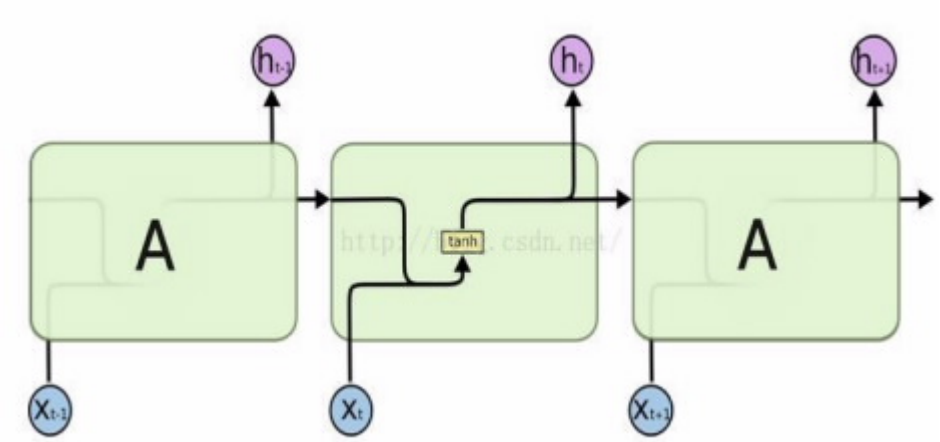
# LSTM长短时记忆网络实践

2015160053 贾鹏

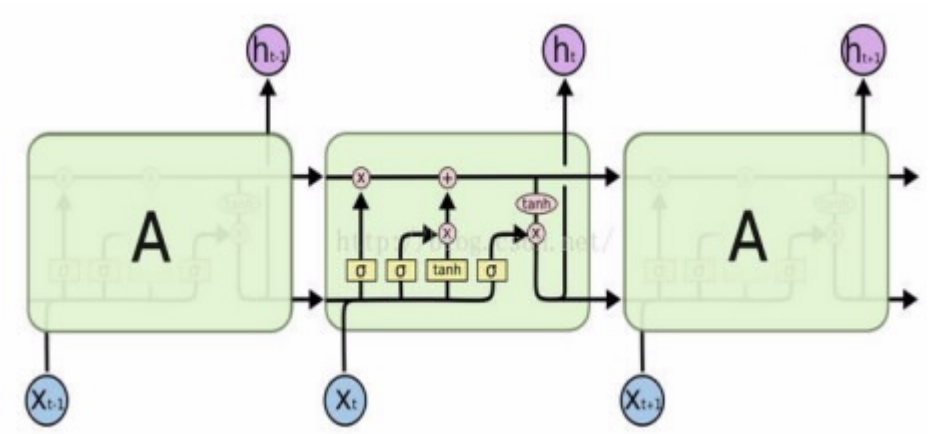
LSTM（Long Short-Term Memory）是长短期记忆网络，是一种时间递归神经网络，适合于处理和预测时间序列中间隔和延迟相对较长的重要事件。

## 1.LSTM网络：

是一种特殊的RNN（循环神经网络），可以解决长句依赖问题，标准的RNN如下：




LSTM也是这种链式的结构，只是重复单元的内部结构不一样，它不是单独的NN层，而是4个NN，这4个相互影响：

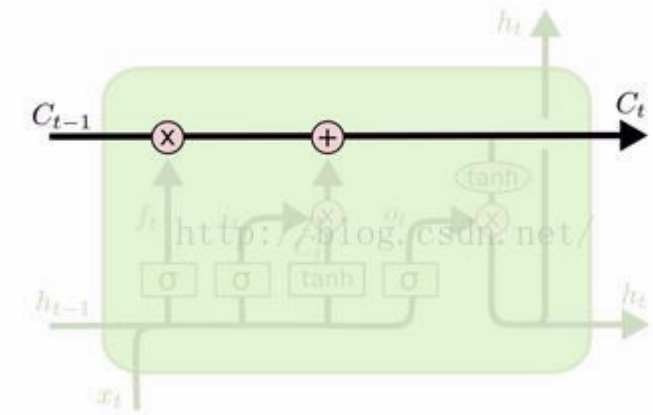


其中，

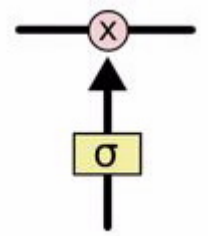

上面框图中，每一行将一个节点的整个向量输出传递到下一个结点做输入。

**2.LSTM的核心**

LSTM的重点是cell state，下面水平这条线从架构的最上面走，cell state就是传送带，整个系统就像一条长直链，只有一些线性关系，信息往下传而不会改变。



