Efficient Exploration of Reinforcement Learning in Non-stationary Environments with More Complex State Dynamics

by Parker Hao

B.S. Computer Science
Massachusetts Institute of Technology, 2019

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

May, 2020

Author: _____ Date: _____
Department of Electrical Engineering and Computer Science

Certified by: _____ Date: _____
Paul F. Mende, Senior Lecturer, Thesis Supervisor

Accepted by: _____ Date: _____
Katrina LaCurts, Chair, Master of Engineering Thesis Committee

Efficient Exploration of Reinforcement Learning in Non-stationary Environments with More Complex State Dynamics

by
Parker Hao

Submitted to the Department of Electrical Engineering and Computer Science on May 8, 2020 in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Exploration technique is the key to reach optimal results via reinforcement learning in a time-efficient manner. When reinforcement learning was first proposed, exploration was implemented as randomly choosing across the action space, resulting in potentially exponential number of state-action pairs to explore from. Over the years, more efficient exploration techniques were proposed, allowing faster convergence and delivering better results across different domains of applications. With the growing interest in non-stationary environments, some of those exploration techniques are explored where the optimal state-action changes across different periods of learning process. In the past, those techniques have performed well in control setups where the targets are non-stationary and continuously moving. However, such techniques have not been extensively tested in environments involving jumps or non-continuous regime changes. This paper analyzes methods for achieving comparable exploration performance under such challenging environments and proposes new techniques for the agent to capture the regime changes of non-stationary environments as more complex states or intrinsic rewards. [1]

Thesis Supervisor: Paul F. Mende
Title: Senior Lecturer

Contents

1. Introduction

Most basic reinforcement learning agents are online, and online learning can usually deal with non-stationary problems. The update rules for state and action value estimators in some control problems are usually written for non-stationary targets, because those problems do not assume constant targets as the policy improves. A rolling recency-weighted mean of historical rewards has been effectively used to capture the dynamics with non-stationary targets as opposed to averaging in an unweighted fashion. [2]

Generally speaking, there are two types of non-stationarity: piece-wise stationarity and time-varying distribution. A piece-wise stationary time series consists of stationary segments with distinct statistical properties, such as mean, variance and autocorrelation, etc. In the case of time-varying distribution, the statistical properties of the time series gradually evolve over time and are usually dependent on the time factor. This paper mainly focuses on the first type of non-stationarity, such as discontinuity or jumps in the statistical properties of the environments.

Non-stationarity can occur across episodes. For example, each episode of a game can be initialized with a different gravity parameter. However, in real world problems instead of games, regime changes are likely to occur within an episode as well. Take trading for instance, there is no set terminal timestamp to define an episode. Therefore, one epoch is typically used as one episode in terms of reinforcement learning for trading. Under such assumptions, weighting more towards recent value updates does help during regime changes, but there is hardly any regime changes between episodes (or epochs).

The problem of "non-stationarity" can also be framed as a more complex MDP (Markov Decision Process) that allows dynamic responses to changes in the state space. Having a more complex MDP can effectively revise the reward structure based on the actions the agent has taken and offers more

flexibility on the timescale. In other words, the non-stationarity in simpler MDPs can be perceived as the additional states in constructing more complex MDPs.

As the complexity of MDPs increases, it is much harder to find the optimal state-action and therefore, efficient exploration is pertinent to achieve faster convergence for each stationary segments, or asymptotic convergence of the piece-wise stationary environment. (Such asymptotic convergence behavior is broadly referred to as convergence in the following sections) Moreover, the probability of exploiting current optima across an episode might not be monotonically increasing. In a non-stationary environment, the optima could be different across sections of regimes (or across periods within an episode). Thus, exploiting existing optima more does not guarantee better rewards. The following work analyzes the problem of efficient exploration, seeks methods for faster convergence, and designs mechanisms for exploring new optima during regime changes.[3]

2. Related Work

Exploration in reinforcement learning can be generally classified as undirected or directed, depending on whether the exploration process memorizes or utilizes the learning knowledge from the past.[4]

2.1. Undirected Exploration

- **Random exploration:** Often applied for cases where exploration does not cost anything and no prior information is known regarding the optima state-action pairs.

- **Utility-driven probabilistic exploration:** Instead of assuming fixed uniform distribution for action selection, the model chooses action α with a probability proportional to action α's expected utility.

