7.0 前言

7.0.1 课程目标

本课程的目标是学习怎么样开发和评估一个Stacked LSTM模型。学习完本课之后，你将会知道：

- 创建多层LSTM的动机以及如何在Keras中开发Stacked LSTM模型；
- 阻尼正弦波预测问题，以及如何准备LSTM模型的例子。
- 如何开发、拟合和评估一个阻尼正弦预测的stacked LSTM模型。

7.0.2 课程概览

本课程被分为7部分，它们是：

1. Stacked LSTM；
2. 阻尼正弦波预测问题；

3. 定义并编译模型；
4. 拟合模型；
5. 评价模型；
6. 用模型做预测；
7. 完成例子。

让我们开始吧！

7.1 Stacked LSTM

Stacked LSTM模型是一个具有多个隐藏LSTM层的模型，其每个层包含多个存储单元。我们将其称为Stacked LSTM以便区分unstacked LSTM (Vanilla LSTM) 以及其他的基础LSTM模型的扩展。

7.1.1 为什么要增加深度?

Stacked LSTM隐藏层使得模型更深，更加准确地描述为深度学习技术。神经网络的深度使得其在具有广泛挑战性的预测问题中取得成功。

> 深度神经网络的成功通常归因于由于几层而引起的层次结构。每一层处理我们希望解决任务的一部分，并将其传递给下一个任务。在这个意义上，DNN可以被看做是一个处理流水线，其中每一层将在任务传递到下一层之前解决任务的一部分，知道最后一层提供输出。
>
> — Training and Analyzing Deep Recurrent Neural Networks, 2013

额外的隐藏层可以添加到多层感知机神经网络，使其更深。额外的隐藏层被理解为从先前的层重新组合学习的表示，并在高层次的抽象创建新的表示。例如，从线性到形状到物体。

一个足够大的单隐藏层多感知机可以用来近似大多数函数。增加网络的深度提供了一种替代的解决方案，需要更少的神经元和更快的训练。最后，增加深度是一种代表性的优化。

> 深度学习是建立在一个假设上的，即一个深层的、分层的模型可以比一些浅层大的模型更能代表某些功能。
>
> — How to Construct Deep Recurrent Neural Networks, 2013

7.1.2 结构

LSTMs也可以得到同样的好处。考虑到LSTMs对序列数据进行操作，这意味着添加层随着时间的推移增加了对输入观测的抽象水平。实际上，对观察值基于时间分块或者在不同时间尺度上代表问题。

> ...通过在彼此智商堆叠多个循环隐藏状态建立一个深度的RNN。这种方法可能允许每个级别的隐藏状态在不同的时间尺度上运行。
>
> — How to Construct Deep Recurrent Neural Networks, 2013

Stacked LSTMs或者深度LSTM由Graves等人介绍的。在LSTMs语音识别的应用中，在一个具有挑战性的标准问题上打了一个基准。

> RNNs在本质上在时间上很深，因为它们的隐藏状态是所有先前隐藏状态的函数。启发本论文的是，RNNs是否也可以从空间的深度，即堆叠多个循环隐藏层在上面，就像前馈层堆叠在传统的深度网络一样。
>
> — Speech Recognition With Deep Recurrent Neural Networks, 2013

在相同的工作中，它们发现网络的深度比在给定层中对模型的学习能力比存储单元的数量更重要。Stacked LSTM现在是一个挑战序列预测问题的稳定技术。Stacked LSTM体系结构可以作为一个由多个LSTM层组成的LSTM模型。上面的LSTM层提供了序列输出，而不是单个值输出到下面的LSTM层。具体来说，每个输入时间步长一个输出，而不是所有输入时间步长一个输出时间步长。

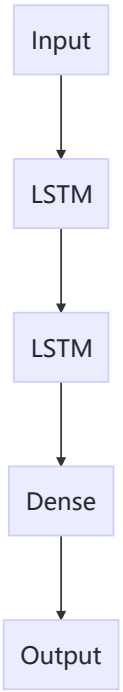

表 7.1 Stacked LSTM结构

7.1.3 实现

我们可以在Keras中轻松的时间Stacked LSTM模型。每个LSTM存储单元需要一个3D输入。当一个LSTM处理时间步长的一个输入序列的时候，每个存储单元将会为整个序列输出一个2D数组的单一值。我们可以用一个具有单个隐藏LSTM层（也是输出层）的模型来演示这一点。

```
# Example of one output for whole sequence
from keras.models import Sequential
from keras.layers import LSTM
from numpy import array
```

```
# define model where LSTM is also output layer
model = Sequential()
model.add(LSTM(1, input_shape=(3,1)))
model.compile(optimizer='adam', loss='mse')
# input time steps
data = array([0.1, 0.2, 0.3]).reshape((1,3,1))
# make and show prediction
print(model.predict(data))
```

