10.0 前言

10.0.1 课程目标

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

- Bidirectional LSTM模型的结构和怎么样在Keras中实现它；
- 积累和问题的；
- 怎么样为积累和问题开发一个Bidirectional LSTM模型。

10.0.2 课程概览

本课程分类为7个部分，它们是：

1. Bidirectional LSTM;
2. 积累和预测问题；
3. 定义和编译模型；
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！

10.1 Bidirectional LSTM

10.1.1 结构

我们已经看到了Encoder-Decoder LSTM的介绍中讨论的LSTMs输入序列的顺序的好处。

> 我们对源句子中颠倒词的改进程度感到惊讶。
>
> — Sequence to Sequence Learning with Neural Networks, 2014.

Bidirectional LSTMs专注于通过输入和输出时间步长在向前和向后两个方向上获得最大的输入序列的问题。在实践中，该架构涉及复制网络中的第一个递归层，使得现在有两个并排的层，然后提供输入序列，作为输入到第一层并且提供输入序列到第二层的反向副本。这种方法是在不久前发展起来的一种用于改善循环神经网络（RNNs）性能的一般方法。

> 为了克服常规RNN的局限性...我们提出了一个双向递归神经网络（BRNN），可以使用所有可用的输入信息在过去和未来的特定时间帧进行训练。...我们的方法是将一个规则的RNN状态神经元分裂成两个部分，一部分负责正时间方向（正向状态），另外一个部分负责负时间方向（后向状态）。
>
> — Bidirectional Recurrent Neural Networks, 1997.

该方法以及被应用于LSTM循环神经网络。向前和向后提供整个序列是基于假设整个序列是可用的假设的。在使用矢量化输入时，在这个实践中通常是一个要求。然而，它可能会引起哲学上的关注，其中理想的时间步长是按顺序和及时（just-in-time）提供的。在语音识别领域中，双向地提供输入序列是合理的，因为有证据表明，在人类中，整个话语的上下文被用来解释所说的话而不是一个线性解释。

> ...依赖于乍一看是违反因果关系的未来知识。我们如何才能理解我们所听到的关于海没有说过的话呢？然而，人类的厅总就是这样做的。在未来的语境中，声音、词语乃至于整个句子都是毫无意义的。我们必须记住的是，任务之间的区别是真的在线的——在每个输入之后需要一个输出，以及在某些输入段的末尾只需要输出。
>
> — Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures, 2005.

