Reinforcement Learning for Stock trading

Rohan Saphal

August 11, 2018

1 Introduction

The academic deep learning and reinforcement learning research community has largely stayed away from the financial markets. Possible reasons are, the finance industry has a bad reputation, the problem doesn't seem interesting from a research perspective, or because data is difficult and expensive to obtain. In this brief report, I hope to argue that training Reinforcement Learning agents to trade in the financial (and cryptocurrency) markets can be an extremely interesting research problem. I believe that it has not received enough attention from the research community but has the potential to push the state-of-the art of many related fields. It is quite similar to training agents for multiplayer games such as DotA, and many of the same research problems carry over. I hope to convey on a more high level about why learning to trade using machine learning is difficult, what some of the challenges are, and where I think reinforcement learning fits in.

2 A few trading strategy metrics

When developing trading algorithms, what are they optimized for? The obvious answer is profit, but if looked at more closely, the profit is the end result or the effect of the optimization process on some metrics. Metrics are also required in order to compare trading strategy to baselines, and compare its risk and volatility to other investments. Here are a few of the most basic metrics that traders are using:

- Net PnL (Net Profit and Loss)
  - Simply how much money an algorithm makes (positive) or loses (negative) over some period of time, minus the trading fees.
- Alpha and Beta
  - Alpha defines how much better, in terms of profit, the strategy is when compared to an alternative, relatively risk-free, investment, like a government bond. Even if the strategy is profitable, it could be better off investing in a risk-free alternative. Beta is closely related, and tells how volatile the strategy is compared to the market. For example, a beta of 0.5 means that the investment moves Rs 1 when the market moves Rs 2.
- Sharpe Ratio
  - The Sharpe Ratio measures the excess return per unit of risk taken. It's basically the return on capital over the standard deviation, adjusted for risk. Thus, the higher the better. It takes into account both the volatility of the strategy, as well as an alternative risk-free investment.
- Maximum Drawdown
  - The Maximum Drawdown is the maximum difference between a local maximum and the subsequent local minimum, another measure of risk. For example, a maximum drawdown of 5 percent means that you lose 5 percent of your capital at some point. You then need to make a 10 percent return to get back to your original amount of capital. Clearly, a lower maximum drawdown is better.

- Value at Risk (VaR)
  - Value at Risk is a risk metric that quantifies how much capital may be lost over a given time frame with some probability, assuming normal market conditions. For example, a 1-day 5 percent VaR of 10 percent means that there is a 5 percent chance that you may lose more than 10 percent of an investment within a day.

3 Supervised Learning

Before looking at the problem from a Reinforcement Learning perspective, it would be helpful to understand how to create a profitable trading strategy using a supervised learning approach. Looking at the supervised learning approach will help to understand the problems with the approach and help to make clear why to use Reinforcement Learning techniques.

The most obvious approach using supervised learning is price prediction. If the prediction is that the market will move up, the reasonable action is to buy , and sell once the market has moved. Or, equivalently, if the prediction is that the market goes down, it is reasonable to go short (borrowing an asset that is not owned) and then buy once the market has moved. However, there are a few problems with this approach. First of all, what price is actually predicted? As seen above, there is not a "single" price to buy at. The final price paid depends on the volume available at different levels of the order book, and the fees needed to be paid. A naive thing to do is to predict the mid price, which is the mid-point between the best bid and best ask. That's what most researchers do. However, this is just a theoretical price, not something we can actually execute orders at, and could differ significantly from the real price we're paying. The next question is time scale. Do we predict the price of the next trade? The price at the next second? Minute? Hour? Day? Intuitively, the further in the future we want to predict, the more uncertainty there is, and the more difficult the prediction problem becomes. Let's look at an example. Let's assume the BTC (bitcoin) price is 10,000 and we can accurately predict that the "price" moves up from 10,000 to 10,050 in the next minute. So, does that mean its possible to make 50 of profit by buying and selling? The following points hope to explain why it would not be easy.

