Reinforcement Learning for Trading in Financial Markets
Theory and Applications

Hanwant Shekhawat
hanwantshekh@gmail.com

Abstract

With trading posed as a partially observable Markov decision process (POMDP), reinforcement learning (RL) offers a framework with which to develop trading strategies. This present study addresses theoretical considerations with regards to choice of state representation, reward definition and learning algorithms. Using idealised data generated from statistical processes, a suite of benchmark tasks are constructed to which development can be standardised. Using these tasks, experiments were conducted to determine the efficacy of several approaches as well as their limitations. To address the realistic objectives of a portfolio manager, a comparative exploration of different methods for risk management is performed, including Sharpe & Sortino based adjustment of rewards and risk distortion methods. Additionally, a novel method is proposed, termed proximal portfolio constraint (PPC), offering tuneable risk sensitive performance for general classes of RL algorithms as well as yielding a dynamic form which may be used in settings such as market making. The formalism and application of value based Deep RL methods in this study motivates further research and application in its use towards developing trading strategies for financial markets.

1 Introduction

1.1 Trading

Public financial markets provide venues of exchange between buyers and sellers of various financial assets. Although initially emergent from the relationship between producers and consumers of underlying assets, much of the traded volume in modern markets is contributed by agents looking to profit off changes in relative value between assets, henceforth termed *trading*. To execute a profitable trading strategy, an agent must posses an actionable *edge* which yields positive expected returns on a portfolio of assets, typically due to an informational and executional advantage over other market participants.

Examples of documented sources of edge include arbitrage opportunities whereby mispricings may occur for identical or related assets trading on different exchanges (Bistarelli et al., 2019), between a derivative contract and its underlying asset (Cummings and Frino, 2011), or between currencies with respect to their relative values (Aiba et al., 2002). Statistical arbitrage describes another class of strategies, whereby asset prices may be modelled as statistical processes yielding opportunities for trades based on statistical *anomalies*, rather than explicit mispricings. A classic example is considering the spread between prices of a cointegrated pair of assets as a mean reverting series (Alexander and Dimitriu, 2005; Avellaneda and Lee, 2010; Krauss, 2017). Along with these strategies which are independent of asset class, exogenous information aside from historical price may also be used. In stock markets, this has historically been the case whereby asset prices are causally associated with a business entity which can be independently analysed. Consideration of fundamental information pertaining to the financial state of a public company constitutes approaches such as factor-based investing (Jensen, Black, and Scholes, 1972). A key distinction between such an approach and the aforementioned arbitrage strategies is the risk profile associated with each. A long-term investor may exercise interest in the underlying asset or entity associated with the asset whereas traders often aim to be neutral with respect to any single asset as well as overall market conditions. As such, traders aim to minimise the negative volatility of their overall equity by constraints such as diversifying positions or minimising time spent in any one position, thereby minimising correlation between asset prices and their own equity. Such risk constraints increase the correlation between an agents equity and their source of edge.

With time and maturation, markets host larger numbers of participants, thus increasing competition between trading agents which can lead to degradation of previously profitable trading strategies. This selection pressure may result in agents specialising in particular bottlenecks within strategies (I.e execution speed), or expanding their set of strategies to consider different or higher order interactions between assets or exogenous variables. It has also been suggested that with increased competition, anomalies may persist but change in behaviour, leading to an evolving but endemic edge (Cross and Kozyakin, 2015). To remain profitable given such a trading environment, it is clear that along with identifying sources of edge, an agent must continually develop trading strategies which are dynamic with respect to market conditions, or specialise in niche areas where a competitive advantage is favourable.

1.2 Machine Learning

Given the problem setting of trading, the process required to discover and consolidate trading strategies is well suited to benefit from machine learning (ML) methods. In particular, algorithms can be used to optimise some objective such as prediction, learning to do so directly from data. The advantage of such approaches is that large amounts of data can be analysed without human labour or bias. A major disadvantage is that the solution space of ML algorithms is restricted by bias inherent in the algorithm and the high-level synthesis of different sources of information that a human trader may perform at their discretion is not trivial to implement in general purpose ML algorithms. Therefore, a combination of ML and discretionary trading is often employed, referred to as the 'quantamental' approach (De Prado, 2018, p. 53).

Indeed, ML methods are increasingly subject to much attention in the world of quantitative finance (Emerson et al., 2019; Snow, 2020). This is especially due to the recent renaissance of Deep Learning owing to Neural Network (NN) models and the software and hardware infrastructure required to implement them, enabling modelling of complex patterns within large datasets. NNs offer a set of tools which can be composed to form 'architectures' with structural priors complementary to a given problem, for example the use of Convolutional NNs (CNN) in image processing or Recurrent NNs (RNN) for sequential data such as time series (LeCun, Bengio, and Hinton, 2015). Given a sufficiently expressive representation space, the primary specification of the learning problem lies in the inputs to the NN and importantly, the objective over the output which the NN 'learns' to optimise. Supervised ML objectives train models for a prediction task such as regression or classification for continuous and discrete targets, respectively. Examples of relevant targets of use for a trading agent may be price returns (relative price change) for regression, or direction of price movement for classification (Tsantekidis et al., 2017; Zhong and Enke, 2019). Such predictions can subsequently be used to inform or enhance a trading strategy. Although this may seem like a promising application, the numerous implementation details of a trading strategy are not trivially solvable by these methods. Specifications for a suitable prediction objective must be chosen, along with conditions for position entry, size and exit. The resultant strategy must then have a statistically justified expectation of positive risk adjusted returns, following consideration of transactions costs, slippage, liquidity restrictions and overall risk profile.

1.3 Reinforcement Learning

Although supervised ML methods provide powerful tools with which trading strategies can be constructed, the process is rather ad-hoc, in that it does not provide a general method which optimises the objective of a trading agent directly. To this end, reinforcement learning (RL) is a particular subset of ML which can offer such a generality. A general overview of RL is provided here, for a comprehensive introduction, please see Sutton and Barto, 2018.

The RL setting is devised for problems characterisable as Markov Decision Processes (MDP). As a discrete time sequential process, an MDP has 4 main components often presented as a tuple (S, A, Pr, R) which for any given timestep t, describes the set of possible states $s_t \in S$, set of possible actions $a_t \in A$, state transition probabilities $Pr(s_{t+1} = s'|s_t = s, a_t = a)$ and a reward function $R(s'|a_t = a, s_t = s)$. This turns the ML problem from one of prediction to one of optimal control where an *agent* interacting with an *environment* aims to maximise rewards r_t received as a result of decisions following its internal policy $\pi(s)$ (Sutton and Barto, 2018). To learn optimal behaviour, agents must be incentivised to not just maximise the immediate reward, but the expected aggregate of future rewards resulting from its actions. For episodic MDPs with finite lengths, a simple sum of future rewards can be used to provide feedback to the agent for a given timestep. In continuous settings, as trading may be intended, a discount factor $\gamma \in [0, 1)$ must be used to prevent the sum of future rewards from diverging[1]. This is generalised in the concept of return R:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1}$$

Thus the goal of an RL agent is to find the optimal policy π_* such that it maximises R.

The methods for solving this optimisation problem can be broadly categorised into two classes; policy and value-based. Policy methods aim to directly learn a policy over the action space which maximises return:

$$\pi(a, s) = Pr(a_t = a|s_t = s) \tag{2}$$

where $\pi(a, s)$ denotes a policy which specifies probabilities for actions given an input state. The action space for a policy may be discrete (I.e Buy or Sell) or continuous (I.e purchase $\pm x$ units or a desired portfolio weighting). Value based methods on the other hand learn the expected return under a given policy π, conditioned on either state in the case of state value functions $V_\pi(s)$ or both state and action for action value functions $Q_\pi(s, a)$. Formally,

$$V_\pi(s_t) = \mathbb{E}_\pi[R_t|s_t = s] \tag{3}$$

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a] \tag{4}$$

[1]As γ approaches 1, future returns are given more weight and conversely as it approaches 0, more weight is given to the most recent observations.

Predicting the expectation of R_t as defined in (1) is increasingly impractical with respect to the number of time steps over which rewards are aggregated, especially in the continuous case. Additionally, aggregation over too large a look-ahead period entails higher variance in R_t, diluting the learnable effect of any one action. Hence, n-step reward structures are often used for R_t via bootstrapping as such:

$$R_t = \sum_{k=0}^{k=n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}) \tag{5}$$

with n=1 being the foundational case for temporal-difference (TD) learning.

Given an optimal $Q_\pi(s, a)$ function, a so-called greedy policy can be derived from Q_π as:

$$\pi(s_t) \sim \arg\max_a Q_\pi(s_t, a) \tag{6}$$

which in turn can be used to bootstrap R_t estimates whereby $\arg\max_a Q(s_{t+n}, a)$ is used in place of $V(s_{t+n})$. The recursive dependency of Q_π and π, along with the recursion induced by bootstrapping value estimates in estimation of R_t, necessitates dynamic programming based techniques for which the conditions for convergence have formed the theoretical foundations for value based RL algorithms (Watkins and Dayan, 1992). Value functions can also be used with policy based methods via bootstrapping to form better R_t estimates as in actor-critic algorithms (Sutton and Barto, 2018, p. 331). The current work explores action value based methods as a preliminary step in applying RL methods in trading.

1.4 Reinforcement Learning for Trading

In the trading setting, the environment is an abstract entity performing the duties of an exchange/broker which sends transaction or quote data for assets to the agent, executes transaction orders, determines accounting pertaining to rewards, as well as being the source of any other information which an agent may use. A key assumption of an MDP is that the current state contains all the information required to determine the next state. This is clearly not realistic when considering financial assets where all the various determinants of price are neither observable, deterministic, nor tractable in any sense. Thus, the environment is posed as a partially observable MDP (POMDP) where the partial states sent by the environment are instead referred to as observations $o_t \in \Omega$ which derive from an underlying true state $\sim O(o|s', a)$ (Kaelbling, Littman, and Cassandra, 1998). The main considerations in the supervised ML setting of specifying inputs and objective functions persist as the representation of o_t must be carefully considered along with the reward function which motivates the agent. Despite these similarities, RL offers the advantage that the objective function for the model being trained is no longer a proxy from which a trading strategy can be recovered, it can be formulated as the direct objective which a trading agent aims to maximise in interaction with an environment. To this end, the current work presents experiments conducted using these methods to develop agents to trade synthetic price series, generated from statistical processes. Rather than a model of asset price behaviour, these settings are devised to study the capability of these algorithms in identifying certain types of edge and subsequently forming trading strategies. In a real-world application, state representation may differ vastly from the sequential time series approach explored in the present study.

2 Methodology

This section specifies design choices for the POMDP and learning algorithms explored in subsequent sections[2].

