The Journal of Finance and Data Science 9 (2023) 100101

Contents lists available at ScienceDirect

The Journal of Finance and Data Science

journal homepage: www.keaipublishing.com/en/journals/jfds

Research Article

Hedging using reinforcement learning: Contextual *k*-armed bandit versus *Q*-learning

Loris Cannelli [a,*], Giuseppe Nuti [b], Marzio Sala [c], Oleg Szehr [a]

[a] Dalle Molle Institute for Artificial Intelligence (IDSIA) - SUPSI/USI, Lugano, Switzerland
[b] UBS Investment Bank, New York, USA
[c] UBS Investment Bank, Zurich, Switzerland

ARTICLE INFO

Keywords:
Hedging
Reinforcement Learning
Q-Learning
Multi-Armed Bandits

ABSTRACT

The construction of replication strategies for contingent claims in the presence of risk and market friction is a key problem of financial engineering. In real markets, continuous replication, such as in the model of Black, Scholes and Merton (BSM), is not only unrealistic but is also undesirable due to high transaction costs. A variety of methods have been proposed to balance between effective replication and losses in the incomplete market setting. With the rise of Artificial Intelligence (AI), AI-based hedgers have attracted considerable interest, where particular attention is given to Recurrent Neural Network systems and variations of the *Q*-learning algorithm. From a practical point of view, sufficient samples for training such an AI can only be obtained from a simulator of the market environment. Yet if an agent is trained solely on simulated data, the run-time performance will primarily reflect the accuracy of the simulation, which leads to the classical problem of model choice and calibration. In this article, the hedging problem is viewed as an instance of a risk-averse contextual *k*-armed bandit problem, which is motivated by the simplicity and sample-efficiency of the architecture, which allows for realistic online model updates from real-world data. We find that the *k*-armed bandit model naturally fits to the Profit and Loss formulation of hedging, providing for a more accurate and sample efficient approach than *Q*-learning and reducing to the Black-Scholes model in the absence of transaction costs and risks.

1. Introduction and motivation

The construction of dynamic hedging strategies lies at the core of the valuation and risk-management of derivative contracts in investment firms. The modern hedging theory came to light with two seminal articles of Black and Scholes (1973) and Merton (1973) on the valuation of European options in an idealized economic model. This model, commonly known as the BSM economy, is characterized by continuous-time trading, a complete and frictionless market and stock price processes of geometric Brownian motion type. Black-Scholes and Merton construct a self-financing trading strategy in a riskless security (cash) and a risky stock, which perfectly replicates the option's payoff at maturity, thus *hedging* the option's risk. The absence of arbitrage implies that the price equals the value of the replicating portfolio over the whole lifetime. In particular, the price of the option at inception is defined as the cost of setting up the replicating portfolio.

* Corresponding author. Via la Santa 1, Viganello, Lugano, 6962, Switzerland.
E-mail addresses: loris.cannelli@idsia.ch (L. Cannelli), giuseppe.nuti@ubs.com (G. Nuti), marzio.sala@ubs.com (M. Sala), oleg.szehr@idsia.ch (O. Szehr).

Peer review under responsibility of KeAi Communications Co., Ltd.

https://doi.org/10.1016/j.jfds.2023.100101
Received 5 December 2022; Received in revised form 15 May 2023; Accepted 18 June 2023
Available online 22 June 2023

In reality, continuous time trading is, of course, impossible. It cannot even serve as a reasonable approximation, due to the high resulting transaction costs. Instead, a *replicating portfolio* is adjusted at discrete times, optimizing between replication error and trading costs. The acceptable deviation from the exact hedge depends on the risk tolerance of the investor. As a consequence, the best hedging strategy constitutes a trade-off between replication error (risk) and trading costs. To quantify this trade-off it is common to introduce a utility function, which reflects the investor's preference. The investor's decisions are then guided by the maximization of expected utility over the investment horizon. Utility-based hedging entails a significant complication of the BSM approach: instead of minimizing the immediate replication error, the investor is faced with a multi-period planning problem, whose goal is the optimal trade-off between risk and accumulated transaction costs. Furthermore, in the context of incomplete markets, it is far less immediate what the price of a derivative contract should be. This can possibly give rise to the bid-ask spread for the derivative contract, which tends to be extraordinarily wide in the absence of a portfolio based approach. In fact various concepts of *price* occur in the literature, two common examples being the "indifference" and "minimum-risk" prices. Indifference purchase/sell prices are the amount of money that makes an investor indifferent, in terms of the expected utility, between trading in a market with and without purchasing/selling the derivative. The minimum-risk price is the amount of initial capital that maximizes an investor's utility given an optimal sequence of hedging decisions. Thus, pricing typically requires the solution of an exogenous optimization problem on top of an existing hedging strategy, e.g. to minimize the global risk. Utility-based hedging and indifference pricing have been introduced in pioneering work by Hodges and Neuenberger (Hodges and Neuberger, 1989). Minimum-risk prices were first studied in (Schweizer, 1995). Hodges and Neuenberger also interpreted hedging in the language of dynamic programming and solved the respective Bellman equations, anticipating some of the modern developments.

With the rise of AI, an entire area of research emerged around learning solutions of dynamic programming problems, which gained wide public audience under the name of Reinforcement Learning (RL). Architectures that combine RL with deep Neural Networks (NNs) have proven super-human capabilities in various domains. The classical examples include video games of increasing complexity such as Atari (Mnih et al., 2013) and Starcraft (Vinyals et al., 2019), and strategic (*P*-) games like Hex (Anthony et al., 2017), Go and Chess (Silver et al., 2016, 2017). Several publications have also reported on promising hedging results by RL agents, see e.g (Hodges and Neuberger, 1989; Kolm and Ritter, 2019; Qlbs, 2020; Cao et al., 2021). In terms of the choice of algorithm (Kolm and Ritter, 2019; Qlbs, 2020; Cao et al., 2021), focus on variations of the *Q*-learning algorithm (Watkins and Dayan, 1992) for the approximation of optimal policies in the hedging problem. The article (Qlbs, 2020) proposes *Q*-learning as a tool for training hedging agents with a quadratic utility functional, but with no transaction costs (Cao et al., 2021). applies double *Q*-learning (Van Hasselt et al., 2016) to take account of stochastic volatility in the hedging problem. The articles (Buehler et al., 2019, 2019) take a slightly different approach to hedging, interpreting multi-period utility optimization as a supervised learning problem. In this setting a recurrent NN is trained via simultaneous optimization over an entire sequence of predictable hedging decisions. The recent contributions (Murray et al., 2022; Mikkilä and Kanniainen, 2023; Cao et al., 2023) in this area appeared during the review of this article. Deep *Q*-Learning (DQN) (Osband et al., 2016) is a variation of *Q*-learning, which combines *Q*-learning with deep NNs to represent quality of state-action values. The article (Kolm and Ritter, 2019) applies DQN to study a discrete-time hedging problem in the presence of transaction costs and served as the motivation for the investigation at hand. As compared to (Hodges and Neuberger, 1989; Qlbs, 2020; Cao et al., 2021; Buehler et al., 2019, 2019) this article focuses on the hedging (rather than valuation) of option contracts assuming that price information is given in an exogenous form (e.g. through a dedicated pricing engine). This assumption leads to what is called the *accounting Profit and Loss* (P&L) formulation of the hedging problem, see (Cao et al., 2021). In this formulation the investor's exercise is to balance between transaction costs and expected deviation from the price process. In contrast, in the *Cash Flow formulation* the investor is only aware of the contractual payoffs of the derivative, which leads to a full-fledged RL problem.

The mentioned publications (Hodges and Neuberger, 1989; Qlbs, 2020; Cao et al., 2021; Buehler et al., 2019, 2019; Kolm and Ritter, 2019) assume an accurate market simulator to train their AI agents. However, setting up such a simulator leads to the classical financial engineering problem of model choice and calibration. This might be seen as a bottleneck for applying those hedging systems in practice as the run-time performance of the AI will often merely reflect the accuracy of the simulator. In this sense it can also hardly be claimed that the trained agents operate in a fully model-free setup. This would apply if enough market data was available for training, which is rarely the case in realistic derivative markets. In reality, run-time performance will be driven by the accuracy of the market simulator - making an AI-based approach awfully similar to standard Monte Carlo methods in use for decades - unless the agent can quickly adapt to unseen scenarios. At this point sample efficiency (see Section 2.6 for a definition) is key, as the comparatively small amount of real-world data that the agent encounters in operation will otherwise not influence the training in a statistically significant way. Similarly, an agent trained on simulated data will usually not acquire the knowledge to appropriately capture market impact in a practical setting. The latter would require training on samples with market impact, which will often be hard to provide in sufficient amount. In the case of a simulated market impact, such samples will simply present the modeled cost of the market impact. The main use-case for current AI systems can thus be seen in hedging of complex derivatives or large derivative portfolios that cannot be efficiently handled using conventional pricing engines as in (Buehler et al., 2019, 2019).

The article at hand argues that - in a practical setting - a bandit-type RL algorithm can be used to address hedging in the P&L formulation. Our theoretical considerations are accompanied with extensive computer experiments comparing the hedging performance of our NN bandit algorithm versus DQN. The *k*-armed bandit is the prototypical instance of RL and has been studied extensively within diverse areas such as advertising selection, clinical trials, finance, etc., see (Sutton and Barto, 2018) for examples. As compared to Supervised Learning, the bandit algorithm adapts to new market data online, i.e. in the course of its execution, by collecting information about realized utility/reward signals. As compared to full-fledged RL, in particular, to variants of *Q*-learning, the bandit is characterized by the assumption that rewards at different time steps are independent and identically distributed. As a consequence each action only determines the immediate reward but has no influence on the future. In terms of hedging, this implies that the agent optimizes for an

immediate utility signal and the agent's choices do not dynamically impact the market. As compared to DQN, bandit algorithms are simpler, better studied, possess stronger performance guarantees, are easier to train, and results can be interpreted more immediately. Most importantly, significantly less samples are required during the training process with well-known optimal regret estimates (Auer et al., 2002). In summary the perspective taken in the article at hand is as follows.

(i) Standard microscopic market models employed to train AI agents do not model market impact. In practice, strong market impact usually signifies a singular situation with little available real-world data.
(ii) Any AI trained on simulated data acquires a behavior typical of the simulation and it will perform well mostly in situations that are similar to the simulation. In particular, it will not be able to make informed decisions in an unusual market state.
(iii) If the number of available training samples is limited, the restricted planning capability of the *k*-armed bandit as compared to more sophisticated RL systems is more than out-weighted by its higher sample efficiency.

The article is structured as follows. Section 2 describes the methodological background for this article, introducing utility-based hedging in Section 2.1 and relevant background from RL in Sections 2.3 and 2.4. Section 3 provides technical details of the algorithms chosen for our experiments. Specifically, Section 3.1 describes the NN-based contextual bandit and Section 3.2 the deep *Q*-learning algorithm. Our experimental findings are presented in Section 4. Section 5 concludes the presentation.

2. Background and methodology

2.1. Hedging and pricing

We first describe the full dynamic-programming approach to hedging in incomplete markets in the Cash Flow formulation and specialize to the P&L formulation later. We assume a random financial market that can be described in terms of Markov processes and where trading occurs at discrete times $t = 0, 1, 2, \ldots, T$. A stochastic process S_t describes the price of a risky asset (stock) and a process B_t describes a riskless security (cash). Suppose a small[1] investor (agent) with bank account holdings B_t enters into a derivative contract with underlying S_t at time $t = 0$. The investor 'sells' the contract, i.e. an immediate profit is credited to the bank account as an initial capital $B_0 \geq 0$ but the contract also creates a random liability of $Z_T \leq 0$ at maturity $T > 0$. In order to off-set the risk from this liability the investor maintains a hedging portfolio. Assume this hedging portfolio is composed of n_t shares of stock S_t, and B_t units of cash on the bank account, and let Π_t denote its value at time t:

$$\Pi_t = n_t S_t + B_t.$$

For shorter notation we assume that there is no compounding on the cash holdings, since this can be incorporated in straight-forward way into our framework. The agent receives new market information at a series of times $t = 1, 2, \ldots. < T$, re-balancing the hedging portfolio immediately once the new information becomes available. In other words at each time step t the agent chooses the number n_{t+1} of shares for the forthcoming period. While the agent adjusts the number of shares, any purchase or sell is equally billed to the bank account B_t, which corresponds to a *self-financing* trading strategy. In the presence of transaction costs any trading activity is accompanied with an additional loss, $cost < 0$, and the trading constraint becomes:

$$\begin{aligned} \Delta\Pi_{t+1} &= n_{t+1}S_{t+1} + \Delta B_{t+1} - n_t S_t + cost(n_t, n_{t+1}, S_t) \\ &= n_{t+1}\Delta S_{t+1} + cost(n_t, n_{t+1}, S_t). \end{aligned} \tag{1}$$

here and in what follows we adopt the convention $\Delta X_{t+1} := X_{t+1} - X_t$. In utility-based investment theory economic agents plan their investments in such a way as to maximize the expected utility of terminal wealth. The terminal wealth w_T is the sum of the replicating portfolio Π_T and the payoff Z_T of the contract at maturity. The hedger's goal is to choose hedging decisions $(n_1, n_2, \ldots, n_T)$ to maximize the expected utility:

$$\max_{(n_1, n_2, \ldots, n_T)} \mathbb{E}[u(w_T)], \tag{2}$$

where the utility function $u(\cdot)$ is assumed to be smooth and increasing. The expectation is computed with respect to the real-world probabilities, although our experimental framework equally applies to the risk-neutral case. Notice that here the price B_0 is given exogeneously, i.e. (2) applies in a situation, where the derivative contract has already been sold irrespective of risk management. Equation (2) can also serve as a basis to introduce a concept of price in an incomplete market setting. For example the investor might want to optimize[2] the sell price B_0 to simultaneously maximize the expected utility over B_0 and $(n_1, n_2, \ldots, n_T)$. Minimum-risk pricing is the prototypical example of this strategy, see (Schweizer, 1995) and more recently (Buehler et al., 2019). Another pricing concept is indifference-pricing (Hodges and Neuberger, 1989), where B_0 is determined such as to stay indifferent in terms of the expected utility between 1) selling and hedging the contract in the market or 2) trading in the market without entering into the contract at all, see again (Buehler et al., 2019) for a recent discussion.

