BITCOIN BOT TRADER WITH DEEP REINFORCEMENT LEARNING

Alexandre Borges
2018403420

Andre Cavalheiro
2019403065

Francisco Carvalho
2019403066

June 24, 2019

ABSTRACT

Given the complex nature of financial markets, recent reinforcement learning methods seem to be dismissed as viable ways to tackle the problem. Developments in the field where these methods outperform supervised learning methods in the same problem domain showed us that this might me an outdated idea. In this project we use new state-of-the-art deep reinforcement learning techniques to build a profitable bitcoin trading bot. For that, we created a new environment for open AI's gym toolkit, we experimented using their baseline implementations with our own set of optimization techniques and parameterization strategies, and tested the resulting agents against established benchmarks.

1 Introduction

A stock market is a stochastic, partialy observed environment. The value of taking an action depends on future actions and states, which makes it difficult to be modeled using a conventional supervised learning method. Deep Reinforcement Learning agents have shown incredible performance on environments like Dota 2, which is a complex, stochastic and partially observed game with many parallels to our problem space, albeit much larger.

In this work, we're going to use OpenAI's state of the art Deep Reinforcement Learning algorithm, Proximal Policy Optimization, to train an LSTM policy network to execute trades on the Bitcoin/US Dollar pair of currencies. In automated trading circles, it seems to be the norm to dismiss any attempt at creating trading bots using reinforcement learning, as it has traditionally been hailed as the wrong way to go about algorithmicaly automating trading. The aim of this DRL approach is to derive a policy that encodes behavioral rules and maps states to actions. Given the noticeable new advances in Reinforcement Learning, our final goal is to acess how viable RL agents trained for policy optimization can be for algorithmic trading.

2 Background

2.1 Reinforcement Learning

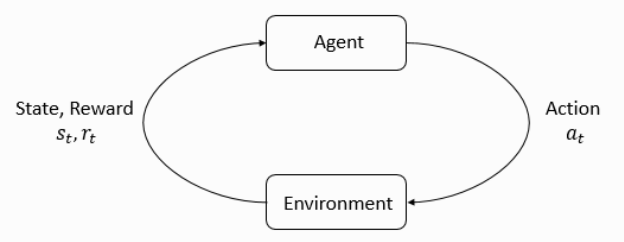

Figure 1: Agent-environment interaction loop [1]

The main concepts of Reinforcement Learning (RL) are the agent and the environment. The environment encompasses the world the agent lives in and interacts with. At each step in the environment, the agent makes an observation of the

state of the world (partial or complete), and then computes what action to take. The environment changes when the agent acts on it, but may also change on its own. The agent also observes a reward value from the environment on each timestep, communicating how 'useful' current world state is. The agent's goal is to maximize its return, the cumulative reward. Different Reinforcement learning methods are patterns through which the agent can learn behaviors to achieve its goal.

2.2 Reinforcement Learning Taxonomy

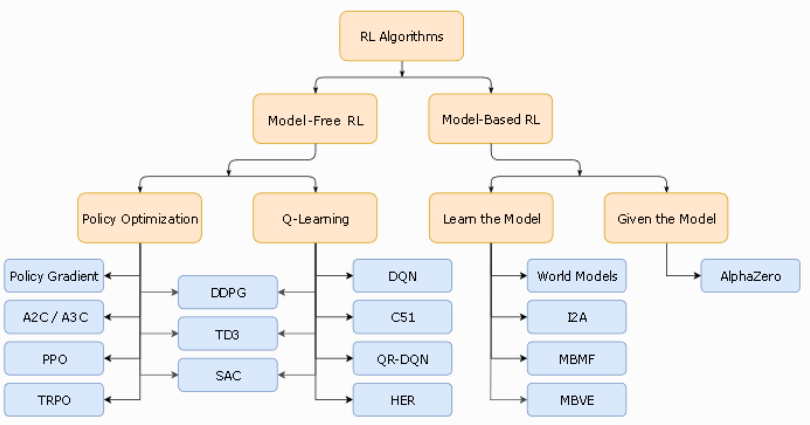

Figure 2: High level taxonomy of reinforcement learning [1]

