教你打造股市晴雨表——通过LSTM神经网络预测股市

【方向】 2017-01-08 20:01:54 浏览16007 评论0

python 机器学习 函数 HTTPS 序列 数组 神经网络 LSTM 时序预测

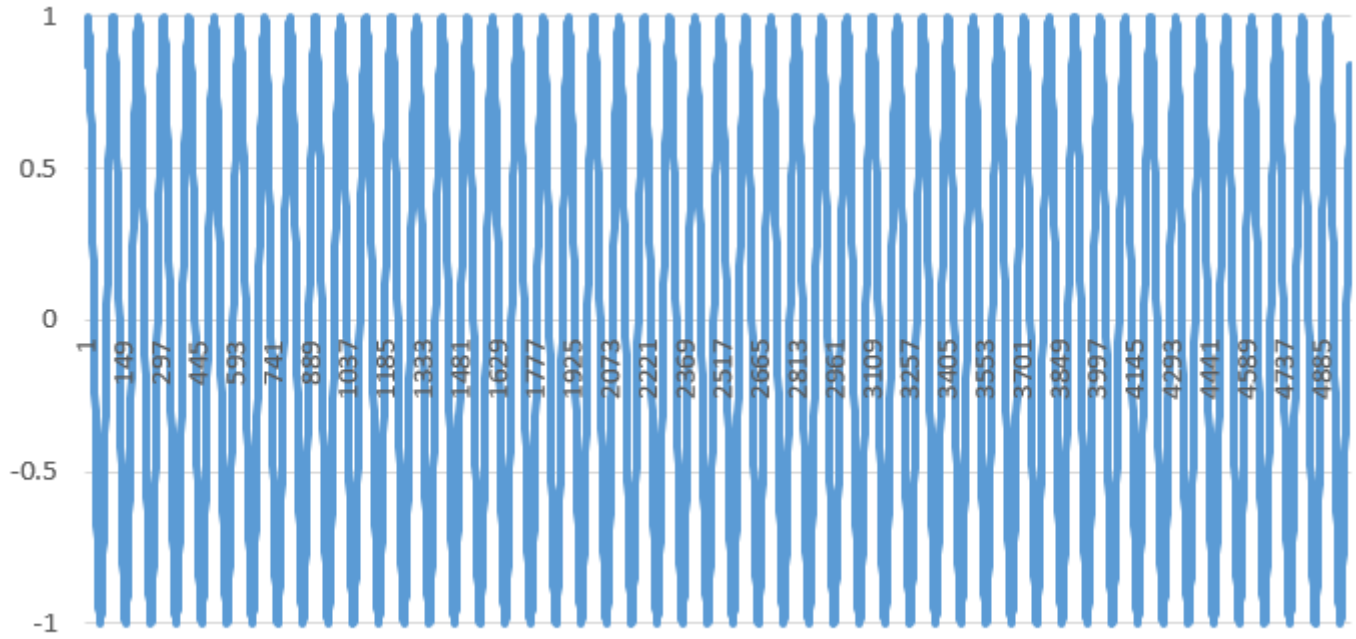
摘要： 神经网络是机器学习中的热门话题。但是网络上有关LSTM在时间序列上的应用却很少，我们不妨透过本文来开拓LSTM的应用视野。

作者介绍：
Jakob Aungiers 现就职于汇丰银行伦敦总部，担任全球资产管理的开发副总裁。擅长机器学习，神经网络等领域。

(以下为译文)
谈及机器学习，神经网络无疑是当前的热门话题。因此，在网络上围绕神经网络的教程和社区多不胜数。现在虽然有大量的公共研究论文和文章涉及LSTM，但我发现，这些理论和例子并没有显示出LSTM在时间序列预测上的真正实力。有鉴于此，我决定以本文作抛砖引玉之用，使用LSTM来预测一些时间序列—例如股市（使用Keras包，对应Python版本为2.7）。
此项目的完整代码可以在GitHub页面上找到（链接）。

