工 作 总 结

冯子朋　2016 年 12 月 23 日

今天，我来公司已经整整一个月。这一个月我主要进行基于 Tensorflow 框架的 LSTM 神经网络股指预测的工作，并利用预测结果编写了一个简单的交易策略。下面按照时间顺序，对所做的工作进行总结。

一、学习基础知识，提出模型思路（第一周）

1.基础知识的学习包括：python 数据分析工具、股指期货相关知识、LSTM 网络相关知识。

2.RNN 网络和 LSTM 网络：

RNN:全连接的深度神经网络存在着一个问题——无法对时间序列上的变化进行建模。然而,样本出现的时间顺序对于自然语言处理、语音识别、时间序列分析等应用非常重要。为了适应这种需求,就出现了另一种神经网络结构——循环神经网络 RNN。在 RNN 中,神经元的输出可以在下一个时间戳直接作用到自身,即第 i 层神经元在 m 时刻的输入,除了(i－1)层神经元在该时刻的输出外,还包括其自身在(m－1)时刻的输出。结构如下图所示：

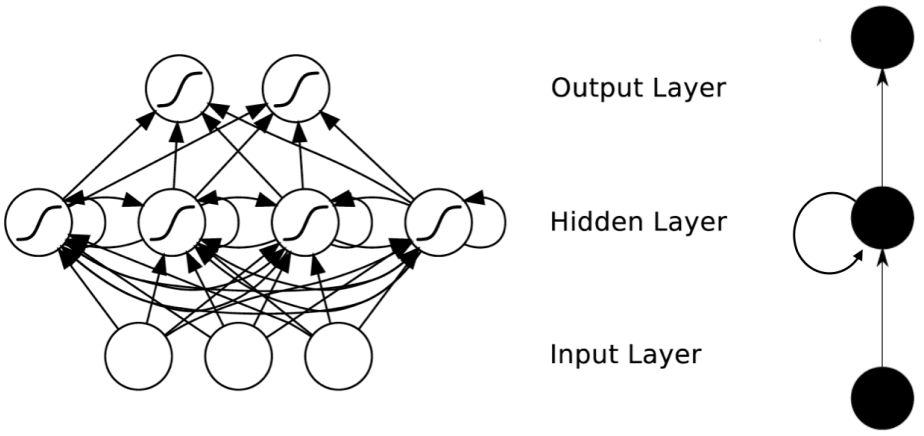


图 1.1 RNN 神经网络结构图

LSTM:而 RNN 仍存在一个显著问题，因为神经元数目和结构受限，只能利用前面几个时间点的数据对当前时间点的数据进行预测，而较长时间之前的数据难以被应用进来。但在股指预测中，因为“历史会不断重演”的特性，长久之前的数据也可能对当前时间的预测有较大的影响，因此 RNN 的这个问题更为显著，需要对 RNN 的结构进行改进，也就是我们所用的 LSTM 网络。

LSTM 通过修改 RNN 神经元的结构，添加控制输入的三个“门”，有了明显的“选择性记忆”特性。能够记忆对当前影响较大的数据，而遗忘对当前影响较小的数据，很好的克服了 RNN 记忆受限的问题。因此理论上非常适合用 LSTM 来进行预测。

模型思路：选取适当的大量历史股指期货数据，加入必要的特征，并分出训练集和测试集。构建 LSTM 神经网络对训练集进行训练，使测试集能够根据前几分钟的特征预测出后面几分钟的某些特征，然后再根据预测目标编写交易策略。

重点：如何对数据进行处理；如何选取训练集和测试集；LSTM 的结构；预测目标

二、学习 TensorFlow, 构建模型一——LSTM 回归模型（第二周）

（一）TensorFlow 相关知识

它的工作模式如下：

· 使用图 (graphs) 来表示计算.
· 在会话 (Session) 中执行图.
· 使用张量 (tensors) 来代表数据,给予固定的格式。
· 通过变量 (Variables) 维护状态,进行赋值、初始化等操作。
· 使用供给 (feeds) 和取回 (fetches) 将数据传入或传出任何操作
· 提供了一系列的计算损失率和最优化的方法,易于调用

对应的典型语句有：

1.xtr = tf.placeholder("float", [None, n_steps, n_input])
ytr = tf.placeholder("float", [None,n_classes])
占位符，表示输入输出数组的格式。其中 None 表示任意大小。

2.weights = { 'out': tf.Variable(tf.random_normal([n_hidden, n_classes]))}
biases = {'out': tf.Variable(tf.random_normal([n_classes]))}
初始化权值和偏移量，采用随机数赋值。

3.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, ytr))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
代价函数和最优化方法，其中利用交叉熵表示实际输出与预测输出之间的差异，然后使用 adam 最优化方法使得代价函数最小。（另外一个常用的最优化方法是梯度下降法）

4.correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(ytr, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
评估模型的准确率

5.sess = tf.InteractiveSession()
acc = sess.run(accuracy, feed_dict={xtr: batch_x, ytr: batch_y})
loss = sess.run(cost, feed_dict={xtr: batch_x, ytr: batch_y})
创建交互式会话并运行，在运行过程中不断输入分段数据，计算准确率和损失度。

（二） LSTM 回归模型

1.选择合约：

考虑到运算时间，选取了 18 个月的数据进行分析。提取每天的 IF 主力合约，拼接为最终的主力合约。这些合约分别为 IF1501-IF1606。

2.提取特征：

选取每分钟的收盘价相对于当日开盘价的涨跌幅，每分钟最大值相对于当日开盘价的涨跌幅， 每分钟最小值相对于当日开盘价的涨跌幅，这三个特征，保存到数组中。

3.训练集和测试集的选取：

前 80%作为训练集，后 20%作为测试集。先考虑预测单分钟的数据，再慢慢增加至多分钟。预测指标为当前时刻后面第三分钟的涨跌幅度。将预测数据与实际数据作比较并作图，观察预测结果。