One of the most important divisions amongst RL algorithms is whether it has access to - or learns a model of the environment. By a model of the environment, we mean a function which predicts state transitions and rewards. The advantage of having such a model is that it allows the agent to plan ahead by testing different courses of action on what would happen. The main disadvantage is that ground truth models for the environments are usually not available and learning a model present myriad challenges, most notably bias that the agent can then exploit to maximize its reward in respect to the learned model, while performing poorly in the real task. Model based learning can provide many potential gains in sample efficiency when they work, but they can often fail to pay off. While model-free methods let go of the potential advantages of a model, they tend to be easier to implement and tune. They have also been developed and tested more widely.

There are two approaches for model-free learning: Q-Learning and Policy Optimization. Policy Optimization methods make use of a policy function which estimates the best action to take given a state of the environment. In Deep RL, this policy is usually a parameterized supervised learning model. The optimization is almost always performed on-policy, which means that each update only uses data collected while acting according to the most recent version of the policy. Q-Learning methods learn an approximator for the optimal action/value function. This optimization is almost always performed off-policy, which means that each update can use data collected at any point during training, regardless of how the agent was choosing to explore the environment when the data was obtained. [citation needed for the following] Q-learning algorithms benefit from being more sample efficient when they work, because they can reuse data more effectively than policy optimization techniques. [citation needed for the following] The primary strength of policy optimization methods is that they are principled, in the sense that you directly optimize for the intended objective. [citation needed] This tends to make them stable and reliable. By contrast, Q-learning only indirectly optimizes for agent performance, by training to satisfy a self-consistency equation. [citation needed] There are many failure modes for this kind of learning, so it tends to be less stable.

3 Related Works

There have been few works that apply reinforcement learning to trading. [2] uses reinforcement learning to solve the problem of determining how to invest the available capital in a portfolio of different assets to maximize the total expected return. They use several reinforcement learning algorithms and test them with only one asset. The highest profit they obtain is of 314.34% over a certain period using PGPE (Policy Gradients with Parameter-based Exploration) compared to the profit of 7.81% of only buying and holding the asset. [3] uses deep reinforcement learning to decided when to sell, buy or hold a stock. They improve on [4] architecture by also doing financial news sentiment analysis

to predict the stock trend instead of only using the stock values and statistics. The sentiment analysis is done using RCNNS (Recurrent Convolutional Neural Networks). They use an actor critic model where both the actor and critic are deep MLPs (Multilayer Perceptrons). They were able to have higher profit than just buying and holding an asset by using a binary reward representing if the action was profitable or not. [5] uses deep q-learning to price time series input. They use several types of networks to model Q values; all agents manage to find a profitable strategy. GRU-based agents (Gated Recurrent Unit) do best in univariate games – where the only input is a wave-like price time series – and MLP-based agents do best in bivariate games - where the input includes a random stepwise price time series and a noisy signal time series, the signal is generated by moving the price ahead by a random number of time steps and then adding a noise term.

Since both [2] and [3] are policy-based algorithms and were able to achieve profit, we decided to go that route by choosing a state-of-the-art Policy Optimization algorithm. We felt like the use of sentiment analysis to predict trends is not enough compared to using time series forecasting to predict the stock trend, but with due time we might have integrated it in our model.

