Selective Experience Replay for Lifelong Learning in Trading

Melisa Civelekoglu, Antoine Dangeard, Martin Nguyen
Group 55
McGill University
Montreal, QC

Abstract

We explore whether employing a long-term memory buffer in stock trading tasks helps agents learn better strategies. Our approach involves implementing Selective Experience Replay, using four prioritization strategies: favoring surprise, favoring reward, matching the global training distribution, and maximizing coverage of the state space. Furthermore, we explore a new prioritization strategy using change-point detection on financial time series.

1 Introduction

Deep reinforcement learning (DRL) has recently emerged as a crucial tool in learning financial markets and trading strategies [9], however, deep nets face the risk of catastrophic forgetting when learning multiple tasks in sequence. Although trading in a market may not necessarily involve sequential tasks in the traditional sense, market conditions can shift from stable to volatile periods, resulting in different data distributions, forcing agents to change strategies. Furthermore, the agent's experience in one market may provide relevant information on strategies for others as they can leverage similar risk management principles.

In trading, traditional experience replay, such as a first-in-first-out (FIFO) buffer, may degrade in performance as new experiences are gained, either due to catastrophic forgetting of strategies from relevant historical market conditions or from preserving outdated data in the buffer from irrelevant market trends, resulting in trading strategies that fail to adapt to new markets. Selective experience replay (SER) for lifelong learning [3], however, could help the agent retain knowledge from prior market conditions, while maintaining the adaptability to new ones.

In this work, we explore using different prioritization strategies within a SER buffer, namely favoring surprise, favoring reward, matching the global training distribution, and maximizing coverage of the state space, in order to help the agent retain critical experiences during training. Furthermore, we present a new strategy, specific for financial time series, where the market is segmented into a specific subset of regimes using change-point detection to allow agents to retain more of the strategies which may be relevant to the current market trend.

2 Background and related work

Due to their volatility, complexity and uncertainty, financial markets suffer from high noise, shifting market regimes, and delayed and sparse rewards. These challenges contribute to the inherent difficulty of financial decisions such as trading even for human experts, who may be prone to biases and inconsistent decision-making under high-stress situations.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

2.1 Deep reinforcement learning in trading

There exist several reviews which discuss the history and details of DRL in financial applications [1, 5, 7], and more specifically in trading [9]. In stock trading, DRL has been particularly advantageous in market model independence and portfolio scalability due to its ability to learn through interactions with a dynamic and unknown environment [4]. Various DRL methods have been applied to trading tasks, with current literature categorized into three main methods, namely critic-only, actor-only and actor-critic approaches. Among these, the critic-only is currently the most commonly studied method in this field, with actor-critic being the least explored [9].

To accelerate research in DRL specifically for stock trading, the open-source framework FinRL was developed [4], which provides a simulation environment emulating multi-asset stock trading through modeling portfolio allocation dynamics, transaction costs and technical indicators, called StockTradingEnv.

StockTradingEnv treats financial tasks as a Markov Decision Process (MDP) Problem, where the state space describes observations that the agent receives from the trading environment similar to what a human trader would need to recognize, including cash holdings, stock prices, technical indicators, and asset positions. On the other hand, the action space includes three main actions: $a \in -1, 0, 1$ representing the action of selling, holding or buying a stock. In the case that the action must be carried across multiple shares, the action space becomes $a = -k, ..., -1, 0, 1, ..., k$, where the agent would take action k to buy k shares, etc. Finally, for the reward function, there are many forms which can be used such as the Sharpe ratio for periods, change of portfolio value and the portfolio log return. FinRL supports multiple reward functions, as well as user-defined ones, which can include risk factor or transaction cost terms (see Appendix B for a more detailed MDP formalization).

In our implementation, we focus on using the following set of technical indicators: MACD, RSI (30), CCI (30), and DX (30). These indicators reflect the range of momentum (MACD), relative strength (RSI), trend (DX), and market deviation (CCI) signals, generally used due to their interpretability and range in capturing different aspects of market behavior [2].

