第 3 章

CHAPTER 3

循环神经网络实战

序列数据是带顺序标签的数据，例如音频、视频和语音。因为序列数据的序列特性，学习序列数据是模式识别领域中最具挑战性的问题之一。在处理顺序数据时，序列各部分之间的依存关系及其变化的长度进一步增加了数据分析的复杂性。随着序列模型和算法[例如，递归神经网络(Recurrent Neural Network，RNN)、长短时记忆模型(Long Short-Term Memory model，LSTM)和门控循环单元(Gated Recurrent Unit，GRU)]的出现，序列数据建模已成功用于多种应用中，例如序列分类、序列生成、语音到文本的转换等。

在序列分类中，目标是预测序列的类别，而在序列生成中，根据输入序列生成新的输出序列。本章将介绍如何使用不同的 RNN 模型来实现序列分类和生成，以及时间序列预测。

本章将介绍以下实战内容：

- 使用 RNN 实现情感分类；
- 使用 LSTM 实现生成文本；
- 使用 GRU 实现时间序列预测；
- 实现双向循环神经网络。

3.1 使用 RNN 实现情感分类

RNN 是一种特殊的人工神经网络，因为它能够对输入数据进行记忆。此功能使其非常适合处理序列数据的问题，例如，时间序列预测、语音识别、机器翻译以及音频和视频序列预测。在 RNN 中，数据以这样的方式遍历：在每个节点上，网络都从当前和之前的输入中学习，并随时间共享权重。这就像在每个步骤上执行相同的任务，只是用不同的输入来减少需要学习的参数总数。

例如，如果激活函数为 tanh，则递归神经元的权重为 W_{aa}，输入神经元的权重为 W_{ax}。可以写出时间为 t 的状态方程 h，如下所示：

$$h_t = \tanh(W_{aa}h_{t-1} + W_{ax}X_t)$$

每个输出点的梯度取决于当前和先前时间步幅。例如，要计算 $t=6$ 处的梯度，需要反

向传播前 5 个时间点的梯度并累加起来。这就是所谓的**时间反向传播**(**BackPropagation Through Time**,**BPTT**)。在时间反向传播过程中,在遍历训练集的同时,修改权值以减少误差。

RNN 可以通过不同的拓扑结构处理具有各种输入和输出类型的数据。主要类型有:

- **一对多**——一个输入可以映射到输出序列的多个节点,如图 3-1 所示。例如,音乐生成(以一个音符作为输入,逐步生成后续一段音符)。
- **多对一**——将输入序列映射到类别或数量预测,如图 3-2 所示。例如,情感分类(以一个文本序列作为输入,输出该文本的情感判断,比如,褒义的或贬义的)。
- **多对多**——输入序列映射到输出序列,如图 3-3 所示。例如,语言翻译。

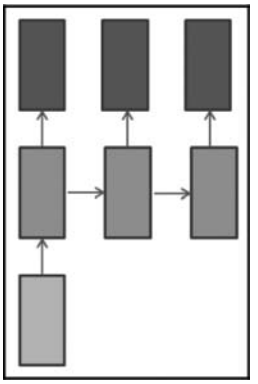


图 3-1　一对多模型拓扑结构

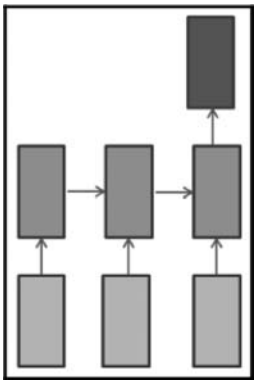


图 3-2　多对一模型拓扑结构

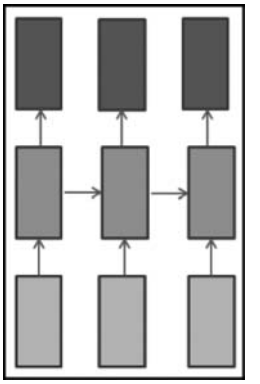


图 3-3　多对多模型拓扑结构

本实例将构建一个 RNN 模型,该模型将对电影评论进行情感分类。

3.1.1　准备工作

本例使用 IMDb 数据集,该数据集包含电影评论及对应情感标签信息。可以从 Keras 库中导入该数据集。这些评论经过预处理并编码为单词索引序列。这些单词按它们在数据集中出现的次数进行索引;例如,单词索引 8 指的是数据中第 8 个最常见的单词。

首先导入 Keras 库和 IMDb 数据集:

```
library(keras)
imdb <- dataset_imdb(num_words = 1000)
```

数据集划分为训练集和验证集:

```
train_x <- imdb$train$x
train_y <- imdb$train$y
test_x <- imdb$test$x
test_y <- imdb$test$y
```

查看训练集和验证集中的样本数量:

```
# number of samples in train and test set
cat(length(train_x), 'train sequences\n')
cat(length(test_x), 'test sequences')
```

从图 3-4 可以看到,训练集和验证集中各有 25 000 条用户评论。

查看训练集的数据的结构信息:

```
str(train_x)
```

```
25000 train sequences
25000 test sequences
```

图 3-4 查看训练集和验证集样本数

训练集中各条评论编码后的数据如图 3-5 所示。

```
List of 25000
 $ : int [1:218] 1 14 22 16 43 530 973 2 2 65 ...
 $ : int [1:189] 1 194 2 194 2 78 228 5 6 2 ...
 $ : int [1:141] 1 14 47 8 30 31 7 4 249 108 ...
 $ : int [1:550] 1 4 2 2 33 2 4 2 432 111 ...
 $ : int [1:147] 1 249 2 7 61 113 10 10 13 2 ...
 $ : int [1:43] 1 778 128 74 12 630 163 15 4 2 ...
 $ : int [1:123] 1 2 365 2 5 2 354 11 14 2 ...
 $ : int [1:562] 1 4 2 716 4 65 7 4 689 2 ...
 $ : int [1:233] 1 43 188 46 5 566 264 51 6 530 ...
 $ : int [1:130] 1 14 20 47 111 439 2 19 12 15 ...
 $ : int [1:450] 1 785 189 438 47 110 142 7 6 2 ...
 $ : int [1:99] 1 54 13 2 14 20 13 69 55 364 ...
 $ : int [1:117] 1 13 119 954 189 2 13 92 459 48 ...
 $ : int [1:238] 1 259 37 100 169 2 2 11 14 418 ...
 $ : int [1:109] 1 503 20 33 118 481 302 26 184 52 ...
 $ : int [1:129] 1 6 964 437 7 58 43 2 11 6 ...
 $ : int [1:163] 1 2 2 11 4 2 9 4 2 4 ...
 $ : int [1:752] 1 33 4 2 7 4 2 194 2 2 ...
 $ : int [1:212] 1 13 28 64 69 4 2 7 319 14 ...
 $ : int [1:177] 1 2 26 9 6 2 731 939 44 6 ...
 $ : int [1:129] 1 617 11 2 17 2 14 966 78 20 ...
 $ : int [1:140] 1 466 49 2 204 2 40 4 2 732 ...
 $ : int [1:256] 1 13 784 886 857 15 135 142 40 2 ...
```

