6.0 前言

6.0.1 课程目标

本课程的目标是学习如何开发和评估Vanilla LSTM模型模型。完成这一课之后，你将会知道：

- 用于序列预测的Vanilla LSTM模型的结构及其一般能力；
- 如何实现echo序列预测问题；
- 怎么样开发Vanilla LSTM模型模型来学习和准确地预测echo序列预测问题。

6.0.2 课程概览

本课程被划分为了7个部分，它们是：

1. Vanilla LSTM模型；
2. echo序列预测问题；
3. 定义和编译模型；
4. 拟合模型；
5. 评价模型；
6. 用模型进行预测；

7. 完成例子。

让我们开始吧！

6.1 Vanilla LSTM模型

6.1.1 结构

LSTM(Long Short Term Memory)是在标准RNN基础上改进而来的一种网络结构，其出现的主要作用是为了解决标准RNN训练过程中的梯度消失问题，LSTM的结构如下图所示。

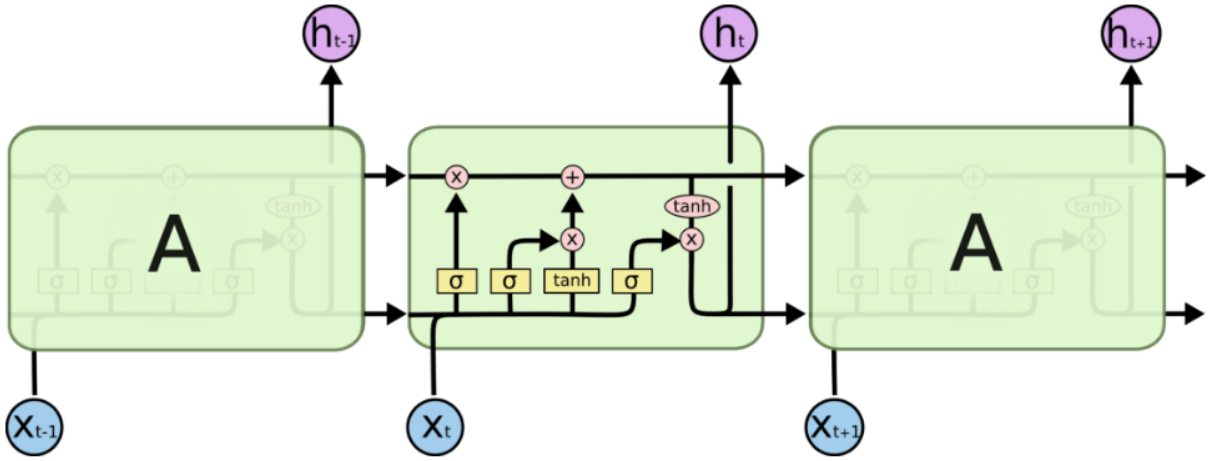


图 6.1 LSTM模型的结构

相比于标准RNN模型，LSTM主要是增加了三个控制门单元：遗忘门，输入门和输出门。如果去掉三个门控制单元（或者将三个门控制函数都设置为1），LSTM就会退化成标准的RNN模型。

