What is the Value of the Cross-Sectional Approach to Deep Reinforcement Learning?

Amine Mohamed Aboussalah*[1], Ziyun Xu†[1], and Chi-Guhn Lee‡[1]

[1]Department of Mechanical and Industrial Engineering, University of Toronto

August 2021

Abstract

Reinforcement learning (RL) for dynamic asset allocation is an emerging field of study. Total return, the common performance metric, is useful for comparing algorithms but does not help us determine how close an RL algorithm is to an optimal solution. In real world financial applications, a bad decision could prove to be fatal. One of the key ideas of our work is to combine the two paradigms of the mean-variance optimization approach (Markowitz criteria) and the optimal capital growth approach (Kelly criteria) via the actor-critic approach. By using an actor-critic approach, we can balance optimization of risk and growth by configuring the actor to optimize the mean-variance while the critic is configured to maximize growth. We propose a Geometric Policy Score used by the critic to assess the quality of the actions taken by the actor. This could allow portfolio manager practitioners to better understand the investment RL policy. We present an extensive and in-depth study of RL algorithms for use in portfolio management (PM). We studied eight published policy-based RL algorithms which are preferred over value-based RL because they are better suited for continuous action spaces and are considered to be state of the art, Deterministic Policy Gradient (DPG), Stochastic Policy Gradients (SPG), Deep Deterministic Policy Gradient (DDPG), Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Twin Delayed Deep Deterministic Policy Gradient (TD3), Soft Actor Critic (SAC), and Evolution Strategies (ES) for Policy Optimization. We implemented all eight and we were able to modify all of them for PM but our initial testing determined that their performance was not satisfactory. Most algorithms showed difficulty converging during the training process due to the non-stationary and noisy nature of financial environments, along with other challenges. We selected the four most promising algorithms DPG, SPG, DDPG, PPO for further improvements. The modification of RL algorithms to finance required unconventional changes. We have developed a novel approach for encoding multi-type financial data in a way that is compatible with RL. We use a multi-channel convolutional neural network (CNN-RL) framework, where each channel corresponds to a specific type of data such as high-low-open-close prices and volumes. We also designed a reward function based on concepts such as alpha, beta, and diversification that are financially meaningful while still being learnable by RL. In addition, portfolio managers will typically use a blend of time series analysis and cross-sectional analysis before making a decision. We extend our approach to incorporate, for the first time, cross-sectional deep RL in addition to time series RL. Finally, we demonstrate the performance of the RL agents and benchmark them against commonly used passive and active trading strategies, such as the uniform buy-and-hold (UBAH) index and the dynamical multi-period Mean-Variance-Optimization (MVO) model.

Keywords: Reinforcement Learning; Deep Learning; Convolutional Neural Networks; Portfolio Allocation; Portfolio Optimization; Optimal Policies; Cross-Sectional Analysis; Computational Finance

*Corresponding author: amine.aboussalah@mail.utoronto.ca

†zy.xu@utoronto.ca

‡cglee@mie.utoronto.ca

1. Introduction

Portfolio management (PM) is the process of making decisions for the allocation of resources among a set of assets while satisfying financial constraints (such as cardinality, round-lot, asset class, etc.) in order to maximize the return, and it is considered to be one of the most important problems in the field of finance. There are vast amounts of data available that can be applied in making PM decisions and a multitude of theories about how to make these decisions. Because of this, many software solutions to assist the portfolio manager have been developed and are in use. While some solutions tend toward mechanical implementations of particular theories, more recently, the trend has been toward systems that learn directly from data using the latest advances in artificial intelligence (AI), in particular, machine learning (ML).

ML is a statistical learning method that automates the parametric model building. It is based on the idea that computer systems can learn from data, identify patterns, and make decisions with minimal human intervention. ML systems have been applied successfully to many real-world problems that previously required manually developed algorithms. (Li and Hoi 2014) formulated online portfolio selection as a sequential decision problem. An area of ML, called reinforcement learning (RL), is the most appropriate ML approach for the PM problem because of its ability to solve sequential decision-making problems, which is a key aspect of PM systems. Specifically, RL aims to teach software agents how to optimally perform a series of decisions by interacting with their environment so as to maximize some notion of cumulative return, and it is considered an active research area with many applications, including finance.

RL algorithms can be classified mainly into two categories: value-based and policy-based (Sutton and Barto 2017). The value-based RL (critic) relies on the notion of a *value-function*, which represents how good a state is for an agent to be in. It is equal to the expected total return for an agent starting from a given state s. In the past, there have been some attempts to use a value-based RL approach in the financial industry: a TD(λ) approach has been applied in finance (Van Roy 1999) and (Neuneier 1996) applied Q-Learning to optimize asset allocation decisions. Another latest paradigm, Deep Q-Network, designed at the outset to play Atari games (Mnih et al. 2015), encouraged the deep Q-trading system, which comprehends the Q-value function for the control problem (Wang et al. 2017). However, such value-function methods are less-than-ideal for online trading due to their inherently delayed feedback (Moody and Saffell 2001). Moreover, the Q-learning approach turns out to be unstable when presented with noisy data, and it is sensitive to the value function selection (Sutton and Barto 2017). In fact, with a value-based method, we have to work out first how to accurately compute a value function and then take the *max* operator over the actions to improve the policy. However, this maximization step can be itself prohibitively expensive. To circumvent the requirement to have this greedy *max* operation step, the policy-based RL (actor) class, also known as policy gradient (PG) or sometimes Direct Reinforcement, is preferred because it is better suited for continuous action spaces.

The actor class offers more flexibility to choose between different utility functions that can be directly optimized, such as profit, wealth, or risk-adjusted performance measures. A comparison study between Direct Reinforcement and Q-Learning methods for asset allocation was conducted by (Moody and Saffell 2001). Recently, other papers that use deep RL in portfolio management have been published. One of them, (Liang et al. 2018), implemented three state-of-art continuous control RL algorithms. All these algorithms are widely used in game-playing and robotics. Another, (Jiang et al. 2017), gave out a financial model-free RL framework to present a deep ML solution to the portfolio management issue using an online stochastic batch learning strategy. They pioneered the concept of Ensemble of Identical Independent Evaluators (EIIE) topology and the Portfolio-Vector Memory (PVM). Yet another, (Zarkias et al. 2019), ushered in an innovative price trailing formulation. In this formulation, the RL agent is directed to trail the price of an asset instead of directly predicting the future price. One more, (Zhengyao and Liang 2017), adopted a convolutional neural network (CNN) trading-based strategy. In this strategy, the historical prices of a class of financial assets from a cryptocurrency exchange were used as the basis to output the portfolio weights. The regime-switching recurrent reinforcement learning model presented by (Maringer and Ramtohul 2012) described its application to investment problems. (Deng et al. 2017) proposed a task-aware scheme to manage the vanishing/exploding gradient in recurrent RL. (Lu 2017) extends long short-term memory (LSTM) to manage the same deficiency. (Almahdi and Yang 2017) proposed a recurrent RL method with a coherent risk-adjusted-performance objective function to obtain both *buy* as well as *sell* signals and asset allocation weights. (Aboussalah and Lee 2020) developed a history-dependent policy gradient (PG) algorithm, called "Stacked Deep Dynamic Recurrent Reinforcement Learning (SDDRRL)," which allows continuous control over multiple assets. Stacking recurrent network structures allows the algorithm to maintain the flow of information through different time steps and balance long-term trends versus short-term losses.

There exists a *hybrid* RL method called Actor-Critic RL. Actor-critic RL aims at combining the advantages of the actor and the critic to guide the learning of the agent and reduce variance (Konda and Tsitsiklis 2000; Grondman et al. 2012). As suggested by its name, actor-critic RL comprises two agents: the actor and the critic. The actor determines the actions and forms the policy of the system. At every time step, the actor receives the current state as input and computes the agent's actions as output. The critic evaluates these actions. It receives the current state as well as the actor's actions as input and computes the discounted future reward as output. The key idea is to gradually adjust the policy parameters of the actor in a way that maximizes the reward predicted by the critic, which is also being adjusted. There are only a few studies employing actor-critic RL in financial markets (Fischer 2018). These works include (Li et al. 2007), who develops an RL agent to improve the forecast of stock returns obtained by an Elman network (Elman 1990), as well as (Bekiros 2010), who combine actor-critic RL with a fuzzy logic inference system. Unfortunately, neither of these works is compatible with the trading techniques re-

quired for PM systems. Actor-critic methods remain very little explored in the context of PM. Because of this, it is not clear how these methods perform compared to the only actor or critic based RL. Due to major advances in the field of RL, we consider that it is time to answer this question. In this paper, we first provide the most up-to-date comprehensive and detailed study of modern deep RL models for PM. For each RL model, we provide a detailed description of the underlying assumptions, highlight the current limitations of the existing models, and give a rational explanation as to why these deep RL algorithms have more difficulty to converge.

The first RL challenge is that financial environments are non-stationary and non-Markovian, plus the fact that we have to deal with noisy and partially observable data. Second, since financial data come under different types and frequencies, we have to deal with noisy, partially observable multi-type and multi-frequency data, and we also have many financial constraints to take into consideration. Third, we need to design a reward function that is financially meaningful. In this context, we want the agent to learn things such as alpha, beta, diversification, etc. Designing a reward function is not that difficult. The difficulty, however, comes when we try to design a reward function that encourages the behaviors we want while still being learnable. Finally, how to incorporate risk measures into RL agents is still a very challenging problem in practice.

One of the key ideas of our work is to combine the two paradigms of the mean-variance optimization approach (Markowitz 1952) and the optimal capital growth approach (Kelly 1956) via the actor-critic approach. By using an actor-critic approach, we can balance optimization of risk and growth by configuring the actor to minimize risk while the critic is configured to maximize growth. The 'policy function' which determines the actions (portfolio weights) given by the actor uses a mean-variance based approach. The 'value function' which evaluates these actions via the critic uses a growth rate based approach. (Cover 1991) proposed the universal portfolio algorithm which is an online portfolio selection algorithm that learns adaptively from historical data and maximizes the log-optimal growth rate in the long run. We used the same approach to derive our geometric policy score used by the critic function to evaluate the policy. The goal is to gradually adjust the policy parameters of the actor in a way that maximizes the growth predicted by the critic.

The contributions in this paper are as follows:

1. We leverage the Kelly theory to evaluate the policy of RL models in finance with more precision using growth than is possible using only total return by defining a Geometric Policy Score.
2. We leverage the Markowitz theory to define a multi-objective and risk-sensitive RL reward function that is financially meaningful.
3. We adapt multi-channel CNN receptive fields to filter multi-type financial data and treat data without assuming the Markov assumption.
4. To find out how the input asset classes move relative to each other at any given time point, we present in this paper for the first time a cross-sectional representation for deep reinforcement learning and show how it can be combined with time series analysis in one single RL agent.
5. The proposed architecture is modular in construction, scalable to realistic portfolio sizes, and therefore deployable for real-time trading platforms.

The remaining sections are organized as follows: section 2 describes the portfolio management model formulation, and section 3 introduces the theory of stochastic dynamic programming in more detail, which constitutes the cornerstone of most deep RL algorithms. Section 4 describes the state of the art of the commonly used deep RL models. Section 5 gives an overview of the main challenges that face RL in the context of finance. Section 6 shows the modifications we made to deep RL models to fit the PM problem. Section 7 summarizes the experimental results and demonstrates the performance of our methodology under a different class of RL algorithms which have been benchmarked against commonly used passive and active trading strategies, such as the uniform buy-and-hold (UBAH) index and the dynamical multi-period Mean-Variance-Optimization (MVO) model. Section 8 summarizes the lessons we have learned from the experimental part and give some recommendations to portfolio manager practitioners. Finally, section 9 concludes our paper.

2. Portfolio Management Model

We utilize the PM model described by (Breiman 1961) and (Algoet and Cover 1988).

Definition 2.1 (Portfolio Return Vector)**.** *Let us consider a portfolio of d assets. A portfolio vector $\mathbf{r} = \left(r^{(1)}, \dots r^{(d)}\right)^T \in \mathbb{R}_+^d$ is a vector of d nonnegative numbers, defining price ratios for a specified trading period. That is, the j-th component $r^{(j)} \geq 0$ of $\mathbf{r}$ indicates the ratio of the consecutive closing prices of asset j. This means $r^{(j)}$ is the factor by which the capital invested in the j-th asset increases or decreases during the trading period.*

We describe the portfolio allocation associated with the returns with a portfolio allocation vector.

Definition 2.2 (Portfolio Allocation Vector)**.** *A portfolio allocation is a vector $\mathbf{a} = \left(a^{(1)}, \dots a^{(d)}\right)^T \in \mathbb{R}^d$ whose j-th component $a^{(j)}$ of $\mathbf{a}$ represents the proportion of the investor's capital invested in asset j.*

The investor is permitted to diversify their capital at the commencement of each trading period in line with a portfolio vector $\mathbf{a} = \left(a^{(1)}, \dots a^{(d)}\right)^T$. Right through the paper, we presume that the portfolio vector $\mathbf{a}$ has nonnegative elements with $\sum_{j=1}^{d} a^{(j)} = 1$. The investment strategy is self-financing because $\sum_{j=1}^{d} a^{(j)} = 1$. Also, the consumption of capital is not included. The non-negativity of the elements of $\mathbf{a}$ implies that short-selling shares on margin are not allowed.

Definition 2.3 (Value of a portfolio)**.** *Let S_0 denote the investor's initial capital. Then, at the conclusion of the first trading period, the value S_1 of a portfolio (investor's wealth) becomes:*

$$S_1 = S_0 \sum_{j=1}^{d} a_1^{(j)} r_1^{(j)} = S_0 \langle \mathbf{a}_1, \mathbf{r}_1 \rangle, \tag{1}$$

where $\langle \cdot, \cdot \rangle$ denotes inner product.

The market evolution in time is defined by a series of market vectors $\mathbf{r}_1, \mathbf{r}_2, \ldots \in R_+^d$, where the j-th component $r_i^{(j)}$ of $\mathbf{r}_i$ indicates the amount derived after investing a unit capital in the j-th asset on the i-th trading period. For $k \leq t$, we abbreviate by $\mathbf{r}_k^t$ the sequence of market vectors $(\mathbf{r}_k, \ldots, \mathbf{r}_t)$ and indicate by Δ_d the simplex of all vectors $\mathbf{a} \in R_+^d$ with nonnegative elements summing up to one.

