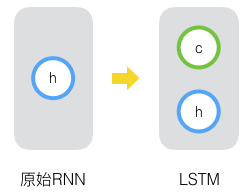
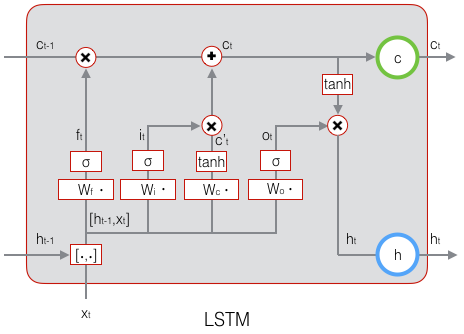
**LSTM算法原理详解以及代码说明**

* LSTM模型说明：

1. 长短时循环神经网络 LSTM 是一种为了解决较长序列训练过程中序列长期依赖的循环神经网络。在 RNN(recurrent neural networks)的基础上，LSTM 将 RNN 中的单一神经网络层更改为具有 4个神经网络层的结构。通过改变神经元的结构，LSTM 引入了门机制并通过门机制删除和增加神经元中的信息。用通俗的话来说，长短时记忆网络的思路比较简单。原始RNN的隐藏层只有一个状态，即h，它对于短期的输入非常敏感。而LSTM增加了一个状态，即c，让它来保存长期的状态，从而解决了对应的问题。新增加的状态c，称为单元状态(cell state)。
2. 在t时刻，LSTM的输入有三个：当前时刻网络的输入值、上一时刻LSTM的输出值、以及上一时刻的单元状态；LSTM的输出有两个：当前时刻LSTM输出值、和当前时刻的单元状态。LSTM的关键，就是怎样控制长期状态c。
3. 
4. 图 3‑1 从RNN到LSTM
5. LSTM用两个门来控制单元状态c的内容，一个是遗忘门(forget gate)，它决定了上一时刻的单元状态有多少保留到当前时刻;另一个是输入门(input gate)，它决定了当前时刻网络的输入有多少保存到单元状态。LSTM用输出门(output gate)来控制单元状态有多少输出到LSTM的当前输出值。
6. 
7. 图 3‑2 LSTM算法结构图
8. 其中具体的计算过程如下：
9. (1)
10. (2)

(3)

(4)

式(1)(2)分别是遗忘门和输入门的计算，(3)则是根据上一次的输出与本次输入计算用于描述当前输入的单元状态。(4)中的符号o表示按元素乘。这样，我们就把LSTM关于当前的记忆和长期的记忆组合在一起，形成了新的单元状态。

输出门(5)控制了长期记忆对当前输出的影响，而LSTM的最终输出(6)，是由输出门和单元状态共同确定的:

1. (5)
2. (6)

**时间域上历史数据步长**

在代码文件中表示为tw，亦即使用过去多长的时间步来预测未来的股价。这个超参数在LSTM网络中较为重要，当tw设置过高时会出现计算代价暴增、内存不足的情况，这也是RNN家族算法最令人诟病的地方——无法提取过去很长时间的时间特征。因为随着预测时间步长的增加，RNN的效果往往会很差。为了解决RNN中的梯度消失和长时依赖的问题，提出了LSTM（长短时记忆单元），通过cell门开关实现时间上的记忆功能，防止梯度消失。LSTM 网络是一种特殊的RNN网络，在RNN网络的结构基础上进行了优化，使得其在序列数据较长时依然能够有效地建模。LSTM 与 RNN 的区别在于其实现了一个更加细化的内部处理单元,来实现上下文信息的有效存储和更新。

RNN中的梯度消失不是指损失对参数的总梯度消失了，而是RNN中对较远时间步的梯度消失了。RNN中反向传播使用的是BPTT方法，损失loss对参数W的梯度等于loss在各时间步对w求导之和。LSTM设计的初衷就是让当前记忆单元对上一记忆单元的偏导为常数，后来为了避免记忆细胞无线增长，引入了“遗忘门”。通常会给遗忘门设置一个很大的偏置项，使得遗忘门在多数情况下是关闭的，只有在少数情况下开启，缓解了由于小数连乘导致的梯度消失问题。但同时，LSTM的参数量是RNN的4倍，参数量过多就会存在过拟合的风险。因此tw设置的另一个目的也是为了避免过拟合情况的出现。