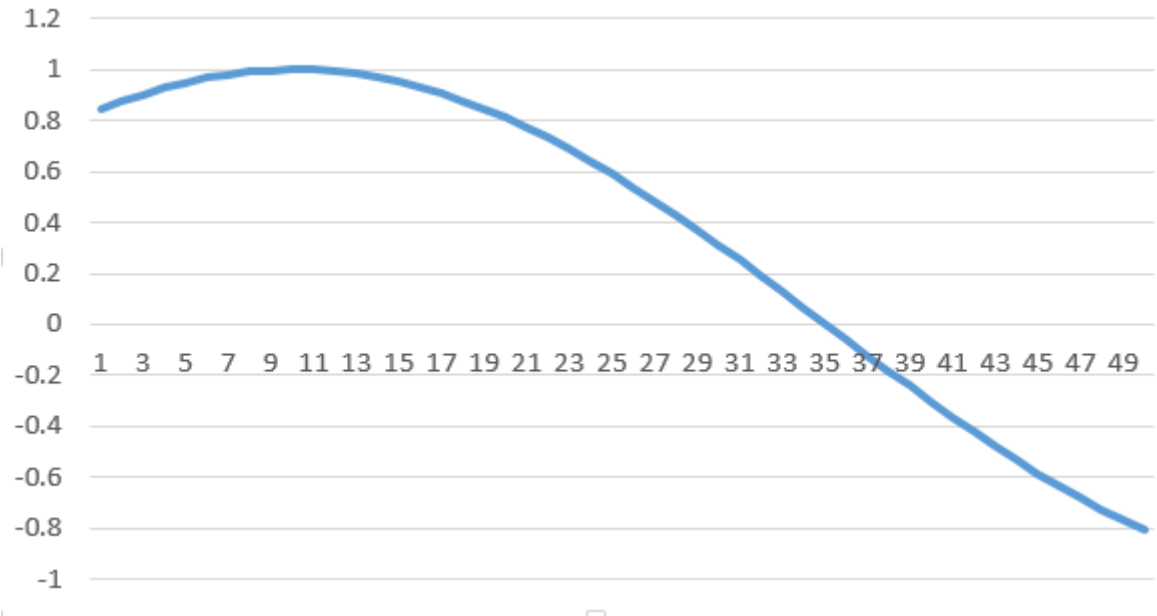
简单的正弦波

让我们从最基本的事情开始----标准正弦波函数。首先创建数据，然后模拟这个函数的多个振荡，以便进行LSTM网络训练。我做了一个excel电子表格，这是一个幅度和频率为1（给出角频率为6.28）的正弦波，我使用该函数获得超过5001个数据点的时间段，时间增量为0.01。其结果看起来是这样的：
(用作训练/测试文件的链接在这里）



这些数据有什么用呢？LSTM将从这一组窗口大小数据值来学习正弦波，然后LSTM会据此来进行时序预测以画出n步后的波形图。

我们首先从CSV文件中转换和载入数据到numpy数组中，然后喂食LSTM。Keras LSTM层的工作方式是通过接收3维（N，W，F）的数字阵列，其中N是训练序列的数目，W是序列长度，F是每个序列的特征数目。我选择的一个序列长度（读取窗口大小）为50，这可让网络在每个序列中知道正弦波的形状，从而教会自身基于先前窗口信息的前提下建立序列的模式。序列本身是滑动的窗口，因此每次移动1长度后，会保持与先前窗口的恒定重叠。



长度为50的序列的示例

下面是将训练数据CSV加载到正确形状numpy数组中的代码：

```
def load_data(filename):
```

```
def load_data(filename):



    f = open(&#39;sinwave.csv&#39;, &#39;rb&#39;).read()
    data = f.split(&#39;\r\n&#39;)

    sequence_length = 50
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])

    result = np.array(result)

    row = round(0.9 * result.shape[0])
    train = result[:row, :]
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = result[row:, :-1]
    y_test = result[row:, -1]

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

    return [x_train, y_train, x_test, y_test]
```

接下来，我们需要实际构建网络本身。我使用了[1,50,100,1]的网络结构，其中我们有1个输入层（由大小为50的序列组成），该输入层喂食50个神经元给LSTM层，接着该LSTM层喂食100个神经元给另一个LSTM层，然后使用一个线性激活函数来喂食一个完全连接的正常层以用于下一个时间步的预测。