表 7.1 单层LSTM的例子

输入序列有3个值。运行例子将输入序列的单个值作为2D数组输出。

```
[[ 0.01703017]]
```

表 7.2 单个层LSTM模型输出的例子

为了堆叠LSTM层，我们需要改变先前LSTM层的配置，以输出3D数组作为后续层的输入。我们可以通过在层上设置return sequences参数为True（默认为False）。这将会返回每一个输入时间步长的一个输出，并提供一个3D数组。下面是return_sequences=True时与上面相同的示例。

```
# Example of one output for whole sequence
from keras.models import Sequential
from keras.layers import LSTM
from numpy import array

# define model where LSTM is also output layer
model = Sequential()
model.add(LSTM(1, return_sequences=True,input_shape=(3,1)))
model.compile(optimizer='adam', loss='mse')
# input time steps
data = array([0.1, 0.2, 0.3]).reshape((1,3,1))
# make and show prediction
print(model.predict(data))
```

表 7.3 返回多序列的一层LSTM例子

运行例子输出输入序列的每个时间步长的单个值。

```
[[[ 0.02506424]
  [ 0.07560402]
  [ 0.1496872 ]]]
```

表 7.4 返回多个序列的一层LSTM模型的例子的输出

下面是定义一个两层隐藏层Stacked LSTM的例子：

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

表 7.5 定义一个具有2个隐藏层的Stacked LSTM例子

我们可以继续添加隐藏的LSTM层，只要先前的LSTM层提供3D输出作为后续层的输入；例如，下面是一个具有4个隐藏层的Stacked LSTM。

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

7.6 定义一个具有4个隐藏层的Stacked LSTM例子

下面我们将会定义一个问题，在这个问题上我们将会证明Stacked LSTM。

7.2 阻尼正弦波预测问题

本章节描述并实现了阻尼正弦波预测问题。本章节被分为下面的部分：

1. 正弦波；
2. 阻尼正弦波；
3. 随机阻尼正弦波；
4. 阻尼正弦波序列；

7.2.1 正弦波

正弦波描述随时间变化的振荡，该振荡具有一致的振幅（从基线运动）和频率（最小值和最大值之前的时间步长）。在没有陷入正弦波方程的情况下，我们可以编写代码来创建正弦波作为一个序列，并绘制它。

```
from math import sin
from math import pi
from matplotlib import pyplot
# create sequence
length = 100
freq = 5
sequence = [sin(2 * pi * freq * (i/length)) for i in range(length)]
# plot sequence
pyplot.plot(sequence)
pyplot.show()
```