Despite its advantages, applying DRL to stock trading still presents challenges. Financial markets not only suffer from high noise and high stochasticity, rewards are often sparse and delayed, and there exist no clear ground truth as to what constitutes as an optimal action. Agents must learn to balance reward and risk sensitivity maximization while minimizing portfolio volatility. These challenges can often lead to overfitting to specific market conditions and high variance in performance of agents.

2.2 Selective experience replay

In the context of trading, SER has the potential to offer stability and improve agent robustness through ensuring exposure to relevant risk management strategies from past experiences across changing market regimes, potentially addressing overfitting and instability issues. Isele and Cosgun propose SER designed for lifelong learning [3], where an agent selectively stores and replays transitions based on four different strategies, aiming to mitigate catastrophic forgetting and improve generalization.

In their work, they introduce a long-term memory storage called episodic memory, in addition to the traditional short term memory such as a FIFO buffer, which retains a subset of past experiences by maintaining the most informative experiences within a priority queue under the chosen selection policy and the restriction of resource constraints [3]. This allows the agent to be exposed to a diverse and meaningful sample of past experiences.

2.2.1 Selection strategies

There are four selection strategies described for SER [3]: favoring surprise, favoring reward, matching the global training distribution, and maximizing coverage of the state space.

Favoring surprise selects transitions with a high temporal difference (TD) error, which describes the prediction error made by the network. Using this strategy will emphasize experiences which the agent currently finds unpredictable or uncertain. The ranking function for this selection follows:

$$\mathcal{R}(e_i) = \left| r_i + \gamma \max_{a'} Q(s_i', a') - Q(s_i, a_i) \right| \tag{1}$$

Favoring reward prioritizes experiences which yield high rewards, encouraging the retention of experiences that are associated with successful behavior. The ranking function uses, for a given experience, the absolute value of the future discounted return:

$$\mathcal{R}(e_i) = |R_i(e_i)| \tag{2}$$

Distribution matching maintains a buffer distribution which matches the overall training distribution in order to prevent overfitting to specific episodes or states, and will use a random value as the key in the priority queue:

$$\mathcal{R}(e_i) \sim \mathcal{N}(0, 1) \tag{3}$$

Finally, coverage maximization selects experiences that promote a broader exploration of the state space by increasing generalization through keeping rare or diverse states within the long-term memory. The ranking function will rank experiences according to the number of neighbors at a fixed distance d:

$$\mathcal{R}(e_i) = -|N_i| \text{ where } N_i = \{e_j \mid \text{dist}(e_i - e_j) < d\} \tag{4}$$

3 Methodology

We use a Soft Actor-Critic (SAC) agent for our implementation and comparison of a classic FIFO buffer implementation (serving as the baseline) against our SER using multiple strategies.

3.1 Selective experience replay buffer implementation

We implement `SERReplayBuffer` using the base class `BaseBuffer` of Stable-Baselines3 (SB3) [6] framework. We implement the required abstract functions from SB3's `BaseBuffer` class, allowing our `SERReplayBuffer` to pass in place of the `ReplayBuffer` to SB3 for training. The four strategies surprise, reward, distribution and coverage are implemented as closely as possible to the original paper's formulation. Note that for the surprise strategy, the training algorithm is required to provide the TD-error estimates for each transition in order to be able to use the ranking function based on unexpectedness. The "selectiveness" of the replay buffer, based on the strategy-specific ranking function, occurs during the insertion of the experience. For our pseudocode (Algorithm 1) describing how each strategy constructs its insertion key and manages experience insertion into episodic memory, please see Appendix A.

During the sampling phase, our `SERReplayBuffer` samples uniformly from both the short-term FIFO memory and the long-term episodic memory. At each sampling step, 50% of the transitions are drawn from the FIFO buffer and the other 50% are sampled from the episodic memory. This allows the agent to be trained on both recent experiences as well as on important historical experiences selectively preserved according to the chosen strategy.