4.神经网络模型构建：

Tensorflow 中有两个常用集成模块：TensorFlowRNNClassfier,用来构建 RNN 分类器；TensorFlowRNNRegressor，用来构建 RNN 回归模型。这两个模块可以方便开发者搭建机器学习模型，极大提高开发效率。最初我认为要预测未来的价格涨跌幅，应该用回归模型更合适，所以选用 TensorFlowRNNRegressor 库。

```python
# 用前25分中的数据对后面第3分钟的数据进行预测
def rnn_data(data, date, timeStep=25, output=3, get_label=False):
    rnn_df = []
    rnn_time = []
    for t in range(len(data)):
        data_tmp = data[t]
        if not get_label:
            for i in range(len(data_tmp) - timeStep - output + 1):
                rnn_df.append(data_tmp[i:i + timeStep])
                rnn_time.append(date[t])
        else:
            for i in range(timeStep, len(data_tmp) - output + 1):
                rnn_df.append(data_tmp[i + output - 1:i + output])
                rnn_time.append(date[t])
    return np.array(rnn_df), rnn_time
```

```python
# 设置rnn回归模型参数
regressor = learn.TensorFlowRNNRegressor(rnn_size=TIME_STEP,
                                         cell_type='lstm',
                                         steps=TRAINING_STEPS,
                                         num_layers=1,
                                         learning_rate=0.003,
                                         batch_size=BATCH_SIZE,
                                         verbose=2, input_op_fn=input_fn)
```

图 2.1 预测代码和网络搭建代码

5.测试结果：

选取 1000 个预测点画图，可见两者吻合程度很高。除了个别数据点，误差均在 0.005 以内。如下图所示：

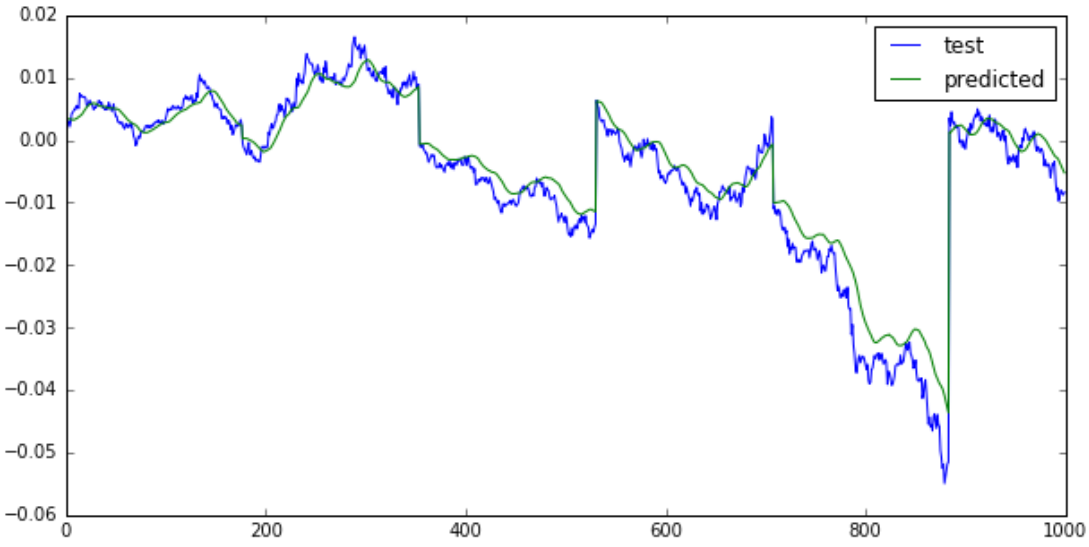


图 2.2 LSTM 回归模型涨跌幅度预测图

6.模型分析：

（1）本模型根据前 n 分钟的数据预测后续某一分钟的数据，能够取得较好的预测效果。但预测连续 n 分钟的数据效果并不好。

（2）该模型只预测单分钟的涨跌幅度，涨跌幅度是一个不好预测的量。并且回归模型的预测以定量指标为主，不好应用到策略中。考虑改为 LSTM 分类模型，定性预测后面时间段内的涨跌趋势等，方面写策略。

（3）机器学习中，训练数据的选择和特征提取是非常重要的，甚至超过模型本身。因此在构建神经网络之前需要首先进行数据处理和特征选择。

三、重点进行数据处理, 构建模型二——LSTM 分类模型（第三周）

1.数据处理

对于给定的数据集，如果其中存在不合理的畸形数据，在训练时会产生非常严重的干扰，极大影响学习的性能和预测的准确率。因此，**首先要做的就是对数据集进行筛选和清理**。步骤如下：

（1）提取主力合约：

考虑到不同年份之间的主力合约数据可能会有较大差距，仅提取 2016 年的主力合约数据进行训练和测试。提取每天的 IF 主力合约，拼接为最终的主力合约。

（2）数据筛选与清理：

1).删除非正常交易时间的数据，交易时间为上午 9：30-11：30，下午 1:00 到 3:00

2). 因为每天开盘的前 10 分钟和收盘的最后 10 分钟数据波动较大，暂时在训练时去掉这些数据

3). 一分钟之内，最大股价和最低股价相差超过 50 点的，那么应该尽量删掉这些数据

4). 删除一分钟成交量大于 1000 的，这些数据可能会影响整体数据

```
for k in range(len(data_temp)):
    if data_temp['Time'].iloc[i].hour < 9:
        delete_list.append(k)
    elif data_temp['Time'].iloc[k].hour == 9 and data_temp['Time'].iloc[k].minute <= 29:
        delete_list.append(k)
    elif data_temp['Time'].iloc[k].hour >= 15:
        delete_list.append(k)
    elif data_temp['MaxPrice'].iloc[k] - data_temp['MinPrice'].iloc[k] > 50:
        delete_list.append(k)
    elif data_temp['Volume'].iloc[k] < 100:
        delete_list.append(k)
for j in range(len(data_temp)):
    if j in delete_list:
        data_temp.drop(j,inplace=True)
data_temp.index = range(len(data_temp))
data_temp.to_csv(new_path + file_list[i].split('.')[0]+ '_cleaned' + '.csv',sep=',')
```

图 3.1 数据清理代码

（3）特征选择：

每个分钟数据都有诸多特征，其中的一些特征还被经常用作交易策略的编写中。有必要把这些特征统统加进去。本模型加入的特征有：

1) 主力合约中原有的特征：

“Latestprice”，“First_Latestprice”，“MaxPrice”，“MinPrice”，“Last_Buy1price”，“Last_Sell1price”，“Last_Buy1quantity”，“Last_Sell1quantity”，“Stockup”，“Volume”，共 10