虽然Bidirectional LSTMs被开发用于语音识别，但是使用双向输入序列是序列预测的主要因素，而LSTMs是提升模型性能的一种方法。

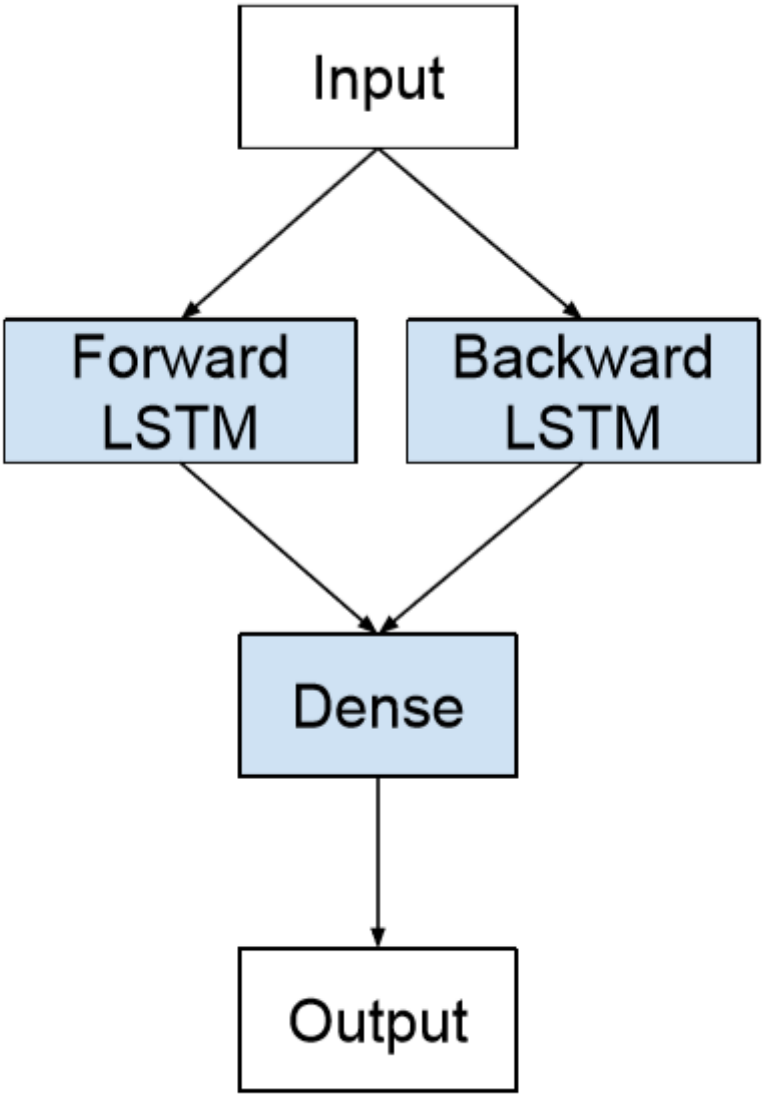

图 10.1 Bidirectional LSTM结构

10.1.2 实现

Keras中的LSTM层使得你指定输入序列的方向变得可能。可以通过设置go backwards参数为True（默认的为False）来完成。

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

表 10.1 具有反向输入序列的Vanilla LSTM模型的例子

Bidirectional LSTMs是这个能力的一个小的步骤。具体来说，Keras所支持的通过Bidirectional层包裹的Bidirectional LSTM，该双向层wrapper实质上合并来自两个并行LSTMs的输出，一个具有向前处理的输入和一个向后处理的输出。这个wrapper将一个递归层（例如，第一LSTM隐藏层）作为一个参数。

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

表 10.2 包裹在LSTM层中的Bidirectional层的例子

Bidirectional wrapper层还允许您指定合并模式；即在向前传递到下一层之前应该如何组合前向和向后输出。选项是：

- “sum”：将输出加在一起；
- “mul”：将输出乘在一起；
- “concat”：默认情况下，将输出连接在一起，为下一层提供两杯的输出数。
- “ave”：输出的平均值。

默认模式是级联，这在双向LSTM研究中经常使用的方法。一般来说，测试您的问题中的每个合并模式可能是个好主意，看看是否可以改进级联的默认选项。

10.2 计算和预测问题

我们将会定义一个简单的序列分类问题来探索Bidirectional LSTMs叫做计算和预测问题。本节本分为下面的几个部分：

1. 计算和；
2. 序列生成；
3. 生成多序列.

10.2.1 计算和

该问题被定义为0和1之间的随机值序列。这个序列作为输入的问题，每个时间提供一个数字一次。二进制标签（0或者1）与每个输入相关联。输出均值为0。一旦序列中的输入值的累计和超过阈值，则输出值从0翻转到1.

使用阈值的四分之一（$\frac{1}{4}$）的序列长度。例如，下面是10个输入时间步长（X）的序列：

```
0.63144003 0.29414551 0.91587952 0.95189228 0.32195638 0.60742236 0.83895793 0.18023048
0.84762691 0.29165514
```

