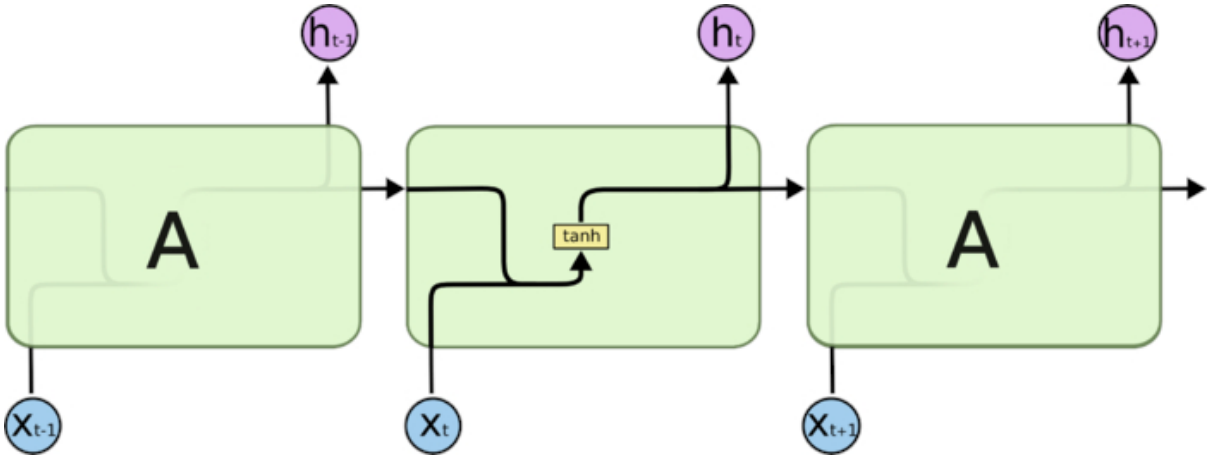
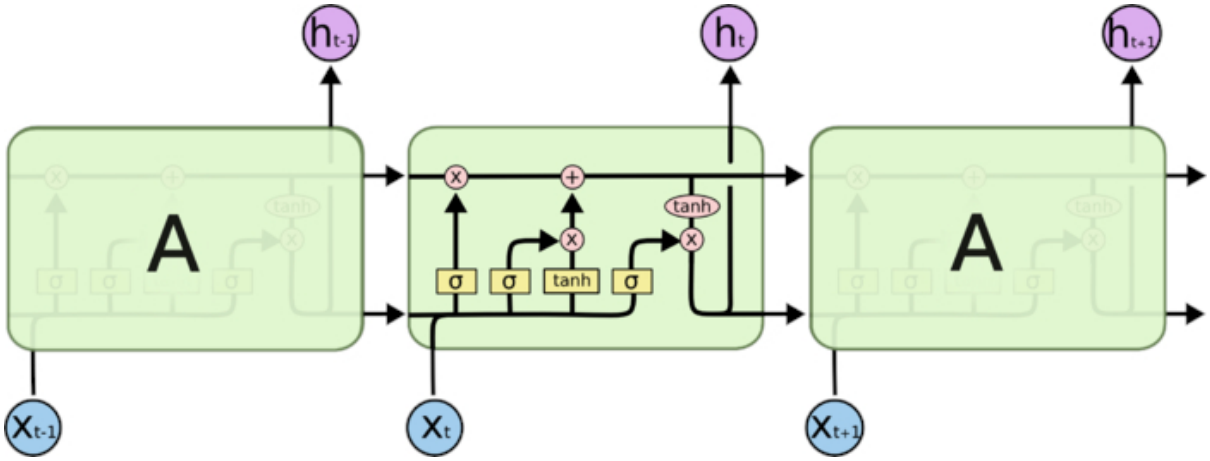
AI学习笔记--sklearn--lstm算法

