9.0 前言

9.0.1 课程目标

本课程的目标是学习怎么样开发Encoder-Decoder LSTM模型。完成本课程之后，你将会学习到：

- Encoder-Decoder LSTM的结构以及怎么样在Keras中实现它；
- 加法序列到序列的预测问题；
- 怎么样开发一个Encoder-Decoder LSTM模型用来解决加法seq2seq预测问题。

9.1 课程概览

本课程被分为7个部分，它们是：

1. Encoder-Decoder LSTM；
2. 加法预测问题；
3. 定义并编译模型；
4. 拟合模型；
5. 评估模型；
6. 用模型做预测；
7. 完成例子

让我们开始吧！

9.2 Encoder-Decoder LSTM模型

9.2.1 序列到序列预测问题

序列预测问题通常涉及预测真实序列中的下一个值或者输出输入序列的类标签。这通常被构造为一个输入时间步长序列到一个输出时间步长（例如，one-to-one）或者多个输入时间步长到一个输出时间步长（many-to-many）类型的序列预测问题。

有一种更具挑战性的序列预测问题，它以序列作为输入，需要序列预测作为输出。这些被称为序列到序列预测问题，或者简称为seq2seq问题。使这些问题具有挑战性的一个建模问题是输入和输出序列的长度可能变化。由于存在多个输入时间步长和多个输出时间步长，这种形式的问题被称为many-to-many序列预测问题。

9.2.2 结构

seq2seq预测问题的一种被证明是非常有效的方法被称为Encoder-Decoder LSTM。该体系结构包括两个模型：一个用于读取输入序列并将其编码成一个固定长度的向量，另一个用于解码固定长度的向量并输出预测序列。模型的使用相应地给出了该体系结构的名字——Encoder-Decoder LSTM，专门针对seq2seq问题而设计。

> ... RNN Encoder-Decoder由两个循环神经元网络（RNN）所组成，它们作为编码和解码对存在。编码器可将可变长度的源序列映射到一个固定长度的向量，同时解码器将矢量使用回可变长度的目标序列。
>
> — Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014

Encoder-Decoder LSTM是为处理自然语言处理问题而开发的，它显示了-of-the-art的性能，特别是在文本翻译领域称为统计机器翻译。这种体系结构的创新是在模型的最核心的部分使用了固定大小的内部表示，这里输入序列被读取并且输出序列从中被读取。由于这个原因，该方法被称为序列嵌入。

在英语与法语翻译的体系结构的一个应用中，编码的英语短语的内部表示被可视化。输出的图像解释了翻译任务中短语管理的一个定性的有意义的学习结构。

> 提出的RNN Encoder-Decoder自然地生成一个短语的连续空间表示。[...]从可视化角度，很明显地RNN Encoder-Decoder捕获语义和句法结构的短语。
>
> — Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014.

在翻译任务上，该模型在输入顺序颠倒时更有效。此外，即使在很长的输入序列上，该模型也被证明是有效的。

> 我们能够很好地完成长句，因为我们颠倒了原来句子的词序，而不是训练和测试集中的目标句子。通过这样做，我们引入了许多短期的依赖关系，是的优化问题变得简单多了。...源句倒换的简单技巧是这项工作的主要技术贡献之一。
>
> — Sequence to Sequence Learning with Neural Networks, 2014.

这种方法也被用于图像输入，其中卷积神经网络被用作输入图像上的特征提取器，然后由解码器LSTM读取。

> ...我们建议遵循这个优雅的配方，有一个深度卷积神经网络（CNN）来取代encoder RNN。[...]使用CNN作为图像编码器是很自然的，首先对图像分类任务进行预训练，最后使用隐藏层作为RNN解码器输出句子的输入。
>
> — Show and Tell: A Neural Image Caption Generator, 2014.

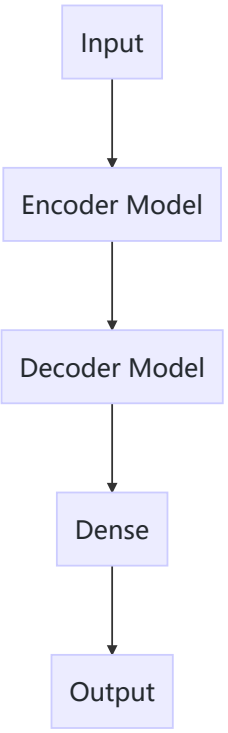

图 9.1 Encoder-decoder LSTM结构

9.2.3 应用

下面的列表突出了Encoder-Decoder LSTM结构的一些有趣的应用。

- **机器翻译**，如短语的英译法语。
- **学习执行**，例如小程序的计算结果；
- **图像标题**，例如用于生成图像；
- **对话建模**，例如对语篇产生的答案的问题；
- **运动序列的分类**，例如对一系列的手势生成一系列的命令；