表 7.7 生成并显示正弦波的例子

运行例子产生正弦波的图像。

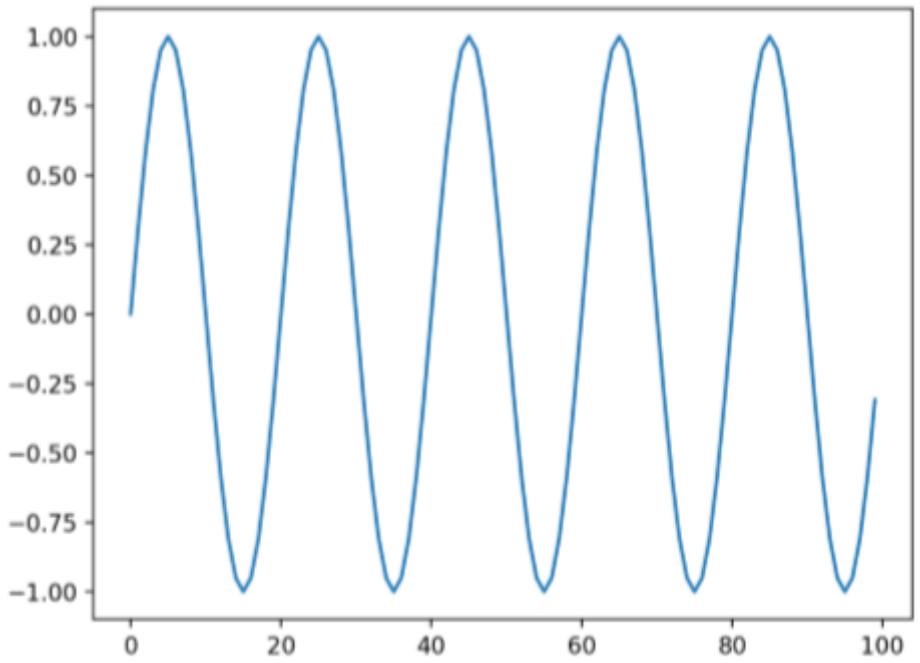

表 7.2 生成正弦波的线图

我们可以看到正弦波序列具有随着时间的变化而上升和下降的特性。这是Vanilla LSTM可以建模的良好的局部运动。LSTM可以记住序列，或者它可以使用最后几个时间步长来预测下一个时间步长。

7.2.2 阻尼正弦波

有一种正弦波随着时间而减少。幅度的减小提供了额外的长期运动，这可能需要在LSTM中附加抽象级别来学习。同样，在不考虑方程的情况下，我们可以在Python中实现这一点，如下所示。实现是谨慎的，以确保所有值都在0到1之间。

```
from math import sin
from math import pi
from math import exp
from matplotlib import pyplot
# create sequence
length = 100
period = 10
decay = 0.05
sequence = [0.5 + 0.5 * sin(2 * pi * i / period) * exp(-decay * i) for i in
range(length)]
# plot sequence
pyplot.plot(sequence)
pyplot.show()
```

表 7.8 生成并显示一个阻尼正弦波的例子

运行实例显示了正弦波和阻尼随时间的变化。周期参数决定了一个完整周期完成之前的多少个时间步长，并且衰减参数决定级数（series）如何快速地向0下降。

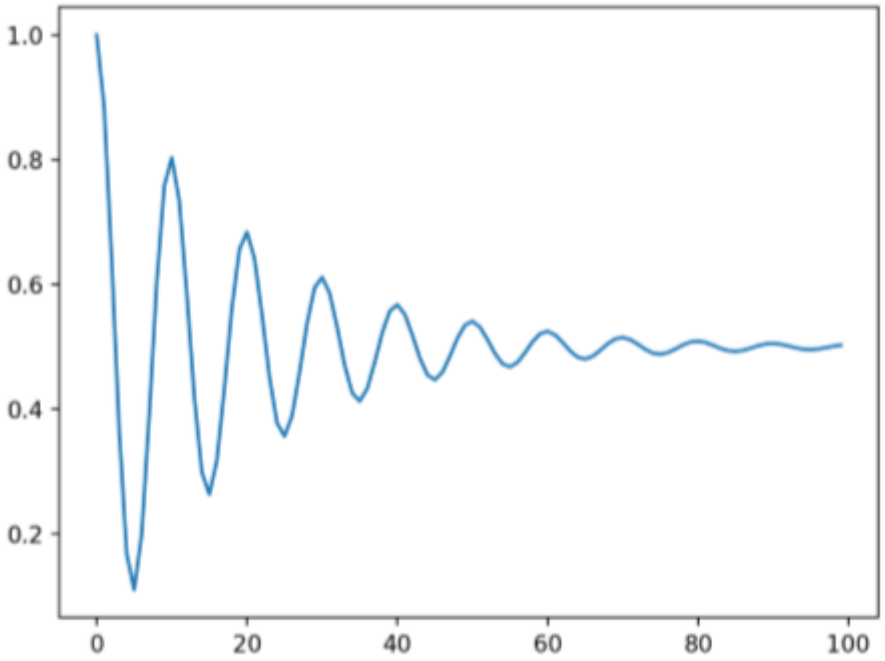

表 7.3 生成阻尼正弦波的线图

我们使用这个作为序列预测问题来证明Stacked LSTMs。

7.2.3 随机阻尼正弦波

我们可以生成具有不同周期和衰减值的阻尼正弦波序列，保留该系列的最后几个点，并对模型进行预测。首先，我们可以得到一个函数来产生一个阻尼的正弦波，称为generate sequence()。

```
# generate damped sine wave in [0,1]
def generate_sequence(length, period, decay):
    return [0.5 + 0.5 * sin(2 * pi * i / period) * exp(-decay * i) for i in
range(length)]
```