2.2. Directed Exploration

- **Counter-based exploration:** Evaluates a state using how many times it has happened during the previous learning process. Such technique can prevent exponential (or brute force) exploration in many cases. However, the most often observed state might not yield the most rewards or utility. [5]

- **Counter-based with decay heuristics:** Assumes that occurrences of states in the beginning of a learning process should be weighted less than those happened later. In other words, occurrences of states are weighted by a time-dependent λ as a decay variable.
- **Counter-based with error estimates:** As the learning process continues, the utility estimates of different states are gradually updated. This approach prefers exploring actions leading to states whose utility values have recently changed the most.
- **Counter-based exploration with selective attention:** Uses an attention parameter that synthesizes behaviors under exploration and exploitation. The attention parameter is updated each step and stays within a range (mostly commonly between 0.1 and 0.9) such that it does not allow full exploration or full exploitation behavior.
- **Recency-based exploration:** The intuition of recency-based exploration is to explore nearby states that did not occur very often during the learning process.

2.3. Confidence Interval-based Exploration

These exploration methods use the confidence interval on the value estimators for each action, and pick the highest interval-upper-bound action. [6]

2.4. Comparison of Exploration Techniques

Table 1 shows the convergence times (quantified by number of steps) in a robot navigation environment that is controlled across all the techniques. The goal of the agent is to navigate a robot to a predetermined goal position with the least number of actions. We can see that overall the counter-based exploration techniques perform better in both deterministic and stochastic settings. Note that the data comes from previous benchmark work on efficient exploration in reinforcement learning. [4]

Yet counter-based exploration techniques have certain limitations as well. First, the above environment did not account for non-stationarity. Secondly, when the state space has a large number of dimensions, such count-based methods cannot be applied because most states will only happen once. [7]

Exploration technique	Deterministic steps	Stochastic steps
Uniform distribution	43000	43000
Boltzmann distribution	43000	43000
Semi-uniform distribution	43000	43000
Counter-based exploration	4700	4800
Counter/error	4700	4800
Counter w/ selective attention	4700	4800
Counter w/ decay ($\lambda = 0.99999$)	5800	7300
Recency-based exploration	7400	8900

Table 1: Comparison of exploration techniques in a controlled robot navigation environment [4]

3. Baseline Model

3.1. Synthetic Trading Environment

The baseline model uses a synthetic time series composed of periods of Ornstein-Uhlenbeck process and monotonically trending process. The length of periods is normally distributed and simulates the timespan of regimes in real-world application. In addition, the frequency of the mean reverting process is randomly drawn from a set of discrete numbers.

Such a synthetic time series can be considered as a price trend of financial products such as stocks. The types of regimes above are commonly referred to as momentum and mean reverting trends in the investment realm. Figure 1 demonstrates a sample time series.

The agent is placed into a simulated trading environment, with an all-cash portfolio to start with. The state space contains the current price of the stock and simple features of historical prices, such as rolling average and rolling variance. At each step, the agent receives the new price information and chooses to buy (if cash is available), hold, or sell (if the portfolio contains shares of the stock) as an action. Note that the trade activity is either zero or one, meaning that when the agent trades, it either spends all the cash or sells all shares available. In addition, this is an un-leveraged trading environment, meaning that the agent cannot trade more than what it owns. The objective is to maximize the portfolio value of the agent, calculated as $AUM_t = Cash + N_{shares} \times Price_t$. Reward is computed as the percentage

change of portfolio value AUM_{t+1}/AUM_t after each action taken at time t.

In addition to the financial returns of the portfolio, there is also the concept of risk, which contains market risk, operational risk, legal risk, etc. In this simulated trading environment, risks are not considered except for the market risk, which translates into the standard deviation of portfolio values over time, also known as portfolio volatility.

The efficient market hypothesis states that asset prices reflect all available information. In other words, no profitable patterns or strategies would be discoverable by the agent in an efficient market environment, which is our null hypothesis. The alternative hypothesis is that in a simulated trading environment with clear mean reverting and momentum trends, the agent can make profits by discovering such patterns and making the appropriate trades.

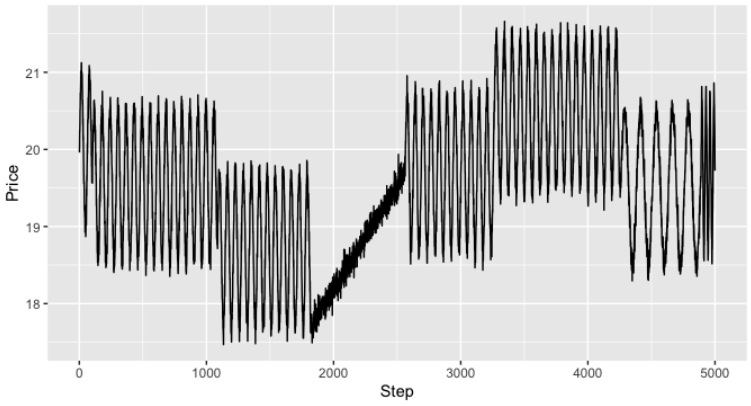

Figure 1: Sample price trend

3.2. Actor Critic

As for the agent, the baseline model implements the actor-critic (A3C) architecture [8], one of the state-of-the-art reinforcement learning models. The actor dictates the policy structure, which the agent relies on to select actions. The critic estimates the value functions, and criticizes the actions made by the actor by judging the new states after actions are taken. TD error, defined as the error function that reports back the difference between the estimated reward at any given state or time step and the actual reward received, drives the learning and feeds the action evaluation of the critic back to action selection. One the other hand, the critic, which maintains an