9.2.4 实现

Encoder-Decoder LSTM可以直接在Keras中实现。我们可以认为模型由两个关键部分组成：编码器和解码器。首先，输入序列一次向网络显示一个编码字符。我们需要一个编码水平来学习输入序列中的步骤之间的关系，并开发这些关系的内部表示。

一个或多个LSTM层可用于显示编码器模型。这个模型的输出时一个固定大小的向量，表示输入许的内部表示。这个层中的存储单元的数目与这个大小的向量长度无关。

```
model = Sequential()
model.add(LSTM(..., input_shape=(...)))
```

表 9.1 Vanilla LSTM模型的例子

解码器必须将所学习的输入序列的内部表示转换成正确的输出序列。还可以使用一个或多个LSTM层来实现编解码模型。这个模型是从编码器模型的大小输出中读取的。与Vanilla LSTM一样，一个Dense层可以被用做网络的输出。通过将Dense层包裹在TimeDistributed层中，同样的权重可以被用来在每个输出序列中输出每个时间步长。

```
model.add(LSTM(..., return_sequences=True))
model.add(TimeDistributed(Dense(...)))
```

表 9.2 用TimeDistributed包裹Dense层的LSTM模型的例子

但是有一个问题。我们必须把编码器和解码器连接起来，但是它们不适合。也就是说，编码器将产生输出的二维矩阵，其中长度由层中的存储单元的数目决定。解码器是一个LSTM层，它期望3D输入（样本、时间步长、特征），以产生由该问题产生的不同长度的解码序列。

如果你试图强迫这些碎片在一起，你会得到一个错误，表明解码器的输出是2D，需要3D解码器。我们可以用重复向量层来解决这个问题。该层简单地多次重复所提出的2D输入以创建3D输出。

RepeatVector层可以像适配器一样使用，以将网络的编码器和解码器部分适配在一起。我们可以配置重复向量以在输出序列中的每个时间步长中重复一个固定长度向量。

```
model.add(RepeatVector(...))
```

表 9.3 一个RepeatVector层的例子

把它们放在一起，我们得到：

```
model = Sequential()
model.add(LSTM(..., input_shape=(...)))
model.add(RepeatVector(...))
model.add(LSTM(..., return_sequences=True))
model.add(TimeDistributed(Dense(...)))
```

表 9.4 Encoder-Decoder模型的例子

总的来说，使用RepeatVector作为编码器的固定大小的2D输出，以适应解码器期望的不同长度和3D输入。TimeDistributed wrapper允许相同的输出层用于输出序列中的每个元素。

9.3 加法预测问题

加法问题是一个序列到序列，或者seq2seq的预测问题。它被用于 Wojciech Zaremba和Ilya Sutskever2014年名为《Learning to Execute》的论文来探索Encoder-Decoder LSTM的能力，其中的体系结构被证明学习计算小程序的输出。

该问题被定义为计算两个输入数的和的输出。这是具有挑战性的，因为每个数字和数学符号被提供为字符类型的，并且预期输出也被预期为字符。例如，输入10+6与输出16将由序列表示：

```
Input: [ 1 , 0 , + , 6 ]
Output: [ 1 , 6 ]
```

表 9.5 加法问题中输入和输出序列的例子

该模型不仅要学习字符的整数性质，还要学习要执行的数学运算的性质。注意序列是如何重要的，并且随机地拖动输入将创建与输出序列无关的无意义序列。还是要注意序列在输入和输出序列中如何变化。在技术上，这使得加法预测问题是一个序列到序列的问题，需要many-to-many的模型来求解。

图 9.2 用many-to-many预测模型构造加法预测问题

我们可以通过添加两个数字来保持事物的简单性，但是我们可以看到如何将其缩放成可变数量的术语和数学运算，这些数学运算可以作为模型的输入来学习和推广。这个问题可以用Python来实现。我们可以把它们分成以下步骤：