表 7.9 生命一个参数化阻尼正弦波的函数

7.2.4 阻尼正弦波序列

下面，我们需要一个函数来生成具有随机选择周期和衰减的序列。我们将使用randint()函数选择均匀的随机周期，在10和20之选择一个均匀的随机衰变。0.01和0.1使用uniform()函数。

```
p = randint(10, 20)
d = uniform(0.01, 0.1)
```

表 7.10 为阻尼正弦波生成随机参数的例子

我们将分离出每个序列的最后n个时间步长，并将其作为输出序列来预测。我们将使得这个是可配置的，包括序列产生的数量和序列的长度。下面提到的generate examples()函数实现了这一点，并返回了一个输入和输出示例的数组，用于训练或者评估LTSM。

```
# generate input and output pairs of damped sine waves
def generate_examples(length, n_patterns, output):
    X, y = list(), list()
    for _ in range(n_patterns):
        p = randint(10, 20)
        d = uniform(0.01, 0.1)
        sequence = generate_sequence(length + output, p, d)
        X.append(sequence[:-output])
        y.append(sequence[-output:])
        X = array(X).reshape(n_patterns, length, 1)
        y = array(y).reshape(n_patterns, output)
    return X, y
```

表 7.11 生成阻尼正弦波序列随机样本函数的例子

我们可以通过生成几个例子和绘制序列来测试这个函数。

```
from math import sin
from math import pi
from math import exp
from random import random
from random import randint
from random import uniform
from numpy import array
from matplotlib import pyplot


# generate damped sine wave in [0,1]
def generate_sequence(length, period, decay):
    return [0.5 + 0.5 * sin(2 * pi * i / period) * exp(-decay * i) for i in
range(length)]


# generate input and output pairs of damped sine waves
def generate_examples(length, n_patterns, output):
    X, y = list(), list()
    for _ in range(n_patterns):
        p = randint(10, 20)
        d = uniform(0.01, 0.1)
        sequence = generate_sequence(length + output, p, d)
        X.append(sequence[:-output])
        y.append(sequence[-output:])
    X = array(X).reshape(n_patterns, length, 1)
    y = array(y).reshape(n_patterns, output)
    return X, y


# test problem generation
```

```
X, y = generate_examples(20, 5, 5)
for i in range(len(X)):
    pyplot.plot([x for x in X[i, :, 0]] + [x for x in y[i]], '-o')
    pyplot.show()
```