2.1 Rewards

To construct R_t such that it best represents the goal of a trading agent, rewards were based on equity returns, the requisite definitions for which follow. The environment keeps a ledger of asset holdings $n_t \in \mathbb{Z}^N$ for the account associated with the agent, termed the portfolio. Holdings in terms of dollar value $h_t \in \mathbb{R}^N$ are computed from the point-wise product of assets holdings and prices: $h_t = n_t p_t$. The net asset value av_t can therefore be calculated as the sum $\sum_{i=0}^{N-1} h_t^i$ or as the dot product $n_t \cdot p_t$. Given

[2]All components were programmed in Python/C++. For implementing NN components, the PyTorch framework was used (Paszke et al., 2019), code available at request.

that positions may be entered using margin or leverage, the net equity value of the account is

$$v_t = av_t + balance_t \tag{7}$$

where $balance_t = cash_t - borrowed\_margin_t$. Thus, equity returns from which reward functions can be derived are based on the difference $\delta v_t = v_t - v_{t-1}$. There are two ways that v_t can change between sequential timesteps; following a change in price δp_t of held assets or via exchange of assets incurring transaction $cost_t$. Hence

$$\delta v_t = n_{t-1} \cdot \delta p_t - cost_t \tag{8}$$

Transactions incur a cost at time t which in this work was computed relative to the dollar value of the transaction. The cost incurred by slippage whereby prices move against the direction of exchange is implicit in δp_t [3].

A direct conversion of equity returns to rewards for an agent can be computed as:

$$r_t = log(\frac{v_t}{v_{t-1}}) \tag{9}$$

as per the commonly used log return measure (Platen and Rendek, 2008). It is widely understood in the financial world that an investment vehicle such as a trading strategy should be evaluated not just with respect to raw returns but also the risk associated with the strategy. Given that this is the criteria by which a strategy is evaluated, this should be reflected in the rewards provided to an RL trading agent. One such adaptation to equity returns is the Sharpe ratio:

$$Sharpe_t = \frac{\mathbb{E}[r^s_t - r^b_t]}{\sqrt{var[r^s_t - r^b_t]}} \tag{10}$$

where r^s is the equity return for the strategy under consideration and r^b is a baseline chosen at discretion I.e the risk-free rate of return, an index or the returns on the asset being traded (Sharpe, 1994). This measure adjusts returns by their standard deviation, representing the volatility of a strategy. Adapting the

[3]Unless specified, experiments were conducted with a cost of 2% of the dollar value of the transaction, with no slippage.

Sharpe ratio for volatility-adjustment of (5):

$$R_t^{sharpe} = \frac{\sum_{k=0}^{k=n-1} \gamma^k (r_{t+k}^s - r_{t+k}^b)}{\sqrt{\frac{\sum_{k=0}^{k=n-1} \gamma^k (r_{t+k}^s - r_{t+k}^b)^2}{n-2}}} + \gamma^n V(s_{t+n}) \tag{11}$$

A limitation of the Sharpe ratio is that desirable volatility such as large positive returns can lead to a deflation of the performance measure. To address this, the Sortino ratio adapts Sharpe by using downside deviation (standard deviation of negative returns) as the denominator, thereby only adjusting for undesirable volatility:

$$\begin{gathered} R_t^{sortino} = \frac{\sum_{k=0}^{k=n-1} \gamma^k (r_{t+k}^s - r_{t+k}^b)}{\sqrt{\frac{\sum_{k=0}^{k=n-1} \gamma^k DR_{t+k}}{n-2}}} + \gamma^n V(s_{t+n}) \\ where\ DR_t = \begin{cases} (r_t^s - r_t^b)^2, & \text{if } (r_t^s - r_t^b) < 0 \\ 0, & \text{otherwise} \end{cases} \end{gathered} \tag{12}$$

Despite being a direct formulation of an investor's objective, a major disadvantage of these risk-adjusted aggregations of R_t is that the horizon over which returns are aggregated must be sufficiently large such that the resultant term is not dominated by the variance inherent in returns (Lo, 2002). A large value for n could be chosen to mitigate this however if the time series is such that the result from an action is expected to transpire over less time steps, then a longer aggregation period results in dilution of the effect of the action resulting in greater variance (Sutton and Barto, 2018, ch7). Hence there are two sources of variance which have opposing relationships with the number of aggregation steps, though potentially yielding a stable range depending on the time series in question. A theoretically better formulation for risk adjustment which avoids this source of variance is the differential Sharpe ratio (DSR) and the Sortino analogue, the differential downside deviation ratio (DDR), first introduced in (Moody et al., 1998) and (Moody and Saffell, 2001) respectively. By maintaining incrementally updated estimates of $\mathbb{E}[r^s - r^b]$ and $var[r^s - r^b]$, the contribution of a future return to the estimates is computed and used as the reward from which R_t is aggregated. Intuitively, rather than the Sharpe or Sortino of the aggregated equity returns, this *differential* update computes how the future returns affect the running estimates of Sharpe and Sortino ratios, thereby removing the aforementioned source of variance. Details of the updates and mathematical derivation can be found in (Moody and Saffell, 2001).

2.1.1 Proximal Portfolio Constraint

In managing a portfolio, traders often have portfolio constraints which are exogenous to individual strategies. For example, despite a strategy having internal specifications for managing risk, the aggregate of positions attributable to independent strategies may lead to undesirable exposure to overall market conditions. As there are many such scenarios which necessitate portfolio adjustments independent of expected return, the proximal portfolio constraint (PPC) is proposed. When used in tandem with any of the previously detailed reward aggregations, this method provides the agent with an additional reward which is inversely proportional to the distance of its portfolio to a specified target portfolio. Using cosine similarity as the measure of distance, the aggregated rewards are thus:

$$R_t^{PPC} = \sum_{k=0}^{k=n-1} \gamma^k (r_{t+k} + \alpha_{t+k} \cdot cos(port_{t+k}, target\_port_{t+k})) + \gamma^n V(s_{t+n}) \tag{13}$$

where α is termed the temperature parameter and cosine similarity is

$$cos(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

Along with being composable with the previous methods, a benefit of this approach is that the target portfolio and α parameters can be varied and explicitly included in the state representation, allowing an agent to respond to dynamic portfolio requirements. In the preliminary experiments conducted, both parameters are held constant to determine the efficacy of the approach in discouraging an agent from trading away from a portfolio of only cash holdings unless the expected returns are higher than the penalty incurred by deviation from the target portfolio. Thereby, the method is directly compared with the Sharpe and Sortino based aggregations in achieving similar objectives.

2.2 Observations

The environment encompasses any component of the learning setup that an agent can observe or interact with but cannot optimise directly. In the experiments conducted, at each discrete timestep, the environment yields an array of prices for each tradeable asset $p_t = \{p_t^1, ..., p_t^N\}$ where N is the number of assets. These are concatenated using a sliding window of size K into observations $o_t = \{p_{t-K}, ..., p_t\}$. This set of observations can be further processed by extracting time series features such as those from technical analysis classically used in finance (Lo, Mamaysky, and Wang, 2000) or more generally, signal processing techniques (Feng and Palomar, 2016). For sake of simplicity and restricting inference to the capabilities of the learning algorithms, experiments were conducted without pre-processing other than a log transformation for series which exhibited large price movements. Given that the pre-processing steps can't be directly optimised by the agent, they are implicitly considered to be part of the environment rather than the agent. In addition to asset prices, a normalised ledger corresponding to $\frac{h_t}{v_t}$, concatenated with normalised cash holdings $\frac{balance_t}{v_t}$ is provided to the agent. This vector of portfolio weights provides the minimal accounting information required for managing a portfolio.

2.3 Interface

To maintain a consistent interface between the environment and any type of agent, the environment is implemented to accepts orders in the form of a vector of desired units to purchase $u_t \in \mathbb{R}^N$ where N is the number of assets [4]. With this interface, it is up to the agent to translate its raw model outputs to transaction units and thus the environment is agnostic to the action space of the agent. The implementation is event based whereby in response to an action sent by an agent, the environment sends the next state, a reward and any information regarding account status. The interaction between an agent and environment continues until termination criteria are met, entailing end of the otherwise continuous 'episode'. For example, if an agent incurs losses on its holdings of greater value than the amount of cash balance in its

[4] In these preliminary experiment, fractional units are permitted

account, a margin call is invoked and the episode ended. This is designed to impose realistic portfolio management criteria. Given a premature ending, a negative reward of -.1 is heuristically given, enhancing the prediction objective for the value based agent in this scenario.

2.4 RL Algorithm

The main agents considered in this work are based upon Q-learning whereby an agent learns an optimal action value function $Q^*(s,a)$ which is subsequently used to derive the optimal policy which maximises R_t:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{14}$$

In particular, agents are based on Deep Q-Learning Networks (DQN) which leverage the representation capabilities of NNs in the RL setting. DQNs have been shown to be able to solve high-dimensional problems such as playing Atari video games using just image representations of the input screen, doing so as well as humans (Mnih et al., 2015). Numerous extensions to the base DQN algorithm have been made with independent performance improvements, a combination of which culminates to the Rainbow DQN architecture (Hessel et al., 2018). All components of Rainbow DQN are utilised in this work except for C51 which is an extension enabling the agent to predict not just a single scalar from $Q(s,a)$, but a distribution of values. Instead, a more recent and improved formulation is used, termed implicit quantile networks (IQN) (Dabney et al., 2018). Along with better general performance on a variety of benchmarks, IQN allows a risk-sensitivity prior to be imposed upon the agent when interacting with the environment, which can bias it towards risk-averse or risk-seeking behaviour.

A major feature of the DQN algorithm is experience replay, whereby interactions with the environment are stored in a sliding window of memory. These experiences are then sampled during the learning phase to leverage the data history generated from agent-environment interaction. There are many such details with respect to specification of the full DQN algorithm which are omitted here, but are critical to learning and importantly, offer opportunity for customisation. Despite the momentum of RL research towards general purpose learning algorithms, domain expertise is vital in engineering solutions for real-world settings, akin to the quantamental approach.

2.5 Action Space

Action value based methods operate on discrete action spaces. The outputs for the agents were constructed to have 4 semantic choices: Hold (do nothing), Buy, Sell, Close (will either Buy or Sell or do nothing). The number of specified action *atoms* allows for integer multiples of the base transaction size for Buy and Sell actions. To determine the base transaction size, a pre-set proportion of the available margin is used[5]. Multiple action atoms give agents the choice to either scale in or enter into large transactions immediately. Experiments in this work were conducted with just one atom, for a total of 4 actions.

2.6 NN Model

For the role of function approximation which is the central 'learner' in the RL algorithm, a CNN based architecture was used. When using 1-dimensional kernels, CNN based architectures have been found to perform well in time series domains (Zhao et al., 2017). Treating asset prices as input channels, the model used 2 convolutional layers with 32 channels each, outputting a latent encoding of price. The normalised portfolio ledger is also embedded into a latent space, using a simple fully connected (FC) layer. Interactions between these representations are enforced via element-wise multiplication, leading to a full state embedding which an output head [6] module transforms into a final vector corresponding to state-action values, from which the max element can be taken to provide an action given the state.