expected future reward given a state and an action, gradually updates itself with a learning rate as well. Therefore, the actor is not only considering the immediate rewards at each step, but also influenced by the long-term expectations of state-action pairs.

The bridge between action selection and evaluation is naturally suited for new exploration techniques. In a traditional learning setting, bad outcomes will directly discourage the agent from selecting such actions under similar scenarios in the future, whereas good actions will be exploited further. Such intuition is correct, but not necessarily efficient during regime changes, or non-stationary jumps of environment's parameters, such as the gravity in games and the frequency of mean reverting financial time series. Via the critic's evaluations, the agent can potentially gather information about the occurrence of regime switch and the optimal states under the new regime. The actor's behavior can be then adjusted to allow for faster convergence in new regimes. Therfore, A3C provides an ideal platform to develop new exploration techniques.

3.3. Transaction Cost

Trade cost is a crucial parameter to be defined properly. Intuitively, when the agent decides to either buy or sell, its portfolio value is penalized with a trade cost (TC), $AUM_t = Cash + N_{shares} \times Price_t - |\Delta_{N_{shares}}| \times Price_t \times TC$. The basis for a proper transaction cost is that the agent should make profits not by exploiting the noise term but by learning the underlying synthetic patterns.

Ideally, for each cycle in a mean reversion price trend, the agent performs a pair of optimal buy and sell trades at the amplitudes. Therefore, if the transaction cost is fair, the optimal amount of trades should roughly equal to twice the number of mean reverting cycles. With a 0.1 standard deviation white noise, 150 bps of transaction cost and $1/20\pi$ frequency (or 64 cycles over 4,000 steps), the agent makes between 100 and 150 upon convergence, which corresponds to twice the number of cycles in the environment. Therefore, 150 bps of transaction cost is set as default because it discourages the agent from exploiting the noise term while allowing profitability on the underlying synthetic trends.

3.4. Convergence Patterns

In the following example regime change, the A3C agent learned to act optimally at approximately the 140th episode with a fixed learning rate of 0.05.

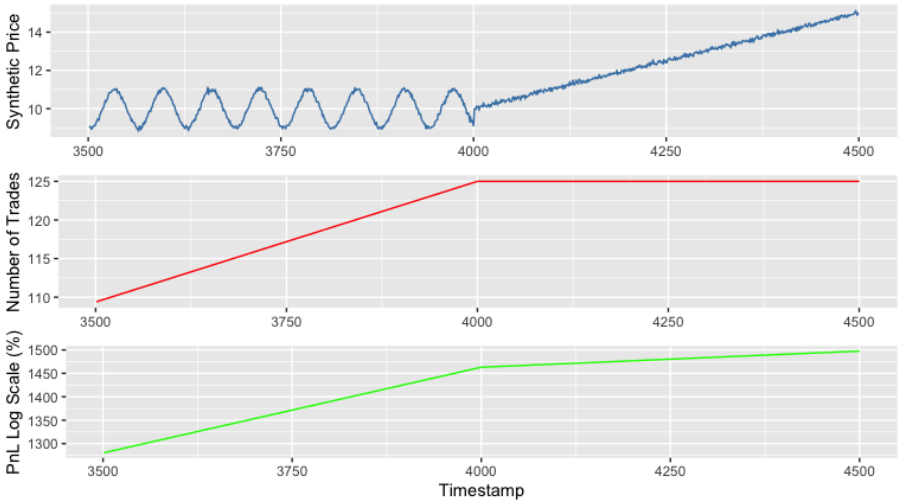

Figure 2: Number of trades and profits upon convergence of a simple price trend

However, one observation is that during the short period after the first 100 episodes, sometimes the agent decided to stay fully in cash during the momentum section, performing sub-optimally. A good portion of the training time is spent on exploring the new regime and finding out the global optimum. Therefore, efficient exploration of new regimes is essential, given that it reduces both the cost of time and the price of unnecessary exploration due to transaction costs.

Figure 2 shows the time series of capital gains and trades executed upon convergence under a simple price trend. The total number of trades matches with twice of the number of cycles and the agent learned not to execute any trades during the "momentum" period. The agent took advantage of the momentum trend by not staying in cash, with zero trades to avoid the trade cost.