Definition 2.4 (Investment strategy)**.** *An investment strategy is a sequence* $\mathbf{A}$ *of functions (actions)*

$$\mathbf{a}_i : \left(R_+^d\right)^{i-1} \longrightarrow \Delta_d, \quad i = 1, 2, \ldots \tag{2}$$

so that $\mathbf{a}_i(\mathbf{r}_1^{i-1})$ *denotes the portfolio vector chosen by the investor in the i-th trading period, upon observing the past behavior of the market. We write* $\mathbf{a}(\mathbf{r}_1^{i-1}) = \mathbf{a}_i(\mathbf{r}_1^{i-1})$ *to ease the notation.*

The question is, what is the best portfolio. Since different agents have different preferences, there is no unique answer to this question.

Corollary 2.1 (Average growth rate)**.** *From the above definitions, we get by induction the value of the portfolio at time t:*

$$\begin{aligned} S_t &= S_0 \prod_{i=1}^{t} \left\langle \mathbf{a}(\mathbf{r}_1^{i-1}), \mathbf{r}_i \right\rangle \\ &= S_0 e^{\sum_{i=1}^{t} \ln \langle \mathbf{a}(\mathbf{r}_1^{i-1}), \mathbf{r}_i \rangle} = S_0 e^{t Q_t(\mathbf{A})}, \end{aligned} \tag{3}$$

where $Q_t(\mathbf{A})$ *denotes the average growth rate of the investment strategy* $\mathbf{A} = \{a_t\}_{t=1}^{+\infty}$*:*

$$Q_t(\mathbf{A}) = \frac{1}{t} \ln S_t = \frac{1}{t} \sum_{i=1}^{t} \ln \left\langle \mathbf{a}(\mathbf{r}_1^{i-1}), \mathbf{r}_i \right\rangle. \tag{4}$$

Remark 2.1 (Characterization of log-optimal portfolios)**.** Without transaction costs, the fundamental limits determined in (Algoet and Cover 1988) reveal that on stationary and ergodic markets, the so-called *log-optimal portfolio* $\mathbf{A}^* = \{\mathbf{a}^*(\cdot)\}$ is the best possible choice. More precisely, in trading period t, let $\mathbf{a}^*(\cdot)$ be such that:

$$\mathbf{a}_t^*(\mathbf{r}_1^{t-1}) = \arg\max_{\mathbf{a}(\cdot)} \mathbb{E}\left[\ln \left\langle \mathbf{a}(\mathbf{r}_1^{t-1}), \mathbf{r}_t \right\rangle \mid \mathbf{r}_1^{t-1} \right]. \tag{5}$$

If $S_t^* = S_t(\mathbf{A}^*)$ denotes the capital achieved by a log-optimal portfolio strategy $\mathbf{A}^*$ subsequent to t trading periods, then for a different investment strategy $\mathbf{A}$ with capital $S_t = S_t(\mathbf{A})$ and for any stationary and ergodic return process $\{\mathbf{r}_t\}_{-\infty}^{+\infty}$,

$$\liminf_{t \to \infty} \frac{1}{t} \ln \frac{S_t^*}{S_t} \geq 0 \quad \text{almost surely} \tag{6}$$

and

$$\lim_{t \to \infty} \frac{1}{t} \ln S_t^* = Q^* \quad \text{almost surely}, \tag{7}$$

where

$$Q^* = \mathbb{E}\left[\max_{\mathbf{a}(\cdot)} \mathbb{E}\left[\ln \langle \mathbf{a}(\mathbf{r}_{-\infty}^{-1}), \mathbf{r}_0 \rangle \mid \mathbf{r}_{-\infty}^{-1} \right] \right] \tag{8}$$

is the maximum possible growth rate of any investment strategy. Moreover, (Györfi and Schäfer 2003) and (Györfi et al. 2006; 2007) constructed empirical (data-dependent) log-optimal strategies for unknown distributions. Note that for first-order Markovian return process:

$$\begin{aligned} \mathbf{a}_t^*(\mathbf{r}_1^{t-1}) &\stackrel{\Delta}{=} \mathbf{a}_t^*(\mathbf{r}_{t-1}) \\ &= \arg\max_{\mathbf{a}(\cdot)} \mathbb{E}\left[\ln \langle \mathbf{a}(\mathbf{r}_{t-1}), \mathbf{r}_t \rangle \mid \mathbf{r}_{t-1} \right]. \end{aligned} \tag{9}$$

Although the traditional technique for solving (Equation (8)) is stochastic dynamic programming, this formalism has practical limitations, as we will show in section 3. (Equation (9)) can be solved efficiently but requires us to make the Markov assumption which may not be valid due to the path dependent nature of financial data. One way to relax this assumption is to use a historical data window size parameter τ which is optimized during the training process:

$$\begin{aligned} \mathbf{a}_t^*(\mathbf{r}_{t-\tau}^{t-1}) &\stackrel{\Delta}{=} \mathbf{a}_t^*(\mathbf{r}_{t-\tau}, \cdots, \mathbf{r}_{t-1}) \\ &= \arg\max_{\mathbf{a}(\cdot)} \mathbb{E}\left[\ln \langle \mathbf{a}(\mathbf{r}_{t-\tau}^{t-1}), \mathbf{r}_t \rangle \mid \mathbf{r}_{t-\tau}, \cdots, \mathbf{r}_{t-1} \right]. \end{aligned} \tag{10}$$

In this paper, we propose variants of policy-based RL approaches including Actor-Critic methods and demonstrate their effectiveness in solving large-scale portfolios (Section 7).

3. From Dynamic Programming to Reinforcement Learning

In problems related to sequential decision-making, an agent interacts with an environment by choosing a series of actions in order to conclude a specific task. In the course of this interaction, the agent gets a numerical reward from the environment. The goal of this interaction is to find the perfect strategy to maximize a certain measure of its performance. Over time, the environment evolves stochastically. This evolution may be influenced by the interaction with the agent. On this account, each action that the agent takes may influence the circumstances in which future decisions will be taken. For this reason, the agents must balance their desire to derive a large reward at a specific time by acting greedily and by evaluating the opportunities that will present themselves in the future. This learning approach is common enough to encompass a broad range of applications in diverse fields. Portfolio allocation problem is a classical noisy decision making example, where an investor must allocate their capital so as to maximize their long-term profits. The primary purpose of the following sections is to introduce the explanation that will be used in the remainder of this work and to recall the elementary concepts and outcomes of the discrete-time stochastic optimal control theory. This is the standard framework to examine problems related to sequential decisions in mathematical terms. Since our discussion will not be exhaustive, we refer the reader to the in-depth literature on the subject, such as (Bertsekas and Shreve 1978; Puterman 1994), (Bertsekas 1995), and (Necchi 2016).

3.1 Markov Decision Processes

A sequential decision-making problem under uncertainty can be defined at a specified time t, the agent (also known as the

decision-maker or controller) monitors the state s_t of the system (also known as the environment) and consequently performs an action a_t. Following this action, the agent obtains an immediate reward r_{t+1} (or suffers an immediate cost). The system, in turn, evolves to a new state in accordance with a probability distribution, which is based on the action the agent chooses. At the subsequent time $t+1$, the agent chooses a new action when presented with the new state of the system. The process repeats as a result. Using a Markov decision process, it is possible to model this interaction rigorously.

Definition 3.1 (Markov Decision Process)**.** *A Markov decision process (MDP) is a stochastic dynamical system specified by the tuple* $<\mathbb{S},\mathbb{A},\mathscr{P},\mathscr{R},\gamma>$*, where*

i) $(\mathbb{S},\mathscr{S})$ *is a measurable space, called the state space.*
ii) $(\mathbb{A},\mathscr{A})$ *is a measurable space, called the action space.*
iii) $\mathscr{P}:\mathbb{S}\times\mathbb{A}\times\mathscr{S}\to\mathbb{R}$ *is a Markov transition kernel, i.e.,*
   a) *for every* $s\in\mathbb{S}$ *and* $a\in\mathbb{A}$, $B\mapsto\mathscr{P}(s,a,B)$ *is a probability distribution over* $(\mathbb{S},\mathscr{S})$.
   b) *for every* $B\in\mathscr{S}$, $(s,a)\mapsto\mathscr{P}(s,a,B)$ *is a measurable function on* $\mathbb{S}\times\mathbb{A}$.
iv) $\mathscr{R}:\mathbb{S}\times\mathbb{A}\to\mathbb{R}$ *is a reward function.*
v) $\gamma\in(0,1)$ *is a discount factor.*

Ordinarily, the state space (and likewise the action space) will be either finite, that is $\mathbb{S}=\{s_1,\ldots,s_d\}$, or continuous, that is $\mathbb{S}\subseteq\mathbb{R}^{dim(\mathscr{S})}$. The kernel $\mathscr{P}$ explains the random evolution of the system: Assume that the system is in state s at time t. And suppose that the agent takes action a. Now, irrespective of the prior history of the system, the following equation gives the probability of finding the system in a state that belongs to $B\in\mathscr{S}$ at time $t+1$,

$$\mathscr{P}(s,a,B)=\mathbb{P}\left(S_{t+1}\in B|S_t=s,A_t=a\right) \tag{11}$$

The agent gains a stochastic reward R_{t+1} following this random transition. The reward function $\mathscr{R}(s,a)$ presents the expected reward gained when action a is taken in state s

$$\mathscr{R}(s,a)=\mathbb{E}\Big[R_{t+1}|S_t=s,A_t=a\Big] \tag{12}$$

This setting can be effortlessly generalized to the following cases:

1. The starting state of the system is a random variable.
2. The state of the system constrains the actions that an agent can choose.

The agent chooses its actions in accordance with a certain policy, at any time step.

Definition 3.2 (Policy)**.** *A policy is a function* $\pi:\mathbb{S}\times\mathscr{A}\to\mathbb{R}$ *such that*

i) for every $s\in\mathbb{S}$, $C\mapsto\pi(s,C)$ *is a probability distribution over* $(\mathbb{A},\mathscr{A})$.
ii) for every $C\in\mathscr{A}$, $s\mapsto\pi(s,C)$ *is a measurable function.*

Intuitively, a policy indicates a stochastic mapping from the ongoing state of the system to actions. Deterministic policies are a specific case of this general definition. We presumed that the agent's policy is stationary and is dependent only on the current state of the system. In this setting, we can often find an optimal policy that is dependent only on the present state. However, in path-dependent applications such as finance, we need to consider more general policies that depend on the entire history of the system. A policy π and an initial state $s_0\in\mathbb{S}$ determine a random state-action-reward series $\{(S_t,A_t,R_{t+1})\}_{t\geq 0}$ with values on $\mathbb{S}\times\mathbb{A}\times\mathbb{R}$ following the mechanism explained above.

Definition 3.3 (History)**.** *Given an initial state* $s_0\in\mathbb{S}$ *and a policy* π*, a history (or equivalently trajectory or roll-out) of the system is a random sequence* $H_\pi=\{(S_t,A_t)\}_{t\geq 0}$ *with values in* $\mathbb{S}\times\mathbb{A}$*, defined on some probability space* $(\Omega,\mathscr{F},\mathbb{P})$*, such that for* $t=0,1,\ldots$

$$\begin{cases} S_0=s_0 \\ A_t\sim\pi(S_t,\cdot) \\ S_{t+1}\sim\mathscr{P}(S_t,A_t,\cdot) \end{cases} \tag{13}$$

we will denote by $(\mathbb{H},\mathscr{H})$ *the measurable space of all possible histories.*

Moreover, we observe that

i) the state sequence $\{S_t\}_{t\geq 0}$ is a Markov process $<\mathbb{S},\mathscr{P}_\pi>$.
ii) the state-reward sequence $\{(S_t,R_{t+1})\}_{t\geq 0}$ is a Markov reward process $<\mathbb{S},\mathscr{P}_\pi,\mathscr{R}_\pi,\gamma>$.

where we denote

$$\begin{aligned} \mathscr{P}_\pi(s,s') &= \int_{\mathbb{A}}\pi(s,a)\mathscr{P}(s,a,s')da \\ \mathscr{R}_\pi(s) &= \int_{\mathbb{A}}\pi(s,a)\mathscr{R}(s,a)da \end{aligned} \tag{14}$$

The goal of the agent in stochastic optimal control is to find a strategy that ensures the agent's long-term performance is maximized. Some objective functions that are generally used in the infinite and finite horizon frameworks are discussed in the next sections.

3.2 Bellman's principle of optimality

In the risk-neutral environment, the agent is only keen on maximizing its reward, without taking into account the risk it needs to take on to accomplish it. In an infinite horizon task, the performance of the agent is typically measured either as the full discounted reward or as the average reward derived at each time step. Both these approaches are radically different from a theoretical as well as an algorithmic point of view. Therefore, they will always be treated separately.

In the discounted reward formulation, the agent's performance is measured as the expected return obtained following a specific policy.

Definition 3.4 (Return)**.** *The return is the total discounted reward obtained by the agent starting from time-step t*

$$G_t=\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \tag{15}$$

where $0<\gamma<1$ *is the discount factor.*

Discounting can be used in domains such as economics to represent the interest earned on rewards. In general, an action that generates an instant reward will be favored over one that generates the same reward a few steps into the future. Hence, discounting is used to model the trade-off between instant and delayed reward: If $\gamma = 0$, the agent chooses its actions in a myopic manner, while if $\gamma \to 1$, it acts in a far-sighted manner. Other probable reasons for discounting future rewards are there as well. The first reason is it is mathematically appropriate, as it avoids infinite returns, and it resolves many convergence problems. Another explanation is that it models the lack of certainty about the time ahead, which may not be fully represented. In reality, the discount factor could be perceived as the probability that the world environment does not stop at a specific time step.

Definition 3.5 (State-Value Function)**.** *The state-value function $V_\pi : \mathbb{S} \to \mathbb{R}$ is the expected return that can be obtained starting from a state and following policy π*

$$V_\pi(s) = \mathbb{E}_\pi\big[G_t | S_t = s\big] \tag{16}$$

where the subscript in $\mathbb{E}_\pi$ specifies that all the actions are chosen according to policy π. Also, the state-value function calculates how beneficial it is for the agent to be in a specific state and follow a certain policy. Likewise, we can initiate an action-value function that calculates how beneficial it is for the agent to be in a state, execute a specific action, and then follow the policy.

Definition 3.6 (State-Action Value Function)**.** *The action-value function $Q_\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the expected return that can be obtained starting from a state, taking an action and then following policy π*

$$Q_\pi(s,a) = \mathbb{E}_\pi\Big[G_t | S_t = s, A_t = a\Big] \tag{17}$$

We have the following relationship between V_π and Q_π

$$V_\pi(s) = \int_{\mathbb{A}} \pi(s,a) Q_\pi(s,a) da \tag{18}$$

Almost all value-based RL algorithms are designed to estimate these value functions and are typically based on the Bellman equations.