表 10.3 随机真值的输入序列的例子

相应的分类输出（y）会是：

```
0 0 0 1 1 1 1 1 1 1
```

表 10.4 输出计算和值的输出的例子

我们将解决这个问题，以充分利用Bidirectional LSTM的结构。输出序列将在整个输入序列被喂进模型之后产生。

技术上，这意味着这是一个序列到序列的预测问题，需要一个多对多的预测模型。输入和输出序列具有相同的时间步长（长度）的情况下也是如此。

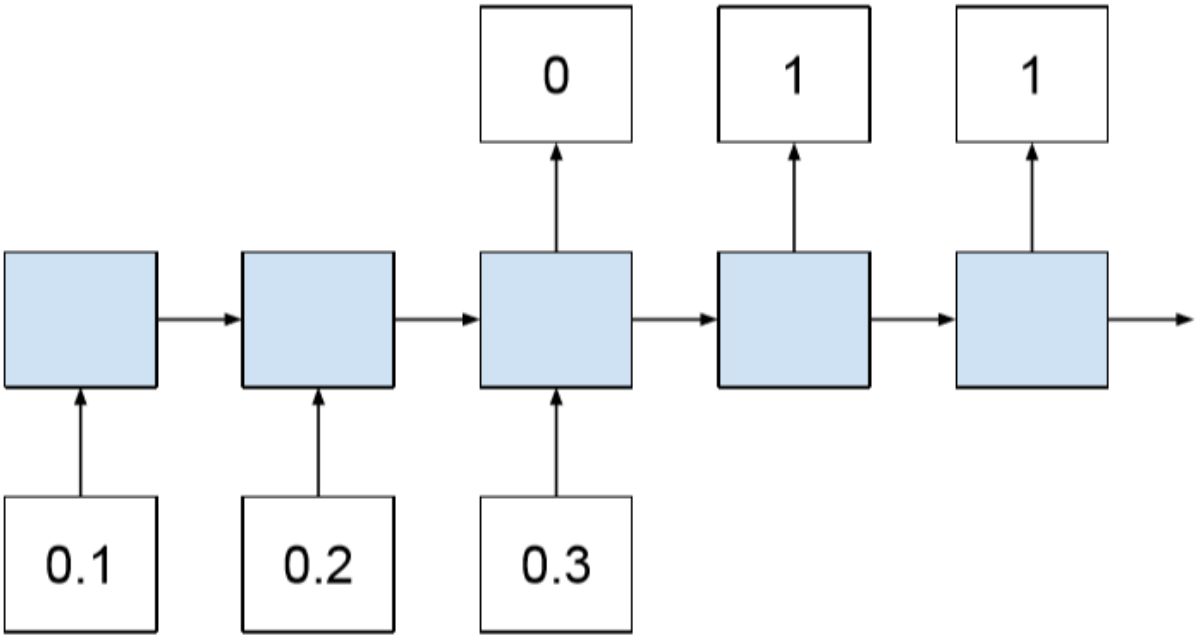

图 10.2 用一个many-to-many的模型计算和预测问题

10.2.2 序列生成

我们可以用Python来实现。第一步是生成随机数值的序列。我们可以使用随机组件中的random()函数。

```
# create a sequence of random numbers in [0,1]
X = array([random() for _ in range(10)])
```

表 10.5 生成随机真值的输入序列的例子

我们可以定义输入序列长度的四分之一为阈值。

```
# calculate cut-off value to change class values
limit = 10/4.0
```

表 10.6 计算累积和阈值的例子

输入序列的累积和可以的计算可以使用NumPy的cumsum()函数。此函数返回一个累积和值序列，例如：

```
pos1, pos1+pos2, pos1+pos2+pos3, ...
```

表 10.7 计算累积和输出序列的例子

然后，我们可以计算每个累积和值是否超过阈值的输出序列。

```
# determine the class outcome for each item in cumulative sequence
y = array([0 if x < limit else 1 for x in cumsum(X)])
```

表 10.8 实现累积和阈值计算的例子

下面的函数，叫做get_sequence()，将所有这些都结合在一起，将序列的长度作为输入，并返回新问题案例的X和y。

```
# create a sequence classification instance
def get_sequence(n_timesteps):
    # create a sequence of random numbers in [0,1]
    X = array([random() for _ in range(n_timesteps)])
    # calculate cut-off value to change class values
    limit = n_timesteps/4.0
    # determine the class outcome for each item in cumulative sequence
    y = array([0 if x < limit else 1 for x in cumsum(X)])
    return X, y
```

表 10.9 产生一个随机输入和输出序列的函数

我们可以用一个新的10-step序列测试这个函数，如下所示：

```
from random import random
from numpy import array
from numpy import cumsum

# create a cumulative sum sequence
def get_sequence(n_timesteps):
    # create a sequence of random numbers in [0,1]
    X = array([random() for _ in range(n_timesteps)])
    # calculate cut-off value to change class values
    limit = n_timesteps/4.0
    # determine the class outcome for each item in cumulative sequence
    y = array([0 if x < limit else 1 for x in cumsum(X)])
    return X, y

X, y = get_sequence(10)
print(X)
print(y)
```