- ## LSTM网络

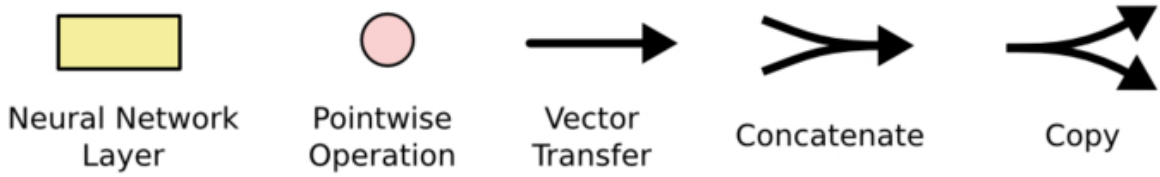
long short term memory(LSTM)，即LSTM算法全称，**是为了解决长短期记忆网络而专门设计出来的，LSTM** 已经在科技领域有了多种应用。基于 **LSTM** 的系统可以学习翻译语言、控制机器人、图像分析、文档摘要、语音识别图像识别、手写识别、控制聊天机器人、预测疾病、点击率和股票、合成音乐等等任务。所有的RNN都具有一种重复神经网络模块的链式形式。在标准RNN中，这个重复的结构模块只有一个非常简单的结构，例如一个tanh层。



LSTM 同样是这样的结构，但是重复的模块拥有一个不同的结构。不同于单一神经网络层，这里是有四个，以一种非常特殊的方式进行交互。



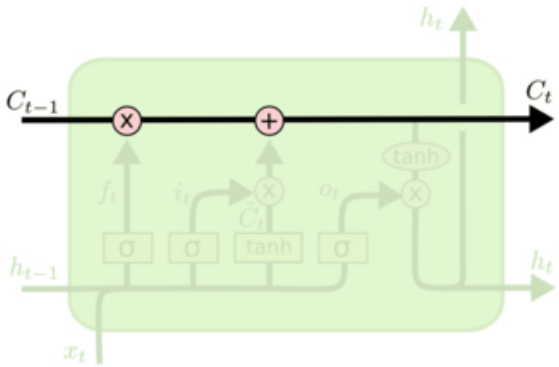
进一步解析LSTM数据结构：



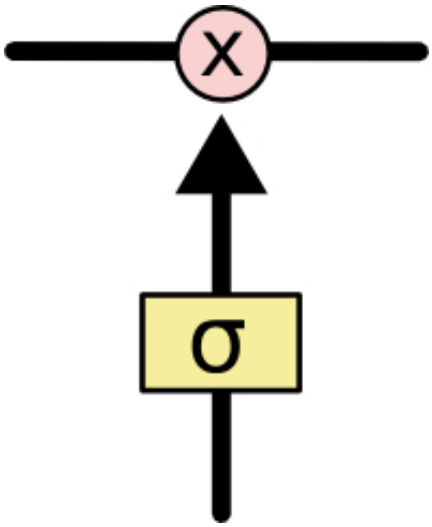
在上面的图例中，每一条黑线传输着一整个向量，从一个节点的输出到其他节点的输入。粉色的圈代表 pointwise 的操作，诸如向量的和，而黄色的矩阵就是学习到的神经网络层。合在一起的线表示向量的连接，分开的线表示内容被复制，然后分发到不同的位置。

- # LSTM核心思想

LSTM的关键在于细胞的状态整个(绿色的图表示的是一个cell)，和穿过细胞的那条水平线。细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变会很容易。



若只有上面的那条水平线是没办法实现添加或者删除信息的。而是通过一种叫做 门（gates） 的结构来实现的。门可以实现选择性地让信息通过，主要是通过一个 sigmoid 的神经层 和一个逐点相乘的操作来实现的。



sigmoid 层输出（是一个向量）的每个元素都是一个在 0 和 1 之间的实数，表示让对应信息通过的权重（或者占比）。比如， 0 表示“不让任何信息通过”， 1 表示“让所有信息通过”。

LSTM通过三个这样的本结构来实现信息的保护和控制。这三个门分别输入门、遗忘门和输出门。

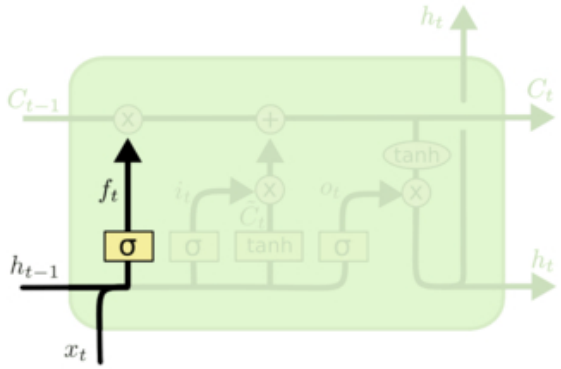
逐步理解LSTM

现在我们就开始通过三个门逐步的了解LSTM的原理。

遗忘门

在我们 LSTM 中的第一步是决定我们会从细胞状态中丢弃什么信息。这个决定通过一个称为忘记门层完成。该门会读取ht−1ht−1和xtxt，输出一个在 0到 1之间的数值给每个在细胞状态 Ct−1Ct−1 中的数字。1 表示“完全保留”，0 表示“完全舍弃”。