1. 生成加法对；
2. 填充字符串的整数；
3. 整数编码序列；
4. one hot编码序列；

5. 序列生成流水线；
6. 解码序列。

9.3.1 生成加法对

第一步是生成随机整数序列及其总和。我们可以把它放在一个名为random_sum_pairs()的函数中，如下所示：

```
from random import seed
from random import randint


# generate lists of random integers and their sum
def random_sum_pairs(n_examples, n_numbers, largest):
    X, y = list(), list()
    for i in range(n_examples):
        in_pattern = [randint(1,largest) for _ in range(n_numbers)]
        out_pattern = sum(in_pattern)
        X.append(in_pattern)
        y.append(out_pattern)
    return X, y

seed(1)
n_samples = 1
n_numbers = 2
largest = 10
# generate pairs
X, y = random_sum_pairs(n_samples, n_numbers, largest)
print(X, y)
```

表 9.6 生成随机序列对的例子

运行这个函数只打印一个在1到10之间添加两个随机整数的例子。

```
[[3, 10]] [13]
```

表 9.7 输出生成一个随机序列对的例子

9.3.2 填充字符串的整数

下一步是将整数转换为字符串。输入字符串将是“10+10”格式，输出字符串将是“20”格式。这个函数的关键是填充数字，以确保每个输出和输出序列具有相同的字符数。填充字符应该与数据无关，因此模型可以学会忽略它们。在这种情况下，我们使用空格字符串（“ ”）填充，并在左侧填充字符串，保持最右边的信息。

还有其他的方法来填充，比如单个填充每个术语。试试看它是否会带来更好的性能。填充需要我们知道最长序列的长度。我们可以通过计算我们可以生成的最大整数的 $log_{10}()$ 和这个数字的上线来计算

每个数字需要多少字符。我们增加了1个最大的数字，以确保我们期望3个字符而不是2个字符，对于一个圆最大的数字，比如200个取结果的上限（例如 $ceil(log10(largest+1))$ ）。然后，我们需要添加正确数目的加符号（例如，`n numbers - 1`）。

```
max_length = n_numbers * ceil(log10(largest+1)) + n_numbers - 1
```

表 9.8 计算输入序列最大长度的例子

我们可以用一个实际的例子来做这个具体的例子，其中总的数量（n个数）是3，最大的值（最大）是10。

```
max_length = n_numbers * ceil(log10(largest+1)) + n_numbers - 1
max_length = 3 * ceil(log10(10+1)) + 3 - 1
max_length = 3 * ceil(1.0413926851582251) + 3 - 1
max_length = 3 * 2 + 3 - 1
max_length = 6 + 3 - 1
max_length = 8
```

表 9.9 最大输入序列长度工作实例

直观来说，我们期望每个词两个空间（例如['1', '0']）乘以3个词，或者最大长度为6个空间的输入序列，如果有加法符号的话就再加两位（例如：[‘1’ , ‘0’ , ‘+’ , ‘1’ , ‘0’ , ‘+’ , ‘1’ , ‘0’]）使得最大的可能序列长度为8个字符。这就是我们在实际例子中看到的。

在输出序列上重复一个类似的过程，当然没有加号。

```
max_length = ceil(log10(n_numbers * (largest+1)))
```

表 9.10 计算输出序列长度的例子

再次，我们可以通过计算期望的最大输出序列长度来具体实现，上面的例子的总数量（n个数）是3，最大值（最大）是10。

```
max_length = ceil(log10(n_numbers * (largest+1)))
max_length = ceil(log10(3 * (10+1)))
max_length = ceil(log10(33))
max_length = ceil(1.5185139398778875)
max_length = 2
```

表 9.11 最大输出序列长度的工作实例

同样的，直观的，我们期望最大可能的加法是10+10+10或者30的 值。这将需要最大长度为2，这就是我们在工作示例中所看到的。下面的示例添加了string()函数，并用一个输入/输出对来演示它的用法。

```
from random import seed
from random import randint
from math import ceil
from math import log10

# generate lists of random integers and their sum
def random_sum_pairs(n_examples, n_numbers, largest):
    X, y = list(), list()
    for i in range(n_examples):
        in_pattern = [randint(1,largest) for _ in range(n_numbers)]
        out_pattern = sum(in_pattern)
        X.append(in_pattern)
        y.append(out_pattern)
    return X, y

# convert data to strings
def to_string(X, y, n_numbers, largest):
    max_length = n_numbers * ceil(log10(largest+1)) + n_numbers - 1
    Xstr = list()
    for pattern in X:
        strp = '+'.join([str(n) for n in pattern])
        strp = ''.join(['' for _ in range(max_length-len(strp))]) + strp
        Xstr.append(strp)
    max_length = ceil(log10(n_numbers * (largest+1)))
    ystr = list()
    for pattern in y:
        strp = str(pattern)
        strp = ''.join(['' for _ in range(max_length-len(strp))]) + strp
        ystr.append(strp)
    return Xstr, ystr

seed(1)
n_samples = 1
n_numbers = 2
largest = 10

# generate pairs
X, y = random_sum_pairs(n_samples, n_numbers, largest)
print(X, y)
# convert to strings
X, y = to_string(X, y, n_numbers, largest)
print(X, y)
```