Figure 3 shows the performance of the agent across different episodes. An increase of action in the graph means a buy activity and a decrease of action means a sell activity, while being flat means the hold action as defined in the baseline environment. When the regime changed during earlier episodes, the agent focused on exploration and aggressively traded to seek the new optimal action. As training goes on, the agent gradually realized that not trading anything and keeping a stock position would be the optimum for the last 1,000 steps. Meanwhile, the trades executed in the first 4,000 steps

have become much more consistent and converged to the right frequency of the underlying sinusoidal trend as training goes on.

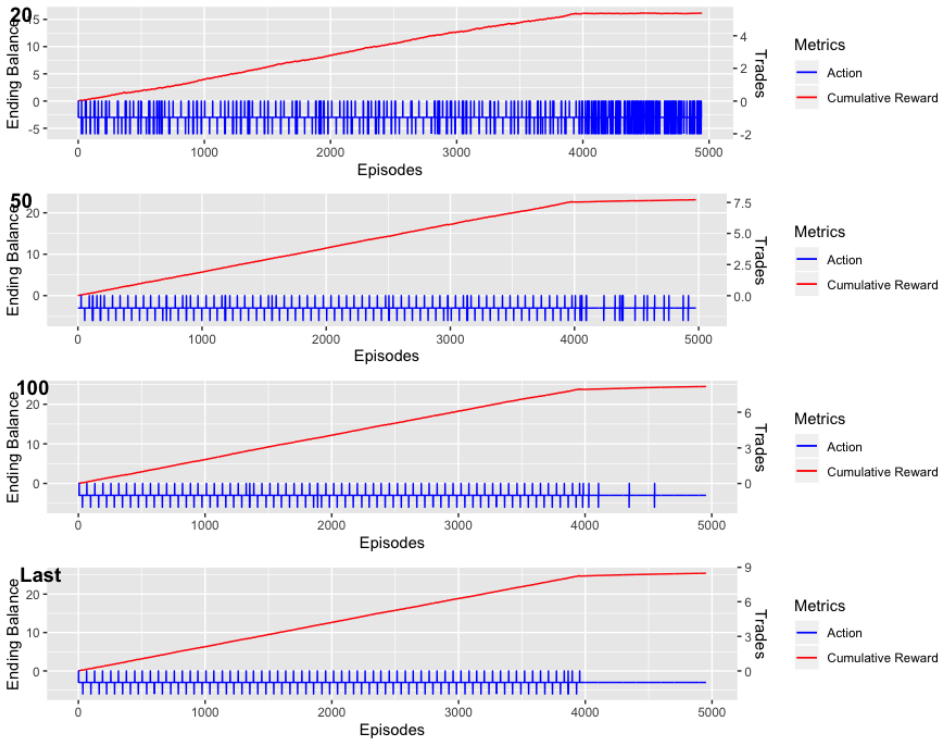

Figure 3: Action sequence of the agent gradually converges to the optimum across episodes

Note that the cumulative reward in this case is different from the capital gains shown in figure 2. The cumulative reward is the summation of each step's reward, whereas the capital gain measures the up-to-date balance of the portfolio with compounding.

It is important that for each episode, the length of the regime is not definitive. The regime change can take place anywhere within 500 steps from the initial data, sampled randomly across episodes. Therefore, the randomness can prevent the agent from using timescale information. If the agent is relying on technical indicators and hidden layer's information, its trading behavior is expected to adjust similarly across datasets with different offsets. If timescale information is heavily relied upon, the agent will not be able to trade optimally across episodes and the capital growth will not be

consistent.

4. Methods

4.1. Detection via Value Estimation

In order for the agent to quickly learn new regimes, it has to identify them first. The value estimate of the critic is a straightforward way to view the preference of states and actions. Therefore, when the regime changes, the value estimates evolve distinctively as well.

The technique first keeps a vector of the value estimates in the recent past and then calculates the variance of the vector, which is assumed to increase upon significant uncertainties of the value estimators. A threshold trigger is placed on the rolling variance. When uncertainties rise and the threshold is triggered, the training has likely started to diverge within the episode, which signals a regime switch.

There is a fine trade-off for determining the length of the rolling window or the amount of history that we can reliably view as the recent past. If the window is too short, training will be affected by the occasionally incorrect gradients from noisy data. Yet if the window is too long, the detection mechanism would face delays in picking up regime switches, and therefore hinder the original objective of quickly detection new regimes and allowing for faster convergence.

Given that the proposed methods should work across different domains of application, the detection mechanism needs to be relatively independent of the specific datasets and dependent on agent's own behavioral changes.