[1] 'Small' means that the investor's actions have no influence on the market behavior.
[2] Depending on technical conditions an optimum might or might not exist, e.g (Schweizer, 1995).

The dynamic-programming view on hedging is as follows: one time step before maturity the agent chooses an action n_T to maximize the conditional expectation:

$$\mathbb{E}[u(Z_T + \Pi_T)|\mathcal{F}_{T-1}].$$

Conditioning on the filtration $\mathcal{F}_{T-1}$ means that all market information at time $T-1$, including S_{T-1}, Π_{T-1}, …, has been realized and is available for decision making. See (Buehler et al., 2019) for a rigorous construction of such a filtration based on market information in the context of hedging in incomplete markets. The agent proceeds by planning backwards in time. Two steps before maturity the agent chooses action n_{T-1} to maximize the above expectation conditioned on $\mathcal{F}_{T-2}$, three steps before maturity n_{T-2} to maximize conditioned on $\mathcal{F}_{T-3}$, and so on. The optimal course of action corresponds to an optimal *value function* for the hedging problem:

$$V^*(t, S_t, \Pi_t) = \max_{n_1, n_2, \ldots} \mathbb{E}[u(Z_T + \Pi_T)|\mathcal{F}_t].$$

It is a classical result that this function can be obtained as a solution of the Bellman optimality equation (Puterman, 1994). RL is concerned with the learning of approximate solutions to this equation. Remark that the successive optimization of utility expectations corresponds to the Cash Flow formulation for the hedging problem.

The situation is simplified drastically in the P&L formulation, where it is assumed that contract prices Z_t are available a priori throughout the planning process. These prices might be spot mid-prices in case of exchange-traded derivatives or they might originate from an external pricing engine in case of OTC derivatives. In the presence of pricing information the agent is merely faced with an optimization between transaction costs and the immediate risk. Assuming transaction costs and a time-additive utility, a possible choice is the minimization of the immediate risk of the form:

$$Risk_t = \mathbb{E}\left[\left(\Delta Z_{t+1} + n_{t+1}\Delta S_{t+1} + cost(n_t, n_{t+1}, S_t)\right)^2 |\mathcal{F}_t\right]. \tag{3}$$

It should be mentioned that if Z_t are computed externally by a pricing engine without accounting for transaction-costs, one might be tempted to interpret equations of the form (3) (or (7) below) as a "way to generalize hedging" to the situation of transaction costs and discretization error. In theory, computing hedges in this way, is not guaranteed to result in the optimal terminal utility, even approximately. The definition of prices via risk-minimization or indifference pricing requires the full solution of the RL problem with transaction costs, while the optimization given pre-computed prices will, in general, not provide a meaningful approximation. However, and this is one of the main messages of the article at hand, in the practical work of an investment firm, the course of hedging decisions is dictated rather by the availability of information than the seek for theoretically optimal pricing. In the absence of a market model the exact computation of indifference/minimum-risk prices becomes elusive, leaving the investor with the task of choosing *reasonable* hedges given available data and infrastructure. In the sequel we follow the line of Kolm and Ritter (2019) focusing on the accounting P&L formulation, which provides an efficient approximate hedging framework in practice. In what follows we argue that, both from a theoretical and practical perspective, if RL is applied to this problem, then a contextual bandit-type algorithm should be employed.

2.2. Dynamic programming and the BSM economy

We illustrate the dynamic programming perspective on the hedging problem by a familiar example. Consider hedging of a European call in a frictionless market and assume that the stock price process S_t is of geometric Brownian motion (GBM) type. In this case the contract liability is the option payoff

$$Z_T = C_T = -(S_T - K)^+.$$

Let the terminal wealth be split into an initial endowment and a series of wealth increments:

$$w_T = w_0 + \sum_t \Delta w_{t+1}.$$

In the absence of transaction costs the wealth increment is $\Delta w_{t+1} = \Delta C_{t+1} + \Delta \Pi_{t+1}$. Suppose an agent maximizes (2) by minimizing a risk measure of individual increments:

$$Risk_t = \mathbb{E}\left[\left(\Delta w_{t+1}\right)^2 |\mathcal{F}_t\right].$$

This corresponds to a time-additive utility of the form $u(w_T) = \sum_t u_t(\Delta w_{t+1})$ with quadratic u_t. The agent then proceeds by choosing n_T to minimize $Risk_{T-1}$, then n_{T-1} to minimize $Risk_{T-2}$, etc. At each time step the price C_t can be defined as the minimizer of $Risk_t$ given the optimal hedge. In the limit of a continuous action space and continuous time this procedure gives rise to the familiar BSM hedging. The optimal action then becomes the δ-hedge:

$$\delta_t = \frac{\partial C_t^*}{\partial S_t},$$

where C_t^* denotes the BSM option price, leading to a complete off-set of risk and a utility of 0. In the presence of transaction costs and discretization error, risk-free hedging is not possible. The computation of optimal hedges requires the solution of the full-fledged RL problem, because transaction costs introduce reward-dependencies even for a time-additive utility.

2.3. Contextual multi-armed bandit (CMAB) model

The k-armed bandit is the prototypical instance of RL and has been studied extensively within various areas of practical interest (Sutton and Barto, 2018). The setup involves a set of $k \in \mathbb{N}_+$ possible choices[3] and a sequence of T periods. In each period $t = 1, \ldots, T$ the learner chooses action $a \in \{a_1, .., a_k\}$ and receives a random reward $r_t(a) = r(a, t)$. The main assumption is that rewards are independent over a and i.i.d. over t. The objective is to identify a "policy", i.e. a probabilistic rule of action, in order to maximize the cumulative expected reward:

$$\sum_{t=1}^{T} \mathbb{E}[r_t],$$

Over the execution episode. In the context of derivatives hedging this form of reward-optimization corresponds to a time-additive utility functional in (2), where each r_t stands for the utility of a wealth increment. In the *contextual* k-armed bandit setting, the agent is faced with non-stationary reward distributions. In each round, prior to taking its decision, the agent receives *context* about the state of the system. Rewards then depend on the context in a hidden way, which has to be learned by the agent. For example, an agent could be presented with the tuple (t, n_t, C_t, S_t) to assist the minimization of (3). The contextual bandit involves both the association of rewards with the given context but also trial-and-error learning to search for strong actions. Contextual search tasks are an intermediate between the prototypical k-armed bandit and the full RL problem (Sutton and Barto, 2018). They are like the full RL problem in that they involve learning a policy, but each action affects only the immediate reward. Like any RL agent, the contextual k-armed bandit must bargain between exploration and exploitation: is it better to choose a lucrative action given a context or should the agent explore in hope to find something even better? A priori exploration implies risk in the sense that the agent must deviate from an optimal action. To take account of the hedging risk we employ a *risk-averse* version of CMAB, which we call R-CMAB.

While in the standard bandit problem, the objective is to select the arm associated to the highest reward, the mean-variance bandit aims at most effectively trading off expected rewards versus variance (i.e. risk). The mean-variance of action a is defined as:

$$MV(a) := \kappa\mu(a) - \sigma^2(a), \tag{4}$$

where $\kappa \in [0, 1]$ measures the risk aversion, and $\mu(\cdot)$ and $\sigma(\cdot)$ are, respectively, the mean and the standard deviation associated to action a. The optimization of the mean-variance under transaction costs for hedging has been proposed in Kolm and Ritter (2019) and constitutes a generalization of the reward structure (3). For learning the agent builds up an empirical reward statistics. For sample outcomes $\{X_l(a)\}_{l=1,\ldots,T}$ of action a, the empirical mean-variance at a given time t is:

$$\widehat{MV}_t(a) = \kappa\widehat{\mu}_t(a) - \widehat{\sigma}_t(a),$$

with empirical mean and standard deviation given by:

$$\widehat{\mu}_t(a) = \frac{1}{t}\sum_{l=1}^{t} X_l(a), \quad \text{and} \quad \widehat{\sigma}_t(a) = \frac{1}{t}\sum_{l=1}^{t} (X_l(a) - \widehat{\mu}_t(a))^2. \tag{5}$$

Notice that the functional (4) does not have the form of an expectation value and is not directly accessible to RL. The mean-variance multi-armed bandit (Sani et al., 2012) is designed for reward signals of this form. In fact, an important motivation to study mean-variance bandits has been the frequent occurrence of mean-variance investors in portfolio optimization (Sattar and Qing, 2016). Similar to ordinary bandits, confidence bound-based algorithms (specifically the Mean-Variance Lower Confidence Bound algorithm) are available for the risk-averse setting and possess of strong performance and convergence guarantees, see (Sani et al., 2012; Sattar and Qing, 2016) for details. An alternative approach to RL with mean-variance objective proposed in Kolm and Ritter (2019) is to expand the variance term and to approximate cross-covariances by 0. Commonly, the availability of data and practical operations at an investment firm dictate a preference for simpler methods, which makes this approximation very reasonable from a practical perspective. To our knowledge the article at hand contains the first application of a mean-variance contextual bandit system. There is currently no rigorous consistency analysis for such algorithms.

2.4. Q-learning

Q-learning is a classical algorithm for the off-policy approximation of solutions to general RL problems beyond the maximization of immediate rewards (Watkins and Dayan, 1992). The agent stores a Q-table that contains estimates of the Q-uality of state–actions pairs (s_t, a_t) in view of their terminal reward.[4] At each time step, the agent *i)* chooses the best action a_t from the table, *ii)* observes the corresponding reward r_t, and *iii)* updates the estimates of the Q-table according to a specific updating rule. For Q-learning this rule is given by:

[3] The name originates from a gambler who chooses from k slot machines.

[4] Remark that the state s_t at time t can contain multiple items of information, e.g. $s_t = (t, n_t, S_t, C_t)$ for option hedging.

$$\widehat{q}_{new}(s_t, a_t) \leftarrow \widehat{q}_{old}(s_t, a_t) + \alpha\left[r_t + \gamma \max_a \ \widehat{q}_{old}(s_{t+1}, a) - \widehat{q}_{old}(s_t, a_t)\right], \tag{6}$$

where $\widehat{q}$ denotes the empirical estimate of Q values. A key point is that being a temporal-difference method, Q-learning performs the update (6) immediately after each time step as opposed to plain Monte Carlo methods that execute:

$$\widehat{q}_{new}(s_t, a_t) \leftarrow \widehat{q}_{old}(s_t, a_t) + \alpha\left[\sum_{k=t}^{T} r_k - \widehat{q}_{old}(s_t, a_t)\right],$$

After each learning episode is completed. Thus (6) contains two types of estimates. First, there is the estimation of $q(s_t, a_t)$ via a Monte Carlo estimator $\widehat{q}_{new/old}(s_t, a_t)$. As for an ordinary Monte Carlo method, the step-size parameter $\alpha \in (0, 1]$ in (6) weights the relevance of the new sample versus the held estimate $\widehat{q}_{old}$. Second, the quantity $r_t + \gamma \max_a \ \widehat{q}(s_{t+1}, a)$ serves as a Bellman estimate of the total expected reward. $\gamma \in (0, 1]$ is a discount factor that reduces the value of rewards that are further away in the future. Q-learning prioritizes immediate rewards for small values of γ. When γ is close to 1 Q-learning takes account of future rewards making it a full-fledged multi-period planning algorithm (Sutton and Barto, 2018). Some benchmarks on control environments are given in Melnikov et al. (2018). The iterative application of the updating procedure is guaranteed to converge to the optimal policy if the step-size is small enough (Sutton and Barto, 2018). DQN algorithms enhance this procedure by making use of a (deep) NN for the representation of Q values. The NN is trained on Monte Carlo samples once sufficient data is available, boosting the quality of the samples of subsequent simulations, see Section 3.2 for details.