- We buy when the best ask is 10,000. Most likely we will not be able to get all our 1.0 BTC filled at that price because the order book does not have the required volume. We may be forced to buy 0.5 BTC at 10,000 and 0.5 BTC at 10,010, for an average price of 10,005. On most brokerage platforms, we also pay a 0.3 percent taker fee, which corresponds to 30 roughly

- The price is now at 10,050, as predicted. We place the sell order. Because the market moves very fast and volatile, by the time the order is delivered over the network the price has slipped already. Let's say it's now at 10,045. Similar to above, we most likely cannot sell all of the 1 BTC at that price. Perhaps we are forced to sell 0.5 BTC are 10,045 and 0.5 BTC at 10,040, for an average price of 10,042.5. Then we pay another 0.3 percent taker fee, which corresponds to roughly 30.

So, a total of -10005 - 30 - 30 + 10,042.5 = -22.5. Instead of making 50, we have lost 22.5, even though we accurately predicted a large price movement over the next minute. In the above example there were three reasons for this: No liquidity in the best order book levels, network latencies, and fees, none of which the supervised model could take into account. What is the lesson here? In order to make money from a simple price prediction strategy, we must predict relatively large price movements over longer periods of time, or be very smart about our fees and order management. And that's a very difficult prediction problem. We could have saved on the fees by using limit instead of market orders, but then we would have no guarantees about our orders being matched, and we would need to build a complex system for order management and cancellation. But there's another problem with supervised learning: It does not imply a policy. In the above example we bought because we predicted that the price moves up, and it actually moved up. Everything went according to plan. But what if the price had moved down? Would you have sold? Kept the position and waited? What if the price had moved up just a little bit and then moved down again? What if we had been uncertain about the prediction, for example 65 percent up and 35 percent down? Would you still have bought? How do you choose the threshold to place an order? Thus, you need more than just a price prediction model (unless the model is extremely accurate and

robust). We also need a rule-based policy that takes as input the price predictions and decides what to actually do: Place an order, do nothing, cancel an order, and so on. How do we come up with such a policy? How do we optimize the policy parameters and decision thresholds? The answer to this is not obvious, and many people use simple heuristics or human intuition.

3.1 A Typical Strategy Development Workflow

There are solutions to many of the above problems however, they are not very effective. Let's look a typical workflow for trading strategy development. It looks something like this:

- Data Analysis: Perform exploratory data analysis to find trading opportunities. Looking at various charts, calculating data statistics, and so on. The output of this step is an "idea" for a trading strategy that should be validated.

- Supervised Model Training: If necessary, training one or more supervised learning models to predict quantities of interest that are necessary for the strategy to work. For example, price prediction, quantity prediction, etc.

- Policy Development: Developing a rule-based policy that determines what actions to take based on the current state of the market and the outputs of supervised models. Note that this policy may also have parameters, such as decision thresholds, that need to be optimized. This optimization is done later.

- Strategy Backtesting: Use of a simulator to test an initial version of the strategy against a set of historical data. The simulator can take things such as order book liquidity, network latencies, fees, etc into account. If the strategy performs reasonably well in backtesting, the next step is to do parameter optimization.

- Parameter Optimization: It is possible now perform a search, for example a grid search, over possible values of strategy parameters like thresholds or coefficient, again using the simulator and a set of historical data. Here, overfitting to historical data is a big risk, and it is required that we be careful about using proper validation and test sets.

- Simulation and Paper Trading: Before the strategy goes live, simulation is done on new market data, in real-time. That's called paper trading and helps prevent overfitting. Only if the strategy is successful in paper trading, it is deployed in a live environment.

- Live Trading: The strategy is now running live on an exchange.

The above mentioned pipeline is a complex process but not very effective. There are a couple of reasons.

