Les Cahiers du GERAD ISSN: 0711–2440

Deep reinforcement learning for optimal stopping with application in financial engineering

A. Fathan, E. Delage

G–2021–30

May 2021

Citation suggérée : A. Fathan, E. Delage (Mai 2021). Deep reinforcement learning for optimal stopping with application in financial engineering, Rapport technique, Les Cahiers du GERAD G–2021–30, GERAD, HEC Montréal, Canada.

Avant de citer ce rapport technique, veuillez visiter notre site Web (https://www.gerad.ca/fr/papers/G-2021-30) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

Suggested citation: A. Fathan, E. Delage (May 2021). Deep reinforcement learning for optimal stopping with application in financial engineering, Technical report, Les Cahiers du GERAD G–2021– 30, GERAD, HEC Montréal, Canada.

Before citing this technical report, please visit our website (https://www.gerad.ca/en/papers/G-2021-30) to update your reference data, if it has been published in a scientific journal.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2021
– Bibliothèque et Archives Canada, 2021

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2021
– Library and Archives Canada, 2021

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Deep reinforcement learning for optimal stopping with application in financial engineering

Abderrahim Fathan
Erick Delage

GERAD & Department of Decision Sciences, HEC Montréal (Québec), Canada, H3T 2A7

abderrahim.fathan@gmail.com
erick.delage@hec.ca

May 2021
Les Cahiers du GERAD
G–2021–30

Abstract : Optimal stopping is the problem of deciding the right time at which to take a particular action in a stochastic system, in order to maximize an expected reward. It has many applications in areas such as finance, healthcare, and statistics. In this paper, we employ deep Reinforcement Learning (RL) to learn optimal stopping policies in two financial engineering applications: namely option pricing, and optimal option exercise. We present for the first time a comprehensive empirical evaluation of the quality of optimal stopping policies identified by three state of the art deep RL algorithms: double deep Q-learning (DDQN), categorical distributional RL (C51), and Implicit Quantile Networks (IQN). In the case of option pricing, our findings indicate that in a theoretical Black-Schole environment, IQN successfully identifies nearly optimal prices. On the other hand, it is slightly outperformed by C51 when confronted to real stock data movements in a put option exercise problem that involves assets from the S&P500 index. More importantly, the C51 algorithm is able to identify an optimal stopping policy that achieves 8% more out-of-sample returns than the best of four natural benchmark policies. We conclude with a discussion of our findings which should pave the way for relevant future research.

Keywords: Optimal stopping, reinforcement learning, deep learning, financial engineering

The authors would like to acknowledge the generous computational support from Google Cloud, Computer Research Institute of Montréal, and Compute Canada. Furthermore, they are thankful for the financial support from the Canadian Natural Sciences and Engineering Research Council [Grant RGPIN–2016–05208] and the Canada Research Chair program [950–230057].

1 Introduction

We consider the problem of Optimal Stopping (OS) in a stochastic system, which can be described as follows: the system evolves from one state to another in discrete time steps up to some fixed horizon T. At each time step the decision maker has the option to stop the process or wait for a later step to do so. If he decides to stop, then he gets a reward that depends on the current state of the system. Otherwise, the decision maker does not receive any reward immediately, but can decide to stop at a future time step.

In spite of its simplicity, the optimal stopping model is of use in many fields of application including asset selling, gambling, and sequential hypothesis testing. Recently, in [1], it is used to determine when to stop the treatment of patients receiving fractioned radiotherapy treatments. [20] proposes an OS framework to perform feature selection in the classification of urban issue requests on civic engagement platforms. [10] combines Bayesian optimization and OS in the design of early-stopping strategies for the training of neural networks. Its most popular application is however in financial engineering. For example, a Bermudan option is a financial derivative product with a predetermined maturity deadline that will pay out at exercise time, chosen among a discrete set of time points, an amount that depends on the value of an underlying financial asset. Based on no-arbitrage theory, these options are usually priced according to the expected return achieved by an optimal exercise (i.e. stopping) strategy under an assumed martingale stochastic process, such as a Geometric Brownian Motion (GBM). Alternatively, the buyer of such an option will often seek to exercise it at the most profitable moment without exact knowledge of the stochastic dynamics of the underlying asset.