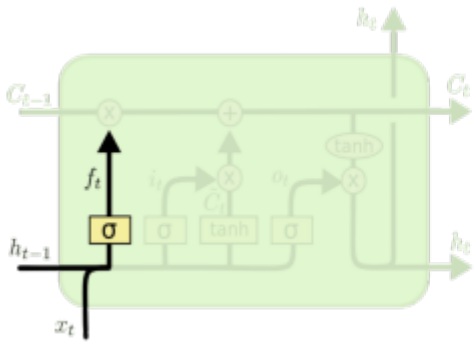

LSTM能删去或者增加信息，依靠的而是门结构。门是信息选择性通过的一种手段，门由sigmoid神经层和一个点乘单元组成。



Sigmoid层的输出介于0和1之间，输出为0表示信息完全不通过，输出为1表示信息全部通过。一个LSTM有3个这样的二门来控制和保护cell state。

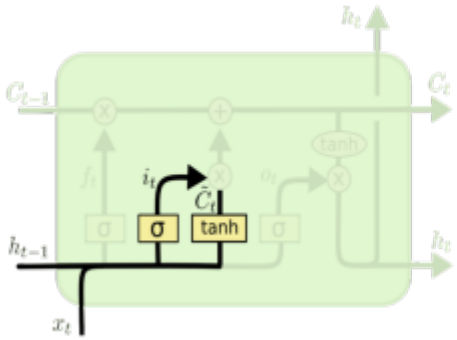
**3.逐步理解 LSTM**

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 h_{t-1} 和 x_t，输出一个在 0 到 1 之间的数值给每个在细胞状态 C_{t-1} 中的数字。1 表示“完全保留”，0 表示“完全舍弃”。决定丢弃信息。




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

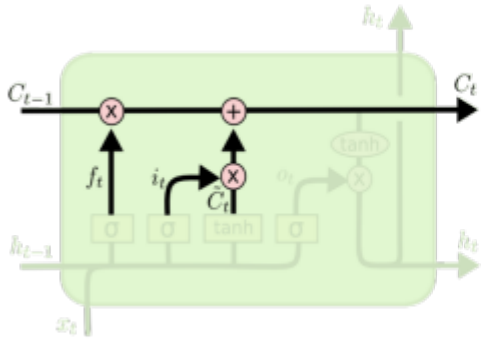
下一步是确定什么样的新信息被存放在细胞状态中。这里包含两个部分。第一，sigmoid 层称 “输入门层”决定什么值我们将要更新。然后，一个 tanh 层创建一个新的候选值向量，\tilde{C}_t，会被加入到状态中。下一步，我们会讲这两个信息来产生对状态的更新。




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

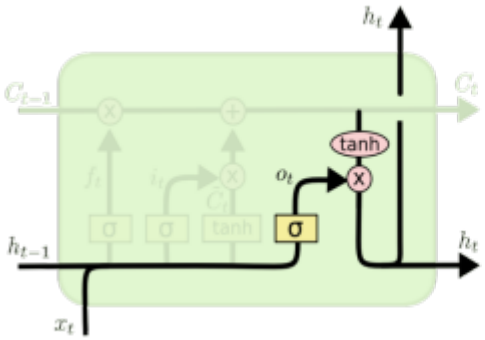
现在是更新旧细胞状态的时间了，C_{t-1}　　更新为 C_t。前面的步骤已经决定了将会做什么，我们现在就是实际去完成。

我们把旧状态与　f_t 相乘，丢弃掉我们确定需要丢弃的信息。接着加上　i_t　* \tilde{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。




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

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。




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

## 4. 使用tensorflow的lstm网络进行时间序列预测

编程环境：python3.5，tensorflow 1.0

本文所用的数据集来自于kesci平台，由云脑机器学习实战训练营提供：真实业界数据的时间序列预测挑战

数据集采用来自业界多组相关时间序列（约40组）与外部特征时间序列（约5组）。本文只使用其中一组数据进行建模。

加载常用的库：

[python] view plain copy

```
1.  <span style="font-size:14px;">#加载数据分析常用库
2.  import pandas as pd
3.  import numpy as np
4.  import tensorflow as tf
5.  from sklearn.metrics import mean_absolute_error,mean_squared_error
6.  from sklearn.preprocessing import MinMaxScaler
7.  import matplotlib.pyplot as plt
8.  % matplotlib inline
9.  import warnings
10. warnings.filterwarnings('ignore')</span>
```