Proposition 3.1 (Bellman Equations)**.**

$$V_\pi(s) = \mathscr{R}_\pi(s) + \gamma T_\pi V_\pi(s) \tag{19}$$
$$Q_\pi(s,a) = \mathscr{R}(s,a) + \gamma T_a V_\pi(s) \tag{20}$$

where we denote by T_a (resp. T_π) the transition operator for action a (resp. for policy π)

$$\begin{aligned} T_a F(s) &= \mathbb{E}\Big[F(S_{t+1}) | S_t = s, A_t = a\Big] \\ &= \int_{\mathbb{S}} \mathscr{P}(s,a,s') F(s') ds' \end{aligned} \tag{21}$$

$$\begin{aligned} T_\pi F(s) &= \mathbb{E}_\pi\Big[F(S_{t+1}) | S_t = s\Big] \\ &= \int_{\mathbb{A}} \pi(s,a) \int_{\mathbb{S}} \mathscr{P}(s,a,s') F(s') ds' da \end{aligned} \tag{22}$$

If we introduce the Bellman operator B_π, defined as

$$B_\pi V_\pi(s) = \mathscr{R}_\pi(s) + \gamma T_\pi V_\pi(s) \tag{23}$$

Then (Equation (19)) can be written as a fixed-point equation

$$V_\pi(s) = B_\pi V_\pi(s) \tag{24}$$

which permits a unique solution by the contraction mapping theorem, based on some plain assumptions on the reward functions. An identical argument holds for (Equation (20)). The goal of the agent is to select a policy π_* that maximizes his expected return in all possible states. Such a policy is called an *optimal policy*.

Definition 3.7 (Optimal State-Value Function)**.** *The optimal state-value function $V_* : \mathbb{S} \to \mathbb{R}$ is the largest expected return that can be obtained starting from a state*

$$V_*(s) = \sup_\pi V_\pi(s) \tag{25}$$

Definition 3.8 (Optimal Action-Value Function)**.** *The optimal action-value function $Q_* : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the largest expected return that can be obtained starting from a state and taking an action*

$$Q_*(s,a) = \sup_\pi Q_\pi(s,a) \tag{26}$$

The optimal value functions satisfy the following Bellman equations.

Proposition 3.2 (Bellman Optimality Equations)**.**

$$V_*(s) = \sup_a Q_*(s,a) = \sup_a \Big\{\mathscr{R}(s,a) + \gamma T_a V_*(s)\Big\} \tag{27}$$

$$\begin{aligned} Q_*(s,a) &= \mathscr{R}(s,a) + \gamma T_a V_*(s) \\ &= \mathscr{R}(s,a) + \gamma \int_{\mathbb{S}} \mathscr{P}(s,a,s') \sup_{a'} Q_*(s',a') ds' \end{aligned} \tag{28}$$

Again, the contraction mapping theorem guarantees the presence and uniqueness of a solution, under some technical assumptions as equations (Equation (27)) and (Equation (28)) are fixed-point equations. An optimal policy can easily be derived beginning at the optimal value functions. Let us now define a partial ordering in the policy space

$$\pi_1 \succeq \pi_2 \Leftrightarrow V_{\pi_1}(s) \geq V_{\pi_2}(s) \quad \forall s \in \mathbb{S} \tag{29}$$

Then the optimal policy is defined such that $\forall \pi$, $\pi_* \succeq \pi$. We have the following results:

Theorem 3.1 (Optimal Policy)**.** *For any Markov decision process,*

i) *There exists an optimal policy π_* such that $\pi_* \succeq \pi$, $\forall \pi$.*
ii) $V_{\pi_*}(s) = V_*(s)$.
iii) $Q_{\pi_*}(s,a) = Q_*(s,a)$.

An optimal control policy can be found by following greedy actions with respect to Q_*. This corresponds to choosing the action that maximizes the action-value function in a given state

$$a_* = \arg\sup_{a \in \mathbb{A}} Q_*(s,a) \tag{30}$$

This policy is deterministic and only depends on the current state of the dynamical system.

However, despite its compact formulation, a stochastic dynamic programming approach is often compromised by several factors, such as the curse of dimensionality when too many state variables are involved. In addition, practical considerations such as transaction costs, financial constraints, non-time-additive utility functions, and other features of financial markets tend to create path dependencies in portfolio optimization problems, which increases the complexity of the problem tremendously. Such problems are difficult to solve in all but the simplest cases, with computational demands that become prohibitive as the number of time periods and assets increases. Even when simplifying assumptions are made, closed-form solutions are rarely available for realistic portfolio optimization problems beyond two or three assets (Haugh and Lo 2001). The alternative approach that might produce good approximate solutions to otherwise intractable portfolio optimization problems is deep RL, as discussed in the introduction.

4. State of the Art Actor-Critic RL Models

In this section, we present five deep RL algorithms considered to be state of the art in game playing: Deterministic Policy Gradient (DPG), Stochastic Policy Gradient (SPG), Deep Deterministic Policy Gradient (DDPG), Trust Region Policy Optimization (TRPO), and Proximal Policy Optimization (PPO).

4.1 Deterministic Policy Gradient (DPG)

The goal of RL is to find an optimal behavior strategy for the agent to obtain maximized rewards. Policy gradient (PG) methods target finding and optimizing a policy, which is a parameterized function with respect to θ, $\pi_\theta(a|s)$.

The objective function is defined as:

$$J(\theta) = \sum_{s\in S} d^\pi(s) V^\pi(s) = \sum_{s\in S} d^\pi(s) \sum_{a\in A} \pi_\theta(a|s) Q^\pi(s,a) \quad (31)$$

where $d^\pi(s)$ is the steady-state, or stationary distribution of the Markov chain generated under π_θ. It accounts for how often a particular state is visited, or how important a particular state is regarded. $V^\pi(s)$ is the state-value function, that is, the expected return of state s. $Q^\pi(s,a)$ is the state-action value function which assesses the expected return of a pair of state and action (s,a) as defined in section 3.

Theorem 4.1 (Policy Gradient Theorem)**.** *Using gradient ascent, we can move θ toward the direction suggested by the gradient $\nabla_\theta J(\theta)$ to find the best θ for π_θ that maximize the objective function. The PG theorem (Sutton and Barto 2017) proved that*

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \sum_{s\in S} d^\pi(s) \sum_{a\in A} Q^\pi(s,a)\pi_\theta(a|s) \\ &\propto \sum_{s\in S} d^\pi(s) \sum_{a\in A} Q^\pi(s,a)\nabla_\theta \pi_\theta(a|s) \\ &= \sum_{s\in S} d^\pi(s) \sum_{a\in A} \pi_\theta(a|s) Q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\ &= \mathbb{E}_{s\sim d_\pi, a\sim\pi_\theta}[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)]\end{aligned} \quad (32)$$

The PG theorem provides a simple reformulation of the derivative of the objective function without involving the derivative of the state distribution $d_\pi(.)$ and simplifies considerably the gradient computation.

Proposition 4.1 (Monte Carlo Policy Gradient Principle)**.** *Monte-Carlo policy gradient (also labeled REINFORCE) relies on an anticipated return computed by Monte-Carlo methods using real sample trajectories to update the policy parameter θ. The Monte Carlo policy gradient works since the expectation of the sample gradient equals the actual gradient:*

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{s\sim d_\pi, a\sim\pi_\theta}[Q^\pi(s_t,a_t)\nabla_\theta \ln \pi_\theta(a_t|s_t)] \\ &= \mathbb{E}_{s\sim d_\pi, a\sim\pi_\theta}[G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]\end{aligned} \quad (33)$$

where $G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$ is the discounted future reward. As defined in section 3, s_t, a_t, r_t refer to the state, action, and reward at time step t on a particular trajectory or episode. (Equation (33)) holds because $Q^\pi(s_t,a_t) = \mathbb{E}_{s\sim d_\pi, a\sim\pi_\theta}[G_t|s_t,a_t]$.

The PG theorem lays the theoretical foundation for various PG algorithms. In our setting, DPG trains a RL policy using an on-policy approach, where policy is constantly updated using the latest learned policy.

4.2 Stochastic Policy Gradient (SPG)

From the PG theorem mentioned above, the policy function $\pi(.|s)$ can be considered stochastic over the action space $\mathscr{A}$ given the observed state s_t (SPG). However, in the DPG setting, we model the policy as a deterministic function: $a = \mu(s)$.

We consider an environment where the state space $\mathscr{S}$ is continuous, and the Markov property holds. Let $\rho^\mu(s \to s', k)$ be the visitation probability density of moving from s to s' after taking k steps by policy μ and let $\rho_0(s)$ be the initial distribution over states. Finally, let $\rho^\mu(s') = \int_{\mathscr{S}} \sum_{k=1}^{\infty} \gamma^{k-1} \rho_0(s) \rho^\mu(s \to s', k) ds$ be the discounted state distribution.

The return objective function is defined as follows:

$$J(\theta) = \int_{\mathscr{S}} \rho^\mu(s) Q^\mu(s, \mu_\theta(s)) ds \quad (34)$$

According to the PG theorem (4.1), the gradient of the objective function can be written as:

$$\begin{aligned}\nabla_\theta J(\theta) &= \int_{\mathscr{S}} \rho^\mu(s) \nabla_a Q^\mu(s,a) \nabla_\theta \mu_\theta(s)|_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s\sim\rho^\mu}[\nabla_a Q^\mu(s,a) \nabla_\theta \mu_\theta(s)|_{a=\mu_\theta(s)}]\end{aligned} \quad (35)$$

When the probability distribution consists of a single extreme non-zero value over a single action, we can regard the deterministic policy as an exceptional case of the stochastic one. In (Silver et al. 2014), the authors have shown that if the stochastic policy $\pi_{\mu_\theta,\sigma}$ is re-parameterized by a deterministic policy μ_θ and a variation variable σ, the stochastic policy is equivalent to the deterministic case when $\sigma = 0$. When compared to the deterministic policy, we expect the stochastic policy to need more samples since it integrates the data over the entire state and action spaces.

In addition, we consider the case where the SPG algorithm is adapted to meet the off-policy actor-critic conditions. In every iteration, there are two steps: the actor update $a = \mu_\theta(s)$ and the

Q-learning update (critic parameters w): Here, the Q-learning updates the off-policy parameters depending on the new gradient that we already computed above:

$$\begin{aligned} \delta_t &= R_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \\ w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_a Q^w(s_t, a_t) \nabla_\theta \mu_\theta(s)|_{a=\mu_\theta(s)} \end{aligned} \tag{36}$$

4.3 Deep Deterministic Policy Gradient (DDPG)

DDPG (Lillicrap et al. 2016), short for Deep Deterministic Policy Gradient, is a model-free off-policy actor-critic algorithm that combines ideas from DPG with DQN (Deep Q-Network) (Mnih et al. 2015). Recall that DQN stabilized the learning of Q-function by experience replay as well as the frozen target network. While the original DQN works in discrete state spaces, DDPG extends it to continuous state spaces using the actor-critic framework, learning a deterministic policy.

For better exploration, an exploration policy μ' is designed by adding noise $\mathcal{N}$:

$$\mu'(s) = \mu_\theta(s) + \mathcal{N}$$

Additionally, DDPG does soft updates ("conservative policy iteration") on both actor and critic parameters, with $\tau \ll 1$: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$. In this manner, the target network values are constrained to change gradually. The target network stays frozen for a certain period of time, which differs from the DQN algorithm.

4.4 Trust Region Policy Optimization (TRPO)

Parameter updates that alter the policy significantly at one step need to be avoided to improve training stability. This idea is carried out by TRPO (Schulman et al. 2017a). At each iteration, a KL divergence constraint is enforced on the size of the policy update.

First, let us consider the case of off-policy RL. The policy β used for gathering sample trajectories is not the same as the policy π to optimize for. In an off-policy model, the objective function measures the total advantage over the state visitation distribution and actions. At the same time, the importance sampling estimator (Hesterberg 1987) compensates the mismatch between the training data distribution and the real policy state distribution:

$$\begin{aligned} J(\theta) &= \sum_{s\in\mathcal{S}} \rho^{\pi_{\theta_{\text{old}}}} \sum_{a\in\mathcal{A}} \left(\pi_\theta(a|s)\hat{A}_{\theta_{\text{old}}}(s,a)\right) \\ &= \sum_{s\in\mathcal{S}} \rho^{\pi_{\theta_{\text{old}}}} \sum_{a\in\mathcal{A}} \left(\beta(a|s)\frac{\pi_\theta(a|s)}{\beta(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right) \\ &= \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}, a\sim\beta}\left[\frac{\pi_\theta(a|s)}{\beta(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right] \end{aligned} \tag{37}$$

where θ_{old} are the policy parameters before the policy update and thus known to us; $\rho^{\pi_{\theta_{\text{old}}}}$ is described in the same way as in (Equation (34)); $\beta(a|s)$ is the behavior policy for gathering sample trajectories. Note that we use an estimated advantage $\hat{A}(.)$ rather than the true advantage function $A(.)$ since the real rewards are usually unknown.

Theoretically, when training on-policy, the policy for gathering data is the same as the policy that we desire to optimize. However, the behavior policy can get stale when the sampling is running in parallel asynchronously. The subtle difference considered by TRPO is that it labels the behavior policy as $\pi_{\theta_{old}}(a|s)$, and thus the objective function becomes:

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}, a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right] \tag{38}$$

The aim of TRPO is to maximize the objective function $J(\theta)$ in (Equation (38)) subject to the trust-region constraint which enforces the distance between new and old policies measured by KL-divergence to be insignificant enough, within a parameter ε:

$$\mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(.|s)\|\pi_\theta(.|s)] \leq \varepsilon \tag{39}$$

4.5 Proximal Policy Optimization (PPO)

TRPO is relatively complex but we can simplify it and maintain similar performance with PPO by implementing a similar constraint using a clipped surrogate objective function (Schulman et al. 2017b).

We denote the probability ratio between new and old policies as:

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \tag{40}$$

Then, the on-policy objective function of TRPO becomes:

$$J^{\text{TRPO}}(\theta) = \mathbb{E}[r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a)] \tag{41}$$

With no limitation on the distance between θ_{old} and θ, the fact that we would like to maximize $J^{TRPO}(\theta)$ would result in instability with very large parameter updates and sizable policy ratios. Here, PPO imposes the same TRPO constraint by forcing $r(\theta)$ to remain within a small interval around 1, exactly $[1-\varepsilon, 1+\varepsilon]$, where ε is a hyperparameter.

$$\begin{aligned} J^{\text{CLIP}}(\theta) = \mathbb{E}\Big[&\min(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a), \text{clip}(r(\theta), \\ &1-\varepsilon, 1+\varepsilon)\hat{A}_{\theta_{\text{old}}}(s,a))\Big] \end{aligned} \tag{42}$$

The ratio within $[1-\varepsilon, 1+\varepsilon]$ is clipped by the function $clip(r(\theta), 1-\varepsilon, 1+\varepsilon)$. The motivation for raising the policy update to extremes for better rewards is lost because the minimum of the real value and the clipped version is used by the objective function of PPO (Equation (42)).