个。

2) 平均值指标：

平均价：MeanPrice = (Latestprice + Minprice + Maxprice + First.Latestprice)/4。

该指标主要用于打预测标签，是一个既容易训练又易于写策略的指标。

3) MA、EMA、MACD、布林带等指标：

分别加入收盘价和成交量的 5 分钟、12 分钟、26 分钟的 MA（移动平均线）、EMA（指数移动平均线）指标，以及几条移动平均线之间的距离指标。共（3+1）*2*2 = 16 个指标。

计算 EMA_12 与 EMA_26 的离差 DIFF 指标，离差平均值 DEA 指标，进而得到 MACD（平滑异同平均线）指标。共 3 个指标。

添加完布林带后，数据集共加入了上轨值、下轨值、%b 指标、通道宽度等 4 个指标。总特征数为 34 个。部分代码截图如下：

```
#  添加收盘价的移动平均线MA和指数平滑移动平均线EMA
    for ma in ma_list:
        newdata1['MA_' + str(ma)] = pd.rolling_mean(newdata1['Latestprice'], ma)
    # 计算移动平均线之间的距离
    newdata1['Dis_MA5_26'] = newdata1['MA_5'] - newdata1['MA_26']
    # 计算指数平滑移动平均线
    for ma in ma_list:
        newdata1['EMA_' + str(ma)] = pd.ewma(newdata1['Latestprice'], span=ma)
    # 计算指数平滑移动平均线之间的距离
    newdata1['Dis_EMA5_26'] = newdata1['EMA_5'] - newdata1['EMA_26']

# 添加成交量的移动平均线MA和指数平滑移动平均线EMA
    for ma in ma_list:
        newdata1['Vol_MA_'+str(ma)] =  pd.rolling_mean(newdata1['Volume'], ma)
# 计算成交量移动平均线之间的距离
    newdata1['Dis_Vol_MA5_26'] = newdata1['Vol_MA_5'] - newdata1['Vol_MA_26']
# 添加成交量的指数平滑移动平均线
    for ma in ma_list:
        newdata1['Vol_EMA_'+str(ma)] = pd.ewma(newdata1['Volume'], span=ma)
# 计算指数平滑移动平均线之间的距离
    newdata1['Dis_Vol_EMA5_26'] = newdata1['Vol_EMA_5'] - newdata1['Vol_EMA_26']
    newdata1['DIFF_12_26'] = newdata1['EMA_12'] - newdata1['EMA_26']
# 计算离差平均值DEA，也就是计算离差值的指数平滑移动平均，设置为5分钟的指数平滑曲线
    newdata1['DEA_12_26'] =  pd.ewma(newdata1['DIFF_12_26'],span = 9)
# 计算MACD值
    newdata1['MACD'] = 2*(newdata1['DIFF_12_26'] - newdata1['DEA_12_26'])
```

图 3.2 诸多特征添加代码

（4）数据归一化

归一化是使不同的量纲之间具有可比性,比如属性 A 的值的范围为 2000 左右,属性 B 的值范围 100 左右,为了使其两者之间具有可比性,分别对属性 A 和属性 B 进行归一化处理,本实验采用线性归一化方式处理数据,其结果范围在 0-1 之间,公式如下:

$$x_t^i := \frac{x_t^i - \min x^i}{\max x^i - \min x^i}$$

（5）组时间序列

我们要研究的问题是时间序列问题，如果只对单一的分钟数据作分析，会大大减弱数据的时间相关性，因此有必要进行时间序列的组合。将每五分钟的归一化数据（已加入各个特征）组合到一起，将原来的二维数组转变为三维数组进行分析。

```
#   数据归一化
    data_array_per = np.array(data_new)
    min_max_scaler = preprocessing.MinMaxScaler()
    data_array_per = min_max_scaler.fit_transform(data_array_per)
#  组时间序列
    for k in range(len(data_array_per)):
        if k < len(data_array_per)-5  and k >= 25:
            seq_new_25.append(data_array_per[k - 25:k])
seq_new_25 = np.array(seq_new_25)
print seq_new_25.shape
```

图 3.3 数据归一化和时间序列代码

得到最终数据集如下所示：

```
array([[[ 1.        ,  1.        ,  1.        , ...,  0.65315817,
          1.        ,  0.51094944],
        [ 0.98349537,  0.9947952 ,  0.98259529, ...,  0.64915289,
          0.98885873,  0.50056431],
        [ 0.97558218,  0.97827563,  0.96959683, ...,  0.64581164,
          0.97517249,  0.54248697],
        [ 0.97354737,  0.97080788,  0.97334215, ...,  0.64486167,
          0.97189232,  0.54957615],
        [ 0.97467782,  0.96922381,  0.97003745, ...,  0.64644872,
          0.97036534,  0.56973569]],

       [[ 0.98349537,  0.9947952 ,  0.98259529, ...,  0.64915289,
          0.98885873,  0.50056431],
        [ 0.97558218,  0.97827563,  0.96959683, ...,  0.64581164,
          0.97517249,  0.54248697],
        [ 0.97354737,  0.97080788,  0.97334215, ...,  0.64486167,
          0.97189232,  0.54957615],
        [ 0.97467782,  0.96922381,  0.97003745, ...,  0.64644872,
          0.97036534,  0.56973569],
        [ 0.97806918,  0.97239194,  0.97554527, ...,  0.65061808,
          0.97381518,  0.54660311]],
```

是一个大小为(35753, 5, 34)的三维数组。

2.训练集与测试集的划分

（1）提取训练数据与测试数据

取前面 10000 个数据作为训练集。前面的数据点因有 na 值填充，会对准确性造成一定影响，因此从第 1000 个点开始，到第 11000 个点结束。

将之后的 2000 个数据作为测试集，因为 11000-12000 这一段的价格起伏过于平缓，不宜测试模型准确性，因此选择 12000-14000 部分的 2000 个数据作为测试集。

（2）打标签

根据下一分钟的涨跌幅情况，将数据分为五类，即：大涨、小涨、平稳、小跌、大跌。阈值设定根据大量数据的涨跌幅统计得出，代码如下：

```
data1 = data[1000:11000]
train_label = []
for i in data1['RaiseDown']:
    if i > 0.002:
        train_label.append([1,0,0,0,0])    #大涨
    elif i > 0.0005:
        train_label.append([0,1,0,0,0])    #小涨
    elif i> -0.0005:
        train_label.append([0,0,1,0,0])    #平稳
    elif i > -0.002:
        train_label.append([0,0,0,1,0])    #小跌
    else:
        train_label.append([0,0,0,0,1])    #大跌
train_label = np.array(train_label)
train_label.shape
```

