Customize loss function to make LSTM model more applicable in stock price prediction

Loss function should not consider price difference only, directional loss also matters !!!

Zedric Cheung Jun 20, 2020 · 8 min read ★

Photo by Nick Chong on Unsplash

Background

There are many tutorials or articles online teaching you how to build a LSTM model to predict stock price. Either it is simple or sophisticated, we can somehow obtain a "desirable" result, something similar to the below graph (Exhibit 1). Yes, it is desirable if we simply judge the model by looking at mean squared error (MSE).

But is it good enough to do well and help us earn big money in real world trading? Sorry to say, the answer is always NO. In this article, we would give a try to customize the loss function to make our LSTM model more applicable in real world.

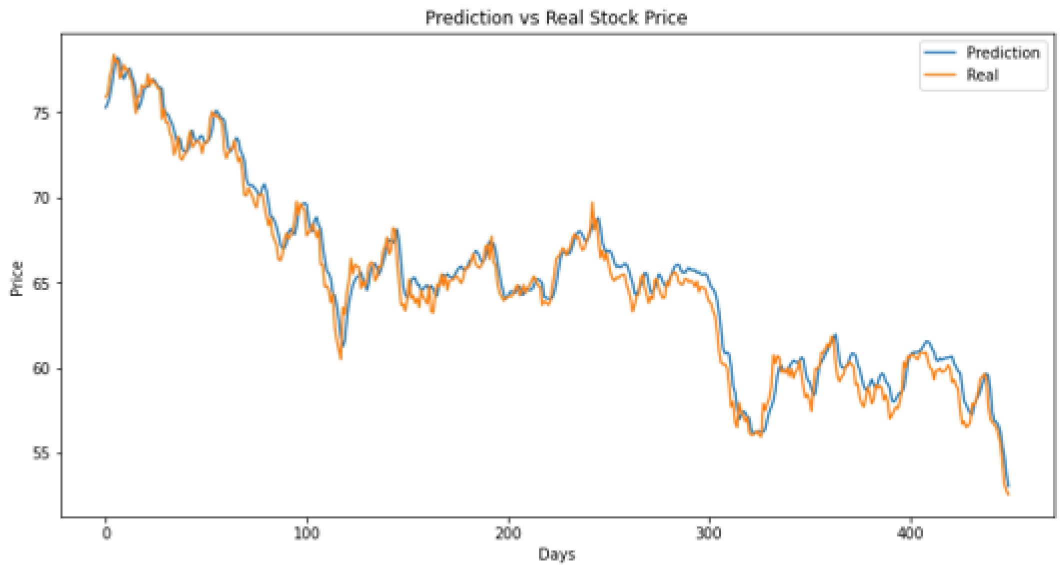

Exhibit 1 — Prediction of HSBC's (0005.HK) stock price based on a simple LSTM model

Problems faced

It's not because something goes wrong in the tutorials or the model is not well-trained enough. But fundamentally, there are several major limitations that are hard to solve.

1. **All free libraries only provide daily data of stock price** — *without real-time data, it's impossible for us to execute any orders within the day*

2. **The commonly used loss function (MSE) is a purely statistical loss function** – *pure price difference doesn't represent the full picture*

3. **LSTM model or any other recurrent neural network model is always a black box** — *trading strategy can only be based on price movement without any reasons to support, and the strategies are hard to extend to portfolio allocation*

The limitations (1) and (3) are hard to solve without any more resources. For (1), the solution may be connecting to real time trading data provider such as Bloomberg, and then train up a real-time LSTM model. The trading orders for next second can then be automatically placed. But sorry to say, it's hard to do so if you are not working on trading floor.

For (3), if aiming to extend to portfolio allocation with some explanations, probably other concepts like mean-variance optimization, with some robust estimators and then considering Value at Risk (VaR) are more appropriate. But those are completely other stories.

In this article, we would like to pinpoint the second limitation and focus on one of the possible ways — **Customize loss function by taking account of directional loss** to make the LSTM model more applicable given limited resources.