数据显示：

[python] view plain copy

```
path = '../input/industry/industry_timeseries/timeseries_train_data/11.csv'
data11 = pd.read_csv(path,names=['年','月','日','当日最高气温','当日最低气温','当日平均气温','当日平均湿度','输出'])
data11.head()
```

| | 年 | 月 | 日 | 当日最高气温 | 当日最低气温 | 当日平均气温 | 当日平均湿度 | 输出 |
|---|---|---|---|---|---|---|---|---|
| 0 | 2015 | 2 | 1 | 1.9 | -0.4 | 0.7875 | 75.000 | 814.155800 |
| 1 | 2015 | 2 | 2 | 6.2 | -3.9 | 1.7625 | 77.250 | 704.251112 |
| 2 | 2015 | 2 | 3 | 7.8 | 2.0 | 4.2375 | 72.750 | 756.958978 |
| 3 | 2015 | 2 | 4 | 8.5 | -1.2 | 3.0375 | 65.875 | 640.645401 |
| 4 | 2015 | 2 | 5 | 7.9 | -3.6 | 1.8625 | 55.375 | 631.725130 |

加载数据：

[python] view plain copy

```
##load data(本文以第一个表为例，其他表类似，不再赘述)
f=open('../input/industry/industry_timeseries/timeseries_train_data/11.csv')
df=pd.read_csv(f)     #读入数据
data=df.iloc[:,3:8].values   #取第3-7列
```

定义常量并初始化权重：

[python] view plain copy

```
#定义常量
rnn_unit=10       #hidden layer units
input_size=4
output_size=1
lr=0.0006         #学习率
tf.reset_default_graph()
#输入层、输出层权重、偏置
weights={
         'in':tf.Variable(tf.random_normal([input_size,rnn_unit])),
         'out':tf.Variable(tf.random_normal([rnn_unit,1]))
         }
biases={
        'in':tf.Variable(tf.constant(0.1,shape=[rnn_unit,])),
        'out':tf.Variable(tf.constant(0.1,shape=[1,]))
        }
```

分割数据集，将数据分为训练集和验证集（最后90天做验证，其他做训练）：

[python] view plain copy

```python
def get_data(batch_size=60,time_step=20,train_begin=0,train_end=487):
    batch_index=[]

    scaler_for_x=MinMaxScaler(feature_range=(0,1))  #按列做minmax缩放
    scaler_for_y=MinMaxScaler(feature_range=(0,1))
    scaled_x_data=scaler_for_x.fit_transform(data[:,:-1])
    scaled_y_data=scaler_for_y.fit_transform(data[:,-1])

    label_train = scaled_y_data[train_begin:train_end]
    label_test = scaled_y_data[train_end:]
    normalized_train_data = scaled_x_data[train_begin:train_end]
    normalized_test_data = scaled_x_data[train_end:]

    train_x,train_y=[],[]   #训练集x和y初定义
    for i in range(len(normalized_train_data)-time_step):
        if i % batch_size==0:
            batch_index.append(i)
        x=normalized_train_data[i:i+time_step,:4]
        y=label_train[i:i+time_step,np.newaxis]
        train_x.append(x.tolist())
        train_y.append(y.tolist())
    batch_index.append((len(normalized_train_data)-time_step))

    size=(len(normalized_test_data)+time_step-1)//time_step  #有size个sample
    test_x,test_y=[],[]
    for i in range(size-1):
        x=normalized_test_data[i*time_step:(i+1)*time_step,:4]
        y=label_test[i*time_step:(i+1)*time_step]
        test_x.append(x.tolist())
```

```
            test_y.extend(y)
        test_x.append((normalized_test_data[(i+1)*time_step:,:4]).tolist())
        test_y.extend((label_test[(i+1)*time_step:]).tolist())

        return batch_index,train_x,train_y,test_x,test_y,scaler_for_y
```

定义LSTM的网络结构：

[python] view plain copy

```
#——————————————————定义神经网络变量——————————————————
def lstm(X):
    batch_size=tf.shape(X)[0]
    time_step=tf.shape(X)[1]
    w_in=weights['in']
    b_in=biases['in']
    input=tf.reshape(X,[-1,input_size])  #需要将tensor转成2维进行计算，计算后的结果作为隐藏层的输入
    input_rnn=tf.matmul(input,w_in)+b_in
    input_rnn=tf.reshape(input_rnn,[-1,time_step,rnn_unit])  #将tensor转成3维，作为lstm cell的输入
    cell=tf.contrib.rnn.BasicLSTMCell(rnn_unit)
    #cell=tf.contrib.rnn.core_rnn_cell.BasicLSTMCell(rnn_unit)
    init_state=cell.zero_state(batch_size,dtype=tf.float32)
    output_rnn,final_states=tf.nn.dynamic_rnn(cell, input_rnn,initial_state=init_state, dtype=tf.float32)  #output_rnn是记录lstm每个输出节点的结果，final_states是最后一个cell的结果
    output=tf.reshape(output_rnn,[-1,rnn_unit]) #作为输出层的输入
    w_out=weights['out']
    b_out=biases['out']
    pred=tf.matmul(output,w_out)+b_out
    return pred,final_states
```