图 3.4 趋势分类代码

根据本模型设计的阈值，训练数据和测试数据的五类涨跌幅情况所占的数据数量如下：

大涨： 244	大涨： 17
小涨： 2136	小涨： 320
平稳： 5124	平稳： 1304
小跌： 2313	小跌： 349
大跌： 183	大跌： 10

（a）训练数据　　（b）测试数据

大涨和大跌数量较少，平稳数量最多，较符合实际。

3.LSTM 分类模型

如果像模型一一样，直接使用 TensorFlowRNNClassfier 模块构建分类器，很多算法都是固定的，不太好修改优化，因此选择使用各功能自己定义的代码，进行神经网络的搭建。经过不断的调参，调出效果最好的神经网络结构。部分代码截图如下：

```
# 定义代价函数和最优化方法
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, ytr))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# 评估模型准确率
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(ytr, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

```
    # 一直迭代，直到最大步长
while step * batch_size < training_iters:
    batch_x, batch_y = next_batch_tr(batch_size)
    batch_x = batch_x.reshape((-1, n_steps, n_input))
        # 利用最优化方法调整权值
    sess.run(optimizer, feed_dict={xtr: batch_x, ytr: batch_y})
    if step % display_step == 0:
            # 计算准确率和损失函数
        acc = sess.run(accuracy, feed_dict={xtr: batch_x, ytr: batch_y})
        loss = sess.run(cost, feed_dict={xtr: batch_x, ytr: batch_y})
        # 每隔一定步长显示训练准确率和损失率
        print("Iter " + str(step * batch_size) + ", Minibatch Loss= " +
                "{:.6f}".format(loss) + ", Training Accuracy= " +
                "{:.5f}".format(acc))
    step += 1
print("Optimization Finished!")
```

图 3.5 神经网络搭建代码

4.预测结果

分别从以下几方面进行预测：

（1）当前时刻后 1-5 分钟的涨跌情况；

（2）上涨趋势（或者下跌趋势）持续的时间

（3）移动平均线、布林带等指标

部分预测的结果如下图：

（1）当前时刻后 1 分钟的涨跌情况的准确率，73.5%

```
Iter 84000, Minibatch Loss= 0.815975, Training Accuracy= 0.70000
Iter 86000, Minibatch Loss= 0.404420, Training Accuracy= 0.85000
Iter 88000, Minibatch Loss= 0.868310, Training Accuracy= 0.60000
Iter 90000, Minibatch Loss= 0.850256, Training Accuracy= 0.70000
Iter 92000, Minibatch Loss= 0.866675, Training Accuracy= 0.60000
Iter 94000, Minibatch Loss= 1.250295, Training Accuracy= 0.45000
Iter 96000, Minibatch Loss= 0.866472, Training Accuracy= 0.60000
Iter 98000, Minibatch Loss= 1.051986, Training Accuracy= 0.50000
Optimization Finished!
('Testing Accuracy:', 0.73500001)
```

（2）当前时刻后 2 分钟的涨跌情况的准确率，59.8%

```
Iter 82000, Minibatch Loss= 0.923656, Training Accuracy= 0.70000
Iter 84000, Minibatch Loss= 0.960349, Training Accuracy= 0.50000
Iter 86000, Minibatch Loss= 0.918003, Training Accuracy= 0.55000
Iter 88000, Minibatch Loss= 0.940913, Training Accuracy= 0.65000
Iter 90000, Minibatch Loss= 0.984426, Training Accuracy= 0.45000
Iter 92000, Minibatch Loss= 0.976268, Training Accuracy= 0.40000
Iter 94000, Minibatch Loss= 0.736631, Training Accuracy= 0.60000
Iter 96000, Minibatch Loss= 0.995859, Training Accuracy= 0.60000
Iter 98000, Minibatch Loss= 0.780405, Training Accuracy= 0.85000
Optimization Finished!
('Testing Accuracy:', 0.59850001)
```

（3）大涨趋势持续时间预测结果：

```
预测大涨持续时间:
第 60 分钟为大涨趋势，直到 6 分钟后开始下跌
第 159 分钟为大涨趋势，直到 2 分钟后开始下跌
第 165 分钟为大涨趋势，直到 3 分钟后开始下跌
第 166 分钟为大涨趋势，直到 2 分钟后开始下跌
第 184 分钟为大涨趋势，直到 3 分钟后开始下跌
第 235 分钟为大涨趋势，直到 17 分钟后开始下跌
第 247 分钟为大涨趋势，直到 5 分钟后开始下跌
第 248 分钟为大涨趋势，直到 4 分钟后开始下跌
第 249 分钟为大涨趋势，直到 3 分钟后开始下跌
第 250 分钟为大涨趋势，直到 2 分钟后开始下跌
第 256 分钟为大涨趋势，直到 18 分钟后开始下跌
第 385 分钟为大涨趋势，直到 8 分钟后开始下跌
第 386 分钟为大涨趋势，直到 7 分钟后开始下跌
第 390 分钟为大涨趋势，直到 3 分钟后开始下跌
第 427 分钟为大涨趋势，直到 3 分钟后开始下跌
第 428 分钟为大涨趋势，直到 2 分钟后开始下跌
第 536 分钟为大涨趋势，直到 3 分钟后开始下跌
第 810 分钟为大涨趋势，直到 3 分钟后开始下跌
```

大涨预测准确率：

```
accuary: 0.318181818182
```

（4）下一分钟布林带下轨线预测准确率，47.9%

```
Iter 82000, Minibatch Loss= 0.560647, Training Accuracy= 0.80000
Iter 84000, Minibatch Loss= 0.585663, Training Accuracy= 0.75000
Iter 86000, Minibatch Loss= 0.558475, Training Accuracy= 0.65000
Iter 88000, Minibatch Loss= 0.124142, Training Accuracy= 1.00000
Iter 90000, Minibatch Loss= 0.554778, Training Accuracy= 0.60000
Iter 92000, Minibatch Loss= 0.319227, Training Accuracy= 0.90000
Iter 94000, Minibatch Loss= 0.523589, Training Accuracy= 0.65000
Iter 96000, Minibatch Loss= 0.652594, Training Accuracy= 0.70000
Iter 98000, Minibatch Loss= 0.130993, Training Accuracy= 0.95000
Optimization Finished!
('Testing Accuracy:', 0.479)
```

（5）某段数据的下一分钟涨跌预测分类与实际分类的对比