- **细胞状态**：LSTM的关键就是细胞状态（如下图所示的水平线），在图上方贯穿运行，它类似于一条传送带，只有少量的信息交互，很容易保存信息不变。

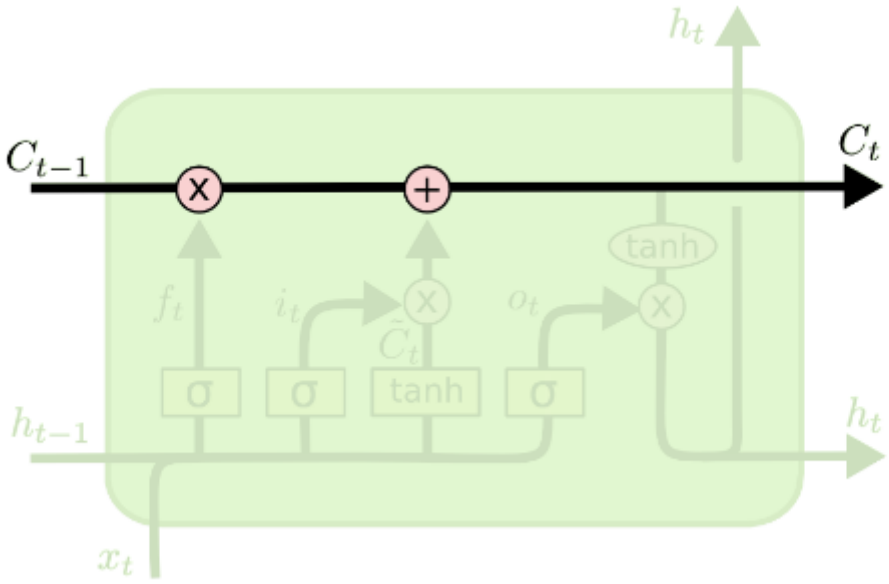
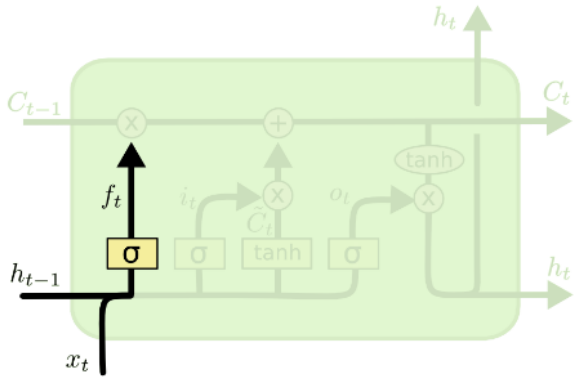


图 6.2 LSTM模型的细胞状态

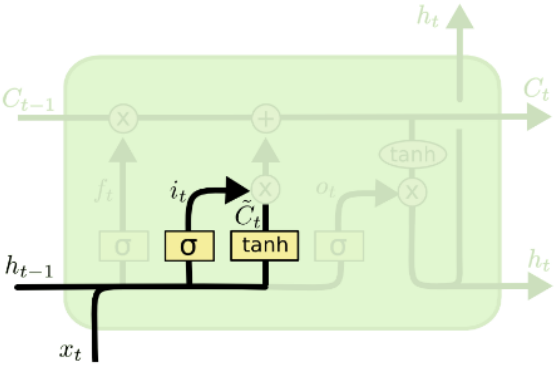
- **遗忘门**：结构如下图所示，决定细胞状态的信息需要丢弃多少。其读取和的信息，输出一个0到1之间的数值个细胞状态，1表示细胞状态完全保留，0表示细胞状态完全丢弃。



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

图 6.3 LSTM模型的遗忘门

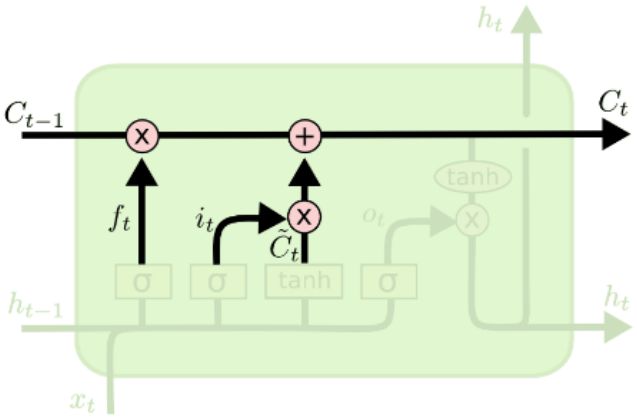
- **输入门**：结构如下图所示，确定什么新的信息需要被添加到细胞状态中。这里包含两方面的内容：1.`sigmod`层决定了什么值需要被更新；2.`tanh`层用来生成新的候选信息向量会被更新到细胞状态中。