2.5. Machine learning algorithms for financial derivatives hedging

Q-learning addresses the computation of optimal hedges in the Cash Flow formulation. In principle it can be employed in situations of strong environment feedback, reflecting the impact of trading decisions to the market. Implementations of Q-learning for the Cash Flow formulation can be found in Qlbs (2020); Cao et al. (2021) and for the P&L formulation in Kolm and Ritter (2019). For hedging in the P&L formulation it is sufficient to optimize the immediate rewards, which can be achieved by contextual bandit-type algorithms. For the P&L formulation we follow (Kolm and Ritter, 2019) in choosing the mean-variance reward function for our experiments (extending (3)):

$$\begin{aligned} MV_t \quad &= \kappa \mathbb{E}[(\Delta Z_{t+1} + n_{t+1}\Delta S_{t+1} + cost(n_t, n_{t+1}, S_t))|\mathcal{F}_t] \\ &\quad -\mathbb{V}[(\Delta Z_{t+1} + n_{t+1}\Delta S_{t+1} + cost(n_t, n_{t+1}, S_t))|\mathcal{F}_t], \end{aligned} \tag{7}$$

where $\mathbb{V}[X]$ stands for the variance of X. If $\kappa = 0$ is chosen the hedging target is simply to minimize immediate variance risk under transaction costs. A choice of $\kappa > 0$ would imply a preference towards generating wealth from the hedging strategy, although this is rarely attempted in practice. This reward is of the form (4) and fits with the reward structure of the R-CMAB algorithm. For DQN we approximate the variance as proposed in (Kolm and Ritter, 2019).

RL algorithms are capable of learning without human supervision and experience. This comes at the cost of a larger computational effort and less sample efficiency as compared to problem-tailored methods and supervised learning. In the derivative hedging area, the articles (Buehler et al., 2019, 2019) interpret the optimization problem (2) as a supervised learning exercise. The expected utility is maximized by an optimization over the whole predictable sequence of hedging decisions (n_1, n_2, …, n_T) by training a recurrent NN on random market paths. In sophisticated domains like the games of Chess or Go the gradient descent-based optimization of rewards will usually get stuck at a poor local maximum because of the high complexity of the value function or non-differentiable reward structure. However, in hedging scenarios, where a market model is available, gradient descent-based optimization provides a sample-efficient and stable approach to the hedging problem, which also scales favorably in multi-asset portfolio optimization tasks (Buehler et al., 2019, 2019). Q-learning, as a model-free RL algorithm, is capable of learning solely from a reward signal and does, in principle, not require an explicit environment for training. However, practical consideration usually dictate the requirement for a market model in the context of derivatives hedging: there is simply no real-world data to train a model-free algorithm like Q-learning, see below for various training procedures and required data sizes. Variations of Q-learning have become somewhat outdated for planning problems, when an explicit model (e.g., the rules of a game, or a market) is available as the capability of learning without an environment model comes at the price of a larger number of required training samples (Sutton and Barto, 2018; Botvinick et al., 2019). Model-based algorithms, such as Neural Monte Carlo Tree Search (MCTS), learn stronger policies faster by leveraging the environment in a target-oriented way, see e.g (Botvinick et al., 2019; Silver et al., 2017; Anthony et al., 2017). MCTS is a genuine planning algorithm, that combines RL with search techniques to direct the learning process to promising courses of actions (Browne et al., 2012). As compared to Monte Carlo methods (including Q-learning), MCTS builds a problem-specific, asymmetric and heavily restricted Monte Carlo decision tree. The performance of the famous Atari DQN (Mnih et al., 2013) has, for instance, been improved using MCTS (Guo et al., 2014). Similar to (Buehler et al., 2019, 2019), MCTS easily learns the hedging of financial derivatives in the Cash Flow formulation (Szehr, 2021) in case that a market model is given a priori. MCTS has also been used for hedging (and portfolio optimization) with real-world data when price information is available (Vittori et al., 2021). In case no market model is available, the exact solution of the hedging and pricing problems in the Cash Flow formulation will often be elusive. The only viable option seems to revert to a bandit model and P&L hedging as a practical approximation. This is illustrated by our experimental findings, Section 4, which give a detailed comparison of our CMAB algorithm to DQN.

Table 1
Heuristic comparison of some features of algorithms for derivatives hedging in the literature.

	Existing Approaches	Market Model	Data Consumption	Training Stability	Generalization Capability	Cash Flow/P&L
Superevised Learning	Recurrent NN/LSTM	Required	Medium	High	Low	Cash Flow
Reinforcement Learning	CMAB(proposed)	Not Required	Medium	High	Low	P&L
	Q-Learning	Not Required	High	Low	Low	Cash Flow/P&L
	MCTS	Required	Medium	High	High	Cash Flow/P&L

As a summary of this discussion and a general orientation for the reader, a heuristic comparison of machine learning algorithms is provided in Table 1. A crude classification of learning methods is given in the "**Existing Approaches**" column. It should be said that many variations of these methods (such as the application of NNs, the specific architecture of NNs, the structure of the reward signal in case of RL) have been studied, and each one comes with its own specific features. The "**Market Model**" column shows if a market model is required in principle (i.e. if the algorithm is model-free or not), although in realistic operations all algorithms will require a market simulator due to lack of data. The absence of an environment model comes at the price of a higher data consumption and a lower stability of training. An orientation on the amount of data required for training and adaptation is given in the "**Data Consumption**" column. For DQN and R-CMAB a comparison of data consumption during training is given in Section 4.1. Section 4.4 reports on data required to adapt to previously unseen market scenarios. The "**Training Stability**" column refers to the empirical variance of terminal rewards obtained over multiple training cycles, see again Section 4.1. The generalization capability "**Generalization Capability**" of ML methods refers to their performance on unseen scenarios. For hedging there is currently not much research on this topic; most systems being designed to "overfit" to a specific market model or data set. The table therefore reflects our experience with such algorithms on other domains. Finally the "**Cash Flow/P&L**" column reports on whether or not the algorithms have been applied with external price information in the literature.

2.6. Sample efficiency

Roughly speaking, the sample efficiency of a RL algorithm is a measure of how much data it is required to achieve a certain level of performance. We give now a formal definition, which suits our online learning setting. In RL an agent/algorithm $\mathcal{A}$ interacts with an environment $\mathcal{E}$ by taking actions and receiving subsequent rewards in an iterative process until a planning horizon T (assumed finite here) is reached. The dynamics follow a Markov decision process (Sutton and Barto, 2018). The agent's actions are guided by a policy π, i.e. a probabilistic rule of action that $\mathcal{A}$ optimizes as to maximize the accumulated reward over a given planning horizon. The value V_π of π is the expected accumulated reward that is generated by $\mathcal{E}$ when $\mathcal{A}$ follows π. During training the interaction of $\mathcal{A}$ with $\mathcal{E}$ is executed over a number K of episodes, and $\mathcal{E}$ is reset to its original state after each episode. $\mathcal{A}$ collects all observations over all episodes in an observation history of the form $\mathcal{H}_K = \{(h_k, k)\}_{k=1}^K$, where each $h_k = \{(s^{(t)}, a^{(t)}, r^{(t)})\}_{t=1}^T$ contains the observed state, the taken action and the observed reward from time step $t = 1, \ldots, T$ of episode k. In online learning, $\mathcal{A}$ executes actions and observes the impact to the environment through state and reward signals. Notice that new observations are sampled based on probabilities that depend on $\mathcal{E}$ but also on the current policy π (and thus on $\mathcal{A}$. $\mathcal{A}$ will also update its policy in the course of the training process (e.g. after each episode). Finally, the choice of policy by $\mathcal{A}$ contains inherent randomness reflecting unobserved internal states in $\mathcal{A}$ (such as the weights of a neural network or the choice of a random batch for stochastic gradient descent). In other words $\mathcal{A}$ can be viewed as a random mapping that assigns a random policy to an observation history $\pi = \mathcal{A}(\mathcal{H}_K)$ (suppressing the dependency on the internal state of the algorithm in this notation). We say that $\mathcal{A}_1$ has higher efficiency than $\mathcal{A}_2$ after K episodes with respect to $\mathcal{E}$, if after K episodes of interaction with $\mathcal{E}$ the realized policy generated by $\mathcal{A}_1$ has higher value than the policy generated by $\mathcal{A}_2$ in expectation:

$$\mathbb{E}_{\mathcal{H}_{K,1}, \pi_1 = \mathcal{A}_1(\mathcal{H}_{K,1})} V_{\pi_1} > \mathbb{E}_{\mathcal{H}_{K,2}, \pi_2 = \mathcal{A}_2(\mathcal{H}_{K,2})} V_{\pi_2}.$$

The expectation values are taken over choices of training histories $\mathcal{H}_{K,i}$ made of K episodes of interactions of $\mathcal{A}_i$ with $\mathcal{E}$, and over policies generated by $\mathcal{A}_i$, $i = 1, 2$. The policy is updated in the course of the training process, which affects the probabilities of consecutive observations, which then affect the choice of the new policy, and so on. This makes the computation of the probabilities behind the above expectation values an iterative process (we assume that, in the absence of a training history, the algorithms return a policy uniformly at random). Given the iterative nature of the involved expectations, the only viable option to assess this inequality is (Monte Carlo) simulation. We do this by i) initializing the algorithms at random, ii) by training them over K episodes, and iii) by estimating for each episode the value of the current policy, see Section 4. For completeness we mention that the sample efficiency of $\mathcal{A}$ with respect to $\mathcal{E}$ at K episodes can be defined as the fraction of the value $V^* = V_{\pi^*}$ of the optimal policy π^* that is gained after K episodes in expectation: $SE(\mathcal{A}, K) := \mathbb{E}_{\mathcal{H}_K, \pi = \mathcal{A}(\mathcal{H}_K)} V_\pi / V^*$. In other words, given $r \in 0, 1$, the sample efficiency of $\mathcal{A}$ with respect to $\mathcal{E}$ is the smallest episode count K so that $SE(\mathcal{A}, K) \geq r$.

3. Algorithms

This section summarizes the chosen architectures for the contextual bandit and Q-learning algorithms, which are well-known in the literature. In general RL algorithms operate by executing episodes of actions and considering the obtained rewards in order to improve subsequent courses of action. In order to achieve high rewards the algorithm needs to exploit the reward-information acquired from

previous episodes; in order to improve performance it needs to explore potentially sub-optimal courses of action. Broadly, the various RL algorithms differ in the ways how the acquired information is represented and how they handle the trade-off between exploration and exploitation. To achieve a good performance, appropriate representation power and an intelligent coordination of explore/exploit decisions over the learning process are key. In the implementation at hand these points are addressed by the usage of NNs to link contexts and rewards and by making use of Bayesian posterior sampling to coordinate the courses of action. Bayesian posterior sampling maintains a Bayesian estimate of a distribution over reward models through the entire learning process. As reward observations are gathered, this distribution is updated according to Bayes' rule, sharpening it around the expected rewards of each action. At each decision step a set of parameters is sampled and the best action is taken according to the resulting model. As compared to just taking the action of highest estimated reward, this sampling procedure ensures a sufficient level of exploration, where the concentration of the posterior around the expected reward reflects the increasing confidence in the reward estimate. The entire procedure leans at Thompson sampling (Thompson, 1933; Chapelle and Li, 2011) in the non-contextual case but can be viewed as a method to approximate non-tractable Bayesian updates also in the presence of a complex context model. Since market moves are random, the RL agents must operate in an environment of high stochasticity. While hedging has been explored with Q-learning, the application of bandit-type algorithms in such an environment seems new. We have experimented with different architectures, including purely linear inference, variational inference and plain Monte Carlo, similar to (Riquelme et al., 2018), where the chosen design has shown the best performance. All experiments were conducted using PyTorch libraries on an ordinary notebook, Intel(R) Core(TM) i7-8565U CPU @ 1.80 GHz and 16 Gig of RAM.

3.1. Neural network R-CMAB algorithm

To achieve high representation power for the contextual bandit the association of contexts to rewards is reflected by a NN. A direct way to coordinate exploration is to assume a linear Bayesian regression model on top of the last hidden layer of the NN as in Riquelme et al. (2018); Snoek et al. (2015). While the NN is trained to accurately predict rewards, a sampling procedure replaces the deterministic output layer. Concretely, the predicted value v_a for action a is given by $v_a = \beta_a^T z_x$. Here z_x denotes the output of the last hidden layer given context x and the parameter vectors β_a are sampled from the posterior corresponding to action a. The posterior at step t is computed following $\pi_t(\beta_a, \sigma_a^2) = \pi_t(\beta_a|\sigma_a^2)\pi_t(\sigma_a^2)$, where σ_a^2 reflects our confidence about the reward from a and we assume $\sigma_a^2 \sim IG$, and $\beta_a|\sigma_a^2 \sim N$ with an Inverse Gamma and Normal distribution, respectively. The exact rules for the parameter updates are given in Riquelme et al. (2018); Snoek et al. (2015). The correlations present in the sequence of contexts might cause instability of the training process as the NN tends to overfit to this correlation. As a common method to remedy this instability we employ a memory buffer, which stores the transitions (s_t, a_t, r_t, s_{t+1}) for a given time step. During the training process random batches are sampled from this buffer, breaking the inter-temporal correlations. The neural network and the posterior model are updated asynchronously at given updating frequencies. The neural R-CMAB algorithm is summarized in Algorithm 1.