让我们回到语言模型的例子中来基于已经看到的预测下一个词。在这个问题中，细胞状态可能包含当前主语的性别，因此正确的代词可以被选择出来。当我们看到新的主语，我们希望忘记旧的主语。



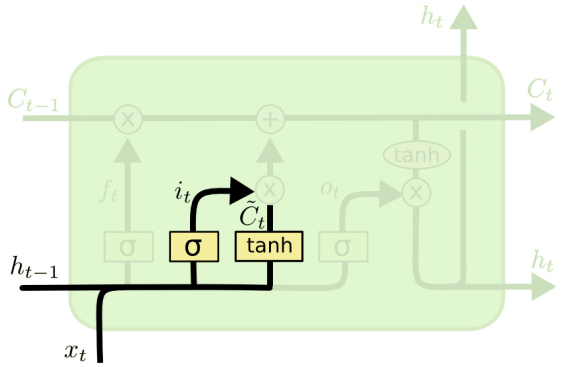
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

其中ht−1表示的是上一个cell的输出，xt表示的是当前细胞的输入。σ表示sigmod函数。

输入门

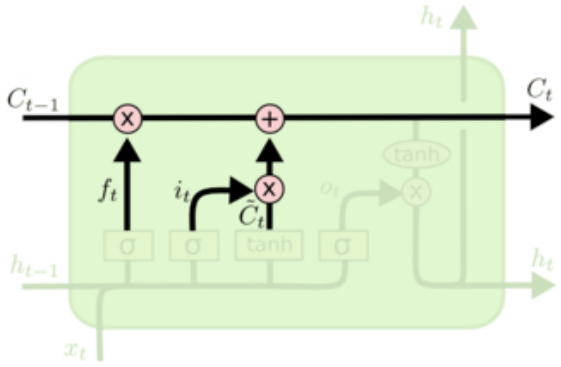
下一步是决定让多少新的信息加入到 cell 状态 中来。实现这个需要包括两个 步骤：首

先，一个叫做“input gate layer ”的 sigmoid 层决定哪些信息需要更新；一个 tanh 层生成一个向量，也就是备选的用来更新的内容，Ĉ tC^t 。在下一步，我们把这两部分联合起来，对 cell 的状态进行一个更新。



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

现在是更新旧细胞状态的时间了，Ct−1Ct−1更新为CtCt。前面的步骤已经决定了将会做什么，我们现在就是实际去完成。我们把旧状态与ftft相乘，丢弃掉我们确定需要丢弃的信息。接着加上it∗C̃ tit∗C~t。这就是新的候选值，根据我们决定更新每个状态的程度进行变化。在语言模型的例子中，这就是我们实际根据前面确定的目标，丢弃旧代词的性别信息并添加新的信息的地方。

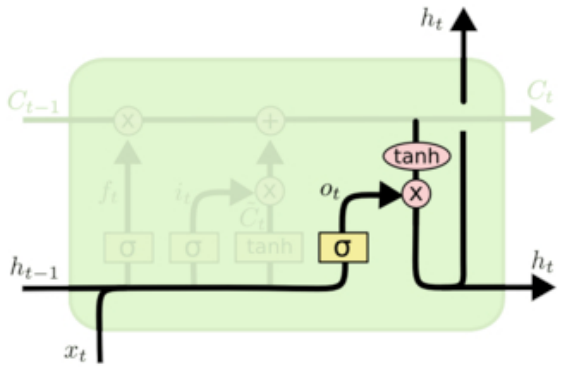


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- **输出门**

最终，我们需要确定输出什么值。这个输出将会基于我们的细胞状态，但是也是一个过滤后的版本。首先，我们运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，我们把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。

在语言模型的例子中，因为他就看到了一个 代词，可能需要输出与一个 动词 相关的信息。例如，可能输出是否代词是单数还是负数，这样如果是动词的话，我们也知道动词需要进行的词形变化。