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

图 6.3 LSTM模型的输入门

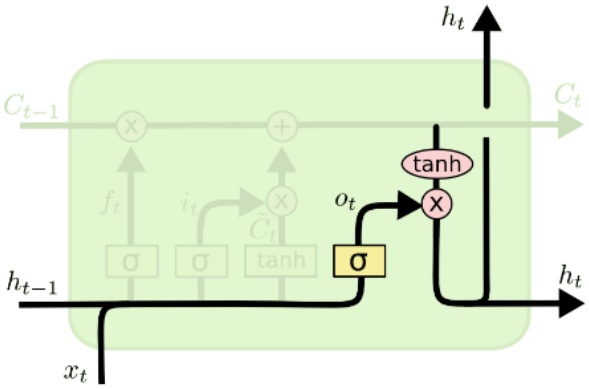
对细胞状态的更新为：将旧状态与相乘，可以丢弃到确定的需要丢弃的状态，然后加上新的需要加入的信息即可完成细胞状态的更新。对细胞状态的更新表示如下图：



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

图 6.4 LSTM模型的细胞状态的更新

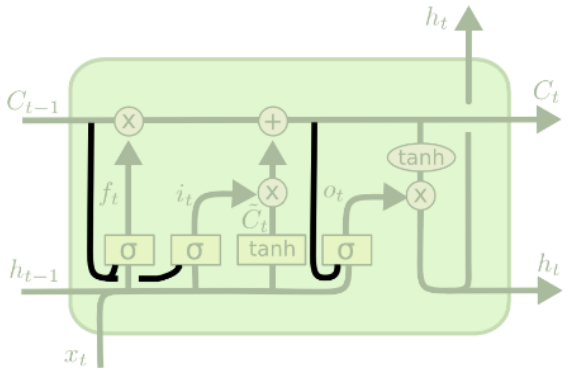
- **输出门**：确定细胞状态的什么信息需要被输出，结构图如下所示。首先是需要一个`sigmod`函数用来确定上一隐层和新输入信息有多少需要被保留，然后将更新后的细胞状态经过`tanh`变到`[-1,1]`的区间后再进行相乘，这样就确定了最终的输出信息。



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

图 6.5 LSTM模型的输出门

Vanilla LSTM模型是LSTM模型的一种简单变种，其主要变化是在三个控制门的输入分别加入了细胞状态信息（下图中的黑线），这个连接被称为`peephole connection`。



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

图 6.6 Vanilla LSTM模型结构

本书中名字叫Vanilla，以区别于更新的LSTM和一套更复杂的配置。它是原来1997年LSTM论文中所定义的LSTM结构，它将在大多数小序列预测问题上给出良好的结果。Vanilla LSTM被定义为：

1. 输入层。
2. 全连接LSTM隐藏层。
3. 全连接输出层。

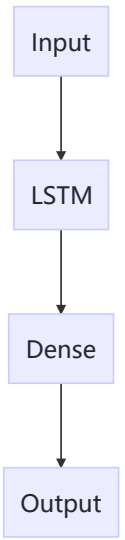


图 6.7 Vanilla LSTM模型整体表示

6.1.2 实现

在Keras中，一个Vanilla LSTM模型被定义如下，这里为了方便大家看清楚结构对于每个神经元的数目做了省略：

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

表 6.7 定义一个Vanilla LSTM模型的例子

这种默认或者标准的LSTM在深入学习LSTM的许多工作和讨论中被引用。Vanilla LSTM具有以下5个有吸引力的特性，具体可以在原论文中查看：

- 基于多分布输入时间步长的序列分类；
- 数千个时间步长的精确输入观察的记忆；
- 作为先前时间步长的函数的序列预测；
- 对输入序列上随机时间步长的插入具有鲁棒性；
- 对输入序列上好好数据的放置具有鲁棒性；

接下里，我们将提出一个简单的序列预测问题，我们可以用它来证明Vanilla LSTM的以上特性。

6.2 Echo序列预测问题

echo序列预测问题是设计来证明LSTM能力的一个问题。该任务是：给定一个随机的整数序列作为输入，在特定时间输入步长（time input step）输出一个随机整数值（该时间输入步长（time input step）是由模型指定的）。

例如，给定随机整数[5, 3, 2]输入序列，而且被选择的时间步长是第二个值，然后期待的输出时3。技术上来说，这是一个序列分类的问题；当在序列的结尾有多个输入时间步长和一个输出时间步长时，它被定义为一个many-to-one预测问题。