* LSTM模型代码说明：
* # -\*- coding: utf-8 -\*-  
  *"""  
   Tensorflow-gpu 1.12.0  
   cuda 9.0  
   cudnn 7.6.5  
   keras 2.1.6  
   h5py 2.9.0  
  """*from \_\_future\_\_ import print\_function  
  import cmath  
  import math  
    
  from keras.layers import (  
   Input,  
   Activation,  
  )  
  from keras.regularizers import l2  
  import keras  
  from keras.models import Model, Sequential, Input  
  from keras.layers import LSTM, Dropout, Dense, Activation, Bidirectional  
  import os  
  import pickle  
  import sys  
  import time  
  import pandas as pd  
  import h5py  
  import matplotlib.pyplot as plt  
  import numpy as np  
  from keras.callbacks import EarlyStopping, ModelCheckpoint  
  from keras.optimizers import Adam  
  from sklearn.preprocessing import MinMaxScaler  
  from sklearn.metrics import mean\_squared\_error  
  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score  
  np.random.seed(1337) # for reproducibility  
  # parameters  
  nb\_epoch = 100 # number of epoch at training stage. To find a nice epochs in the valid dataset.  
  batch\_size = 64 # batch size  
  lr = 0.001 # learning rate lr = 0.0002  
  # divide data into two subsets: Train & Test, of which the test set is the last "days\_test" days  
  days\_test = 3  
  len\_test = 140  
  tw = 1  
    
    
  class MinMaxNormalization(object):  
   *'''MinMax Normalization --> [-1, 1]  
   x = (x - min) / (max - min).  
   x = x \* 2 - 1  
   '''* def \_\_init\_\_(self):  
   pass  
    
   def fit(self, X):  
   self.\_min = X.min()  
   self.\_max = X.max()  
   print("min:", self.\_min, "max:", self.\_max)  
    
   def transform(self, X):  
   X = 1. \* (X - self.\_min) / (self.\_max - self.\_min)  
   X = X \* 2. - 1.  
   return X  
    
   def fit\_transform(self, X):  
   self.fit(X)  
   return self.transform(X)  
    
   def inverse\_transform(self, X):  
   X = (X + 1.) / 2.  
   X = 1. \* X \* (self.\_max - self.\_min) + self.\_min  
   return X  
  def creat\_dataset(dataset, tw):  
   data\_x = []  
   data\_y = []  
   for i in range(len(dataset)-tw):  
   data\_x.append(dataset[i:i+tw])  
   data\_y.append(dataset[i+tw])  
   return np.asarray(data\_x), np.asarray(data\_y) # 转为ndarray数据  
    
  def main():  
   # load data  
   print("loading data...")  
   # x\_data\_all = []  
   # y\_data\_all = []  
   dataframe = pd.read\_csv('zgpa\_train.csv',  
   header=0, parse\_dates=[0],  
   index\_col=0, usecols=[0, 5], squeeze=True)  
   print(dataframe)  
   # dataset = dataframe.values  
   # data = pd.read\_csv('zgpa\_train.csv', header=None, usecols=[5], dtype='int')  
   dataset = dataframe.values  
   print('dataset:', dataset)  
   scaler = MinMaxScaler(feature\_range=(0, 1))  
   dataset = scaler.fit\_transform(dataset.reshape(-1, 1))  
   print(dataset.shape)  
   len\_all = dataset.shape[0]  
   x\_data\_train = dataset[:-len\_test]  
   x\_data\_test = dataset[-len\_test:]  
   x\_train, y\_train = creat\_dataset(x\_data\_train, tw)  
   x\_test, y\_test = creat\_dataset(x\_data\_test, tw)  
    
    
   model = Sequential()  
   model.add(LSTM(input\_dim=1, output\_dim=50, return\_sequences=True))  
   # model.add(Dropout(0.2))  
    
   model.add(LSTM(input\_dim=50, output\_dim=100, return\_sequences=True))  
   # model.add(Dropout(0.2))  
    
   model.add(LSTM(input\_dim=100, output\_dim=200, return\_sequences=True))  
   # model.add(Dropout(0.2))  
    
   model.add(LSTM(300, return\_sequences=False))  
   model.add(Dropout(0.2))  
    
   model.add(Dense(100))  
   model.add(Dense(output\_dim=1))  
    
   model.add(Activation('relu'))  
   start = time.time()  
   adam = Adam(lr=lr)  
   model.compile(loss='mse', optimizer=adam)  
   model.summary()  
   print("training model...")  
   history = model.fit(x\_train, y\_train,  
   nb\_epoch=nb\_epoch,  
   batch\_size=batch\_size,  
   validation\_split=0.1,  
   verbose=2)  
   score = model.evaluate(  
   x\_test, y\_test, batch\_size=64, verbose=0)  
   model.save('lstm.h5')  
   # model.save()  
   predict = model.predict(x\_test, batch\_size=64)  
   predict = scaler.inverse\_transform(predict)  
   y\_test = scaler.inverse\_transform(y\_test)  
   rmse = math.sqrt(mean\_squared\_error(y\_test, predict))  
   print('specific rmse = ', rmse)  
   plt.figure(figsize=(16, 8))  
   plt.plot(y\_test, 'b', label='real')  
   plt.plot(predict, ls='-.', c='r', label='predict')  
   plt.legend(loc='best')  
   plt.grid(True)  
   plt.savefig('LSTM.png', dpi=800)  
   plt.show()  
  if \_\_name\_\_ == '\_\_main\_\_':  
   main()