表 10.10 生成随机输入输出序列的例子

运行例子首先打印生成的输入序列，跟着的是匹配输出序列：

```
[ 0.22228819 0.26882207 0.069623 0.91477783 0.02095862 0.71322527 0.90159654 0.65000306
0.88845226 0.4037031 ]
[0 0 0 0 0 0 1 1 1 1]
```

表 10.11 生成一个随机输入和输出序列的输出的例子

10.2.3 生成多个序列

我们可以定义一个函数来创建多个序列。下面名为get_sequences()的函数使用序列的数量来生成，同时每个序列的时间步长作为参数并使用get_sequence()来生成序列。一旦特定数量的序列被生成，输入和输出序列的列表被变型为三维的，适合于LSTMs一起使用。

```
# create multiple samples of cumulative sum sequences
def get_sequences(n_sequences, n_timesteps):
    seqX, seqY = list(), list()
    # create and store sequences
    for _ in range(n_sequences):
        X, y = get_sequence(n_timesteps)
        seqX.append(X)
        seqY.append(y)
    # reshape input and output for lstm
    seqX = array(seqX).reshape(n_sequences, n_timesteps, 1)
    seqY = array(seqY).reshape(n_sequences, n_timesteps, 1)
    return seqX, seqY
```

表 10.2 生成序列的函数和适应LSTM模型格式

我们现在准备好开始为整个问题开发一个Bidirectional LSTM模型。

10.3 定义和编译模型

首先，我们定义了一个复杂的问题。我们会限制输入时间步长的数目在一个合适的大小；在这种情况下，10。这意味着输入形状将是具有1个特征的10个时间步长。

```
# define problem
n_timesteps = 10
```

表 10.13 配置问题的例子

下面，我们需要定义隐藏在一个Bidirectional层中的隐藏的LSTM层。我们将在LSTM隐藏层中使用50个存储单元。Bidirectional wrapper将加倍，创建一个平行于第一层的第二层，也有50个存储单元。

```
model.add(Bidirectional(LSTM(50, return_sequences=True), input_shape=(n_timesteps, 1)))
```

表 10.14 添加Bidirectional输入层的例子

将从前向和后向LSTM隐藏层中的每一个输出50个向量的向量（Bidirectional wrapper 层的默认合并方法）以创建100个元素的向量输出。这是作为输入到Dense层的输入，该Dense层被包裹在TimeDistributed层中。这具有重用Dense层的权重以创建每个输出时间步长的效果。

```
model.add(TimeDistributed(Dense(1, activation= sigmoid )))
```

表 10.15 添加TimeDistributed输出层的例子

Bidirectional LSTM层返回给包裹着的Dense层序列。这具有向每个输出时间步长提供Dense层的一个级联的100个元素矢量作为输入的效果。如果不使用TimeDistributed wrapper，一个100个元素的向量将会被提供给Dense层，从该Dense层将需要输出10个时间步长的分类。对于模型来说，这似乎是一个更具有挑战性的问题。

把这些放在一起，模型定义如下。在Dense输出层sigmoid被用作激活函数，并且二项log损失被优化，因为每个输出时间步长是一个累积和的阈值是否超过的二分类问题。在模型训练和评价过程中，采用Adam梯度下降法来优化权值，并计算分类精度。

```
# define LSTM model = Sequential()
model.add(Bidirectional(LSTM(50, return_sequences=True), input_shape=(n_timesteps, 1)))
model.add(TimeDistributed(Dense(1, activation= 'sigmoid' )))
model.compile(loss= 'binary_crossentropy' , optimizer= 'adam' , metrics=[ 'acc' ])
print(model.summary())
```