```
Test 0 Prediction: 2 True Class: 2
Test 1 Prediction: 2 True Class: 2
Test 2 Prediction: 2 True Class: 2
Test 3 Prediction: 2 True Class: 2
Test 4 Prediction: 2 True Class: 2
Test 5 Prediction: 2 True Class: 2
Test 6 Prediction: 2 True Class: 2
Test 7 Prediction: 2 True Class: 2
Test 8 Prediction: 2 True Class: 1
Test 9 Prediction: 2 True Class: 2
Test 10 Prediction: 2 True Class: 2
Test 11 Prediction: 2 True Class: 2
Test 12 Prediction: 2 True Class: 1
Test 13 Prediction: 3 True Class: 2
Test 14 Prediction: 3 True Class: 2
Test 15 Prediction: 3 True Class: 1
Test 16 Prediction: 3 True Class: 3
Test 17 Prediction: 1 True Class: 2
Test 18 Prediction: 1 True Class: 1
Test 19 Prediction: 2 True Class: 2
Test 20 Prediction: 1 True Class: 1
```

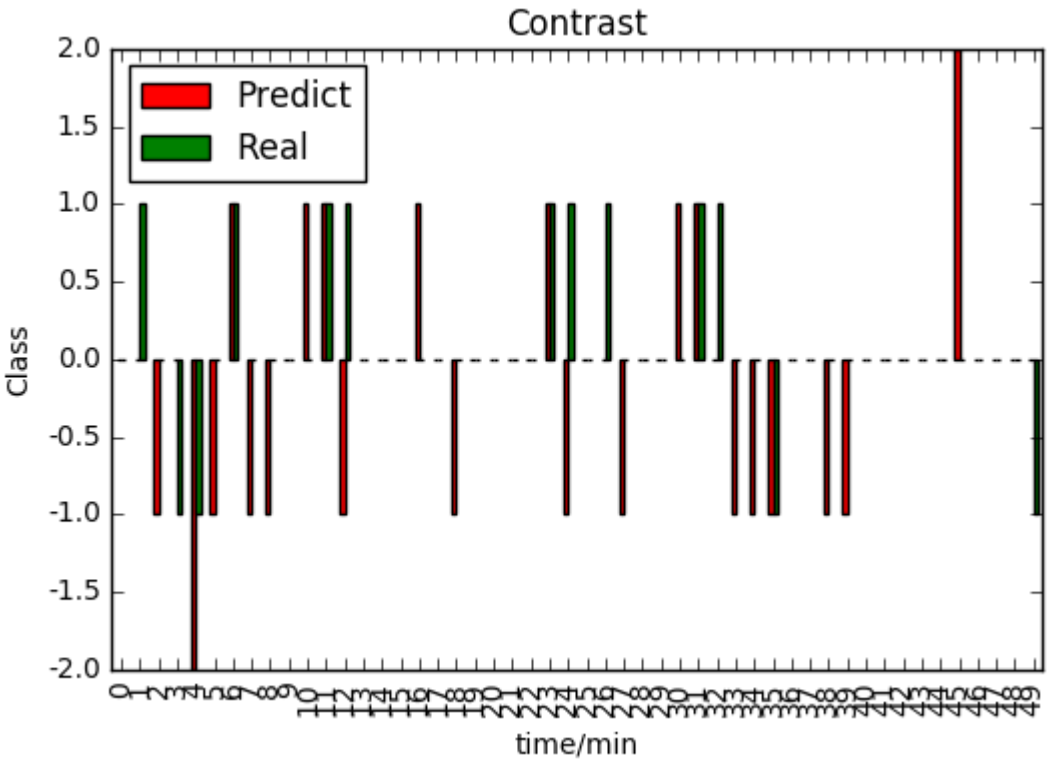


图 3.6 实际分类与预测分类的对比

5.模型的保存与加载

模型训练和测试时，每次都需要重新训练。但如果应用到策略中，实时更新的数据与训练需要的漫长时间有矛盾。因此必须进行模型的保存与加载。部分代码如下：

```
saver = tf.train.Saver()
saver.restore(sess, "/home/chocolate/Model_LSTM-Future_20161123/models/predict_1min.ckpt")
print "Model restored."
test_data1 = test_data.reshape((-1, n_steps, n_input))
# test_label1 = test_label[:test_len]
print("Testing Accuracy:",
        sess.run(accuracy, feed_dict={xtr: test_data1, ytr: test_label}))
```

图 3.7 模型加载代码

三、第一个交易策略的编写（第四周）

1.策略内容：

完成对以上指标的预测，就开始着手写交易策略了。目前第一个策略，只用到了当前时刻后 1-5 分钟的涨跌分类。该策略主要包含以下部分：

（1）获取当前的 bar 数据，并添加上述的 34 个特征。归一化，生成时间序列，使格式与训练时用的数据集格式完全相同。

（2）加载 LSTM 网络模型，获得后 1-5 分钟的分类结果。分类中，‘5’表示大涨，‘4’表示小涨，‘3’表示平稳，‘2’表示小跌，‘1’表示大跌。

代码如下：

```
# 将当前数据代入模型，获得1-5分钟的分类类别
# 5: 大涨    4: 小涨    3: 平稳    2: 小跌    1: 大跌
        pred_1 = sess_1.run(pred, feed_dict={xtr: seq_data})
        predict_1 = 5 - pred_1.argmax()
        pred_2 = sess_2.run(pred, feed_dict={xtr: seq_data})
        predict_2 = 5 - pred_2.argmax()
        pred_3 = sess_3.run(pred, feed_dict={xtr: seq_data})
        predict_3 = 5 - pred_3.argmax()
        pred_4 = sess_4.run(pred, feed_dict={xtr: seq_data})
        predict_4 = 5 - pred_4.argmax()
        pred_5 = sess_5.run(pred, feed_dict={xtr: seq_data})
        predict_5 = 5 - pred_5.argmax()

        return predict_1, predict_2, predict_3, predict_4, predict_5
```

（3）对 1-5 分钟的预测值赋予不同权重，时间越短权值越高，得到一个综合得分。