Algorithm 1: Neural R-CMAB Algorithm

```
Input Init. action-value NN with random parameters; Set up prior
  distribution over models, π_0 : θ ∈ Θ → [0,1]. Init. replay memory to
  given capacity;
for episode = 1 : numberOfEpisodes do
    for t = 1 : length(episode) do
        Sample parameters θ_t ~ π_t;
        Get a context vector x_t ∈ R^d;
        Compute a_t = BestAction(x_t, θ_t) through NN followed by linear
          Bayesian model;
        Perform action a_t on hedging environment and observe reward
          r_t;
        Add (x_t, a_t, r_t) to memory buffer;
        if updateModelFrequencyCriterium then
            Update Bayesian estimate of the posterior distribution to
              π_{t+1} using most recent transitions;
        end
        if updateNNFrequencyCriterium then
            Sample random batches from memory buffer to train NN;
        end
    end
end
```

3.2. Neural network Q-learning

The DQN algorithm has been originally designed for learning from alternating visual data (Mnih et al., 2013). Our architecture orients itself broadly at the Atari game example of Mnih et al. (2013) and the R-CMAB above. Similarly to the CMAB algorithm, the Q-values are represented internally by a NN, which is trained on approximate reward targets and exploration is managed as in Mnih et al. (2013) by adding random noise with decaying probability. Following the temporal-difference paradigm the parameters of the NN are updated intra-episode as in (6) on Bellman targets of the form $r_{t+1} + \gamma \max_a \widehat{q}(s_{t+1}, a)$. The gradient then points into the direction of $r_{t+1} + \gamma \max_a \widehat{q}_{old}(s_{t+1}, a) - \widehat{q}_{old}(s_t, a_t)$ as in (6), where α takes the role of the learning rate. As opposed to supervised learning, where

learning targets are fixed, the Bellman targets depend on the NN parameters themselves, which would lead to training the NN on a sequence of targets that change at each learning iteration. Two instances of the same NN equipped with different suits of parameters are employed to stabilize the learning process. The policy NN provides the estimates $\widehat{q}_{old}$ and is trained at each iteration on the Bellman targets. The target NN provides the action values for the computation of Bellman targets. The parameters of the target NN are updated asynchronously every C iteration of the learning process with the parameters of the policy NN. As for the CMAB we employ a replay memory buffer, which stores the transitions (s_t, a_t, r_t, s_{t+1}) for a given time step. During the training of the policy NN random batches are sampled from this buffer. The NN is then updated by the batch stochastic gradient descent algorithm, which serves to stabilize the update via (6). The described DQN algorithm is summarized in Algorithm 2. Typical values for the discount factor γ found in the literature fall in the range 0.99, chosen in Mnih et al. (2013), to 0.999 (Lin, 1992). The effect of a small value for γ is that DQN prioritizes the optimization of immediate rewards over planning for potential future rewards. Finally, let us mention that many variations of the DQN design have appeared in the literature, where more recent architectures are known to outperform the original design in terms of stability and sample efficiency, see e.g (Hessel et al., 2018).

Algorithm 2: Neural (Deep) Q-learning Algorithm

Input Init. action-value target and policy NNs with random parameters; Set up exploration parameter: ϵ. Init. replay memory to given capacity;

for *episode* $= 1 :$ *numberOfEpisodes* **do**

  for $t = 1 :$ *length(episode)* **do**

    Get current state of environment $s_t \in \mathbb{R}^d$;

    Compute action $a_t = \text{BestAction}(s_t, \theta_t)$ through policy NN followed by ϵ-greedy action;

    Perform a_t on hedging env. and observe reward r_t, new state s_{t+1};

    Store transition (s_t, a_t, r_t, s_{t+1}) in the replay memory;

    Sample mini-batch of transitions from the replay memory;

    Compute Bellman targets for states in this mini-batch by evaluating target NN;

    Update the policy NN parameters by one step of stochastic batch gradient descent on Bellman targets;

    if *updateModelFrequencyCriterium* **then**

      Update exploration parameter ϵ;

    end

  end

  if *updateTargetNNFrequencyCriterium* **then**

    Update parameters of target NN by policy NN;

  end

end

4. Proof of concept and experimental findings

This section compares the performance of R-CMAB and DQN in view of a potential deployment in a setting, where training data is limited. We study the P&L formulation, which means that the hedger has access to price information, e.g. through an exogenous pricing engine, and we compute rewards via (7). We focus on the hedging of a short European option contract in a discrete-time setting, but we expect similar outcomes for more general derivative contracts. As is standard in RL the training process is organized in independent "training episodes", see (Sutton, 1988) for an introduction. An episode corresponds to the full life-time of the option contract from inception to maturity. All relevant option and market data is reset to the same values at the beginning of each episode. Each episode is split into a number of equal time intervals, where hedging decisions occur in the beginning of each interval. Training proceeds by considering "reward-samples" (i.e. market-feedback) that the agent receives as a consequence of the preceding hedging decision over multiple episodes. In what follows we will usually start the training process "tabula rasa", i.e., the agent possesses no prior information but learns solely from the observed reward statistics. We will not attempt to train an agent "to perfection" on simulated data, but rather we are interested in the performance after a comparatively small number of training episodes. For better orientation we provide a summary of this section.

Overview over Section 4: Sections 4.1 and 4.2 compare the training and test performance of R-CMAB versus DQN hedgers. Section 4.3 compares the hedgers in the presence of transaction costs. Section 4.4 addresses the capability of pre-trained hedgers to adapt to new market scenarios. Section 4.5 discusses the practical implications of our experimental findings.

Summary of basic experimental setup.

- **Option contract:** Short Euro vanilla call with strike $K = 1.0$ and maturity $T = 60$ days.
- **Market:** Underlying stock price is given by a Geometric Brownian Motion, $S_0 = 1.0$, with $\mu = 0.01$, $\sigma = 0.3$. Flat interest rate market.
- **Hedging agent:** Hedging proceeds in discrete time. Each training episode (corresponding to $T = 60$ days) is discretized into a number 10, 20 or 60 time steps (samples).
- **State space:** The state variable (i.e., the context in case of R-CMAB) is given by the tuple $(S_t, T - t)$. In Section 4.3 the state will also include the number of current holdings n_t. In Section 4.4 the state will include information about market events.
- **Action space:** The agents can choose from 51 actions, corresponding to 25 buy/sell actions and one hold. These actions are gauged to the interval $[-1, 1]$. In Section 4.4 short selling will be switched off, which results in 26 actions. The actions are gauged to $[0, 1]$ in this case.
- **Reward:** The reward function for both DQN and R-CMAB agents is given in (7). For DQN we approximate the variance as outlined in Kolm and Ritter (2019).
- **Transaction costs:** Transaction costs are considered in Section 4.3 only. They are set to 0 for all other experiments.

Summary of algorithm setup.

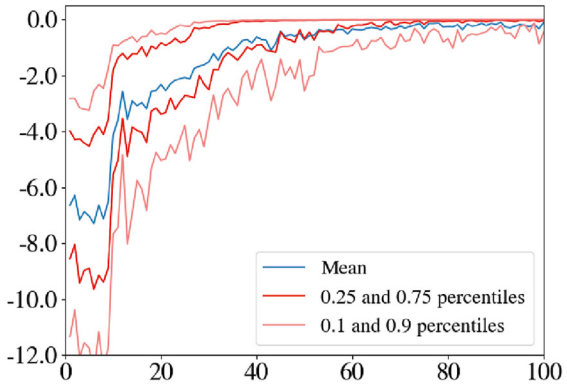
- **R-CMAB algorithm:** NN composed of 3 layers with 20 fully connected nodes each. ReLu activation. SGD optimizer with initial learning rate of 0.001.
- **DQN algorithm:** NN composed of 3 layers with 20 fully connected nodes each. ReLu activation. SGD optimizer with initial learning rate of 0.001.

On the role of the discount factor γ: For the comparison of training and test performance in Sections 4.1 and 4.2 a small discount factor of $\gamma = 0.5$ has been chosen in DQN. In multi-period problems discount factors are typically chosen close to 1, see above. The small value of γ brings DQN closer to a contextual bandit algorithm, which creates favorable training conditions in the P&L formulation of the hedging problem, see (7). Our choice gives DQN an artificial advantage for the comparison with our CMAB algorithm. However, we stress that this choice defies the purpose of a multi-period algorithm in the first place. Sections 4.3 and 4.4 cover our experiments on transaction costs and pre-training. Because these experiments concern DQN's ability to plan ahead, we run them with the more realistic value $\gamma = 0.9$, see below. Unsurprisingly the training speed of DQN deteriorates as γ is increased towards higher values, which is illustrated in Section 4.2 and discussed in Section 4.5. In addition we grant DQN a two times larger training set for all experiments.

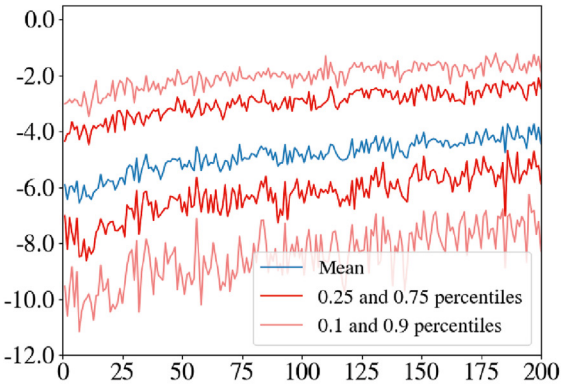
4.1. The training process

We compare the training processes of CMAB and DQN algorithms by presenting some descriptive statistics from learning in a quintessential hedging scenario. In this scenario transaction costs are switched off and we consider purely variance-based hedging (i.e.$\kappa = 0$ in (7)). For DQN $\gamma = 0.5$ is chosen. At each time step the agent chooses from 51 possible hedging decisions, corresponding to 25 buy/sell actions and one hold. (Short-selling is explicitly allowed here, although we are hedging a short call option). To gain a basic quantitative picture we begin with a situation, where we pass the exact expectation value (7) as a reward after each time step. Notice that the computation of this expectation requires information about the underlying market model, which often will not be available in practice. In what follows we call this the "deterministic scenario". The deterministic scenario serves as a benchmark for the subsequent experiments providing estimates on the number of required training samples and training stability in a "trivial" setup. Fig. 1 presents a comparison of episodic rewards during the training process in the deterministic situation. We ran 100 independent training cycles of CMAB (with 100 training episodes) and DQN (with 200 training episodes) for this comparison. Rewards are compared for three distinct hedging frequencies. In Fig. 1(a)–(b) the portfolio is re-balanced 10 times over the option's lifetime (i.e. hedging occurs once every 6 days), in Fig. 1(c)–(d) it is re-balanced 20 times (i.e. hedging occurs once every 3 days) and in Fig. 1(e)–(f) it is re-balanced 60 times (i.e. hedging occurs daily). Notice that a larger hedging frequency implies a slower training progress because less samples are presented to the algorithms until maturity. Rewards increase consistently for both algorithms but CMAB learns faster on the deterministic benchmark reaching a near 0 mean reward in all cases.

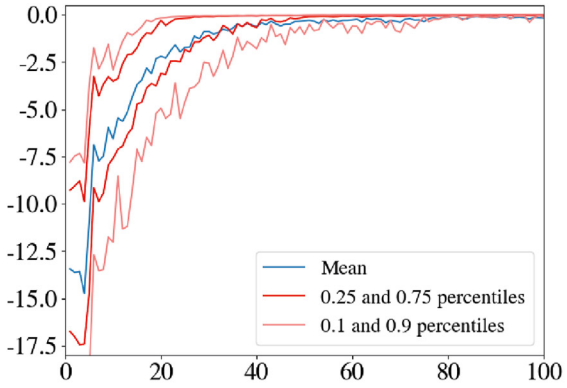
In the absence of a market model, the expectation (7) is unknown and the agent learns from the random rewards obtained during the training process. In this situation the agent is left with choosing hedges and building up an internal model of the market's hidden reward structure according to the realized rewards. We call this the "non-deterministic scenario". The training progress in the non-deterministic situation is presented in Fig. 2. As for the deterministic case we ran 100 independent training cycles of CMAB (with 100 training episodes) and DQN (with 200 training episodes) for this comparison. As before rewards are compared for three distinct hedging frequencies. In Fig. 2(a)–(b), the portfolio is re-balanced 10 times over the option's lifetime, in Fig. 2(c)–(d) it is re-balanced 20 times and in Fig. 2(e)–(f) it is re-balanced 60 times. As before a larger hedging frequency implies a slower training progress because less samples are presented to the algorithms until maturity. Since the agent decides on a hedge first and the reward is realized only one time-step later, this setting comes with an inevitable residual risk (here "residual variance" since $\kappa = 0$) that cannot be hedged away. In other words, as compared to the familiar BSM economy, time- and action-space discretization entail a minimum rate of hedging error in the setting at



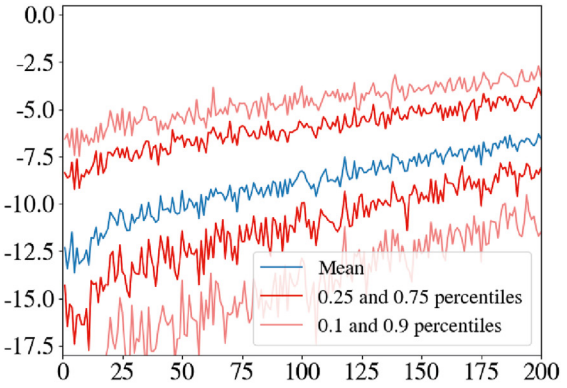
(a) CMAB - 10 samples per episode



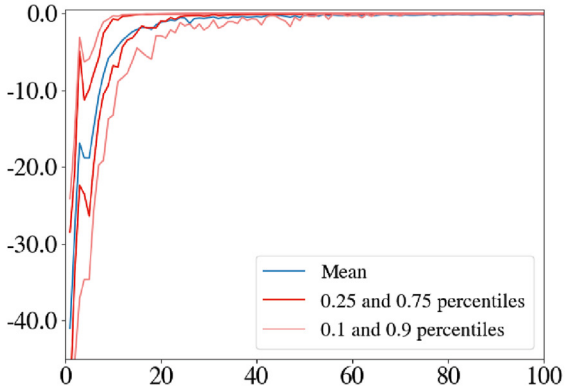
(b) DQN - 10 samples per episode



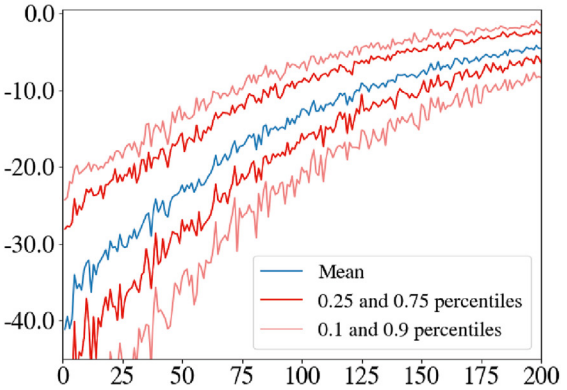
(c) CMAB - 20 samples per episode



(d) DQN - 20 samples per episode



(e) CMAB - 60 samples per episode



(f) DQN - 60 samples per episode

Fig. 1. Comparison of training progress of CMAB (left column) and DQN (right column) in the deterministic scenario. Graphs show descriptive statistics of episodic rewards obtained from 100 independent training cycles.

