A Forecasting FX Trading Strategy with LSTM model

1.Abstract

The overall task involves using LSTM network as training methodology to analyze effectiveness in forecasting prices of the FX futures mainly on 5-minute USD/CNY data from July 2019 till May 2020 for trading.

As for the trading strategy, first, use the prediction value to decide when to buy. If the prediction value is higher than the real value, set up an indicator entry as +1, if lower, set -1. Then, calculate the entry differences in each period, buy when entry is +2, sell when entry is -2. Finally, plot the strategy return, and perform technical analysis about the back testing result. The result has shown that LSTM model is not that accuracy in forecasting currency futures but is a feasible way with high return rate to trade currency futures.

2.LSTM model review

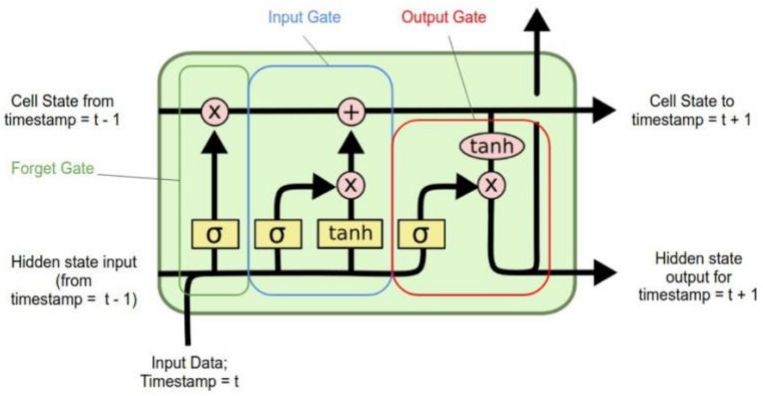
As we discussed in our lecture, LSTM model includes these three parts: Forget Gate, Input Gate, Output Gate.

Qi hang (2020) constructed comparison of ARIMA, ANN and LSTM model for Stock Price Prediction. He concluded that LSTM model performed better and showed certain advantages in prediction process with LSTM model.

Ching-I (2019) discussed the problem of next day exchange rate prediction by using LSTM and concluded the last seven-day historical data outperformed.

For this model, I'm interested in using LSTM model to predict exchange rate in USD/CNY futures' prices.

Figure 1: LSTM definition



3.LSTM model implement

3.1 Dataset preparation

First, based on conclusions from Ching-I (2019). I choose 5-minute USD/CNY data from 7/1/2019 to 5/20/2020. This is the tail data I utilize to train/develop and test my model with 26277 data.

Figure 2: dataset tail

	open	low	close	high
26272	7.1042	7.1042	7.1048	7.1055
26273	7.1047	7.1038	7.1038	7.1047
26274	7.1039	7.1039	7.1046	7.1047
26275	7.1048	7.1043	7.1045	7.1049
26276	7.1049	7.1049	7.1057	7.1057

Then, set up the MinmaxScaler with preprocessing package from sklearn. This method is to reshape and transform the data range 0 to 1.

Figure 3: MinMaxScaler function

```
#MinMaxScaler
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
df0=min_max_scaler.fit_transform(df1)
df = pd.DataFrame(df0, columns=df1.columns)
input_size=len(df.iloc[1,:])
```

Finally, convert and reshape an array of values into a dataset matrix, split it into train and test data. We have x train & test, y train & test data, since we have four columns, we employ the sequence length as four plus one, which is five.

Figure 4: Convert and reshape the dataset

```
stock=df
seq_len=4
amount_of_features = len(stock.columns)#column
data = stock.to_numpy() #pd.DataFrame(stock) transform to matrix
sequence_length = seq_len + 1#len4+1
result = []
for index in range(len(data) - sequence_length):#recursive -5 times
    result.append(data[index: index + sequence_length])#from i to i+5
result = np.array(result)#get161sample, 6*3
row = round(0.9 * result.shape[0])#set train set
train = result[:int(row), :]
## reshape input to be [samples, time steps, features] which is required for LSTM
x_train = train[:, :-1]
y_train = train[:, -1][:,-1]
x_test = result[int(row):, :-1]
y_test = result[int(row):, -1][:,-1]
#reshape 5*3
X_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], amount_of_features))
X_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], amount_of_features))
```

3.2 GRU model

Here is GRU model result. Dense means return a sequence of vectors of dimension 8. The last two layer is added to get output in format needed by the user. Here Dense (4) means 4 different classes output will be generated with 10 epochs with MSE loss function. Batch Size determines training number for one train. Adam optimization is stochastic gradient descent method that is based on adaptive estimation of first order and second-order moments, which is suitable for financial market. The accuracy metrics calculates how often predictions equal labels. Fit the model.