表 9.12 将一个序列对转换成插补字符的例子

运行例子首先输出整数序列，并插补同样序列的字符串表达。

```
[[3, 10]] [13]
['3+10'] ['13']
```

表 9.13 将一个序列对转换为插补字符的输出的例子

9.3.3 整数编码序列

接下来，我们需要将字符串中的每个字符编码为整数值。在神经网络中我们必须用数字进行工作，而不是字符。整数编码将问题转化为一个分类问题，其中输出序列可以被认为是具有11个可能值的类输出。这恰好是具有一些序数关系的整数（前10类值）。为了执行此编码，我们必须确定字符串编码中可能出现的符号的完整字母表，如下：

```
alphabet = [ 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , + , ]
```

表 9.14 定义一个字符表的例子

然后，整数编码成为一个简单的过程，构建一个字符串到整数偏移的查找表，并逐个转换每个字符串的每个字符。下面的示例提供整数编码的integer_encode()函数，并演示它如何使用。

```
from random import seed
from random import randint
from math import ceil
from math import log10


# generate lists of random integers and their sum
def random_sum_pairs(n_examples, n_numbers, largest):
    X, y = list(), list()
    for i in range(n_examples):
        in_pattern = [randint(1,largest) for _ in range(n_numbers)]
        out_pattern = sum(in_pattern)
        X.append(in_pattern)
        y.append(out_pattern)
    return X, y

# convert data to strings
def to_string(X, y, n_numbers, largest):
    max_length = n_numbers * ceil(log10(largest+1)) + n_numbers - 1
    Xstr = list()
    for pattern in X:
        strp = '+ '.join([str(n) for n in pattern])
        strp = ''.join(['' for _ in range(max_length-len(strp))]) + strp
        Xstr.append(strp)
    max_length = ceil(log10(n_numbers * (largest+1)))
    ystr = list()
    for pattern in y:
        strp = str(pattern)
        strp = ''.join(['' for _ in range(max_length-len(strp))]) + strp
        ystr.append(strp)
    return Xstr, ystr
# integer encode strings
def integer_encode(X, y, alphabet):
    char_to_int = dict((c, i) for i, c in enumerate(alphabet))
    Xenc = list()
    for pattern in X:
        integer_encoded = [char_to_int[char] for char in pattern]
        Xenc.append(integer_encoded)
```

```
    yenc = list()
    for pattern in y:
        integer_encoded = [char_to_int[char] for char in pattern]
        yenc.append(integer_encoded)
    return Xenc, yenc

seed(1)
n_samples = 1
n_numbers = 2
largest = 10
# generate pairs
X, y = random_sum_pairs(n_samples, n_numbers, largest)
print(X, y)
# convert to strings
X, y = to_string(X, y, n_numbers, largest)
print(X, y)
# integer encode
alphabet = [ '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' , '9' , '+' , ' ']
X, y = integer_encode(X, y, alphabet)
print(X, y)
```

表 9.15 整数编码插补序列的例子

运行例子打印每个字符串编码模式的整数编码版本。我们可以看到空字符串（“”）被编码成了11，字符三（“3”）被编码成了3，等等。

```
[[3, 10]] [13]
['3+ 10'] ['13']
[[3, 10, 11, 1, 0]] [[1, 3]]
```