图 3-5 训练集中各条评论编码后的数据

查看训练集中各条评论的情感标签信息:

```
str(train_y)
```

训练集中因变量的描述信息如图 3-6 所示。

从以上输出可以看到,训练集是评论和情感标签的列表。查看第一个评论及其中的单词数:

```
int [1:25000] 1 0 0 1 0 0 1 0 1 0 ...
```

图 3-6 训练集中因变量的描述信息

```
train_x[[1]]
cat("Number of words in the first review is",length(train_x[[1]]))
```

图 3-7 以编码形式显示了第一条评论。

```
1  14  22  16  43  530  973  2  2  65  458  2  66  2  4  173  36  256  5  25  100  43  838  112  50  670  2  9  35  480  284  5
150  4  172  112  167  2  336  385  39  4  172  2  2  17  546  38  13  447  4  192  50  16  6  147  2  19  14  22  4  2  2  469  4
22  71  87  12  16  43  530  38  76  15  13  2  4  22  17  515  17  12  16  626  18  2  5  62  386  12  8  316  8  106  5  4  2  2
16  480  66  2  33  4  130  12  16  38  619  5  25  124  51  36  135  48  25  2  33  6  22  12  215  28  77  52  5  14  407  16
82  2  8  4  107  117  2  15  256  4  2  7  2  5  723  36  71  43  530  476  26  400  317  46  7  4  2  2  13  104  88  4  381  15
297  98  32  2  56  26  141  6  194  2  18  4  226  22  21  134  476  26  480  5  144  30  2  18  51  36  28  224  92  25  104  4
226  65  16  38  2  88  12  16  283  5  16  2  113  103  32  15  16  2  19  178  32

Number of words in the first review is 218
```

图 3-7 以编码形式显示第一条评论

请注意，在导入数据集时，将 num_words 参数的值设置为 1000。这意味着在编码的评论中仅保留前千个常用单词。确认一下评论列表中的最大编码值：

```
cat("Maximum encoded value in train ",max(sapply(train_x, max)),"\n")
cat("Maximum encoded value in test ",max(sapply(test_x, max)))
```

执行前面的代码会输出训练集和验证集中的最大编码值。

3.1.2　操作步骤

至此已对 IMDb 数据集有所认识，下面对数据集做详细分析。

（1）导入 IMDb 数据集的单词词频索引：

```
word_index = dataset_imdb_word_index()
```

可以使用以下代码查看单词词频索引的前几条数据：

```
head(word_index)
```

从图 3-8 可以看到一个键值对列表，其中键是单词，值是单词在数据集中出现的次数。

查看单词索引中的单词数量：

```
length((word_index))
```

从图 3-9 可以看到单词索引中有 88 584 个单词。

（2）创建一个单词索引的键值对的反向列表。将使用这个列表来解码 IMDb 数据集中的评论。

```
reverse_word_index <- names(word_index)
names(reverse_word_index) <- word_index
head(reverse_word_index)
```

图 3-10 显示了反向的词索引列表，它是一个键-值对列表，其中键是整型索引，值是相关联的单词。

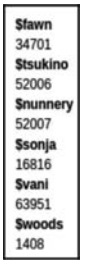


图 3-8　单词和单词出现次数列表

图 3-9　单词索引列表中的单词数

图 3-10　反向的词索引列表

(3) 解码第一个用户评论。注意,单词编码的偏移量为3,因为0、1、2分别预留作填充、文本序列的开始和词汇表外的单词。

```
decoded_review <- sapply(train_x[[1]], function(index) {
 word <- if (index >= 3) reverse_word_index[[as.character(index - 3)]]
 if (!is.null(word)) word else "?"})
cat(decoded_review)
```

第一条用户评论的解码文本如图3-11所示。

```
? this film was just brilliant casting ? ? story direction ? really ? the part they played and you could just imagi
ne being there robert ? is an amazing actor and now the same being director ? father came from the same ? ? as myse
lf so i loved the fact there was a real ? with this film the ? ? throughout the film were great it was just brillia
nt so much that i ? the film as soon as it was released for ? and would recommend it to everyone to watch and the ?
? was amazing really ? at the end it was so sad and you know what they say if you ? at a film it must have been goo
d and this definitely was also ? to the two little ? that played the ? of ? and paul they were just brilliant child
ren are often left out of the ? ? i think because the stars that play them all ? up are such a big ? for the whole
film but these children are amazing and should be ? for what they have done don't you think the whole story was so
? because it was true and was ? life after all that was ? with us all
```

图3-11 第一条用户评论的解码文本

(4) 填充/截取所有的文本序列,使它们的长度相同:

```
train_x <- pad_sequences(train_x, maxlen = 80)
test_x <- pad_sequences(test_x, maxlen = 80)
cat('x_train shape:', dim(train_x), '\n') cat('x_test shape:', dim(test_x), '\n')
```

如图3-12所示,所有的文本序列都被填充/截取至长度为80个索引。

```
x_train shape: 25000 80
x_test shape: 25000 80
```

图3-12 填充/截取所有的文本序列至长度为80

查看填充后的第一条用户评论:

```
train_x[1,]
```

从图3-13可以看到第一条用户评论只剩下80个索引(原来有128个索引)。

```
15  256  4  2  7  2  5  723  36  71  43  530  476  26  400  317  46  7  4  2  2  13  104  88  4  381  15  297  98  32  2  56  26
141  6  194  2  18  4  226  22  21  134  476  26  480  5  144  30  2  18  51  36  28  224  92  25  104  4  226  65  16  38  2  88
12  16  283  5  16  2  113  103  32  15  16  2  19  178  32
```

图3-13 第一条用户评论截取至长度为80

(5) 构建情感分类模型,并查看模型摘要信息:

```
model <- keras_model_sequential()
model %>%
 layer_embedding(input_dim = 1000, output_dim = 128) %>%
 layer_simple_rnn(units = 32) %>%
 layer_dense(units = 1, activation = 'sigmoid')
summary(model)
```

模型摘要信息如图 3-14 所示。

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 128)         256000
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 32)                5152
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 261,185
Trainable params: 261,185
Non-trainable params: 0
_________________________________________________________________
```

图 3-14　模型摘要信息

(6) 编译和训练模型：

```
# 编译模型
model %>% compile(
 loss = 'binary_crossentropy',
 optimizer = 'adam',
 metrics = c('accuracy') )
# 训练模型
model %>% fit(
 train_x,train_y,
 batch_size = 32,
 epochs = 10,
validation_split = .2
)
```

(7) 在验证集上评估模型性能，并输出评价指标：

```
scores <- model %>% evaluate(
 test_x, test_y,
 batch_size = 32
)
cat('Test score:', scores[[1]],'\n')
cat('Test accuracy', scores[[2]])
```