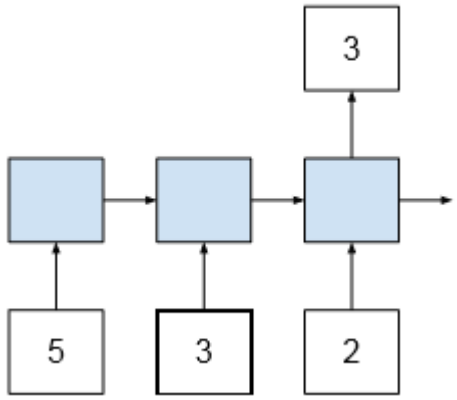


图 6.8 echo序列预测问题构建成many-to-one预测模型

我们很小心地选择了这个问题用来阐释Vanilla LSTM存储能力。此外，我们将手动执行模型生命周期中的一些元素，例如，对模型进行拟合和评估，以便对其中所含的原理有更深刻的理解。接下来，从代码中来解读其运行的原理及过程。具体包括以下步骤：

1. 生成随机序列；
2. one hot编码序列；
3. 实例；
4. 序列变型。

6.2.1 生成随机序列

在Python中使用`randint()`函数可以生成随机整数，该函数需要指定两个整数以便明确从什么返回内生成随机数。在本课中，我们将会定义从0到99之间生成唯一的值。

```
ranint(0, 99)
```

表 6.2 生成随机整数

我们可以将这个函数放到一个叫做`generate_sequence()`的函数中，该函数将会生成一个所期望长度序列的随机整数。这个函数被列出来如下：

```
# generate a sequence of random numbers in [0, n_features)
def generate_sequence(length, n_features):
    return [randint(0, n_features-1) for _ in range(length)]
```

表 6.3 生成随机整数序列的函数

6.2.2 one hot编码序列

我们生成了随机整数序列之后，我们需要将其转换成一个合适LSTM网络训练的数据格式。其中一个选择是将其缩放到[0,1]的范围内。实践证明这是可行的，而且我们可以把这个问题归结为回归问题。

因为我们感兴趣的是预测正确的数字，而不是接近预期值的数字。这意味着我们更喜欢将问题化为分类而不是回归，其中预期的输出是一个类，并且有100个可能的类值。在这种情况下，我们可以使用整数值的热编码，其中每个值由100个元素的二进制向量表示。

下面的函数称为`one_hot_encode()`定义了怎么样在一个序列的整数值上迭代，并为每个创建一个二进制向量表达，并将结果作为2维数组返回。

```
# one hot encode sequence
def one_hot_encode(sequence, n_features):
    encoding = list()
    for value in sequence:
        vector = [0 for _ in range(n_features)]
        vector[value] = 1
        encoding.append(vector)
return array(encoding)
```

表 6.4 随机整数序列one hot编码函数

完成编码之后，我们也需要对已经编码的结果进行解码，以便我们可以利用预测；在这种情况下，仅仅浏览下它们。用NumPy中的`argmax`函数可以转换one hot编码，用最大的值返回向量中的值的索引。下面的函数，叫做`one_hot_decode()`，将会解码一个编码了的序列，它后面可以被用作解码你网络的预测。

```
# decode a one hot encoded string
def one_hot_decode(encoded_seq):
    return [argmax(vector) for vector in encoded_seq]
```

表 6.5 解码编码序列的函数

6.2.3 实例

我们将其整合到一起。下面的是一个完整的用于生成25个序列的随机整数并将每个整数编码为二进制向量的例子：

```
from random import randint
from numpy import array
from numpy import argmax
```

```
# generate a sequence of random numbers in [0, n_features)
def generate_sequence(length, n_features):
    return [randint(0, n_features-1) for _ in range(length)]


# one hot encode sequence
def one_hot_encode(sequence, n_features):
    encoding = list()
    for value in sequence:
        vector = [0 for _ in range(n_features)]
        vector[value] = 1
        encoding.append(vector)
    return array(encoding)


# decode a one hot encoded string
def one_hot_decode(encoded_seq):
    return [argmax(vector) for vector in encoded_seq]


# generate random sequence
sequence = generate_sequence(25, 100)
print(sequence)
# one hot encode
encoded = one_hot_encode(sequence, 100)
print(encoded)
# one hot decode
decoded = one_hot_decode(encoded)
print(decoded)
```