- Iteration cycles are slow. Step 1-3 are largely based on intuition, and it is not known if the strategy works until the optimization in step 4-5 is done, possibly forcing you to start from scratch. In fact, every step comes with the risk of failing and forcing you to start from scratch.

- Simulation comes too late. environmental factors such as latencies, fees, and liquidity are not explicitly taken into account until step 4.These things should directly inform the strategy development or the parameters of the model.

- Policies are developed independently from supervised models even though they interact closely. Supervised predictions are an input to the policy.It would make sense to jointly optimize them.

- Policies are simple. They are limited to what humans can come up with.

- Parameter optimization is inefficient. For example, assume the optimization process is for a combination of profit and risk, and the goal is to find parameters that give you a high Sharpe Ratio. Instead of using an efficient gradient-based approach, an inefficient grid search is being deployed and in the hope that the process will find something good (while not overfitting).

4 Deep Reinforcement Learning for Trading

The traditional Reinforcement Learning problem can be formulated as a Markov Decision Process (MDP). We have an agent acting in an environment. Each time step the agent receives as the input the current state , takes an action , and receives a reward and the next state . The agent chooses the action based on some policy . The goal is to find a policy that maximizes the cumulative reward over some finite or infinite time horizon.

Agent
The agent in this context is the trading agent. The agent can be compared to a human trader who opens the GUI of an exchange and makes trading decision based on the current state of the exchange on his or her account.

Environment
The exchange is our environment. The important thing to note is that there are many other agents, both human and algorithmic market players, trading on the same exchange. Let's assume that actions are being taken on a minutely scale (more on that below). Some action is taken, wait a minute, get a new state, take another action, and so on. When a new state is observed, it will be the response of the market environment, which includes the response of the other agents. Thus, from the perspective of our agent, these other agents are also part of the environment. They're not under our control. However, by putting other agents together into some big complex environment the ability to explicitly model them is lost. For example, one can imagine an agent to learn to reverse-engineer the algorithms and strategies that other traders are running and then learn to exploit them. Doing so would put us into a Multi-Agent Reinforcement Learning (MARL) problem setting, which is an active research area. For simplicity,its best to assume that our agent is interacting with a single complex environment that includes the behavior of all other agents.

State
In the case of trading on an exchange, it is not possible to observe the complete state of the environment. For example, it is not known about the other agents in the environment, how many there are, what their account balances are, or what their open limit orders are. Therefore it is a a Partially Observable Markov Decision Process (POMDP) that is being dealt with here. What the agent observes is not the actual state of the environment, but some derivation of that. Let that be known as the observation , which is calculated using some function of the full state . In this case, the observation at each timestep is simply the history of all exchange events (described in the data section above) received up to time . This event history can be used to build up the current exchange state. However, in order for the agent to make decisions, there are a few other things that the observation must include, such as the current account balance, and open limit orders, if any.

Time Scale
It is important to decide what time scale to act on. Days? Hours? Minutes? Seconds? Milliseconds? Nanoseconds? Variables scales? All of these require different approaches. Someone buying an asset and holding it for several days, weeks or months is often making a long-term bet based on analysis, such as "Will Bitcoin be successful?". Often, these decisions are driven by external events, news, or a fundamental understanding of the assets value or potential. Because such an analysis typically requires an understanding of how the world works, it can be difficult to automate using Machine Learning techniques. On the opposite end, we have High Frequency Trading (HFT) techniques, where decisions are based almost entirely on market microstructure signals. Decisions are made on nanosecond timescales and trading strategies use dedicated connections to exchanges and extremely fast but simple algorithms running of FPGA hardware. Another way to think about these two extremes is in term of "humanness". The former requires a big picture view and an understanding of how the world works, human intuition and high-level analysis, while the latter is all about simple, but extremely fast, pattern matching. Neural Networks are popular because, given a lot of data, they can learn more complex representations than algorithms such as Linear Regression or Naive Bayes. But Deep Neural Nets are also slow, relatively speaking. They can't make predictions on nanosecond time scales and thus cannot compete with the speed of HFT algorithms. Therefore I believe the sweet spot is somewhere in the middle of these two extremes.

