐 殷家 军 记者 傅亚雨 沈阳 报道 来到 沈阳 ， 国奥队 依然 没
有 摆脱 雨水 的 困扰 。 7 月 31 日 下午 6 点 ， 国奥队 的 日常 训练 再度 受到 大雨 的 干扰 ， 无奈 之下 队员
们 只 慢跑 了 25 分钟 就 草草收场 。 31 日 上午 10 点 ， 国奥队 在 奥体中心 外场 训练 的 时候 ， 天 就是 阴
沉沉 的 ， 气象预报 显示 当天 下午 沈阳 就 有 大雨 ， 但 幸好 队伍 上午 的 训练 并 没有 受到 任何 干扰 。 下
午 6 点 ， 当 球队 抵达 训练场 时 ， 大雨 已经 下 了 几个 小时 ， 而且 丝毫 没有 停下来 的 意思 。 抱 着 试一
试 的 态度 ， 球队 开始 了 当天 下午 的 例行 训练 ， 25 分钟 过去 了 ， 天气 没有 任何 转好 的 迹象 ， 为了
保护 球员 们 ， 国奥队 决定 中止 当天 的 训练 ， 全队 立即 返回 酒店 。 在 雨 中 训练 对 足球队 来说 并 不是
什么 稀罕 事 ， 但 在 奥运会 即将 开始 之前 ， 全队 变得 “ 娇贵 ” 了 。 在 沈阳 最后 一周 的 训练 ， 国奥队
首先 要 保证 现有 的 球员 不再 出现意外 的 伤病 情况 以免 影响 正式 比赛 ， 因此 这一 阶段 控制 训练 受伤 、
控制 感冒 等 疾病 的 出现 被 队伍 放在 了 相当 重要 的 位置 。 而 抵达 沈阳 之后 ， 中 后卫 冯萧霆 就 一直
没有 训练 ， 冯萧霆 是 7 月 27 日 在 长春 患上 了 感冒 ， 因此 也 没有 参加 29 日 跟 塞尔维亚 的 热身赛 。
队伍 介绍 说 ， 冯萧霆 并 没有 出现 发烧 症状 ， 但 为了 安全 起 见 ， 这 两天 还是 让 他 静养 休息 ， 等 感冒
彻底 好 了 之后 再 恢复 训练 。 由于 有 了 冯萧霆 这个 例子 ， 因此 国奥队 对雨中 训练 就 显得 特别 谨慎 ，
主要 是 担心 球员 们 受凉 而 引发 感冒 ， 造成 非战斗 减员 。 而 女足 队员 马晓旭 在 热身赛 中 受伤 导致 无缘
奥运 的 前科 ， 也 让 在 沈阳 的 国奥队 现在 格外 警惕 ， “ 训练 中 不断 嘱咐 队员 们 要 注意 动作 ， 我们 可
不能 再出 这样 的 事情 了 。 ” 一位 工作人员 表示 。 从 长春 到 沈阳 ， 雨水 一路 伴随 着 国奥队 ， “ 也 邪
了 ， 我们 走 到 哪儿 雨 就 下 到 哪儿 ， 在 长春 几次 训练 都 被 大雨 给 搅和 了 ， 没想到 来 沈阳 又 碰到
这种 事情 。 ” 一位 国奥 球员 也 对 雨水 的 “ 青睐 ” 有些 不解 。
```

图 8-19　分词后的字符串

- train_content.append(text)

将分词后的新闻内容保存到 train_content 数组中，数组中的每一项就是一个分词后的字符串。

- print("训练数据一共有"+str(n)+"条新闻")

打印训练数据的条数。

- file_tl=open('train_label.txt','w')

创建 train_label.txt 文件用来保存分词后的新闻类别

- file_tl.write(str(train_label))

将 train_label 数组中的内容写入 train_label.txt 文件中

- file_tl.close()

关闭文件

- fp=open('cnews.test.txt')
- while True:
- cur_text=fp.readline()
- if cur_text=="":
- break
- item=cur_text.split()
- test_label.append(newstype[item[0]])
- tmp=''.join(item[1:])
- text=fenci.fenci1(tmp)
- test_content.append(text)
- n=n+1
- print("测试数据一共有"+str(n)+"条新闻")
- file_testc=open('test_content.txt','w')
- file_testc.write(str(test_content));
- file_testc.close() #关闭文件
- file_testl=open('test_label.txt','w')
- file_testl.write(str(test_label));
- file_testl.close()

对测试数据的分词，分词过程同训练数据的分词。

- print(train_label[0])

打印训练数据中第一条新闻的类别。

- print(train_content[0])

打印训练数据中第一条新闻进行分词后的结果。

- print('**********success to finish fenci readNewsData**********')

8.5.3 创建中文词字典

（1）在 jupyter notebook 中输入图 8-20 中显示的代码，并确认代码无错误。

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

token= Tokenizer(num_words=1000)    #设置字典中词的数量
token.fit_on_texts(train_content)        #读取所有词，并将出现次数前3000的词放入字典

print(token.document_count)     #显示读取的新闻条数
```

图 8-20 建立 MLP 学习模型代码

（2）按 Ctrl+Enter 组合键执行代码确认代码正确无误。

（3）按下 Shift+Enter 组合键新建下一个单元格。

（4）下面对创建中文词字典的过程进行逐行解析，对整个模型的创建进行更深入的了解。

- from keras.preprocessing import sequence

导入 keras 中 preprocessing 模块中的 sequence 函数

- from keras.preprocessing.text import Tokenizer

导入 keras 中 preprocessing.text 模块中的 Tokenizer 函数

- token= Tokenizer(num_words=3000)

设置字典中词的数量为 3000。num_words 用来初始化一个 Tokenizer 类，表示用多少词语生成词典（vocabulary），给定以后，就用出现频次最多的 3000 个数生成词典，其余的低频词丢掉.