表 10.16 定义和编译Bidirectional LSTM模型的例子

运行这个代码打印编译模型的总结。我们可以确认Dense层由100个权重（加上偏置），一个用于从Bidirectional包裹的LSTM隐藏层提供的100个元素级联向量中的每个单元。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_1 (Bidirection (None, 10, 100)           20800
_________________________________________________________________
time_distributed_1 (TimeDist (None, 10, 1)             101
=================================================================
Total params: 20,901
Trainable params: 20,901
Non-trainable params: 0
_________________________________________________________________
None
```

表 10.17 定义和编译Bidirectional LSTM模型的输出的例子

10.4 拟合模型

在这个模型中，我们使用get_sequences()函数来生成大量的随机例子。我们可以通过使用速记生成序列的数目作为周期（epoch）的代理来简化训练。这使得我们能够生成大量的例子，在这种情况下，5000，将它们存储在内存中，并且在一个Keras周期（epoch）中存储它们。

使用10个批次大小（batch size）来平衡学习速度和计算效率。在实验和误差的基础上，发现了样本的数量和皮批次的大小。用不同的值进行实验，看看能否用较少的计算量来训练一个精确的模型。

```
# train LSTM
X, y = get_sequences(50000, n_timesteps)
model.fit(X, y, epochs=1, batch_size=10)
```

表 10.18 拟合和编译Bidirectional LSTM模型的例子

拟合模型并不需要很长的时间。进度条提供用于训练，并且log损失和模型准确度将会在每个批次（batch）被更新。

```
50000/50000 [==============================] - 97s - loss: 0.0508 - acc: 0.9817
```

表 10.19 拟合一个编译的Bidirectional LSTM模型的输出的例子

10.5 模型评价

我们可以通过生成100个新的随机序列来评估模型，并计算拟合模型的预测的准确性。

```
# evaluate LSTM
X, y = get_sequences(100, n_timesteps)
loss, acc = model.evaluate(X, y, verbose=0)
print( Loss: %f, Accuracy: %f % (loss, acc*100))
```

表 10.20 评价一个拟合Bidirectional LSTM模型输出的例子

运行例子打印log损失和准确率。我们可以看到模型达到了100%的准确率。当给定算法的随机性质时，该示例的精度可能会有所不同。你可以看到模型的学习能力在很高的90s。尝试运行例子几次。

```
Loss: 0.016752, Accuracy: 100.000000
```

表 10.21 从评估一个拟合Bidirectional LSTM模型的输出的例子

10.6 用模型进行预测

我们可以用类似于评估模型的方式进行预测。在这种情况下，我们将生成10个新的随机序列，对每个进行预测，将预测的输出序列与期望的输出序列进行比较。

```
# make predictions
for _ in range(10):
    X, y = get_sequences(1, n_timesteps)
    yhat = model.predict_classes(X, verbose=0)
    exp, pred = y.reshape(n_timesteps), yhat.reshape(n_timesteps)
    print( y=%s, yhat=%s, correct=%s % (exp, pred, array_equal(exp,pred)))
```

表 10.22 使用拟合Bidirectional LSTM模型做出预测的例子

运行示例打印预期的（y）和预测的（yhat）输出序列以及预测序列是否正确。我们可以看到，至少在这种情况下，10个序列中的2个在一个时间步长中被预测错误。

你具体的结果会有所不同，但你应该看到类似的行为平均。这是一个具有挑战性的问题，及时对于一个模型，它具有大量的例子，并显示出良好的精度，它仍然可以在预测新序列中产生错误。

```
y=[0 0 0 0 0 0 1 1 1 1], yhat=[0 0 0 0 0 0 1 1 1 1], correct=True
y=[0 0 0 0 1 1 1 1 1 1], yhat=[0 0 0 0 1 1 1 1 1 1], correct=True
y=[0 0 0 1 1 1 1 1 1 1], yhat=[0 0 0 1 1 1 1 1 1 1], correct=True
y=[0 0 0 0 0 0 0 1 1 1], yhat=[0 0 0 0 0 0 0 0 1 1], correct=False
y=[0 0 0 0 0 1 1 1 1 1], yhat=[0 0 0 0 0 1 1 1 1 1], correct=True
y=[0 0 0 1 1 1 1 1 1 1], yhat=[0 0 0 1 1 1 1 1 1 1], correct=True
y=[0 0 0 0 0 1 1 1 1 1], yhat=[0 0 0 0 0 0 1 1 1 1], correct=False
y=[0 0 0 0 1 1 1 1 1 1], yhat=[0 0 0 0 1 1 1 1 1 1], correct=True
y=[0 0 0 0 0 0 0 0 1 1], yhat=[0 0 0 0 0 0 0 0 1 1], correct=True
y=[0 0 0 1 1 1 1 1 1 1], yhat=[0 0 0 1 1 1 1 1 1 1], correct=True
```

表 10.23 用一个拟合的Bidirectional LSTM模型做预测的输出的例子

10.7 完整例子

为了完整性全部的例子列出如下供您参考。