Therefore, additional learning is built-in for such purpose. The window size gradually adjusts itself between episodes through historical detections. After an episode, while the gradients are back-propagated, if the new regime's value estimation variance remains high, the detection mechanism's window size increases, and vice versa. Intuitively, a persistent period of high variance among value estimation variance suggests that the agent is going through too many incorrect gradients that causes the short-term reward table to jump up and down. If the window size is too long, then the detection mechanism hardly triggers itself, which will manifest as steady variance of the value estimators.

4.2. Detection via Future Reward Boundary

The second detection technique, instead of using uncertainty metrics, relies on an assumption that the regime remains unchanged throughout. The intuition is to make predictions on the short-term rewards in the near future under the same regime as the past. If the actual future rewards are significantly different than our range of predictions in the past, the agent will then recognize the new data as a different regime, and incentivize exploration in a timely manner to promote convergence.

This technique is implemented by fitting a low-order polynomial on historical rewards in a recency-weighted fashion. Then the coefficients and standard errors of the regression are used to predict lower bounds and upper bounds for the rewards in the near future.

There are a few key parameters.

- **Order of Polynomial Fit:** an order of two is generally sufficient to describe most convergence patterns under the constant regime assumption. A larger order would make the agent prone to temporary uncertainties and cause the predictions to carry stronger bias. Yet a linear trend does not capture the convexity of convergence under the assumption of a usually fixed learning rate.
- **Look-back Window Size:** here we empirically define one tenth of an episode as our look-back window size. (For example, if an episode contains 5000 timestamps, then the look-back window is 500 timestamps.) Such metric might be hard to generalize across different scenarios. A short window could drive over-fitting whereas a longer window only applies to highly stationary environments.
- **Conviction Window Size:** the near future predictions are defined with a fixed time window, as the "conviction" window. During this window, the boundaries are calculated and the distances between realized future rewards and the boundaries are referred to as residuals. The residuals are recency weighted as well.

4.3. Updating Exploration Behavior

When the regime changes are detected in the environment, the agent needs to adjust its trade-off between exploitation and exploration to quickly learn the new regime. In the setting of continuous non-stationary targets, existing exploration techniques can adapt easily with certain recency weightings so

that intuitively, the agent is "chasing" after the dynamic targets. However, when the environment is going through discrete changes or jumps in optimal value functions, exploration should be immediately more aggressive rather than conservative. [9]

- In counter-based exploration, which is a benchmark technique, the counters for each action go through a "smoothing" process such that the counters values tend to be more uniform across actions and states, allowing for a partial transition from pure exploitation into unbiased exploration.

- When the new regime is significantly different from the past, or even opposite to the past, extreme measures and adjustments need to be taken. The approach is to reset the value functions such that the new regime can be effectively considered as a new episode. In scenarios where the old policy completely fails, slightly negating the value functions can yield above-par results as well if the downside is not too costly.

- Not every regime change is determined to be totally new. The set of possible regimes in most applications tend to be finite. In addition, the regimes within such a finite set might carry similarities. Therefore, it might not be necessary for the agent to fully re-train itself to learn new regimes all the time. To make the agent more efficient for training similar regime switches in the future, memory structures can be utilized and the regimes can be compared to each other based on patterns of state-action values or intrinsic rewards. As part of the future implementation, the agent can keep a mapping from regimes to snapshots of their average state-action values. The state-action pairs from new regimes can be then compared with the previous snapshots. If similar regimes are found, the agent can "smooth" its exploration counters and reset its action probabilities towards the snapshots of the similar regimes that are already trained. The intrinsic reward values [10] can also be used in substitute of the state-action pairs, given that one prospect of the intrinsic reward mechanism is to build a hierarchy of reusable skills.

5. Results

5.1. Purely Synthetic Data

5.1.1. Detection Mechanisms

Under the baseline's synthetic data assumptions, the detection mechanism was triggered while the agent was struggling under the new regimes in most of the trials. Figure 4 illustrates the distance between rewards and our expected boundaries at different episodes. When there is only one regime switch in our synthetic data, a strong residual is observed at the regime change timestamp (roughly at the 4,000th step) after approximately 20 episodes of training. The regime switch detection was quickly triggered near the 30th episode and the agent adapted its exploration versus exploitation behavior.

