Received: 1 December 2022 | Revised: 2 August 2023 | Accepted: 14 September 2023

DOI: 10.1002/asmb.2825

RESEARCH ARTICLE

WILEY

A reinforcement learning algorithm for trading commodities

Federico Giorgi | **Stefano Herzel** | **Paolo Pigato**

Department of Economics and Finance, University of Rome Tor Vergata, Rome, Italy

Correspondence
Paolo Pigato, Department of Economics and Finance, University of Rome Tor Vergata, Rome, Italy.
Email: paolo.pigato@uniroma2.it

Funding information
University of Rome Tor Vergata via the Project AIF, Grant/Award Number: E83C22002610005

Abstract

We propose a reinforcement learning (RL) algorithm for generating a trading strategy in a realistic setting, that includes transaction costs and factors driving the asset dynamics. We benchmark our algorithm against the analytical optimal solution, available when factors are linear and transaction costs are quadratic, showing that RL is able to mimic the optimal strategy. Then we consider a more realistic setting, including non-linear dynamics, that better describes the WTI spot prices time series. For these more general dynamics, an optimal strategy is not known and RL becomes a viable alternative. We show that on synthetic data generated from WTI spot prices, the RL agent outperforms a trader that linearizes the model to apply the theoretical optimal strategy.

KEYWORDS

commodities, portfolio optimization, reinforcement learning, SARSA, threshold models

1 | INTRODUCTION

We propose an algorithm, based on Reinforcement Learning (RL), to determine an effective trading strategy for a dynamic model that includes transaction costs and where the asset price dynamics is driven by factors that can be either internal, like the average of previous returns, or external, like the level of a market index or its volatility. We tackle the classical problem of optimal portfolio allocation, initiated with the seminal work by Markowitz, who formulated the problem in a static setting under mean-variance utility as a quadratic programming. Markowitz result was extended to a dynamic setting by Merton,[1] who considered a model where prices follow a geometric Brownian motion and solved the associated dynamic optimization problem through the Bellman Equation. Later on, Gârleanu and Pedersen[2] (henceforth GP), provided an analytical solution to a problem where prices follow a diffusion and are linearly predicted by mean reverting factors, in a market where transactions costs are quadratic in the number of shares traded. In their paper, GP also apply their solution to determine the optimal investment strategy on a portfolio of futures on commodities, showing that, by calibrating their model in-sample and by assuming quadratic transaction costs, their strategy significantly outperform, in terms of Sharpe ratio, the Markowitz–Merton one.

The GP result is a significant theoretical breakthrough but their model assumptions are not always realistic. For instance, predicting factors, albeit present, may not be observable, their relation to price changes may be non-linear, transaction costs may not be quadratic, price dynamics may not be well represented by a diffusion. To deal with more realistic settings, where an analytical solution is not known, we propose to use RL, as it is often effective under general assumptions (see Sutton and Barto[3] for a general discussion). In RL, an agent is interacting with the environment over time. At each time, the agent observes the state of the environment and chooses an action; at the following time, the agent observes a new state of the environment and receives a reward for its action. RL algorithms provide a way to train

the agent by letting it learn through positive reinforcement with the goal of maximizing its reward. The agent does not know the dynamics of the environment and learns how to improve its strategy from the observation of a large number of trials.

Our algorithm combines the standard SARSA (State-Action-Reward-State-Action) with a neural network to estimate the value function on a continuous set of states (current shares, market realization) and actions (shares to trade). The value function is estimated by generating consecutive batches of episodes: within each batch, the agent trained on previous batches is initially used, and then its choices are gradually improved via simulations of new episodes. RL algorithms, even if effective in most cases, are not guaranteed to work on any possible instance of the problem at hand. For this reason, we perform a comparative test in a situation where the optimal solution is known. The application of the algorithm to crude oil West Texas Intermediate (WTI) shows that RL may bring some statistically significant advantages over the current best theoretical choice represented by the GP model. The comparison of several models shows that the best fit is attained for a specification of the factor dynamics that is not linear and hence does not fit GP modeling assumptions. This is a clear example of an instance where the RL approach may present some advantage over the theoretical model.

We adopt a value-based algorithm, where an approximation to the optimal value function is computed on simulations. Value-based algorithms have been originally developed in a tabular fashion for discrete and finite state-action spaces (refer to Reference 3 for more insights). In finance, several strategies have been adopted in order to generalize to the continuous case. In Reference 4, tabular methods have been extended to the continuous case by means of kernel approximation methods. Value-based algorithms have been used for example by References 5 and 6 for hedging derivatives and by Reference 7 for optimal trading, as they are easy to implement for arbitrarily complex reward functions. On the negative side they are sensitive to hyperparameters and they need a large number of simulations to converge (see Reference 8).

A different approach, called "policy search method", optimizes directly on the policies space. These methods have been applied to finance in Reference 9, to compare the performances of an agent using RL to one that adopts the GP formula, in References 10 and 11 to produce a trading algorithm and in References 12 and 13 for hedging derivatives. In Reference 14, RL is used to construct real-time trading strategies in the presence of trading signals. They are considered to be robust on noisy datasets and fast converging. Furthermore, it is possible to build algorithms with safety guarantees that ensure an improvement of the policy at each update (see References 15-18). However, policy search methods are often complex to implement and very objective-specific (see Reference 19 for a survey).

Outline. In Section 2 we revisit the RL theory and describe the modified SARSA algorithm. In Section 3 we compare the trained algorithm to the GP optimal strategy for trading with transaction costs. In Section 4 we calibrate several non-linear models to the WTI time-series and combine the RL algorithm with the most realistic one. We prove by simulation that, in this case, the RL outperforms GP. In Section 5 we conclude the paper and propose possible developments.

2 | THE REINFORCEMENT LEARNING ALGORITHM

Let us introduce the basic ideas and definitions on RL. The RL agent learns how to interact with an environment by receiving some rewards for its actions: it observes the state of the environment s_t, selects an action a_t, receives a reward R_{t+1} and observes a new state s_{t+1} as a consequence, possibly random, of s_t and a_t. The procedure is then repeated until a final time T (possibly infinite). The sequence of states, action and rewards is stochastic, and the mathematical framework used for modeling such decision making problem is provided by a Markov decision process (MDP). A MDP is a Markov process that is embedded with the possibility for an agent to interact with its evolution through its actions. In a MDP, the law of the time $t+1$ state and reward does not depend on the full path of state, action and rewards up to t, but only on the most recent state and action

$$\mathcal{L}(s_{t+1}, R_{t+1}|s_0, a_0, R_1, s_1, a_1, \ \dots \ R_t, s_t, a_t) = \mathcal{L}(s_{t+1}, R_{t+1}|s_t, a_t) \tag{1}$$

where $\mathcal{L}(X|y)$ denotes the law of the random variable X conditional on the observation of $Y = y$. The policy, or strategy, followed by the agent of selecting an action after observing a state is expressed as a function $\pi(s) = a$. For ay initial probability distribution on the states s_0, the transition probability $\mathcal{L}(s_{t+1}, R_{t+1}|s_t, a_t)$ (1), together with the policy π completely determines the progress of the system through a *homogeneous Markov chain*. Refer to Reference 20 for details on how to

generalize the setting to time-changing environments. The agent's goal is to maximize the expectation of the cumulative (discounted) rewards, that is to determine the optimal policy

$$\pi^* = \arg\max_{\pi} \mathbb{E}^{\pi}\left[\sum_{k\geq 0}\gamma^k R_{t+k+1}\right] \tag{2}$$

where $\mathbb{E}^{\pi}$ is the expectation operator based on the MDP distribution (1) and following the policy π, while γ is a discount factor.

The state-action value function of a policy π is defined as the expected value of cumulative rewards when the initial state is s, the selected action is a and the policy π is followed,

$$q_{\pi}(s,a) = \mathbb{E}^{\pi}\left[\sum_{k\geq 0}\gamma^k R_{t+k+1}|s_t = s, a_t = a\right]. \tag{3}$$

Observe that, due to the stationarity of the policy, both the problem definition (2) and the action-value function (3) are time independent.