Existing difficulties

Let's back to the above graph (Exhibit 1). It looks perfect and indicates that the model's prediction power is very high. True, its MSE for training loss is only 0.000529 after training 300 epochs, but **its accuracy on predicting the direction of next day's price movement is only 0.449889**, even lower than flipping the coins !!!

MSE mainly focuses on the difference between real price and predicted price without considering whether the predicted direction is correct or not. This characteristic would create huge troubles if we apply trading strategies like put / call options based on the prediction from LSTM model. From such perspective, correctness in direction should be emphasized.

Furthermore, the model is daily price based given data availability and tries to predict the next day's close price, which doesn't capture the price fluctuation within the day. Under such condition, directional accuracy is even more important than the price difference. Even you may earn less on some of the days, but at least it won't lead to money loss.

Methodology

Now, let's start to customize the loss function. For the details of data pre-processing and how to build a simple LSTM model stock prediction, please refer to the Github link here. Full codes could be also found there.

Step 1: Extract necessary information from the input tensors for loss function

```
def custom_loss(y_true, y_pred):

        #extract the "next day's price" of tensor
        y_true_next = y_true[1:]
        y_pred_next = y_pred[1:]
        #extract the "today's price" of tensor
        y_true_tdy = y_true[:-1]
        y_pred_tdy = y_pred[:-1]
```

Always remember that the inputs for the loss function are two tensors, *y_true* (the true price) and ***y_pred*** (the predicted price)**.** First, we have to create four new tensors to store the “next day’s price” and “today’s price” from the two input sensors for further use.

Step 2: Create new tensors to record the price movement (up / down)

```
#substract to get up/down movement of the two tensors
y_true_diff = tf.subtract(y_true_next, y_true_tdy)
y_pred_diff = tf.subtract(y_pred_next, y_pred_tdy)

#create a standard tensor with zero value for comparison
standard = tf.zeros_like(y_pred_diff)

#compare with the standard; if true, UP; else DOWN
y_true_move = tf.greater_equal(y_true_diff, standard)
y_pred_move = tf.greater_equal(y_pred_diff, standard)
```

The **tf.substract** is to substract the element-wise value in ***y_true_tdy*** tensor from that in ***y_true_next*** tensor. We then compare the two difference tensors (***y_true_diff*** and ***y_pred_diff***) with a standard zero tensor. If the value is greater than or equal to zero, then it belongs to an upward movement, otherwise downward.
The **tf.greater_equal**will return a boolean tensor.

Step 3: Find out indices when the movement of the two tensors are not in same direction

```
#find indices where the directions are not the same
condition = tf.not_equal(y_true_move, y_pred_move)
indices = tf.where(condition)

ones = tf.ones_like(indices)
indices = tf.add(indices, ones)
```

(a) The **tf.not_equal** compares the two boolean tensors, ***y_true_move*** and ***y_pred_move***, and generates another new boolean tensor — **condition**. If the direction in the next day is the same between the true movement and the predicted movement, **True** is returned, otherwise **False**.

(b) The **tf.where** returns the position of "True" in the **condition** tensor.

(c) The **tf.add** adds one to each element in **indices** tensor. If you are careful enough, you may notice that the shape of any processed tensors is (49, 1) , one unit shorter than the that of original inputs (50, 1). Adding one means that we move the indices one day later, which represents the true location of next day within the original input tensors.

Step 4: Create a tensor to store directional loss and put it into custom loss output

```
direction_loss = tf.Variable(tf.ones_like(y_pred), dtype='float32')
updates = K.cast(tf.ones_like(indices), dtype='float32')
alpha = 1000
direction_loss = tf.scatter_nd_update(direction_loss, indices,
alpha*updates)

custom_loss = K.mean(tf.multiply(K.square(y_true - y_pred),
direction_loss), axis=-1)
```

Now, we are creating the most important tensor — ***direction_loss***. Since it should be a trainable tensor and be put into the final output — ***custom_loss***, it has to be set as a variable tensor using ***tf.Variable.***

The tensor ***indices*** has stored the location where the direction doesn't match between the true price and the predicted price. Through ***tf.scatter_nd_update***, we can update the values in tensor ***direction_loss*** by specifying the location and replaced with new values. But keep in mind that shapes of indices and updates have to be the same. The end product of ***direction_loss*** is a tensor with value either 1 or 1000.

