Expert Systems With Applications 103 (2018) 1–13

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Trading financial indices with reinforcement learning agents

Parag C. Pendharkar*, Patrick Cusatis

School of Business Administration, Pennsylvania State University at Harrisburg, 777 West Harrisburg Pike, Middletown, PA 17057, United States

ARTICLE INFO

Article history:
Received 16 July 2017
Revised 22 February 2018
Accepted 23 February 2018
Available online 6 March 2018

Keywords:
Reinforcement learning
Multi-agent systems
Markov decision process
Portfolio management

ABSTRACT

Intelligent agents are often used in professional portfolio management. The use of intelligent agents in personal retirement portfolio management is not investigated in the past. In this research, we consider a two-asset personal retirement portfolio and propose several reinforcement learning agents for trading portfolio assets. In particular, we design an on-policy SARSA (λ) and an off-policy Q(λ) discrete state and discrete action agents that maximize either portfolio returns or differential Sharpe ratios. Additionally, we design a temporal-difference learning, TD(λ), agent that uses a linear valuation function in discrete state and continuous action settings. Using two different two-asset portfolios, the first asset being the S&P 500 Index and the second asset being either a broad bond market index or a 10-year U.S. Treasury note (T-note), we test the performance of different agents on different holdout (test) samples. The results of our experiments indicate that the high-learning frequency (i.e., adaptive learning) TD(λ) agent consistently beats both the single asset stock and bond cumulative returns by a significant margin.

1. Introduction

There are extensive studies in the finance literature that deal with trading strategies. Early studies involve the use of filter rules to determine when to buy or sell a stock and conclude that a buy and hold strategy dominates trading strategies based on filters (Alexander, 1961; Fama & Blume, 1966). Other studies focus on momentum and contrarian strategies. Momentum trading strategies assume that winners will continue to be winners, and losers will continue to be losers. Jegadeesh and Titman (1993, 2001) find that momentum trading produces abnormal trading profits. Contrarian strategies seek to exploit market overreactions by buying losers and selling winners. A special case of the contrarian strategy is called pairs trading. Pairs trading involves identifying pairs of stocks that are close substitutes and buying the losers and selling the winners of the pairs. Do and Faff (2010, 2012) and Gatev, Goetzmann, and Geert Rouwenhorst (2006) find significant gains from using a pairs trading strategy.

In addition to trading strategies, intelligent systems are often used in dynamic control problems including financial investing (Almahdi & Yang, 2017). These dynamic investment decisions are often modeled as a Markov process. Peck and Yang (2011) use a Markov decision process (MDP) to model investment returns. In their model, investors in each period observe prior investment decisions. Based on this history, the investors make their future investment decisions. Zhang (2001) presents an investing model that identifies an optimal selling rule for stocks based on a finite-state Markov process. MDP has been applied to a variety of applications in finance (Bauerle & Rieder, 2011).

The opponents of MDP argue that it is rare that complex modern financial markets are truly Markovian (Nevmyvaka, Feng, & Kearns, 2006). Thus, in financial securities' trading and investment decision making, it is rare to have prior knowledge of a stochastic transition model and any knowledge of the reward function. In such cases, using the traditional MDP model poses a few challenges because a partially-observable environment is treated as a fully-observable environment (Nevmyvaka et al., 2006). These challenges can be handled by breaking down the MDP model into two stages. In the first stage, an algorithm can be designed to learn the state transition model and a reward function. In the second stage, the MDP can be solved to learn policy mapping. Alternately, the policy can be learned directly through trial and error. A variety of algorithms exist to learn policies using trial-and-error, *model-free* approaches. These algorithms fall into a broad category known as reinforcement learning (RL) or adaptive or approximate (Kara & Dogan, 2018; Li & Womer, 2015; Ohno, Boh, Nakade, & Tamura, 2016) dynamic programming. Over the last two decades, a number of studies have applied RL to financial trade execution (Bertoluzzo & Corazza, 2012; Bertsimas & Lo, 1998; Moody & Saffell, 2001; Moody et al., 1998; Nevmyvaka et al., 2006).

There is a special class of problems broadly defined as multi-armed bandit (casino slot machine) problems. In these problems, a player may observe a current state (earnings) and select an action (an arm to pull among many arms at a slot machine), but the

* Corresponding author.
E-mail addresses: pxp19@psu.edu (P.C. Pendharkar), pjc101@psu.edu (P. Cusatis).

https://doi.org/10.1016/j.eswa.2018.02.032
0957-4174/

future reward of the action and its probability distribution are unknown to the player and are independent of the player's actions. We assume that our individual two-asset investment problem falls into the multi-arm bandit class of problems, where the probability distribution of the reward of an investor's action is assumed to be unknown. Algorithms designed to solve multi-arm bandit problems have to balance the exploration–exploitation tradeoff (Kuleshov & Precup, 2000). Exploitation occurs when an investor uses trading strategies that provided the largest gains in the past, and exploration occurs when new actions are attempted to learn previously unknown strategies to maximize gains. Kuleshov and Precup (2000) argue that simple heuristics, such as ε-greedy and Boltzmann exploration, outperform other theoretically sound algorithms by significant margins. For the multi-arm bandit class of problems, the performance of algorithms is sensitive to the choice of parameters that manage exploration and exploitation tradeoffs, and these parameters must be chosen carefully to obtain superior results (Kuleshov & Precup, 2000).

Traditionally, researchers have used RL agents for professional portfolio management (Almahdi & Yang, 2017) and strategic foreign exchange asset allocations (Dempster & Leemans, 2006). The current research focuses on the application of RL agents for individual retirement portfolio management. Professional portfolio management (e.g., managing active mutual funds or hedge fund portfolios) involves different sets of tools and strategies. Professional portfolio managers have access to real-time information, can make trades at significant discounts and can make frequent trades at a fraction of a second (Bertsimas & Lo, 1998). Individual retirement portfolio management is a different problem because individuals are often restricted from making frequent trades by their brokerages. Additionally, information access and the number of assets in an individual portfolio are also limited. We assume a simple retirement portfolio containing two asset classes, one containing S&P 500 Index fund/ETF and another asset class containing either AGG Bond Index or 10 year US Treasury note, and illustrate that RL agents can be successfully used by individuals to manage their retirement portfolios. We propose and use a model-free RL agent to learn retirement portfolio trading strategies using major stock and bond indices/U.S. Treasury note (T-note) data. We assume that the first security in the retirement portfolio is a stock market security, which is an exchange trade fund (ETF) or an index mutual fund (IMF) that represents the return on the S&P 500 Index (S&P 500). The second security is a bond market security, which is either an ETF or IMF that mimics the return on the Barclays Capital U.S. Aggregate Bond Index (AGG) or a 10-year U.S. T-note. The investment objective is to determine a trading strategy that maximizes either portfolio returns or differential Sharpe ratios over long investment periods of 10 years or longer. We assume that the two-asset portfolio is reallocated exactly once per fixed-time trading period using information on the last trading day of the trading period. On this day, an investor uses the information on returns of the S&P 500 ETF and the bond assets (AGG ETF or T-note) to determine the portfolio allocation for the next trading period. Fixed trading periods in our research may be quarterly, semi-annual or annual. In their decision-making, an investor only considers whether previous trading period returns of the S&P 500 and the bond asset (AGG or T-note) are positive or negative, and does not consider the magnitude of these returns. We use the S&P 500 and the AGG (introduced in 1973) data for the years 1976–2016. For our experiments with T-note data, we use 46 years of data from years beginning in 1970 and ending in 2016. The use of T-notes allows us to obtain more finely grained return data (quarterly and semi-annual) over a longer time horizon, since, unlike the AGG, quarterly data is available. Thus, our experiments with the S&P 500 Index and the AGG portfolio assume annual trading. In contrast, our experiments with the S&P 500 and the T-note portfolio allow us to use three different trading periods: quarterly, semi-annual and annual.

We divide our datasets into two parts—training and holdout/test datasets. We learn trading strategies using the training dataset and apply them to the holdout dataset to monitor their performance. Since we use a portfolio of only two assets, the yearly return on our portfolio is a convex combination of the returns on the two assets. Thus, finding a trading strategy that uses two indices and beats the cumulative performances of both indices over a decade is no trivial task. If such a trading strategy can be found then it will be efficient on the risk-return efficient frontier since it will provide a higher return at a lower level of risk. Our RL algorithms use an $\varepsilon-$greedy strategy, and we experiment with different performance parameters to manage the exploration–exploitation tradeoff.

The rest of our paper is organized as follows: In the next section, we review some of the studies that applied RL for trading financial securities. In Section 3, we define the stochastic decision process model that applies to our research problem and introduce different reinforcement learning agents used in our research. In Section 4, we describe our data and report the results of our experiments. We provide our conclusions and propose directions for future work in Section 5.