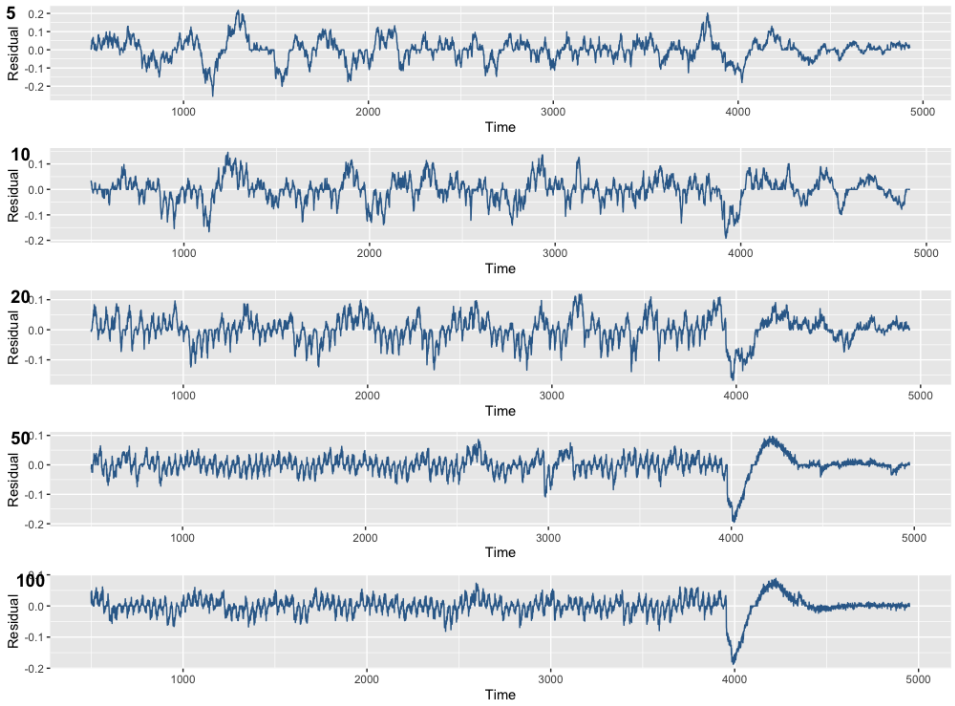

Figure 4: Residuals outside predicted boundaries across episodes

5.1.2. Convergence Times

The above baseline model makes simple assumptions about only one regime change with sufficient data for both regimes. However, most real-world applications have much more complex environments. We then investigate the

Regime Switch	Convergence	Detection	Efficient Exploration
1	132	30	106
2	344	70	279
3	785	110	543

Table 2: Number of episodes needed for baseline convergence, successful detection of regimes, and convergence with efficient exploration

convergence times as the non-stationarity of the purely synthetic environment rises under the same assumptions of two general types of regimes: mean reversion and momentum.

Table 2 indicates the number of episodes that the agent spent to converge under the baseline, to successfully detect new regimes and to converge under the efficient exploration mechanisms upon observed regime switch. An increasing number of regime switches represents a higher degree of non-stationarity in the environment. The experiment was run at least 3 times for each row in Table 2 and the average number of episodes was computed.

5.2. Hybrid Synthetic Data

In addition to the purely synthetic environment, a hybrid approach is also implemented, where the real-world price trends are combined with the above synthetic assumptions. For the real-world piece, the exchange rate ratio between USD/CNH (offshore Chinese Yuan) and USD/CNY (onshore Chinese Yuan) is used, given its two patterns: convergence guarantee towards 1 in the long run and distinct periods of sustained volatility. Figure 5 shows a sample historical exchange rate ratio, where the regimes of high and low volatility are clearly divided, both centered at 1 in the long run.
The hybrid approach uses $Noise = (CNY/CNH)^{Power}$ as the noise multiplier for the hybrid time series. The two patterns above suggests that the noise term has an average of one in the long run and carries segments of distinct variance levels. The noise sequence is repeated to match the length of the synthetic data. The hybrid prices are then calculated as $P_{Hybrid,t} = P_{Synthesized,t} \times Noise_t$. As the power on the noise term increases, the hybrid prices will be more affected by the noise term and tend to look slightly more like a random walk. In other words, the hybrid prices are more dominated by the real-world (CNY/CNH) exchange rate ratio and behaves less synthetically. If the power of the noise term is small, then the hybrid

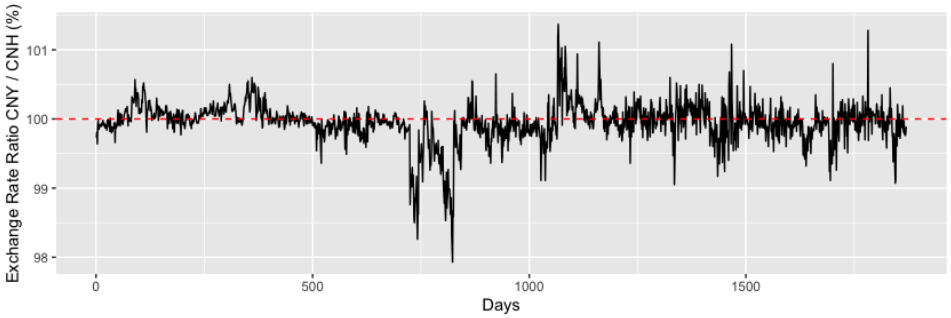

Figure 5: Exchange rate ratio

prices will be more dominated by the synthetic component $P_{Synthesized,t}$.

