Making Algorithmic Trading Strategies Better with Deep Learning

Dennis Kuzminer
FINUY4903A Blog Post

During volatile market conditions, which seemingly are becoming more common, investors who employ algorithmic trading strategies have copious opportunities for making a profit on price swings. As machine learning and deep learning research strengthen, some wonder whether those models can be adapted to predict those swings. In this post, let's explore how we can predict prices using deep-learning tactics and how to incorporate them into a simple strategy.

What are we looking for?

In order to predict securities prices, it is important to understand what kind of movements stocks take on. We are looking to identify when there are continuous upward or downward trends. Short-term volatility is also responsible for the swings that we are after. The model, however, should capture the intricacies of market data, namely cyclicality such as earnings calls, corporate actions, or possibly even the "January effect," mean-reversion—the idea that prices retrace to a long-running average price—and noise.

What information could be useful to us?

Obviously, price data, represented in intervals with aggregated Open, High, Low, Close, and Volume values, is our main dataset. However, to derive additional data from the data set, we can engineer more "features" or variables to potentially increase the predictive capabilities of our model by introducing additional relationships. This could come in the form of two separate categories. First, we can include certain technical

indicators, including moving averages (MA), moving average convergence divergence (MACD), commodity channel index (CCI), average true range (ATR), Bollinger bands (BOLL), price rate of change (ROC), among others. Second, adding broader market movements, like the S&P 500 and the VIX as well as sentiment analysis of the news, could give context to the meaning of the traits determined by the indicators. For this post, we'll keep it simple.

Deeper into deep learning

Because of the non-linear, cyclical, somewhat random, multi-variate nature of stock market data, deep learning models, created to capture those sorts of relationships, are useful in this context. Specifically, let's examine a Long Short-Term Memory (LSTM) model, which is a type of neural network.

A neural network is designed to make predictions based on input data in a similar fashion to the way in which our brain may recognize or classify information. The model constructs a graph-like structure that maps the input layer (set) to the output layer (set) with hidden layers in between. The nodes in the graph are connected by edges to every node within the next layer with corresponding weights. The weights in conjunction with the node's (neuron's) value (activation) can be used (to calculate a weighted sum) to determine the next node or set of nodes. The hidden layers are sets of nodes that are each a partially combined representation of the previous layer. As a new input value traverses the graph, the model will find the predicted output based on how closely each subcomponent of the input value matches the subcomponents it is familiar with. This is why such models are used for image recognition and why, with their pattern-matching abilities, we will use a variant of it for pattern-heavy stock data.

In order to correctly classify what an important pattern to follow is and what is just the inherent randomness of the data, the LSTM model employs the same ideas of a neural network but with some added conditions. First, upon computing the value of the node, it feeds the output value of the activation function back into itself in addition to the

next input. Next, it maintains a memory cell in each node in the form of an input, output, and forget gate. The input gate determines whether new information should be added to the memory cell; the output gate determines what information should be passed onto the next cell; and the forget gate determines which long-term information is relevant to the cell state, which affects the activation function.

Applying these tools

For the LSTM model, we will use the `tensorflow.keras` library. Import the necessary packages.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
```

Next, import the data, extract the prices, and use the `MinMaxScaler` to scale the prices. I am using S&P 500 data with a 5-minute frequency over the last 15 years. That is a dataset with over 750,000 records, which tends to be quite slow, especially with something like the LSTM model. As a result, we can make use of the `resample` function with `agg` to combine the data into a daily frequency.

```
data = pd.read_csv("data.txt")
data = data.infer_objects()
data.columns = ['DateTime', 'Open', 'High', 'Low', 'Close', 'Volume']
data['DateTime'] = pd.to_datetime(data['DateTime'])
data = data.set_index("DateTime").resample('D').agg({'Open': 'first', 'High':
'max', 'Low': 'min', 'Close': 'last', 'Volume': 'sum'}).reset_index().dropna()
prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler()
prices = scaler.fit_transform(prices)
```

Now, we must prepare the data for the model. For the LSTM model to understand the current stock's trend or environment, we can create rolling windows of context for the model to generate the logical next movement given the previous `window_size` number of points. This ensures that extremely old points do not have much of an influence on the prediction. We can then split the data into a training and testing set.