表 7.12 生成并显示多个随机阻尼正弦波序列的例子

运行示例创建5个阻尼正弦波序列，每个都具有20个时间步长。额外的5个时间步长在序列的结尾生成，将作为测试数据集保留。我们将把这个扩展到实际问题的50个时间步长。

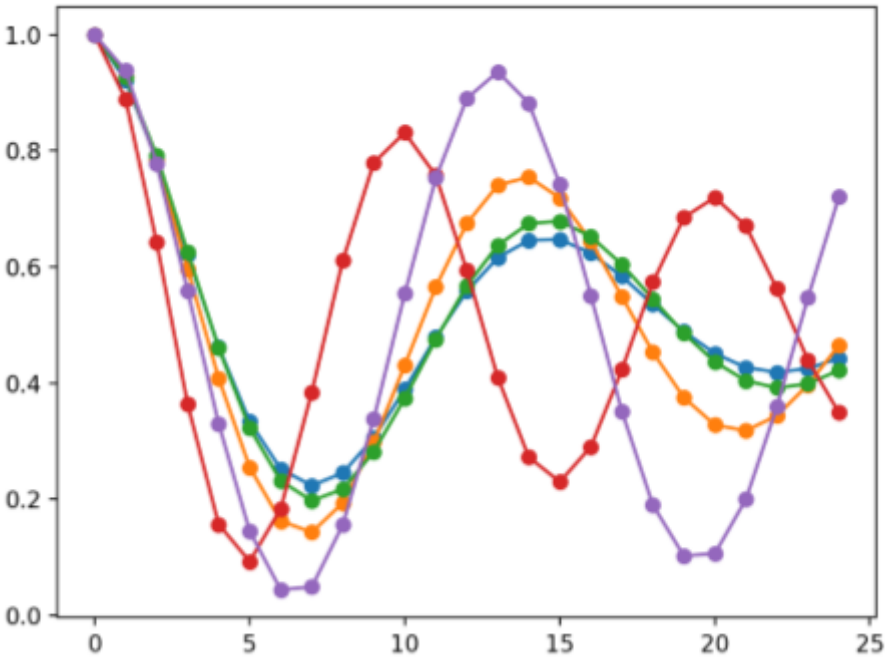

表 7.4 多个随机产生的阻尼正弦波的线图，其中点表示序列值

这是一个回归序列预测问题。它可以被认为是一个时间序列预测回归问题。在时间序列预测中，使序列平稳是一种很好的做法，即在建模问题之前从系列中删除任何系统趋势和季节性。这在使用LSTM时是推荐的。在本课中，我们故意不使该系列静止，以演示Stacked LSTM的能力。

从技术上来讲，这是many-to-one的序列预测问题。这可能是混乱的，因为我们显然打算预测一系列的输入时间步长。这是一个many-to-one的预测问题，原因是因为模型不会分段地预测输出时间步长，整个预测将同时产生。

从模型的角度来看，N个时间步长序列被喂入，然后在序列的末尾进行单个预测；恰好如此，预测是N个特征的向量，我们将其解释为时间步长。我们可以通过改变结构把这个模型调整为提出的many-

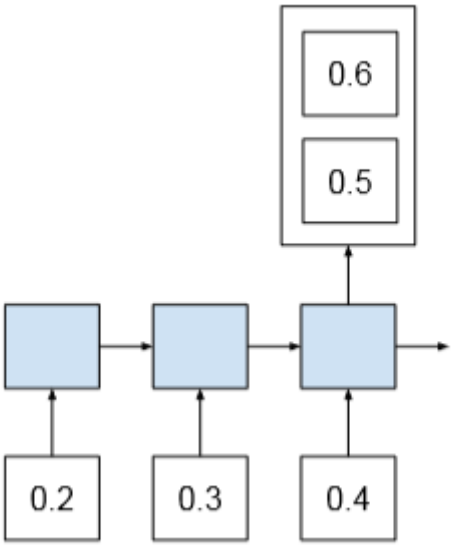

图 7.5 随机阻尼正弦波问题被构建为一个many-to-one的预测模型

to-many的LSTM模型。考虑将此作为扩展。

下面我们将为整个问题开发一个Stacked LSTM模型。

7.3 定义并编译模型

我们将会定义一个具有两个隐藏层的Stacked LSTM模型。每个LSTM层都有20个存储单元。输入维度将是1个特征，20个时间步长。模型的输出维度将是5个值的向量。我们将其解释为5个时间步长。输入层将使用线性激活函数，这是当没有函数被指定时使用的默认值。平均绝对误差（MAE）损失函数将被优化，并且将使用梯度下降优化算法的Adam实现。下面列出了模型定义的代码：

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

# configure problem
length = 50
output = 5
# define model
model = Sequential()
model.add(LSTM(20, return_sequences=True, input_shape=(length, 1)))
model.add(LSTM(20))
model.add(Dense(output))
model.compile(loss= 'mae', optimizer='adam')
print(model.summary())
```

表 7.13 为阻尼正弦波定义定义一个Stacked LSTM模型

该模型配置，特别是2层的使用和每层20个单元的使用，是在一次次实验和错误之后被选择的。模型配置是有竞争力的，但没有调谐到这个问题。运行此部分定义并编译模型，然后打印模型结构的摘要。在该结构中，我们可以确认模型的输入和输出的形状以及两个隐藏LSTM的使用。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 50, 20)            1760
_________________________________________________________________
lstm_2 (LSTM)                (None, 20)                3280
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 105
=================================================================
Total params: 5,145
Trainable params: 5,145
Non-trainable params: 0
_________________________________________________________________
None
```