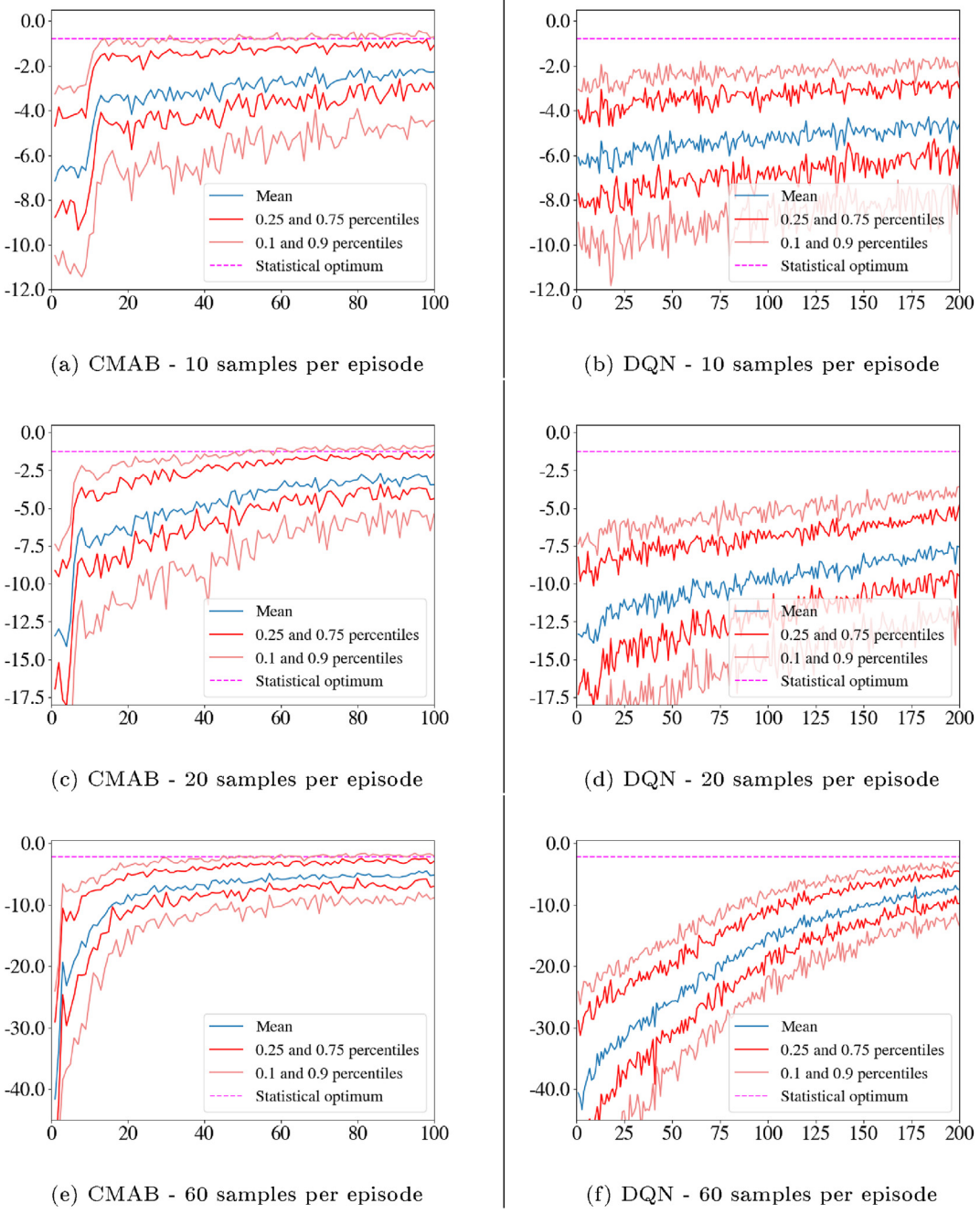
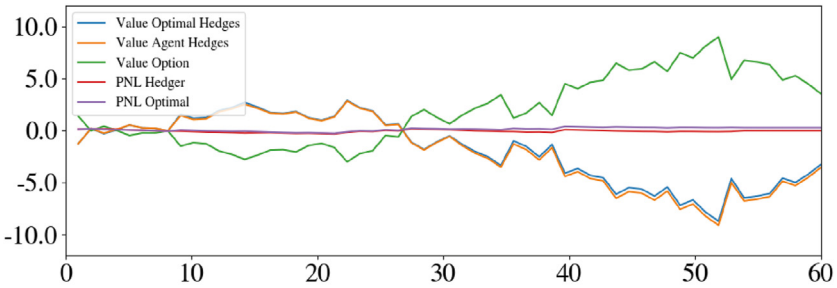


Fig. 2. Comparison of training progress of CMAB (left column) and DQN (right column) in the non-deterministic scenario. Graphs show descriptive statistics of episodic rewards obtained from 100 independent training cycles.

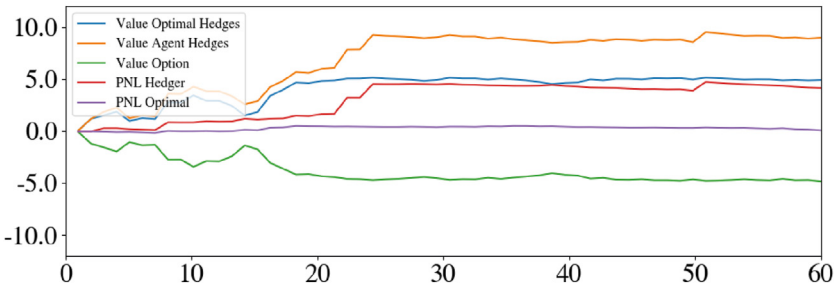
hand. We compute this risk for an optimal oracle hedger that knows the underlying market model and chooses hedges to maximize the reward (7) and depict it by horizontal lines in Fig. 2. For CMAB the terminal mean episodic rewards in Fig. 2 (a),(c),(e) demonstrate that the mean error per hedging decision obtained after training decreases as the hedging frequency increases. This is in line with the theoretical expectation that CMAB hedging should converge to riskless BSM hedging in the continuous limit.

4.2. Comparison after training

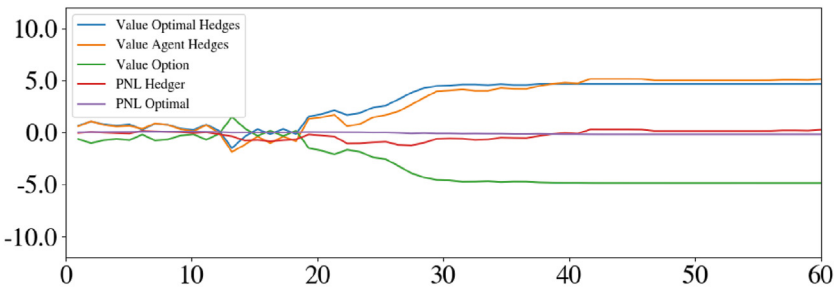
We compare the performance of CMAB and DQN hedgers after training in the scenario of Section 4.1. As an illustration Fig. 3 shows exemplary P&L hedging paths obtained after training. To highlight the impact of discretization error, we also added the aforementioned oracle agent, that knows the underlying market model and takes the optimal actions accordingly. We evaluate the hedging performance by measuring the terminal P&L on 100 GBM paths that share the same parameters but are not contained in the original training set. The



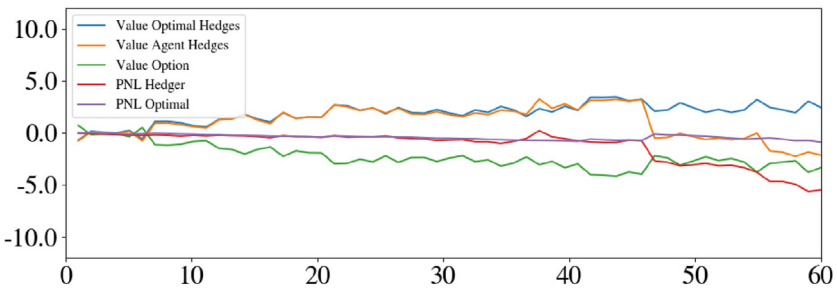
(a) CMAB - 100 episodes, 60 samples per episode, deterministic scenario



(b) DQN - 200 episodes, 60 samples per episode, deterministic scenario



(c) CMAB - 100 episodes, 60 samples per episode, non-deterministic scenario



(d) DQN - 200 episodes, 60 samples per episode, non-deterministic scenario

Fig. 3. Exemplary P&L of hedging paths from trained CMAB and DQN agents.

resulting histograms of terminal P&L are shown in Fig. 4 for the deterministic and in Fig. 5 for the random market. As for the training progress P&Ls are compared for three distinct hedging frequencies (where in Figs. 4 and 5(a)–(b) the portfolio is re-balanced 10 times, in (c)-(d) 20 times and in (e)-(f) 60 times). Increasing the hedging frequency leads to an increasing concentration of the P&L statistics around 0 witnessing an improvement in the overall hedging performance. To obtain a clearer picture about the actions chosen by the trained agents we compare them to the respective BSM δ on 1000 test samples, where the impact of discretization becomes apparent. Figs. 6 and 7 show the dependency of the chosen actions on the underlying stock price for three hedging frequencies (corresponding to 10, 20 and 60 re-balancings). Finally Fig. 8 demonstrates the deterioration of DQN's performance as γ increases in the non-deterministic scenario.