The size of the episodic memory is kept significantly lower than the capacity of the FIFO buffer in order to ensure the retention of only the crucially relevant transitions while maintaining a diverse training sample. In our experiments, the FIFO buffer is set to a capacity of 100,000 transitions whereas the episodic memory is set to hold a maximum of 20,000 transitions.

3.2 Change-point detection

Furthermore, we explored the integration of change point detection techniques in order to create a new ranking strategy for storing experiences within the long-term episodic memory. The main idea is that by separating the financial time series data into different market regimes, we could construct such a ranking function which will give higher priority to previous regimes currently stored in the long-term memory which match the currently detected regime. For this, a dynamic re-scoring of the transitions within the episodic memory would ideally be the best strategy.

We used the STUMPY library [8] to implement change-point detection on the financial time series (see Figure 1 for an example plot). Although our implementation differs slightly from the original

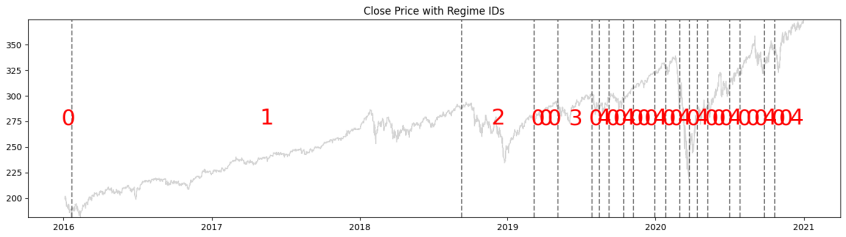

Figure 1: Change-point detection in 4 regimes using clustering

formulation [8], it still provides reasonable segments of market regimes to be used in the SER buffer. Each detected segment corresponds to a period of relatively consistent market behaviour, which we call regimes, which are clustered using KMeans based on the extracted features and tagged with a "trend id", allowing the replay buffer to distinguish between them during training. For pseudocode outlining this method, please see Appendix C.

4 Results

During our experimentations, there were several complications related to training stability and our environment behaviour. One of our major limitations was due to insufficient hardware and time constraint, as specifically in trading the rewards are very sparse and noisy, therefore the agent must be trained on long sequences until it can learn meaningful trading strategies.

We initially explored using environments `ForexEnv` and StocksEnv from gym-anytrading, however, their simplified structure and limited features led us to explore FinRL's `StockTradingEnv` further, which offers a much more comprehensive and realistic simulation of financial trade markets. We used backtracking as described in their framework to plot our results to get more accurate and meaningful, trade-domain specific plots. Despite experimenting with FinRL, however, we observed that the agent's behaviour often flatlined, either taking no action at all or consistently repeating the same action, even when experimenting with different strategies (see Appendix D).

Surprisingly, our outcomes were not consistent across runs, while using the same hyperparameters, some training runs would show some learning progress, whereas others would not, regardless of the SER strategy used nor SB3's own `ReplayBuffer`. We suspect that this inconsistency might have emerged due to issues within the FinRL implementation itself, due to several issues that we observed while training. For example, several notebooks provided under FinRL's examples were not runnable out of the box and the library (currently in version 0.3.1) appears to be outdated and potentially buggy.

Similarly, for our change-point detection, despite successfully segmenting the data into regimes, we observed that the agent's learning showed little to no learning. We believe this is likely due to the instability of `StockTradingEnv`, as described above.

5 Conclusion

In this project, we implemented Selective Experience Replay strategies for a stock trading specific domain. While the initial motivation for our project was to improve the robustness of the agent learning financial tasks (buy/sell/hold stocks) and mitigate catastrophic forgetting, our experiments revealed framework and environment specific challenges in the trading domain, such as reward sparsity and environment instability.