All optimal policies share the same optimal action-value function $q_*(s,a) = \max_{\pi} q_{\pi}(s,a)$, which satisfies the Bellman optimality equations (see Reference 3 for a proof)

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a_{t+1}\in\mathcal{A}(s_{t+1})} q_*(s_{t+1}, a_{t+1})|s_t = s, a_t = a\right] \tag{4}$$

where $\mathcal{A}(s_{t+1})$ determines the state-dependent action space. To each action-value function $(s,a)\to q(s,a)$ it is possible to associate its q-greedy policy. Such policy consists in choosing the action a that, in a given state s, maximizes the action-value function (3)

$$\pi^q_{greedy}(s) = \arg\max_{a\in\mathcal{A}(s)} q(s,a) \tag{5}$$

The solution to (5) does not need to exist or to be unique, depending on the nature of the RL problem[1]. From the Bellman equation (4), the greedy policy (5) associated to the optimal value function q_* is a solution to the RL problem (2), or $\pi^* = \pi^{q_*}_{greedy}$. In turn, the Bellman equation (4) can be expressed as

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma q_*(s_{t+1}, \pi^{q_*}_{greedy}(s_{t+1}))|s_t = s, a_t = a\right] \tag{6}$$

The main idea behind value-based algorithms is to approximate the solution to (6) by an iterative procedure

$$q_{k+1}(s,a) = \mathbb{E}\left[R_{t+1} + \gamma q_k(s_{t+1}, \pi^{q_k}_{greedy}(s_{t+1}))|s_t = s, a_t = a\right], \quad k = 0, 1, \ldots \tag{7}$$

This is called "iterative policy evaluation" and its convergence to the value function is a consequence of the Fixed-Value Theorem. Therefore, an approximation to the optimal policy can be obtained by a sequence of greedy policies $\pi^{q_k}_{greedy}$, a procedure called "iterative policy improvement". Refer to Reference 3 and references therein.

The estimation of the optimal value function is typically done by simulations, where the MDP assumption is used to generate a large number of paths, or "episodes", which are then used to approximate the solution of the Bellman equation (4).

Indeed, ideas and definitions introduced so far are well known in dynamic programming. A relevant difference between RL and dynamic programming is that the RL agent does not need to know the distribution of the MDP and learns the optimal strategy after repeated interactions with the environment. In RL, the process of learning about the environment, and how it reacts to different actions, is called exploration, while the process of following the greedy

[1]Consider, for instance, the the action-value function $q(s,a) = a^2$ on the unbounded action space $\mathcal{A}(s) = \mathbb{R}$, in which case the maximum does not exist, or on the symmetric action space $\mathcal{A}(s) = [-A, A]$, in which case the maximum is not unique.

policy associated to the current estimate of the state-value function is called exploitation. RL algorithms alternate between exploration and exploitation.

In particular the RL algorithm adopted in this paper belongs to the class of SARSA methods, proposed by Reference 21, and refined by Reference 3. To select the action after observing the state, the SARSA algorithm balances exploration and exploitation adopting the ϵ-greedy policy (of a value function q) defined as

$$\pi^q_{\epsilon\text{-greedy}}(s) = \begin{cases} \tilde{a}, & \text{if } u < \epsilon \\ arg\,max_a\, q(s,a), & u \geq \epsilon \end{cases} \tag{8}$$

where $\epsilon \leq 1$ is a positive constant, u is uniformly sampled on $(0,1)$ and $\tilde{a}$ is randomly chosen according to the uniform distribution over $\mathcal{A}(s)$. The $\pi^q_{\epsilon\text{-greedy}}$ policy selects a random action with probability ϵ (exploration) and coincides with the π^q_{greedy} policy with probability $1-\epsilon$ (exploitation). At the first step of the algorithm the value of ϵ is large, so that more weight is assigned to exploration, and after each step of the algorithm it is decreased towards zero so that the policy becomes more exploitative (closer to the greedy one).

The name SARSA derives from the fact that the value function is estimated by successive simulations of the sequence State-Action-Reward-State-Action $s_0^{(j)}, a_0^{(j)}, r_1^{(j)}, s_1^{(j)}, a_1^{(j)}, \ldots$ as produced by the MDP (1), which are then used to estimate the optimal value function q^* by iteratively solving the Bellman equation (4) by means of the SARSA updating formula

$$q_{k+1}\left(s_{t-1}^{(j)}, a_{t-1}^{(j)}\right) = q_k\left(s_{t-1}^{(j)}, a_{t-1}^{(j)}\right) + \alpha\left(R_t^{(j)} + \gamma q_k\left(s_t^{(j)}, a_t^{(j)}\right) - q_k\left(s_{t-1}^{(j)}, a_{t-1}^{(j)}\right)\right) \tag{9}$$

where $\alpha \in [0,1]$ is a "learning rate" that allows to fine tune the learning speed: setting it equal to 0 means that the value function is never updated, whereas setting it equal to 1 means that learning can occur quickly. The reader may notice that the full-strength SARSA updating formula (9) resembles the term in the expectation on the right-hand-side of the iteration formula (7). Originally, the SARSA algorithm was developed under the assumption of finite state space and action space $\mathcal{A}$. In such situation, it can be shown that, provided that enough episodes are simulated, then all the possible state-action couples (s,a) are updated often enough to reach convergence of the value function q_k on the entire grid. Under suitable conditions, the algorithm produces a sequence of value functions converging to the optimal one (see Reference 3 and references therein.)

Since in our case the action space is a continuous set, we propose a modification of the standard SARSA algorithm that uses a supervised regressor to estimate the value function. An extensive literature has been developed to extend the application of SARSA to continuous state-action spaces. One possibility, is to discretize the state-action spaces and then to apply the standard SARSA. With this approach, convergence is ensured from the results obtained for the finite case (see Reference 3). However, providing a discretization that is dense enough to capture the continuous features of states and actions and yet coarse enough to ensure reasonable computational effort is not an easy task. A more refined extension of SARSA to the continuous case has been studied in Reference 22 by means of Newton's Method for direct optimization of the state-action value function for action selection, in order to allow continuous action reinforcement learning without a separate policy function. In this paper we propose to use Neural Networks.

Let us now describe the algorithm, whose pseudo-code is shown in Algorithm 1. The training process is performed on consecutive "batches" of episodes, obtained by simulating the MDP (1). At the beginning of the algorithm, an estimate for the value function $\hat{q}^{(0)}(\cdot,\cdot)$ must be initialized. Such initialization imposes a bias on the algorithm and can be tackled in different ways. The most agnostic approach is the zero initialization $\hat{q}^{(0)}(s,a) = 0$, reflecting the fact that no episode has yet been generated. A somewhat opposite approach is the "optimistic" initialization. In this case, the initial estimate is set to a constant, large value $\hat{q}^{(0)}(s,a) = Q \gg 0$; then, as the training proceeds, the estimate on the state-action pairs that are actually visited tends to decrease to its true value, leaving the large value for un-visited state-action pairs. Then, by following the greedy policy (5), the agent tends to select actions that are yet-to-be visited, thus encouraging exploration. In this setting, exploration is tuned by means of the ϵ-greedy policy (8), therefore we do not use the optimistic initialization approach. However, the zero initialization approach presents some technical difficulties in the definition of greedy policy (5): in fact, if the value function is constant, the action providing the maximum value is not well defined. Therefore, we choose an approach whereby the initial value function is set $\hat{q}^{(0)}(s,a) = 0$ and the ϵ parameter in the ϵ-greedy policy is set to $\epsilon_0 = 1$.

In the exploration phase occurring in the ϵ-greedy policy (8), the agent can explore the action space in different fashions by choosing different strategies to randomize the exploring action $\tilde{a} \in \mathcal{A}(s)$. The most agnostic way is to draw $\tilde{a}$

uniformly on the action space $\mathcal{A}(s)$. Alternatively, more probability weight can be assigned to areas where one believes better rewarding actions are located.

In the exploitation phase occurring in the ϵ-greedy policy (8), an optimization of the current estimate of the state-action value function $\hat{q}^{(n)}(s,a)$ must be performed on the action space $\mathcal{A}(s)$. In order to do so, we use the SHGO optimization algorithm, which has been proven to be appropriate for solving blackbox low-dimensional global optimization problems (refer to Reference 23 for details).

The simulations of the episodes are used to obtain simulations of the SARSA updating formula scheme (9)

$$q^{(j)}\left(s_t^{(j)}, a_t^{(j)}\right) = \hat{q}^{(n-1)}\left(s_t^{(j)}, a_t^{(j)}\right) + \alpha\left(R_{t+1}^{(j)} + \gamma \hat{q}^{(n-1)}\left(s_{t+1}^{(j)}, a_{t+1}^{(j)}\right) - \hat{q}^{(n-1)}\left(s_t^{(j)}, a_t^{(j)}\right)\right) \tag{10}$$

At the end of each batch, a Neural Network interpolation $N(s,a;\boldsymbol{\theta}^*)$ is fitted on the relation $(s_t^{(j)}, a_t^{(j)}) \to q^{(j)}\left(s_t^{(j)}, a_t^{(j)}\right)$ by using the grid of points generated within the batch. Then, a new state value function is produced by model averaging

$$\hat{q}^{(n)}(s,a) = \eta N(s,a;\boldsymbol{\theta}^*) + (1-\eta)\hat{q}^{(n-1)}(s,a) \tag{11}$$

where we set the parameter $\eta \in [0,1]$ to assign the proper weight to previous estimates of the state value function. Finally, the exploration parameter ϵ is reduced at the end of each step to get convergence to the greedy policy. In our implementation[2], a Neural Network $N(s,a;\boldsymbol{\theta}^*)$ with only 3 layers, with 64 hidden neurons on the first layer, 32 neurons on the second layer and 8 neurons on the last proved to be sufficient. The activation function defining each hidden neuron is the standard rectified linear unit, or ReLU (see Reference 24). The solver for the weights optimization is the standard Adam algorithm by Reference 25.