Tokenizer 是一个用于向量化文本，或将文本转换为序列（即单词在字典中的下标构成的列表，从 1 算起）的类。其函数原型为，

keras.preprocessing.text.Tokenizer(num_words=None,

filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',

lower=True,

split=" ",

char_level=False)

其输入参数为：

- filters：需要滤除的字符的列表或连接形成的字符串，例如标点符号。默认值

为 '!"#$%&()*+,-./:;<=>?@[]^_`{|}~\t\n'，包含标点符号，制表符和换行符等

- lower：布尔值，是否将序列设为小写形式
- split：字符串，单词的分隔符，如空格
- num_words：None 或整数，处理的最大单词数量。若被设置为整数，则分词器将被限制为待处理数据集中最常见的 num_words 个单词
- char_level: 如果为 True, 每个字符将被视为一个标记

表 8-2　Tokenizer 的类方法和属性

方法	fit_on_texts(texts)	• texts：要用以训练的文本列表
	texts_to_sequences(texts)	• texts：待转为序列的文本列表 • 返回值：序列的列表，列表中每个序列对应于一段输入文本
	texts_to_sequences_generator(texts)	• 本函数是 texts_to_sequences 的生成器函数版 • texts：待转为序列的文本列表 • 返回值：每次调用返回对应于一段输入文本的序列
	texts_to_matrix(texts, mode)	• texts：待向量化的文本列 • mode：‘binary’，‘count’，‘tfidf’，‘freq’之一，默认为‘binary’ • 返回值：形如 (len(texts), nb_words) 的 numpy array
	fit_on_sequences(sequences):	• sequences：要用以训练的序列列表
	sequences_to_matrix(sequences):	• sequences：待向量化的序列列表 • mode：‘binary’，‘count’，‘tfidf’，‘freq’之一，默认为‘binary’ • 返回值：形如(len(sequences), nb_words)的 numpy array
属性	word_counts	• 字典，将单词（字符串）映射为它们在训练期间出现的次数。仅在调用 fit_on_texts 之后设置。
	word_docs	• 字典，将单词（字符串）映射为它们在训练期间所出现的文档或文本的数量。仅在调用 fit_on_texts 之后设置。
	word_index	• 字典，将单词（字符串）映射为它们的排名或者索引。仅在调用 fit_on_texts 之后设置。

	document_count	• 整数。分词器被训练的文档（文本或者序列）数量。仅在调用 fit_on_texts 或 fit_on_sequences 之后设置。

- token.fit_on_texts(train_content)

读取所有词，并将出现次数前 3000 的词放入字典，运行完这一句 token 就将所有单词的出现过的次数统计好了 1 代表出现频率最高的单词，2 代表出现频率第二高的单词，3 代表.....可以用 token.word_index 查看单词排列顺序。

- print(token.document_count)

显示读取的新闻条数，我们知道训练数据有 50000 条，因此，读取新闻的条数为 50000。

8.5.4 数据预处理

- trainSqe=token.texts_to_sequences(train_content)

使用 token 中的字典将新闻内容转换成数字列表。token.texts_to_sequences 的函数原型为：

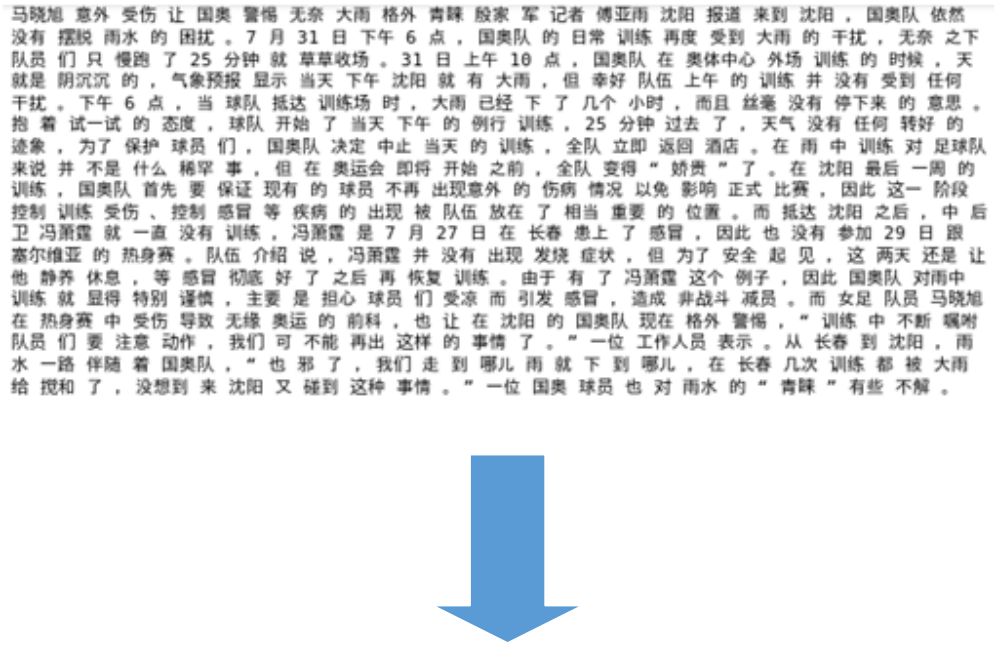
```
keras.preprocessing.text.text_to_word_sequence(text,
                                                filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',
                                                lower=True,
                                                split=" ")
```

本函数将一个句子拆分成单词构成的列表，其输入参数为，

- text：字符串，待处理的文本
- filters：需要滤除的字符的列表或连接形成的字符串，例如标点符号。默认值为 '!"#$%&()*+,-./:;<=>?@[]^_`{|}~\t\n'，包含标点符号，制表符和换行符等
- lower：布尔值，是否将序列设为小写形式
- split：字符串，单词的分隔符，如空格

返回值为字符串列表。

如下图 8-21 输入的文本 train_content[0]的内容经转换后变成数字列表，



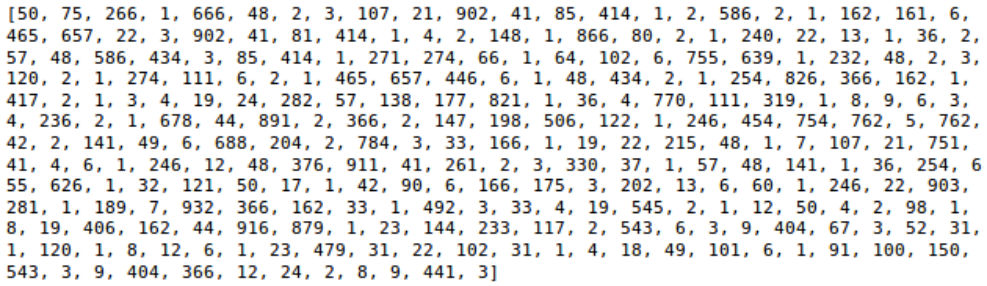


图 8-21　中文内容转换成数字列表

- testSqe=token.texts_to_sequences(test_content)