Last by not least, we multiply the squared difference between true price and predicted price with the ***direction_loss*** tensor. **The concept here is that if the direction matches between the true price and the predicted price for the day, we keep the loss as**

squared difference. If it doesn't match, then we multiply the squared difference by alpha (1000).

Finally, a customized loss function is completed. We have now taken consideration of whether the predicted price is in the same direction as the true price. If we apply LSTM model with the same settings (batch size: 50, epochs: 300, time steps: 60) to predict stock price of HSBC (0005.HK), the accuracy to predict the price direction has increased from 0.444343 to 0.561158. A big improvement but still far from perfect. We will discuss some hurdles to overcome at the last part of this article if we want to build an even better loss function.

Some tricks that can help to save your time

Most of the time, we may have to customize the loss function with completely different concepts from the above. Below are some tricks that can help to save your time or track errors during the process.

(a) ***get_shape*** — when you are not sure about the tensor's shape, never hesitate to use this function to print it out. Nearly all the processing functions require all inputted tensors' shape to be the same.

(b) ***keras.backend.cast*** — when the error message says the format of elements in the tensor doesn't match with others', try to use this function to change the format of the tensors' elements into specific type.

(c) ***tensorflow.reshape*** — when the error message says the shape doesn't match with the original inputs, which should hold a consistent shape of (x, 1), try to use this function ***tf.reshape(tensor, [-1])*** to flatten the tensor.

(d) ***custom_loss*** — keep in mind that the end product must consist of the two inputted tensors, ***y_true*** and ***y_pred***, and will be returned to the main body of the LSTM model to compile.

More hurdles to overcome…

The result now has shown a big improvement, but still far from perfect. However, to step further, many hurdles are waiting us, and below are some of them.

(a) Hard to balance between price difference and directional loss — if alpha is set to be too high, you may find that the predicted price shows very little fluctuation. If we plot it, it's nearly a flat line. This means that directional loss dominates the loss function. Under such situation, the predicted price becomes meaningless but only its direction is meaningful. We are simply betting whether the next day's price is upward or downward.

(b) Hard to apply categorical classifier on stock price prediction —many of you may find that if we are simply betting the price movement (up/down), then why don't we apply categorical classifier to do the prediction or turn the loss function as ***tf.binary_crossentropy***. Sorry to say, the result shows no improvement. I have tried to first convert all the price data into "movement data" represented by 0 (down) or 1 (up), and input them for training. But since the nature of the data is time series, unlike handwriting recognition, the 0 or 1 arrays in every training batch are not distinguished enough to make the prediction of next day's price movement. Some methods like support vector machine (SVM) and convolutional neural network (CNN), which perform very well in classification, are hard to apply to this case.

(c) Alpha is very specific for every stock —I have tried to apply the same model on stock price prediction for other 10 stocks, but not all show big improvements. For every stock, the relationship between price difference and directional loss seems very unique. So we may have to spend lots of time to figure out what's the best combination for each stock.

Conclusion

It's always not difficult to build a desirable LSTM model for stock price prediction from the perspective of minimizing MSE. But it is far from applicable in real world. This article introduces one of the possible ways — **Customize loss function by taking account of directional loss**, and have discussed some difficulties during the journey and provide some suggestions. I hope that it would open the discussion on how to improve our LSTM model. As mentioned, there are many hurdles have to be overcome if we want to step further, especially given limited resources.

This article is also my first publication on Medium. In the future, I will try to explore more about application of data science and machine learning techniques on economics and finance areas. Thanks for supports !!!

Reference

1. Asutosh Nayak. (2019). Predicting Stock Price with LSTM.
2. Dhiraj K. (2019). How to create a custom loss function in Keras.
3. Eyal Zakkay. (2019). Advanced Keras — Constructing Complex Custom Losses and Metrics.