3 | THE GARLEANU–PEDERSEN MODEL: REINFORCEMENT LEARNING VS DYNAMIC PROGRAMMING

To validate the procedure explained in the previous section, we benchmark our RL algorithm against the model proposed by GP in Reference 2 for which the optimal strategy can be computed in closed form using dynamic programming. GP consider the prices p_t of S securities and the changes in prices in excess of the risk-free return

$$x_{t+1} = p_{t+1} - (1+r^f)p_t.$$

The price changes are driven by a K-dimensional factor f as

$$x_{t+1} = \mu_r + Bf_t + u_{t+1} \tag{12}$$

$$f_{t+1} - f_t = \mu_f - \Phi f_t + \varepsilon_{t+1} \tag{13}$$

where Φ is a constant matrix (subject to conditions to make the process stationary) and the noises are Gaussian and uncorrelated, $u_t \sim N(0,\Sigma)$ i.i.d., $\varepsilon_t \sim N(0,\Omega)$ i.i.d., $u_t \perp \varepsilon_t$. The goal is to determine the strategy π, defined by the number of shares n_t hold in the portfolio at time t, maximizing the risk-cost penalized present value of the future expected gains

$$\pi^* = \underset{\pi}{arg\,max}\, \mathbb{E}^{\pi}\left\{\sum_{t=1}^{\infty}\gamma^{t-1}\left(\gamma\left(n_{t-1}'x_t - \frac{\kappa}{2}n_{t-1}'\Sigma n_{t-1}\right) - c(\Delta n_{t-1})\right)\right\} \tag{14}$$

where γ and κ are constants representing the discount factor and the risk-aversion, $c(\Delta n)$ are the transaction costs paid when trading $\Delta n_{t-1} = n_t - n_{t-1}$ shares, and $(\cdot)'$ denotes the transpose.

[2]Source code available on GitHub at https://github.com/dynamic-trading-RL/dynamic-trading.

Algorithm 1. The pseudo-code for the SARSA algorithm.

Set $n = 0$, the initial value function estimate $\hat{q}^{(0)}(\cdot,\cdot)$, the value of $\epsilon = \epsilon_0$, the value of the decreasing factor k, the model averaging parameter η, and the number of episodes J in each batch.Repeat

1 Set $n = n + 1$

2 For $j = 1, \dots, J$

a Simulate state $s_0^{(j)}$, compute $a_0^{(j)} = \pi^{\hat{q}^{(n-1)}}_{\epsilon\text{-greedy}}\left(s_0^{(j)}\right)$

b For $t = 1, \dots, T$

i Simulate the new state $s_t^{(j)}$ and compute the reward $R_t^{(j)}$

ii Compute the new action $a_t^{(j)} = \pi^{\hat{q}^{(n-1)}}_{\epsilon\text{-greedy}}\left(s_t^{(j)}\right)$

iii Compute the estimate of the value function for the previous state and action

$$q^{(j)}\left(s_{t-1}^{(j)}, a_{t-1}^{(j)}\right) = \hat{q}^{(n-1)}\left(s_{t-1}^{(j)}, a_{t-1}^{(j)}\right) + \alpha\left(R_t^{(j)} + \gamma\hat{q}^{(n-1)}\left(s_t^{(j)}, a_t^{(j)}\right) - \hat{q}^{(n-1)}\left(s_{t-1}^{(j)}, a_{t-1}^{(j)}\right)\right)$$

3 Fit the Neural Network $N(s, a; \boldsymbol{\theta})$ over all the episodes of the batch

$$\boldsymbol{\theta}^* = arg\min_{\boldsymbol{\theta}} \sum_{t,j}\left(q^{(j)}\left(s_t^{(j)}, a_t^{(j)}\right) - N\left(s_t^{(j)}, a_t^{(j)}; \boldsymbol{\theta}\right)\right)^2$$

4 Update the value function for the next batch

$$\hat{q}^{(n)}(s, a) = \eta N(s, a; \boldsymbol{\theta}^*) + (1 - \eta)\hat{q}^{(n-1)}(s, a)$$

5 Set $\epsilon = k\epsilon$

Until convergence. Return $\hat{q}^{(n)}(\cdot,\cdot)$

The optimal strategy for quadratic trading costs $c(a) = \frac{\lambda}{2}a'\Sigma a$[3] satisfies the relation

$$n_t = (1 - \eta)n_{t-1} + \eta \times aim_t \tag{15}$$

where η is a constant depending on the parameters of the model and aim_t is a weighted combination of the expected values of the future Markowitz portfolios (see Reference 26) defined as

$$Markowitz_t := (\kappa\Sigma)^{-1}(\mu_f + Bf_t) \tag{16}$$

When trading costs are very high, that is when λ grows to infinite, the solution converges to the static strategy $n_t = n_{t-1}$; when there are no trading costs, or $\lambda \to 0$, the solution converges to the Markowitz solution.

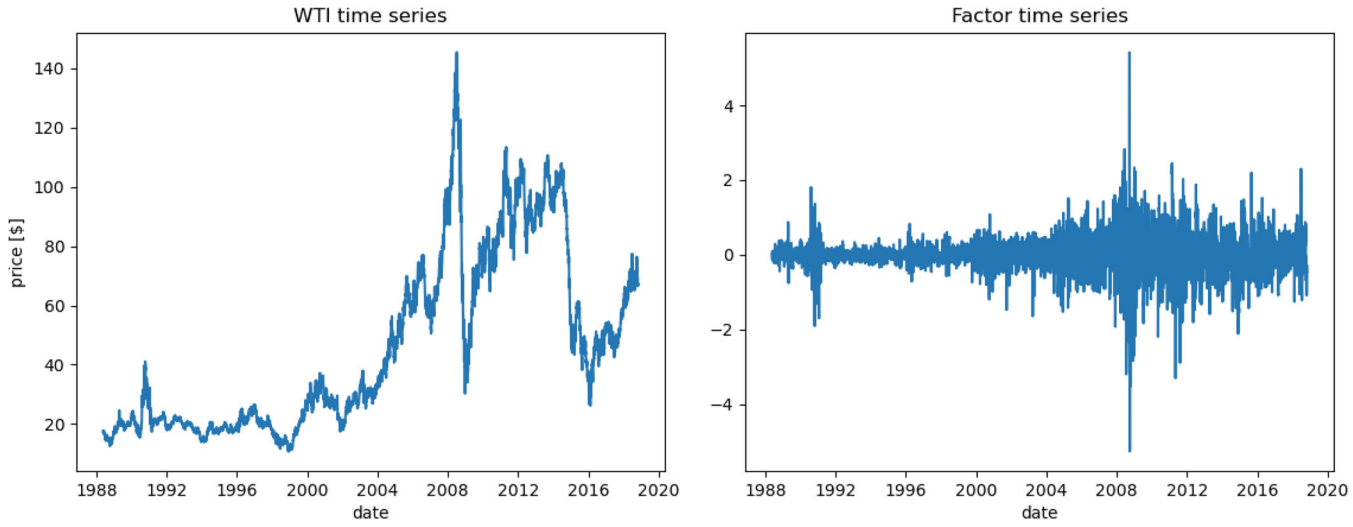
For the RL algorithm we need to define state and actions spaces and reward function. The definition of the state space is very important, as it should contain all the relevant information needed by the agent to select an action, but to keep the computational workload feasible it should not be too large. The state variables should include the current time t, the price change x_t, the observed factor f_t and the current number of shares n_{t-1}. The action variable is given by the number of shares traded

$$a_t = \Delta n_t = n_t - n_{t-1}. \tag{17}$$

The reward is obtained from the objective function (14) as

$$R_{t+1} = \gamma\left(n_t'x_{t+1} - \frac{\kappa}{2}n_t'\Sigma n_t\right) - c(\Delta n_t) \tag{18}$$

[3]More precisely, Gârleanu and Pedersen solve the problem with the more general cost formulation $c(a) = \frac{1}{2}a'\Lambda a$, where Λ is a symmetric positive-definite matrix. Then, they specialize the discussion on the case where transaction costs are proportional to the P&L covariance Σ in terms of a cost friction parameter λ. In fact, such formulation provides intuitive results (as discussed below) as well as having micro-foundations.



(A) WTI time series (B) Factors time series

FIGURE 1 WTI observed prices (A) and increments of the corresponding factor (B) computed as in (20).

TABLE 1 Maximum likelihood estimated parameters for the GP model (12)–(13).