同上，将测试新闻内容转换成数字列表。

- print(trainSqe[1])

显示第二条新闻的数字列表。

- trainPadSqe=sequence.pad_sequences(trainSqe,maxlen=128)

将数字列表统一长度，缺少的在前面补 0。pad_sequences()函数的功能是填充序列，上面将 trainSqe 数字序列统一填充为 128 长度的数字向量。

pad_sequences()函数原型为

keras.preprocessing.sequence.pad_sequences(sequences, maxlen=None, dtype='int32',

padding='pre', truncating='pre', value=0)

该函数将长为 nb_samples 的序列（标量序列）转化成尺寸大小是

(nb_samples,nb_timesteps)的二维 numpy 数组。如果提供了参数 maxlen，nb_timesteps=maxlen，否则其值为最长序列的长度。其他短于该长度的序列都会在后部填充 0 以达到该长度。长于 nb_timesteps 的序列将会被截断，以使其匹配目标长度。padding 和截断发生的位置分别取决于 padding 和 truncating，函数的输入参数为：

- sequences：浮点数或整数构成的两层嵌套列表
- maxlen：None 或整数，为序列的最大长度。大于此长度的序列将被截短，小于此长度的序列将在后部填 0.
- dtype：返回的 numpy array 的数据类型
- padding：‘pre’或‘post’，确定当需要补 0 时，在序列的起始还是结尾补
- truncating：‘pre’或‘post’，确定当需要截断序列时，从起始还是结尾截断
- value：浮点数，此值将在填充时代替默认的填充值 0

函数的返回值形如(nb_samples,nb_timesteps)的 2D 张量。

- testPadSqe=sequence.pad_sequences(testSqe,maxlen=128)
- print(len(trainSqe[0]))

第一条新闻转换成数字列表后长度为 279。

- print(len(trainPadSqe[0]))

第一条新闻的数字列表统一长度后为 128。

- print(trainPadSqe[0])

显示统一长度后的第一条新闻数字列表,如下图 8-22 ，经过统一长度后，第一条新闻的数字列表。

```
[50, 75, 266, 1, 666, 48, 2, 3, 107, 21, 902, 41, 85, 414, 1, 2, 586, 2, 1, 162, 161, 6,
465, 657, 22, 3, 902, 41, 81, 414, 1, 4, 2, 148, 1, 866, 80, 2, 1, 240, 22, 13, 1, 36, 2,
57, 48, 586, 434, 3, 85, 414, 1, 271, 274, 66, 1, 64, 102, 6, 755, 639, 1, 232, 48, 2, 3,
120, 2, 1, 274, 111, 6, 2, 1, 465, 657, 446, 6, 1, 48, 434, 2, 1, 254, 826, 366, 162, 1,
417, 2, 1, 3, 4, 19, 24, 282, 57, 138, 177, 821, 1, 36, 4, 770, 111, 319, 1, 8, 9, 6, 3,
4, 236, 2, 1, 678, 44, 891, 2, 366, 2, 147, 198, 506, 122, 1, 246, 454, 754, 762, 5, 762,
42, 2, 141, 49, 6, 688, 204, 2, 784, 3, 33, 166, 1, 19, 22, 215, 48, 1, 7, 107, 21, 751,
41, 4, 6, 1, 246, 12, 48, 376, 911, 41, 261, 2, 3, 330, 37, 1, 57, 48, 141, 1, 36, 254, 6
55, 626, 1, 32, 121, 50, 17, 1, 42, 90, 6, 166, 175, 3, 202, 13, 6, 60, 1, 246, 22, 903,
281, 1, 189, 7, 932, 366, 162, 33, 1, 492, 3, 33, 4, 19, 545, 2, 1, 12, 50, 4, 2, 98, 1,
8, 19, 406, 162, 44, 916, 879, 1, 23, 144, 233, 117, 2, 543, 6, 3, 9, 404, 67, 3, 52, 31,
1, 120, 1, 8, 12, 6, 1, 23, 479, 31, 22, 102, 31, 1, 4, 18, 49, 101, 6, 1, 91, 100, 150,
543, 3, 9, 404, 366, 12, 24, 2, 8, 9, 441, 3]
```

```
[  7 107  21 751  41   4   6   1 246  12  48 376 911  41 261   2   3 330
  37   1  57  48 141   1  36 254 655 626   1  32 121  50  17   1  42  90
   6 166 175   3 202  13   6  60   1 246  22 903 281   1 189   7 932 366
 162  33   1 492   3  33   4  19 545   2   1  12  50   4   2  98   1   8
  19 406 162  44 916 879   1  23 144 233 117   2 543   6   3   9 404  67
   3  52  31   1 120   1   8  12   6   1  23 479  31  22 102  31   1   4
  18  49 101   6   1  91 100 150 543   3   9 404 366  12  24   2   8   9
 441   3]
```

图 8-22　统一长度为 128 的第一条新闻数字列表

print(trainPadSqe.shape)

其结果显示为(50000, 128)，即训练数据集转换成有 50000 条数据，每条数据长为 128 的二维数组。

- train_label_ohe=np_utils.to_categorical(train_label)
- test_label_ohe=np_utils.to_categorical(test_label)

将训练数据分类标签和测试数据分类标签进行 one-hot-encoding 编码。

- print(train_label_ohe[0])

显示数据分类标签进行 one-hot-encoding 编码后的结果，其结果显示为[1,0,0,0,0,0,0,0,0,0,0],表示标签数为 0，对应的“体育”类别。

8.5.5 创建基于 RNN 的训练模型

（1）在 jupyter notebook 中输入图 8-23 中显示的代码，并确认代码无错误。

```
from keras.models import Sequential
from keras.layers import Dense, Dropout,Activation,Flatten
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM
from keras.layers.recurrent import SimpleRNN

model =Sequential() #搭建线性框架
model.add(Embedding(output_dim=32,
                    input_dim=3000,
                    input_length=128))  #增加嵌入层,将数字列表装换成向量列表

model.add(Dropout(0.25))
model.add(SimpleRNN(units=32))
model.add(Dropout(0.3))
model.add(Dense(units=256,activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(units=10,activation='softmax'))
print(model.summary())
```