2. Reinforcement learning applications for stock trade executions

RL is a type of learning that is used for sequential decision-making problems (Sutton & Barto, 1998). An RL agent recognizes different states and takes an action where it receives a feedback (reward) and then it learns to adjust its actions to maximize its future rewards. RL algorithms have been previously used in quantitative finance for optimal execution of trades (Almahdi & Yang, 2017). Part of the interest in the application of RL algorithms in finance is due to the application of automated agents that process real time high frequency microstructure data (millisecond time scale) to execute trades. Bertsimas and Lo (1998) study an application of RL (also known as adaptive dynamic programming) for trading large equity blocks over a fixed finite number of time periods so that the expected cost of executing trades is minimized. Bertsimas and Lo (1998) look at equity trading from institutional investors' point of view because these investors execute high volume trades over the course of several days. They define best execution as a strategy that unfolds over several days. Because current trading affects current equity prices which in turn affect future trading costs, trading strategies must adapt to changing market conditions. Naïve strategies, such as equally dividing the sale (or purchase) of a block of shares over a fixed time interval or selling (or purchasing) all shares at once on the first day, are generally not optimal, because equity purchases constantly impact equity prices. In their Monte Carlo experiments, Bertsimas and Lo (1998) find that the RL algorithm strategy saved between 25% and 40% in execution costs when compared to the naïve strategy of trading in equal-size lots. The primary limitation of this study is the assumption that the volume of each buy trade is still large enough to increase the price of the traded security, excluding any random noise in security prices. This assumption of an institutional investor influencing security prices implicitly assumes that the security markets are small because for large markets security prices are beyond the control of an individual investor (Hildenbrand and Kirman, 1988).

Nevmyvaka et al. (2006) use an RL algorithm for optimizing trade execution using 1.5 years of millisecond time-scale limit order data from companies that trade on the NASDAQ. In limit orders, buyers and sellers specify prices at which they will buy or sell a security. In these cases, trade-offs may arise due to speed of

trade execution and stock price updates. The focus of their study was short-term price execution, which can be predicted using the distribution of outstanding limit orders and the sequence of prices of already-executed trades. The authors make a case that most real-world traders deal with large diversified portfolios, and do not have time to deal with single stock and millisecond time-scale data. This makes the use of automated trading agents necessary. The problem formulation is to sell a fixed number of shares in a fixed time horizon. At any given time, the state of the system is given by a multi-tuple vector that included elapsed time, number of unsold shares at hand, number of limit orders on the books, and recent market activity in the stock. The RL agent's action is the limit order price at which to reposition remaining unsold shares of the stock. The selling of stock was mandatory over the fixed time and any unsold shares at the end of fixed time horizon are sold at prevailing market prices at the termination of the fixed time horizon. The authors use real-world data on three NASDAQ stocks and compare the performance of their *Q*-learning RL algorithm with a simple submit-and-leave strategy to show that the RL algorithms result in significant improvements

Many studies use Sharpe ratios as a measure of performance (Almahdi & Yang, 2017). For a series of portfolio returns, the Sharpe ratio is computed as the ratio of the average of these returns divided by their standard deviation. Higher levels of Sharpe ratios indicate allocations that are consistent with Markowitz (1959) mean-variance portfolio optimization problem. A differential Sharpe ratio is the first order term used in approximation of the Sharpe ratio, and is often used in on-line optimization of trading system performance.

Moody, Wu, Liao, and Saffell (1998) and Moody and Saffell (2001) use a recurrent RL algorithm to learn asset allocation systems based on simulated and 25 years of monthly data on the S&P 500 and U.S. Treasury bills. Moody and Saffell (2001) use additional intra-daily foreign exchange data for currency trading. The objective of both studies is to maximize differential Sharpe ratios, which maximize risk-adjusted returns by accounting for transaction costs. The differential Sharpe ratio allows for online optimization of trading performance. The downside of using Sharpe ratios is that it penalizes returns larger than a certain amount, weighs recent returns more than past returns, and represents an adaptive utility function. The Sharpe ratio does not distinguish between the upside and downside potential growth of a portfolio. Generally, the standard deviation of returns is minimized to increase the value of the Sharpe ratio.

Bertoluzzo and Corazza (2012) model *Q*-learning and Kernel-based RL algorithms for automatic financial trading. The authors use weekly (5 days) stock performance to compute Sharpe ratios and define a state. The RL agent actions are defined as buy, sell or hold. Using simulated and real-world data from three Italian stocks, the authors compare the performance of *Q*-learning and Kernel-based RL algorithms. Among their finds were: *Q*-learning algorithms perform better than Kernel-based RL algorithms, using weekly performance to define a state is a naïve choice, and Sharpe ratio as a performance measure suffers from several limitations.

The two performance criteria used in the literature are portfolio returns and differential Sharpe ratio. Maximizing each performance criterion serves a different purpose. Maximizing portfolio returns is simple and it represents a risk-neutral decision-making utility function. Maximizing Sharpe ratio is more complicated. Differential Sharpe ratio models assume an adaptive utility function whereby the decision-maker's risk aversion changes over time and is significantly impacted by recent returns instead of returns from the distant past (Moody et al., 1998). Recently, mean-variance portfolio analysis and metrics have been criticized for not being able to properly model stock market volatility (Moshe & Guy, 2015). This criticism makes maximizing portfolio returns a slightly more desirable metric both for its simplicity and interpretation.

Table 1
Reinforcement agent action set.

A	1	2	3	4	5
S&P 500 (%)	0	25	50	75	100
AGG or T-bill bond (%)	100	75	50	25	0

3. Stochastic decision process model and reinforcement learning agent learning algorithms

In our study, we consider that the unit of analysis is a retirement portfolio investor and not a financial institution. Our assumption ensures that an individual investor's transactions do not have any impact on market prices of portfolio assets because the trade size is small. This assumption makes our problem analogous to multi-arm bandit problems. Next, we consider a stochastic decision process model of a two-asset financial market process where investment decisions are taken at the last minute of the trading period (end of the quarter, half-year or day of the year). We assume that transaction costs and taxes are zero. This assumption is applicable in most retirement portfolios where brokerage/mutual fund companies allow for periodic reallocation of the portfolio without applying any transactions costs and investments grow tax-free. Some brokerage companies (e.g., Vanguard) impose asset reallocation restrictions and do not allow asset reallocations with frequencies of less than two months. As a result, we assume that the maximum frequency of trading is quarterly, and the minimum is yearly. Thus, the time variable t has a span of one quarter, a half year or a year. We consider two different two-asset portfolios: S&P 500 and AGG; and S&P 500 and T-note. We assume that the investment horizon is long-term (10 years or longer). For short-term investment horizons (such as 1–3 years), stocks may not be a viable investment due to their short-term volatility. Short-term investment horizons are too simplistic for retirement portfolios; therefore, we do not consider short-term bonds. We believe that the choice between medium-term bonds, such as the AGG index or the 10-year T-note, best represents an average retirement portfolio planning decision. As a result, we consider one of these two asset classes in our bond asset allocations.

The market state variable on the last minute of the trading period is two-tuple binary variable indicating whether the S&P 500 was non-negative (1) or negative (0) and whether the AGG or T-note was non-negative or negative. This gives us four states for our research problem $E = \{11, 01, 10, 00\}$. The first element in the set E indicates that both the S&P 500 and the AGG or T-note portfolio was non-negative for the period on the last trading day of the trading period (at the last minute). The actions of our agents are either discrete or continuous giving us two different types of RL agents.

Our discrete agent's action set has five actions shown in Table 1. The first action is to assign all portfolio allocation into the AGG or T-note and the last (fifth) action is a portfolio containing all money into the S&P 500 ETF. Between these two extremes are fixed diversified allocations containing some money assigned to the S&P 500 and remaining money assigned to the AGG or T-note. For benchmarking our RL agent results, we consider pure stock and bond portfolios and fixed diversified allocations as A2, A3, A4 portfolios, where "A" stands for action and the following number represents the column number in Table 1. That is, A2 is fixed portfolio allocation with 25% assets invested in S&P 500 and remaining 75% of portfolio assets invested in bonds.

In our research, the discrete RL agent only considers five discrete actions at the beginning of the next trading period. The state

at the end of the current trading period is determined, and the agent's action represents the portfolio allocation at the beginning of the next trading period.

The action set for a continuous agent is determined by parameter θ_1, where $0 \le \theta_1 \le 1$ represents next period's portfolio percentage allocation in the S&P 500. The remaining $(1-\theta_1)$ is allocated to the AGG or T-note. Let μ_θ represent average stock allocations over all the planning periods over which θ_1 values are computed; then $(1-\mu_\theta)$ will be the average bond allocations over the same planning periods. We call these two averages collectively the "all-time average" (ATA) and create a benchmarking portfolio using the ATA allocations. We report the ATA cumulative portfolio returns along with A2, A3, and A4 benchmarks proposed earlier.

For discrete agents, we use two different RL learning algorithms and two different reward criteria. For a continuous agent, we learn the value of parameter θ_1 using a linear regression-like online learning method. We describe our approaches in following sections.

3.1. Discrete action reinforcement learning algorithms

A reinforcement learning agent learns an optimal state-action value function Q^* for an unknown model. In our research, for four states and five actions, we create a matrix $Q \in \Re^{4\times5}$ that is initialized with random values. Thus, $Q(s,a)$ represents the Q-value for state s and action a. These random values are then updated by sampling new states and actions using a training dataset that provides a reward (reinforcement) for the selected action. We consider that this reward is represented as $r(s,a)$. There are two ways (algorithms) to update values in matrix Q via sampling. The first algorithm is known as an *off-policy* algorithm, where the update rule does not depend on the agent learning policy. The second algorithm is an *on-policy* algorithm, where the update rule depends on the agent learning policy. Among the popular on-policy and off-policy algorithms are SARSA and Q-Learning algorithms (Sutton & Barto, 1998).