Figure 5: GRU model

```
#get GRU
d = 0.01
model = Sequential()
model.add(layers.LSTM(8, input_shape=(window, input_size), return_sequences=False))
model.add(Dropout(d))
model.add(Dense(4))
model.add(Dense(1))
model.compile(loss='mse',optimizer='adam',metrics=['accuracy'])
model.fit(X_train, y_train, epochs = 10, batch_size = 1) #train 1000
```

```
Epoch 1/10
23648/23648 [==============================] - 18s 714us/step - loss: 4.1044e-04 - accuracy: 8.4574e-05
Epoch 2/10
23648/23648 [==============================] - 16s 690us/step - loss: 1.5254e-04 - accuracy: 8.4574e-05
Epoch 3/10
23648/23648 [==============================] - 17s 705us/step - loss: 1.4979e-04 - accuracy: 8.4574e-05
Epoch 4/10
23648/23648 [==============================] - 17s 703us/step - loss: 1.3894e-04 - accuracy: 8.4574e-05
Epoch 5/10
23648/23648 [==============================] - 16s 689us/step - loss: 1.3973e-04 - accuracy: 8.4574e-05
Epoch 6/10
23648/23648 [==============================] - 16s 687us/step - loss: 1.4548e-04 - accuracy: 8.4574e-05
Epoch 7/10
23648/23648 [==============================] - 16s 688us/step - loss: 1.3366e-04 - accuracy: 8.4574e-05
Epoch 8/10
23648/23648 [==============================] - 17s 703us/step - loss: 1.3705e-04 - accuracy: 8.4574e-05
Epoch 9/10
23648/23648 [==============================] - 17s 707us/step - loss: 1.4587e-04 - accuracy: 8.4574e-05
Epoch 10/10
23648/23648 [==============================] - 18s 745us/step - loss: 1.3555e-04 - accuracy: 8.4574e-05

<keras.callbacks.History at 0x1a4767e80>
```

Then test it on the training data, and testing data. Plot the real price and the predicted price. It seems accurate but shows lag prediction in each graph because of the Markov property.

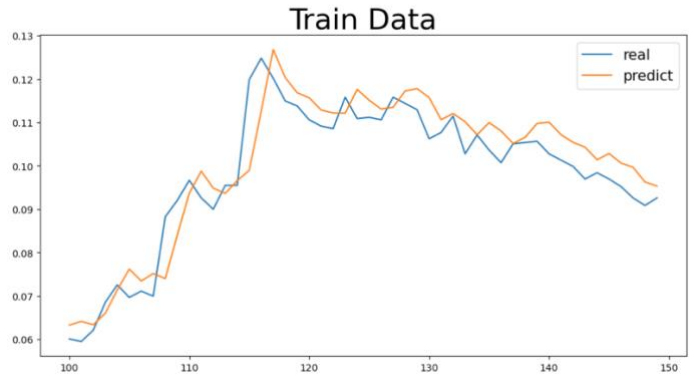
Figure 6: Train Data & Test data

```
#show the fit outcome
y_train_predict=model.predict(X_train)
y_train_predict=y_train_predict[:,0]
```

```
739/739 [==============================] - 1s 456us/step
```

```
plt.rcParams['figure.figsize']=16,8
draw=pd.concat([pd.DataFrame(y_train),pd.DataFrame(y_train_predict)],axis=1)
draw.iloc[100:150,0].plot(figsize=(12,6))
draw.iloc[100:150,1].plot(figsize=(12,6))
plt.legend(('real', 'predict'),fontsize='15')
plt.title("Train Data",fontsize='30')
#show the train data
```

```
Text(0.5, 1.0, 'Train Data')
```




```
#test on test data
y_test_predict=model.predict(X_test)
y_test_predict=y_test_predict[:,0]
```

```
83/83 [==============================] - 0s 513us/step
```

```
plt.rcParams['figure.figsize']=16,8
draw=pd.concat([pd.DataFrame(y_test),pd.DataFrame(y_test_predict)],axis=1);
draw.iloc[200:250,0].plot(figsize=(12,6))
draw.iloc[200:250,1].plot(figsize=(12,6))
plt.legend(('real', 'predict'),loc='upper right',fontsize='15')
plt.title("Test Data",fontsize='30')
#show the test data
```

```
Text(0.5, 1.0, 'Test Data')
```

Finally show the outcome with MAE/MSE/MAPE value in training set and testing set.