```
Test score: 0.8564493
Test accuracy 0.71648
```

图 3-15　模型在验证集上的损失函数值和准确率

模型在验证集上的性能指标如图 3-15 所示。

模型在验证集上达到约 71.6%的准确率。

3.1.3　原理解析

本例使用了 Keras 库中内置的 IMDb 评论数据集。首先加载训练集和验证集并查看数据的结构信息，可以看到用户评论被映射为一个特定的整数值序列，每个整数值对应单词索引列表中的一个特定单词。这个单词索引列表收录了丰富的词汇，根据语料库中每个单词的使用频率作为索引进行排列。由此，可以看到单词索引列表是一个键-值对列表，键表示

单词,值表示单词在字典中的索引。为了丢弃不经常使用的单词,实例中只使用单词索引列表中前 1000 个单词索引,也就是说,只保留了训练数据集中最常用的 1000 个单词,而忽略了其余的单词。认识和预处理完数据集后,开始具体的操作。

在 3.1.2 节的步骤(1)中导入了 IMDb 数据集的单词索引。在这个单词索引中,数据中的词是根据其数据集的频率进行编码和索引的。步骤(2)创建了单词索引的键-值对的反向列表,用于将句子从一系列编码的整数解码回其原始文本。步骤(3)展示了如何对一条编码后的用户评论进行解码。

3.1.2 节的步骤(4)预处理数据,以便可以将其输入到模型中。由于不能直接向模型传递任意长度列表,所以将用户评论序列转换成一致尺寸的张量。为了使所有评论序列的长度一致,可以采用以下两种方法之一:

- **独热编码(one-hot encoding)**——将序列转换成相同长度的张量。矩阵的尺寸是单词数×用户评论数,这种方法计算量很大。
- **填充评论(pad the reviews)**——填充或截取所有的用户评论序列,使评论序列具有相同的长度。这个操作将创建尺寸为序列最大长度(max_length)×用户评论数的整数张量。max_length 参数用于限制所有用户评论中保留的最大单词数。

由于第二种方法的时间和空间复杂度比第一种方法小,因此本实例选择了第二种方法,将评论序列填充/截取为最大长度为 80 的序列。

在 3.1.2 节的步骤(5)中,定义了一个顺序 Keras 模型并配置了它的各层。第一层是嵌入层,用于从数据中生成单词序列的上下文,并提供相关特征的信息。在一个嵌入层中,单词用稠密向量表示。每个向量表示单词在向量空间中的投影,向量空间是从文本中学习而来,向量空间的维度由特定词决定。单词在向量空间中的位置被称为**词嵌入(embedding)**。在进行词嵌入时,每一条用户评论都用一个词向量来替代。例如,brilliant 这个词可以用一个向量表示,比如向量[0.32, 0.02, 0.48, 0.21, 0.56, 0.15]。处理大数据集时,词嵌入方法的计算效率同样很高,因为词嵌入方法相比独热编码降低了维数。在深度神经网络的训练过程中,对词嵌入向量进行更新,有助于在多维空间中识别相似词。词嵌入也反映了词语在语义上是如何相互关联的。例如,talking 和 talked 这两个词可以被认为是相互关联的,就像 swimming 与 swam 之间的联系一样。

词嵌入的例子如图 3-16 所示。

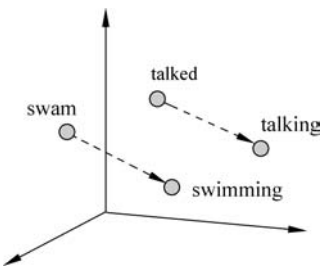


图 3-16 词嵌入示例

嵌入层通过以下 3 个参数来定义:

- **input_dim**——文本数据中单词索引列表的大小。在本实例中,文本数据是一个被编码为 0～999 的值的整数。因此,单词索引列表的大小是 1000 个单词;
- **output_dim**——词嵌入的向量空间的大小,本实例指定为 128;
- **input_length**——输入序列的长度,Keras 模型的任何输入层都有定义该变量。

在下一层中，定义了一个包含 32 个隐藏单元的简单 RNN 模型。如果 n 为输入维数，d 为 RNN 层中隐藏层神经元的个数，则要训练的参数个数可由下式给出：

$$((n+d)\times d)+d$$

最后一层与单个输出节点全相连。输出层神经元使用 sigmoid 激活函数，因为本实例是一个二元分类任务。在 3.1.2 节的步骤(6)中，编译模型，指定 binary_crossenropy 作为损失函数，因为处理的是二元分类，所以优化器采用 adam 算法。训练模型时预留 20%样本用于验证集。在最后一步中，在验证集上评估模型性能，并输出评价指标。

3.1.4　内容拓展

在前面各节已经学习了时间反向传播(BPTT)在 RNN 中的工作方式。在每次迭代中误差反向传播计算误差相对于权值的梯度。在反向传播过程中，越往输入层方面，梯度变得越小，从而使这些层次的神经元学习非常缓慢。对于精确的模型，对越靠近输入层的层进行准确的训练是至关重要的，因为这些层负责从输入中学习简单的模式，并将相关信息相应地传递给下层。当训练具有更多层依赖性的大型网络时，RNN 经常面临**梯度消失问题**(**vanishing gradient problem**)，它会使得网络收敛速度很慢。这也意味着，网络训练停止后准确率不高。通常建议使用 ReLU 激活函数来避免大型网络中的梯度消失问题。处理这个问题的另一种常见方法是使用**长短时记忆**(**Long Short-Term Memory，LSTM**)模型。下一个实战案例中将讨论 LSTM。

RNN 遇到的另一个挑战是**梯度爆炸问题**(**exploding gradient problem**)。在这种情况下，可以看到很大的梯度值，这反过来使模型学习速度过快和不准确。在某些情况下，由于计算中的数值溢出，梯度也可能变成 NaN。当这种情况发生时，在训练时，网络中的权值会在更短的时间内大幅增加。防止此问题的最常用的补救方法是**梯度裁剪**(**gradient clipping**)，它防止梯度逐渐增加而超过指定的阈值。

3.1.5　参考阅读

要了解更多关于递归神经网络正则化的知识，请阅读论文 https://arxiv.org/pdf/1409.2329.pdf.

3.2　使用 LSTM 实现文本生成

循环神经网络无法建立起和较早时间步信息的依赖关系，当神经网络层数很多并且各层之间存在长程依赖时这个问题更加突出。**长短时记忆网络**(**long-short term memory network**)是循环神经网络的一种变体，它能够改善 RNN 的**长程依赖**(**long-term dependency**)问题，被广泛应用于深度学习中以解决 RNN 面临的**梯度消失问题**(**vanishing gradient problem**)。LSTM 通过**一种门控机制**(**gating mechanism**)来解决梯度消失，并且能够向状态

单元中删除或添加信息。这种状态单元的状态受到“门”的严格控制,“门”控制着通过状态单元的信息。LSTM 有 3 种门:输入门、输出门和遗忘门。

- **遗忘门**控制想要传递多少来自前一个状态的信息到下一个状态单元;
- **输入门**控制将多少新计算的状态信息传递给当前输入 x_t 的后续状态;
- **输出门**控制将多少内部状态信息传递给下一个状态。

LSTM 网络结构如图 3-17 所示。

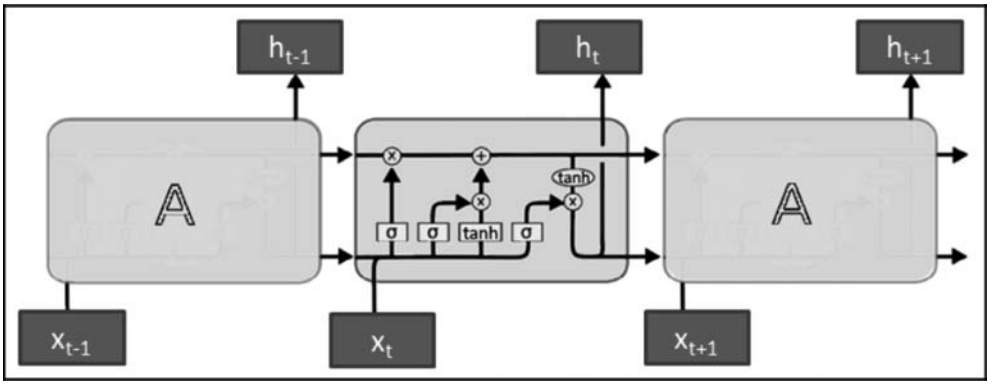


图 3-17 LSTM 网络结构

本实例实现一个用于序列预测的 LSTM 模型(本例是多对一模型)。该模型将根据之前的单词序列预测单词的出现情况,即所谓的**文本生成**(**text generation**)。

3.2.1 准备工作

本实例使用童谣《杰克和吉尔》作为源文本来构建语言模型。实例中创建一个包含押韵的文本文件,并将其保存在当前工作目录中。该语言模型以两个单词作为输入来预测下一个单词。

首先导入所需的库和读取所需的文本文件。