4 Approach

4.1 Feature Engineering

4.1.1 Stationariety

A stationary series is one in which the properties – mean, variance and covariance, do not vary with time. Making the time series stationary is critical if we want the forecasting model to work well because our data has non-stationary trends that make effective and precise predictions impossible. To make the time-seriues data stationary, we compute the difference between consecutive terms in the series. Differencing is typically performed to get rid of the varying mean. Transformations are used to stabilize the non-constant variance of a series. Common transformation methods include power transform, square root, and log transform. Here we use a log transformation for each term before differencing them. The final expression can be interpreted as the proportional change of the the values through time, given the log's algebraic properties. The formula is as follows:

$$\log(a_2) - \log(a_1) \tag{1}$$

where a_2 and a_1 are consecutive points.

To verify the stationariety property, we can use a popular statistical test called "Augmented Dickey-Fuller" test. Doing this gives us a p-value of 0.00, allowing us to reject the test's null hypothesis and confirm our time series is stationary.

4.1.2 Technical Analysis (TA)

In finance, technical analysis is an analysis methodology for forecasting the direction of prices through the study of past market data, primarily price and volume. Behavioral economics and quantitative analysis use many of the same tools of technical analysis. Many professional traders use technical analysis indicators, which are formulaically calculated based on past market data, to make decisions and manage risk when it comes to trading. We use the Python Technical Analysis library which has 58 features from the 32 implemented indicators. We chose the least correlated features, including 38 of them in our observation space. For example, one of the TAs we use is Relative Strength Index (RSI) which measures how recent prices changed to evaluate overbought or oversold conditions in the price of an asset. We also use a Simple Moving Average (SMA) which is the sum of recent closing prices divided by the number of time periods in the average calculation.

4.1.3 SARIMAX - Seasonal Auto-regressive Integrated Moving Average

For time series forecasting we decided to use SARIMAX, this model can be split into several components:

- S: Seasonal. Supports univariate time series data with a seasonal component.
- AR: Autoregression. Uses dependent relationship between an observation and some number of lagged observations.
- I: Integrated. Use of differencing of raw observations to make the time series stationary.
- MA: Moving Average. Uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.
- X: Exogenous variables. Allows the use of external variables to be considered in the model.

Each of these components are explicitly specified in the model as one or more parameters.

4.2 Model

Given what we know from the families of RL, we decided to go with a Policy Optimization algorithm, for which we have two options which differ in their value functions. A2C, which directly maximizes performance through gradient ascent. PPO, whose updates indirectly maximize performance, by maximizing a surrogate objective function which prevents the policy from changing too much, and thus causing accidental performance collapse. The intuition behind PPO is that:

- PPO is a policy optimization algorithm that follows the same intuition as the previous SOTA (TRPO): trying to maximize the size of the improvement step without stepping so far that it causes accidental performance colapse. Where TRPO uses second-order methods based on the KL-divergence between consecutive policies, PPO is significantly simpler to implement and empirically seems to perform at least as well as TRPO.
- PPO-clip doesn't have a KL-divergence term in the objective and doesn't have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

We decided to use the PPO2 algorithm from the baselines repository by OpenAI, and train it in a custom OpenAI Gym environment for bitcoin trading.

4.3 Action Space

The action space is discrete, as we can only buy, sell, or hold a given amount, and we stipulate that the agent can trade only discrete proportions of its own funds. eg: buy BTC with 1/3 of the agent's USD.

4.4 Observation Space

A state for our stock trading environment is partially observed, as we only see market data in aggregate, like prices and total volume of trading, which are the result of many different trades happening sequentially. Our agent's observation space includes the OHLC data from the dataset, data from the past trades (amounts of BTC bought/sold and their value in USD), the final set of 38 TA features, and the SARIMAX predictions of closing prices for a certain forecast window.

4.4.1 Dataset

The dataset used was the hourly price data from the Coinbase exchange for the BTC/USD pair. The data format is as follows.

Date	Trading Pair	Open	High	Low	Close	Volume BTC	Volume USD

Figure 3: Fields in the dataset

In which Open and Close correspond to the price of BTC in USD at the beginning and end of the hour. High and Low correspond to the highest and lowest price during that time. Volume means the total amount of currency that was traded in that span.