图 8-23 建立 RNN 学习模型代码

（2）按 Ctrl+Enter 组合键执行代码，代码没有错误提示后按 Shift+Enter 组合键新建下一个单元格，结果显示如图 8-24 所示：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 128, 32)           96000
_________________________________________________________________
dropout_9 (Dropout)          (None, 128, 32)           0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dropout_10 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               8448
_________________________________________________________________
dropout_11 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                2570
=================================================================
Total params: 109,098
Trainable params: 109,098
Non-trainable params: 0
_________________________________________________________________
None
```

图 8-24 RNN 学习模型内容

（3）下面对学习模型创建代码进行逐行解析，对整个模型的创建进行更深入的了解。

- from keras.models import Sequential

  导入贯序模型创建函数。

- from keras.layers import Dense, Dropout,Activation,Flatten

  导入 Dense 层、Dropout 层、Flatten 层创建函数，导入激活函数。

- from keras.layers.embeddings import Embedding

导入嵌入层创建函数。

- from keras.layers.recurrent import SimpleRNN

导入 RNN 层创建函数。

- from keras.layers.recurrent import LSTM

导入长短时记忆网络层创建函数。

- model =Sequential()

搭建线性框架。

- model.add(Embedding(output_dim=32,input_dim=3000,input_length=128))

增加词嵌入层，将数字列表装换成向量列表。词嵌入是使用密集向量表示来表示单词和文档的一类方法。这是对传统的袋型（bag-of-word）模型编码方案的改进，其中使用大的稀疏向量来表示每个单词或向量中的每个单词进行数字分配以表示整个词汇表。这些表示是稀疏的，因为词汇是广泛的，这样一个给定的单词或文档将由一个主要由零值组成的向量几何表示。相反，在词嵌入中，词由密集向量表示，其中矢量表示单词投射到连续向量空间中。一个单词在向量空间中的位置是从文本中学习的，并且基于使用该文本时的单词。在学习向量空间中的单词的位置称为嵌入位置。

例如同样的“今天”、“天空”、“很”、“蓝”这四个词（同样假设全世界的中文词就这 4 个），经过编码以后可能变成了：“今天” 对应（0.2），“天空”对应（0.32），“很”对应（0.1），“蓝”对应（0.35）。

我们假设嵌入的空间为 256 维（一般是 256，512 或者 1024 维，词汇表越大，对应的空间维度越高）。那么

“今天” 对应（0.1，0.2，0.4，0，…）（向量长度为 256）

“天空”对应（0.23，0.14，0，0,…）

“很”对应（0，0，0.41，0.9,…）

“蓝”对应（0，0.82，0，0.14,…）

嵌入层将正整数（下标）转换为具有固定大小的向量，如[[4],[20]]->[[0.25,0.1],[0.6,-0.2]]，Embedding 层只能作为模型的第一层，嵌入层函数原型为 keras.layers.embeddings.Embedding(input_dim,

output_dim,