It is best to act on a time scale where we can analyze data faster than a human possibly could, but where being smarter allows the algorithm to beat the "fast but simple" algorithms. My guess is that this corresponds to acting on timescales somewhere between a few milliseconds and a few minutes. Humans traders can act on these timescales as well, but not as quickly as algorithms. And they certainly cannot synthesize the same amount of information that an algorithm can in that same time period. That's the advantage we gain. Another reason to act on relatively short timescales is that patterns in the data may be more apparent. For example, because most human traders look at the exact same (limited) graphical user interfaces which have pre-defined market signals their actions are restricted to the information present in those signals, resulting in certain action patterns. Similarly, algorithms running in the market act based on certain patterns. My hope is that Deep RL algorithms can pick up those patterns and exploit them. Note that we could also act on variable time scales, based on some signal trigger. For example, we could decide to take an action whenever a large trade occurred in the market. Such as trigger-based agent would still roughly correspond to some time scale, depending on the frequency of the trigger event.

Action Space
In Reinforcement Learning,there is a distinction between discrete (finite) and continuous (infinite) action spaces. Depending on how complex the agent should be, there are a couple of choices here. The simplest approach would be to have three actions: Buy, Hold, and Sell. That works, but it limits to placing market orders and to invest a deterministic amount of money at each step. The next level of complexity would be to let the agent learn how much money to invest, for example, based on the uncertainty of our model. That would put it into a continuous action space, as we need to decide on both the (discrete) action and the (continuous) quantity. An even more complex scenario arises when we want our agent to be able to place limit orders. In that case our agent must decide the level (price) and the quantity of the order, both of which are continuous quantities. It must also be able to cancel open orders that have not yet been matched.

Reward Function
There are several possible reward functions we can pick from (referring to the metrics mentioned earlier). An obvious one would the Realized PnL (Profit and Loss). The agent receives a reward whenever it closes a position, e.g. when it sells an asset it has previously bought, or buys an asset it has previously borrowed. The net profit from that trade can be positive or negative. That's the reward signal. As the agent maximizes the total cumulative reward, it learns to trade profitably. This reward function is technically correct and could lead to the optimal policy in the limit. However, rewards are sparse because buy and sell actions are relatively rare compared to doing nothing. Hence, it requires the agent to learn without receiving frequent feedback. An alternative with more frequent feedback would be the Unrealized PnL, which the net profit the agent would get if it were to close all of its positions immediately. For example, if the price went down after the agent placed a buy order, it would receive a negative reward even though it hasn't sold yet. Because the Unrealized PnL may change at each time step, it gives the agent more frequent feedback signals. However, the direct feedback may also bias the agent towards short-term actions when used in conjunction with a decay factor. Both of these reward functions naively optimize for profit. In reality, a trader may want to minimize risk. A strategy with a slightly lower return but significantly lower volatility is preferably over a highly volatile but only slightly more profitable strategy. Using the Sharpe Ratio is one simple way to take risk into account, but there are many others. We may also want to take into account something like Maximum Drawdown, described above. One can image a wide range of complex reward function that trade-off between profit and risk.

5 Reinforcement Learning vs Traditional methods

In the following section, I hope to explain how Reinforcement Learning is better and why to use it over supervised techniques. Developing trading strategies using RL could result in a pipeline which is much simpler, and more principled than the approach we saw in the previous section.

End-to-End Optimization
In the traditional strategy development approach there are several steps before we get to the metric we actually care about. For example, to find a strategy with a maximum drawdown of 25 per-

cent, it is needed to train a supervised model, come up with a rule-based policy using the model, backtest the policy and optimize its hyper-parameters, and finally assess its performance through simulation. Reinforcement Learning allows for end-to-end optimization and maximizes (potentially delayed) rewards. By adding a term to the reward function, it is possible to directly optimize for this drawdown, without needing to go through separate stages. For example, imagine giving a large negative reward whenever a drawdown of more than 25 percent happens, forcing the agent to look for a different policy. Of course, we can combine drawdown with many other metrics. This is not only easier, but also a much more powerful model.