模型训练与预测：

[python] view plain copy

```
#——————————————————训练模型——————————————————
```

```
def train_lstm(batch_size=80,time_step=15,train_begin=0,train_end=487):
    X=tf.placeholder(tf.float32, shape=[None,time_step,input_size])
    Y=tf.placeholder(tf.float32, shape=[None,time_step,output_size])
    batch_index,train_x,train_y,test_x,test_y,scaler_for_y = get_data(batch_size,time_step,train_begin,train_end)
    pred,_=lstm(X)
    #损失函数
    loss=tf.reduce_mean(tf.square(tf.reshape(pred,[-1])-tf.reshape(Y, [-1])))
    train_op=tf.train.AdamOptimizer(lr).minimize(loss)
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        #重复训练5000次
        iter_time = 5000
        for i in range(iter_time):
            for step in range(len(batch_index)-1):
                _,loss_=sess.run([train_op,loss],feed_dict={X:train_x[batch_index[step]:batch_index[step+1]],Y:train_y[batch_index[step]:batch_index[step+1]]})
            if i % 100 == 0:
                print('iter:',i,'loss:',loss_)
        ####predict####
        test_predict=[]
        for step in range(len(test_x)):
            prob=sess.run(pred,feed_dict={X:[test_x[step]]})
            predict=prob.reshape((-1))
            test_predict.extend(predict)

        test_predict = scaler_for_y.inverse_transform(test_predict)
        test_y = scaler_for_y.inverse_transform(test_y)
        rmse=np.sqrt(mean_squared_error(test_predict,test_y))
        mae = mean_absolute_error(y_pred=test_predict,y_true=test_y)
        print ('mae:',mae,'   rmse:',rmse)
```

```
31.     return test_predict
```

调用train_lstm()函数，完成模型训练与预测的过程，并统计验证误差（mae和rmse）：

[python] view plain copy

```
1.  test_predict = train_lstm(batch_size=80,time_step=15,train_begin=0,train_end
    =487)
```

迭代5000次后的结果：

[python] view plain copy

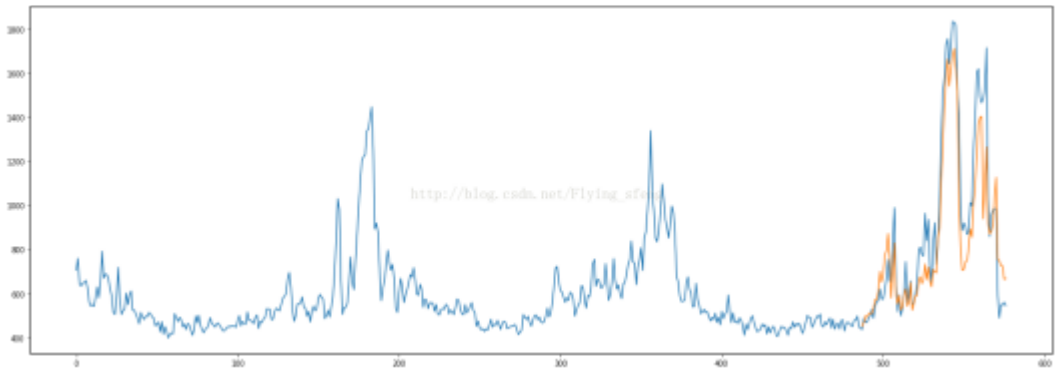
```
1.  iter: 3900 loss: 0.000505382
2.  iter: 4000 loss: 0.000502154
3.  iter: 4100 loss: 0.000503413
4.  iter: 4200 loss: 0.00140424
5.  iter: 4300 loss: 0.000500015
6.  iter: 4400 loss: 0.00050004
7.  iter: 4500 loss: 0.000498159
8.  iter: 4600 loss: 0.000500861
9.  iter: 4700 loss: 0.000519379
10. iter: 4800 loss: 0.000499999
11. iter: 4900 loss: 0.000501265
12. mae: 121.183626208    rmse: 162.049017904
```

画图分析：

[python] view plain copy

```
1.  plt.figure(figsize=(24,8))
2.  plt.plot(data[:, -1])
3.  plt.plot([None for _ in range(487)] + [x for x in test_predict])
4.  plt.show()
```

结果如下：



可以看到，lstm模型基本能预测出序列的趋势。

为了简化流程，本实验在特征工程及参数调优方面并没有下功夫，适合初学者探索lstm模型在时间序列问题上的应用。