表 6.6 生成序列并进行编码的例子

运行例子，首先打印25个随机整数，然后是一个所有整数序列的截断的二进制编码，每个向量一行，然后解码序列。由于每次运行的时候所生成的随机整数不一样，所以每个人的结果可能都是不一样的。

```
[50, 65, 99, 36, 5, 60, 74, 77, 64, 22, 79, 8, 56, 70, 95, 52, 59, 32, 45, 49, 85, 67,
98, 82, 6]
[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 1]
 ...,
 [0 0 0 ..., 0 1 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]]
[50, 65, 99, 36, 5, 60, 74, 77, 64, 22, 79, 8, 56, 70, 95, 52, 59, 32, 45, 49, 85, 67,
98, 82, 6]
```

表 6.7 生成序列并编码它们的输出的例子

这里`...`表示的是省略了的，该部分显示了25个长度为100的向量。

6.2.4 序列变型

最后一步是将one hot编码序列变型到一个可以用于LSTM输入的格式。这包括将编码的序列变型为`n`个时间步长和`k`个特征，这里`n`是生成序列的整数数目，而`k`是每个时间步长可能整数的集合（例如100）。

一个序列然后可以被变型为一个样本、时间步长和特征的3维的矩阵，或者对于一个具有25个整数的单一的序列`[1, 25, 100]`。例如：

```
X = encoded.reshape(1, 25, 100)
```

表 6.8 变换一个编码序列的例子

序列的输出是在特定预定义位置的单纯的编码整数。对于一个模型生成的所有示例，该位置必须保持一致，以便模型可以学习。例如，我们可以通过直接从编码序列取编码值，使用第二个时间步长作为具有25个时间步长的序列的输出。

```
y = encoded[1, :]
```

表 6.9 获取解码序列的值的例子

我们可以将这个以及上述所生产的以及编码步长一起放到一个新的叫做generate_example()的函数中来生成一个序列，编码它并返回输入（X）和输出（y）组件来训练LSTM。

```
# generate one example for an lstm
def generate_example(length, n_features, out_index):
    # generate sequence
    sequence = generate_sequence(length, n_features)
    # one hot encode
    encoded = one_hot_encode(sequence, n_features)
    # reshape sequence to be 3D
    X = encoded.reshape((1, length, n_features))
    # select output
    y = encoded[out_index].reshape(1, n_features)
    return X, y
```

表 6.10 生成序列、编码它们并对其变型的例子

我们可以将这些放在一起，测试一个例子的生成，以拟合或者评估LSTM如下：

```
from random import randint
from numpy import array
from numpy import argmax


# generate a sequence of random numbers in [0, n_features)
def generate_sequence(length, n_features):
```

```
    return [randint(0, n_features-1) for _ in range(length)]

# one hot encode sequence
def one_hot_encode(sequence, n_features):
    encoding = list()
    for value in sequence:
        vector = [0 for _ in range(n_features)]
        vector[value] = 1
        encoding.append(vector)
    return array(encoding)


# decode a one hot encoded string
def one_hot_decode(encoded_seq):
    return [argmax(vector) for vector in encoded_seq]

# generate one example for an lstm
def generate_example(length, n_features, out_index):
    # generate sequence
    sequence = generate_sequence(length, n_features)
    # one hot encode
    encoded = one_hot_encode(sequence, n_features)
    # reshape sequence to be 3D
    X = encoded.reshape((1, length, n_features))
    # select output
    y = encoded[out_index].reshape(1, n_features)
    return X, y

X, y = generate_example(25, 100, 2)
print(X.shape)
print(y.shape)
```