```
embeddings_initializer='uniform',
embeddings_regularizer=None,
activity_regularizer=None,
embeddings_constraint=None,
mask_zero=False,
input_length=None)
```

其输入参数为：

- input_dim：大或等于 0 的整数，字典长度，即输入数据最大下标+1
- output_dim：大于 0 的整数，代表全连接嵌入的维度
- embeddings_initializer: 嵌入矩阵的初始化方法，为预定义初始化方法名的字符串，或用于初始化权重的初始化器。参考 initializers
- embeddings_regularizer: 嵌入矩阵的正则项，为 Regularizer 对象
- embeddings_constraint: 嵌入矩阵的约束项，为 Constraints 对象
- mask_zero：布尔值，确定是否将输入中的‘0’看作是应该被忽略的‘填充’（padding）值，该参数在使用递归层处理变长输入时有用。设置为 True 的话，模型中后续的层必须都支持 masking，否则会抛出异常。如果该值为 True，则下标 0 在字典中不可用，input_dim 应设置为|vocabulary| + 1。
- input_length：当输入序列的长度固定时，该值为其长度。如果要在该层后接 Flatten 层，然后接 Dense 层，则必须指定该参数，否则 Dense 层的输出维度无法自动推断。

输入参数的 shape 形如（samples，sequence_length）的 2D 张量，例如（50000,128），表示 50000 个样本，每个样本长度为 128。

输出参数的 shape 形如(samples, sequence_length, output_dim)的 3D 张量，例如（50000,128,32），表示有 50000 个样本，每个样本长度为 128，这 128 个单元每个是长度为 32 的向量，如下图 8-25 所示。

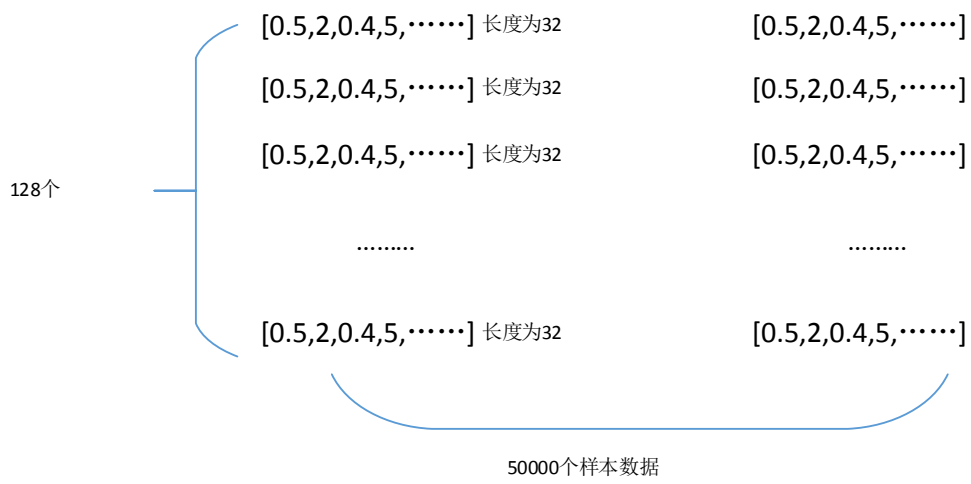


图 8-25　嵌入层转换后的结果

- model.add(Dropout(0.25))

增加 Dropout 层，每次丢弃 25%的神经元。

- model.add(SimpleRNN(units=32))

增加 RNN 层，该层有 32 个神经元节点。SimpleRNN 为全连接 RNN 网络，RNN 的输出会被回馈到输入。其函数原型为

keras.layers. SimpleRNN (units,

activation='tanh',

recurrent_activation='hard_sigmoid',

use_bias=True,

kernel_initializer='glorot_uniform',

recurrent_initializer='orthogonal',

bias_initializer='zeros',

kernel_regularizer=None,

recurrent_regularizer=None,

bias_regularizer=None,

activity_regularizer=None,

kernel_constraint=None,

recurrent_constraint=None,

bias_constraint=None,

dropout=0.0,

recurrent_dropout=0.0,

implementation=1,

return_sequences=False,

return_state=False,

go_backwards=False,

stateful=False,

unroll=False)

输入参数的定义为：

- units：输出维度
- activation：激活函数，为预定义的激活函数名（参考激活函数）
- use_bias: 布尔值，是否使用偏置项
- kernel_initializer：权值初始化方法，为预定义初始化方法名的字符串，或用于初始化权重的初始化器。参考 initializers
- recurrent_initializer：循环核的初始化方法，为预定义初始化方法名的字符串，或用于初始化权重的初始化器。参考 initializers
- bias_initializer：权值初始化方法，为预定义初始化方法名的字符串，或用于初始化权重的初始化器。参考 initializers
- kernel_regularizer：施加在权重上的正则项，为 Regularizer 对象
- bias_regularizer：施加在偏置向量上的正则项，为 Regularizer 对象
- recurrent_regularizer：施加在循环核上的正则项，为 Regularizer 对象
- activity_regularizer：施加在输出上的正则项，为 Regularizer 对象
- kernel_constraints：施加在权重上的约束项，为 Constraints 对象
- recurrent_constraints：施加在循环核上的约束项，为 Constraints 对象
- bias_constraints：施加在偏置上的约束项，为 Constraints 对象
- dropout：0~1 之间的浮点数，控制输入线性变换的神经元断开比例
- recurrent_dropout：0~1 之间的浮点数，控制循环状态的线性变换的神经元断开比例。

- model.add(Dense(units=1024,activation='relu'))

构建隐藏层，全连接结构，神经元个数 1024 个，初始化权重为默认值,激活

函数为'relu'。

- model.add(Dropout(0.4))

构建 dropout 层，每次计算随机丢弃 40%的神经元

- model.add(Dense(units=10,activation='softmax'))

构建输出层，全连接结构，神经元个数 10 个，初始化权重为默认值（kernel_initializer='glorot_uniform'），激活函数为 softmax。

- print(model.summary())

打印出模型概况，如图 5-12 所示。它实际调用的是 keras.utils.print_summary。

embedding_5 (Embedding) 为嵌入层，有 96000 个参数，simple_rnn_1 (SimpleRNN)为循环神经网络层，有 2080 个参数。

dense_7 为隐藏层，有 8448 个参数，因为输入层有 32 个单元，隐藏层有 256 个单元，按照全连接模式，一共需要（32+1）*256=8448 个权重参数进行训练。dense_8 为输出层，按照全连接模式，一共有参数（256+1）x10=2570 个参数。

整个模型参数一共有 109098 个参数需要通过数据集进行训练获得。此外，还有 dropout 层（dropout_7 和 dropout_8），由于 dropout 层只随机丢弃神经元，不需要权重参数，因此，权重参数个数均为 0。

8.5.6 对模型进行训练

（1）在 jupyter notebook 中输入图 8-26 中显示的代码，并确认代码无错误。

```
try:
    model.load_weights("newstpye.h5")
    print("成功加载已有模型,继续训练该模型")
except:
    print("没有模型加载,开始训练新模型")

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['categorical_accuracy'])

train_history=model.fit(x=trainPadSqe,y=train_label_ohe,
                        validation_split=0.2,epochs=5,
                        batch_size=100,verbose=1)

model.save_weights("newstpye.h5")
print("保存刚训练的模型")
```

图 8-26 对模型进行训练的代码

（2）按下 Ctrl+Enter 组合键,显示代码的运行结果

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/5
40000/40000 [==============================] - 26s 640us/step - loss: 1.2784 - categorica
l_accuracy: 0.4890 - val_loss: 7.5441 - val_categorical_accuracy: 0.0000e+00
Epoch 2/5
40000/40000 [==============================] - 23s 585us/step - loss: 0.7027 - categorica
l_accuracy: 0.7419 - val_loss: 9.2274 - val_categorical_accuracy: 0.0000e+00
Epoch 3/5
40000/40000 [==============================] - 27s 663us/step - loss: 0.5987 - categorica
l_accuracy: 0.7979 - val_loss: 10.2355 - val_categorical_accuracy: 0.0000e+00
Epoch 4/5
40000/40000 [==============================] - 23s 586us/step - loss: 0.5225 - categorica
l_accuracy: 0.8301 - val_loss: 10.3943 - val_categorical_accuracy: 0.0000e+00
Epoch 5/5
40000/40000 [==============================] - 21s 537us/step - loss: 0.4710 - categorica
l_accuracy: 0.8541 - val_loss: 13.2955 - val_categorical_accuracy: 0.0000e+00
```

图 8-27 模型训练过程

（3）按下 Shift+Enter 组合键新建下一个单元格。

（4）下面对模型进行训练的代码进行逐行解析，能对模型的学习设置和学习过程进行更深入的了解。

- try:
- model.load_weights("newstpye.h5")
- print("成功加载已有模型,继续训练该模型")
- except:
- print("没有模型加载，开始训练新模型")

如果已有模型参数，则加载，在此基础上进行计算；如果没有，则重新开始计算。

- model.compile（loss='categorical_crossentropy',optimizer='adam', metrics=['categorical_accuracy']）

调用 model.compile()函数对训练模型进行设置，参数设置为：

loss='categorical_crossentropy'：loss（损失函数）设置为交叉熵模式，在深度学习中用交叉熵模式训练效果会比较好。

optimizer='adam'：optimizer（优化器）设置为 adam，在深度学习中可以让训练更快收敛，并提高准确率。

metrics=[' categorical_accuracy ']：对多分类问题,计算在所有预测值上的平均正确率。

- train_history=model.fit(x=trainPadSqe,y=train_label_ohe,validation_split=0.2,epochs=5,batch_size=128,verbose=1)