```
def partition(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:(i + seq_length)])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

window_size = 60
X, y = partition(prices, window_size)

train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

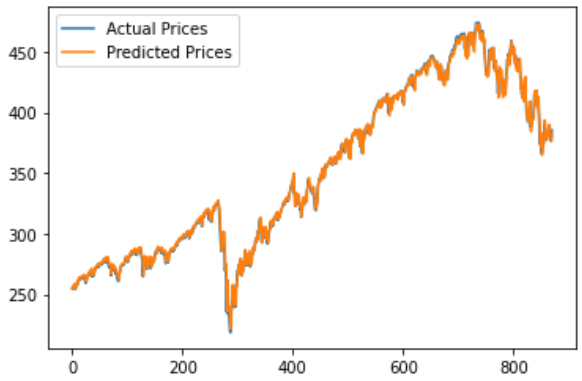
Finally, fit the LSTM model on the training data, predict the testing data, and then transform it back.

```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100, batch_size=32)

y_pred = model.predict(X_test)
y_test = scaler.inverse_transform(y_test)
y_pred = scaler.inverse_transform(y_pred)
```

Now, let's see the predictions.



Looks like the predictions are fairly on par with the actual prices. These are no normal market conditions either; as you could probably guess by the shape of the line, this was the year preceding and 2 years following the COVID-19 pandemic event.

Performance?

Let's now compare the performance of a buy-and-hold strategy against a simple strategy: buy if the model thinks the next day's price will be higher than today's and sell if the opposite.

```
y_test = y_test.flatten()
y_pred = y_pred.flatten()

signals = np.where(y_pred > y_test, 1, -1)

lstm_returns = signals[:-1] * (y_test[1:] - y_test[:-1])
buy_and_hold_returns = y_test[1:] - y_test[:-1]

lstm_cumulative_returns = np.cumsum(lstm_returns)
buy_and_hold_cumulative_returns = np.cumsum(buy_and_hold_returns)

plt.plot(lstm_cumulative_returns, label="LSTM Strategy")
plt.plot(buy_and_hold_cumulative_returns, label="Buy-and-Hold Strategy")
plt.xlabel("Days")
plt.ylabel("Cumulative Returns")
plt.legend()
plt.show()
```


```python
def sharpe_ratio(returns, risk_free_rate=0.0, trading_days=252):
    excess_returns = returns - risk_free_rate
    annualized_return = np.mean(excess_returns) * trading_days
    annualized_volatility = np.std(excess_returns) * np.sqrt(trading_days)
    return annualized_return / annualized_volatility

lstm_sharpe_ratio = sharpe_ratio(lstm_returns)
buy_and_hold_sharpe_ratio = sharpe_ratio(buy_and_hold_returns)

lstm_annualized_return = np.mean(lstm_returns) * 252
lstm_annualized_volatility = np.std(lstm_returns) * np.sqrt(252)
buy_and_hold_annualized_return = np.mean(buy_and_hold_returns) * 252
buy_and_hold_annualized_volatility = np.std(buy_and_hold_returns) * np.sqrt(252)

print("LSTM Strategy:")
print(f"Sharpe Ratio: {lstm_sharpe_ratio}")
print(f"Annualized Return: {lstm_annualized_return}")
print(f"Annualized Volatility: {lstm_annualized_volatility}")

print("\nBuy-and-Hold Strategy:")
print(f"Sharpe Ratio: {buy_and_hold_sharpe_ratio}")
print(f"Annualized Return: {buy_and_hold_annualized_return}")
print(f"Annualized Volatility: {buy_and_hold_annualized_volatility}")
```

```
LSTM Strategy:
Sharpe Ratio: 1.0544738161079705
Annualized Return: 74.32708137931017
Annualized Volatility: 70.4873655883169
```

```
Buy-and-Hold Strategy:
Sharpe Ratio: 0.5333237638444049
Annualized Return: 37.654187586206895
Annualized Volatility: 70.60286853670438
```

We can see that all metrics, the Sharpe ratio, returns, and volatility, are better for the LSTM strategy without factoring in transaction costs.

There are ways we can make it better, however. Mainly, we could limit the number of trades so as to not incur heavy fees. This can be done by adding additional signals or criteria to be met in order to allow the model to trade (like the technical indicators mentioned earlier). We could also add a downward stop loss so that our strategy could pull out of the market for some time whenever it experiences a sharp dip in returns. Lastly, while this was just a simple exploration, we could use the `keras-tuner` library to optimize the hyperparameters for the model such as output space dimensionality, loss function, epochs, and batch size (if you have a powerful computer).

Conclusion

LSTM models have been a large topic of research in their ability to be a powerful and effective tool for demystifying trends. Their features allow them to model the complex characteristics of financial markets. As we demonstrated in this blog post, incorporating an LSTM model into a trading strategy can provide valuable insights when it comes to buying or shorting stocks.

This blog post uses the ideas put forward in these articles:

1. http://cs229.stanford.edu/proj2017/final-reports/5241098.pdf
2. https://arxiv.org/pdf/1904.04912.pdf