In stochastic control, one way to solve the optimal stopping problem is by using Approximate Dynamic Programming (ADP) [4] to approximate the value function that specifies the best expected reward one can receive starting from a given state. Once this is achieved, a greedy policy based on the approximate value function is expected to provide good decisions. In this paper, we apply some of the latest advancements in reinforcement learning (RL), which had great success in controlling several Atari 2600 games [3, 9, 22, 23], to address the optimal stopping problem. The original algorithms have been modified to adapt to time series data and a Long Short Term Memory (LSTM) [16] recurrent neural network is implemented to model long sequences and integrate history. Following the work by [15], we also combine three additional techniques in each of our three customized RL approaches, the first of which is Double Q-learning, first introduced in [26] to address the problem of over-estimation of action values, and has been subject to improvements in [14] to attain better performance. The second is the dueling architecture [27] that uses two separate estimators: one for the state value function and one for the state-dependent action advantage function, which leads to better policy evaluation in the presence of many similar-valued actions. The third is multi-step bootstrapping of targets [11] which helps accelerate the propagation of newly observed rewards to earlier visited states and balances the bias-variance trade-off.

In this work, we additionally perform the first comprehensive empirical study of the use of recent deep reinforcement learning algorithms to solve the problem of optimal stopping with application in option pricing and optimal exercising. We show, for the first time, that with well-designed modifications to the original algorithms, deep RL architectures such as Double Deep Q-Network (DDQN) [23], Categorical Distributional RL (C51) [3] and Implicit Quantile Networks (IQN) [9] are able to identify policies that achieve near optimal performance in terms of pricing and to outperform predictive financial models, such as the binomial tree model, in an option exercising problem with real US stock market data. Furthermore, our experiments demonstrate that: (1) models based on deep reinforcement learning have a high ability to learn and adapt to stochastic environments with high volatility and randomness; (2) C51 and IQN algorithms outperform DDQN in terms of performance, at a cost of more computation time; (3) C51 slightly outperforms IQN when confronted to real stock data movements, identifying an option exercise policy that achieves 8% more out-of-sample returns than the best of four natural benchmark policies.

The rest of this paper is organized as follows: In Section 2, we discuss related work to our problem, then define the problem in Section 3. Section 4 describes the three RL architectures that will be evaluated. Section 5 then presents two experiments involving financial engineering applications that compare the performance of RL to natural benchmarks. Finally, in Section 6 we conclude with a discussion of our findings which should pave the way for relevant future research. Interested readers can find all our data, implementations, and experiments at https://github.com/osrlpaper/os_rl_papercode.

2 Related work

Among the state-of-the-art ADP approaches that solve the optimal stopping problem, one finds the simulation-regression approach of (see [6, 21, 24]) which uses regression to approximate the optimal continuation value at each state of the system, and the martingale duality-based approach of [25]. The latter relaxes the non-anticipativity requirement of the policy by allowing it to use future information, but on the other hand penalizes any policy that uses such information. Several other approaches have been derived from these two including [5], and [13]. However, such numerical methods either suffer from the well-known curse of dimensionality, assume that the underlying stochastic model is known (or in the very least the states that makes it Markovian), or require the fine tuning of basis functions.

On the RL side, [2] proposes an approach in which multilayer feed-forward neural networks are used. [19] also applies the least-squares policy iteration RL method to the problem of learning exercise policies for American options and shows their good quality. [8] on the other hand addresses the OS problem by constructing interpretable optimal stopping policies from data using binary trees. [12] proposes a model-free policy search method that reuses data for sample efficiency by leveraging problem structure to simultaneously learn and plan. On the other hand, [28] introduces alternative algorithms to Q-learning for OS, which are based on projected value iteration, linear function approximation, and least squares. [2] propose a value-based reinforcement learning for OS learning from Monte Carlo samples, with application to derivative pricing. It is the closest paper to our work, and the reader is referred to it for more details of the problem setup.[17] consider the problem of ranking response surfaces as image segmentation, using feed-forward neural networks to approximate the value function. Reformulating the optimal stopping problem as a surface ranking problem, they apply this scheme to pricing Bermudan options. [7] propose a Q-learning based algorithm for OS with an application to derivative pricing. In their paper, they prove convergence of the algorithm using ODE analysis, and also observe that it achieves optimal asymptotic variance.