表 6.11 测试函数来生成编码序列并变型的例子

运行代码生成编码的序列并打印LSTM序列输入和输出组件的形状。

```
(1, 25, 100)
(1, 100)
```

表 6.112 输出从生成编码序列并变型的例子

现在我们知道怎么样准备和表达随机整数序列了，我们可以试着使用LSTMs来学习它们。

6.3 定义并编译模型

我们将从定义和编译模型开始。为了保证模型在一个合理的时间内拟合，我们将会通过将序列长度减少到5个整数和特征数为10（例如0~9）来大大简化问题。模型必须制定输入数据的预期维数。在这种情况下，根据时间步长（5）和特征（10）。我们将使用一个单一的隐藏层LSTM与25个内存单元，选择一个小的尝试和错误。

输出层时一个全连接层（Dense），具有10个神经元，用于10个可能输出的整数。在输出层上使用softmax()激活函数，以允许网络在可能的输出值上学习和输出分布。

网络将在训练是使用log损失函数，适用于多分类问题，以及效率Adam优化算法。精度度量将会在每个训练周期内给出，以便根据损失函数查看模型的学习能力。

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense

# define model
length = 5
n_features = 10
out_index = 2
model = Sequential()
model.add(LSTM(25, input_shape=(length, n_features)))
model.add(Dense(n_features, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

表 6.13 为echo问题定义一个Vanilla LSTM的例子

运行例子定义和编译模型，然后输出模型的总的结构。打印一个模型结构是一个很好的实践，一般来说，确认模型是根据你的意图定义和编译的。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 25)                3600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                260
=================================================================
Total params: 3,860
Trainable params: 3,860
Non-trainable params: 0
_________________________________________________________________
None
```

表 6.14 从定义模型中输出的例子

6.4 拟合模型

我们现在可以在样本序列上拟合模型了。我们编写的用于echo序列预测问题的代码可以生成随机序列。我们可以生成一个很大数量的例子序列并将其传递给模型的fit()函数。数据集将会被加载到内存，训练会很快，并且我们可以通过变换周期（epoch）的数量、数据集的大小以及批次（batch）的数量来实验。

一个更简单的方法是来手工管理训练过程，其中生成一个训练样本用于更新模型，并且清楚任何的内部状态。周期（epoch）的数目是生成样本的迭代次数，并且基本上批次（batch）处理的大小是1个

样本。下面是一个例子，说明模型的1000个周期（epoch）发现的一个小的尝试和错误。

```
# fit model
for i in range(10000):
    X, y = generate_example(length, n_features, out_index)
    model.fit(X, y, epochs=1,verbose = 2)
```

表 6.15 拟合定义的LSTM模型的例子

拟合模型将会报告log损失函数以及每个模式的准确率。这里，准确率既不是0或1（0%或者100%），因为我们对每一个样本进行序列分类预测并报告结果。

```
...
Epoch 1/1
0s - loss: 0.1610 - acc: 1.0000
Epoch 1/1
0s - loss: 0.0288 - acc: 1.0000
Epoch 1/1
0s - loss: 0.0166 - acc: 1.0000
Epoch 1/1
0s - loss: 0.0013 - acc: 1.0000
Epoch 1/1
0s - loss: 0.0244 - acc: 1.0000
```

表 6.16 输出拟合定义模型的例子

6.5 评估模型

一旦模型拟合了，我们就可以估计模型在分类新的随机序列的能力。我们可以通过简单地对100个随机生成的序列进行预测并计算正确预测的数量来做到这一点。

当我们拟合了模型，我们将会生成一个很大数量的样本，将其连接在一起，然后使用evaluate()函数来评估模型。在这种情况下，我们将会手动预测并计算正确的输出结果。我们可以在一个循环中完成生成样本、做出预测、并在结果是正确的时候计数器加1。

```
# evaluate model
correct = 0
for i in range(100):
    X, y = generate_example(length, n_features, out_index)
    yhat = model.predict(X)
    if one_hot_decode(yhat) == one_hot_decode(y):
        correct += 1
    print('Accuracy: %f' % ((correct/100)*100.0))
```

表 6.17 评价拟合LSTM模型的例子

评价模型报告模型的学习能力是100%。