```
# 计算加权得分
self.score = 0.4 * self.predict_1 + 0.2 * self.predict_2 + \
    0.2 * self.predict_3 + 0.1 * self.predict_4 + \
    0.1 * self.predict_5
```

（4）根据得分判断是否作出买卖请求。当得分大于一个设定的最高阈值时，表示后 1-5 分钟基本属于大涨趋势，可以买多；当得分小于一个设定的最低阈值时，表示后 1-5 分钟基本属于大跌趋势，可以买空。

```
if self.score >= self.buy_raise_score and self.zhisun_label is False:
    self.buy(self.askprice1, 1)
    self.rec_price = self.askprice1  # 记录当前的价格作为比较价格
    self.pos_rec += 1
    self.writeCtaLog(u'buy!' + str(self.rec_price) + u'pos_rec' + str(self.pos_rec))
    self.trade_records.append([bar.datetime, self.price, u'buy'])
    self.init_price = self.askprice1

if self.score <= self.buy_down_score and self.zhisun_label is False:
    self.short(self.bidprice1, 1)
    self.rec_price = self.bidprice1
    self.init_price = self.bidprice1
    self.pos_rec -= 1
    self.writeCtaLog(u'short!' + str(self.rec_price) + u'pos_rec' + str(self.pos_rec))
    self.trade_records.append([bar.datetime, self.price, u'short'])
```

（5）止盈止损策略。同样的，设置一个止盈系数和一个止损系数。手中持有仓时，当前买一价（卖一价）已经超过了止盈（止损）所限定的最大范围，则平仓。

```
def long_pos_sell(self, long_pos):##多仓的平仓考虑。

    if long_pos.bidPrice1 < (1 - self.zhisun)*self.rec_price:#止损策略
        self.sell(self.bidprice1, 1)
        self.pos_rec -= 1
        self.writeCtaLog(u'多仓'+u'止损价'+str((1 - self.zhisun) * self.rec_price)+u'当前价'+str
        self.zhisun_label = True
        self.zhisun_bar = 0
        self.trade_records.append([long_pos.datetime,self.price,u'sell'])

    if long_pos.bidPrice1 > self.rec_price*(1+self.zhiying):##止盈策略
        self.sell(self.bidprice1, 1)
        self.pos_rec -= 1

        self.writeCtaLog(u'多仓'+u'止盈价'+str(self.rec_price*(1+self.zhiying))+u'当前价'+str(lo
        self.trade_records.append([long_pos.datetime,self.price,u'sell'])

def short_pos_cover(self, short_pos):##持有空仓时的平仓考虑。

    if short_pos.askPrice1 > (1+self.zhisun)*self.rec_price: ##做空的时候，实时价格高于止损线
        self.cover(self.askprice1, 1)
        self.pos_rec += 1
        self.writeCtaLog(u'空仓'+u'止损价'+str((1 + self.zhisun) * self.rec_price)+u'当前价'+str

        self.zhisun_label = True
        self.zhisun_bar = 0
        self.trade_records.append([short_pos.datetime,self.price,u'cover'])

    if short_pos.askPrice1 < self.rec_price*(1-self.zhiying):#做空的时候，实时价格已经低于止盈线了
        self.cover(self.askprice1, 1)
        self.pos_rec += 1
        self.writeCtaLog(u'空仓'+u'止盈价'+str(self.rec_price * (1 - self.zhiying))+u'当前价'+st

        self.trade_records.append([short_pos.datetime,self.price,u'cover'])
```

（6）优化策略：防止开多仓的策略；以及亏损后考虑停止 5 分钟再次建仓，防止立刻反向操作再次亏损的情况；不留过夜仓的策略。

```
def pos_rec_concert(self):  # pos与pos_rec不一致时进行调整。
    if self.pos != self.pos_rec:
        self.writeCtaLog(u'调整pos_rec，由' +
                         str(self.pos_rec) + u'变为' + str(self.pos))
        self.pos_rec = self.pos

    if self.lastOrder is not None and self.lastOrder.status == u'未成交':
        self.cancelOrder(self.lastOrder.vtOrderID)
        self.lastOrder = None
        self.writeCtaLog(u'撤销上一单')
```

```
#每次止损平仓后，考虑停止5分钟进行操作，因此设置zhisun_label作为是否能重新建仓的标志
    def zhisun_set(self):
        if self.zhisun_label == True:
            self.zhisun_bar += 1

        if self.zhisun_bar >= 5:
            self.zhisun_bar = 0
            self.zhisun_label = False
```