[5]For example, if 100,000 is currently available in Margin and a Buy action is chosen with a pre-specified proportion of .05, a transaction order or 5,000 is sent to the environment

[6]The output head may be a single FC layer or in the case of dueling networks as was used in this work, a combination of two independent FC layers.

3 Experiments

This section outlines results obtained from training the Rainbow based extension to the base DQN algorithm on environments with various data generating processes. Specifically, this extension makes use of IQN for distributional RL, noisy nets for exploration, dueling and double DQN for value estimation stability, n-step returns and prioritised experience replay (Hessel et al., 2018) [7]. The agents are trained with the objective of maximising equity returns, until subsection 3.5 where risk adjusted returns are explored. Throughout training, agent performance is evaluated every 5K steps and the metric of mean returns per episode is presented with approximate confidence intervals[8]. These idealised settings provide a suite of increasingly difficult problems that new agents can be tested on. Additionally, performance with different data processes provides insight into the types of statistical edge the agents can identify as well the trading strategies that are learned. Such benchmarks are important as the full problem of trading financial markets presents so many factors that lack of performance of any RL method can be difficult to diagnose.

3.1 Mean Reversion

Sine Wave

Figure 1b shows the equity curve for a DQN agent trained to trade a periodic price series. Although the exponential growth in equity is achieved early, it is interesting to note the evolution of the trading strategy from training steps 10000 (c-e) to 20000 (f-h). At 10K steps, figure 1d shows the agent continuously buying when the series is rising from a low and closing the accumulated position (large negative spikes in transaction) when the series is at its easily predictable high. At 20K steps, the agent learns to profit from the downward movement as well, establishing a short position after closing its net long, and vice versa. As only 3 total action atoms are provided to the agent corresponding to -1, 0 and 1 multiples of a small proportion of available margin, the agent learns to compound positions to maximise profit. To

[7]Hyper-parameters are provided in appendix A

[8]Calculated by taking the 50-period rolling sd. of mean returns per episode

note is the oscillations in the equity curve, which the agent tolerates though is not ideal in minimising downside volatility.

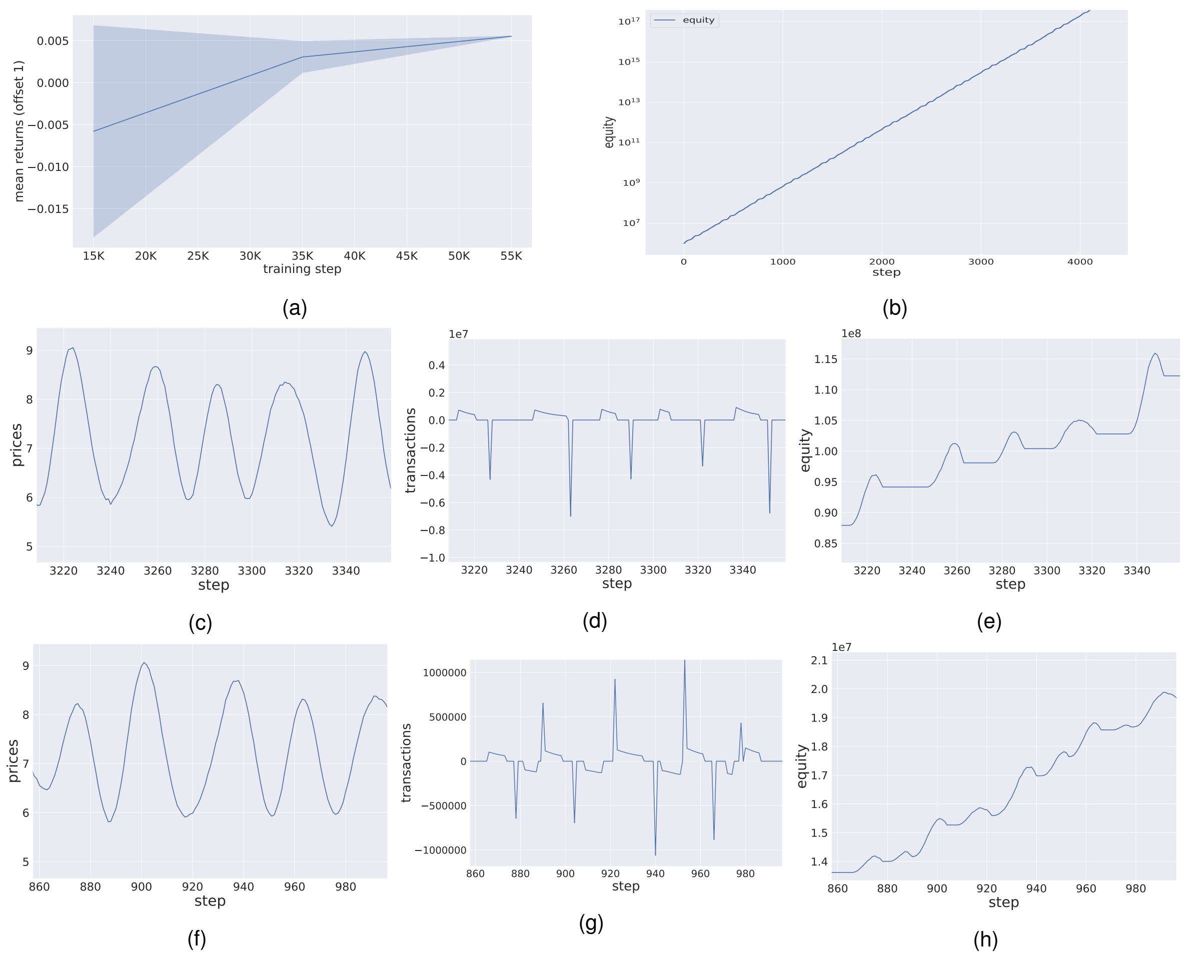
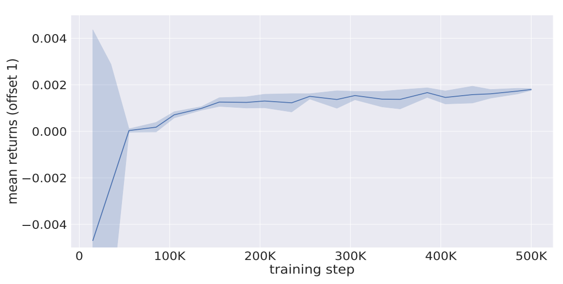
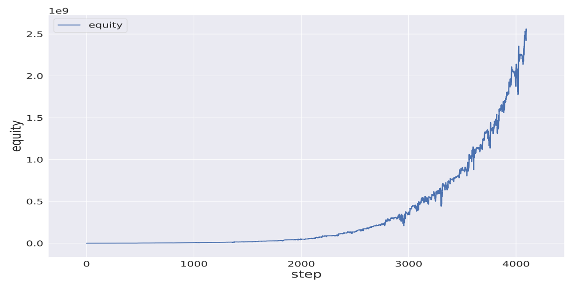


Figure 1: Agent trained on periodic sine wave for ~ 60K training iterations. (a) Mean equity returns for test episodes taken periodically throughout training . (b) Equity Curve (log scale) for a test episode lasting 4096 steps, taken at end of training. (c-e) Price, transaction and equity for ~140 steps of a test episode after 10K training steps. (f-h) Price, transaction and equity for steps ~140 steps of a test episode after 20K training steps.

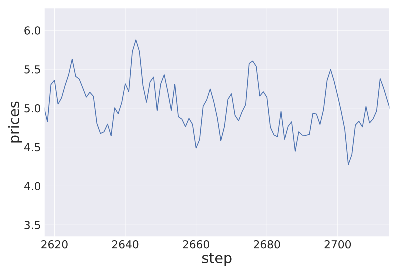
Ornstein Uhlenbeck Process



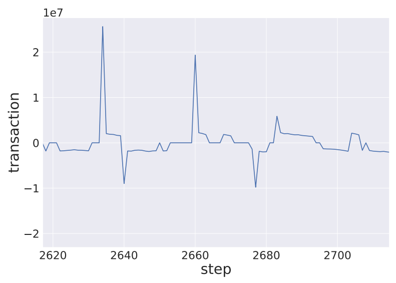
(a)



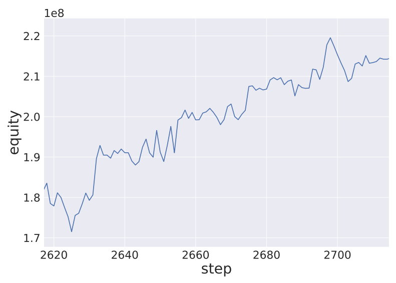
(b)



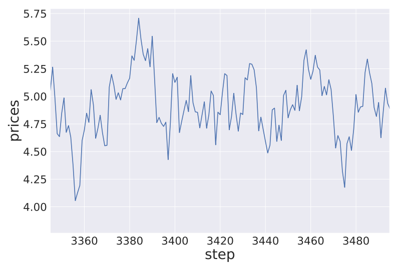
(c)



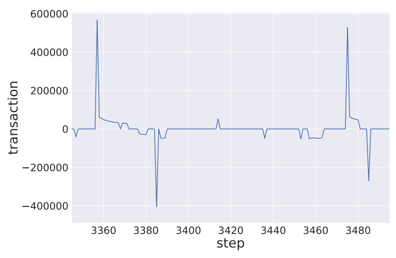
(d)



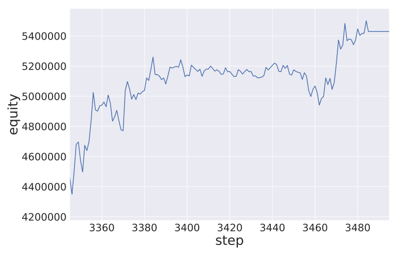
(e)



(f)



(g)



(h)

Figure 2: Agent trained on an OU process for 500K training steps. (a) Mean equity returns for test episodes taken throughout training. (b) Equity Curve for a test episode lasting 4096 steps, taken at end of training. (c-e) Price, transactions and equity for ~100 steps of a test episode at end of training. (f-h), Price, transactions and equity for ~140 steps of a test episode taken at the end of training an agent trained on the same series, with the same hyper-parameters and number of steps but with a relative transaction cost of .05 instead of .02. Distribution parameters were $\mu = 5$, $\theta = .15$ and $\sigma = .2$.