To the best of our knowledge, our paper is the first to apply and compare the performance of Deep DQN [22, 23], Categorical Distributional RL [3], and Implicit Quantile Networks [9] on optimal stopping problems.

3 Problem definition

In this work we adopt the following notation:

- β : the discount factor $\in$ [0,1]
- $\mathcal{A}_t$: the set of possible actions at time t
- $\Omega \subseteq \mathcal{S}^{T+1}$: the set of possible trajectories
- T : the horizon of the problem.
- $\mathcal{S} \subseteq \mathbb{R}^L$: the set of possible states.
- Π : the set of possible policies $\pi : \mathcal{S} \to \mathcal{A}$.

In particular, we let $\mathcal{A}_t := \{\text{continue}, \text{stop}\}$ for $t < T$, and $\mathcal{A}_T := \{\text{stop}\}$, and always assume that the time (and remaining time $T - t$) can be inferred from a state of s, i.e. $t = \mathfrak{t}(s)$ iff s is observed at time t. The stopping time with policy π is defined as: $\tau_\pi = \min\{t \in [0, \ldots, T] : \pi(s_t) = \text{stop}\}$. Our goal is to find the optimal policy, that maximizes the average discounted return received over all trajectories: $\pi^* = \arg\max_{\pi\in\Pi} \mathbb{E}\left[\beta^{\tau_\pi} g_{\tau_\pi}(\boldsymbol{s}_{\tau_\pi:T})\right]$, where $\boldsymbol{s}_{t:T}$ is shorthand for the sub-trajectory $[s_t, \ldots, s_T]$ and the expectation is taken based on the distribution of $\boldsymbol{s}_{0:T}$. Furthermore, $g_t(\boldsymbol{s}_{t:T})$ refers to the payout received when stopping at time t under any trajectory with a tail trajectory matching $\boldsymbol{s}_{t:T}$.

Alternatively, one can also define

$$\pi^*(s_t) = \begin{cases} \text{stop} & \text{if } \mathbb{E}[\beta^t g_t(s_{t:T})|s_t] \geq \mathbb{E}[\beta^{\tau_\pi^{t+1}} g_{\tau_\pi^{t+1}}(\boldsymbol{s}_{\tau_\pi^{t+1}:T})|s_t] \\ \text{continue} & \text{otherwise} \end{cases}$$

where $\tau_\pi^t = \min\{t' \in [t, \ldots, T] : \pi(s_{t'}) = \text{stop}\}$.

In the case of a Bermudan option pricing or exercising problem, s includes information about the current value of the financial asset, which can be recovered through some $S(s)$, and $g_t(\boldsymbol{s}_{t:T}) := \max(0,\, S(s_t) - K)$ with strike price K for a call option or $g_t(\boldsymbol{s}_{t:T}) := \max(0,\, K - S(s_t))$ for a put option. In other words, a call option will pay the difference between the value of the asset and the strike price if it is positive, and the opposite occurs for the put option. When K is set to be $S(s_0)$, the option is said to be at-the-money.

We also wish to emphasize that our payout function is flexible enough to model any general payout $h(t, \boldsymbol{s}_{0:T})$ as long as the information about $\boldsymbol{s}_{0:t-1}$ (or some "sufficient statistics") is included in s_t, obtaining $g_t(\boldsymbol{s}_{t:T}) := h(t, [\boldsymbol{s}_{0:t-1}(s_t),\ \boldsymbol{s}_{t:T}])$. For example, in an exercise problem involving a call option, one might be instead interested in maximizing $\mathbb{E}[\max(0,\, S(s_{\tau_\pi}) - K)/\sup_t \max(0,\, S(s_t) - K)]$. This can easily be implemented by including in the state $\theta_t =: \max(\theta_{t-1}, \max(0,\, S(s_t) - K))$, and defining $g_t(\boldsymbol{s}_{t:T}) := \max(0,\, S(s_t) - K)/\max(\theta_t, \sup_{t'>t} \max(0,\, S(s_{t'}) - K))$ with $\beta = 1$.