4.3. Transaction costs

DQN is designed to address the full-fledged RL problem surpassing the planning capability of the CMAB algorithm. In this section we aim for evidence whether or not this capability aids in planning hedging decisions in the presence of transaction costs. For our experiment we consider proportional transaction costs of the form

$$cost(n_t, n_{t+1}, S_t) = -\eta \cdot S_t \cdot |n_{t+1} - n_t|,$$

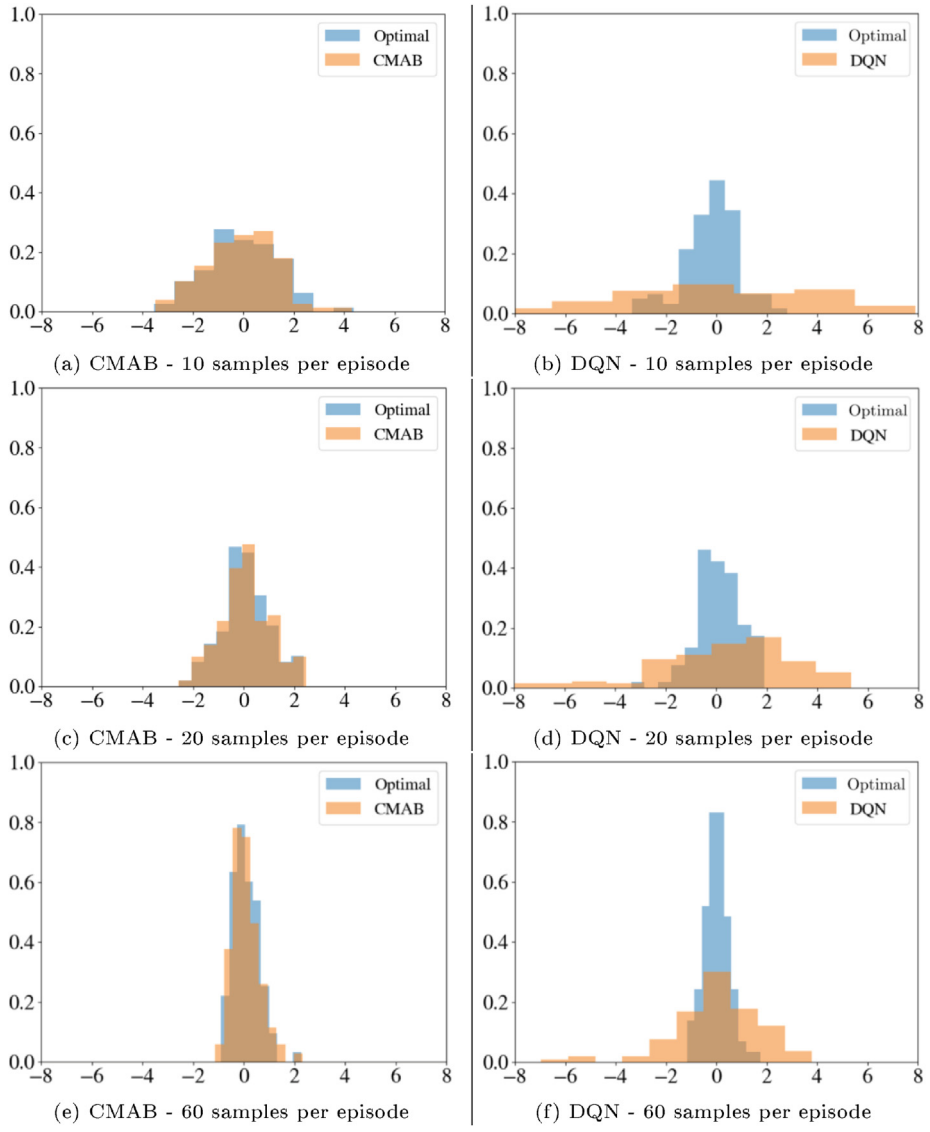


Fig. 4. Histograms of the terminal P&L measured on 100 test paths of CMAB (left column) and DQN (right column) agents in the deterministic scenario.

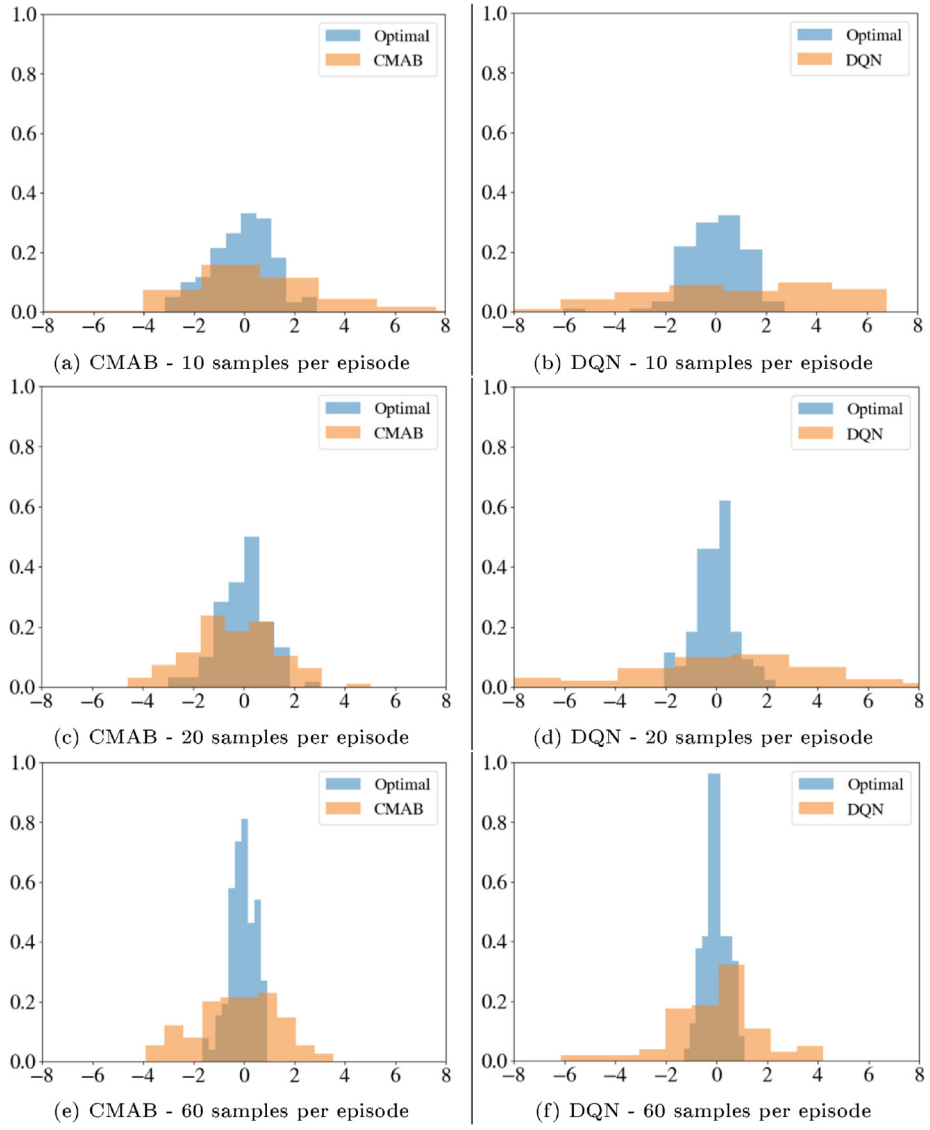
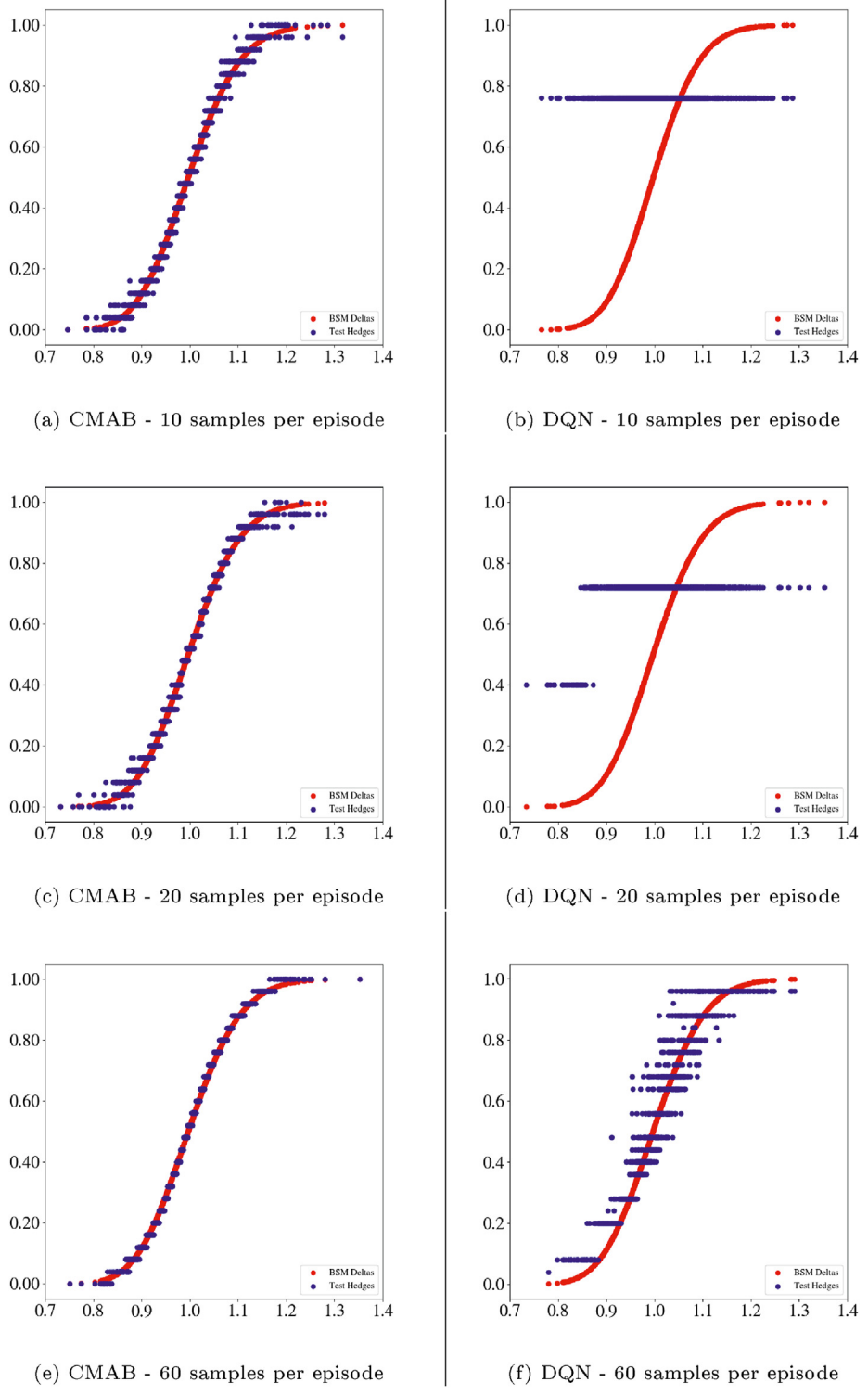


Fig. 5. Histograms of the terminal P&L measured on 100 test paths of CMAB (left column) and DQN (right column) agents in the non-deterministic scenario.

where n_t denotes the number of shares currently held, n_{t+1} is the number of shares chosen for the coming period, and $\eta \geq 0$ is a constant pre-factor. We choose $\eta = 0.1$ for our experiments. The current holdings n_t are passed to the agents as part of the context or the state. To asses whether the planning capability of DQN reduces transaction costs we choose a slightly different experimental setup for both agents. DQN is set up as a multi-period planning algorithm, which minimizes transaction costs over the entire lifetime of the option contract ($\gamma = 0.9$). The current holdings n_t are part of the state, both during training and testing. In contrast, the CMAB algorithm is trained to optimize immediate rewards by providing *random* context about the number of shares. For testing, however, the choice n_t of the previous period becomes the context of the next period. This corresponds to hedging in the real world, where the current holdings are a consequence of the preceding investment decision.

As before we grant 100 training episodes to CMAB but, this time, 5000 episodes to DQN. The accumulated transaction costs from a test set of 1000 unseen episodes are compared in Fig. 9. As an immediate benchmark we also added the transaction costs that would emerge from rounding BSM model hedges to the closest possible action. Notably, our transaction cost histogram for CMAB is comparable to the one obtained for DQN in [15, Exhibit 4]. While (Kolm and Ritter, 2019) reports training on 5 batches of 15000 episodes with 50 time steps each, we have used 100 episodes with 60 steps.



(a) CMAB - 10 samples per episode

(b) DQN - 10 samples per episode

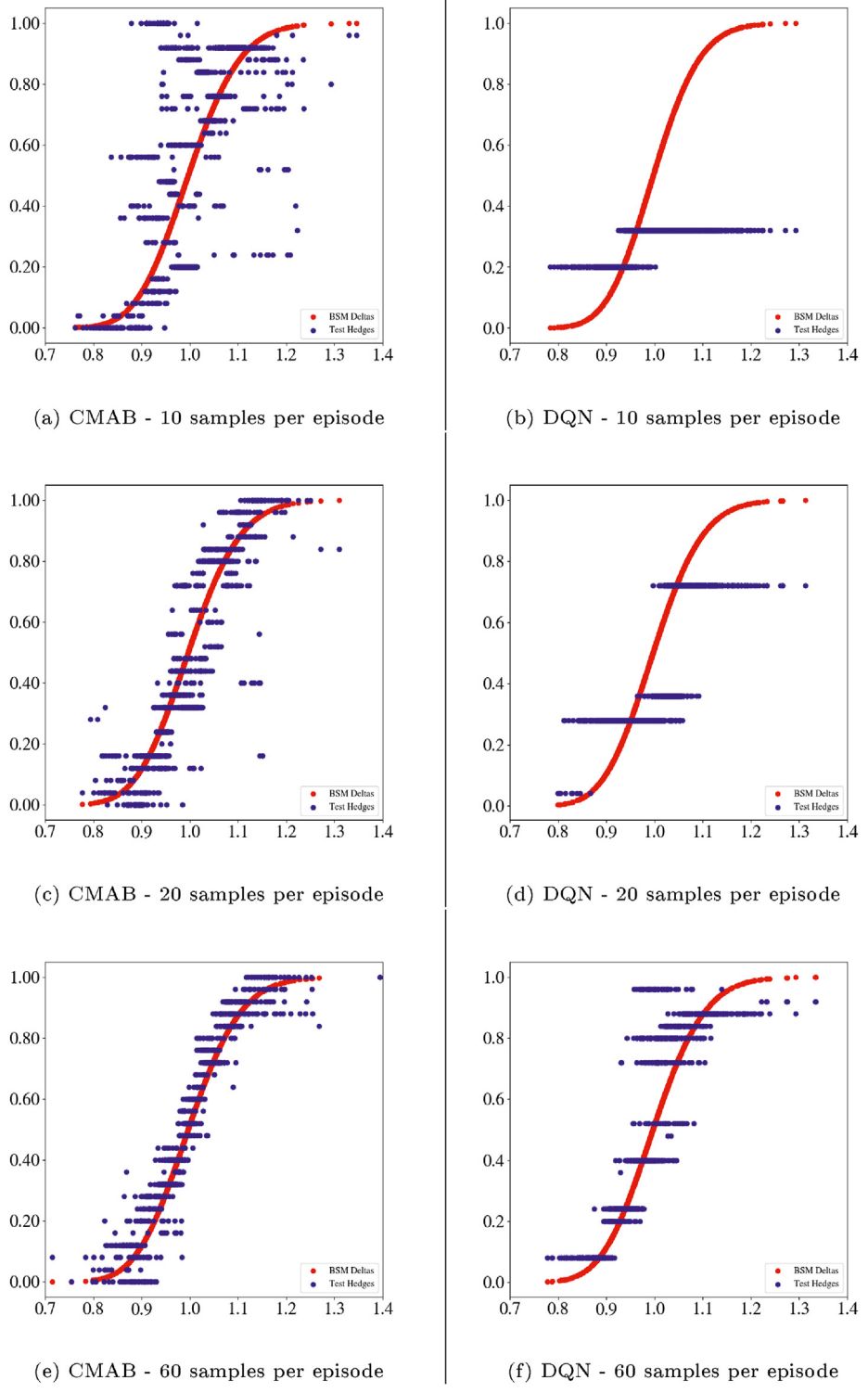
(c) CMAB - 20 samples per episode

(d) DQN - 20 samples per episode

(e) CMAB - 60 samples per episode

(f) DQN - 60 samples per episode

Fig. 6. Comparison of agent actions versus BSM δ at the mid-point of the investment horizon in deterministic scenario. *X*-axis: Value of underlying, *y*-axis: hedging action.