```
library(keras)
library(readr)
library(stringr)
data <- read_file("data/rhyme.txt") %>% str_to_lower()
```

在自然语言处理中,将数据称为语料库,语料库是大量文本的集合。查看本实例的语料库:

```
data
```

语料库中的文本如图 3-18 所示。

将使用图 3-18 所示的文本来生成整数序列。

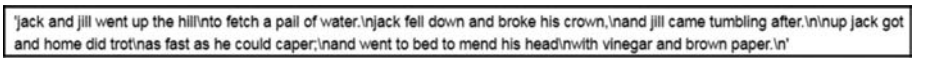


图 3-18　语料库中的文本示例

3.2.2　操作步骤

到目前为止，已经在 R 环境中导入了一个语料库。为构建语言模型，需要将语料库转换成一个整数序列。第一步先做数据预处理。

(1) 创建分词器(tokenizer)，用于将文本转换为整数序列：

```
tokenizer = text_tokenizer(num_words = 35,char_level = F)
tokenizer %>% fit_text_tokenizer(data)
```

查看语料库中的单词数量：

```
cat("Number of unique words", length(tokenizer$word_index))
```

语料库中有 37 个不同的单词。

使用以下命令查看单词索引表前几条记录：

```
head(tokenizer$word_index)
```

使用前面创建的分词器将语料库转换为整数序列：

```
text_seqs <- texts_to_sequences(tokenizer, data)
str(text_seqs)
```

生成的整数序列的形式如图 3-19 所示。

可以看到，texts_to_sequences()函数返回了一个整数列表。将它转换成一个向量并输出其长度。

```
text_seqs <- text_seqs[[1]]
length(text_seqs)
```

图 3-20 所示的输出结果表明该语料库的文本包含 48 个单词。

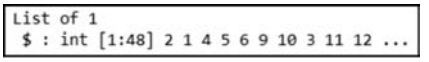


图 3-19　生成的整数序列的形式

图 3-20　语料库包含的单词数

(2) 将文本序列转换为输入(特征)序列和输出(标签)序列，其中输入是两个连续单词的序列，而输出是序列中出现的下一个单词。

```
input_sequence_length <- 2
feature <- matrix(ncol = input_sequence_length)
label <- matrix(ncol = 1)
for(i in seq(input_sequence_length, length(text_seqs))){
 if(i >= length(text_seqs)){
```

```
 break()
 }
 start_idx <- (i - input_sequence_length) + 1
 end_idx <- i + 1
 new_seq <- text_seqs[start_idx:end_idx]
 feature <- rbind(feature,new_seq[1:input_sequence_length])
 label <- rbind(label,new_seq[input_sequence_length + 1])
}
feature <- feature[ - 1,]
label <- label[ - 1,]
paste("Feature")
head(feature)
```

图 3-21 显示了特征序列。

查看创建的标签序列：

```
paste("label")
head(label)
```

图 3-22 显示了前几个标签序列。

将标签采用独热编码：

```
label <- to_categorical(label,num_classes = tokenizer$num_words)
```

查看到特性和标签数据的维度：

```
cat("Shape of features",dim(feature),"\n")
cat("Shape of label",length(label))
```

图 3-23 显示了特性和标签序列的维度。

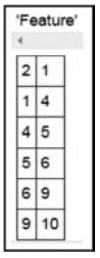


图 3-21 特征序列

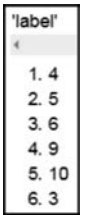


图 3-22 标签序列

图 3-23 特性和标签序列的维度

(3) 创建一个文本生成模型并输出模型的摘要信息：

```
model <- keras_model_sequential()
model %>%
 layer_embedding(input_dim = tokenizer$num_words,output_dim =
10,input_length = input_sequence_length) %>%
 layer_lstm(units = 50) %>%
 layer_dense(tokenizer$num_words) %>%
 layer_activation("softmax")
summary(model)
```

模型的摘要信息如图 3-24 所示。

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 2, 10)	350
lstm (LSTM)	(None, 50)	12200
dense (Dense)	(None, 35)	1785
activation (Activation)	(None, 35)	0

Total params: 14,335
Trainable params: 14,335
Non-trainable params: 0

图 3-24　模型的摘要信息

编译和训练模型：

```
# 编译模型
model %>% compile(
 loss = "categorical_crossentropy",
 optimizer = optimizer_rmsprop(lr = 0.001),
 metrics = c('accuracy') )
# 训练模型
model %>% fit(
 feature, label,
# batch_size = 128,
 epochs = 500
)
```

下面代码块实现了一个函数，按照语言模型生成一个序列：

```
generate_sequence <- function(model, tokenizer, input_length,
seed_text, predict_next_n_words){
 input_text <- seed_text
 for(i in seq(predict_next_n_words)){
 encoded <- texts_to_sequences(tokenizer,input_text)[[1]]
 encoded <- pad_sequences(sequences = list(encoded),maxlen =
input_length,padding = 'pre')
 yhat <- predict_classes(model,encoded, verbose = 0)
 next_word <- tokenizer$index_word[[as.character(yhat)]]
 input_text <- paste(input_text, next_word)
 }
 return(input_text) }
```

使用自定义函数 generate_sequence()从整数序列生成文本：