On the network architecture with shared parameters, when applying PPO for policy (actor) as well as value (critic) functions, besides the clipped reward, the objective function is strengthened with an error term on the value estimation and an entropy term to incentivize sufficient exploration.

$$\begin{aligned} &J^{\text{CLIP'}}(\theta) \\ &= \mathbb{E}\Big[J^{\text{CLIP}}(\theta) - c_1(V_\theta(s) - V_{\text{target}})^2 + c_2 H(s, \pi_\theta(.))\Big] \end{aligned} \tag{43}$$

where both c_1 and c_2 are hyperparameter constants.

On a set of benchmark tasks, PPO has been tested and proved to produce good results with much greater simplicity compared to TRPO. As a result, we decided to consider only PPO and adapt it to PM, leading to four RL models in our experimental section.

5. Reinforcement Learning Challenges in Finance

Financial dynamic PM is still a challenging task for RL due to several characteristics.

Non-stationarity and partial observability: RL algorithms may perform particularly well when trained in closed and synthetic environments. For instance, in game playing, conditions under which the RL agent repeats its decision process do not change. That is not the case for real-world problems such as finance. Financial market data is highly volatile and non-stationary, which results in a very low signal to noise ratio. In addition, the price information does not capture the complete state information of the markets dynamics, which leads to a partially observable state space representation. Noisy, non-stationary, and partially observable market data results in inefficient state representations for RL algorithms that cause performance degradation.

State-space visitation and Markov property: In game playing, the state space can often be exhaustively explored, and we may revisit some states during the training process but in PM we cannot; therefore, we need to validate (backtest) our deep RL models with data not used for training. Hence, the training process (including exploration) is much more challenging in the context of finance. Moreover, the majority of the RL models assume that the environment is Markov. This is not the case in finance, where the variables at play are path-dependent.

Reward function: Reward engineering is one of the most critical components in designing RL algorithms. The challenge in designing a reward function is to incentivize the behaviors we want while still being learnable. In fact, it is hard to design a reward function that is financially meaningful. Typical game reward functions do not need to deal with multiple objectives as required in financial domains. Complex reward functions complicate the training of the RL agent, which may prevent achieving desirable performance. In the financial context, we want the agent to learn financial concepts such as alpha, beta, diversification, etc.

Transaction Costs: Without considering transaction costs (TC), a greedy algorithm can achieve optimal results. To be specific, allocating the entire wealth into the asset with the highest expected increase rate at each time increment is, of course, the optimal policy in such a naive setting. In our setting, the RL agent has to always incur a transaction fee penalty regardless of the action taken (good or bad), which will discourage frequent portfolio changes in response to minor changes in immediate return. In addition, what makes TC a challenging problem for RL in PM is the fact that we need to handle asynchronous reward delays longer than most other RL applications. For example, a security transaction may encounter a delay to implement in the market while a decision in a computer game can be implemented immediately. Additionally, different delays will be encountered for different securities in the portfolio due to their market liquidity. These asynchronous transaction delays result in asynchronous delays in the reward information required for the RL optimization (assumptions (c) and (d) in Assumption 6.1).

Constraints: One of the most important problems in RL is how to handle the financial constraints associated with multi-dimensional PM. One of the primary constraints in multi-asset investment that must be satisfied is the portfolio weight constraint. That is, each time the portfolio is rebalanced, the trading control weight of each asset must be a fraction of one and all of the control weights must sum to one. Additional critical constraints, such as maximum allocation per individual asset or per economic sector, are not yet taken into consideration by current RL models.

Multi-type and multi-frequency: In finance, the data come under different types and frequencies. We talk about multi-type and multi-frequency data. Usually, in finance, we deal with Technical data indicators that use high, low, open, close prices and volume information, Fundamental data indicators such as revenues, earnings, future growth, etc., or another type of data such as sentiment data. However, it is still unclear how to represent multi-type and multi-frequency data in a way that is compatible with RL.

Feature combination: Few works discuss the combinations of features in RL (Fischer 2018). In game playing or robot control, deep RL models learn using low-level pixel information. In PM, many high-level features are available, giving the opportunity for feature selection to affect performance. Common features include the closing price, the open price, the high price, the low price, and volume and non-price information data such as Price-to-Earning Ratio (PE) and Price-to-book ratio (PB) that can also provide insights into market movements. The investigation of feature combinations is still an open question.

Cross-sectional analysis: A cross-sectional regression in statistics and econometrics is a type of regression in which the explained and explanatory variables are all related to the same single period or point in time. This type of cross-sectional analysis is in direct contrast to a time-series regression or longitudinal regression in which the variables are viewed as being associated with a sequence of points in time (Andrews 2005; Wooldridge 2009). In practice, before making a decision, investors will typically use a fine blend of time series analysis and cross-sectional analysis. However, there is still no work in the RL literature that exposes cross-sectional analysis. The challenge for cross-sectional analysis is not with developing cross-sectional feature sets but with combining those features with the time series features in a neural network architecture.

Time-horizon: Conventional game playing RL algorithms are designed for infinite-horizon MDP while PM seeks to maximize absolute portfolio value or other related objective functions in a finite time horizon. The optimal policy in the infinite-horizon MDP case using discount rewards is the stationary policy, while the optimal policy in the finite horizon case using average rewards is not stationary. The use of discount rewards in RL is well established while research on average rewards is very preliminary. While an optimal non-stationary policy should be preferred for the PM problem, it is not clear how much advantage it would offer over an optimal stationary policy. We decided to focus on other challenges for RL enhancement which were more promising.

Risk: Deep RL models trained under a particular market dynamic may underestimate the risk when the dynamics of the market changes, which may result in disastrous deterioration

of its performance in a real trading environment. (Liang et al. 2018) incorporated volatility as a second moment in the reward function and used also Sharpe ratio (SR) (Sharpe 1994) in the utility function but without success in training. They claim that these modifications make the objective function too complex and hard to train. In our case, we found that the use of volatility in combination with our other adaptations improves performance (please refer to section 6).

Interpretability of the investment policy: Finance is unforgiving environment, where a bad decision could prove to be fatal. Recently, several publications using RL in finance have emerged and the field is growing in popularity (section 1). Total return, the common performance metric, is useful for comparing algorithms but does not help us determine how close an RL algorithm is to an optimal solution.

These PM problem characteristics have not been studied sufficiently in the general framework of RL and, more specifically, in relation to RL in finance. In this work, we will show how we tackle some of these issues.

6. Modifications to RL Algorithms for Application to Finance

We have developed four modifications of deep RL algorithms optimized for game playing and robot control to adapt them for PM.

Assumptions 6.1 (Financial RL environment)**.** In the following, we will use these assumptions for our financial RL environment:

(a) The market data are continuous, where the opening price on any given day equals the closing price on the previous day.
(b) The actions of the investor do not affect the dynamics of the market.
(c) The asset prices are infinitely divisible which means that it is possible for the RL agent to buy or sell fractional shares of any asset without imposing minimum orders sizes.
(d) At any given trading period, the assets are immediately available in limitless quantities at the current price.

6.1 Characterizing optimality in deep reinforcement learning models for finance

By bridging the gap between financial theory and RL theory, the goal of this section is to take a few steps in the direction that could allow us to assess whether or not the actions taken by the AI investment agent make sense. This would complement the portfolio manager and give him a way to interpret the investment policy and decide whether to trust the agent.

Recall from section 2, the portfolio weight vector was defined to be $\mathbf{a} = \left(a^{(1)}, \dots a^{(d)}\right)^T \in \mathbb{R}^d$ whose j-th component $a^{(j)}$ represents the proportion of the investor's capital invested in asset j. We also wrote the random return vector as $\mathbf{r} = \left(r^{(1)}, \dots r^{(d)}\right)^T \in \mathbb{R}_+^d$. Here, the j-th component $r^{(j)} \geq 0$ of $\mathbf{r}$ indicates the ratio of the consecutive closing prices of asset j. This means $r^{(j)}$ is the factor by which the capital invested in the j-th asset increases during the trading period. We define S_T to be the total *log-return* of the portfolio after T periods:

$$S_T = S_0 \prod_{i=1}^{T} \left\langle \mathbf{a}(\mathbf{r}_1^{i-1}), \mathbf{r}_i \right\rangle, \tag{44}$$

According to (Eq.4), we can re-write the financial *growth function*:

$$Q(\boldsymbol{a}; \xi) = \int \log\left(\langle \boldsymbol{a}, \boldsymbol{r} \rangle\right) d\xi(\boldsymbol{r}) = \mathbb{E}\Big[\log\left(\langle \boldsymbol{a}, \boldsymbol{r} \rangle\right)\Big], \tag{45}$$

where $\xi(\boldsymbol{r})$ is the distribution function of the stochastic return vector $\boldsymbol{r}$. In order to align the financial growth function with the notion of the value function used in RL, we define the optimal *growth value function* by:

$$V^*(\xi) = \max_{\boldsymbol{a} \in \Delta_d} Q(\boldsymbol{a}; \xi). \tag{46}$$

where Δ_d indicates the simplex of all vectors $\mathbf{a} \in R_+^d$ with non-negative elements summing up to one.

Proposition 6.1.

1. *For a given control action $\boldsymbol{a}$, $Q(\boldsymbol{a}; \xi)$ is a linear function of the return distribution function ξ.*
2. *For a given return distribution function ξ, $Q(\boldsymbol{a}; \xi)$ is a concave function on $\boldsymbol{a}$.*
3. *$V^*(\xi)$ is a convex function on ξ.*

Proof. 1. Let a be the control action, ξ_1, ξ_2 two return distribution functions and α a real scalar:

$$\begin{aligned} Q(\boldsymbol{a}; \xi_1 + \alpha\xi_2) &= \int \log\left(\langle \boldsymbol{a}, \boldsymbol{r} \rangle\right) d(\xi_1 + \alpha\xi_2)(\boldsymbol{r}) \\ &= \int \log\left(\langle \boldsymbol{a}, \boldsymbol{r} \rangle\right) d(\xi_1)(\boldsymbol{r}) + \alpha \int \log\left(\langle \boldsymbol{a}, \boldsymbol{r} \rangle\right) d(\xi_2)(\boldsymbol{r}) \\ &= Q(\boldsymbol{a}; \xi_1) + \alpha Q(\boldsymbol{a}; \xi_2) \end{aligned} \tag{47}$$

□

Proof. 2. Let ξ be a distribution that generates the stochastic return vector $\boldsymbol{r}$. From the concave property of the $log(.)$ function, we can write:

$$\begin{aligned} \forall \theta \in [0,1], \forall \boldsymbol{a}_1, \boldsymbol{a}_2 \in \Delta_d : \log\Big(\theta \boldsymbol{a}_1 + (1-\theta)\boldsymbol{a}_2\Big) \cdot \boldsymbol{r} \\ \geq \theta \log \boldsymbol{a}_1 \cdot \boldsymbol{r} + (1-\theta) \log \boldsymbol{a}_2 \cdot \boldsymbol{r}. \end{aligned} \tag{48}$$

The expectation is just an integral over the underlying probability space. Since integrals respect the order relation in inequalities, the expectation operator respects the inequality order as well. We obtain the concave property on $\boldsymbol{a}$ by simply applying the expectation operator on both sides:

$$\begin{aligned} \forall \theta \in [0,1], \forall \boldsymbol{a}_1, \boldsymbol{a}_2 \in \Delta_d : Q(\theta \boldsymbol{a}_1 + (1-\theta)\boldsymbol{a}_2; \xi) \\ \geq \theta Q(\boldsymbol{a}_1; \xi) + (1-\theta) Q(\boldsymbol{a}_2; \xi). \end{aligned} \tag{49}$$

□

Proof. 3. Now let us show the convexity property of V^* with respect to the underlying distribution function of stochastic returns ξ. Consider two distribution functions ξ_1 and ξ_2 and denote their corresponding optimal portfolio weights by $\boldsymbol{a}^*(\xi_1)$ and $\boldsymbol{a}^*(\xi_2)$.

With the above notations, $\boldsymbol{a}^*(\xi_1)$, $\boldsymbol{a}^*(\xi_2)$ and $\boldsymbol{a}^*(\theta\xi_1 + (1-\theta)\xi_2)$ represent the portfolio weight vector (control actions)

that maximize $Q(\boldsymbol{a};\xi_1)$, $Q(\boldsymbol{a};\xi_2)$, and $Q(\boldsymbol{a};\theta\xi_1+(1-\theta)\xi_2)$ respectively. We write:

$$\begin{aligned}\forall\theta\in[0,1]\quad & V^*\Big(\theta\xi_1+(1-\theta)\xi_2\Big)\\ &=\max_{\boldsymbol{a}\in\Delta_d} Q(\boldsymbol{a};\theta\xi_1+(1-\theta)\xi_2)\\ &=Q\Big(\boldsymbol{a}^*(\theta\xi_1+(1-\theta)\xi_2);\theta\xi_1+(1-\theta)\xi_2\Big)\\ &=\theta Q\Big(\boldsymbol{a}^*(\theta\xi_1+(1-\theta)\xi_2);\xi_1\Big)\\ &\quad+(1-\theta)Q\Big(\boldsymbol{a}^*(\theta\xi_1+(1-\theta)\xi_2;\xi_2\Big)\\ &\le\theta Q(\boldsymbol{a}^*(\xi_1);\xi_1)+(1-\theta)Q(\boldsymbol{a}^*(\xi_2);\xi_2)\\ &=\theta\max_{\boldsymbol{a}\in\Delta_d} Q(\boldsymbol{a};\xi_1)+(1-\theta)\max_{\boldsymbol{a}\in\Delta_d} Q(\boldsymbol{a};\xi_2)\\ &=\theta V^*(\xi_1)+(1-\theta)V^*(\xi_2).\quad\square\end{aligned}\tag{50}$$

Proposition 6.2. *Let us consider a portfolio of d assets. An optimal policy (investment strategy) $\boldsymbol{a}^*$ of a RL agent exists if and only if the following condition is satisfied:*

$$\forall j\in\{1,\dots,d\}\qquad \mathbb{E}\left[\frac{r^{(j)}}{\langle\boldsymbol{a}^*,\boldsymbol{r}\rangle}\right]\le 1.\tag{51}$$

Proof. We know from Proposition 6.1 that $Q(\boldsymbol{a};\xi)$ is a concave function on $\boldsymbol{a}$, and the domain of definition of $\boldsymbol{a}$ is a simplex Δ_d. The necessary and sufficient condition (NSC) for $\boldsymbol{a}^*$ to be an optimal policy is that the directional derivative of $Q(\boldsymbol{a};\xi)$ at $\boldsymbol{a}^*$ along any curve must be negative.