The OU process is a stochastic mean reverting process x_t, described by:

$$dx_t = -\theta(x_t - \mu)d_t + \sigma dW_t \tag{15}$$

where μ is the mean around which the process is centered, $\theta > 0$ and $\sigma > 0$ are parameters corresponding to degree of mean reversion and noise respectively and dW represents a random walk process. Figure

2 shows training and test results for an agent trained on OU series. While the agent training on the deterministically mean reverting sine series learned optimal trading strategies almost immediately, it was not until ~ 80K training steps that the OU agent reached profitability. Similar to the sine series, figure 2d shows that the learned trading strategy essentially accumulates net long and short positions when the series is expected to rise and fall, respectively. The agent seems comfortable accumulating positions and waiting for the opportunity to exit and reverse, which always comes with such a process, though it must do so in expectation that the mean entry price relative to its mean exit price covers transaction costs. The agent for which results are displayed in figures 2(a-e) was subject to a relative transaction cost of .02 while figures 2(f-g) show the trading strategy for an agent which other than being subject to a transaction cost of .05, was identically trained. It is clear that the strategy is more conservative in actions taken, choosing to hold much more frequently and entering transaction much more sparsely. The total amount of time spent in transactions however, was comparable for both agents.

3.2 Trends

Simple Trend

Trending series are constructed by devising a simple formula based on the sentiment that price direction which has consistently persisted in the past, tends to persist for a while into the future. This is not necessarily a model of real asset prices, though the general notion has been shown to underlie some long-term price behaviour as well as institutional investment strategies (Lempérière et al., 2014; Baltas and Kosowski, 2013). The algorithm used to construct the series is provided in B.1. As seen in figure 3, the agent executes less actions than the OU strategy though similarly accumulates positions before exiting once the trend stops persisting. Although, there is typically a delay where the agent tolerates small draw-downs before fully exiting the position.

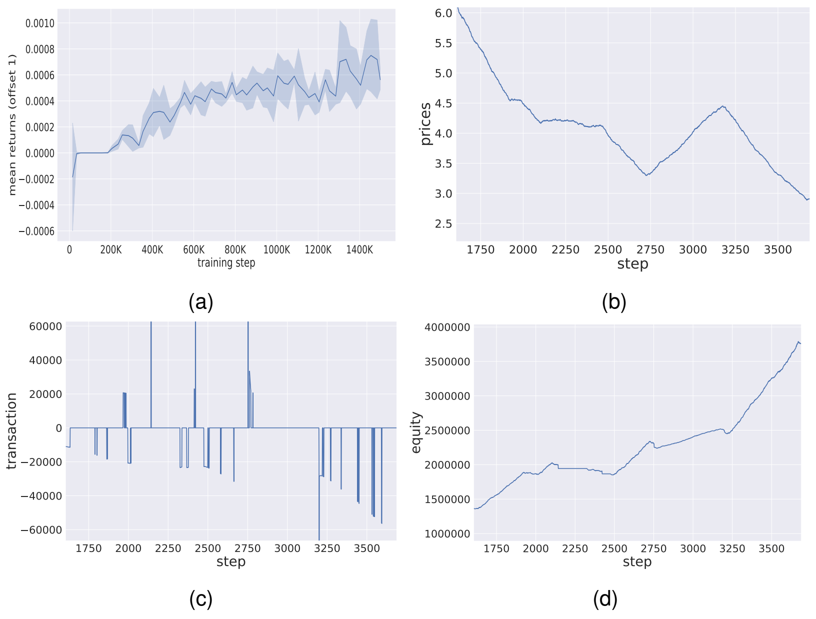


Figure 3: Agent trained on a trending process for ~1.5M training iterations. (a) Mean equity returns for test episodes. (b-d) Price, transactions and equity for steps ~1700 - 3600 of a test episode conducted after 500K training steps. Algorithm parameters (see B.1) were $trendProb = .01$, $minTrendLen = 100$, $maxTrendLen = 500$, $minDy = .001$, $maxDy = .005$ and $noise = .001$.

3.3 Composite Series

The following experiments pertain to data generating processes which possess a combination of the characteristics explored in previous sections.

TrendOU

TrendOU denotes a series which switches from regimes of mean reversion and trend persistence, identical to the OU and trending processes described in the previous section, respectively. The training progress for this series was much more variable than the previous series, as shown in figure 4a. The trading strategy appears to be a combination of the individual strategies for the OU and trending series, as shown in figures 4(b-e). At step ~3620 the agent incurs a loss on a net negative position (figure 4e) which is sustained until an incremental reversal, eventually maintained as a net long position in response to the upwards trend, showing an adaptive response to a trend occurring while implementing a mean reverting strategy. Note that the agent does not receive explicit profit and loss information and so reacts primarily to price in such scenarios.

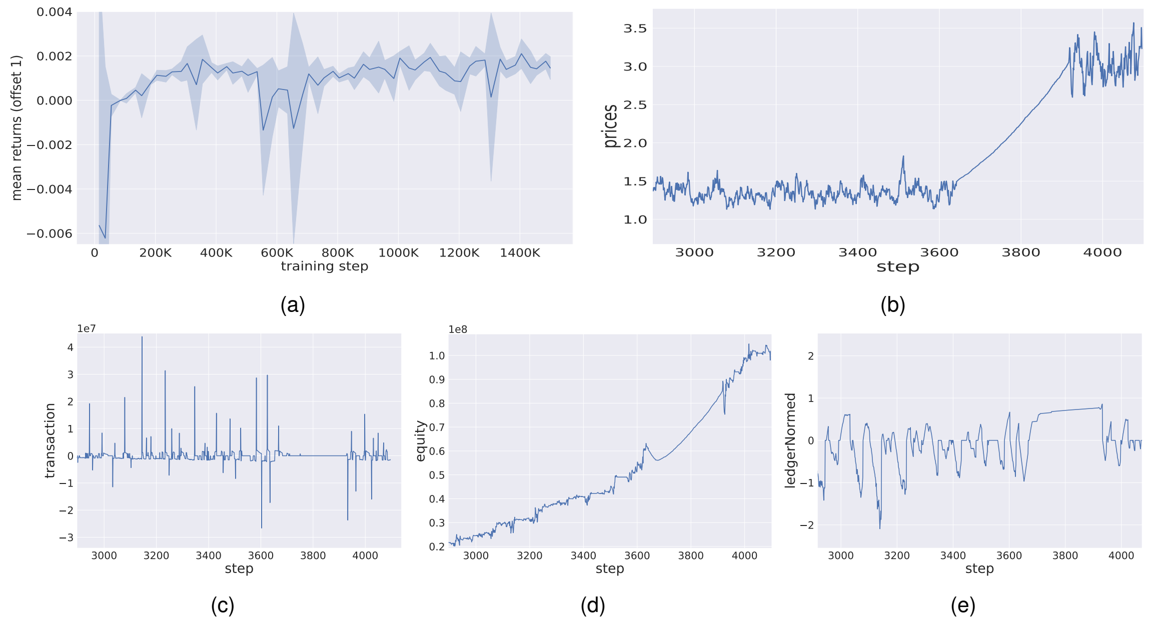
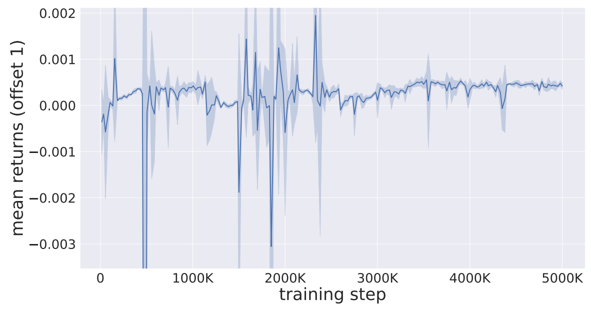


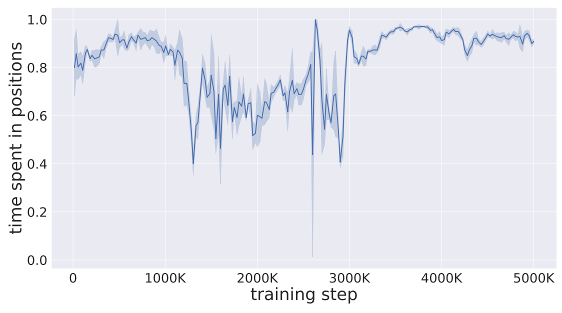
Figure 4: Agent trained on a TrendOU process for ~ 1.5M training iterations. (a) Mean equity returns for test episodes. (b-e) Price, transaction, equity and the normalised ledger from steps ~ 3000-4000 for the a test episode performed after 500K training steps. The ledger represents the value of the position normalised to the net equity in the account. Data Process parameters were the same as the previous OU and Simple Trend series except with $trendProb = .001$ and $\sigma = .02$ as a proportion of the current value thus implementing a geometric OU process.

TrendingOU

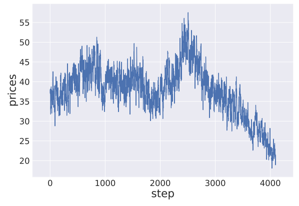
The TrendingOU series exhibits mean reverting behaviour concurrently with trends, detailed in B.2. Figure 5 shows training progression both in terms of overall performance as well as style of trading as indicated by time spent in positions. In figures 5(c-f), a test episode taken at ~ 2.5M steps shows an agent successfully trading the mean reversion component of the series though with an exclusively long bias (figure 5f). As shown in figures 5(a-b), this period was amidst some training instability in mean returns as well as time spent in positions. Following this period and at the end of training, figures 5(g-j) show that the trading strategy adapts to the overall trend, by accumulating both net long and short positions in response to trends while simultaneously trading the mean reverting component. To achieve sensitivity to the overall trend, an augmentation to the state observations o_t was made where in addition to concatenating consecutive observations $p_t = \{p_{t-K}, p_{t-K+1}, ..., p_t\}$, the agent was also given a strided series $p_t^s = \{p_{t-(K \times s)}, p_{t-((K-1) \times s)}, ..., p_t\}$, with stride $s = 5$, enabling both short and long time windows to be considered in parallel without modification of the NN architecture. Although not shown, this adjustment was found to be important in achieving the adaptability to trends shown in figures 5(f-h). This illustrates the role of domain expertise, whereby the information provided to any learning algorithm must be tailored to be able to represent the relationships which are sought.



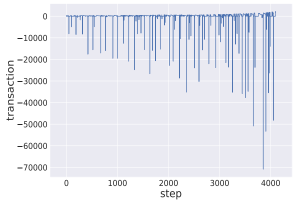
(a)



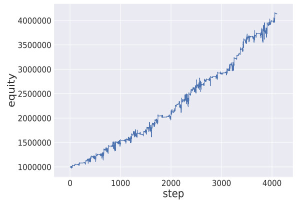
(b)



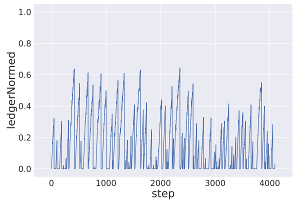
(c)



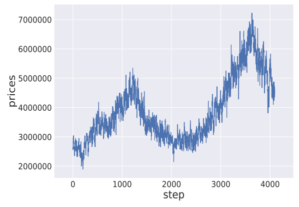
(d)



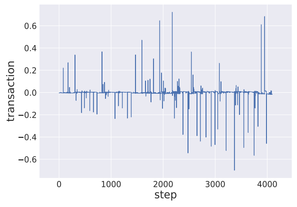
(e)



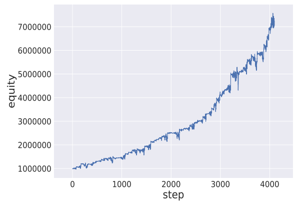
(f)



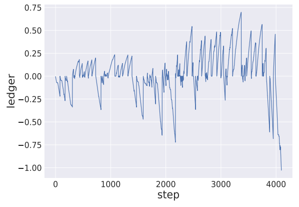
(g)



(h)



(i)



(j)

Figure 5: Agent trained on a TrendingOU process for ~5M steps. (a) Mean equity returns for test episodes throughout training. (b) total proportion of time spent in test episodes. (c-f) Prices, transactions, equity and normed ledger respectively, for a test episode taken at ~ 2.5M steps, showing a long bias.(g-j) Prices, transactions, equity and normed ledger respectively, for a test episode taken at ~ 5M steps, showing adaptability to trends by accumulating net long/short positions. Algorithm parameters were the same as TrendOU except $trendProb = .01$, $minDy = .0005$ and $maxDy = .001$.