```
if bar.datetime.hour == 14 and bar.datetime.minute >= 58:
    print(u'stop time')
    if self.pos == 0:
        pass
    if self.pos > 0:
        self.sell(self.bidprice1, 1)
        self.zhisun_label = True
        self.trade_records.append([bar.datetime, self.price, u'sell'])
    if self.pos < 0:
        self.cover(self.askprice1, 1)
        self.zhisun_label = True
        self.trade_records.append([bar.datetime, self.price, u'cover'])
```

从以上分析可以看出，该策略十分依赖于预测的准确性。若预测结果出现太多大涨大跌的情况，则会频繁开仓；若预测结果过于平稳，则会出现不交易的情况。若预测趋势与实际趋势相反，则会造成较大损失。

2.执行结果

（1）实时分类结果与得分显示：

inited	trading	pos	pos_rec	count	count_bar	class_1	class_2	class_3	class_4	class_5	score
True	True	1	0	33	33	3	1	1	1	1	1.8

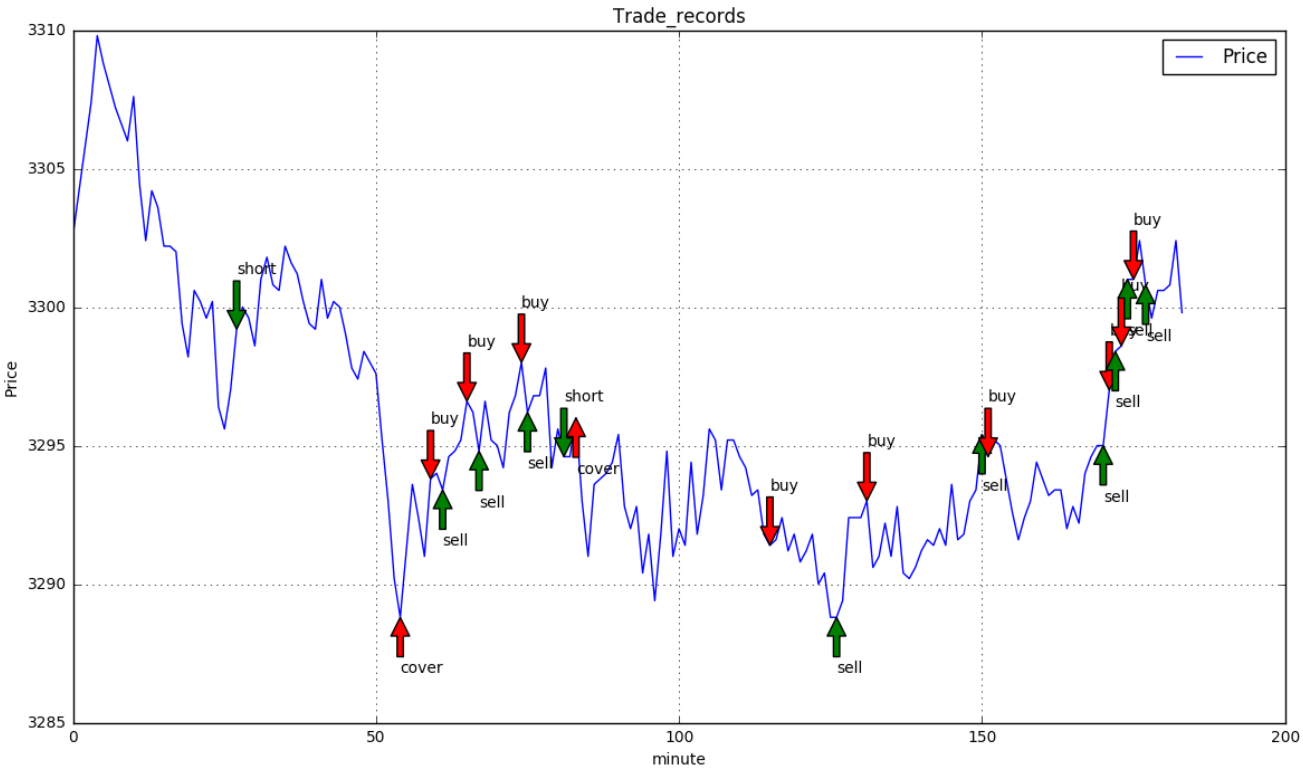
（2）日内交易情况表（部分）

71617	8	IF1701	沪深300股指1701	空	平仓	3296.0	1	10:51:28	CTP
71086	7	IF1701	沪深300股指1701	多	开仓	3298.0	1	10:51:02	CTP
67796	6	IF1701	沪深300股指1701	空	平仓	3294.4	1	10:43:32	CTP
66981	5	IF1701	沪深300股指1701	多	开仓	3297.0	1	10:42:01	CTP
65427	4	IF1701	沪深300股指1701	空	平仓	3291.2	1	10:37:13	CTP
64967	3	IF1701	沪深300股指1701	多	开仓	3293.8	1	10:36:01	CTP
62450	2	IF1701	沪深300股指1701	多	平仓	3290.6	1	10:30:12	CTP
56610	1	IF1701	沪深300股指1701	空	开仓	3298.8	1	10:04:01	CTP

（3）日内资金额

账户	昨结	净值	可用	手续费	保证金	平仓盈亏	持仓盈亏	接口
079240	996113.66	995611.1639	994591.1639	1522.4961		1020.0		CTP

（4）日内交易情况图



3.执行结果分析

目前来看，执行结果不太令人满意。主要不足有：

（1）预测结果有太多的大涨大跌趋势，导致频繁开仓。

（2）预测趋势准确率较低。从交易图示来看，11 笔交易中，趋势预测准确的只有 5 笔，未过半。

（3）止盈止损系数设置的过低，稍有波动就会平仓，导致持仓时间过短。如第二笔，预

测是正确的，第一分钟下跌，后几分钟都是上涨。但第一分钟就止损平仓了，未能抓住大趋势。

四、重新修改预测模型（目前）

1.修改内容：

针对策略实现显示出来的种种不足，需要重新对模型进行修改和优化。

（1）增大数据量。之前的预测模型的训练集只取了 10000 个序列，这对于深度网络需要的数据量远远不足。考虑将 IC、IH 合约的 bar 数据也进行处理，应用到模型中来。因为 tick 数据缺少最高价最低价等指标，后续指标难以添加，先不予考虑。目前的数据集增大到 90000 个序列。先用它们来训练，效果不理想时继续增大数据量。

（2）更改时间序列的长度。原来的时间序列为 5 分钟，相关性还不是很高。考虑到特征中有 26 分钟移动平均线这一特征，使得开盘后 26 分钟只收集数据，不做预测。同时因为第一分钟接收到的数据有误，予以删除，因此将时间序列调整为 25 分钟，既保证不会有数据浪费，又能使预测时参考数据大大增加，提高预测准确率。

（3）将数据集打乱重排再进行训练，能在一定程度上提高准确率。

2.修改后的结果：

增大数据量，拉长序列长度后，训练时间有了成倍的增加。训练完之后，准确率有了极大的提高。

（1）后一分钟涨跌分类准确率，84.79%：

```
Iter 850000, Minibatch Loss= 0.573726, Training Accuracy= 0.86400
Iter 875000, Minibatch Loss= 0.605627, Training Accuracy= 0.85000
Iter 900000, Minibatch Loss= 0.624047, Training Accuracy= 0.84800
Iter 925000, Minibatch Loss= 0.631733, Training Accuracy= 0.84600
Iter 950000, Minibatch Loss= 0.630488, Training Accuracy= 0.84400
Iter 975000, Minibatch Loss= 0.579400, Training Accuracy= 0.86000
Optimization Finished!
Model saved.