SARSA is an on-policy algorithm and its name comes from fact that the updating rule considers a quintuple $Q(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ where s_t, and a_t are the current state and the action (at the end of the current trading period); r_t is the reward observed (next trading period), and s_{t+1}, a_{t+1} are the next state and action pair (at the end of the next trading period). In our research, we use a SARSA (λ) algorithm that uses eligibility traces. The variable λ refers to an eligibility trace. Eligibility trace methods bridge the gap between temporal difference and Monte Carlo methods for faster and reliable learning (Sutton & Barto, 1998). An eligibility trace, $e(s,a)$, discounts past state-action credits and assigns higher credit to rewards observed for current states and actions. The use of eligibility traces allows for efficient learning of reinforcement learning algorithms (Sutton & Barto, 1998). Eligibility traces can be either accumulating traces or replacing traces. There is a minor difference between these two kinds of eligibility traces, but replacing traces generally lead to improved results. If $e_t(s,a)$ is an eligibility state matrix at time t and s_t and a_t are actions chosen by the agent then replacing traces update the entire eligibility trace matrix using following formula:

$$e_{t+1}(s,a) = \begin{cases} 1 \ if \ s = s_t \ and \ a = a_t \\ \gamma\lambda e_t(s,a) \ Otherwise \end{cases} \tag{3.1}$$

The values of eligibility traces decay exponentially for states and actions that are not frequently visited. The eligibility trace variable λ controls the rate of decay.

We implement replacing traces in our algorithms. Algorithm 1 illustrates the SARSA (λ) procedure for updating values for matrix Q (Sutton & Barto, 1998). The parameter α in Algorithm 1 is the learning rate that is typically set at a value between 0 and 1. Low values of alpha represent slower learning and high values of alpha represent faster learning (updates) of Q-values. The parameter γ is also assigned a value between 0 and 1, and is a discount factor that discounts future rewards and weighs current rewards more heavily. The exploration–exploitation tradeoff is achieved by an ε-greedy action policy that selects an action with highest Q-value most of the time. However, with a low probability value ε, an action is selected at random using uniform probability distribution.

Algorithm 1 The on-policy SARSA(λ) algorithm.

```
Initialize Q (s, a) randomly and e(s,a) = 0 for all s,a
Repeat (for each episode):
    t ← 0
    Initialize s_t
    Choose a_t from s_t using policy derived from Q(ε−greedy)
    Repeat (for each step of episode)
        Take action a_t, observe r_t and s_{t+1}
        Choose a_{t+1} from s_{t+1} using policy derived from Q(ε−greedy)
        δ ← r_t + γQ(s_{t+1}, a_{t+1}) − Q(s_t,a_t)
        e(s_t,a_t) ← 1
        For all s,a:
            Q(s, a) ← Q(s,a) + αδe(s, a)
            e(s, a) ← γλe(s, a)
        End For
        s_t ← s_{t+1}; a_t ← a_{t+1};
        t ← t + 1
    until s_t is terminal state
```

Algorithm 2 The off-policy $Q(\lambda)$ algorithm.

```
Initialize Q(s, a) randomly and e(s,a) = 0 for all s,a
Repeat (for each episode):
    t ← 0
    Initialize s_t
    Choose a_t from s_t using policy derived from Q(ε−greedy)
    Repeat (for each step of episode)
        Take action a_t, observe r_t and s_{t+1}
        Choose a_{t+1} from s_{t+1} using policy derived from Q(ε−greedy)
        a* ← arg max_b Q(s_{t+1}, b) (if a_{t+1} ties for the max, then a* ← a_{t+1})
        δ ← r_t + γQ(s_{t+1}, a* ) − Q(s_t,a_t)
        e(s_t,a_t) ← 1
        For all s,a:
            Q(s, a) ← Q(s, a) + αδe(s, a)
            If a_{t+1} = a*, then e(s, a) ← γλe(s, a)
            else e(s, a) ← 0
        End For
        s_t ← s_{t+1}; a_t ← a_{t+1};
        t ← t + 1
    until s_t is terminal state
```

Algorithm 2 illustrates the off-policy $Q(\lambda)$-learning algorithm. In this algorithm, the Q-value update rule always picks maximum Q-values for actions in the next state.

3.2. Discrete agent reward criteria

We choose two different reward criteria for our discrete agents. The first criterion is that of maximizing returns. Here, given the current state (s_t) at the end of the current trading period and action (a_t), the allocation for next trading period, the reward (r_t) is computed at the end of the next trading period based on action a_t. As an example, assume an annual trading period for a two-asset S&P 500 and AGG portfolio and the S&P 500 return was negative and the AGG return was non-negative, as it was at the end of 2000, giving $s_t = 01$. Further assume that the RL agent, using its policy and Q-matrix values, selects an action $a_t = 2$ (see Table 1 for allocation). Since the S&P 500 and the AGG returns for year 2001 were – 11.89% and 8.44%, respectively, the portfolio return (R_t) and

reward (r_t) for the agent will be:

$$r_t = R_t = 0.25 \times (-11.89\%) + 0.75 \times (8.44\%) = 3.35\%.$$

The maximizing portfolio returns criterion represents a risk neutral decision maker with multiplicative utility function (Moody et al., 1998).

Our second reward criterion is that of maximizing the differential Sharpe ratio (DSR). Details of differential Sharpe ratio can be found in Moody et al. (1998). The DSR represents an adaptive utility function that considers both the current wealth and past performance. The DSR for time t, DE_t, is computed using the following equation:

$$DE_t = \begin{cases} \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{\left(B_{t-1} - A_t^2\right)^{3/2}}, & if\ \left(B_{t-1} - A_t^2\right)^{3/2} \neq 0 \\ 0, & if\ \left(B_{t-1} - A_t^2\right)^{3/2} = 0 \end{cases}. \quad (3.2)$$

Where,

$$\begin{cases} A_t = A_{t-1} + \eta\Delta A_t = A_{t-1} + \eta(R_t - A_{t-1}) \\ B_t = B_{t-1} + \eta\Delta B_t = B_{t-1} + \eta\left(R_t^2 - B_{t-1}\right) \end{cases}. \quad (3.3)$$

The variables A_t and B_t are the first and second moments of portfolio returns' distribution. The values of $A_0 = B_0$ are initialized at 0, parameter η controls the magnitude of influence of return R_t on the Sharpe ratio. The variable η smoothes the fluctuations out and is often determined by assuming a forgetting time scale for the stock market. Moody et al. (1998) recommend that the value of this variable should be the inverse of the forgetting time scale. Since the objective of our research is to design a portfolio that performs well over a period of a decade or more, we assume that the forgetting time scale for the stock market is a decade, and we choose a value of $\eta = 0.1$. The return R_t is computed using the method described above when using the first criterion of maximizing returns. The RL agent reward (r_t), when using the second criterion, is the value of DE_t.

3.3. Continuous action reinforcement learning agent

For our two-asset portfolio, the continuous RL agent attempts to learn to predict portfolio returns by learning a parameter vector $\boldsymbol{\theta} = (\theta_1, \theta_2)^T$ for an online regression based linear valuation model. The first component of the vector, $\theta_1 \in [0,1]$, is a regression parameter representing the percentage of portfolio allocation to stocks. The bond percentage allocation in the portfolio can be computed from the first component as $(1 - \theta_1)$. The second component of the vector, $\theta_2 \in \Re$, accounts for the regression intercept as described in the following paragraph.

Assuming that for each state a portfolio return is represented as y^E then true portfolio return for state E and period t is represented: $y_t^E = \theta_1^E R_t^S + (1 - \theta_1^E)R_t^B$, where R_t^S and R_t^B represent S&P 500 and AGG returns for period t and θ_1^E is the percentage of portfolio allocation to stocks. The RL agent attempts to learn a state dependent and time independent portfolio return (y_{RL}^E) prediction function as follows:

$$y_{RL}^E = \theta_1^E R_t^S + \left(1 - \theta_1^E\right)R_t^B + K, \quad (3.4)$$

where K is the regression intercept. Eq. (3.4), after some rearrangement of terms, and introducing a variable $\theta_2^E = R_t^B + K$ can be written as the following linear regression model:

$$y_{RL}^E = \theta_1^E\left(R_t^S - R_t^B\right) + \theta_2^E. \quad (3.5)$$

The continuous RL learning agent learns parameter vectors $\boldsymbol{\theta^E}$ for each state E, so that the parameter vector minimizes the following mean square error (MSE) for each state.

$$MSE\left(\boldsymbol{\theta}^E\right) = \sum_{t,E}\left(y_t^E - y_{RL}^E\right)^2. \quad (3.6)$$

Algorithm 3 On-line gradient descent TD (λ) for learning continuous agent actions.

Initialize $\boldsymbol{\theta} = (\theta_1, \theta_2)^T$ arbitrarily
Repeat (for each episode):
  $\boldsymbol{e} = (e_1, e_2)^T = (0,0)^T$
  $s \leftarrow$ initial state of episode
  Repeat (for each step of episode):
    $a \leftarrow$ action given by policy π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma\ V(s') - V(s)$
    $\boldsymbol{e} \leftarrow \gamma\lambda\boldsymbol{e} + \nabla_{\boldsymbol{\theta}}V(s)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\ \boldsymbol{e}$
    if $(\theta_1 > 1)$ then $\theta_1 = 1$
    if $(\theta_1 < 0)$ then $\theta_1 = 0$
    $s \leftarrow s'$
  until s is terminal state

The MSE minimizing procedure requires a special consideration so that θ_1 must take a value between zero and one as it represents the stock asset portfolio allocation (regression parameter). The intercept related variable θ_2 is allowed to vary freely. We use a special version of online gradient descent algorithm *TD* (λ) (Sutton & Barto, 1998) to learn parameter vector $\boldsymbol{\theta^E}$. Algorithm 3 illustrates our procedure, for a state $s \in E$, $V(s) = y_{RL}^E$ from Eq. (3.5); whereas, $\nabla_{\boldsymbol{\theta}} = (R_t^S - R_t^B,\ 1)^T$. The policy π in the Algorithm 3 is an $\varepsilon-$greedy action policy that selects a current value of θ_1 most of the time. However to manage exploration–exploitation tradeoff, with a low probability, ε, a random value of $\theta_1 \in [0,1]$ is selected using a uniform probability distribution.