Learned Policies
Instead of needing to hand-code a rule-based policy, Reinforcement Learning directly learns a policy. There's no need to specify rules and thresholds such as "buy when you are more than 75 percent sure that the market will move up". That's inherent in the RL policy, which optimizes for the metric. Also because the policy can be parameterized by a complex model, such as a deep neural network, it is possible to learn policies that are more complex and powerful than any rules a human trader could possibly come up with. And as we've seen above, the policies implicitly take into account metrics such as risk, if that's something we're optimizing for.

Trained directly in Simulation Environments
It was required to have separate backtesting and parameter optimization steps because it was difficult for the strategies to take into account environmental factors, such as order book liquidity, fee structures, latencies, and others, when using a supervised approach. It is not uncommon to come up with a strategy, only to find out much later that it does not work, perhaps because the latencies are too high and the market is moving too quickly so that it is not possible to get the trades that was expected. **Since Reinforcement Learning agents are trained in a simulation, and that simulation can be as complex as desired, taking into account latencies, liquidity and fees, we possibly will not have this problem. Getting around environmental limitations is part of the optimization process. For example, if we simulate the latency in the Reinforcement Learning environment, and this results in the agent making a mistake, the agent will get a negative reward, forcing it to learn to work around the latencies.** It is possible to take this a step further and simulate the response of the other agents in the same environment, to model impact of our own orders, for example. If the agent's actions move the price in a simulation that's based on historical data, we don't know how the real market would have responded to this. Typically, simulators ignore this and assume that orders do not have market impact. However, by learning a model of the environment and performing roll-outs using techniques like a Monte Carlo Tree Search (MCTS), we could take into account potential reactions of the market (other agents). By being smart about the data we collect from the live environment, we can continuously improve our model. There exists an interesting exploration/exploitation trade-off here: Do we act optimally in the live environment to generate profits, or do we act sub optimally to gather interesting information that we can use to improve the model of our environment and other agents? That's a very powerful concept. By building an increasingly complex simulation environment that models the real world you can train very sophisticated agents that learn to take environment constraints into account.

Learning to adapt to market conditions
Intuitively, certain strategies and policies will work better in some market environments than others. For example, a strategy may work well in a bearish environment, but lose money in a bullish environment. Partly, this is due to the simplistic nature of the policy, which does not have a parameterization powerful enough to learn to adapt to changing market conditions. Because RL agents are learning powerful policies parameterized by Neural Networks, they can also learn to adapt to various market conditions by seeing them in historical data, given that they are trained over a long time horizon and have sufficient memory. This allows them to be much more robust to changing markets. In facts, we can directly optimize them to become robust to changes in market conditions, by putting appropriate penalties into the reward function.

Ability to model other agents
A unique ability of Reinforcement Learning is that we can explicitly take into account other agents. So far we've always talked about "how the market reacts", ignoring that the market is really just a

group of agents and algorithms, just like us. However, if we explicitly modeled the other agents in the environment, our agent could learn to exploit their strategies. In essence, we are reformulating the problem from "market prediction" to "agent exploitation". This is much more similar to what we are doing in multiplayer games, like DotA.

6 RL trading agent research

The goal with this brief report on Reinforcement Learning for Trading is also to convince more researchers to take a look at the problem. The following makes trading an interesting research problem.