```
seed_1 = "Jack and"
cat("Text generated from seed 1: "
,generate_sequence(model,tokenizer,input_sequence_length,seed_1,11) ,"\n ")
seed_2 = "Jack fell"
cat("Text generated from seed 2:
```

```
",generate_sequence(model,tokenizer,input_sequence_length,seed_2,11
))
```

图 3-25 显示了模型从输入的文本中生成的文本。

```
Text generated from seed 1:  Jack and jill went up the hill to fetch a pail of water
 Text generated from seed 2:  Jack fell down and broke his crown and jill went up the hill
```

图 3-25 模型生成的文本

可以看出,模型在预测序列方面做得很好。

3.2.3 原理解析

要构建任何语言模型,都需要先将文本进行分词。分词是将文本分隔为一个个词语。默认情况下,Keras 分词器将语料库进行分隔生成一个单词列表,删除所有标点,将单词的字母转换为小写,并构建一个内部词汇表。由分词器生成的词汇表是一个单词索引列表,其中单词根据它们在数据集中的出现频率进行索引。在这个实例中,可以看到在童谣《杰克和吉尔》中,and 是最常见的单词,而 up 是第五常见的单词。语料库总共有 37 个单词。

在 3.2.2 节的步骤(1)中,将语料库转换为一个整数序列。请注意,text_tokenizer()的 num_words 参数根据词频定义了要保留的最大单词数。这意味着只有最前面的 n 个频繁词被保存在编码序列中。在步骤(2)中,从语料库生成特征列表和标签列表。

在 3.2.2 节的步骤(3)中,定义了 LSTM 神经网络。首先,对序列模型进行初始化,然后在模型中加入嵌入层。嵌入层将输入特征空间转化为一个 d 维的潜在特征,本实例中将其转换为 128 个潜在特征。接下来,添加一个有 50 个神经元的 LSTM 层。单词预测是一个分类问题,预测词汇表中的下一个单词。因此,添加了一个全连接层,神经元数量等于词汇表中单词的数量,采用 softmax 激活函数。

在 3.2.2 节的步骤(4)中,定义了一个函数,该函数将从给定的两个单词的初始集合生成文本,即模型从原来的前两个单词中预测下一个单词。在本实例中,第一个样本特征是“Jack and”,预测的单词是“jill”,从而创建了 3 个单词序列。在下一次迭代中,取句子的最后两个单词“and jill”,并预测下一个单词“went”。该函数继续生成文本,直到生成的单词数等于 predict_next_n_words 参数的值为止。

3.2.4 内容拓展

在开发自然语言处理应用程序时,从文本数据中构造有意义的特征。可以使用许多技术来构建这些特征,如计数向量化、二进制向量化、词频-逆向文本频率(Term Frequency-Inverse Document Frequency,TF-IDF)、词嵌入等。下面的代码块演示了如何使用 R 中的 Keras 库为各种自然语言处理应用程序构建 TF-IDF 特征矩阵:

```
texts_to_matrix(tokenizer, input, mode = c("tfidf"))
```

mode 参数还可以取值为 binary、count、freq。

3.2.5　参考阅读

- 要了解更多关于循环神经网络或长短时记忆网络中添加编码器-解码器网络的知识，请查阅论文 https://cs224d.stanford.edu/reports/Lambert.pdf。
- 要了解更多关于基于 Word2Vec 的神经网络的知识，请查阅文档 http://mccormickml.com/assets/word2vec/Alex_Minnaar_Word2Vec_Tutorial_Part_I_The_Skip-Gram_Model.pdf。

3.3　使用 GRU 实现时间序列预测

不同于 LSTM，**门控循环单元**（**Gate Recurrent Unit，GRU**）网络不使用记忆单元来控制信息流，可以直接利用所有隐状态。GRU 使用隐状态（短时记忆向量）来传输信息，而不是使用长时记忆向量（cell state）。GRU 通常比其他基于记忆的神经网络训练得更快，因为 GRU 模型需要相对较少的训练参数和更少的张量操作，并且可以在更少的数据下很好地工作。GRU 模型有两个控制门，称为**重置门**（**reset gate**）和**更新门**（**update gate**）。重置门用来决定如何将新的输入与先前的记忆相结合，而更新门决定从先前的状态保留多少信息。与 LSTM 相比，GRU 中的更新门与 LSTM 中的输入门＋遗忘门的作用相当，更新门控制当前状态需要从历史状态中保留多少信息。GRU 网络通过更新门合并短时记忆向量和长时记忆向量来简化模型。

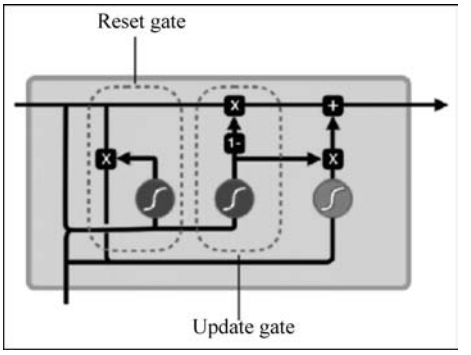


图 3-26　GRU 模型结构

GRU 模型结构如图 3-26 所示。

本实例使用 GRU 网络来预测洗发水的销售情况。

3.3.1　准备工作

在构建模型之前，先分析数据的趋势。

首先导入 Keras 库：

```
library(keras)
```

本实例使用洗发水的销售数据，可以从本书的 GitHub 存储库下载。该数据集包含 3 年期间洗发水的月销售额，由 36 行组成。原始数据集是由 Makridakis，Wheelwright，Hyndman (1998)提供的。

```
data = read.table("data/shampoo_sales.txt",sep = ',')
```

```
data <- data[-1,]
rownames(data) <- 1:nrow(data)
colnames(data) <- c("Year_Month","Sales")
head(data)
```

Year_Month	Sales
1-01	266.0
1-02	145.9
1-03	183.1
1-04	119.3
1-05	180.3
1-06	168.5

图 3-27 查看数据集中部分数据

数据集的部分数据如图 3-27 所示。

分析 Sales 列的数据趋势：