$\mu_r = 0.007$	$B = -0.083$	$u_t \sim N(0, 1.349)$
$\mu_f = 0.001$	$\Phi = 0.228$	$\varepsilon_t \sim N(0, 0.100)$

In order to get a bounded action space we introduce an upper bound on the absolute value of the shares

$$\sup_t \|n_t\|_\infty < M. \tag{19}$$

where $M \in \mathbb{R}^S_{>0}$ is a vector of positive bounds for each of the assets positions. Therefore we impose that, for all times t,

$$|a^i_t| < M^i - n^i_{t+1}$$

for all $i = 1, \dots, S$, thereby considering a state-dependent action space. We remark that, by appropriately defining a lower and an upper bound on the trades, one can also include short sale or budget constraints in the RL formulation, generalizing the unconstrained GP solution (15).

To compare the performances of the RL algorithm to that of the theoretical optimum by GP, we consider the West Texas Intermediate (WTI) daily spot prices from Jan-1986 to Jul-2019[4]. We assume that the factor depends on previous observations of the process x, as could be believed by an investor following a momentum strategy

$$f_t = \frac{1}{5}(x_t + x_{t-1} + x_{t-2} + x_{t-3} + x_{t-4}). \tag{20}$$

In Figure 1A we display the WTI time series, in Figure 1B the corresponding factor time series computed as in (20). Maximum likelihood estimation of the GP model (12)–(13) returns the parameters in Table 1[5]. We consider a trading period of $T = 50$ days. Following GP in Reference 2, we assume the costs parameter to be $\lambda = 0.015$ (which yields a cost for unit trade of $c(1) = \$0.01$), a risk aversion parameter $\kappa = 10^{-3}$ and a 2% continuously compounded annual rate, providing a daily discount factor $\gamma = e^{-0.02/252}$.

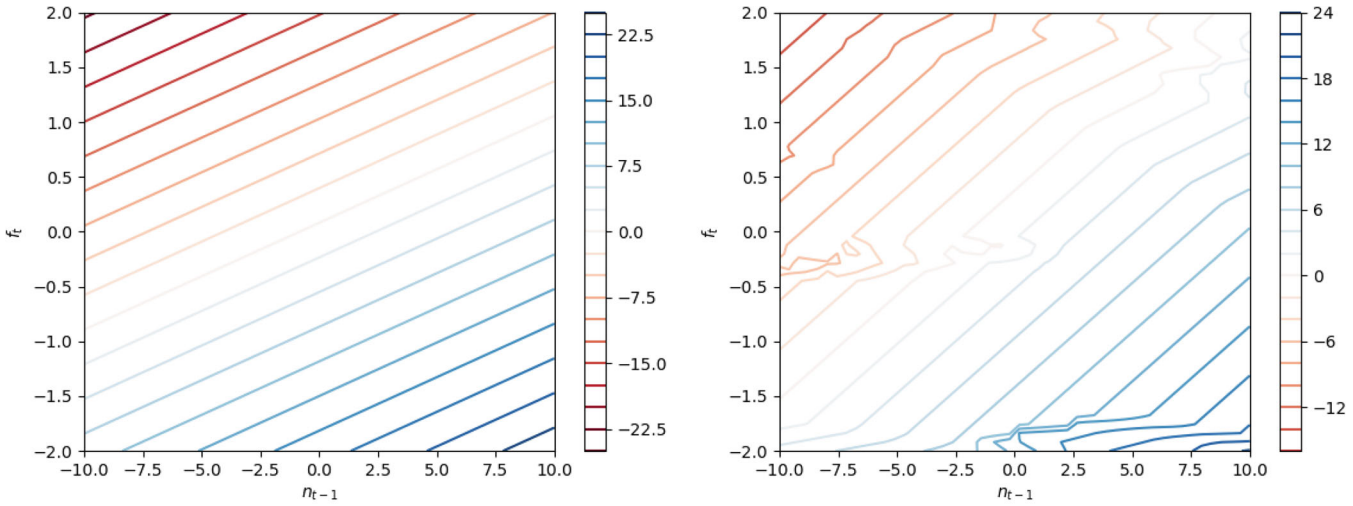
We train the agent on consecutive batches; in each batch, $J = 15{,}000$ episodes are generated by using the model (12)–(13). The ϵ-greedy policy starts with a parameter $\epsilon = 1\%$, which is then decreased as $\epsilon \leftarrow \epsilon/3$ at each batch iteration.

[4]Source: FRED (Federal Reserve Economic Data)–Economic Research Division–Federal Reserve Bank of St. Louis.

[5]We fit all the models presented in this paper on the time period from 1988-05-17 to 2018-10-29; then, we test our strategies on the time period that was held out from the estimation, that is, from 2018-10-30 to 2019-01-07, which represents 50 trading days.

TABLE 2 Progress of the RL algorithm: n is the number of batches, ϵ is the exploration parameter, $\overline{q}^{(n)}$ is the average of the value function at step n.

n	ϵ	$\overline{q}^{(n)}$
0	1.000%	−1238.259
1	0.333%	5.445
2	0.111%	7.084
3	0.037%	7.790
4	0.012%	8.176
5	0.004%	8.425



(A) GP strategy: shares (B) RL strategy: shares

FIGURE 2 The number of shares n_t given the factor f_t and previous number of shares n_{t-1} according to GP (A) and RL (B).

Model averaging is also applied with $\eta = 0.5$ in (11), in order to avoid overfitting and get a more robust convergence, less dependent on the specific simulations of the current batch. The algorithm stops when the update of the value function is negligible, in the sense that $\|\hat{q}^{(n)}(\cdot,\cdot) - \hat{q}^{(n-1)}(\cdot,\cdot)\|$ is smaller than a pre-determined level. Table 2 displays the values $\overline{q}^{(n)}$ of the average of the value function at step n for a random sample of starting points extracted from the stationary distribution of the process in (12)–(13). Table 2 shows that the average of the value function is improving at each step of the algorithm.

To set the bound M in (19) so that the action space is sufficiently large to include the optimal strategy, we compute the trades of the Markowitz strategy (16) on simulated paths and set M as the 99.5 upper percentile of their absolute values. This is motivated by the fact that we expect the trades given by the Markowitz strategy (which does not take into account trading costs) to be an upper bound for the optimal strategy (15).

The maximization over the action space required by the ϵ-greedy policy (8) is performed by using the SHGO global optimization algorithm.[23]

The left (right) panel of Figure 2 reports the shares hold at time t by the GP (RL) trader as a function of the values of the factor f_t and of the previous number of shares. We see how the strategy of the RL agent mimics the analytical optimal one. If we focus on the vertical line $n_{t-1} = 0$, we can see that, when the factor f_t is positive, the next step shares are negative, whereas when the factor f_t is negative, the next step shares are positive. This behavior is expected, since the estimated parameter $B = -0.083$ implies that positive factors predict negative price changes.

To backtest the strategies, we consider a trading period of T days on the historical time series and compare the three strategies GP, Markowitz and RL. For each strategy, we report in Figure 3C the realized cumulative risk-cost penalized