Model restored.
('Testing Accuracy:', 0.84793115)
```

（2）后两分钟涨跌分类准确率，69.12%：

```
Iter 850000, Minibatch Loss= 1.057175, Training Accuracy= 0.68000
Iter 875000, Minibatch Loss= 0.999548, Training Accuracy= 0.70800
Iter 900000, Minibatch Loss= 1.013177, Training Accuracy= 0.70000
Iter 925000, Minibatch Loss= 1.028159, Training Accuracy= 0.68600
Iter 950000, Minibatch Loss= 1.000371, Training Accuracy= 0.71200
Iter 975000, Minibatch Loss= 1.003331, Training Accuracy= 0.70400
Optimization Finished!
Model saved.

Model restored.
('Testing Accuracy:', 0.69120592)
```

（3）后三分钟涨跌分类准确率，58.4%：

```
Iter 850000, Minibatch Loss= 1.139645, Training Accuracy= 0.58200
Iter 875000, Minibatch Loss= 1.080260, Training Accuracy= 0.63400
Iter 900000, Minibatch Loss= 1.161784, Training Accuracy= 0.58000
Iter 925000, Minibatch Loss= 1.114544, Training Accuracy= 0.62200
Iter 950000, Minibatch Loss= 1.071264, Training Accuracy= 0.64200
Iter 975000, Minibatch Loss= 1.098840, Training Accuracy= 0.62800
Optimization Finished!
Model saved.

Model restored.
('Testing Accuracy:', 0.58405048)
```

（4）后四分钟涨跌分类准确率，54.6%：

```
Iter 850000, Minibatch Loss= 1.240263, Training Accuracy= 0.53600
Iter 875000, Minibatch Loss= 1.249962, Training Accuracy= 0.53400
Iter 900000, Minibatch Loss= 1.231785, Training Accuracy= 0.55600
Iter 925000, Minibatch Loss= 1.232758, Training Accuracy= 0.54000
Iter 950000, Minibatch Loss= 1.233661, Training Accuracy= 0.54800
Iter 975000, Minibatch Loss= 1.255963, Training Accuracy= 0.52800
Optimization Finished!
Model saved.

Model restored.
('Testing Accuracy:', 0.54603326)
```

（5）后五分钟涨跌分类准确率，49.88%：

```
Iter 850000, Minibatch Loss= 1.309224, Training Accuracy= 0.50200
Iter 875000, Minibatch Loss= 1.283159, Training Accuracy= 0.52800
Iter 900000, Minibatch Loss= 1.342526, Training Accuracy= 0.49800
Iter 925000, Minibatch Loss= 1.278099, Training Accuracy= 0.53000
Iter 950000, Minibatch Loss= 1.328031, Training Accuracy= 0.47600
Iter 975000, Minibatch Loss= 1.345263, Training Accuracy= 0.49400
Optimization Finished!
Model saved.

Model restored.
('Testing Accuracy:', 0.49888012)
```

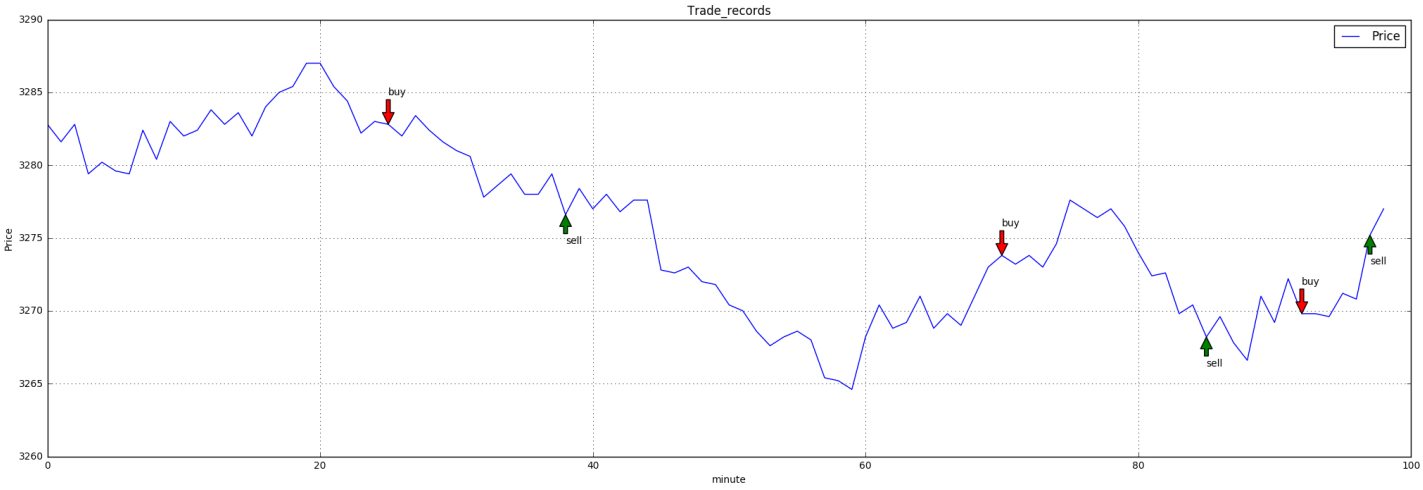
3.修改后的交易情况

（1）实时分类结果与得分显示：

inited	trading	pos	pos_rec	count	count_bar	real_class	predict_1	predict_2	predict_3	predict_4	predict_5	score	acc
True	True	0	0	42	42	3	3	3	5	5	5	3.8	6

其中，real_class 是当前时刻的实际分类，predict_1 至 predict_5 为预测的后 1-5 分钟的分类。acc 是今天所有分钟的 predict_1 与 real_class 的比值，即后一分钟预测成功率。（显示不合常理，可能代码有误，仍需修改）

（2）日内交易情况图



五、模型和策略分析

（1）修改后的模型在测试时性能有了明显改善，但是在实际应用中还是效果不好。之前是太多的出现大涨大跌的预测，现在是预测结果不灵敏，经常连续几分钟始终保持同样的预测结果。怀疑是 IC 和 IH 的涨跌幅与 IF 的涨跌幅区间不同，导致分类阈值设置的不准确有关。考虑更多的使用 IF 合约，加上前几年的和最近时间的。

（2）策略上，目前只能预测后五分钟的涨跌情况，即使预测准确率够高，开仓后要保证足够的持仓时间（大于五分钟），当出现前面几分钟涨后面几分钟大跌的情况，也是无法盈利的。另一方面，预测的时间过长，准确率肯定不会高。因此还需要对策略进行修改，确定更合理的建仓时间。