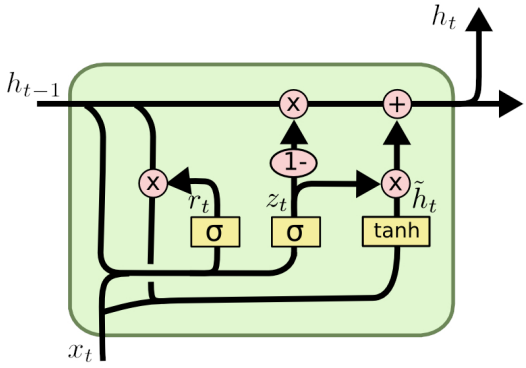


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- **LSTM变体**

原文这部分介绍了 LSTM 的几个变种，还有这些变形的作用。在这里我就不再写了。有兴趣的可以直接阅读原文。

下面主要讲一下其中比较著名的变种 GRU（Gated Recurrent Unit ），这是由 Cho, et al. (2014) 提出。在 GRU 中，如下图所示，只有两个门：重置门（reset gate）和更新门（update gate）。同时在这个结构中，把细胞状态和隐藏状态进行了合并。最后模型比标准的 LSTM 结构要简单，而且这个结构后来也非常流行。



$$z_t = \sigma\left(W_z \cdot \left[h_{t-1}, x_t\right]\right)$$
$$r_t = \sigma\left(W_r \cdot \left[h_{t-1}, x_t\right]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot \left[r_t * h_{t-1}, x_t\right]\right)$$
$$h_t = \left(1 - z_t\right) * h_{t-1} + z_t * \tilde{h}_t$$

其中， rtrt表示重置门，ztzt表示更新门。重置门决定是否将之前的状态忘记。(作用相当于合并了 LSTM 中的遗忘门和传入门）当rtrt趋于0的时候，前一个时刻的状态信息ht−1ht−1会被忘掉，隐藏状态ĥ th^t会被重置为当前输入的信息。更新门决定是否要将隐藏状态更新为新的状态ĥ th^t（作用相当于 LSTM 中的输出门） 。

和 LSTM 比较一下：

- GRU 少一个门，同时少了细胞状态CtCt。
- 在 LSTM 中，通过遗忘门和传入门控制信息的保留和传入；GRU 则通过重置门来控制是否要保留原来隐藏状态的信息，但是不再限制当前信息的传入。
- 在 LSTM 中，虽然得到了新的细胞状态 Ct，但是还不能直接输出，而是需要经过一

个过滤的处理:ht=ot∗tanh(Ct)ht=ot∗tanh(Ct)；同样，在 GRU 中, 虽然我们也得到了新的隐藏状态ĥ th^t， 但是还不能直接输出，而是通过更新门来控制最后的输出：ht=(1−zt)∗ht−1+zt∗ĥ tht=(1−zt)∗ht−1+zt∗h^t

- # LSTM Python demo

编写代码：

```
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

# load the dataset
# dataframe = read_csv('1.csv', usecols=[1], engine='python', skipfooter=3)
dataframe = read_csv('123.csv', usecols=[0], engine='python', skipfooter=3)

dataset = dataframe.values

# 将整型变为float
dataset = dataset.astype('float32')
#plt.plot(dataset,'.')
#plt.plot(dataset,'.')
#plt.show()

# X is the number of passengers at a given time (t) and Y is the number of
passengers at the next time (t + 1).
# convert an array of values into a dataset matrix

def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)

numpy.random.seed(7)
```

```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]

# use this function to prepare the train and test datasets for modeling
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)

# 转换成lstm需要的数据格式
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)) )
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=10, batch_size=1, verbose=2)

# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))

# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] =
testPredict
```

```
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot,'.')
plt.plot(testPredictPlot,'.')

plt.plot(trainPredictPlot)
plt.plot(scaler.inverse_transform(dataset)[-(len(testPredict)):])
plt.plot(testPredict)
plt.show()

# Test Score: 11.96 RMSE
```

可能出现的异常：

```
/Users/genesis/Workplace/Python_Workplace/isum.py in <module>()
      3 from pandas import read_csv
      4 import math
----> 5 from keras.models import Sequential
      6 from keras.layers import Dense
      7 from keras.layers import LSTM
ImportError: No module named keras.models

解决方案：
→  Python_Workplace pip install keras
```

