Reinforcement Learning for Automated Trading

Krzysztof Wojdalski based on Pierpaolo G. Necchi's paper

11/23/2017

Reinforcement Learning for Automated Trading

- Many trading application can be viewed as sequential decision problems which allows us to apply reinforcement learning algorithms to them
- In this paper the author tried to use the framework for asset allocation (an agent has to divide his capital into a set of financial instruments)
- The goal for the research was to implement the algorithm that works as expected and potentially beat the benchmark

Reinforcement Learning - what is it?

- A general class of algorithms in the ML field
- Allows an agent to learn how to behave in a stochastic and possibly unknown environment
- The only feedback consists of a scalar (1×1 matrix) reward signal
- Based on actual experiences, not programmer's assumptions
- RL algorithms can be divided into three general categories:
  - *Model-Based*
  - *Value-Based*
  - *Policy-Based*

Example for Value-Based (Q-learning) method - Grid World

- Q-learning is the most popular and seems to be the most effective model-free algorithm for learning from delayed reinforcement.
- Directly estimates the optimal Q-values of each action in each state from which a policy may be derived by choosing the action with the highest Q-value in the current state
- It may converge quite slowly to a good policy and does not generalize very well for more complicated problems

What is the problem with standard RL algorithms? - cont'd

- They work well but only if the number of states is small enough i.e. computationally possible. For instance, simple Tic Tac Toe game (3×3 states)
- The major disadvantage of methods such as **SARSA**, **TD(0)**, **Q-learning** is that we need to estimate the value function for each state and in the case of the action value function each state and each action
- For financial markets, environment is much more complicated than in the example. Variables, e.g. volatility, are continuous and the number of states is not limited. It is not feasible (or computationally very absorbing) to calculate values for each combination of input values
- Hence, it's usually needed to use approximation methods, i.e. $\hat{f}(x, \theta) \approx V(s)$ where $x = \varphi(s)$

Policy gradient or value-based methods?

When should I use policy gradient?

- When there is a parameterized policy
- **When there is a high-dimensional state space**
- When we expect the gradient to be smooth

When should I use a value-based method?

- When there is no parameterized policy
- When we have no idea how to solve the problem
- Often π can be simpler than $Q(s,a)$ or $V(s)$
- V does not prescribe actions
- Q need to be efficiently solve $argmax_u Q_\theta(s,u)$ - it's challenging

Policy gradient methods

- Increases probability of paths with positive R
- Decrease probability of paths with negative R
- In the Q-learning algorithm we try to find value for $Q(s, a)$ and then set the policy accordingly (for each state choose such an action that value for the function is maximized)
- In policy gradient methods we improve a policy but it is never fully deterministic
  - *For instance, in the grid world our action set is* $[U, D, L, R]$*. The output for such an algorithm might look like this (for a given state):* $[0.4, 0.3, 0.2, 0.1]$

The investment process in the research

- At time t, the investor observes the state of the market S_t, consisting of the past asset prices ~~and other relevant economic variables~~
- Subsequently, he chooses how to rebalance his portfolio, by specifying the units of each stock $n_t = (n_t^0, n_t^1, \ldots, n_t^I)^T$ be held between t and $t + 1$.
- He needs to take into account the transaction costs that he has to pay to the broker to change his position.
- At time $t + 1$, the investor realizes a profit or a loss from his investment due to the stochastic variation of the stock values

Problem formulation

Investor's goal can be described in the following way:

- How to (dynamically) invest the available capital in a portfolio of different assets in order to maximize the expected total return or another relevant performance measure

$$R_{t+1} = log\{1 + \sum_{i=0}^{I}[a_t^i X_{t+1}^i - \delta_i |a_t^i - \tilde{a}_t^i| - \delta_s (a_t^i)^-] - \delta_f 1_{a_t \neq \tilde{a}_{t-1}}\} \text{ - rewards}$$

$$\{a_t^i\}_{i=0}^{I} \text{s.t.} \quad \sum_{i=0}^{I} a_t^i = 1 \qquad \forall t \in \{0, 1, 2, \dots\} \text{ - actions}$$

$$S_t = \{X, X_t, X_{t-1}, \dots, X_{t-P}, \tilde{a}_t\} \text{ - states}$$

where:

- The agent invests all his wealth at each step $\sum_{i=0}^{I} a_t^i = 1 \quad \forall t \in \{0, 1, 2, \dots\}$
- a_t^i is the fraction of wealth invested in the i-th stock between time t and $t+1$
- $a_t^i X_{t+1}^i$ - the returns multiplied by weights at time t for i-th stock
- $\tilde{a}_t^i$ is the fraction of wealth invested in the i-th stock just before the reallocation
- δ_i - the transaction cost
- δ_s - the borrowing cost for short positions
- δ_f - the fixed cost if reallocation takes place