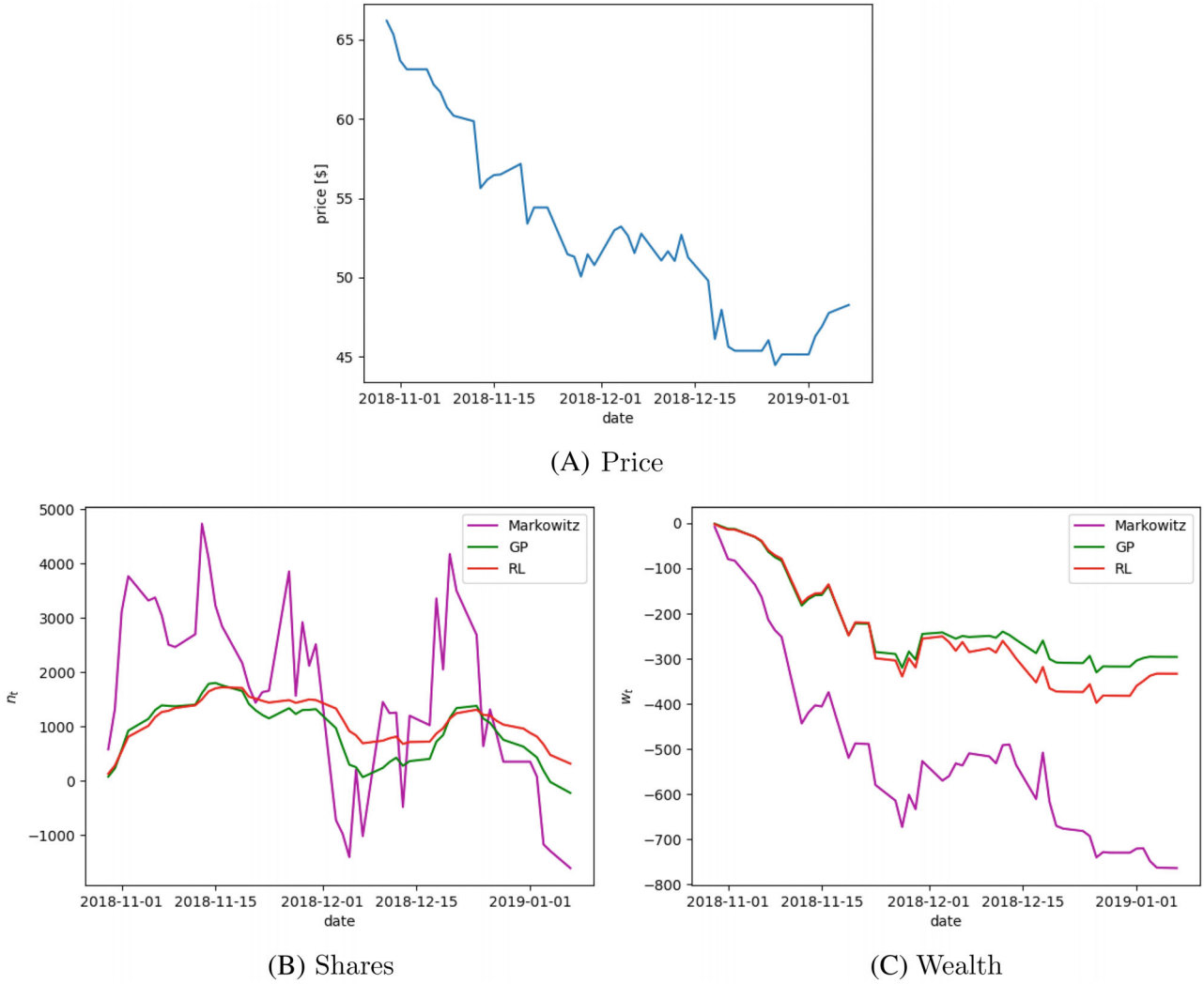


FIGURE 3 Evolution of portfolio position (B) and wealth (C) according to Markowitz, GP and RL strategies.

present value of the future expected gains, which, following References 5 or 7, we refer to as "wealth" (even if it should be noted that this is not a monetary quantity, but rather a total utility indicator)

$$w_t = \sum_{s=1}^{t} \gamma^{s-1} \left(\gamma \left(n'_{s-1} x_s - \frac{\kappa}{2} n'_{s-1} \Sigma n_{s-1} \right) - c(\Delta n_{s-1}) \right) \tag{21}$$

In Figure 3B we see that the Markowitz strategy shows wider oscillations in the number of shares than the optimal and RL strategies. This behaviour is due to the fact that the Markowitz model does not consider transaction costs and hence returns a lower cumulative wealth. We also see that the RL strategy closely follows the optimal one. For all three strategies, the return is strongly negative. This fact can be explained by looking at the negative trend of the asset in the testing period (see Figure 3A), which appears to be a disruptive event if compared to the long term behavior of the asset time series (see Figure 1A) that was used for training. However, we observe that both the GP and the RL strategies are more successful in mitigating the losses than the Markowitz one.

For a statistical comparison we simulate 10,000 paths of the factors f_t and the price changes x_t according to the model in (12)–(13), we apply the GP and RL strategies and we store the final wealths (21) obtained on each path. We get the following estimates for the sample mean and standard deviation

$$\mathbb{E}\left\{w_T^{GP}\right\} = 11.24, \qquad \mathbb{S}d\left\{w_T^{GP}\right\} = 100.25,$$
$$\mathbb{E}\left\{w_T^{RL}\right\} = 8.55, \qquad \mathbb{S}d\left\{w_T^{RL}\right\} = 101.88.$$

To test the hypothesis that the expected final wealth of the optimal and RL agents are equal, we use a two-sided Welch's t-test

$$H_0 : \mathbb{E}\left\{w_T^{RL}\right\} = \mathbb{E}\left\{w_T^{GP}\right\}$$
$$H_1 : \mathbb{E}\left\{w_T^{RL}\right\} \neq \mathbb{E}\left\{w_T^{GP}\right\}$$
$$t\text{-statistic} = -1.879, \; p = 0.060$$

The null hypothesis of equality between the final wealth attained by RL and GP is not rejected. See also next Figure 9A.

4 | REINFORCEMENT LEARNING WITH NONLINEAR MODELS

The recent empirical study[27] uses data on commodity futures transactions in the 2004–2013 period, including energy commodities, to uncover several stylized facts on commodity futures prices and volatilities, that clearly do not fit the GP framework. With this in mind, in what follows we determine a non-linear model that better fits the WTI and factors time series x_t and f_t (see Figure 1), and then compare the out-of-sample performances of a trading strategy produced by the RL algorithm trained on the non-linear model to those produced by GP.

We consider a non-linear model, where two different regimes are in force, depending on the value taken by the factor f_t

$$x_{t+1} = \begin{cases} \mu_r^{(0)} + B^{(0)} f_t + u_{t+1}^{(0)} & \text{if } f_t < c \\ \mu_r^{(1)} + B^{(1)} f_t + u_{t+1}^{(1)} & \text{if } f_t \geq c \end{cases}, \quad u_{t+1}^{(i)} \sim N(0, \Sigma^{(i)}) \tag{22}$$

where c is a constant threshold (that we set equal to 0) and $u_t^{(0)}, u_t^{(1)}$ are mutually independent. By calibrating the model (22) to the WTI time series and using the same factors as in (20) we get the parameters displayed in Table 3. Figure 4 represents the expected price change as a function of the observed factor for the linear and the non-linear model. The estimated slope of the linear model is $B = -0.083$, therefore a negative realization of the factor predicts a positive

TABLE 3 Parameters of model (22), with $c = 0$, fitted to WTI time series.

$\mu_r^{(0)} = 0.025$	$B^{(0)} = 0.014$	$u_t^{(0)} \sim N(0, 1.370)$
$\mu_r^{(1)} = 0.081$	$B^{(1)} = -0.276$	$u_t^{(1)} \sim N(0, 1.325)$

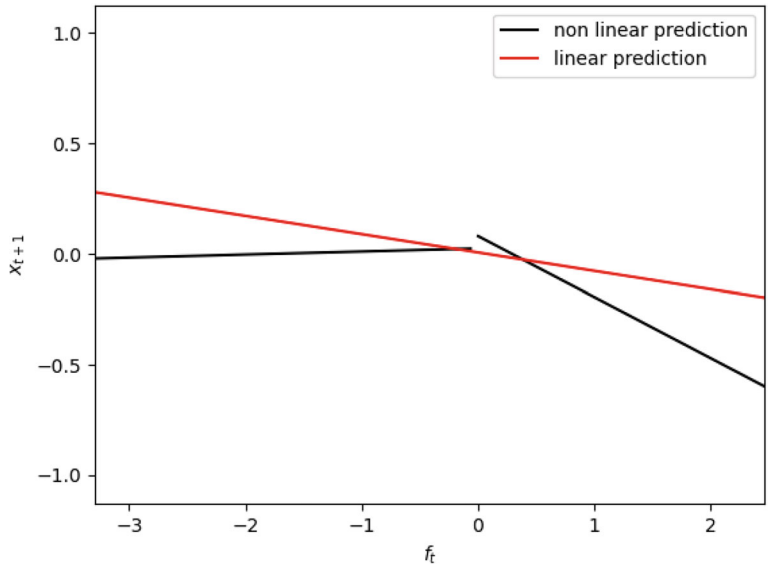


FIGURE 4 Expected value $E[x_{t+1}|f_t]$ as a function of f_t, for the linear (13) and non-linear (22) model.

realization of x. On the other hand, the nonlinear model predicts negative values for all the occurrences of the factor, both positive and negative, provided that their absolute value is sufficiently large. In particular, large positive factors predict price changes that are more negative than those predicted by the linear model.

Let us turn our attention to the forecasting factors. Figure 1B shows that the factor time series is not homoskedastic and displays volatility clustering. To model it we propose five alternatives:

1. AR

$$f_{t+1} - f_t = \mu_f - \Phi f_t + \varepsilon_{t+1}$$

This is the dynamics assumed by GP. As it is well known, it does not support the homoskedasticity and volatility clustering that we observe in the data.

2. SETAR

$$f_{t+1} - f_t = \begin{cases} \mu_f^{(0)} - \Phi^{(0)} f_t + \varepsilon_{t+1}^{(0)} & \text{if } f_t < c \\ \mu_f^{(1)} - \Phi^{(1)} f_t + \varepsilon_{t+1}^{(1)} & \text{if } f_t \geq c \end{cases}, \quad \varepsilon_{t+1}^{(i)} \sim N(0, \Omega^{(i)}) \tag{23}$$

The SETAR (Self-Exciting Threshold AutoRegressive) model was proposed by Reference 28. Similarly to the AR model, it is mean-reverting. However, unlike the AR model, it allows different autoregression and volatility intensities, according to the position of the process at the previous step, above or below a given threshold c (that we set equal to zero in our calibration).