(a) CMAB - 10 samples per episode

(b) DQN - 10 samples per episode

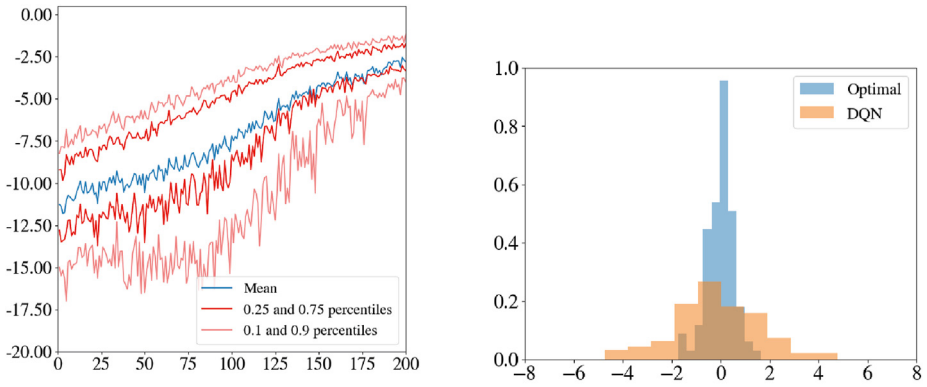
(c) CMAB - 20 samples per episode

(d) DQN - 20 samples per episode

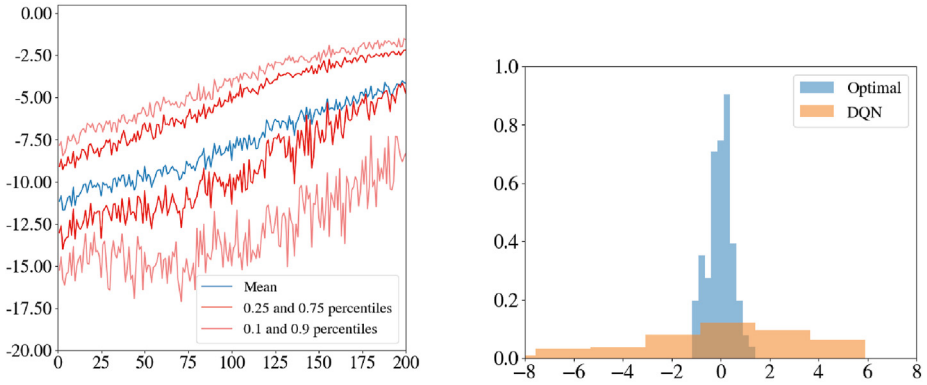
(e) CMAB - 60 samples per episode

(f) DQN - 60 samples per episode

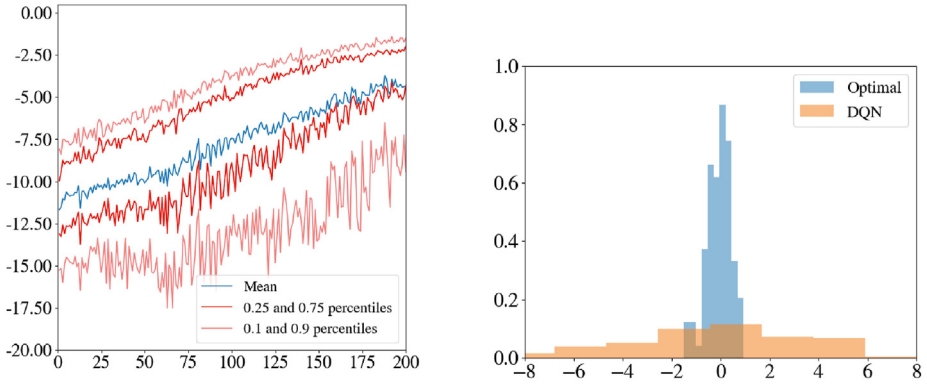
Fig. 7. Comparison of agent actions versus BSM δ at the mid-point of the investment horizon in non-deterministic scenario. *X*-axis: Value of underlying, *y*-axis: hedging action.



(a) DQN - $\gamma = 0.5$



(b) DQN - $\gamma = 0.9$



(c) DQN - $\gamma = 0.99$

Fig. 8. Deterioration of training and test performance of DQN for P&L hedging. Left column: Descriptive statistics over 100 training cycles with 60 samples per episode. Right column: Exemplary terminal P&L histograms on 100 test GBM paths.

4.4. Pre-training and adaptation

In the practical work of an investment firm the availability of real-world data for training will be the bottle-neck for the deployment of any form of AI for hedging. We have seen in Section 4.1 that training from real-world market data (i.e. in the non-deterministic scenario) is data costly, even for the minimalist' CMAB agent. A potential way out is that in applications the AI is *pre-trained* on

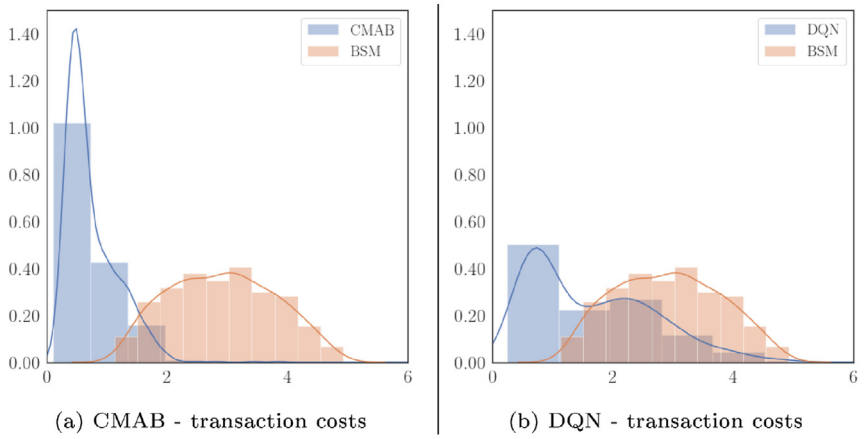


Fig. 9. Histograms of accumulated transaction costs measured on 1000 testing episodes for CMAB (left) and DQN (right).

simulated data and then *fine-tuned* on available real-world data. In this process sample-efficiency is key because only an efficient AI can adapt sufficiently fast to the new information. An inefficient AI, to the contrary, would simply over-fit to the chosen simulation models (for markets, transaction costs, etc …) and ignore the important real-world information. To analyse the adaptation capability of the CMAB and DQN algorithms we have designed the following experiment:

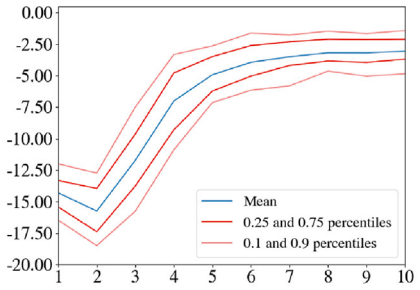
In the non-deterministic scenario the agent is pre-trained until it reaches the maximum reward threshold (300 episodes of 60 samples for CMAB, 600 episodes of 60 samples for DQN, $\gamma = 0.9$). This training process mimics the "training on simulated data" in practice. Once the model is fully trained we investigate how quickly it adapts to a previously unseen market reality. For this we invent random *market catastrophe events*, which are provided as an additional binary context to the trained agent. In case of market catastrophe the agent should sell all holdings in the underlying immediately, otherwise it receives a large negative reward. In the absence of a catastrophe the agent should hedge as learned in pre-training. In the sequel we investigate the adaptation of the pre-trained agents to the new scenario. Since catastrophic events are rather sparse on the real markets we choose 10 such events per training episode. An important point is that while catastrophes occur at the same time steps at each episode during training, in testing catastrophe events occur at random times. In other words we recycle the same sparse catastrophe data for training but we test in a more realistic setting with unpredictable times of catastrophe. Given the large consumption of data of DQN in preceding experiments we switched off transaction costs and short-selling for this experiment. The re-training progress of the pre-trained CMAB and DQN agents on episodes with added catastrophe events is depicted in Fig. 10.

The experimental findings demonstrate that after approximately 8 training episodes the pre-trained CMAB agent has adapted to the new market context in terms of accumulated rewards and erroneous actions on catastrophe events. First CMAB agents show 0 erroneous actions after 4 training episodes. The DQN agent did not adapt fully to the new market setting even after 50 training episodes. To gain a clearer picture on how well the CMAB agent adapted to the new context we compare the terminal P&L statistics and the accumulated erroneous actions before and after re-training on 100 previously unseen samples, see Fig. 11. We observe that during the test cycle the trained agent did not choose a single erroneous action in case of market catastrophe, although such events occurred at random moments in time. The P&L performance of the hedger has shown only slight deterioration. We conclude that the pre-trained CMAB agent successfully adapted to the new market information on a set of sparse market events (that have been used multiple times in the training cycle). This is a key feature of the CMAB agent as it is prerequisite to the deployment of pre-trained models in a practical setting.

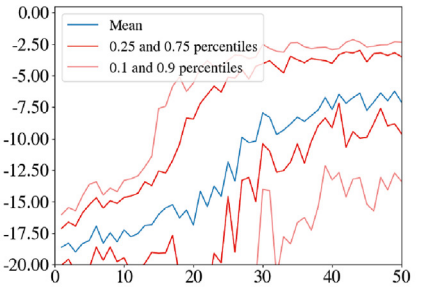
4.5. R-CMAB versus DQN for derivatives hedging in practice

Conceptually our choice of a bandit-type formulation resembles business operations at an investment firm. It is rare that hedging decisions target the optimization of multi-period transaction costs and prices as in the full-fledged RL problem. Instead, traders consider the current market state and optimize for the immediate monetary targets within thresholds of immediate risk (which will almost certainly include a mark-to-market of the traded structure to an internally approved model). Transaction costs and risk quantifiers are monitored on a daily basis and reported to management and regulators. Most trading desks are subject to a forced liquidation upon reaching a maximum drawdown (or risk limit): this absorbing state has a profound impact on the theoretical value of an option, yet it is wholesale ignored in most modeling approaches. Long-term planning in the sense of taking sacrifice temporarily to achieve better terminal reward (as it is typical for games like Chess) is therefore often an impractical goal within realistic business constraints.

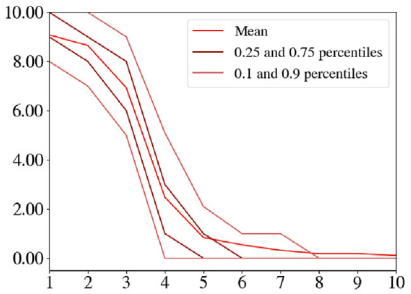
Notably, in the absence of transaction costs and adjustment for trading gains the CMAB algorithm minimizes the hedging error for each individual time period. In theory this procedure converges to the BSM δ in the limit of a continuous action space and continuous time. Consistently, in our experiments the risk associated with CMAB hedges decreases as the hedging frequency increases. This feature is desirable for benchmarking and not present in the Q-learning framework: While the value of an option contract in the BSM economy follows the BSM differential equation, Q-learning is designed to approximate solutions of the Bellman equation. As a consequence it is suitable for terminal utility hedging in the Cash Flow formulation, when the value function follows a Bellman equation, such as in



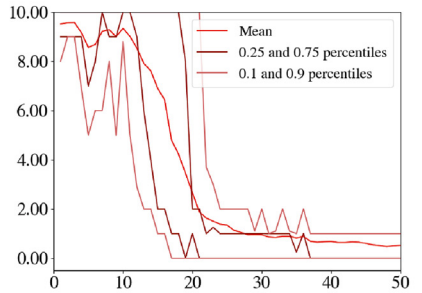
(a) CMAB - Episodic rewards.