Let's consider an element in the simplex Δ_d called $\boldsymbol{a}_\theta=(1-\theta)\boldsymbol{a}^*+\theta\boldsymbol{a}$, where $\theta\in[0,1]$, that moves from $\boldsymbol{a}^*$ to an arbitrary control vector $\boldsymbol{a}$ in Δ_d (figure 1).

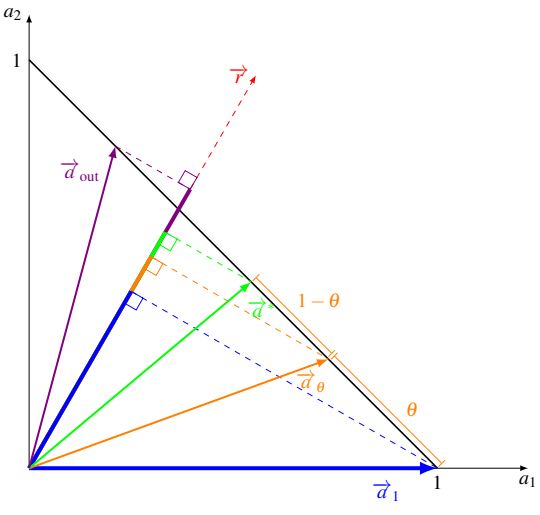

Figure 1: Set of feasible reinforcement learning solutions (simplex in two dimensions).

The above NSC translates to the following condition:

$$\theta.\nabla Q(\boldsymbol{a}_\theta;\xi)\big|_{\theta=0^+}\le 0\quad\forall\boldsymbol{a}\in\Delta_d.\tag{52}$$

On the other hand, by definition:

$$\begin{aligned}\forall\boldsymbol{a}\in\Delta_d\ \theta.\nabla Q(\boldsymbol{a}_\theta;\xi)\big|_{\theta=0^+}&=\theta.\nabla\mathbb{E}[\log(\langle\boldsymbol{a}_\theta,\boldsymbol{r}\rangle)]\Big|_{\theta=0^+}\\ &=\lim_{\theta\to0^+}\frac{d}{d\theta}\mathbb{E}\left[\log\left(\frac{(1-\theta)\langle\boldsymbol{a}^*,\boldsymbol{r}\rangle+\theta\langle\boldsymbol{a},\boldsymbol{r}\rangle}{\langle\boldsymbol{a}^*,\boldsymbol{r}\rangle}\right)\right]\\ &=\mathbb{E}\left[\lim_{\theta\to0^+}\frac{d}{d\theta}\log\left(1+\theta\left(\frac{\langle\boldsymbol{a},\boldsymbol{r}\rangle}{\langle\boldsymbol{a}^*,\boldsymbol{r}\rangle}-1\right)\right)\right]\\ &=\mathbb{E}\left[\frac{\langle\boldsymbol{a},\boldsymbol{r}\rangle}{\langle\boldsymbol{a}^*,\boldsymbol{r}\rangle}-1\right]\le 0\end{aligned}\tag{53}$$

In particular, for $\boldsymbol{a}=\big(0,\dots,1,\dots,0\big)^T\in\Delta_d$, we get:

$$\forall j\in\{1,\dots,d\}\qquad\mathbb{E}\left[\frac{r^{(j)}}{\langle\boldsymbol{a}^*,\boldsymbol{r}\rangle}\right]\le 1.\tag{54}$$

□

We will use the theoretical results of Proposition 6.2 to experimentally assess the quality of the investment policies of different RL models. Typical deep RL approaches for PM focus on total return (profit generated) but total return can be deceptive as shown in the experimental section. We desired a deeper understanding of the quality of the RL policy, so we developed a scoring approach for PM that generally rewards good decisions without focusing too narrowly on the value of the returns.

In figure 1, we only show the set of feasible portfolio weights for two assets. But for N-dimensional portfolios we cannot visualize a simplex in N-dimensions. Our Geometric Policy Score shown in figure 2 is inspired by the visualization of figure 1 and the necessity of reducing the dimensionality for a better visualization. We divide our portfolio securities into two groups on a weekly basis: the top-half performing securities and the bottom-half performing securities. We score the RL policy by comparing the weight of securities in first group with the weight of securities in the second group. The score is the weekly average percent allocation to the top half performing securities. More weight of securities in the first group should in principle lead to profitable strategies in the long term (please refer to section 7). Note that our score does not depend on the actual returns of each individual security but it does depend on the allocation of the individual securities. By rewarding allocations to the top half performing securities, our Geometric Policy Score encourages beneficial diversified portfolio strategies without focusing too narrowly on total returns. In Proposition 6.2, we used θ to measure how close we were to an optimal solution $\boldsymbol{a}^*$. Our Geometric Policy Score also defines a θ value that becomes smaller as we approach a good solution.

Figure 2 shows an example where 70% of the portfolio allocation went to the top half performing securities and 30% went to the bottom half. $\theta_{70\%}=0.42$ is computed geometrically from those values. When only 50% of the portfolio allocation went to the top half performing securities, $\theta_{50\%}=\sqrt{2}/2\cong 0.70$. Smaller values of θ represent better stock allocations as more of the portfolio is invested in the top half performing securities. The θ associated with our Geometric Policy score serves the same purpose as the θ in figure 1.

6.2 Multi-objective and risk-sensitive utility function

In the financial context, we want the agent to learn financial concepts such as alpha, beta, diversification, etc., so that the

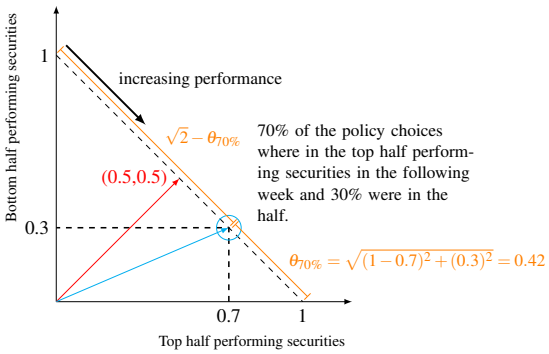

Figure 2: Geometric Policy Score: allocation to top performing securities vs. bottom performing securities.

RL reward function becomes financially meaningful. Thus, the portfolio optimization model for the actor can be formulated at time t as follows:

$$
\begin{aligned}
&\max_{\theta} \sum_{t=1}^{T} \gamma^t \Big(\mathscr{R}_t - \alpha \mathscr{R}_t^{EW} - \beta \Sigma_t^2 - \gamma \max(a_{1,t}^{\theta}, \ldots, a_{m,t}^{\theta}) \Big) \\
&\text{s.t. } \mathscr{R}_t = \ln \Big(\langle \mathbf{a}_{t-1}, \mathbf{r}_t \rangle - c \sum_{i=1}^{m} \mid a_{i,t} - a_{i,t-1} \mid \Big) \\
&\quad a_t = softmax(W^{(3)} \tilde{h}_2 + b^{(3)}) \quad \textit{\#Portfolio vector weights @ time-step t} \\
&\quad \tilde{h}_2 = ReLU(W^{(2)} \tilde{h}_1 + b^{(2)}) \quad \textit{\#Dense layer} \\
&\quad \tilde{h}_1 = ReLU(W^{(1)} \tilde{\Phi}_t + b^{(1)}) \quad \textit{\#Dense layer} \\
&\quad \tilde{\Phi}_t = \mathscr{H} \odot \Phi_t \quad \textit{\# } \Phi_t \textit{ is dropped out using the rand(m) kernel } \mathscr{H} \\
&\quad \mathscr{H}_{i,j} \sim Bernouilli(p) \\
&\quad \ldots \\
&\quad \Phi_t = \{\Phi_t^{\text{Row}}\} \oplus \{a_{t-1}\} \quad \textit{\#Feature combination with previous action} \\
&\quad \Phi_t^{\text{Row}} = Conv(K^{(1)} * \chi_t + b^{(1)}) \quad \textit{\#Feature extraction of multi-type row data}
\end{aligned}
\tag{55}
$$

where:

- $\mathscr{R}_t$ is the periodic log-return including the transaction commission rate c.
- $\mathscr{R}_t^{EW}$ is the return of the equal-weighted portfolio which serves as a baseline we would like the agent to outperform.
- Σ_t^2 is the covariance matrix. The loss function is constrained by reducing the profit from investing in highly volatile assets.
- $\max(a_{1,t}^{\theta}, ..., a_{m,t}^{\theta})$ represents the maximum of the portfolio weights. This term is set up to make sure the agent avoids investing fully in one stock.
- r_t indicates the rate of price change within a trading period (one day in our model).
- χ_t represents multi-type row data (high-low-open-close prices and volume) (figure 3).
- $W^{(l)} = \left[w_{i,j}^{(l)} \right] \in \mathbb{R}^{N_l \times N_{l-1}}$ is the weight matrix and $b^{(l)} \in \mathbb{R}^{N_l}$ the bias vector for layer l; N_l denotes the number of neurons in a given layer l.
- $\boldsymbol{\theta} = \{(W^{(1)}, b^{(1)}), ..., (W^{(3)}, b^{(3)})\}$ represents the trading parameters of the policy network.

In our model, we considered a periodic logarithmic reward function because the returns are compounded via a product formula. The product turns into a sum when we take the logarithm, which will simplify the optimization and preserve the concavity property of the reward function and, hence, may potentially help us to better study the value growth function and/or the policy structure as discussed in section 6.1. We believe this is useful in the context of PM, so we incorporated the log-utility function in all the deep RL models considered in our work. In addition, we considered a term that takes into account the transaction costs (TC).

6.3 Multi-type tensor representation and Markov property relaxation

Figure 3 shows how we build the data structure used by the RL agent for one security. At the bottom we show a sequence samples over time t. One sample consists of a tensor of five types of data at that time point: open, close, high, low prices, and volume. With the Markov assumption we will also have a similar tensor with the previous state. We relaxed the Markov assumption by including a time window of historical data from t to $t-k$. If we view the tensor as a circle, then the historical window represents a tube, which we unroll to form a grid. With multiple securities, we stack the grids to form a block (top left of figure 4). This is a different approach than used in (Aboussalah and Lee 2020) where Recurrent Neural Networks (RNNs) are used. RNNs support feedback for trend analysis and hence relax the Markov assumption in a different way. Our approach uses CNNs which allow multi-type data in combination with historical time windows to relax the Markov assumption and enable trend analysis without the need for recurrent feedback. Our approach is general enough to also consider Fundamental data (FD) such as revenues, earnings, future growth, etc. or other non-price data such as sentiment data.

The second challenge is dealing with multi-type data when we have data from multiple securities. Our solution is to create a channel for each type of data. With this approach, we can use a multi-channel convolutional neural network (CNN) to extract features through receptive fields and pooling operations, where each channel corresponds to a specific type of data such as high-low-open-close prices and volumes.

6.4 Cross-sectional deep reinforcement learning

A cross-sectional regression in statistics and econometrics is a type of regression in which the explained and explanatory variables are all related to the same single period or point in time. This type of cross-sectional analysis is in direct contrast to a time-series regression or longitudinal regression in which the variables are viewed as being associated with a sequence of points in time (Andrews 2005; Wooldridge 2009).

Cross-sectional analysis is one of the two overarching comparison methods for stock analysis. This analysis examines data collected at a single point in time, instead of data collected over a period of time. The analysis starts by establishing research goals and defining the variables that an analyst wants to measure. The next step is identifying the cross-section, such as a group of peers or an industry, and setting the specific point in time to be assessed. The final step is conducting analysis, based

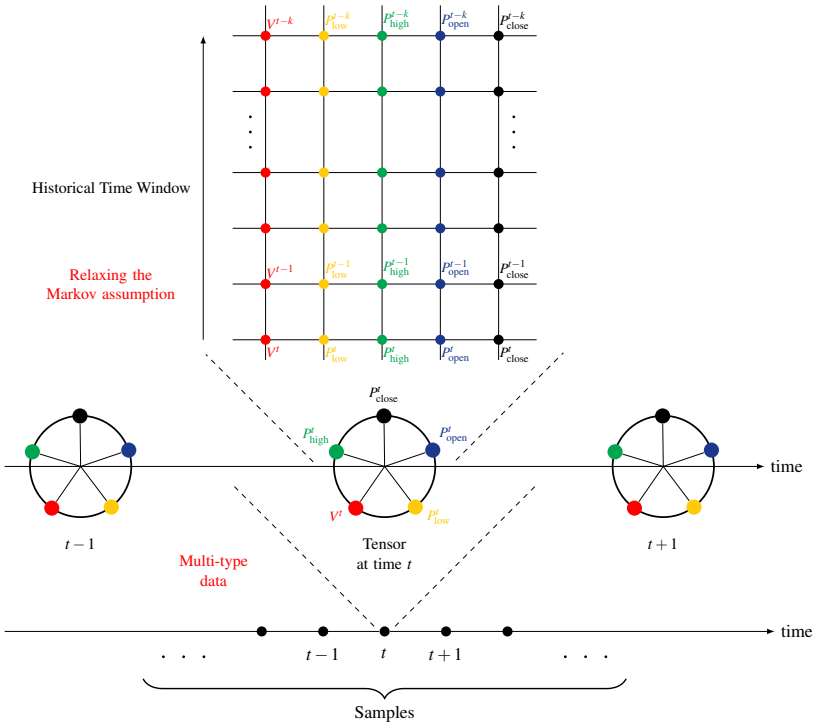

Figure 3: Building multi-type tensor representation from multiple row data for one security.

on the cross-section, and coming to a conclusion on the performance of a company or organization. Cross-sectional analysis, essentially, shows an investor which company is best in relation to the metrics that are important to him.

On the other hand, time series analysis, also labeled trend analysis, concentrates on a single company over time. In this case, past performance is the sole criterion on which the company is being judged. Time series analysis indicates to an investor whether the company is performing better or worse than before by the metrics that interest him. Often, these metrics will be classic key performance indicators (KPIs) such as earnings per share (EPS), debt-to-equity ratio, free cash flow, and so on.

Figure 4 shows the overall neural network architecture used to build variants of the deep RL models. We start by building the multi-type tensor from the multi-row data as shown in figure 3. The multi-type tensor forms the input data block with dimensions of the historical window size, the number of stocks, and the number of data types which correspond to receptive field channels. Our architecture supports two options individually or in combination: time series analysis (top of the figure) and cross-sectional analysis (bottom of the figure). Each option requires a different kernel size for the convolution networks.