Our Algorithm 3 has certain advantages and disadvantages. Two advantages are that the algorithm is computationally efficient, and quickly provides stock asset allocation from which bond allocation can be easily computed. However, the disadvantage of the algorithm is that it is specialized for two-asset portfolios, and will not work for portfolios with more than two assets. For portfolios containing more than two assets, linear value function approximations and computational intensive fixed point methods may be deployed (Parr, Li, Taylor, Painter-Wakefield, & Littman, 2008). Linear value functions are essentially regression functions that will predict a portfolio return for different states. Computing asset allocations from linear regression coefficients will still pose a challenge for multiple asset allocation portfolios. Multiple assets will have multiple degrees of freedom and may require larger datasets for reliable learning. Table 2 provides a summary of advantages and disadvantages of the three algorithms used in our research.

3.4. Agent learning frequency

In our study, we experiment with the agents' learning frequencies as well. Specifically, we experiment with two different learning frequencies. First, we introduce some notation. For discrete action agents, the data set for learning RL *Q*-matrices contains a 2-tuple vector $\boldsymbol{R}_t = (R_t^S, R_t^B)$. The first component is the S&P 500 (stock) return for trading period t and the second component is the AGG (bond) or T-note return for period t. The scalar $R_t = w_1 R_t^S + w_2 R_t^B$, defined and used earlier, represents the portfolio return with w_1 and w_2 as action (a_t) dependent percentage allocations to stocks and bonds determined from Table 1. Assuming that these returns are available for a total of $(k+n)$ periods, we can split the dataset into two disjoint training and test datasets. The training dataset contains the first k examples and the test dataset contains the remaining n examples. In supervised learning literature, it is common to use approximately 60% of total examples for the training dataset and 40% of the remaining examples for testing performance of the RL agent (Nevmyvaka et al., 2006). Fig. 1 illustrates such a setup. The knowledge learned by the RL (i.e., *Q*-matrix values) agent from the training dataset is used for

Table 2
Advantages and disadvantages of the three algorithms.

Advantages	Disadvantages
Algorithm 1. *On-policy SARSA (λ)* The on-policy algorithm and maintains better exploration and is more adaptive than Q-learning Algorithm 2.	Will always keep exploring even when best solution is found
Algorithm 2. *Off-policy Q(λ)* The off-policy algorithm that may accidentally find good solutions quickly.	Greedy search will not always explore all actions. It is highly sensitive to design parameters and may not always find good solutions.
Algorithm 3. *TD(λ)* Learns value functions directly.	Performance depends on value function learning algorithm (linear vs. non-linear)

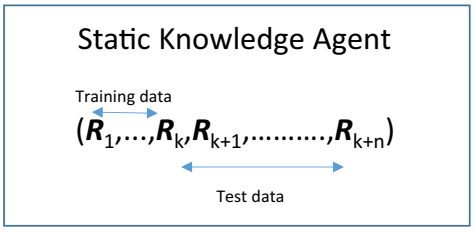

Fig. 1. A static knowledge agent.

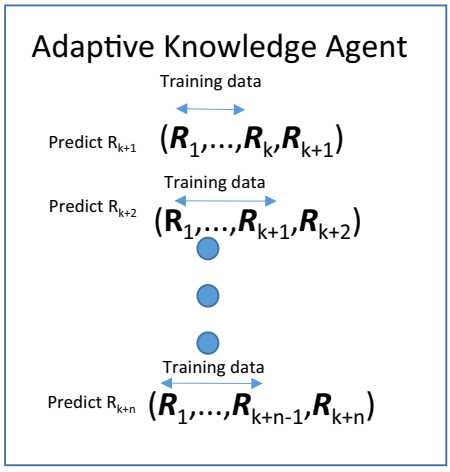

Fig. 2. An adaptive knowledge agent.

periodic portfolio allocations for the test dataset. For learning and test setup shown in Fig. 1, our agent's knowledge is static (for all test dataset periods) and, therefore, we call our RL agent a Static Knowledge Agent (SKA). Depending on reward criteria used by the agent, we can further classify the agent as either a return maximizing SKA (R-SKA) or a differential Sharpe ratio maximizing SKA (S-SKA). A continuous action learning agent learns the vector $\boldsymbol{\theta}=(\theta_1, \theta_2)^T$ from the training dataset and uses this knowledge on the test dataset. In a static setup with a fixed number of examples in the training and test datasets, we label our continuous action SKA as CA-SKA.

Another approach to learning *Q*-matrix values is to increase the frequency of learning. In this case, an agent learns from first k examples to predict performance for period $k+1$. However, for subsequent periods, it uses the training dataset from all available examples from all previous periods as the new training dataset. More specifically, for predicting allocation for period $k+t$, the RL agent will use all $(k+t-1)$ examples as the training dataset. Fig. 2 illustrates such training and test setup. Unlike SKA where the *Q*-matrix values remain static, the *Q*-matrix values in high-frequency learning setup may change every year due to additional training examples. Thus, in this new setup, RL agent's knowledge constantly adapts to additional information that becomes available in each new trading period. As a result, we call this high-frequency learning setup an Adaptive Knowledge Agent (AKA). As before, depending on reward criterion used by the agent, it may be classified as either an R-AKA, an S-AKA, or a CA-AKA agent.

Algorithms 1–3 all require generating an initial state at the beginning of each episode. For a given training dataset, using the same initial state may lead to memorization of the training dataset and poor generalization. One way to avoid this is to randomly generate the initial state (for each episode) beginning with the first period to period $k-4$. This gives each episode a random sequence of training examples. This randomization of the starting state minimizes memorization of training data sequences. Table 3 summarizes the RL agents used in our research and their computational complexity upper bound. For computing computational complexity, we assume that the number of episodes is constant across all problem types, $|E|$ represents the number of discrete states, $|A|$ represents the number of discrete actions, $z=k+n$ is the total number of periods in the overall dataset (inclusive of training and test samples). Assuming a constant number of states and actions, the computational complexity of all static agents may be further generalized to $O(z)$, and corresponding computational complexity for all adaptive agents may be generalized to $O(z^2)$, respectively.

4. Market data experiments and results

For our experiments, we use two different types of two-asset portfolios. The first portfolio consists of annual returns for the S&P 500 and the AGG. The second portfolio consists of three different scenarios with quarterly, semi-annual and annual returns for S&P 500 and 10-year T-note assets. The complexity of the AGG value computation restricted us from obtaining quarterly and semi-annual returns, and we only use single trading period of one year for this portfolio.

For the holdout sample, we compare our agents to all fixed trading strategies shown in Table 1. In Table 1, first and last column allocations are single asset allocations. The remaining allocations are A2, A3, and A4 described earlier. Additionally, we also compare our results with a theoretical ceiling (upper bound) and the ATA allocation. The ceiling for holdout samples is computed by allocating all portfolio assets, at the beginning of trading time period t, to an asset that maximizes (R_t^S, R_t^B) at the end of that trading period.

Including several agents, ceiling and fixed trading strategies lead to cluttered graphs. So we divide our graphs into two parts. In the first part, we include portfolio growth graphs for on-policy agents (SARSA (λ) learning), continuous action agents, one asset allocation stock and bond benchmarks; and a ceiling value of the portfolio. In the second part, we include off-policy (Q(λ) learning)

Table 3
The computational complexity upper bound of reinforcement learning agents.

Learning algorithm	Performance objective	Static agent label	Adaptive agent label	Computational complexity
Discrete state discrete actions agents				
On policy SARSA (λ)	Maximize returns	SKA	AKA	SKA $= O(z \times \|A\| \times \|E\|)$ AKA $= O(z^2 \times \|A\| \times \|E\|)$
Off policy Q (λ)	Maximize returns	Q-SKA	Q-AKA	Q-SKA $= O(z \times \|A\| \times \|E\|)$ Q-AKA $= O(z^2 \times \|A\| \times \|E\|)$
On policy SARSA (λ)	Maximize differential Sharpe ratio	S-SKA	S-AKA	S-SKA $= O(z \times \|A\| \times \|E\|)$ S-AKA $= O(z^2 \times \|A\| \times \|E\|)$
Off policy Q (λ)	Maximize differential Sharpe ratio	QS-SKA	QS-AKA	QS-SKA $= O(z \times \|A\| \times \|E\|)$ QS-AKA $= O(z^2 \times \|A\| \times \|E\|)$
Discrete state continuous actions agents				
On policy TD (λ)	Maximize returns	CA-SKA	CA-AKA	CA-SKA $= O(z)$ CA-AKA $= O(z^2)$

Table 4
Annual returns for S&P 500 Index and AGG Bond Index for years 1976–2016.