表 9.16 从整数编码输入和输出序列输出的例子

9.3.4 one hot编码序列

下一步是对整数编码序列进行二进制编码。这涉及到将每个整数转换成与字母相同长度的二进制向量，并用1标记特定的整数。例如，0个整数表示“0”字符，并将其编码为11个向量元素的第0位位1的二进制向量：[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]。下面的例子为二进制编码定义了one_hot_encode() 函数，并演示了如何使用它。

```
from random import seed
from random import randint
from math import ceil
from math import log10


# generate lists of random integers and their sum
def random_sum_pairs(n_examples, n_numbers, largest):
    X, y = list(), list()
    for i in range(n_examples):
        in_pattern = [randint(1,largest) for _ in range(n_numbers)]
        out_pattern = sum(in_pattern)
```

```
        X.append(in_pattern)
        y.append(out_pattern)
    return X, y

# convert data to strings
def to_string(X, y, n_numbers, largest):
    max_length = n_numbers * ceil(log10(largest+1)) + n_numbers - 1
    Xstr = list()
    for pattern in X:
        strp = '+'.join([str(n) for n in pattern])
        strp = ''.join(['' for _ in range(max_length-len(strp))]) + strp
        Xstr.append(strp)
    max_length = ceil(log10(n_numbers * (largest+1)))
    ystr = list()
    for pattern in y:
        strp = str(pattern)
        strp = ''.join(['' for _ in range(max_length-len(strp))]) + strp
        ystr.append(strp)
    return Xstr, ystr

# integer encode strings
def integer_encode(X, y, alphabet):
    char_to_int = dict((c, i) for i, c in enumerate(alphabet))
    Xenc = list()
    for pattern in X:
        integer_encoded = [char_to_int[char] for char in pattern]
        Xenc.append(integer_encoded)
    yenc = list()
    for pattern in y:
        integer_encoded = [char_to_int[char] for char in pattern]
        yenc.append(integer_encoded)
    return Xenc, yenc

# one hot encode
def one_hot_encode(X, y, max_int):
    Xenc = list()
    for seq in X:
        pattern = list()
        for index in seq:
            vector = [0 for _ in range(max_int)]
            vector[index] = 1
            pattern.append(vector)
        Xenc.append(pattern)
    yenc = list()
    for seq in y:
        pattern = list()
        for index in seq:
            vector = [0 for _ in range(max_int)]
            vector[index] = 1
            pattern.append(vector)
        yenc.append(pattern)
    return Xenc, yenc

seed(1)
n_samples = 1
n_numbers = 2
largest = 10
# generate pairs
```

```
X, y = random_sum_pairs(n_samples, n_numbers, largest)
print(X, y)
# convert to strings
X, y = to_string(X, y, n_numbers, largest)
print(X, y)
# integer encode
alphabet = [ '0' , '1' , '2' , '3', '4' , '5' , '6' , '7' , '8' , '9' , '+' ,' ']
X, y = integer_encode(X, y, alphabet)
print(X, y)
# one hot encode
X, y = one_hot_encode(X, y, len(alphabet))
print(X, y)
```

表 9.17 one hot编码一个整数编码序列的例子

运行示例为每个整数编码打印二进制编码序列。我添加了一些新行，使输入和输出的二进制编码更加清晰。可以看到，一个和模式变成5个二进制编码向量的序列，每一个都有11个元素。输出或者和成为2个二进制编码向量的序列，每一个都具有11个元素。

```
[[3, 10]] [13]
['3+10'] ['13']
[[3, 10, 1, 0]] [[1, 3]]
[[[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]] [[[0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]]]
```

表 9.18 one hot编码一个整数编码序列的输出的例子

9.3.5 序列生成流水线

我们可以将所有这些步骤结合到一个名为generate_data()的函数中，如下所示。给定设计的样本数量、术语数量、每个术语的最大值和可能字符的字母表，函数将生成一组输入和输出序列。

```
# generate an encoded dataset
def generate_data(n_samples, n_numbers, largest, alphabet):
    # generate pairs
    X, y = random_sum_pairs(n_samples, n_numbers, largest)
    # convert to strings
    X, y = to_string(X, y, n_numbers, largest)
    # integer encode
    X, y = integer_encode(X, y, alphabet)
    # one hot encode
    X, y = one_hot_encode(X, y, len(alphabet))
    # return as NumPy arrays
    X, y = array(X), array(y)
    return X, y
```

表 9.19 生成一个序列、编码和对其进行变型以适应LSTM模型的例子

9.3.6 解码序列

最后，我们需要反转编码来将输出向量转换成数字，这样我们就可以将预期输出整数有预测整数进行比较。下面的invert()函数执行此操作。关键是使用argmax()函数将二进制编码转回到整数，然后将整数转换成字符，使用整数的反向映射到字母表中的字符。

```
# invert encoding
def invert(seq, alphabet):
    int_to_char = dict((i, c) for i, c in enumerate(alphabet))
    strings = list()
    for pattern in seq:
        string = int_to_char[argmax(pattern)]
        strings.append(string)
    return ''.join(strings)
```

表 9.20 决定一个编码输入或者输出序列的例子

现在我们为这个例子准备好了所有的事情了。

9.4 定义并编译模型

第一步是定义一个特定的序列预测问题。我们必须指定3个参数作为generate_data()函数(如上)的输入来生成输入-输出序列的样本：

- **n_term**:等式中单词的数目（例如，2则为10+10）。
- **largest**:每个单词的最大数（例如，10则为值在0-10之间）。
- **alphabet**:用于编码输入和输出序列的字母表（例如，0-9，+和“ ”）。