4.5 Policy

When doing policy estimation, we wanted to use information from previous states, to achieve this we used LSTM, [6] also used LSTMs with reinforcement learning achieving good results. We could also have used MLPs with a sliding window, but this wouldn't encode information from all previous states and besides that the implementation would be more complex.

VPG trains a stochastic policy in an on-policy way. This means that it explores by sampling actions according to the latest version of its stochastic policy.

4.6 Reward Functions

Reward functions are a crucial aspect of any RL algorithm and can make or break an agent's success, as they are what it is trying to optimize; no more and no less. It's where the 'art of the work' is, so to speak, because finding a metric that produces our desired outcomes often isn't trivial. A naive approach to a reward function would be to try to optimize the agent's net worth, but we will expand on this in the Experiments section.

5 Experiments

We used PPO as our main Policy Optimization algorithm and experimented with various reward functions of increasing complexity. We measured the performance of the agents through back testing on the data that was left out of the training set. This is standard practice when evaluating trading automation and is a reasonable simplification of a live trading environment, especially when trading with not-too-large amounts of money. Evaluation will be done in comparison with a few baseline strategies: HODL, Holding On for Dear Life, refers to the practice within the community to just buy the asset and never sell, while believing it will appreciate in value. RSI and SMA are simple TA strategies based on the respective indicators, that trigger actions when the indicators cross certain thresholds. A minimum of profitability (not losing money) would be considered a success. However, for the usage of this bot to be desirable from an asset management point-of-view, the bot's performance would need to beat our established baselines' performance. For these experiments, the environment was set up with an initial budget of 10,000USD and every transaction is subject to a commission of 0.0025, higher than the ratio practiced by Binance, one of the largest cryptocurrency exchanges in the world.

5.1 Hyper-parameters

To find optimal hyper-parameters for our algorithm (or at least close to optimal) to we used a method called Bayesian optimization which works as follows:

Assuming we have a certain function, we want to optimize, we describe it as a posterior distribution of functions (gaussian process). We do this by experimenting several combinations of parameters and verifying which values our function produces with these combinations. As the number of observations grows, the distribution improves, and the algorithm becomes more certain of which regions in the parameter space are worth exploring and which are not. In our case the function to maximize is obviously our reward function. We had to optimize parameters from both the environment (the forecast length and the confidence interval for the SARIMAX function) and the agent (PPO's arguments such as the learning rate, the entropy coefficient for the loss calculation or the number of minibatches to run per update just to name a few).

5.2 Reward Functions

We experimented with several reward functions.

5.2.1 Networth

Using networth has the reward function is a naive approach.

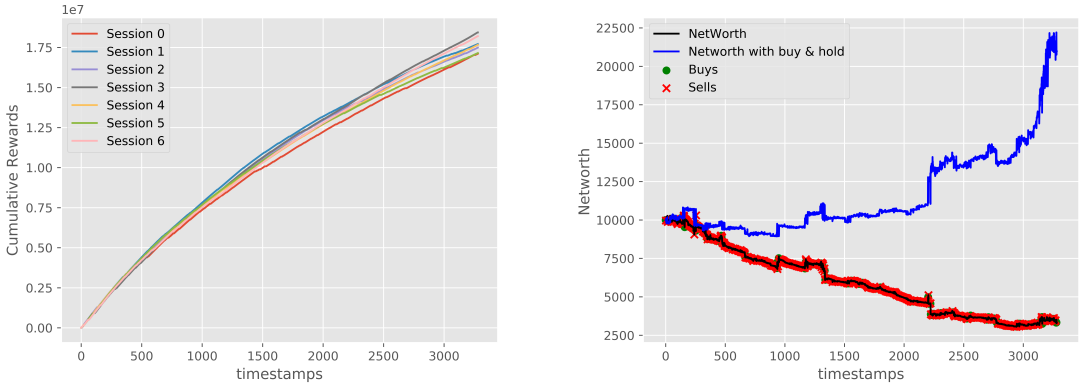

Figure 4: Networth and reward using networth as the reward function

As you can see from 5, doesn't work because it is constantly being rewarded for the net worth it already has so it reinforces random behaviour, which naturally performs badly.