For the time series analysis pipeline (top of figure 4), we feed the multi-type tensor into two layers of convolution networks. The convolution operators are used to de-noise the signals and capture the trend. The time series analysis pipeline only looks at the trend of each individual asset. We built the cross-sectional pipeline (bottom of figure 4) in order to learn how the assets move relative to each other at any given time step by applying a vertical CNN kernel at each time step, hence exposing complementary information compared to the time series analysis. CNNs make it easy to do cross-sectional analysis which is an advantage over RNNs.

At the end of the convolution layers, whether it is for trend or cross-sectional analysis, we either directly take the decision to compute the portfolio weights based only on the features captured via the convolution layers; or we combine them with dense neural networks before outputting the portfolio weights. Before the final step, we concatenate the portfolio allocation vector of the previous time step (a_{t-1}) to the feature vector in order to help the agent to better make a decision at the current step (a_t). Then we present this richer representation to the final layer (policy layer) responsible for making the decisions and updating the portfolio weights.

7. Experimental results

7.1 Dataset

In our experiments, the investment decisions are made daily and each input signal represents a multidimensional tensor that aggregates historical open, low, high, close prices, and volume (figure 4). Our deep RL models are trained and tested using a sliding window allowing the agents to adapt to new market conditions. The models are trained and tested using 50 successive rounds of training and testing with a shift of 10 days in the sliding window between rounds. Each round consists of a training period of 200 days followed by a 10 day testing period. We found a 10 day testing window to be the largest window size appropriate for daily frequency data. It should be noted that our training and testing include the transaction costs (TC). We used the typical cost due to bid-ask spread and market impact that is 0.25%. We believe these are reasonable transaction costs for the portfolio trades.

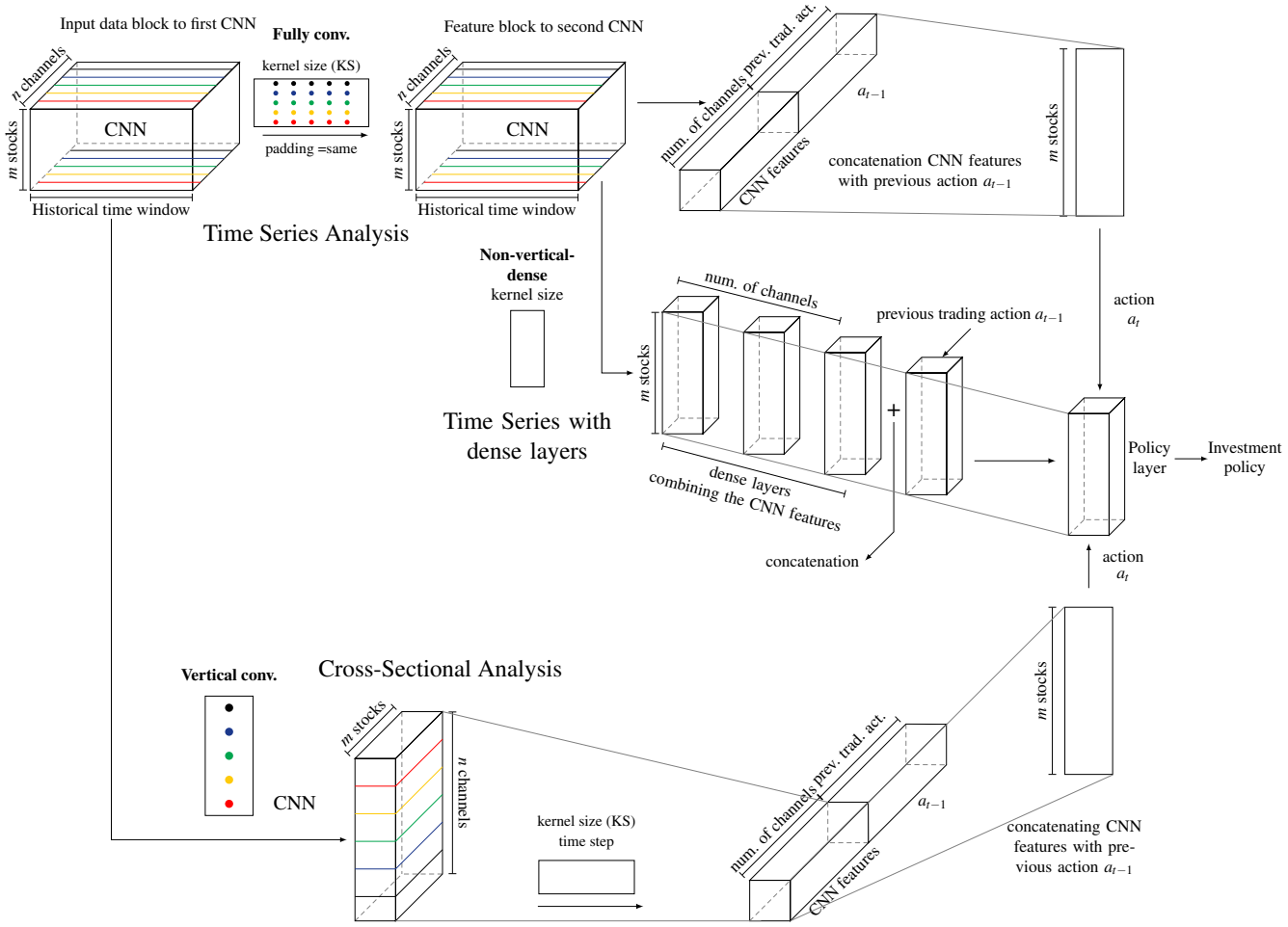

Figure 4: The overall neural network architecture used to build variants of the deep RL models supports Cross-Sectional and Time Series with optional dense layers.

We have 96 weeks of testing data ($\sim$ 2 years) with 5 trading days per week from the beginning of 2013 to the end of 2014. The RL agents rebalance the portfolio on a daily basis and are evaluated on a portfolio consisting of the following ten selected securities: American Tower Corp. (AMT), American Express Company (AXP), Boeing Company (BA), Chevron Corporation (CVX), Johnson & Johnson (JNJ), Coca-Cola Co (KO), McDonald's Corp. (MCD), Microsoft Corporation (MSFT), AT&T Inc. (T) and Walmart Inc (WMT). To promote the diversification of the portfolio, these stocks were selected from different sectors of S&P 500, so that they are uncorrelated as much as possible as shown in figure 5.

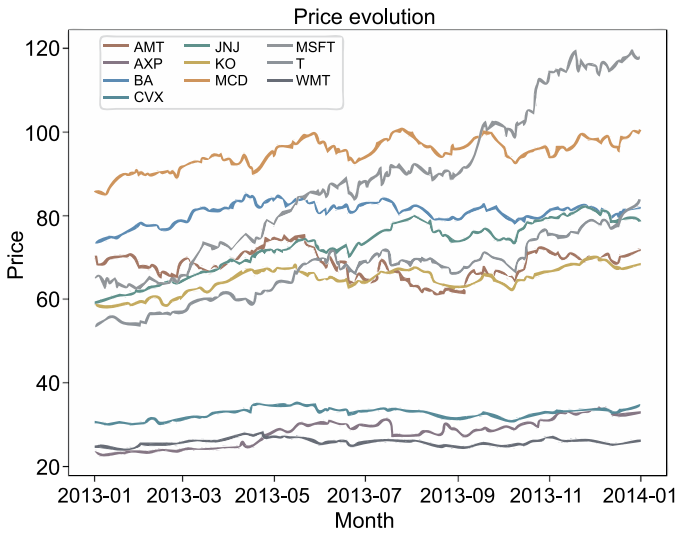

Figure 5: Evolution of stock prices over one year.

7.2 Baseline active trading strategy

For evaluation purposes, we compare our deep RL results with an active trading strategy namely the multi-period mean-variance optimization (MVO). The MVO framework proposed by (Markowitz 1952; 2010) is a quantitative tool traditionally used in dynamic portfolio allocation problem where risk and return are traded off. The solution of the MVO problem using risk-adjusted utilities may be difficult to obtain because of the nature of its objective: 1) non-linear, 2) possibly non-convex. However, under reasonable assumptions, it can be reduced to a standard convex quadratic program (Cornuejols and Tutuncu

2006):

$$\begin{aligned} \min_{y\in\mathbb{R}^m,\kappa\in\mathbb{R}} \quad & y^T Q y \\ \text{s.t.} \quad & \sum_i (\mu_i - r_f) y_i = 1 \\ & \sum_i y_i = \kappa \\ & l.\kappa \leq Ay \leq u.\kappa \\ & \kappa \geq 0 \end{aligned} \tag{56}$$

where,

1. μ, the vector of mean returns.
2. Q, the covariance matrix.
3. $\sum_i y_i = \kappa$; (cardinality constraint).
4. $l.\kappa \leq Ay \leq u.\kappa$, (other linear constraints if needed).
5. $y \geq 0$, portfolio weight vector[1].
6. r_f, risk-free rate of return.

Figure 6 shows the evolution of the optimal decision weights for the multi-period MVO model.

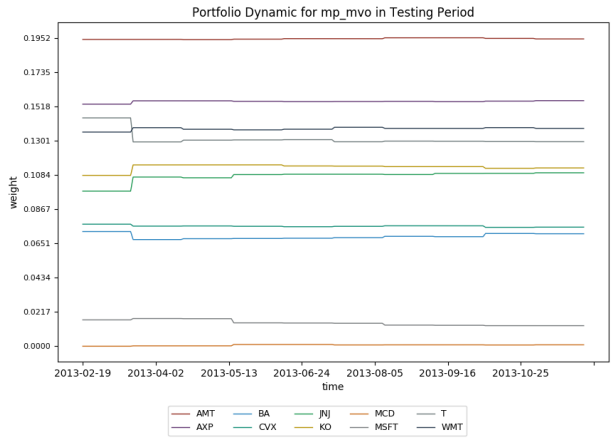

Figure 6: The distribution of portfolio weights of the rolling horizon MVO model over the test period.

7.3 Setup

Experiments were run on a 24-core machine with 33GB of memory and 2 NVIDIA Quadro P5000 (16GB) GPUs. All algorithms were implemented in Python using Keras and Tensorflow libraries. Each method is executed in an asynchronously parallel setup of 2 GPUs, that is, it can evaluate multiple models in parallel, with each model on a single GPU. When the evaluation of one model finishes, the methods can incorporate the result and immediately re-deploy the next job without waiting for the others to finish.

7.4 Hyperparameters

The hyperparameter space is represented by a hypercube: the more values it contains the harder it is to explore all the possible combinations. To efficiently find the optimal set of hyperparameters, we explored the hyperparameter space using Bayesian optimization $\mathscr{BO}$ (Hutter et al. 2011).

[1] $y \geq 0$ means that short-selling is disallowed.

Table 1 shows the range of values for the hyperparameters used during the training and validation phase. The learning rate controls the speed at which neural network parameters are updated. The window (as shown in figure 3) is used to allow the deep RL agents to utilize a range of historical data values to relax the Markov assumption. We allow the use of 2 days up to 30 days of historical data. The regularization coefficients α, β, and γ are used in the policy network loss function to account for the equally weighted portfolio baseline, covariance, and diversification respectively. The number of filters and kernel strides are the hyperparameters for the convolution neural networks. It is important to carefully optimize these parameters in order to capture the best feature representations used by the policy networks. Finally, the number of units per layer controls the model complexity. A model that is not too complex may not be able to generalize the learning while a model that is too complex may overfit.

Table 1: Hyperparameters used by our RL algorithms.

Parameters	**Bounds**	**Type**
Learning rate	10^{-5}–5.10^{-1}	Discrete
Window	2–30	Discrete
Regularization coef.	0–1	Continuous
Number of filters	2–52	Discrete
Kernel Strides	2–10	Discrete
Number of units per layer	10–2010	Discrete

The hyperparameter training and validation involved 100 trials of Bayesian Optimization ($\mathscr{BO}$) for each of the 20 classes (4 deep RL models $\times$ 5 variants). The four deep RL models correspond to: DPG, SPG, DDPG, PPO, respectively. The five variants enabled by our architecture (figure 4) are shown in table 2.

Table 2: Characteristics of Variants

RL Variant	CNN	DENSE	Time Seies (TS)	Cross-Sectional (CS)
VANILLA			X	
CNN_TS	X		X	
CNN_TS_DENSE	X	X	X	
CNN_CS	X			X
CNN_CS+TS	X		X	X

Table 3 shows the best hyperparameter configuration of the top performing RL variant for each RL model. For each deep RL model, we show the best performing variant and the combination of hyperparameters that comes with it as well as the performance measured in terms of annualized return. We benchmark the performance of the RL agents against commonly used passive and active trading strategies, the uniform buy-and-hold (UBAH) index and the dynamical multi-period Mean-Variance-Optimization (MP_MVO) model. The DDPG model and its variants fail to beat the benchmarks, while the SPG and PPO models show very similar performance to the benchmarks. Only the DPG with the cross-sectional module seems to have a better performance.

Table 3: Top hyperparameters for each of our four RL models.

Best RL Model	Number of units in each layer	Learning rate Actor	Learning rate Critic	Window	Beta coef.	Gamma coef.	Annualized Return
DPG_CNN_CS	310–290–20	0.1	–	7	0.5	0.5	35.1%
SPG_VANILLA	420–240	0.1	0.0005	10	0	0.5	16.7%
DDPG_CNN_TS	270–150	0.01	0.001	10	0	0.5	12.8%
PPO_CNN_TS	1640–150	0.00001	0.05	14	0.63	0.5	16.4%
MP_MVO	–	–	–	–	–	–	15.2%
UBAH	–	–	–	–	–	–	16.3%

7.5 Results

We produce three types of plots:

1. The first shows the total returns over a testing horizon of 96 weeks ($\sim$ 2 years). We show a set of results corresponding to different variants of the same RL class (plot (a) in figures 7-10).
2. The second shows the average Geometric Policy Score over 96 weeks. We divide the securities into top and bottom half performing sets on a weekly basis. The weekly scores are averaged to get the overall policy score over 96 testing weeks. We plot the scores with the top half performing securities on the x-axis and the bottom half performing securities on the y-axis. The data points are all on the diagonal with performance increasing from the top left to the lower right. We show a set of policy scores corresponding to different variants of the RL class (plot (b) in figures 7-10).
3. The third shows the policy weight distribution over the testing period where the y-axis is the weight of each security. The plots show 10 securities for one variant of the RL model. We plot each variant RL model separately (plots (c-g) in figures 7-10). We show only the first year so that the portfolio activity is clearly visible. The second year portfolio activity is similar to the first.

Figures 7-10 each show a set of the above plots for a different RL class. The notation for the RL model variants in each figure uses the abbreviations shown in table 4.

Table 4: Acronyms used to label RL algorithms.