On the other hand, we believe our integration of change-point detection using trend-id tagging opens a new direction for selective experience prioritization specific to the trading domain.

Future directions for this project given more time should include exploring different financial simulation environments or refining the existing environments more thoroughly, investigating further where the instabilities encountered arise. Furthermore, an extension would include change-point aware SER strategies with online updates and evaluating these with broader market datasets and much longer training durations.

References

[1] Yahui Bai, Yuhe Gao, Runzhe Wan, Sheng Zhang, and Rui Song. A review of reinforcement learning in financial applications. *arXiv preprint arXiv:2411.12746*, 2024. License: CC BY 4.0. Equal contribution (authors listed alphabetically). Corresponding author: Rui Song.

[2] Jingyuan Huang, Yusen Ye, Yuze Zhang, Xiao-Yang Liu, and Christina Dan Wang. Finrl-meta: A meta-learning library for financial reinforcement learning. *arXiv preprint arXiv:2111.05011*, 2021.

[3] Donald Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[4] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. Finrl: A deep reinforcement learning library for automated stock trading in quantitative finance. *arXiv preprint arXiv:2011.09607*, 2020.

[5] Alireza Mohammadshafie, Akram Mirzaeinia, Haseebullah Jumakhan, and Amir Mirzaeinia. Deep reinforcement learning strategies in finance: Insights into asset holding, trading behavior, and purchase diversity. *arXiv preprint arXiv:2407.09557*, 2024. Regular Research Paper (CSCE-ICAI'24). License: CC0.

[6] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, and Noah Ernestus. Stable-baselines3: Reliable reinforcement learning implementations. `https://stable-baselines3.readthedocs.io/en/master/`, 2021. Accessed: 2025-04-25.

[7] Ashwin Rao and David Blei. *Reinforcement Learning for Finance*. 2022. Available online: `https://stanford.edu/~ashlearn/RLForFinanceBook/book.pdf`.

[8] TDAmeritrade. Stumpy: A powerful and scalable library for time series data mining. `https://github.com/TDAmeritrade/stumpy`, 2024. Accessed: 2025-04-25.

[9] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deep reinforcement learning for trading. *arXiv preprint arXiv:2003.10719*, 2020.

Appendix

A Experience insertion logic for long-term memory

The following pseudocode describes the insertion logic our `SERReplayBuffer` uses for each prioritization strategy. Transitions are added or discarded from the episodic memory depending on their computed ranking scores and long-term memory space availability.

Algorithm 1 Experience insertion based on SER strategy

Input: Transition (s, a, r, s', d)
Store transition (s, a, r, s', d) in short-term FIFO buffer
if strategy == *distribution* **then**
  if episodic memory has space **then**
    Append transition to episodic memory
  else
    With probability $1/n_{steps}$, replace random element
  end if
else if strategy == *reward* **then**
  Compute $key = |r|$
  if episodic memory has space **then**
    Insert transition with key into episodic memory
  else
    if key is greater than the minimum key stored in memory **then**
      Replace minimum priority transition with the new transition
    end if
  end if
else if strategy == *surprise* **then**
  Compute $key = |\text{TD error}|$
  if episodic memory has space **then**
    Insert transition with key into episodic memory
  else
    if key is greater than the minimum key stored in memory **then**
      Replace minimum priority transition with the new transition
    end if
  end if
else if strategy == *coverage* **then**
  if episodic memory has space **then**
    Append transition and build KD-tree
  else
    if current observation in a sparse region with less neighbors **then**
      Replace densest transition in episodic memory
      Rebuild KD-tree
    end if
  end if
end if=0

B MDP formalization for trading

We define trading as an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$:

- **State** $s_t \in \mathcal{S}$: a vector of price data (open, high, low, close, volume), 80+ technical indicators (MACD, RSI, ATR, Bollinger Bands, etc.), and current portfolio state (cash, shares held, total value).
- **Action** $a_t \in \mathcal{A} = \{\text{Buy}, \text{Sell}, \text{Hold}\}$: discrete trading decisions on a single asset with a max position size h_{max}.
- **Reward** r_t: change in net portfolio value penalized by transaction cost:

$$r_t = \Delta\text{portfolio value} - \text{trading cost}$$

- **Transitions** $\mathcal{P}$: ticker data downloaded from sources like Yahoo Finance.