模型构建函数代码如下：

```
def build_model(layers):



    model = Sequential()

    model.add(LSTM(
        input_dim=layers[0],
        output_dim=layers[1],
        return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(
        output_dim=layers[3]))
```

```
    model.add(Activation(&quot;linear&quot;))

    start = time.time()
    model.compile(loss=&quot;mse&quot;, optimizer=&quot;rmsprop&quot;)
    print &quot;Compilation Time : &quot;, time.time() - start
    return model
```

最后是训练网络上的数据，看看我们得到了什么。在该LSTM我只使用了1个训练时期，这有别于需要大量训练数据的传统网络。因为我们使用的是简的具可预测模式的正弦波，1训练时期将足够获得非常近似的全sin函数。

放入run.py模块后的代码：

```
epochs  = 1
seq_len = 50
```

print 'Loading data... '

X_train, y_train, X_test, y_test = lstm.load_data('sinwave.csv')

print 'nData Loaded. Compiling...n'

model = lstm.build_model([1, 50, 100, 1])

model.fit(

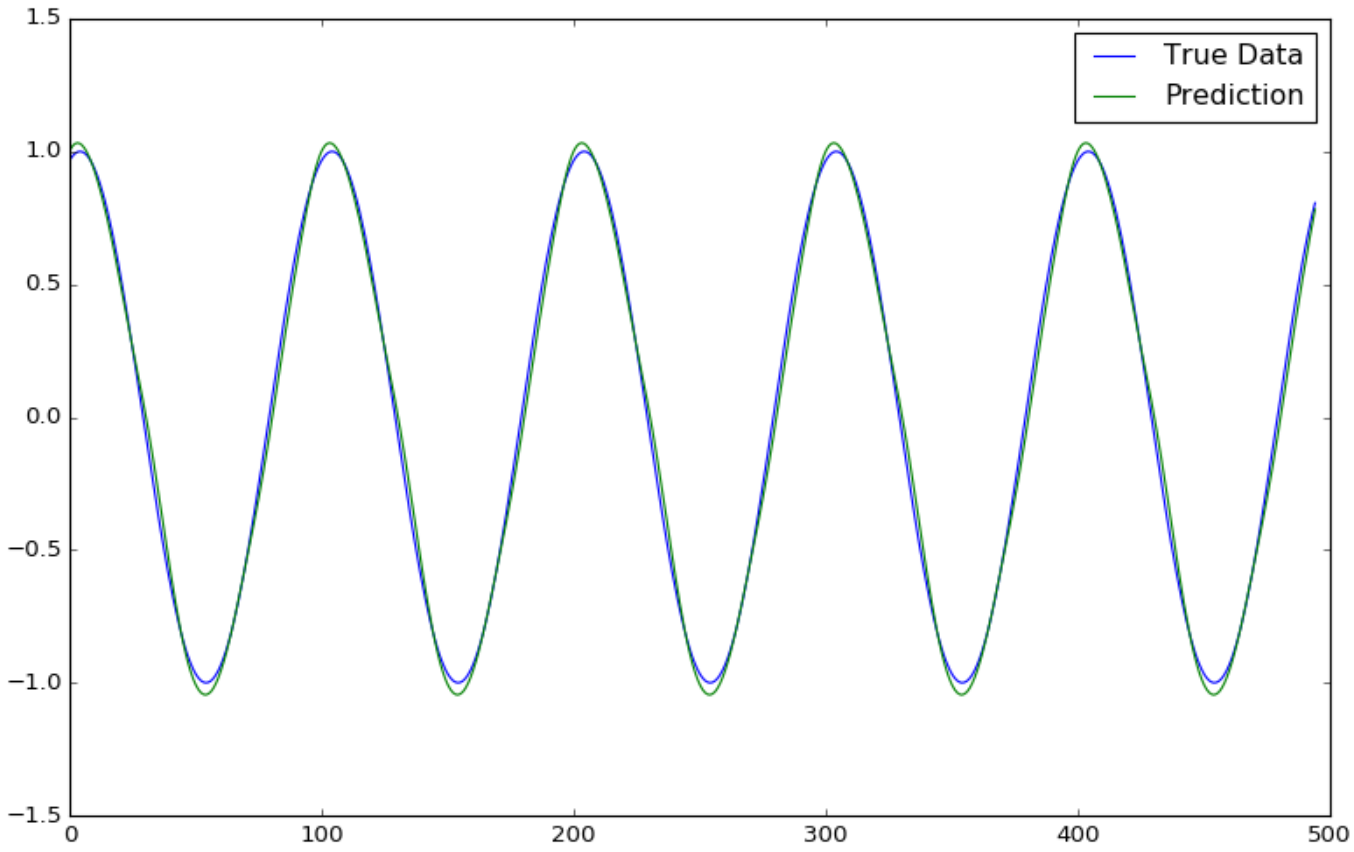
```
    X_train,
    y_train,
    batch_size=512,
    nb_epoch=epochs,
    validation_split=0.05)
```