5.2.2 Profit

This reward uses the profit made by the agent. This solves the problem with the networth approach which when the agent has high enough network will receive a high reward by doing nothing, by using profit we encourage the agent to try and make money.

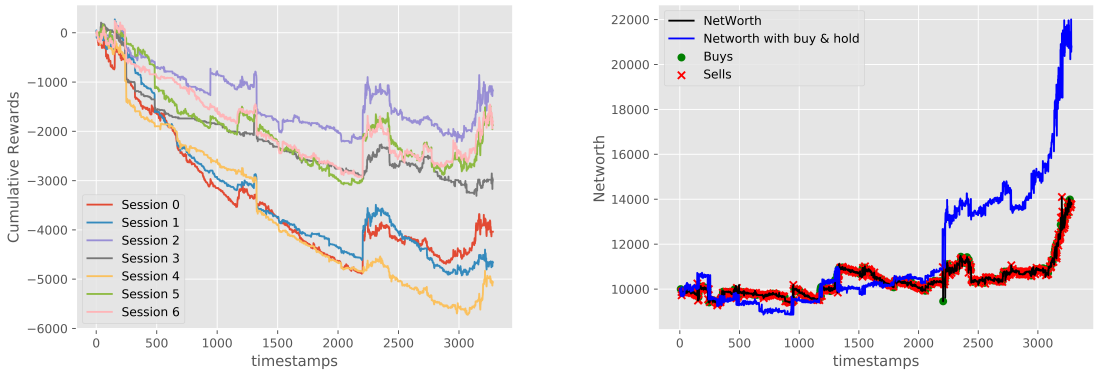

Figure 5: Networth and reward using profit as the reward function

This produces profitable results, although they are still below the benchmark of just buying and holding, so there must be room for improvement.

5.2.3 Sortino

$$S = \frac{R - T}{DR} = \frac{Portfolio \quad returns}{Downside} \tag{2}$$

The Sortino ratio is a volatility-based metric that is the ratio between the portfolio's excess returns and downside volatility over a certain period. We only consider the downside volatility instead of both the downside and upside volatility - as is it is done with the Sharpe ratio - since bitcoin upside volatility can be profitable for our model.

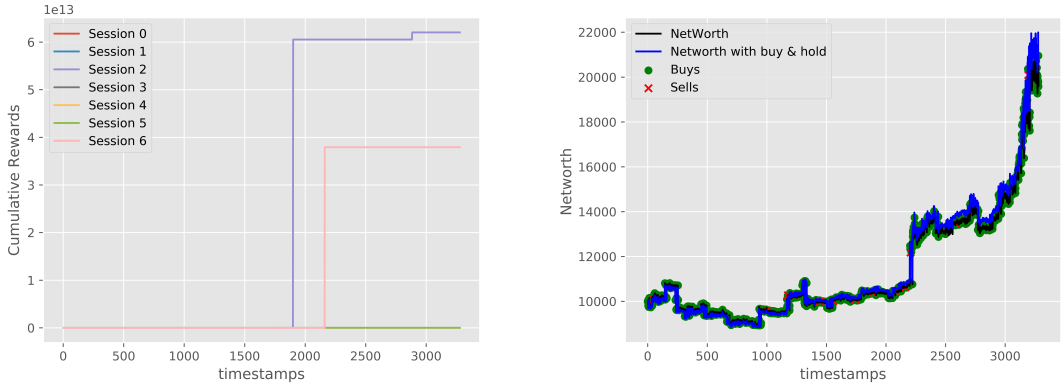

Figure 6: Networth and reward using sortino as the reward function

The best Sortino ratio agent produces a final performance very close to the holding strategy. What's more impressive, even though it is making many trades, its net worth curve seems to follow the one from the HODL strategy very closely. This doesn't always happen so neatly with every sortino agent, there are less profitable sortino agents that depart from the BTC price curve. We are still intrigued by these odd results, so we would not jump to conclusions without further investigation. Most alarmingly, one might infer from this graph that in a BTC price downtrend, the bot's performance might go down as well, so this would be the priority in future testing. It is plausible, however, that due to the Sortino ratio's property of accounting for downside volatility, it might perform reasonably.