调用 model.fit 配置训练参数，开始训练，并保存训练结果。

x=x_train_normalize：数据集中已经经过预处理的训练集图像

y=y_label_ohe:数据集中已经经过预处理的训练集 label

validation_split=0.2：训练之前将输入的训练数据集中 80%作为训练数据，20%作为测试数据。

epochs=5：设置训练周期为 5 次。

batch_size=128：设置每一次训练周期中，训练数据每次输入多少个。

verbose=1：设置成输出进度条记录。

8.5.7 利用测试数据进行预测评估与识别

（1）在 jupyter notebook 中输入图 8-28 中显示的代码，并确认代码无错误。

```
results=model.evaluate(testPadSqe,test_label_ohe)
print('acc=',results[1])
predic_pro=model.predict(testPadSqe)
def show_categorical(num):
    print('the news label is :'+str(test_label[num]))
    for j in range(10):
        print(' newstype '+str(j)+' probility is: %1.9f'%(predic_pro[num][j]))
    print(test_content[num])
show_categorical(2345)
```

图 8-28 利用测试数据进行预测评估与识别代码

（2）按下 Ctrl+Enter 组合键,显示代码的运行结果，按下 Shift+Enter 组合键新建下一个单元格。

```
10000/10000 [==============================] - 3s 315us/step
acc= 0.6727
the news label is :2
 newstype 0 probility is: 0.000295031
 newstype 1 probility is: 0.001610436
 newstype 2 probility is: 0.836724639
 newstype 3 probility is: 0.000527066
 newstype 4 probility is: 0.024743959
 newstype 5 probility is: 0.120044328
 newstype 6 probility is: 0.007421518
 newstype 7 probility is: 0.008623060
 newstype 8 probility is: 0.000005036
 newstype 9 probility is: 0.000004948
地板 专家 价格 过低 强化 地板 选购 时 当心 一般 家庭 选用 强化 地板 及 多层 实 木地板 的 较 多 。目前 强化
地板 的 基材 用 的 大都 是 高 密度板 ， 是 由 速生林 打碎 后用 胶 通过 高温高压 制成 ， 有 很 高 的 强度 ，
同时 耐冲击 。 基材 是 这类 地板 的 主要 部分 。 这种 地板 一般 分 四层 ， 第一层 是 表面 耐磨 层 ， 由 三氧化
二铝 构成 ； 然后 是 装饰 层 ， 由 电脑设计 的 木纹 装饰 纸 ； 再往 下 就是 基材 ； 最后 一层 是 背板 ， 起 防
潮 平衡 的 作用 。 由于 这种 地板 需要 这么 多 原材料 ， 同时 还要 加上 生产成本 （ 机器设备 的 磨损 、 人员
费用 等 ） 、 销售费用 、 安装 成本 ， 所以 它 的 价格 应该 高于 每平方米 70 元 ， 如果 这 类产品 低于 70 元 ，
那么 选择 上 就要 谨慎 为 好 。 还有 一种 多层 实 木地板 ， 一般 是 由 三层 或 多层 结构 组成 。 最 表面 一层
一般 是 由 0.6 - 4 毫米 厚 的 珍稀 树种 制作 ， 比较 硬 ； 中间 和 最 下面 一层 用 的 木质 比表面 稍软 一些
， 按照 上中下 多 层级 纵横交错 地 粘合 起来 ， 因为 木材 的 横纹 方向 变型 较大 ， 顺纹 方向 的 变形 基本 可
```

图 8-29 测试数据在 RNN 模型中进行预测的结果显示

（3）下面对使用测试数据进行预测的代码进行详细的解析，以便掌握使用模型进行预测的方法。

- results=model.evaluate(testPadSqe,test_label_ohe)

testPadSqe：输入数据为预处理后的测试数据集

test_label_ohe：测试数据分类标签预处理后的测试标签集

- print('acc=',results[1])

显示测试数据集的预测准确率。有图 8-29 可知，测试数据的预测准确率为 72.85%。

- predic_pro=model.predict(testPadSqe)

计算预测概率，因为本结果有 10 中类别，该方法可以获得在每种类别下的预测概率值。返回结果为二维数组。如 predic_pro[i][j]定义为表示第 i 个数据在第 j 类标签上的预测概率。如 predic_pro[0][0]=1，表示第 0 个数据在第 0 类标签上的预测概率为 100%。

- def show_categorical(num):

定义显示分类预测概率函数，输入值为测试新闻内容的序号值。

- print('the news label is :'+str(test_label[num]))

打印测试新闻内容对应的分类标签值。

- for j in range(10):
- print(' newstype '+str(j)+' probility is: %1.9f'%(predic_pro[num][j]))

循环输出从 0~9 十个新闻类别标签下的预测概率值。

- print(test_content[num])

输出选择显示的测试新闻的内容。

- show_categorical(2345)

显示第 2345 条新闻的对应分类标签值、在每个类别下的预测概率以及该新闻的内容。

8.5.8 创建基于 LSTM 的模型进行训练和预测

（1）在 jupyter notebook 中将 8.5.5 节中的代码修改为如下图 8-30 所示，

```
from keras.models import Sequential
from keras.layers import Dense, Dropout,Activation,Flatten
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM
from keras.layers.recurrent import SimpleRNN

model =Sequential() #搭建线性框架
model.add(Embedding(output_dim=32,
                    input_dim=3000,
                    input_length=128))  #增加嵌入层,将数字列表装换成向量列表

model.add(Dropout(0.25))
model.add(LSTM(48))
model.add(Dropout(0.3))
model.add(Dense(units=256,activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(units=10,activation='softmax'))
print(model.summary())
```

图 8-30 LSTM 的模型创建

将 8.5.6 节中的代码修改为如下图所示，并确认代码无错误。

```
try:
    model.load_weights("newstpyelstm.h5")
    print("成功加载已有模型,继续训练该模型")
except:
    print("没有模型加载,开始训练新模型")

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['categorical_accuracy'])

train_history=model.fit(x=trainPadSqe,y=train_label_ohe,
                        validation_split=0.2,epochs=5,
                        batch_size=100,verbose=1)

model.save_weights("newstpyelstm.h5")
print("保存刚训练的模型")
```