(b) DQN - Episodic rewards.

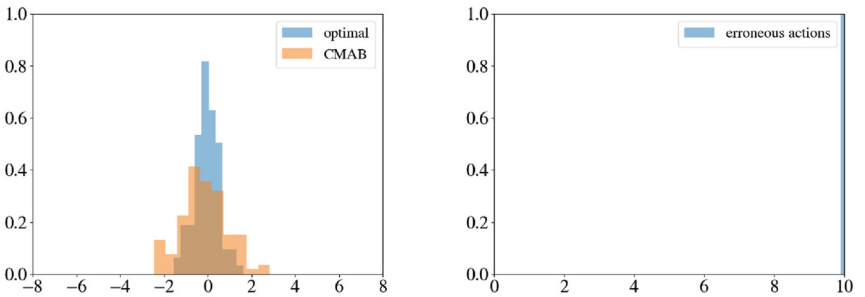


(c) CMAB - Erroneous actions in the presence of market catastrophe, accumulated over training episodes.

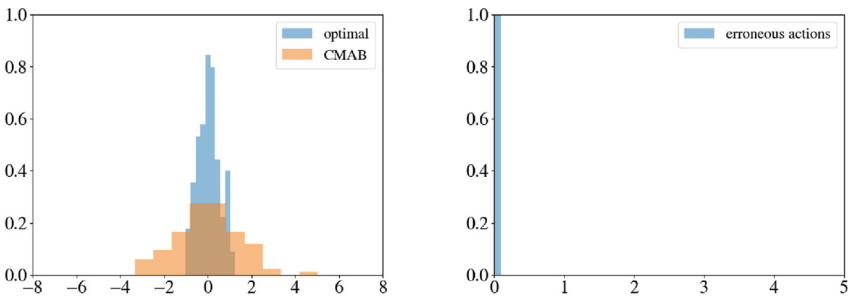


(d) DQN - Erroneous actions in the presence of market catastrophe, accumulated over training episodes.

Fig. 10. Adaptation of pre-trained CMAB (left column) and DQN (right column) agents to catastrophe context. Graphs show descriptive statistics from independent 100 training cycles with 60 samples per episode.



(a) Pre-trained agent, before adaptation to market catastrophe context.



(b) After adaptation to market catastrophe context.

Fig. 11. Comparison of terminal P&L (left column) and erroneous actions (right column) accumulated over test episodes for the CMAB agent.

Hodges and Neuberger (1989). This discrepancy is also reflected in the optimal hedges: in the classical BSM economy the optimal hedge is provided by the δ-hedge portfolio, where δ only depends on the current price of the underlying and the remaining time to maturity. DQN anticipates the expected reward structure, a feature that fits naturally with the optimization of terminal utility, see Section 2.1, but not the BSM model.

As a consequence the role of the parameter γ in DQN (see Section 2.4) is important: apart from being a discount factor, it also regulates, in terms of the Bellman equation, the dependency between the immediate reward resulting from a given action and the future rewards. Experimental results are in line with the expectation that hedging performance become weaker (when benchmarked against BSM) when the DQN agent attempts to anticipate the future reward structure, see Fig. 8. But what happens in the presence of propagating transaction costs? If costs are small, then probably, γ should be comparatively small. To find the optimal value for γ is a challenging task. In reality γ can thus be viewed as a delicate hyperparameter that should be tuned to the real markets. The right choice of risk-measure is not obvious either. Indeed, the preceding articles (Hodges and Neuberger, 1989; Kolm and Ritter, 2019; Qlbs, 2020; Cao et al., 2021; Ritter, 2017) rely on (multiple) different measures of risk. This implies that the same hedging path would be valued differently within the various setups and makes setting up a planning algorithm even more delicate. From a practical perspective, we remark that often *Q*-learning is tedious to train and unstable, especially when used in conjunction with NNs and a sparse reward signal (Tsitsiklis and Van Roy, 1997): We experimented with hedging in the Cash Flow formulation, where the entire reward was granted to DQN at contract maturity. We observed frequent convergence of DQN to a fixed hedging decision independent of the market state (i.e. a poor local optimum), frequent divergence of *Q*-values and an overall unstable training process. In this setting DQN required an architecture that stores snapshots of the algorithm state to allow restarting from the snapshot.

5. Conclusions

We have confirmed, following the preceding publications (Hodges and Neuberger, 1989; Kolm and Ritter, 2019; Qlbs, 2020; Cao et al., 2021), that RL agents can be trained to hedge derivative contracts. We have introduced the R-CMAB algorithm for P&L hedging, which is motivated by its relative simplicity and sample efficiency. During training, RL agents learn to operate within their specific training environment. RL agents trained on simulated data perform well in market situations that are similar to the training simulations. This reveals an important shortcoming of RL methods when applied to hedging in practice: the lack of real-world data leaves the practitioner with little options but to train on simulated data. For full-fledged RL algorithms like DQN, the amount of simulated training data is huge as compared to the real-world data that the agent encounters in operation. The statistical influence of the real-world data will therefore often be negligible, leaving not much but an overfitting to the simulation models. We have demonstrated that the R-CMAB algorithm addresses this issue by a much more efficient and stable learning process.

During execution, hedging agents should adapt to potential changes in the market. We have seen that the data cost of *Q*-learning will often render model updates unrealistic and the proposed CMAB algorithm reacts to new market conditions faster. In similar vein, although *Q*-learning is well-suited for multi-period hedging problems and for the computation of reservation and minimum risk prices, obtaining sufficient real-world data remains elusive. Given the constraints of investment firms the practical benefit of the theoretically more accurate prices is not obvious.

Finally, more modern algorithms, such as Supervised Learning with recurrent neural networks or neural MCTS, are available to address the multi-period problems more efficiently. The presence of hyperparameters, such as γ in DQN, which regulates the interdependency of rewards and the level of risk-aversion, which tunes between different measures of risk, makes the evaluation of hedging performance subtle. Indeed, hedges obtained from R-CMAB naturally converge to BSM deltas when no risk adjustment, discretization error and transaction costs are present. This makes R-CMAB a useful benchmark for performance comparison and the development of more sophisticated algorithms. Direct minimization of terminal risk using Supervised Learning addresses the important use case of hedging complicated structures that cannot be priced with standard tools, but is much simpler than DQN. While our findings are no surprise from the perspective of AI research, the question of training-data availability has not been discussed in detail in the hedging literature. To the contrary, we have observed a tendency towards more complex architectures in the recent publications, but data-availability might dictate a simple model. It is not uncommon that RL is used in Finance in situations, where large data sets are available for training. A typical example, where RL algorithms are used successfully is market-making, where trading occurs on a submicrosecond time scale and produces an abundance of real-world data. The application of RL algorithms in a settings that provides relatively few data points, such as options hedging, remains an area of active research. As a future line of research and complementary to this paper we indend to study the performance of contextual bandit algorithms on real-world data.

Financial disclosure

This work is supported by UBS project LP-15403/CW-202427. The views and opinions expressed in this material are those of the respective speakers and are not those of UBS AG, its subsidiaries or affiliate companies (“UBS”). Accordingly, neither UBS nor any of its directors, officers, employees or agents accepts any liability for any loss or damage arising out of the use of all or any part of this material or reliance upon any information contained herein.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Anthony, T., Tian, Z., Barber, D., 2017. Thinking fast and slow with deep learning and tree search. Adv. Neural Inf. Process. Syst. 30.

Auer, P., Cesa-Bianchi, N., Fischer, P., 2002. Finite-time analysis of the multiarmed bandit problem. Mach. Learn. 47 (2–3), 235–256.

Black, F., Scholes, M., 1973. The pricing of options and corporate liabilities. J. Polit. Econ. 81 (3), 637–654.

Botvinick, M., Ritter, S., Wang, J.X., Kurth-Nelson, Z., Blundell, C., Hassabis, D., 2019. Reinforcement learning, fast and slow. Trends Cognit. Sci. 23 (5), 408–422.

Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S., 2012. A survey of Monte Carlo tree search methods. IEEE Trans. Comput. Intell. AI Game. 4 (1), 1–43.

Buehler, H., Gonon, L., Teichmann, J., Wood, B., 2019. Deep hedging. Quant. Finance 19 (8), 1271–1291.

Buehler, H., Gonon, L., Teichmann, J., Wood, B., Mohan, B., Kochems, J., 2019. Deep Hedging: Hedging Derivatives under Generic Market Frictions Using Reinforcement Learning. Swiss Finance Institute.

Cao, J., Chen, J., Farghadani, S., Hull, J., Poulos, Z., Wang, Z., Yuan, J., 2023. Gamma and vega hedging using deep distributional reinforcement learning. Frontiers in Art. Int. 6.

Cao, J., Chen, J., Hull, J., Poulos, Z., 2021. Deep hedging of derivatives using reinforcement learning. J. Financ. Data Sci. 3 (1), 10–27.

Chapelle, O., Li, L., 2011. An empirical evaluation of thompson sampling. Adv. Neural Inf. Process. Syst. 24, 2249–2257.

Guo, X., Singh, S., Lee, H., Lewis, R.L., Wang, X., 2014. Deep learning for real-time atari game play using offline monte-carlo tree search planning. Adv. Neural Inf. Process. Syst. 3338–3346.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D., 2018. Rainbow: Combining Improvements in Deep Reinforcement Learning. Thirty-Second AAAI Conf. on Art. Int.

Hodges, S., Neuberger, A., 1989. Option Replication of Contingent Claims under Transactions Costs. Technical Report, Financial Options Research Centre,University of Warwick.

Kolm, P.N., Ritter, G., 2019. Dynamic replication and hedging: a reinforcement learning approach. J. Financ. Data Sci. 1 (1), 159–171.

Lin, L.J., 1992. Reinforcement Learning for Robots Using Neural Networks. Carnegie Mellon University.

Melnikov, A.A., Makmal, A., Briegel, H.J., 2018. Benchmarking projective simulation in navigation problems. IEEE Access 6, 64639–64648.

Merton, R.C., 1973. Theory of rational option pricing. Bell J. Econ. Manag. Sci. 141–183.

Mikkilä, O., Kanniainen, J., 2023. Empirical deep hedging. Quant. Finance 23 (1), 111–122.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Murray, P., Wood, B., Buehler, H., Wiese, M., Pakkanen, M., 2022. Deep hedging: continuous reinforcement learning for hedging of general portfolios across multiple risk aversions. In: Proc. of the Third ACM Int. Conf. on AI in Finance, pp. 361–368.

Osband, I., Blundell, C., Pritzel, A., Van Roy, B., 2016. Deep exploration via bootstrapped dqn. Adv. Neural Inf. Process. Syst. 4026–4034.

Puterman, M.L., 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc.

Qlbs, I. Halperin, 2020. Q-learner in the black-scholes (-merton) worlds. J. Deriv. 28 (1), 99–122.

Riquelme, C., Tucker, G., Snoek, J., 2018. Deep bayesian bandits showdown: an empirical comparison of bayesian deep networks for thompson sampling. arXiv preprint arXiv:1802.09127.

Ritter, G., 2017. Machine Learning for Trading. Available at: *SSRN 3015609*.

Sani, A., Lazaric, A., Munos, R., 2012. Risk-aversion in multi-armed bandits. Adv. Neural Inf. Process. Syst. 3275–3283.

Sattar, V., Qing, Z., 2016. Risk-averse multi-armed bandit problems under mean-variance measure. IEEE J. Sel. Top. Signal Process. 10 (6), 1093–1111.

Schweizer, M., 1995. Variance-optimal hedging in discrete time. Math. Oper. Res. 20 (1), 1–32.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., 2016. Mastering the game of go with deep neural networks and tree search. Nature 529 (7587), 484.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., 2017. Mastering the game of go without human knowledge. Nature 550 (7676), 354–359.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M.A., Prabhat, Adams, R., 2015. Scalable Bayesian Optimization Using Deep Neural Networks. International Conference on Machine Learning, pp. 2171–2180.

Sutton, R.S., 1988. Learning to predict by the methods of temporal differences. Mach. Learn. 3 (1), 9–44.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: an Introduction. MIT Press.

Szehr, O., 2021. Hedging of financial derivative contracts via Monte Carlo tree search. arXiv preprint arXiv:2102.06274.

Thompson, W.R., 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika 25 (3/4), 285–294.

Tsitsiklis, J.N., Van Roy, B., 1997. Analysis of temporal-difference learning with function approximation. Adv. Neural Inf. Process. Syst. 1075–1081.

Van Hasselt, H., Guez, A., Silver, D., 2016. Deep Reinforcement Learning with Double Q-Learning. Thirtieth AAAI Conference On Artificial Intelligence.

Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al., 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature 575 (7782), 350–354.

Vittori, E., Likmeta, A., Restelli, M., 2021. Monte Carlo tree search for trading and hedging. In: Proc. of the Second ACM Int. Conf. on AI in Finance, pp. 1–9.

Watkins, C.J.C.H., Dayan, P., 1992. Q-learn. Mach. learn. 8 (3–4), 279–292.