Acronym	Name
CNN	Convolution Neural Network
TS	Time Series
CS	Cross-Sectional
VANILLA	Plain RL model
DENSE	With Dense Layers After CNN Layer
MP_MVO	Multi-Period Mean Variance Optimization
UBAH	Uniform Buy And Hold

Figure 6 shows a MVO weight plot demonstrating a balanced portfolio with few rebalancing trades. This policy behavior is due to the fact that MVO models are very sensitive to transaction costs.

Figure 7 shows different variants of DPG class which is an actor RL model. We can see in plots (d), (e) and (g) that the structure of the DPG policy often tends to concentrate the weights around a more or less equally weighted strategy, which corresponds to a very conservative strategy focusing more on diversifying the portfolio to protect against unsystematic risk. Plots (c) and (f) show an agent policy that focuses on one stock at a time where it is trying to buy the best stock. We can explain this RL policy behavior by the fact that the RL agents use additional information, namely the cross-sectional module which tends to incentivize the agent to break the equal weight distribution shown above. These also offer the highest returns (a) and the best policy scores (b) but clearly the strategy is risky because it is not diversified.

Figure 8 shows different variants of SPG class adapted to use an actor-critic RL approach as shown in section 4.2. We can see that in all the plots from (c) to (g) except (e), the structure of the SPG policy tends to select one stock at a time most of the time. This is probably due to the use of the action-state value function used in the actor-critic setting. Indeed, the optimization of the action-state value function will often encourage greedy actions, which could induce the RL agents to take extreme decisions (most likely not optimal in high dimensional continuous action spaces such as in PM). In this setting, these investment strategies are clearly risky because they are not diversified.

Figure 9 shows different variants of DDPG class, which is one of the most studied actor-critic RL approaches in the RL literature. In this class, we can see that in all the plots from (c) to (g), the structure of the DDPG policy tends to select one stock at a time most of the time similar to the SPG class with the difference that here the investment decisions are even more extreme. This is due to the nature of DDPG which uses two critic networks instead of one used in the SPG class. For the same reason, it is clear that the DDPG investment strategies are risky because they are not diversified.

Figure 10 shows different variants of PPO class which is another class of actor-critic RL model. We can see in almost all the plots from (c) to (g) that the structure of the PPO policy tends most of the time to concentrate the weights around a more or less equally weighted strategy, which corresponds to a very conservative strategy focusing more on diversifying the portfolio to protect against unsystematic risk. This RL policy behavior is similar to the variants of the DPG class that do not use the cross-sectional module, but with the difference here that the

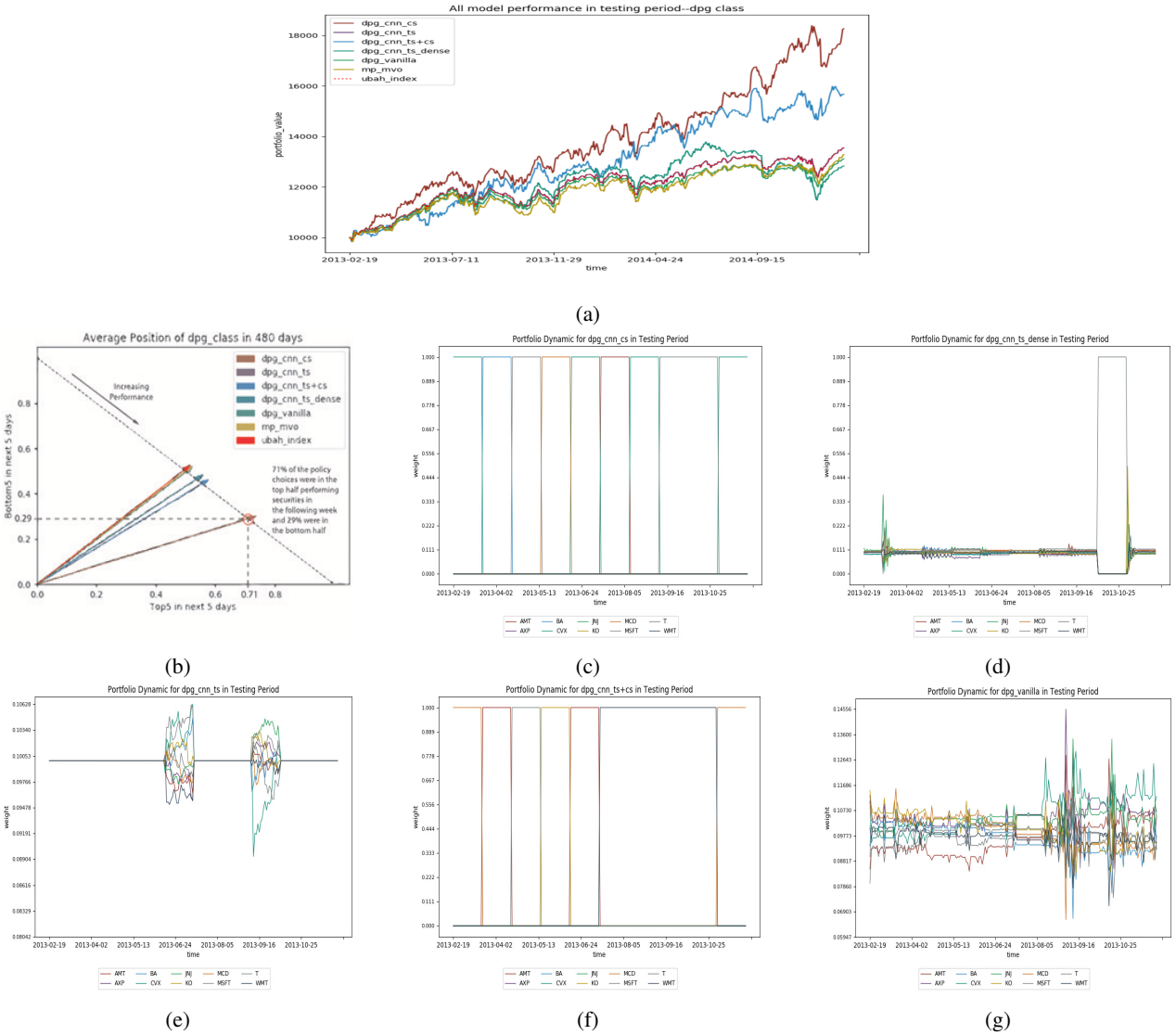

Figure 7: (a): Benchmarking DPG class returns against the performance of multi-period MVO and UBAH index; (b): Quality of DPG class RL policies; (c-d-e-f-g): Trading policy structure for DPG models.

cross-sectional module seems to not influence the PPO investment decisions to break down the equal weights. This could be explained by the nature of the TRPO and PPO algorithms that use a trust-region constraint in the policy space to discourage the RL agent from taking aggressive actions.

On the total return plots (a) the multi-period MVO closely tracks the market (UBAH index). The only RL model that outperforms the MVO and the UBAH index is the DGP class combined with the cross-sectional module. On the score plots (b), multi-period MVO and UBAH index are near the 50–50 point.

One of the contributions of this paper is the proposed methodology to evaluate the quality of deep RL policy models in finance. A positive total return does not necessarily imply that the corresponding policy is better and vice versa. As an example, we show in figure 10 (a) the green curve (PPO_Vanilla) which is the only declining curve (loss of money) among the other curves. The policy that corresponds to this declining curve is shown in figure 10 (g). This decline can be explained by the fact that the corresponding agent changed from a diversified approach to focus on one stock for a month from mid May 2013 to mid June 2013 and that stock performed poorly. Consequently, this had an impact on the overall performance due to the compounding effect. Even though the agent performed well afterward it never recovered from the loss incurred during that period.

Figure 10 (a) shows that total return can be deceptive. One bad selection makes the algorithm look much worse than the others, but figure 10 (b) shows that all the policies have similar scores although they have different total returns.

Our score provides insights into the reliability of the RL agents. The fact that these agents have similar scores but only one had a negative total return indicates that the other RL agents with the positive returns were just as likely to make a bad mistake during this test but were lucky enough not to. This mistake is due to the optimization approach used in neural networks. Gradient descent optimization can easily get stuck in a bad lo-

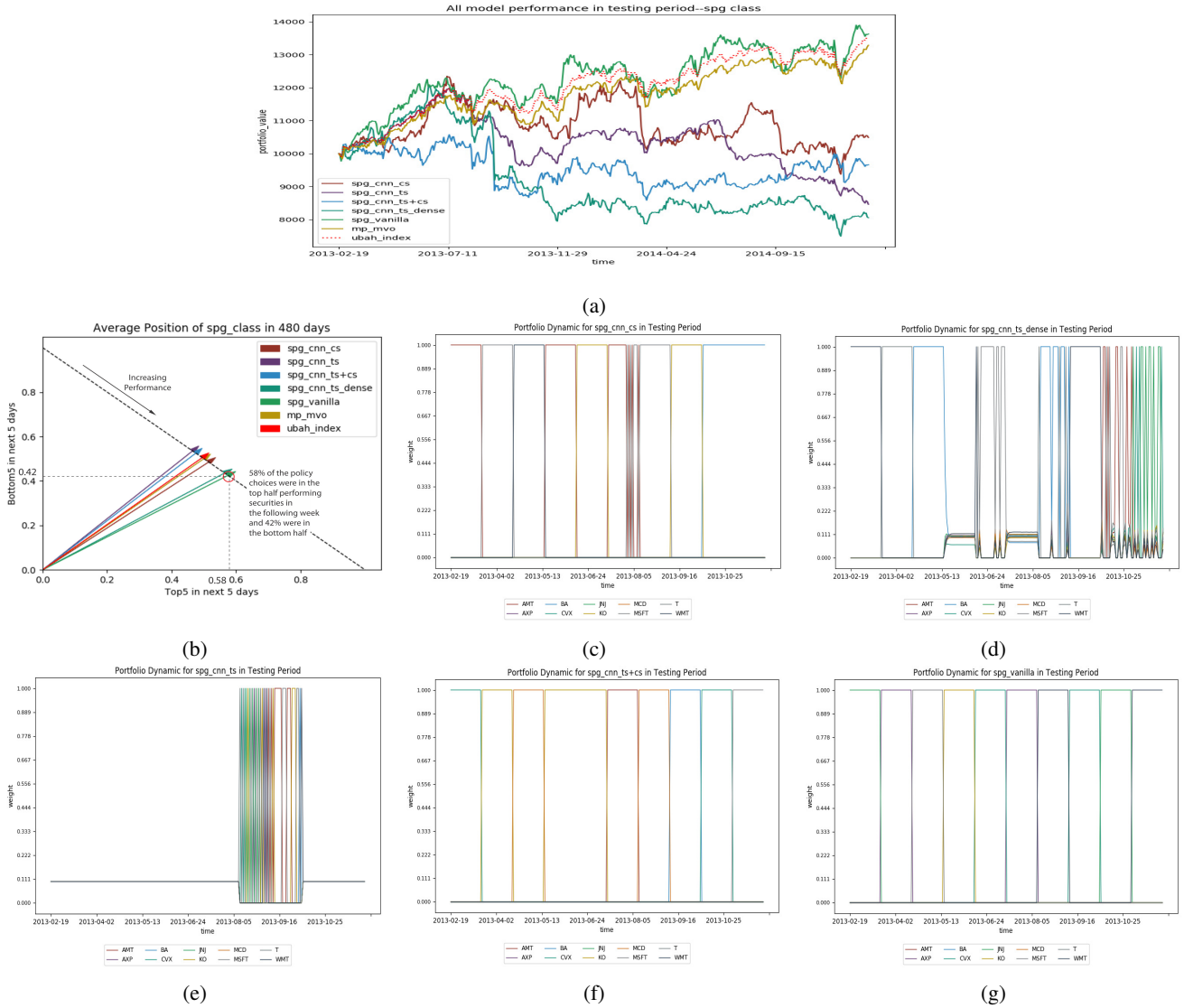

Figure 8: (a): Benchmarking SPG class returns against the performance of multi-period MVO and UBAH index; (b): Quality of SPG class RL policies; (c-d-e-f-g): Trading policy structure for SPG models.

cal optima which could lead the agent to make a bad choice. We believe our RL policy quality score provides valuable insight into the RL policies that is complementary to total return. Another contribution of this paper is the proposed cross-sectional deep RL framework. We determined that cross-sectional RL improves the quality of the RL policy and hence total return only in the case of an actor only class with the cross-sectional module. The addition a critic counteracts the benefits of cross-sectional RL. Another contribution of this paper is the proposed cross-sectional deep RL framework. We determined that cross-sectional RL improves the quality of the RL policy and hence total return only in the case of an actor only class with the cross-sectional module. The addition a critic counteracts the benefits of cross-sectional RL.

8. Lessons Learned

One of the key goals of this research was to understand the value of the cross-sectional RL to PM. The DPG RL class performed better overall and especially when combined with cross-sectional analysis where it consistently outperforms the benchmarks UBAH index and MP_MVO. SPG, DDPG and PPO outperform the benchmarks in some cases but not in others. Cross-sectional analysis does not help these classes. This is likely due to the fact that SPG, DDPG and PPO all randomize/shuffle the data during the training process for better generalization. This turns out to be incompatible with the cross-sectional methods because they require preservation of the order of the samples as is the case with DPG.