3. AR-TARCH

$$\begin{cases} f_{t+1} - f_t = \mu_f - \Phi f_t + \varepsilon_{t+1} \\ \varepsilon_{t+1} = \sigma_{t+1} e_{t+1}, \quad e_{t+1} \sim N(0,1) \\ \sigma_{t+1}^2 = \omega + \alpha \varepsilon_t^2 + \gamma \varepsilon_t^2 1_{\varepsilon_t < 0} + \beta \sigma_t^2 \end{cases} \tag{24}$$

The factor f follows the AR model with TARCH (Threshold ARCH) volatility (see Reference 29). In the TARCH model, the volatility process includes a threshold (that we set to 0) with an extra term that is activated only when the previous realization of the factor innovation is below the threshold. This model should be able to fit both the mean reversion on the factor process and the (threshold) volatility clustering on the volatility process.

4. TARCH. This is a sub-case of the AR-TARCH model, with the parameter Φ set to zero, that is, the process f is not mean-reverting.
5. GARCH. This is also a sub-case of the AR-TARCH model, with the parameters Φ and γ both set to zero. The GARCH model is ubiquitous in finance and is used to capture volatility clustering of time series by modeling the volatility process in an autoregressive fashion.

In Figure 5 we show the residuals obtained after fitting each model. The time series are clearly non-stationary for AR and SETAR models while look more stationary for TARCH, GARCH and AR-TARCH. Figure 6 shows the autocorrelation functions for the AR model and for the AR-TARCH model; as we can see, the AR provides residuals with stronger autocorrelation than the more refined AR-TARCH, which is able to capture volatility clustering. To chose between these models we apply the Akaike information criterion (AIC),[30] and the Bayesian information criterion (BIC).[31] BIC and AIC are model selection criteria that rank the explanatory power of the models taking into account their likelihood as well as the number of parameters. We report in Table 4 the Log-likelihood, AIC and BIC for the models under consideration. The model to be preferred is the AR-TARCH, since its BIC and AIC are the lowest: therefore, in what follows we use AR-TARCH as the data generating process to train the RL algorithm. The best fit of model (24) with $c = 0$ on the factors time series (20) gives the parameters in Table 5.

Let us now compare the strategies of two traders. The first trader (GP) fits the model (12)–(13), that is linear on the price changes and autoregressive on the factors, and then applies the GP solution (15). The second trader (RL) fits the non-linear model on the price changes (22) and the AR-TARCH (24) model on the factors and applies the RL algorithm.

In Figure 7 we show the policies defining the shares owned by the GP investor and by the RL-trained investor. Focusing on the vertical line $n_{t-1} = 0$, we see that the monotonic behavior shown in the GP solution (Figure 7A) is not preserved by

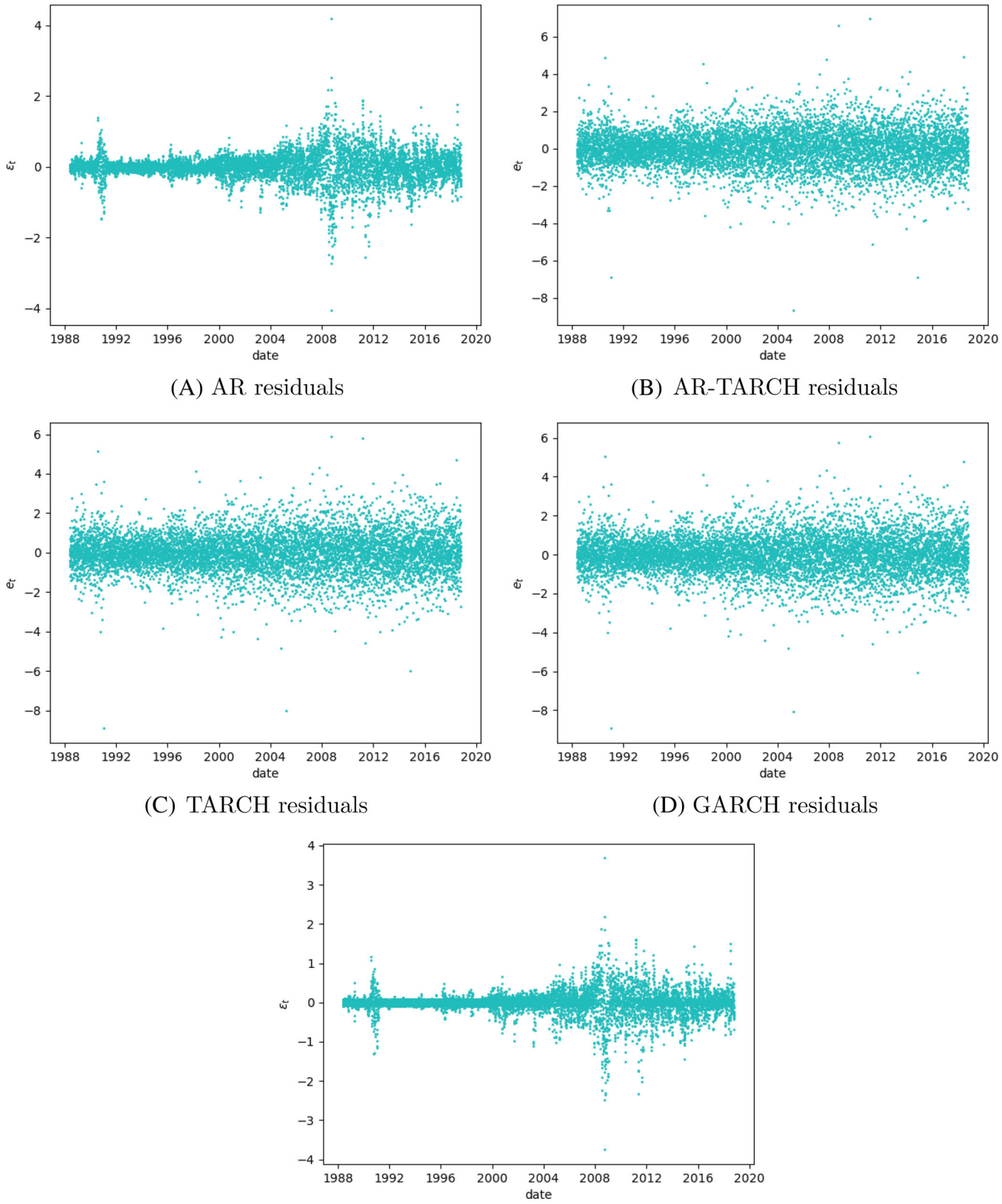


FIGURE 5 Time series of residuals ε_t for different non-linear models (A) AR; (B) AR-TARCH; (C) TARCH; (D) GARCH; (E) SETAR.

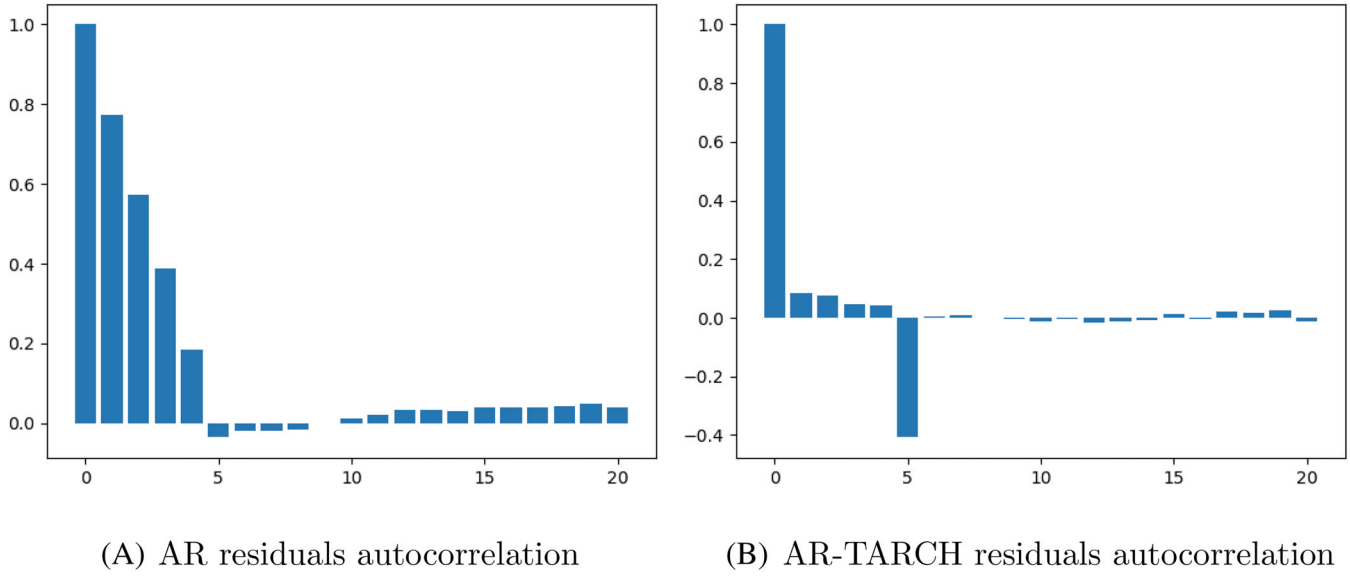


FIGURE 6 Autocorrelation functions of the time series of residuals ϵ_t for different non-linear models (A) AR; (B) AR-TARCH.