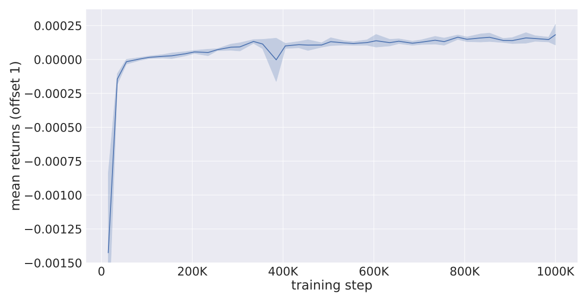
3.4 Multiple Assets

In considering trading independent assets, it is interesting to consider how a single agent may approach parallel tasks. In turn, the efficacy of the DQN approach in trading assets with inter-related relationships is also important. Such scenarios require careful consideration of the action space available to the agent.

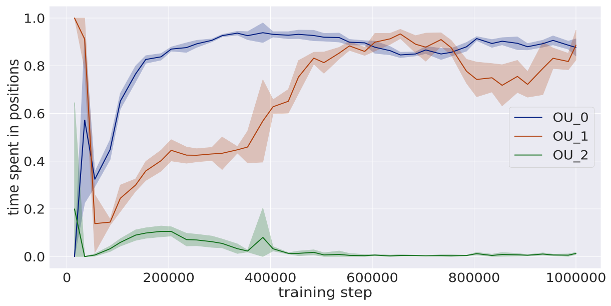
Independent Assets

For independent assets, due to the loss of information when using a scalar reward, the reward was decomposed into a vector of returns attributable to each asset. This included costs incurred when executing transactions for each respective asset. This, along with a slight modification of the DQN loss function allowed training of an agent with multiple sets of actions in parallel[9]. Although not shown, this formulation was found to perform better than using a scalar reward. Figure 6 shows training progression with the decomposed rewards approach. Specifically, the data process consisted of three independent OU process OU_0, OU_1 & OU_2 with variances .04, .02 & .01 respectively. Given that variance determines the number of trading opportunities, figure 6b shows a differential distribution of time spent in positions for each of the assets. Clearly, the agent hardly engages with OU_2 throughout the learning phase, as it has the least amount of trading opportunities, while there is a clear bias towards OU_0. The activity regarding OU_1 is quite variable and doesn't seem to reach a steady state throughout training. It is possible that given more training steps, convergence leading to a steady state of time spent in positions would be reached, reflecting the ratio of trading opportunities.

[9]Action value estimates were made for each asset independently, hence eq. (14) was optimised for assets in parallel, using a single agent.



(a)



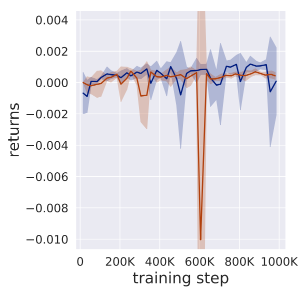
(b)

Figure 6: Results for an agent trained on 3 independent OU processes. (a) Mean Returns for test episodes throughout training. (b) Proportion of time spent in positions for each asset, throughout training. Distribution parameters were the same as the previous OU process except series OU_0, OU_1 and OU_2 had σ values of .04, .02 and .01, respectively.

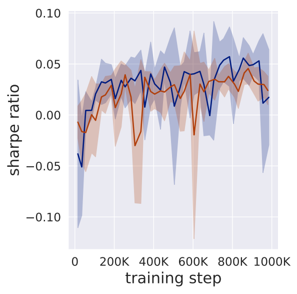
Cointegrated Pairs

To explore trading assets with inter-dependencies, a pair of time series were implemented which exhibited mean reversion around a common mean which itself is a random walk, detailed in B.3. Thus, the ideal strategy is to trade the spread between the assets when it diverges from the mean of 0, provided that the divergence is greater than the transaction cost. In addition to the action space formulation used for independent assets, another approach is to explicitly determine the set of actions by taking the combinatorial product of individual actions for each asset. With n assets and k actions per asset, this results in k^n total actions, compared to $n \times k$ for the independent formulation. In the case of 4 actions and 2 assets, the set is $4^2 = 16$ actions. Despite the theoretical benefit of enabling modelling of co-dependent actions, the scaling relationship between number of assets and total number of action combinations is clearly undesirable. Thus, to restrict the action space to useful actions, this set was filtered to 4 actions which treated the spread as a tradeable asset (Hold, Close, (Buy A, Sell B), (Sell A, Buy B)), with rewards being aggregated into scalar values.

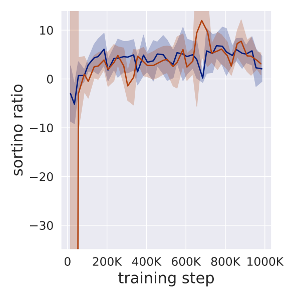
Unexpectedly, figures 7(a-b) show similar training progressions using both action space formulations. This is especially surprising considering that the combined action space is tailored for trading a spread while the agent with the independent action space can not explicitly factor into its value estimate that when it enters one part of a spread trade, that the other side will also be entered. Figures 7(e-h) compare the strategies of the two agents which in effect are similar, though differ in timing. Whereas the combined action space entails that the positions for both assets are entered simultaneously, the independent actions yield much looser timing, with no constraint that the complementary pair trade is entered simultaneously. Although these two methods worked in the setting of two related assets, they may not necessarily scale when the number of assets is increased, greatly increasing the implicit and explicit combinatorial complexity in the independent and combined action approaches, respectively. For combined actions, the set of actions would have to be specified to restrict the total number of actions, thus reducing the span of possible strategies. Yet for independent actions, it is not obvious whether strategies such as trading spreads can be reliably learned as the number of assets and relationships between assets increases. Solutions to this are discussed in section 4.



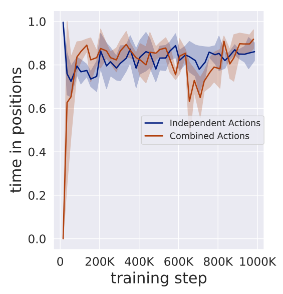
(a)



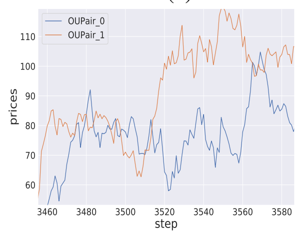
(b)



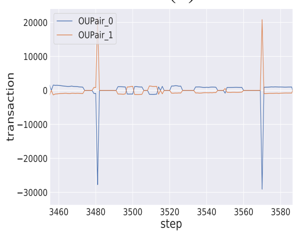
(c)



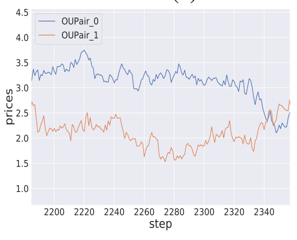
(d)



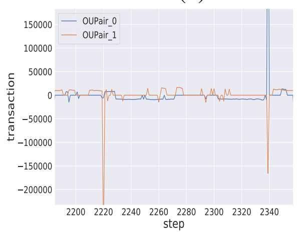
(e)



(f)



(g)



(j)

Figure 7: Comparison between two action spaces (independent and combined) for trading a cointegrated pair. (a-d) Mean returns, Sharpe ratio, Sortino ratio and time spent in positions for test episodes throughout training. (e-f) Prices and transactions for ~120 steps of a test episode taken at the end of training for an agent trained with the combined action space, explicitly trading the spread.(g-h) Prices and transactions for ~160 steps of a test episode taken at the end of training for an agent trained with the independent action space.

3.5 Risk Management

This section details different approaches to risk management. The previous results have been presented by primarily showing mean returns over test episodes. As volatility of returns is critical in characterising a trading strategy, along with mean returns, the following results primarily compare Sharpe ratio, Sortino ratio and the proportion of time spent in positions for each of the approaches to modulating risk. Each experiment was conducted using an OU series with $\theta = .015$, $\sigma = .04$ and an agent trained as before but with 20 n-step returns, to provide a sufficiently long period for the return aggregation approach.

Volatility Adjustment of Rewards

To directly optimise for volatility adjusted returns, the naive approach of Sharpe and Sortino based return aggregation was tested, as specified in equations (11) & (12) respectively. Additional to the standard formulation of the Sortino ratio (henceforth referred to as Sortino A), another Sortino inspired aggregation (Sortino B) was formulated as:

$$R_t^{sortinoB} = \sum_{k=0}^{k=n-1} \gamma^k Diff_k + \gamma^n V(s_{t+n})$$

$$where\ Diff_k = \begin{cases} (r_k^s - r_k^b)^{\frac{1}{\alpha}}, & \text{if } (r^s - r^b) < 0 \\ (r_k^s - r_k^b), & \text{otherwise} \end{cases} \tag{16}$$

where α is a temperature parameter modulating the degree of emphasis on negative returns. Intuitively, this adjustment simply penalises negative rewards more than positive rewards[10]. As well as adjusting α in Sortino B, the square operation in the DR_i term for Sortino A (eq. 12) offers opportunity for modulating the degree of downside-volatility adjustment by generalising the operation from $(\mathbb{E}(r_i^s - r_i^b)^2)^{\frac{1}{2}}$ to fractional central moments, $(\mathbb{E}(r_i^s - r_i^b)^{\alpha})^{\frac{1}{\alpha}}$. The naive aggregations were compared with the standard DSR and DDR formulations as outlined in (Moody and Saffell, 2001).