```
from random import random
from numpy import array
from numpy import cumsum
from numpy import array_equal
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import TimeDistributed
from keras.layers import Bidirectional


# create a cumulative sum sequence
def get_sequence(n_timesteps):
    # create a sequence of random numbers in [0,1]
```

```
    X = array([random() for _ in range(n_timesteps)])
    # calculate cut-off value to change class values
    limit = n_timesteps/4.0
    # determine the class outcome for each item in cumulative sequence
    y = array([0 if x < limit else 1 for x in cumsum(X)])
    return X, y

# create multiple samples of cumulative sum sequences
def get_sequences(n_sequences, n_timesteps):
    seqX, seqY = list(), list()
    # create and store sequences
    for _ in range(n_sequences):
        X, y = get_sequence(n_timesteps)
        seqX.append(X)
        seqY.append(y)
    # reshape input and output for lstm
    seqX = array(seqX).reshape(n_sequences, n_timesteps, 1)
    seqY = array(seqY).reshape(n_sequences, n_timesteps, 1)
    return seqX, seqY

# define problem
n_timesteps = 10

# define LSTM
model = Sequential()
model.add(Bidirectional(LSTM(50, return_sequences=True), input_shape=(n_timesteps, 1)))
model.add(TimeDistributed(Dense(1, activation= 'sigmoid' )))
model.compile(loss= 'binary_crossentropy' , optimizer= 'adam' , metrics=[ 'acc' ])
print(model.summary())

# train LSTM
X, y = get_sequences(50000, n_timesteps)
model.fit(X, y, epochs=1, batch_size=10)
# evaluate LSTM
X, y = get_sequences(100, n_timesteps)
loss, acc = model.evaluate(X, y, verbose=0)
print( 'Loss: %f, Accuracy: %f' % (loss, acc*100))
# make predictions
for _ in range(10):
    X, y = get_sequences(1, n_timesteps)
    yhat = model.predict_classes(X, verbose=0)
    exp, pred = y.reshape(n_timesteps), yhat.reshape(n_timesteps)
    print( 'y=%s, yhat=%s, correct=%s' % (exp, pred, array_equal(exp,pred)))
```

表 10.24 Bidirectional LSTM在积累和问题的完整的例子

10.8 扩展阅读

本章节提供了一些扩展阅读的资源。

10.8.1 研究论文

- Bidirectional Recurrent Neural Networks, 1997.
- Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures, 2005.
- Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition, 2005.
- Speech Recognition with Deep Recurrent Neural Networks, 2013.

10.8.2 APIs

- random() Python API.
- cumsum() NumPy API.
- Bidirectional Keras API.

10.9 扩展

你想更深入的了解Bidirectional LSTM吗？本章节将会列出本课程中具有挑战性的扩展：

- 列出5个可能从Bidirectional LSTM中获益的序列预测问题的例子；
- 调整记忆单元的数目、训练样本和批次大小（batch size）来开发一个更小的或者训练更快的具有100%准确度的模型；
- 设计并执行一个实验来比较模型大小和问题的复杂度（例如，序列长度）；
- 定义和执行一个实验来比较前向、后向和Bidirectional LSTM输入方向；
- 定义和执行一个实验来比较将Bidirectional LSTM wrapper层结合的方法。

将你的扩展放到网上并将链接分享给我，我很希望看到你是怎么样想的！

10.10 总结

在本课中，你学习到了怎么样开发一个Bidirectional LSTM模型。特别地，你学习到了：

- Bidirectional LSTM模型的结构和怎么样在Keras中实现它；
- 积累和问题的；
- 怎么样为积累和问题开发一个Bidirectional LSTM模型。

在下一课中，你将会学习到怎么样开发和评估一个Generative LSTM模型。