TABLE 4 In this table we report log-likelihood, Akaike information criterion (AIC) and Bayesian information criterion (BIC) for the models considered.

Model	Log-likelihood	AIC	BIC
AR-TARCH	848.31	−1684.61	−1642.73
TARCH	419.78	−829.55	−794.65
GARCH	389.13	−770.27	−742.34
SETAR	−88.29	−182.58	−201.62
AR	−2136.98	4279.97	4300.91

Note: We can see how the best model in terms of likelihood is given by the AR model with TARCH residuals (24). In order to verify that the higher log-likelihood is not a consequence of overfitting, we verify that the AIC and BIC for such model is the lowest among the models considered.

TABLE 5 Parameters of model (24), with $c = 0$, fitted to WTI factors.

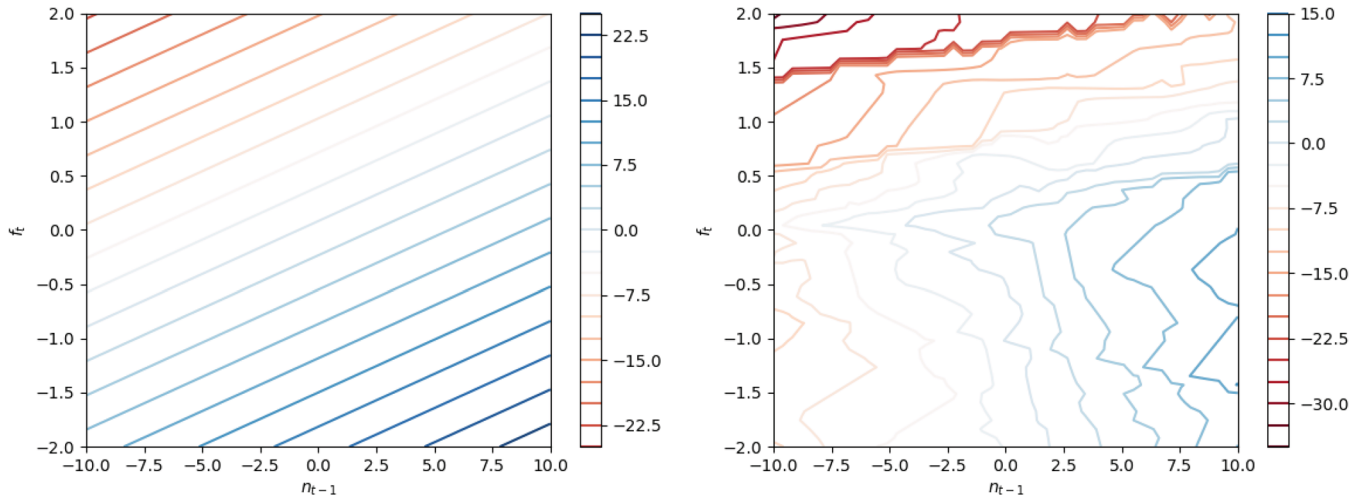
$\mu_f = 0.001$	$\Phi = 0.228$	$\omega = 0.002$
$\alpha = 0.200$	$\gamma = 0.010$	$\beta = 0.775$

the RL solution (Figure 7B). In fact, since $B^{(0)} = 0.014, B^{(1)} = -0.276$, both positive and negative factors predict negative price changes. For this reason it appears that the allocation strategy of the RL trader takes on less extreme values than that of the GP trader.

In Figure 8 we see the results of backtesting obtained by applying the policies of the three agents on the WTI time series from November 2018 to January 2019. In Figure 8A we see that allocation policy of the RL trader is more static than the one of the GP trader, in Figure 8B that it results in a larger cumulative wealth (21).

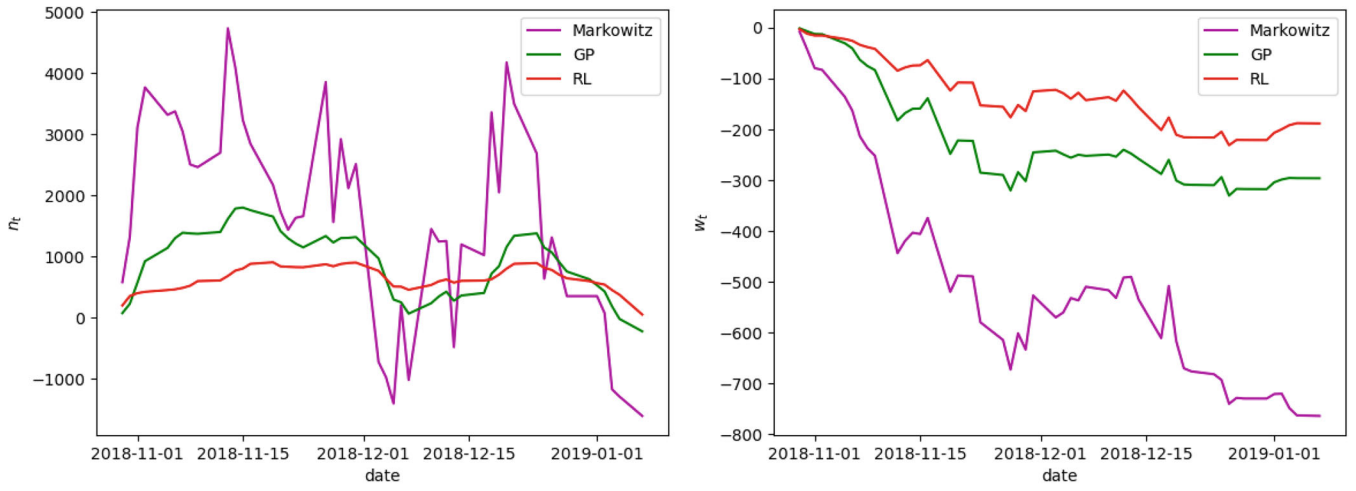
For a statistical comparison we generate data from the calibrated model.

In Figure 9 we compare final wealths according to Markowitz, GP and RL, produced on simulations generated by the GP model (12)–(13) as in Section 3, and on simulations generated by the AR-TARCH model. We observe that the final wealth depends both on the generating model used to produce the data and the choice of the strategy. In Figure 9A we see that when the paths are generated using the GP model (12)–(13), the distributions of GP and RL are almost indistinguishable, while the Markowitz one exhibits a larger variance and a smaller mean. In Figure 9B we see that when the paths are generated using the AR-TARCH model, RL performs better than GP producing a wealth distribution with higher mean and smaller variance, while both these strategies still largely outperform the Markowitz one both in terms of variance and mean.



(A) GP strategy: shares (B) RL strategy: shares

FIGURE 7 Policy (number of shares) of the GP trader (A) and RL trader (B) as a function of the current values of factor and shares.



(A) Shares (B) Wealth

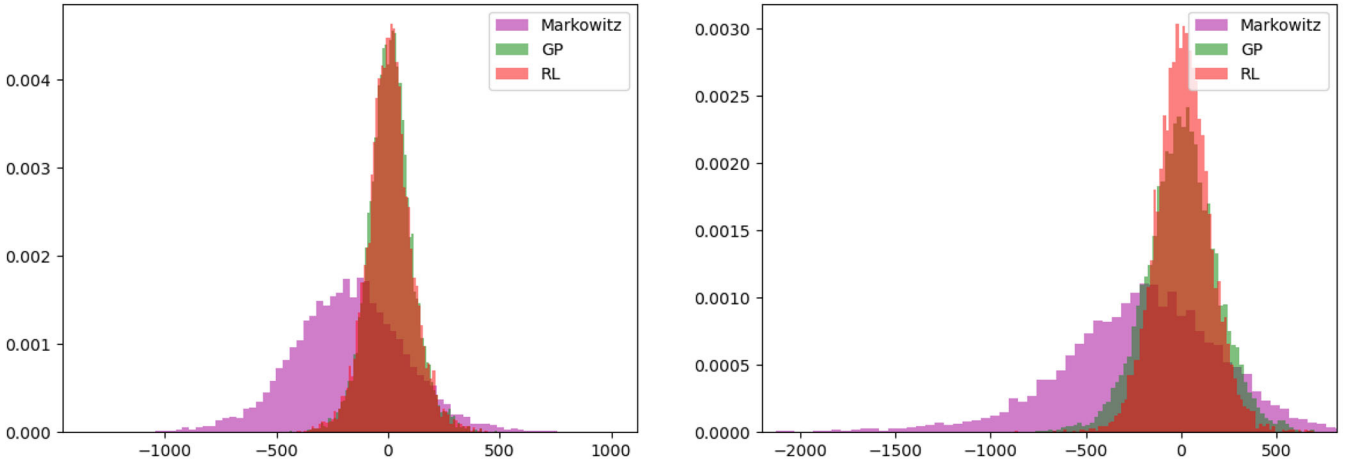
FIGURE 8 Evolution of the portfolio position according to Markowitz, GP and RL (A) and evolution of the corresponding wealth (B).