5.2.1. Synthetic-Dominated Data

Figure 6 shows a price trend dominated by the synthetic data (with power = 3). Given that the regime changes are more frequent and that the noise term resembles a random walk, the agent had a difficult time making gains in most of the iterations as seen in Table 3. However, compared with the baseline model where no detection mechanism is used, the performance is slightly more stable and the trading portfolio did not suffer as much loss on average.

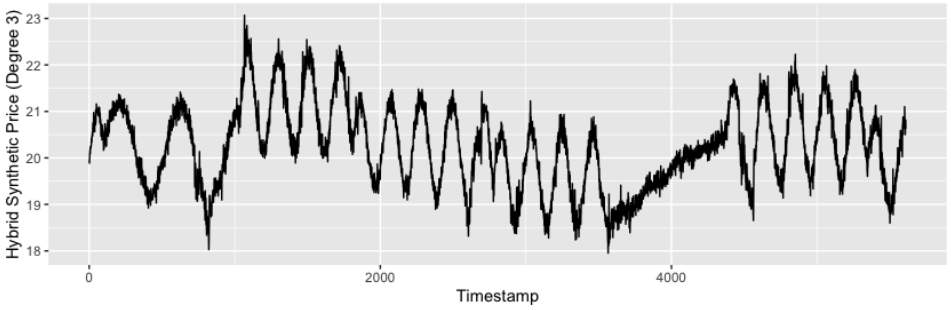

Figure 6: Synthetic-dominated price trend

5.2.2. Real-World-Dominated Data

Figure 7 shows the price trend with a power of 6. In this real-world-dominated environment, both the baseline and the proposed techniques have failed to converge and ended up with insignificant portfolio values at the end of training. Therefore, under a much higher influence of the real-world noise term, the proposed techniques did not perform well or improve the baseline model.

Model	5 Percentile PnL	Median PnL	95 Percentile PnL
Baseline	-9%	-6%	-1%
Detection Enabled	-43%	-20%	2%

Table 3: Portfolio performance on synthetic-dominated data with baseline or efficient exploration techniques

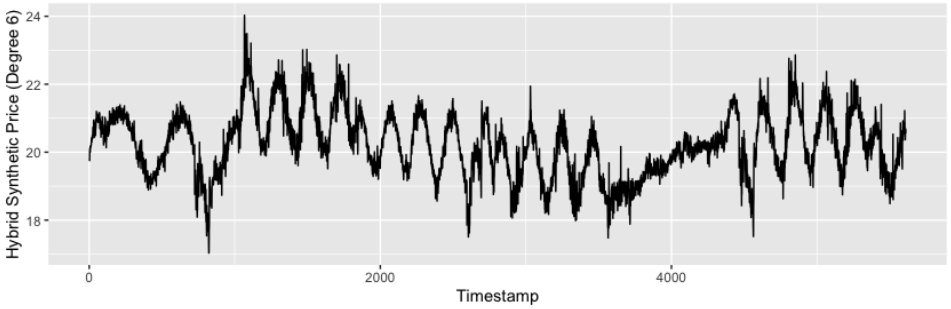

Figure 7: Real-world-dominated price trend

6. Analysis

6.1. Successful Detection for Synthesized Data

Both the value estimation and the future reward boundary approaches have shown adequate detection capabilities early on in the training process, in purely synthetic environments. As Table 2 suggests, the agent detected the new regimes in advance, about 30-40 percent into the training process (with asymptotic convergence defined as the endpoint), before the agent sufficiently learned about the optima under the new regimes.

6.2. Faster Convergence for Synthesized Data

In the purely synthetic scenarios, the regime detection techniques and exploration update technique have allowed for a 20-30% faster convergence in Table 2. As the environment's complexity grows and the number of regimes increases, the convergence time of the baseline from Table 2 is almost polynomially larger, whereas the time it takes for the agent to detect new regimes seems to increase linearly. As the number of regimes increases, the effect of the detection and exploration techniques on reducing convergence time is more significant. In other words, the proposed techniques perform better relative to the baseline as the piece-wise synthetic environment becomes more

complex. However, the proposed techniques did not demonstrate significant advantage over the baseline in the hybrid synthetic environments.