```
Accuracy: 100.000000
```

表 6.18 输出评估拟合模型的例子

6.6 用模型做出预测

最后，我们使用拟合模型在一个新的随机生成的序列上来做出预测。对于这个问题，这与评估模型的情况大致相同。因为这是一个面向用户的活动，我们可以解码整个序列、预期输出和预测，并将它们打印在屏幕上。

```
# prediction on new data
X, y = generate_example(length, n_features, out_index)
yhat = model.predict(X)
print('Sequence: %s' % [one_hot_decode(x) for x in X])
print('Expected: %s' % one_hot_decode(y))
print('Predicted: %s' % one_hot_decode(yhat))
```

表 6.19 用拟合LSTM模型做出预测的例子

运行该示例将打印解码的随机生成的序列、预期的结果以及（希望）满足预期值的预测。你的具体结果会有所不同。

```
Sequence: [[7, 0, 2, 6, 7]] Expected: [2]
Predicted: [2]
```

表 6.20 输出从拟合模型预测结果的例子

如果模型显示错误请不要慌张。LSTMs是随机的，有可能单一模型的运行可能收敛在一个不完全了解问题的解决方案上。如果这发生在你的身上，试试多运行这个例子几次。

6.7 完整例子

本节列出了列出了完整的工作来供你进行参考。

```
from random import randint
from numpy import array
from numpy import argmax
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense


# generate a sequence of random numbers in [0, n_features)
def generate_sequence(length, n_features):
    return [randint(0, n_features-1) for _ in range(length)]


# one hot encode sequence
```

```
def one_hot_encode(sequence, n_features):
    encoding = list()
    for value in sequence:
        vector = [0 for _ in range(n_features)]
        vector[value] = 1
        encoding.append(vector)
    return array(encoding)


# decode a one hot encoded string
def one_hot_decode(encoded_seq):
    return [argmax(vector) for vector in encoded_seq]

# generate one example for an lstm
def generate_example(length, n_features, out_index):
    # generate sequence
    sequence = generate_sequence(length, n_features)
    # one hot encode
    encoded = one_hot_encode(sequence, n_features)
    # reshape sequence to be 3D
    X = encoded.reshape((1, length, n_features))
    # select output
    y = encoded[out_index].reshape(1, n_features)
    return X, y

# define model
length = 5
n_features = 10
out_index = 2
model = Sequential()
model.add(LSTM(25, input_shape=(length, n_features)))
model.add(Dense(n_features, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())

# fit model
for i in range(10000):
    X, y = generate_example(length, n_features, out_index)
    model.fit(X, y, epochs=1, verbose=2)

# evaluate model
correct = 0
for i in range(100):
    X, y = generate_example(length, n_features, out_index)
    yhat = model.predict(X)
    if one_hot_decode(yhat) == one_hot_decode(y):
        correct += 1
    print('Accuracy: %f' % ((correct/100)*100.0))

# prediction on new data
X, y = generate_example(length, n_features, out_index)
yhat = model.predict(X)
print('Sequence: %s' % [one_hot_decode(x) for x in X])
print('Expected: %s' % one_hot_decode(y))
print('Predicted: %s' % one_hot_decode(yhat))
```

表 6.21 将Vanilla LSTM模型应用到Echo问题的例子

6.8 扩展阅读

本章节提供了扩展阅读的一些资源。

- Long Short-Term Memory, 1997.
- Learning to Forget: Continual Prediction with LSTM, 1999.

6.9 扩展

你想更深入地了解Vanilla LSTM吗？本章节列出了本课程中一些具有挑战性的扩展。

- 更新例子使用更长的序列长度，并且仍然达到100%的准确度；
- 更新例子使用更大数量的特征，并且仍然达到100%的准确度；
- 更新例子使用SGD优化算法并调整学习率和momentum；
- 更新准备一个大型数据集来更新示例并探索不同的批大小（batch size）；
- 改变序列输出和训练周期（epoch）的时间步长指数，看看索引之间是否存在关系，以及问题学习的难度。

把你的扩展放在网上，并将连接分享给我。我很想知道你是怎么样想的。