The trading patterns of the off-policy actor-critic RL algorithms (SPG and DDPG) seem to take the opposite approach compared to the on-policy actor-critic (DPG and PPO). We believe this is due primarily to the greediness levels of the algorithms. The off-policy actor-critic methods (SPG and DDPG) are very greedy and tend to choose a single security, which is a risky approach. For the on-policy actor-critic (DPG and PPO), the agents try to directly optimize the policy in a continuous

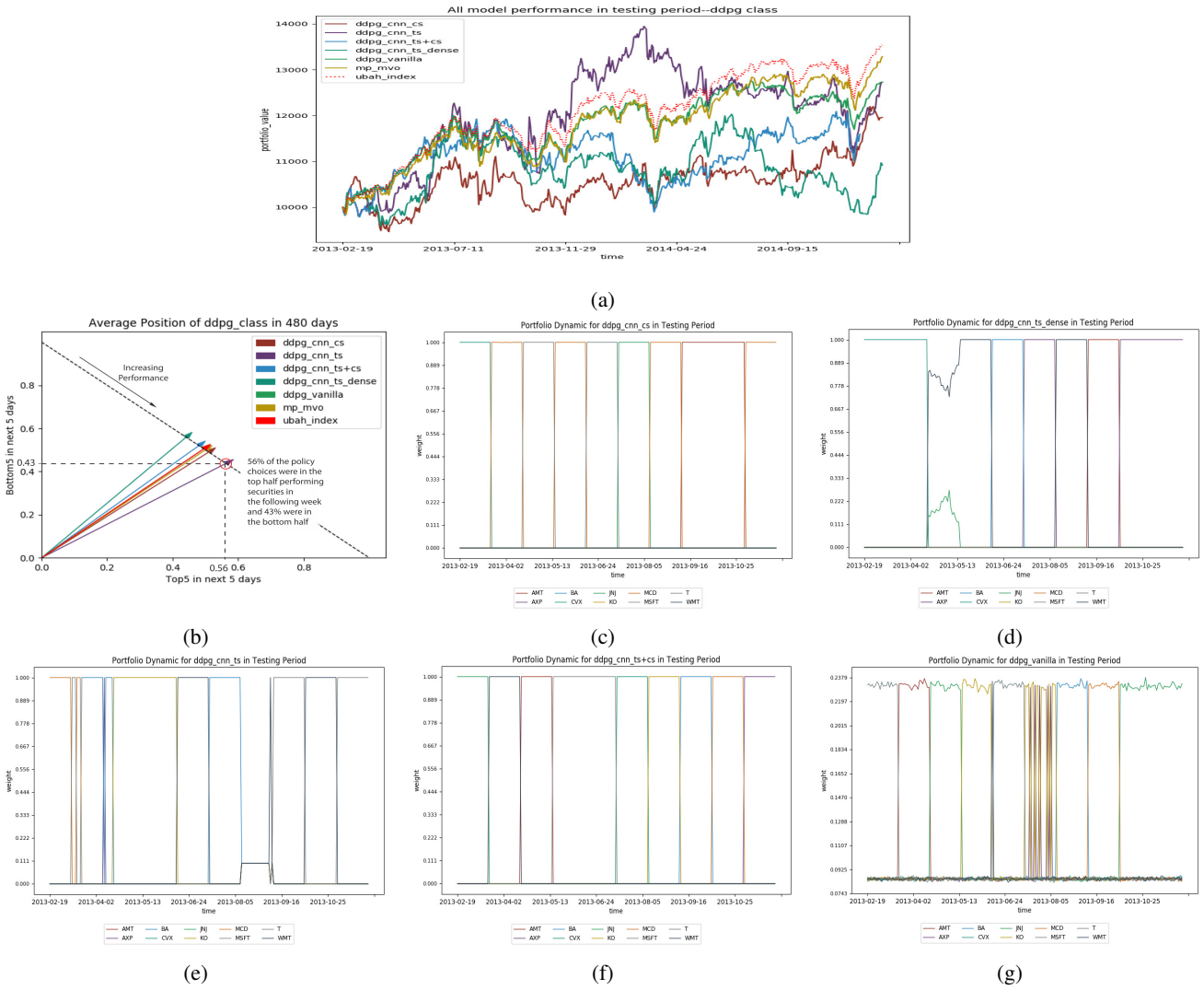

Figure 9: (a): Benchmarking DDPG class returns against the performance of multi-period MVO and UBAH index; (b): Quality of DDPG class RL policies; (c-d-e-f-g): Trading policy structure for DDPG models.

action space, in which case it is natural for the agent to incrementally adjust the policy ending up with a balanced portfolio pattern, hence reducing unsystematic risk.

(Liang et al. 2018) reported similar behavior based on return performance without taking risk into account when evaluating three of the eight RL algorithms we evaluated. They do not provide portfolio weights so we cannot evaluate the policy structure taken by the RL agents. We were able to successfully train our RL models by incorporating a risk-sensitive and multi-objective reward function (Equation (55)) where they were not successful.

We found it very interesting that our RL agents developed strategies that mimic well-known financial strategies. The variant CNN_TS for DPG and PPO models very closely follow the UBAH index (DPG_CNN_TS in figure 7 (a) and PPO_CNN_TS in figure 10 (a)). In addition, the DPG class without cross-sectional (figure 7 (d), (e), and (g)) and PPO class (figure 10 (c–g)) tend to use a strategy similar to equally weighted portfolios. So, we can imagine these RL methods could be a good approximation to ETF trading strategies. In addition, DPG_CNN_TS (figure 7 (e)) appears to duplicate Risk Parity models by largely using equal weights with occasional segments with different portfolio weight distributions.

While we found RL models performed well, we do not fully understand the stability of the algorithms and we would like to study further the circumstances that lead to bad decisions.

9. Conclusions

The instability of RL models has prevented them from deployment in real-time trading or widespread use in the financial world. We developed new algorithms based on the RL literature and successfully adapted them for use in portfolio management (section 6) by representing multi-type data through CNN channels. Future work may include adding multi-frequency data as well. We found that most of the actor-critic RL models cannot beat the MVO or UBAH. The only agents who were able to beat the market (UBAH) and the MVO strategy were the DPG class using the cross-sectional module. Our conclusion is that

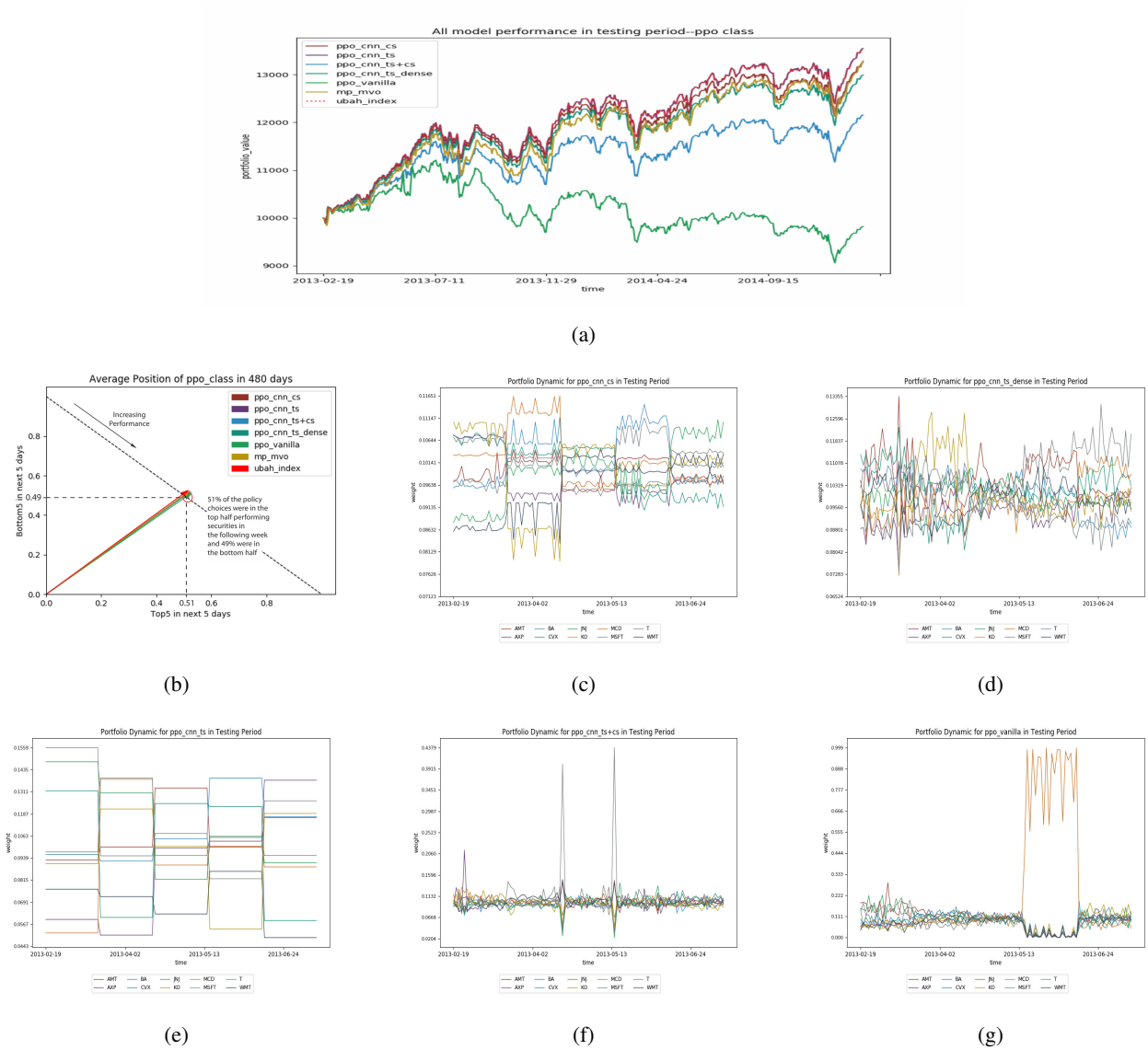

Figure 10: (a): Benchmarking PPO class returns against the performance of multi-period MVO and UBAH index; (b): Quality of PPO class RL policies; (c-d-e-f-g): Trading policy structure for PPO models.

cross-sectional approach only helps when combined with the DPG agent.

In finance, it is important for portfolio managers to understand the trustworthiness of an algorithm. Most researchers use total return as the performance metric for evaluating the performance of an algorithm. We believe total return can be deceptive and it should not be the only way to evaluate the performance of an RL algorithm. Our attempt to move the field forward focuses on assessing whether the actions taken by the RL agent make sense by evaluating the quality of the policy. We developed an RL Geometric Policy Score that goes beyond total return to give a complementary view point on the performance and reliability of the agents which can increase the trust of the portfolio manager.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) [grant number 411296449], Canada's federal funding agency for university-based research.

References

A. M. Aboussalah and C.-G. Lee. Continuous control with stacked deep dynamic recurrent reinforcement learning for portfolio optimization. *Expert Systems with Applications*, 2020.

P. H. Algoet and T. M. Cover. Asymptotic optimality and asymptotic equipartition properties of log-optimum investment. *The Annals of Probability*, 16:876–898, 1988.

S. Almahdi and S. Y. Yang. 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, 2017.

D. W. K. Andrews. Cross-section regression with common shocks. *Econometrica*, 73 (5):1551, 2005.

S. D. Bekiros. Heterogeneous trading strategies with adaptive fuzzy actor–critic reinforcement learning: A behavioral approach. *Journal of Economic Dynamics and Control*, 34(6): 1153–1170, 2010.

D. P. Bertsekas. *Dynamic programming and optimal control, vol. 1*. Athena Scientific, Belmont, 1995.

D. P. Bertsekas and S. E. Shreve. *Stochastic optimal control: the discrete-time case*, volume 23. Academic Press, Cambridge, 1978.

L. Breiman. Optimal gambling systems for favorable games. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 65–78, Berkeley, 1961. University of California Press.

G. Cornuejols and R. Tutuncu. *Optimization Methods in Finance*. Carnegie Mellon University, Pittsburgh, 2006.

T. Cover. Universal portfolios. *Mathematical Finance*, 1(1): 1–29, 1991.

Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning systems*, 28(3):653–664, 2017.

J. L. Elman. Finding structure in time. *Cognitive Science*, 14 (2):179–211, 1990.

T. G. Fischer. Reinforcement learning in financial markets - a survey. *Institute for Economics*, 2018.

I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews)*, 42(6): 1291–1307, 2012.

L. Györfi and D. Schäfer. *Nonparametric prediction*. Budapest University of Technology and Economics, Budapest, 2003.

L. Györfi, G. Lugosi, and F. Udina. Nonparametric kernel-based sequential investment strategies. *Mathematical Finance*, 16:337–357, 2006.

L. Györfi, A. Urbán, and I. Vajda. Kernel-based semi-log-optimal empirical portfolio selection strategy. *International Journal of Theoretical and Applied Finance*, 10:505–516, 2007.

M. Haugh and A. Lo. Computational challenges in portfolio management. *Computing in Science & Engineering, IEEE*, 3:54–59, 2001.

T. Hesterberg. Importance sampling in multivariate problems. In *In Proceedings of the Statistical Computing Section*, pages 412–417, San Francisco, 1987. American Statistical Association 1987 Meeting.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523, Rome, 2011.

Z. Jiang, D. Xu, and J. Liang. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv:1706.10059*, 2017.

J. L. Kelly. A new interpretation of information rate. *ell System Technical Journal*, 35(4):917–926, 1956.

V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *The conference on Advances in Neural Information Processing Systems*, pages 1008–1014, Denver, 2000.

B. Li and S. Hoi. Online portfolio selection: A survey. *ACM Computing Surveys. Research Collection School Of Computing and Information Systems*, 46(3):1–33, 2014.

H. Li, C. H. Dagli, and D. Enke. Short-term stock market timing prediction under reinforcement learning schemes. In *The IEEE international symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 233–240, Honolulu, 2007.

Z. Liang, H. Chen, J. Zhu, K. Jiang, and Y. Li. Adversarial deep reinforcement learning in portfolio management. *arXiv:1808.09940*, 2018.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1506.02971v4*, 2016.

D. W. Lu. Agent inspired trading using recurrent reinforcement learning and lstm neural networks. *arxiv*, 2017. doi: https://arxiv.org/pdf/1707.07338.pdf.

D. Maringer and T. Ramtohul. Regime-switching recurrent reinforcement learning for investment decision making. *Computational Management Science*, 9:89–107, 2012.

H. Markowitz. Portfolio selection. *The Journal of Finance*, 7: 77–91, 1952.

H. Markowitz. Portfolio theory as i still see it. *Annual Review of Financial Economics*, 2:1–23, 2010. URL https://doi.org/10.1146/annurev-financial-011110-134602.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

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

P. G. Necchi. *Policy gradient algorithms for the asset allocation problem*. Politecnico di Milano, Scuola di Ingegneria Industriale e dell' Informazione, Milan, 2016.

R. Neuneier. optimal asset allocation using adaptive dynamic programming. In *Advances in Neural Information Processing Systems*, Denver, 1996.

M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Son, Hoboken, New Jersey, 1994.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, Sydney, 2017a.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017b.

W. F. Sharpe. The sharpe ratio. *The Journal of Portfolio Management*, 21(1):49–58, 1994.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 32(1):387–395, 2014.

R. Sutton and A. Barto. *Reinforcement learning: An introduction*. The MIT Press, Cambridge, 2017.

B. Van Roy. Temporal-difference learning and applications in finance. In *Computational finance Conference, MIT press*, New York, 1999.

Y. Wang, D. Wang, S. Zhang, Y. Feng, S. Li, and Q. Zhou. Deep q-trading. *CSLT Technical Report-20160036*, 2017.

J. M. Wooldridge. *Part 1: Regression Analysis with Cross Sectional Data*. south-western cengage learning, Boston, 2009.

K. S. Zarkias, N. Passalis, A. Tsantekidis, and A. Tefas. Deep reinforcement learning for financial trading using price trailing. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE*, Brighton, United Kingdom, 2019.

J. Zhengyao and J. Liang. Cryptocurrency portfolio management with deep reinforcement learning. In *Intelligent Systems Conference (IntelliSys), IEEE*, London, United Kingdom, 2017.