我们将使用具有适度复杂性的问题的配置。每个实例由3个术语组成，每个术语的最大值为10.不管值为0-9，+，还是“0”，字母表保持不变。

```
# number of math terms
n_terms = 3
# largest value for any single input digit
largest = 10
# scope of possible symbols for each input or output time step
alphabet = [str(x) for x in range(10)] + [ '+' , ' ']
```

表 9.21 配置问题实例的例子

由于加法问题的特殊性，网络需要三个配置值。

- n_chars:一个时间步长的字母表的大小（例如，12对应0,9，'+'和‘ ’）。
- n_in_seq_length:编码输入序列的时间步长（例如，8的时候对应'10+10+10'）
- n_out_seq_length:编码输出序列的时间步长（例如，2时候对应'30'）。

n_chars变量用于对输入层中的特征数目和输出层中的每个输入和输出时间步长的特征数进行分解。使用n_in_seq_length变量来定义时间步长的数量以在RepeatVector中重复编码输入，这反过来定义了序列喂入用于产生输出序列的解码器中的长度。n_in_seq_length和n_out_seq_length的定义使用了来自to_string()函数相同的代码，to_string()函数是用作将整数序列映射为字符串的。

```
# size of alphabet: (12 for 0-9, + and )
n_chars = len(alphabet)
# length of encoded input sequence (8 for 10+10+10)
n_in_seq_length = n_terms * ceil(log10(largest+1)) + n_terms - 1
# length of encoded output sequence (2 for 30 )
n_out_seq_length = ceil(log10(n_terms * (largest+1)))
```

表 9.22 在问题实例的基础上定义网络配置的例子

现在我们准备好定义Encoder-Decoder LSTM了。我们将使用一个单一的LSTM层的编码器和另一个单一层的解码器。编码器具有75个存储单元和50个存储单元的解码器。记忆细胞的数量是通过一次次的实验和错误确定的。由于输入序列相对输出序列较长，所以编码器和解码器中的层的大小不对称似乎是一种自然的组织。

输出层使用可预测的12个可能类别的分类log损失。使用了有效的Adam算法实现梯度下降法，并且在训练和模型评估期间计算精度。

```
# define LSTM
model = Sequential()
model.add(LSTM(75, input_shape=(n_in_seq_length, n_chars)))
model.add(RepeatVector(n_out_seq_length))
model.add(LSTM(50, return_sequences=True))
model.add(TimeDistributed(Dense(n_chars, activation= 'softmax' )))
model.compile(loss= categorical_crossentropy , optimizer= 'adam' , metrics=[ 'accuracy'
])
print(model.summary())
```

表 9.23 定义并编译Encoder-Decoder LSTM的例子

运行示例打印网络结构的摘要。我们可以看到，编码器将输出一个固定大小的向量，对于给定的输入序列长度为75。该序列被重复2次，以提供75个特征的2个时间步长序列到解码器。解码器将50个特征的两个时间步长输入到Dense输出层，通过一个TimeDistributed wrapper一次处理这些输出，以每次输出一个编码字符。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 75)                26400
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 2, 75)             0
_________________________________________________________________
lstm_2 (LSTM)                (None, 2, 50)             25200
_________________________________________________________________
time_distributed_1 (TimeDist (None, 2, 12)             612
=================================================================
```

```
Total params: 52,212
Trainable params: 52,212
Non-trainable params: 0
_________________________________________________________________
None
```

表 9.24 定义和编译Encoder-Decoder LSTM输出的例子

9.5 拟合模型

该模型适合于75000个随机生成的输入输出对实例的单个周期（epoch）。序列的数目是训练周期（epoch）的代理。总共有75000个数，选定批次大小（batch size）为32个是通过一次次尝试和错误得来的，并不是一个最佳的配置。

```
# fit LSTM
X, y = generate_data(75000, n_terms, largest, alphabet)
model.fit(X, y, epochs=1, batch_size=32)
```

表 9.25 拟合定义的Encoder-Decoder LSTM的例子

拟合提供进度条，显示模型在每个批次结束时的损失和准确性。该模型不需要很长的时间就可以安装在CPU上。如果进度条干扰您的开发环境，您可以通过在fit()函数中设置verbose=0来关闭它。