Live Testing and Fast Iteration Cycle
When training Reinforcement Learning agents, it is often difficult or expensive to deploy them in the real world and get feedback. For example, if an agent is trained to play Starcraft 2, how would you let it play against a larger number of human players? Same for Chess, Poker, or any other game that is popular in the RL community. You would probably need to somehow enter a tournament and let the agent play there. Trading agents have characteristics very similar to those for multi-player games. It is possible to test them live by deploying the agent on an exchange through their API and immediately get real-world market feedback. If the agent does not generalize and loses money you know that you have probably over-fit to the training data. In other words, the iteration cycle can be extremely fast.

Large Multi-player Environments
The trading environment is essentially a multi-player game with thousands of agents acting simultaneously. This is an active research area and is making progress at multi-player games such as Poker, Dota2, and others, and many of the same techniques will apply here. In fact, the trading problem is a much more difficult one due to the sheer number of simultaneous agents who can leave or join the game at any time. Understanding how to build models of other agents is only one possible direction one can go into. As mentioned earlier, one could choose to perform actions in a live environment with the goal maximizing the information gain with respect to kind policies the other agents may be following.

Learning to Exploit other Agents and the Market
Closely related is the question of whether we can learn to exploit other agents acting in the environment. For example, if we knew exactly what algorithms were running in the market we can trick them into taking actions they should not take and profit from their mistakes. This also applies to human traders, who typically act based on a combination of well-known market signals, such as exponential moving averages or order book pressures. Disclaimer: The agent should not be allowed to do anything illegal. It is required that we comply with all applicable laws in our jurisdiction. Also, past performance is no guarantee of future results.

Sparse Rewards and Exploration
Trading agents typically receive sparse rewards from the market. Most of the time the agent will do nothing. Buy and sell actions typically account for a tiny fraction of all actions taken. Naively applying "reward-hungry" Reinforcement Learning algorithms will fail. This opens up the possibility for new algorithms and techniques, especially model-based ones, that can efficiently deal with sparse rewards. A similar argument can be made for exploration. Many of today's standard algorithms, such as DQN or A3C, use a very naive approach to exploration, basically adding random noise to the policy. However, in the trading case, most states in the environment are bad, and there are only a few good ones. A naive random approach to exploration will almost never stumble upon those good state-actions pairs. A new approach is necessary here.

Multi-Agent Self-Play
Similar to how self-play is applied to two-player games such as Chess or Go, one could apply self-play techniques to a multi-player environment. For example, simultaneously training a large number of competing agents, and investigate whether the resulting market dynamic somehow resembles the dynamics found in the real world. It is possible also mix the types of agents that are training, from different RL algorithms, the evolution-based ones, and deterministic ones. One

could also use the real-world market data as a supervised feedback signal to "force" the agents in the simulation to collectively behave like the real world.

Continuous Time
Markets change on micro- to milliseconds times scales, the trading domain is a good approximation of a continuous time domain. In the previous example above time period is fixed and made that decision for the agent. However, imagine making this part of the agent training. Thus, the agent would not only decide what actions to take, but also when to take an action. Again, this is an active research area useful for many other domains, including robotics.

Non stationary, Lifelong Learning, and Catastrophic Forgetting
The trading environment is inherently non stationary. Market conditions change and other agent join, leave, and constantly change their strategies. Is it possible to train agents that learn to automatically adjust to changing market conditions, without "forgetting" what they have learned before? For example, can an agent successfully transition from a bear to a bull market and then back to a bear market, without needing to be re-trained? Can an agent adjust to other agent joining and learning to exploit them automatically?

Transfer Learning and Auxiliary Tasks
Training Reinforcement Learning from scratch in complex domains can take a very long time because they not only need to learn to make good decisions, but they also need to learn the "rules of the game". There are many ways to speed up the training of Reinforcement Learning agents, including transfer learning, and using auxiliary tasks. For example, we could imagine pre-training an agent with an expert policy, or adding auxiliary tasks, such as price prediction, to the agent's training objective, to speed up the learning.

7 Conclusion

The goal of this very brief report was to make an argument for Reinforcement Learning based trading agents and why they are superior to current trading strategy development models, and make an argument for why I believe more researchers should be working on this.