Algorithms used

- ARAC - Average Reward Actor-Critic agent (ARAC), which includes most of the features of the other algorithms tested in this thesis
- PGPE - (policy gradient with parameter-based exploration) employs a Monte Carlo approximation of the gradient, where the action-value function is estimated using the returns on a sampled trajectory of the MDP
  - *The controller is deterministic*
  - *The actions do not need to be sampled at each time step*
  - *It is sufficient to sample the parameters once at the beginning of the episode ($t = 1$ instead of $1, \ldots, 7000$) and then generate an entire trajectory following the deterministic policy F*
- NPGPE - an enhancement of the PGPE algorithm based on the natural gradient technique

Asset used for RL methods evaluation

- Author introduced a synthetic asset whose behavior gimmicked assets that might be subject to profitable trading strategies

$$z_t = z_{t-1} + \beta_{t-1} + \kappa\epsilon$$

$$\beta_t = \alpha\beta_{t-1} + \upsilon_t$$

The synthetic price series

$$Z_t = exp(\frac{z_t}{max_t z_t - min_t z_t})$$

The model (synthetic price series) is often used as a benchmark test/sanity check

Experiment setup

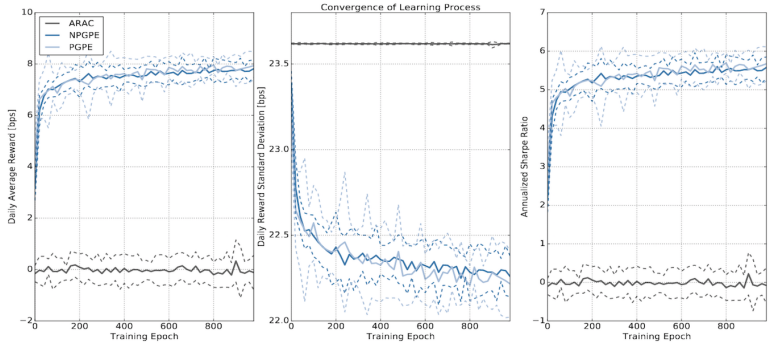
Parameters used:

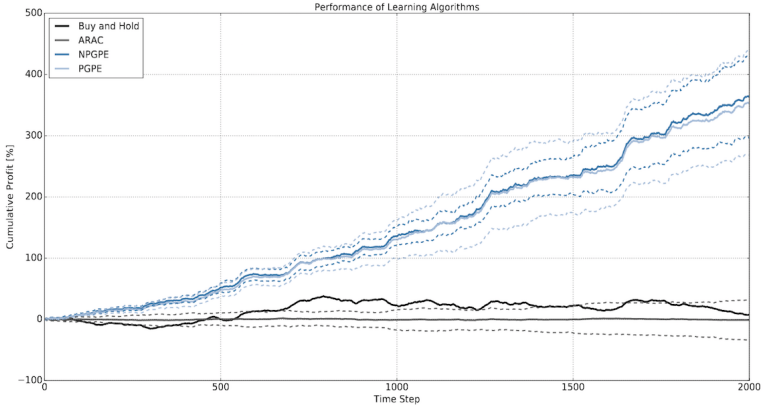
- length of $Z_t = 9000$
- $\alpha = 0.9$
- $\kappa = 3$
- Number of epochs (episodes) = 500
- Training set = the 7000 first days with decreasing learning rate α after each epoch
- Test set = 2000 final days
- Presented results are the average of 10 independent experiments that used "slightly different random initialization of the policy parameters"

Steps

- Preprocessing - creates a series of a synthetic asset and initialize the MarketEnvironment objects. For real market data, the author used `market_data_collector` which utilizes Yahoo! Finance database
- Experiment Launcher
  - *Parameters setup*
  - *Running experiment*
  - *Saving the output*
- Postprocessing - outputs figures and results produced by main_thesis, so they can be easily compared and assessed It computes the average and confidence intervals for the learning curves of the algorithms and the backtest cumulative profits of the learned strategies. Moreover, it computes some performance measures typically used in Finance to evaluate a trading strategy, such as the Sharpe ratio and the maximum drawdown