The statistics resulting from the histograms in Figure 9B is reported below

$$\mathbb{E}\left\{w_T^{GP}\right\} = 6.37, \quad \mathbb{S}d\left\{w_T^{GP}\right\} = 77.51,$$
$$\mathbb{E}\left\{w_T^{RL}\right\} = 11.52, \quad \mathbb{S}d\left\{w_T^{RL}\right\} = 94.69.$$

As we can see, the RL trader outperforms the GP trader in terms of wealth; this is confirmed by the one-sided Welch's t-test, which results in rejecting the null hypothesis of under performance.

$$H_0 : \mathbb{E}\left\{w_T^{RL}\right\} \leq \mathbb{E}\left\{w_T^{GP}\right\}$$
$$H_1 : \mathbb{E}\left\{w_T^{RL}\right\} > \mathbb{E}\left\{w_T^{GP}\right\}$$
$$t\text{-statistic} = 4.206, p < 10^{-3}$$



(A) Wealth distribution based simulations of the GP model

(B) Wealth distribution based simulations of the AR - TARCH model

FIGURE 9 Distribution of final wealths according to Markowitz, GP and RL on 10,000 simulations generated by the GP model (A), and on 10,000 simulations generated by the AR-TARCH model (B), calibrated to the WTI time series.

5 | CONCLUSION

We constructed an RL algorithm, combining the SARSA approach with neural networks, applicable to a continuous state-action space, to determine a trading strategy for a dynamic model that includes transaction costs.

We first considered a setting with linear factors and quadratic transaction costs, where an analytical optimal solution is known from Reference 2, finding that the RL approach is able to recover the optimal strategy. Then, we considered a setting with more realistic, non-linear dynamics, where a theoretical optimal solution is not known and the RL approach can still be applied, since it only requires to be able to simulate the model. We provided statistical evidence that, on synthetic data generated from WTI spot prices, the RL agent outperforms a trader that linearizes the model to apply the theoretical optimal strategy.

The main advantage of using RL is its flexibility, as it works under any model dynamics. The main disadvantage is that, to train it in a satisfactory way, it needs a significant number of trials. The problem with simulating these trials is not only the computational burden, but also that, in this setting, the user needs to specify a parametric model to produce the data. We think that an important improvement on the methodology proposed here will be achieved when new simulated data can be produced in a non-parametric way, so that the approach becomes really "model-free". The use of signatures[32] and/or Generating Adversarial Network[33] are promising tools in this direction.

Our algorithm produces an approximation to the optimal value function by employing a modification of the SARSA method to consider a continuous state-action space. A promising different approach, used in a similar setting by Reference 9, is based on policy approximation methods.

The selection of the predicting factor is key for a successful use of our methodology. In this respect, the empirical analysis[27] points out that commodity volatility is strongly related with volatility in other markets, and in particular to stock market volatility. This may be taken into account by including an implied or realized volatility index as a factor.

ACKNOWLEDGMENTS

The authors gratefully acknowledges financial support from University of Rome Tor Vergata via the Project AIF–E83C22002610005. We thank Riccardo Cogo, Giovanni Nappi, Andrea Cosentini and Ranieri Dugo for valuable inputs, and the participants of the XXIII Workshop on Quantitative Finance, the EFMA Meeting 2022, the AMASES Conference 2022 for their feedback. We are very grateful to an anonymous reviewer for his/her careful reading and valuable suggestions.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

1. Merton RC. An intertemporal capital asset pricing model. *Econometrica*. 1973;41:867-887.
2. Gârleanu N, Pedersen LH. Dynamic trading with predictable returns and transaction costs. *J Financ*. 2013;68(6):2309-2340.
3. Sutton RS, Barto AG. *Reinforcement Learning: an Introduction*. The MIT Press; 2018.
4. Bertoluzzo F, Corazza M. Testing different reinforcement learning configurations for financial trading: introduction and applications. *Procedia Econ Finan*. 2012;3:68-77.
5. Kolm PN, Ritter G. Dynamic replication and hedging: a reinforcement learning approach. *J Finan Data Sci*. 2019;1(1):159-171.
6. Cao J, Chen J, Hull J, Poulos Z. Deep hedging of derivatives using reinforcement learning. *J Finan Data Sci*. 2021;3(1):10-27.
7. Ritter G. Machine learning for trading. Available at SSRN. 2017 3015609.
8. Munos R, Baird LC, Moore AW. Gradient descent approaches to neural-net-based solutions of the Hamilton-jacobi-bellman equation. Paper presented at: IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339), volume 3, IEEE. 1999 2152–2157.
9. Brini A, Tantari D. Deep reinforcement trading with predictable returns. *Phys A: Stat Mech Appl*. page arXiv:2104.14683. 2021.
10. Moody J, Saffell M. Learning to trade via direct reinforcement. *IEEE Trans Neural Netw*. 2001;12(4):875-889.
11. Zhang Z, Zohren S, Roberts S. Deep reinforcement learning for trading. *J Finan Data Sci*. 2020;2(2):25-40.
12. Buehler H, Gonon L, Teichmann J, Wood B. Deep hedging. *Quant Finan*. 2019;19(8):1271-1291.
13. Vittori E, Trapletti M, Restelli M. Option hedging with risk averse reinforcement learning. arXiv preprint arXiv:2010.12245. 2020.
14. Deng Y, Bao F, Kong Y, Ren Z, Dai Q. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Trans Neural Netw Learn Syst*. 2016;28(3):653-664.
15. Kakade S, Langford J. Approximately optimal approximate reinforcement learning. Paper presented at: Proc. 19th International Conference on Machine Learning. Citeseer. 2002.
16. Papini M, Pirotta M, Restelli M. Adaptive batch size for safe policy gradients. Paper presented at: The Thirty-First Annual Conference on Neural Information Processing Systems (NIPS). 2017.
17. Schulman J, Levine S, Abbeel P, Jordan M, Moritz P. Trust region policy optimization. Paper presented at: International Conference on Machine Learning, PMLR. 2015 1889–1897.
18. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347. 2017.
19. Deisenroth MP, Neumann G, Peters J, et al. A survey on policy search for robotics. *Found Trends Robot*. 2013;2(1-2):388-403.
20. Csáji BC, Monostori L. Value function based reinforcement learning in changing markovian environments. *J Mach Learn Res*. 2008;9(8):1679-1709.
21. Rummery GA, Niranjan M. *On-Line Q-Learning Using Connectionist Systems*. Vol 37. University of Cambridge, Department of Engineering; 1994.
22. Nichols BD, Dracopoulos DC. Application of newton's method to action selection in continuous state-and action-space reinforcement learning. *ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. D facto, Bruges (Belgium); 2014:23-25.
23. Endres SC, Sandrock C, Focke WW. A simplicial homology algorithm for lipschitz optimisation. *J Global Optim*. 2018;72(2):181-217.
24. Ramachandran P, Zoph B, Le QV. Searching for activation functions. arXiv preprint arXiv:1710.05941. 2017.
25. Kingma DP, Ba J. Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980. 2014.
26. Markowitz H. Portfolio selection. *J Finan*. 1952;7:77-91.
27. Christoffersen P, Lunde A, Olesen KV. Factor structure in commodity futures return and volatility. *J Finan Quant Anal*. 2019;54(3):1083-1115.
28. Tong H. *Threshold Models in Non-linear Time Series Analysis*. Vol 21. Springer Science & Business Media; 2012.
29. Zakoian JM. *Threshold Heteroskedastic Models*. Institut National de la Statistique et des Etudes Economiques, Unpublished paper; 1991.
30. Akaike H. Information theory and an extension of the maximum likelihood principle. *Selected Papers of Hirotugu Akaike*. Springer; 1998:199-213.
31. Schwarz G. Estimating the dimension of a model. *Ann Stat*. 1978;6:461-464.
32. Bühler H, Horvath B, Lyons T, Arribas IP, Wood B. A Data-driven Market Simulator for Small Data Environments. *arXiv e-prints*, page arXiv:2006.14498. 2020.
33. Xu T, Wenliang LK, Munn M, Acciaio B. Cot-Gan: generating sequential data via causal optimal transport. arXiv preprint arXiv:2006.08571. 2020.

How to cite this article: Giorgi F, Herzel S, Pigato P. A reinforcement learning algorithm for trading commodities. *Appl Stochastic Models Bus Ind*. 2023;1-16. doi: 10.1002/asmb.2825