图 8-31 LSTM 的学习

（2）重新运行整个程序，模型搭建的结果显示如图 8-32 所示。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 128, 32)           96000
_________________________________________________________________
dropout_12 (Dropout)         (None, 128, 32)           0
_________________________________________________________________
lstm_4 (LSTM)                (None, 48)                15552
_________________________________________________________________
dropout_13 (Dropout)         (None, 48)                0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               12544
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                2570
=================================================================
Total params: 126,666
Trainable params: 126,666
Non-trainable params: 0
_________________________________________________________________
None
```

图 8-32　LSTM 的模型的内容

（3）下面对长短时记忆网络学习模型创建代码进行逐行解析，对整个模型的创建进行更深入的了解。

- model.add(LSTM(48))

创建一个有 48 个神经元节点的长短时记忆网络层。LSTM（长短时记忆网络）层创建使用 LSTM()函数，其函数原型为

```
keras.layers.LSTM(units,
                  activation='tanh',
                  recurrent_activation='hard_sigmoid',
                  use_bias=True,
                  kernel_initializer='glorot_uniform',
                  recurrent_initializer='orthogonal',
                  bias_initializer='zeros',
                  unit_forget_bias=True,
                  kernel_regularizer=None,
                  recurrent_regularizer=None,
                  bias_regularizer=None,
                  activity_regularizer=None,
                  kernel_constraint=None,
                  recurrent_constraint=None,
                  bias_constraint=None,
                  dropout=0.0,
                  recurrent_dropout=0.0,
                  implementation=1,
                  return_sequences=False,
                  return_state=False,
                  go_backwards=False,
                  stateful=False,
                  unroll=False)
```

其输入参数为：

- units: 正整数，输出空间的维度。
- activation: 要使用的激活函数 (详见 activations)。 如果传入 None，则不使用激活函数 (即 线性激活：a(x) = x)。
- recurrent_activation: 用于循环时间步的激活函数 (详见 activations)。 默认：分段线性近似 sigmoid (hard_sigmoid)。 如果传入 None，则不使用激活函数 (即 线性激活：a(x) = x)。
- use_bias: 布尔值，该层是否使用偏置向量。
- kernel_initializer: kernel 权值矩阵的初始化器， 用于输入的线性转换 (详见 initializers)。
- recurrent_initializer: recurrent_kernel 权值矩阵 的初始化器，用于循环层状态的线性转换 (详见 initializers)。
- bias_initializer:偏置向量的初始化器 (详见 initializers).
- unit_forget_bias: 布尔值。 如果为 True，初始化时，将忘记门的偏置加 1。将其设置为 True 同时还会强制 bias_initializer="zeros"。 这个建议来自 Jozefowicz et al.。
- kernel_regularizer: 运用到 kernel 权值矩阵的正则化函数 (详见 regularizer)。
- recurrent_regularizer: 运用到 recurrent_kernel 权值矩阵的正则化函数 (详见 regularizer)。
- bias_regularizer: 运用到偏置向量的正则化函数 (详见 regularizer)。
- activity_regularizer: 运用到层输出（它的激活值）的正则化函数 (详见 regularizer)。
- kernel_constraint: 运用到 kernel 权值矩阵的约束函数 (详见 constraints)。
- recurrent_constraint: 运用到 recurrent_kernel 权值矩阵的约束函数 (详见 constraints)。
- bias_constraint: 运用到偏置向量的约束函数 (详见 constraints)。
- dropout: 在 0 和 1 之间的浮点数。 单元的丢弃比例，用于输入的线性转换。
- recurrent_dropout: 在 0 和 1 之间的浮点数。 单元的丢弃比例，用于循环

层状态的线性转换。

- implementation: 实现模式，1 或 2。 模式 1 将把它的操作结构化为更多的小的点积和加法操作， 而模式 2 将把它们分批到更少，更大的操作中。 这些模式在不同的硬件和不同的应用中具有不同的性能配置文件。
- return_sequences: 布尔值。是返回输出序列中的最后一个输出，还是全部序列。
- return_state: 布尔值。除了输出之外是否返回最后一个状态。
- go_backwards: 布尔值 (默认 False)。 如果为 True，则向后处理输入序列并返回相反的序列。
- stateful: 布尔值 (默认 False)。 如果为 True，则批次中索引 i 处的每个样品的最后状态 将用作下一批次中索引 i 样品的初始状态。
- unroll: 布尔值 (默认 False)。 如果为 True，则网络将展开，否则将使用符号循环。 展开可以加速 RNN，但它往往会占用更多的内存。 展开只适用于短序列。

使用该模型进行训练的结果如下图 8-33 所示：

```
成功加载已有模型,继续训练该模型
Train on 40000 samples, validate on 10000 samples
Epoch 1/5
40000/40000 [==============================] - 64s 2ms/step - loss: 0.1963 - categorical_
accuracy: 0.9457 - val_loss: 15.5583 - val_categorical_accuracy: 0.0000e+00
Epoch 2/5
40000/40000 [==============================] - 62s 2ms/step - loss: 0.1809 - categorical_
accuracy: 0.9499 - val_loss: 15.8022 - val_categorical_accuracy: 0.0000e+00
Epoch 3/5
40000/40000 [==============================] - 63s 2ms/step - loss: 0.1722 - categorical_
accuracy: 0.9519 - val_loss: 15.9104 - val_categorical_accuracy: 0.0000e+00
Epoch 4/5
40000/40000 [==============================] - 66s 2ms/step - loss: 0.1851 - categorical_
accuracy: 0.9472 - val_loss: 15.8746 - val_categorical_accuracy: 0.0000e+00
Epoch 5/5
40000/40000 [==============================] - 66s 2ms/step - loss: 0.1645 - categorical_
accuracy: 0.9543 - val_loss: 15.9354 - val_categorical_accuracy: 0.0000e+00
保存刚训练的模型
```

图 8-33 LSTM 的模型的训练过程

模型训练好后，对模型进行测试，测试结果如下图所示：