predicted = lstm.predict_point_by_point(model, X_test)

如果你注意到你会注意到我们在上面的load_data（ ）函数中，我们将数据分为训练/测试集，这是机器学习问题的标准做法。然而，我们需要注意的是，我们实际上想要在时间序列实现预测。

如果我们使用测试集，我们将运行每个窗口的真实数据来预测下一个时间步。下图是使用该方法后

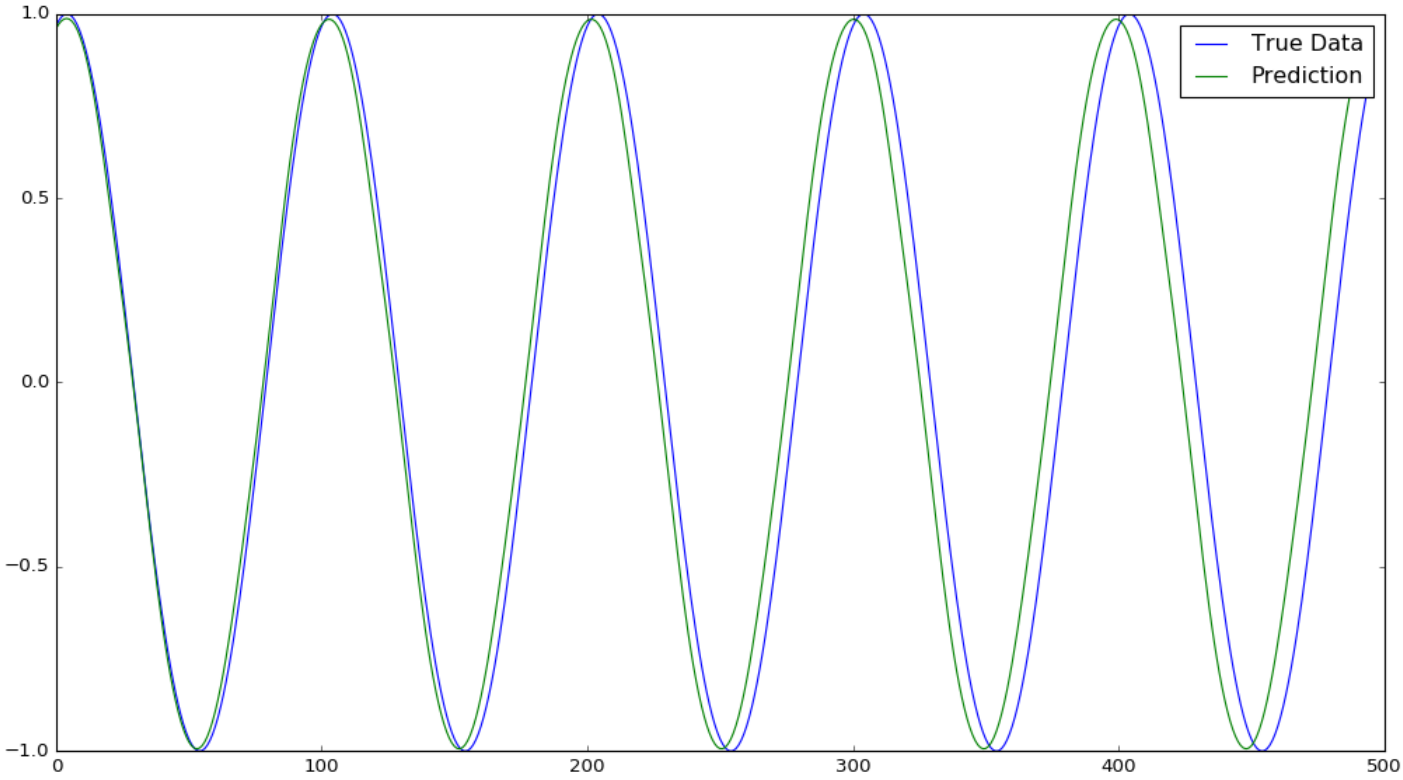
的结果：



epochs = 1, window size = 50

然而，如果我们想要预测许更多的时间步，我们需使用来自测试数据的第一个窗口作为启动窗口。在每个时间步，弹出窗口后面的最先条目，并将下一个时间步预测附加到窗口的前面，实质上是移动窗口，所以它是慢慢地用预测建立自己，直到窗口都是预测值（在我们的例子中，因为我们的窗口大小为50，这将发生在50个时间步之后）。 然后，我们无限期地保持这一点，预测下一步对未来时间步骤的预测，从而实现看到趋势预测。

下图显示了仅从真实测试数据的初始开始窗口预测的正弦波时间序列，然后预测约500步：



epochs = 1, window size = 50

除了简单的正弦波预测，LSTM还能做更复杂的预测吗？答案是肯定的，例如以下的有关股票市场的时间序列预测。什么？！股票预测！！是的。这是LSTM另一个技能--潜在隐藏趋势预判。

首先我准备了一个CSV文件(链接)，其保存的是标准普尔500股权指数从2000年1月到2016年8月的收盘数据。我使它与我们的正弦波数据具完全相同的格式，然后运行相同的模型。

这之前我们需要对我们的数据做一个微小的改变，因为正弦波已经是一个很规范的重复模式，但股票数据是不规范的。因此为了应对这种情况，我们需要使训练/测试数据的每个n大小的窗口进行标准化，以反映从该窗口开始的百分比变化（因此点i = 0处的数据将始终为0）。我们将使用以下方程式进行归一化，然后在预测过程结束时进行反标准化，以得到预测中的真实世界数：