Year	S&P 500 Index annual returns (%)	AGG Bond Index annual returns (%)	Year	S&P 500 Index annual returns	AGG Bond Index annual returns
1976	23.84	15.60	1997	33.36%	9.65%
1977	−7.18	3.00	1998	28.58%	8.69%
1978	6.56	1.40	1999	21.04%	−0.82%
1979	18.44	1.90	2000	−9.10%	11.63%
1980	32.50	2.70	2001	−11.89%	8.44%
1981	−4.92	6.30	2002	−22.10%	10.26%
1982	21.55	32.60	2003	28.69%	4.10%
1983	22.56	8.40	2004	10.88%	4.34%
1984	6.27	15.15	2005	4.91%	2.43%
1985	31.73	22.11	2006	15.79%	4.33%
1986	18.67	15.26	2007	5.49%	6.97%
1987	5.25	2.76	2008	−37.00%	5.24%
1988	16.61	7.89	2009	26.46%	5.93%
1989	31.69	14.53	2010	15.06%	6.54%
1990	−3.10	8.96	2011	2.11%	7.84%
1991	30.47	16.00	2012	16.00%	4.21%
1992	7.62	7.40	2013	32.40%	−2.02%
1993	10.08	9.75	2014	13.69%	5.97%
1994	1.32	−2.92	2015	1.38%	0.55%
1995	37.58	18.47	2016	11.96%	2.65%
1996	22.96	3.63			

agents, fixed agent portfolio growths (A2, A3, A4), and single asset benchmarks. Our graphs also report numbers in certain cases. These numbers are reported as the final cumulative portfolio values for fixed value portfolios (A2, A3, A4, and the ATA), best performing agent and ceiling portfolio. In some cases, the magnitude of ceiling portfolio is so high that it compresses the growth of RL agent portfolios as it requires the use of a very large maximum number on the *Y*-axis. In such cases, we do not show the growth of ceiling portfolio and only report its final portfolio value on the graph. For the ATA benchmark, we report the stock and bond allocation averages in the text; and final cumulative portfolio values in graphs containing other related benchmarks (e.g., A2, A3, and A4).

For our preliminary experiments using the first two-asset portfolio, we obtain annual returns on the S&P 500 Index including dividends (S&P 500) and the Barclays Capital U.S. Aggregate Bond Index (AGG) for the years 1976–2016. The S&P 500 is a broad-based, value-weighted stock market index. The AGG is a composite bond index consisting of: U.S. Government Index; U.S. Credit Index; U.S. Mortgage Backed Securities Index (since 1986); and (since 1992) U.S. Asset Backed Securities Index. Table 4 lists the 41-year returns data used in our research.

To manage exploration–exploitation tradeoff, parameters for algorithms need to be carefully selected for experiments. For determining these RL agent parameters, we use some principles from the literature, where it is common to set the values of γ and λ slightly less than one and assign smaller values to the parameters ε, η and α, which are less than 0.15. Using these bounds, we perform a few experiments with different values for different parameters. For our experiments, we allow values of $\gamma = \lambda$ to vary in set {0.9, 0.95}, ε in set {0.01, 0.05}, and $\eta = \alpha$ in set {0.1, 0.05}. Using the dataset from Table 4, we use data from first 25 years as training dataset, and remaining data as test dataset to conduct 8 different training and test experiments for our agents. The objectives of final parameter set selection were two fold. First, the final parameters set should minimize the variance of training and test dataset final portfolio cumulative value across different agents, and second, the selected set of parameters should maximize both training and test sample cumulative average portfolio values. Tables 5–7 illustrate training and test data performance results. For training dataset, we only consider performances of static agents only because adaptive agents have multiple training periods and mixing cumulative performances will mess up standard deviations values due to unequal training periods between static and adaptive agents. However, for holdout samples, we consider performances of all agents. The last three columns of Table 5 contain average and standard deviations of cumulative portfolio values for a given set of parameters. The last column, ratio 1, is standardized ratio computed by dividing average portfolio value by standard deviation. The higher value of ratio 1 indicates that a set of parameters achieved higher training portfolio returns at lower standard deviation. A similar average portfolio value, standard deviation and ratio 2 was computed for test dataset portfolio value. Fig. 3 illustrates

Table 5
Training data cumulative portfolio values for different parameters and different static learning agents.

γ, λ	ε	η, α	SKA	S-SKA	CA-SKA	Q-SKA	QS-SKA	Average	Std. dev	Ratio 1
0.9	0.01	0.1	211,885	208,841	266,741	215,696	219,784	224589.4	23918.3	9.4
0.9	0.01	0.05	286,564	228,093	273,217	220,077	210,519	243694.00	33952.8	7.2
0.9	0.05	0.1	211,885	219,784	267,194	285,912	215,266	240008.2	34126.0	7.0
0.9	0.05	0.05	116,032	159,446	273,565	215,696	168,947	186737.2	60079.5	3.1
0.95	0.01	0.1	269,242	218,032	118,318	156,942	118,160	176138.8	66137.2	2.7
0.95	0.01	0.05	291,718	219,784	201,366	291,718	215,266	243970.4	44112.7	5.5
0.95	0.05	0.1	280,860	117,289	122,353	113,982	219,652	170827.2	75727.3	2.3
0.95	0.05	0.05	156,942	168,947	204,765	153,945	219,784	180876.6	29685.5	6.1

Table 6
Test data cumulative portfolio values for different parameters and different agents.

γ,λ	ε	η,α	SKA	AKA	S-SKA	S-AKA	CA-SKA	CA-AKA	Q-SKA	Q-AKA	QS-SKA	QS-AKA
0.9	0.01	0.1	24,633	17,656	25871.4	13635.1	26247.8	38070.5	23094.3	15973.5	24097.2	25783.8
0.9	0.01	0.05	24412.3	19546.1	21526.1	22252.1	26272.1	36425.3	22702.2	22478.1	25408.6	17204.6
0.9	0.05	0.1	24,633	19534.1	24097.2	15716.3	26241.6	40969.9	23282.5	15203.4	24917.6	15699.9
0.9	0.05	0.05	20960.3	14594.3	24027.3	17334.7	26269.3	40,522	23094.3	14791.4	20756.9	13523.6
0.95	0.01	0.1	27,015.3	16,232	24,536	17,408	24,185.6	46,196.9	23,752.9	17,563.8	21,597.9	13,943.5
0.95	0.01	0.05	22887.3	27590.9	24097.2	10895.9	26433.5	55588.3	22887.3	28641.2	24917.6	11024.8
0.95	0.05	0.1	24833.8	15364.1	22268.9	14,631	24366.4	46196.9	22357	16327.1	23796.7	22247.5
0.95	0.05	0.05	23752.9	17059.3	20756.9	24,263	26,468	55145.6	24,929	17,966	24097.2	33562.7

Table 7
Test data set row means and standardized ratio.

γ, λ	ε	η, α	Average	Standard dev.	Ratio 2
0.9	0.01	0.1	23822.75	5170.22	4.61
0.9	0.01	0.05	23757.24	5479.45	4.34
0.9	0.05	0.1	22851.38	8041.54	2.84
0.9	0.05	0.05	21587.41	7943.87	2.72
0.95	0.01	0.1	23243.19	9120.03	2.55
0.95	0.01	0.05	25496.40	12289.05	2.07
0.95	0.05	0.1	23238.94	8930.46	2.60
0.95	0.05	0.05	28398.55	11714.92	2.42

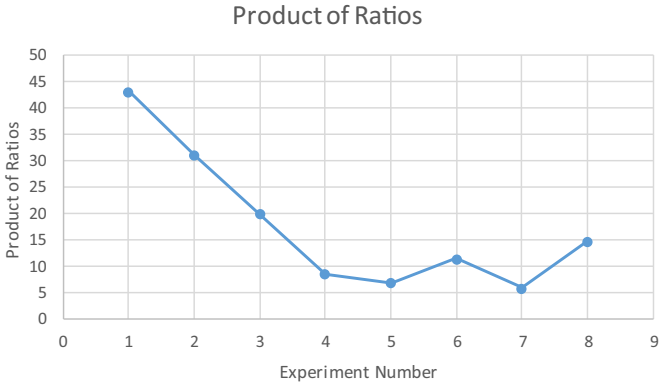

Fig. 3. Product of ratios.

the results of the product of these two ratios for each row in the tables.

A higher product indicates that a set of parameters achieved the highest value on the two performance criteria that we had set to select parameters for our experiments. Since the first experiment achieved the highest product value, we use the following values for our experiments: $\gamma = 0.9$, $\lambda = 0.9$, $\varepsilon = 0.01$, $\eta = 0.1$ and $\alpha = 0.1$. In all our procedures, these values were kept constant. Since it is customary to use approximately 60% of examples from a dataset as the training set and the remaining 40% as a test set, we create pairs of two training and test datasets. In the first pair, our training dataset contains the first 26 examples (years 1976–2001) and the test dataset contains the remaining 15 examples (years 2002–2016). In the second pair, our training dataset contains the first 25 examples (years 1976–2000) and the test dataset contains the remaining 16 examples (years 2001–2016). In the next section, we describe the results for each pair.

4.1. The first training (1976–2001) and test (2002–2016) data pair results

We divide this section into two sections. In the first section, we consider the performance of on-policy (i.e., SARSA) and continuous action agents and, in the second section, we consider the performance of an off-policy *Q*-learning agent.