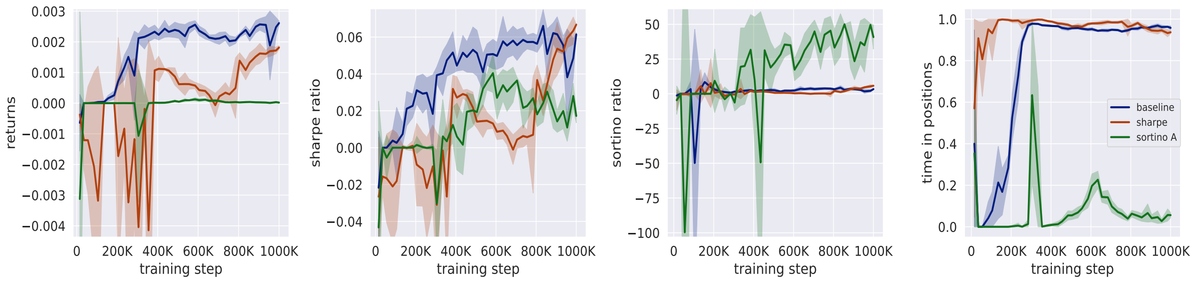
As shown in figure 8a, both Sharpe and Sortino A aggregations result in lower mean returns across

[10]Negative rewards are penalised more when $\alpha > 1$ and $|r^s - r^b| < 1$, this was enforced by clipping raw rewards (log returns) to $-1 < r < 1$

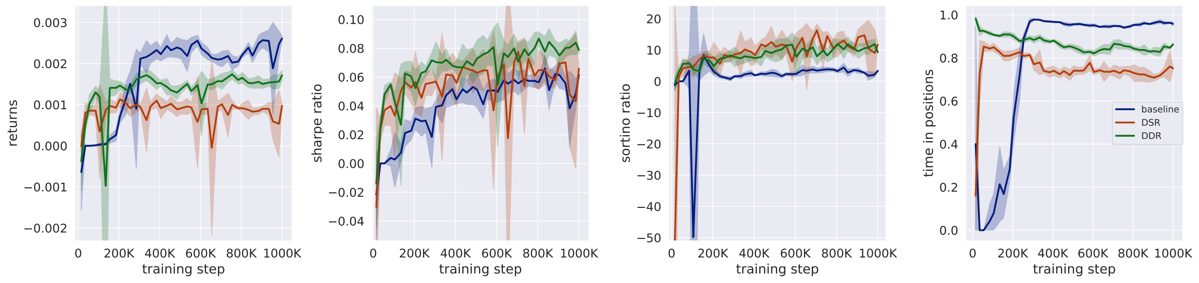
training, with Sortino A being magnitudes smaller. The Sharpe aggregation seems to approach the performance of the baseline in terms of mean returns, Sharpe and Sortino ratio near the end of training, hinting at a potential improvement at least up to the level of the baseline, with comparable time spent in positions. Thus, while admitting a pre-maturely ended experiment, the advantage of the Sharpe aggregation is not apparent. This may be expected from the theoretical consideration of penalising large positive rewards as much as large negative rewards, though it may just be that the Sharpe aggregation results in slower training but converges to better long term performance. Conversely, despite the naive Sortino (A) aggregation not spending much time in positions and having much lower mean return, its Sortino ratio can be seen in figure 8a to be orders of magnitude higher than either the baseline or Sharpe aggregations and also approximately double that of the DDR and DSR methods. Though it may seem like a trivial result that the agent optimised to predict Sortino aggregated rewards has the highest Sortino ratio upon testing, the same was not seen with the Sharpe aggregation. Compared with baseline, the DDR and DSR methods both showed a similar trend of smaller mean returns and time spent in positions but higher Sortino ratios and at least as high Sharpe ratios.

For the naive aggregation methods, the effects of adjusting the degree of downside-risk adjustment via the α parameter, are shown in figure 8c. Clearly, it is possible to use this parameter to tune the degree of risk aversion imposed by the Sortino A aggregation, as evidenced by the differing performance metrics and time spent in positions. Similarly, the Sortino B aggregation was found to be effective in producing downside-risk adjusted performance superior to baseline except only when the α parameter was reduced to 1.05. Similar adjustments to degree of downside risk penalisation were not performed for the DDR method however will be addressed in future work.

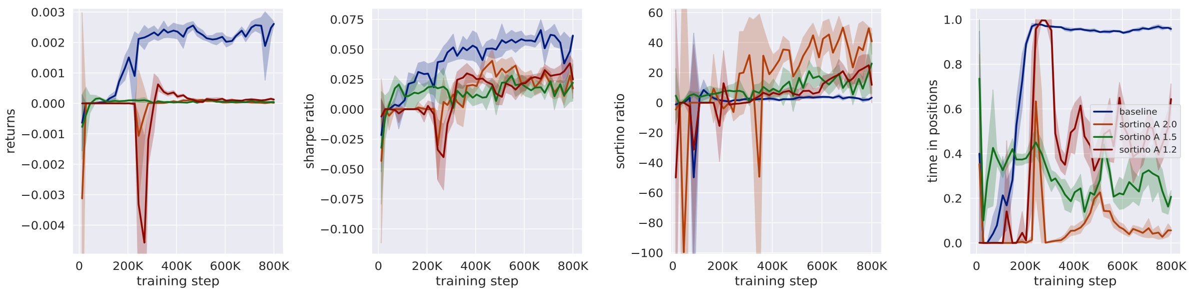
These results show the empirical benefits of such risk adjustment methods and that the naive approach can perform comparably with the DSR and DDR which have better theoretical properties. However, this must be considered in context of the fact that only one experiment per condition was run and that n-step returns were aggregated with n=20. While this was suitable for the OU time series tested, if a time series does not exhibit causal behaviour at that range, this would introduce variance in the naive aggregation, as discussed in 2.1. Thus, this would require the number of n-step returns to be tuned for the specific task, with too high and too low number of steps both entailing higher variance. Since DSR and DDR don't share this source variance, they may exhibit better generalisation. In summary, these two methods of incentivising risk sensitive performance result in classes of agents which learn trading strategies with varying sensitivities to risk. Additionally, explicit factorisation of downside risk can result in an agent with a clearly risk sensitive trading strategy whereby positions are entered sparsely where the expectation of positive return has much higher precision than a risk neutral strategy.



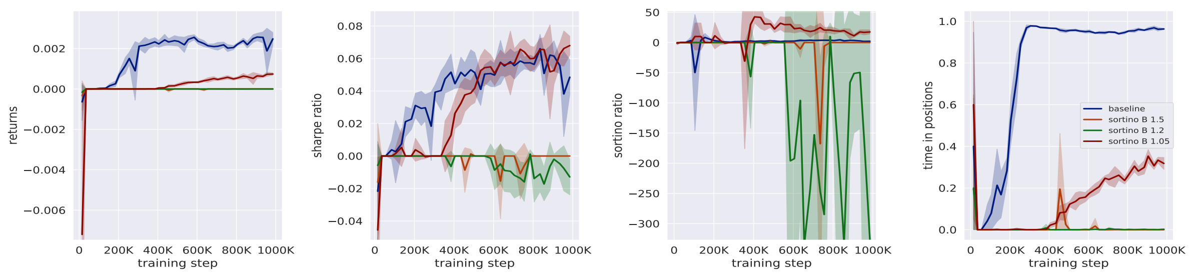
(a) Comparison between baseline (risk neutral), Sharpe and Sortino A ($\alpha = 2$) based volatility adjustment of rewards



(b) Comparison between baseline, DSR and DDR



(c) Comparison between baseline and Sortino A with varying $\alpha \in (2, 1.5, 1.2)$



(d) Comparison between baseline and Sortino B with varying $\alpha \in (1.5, 1.2, 1.05)$

Figure 8

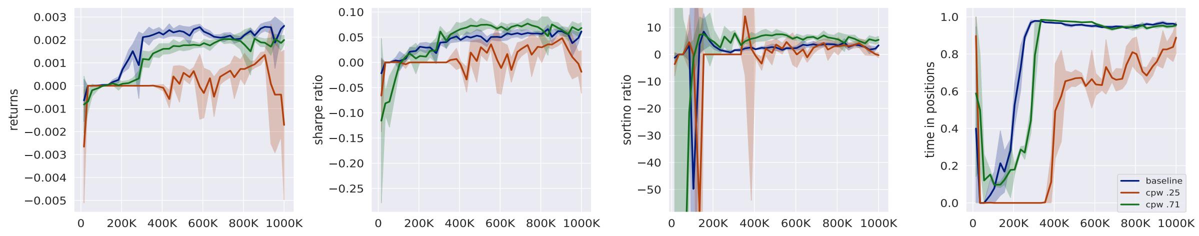
Risk Distortion in Distribution RL

For a dynamic and general approach to specifying the risk sensitivity of an agent, the formulation of IQN offers a principled method which fits naturally in its formulation. Importantly, this method addresses the limitations of both the naive aggregation and the differential update as it does not depend on the aggregation period of n-step returns and can be dynamically altered during test time (Dabney et al., 2018). The method leverages a distortion risk function which alters the cumulative sampling probabilities of a random variable (Sereda et al., 2010). As the IQN agent predicts a distribution of rewards for state-action values which it then averages to return a scalar, the weighting of the distribution can be altered to reflect a risk averse or risk seeking bias. As in Dabney et al. (2018), the distortion measures explored are cumulative probability weighting (cpw) (Tversky and Kahneman, 1992), wang (Wang, 2000), conditional value at risk (cvar) (Chow and Ghavamzadeh, 2014) and the power (pow) distortion proposed by the authors[11]. All these risk distortion measures require specification of a parameter η which determines the shape of the resulting sampling distribution, with globally concave and convex shapes associated with risk-averse and risk-seeking distortions respectively.

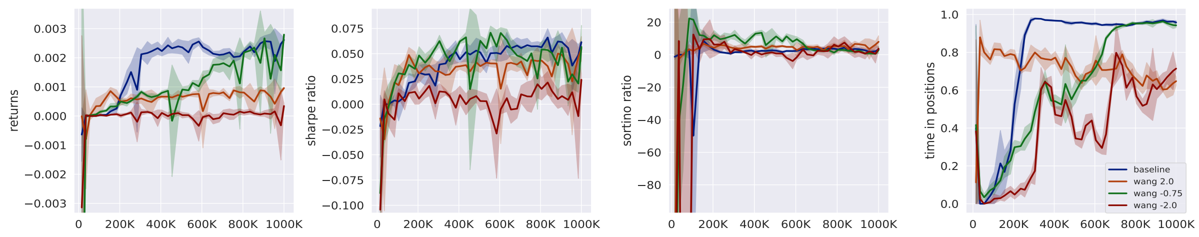
Figure 9 shows experimental results using these risk distortion measures with varying η, compared to the same baseline as the previous experiments. In summary, as with the previous class of methods, the risk distortion methods generally result in lower mean returns, higher Sortino ratios and less time spent in positions. The Sharpe ratios are comparable, though not as informative as the Sortino ratios. To note, the agents in these experiments were trained and tested using the same distortion measure. Differing combinations of risk measures and η in the training and testing settings may yield further useful behaviour. For example, an agent may act in a risk neutral manner when interacting with the environment during training, gaining varied experiences and then subject to a stringent distortion measure when being tested. This may overcome the biased data distribution incurred using a risk averse distortion which incentivises an agent to interact conservatively with the environment. Although this method may be

[11]Equations are omitted here, please see (Dabney et al., 2018).