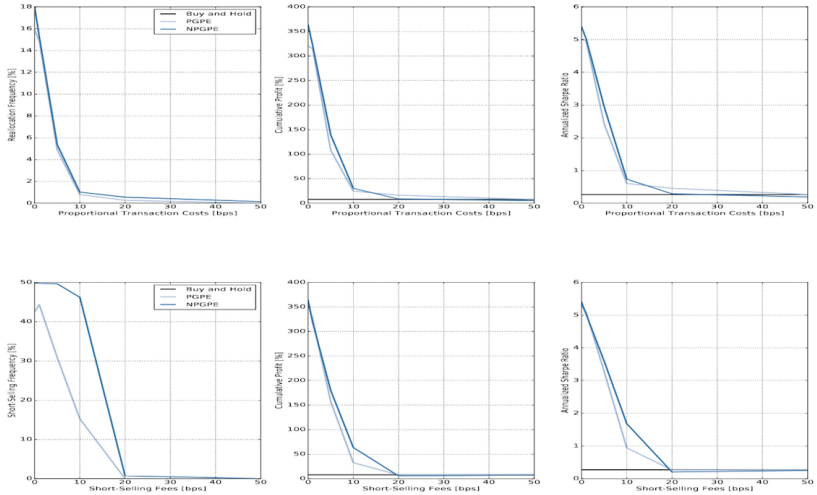
Performance - results





Impact of Transaction Costs - results

- The Author argues that most works are based on methods that simply invest according to the prediction of the future returns
- Lack of (explicit) taking into account transaction costs tends to degrade when shorting position/executing an order gets more expensive
- We can see from the figures that for RL based algorithm it is not the case as reallocation frequency adjusts for Proportional Transaction Costs [bps]



- Short-selling gets less frequent as the cost increases what complies with author's expectations

Conclusions

- For a synthetic asset, the produced strategies outperform the simple Buy & Hold strategy
- Even when investment decisions are only based on extremely basic autoregressive features
- The learning algorithm was able to adapt to the introduction of transaction costs by reducing the frequency of reallocation and of short positions
- RL techniques might effectively dealing with complex sequential decision problems that are typical in financial applications
- On historical data the used algorithm did not outperform Buy & Hold strategy

Conclusions

Possible explanations:

- Weak features - author believes that it was due that the simple features he has used were not good enough. More sophisticated set of features could possibly beat the benchmark
  - *Concretely, state space was based solely on past returns "We might also include some external variables Yt that may be relevant to the trader, such as the common technical indicator used in practice" "*
- Non-stationarity of financial time-series: a signal needs to be persistent
- Quality of data - according to the author, it's unlikely to find patterns in daily data for liquid assets
- It's generally extremely hard to find tradable patterns in financial markets

Research Directions

- Improvement of the algorithms on actual historical data
- More complex features for the trading strategy
- Combine the used technique with other techniques

My impressions

What I liked about the paper

- Design of the experiment was clear to me
- The paper is reproducible - the author's repository can be easily found on the internet
- I have the idea what is needed to be done in order to create such an agent

What I did not like

- The optimization algorithms are "black boxes" in Theano/Tensorflow libraries in Python
- The method of policy gradient is still not clear to me
- Author did not describe well results for actual data. He only mentioned his algorithms did not outperform the benchmark

Data for my thesis

- Tick database - from a vendor aggregating quotations from liquidity providers, based on Google cloud service (bigQuery)
- Reuters D3000 - market for FX dealers
- Interbank FX market based on liquidity providers (price makers) which quote bid and ask prices
- Uneven interval (high frequency dataset)
- Different tiers
- Different liquidity providers - FX market is decentralized - there is no relevant exchange. Flow goes mostly through big banks/sophisticated hedge funds
- Covers years 2015-2017, G10 currencies (pairs), e.g. EUR/USD

Data for my thesis

- Complexity of the data might lead me to simplifications
- Use only 1 tier (for 1mio which is the most frequent traded amount)
- Aggregate tick-by-tick dataset into 1 sec interval
- Use only relevant/consistent liquidity providers

Algorithm

- The final form of the algorithm implemented is yet-to-be decided
- It might be based on Policy Gradient approach which works but unfortunately I have not implemented it yet by myself
- It is very likely that it will address remaining issues outlined by the author
- I consider implementation and the optimization as the possibly biggest obstacles

References

- Richard S Sutton and Andrew G Barto. Introduction to reinforcement learning, volume 135. MIT Press Cambridge, 1998.
- David Silver. Reinforcement Learning Course on YouTube.com, Google DeepMind
- LazyProgrammer Inc. (Deep) Reinforcement Learning Course. Udemy.com