C Change-point detection implementation

The following is our pseudcode outlining the proposed change-point detection integration:

Algorithm 2 Change-Point Detection and Trend ID Assignment

Input: Financial time series df with datetime and close price
Output: Dataframe df with assigned trend IDs
Resample df to 1-hour intervals and compute log-returns
Compute the Matrix Profile of log-returns using STUMPY
Apply FLUSS algorithm to identify change-points based on Matrix Profile
Partition time series at detected change-points into segments
for each segment **do**
  Extract features: mean and standard deviation of log-returns, average volatility ATR, BBW, DCW, and segment length
end for
Cluster segment features using K-Means into n_{clusters} regimes
Assign each original time step a trend ID according to its corresponding segment cluster =0

D Analysis of experimental challenges

Both of the following plots reveal core issues we have encountered with the FinRL `StockTradingEnv` environment.

We suspect the causes lie within a range of factors:

- Unstable, sparse or poorly specified reward signal in `StockTradingEnv`, resulting in insufficient learning signals.
- Environment may not have sufficient variablitity of actions, as suggested by Figure 3.
- Figure 3 suggests overly constrained low-action granularity, where the agent always buys a fixed quantity, limiting policy diversity.

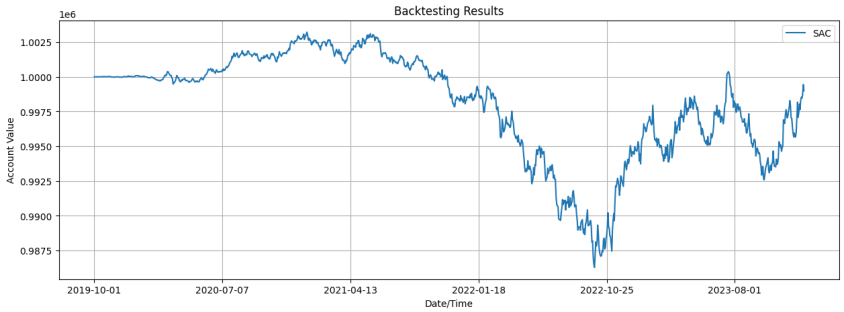

Figure 2: Account value over time for SAC agent.

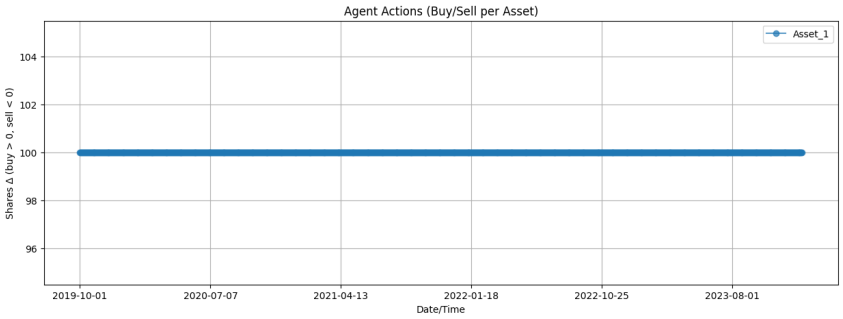

Figure 3: Agent action history across the episode

Overall, figures 2 and 3 reinforce the idea that the current environment setup used with `StockTradingEnv` needs refinement. Our experimentations highlight the importance of validating environment dynamics when designing reinforcement learning experiments specifically in the financial domain.