	Profit	HODL	RSI	SMA
Sortino	104%	-3%	75%	103%
Profit	40%	-67%	11%	39%

Table 1: Profit Comparison of PPO agents with different baselines

From table 1, we can see that the best agent - the one that uses sortino as it's reward - outperforms most of the baselines and in the case of HODL, it's only outperformed by 3%.

5.3 Another Policy Optimization Algorithm: A2C

Finally, we'd like to briefly explore another PO algorithm to see how it performs when compared to PPO. A2C simply tries to optimize for the reward function, without step maximizing or regularizing like PPO. We use the best performing reward function under PPO, Sortino, and - due to time constraints - the same parameters we got from the Bayesian Optimization of the PPO-Sortino agent. The performance isn't impressive, but this might be due to the use of untuned parameters.

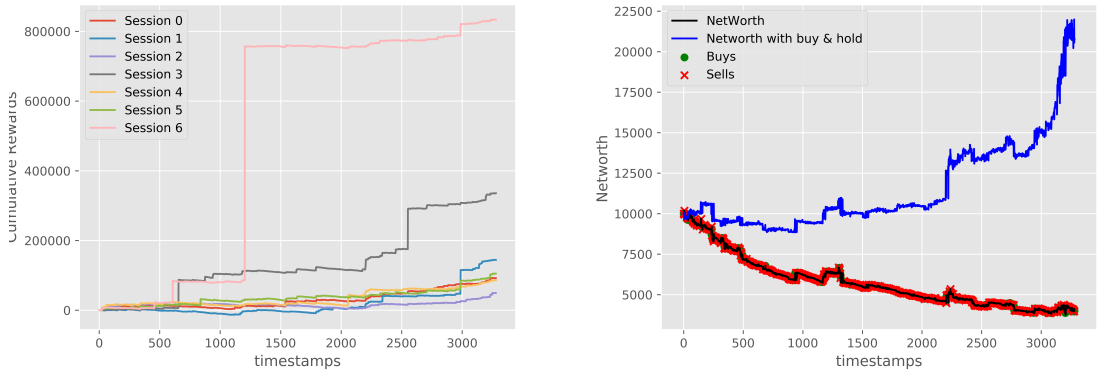

Figure 7: Networth and reward using sortino as the reward function and A2C as the policy optimization algorithm

6 Conclusion

With this report, we showed that - under back-testing conditions, our tested ratio of commission, and our initial trading budget - RL agents can trade profitably on previously unseen data. Future work might include increased training and parameter optimization times. Contextual data like the prices of other assets, or public sentiment analysis might be experimented with in order to produce even higher returns. The obvious next step would be to deploy the bot in a live setting.

References

[1] https://spinningup.openai.com

[2] Pierpaolo G. Necch. Reinforcement Learning For Automated Trading. Politecnico di Milano, IT 20123

[3] Azhikodan, Akhil and Bhat, Anvitha and Jadhav, Mamatha. Stock Trading Bot Using Deep Reinforcement Learning. 2019

[4] M A H Dempster and V Leemans. An Automated FX Trading System Using Adaptive Reinforcement Learning. 2004

[5] Xiang Gao. Deep reinforcement learning for time series : playing idealized trading games. 2018

[6] Bram Bakker Reinforcement Learning with Long Short-Term Memory. 2002