```
75000/75000 [==============================] - 37s - loss: 0.6982 - acc: 0.7943
```

表 9.26 拟合定义的Encoder-Decoder LSTM的输出的例子

9.6 评价模型

我们可以通过在100个不同的随机产生的输入-输出对上生成预测来评估模型。结果将给出一般随机生成示例的模型学习能力的估计。

```
# evaluate LSTM
X, y = generate_data(100, n_terms, largest, alphabet)
loss, acc = model.evaluate(X, y, verbose=0)
print( 'Loss: %f, Accuracy: %f' % (loss, acc*100))
```

表 9.27 评价拟合Encoder-Decoder LSTM拟合的例子

运行该示例同时打印模型的log损失和准确性。由于神经网络的随机性，您的特定值可能有所不同，但是模型的精度应该是在90%以内的。

```
Loss: 0.128379, Accuracy: 100.000000
```

表 9.28 评估拟合Encoder-Decoder LSTM输出的例子

9.7 用模型做预测

我们可以使用拟合模型进行预测。我们将演示一次做出一个预测，并提供解码输入、预期输出和预测输出的摘要。打印解码输出使我们对问题和模型能力有了更具体的联系。在这里，我们生成10个新的随机输入-输出序列对，使用每一个拟合模型进行预测，解码所涉及的所有序列，并将它们打印到屏幕上。

```
# predict
for _ in range(10):
    # generate an input-output pair
    X, y = generate_data(1, n_terms, largest, alphabet)
    # make prediction yhat = model.predict(X, verbose=0)
    # decode input, expected and predicted
    in_seq = invert(X[0], alphabet)
    out_seq = invert(y[0], alphabet)
    predicted = invert(yhat[0], alphabet)
    print( '%s = %s (expect %s)' % (in_seq, predicted, out_seq))
```

表 9.29 使用Encoder-Decoder LSTM做预测的例子

运行该示例表明，模型使大部分序列正确。你生成的叶鼎序列和模型的学习能力在10个例子中会有所不同。尝试运行预测几次，以获得良好的模型行为的感觉。

```
9+10+9 = 27 (expect 28)
9+6+9 = 24 (expect 24)
8+9+10 = 27 (expect 27)
9+9+10 = 28 (expect 28)
2+4+5 = 11 (expect 11)
2+9+7 = 18 (expect 18)
7+3+2 = 12 (expect 12)
4+1+4 = 9 (expect 9)
8+6+7 = 21 (expect 21)
5+2+7 = 14 (expect 14)
```

表 9.30 用拟合Encoder-Decoder LSTM做预测输出的例子

9.8 完整例子

为了完整性，我们将全部的代码列表提供如下供你参考。

```
from random import seed
from random import randint
from numpy import array
```

```
from math import ceil
from math import log10
from math import sqrt
from numpy import argmax
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import TimeDistributed
from keras.layers import RepeatVector

# generate lists of random integers and their sum
def random_sum_pairs(n_examples, n_numbers, largest):
    X, y = list(), list()
    for i in range(n_examples):
        in_pattern = [randint(1,largest) for _ in range(n_numbers)]
        out_pattern = sum(in_pattern)
        X.append(in_pattern)
        y.append(out_pattern)
    return X, y

# convert data to strings
def to_string(X, y, n_numbers, largest):
    max_length = n_numbers * ceil(log10(largest+1)) + n_numbers - 1
    Xstr = list()
    for pattern in X:
        strp = '+' .join([str(n) for n in pattern])
        strp = ''.join(['' for _ in range(max_length-len(strp))]) + strp
        Xstr.append(strp)
    max_length = ceil(log10(n_numbers * (largest+1)))
    ystr = list()
    for pattern in y:
        strp = str(pattern)
        strp = ''.join(['' for _ in range(max_length-len(strp))]) + strp
        ystr.append(strp)
    return Xstr, ystr

# integer encode strings
def integer_encode(X, y, alphabet):
    char_to_int = dict((c, i) for i, c in enumerate(alphabet))
    Xenc = list()
    for pattern in X:
        integer_encoded = [char_to_int[char] for char in pattern]
        Xenc.append(integer_encoded)
    yenc = list()
    for pattern in y:
        integer_encoded = [char_to_int[char] for char in pattern]
        yenc.append(integer_encoded)
    return Xenc, yenc

# one hot encode
def one_hot_encode(X, y, max_int):
    Xenc = list()
    for seq in X: pattern = list()
    for index in seq:
        vector = [0 for _ in range(max_int)]
        vector[index] = 1
        pattern.append(vector)
        Xenc.append(pattern)
```