1.MAE refers to Mean Absolute Error, which is $\frac{1}{n}\sum_1^n |\, y_i - \hat{y}_i \,|$.

This gives less weight to outliers, which is not sensitive to outliers. The closer MAE is 0, the more accurate the model is. Two set seems accurate (close to 0) on this estimator measure.

2.MAPE refers to Mean Absolute Percentage Error which is $\frac{100}{n}\sum_i^n \frac{y_i - \hat{y}_i}{y_i}$

Like MAE but normalized by true observation. Over 100% in two sets, this means the error is greater than the actual value.

3.MSE refers to Mean Squared Error, which is $\frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2$

MSE is like a combination measurement of bias and variance of your prediction.

Training/ testing set MAE/MSE are close which is higher than testing set. That means the training model could find an optimization well. MAPE results are higher than 25%.

The total accuracy is 48%, which is not that high. Consider the foreign exchange rate price has low volatility and outliers the performance should base more on MAE/MSE. Now let's perform it in our trading strategy.

Figure 7: Outcome

```
: #show the outcome
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
import math
def mape(y_true, y_pred):
    return np.mean(np.abs((y_pred - y_true) / y_true)) * 100
print('train set MAE/MSE/MAPE')
print(mean_absolute_error(y_train_predict, y_train))
print(mean_squared_error(y_train_predict, y_train) )
print(mape(y_train_predict, y_train) )
print('test set MAE/MSE/MAPE')
print(mean_absolute_error(y_test_predict, y_test))
print(mean_squared_error(y_test_predict, y_test) )
print(mape(y_test_predict,  y_test) )
y_var_test=y_test[1:]-y_test[:len(y_test)-1]
y_var_predict=y_test_predict[1:]-y_test_predict[:len(y_test_predict)-1]
txt=np.zeros(len(y_var_test))
for i in range(len(y_var_test-1)):
    txt[i]=np.sign(y_var_test[i])==np.sign(y_var_predict[i])
result=sum(txt)/len(txt)
print('accuracy:',result)
print('train time:',54.56)
```

```
train set MAE/MSE/MAPE
0.00297978410896873 13
2.6603637225497838e-05
1.0556845787499456
test set MAE/MSE/MAPE
0.002355884565188878
1.7169048481157627e-05
0.330892471545489
accuracy: 0.4885757806549886
train time: 54.56
```

4.Trading Strategy implement

The idea about this trading strategy is that we compare the predict value (pv) with test value (tv).

If pv is less than tv, set signal +1, else 0 means buy. If pv is higher than tv, set signal -1 means sell. Then calculate the holding return which is the return you buy asset and hold it till the end, calculate the system return which is the return you buy asset with this strategy.

Build an entry column to storage the difference between each signal column.

Buy the asset when entry = +2, sell the asset when entry = -2.

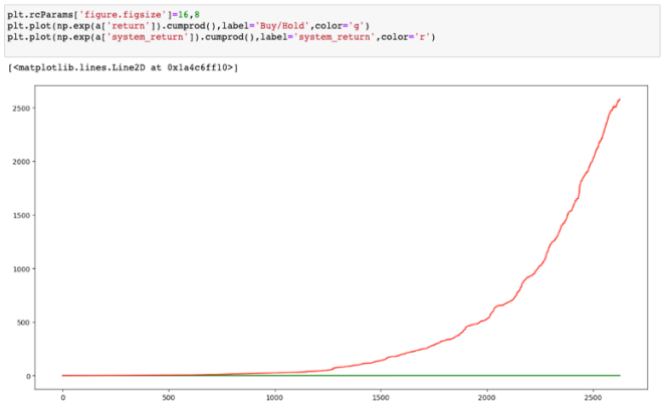
Figure 8: Trading implement

```
#reshape it into df
a = pd.DataFrame(y_test)
#set up predict,test column
a['predict'] = pd.DataFrame(y_test_predict)
a.columns=['test','predict']
#if predict value is lower than the test value, buy
a['signal']=np.where(a['predict']< a['test'],1,0)
#if predict value is higher than the test value,sell
a['signal']=np.where(a['predict']> a['test'],-1,a['signal'])
#drop the NA value
#calculate the return when you buy the asset and hold it
a['return']=np.log(a['test']).diff().dropna()
#calculate the return when you employ the trading system
a['system_return']=a['signal']*a['return']
#calculate the entry column
a['entry']=a['signal'].diff()
pd.options.display.max_rows=None
```