实际打印结果：

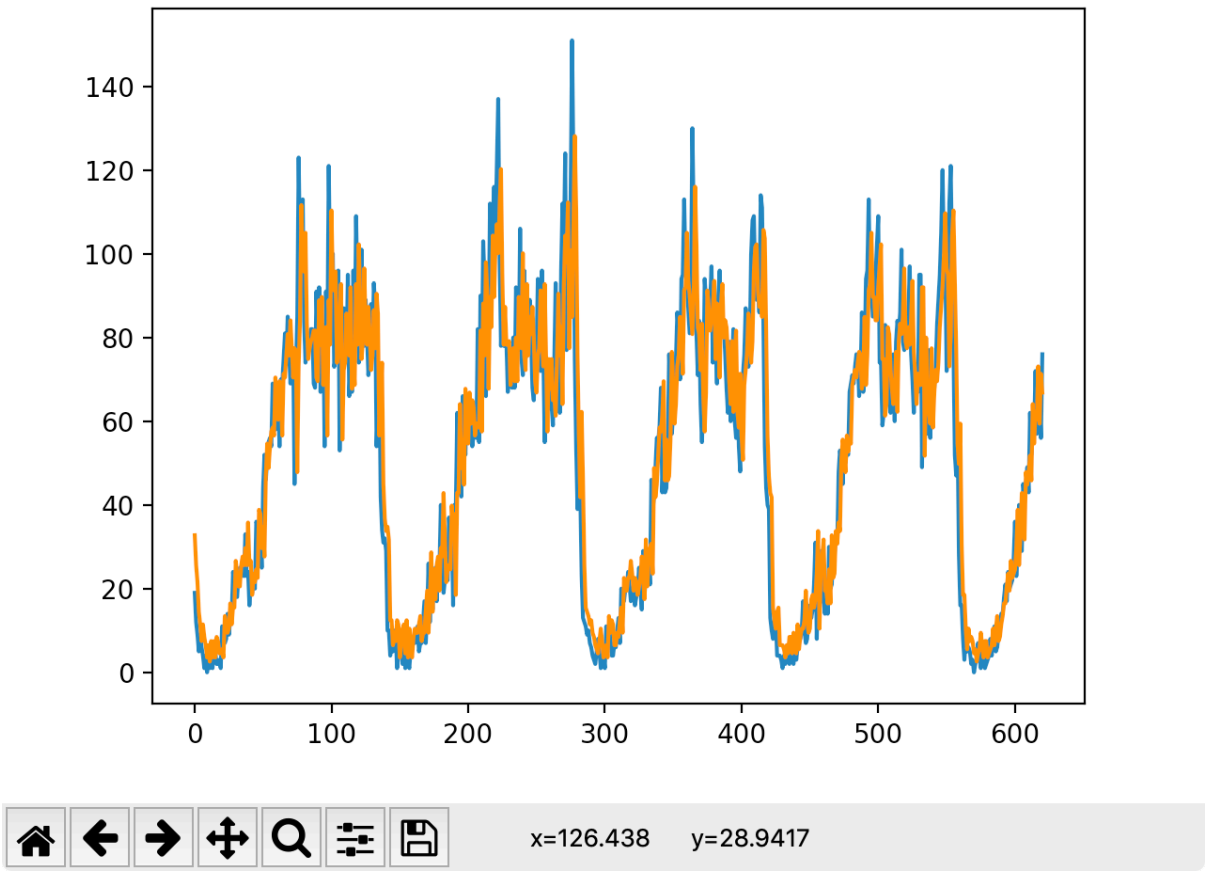
```
→  Python_Workplace ipython isum.py
Using TensorFlow backend.
Epoch 1/10
2019-02-14 17:22:52.752678: I tensorflow/core/platform/cpu_feature_guard.cc:141]
Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
 - 5s - loss: 0.0327
Epoch 2/10
 - 4s - loss: 0.0081
Epoch 3/10
 - 4s - loss: 0.0074
Epoch 4/10
 - 4s - loss: 0.0073
Epoch 5/10
 - 4s - loss: 0.0072
Epoch 6/10
 - 4s - loss: 0.0072
Epoch 7/10
 - 4s - loss: 0.0070
Epoch 8/10
 - 4s - loss: 0.0069
Epoch 9/10
 - 4s - loss: 0.0068
```

```
Epoch 10/10
 - 4s - loss: 0.0068
Train Score: 12.35 RMSE
Test Score: 12.21 RMSE
```

实验结果：

123.csv

Figure 1



- demo2

```
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import lstm, time #helper libraries
#Step 1 Load Data
X_train, y_train, X_test, y_test = lstm.load_data('sp500.csv', 50,
True)
#Step 2 Build Model
model = Sequential()
```

```
model.add(LSTM(
    input_dim=1,
    output_dim=50,
    return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(
    100,
    return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(
    output_dim=1))
model.add(Activation('linear'))
start = time.time()
model.compile(loss='mse', optimizer='rmsprop')
#Step 3 Train the model
model.fit(
    X_train,
    y_train,
    batch_size=512,
    nb_epoch=1,
    validation_split=0.05)
#Step 4 - Plot the predictions!
predictions = lstm.predict_sequences_multiple(model, X_test, 50, 50)
lstm.plot_results_multiple(predictions, y_test, 50)
```