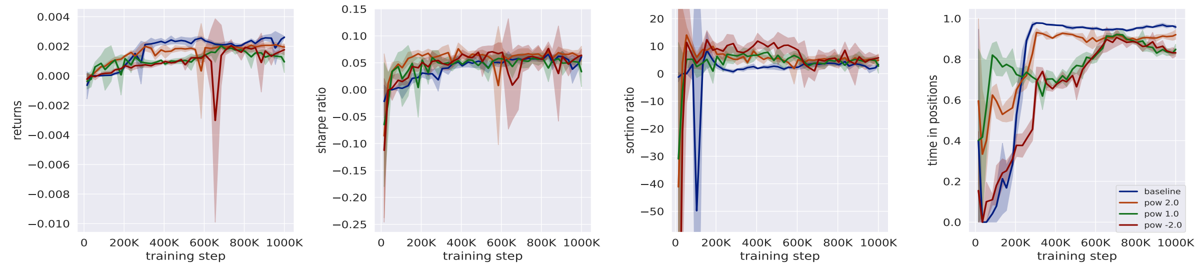
combined with the aggregation methods, the aforementioned benefits of this method warrant an attempt to optimise this method with priority.



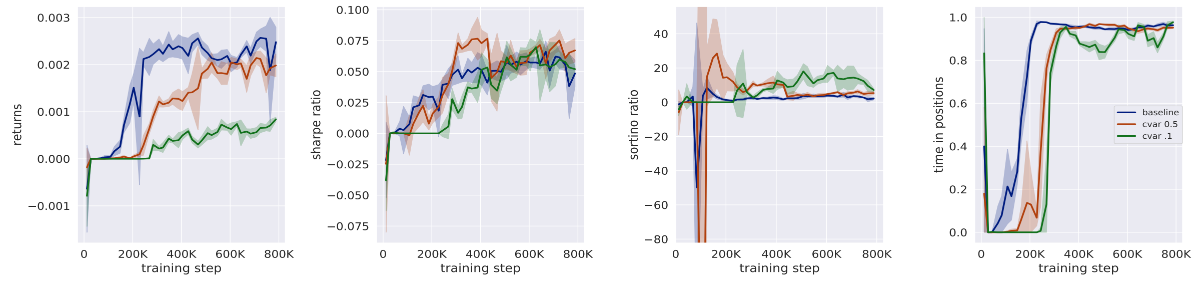
(a) Comparison between baseline and cpw risk distortion with varying η. Distortion is neither globally convex nor concave but locally concave for small η and locally concave for large η.



(b) Comparison between baseline and wang risk distortion with varying η. Distortion is risk averse for $\eta < 0$ and risk seeking for $\eta > 0$.



(c) Comparison between baseline and pow risk distortion with varying η. Distortion is risk averse for $\eta < 0$ and risk seeking for $\eta > 0$.



(d) Comparison between baseline and cvar risk distortion with varying η. Distortion is increasingly risk averse as η is decreased.

Figure 9

Proximal Portfolio Constraint

As indicated in 2.1.1, to compare PPC towards the same objective as the previously explored risk adjustment methods, the α parameter and the desired portfolio of full cash holdings is held constant throughout an experiment. Figure 10 shows results obtained by varying the α parameter between experiments. As one would expect, there is an inverse relationship between α and mean returns as well as time spent in positions, with higher α discouraging the agent from entering into positions. However, it appears that $\alpha = .025$ was within a favourable range such that its Sortino ratio was higher than α values higher and lower, as well as the baseline. Thus, in its most basic formulation, the PPC method successfully provides a method for risk adjustment in the trading setting.

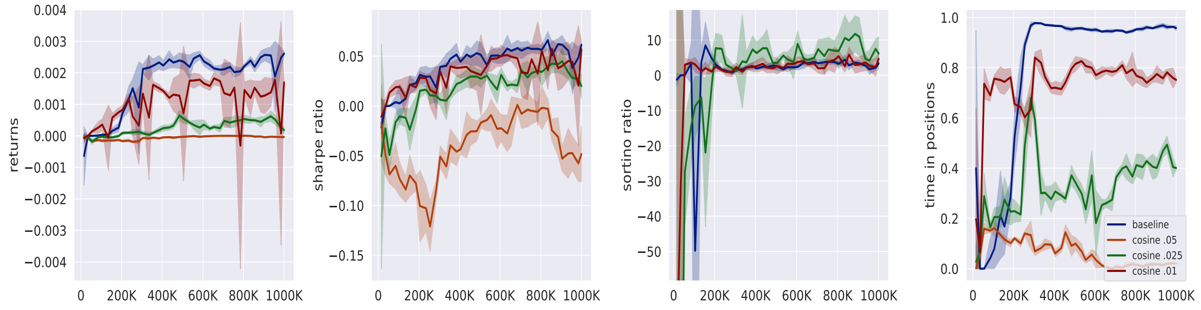


Figure 10: Comparison between baseline and cosine portfolio penalties with differing temperatures. Sortino Ratio for penalty .05 was large and negative, omitted for visual clarity

4 Related and Future Applications

Many applications of RL in trading have been presented in literature, with some seminal work having been done long before the current advancements in deep RL (Moody and Saffell, 1999). Utilising the DSR measure, an early application of neural networks in using RL for trading was (Gold, 2003) which utilised recurrent NNs in trading currency exchange markets. To extend upon the recurrent idea in encoding the state space, further work may endow an agent with auto-regression with respect to past rewards and actions, congruent with auto-regressive time series models such as ARIMA and GARCH, commonly used in analysis of financial timeseries (Gençay et al., 2001). Although the present work has approached the general problem of active trading, there have been applications of RL in specific areas of financial markets such as optimal trade execution and the inventory constrained setting of market making, requiring interaction with a limit order book (Nevmyvaka, Feng, and Kearns, 2006; Spooner et al., 2018). The proposed PPC method in its dynamic formulation may prove useful in such settings.

In real world applications, the feature extraction capabilities of NNs may be leveraged to extract trading strategies from a suitable representation of data provided that an extractable edge *exists* given the representation. Although this is congruent with recent deep learning trends whereby NNs are applied to problems in an end-to-end manner, it is expected that such methods are best when engineered by domain experts. For example, an RL agent may be trained on a sparse subset of data where a previously used trading signal reaches some threshold, thereby semantically segmenting the dataset and applying the RL method to specific market conditions. As another approach, agents may be pre-trained on idealised data processes parameterised by market data and then fine-tuned on real market data. Many RL agents trained in the same manner may also be deployed as independent trading strategies in parallel, an ensembling strategy which fairs well in machine learning when the optimisation method does not tend to find a global minimum. Additionally, when trained under differing conditions, resulting equity curves from individual agents may themselves be treated as assets and given weightings via classic portfolio optimisation methods (Black and Litterman, 1992). For example, a mean reverting agent with recent success may be allocated higher capital, provided a validated expectation of persistence of the market condition associated with the agent.

Independent to the specific setting of trading are improvements in the core RL algorithms used. A natural progression from the value based methods explored in this work are actor-critic (AC) methods which employ policy gradient based agents (actors) in defining explicit policies which are improved with the help of value based components (critics). In addition to providing general performance improvements in many benchmarks (Haarnoja et al., 2018), AC methods allow direct and scaleable modelling of interactions when dealing with multiple outputs, without requiring an explicit factorisation [12] (Berner et al., 2019). Furthermore, explicit policies allow use of continuous outputs, allowing agents to specify fine-grained actions via portfolio weightings or direct transaction sizes[13]. Another class of methods which are independent to choice of policy are on-line methods whereby the agent exploring the environment learns from its consecutive interactions. This is in contrast with the DQN approach where a *behaviour* agent is used to explore the environment and accumulate experiences from which it learns by sampling randomly. Thus, online methods do not rely on a replay buffer and may exhibit greater adaptability in dynamic market conditions if subject to continual learning. Lastly, a progression of the previously mentioned theme of auto-regression would be planning methods which model the transition functions of the MDP such as value prediction networks and the Mu-Zero algorithm (Oh, Singh, and Lee, 2017; Schrittwieser et al., 2020).

Aside from the base RL algorithm, the NN architecture used as the learned model also offers opportunity for further optimisation. The CNN based architecture used in the experiments was not significantly tuned, in interest of keeping components as simple as possible whilst being able to complete the tasks presented. Many extensions to the CNN have been proposed, designed specifically for time series forecasting (Zhao et al., 2017; Cui, Chen, and Chen, 2016; Borovykh, Bohte, and Oosterlee, 2017). Also, the transformer architecture based on self-attention proposed in (Vaswani et al., 2017) for sequential problems has promising applications to time series data (Li et al., 2019; Zhou et al., 2020).

5 Conclusion

The present work presents two main contributions. Firstly, the development of a set of benchmarks tests which can be used to validate, compare and develop trading strategies based on themes such as mean reversion and trend persistence, before application to real financial data. The significance of this contribution lies in the opportunity for diagnostic analysis into what may otherwise be a black-box when using

[12]For which the number of actions k and assets n were shown to scale as $O(k^n)$

[13]I.e As a proportion of available margin or overall equity

methods such as RL, especially in conjunction with deep NNs. Secondly, a set of approaches extending upon the base DQN algorithm was validated with respect to specification of state inputs, action space and rewards to achieve positive risk-adjusted performance in the benchmark tasks. Several approaches to risk management were tested, some of which provided opportunity for tuning the degree of risk sensitivity of an agent. Of these, the risk distortion method of adjusting the sampling distribution of the distributional RL approach enabled by IQN provides the most general and dynamic method for enabling such agents to behave in a risk sensitive manner. Additionally, a novel portfolio constraint approach (PPC) was proposed and tested, allowing specification of degree of risk aversion by differentially constraining an agent towards a target portfolio. In its generalised form, it yields a method for executing dynamic portfolio constraints which implores further testing. In summary, the set of experimental results in this study motivate the application and further research of RL methods in trading.

References

Aiba, Yukihiro et al. (2002). “Triangular arbitrage as an interaction among foreign exchange rates”. In: *Physica A: Statistical Mechanics and its Applications* 310.3-4, pp. 467–479.

Alexander, Carol and Anca Dimitriu (2005). “Indexing and statistical arbitrage”. In: *The Journal of Portfolio Management* 31.2, pp. 50–63.

Avellaneda, Marco and Jeong-Hyun Lee (2010). “Statistical arbitrage in the US equities market”. In: *Quantitative Finance* 10.7, pp. 761–782.

Baltas, Nick and Robert Kosowski (2013). “Momentum strategies in futures markets and trend-following funds”. In: *Paris December 2012 Finance Meeting EUROFIDAI-AFFI Paper*.