4.1.1. Results of SARSA and continuous action agents

In this section, we report the results for both static and adaptive agents. Fig. 4 illustrates the results of the test dataset experiments. The results indicate that two adaptive agents (CA-AKA and S-AKA) beat both the AGG and the S&P 500 indices. The S&P 500 beats the AGG; and all other agents had final performances (cumulative returns) that were lower than the growth of the S&P 500 Index portfolio. The SARSA adaptive agent (AKA) had the worst performance (cumulative returns). However, this agent also had the lowest standard deviation (2759.4) and had the most steady investment growth over the test years. The best performing agent (CA-AKA) performance was less than 50% of the best theoretical performance indicated by the ceiling. The CA-AKA agent beats fixed allocation portfolio returns (see Fig. 5 for these values).

4.1.2. Results of Q-learning agent

Fig. 5 illustrates test results for an off-policy *Q*-learning agent for both maximizing performance (Q-SKA) and maximizing differential Sharpe ratio (QS-SKA) objectives. All *Q*-learning agents (static and dynamic) had poor performance on the test dataset. The S&P 500 Index returns performed the best. Static agents closely followed S&P 500 Index returns in both the training and test datasets. In case of the training dataset results, both static *Q*-learning agents learned to outperform the S&P 500 Index at the end of the training dataset year. For the ATA computation, the values of $\mu_\theta = 80.04\%$ and $(1 - \mu_\theta) = 19.96\%$.

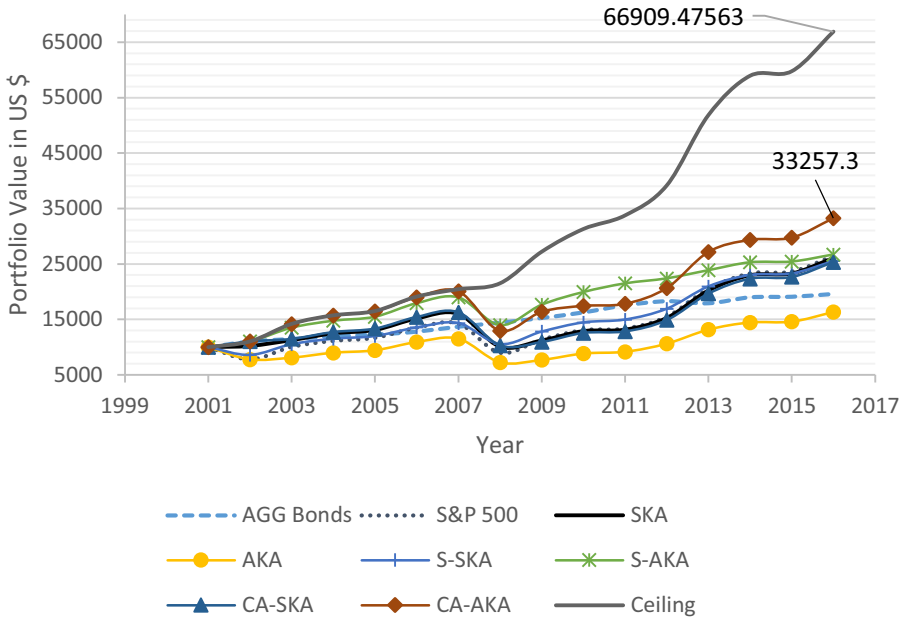

Fig. 4. Agents' performance on test dataset.

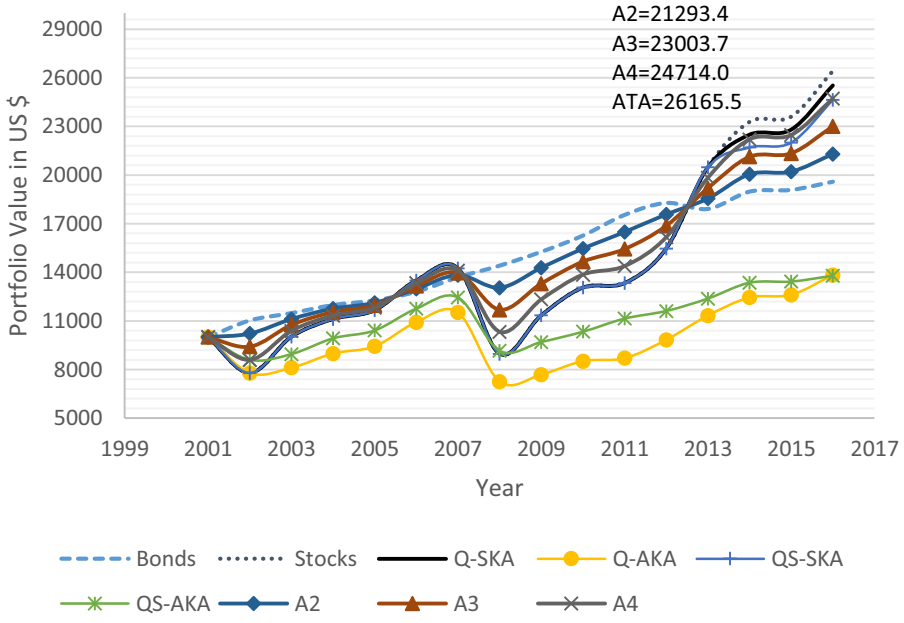

Fig. 5. Agents' performance on test dataset.

4.2. The second training (1976–2000) and test (2001–2016) data pair results

In our second set of experiments, the test dataset in represents a different set of circumstances than our first experiment. The following sections report the results of our experiments for the second training and test pair.

4.2.1. Results of SARSA and continuous action agents

Fig. 6 illustrates the results of the test dataset experiments. The results indicate that all three static agents (SKA, S-SKA, and CA-SKA) beat both index benchmarks (S&P 500 and AGG). The best performance, however, was obtained by the continuous action adaptive knowledge agent (CA-AKA). The other two adaptive agents (AKA and S-AKA) perform worse than the index benchmarks. The performance of the best performing CA-AKA agent was slightly higher than 50% of the theoretical ceiling value. The CA-AKA portfolio also beats fixed allocation portfolios (see Fig. 7 for results).

4.2.2. Results of Q-learning agent

Fig. 7 illustrates the results of Q-learning agents. Barring the adaptive knowledge agent (Q-AKA) with maximizing return objective, all of the other three agents perform higher than the benchmark indices with the Q-learning based adaptive agent maximizing differential Sharpe ratio performing the best. For the ATA computation, the values of $\mu_\theta = 79.79\%$ and $(1 - \mu_\theta) = 20.21\%$, respectively.

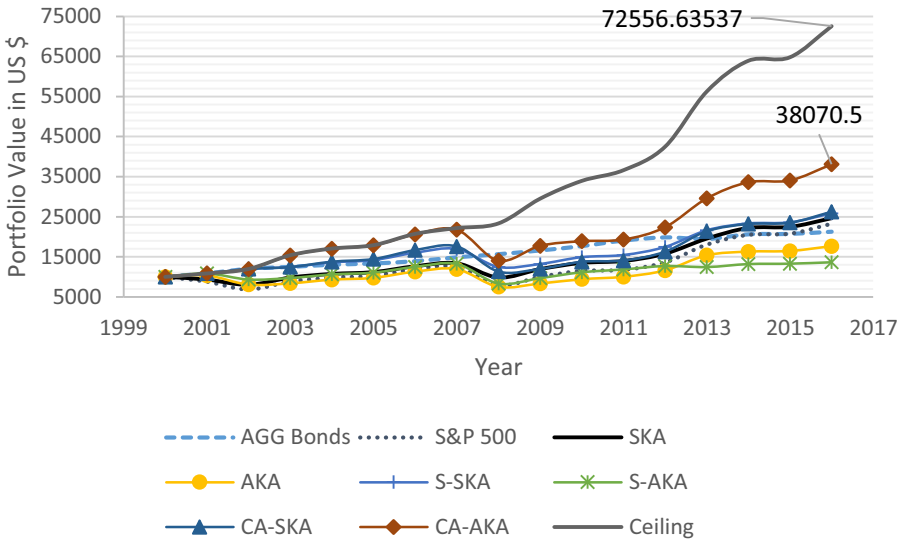

Fig. 6. Agents' performance on test dataset.

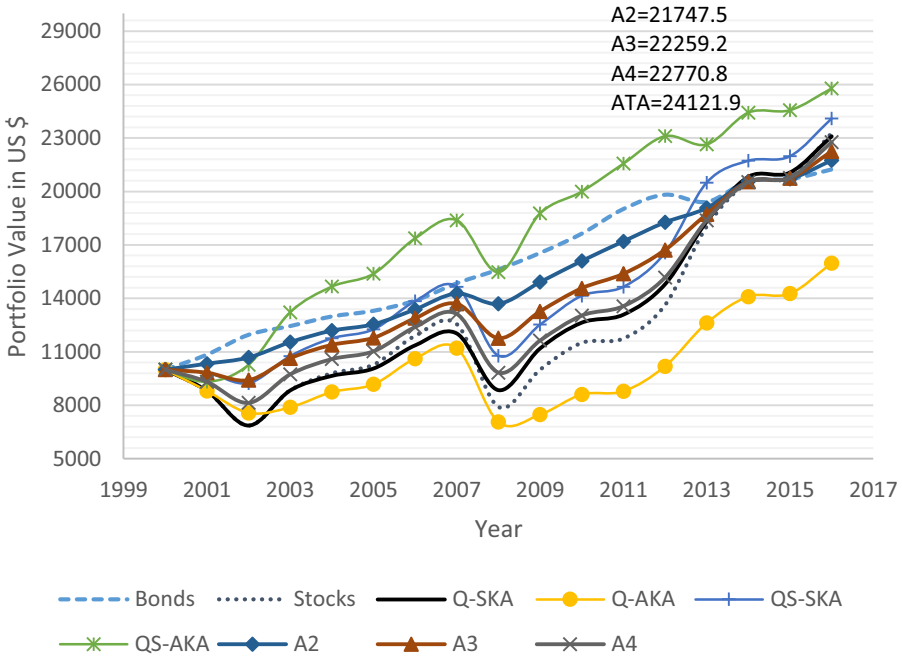

Fig. 7. Agents' performance on test dataset.