表 7.14 定义一个Stacked LSTM模型的例子

7.4 拟合模型

现在我们可以在随机生成的阻尼正弦波例子的数据集上拟合模型。该模型推广了一种阻尼正弦波时间序列的最后几个时间步长的解决方案。我们可以生成一个小的例子，并在多个时间的UI及例子上拟合模型。缺点是，模型会看到相同的随机例子很多次，并可能试图记住它们。

另一个方面，我们可以生成大量的随机实例，并在该数据集的一个周期（epoch）上建立模型。这将需要更多的内存，但可能提供更快的训练和更广义的解决方案。这就是我们要使用的方法。

我们将产生10000个随机阻尼正弦波的例子来对模型进行建模，并且使用该数据集的周期（epoch）模型。这就像是10000个周期（epoch）的模型。理想情况下，通过将批处理大小（batch size）设置为1，我们将在每个样本之后重置模型的内部状态。在这种情况下，我们将权衡训练速度的纯度，并将批次大小（batch size）设置为10。这将意味着模型权重将被更新，并且在每10个样本之后将重置LSTM存储单元内部状态。

```
# fit model
X, y = generate_examples(length, 10000, output)
model.fit(X, y, batch_size=10, epochs=1)
```

表 7.15 拟合定义Stacked LSTM模型的例子

训练可能需要几分钟，因此，进度条将显示为模型拟合。如果这在笔记本或者开发环境中引起问题，可以通过调用fit()函数时设置verbose=0来将其关闭。

```
10000/10000 [==============================] - 169s - loss: 0.0481
```

表 7.16 拟合Stacked LSTM模型的输出的例子

7.5 评价模型

一旦模型拟合，我们可以评价它。这里，我们生成一个新的1000个随机序列的新集合，并报出平均绝对误差（MAE）。

```
# evaluate model
X, y = generate_examples(length, 1000, output)
loss = model.evaluate(X, y, verbose=0)
print('MAE: %f' % loss)
```

表 7.17 评估拟合Stacked LSTM模型的例子

评估模型显示模型的技能（skill）在0.02MAE左右。由于神经网络的随机性，特定技能（skill）得分在运行时可能会有所不同。

```
MAE: 0.021665
```

表 7.18 评估Stacked LSTM模型输出的例子

这有助于比较模型和根据它们的技能（skill）进行的模型配置，但是很难弄清楚到底发生了什么。

7.6 用模型做预测

我们可以通过生成一个独立的预测并将其与预期的输出序列作图来更好地了解模型的技巧。我们可以调用generate examples() 函数来生成一个例子，然后使用拟合模型来预测。预测和预期的序列被绘制出来进行比较。

```
# prediction on new data
X, y = generate_examples(length, 1, output)
yhat = model.predict(X, verbose=0)
pyplot.plot(y[0], label= y )
pyplot.plot(yhat[0], label= yhat )
pyplot.legend()
pyplot.show()
```