Finally, one can show that the sequential decision making problem described above can be reformulated in standard reinforcement learning notation as:

$$Q(s,a) := \begin{cases} r(s, \text{stop}) & \text{if } a = \text{stop} \\ r(s, \text{continue}) + \beta \mathbb{E}[\max(Q(s', \text{stop}),\, Q(s', \text{continue}))] & \text{otherwise.} \end{cases} \quad (1)$$

where $r(s, \text{continue}) := 0$ while $r(s, \text{stop}) := g_{\text{t}(s)}(\tilde{\boldsymbol{s}}_{\text{t}(s):T})$ for a random trajectory $\tilde{\boldsymbol{s}}_{0:T}$ with $\tilde{\boldsymbol{s}}_{\text{t}(s)} = s$. In what follows, we provide for the first time customized implementations of a number of state-of-the-art deep RL algorithms to this most general form of the OS problem. In particular, the original Double Q-learning [23] only uses plain fully-connected layers that exhibit low performance when applied on strategic games with long time dependencies. In our implementation, we integrated LSTM in order to learn the representation of states, aggregate partial information from the past, and capture long-term dependencies in our sequential data.

It is also worth mentioning that OS problems cannot be exactly cast as a regression of the optimal stopping time, or the classification of X as either "stop" or "continue" given that the trajectories are unlabeled, and that the consequences of continuing are delayed.

4 Architectures

We implemented three deep reinforcement learning algorithms to identify the optimal policy and value of the OS problem: double deep Q-Learning (DDQN), categorical distributional RL (C51), and implicit quantile networks (IQN). In short, DDQN attempts to learn an optimal action-value Q-network (as defined in Equation 1), while C51 and IQN aim at learning the full distribution of the total discounted reward, i.e. the optimal stopping value $\beta^{\tau_{\pi^*}} g_{\tau_{\pi^*}}(\boldsymbol{s}_{\tau_{\pi^*}:T})$, given s. For conciseness, we push the pseudo-code description of DDQN and C51 to Appendix A. We also refer interested readers to [23], [3], and [9] for additional details on the original implementations of these three approaches. Since we are dealing with time series of varying lengths (they end when the agent stops the process), the core of our model uses a dynamic LSTM layer: specifically, we used three layers with cells of size 512. Our architecture integrates two neural networks: a primary network to choose an action given the current state, and a target network that generates the target Q-value for that action. Adam gradient descent [18] was used to optimize our networks using Huber loss as our temporal difference error.

During each episode of training, we first decide whether the episode will employ a random policy or not. If it is random, then a random stopping time is chosen. Otherwise, the policy learned so far is used throughout the episode. The probability of employing a random policy ϵ is annealed from 1 to 0.01 over time, and no random action is taken during validation or test. We observed that this approach improved learning efficiency when compared to ϵ-greedy policies since it avoided unnecessary use of the neural network in trajectories where a random action eventually ends up being taken. Our exploration strategy also quickly provided to the agent a more diversified set of experiences to learn from, with a stopping time that is uniformly distributed over the horizon compared to ϵ-greedy which had a bias towards stopping early. Given the nature of our application, it would be in theory possible to dismiss exploration altogether if the whole trajectory (even passed the stopping time) was used in training. However, we found that learning only from the part of the trajectory that precedes stopping time together with some annealed exploration improved the quality of final policies. We suspect that this is due to the fact that the neural network's predictive power ends up focusing more on relevant states. We also consider that the trajectory past the stopping time might be unobservable in some applications, e.g. the secretary problem.

It is worth mentioning that, unlike in the original DDQN algorithm [23], we take into consideration the sequential nature of data. Hence, when sampling a mini-batch of buffers B, we provide a sequence of T time steps and a maximum of $T*$ batch-size sequence transitions are used in every mini-batch, depending on the stopping time of every episode ($(T-n+1)*$ batch-size in case of n-step bootstrapping). Furthermore, the maximum dueling architecture and multi-step bootstrapping of 7-steps [11] have been integrated and optimised into our version of DDQN, however, they degraded the results for IQN and C51 during tests and hence their use was omitted in all versions.

Finally, we added a dropout wrapper around the LSTM and fully connected layers, with a drop probability of 20%, to reduce the risk of over-fitting during training. Soft-updates ($\tau = 0.001$) of the target network were implemented and tested, where the network is smoothly and gradually updated in contrast to hard updates that assign the whole online network to the primary network at each update. Overall, while hard update appeared more stable and better performing when trained with synthetic data in Section 5.1, soft-updates were the favoured configuration for real data training (in Section 5.2) for both C51 and IQN.