5.Results analysis

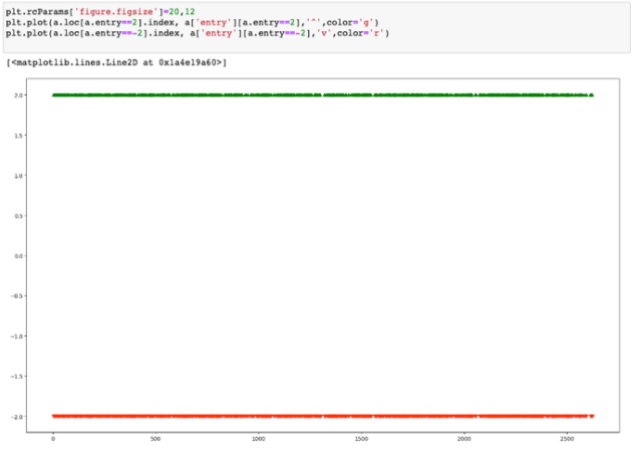
The system return outcome seems crazy but it's possibly true with this 5-minute dataset high-frequency trading. Since exchange rate value is not that volatility and transaction fee cost is low (almost 0).

Figure 9: Return outcome



These are time spots we should buy & sell. We buy 574 times and sell 575 times in 26277 data. Sometimes, we buy and sell the price within 5 minutes.

Figure 10: Buy & sell timeline



```
print(a.loc[a.entry==2].index)
```

```
Int64Index([   3,    7,   10,   13,   19,   23,   28,   30,   36,   42,
            ...
            2563, 2566, 2572, 2581, 2592, 2594, 2612, 2615, 2617, 2623],
           dtype='int64', length=574)
```

```
print(a.loc[a.entry==-2].index)
```

```
Int64Index([   2,    6,    9,   11,   14,   21,   24,   29,   34,   41,
            ...
            2564, 2569, 2577, 2583, 2593, 2595, 2614, 2616, 2618, 2624],
           dtype='int64', length=575)
```

Now, let's consider the Sharpe Ratio and the Maximum drawdown.

I calculate these two indicators with two functions. The buy & hold strategy Sharpe ratio is about 6%, and the system strategy Sharpe ratio is 58% which seems better.

The maximum drawdown in system strategy is extremely low which means the profit is stable. Due to the low volatility of the exchange rate, we should employ system strategy for a longer period.

Figure 11: Sharpe Ratio

```
def calculate_sharpe(data):
    avg_return=data.mean()
    std_return=data.std()
    sharp= avg_return/std_return
    return sharp
```

```
dfbuy=a['return'].dropna()
dfsystem=a['system_return'].dropna()
```

```
calculate_sharpe(dfbuy)
```

```
0.007721150439567882
```

```
calculate_sharpe(dfsystem)
```

```
0.5798021224729354
```

Figure 12: Maximum drawdown

```
dfbuy=list(dfbuy)
dfsystem=list(dfsystem)
```

```
def get_max_drawdown_fast(array):
    drawdowns=[]
    max_so_far=array[0]
    for i in range(len(array)):
        if array[i]> max_so_far:
            drawdown=0
            drawdowns.append(drawdown)
            max_soz_far=array[i]
        else:
            drawdown=max_so_far- array[i]
            drawdowns.append(drawdown)
    return max(drawdowns)
```

```
get_max_drawdown_fast(dfbuy)
```

```
0.06145121240275331
```

```
get_max_drawdown_fast(dfsystem)
```

```
0.0075419421908195505
```

6. Improvements with relevant topic

1. Consider the turnover rate and try to add transaction cost.
2. Combine with more restrict rules such as Bollinger Bands, RSI indicator to choose the signal.
3.Employ this strategy with different exchange rate in G7 currencies and commodities futures (Gold, Oil).
4.Improve strategy order system to different types of orders (Limit/Market/FAK) based on the trading signals.

7.References

[1]. Qihang Ma, “Comparison of ARIMA, ANN and LSTM for Stock Price prediction” Web of Conferences 218, 01026, 2020.
[2]. Ching-I-Lee, Chia-Hui Chang, “Currency Exchange Rate Prediction with LSTM Networks Based on Attention and News Sentiment Analysis. 2019.