实验数据  sp500.csv

执行结果：

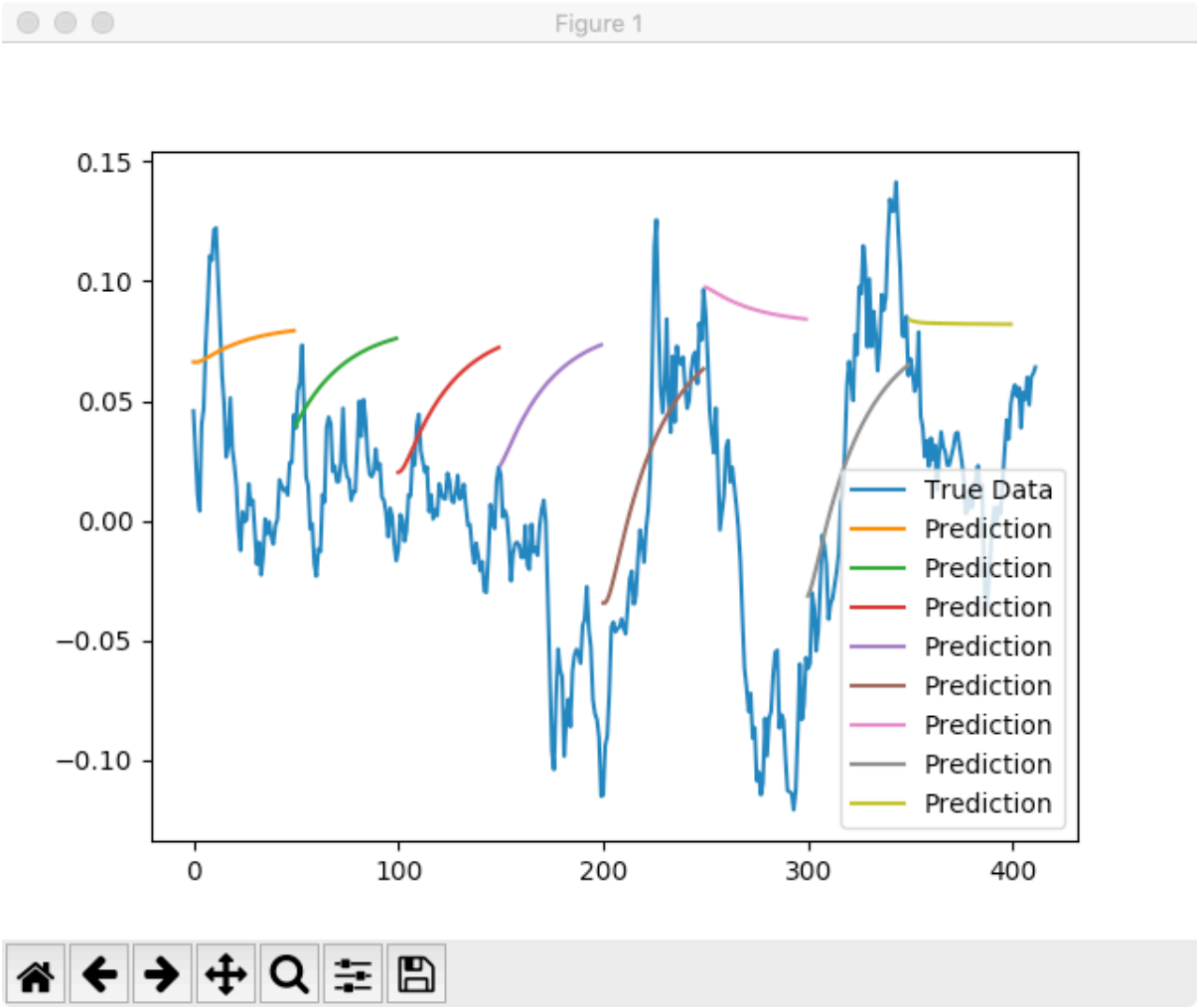


Figure 1
0.15
0.10
0.05
0.00
−0.05
−0.10
0
100
200
300
400
True Data
Prediction
Prediction
Prediction
Prediction
Prediction
Prediction
Prediction
Prediction