n =价格变化的标准化列表[窗口]
p =原始列表[窗口]调整的每日回报价格
标准化公式：
$n_i = (p_i/p_0\text{-}1)$

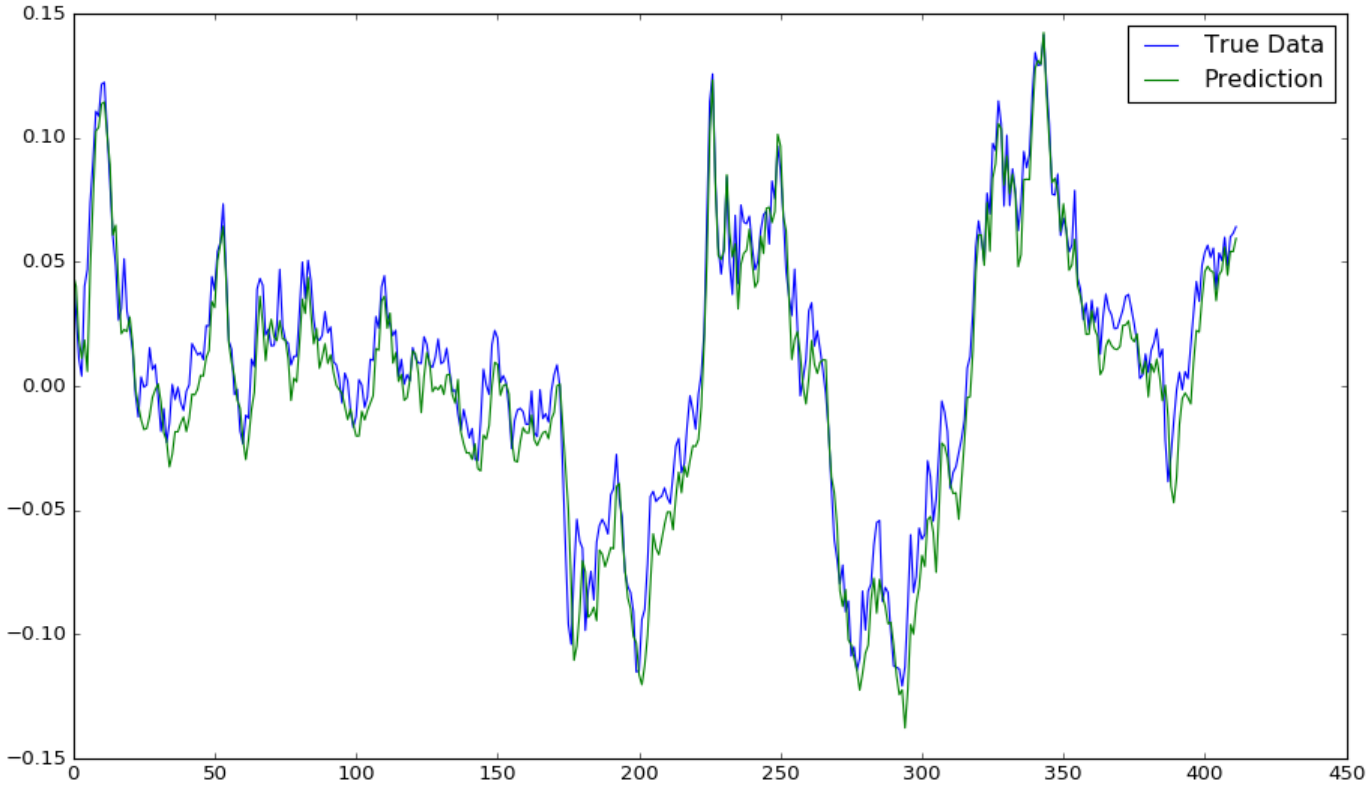
反标准化公式：
$p_i = p_0(n_i + 1)$

在代码中添加一个normalise_windows（window_data）函数，并更新load_data（filename）函数以包含一个条件调用，以及获取序列长度和规范化标志load_data（filename，seq_len，normalise_window）：

代码如下：

```
def load_data(filename, seq_len, normalise_window):


    f = open(filename, &#39;rb&#39;).read()
    data = f.split(&#39;\r\n&#39;)

    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])

    if normalise_window:
        result = normalise_windows(result)

    result = np.array(result)

    row = round(0.9 * result.shape[0])
    train = result[:row, :]
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = result[row:, :-1]
    y_test = result[row:, -1]

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

    return [x_train, y_train, x_test, y_test]


def normalise_windows(window_data):


    normalised_data = []
    for window in window_data:
        normalised_window = [((float(p) / float(window[0])) - 1) for p in window]
        normalised_data.append(normalised_window)
    return normalised_data
```