```
10000/10000 [==============================] - 7s 720us/step
acc= 0.7285
the news label is :2
 newstype 0 probility is: 0.000003726
 newstype 1 probility is: 0.000023248
 newstype 2 probility is: 0.994669259
 newstype 3 probility is: 0.000014045
 newstype 4 probility is: 0.000540441
 newstype 5 probility is: 0.004594720
 newstype 6 probility is: 0.000135063
 newstype 7 probility is: 0.000019564
 newstype 8 probility is: 0.000000000
 newstype 9 probility is: 0.000000000
地板 专家 价格 过低 强化 地板 选购 时 当心 一般 家庭 选用 强化 地板 及 多层 实 木地板 的 较 多 。目前 强化
地板 的 基材 用 的 大都 是 高 密度板 ， 是 由 速生林 打碎 后用 胶 通过 高温高压 制成 ， 有 很 高 的 强度 ，
同时 耐冲击 。基材 是 这类 地板 的 主要 部分 。这种 地板 一般 分 四层 ， 第一层 是 表面 耐磨 层 ， 由 三氧化
二铝 构成 ； 然后 是 装饰 层 ， 由 电脑设计 的 木纹 装饰 纸 ； 再往 下 就是 基材 ； 最后 一层 是 背板 ， 起 防
潮 平衡 的 作用 。由于 这种 地板 需要 这么 多 原材料 ， 同时 还要 加上 生产成本 （ 机器设备 的 磨损 、 人员
费用 等 ） 、 销售费用 、 安装 成本 ， 所以 它 的 价格 应该 高于 每平方米 70 元 ， 如果 这 类产品 低于 70 元 ，
那么 选择 上 就要 谨慎 为 好 。还有 一种 多层 实 木地板 ， 一般 是 由 三层 或 多层 结构 组成 。最 表面 一层
一般 是 由 0.6 - 4 毫米 厚 的 珍稀 树种 制作 ， 比较 硬 ； 中间 和 最 下面 一层 用 的 木质 比表面 稍软 一些
， 按照 上中下 多 层级 纵横交错 地 粘合 起来 ， 因为 木材 的 横纹 方向 变型 较大 ， 顺纹 方向 的 变形 基本 可
以 忽略 ， 由于 多层 实木 交错 层叠 的 均衡 结构 ， 相邻 实 木板 受力 方向 垂直 ， 相互 抵消 ， 因而 可以 减少
```

图 8-34 LSTM 的模型的测试结果

8.6 项目代码完整示例

```python
### rask5-1.py
#使用MNIST数据集
#采用MLP模型进行训练
#
#**************** 第一步：准备知识库，导入手写数字图像数据集************************
from keras.utils import np_utils #导入keras模块中的utils函数内容
import numpy as np           #导入numpy模块
#用于指定随机数生成时所用算法开始的整数值，不指定则每次随机数都一样
np.random.seed(10)
from keras.datasets import mnist #导入keras模块中的datasets函数内容
#下载mnist数据集
(x_train_image, y_train_label),(x_test_image, y_test_label)=mnist.load_data()
#将二维图像数据转换成一维向量并改变类型为浮点数类型
x_train=x_train_image.reshape(60000,784).astype('float32')
x_test=x_test_image.reshape(10000,784).astype('float32')
#将一维向量数据归一化使得数据处于[0,1]区间
x_train_norm=x_train/255
x_test_norm=x_test/255
#将测试数据进行one-hot-encode处理
y_train_ohe=np_utils.to_categorical(y_train_label)
y_test_ohe=np_utils.to_categorical(y_test_label)
print('第一张训练照片数据为：')
print(x_train_image[0])
print('第一张训练照片转换成一维向量后数据为：')
print(x_train[0])
print('前3个训练label进行one-hot-encoding转换后的数据为：')
print(y_train_ohe[:3])
```

```python

#*************** 第二步：创建空白大脑，建立MLP学习模型************************
from keras.models import Sequential #导入keras模块中的models函数内容
from keras.layers import Dense#导入keras模块中的layers函数内容
model=Sequential() # 建立线性堆叠模型
model.add(Dense(units=256,input_dim=784,kernel_initializer='normal',activation='relu'))  #添加隐藏层
model.add(Dense(units=10,kernel_initializer='normal',activation='softmax')) #添加输出层
print(model.summary())#打印建立的模型内容
```

```
#*************** 第三步：将手写数字图像知识送给大脑学习，对模型进行训练**********************
model.compile(loss='categorical_crossentropy',/
optimizer='adam',metrics=['accuracy']) #对训练模型进行参数设置
train_history=model.fit(x=x_train_normalize,/
    y=y_label_onddd,validation_split=0.2,/
    epochs=10,batch_size=200,verbose=2) #设置训练参数，并启动训练
```

```
#*************** 第四步：显示大脑的学习过程和效果，显示模型训练过程中的准确率和误差变化***
import matplotlib.pyplot as plt #导入matplotlib模块中的pyplot函数进行图形绘制
def  show_train(train_histroy,train,validation):  #定义训练准确率和误差绘制函数
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.show()
#绘制训练数据准确率变化曲线和测试数据准确率变化曲线对比图
show_train(train_history,'acc','val_acc')
#绘制训练数据误差变化曲线和测试数据误差变化曲线对比图
show_train(train_history,'loss','val_loss')
```

```
#*************** 第五步：看看我创建的大脑是否足够聪明，利用模型进行预测和识别**********
results=model.evaluate(x_test_norm,y_test_ohe)#利用测试数据对模型进行评估
print('acc=',results[1]) #打印测试数据进行评估的准确率
prediction=model.predict_classes(x_test) #使用训练好的模型对测试数据进行分类
prediction[:10] #打印前10个测试数据预测分类的结果
```

8.7 小结与应用

本节简要介绍了自然语言处理的应用，详细介绍了中文分词工具 jieba 与中文分词的内容，并介绍了循环神经网络和长短时记忆网络模型，利用循环神经网络和长短时记忆网络模型进行多分类的处理，训练过程中每次训练都会保存训练后的模型数据，下次使用时会检查是否有已经训练过的模型，如果有，则读取模型数据继续进行训练，如果没有，则重新训练一个新模型。

习题

1.简述循环神经网络和长短时记忆网络。

2.在本章训练模型中，改变循环神经网络和长短时记忆网络的神经元个数、中文字典数量、隐藏层层数、对应的 dropout 层配置、模型的损失函数、优化器等，记录测试数据进行预测的准确率。

测试序号	隐藏层层数	模型的损失函数	优化器	测试数据准确率评估
1	例如：2 层隐藏层 隐藏层 1（2000），dropout 层（0.5） 隐藏层 2（1800），dropout 层（0.4）	categorical_crossentropy	adam	
2				

3				
4				
5				

3.编写函数，显示指定图像以及该图像的真实值和图像在模型中的预测结果。