```
# 绘制折线图查看数据趋势
library(ggplot2)
q = ggplot(data = data, aes(x = Year_Month, y = Sales,group = 1)) +
geom_line()
q = q + theme(axis.text.x = element_text(angle = 90, hjust = 1))
q
```

数据的趋势如图 3-28 所示。

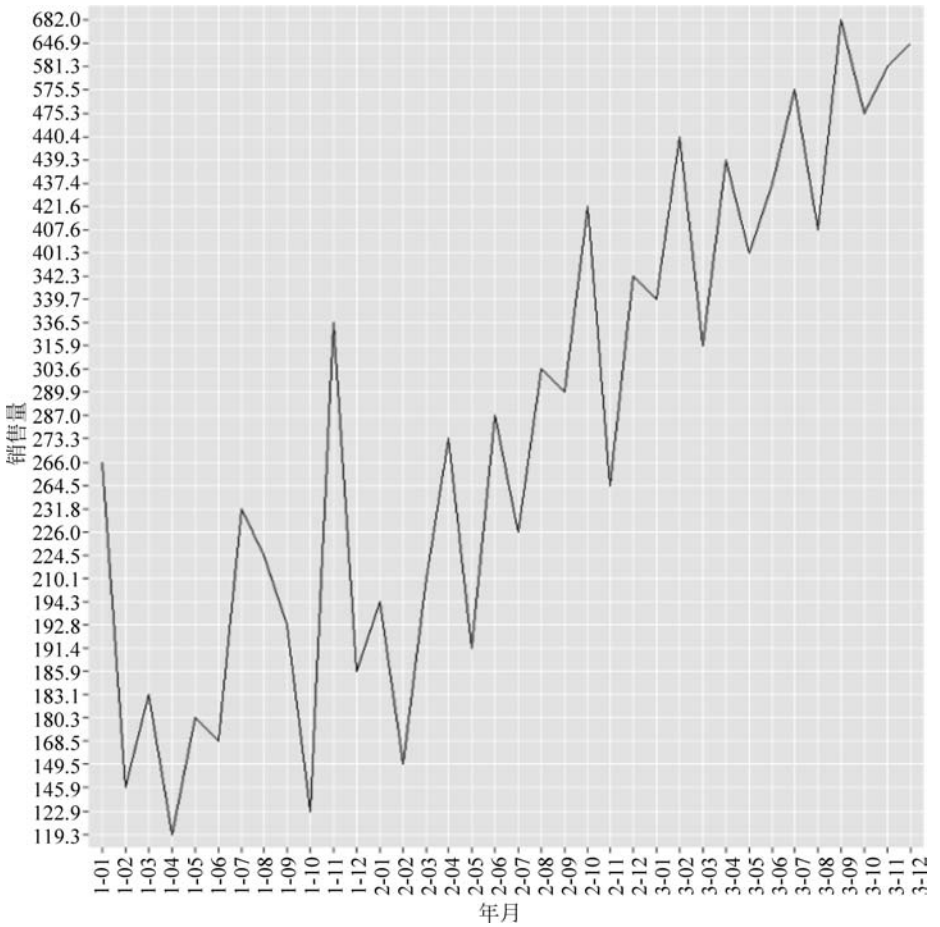


图 3-28 洗发水销售数据折线图

可以看到数据有上升趋势。

3.3.2　操作步骤

进入数据处理部分。

(1) 查看数据中 Sales 列的数据类型：

```
class(data$Sales)
```

注意，在数据中，Sales 列是 R 的因子数据类型(分类变量)。为了后续分析该数据，需要将它转换为数值型：

```
data$Sales <- as.numeric(as.character(data$Sales))
class(data$Sales)
```

代码执行后 Sales 列转为数值型。

(2) 为了实现时间序列预测，需要将数据转换为平稳序列。可以使用 diff()函数迭代计算序列数据的差分值(序列中的后项减前项)。将参数 differences 的值设为 1，表示差分的延迟为 1。

```
data_differenced = diff(data$Sales, differences = 1)
head(data_differenced)
```

差分运算后生成一部分数据如图 3-29 所示。

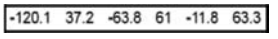


图 3-29　差分运算后生成的部分数据

(3) 创建一个用于监督学习的数据集，以便应用 GRU。将 data_differenced 序列中的各元素都往后移动 1 格作为输入，即将时刻(t－1)的值作为输入，将时刻 t 的值作为输出。

```
data_lagged = c(rep(NA, 1),
data_differenced[1:(length(data_differenced) - 1)])
data_preprocessed =
as.data.frame(cbind(data_lagged,data_differenced))
colnames(data_preprocessed) <- c( paste0('x - ', 1), 'x')
data_preprocessed[is.na(data_preprocessed)] <- 0
head(data_preprocessed)
```

为监督学习构造的数据集如图 3-30 所示。

x-1	x
0.0	-120.1
-120.1	37.2
37.2	-63.8
-63.8	61.0
61.0	-11.8
-11.8	63.3

图 3-30　构造的数据集

(4) 需要将数据分成训练集和验证集。在时间序列问题中，不能对数据进行随机抽样，因为数据的顺序很重要。因此，需要对数据进行分割，将序列的前 70%作为训练数据，剩余 30%作为测试数据。

```
N = nrow(data_preprocessed)
n = round(N * 0.7, digits = 0)
```

```
train = data_preprocessed[1:n, ]
test = data_preprocessed[(n + 1):N,]
print("Training data snapshot :")
head(train)
print("Testing data snapshot :")
head(test)
```

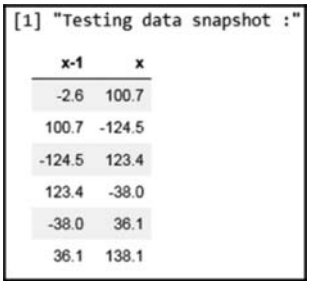
训练集的部分记录如图 3-31 所示。

验证集的部分记录如图 3-32 所示。

[1] "Training data snapshot :"

x-1	x
0.0	-120.1
-120.1	37.2
37.2	-63.8
-63.8	61.0
61.0	-11.8
-11.8	63.3

图 3-31 训练集的部分记录



[1] "Testing data snapshot :"

x-1	x
-2.6	100.7
100.7	-124.5
-124.5	123.4
123.4	-38.0
-38.0	36.1
36.1	138.1

图 3-32 验证集的部分记录

(5) 将数据归一化处理成适合于模型所选激活函数的范围。模型中使用 tanh 函数作为激活函数,tanh 函数的值域是(−1,1),这里采用最小-最大归一化方法来处理数据。

```
scaling_data = function(train, test, feature_range = c(0, 1)) {
 x = train
 fr_min = feature_range[1]
 fr_max = feature_range[2]
 std_train = ((x - min(x) ) / (max(x) - min(x) ))
 std_test = ((test - min(x) ) / (max(x) - min(x) ))
 scaled_train = std_train * (fr_max - fr_min) + fr_min
 scaled_test = std_test * (fr_max - fr_min) + fr_min
 return( list(scaled_train = as.vector(scaled_train), scaled_test =
as.vector(scaled_test) ,scaler = c(min = min(x), max = max(x))) )
}
Scaled = scaling_data(train, test, c( - 1, 1))
y_train = Scaled$scaled_train[, 2]
x_train = Scaled$scaled_train[, 1]
y_test = Scaled$scaled_test[, 2]
x_test = Scaled$scaled_test[, 1]
```

编写一个函数来将模型预测值还原为原始变量尺度。在得出最终预测值时要使用这个函数。

```
##反向转换
invert_scaling = function(scaled, scaler, feature_range = c(0, 1)){
 min = scaler[1]
```

```
 max = scaler[2]
 t = length(scaled)
 mins = feature_range[1]
 maxs = feature_range[2]
 inverted_dfs = numeric(t)
 for( i in 1:t){
 X = (scaled[i]- mins)/(maxs - mins)
 rawValues = X *(max - min) + min
 inverted_dfs[i] <- rawValues
 }
 return(inverted_dfs)
}
```

(6) 定义模型并配置层。将数据重构为 3D 格式,以便将其输入模型。