归一化了如上所述的窗口后，我们现在可以通过LSTM网络运行我们的股票数据。 让我们看看它的运行情况：

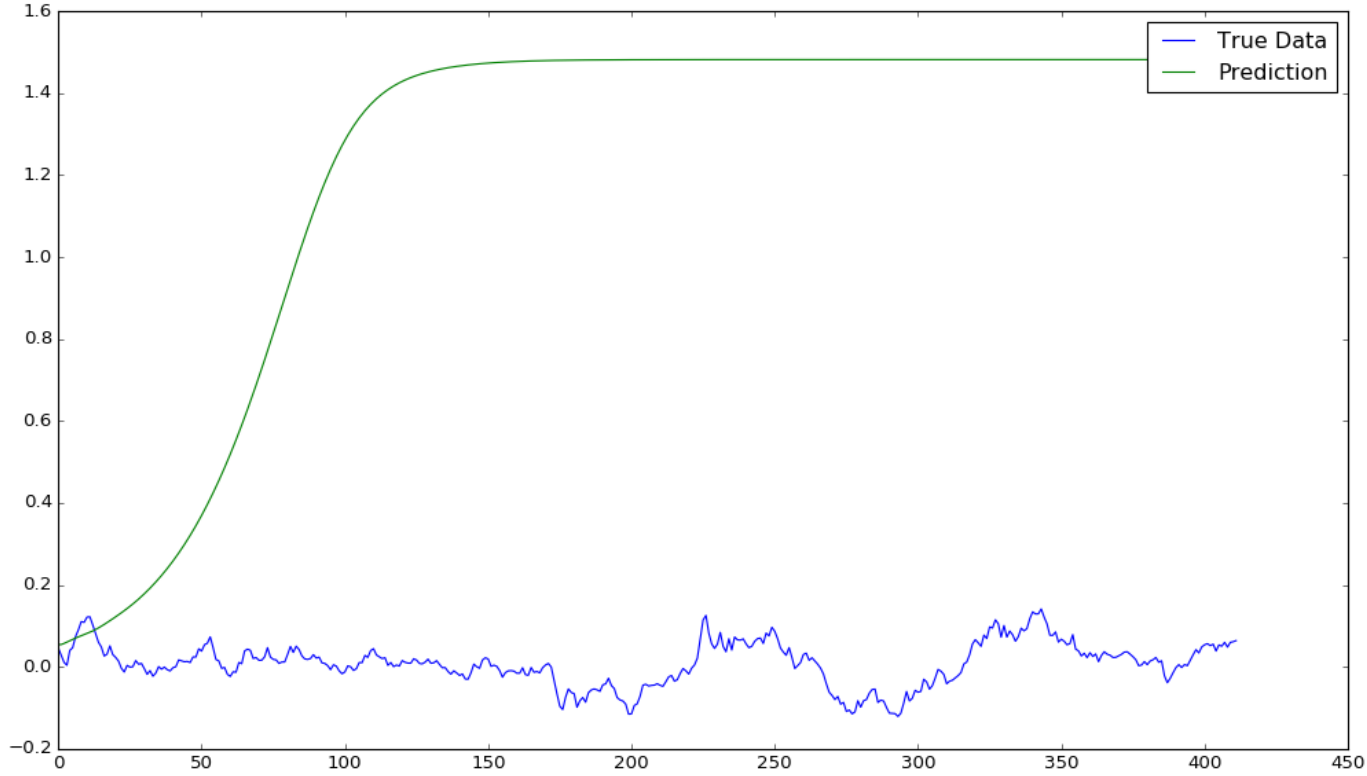


在如上所述的单个逐点预测上运行数据给出了与实际相当接近的图形。但这是欺骗性的！为什么？如果你更仔细地看，预测线由单一的预测点组成，它们在它们之后具有整个先前的真实历史窗口。因此，网络并不依赖时间序列本身，除了每下一个点可能不会离最后一点太远。因此，即使它得到错误预测的点，下一个预测可在考虑真实历史后忽略不正确的预测，并允许再次出现错误。

那么，我们来看看是否真的有一些潜在的模式在价格中变动切可辨别？接下来让我们做对正弦波问题做的同样事情，让网络预测点序列而不是下一个点。

这样，我们现在可以看到，与作为正弦波序列的正弦波不同，它与真实数据几乎相同，我们的股票

数据预测很快地收敛到某种平衡。



epochs = 50, window size = 50

epochs = 1, window size = 50

让我们进一步研究回归收敛，将我们的预测序列限制到50个未来时间步长，然后每次将启动窗口移动50单位：

epochs = 1, window size = 50, sequence shift = 50

而当epochs增加到400时（这应该使模型模式更准确），我们看到，实际上它现在只是试图预测几乎每个时间段的向上动量。

epochs = 400, window size = 50, sequence shift = 50

小结：

LSTM的应用日益广泛，例如文本预测，AI智能聊天，自驾车等许多前沿领域。 希望本文能有助你开拓LSTM在时间序列中应用的视野。

本文由北邮@爱可可-爱生活 老师推荐，阿里云云栖社区组织翻译。

文章原标题《LSTM NEURAL NETWORK FOR TIME SERIES PREDICTION》，作者：Jakob Aungiers，译者：伍昆