All the code was implemented in tensorflow 1.14 using, among others, CudnnCompatibleLSTMCell on a GPU, which is 3-5x faster than normal LSTM implementations, and is platform-independent. To further accelerate learning, We first anneal ϵ rapidly to allow the algorithm to learn from more meaningful samples, then we decelerate the annealing speed through time. This has proven to be more efficient during our experiments.

5 Empirical results

In this section, we assess the performance of three different RL algorithms on two financial engineering problems. In the first one, RL is used to price a Bermudan put option in a context where the underlying stock dynamics are assumed to be known. This is a case where a unique price exists and can be computed numerically by employing approximation methods such as binomial tree models. We are therefore able to compare the performance of RL to a ground truth which will validate the potential of C51 and IQN at identifying truly optimal policies. The second setting involves an optimal exercise problem where the underlying stock's dynamics are unknown and based purely on historical data. We will show that in this real world setting, state of the art methods like C51 can learn policies that significantly outperform traditional benchmarks out-of-sample.

In both applications, the state $s \in \mathcal{S}$ will be defined as a sequence of $L = 15$ scalar values (history of prices), concatenated with the amount of remaining time $(T - t)$ to maturity of the option and the relative position of the stock value compared to strike price $\beta^t \max(K - S(s_t), 0) - \max(S(s_t) - K, 0))$, a feature that either returns the discounted reward that will be received (if strictly positive), or otherwise returns how far the stock is from the strike price. This makes the real size of states fed to neural networks $L + 2$. In order to warm start the LSTM, each episode is started 12 days earlier while the policy is only implemented from day 1. Finally, we limited the number of epochs of training to 5 to avoid overfitting and also to limit computation time. In the special case of C51, in order to have comparable training times, C51 was trained on a subset of only 48 trajectories (instead of 160) in Section 5.1, while in Section 5.2 it was trained for only three epochs.

Our experiments will systematically involve three steps of execution. First, we calibrate the hyper-parameters of each algorithm using a training (Training) and validation set (Valid_HP). Once the optimal setting is found, we employ a second validation set (Valid_Model) in order to assess in an unbiased way which algorithm is best performing and finally test this best performing algorithm on a reserved set of test data. This process allows us to make claims about the statistical significance of our results. Performance of the RL policies will be compared to three natural benchmarks: "Rand" which chooses uniformly at random the exercise among the T alternatives, "First" and "Last" which exercise respectively on the first $\tau = 0$ and last $\tau = T$ days, and a binomial tree model (B.M.) that is either calibrated on the true stock dynamics (in Section 5.1) or on the available set of L historical prices (in Section 5.2).

5.1 Bermudan option pricing under Black-Schole setting

Our first experiment consists of a classical Black-Schole option pricing problem. When a financial asset is assumed to behave according to a Geometric Brownian Motion (GBM) model, it is well known that, in order to avoid giving rise to arbitrage opportunities, a financial derivative of this asset needs to be priced according to the optimal expected revenue that can be obtained under the GBM's so called risk neutral martingale measure. Here, we focus on the case of pricing an at-the-money Bermudan put option with daily exercise opportunities (often used as a proxy for pricing American options), where the daily discretized risk neutral measure takes the form of $S_t = S_{t-1}e^{(r-\frac{\sigma^2}{2})\Delta t+\sigma\sqrt{\Delta t}\varepsilon}$, with r as the risk-free continuously compounded yearly interest rate, σ as the volatility of the asset, ε as the standard normal distribution, and Δt as the amount of time elapsed between $t-1$ and t. Specifically, in our experiments, we let $S_0 = 1$, $\sigma = 20\%$, $\Delta t = 1/252$, $r = 5\%$, and the horizon $T = 38$. Similarly as in [19], RL will consider a discount factor of $\beta = e^{-r\Delta t}$ which effectively prices an option that pays $e^{-rt\Delta t}\max(0, S_0 - S_t)$ at exercise time. Hence, in this experiment, we train the three RL algorithms on simulated trajectories in order to use the expected reward from the best trained model as an estimation of the arbitrage-free option price (AFOP). While such a price can be obtained with high precision much more efficiently using binomial tree models (B.M.), our aim is to verify whether modern RL algorithms are mature and flexible enough to reach optimality and retrieve such a price.