6.3. Limitations

From the hybrid synthetic environment results, certain limitations of the new techniques are observed. First, although the value estimation detection process does not require many hyper-parameters, both the future reward boundary detection and updating exploration methods require certain settings, which are specific to the environment. Therefore, in order to detect and explore the new regimes efficiently, the hyper-parameters of the techniques need to set properly in accordance with the environment. Secondly, as the environment evolves and turns into a real-world scenario, the proposed techniques seem to struggle more due to the increasingly complex regime dynamics. It is harder for the agent to differentiate new state dynamics from occasional noisy gradients. The line between over-fitting with more regime transitions and under-fitting with more essential regime properties becomes blurry as well.

7. Future Work

Efficient exploration is one of the most popular topics in reinforcement learning and people have been inventing new techniques to cope with dynamic environments. It is hard for an agent to efficiently explore non-stationary environments, particularly piece-wise stationary environments. This paper proposes a new set of techniques, by differentiating such jumps within episodes and fine-tuning exploration probabilities to learn the new regimes or new segments of a piece-wise stationary environment. Performance results are demonstrated though synthetic data that carries certain real-world assumptions regarding regime switching in financial markets. In the future, memory-based structures could potentially analyze and save more complex state dynamics as we step into real-world scenarios. The simultaneous off-policy learning approach taken by the future reward boundary detection method can be expanded to the complete set of state-action pairs, which provide far more cost-free insights as the agent learns about the new regimes.

References

[1] *D. Pathak, P. Agrawal, A. A. Efros, T. Darrell, Curiosity-driven exploration by self-supervised prediction, CoRR abs/1705.05363. arXiv:1705.05363.*
URL `http://arxiv.org/abs/1705.05363`

[2] *I. Arel, Reinforcement learning in artificial intelligence: Evaluative feedback (exploration vs. exploitation) (Fall 2016) 16.*
URL `http://web.eecs.utk.edu/ ielhanan/courses/ECE-517/notes/lecture2.pdf`

[3] *B. C. Stadie, S. Levine, P. Abbeel, Incentivizing exploration in reinforcement learning with deep predictive models, CoRR abs/1507.00814. arXiv:1507.00814.*
URL `http://arxiv.org/abs/1507.00814`

[4] *S. B. Thrun, Efficient exploration in reinforcement learning, Carnegie Mellon Universitydoi:EER:865072.*

[5] *M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, R. Munos, Unifying count-based exploration and intrinsic motivation, CoRR abs/1606.01868. arXiv:1606.01868.*
URL `http://arxiv.org/abs/1606.01868`

[6] *A. K. McCallum, Efficient exploration inreinforcement learning with hidden state.*

[7] *H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. D. Turck, P. Abbeel, exploration: A study of count-based exploration for deep reinforcement learning, in: NIPS, 2017.*

[8] *V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning.*
URL `https://arxiv.org/abs/1602.01783`

[9] *S. B. Thrun, K. Möller, Active exploration in dynamic environments (1992) 531–538.*

[10] *O. Şimşek, A. G. Barto, An intrinsic reward mechanism for efficient exploration (2006) 833–840doi:10.1145/1143844.1143949.*
URL `http://doi.acm.org/10.1145/1143844.1143949`

Appendix A. System Overview

The model is implemented via PyTorch. Most of the code (synthesizing data, generating state space, A3C, environments and background assisting scripts) are written in Python, whereas the analytical and plotting scripts are written in R.

Appendix A.1. Network Architecture

The network architecture consists of two simple hidden layers (each 128 nodes, with ReLU activation function) that feeds into both the actor's layer and the critic's layer. To achieve controlled experiments, the network specification is kept constant across the baseline and the proposed techniques.

Appendix A.2. Training

A maximum of 500 training episodes is put in place to avoid redundant training that does not converge (or asymptotically converge). Adam optimizer is used in implementation. Policy losses, value losses and the true values (using rewards from the environment) are stored and computed as $float32$.

Appendix A.3. Logging

Logs of general training performance are saved every 100 steps for each episode to reduce file size. For every tenth episode, detailed policy loss and value loss are logged at each timestamp. The file format of logs is csv.

Appendix A.4. Features

For each price trend, whether synthesized or hybrid, both rolling average and rolling standard deviation are computed with a set of time windows as the features (2, 4, 8, 16, 32, 64 days). The computed features are always aligned on the right to avoid using any future price information. Days with incomplete or NA features are deleted.

Appendix A.5. Smoothing

Given that the action probabilities are always non-negative, smoothing is implemented in a square-root fashion to avoid significantly large or small outputs. Take a simple example, smoothing(A, B) = sqrt(A) / (sqrt(A) + sqrt(B)).