4.3. Discussion of results

Table 8 illustrates the test dataset allocations for both experiments for the best performing agent, CA-AKA. The year 2017 allocation is predicted based on the year 2016 state. For the year 2017, agents from both experiments recommend an allocation of approximately 98% of their portfolios in stocks and rest in bonds.

Portfolio allocation problems are notoriously difficult (Kao & Steuer, 2016), and beating both the S&P 500 and AGG indices over 14 or more years of cumulative returns is not a trivial task. One proof of such difficulty is that the majority of agents considered in our research cannot beat both indices consistently. The only exception is the CA-AKA agent. Our preliminary experiments indicate that the CA-AKA agent not only beats both indices in the test dataset for both experiments but also beats these indices by a very large magnitude. Adaptive agents in our research have large differences in performance, but the CA-AKA agent always performs well and the lowest performance is obtained by one of the discrete action adaptive agents. Static agents' performances were generally bound between the performance curves of the best performing and worst performing adaptive agents. These behaviors seem to indicate that, to create high-performance portfolios, allocations require regular (i.e., adaptive and not static knowledge-based) fine tunings (i.e., finer than the 25% discrete categories used in discrete action agents). On different performance objectives (maximize returns vs. maximize differential Sharpe ratio) for discrete action agents, on-policy agents maximizing Sharpe ratios seem to have slightly superior performance than static agents. Overall, adaptive learning for discrete action agents is highly unreliable. In our first two-asset portfolio, state 00 was missing in both training and test datasets. As a result, the Q-value matrix will contain initial random values for these states. However, all states were present in our second two-asset portfolio data analysis.

Table 8
The test dataset portfolio allocations for the CA-AKA agent.

Year	First experiment		Second experiment	
	S&P 500 Allocation	AGG Allocation	S&P 500 Allocation (%)	AGG Allocation (%)
2001	–	–	0.61	99.39
2002	0.25%	99.75%	0.00	100.00
2003	100.00%	0.00%	100.00	0.00
2004	99.72%	0.28%	99.22	0.78
2005	97.16%	2.84%	97.46	2.54
2006	96.15%	3.85%	96.81	3.19
2007	94.72%	5.28%	93.06	6.94
2008	96.40%	3.60%	96.60	3.40
2009	99.96%	0.04%	100.00	0.00
2010	0.09%	99.91%	0.00	100.00
2011	99.64%	0.36%	99.39	0.61
2012	99.49%	0.51%	99.52	0.48
2013	97.51%	2.49%	99.13	0.87
2014	26.60%	73.40%	99.93	0.07
2015	94.63%	5.37%	97.16	2.84
2016	98.36%	1.64%	97.83	2.17
2017	97.98%	2.02%	98.71	1.29

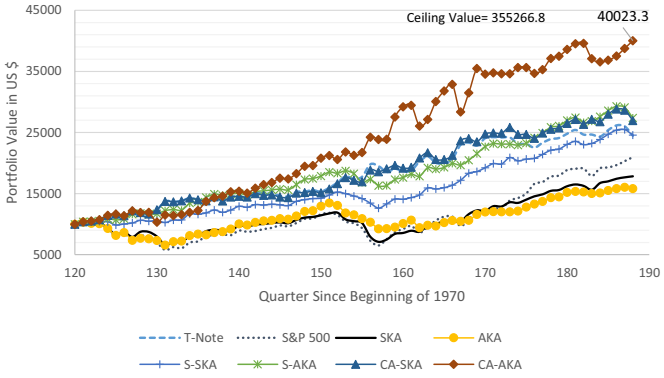

Fig. 8. Quarterly trading on-policy agent performance on test data.

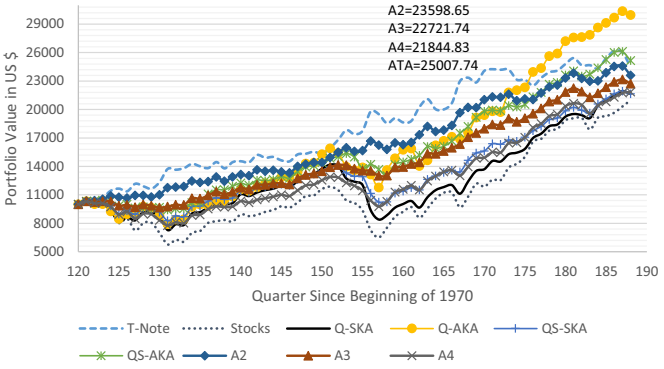

Fig. 9. Quarterly trading off-policy agent performance on test data.

4.4. Experiments with trading frequency on second portfolio

Our first two-asset portfolio with S&P 500 and AGG was limited in that we only had yearly data. To test whether increasing trading frequency to quarterly and semi-annual basis has any impact on portfolio returns, we construct a second two-asset portfolio based on returns from the S&P 500 Index including dividend reinvestments, and total returns on the 10-year T-note. The data consist of 47 years of quarterly returns from the beginning of 1970 to the end of 2016. We use the first 30 years of data as the training period and the next 17 years as the holdout/test period. When the training period is quarterly, our training data contains data for the first 120 quarters; and when it is semi-annual, our training data contains data for the first 60 semi-annual periods. Figs. 8 and 9 illustrate the quarterly trading performance of our agents, fixed allocation portfolios; and report theoretical ceiling and the ATA values. For the ATA computation, the values of $\mu_\theta = 50.22\%$ and $(1 - \mu_\theta) = 49.78\%$. The results indicate that the CA-AKA agent performed the best.

Figs. 10 and 11 report similar results for semi-annual trading and Figs. 12 and 13 report results for annual trading. For semi-annual trading, the ATA values are: $\mu_\theta = 22.55\%$ and $(1 - \mu_\theta) = 77.45\%$. The values for annual trading are: $\mu_\theta = 39.28\%$ and $(1 - \mu_\theta) = 60.72\%$.

The results of our experiments indicate that the CA-AKA agent performs the best in all three trading situations. Additionally, the ATA portfolio consistently beats all static allocation portfolios. To identify which trading situation is desirable, we compare the relative performances of CA-AKA agent with its ceiling values. For quarterly trading, the final portfolio value for CA-AKA agent is about 11.26% of its ceiling value. However, that value increases to 21.05% for semi-annual trading and 68.5% for annual trading. The magnitude of final CA-AKA portfolio value for annual trading also beats the final portfolio values for semi-annual and quarterly trading.

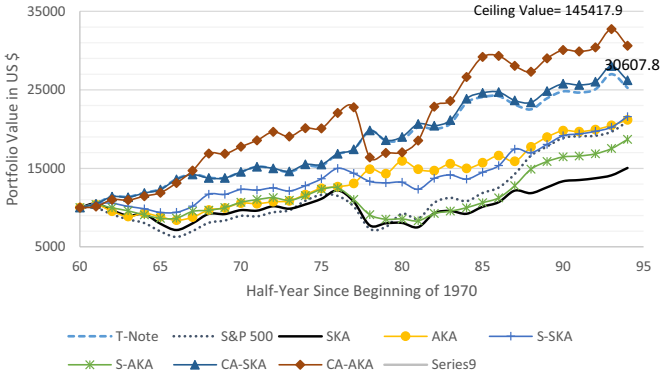

Fig. 10. Semi-annual trading on-policy agent performance on test data.

There are two reasons why annual trading frequency is better. The first reason is the selection of parameters. As mentioned earlier, the performance of RL agents is highly sensitive to the selection of parameters. In our parameter selection experiments, parameters were optimized for annual trading frequencies. As a result,

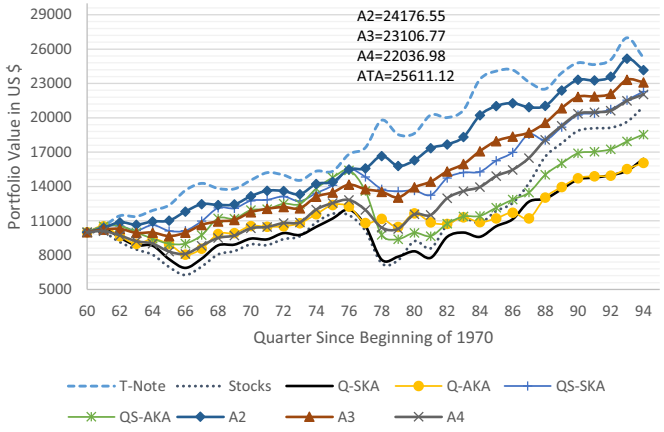

Fig. 11. Semi-annual trading off-policy agent performance on test data.