Berner, Christopher et al. (2019). “Dota 2 with large scale deep reinforcement learning”. In: *arXiv preprint arXiv:1912.06680*.

Bistarelli, Stefano et al. (2019). “Model-based arbitrage in multi-exchange models for Bitcoin price dynamics”. In: *Digital Finance* 1.1-4, pp. 23–46.

Black, Fischer and Robert Litterman (1992). “Global portfolio optimization”. In: *Financial analysts journal* 48.5, pp. 28–43.

Borovykh, Anastasia, Sander Bohte, and Cornelis W Oosterlee (2017). “Conditional time series forecasting with convolutional neural networks”. In: *arXiv preprint arXiv:1703.04691*.

Chow, Yinlam and Mohammad Ghavamzadeh (2014). “Algorithms for CVaR optimization in MDPs”. In: *Advances in neural information processing systems* 27, pp. 3509–3517.

Cross, Rod and Victor Kozyakin (2015). “Fact and fictions in FX arbitrage processes”. In: *Journal of Physics: Conference Series*. Vol. 585, pp. 1–9.

Cui, Zhicheng, Wenlin Chen, and Yixin Chen (2016). “Multi-scale convolutional neural networks for time series classification”. In: *arXiv preprint arXiv:1603.06995*.

Cummings, James Richard and Alex Frino (2011). “Index arbitrage and the pricing relationship between Australian stock index futures and their underlying shares”. In: *Accounting & Finance* 51.3, pp. 661–683.

Dabney, Will et al. (2018). “Implicit Quantile Networks for Distributional Reinforcement Learning”. In: *International Conference on Machine Learning*, pp. 1096–1105.

De Prado, Marcos Lopez (2018). *Advances in financial machine learning*. John Wiley & Sons.

Emerson, Sophie et al. (2019). “Trends and applications of machine learning in quantitative finance”. In: *8th International Conference on Economics and Finance Research (ICEFR 2019)*.

Feng, Yiyong, Daniel P Palomar, et al. (2016). *A signal processing perspective on financial engineering*. Vol. 9. Now Publishers.

Gençay, Ramazan et al. (2001). *An introduction to high-frequency finance*. Elsevier.

Gold, Carl (2003). “FX trading via recurrent reinforcement learning”. In: *2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings.* IEEE, pp. 363–370.

Haarnoja, Tuomas et al. (2018). “Soft Actor-Critic Algorithms and Applications”. In: *CoRR* abs/1812.05905. arXiv: `1812.05905`. URL: `http://arxiv.org/abs/1812.05905`.

Hendrycks, Dan and Kevin Gimpel (2016). “Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units”. In: *CoRR* abs/1606.08415. arXiv: `1606.08415`. URL: `http://arxiv.org/abs/1606.08415`.

Hessel, Matteo et al. (2018). “Rainbow: Combining improvements in deep reinforcement learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Jensen, Michael C, Fischer Black, and Myron S Scholes (1972). “The capital asset pricing model: Some empirical tests”. In:

Kaelbling, Leslie Pack, Michael L Littman, and Anthony R Cassandra (1998). “Planning and acting in partially observable stochastic domains”. In: *Artificial intelligence* 101.1-2, pp. 99–134.

Krauss, Christopher (2017). “Statistical arbitrage pairs trading strategies: Review and outlook”. In: *Journal of Economic Surveys* 31.2, pp. 513–545.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *nature* 521.7553, pp. 436–444.

Lempérière, Yves et al. (2014). “Two centuries of trend following”. In: *arXiv preprint arXiv:1404.3274*.

Li, Shiyang et al. (2019). “Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting”. In: *Advances in Neural Information Processing Systems*, pp. 5243–5253.

Lo, Andrew W (2002). “The statistics of Sharpe ratios”. In: *Financial analysts journal* 58.4, pp. 36–52.

Lo, Andrew W, Harry Mamaysky, and Jiang Wang (2000). “Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation”. In: *The journal of finance* 55.4, pp. 1705–1765.

Mnih, Volodymyr et al. (2015). “Human-level control through deep reinforcement learning”. In: *nature* 518.7540, pp. 529–533.

Moody, John and Matthew Saffell (2001). “Learning to trade via direct reinforcement”. In: *IEEE transactions on neural Networks* 12.4, pp. 875–889.

Moody, John et al. (1998). “Performance functions and reinforcement learning for trading systems and portfolios”. In: *Journal of Forecasting* 17.5-6, pp. 441–470.

Moody, John E and Matthew Saffell (1999). “Reinforcement learning for trading”. In: *Advances in Neural Information Processing Systems*, pp. 917–923.

Nevmyvaka, Yuriy, Yi Feng, and Michael Kearns (2006). “Reinforcement learning for optimized trade execution”. In: *Proceedings of the 23rd international conference on Machine learning*, pp. 673–680.

Oh, Junhyuk, Satinder Singh, and Honglak Lee (2017). “Value prediction network”. In: *arXiv preprint arXiv:1707.03497*.

Paszke, Adam et al. (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Platen, Eckhard and Renata Rendek (2008). "Empirical evidence on Student-t log-returns of diversified world stock indices". In: *Journal of statistical theory and practice* 2.2, pp. 233–251.

Schrittwieser, Julian et al. (2020). "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839, pp. 604–609.

Sereda, Ekaterina N et al. (2010). "Distortion risk measures in portfolio optimization". In: *Handbook of portfolio construction*. Springer, pp. 649–673.

Sharpe, William F (1994). "The sharpe ratio". In: *Journal of portfolio management* 21.1, pp. 49–58.

Snow, Derek (2020). "Machine Learning in Asset ManagementPart 1: Portfolio ConstructionTrading Strategies". In: *The Journal of Financial Data Science* 2.1, pp. 10–23.

Spooner, Thomas et al. (2018). "Market making via reinforcement learning". In: *arXiv preprint arXiv:1804.04216*.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Tsantekidis, Avraam et al. (2017). "Forecasting stock prices from the limit order book using convolutional neural networks". In: *2017 IEEE 19th Conference on Business Informatics (CBI)*. Vol. 1. IEEE, pp. 7–12.

Tversky, Amos and Daniel Kahneman (1992). "Advances in prospect theory: Cumulative representation of uncertainty". In: *Journal of Risk and uncertainty* 5.4, pp. 297–323.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008.

Wang, Shaun S (2000). "A class of distortion operators for pricing financial and insurance risks". In: *Journal of risk and insurance*, pp. 15–36.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8.3-4, pp. 279–292.

Zhao, Bendong et al. (2017). "Convolutional neural networks for time series classification". In: *Journal of Systems Engineering and Electronics* 28.1, pp. 162–169.

Zhong, Xiao and David Enke (2019). "Predicting the daily return direction of the stock market using hybrid machine learning algorithms". In: *Financial Innovation* 5.1, p. 4.

Zhou, Haoyi et al. (2020). "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting". In: *arXiv preprint arXiv:2012.07436*.

A Hyperparameters

The following table outlines a common set of parameters shared by most experiments discussed, unless otherwise indicated in the experiments section. Replay buffer size was set to 300K to accommodate computational constraints when running experiments in parallel. N and N' refer to the number of samples to get distribution quantiles τ and τ' respectively, in IQN (Dabney et al., 2018).

Hyperparameters	Value
Discount γ	.99
Nsteps	5
N	32
N'	8
Replay buffer size	300K
Minimum replay buffer size	20K
Batch Size	32
Num layers in CNN	2
Number of channels in 1D Convolutions	32
Size of kernels in 1D Convolutions	5
Size of fully connected layers	256
Activation function used in CNN	GELU (Hendrycks and Gimpel, 2016)
Optimizer for gradient descent	Adam
Learning rate	.001
Preprocessor concatenation window length K	64
Maintenance Margin (proportion of total equity in cash)	.2
Base transaction units (as proportion of available margin)	.05
Transaction cost (relative to transaction value)	.02

B Data Generating Processes

The following algorithms formalise some of the data generating processes tested, which aren't applications of common processes with referable literature.

B.1 Trend Persistence

Algorithm 1 Trend Persisting Series

```
Init $y, dy, trendProb, noise$
Init $minTrendLen, maxTrendLen, mindy, maxdy$
$trending \leftarrow$ **false**
$trendDirection \leftarrow 1$
**loop**
   **if** $trending$ **then**
      $y \leftarrow y + y \cdot dy \cdot trendDirection$
      $trendDirection \leftarrow -1$
      **if** $trendDirection == 0$ **then**
         $trending \leftarrow$ **false**
      **end if**
   **else**
      $rand \leftarrow sample \sim Uniform[0, 1]$
      **if** $rand < trendProb$ **then**
         $trending \leftarrow$ **true**
         $currentDirection \leftarrow sample \in \{-1, 1\}$
         $currentTrendLen \leftarrow sample \sim Uniform[minTrendLen, maxTrendLen]$
         $dy \leftarrow sample \sim Uniform[mindy, maxdy]$
      **end if**
   **end if**
   $y \leftarrow y + y \cdot (sample \sim N(0, noise))$
**end loop**
```

B.2 TrendingOU

Algorithm 2 Geometric OU centered around a trending process.

```
Init $y, trend, ou, dy, trendProb$
Init $ouNoise, ouRevertingComponent$
Init $\theta, \sigma$                                              // as defined in (15)
Init $minTrendLen, maxTrendLen, mindy, maxdy$
$trending \leftarrow$ **false**
$trendDirection \leftarrow 1$
loop
    $ouNoise \leftarrow trend \cdot noise \sim N(0, \sigma)$
    $ouRevertingComponent \leftarrow \theta \cdot -ou$                // Centered at 0.
    $ou \leftarrow ou + ouNoise + ouRevertingComponent$
    if $trending$ then
        $trend \leftarrow trend + trend \cdot dy \cdot trendDirection$
        $trendDirection \leftarrow -1$
        if $trendDirection == 0$ then
            $trending \leftarrow$ false
        end if
    else
        $rand \leftarrow sample \sim Uniform[0, 1]$
        if $rand < trendProb$ then
            $trending \leftarrow$ true
            $currentDirection \leftarrow sample \in \{-1, 1\}$
            $currentTrendLen \leftarrow sample \sim Uniform[minTrendLen, maxTrendLen]$
            $dy \leftarrow sample \sim Uniform[mindy, maxdy]$
        end if
    end if
    $y \leftarrow y + ou + trend$             // OU and Trending components maintained separately
end loop
```

B.3 Cointegrated Pairs

Algorithm 3 Geometric OU pair centered around a random walk process.

```
Init $ouA, ouB, mean$
Init $\theta, \sigma, noise$
loop
    $mean \leftarrow mean + mean \cdot sample \sim N(0, noise)$
    $ouA \leftarrow ouA + theta \cdot (mean - ouA) + mean \cdot sample \sim N(0, \sigma)$
    $ouB \leftarrow ouB + theta \cdot (mean - ouB) + mean \cdot sample \sim N(0, \sigma)$
end loop
```