表 7.19 用拟合Stacked LSTM模型进行预测并绘制结果的例子

生成的图像显示，至少对于这次运行和在这个特定的例子中，预测对于预期的序列看来是合理的。

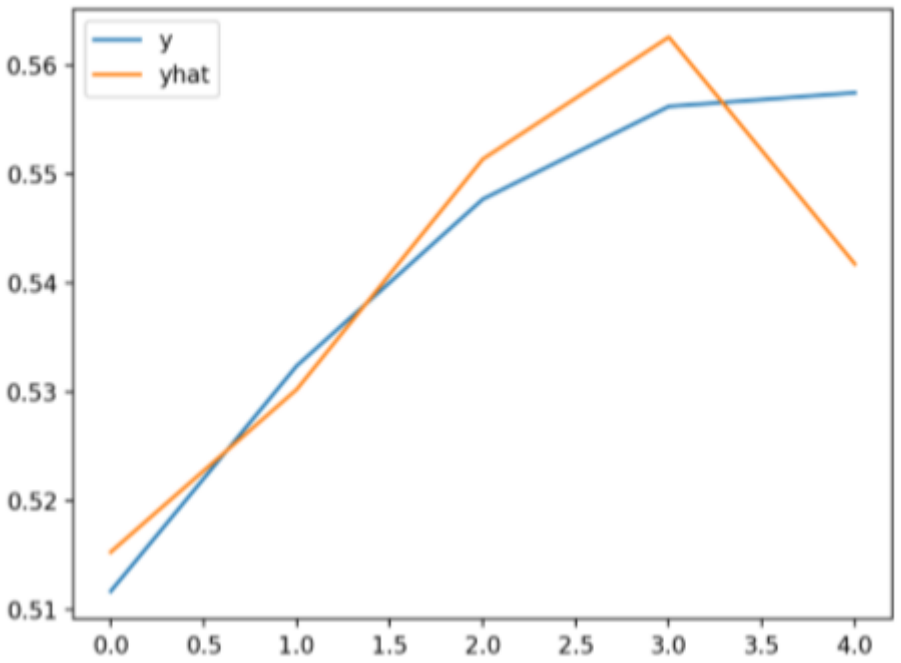

图 7.6 期望值vs预测值的线图

7.7 完整的例子

完整的例子在下面列出来了供你参考：

```
from math import sin
from math import pi
from math import exp
from random import random
from random import randint
from random import uniform
from numpy import array
from matplotlib import pyplot
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense


# generate damped sine wave in [0,1]
def generate_sequence(length, period, decay):
    return [0.5 + 0.5 * sin(2 * pi * i / period) * exp(-decay * i) for i in
range(length)]

# generate input and output pairs of damped sine waves
def generate_examples(length, n_patterns, output):
    X, y = list(), list()
    for _ in range(n_patterns):
        p = randint(10, 20)
        d = uniform(0.01, 0.1)
        sequence = generate_sequence(length + output, p, d)
```

```
        X.append(sequence[:-output])
        y.append(sequence[-output:])
        X = array(X).reshape(n_patterns, length, 1)
        y = array(y).reshape(n_patterns, output)
    return X, y

# configure problem
length = 50
output = 5

# define model
model = Sequential()
model.add(LSTM(20, return_sequences=True, input_shape=(length, 1)))
model.add(LSTM(20))
model.add(Dense(output))
model.compile(loss='mae', optimizer='adam')
print(model.summary())

# fit model
X, y = generate_examples(length, 10000, output)
history = model.fit(X, y, batch_size=10, epochs=1)

# evaluate model
X, y = generate_examples(length, 1000, output)
loss = model.evaluate(X, y, verbose=0)
print('MAE: %f' % loss)

# prediction on new data
X, y = generate_examples(length, 1, output)
yhat = model.predict(X, verbose=0)
pyplot.plot(y[0], label='y')
pyplot.plot(yhat[0], label='yhat')
pyplot.legend()
pyplot.show()
```

表 7.20 完整的LSTM模型在阻尼符号波上的工作实例

7.8 扩展阅读

本章节提供了一些扩展阅读的资料。

7.8.1 研究论文

- How to Construct Deep Recurrent Neural Networks, 2013.
- Training and Analyzing Deep Recurrent Neural Networks, 2013.
- Speech Recognition With Deep Recurrent Neural Networks, 2013.
- Generating Sequences With Recurrent Neural Networks, 2014.

7.8.2 论文

- Sine Wave on Wikipedia.
- Damped Sine Wave on Wikipedia.

7.9 扩展

你想进一步深入地学习Stacked LSTM模型吗？本章节列出了本课程中一些具有挑战性的扩展：

- 列出5个你认为可能是Stacked LSTM的好例子；
- 调整存储单元的数量、批大小（batch size）和训练样本的数量，以进一步降低模型误差（例如，尝试50K个样本和批大小为1）。
- 对某一个问题开发一个Vanilla LSTM并比较模型的性能；
- 设计并执行一个实验，用阻尼正弦波序列的长度增加训练和/或记忆细胞所需的增加量；
- 设计一种适合于Stacked LSTM的新的序列预测问题，并设计一个Stacked LSTM模型来巧妙地解决它。

把你的扩展放到网上，并把链接分享给我，我很想知道你是怎么样想的！

7.10 总结

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

- 创造一个多层LSTM的动机以及怎么样在Keras中开发一个Stacked LSTM模型；
- 阻尼正弦波预测问题以及怎么样准备拟合LSTM模型的例子；
- 怎么样开发、拟合和评估一个Stacked LSTM模型用于阻尼正弦波预测问题。

在下面的课程中，你将会学习到怎么样开发和评价CNN LSTM模型。