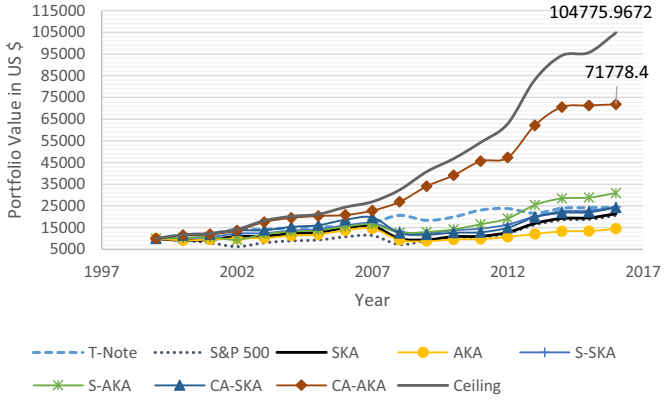

Fig. 12. Annual trading on-policy agent performance on test data.

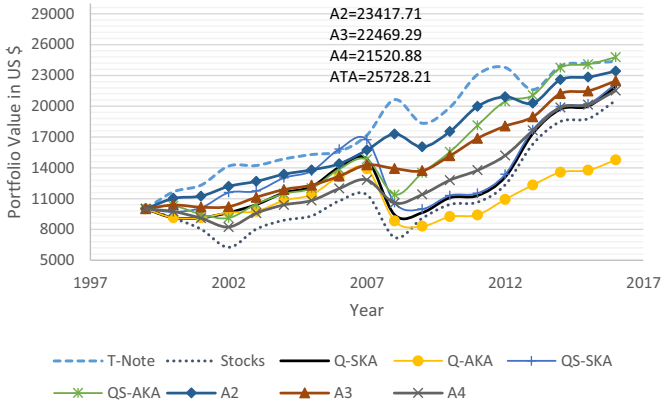

Fig. 13. Annual trading off-policy agent performance on test data.

RL agent performance degradation is expected when the trading frequency is increased. The second reason is that stock allocations and expected variance are impacted by semi-annual and quarterly trading frequencies. For a two asset portfolio, let π be the probability that an RL agent believes that stocks will outperform bonds and let a binary variable X take a value 1 when $\theta_1 \geq 0.5$, and a value 0 when $\theta_1 < 0.5$. The variable X will have a Binomial distribution with mean $n\pi$ and variance $n\pi\ (1-\pi)$. The values of n for test datasets are: 17 for annual trading frequency, 34 for semi-annual trading frequency and 68 for quarterly trading frequency. In our experiments, we compute the value of X, compute its average, and then equate it to a Binomial distribution mean to compute the value of π using following formula:

$$\pi = \frac{\sum_{i=1}^{n} X_i}{n^2}. \tag{4.1}$$

Table 9
Probability of stocks outperforming bonds in a trading period.

Trading period	π	$n\pi\ (1-\pi)$
Annual	0.024	0.401
Semi-annual	0.007	0.236
Quarterly	0.0073	0.493

We report the values of π and variance $n\pi\ (1-\pi)$ in Table 9.

The results indicate that increasing the trading frequency lowers the probability that an agent will believe that stocks will outperform bonds. This is due to the higher volatility of stocks for shorter trading periods. As shown in Table 9, the belief that stocks will outperform bonds is the highest for annual trading and the same for semi-annual and quarterly trading. The variance is the highest for quarterly trading and the lowest for semi-annual trading. The higher variance of the quarterly trading agent leads to either significantly better or significantly worse performance than that of the semi-annual trading agent. In our case, worse performance was observed.

In the end, the annual trading frequency is the best for maximizing overall portfolio returns. This is in part due to lower year-to-year volatility in annual stock returns relative that of semi-annual or quarterly returns. The annual trading agent takes the higher risk (i.e., higher π) than semi-annual and quarterly trading agents, and it also adapts more quickly than a semi-annual trading agent (i.e., higher standard deviation). An investor may also find annual trading more convenient and easier to execute.

5. Conclusions and directions for future research

In this paper, we proposed and tested several RL agents for trading financial indices in a personal retirement portfolio. In particular, we tested several on-policy and off-policy discrete state and discrete action agents that maximized either total returns or differential Sharpe ratios. Additionally, we tested on-policy discrete state and continuous action agents. We also experimented with the impact of RL agent learning and trading frequencies. The results of our experiments indicate that an adaptive continuous action agent consistently performs the best in predicting next period portfolio allocations. The learning frequency of an agent plays a role in determining the agent's predictive performance. However, generally discrete action agents are unable to fully utilize the benefits of higher learning frequency and, in several cases, have lower performance than static agents that only learn once from the training dataset. We also find that annual trading frequency gives excellent returns when compared to quarterly or semi-annual trading. This is an interesting observation because higher trading frequencies often mean learning on bigger training datasets where data is acquired more frequently. Clearly, we did not find that large training datasets are always beneficial. There are tradeoffs between the size of training datasets and learning algorithm. Our study indicates that adaptive agents with annual trading provide the best performance for our portfolio allocation problem.

The results of the continuous action adaptive knowledge agent are very promising. This agent beats the S&P 500 and AGG portfolio, as well as the S&P 500 and 10-year T-note portfolio by a large magnitude. While it will be very difficult to beat the performance of this agent, future researchers may consider model adjustments in an effort to improve the performance of the best performing agent in the current research. Among these possible adjustments are non-linear value functions and continuous state

space. Our continuous action space agent uses a linear value function. Non-linear value functions, including neural networks, may produce better valuations and predictions. Additionally, our state space throughout the research is discrete regardless of whether the S&P 500 and the AGG returns are non-negative. We do not consider the magnitude of these returns, which would require a continuous state space. Continuous state spaces and continuous action spaces are much more realistic, but these considerations significantly increase the complexity of agent learning. This complex learning may require large datasets for better generalization. Also, the exact magnitude of trading period returns may not be available on the last trading day of the period and would have to be estimated with some degree of error. Future researchers must consider these tradeoffs in their reinforcement learning agent designs.

References

Alexander, S. (1961). Price movements in speculative markets: Trends or random walks. *Industrial Management Review, 5*, 7–26.

Almahdi, S., & Yang, S. Y. (2017). An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. *Expert Systems with Applications, 87*, 267–279.

Bauerle, N., & Rieder, U. (2011). *Markov decision processes with applications to finance*. BerlinHeidelberg: Springer-Verlag.

Bertoluzzo, F., & Corazza, M. (2012). *Reinforcement learning for automatic financial trading: Introduction and some applications*. Department of Economics, Ca' Foscari University of Venice Working paper, 33.

Bertsimas, D., & Lo, A. W. (1998). Optimal control of execution costs. *Journal of Financial Markets, 1*, 1–50.

Dempster, M. A., & Leemans, V. (2006). An automated FX trading system using adaptive reinforcement learning. *Expert Systems with Applications, 30*(3), 543–552.

Do, B., & Faff, R. (2012). Are pairs trading profits robust to trading costs. *The Journal of Financial Research, 35*(2), 261–287.

Do, B., & Faff, R. (2010). Does simple pairs trading still work. *Financial Analysts Journal, 66*(4), 83–95.

Fama, E., & Blume, M. (1966). Filter rules and stock market trading profits. *Journal of Business, 39*, 226–241.

Gatev, E., Goetzmann, W. N., & Geert Rouwenhorst, K. (2006). Pairs trading: Performance of a relative-value arbitrage rule. *The Review of Financial Studies, 19*(3), 797.

Hildenbrand, W., & Kirman, A. P. (1988). *Equilibrium analysis*. Amsterdam, The Netherlands: Elsevier Science Publishers.

Jegadeesh, N., & Titman, S. (2001). Profitability of momentum strategies: An evaluation of alternative explanations. *The Journal of Finance, 56*(2), 699–720.

Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance, 48*(1), 65.

Kao, C., & Steuer, R. E. (2016). Value of information in portfolio selection, with a Taiwan stock market application illustration. *European Journal of Operational Research, 253*, 418–427.

Kara, A., & Dogan, I. (2018). Reinforcement learning approaches for specifying ordering policies of perishable inventory systems. *Expert Systems with Applications, 91*, 150–158.

Kuleshov, V., & Precup, D. (2000). Algorithms for the multi-armed bandit problem. *Journal of Machine Learning Research, 1*, 1–48.

Li, H., & Womer, N. K. (2015). Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming. *European Journal of Operational Research, 246*, 20–33.

Markowitz, H. (1959). *Portfolio selection: Efficient diversification of investments*. New York: Wiley.

Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks, 12*(4), 875–889.

Moody, J., Wu, L., Liao, Y., & Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting, 17*, 441–470.

Moshe, L., & Guy, K. (2015). Portfolio selection in a two-regime world. *European Journal of Operational Research, 242*(2), 514–524.

Nevmyvaka, Y., Feng, Y., & Kearns, M. (2006). Reinforcement learning for optimized trade execution. In *Proceedings of the twenty-third international conference on machine learning* (pp. 1–8).

Ohno, K., Boh, T., Nakade, K., & Tamura, T. (2016). New approximate dynamic programming algorithms for large-scale undiscounted Markov decision processes and their application to optimize a production and distribution system. *European Journal of Operational Research, 246*, 22–31.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., & Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the twenty-fifth international conference on machine learning* (p. 8). 2008.

Peck, J., & Yang, H. (2011). Investment cycles, strategic delay, and self-reversing cascades. *International Economic Review, 52*(1), 259.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Zhang, Q. (2001). Stock trading: An optimal selling rule. *SIAM Journal on Control and Optimization, 40*(1), 64–87.