```
# Reshaping the input to 3-dimensional
dim(x_train) <- c(length(x_train), 1, 1)
# specify required arguments
batch_size = 1
units = 1
model <- keras_model_sequential()
model %>%
 layer_gru(units, batch_input_shape = c(batch_size,
dim(x_train)[2], dim(x_train)[3]), stateful = TRUE) %>%
 layer_dense(units = 1)
```

查看模型摘要信息:

```
summary(model)
```

模型摘要信息如图 3-33 所示。

```
_____________________________________________________________________________
Layer (type)                 Output Shape              Param #
=============================================================================
gru_1 (GRU)                  (1, 1)                    9
_____________________________________________________________________________
dense_1 (Dense)              (1, 1)                    2
=============================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
_____________________________________________________________________________
```

图 3-33　模型的摘要信息

接着,编译模型:

```
model %>% compile(
 loss = 'mean_squared_error',
 optimizer = optimizer_adam( lr = 0.01, decay = 1e-6 ),
 metrics = c('accuracy')
)
```

(7) 每次迭代,模型拟合一次训练数据并重置模型状态,模型迭代 50 步。

```
for(i in 1:50 ){
 model %>% fit(x_train, y_train, epochs = 1, batch_size = batch_size,
verbose = 1, shuffle = FALSE)
 model %>% reset_states()
}
```

(8) 对验证集进行预测,并使用 inverse_scaling 函数将预测值变换为原数据的尺度。

```
scaler = Scaled$scaler
predictions = vector()
for(i in 1:length(x_test)){
 X = x_test[i]
 dim(X) = c(1,1,1)
 yhat = model %>% predict(X, batch_size = batch_size)
 # invert scaling
 yhat = invert_scaling(yhat, scaler, c(-1, 1))
 # invert differencing
 yhat = yhat + data$Sales[(n + i)]
 # store
 predictions[i] <- yhat
}
```

查看验证集的预测结果:

```
Predictions
```

验证集的预测值如图 3-34 所示。

348.112441666424	359.491248017549	379.598579031229	354.127689468861	413.616816712916	423.328302359581	455.889744898677
461.995177252591	513.082208752632	505.667284664512	502.387261849642			

图 3-34 验证集的预测值

从测试数据的预测值,可以发现模型的预测效果很好。

3.3.3 原理解析

在 3.3.2 节的步骤(1)中查看数据集 Sales 列的数据类型,该列是模型要预测的列。将 Sales 列的数据类型转换为数值型。步骤(2)将输入数据转换为平稳序列,通过一阶差分操作可以消除序列中的随机性趋势(随机波动)。由图 3-28 可以看出输入数据有递增趋势。在时间序列预测中,建议在建模前去除随机性趋势因素。随机性趋势因素可以在以后添加回预测值中,这样就可以在原始数据尺度中得到预测值。在实例中,通过对数据进行一阶差分来消除随机性趋势,也就是说,用当前的观测值减去前一个观测值。

在使用 LSTM 和 GRU 等算法时,需要提供监督学习形式的训练数据,也就是说,以预测变量(自变量)和目标变量(因变量)的形式。在时间序列问题中,对于任意滞后 k 步的

数据序列，时间$(t-k)$值作为时间t值的输入。在本实例中，k等于1，所以在3.3.2节的步骤(3)中，通过将当前数据右移一格创建了一个滞后数据集。通过这样做，在数据中看到了$X=t-1$和$Y=t$的模式。创建的滞后数据序列作为预测变量。

在3.3.2节的步骤(4)中，将数据分成训练集和验证集。随机抽样不能保证时间序列数据中观测结果的顺序。因此，在保持观察顺序不变的情况下切分数据。将前70%的数据作为训练集，其余30%作为验证集。n表示分切点，是训练集的最后一个样本序号，$n+1$表示测试数据的起始序号。步骤(5)对数据做归一化处理。GRU模型要求训练数据在网络使用的激活函数的值域范围内。由于本例使用tanh作为激活函数，值域在(-1,1)范围内，因此，将训练集和验证集规范化为(-1,1)范围内。使用了**最小-最大归一化方法(min-max scaling)**，计算公式如下：

$$z_i=\frac{x_i-x_{\min}}{x_{\max}-x_{\min}}$$

在训练集上求得$x_{\max}$和$x_{\min}$，在验证集进行归一化时用同样的系数。这样做是为了避免测试数据集的最大值和最小值与训练集有差异造成计算结果的偏差。这就是为什么这里使用训练数据的最大值和最小值作为最小-最大归一化公式中的系数来对训练集和验证集以及预测值进行缩放。实例中还创建了一个名为invert_scaling的函数来对缩放后的值进行反向缩放，并将预测值映射回原数据尺度。

GRU模型输入的数据格式为[batch_size, timesteps, features]。batch_size定义了每次迭代向模型输入的批量样本数。timesteps表示模型进行预测时需要读入多少个历史数据(即滑动窗口的尺寸)。在本例中，将其设置为1。features参数表示使用的预测器的数量，在本例中为1。在3.3.2节的步骤(6)中，将输入数据重构为模型所需格式，并将其输入GRU层。注意，实例中指定了参数stateful=TRUE(有状态设置)，则批次中索引i处的每个样本的最后状态将用作后续批次中索引i的样本的初始状态。

假定不同连续批次的样本之间存在一一对应的映射。units参数表示输出空间的维数。因为本例处理的是预测连续值，取units=1。定义模型参数后，编译模型，并指定mean_squared_error作为损失函数，adam作为优化器，学习率为0.01。使用准确率作为模型的评价指标。接着查看模型的概要信息。

定义以下符号表示：

- f为前馈神经网络(FeedForward Neural Networks，FFNN)的神经元个数(在GRU中为3)；
- h是隐藏单元的大小；
- i是输入数据的尺寸。

由于每个FFNN有$h(h+i)+h$个参数，可以计算出GRU中要训练的参数个数是：

$$\text{num\_params}=f\times[h(h+i)+h];\quad \text{GRU的}\ f=3$$

在3.3.2节的步骤(7)中，每次迭代，模型拟合训练数据。shuffle参数的值指定为

false,以避免在构建模型时对训练数据进行的随机洗牌,这是因为样本之间是时间相依的。通过 reset_states()函数在每次迭代后重置网络状态,这是因为在步骤(6)中 GRU 模型定义了 stateful=true,因此需要在每次迭代后重置 LSTM 的状态,以便下一次迭代从新的状态开始训练。在最后一步中,预测测试数据集的值。为了将预测值缩放回原始数据的尺度,使用了步骤(5)中定义的 inverse_scaling 函数。

3.3.4 内容拓展

在处理大型数据集时,经常会在训练深度学习模型时耗尽内存。R 中的 Keras 库提供了各种实用的生成器函数,这些函数在训练过程中动态生成批量训练数据。Keras 还提供了一个用于创建批量时间序列数据的函数。下面的代码创建了一个监督学习训练集,类似于 3.3.2 节中创建的数据集。使用生成器的程序如下:

```
# 导入所需的库
library(reticulate)
library(keras)
# 生成序列数据(1,2,3,4,5,6,7,8,9,10)
data = seq(from = 1,to = 10)
# 定义时间序列生成器
gen = timeseries_generator(data = data,targets = data,length = 1,batch_size= 5)
# 输出第一个批次样本
iter_next(as_iterator(gen))
```

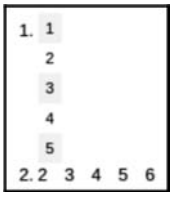


图 3-35 生成器生成的第一批数据

图 3-35 显示了生成器的第一批数据。可以看到 generator 对象生成了一个包含两个序列数据的列表;第一个序列为特征向量(自变量),第二个序列为对应的标签向量(因变量)。

下面的代码为时间序列数据实现了一个自定义生成器。lookback 参数可以非常便捷地控制使用多少历史值来预测未来的值或序列。future_steps 定义要预测的未来时间步数。

```
generator <- function (data,lookback = 3 ,future_steps = 3,batch_size = 3 ){
 new_data = data
 for(i in seq(1,3)){
 data_lagged = c(rep(NA, i), data[1:(length(data) - i)])
 new_data = cbind(data_lagged,new_data)
 }
 targets = new_data[future_steps:length(data),(ncol(new_data) - (future_steps - 1)):ncol
(new_data)]
 gen = timeseries_generator(data = data[1:(length(data) - (future_steps - 1))],targets =
targets,length = lookback,batch_size = batch_size) }
cat("First batch of generator:")
iter_next(as_iterator(generator(data = data,lookback = 3,future_steps = 2)))
```

图 3-36 显示了自定义生成器生成的第一批样本。

由图 3-36 可以看到，在列表 1 中，按参数 lookback＝3 生成特征序列(参数 batch_size＝3，因此列表 1 中有 3 行，代表 3 个特征序列；参数 lookback＝3，因此每序列中有 3 个元素)；在列表 2 中，按参数 future_steps＝2，生成列表 1 中对应 3 个序列预测两步的输出，比如，输入(1，2，3)，预测输出(4，5)。

```
First batch of generator:
1. 1 2 3
   2 3 4
   3 4 5
2. 4 5
   5 6
   6 7
```

图 3-36　生成器生成的第一批样本

3.3.5　参考阅读

- 要了解如何利用 LSTM 处理具有季节因素的时间序列数据，请查阅论文 https://arxiv.org/pdf/1909.04293.pdf。

3.4　实现双向循环神经网络

双向循环神经网络(Bidirectional Recurrent Neural Networks，Bi-RNN)是 RNN 的一种变体，它将输入数据按时间顺序和时间逆序输入到两个网络中。可以使用各种归并模式对顺序和逆序两个网络的输出的每个时间步进行归并，归并模式有求和、连接、乘法和平均。双向循环神经网络主要用于解决诸如整个语句或文本的语意与整个文本序列相关联，而不仅仅是与部分相邻上下文相关联的问题。Bi-RNN 训练要借助于很长的梯度链，训练代价很高。图 3-37 是 Bi-RNN 的结构图。

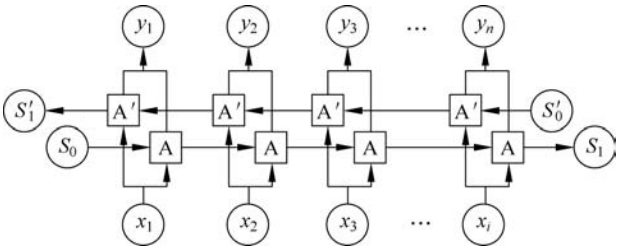


图 3-37　Bi-RNN 的结构图

本实例将实现一种 Bi-RNN 模型用于 IMDb 评论的情感分类。

3.4.1　操作步骤

本节使用 IMDb 评论数据集。数据预处理步骤与 3.1 节的操作相同。因此这里跳过数据预处理，直接进入模型构建部分。

(1) 创建序贯模型对象：

```
model <- keras_model_sequential()
```

(2) 添加一些神经网络层到模型中,并输出模型的摘要信息:

```
model %>%
 layer_embedding(input_dim = 2000, output_dim = 128) %>%
 bidirectional(layer_simple_rnn(units = 32),merge_mode = "concat")
%>%
 layer_dense(units = 1, activation = 'sigmoid')
summary(model)
```

模型的概要信息如图 3-38 所示。

```
Layer (type)                 Output Shape              Param #
================================================================
embedding (Embedding)        (None, None, 128)         256000
________________________________________________________________
bidirectional (Bidirectional) (None, 64)               10304
________________________________________________________________
dense (Dense)                (None, 1)                 65
================================================================
Total params: 266,369
Trainable params: 266,369
Non-trainable params: 0
________________________________________________________________
```

图 3-38 模型的摘要信息

(3) 编译和训练模型:

```
# 编译模型
model %>% compile(
 loss = "binary_crossentropy",
 optimizer = "adam",
 metrics = c("accuracy") )
# 训练模型
model %>% fit(
 train_x,train_y,
 batch_size = 32,
 epochs = 10,
 validation_split = .2
)
```

(4) 评估模型性能并输出性能指标:

```
scores <- model %>% evaluate(
 test_x, test_y,
 batch_size = 32
)
cat('Test score:', scores[[1]],'\n')
cat('Test accuracy', scores[[2]])
```

图 3-39 显示了模型在验证集上的性能指标。

模型正确率约为 75.7%。

```
Test score: 1.067133
Test accuracy 0.75688
```

图 3-39 模型在验证集上的损失函数值和准确率

3.4.2　原理解析

在建模前，需要先准备数据。想了解更多关于数据预处理部分的知识，可以参考 3.1 节的内容。

在 3.4.1 节的步骤(1)中实例化了一个 Keras 序贯模型。步骤(2)向序贯模型添加神经网络层。首先加入嵌入层，对输入特征空间进行降维处理。然后，在模型中添加一个双向循环神经网络层(参数 merge_mode＝"concat")。归并模式定义了如何组合顺序和逆序网络的输出，归并模式有求和、乘法、平均和无操作。最后，添加了一个带有一个隐藏层的全连接网络层，并使用 sigmoid 作为激活函数。

在 3.4.1 节的步骤(3)中，使用二元交叉熵(binary_crossenropy)作为损失函数来编译模型，因为本例是解决一个二元分类问题。模型使用了 adam 优化器，在训练数据集上训练模型。在步骤(4)中，评估模型在验证集上的准确率，评价模型在验证集上的性能。

3.4.3　内容拓展

尽管双向循环神经网络是一种最先进的技术，但在使用它们时存在一些限制。由于双向循环神经网络的作用方向有顺序和逆序两个方向，所以它们的梯度链很长，造成训练速度非常慢。此外，向循环神经网络也只用于非常特定的应用领域，如填补缺失的单词、机器翻译等等。该算法的另一个主要问题是，如果机器的内存受限，训练很难进行。