A neural network approach to returns prediction

Introduction and rules

The data you will be working on is going to be a financial time-series consisting of date, open, high, low, close and volume. The security is the Bitcoin price in US dollar, but it doesn't really matter what asset the data is from, the methodology can be applied to any asset.

The goal is to use some indicators of past price action to try and predict future direction. Some of these indicators are known as "technical analysis".

The libraries used will be pandas for handling the time series data, pandas-ta for the library of techincal indicators (https://pypi.org/project/pandas-ta/), and any neural network package you're comfortable using for the neural network. I quite like PyTorch.

From myself you will receive:
(1) the dataset consisting of the financial time series
(2) an initial .py file

What you need to deliver is a single .py file that sequentially performs all the tasks listed below. Please save it as "*Name_Surname.py*". It should run by simply changing the initial path file for the dataset. You can load into the main file any package you like, but please don't use any "own"/private packages. If you have done something really exotic, then you may consider also submitting a word document where you explain what you've done. But please do this only if necessary, in most cases commenting your Python code will be enough.

The winner will be the project that achieves the highest Sharpe ratio (provided everything else is computed correctly). From my experience, maximizing the accuracy of the network also maximises the Sharpe ratio. Other factors may be considered as well, such as the depth of the research (tried many time horizons, improved the network beyond the basic version, etc.) or the clarity of the code. The winner will win a £100 Amazon voucher.

The deadline is the 12th of April at 23:59.

For any queries, drop me an e-mail at ggvecchio@gmail.com.

Best of luck,

Dr Giovanni Gabriele Vecchio

Outline of the project

Part A: Preliminary data preparation.

a) Set up your machine with the required packages. I personally use Anaconda and code with Spyder. Anacoda has a very convenient package management system.

b) Finish adding all indicators to the dataframe. Note some indicators need to be computed on a secondary dataframe first, and then added to the main dataframe.

c) Stationarize the data. The data is now in "price" units, so it has a scale. In most machine learning applications, data must be standardized before being fed into a neural network.

c.1) Prices are stationarized by computing log returns:

$$r(t) = \log(Close(t) / Close(t-1))$$

c.2) Moving averages are stationarized in a similar fashion:

$$ma(t) = \log(MA(t) / Close(t))$$

Note that this means that trading signals change a bit. For example the sell condition $MA(t) > Close(t)$ becomes $ma(t) > 0$.

c.n) Repeat this process for all indicators.

The collection of all standardized indicators will be our dataset of predictors X.

d) We need to create our target data Y. We are trying to predict if the market is going to go up or down, so we proceed in the following way.

d.1) Estimate the transaction costs for a single roundtrip (buy and sell) trade. This will include commissions and bid-ask spread. Imagine you are trading with £10.000. Let's call this number "c".

d.2) Pick a future time horizon h, for example h=5. At each point in time t,the future return will be the sum of all returns from r(t+1) to r(t+h). This will be r(t,h).
We can then classify as:

- if $r(t,h) > c$ then state="UP", coded as a vector [1 0 0]
- if $-c<r(t,h)<c$ then state="FLAT", coded as a vector [0 1 0]
- if $r(t,h) < -c$ then state = "DOWN", coded as a vector [0 0 1]

The set of vectors at each time t is our set of target data to predict Y.

e) Merge the two datasets X and Y into D. Be careful to align the data correctly by time. Our final dataset D = [X Y] has to be in the correct format to be fed into a neural network. Notice that each row is self contained, so we can scramble the order of the rows with no loss. This is also a good moment to split the dataset D randomly into training and testing sets. I recommend starting with an 80%-20% random split, but other splits are also possible.

Part B: Training and testing the neural network

You are free to design your own neural architecture, but the following characteristics must be kept.

a) The input layer is made of 84 nodes, each input node corresponding to one indicator.

b) The output layer is made of three nodes. The output of these three nodes is then fed to a softmax function to generate probabilities.

c) The cost function is the cross-entropy function.

My suggestion (not mandatory) is to start with one middle layer of 42 nodes, the sigmoid or tanh activation function and stochastic gradient descent with batch size of around 30.

d) To test the accuracy of the network at labelling returns, we don't use the cost function, but we need a binary result "correct / incorrect classification". To achieve this we do the following:

> d.1) The output of our network is a vector assigning probabilities to the three states. Our predicted state is the one with the highest probability, so a max function is all we need.
> If the outcome of the network is [0.35 0.40 0.25], our prediction is [0 1 0] ("FLAT").
>
> d.2) We then need to record accuracy. This is just a way to count how many mistakes we make in classifying. We achieve this using:
>
> 0.5*sum((predicted vector of state– actual vector of state)^2)
>
> This will give us a 1 if the prediction is a mistake, and 0 if it is accurate. We need to measure accuracy in the whole sample N, so something like (N – sum of all mistakes) / N will give us an accuracy rate in %.

e) (Optional) Improvements of neural network

If you are an expert in neural networks, this task will be pretty easy for you so far. Now you can improve on the basic version of the network. Here some suggestions:

- repeat for different time horizons h (point d.2)
- try adding a second layer
- use a different algorithm to train the network (eg. momentum)
- implement parameter L2-regularization
- implement dropout (cut nodes, cut connections, etc.)
- try different ML forecasting techinques (probit, random forests, CARTs, RNNs, LSTMs, etc.)

Part C: Evaluating results.

a) Track the performance of a strategy that buys whenever P(“UP”)>p*, and sells when P(“DOWN”)> p*. You can assume a capital of £10.000 and transaction costs of c per trade. The outcome of this stage should be a daily time series of capital, that starts at £10.000 and (hopefully) increases over time.

b) Produce summary statistics for the strategy, such as annualized return, annualized volatility, Sharpe ratio, maximum drawdown, etc.