```
    yenc = list()
    for seq in y:
        pattern = list()
        for index in seq:
            vector = [0 for _ in range(max_int)]
            vector[index] = 1
            pattern.append(vector)
        yenc.append(pattern)
    return Xenc, yenc

# generate an encoded dataset
def generate_data(n_samples, n_numbers, largest, alphabet):
    # generate pairs
    X, y = random_sum_pairs(n_samples, n_numbers, largest)
    # convert to strings
    X, y = to_string(X, y, n_numbers, largest)
    # integer encode
    X, y = integer_encode(X, y, alphabet)
    # one hot encode
    X, y = one_hot_encode(X, y, len(alphabet))
    # return as numpy arrays
    X, y = array(X), array(y)
    return X, y

# invert encoding
def invert(seq, alphabet):
    int_to_char = dict((i, c) for i, c in enumerate(alphabet))
    strings = list()
    for pattern in seq:
        string = int_to_char[argmax(pattern)]
        strings.append(string)
    return ''.join(strings)

# configure problem
# number of math terms
n_terms = 3
# largest value for any single input digit
largest = 10
# scope of possible symbols for each input or output time step
alphabet = [str(x) for x in range(10)] + [ '+' , ' ']

# size of alphabet: (12 for 0-9, + and )
n_chars = len(alphabet)
# length of encoded input sequence (8 for 10+10+10)
n_in_seq_length = n_terms * ceil(log10(largest+1)) + n_terms - 1
# length of encoded output sequence (2 for 30 )
n_out_seq_length = ceil(log10(n_terms * (largest+1)))

# define LSTM
model = Sequential()
model.add(LSTM(75, input_shape=(n_in_seq_length, n_chars)))
model.add(RepeatVector(n_out_seq_length))
model.add(LSTM(50, return_sequences=True))
model.add(TimeDistributed(Dense(n_chars, activation= 'softmax' )))
model.compile(loss= 'categorical_crossentropy' , optimizer= 'adam' , metrics=[ 'accuracy'
])
print(model.summary())
```

```
# fit LSTM
X, y = generate_data(75000, n_terms, largest, alphabet)
model.fit(X, y, epochs=1, batch_size=32)

# evaluate LSTM
X, y = generate_data(100, n_terms, largest, alphabet)
loss, acc = model.evaluate(X, y, verbose=0)
print('Loss: %f, Accuracy: %f' % (loss, acc*100))

# predict
for _ in range(10):
    # generate an input-output pair
    X, y = generate_data(1, n_terms, largest, alphabet)
    # make prediction
    yhat = model.predict(X, verbose=0)
    # decode input, expected and predicted
    in_seq = invert(X[0], alphabet)
    out_seq = invert(y[0], alphabet)
    predicted = invert(yhat[0], alphabet)
    print('%s = %s (expect %s)' % (in_seq, predicted, out_seq))
```

表 9.31 Encoder-Decoder LSTM在加法预测问题上的完整例子

9.9 扩展阅读

本章节提供了一些扩展阅读的资料。

9.9.1 Encoder-Decoder LSTM论文

- Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014.
- [Sequence to Sequence Learning with Neural Networks, 2014. https://arxiv.org/abs/1409.3215
- Show and Tell: A Neural Image Caption Generator, 2014.
- Learning to Execute, 2015.
- {A Neural Conversational Model, 2015.](https://arxiv.org/abs/1506.05869)

9.9.2 Keras API

- RepeatVector Keras API.
- TimeDistributed Keras API.

9.10 扩展

你想更深度地了解Encoder-Decoder LSTMs吗？本章节列出了本课程的一些具有挑战性的扩展。

- 列出10个可以从Encoder-Decoder LSTM结构中获益的序列到序列预测问题；
- 增加terms的数量或者数字的数量，并调整模型以获得100%的准确度；
- 设计一个比较模型大小与问题序列问题复杂度（term和/或数字）的研究；
- 更新示例以支持给定实例中的可变数量的术语，并调整模型以获得100%的准确度。
- 增加对其他数学运算的支持，例如减法、除法和乘法。

把你的扩展张贴到网上，并把链接分享给我。我很想知道你是怎么样想的。

9.11 总结

在本课程中，你学习到了怎么样开发一个Encoder-Decoder LSTM模型。特别地，你学习到了：

- Encoder-Decoder LSTM的结构以及怎么样在Keras中实现它；
- 加法序列到序列的预测问题；
- 怎么样开发一个Encoder-Decoder LSTM模型用来解决加法seq2seq预测问题。

在下一个章节中，我们将会学习到怎么样开发并评估一个Bidirectional LSTM模型。