In this experiment, the Training set is composed of 160 sampled trajectories of 928 days each, from which are drawn 135600 episodes used in training. The Valid_HP set (for hyper-parametrization) consists of 40 independently and identically drawn trajectories of 928 days (24000 episodes), while the Valid_Model (for algorithm selection) and Test set consist of 200 and 400 i.i.d. trajectories over 928 days respectively (159600 and 319200 episodes). We refer the reader to Appendix B, which descibes the final choice of hyper-parameter values.

Table 1 presents the results for this experiment. Looking at the numbers, we observe that both C51 and IQN achieve high Expected Reward (ER) in training and both steps of validation. While C51 appeared to be the best performing approach on Valid_HP , we suspect that the selection of hyperparameters overfitted the Valid_HP set given that 1) it outperformed the theoretically optimal policy generated by the binomial tree model; 2) the performance degraded when validating on Valid_Model set. Given its better performance in Valid_Model , IQN was selected for the final out-of-sample test where it estimates a AFOP of 0.0284 ± 0.0001 compared to a ground truth of 0.0283 ± 0.0001. This confirms that the resulting IQN exercise policy is statistically equivalent to the theoretical policy.

Overall, we can conclude that, despite the context of high stochasticity of GBMs, RL models such as IQN are flexible enough to learn optimal exercise policies. This shows the high potential of RL algorithms to replace conventional approaches in situations where the dynamics of the risk neutral martingale require a large state space in order to become Markovian, and should be easier to adapt to situations were the market is incomplete or stock dynamics are unknown. On the other hand, one needs to be aware of the heavy computational burden imposed by current state-of-the-art RL algorithms. Beyond requiring substantial training time due to their model-free nature, the selection of best performing hyper-parameters is still more of an art than a science. In particular, we observed that regions of best performing hyperparameter values were sensitive to factors such as the number of trajectories and epochs that were used.

Table 1: Performance of the 3 versions of RL vs Baselines for GBM data

Data \ Method		DDQN	C51	IQN	Rand	Last	First	B.M.
Training	ER	0.0263	0.0267	**0.0268**	0.0229	0.0263	0.0160	0.0267
	Time (sec)	**0.4666**	1.3029	0.6668				
Valid_HP	ER	0.0279	**0.0280**	0.0275	0.0234	0.0279	0.0167	0.0276
Valid_Model	ER	0.0270	0.0273	**0.0275**	0.0236	0.0271	0.0163	0.0275
	CI	0.0002	0.0002	0.0001	0.0001	0.0002	0.0001	0.0001
Test	ER/AFOP	0.0282	0.0283	<u>**0.0284**</u>	0.0243	0.0282	0.0168	0.0283
	CI	0.0001	0.0001	<u>0.0001</u>	0.0001	0.0001	0.0001	0.0001

Best values are marked in **bold**. Training time is per episode (on a Titan X GPU). CI refers to a 90% confidence interval. Out-of-sample performance of RL model selected with Valid_Model is underlined.

5.2 S&P 500 stock data

In this section, we consider the optimal exercise problem of a Bermudan option. In particular, we consider the distribution of T days stock trajectories in which one first draws the stock randomly from 111 stocks that compose the S&P 500 index, and a random date on the period 2014-03-27 to 2019-12-10. The Training set considers trajectories from a subset of 60 different stocks and dates from the period 2014-03-27 to 2016-03-29 (733 trading days), Valid_HP considers the same set of stocks with period 2016-03-29 to 2017-11-10 (183 days). The Valid_Model set considers 51 other stocks over the period 2014-03-27 to 2017-11-10. Finally, the test set is composed of all 111 stocks over a "future" period 2017-11-11 to 2019-12-10 (522 working days). In order for the policies to treat similarly stocks with different starting price, we focus on the task of maximizing the Expected Relative Option Payout (EROP) for an at-the-money put option with horizon $T = 38$: i.e. $g_t(\boldsymbol{s}_{t:T}) := (1/S(s_0))\max(0, S(s_0) - S(X_t))$. Once again, the performance is compared to Rand, First, and Last policies, while B.M. captures the optimal policy for a GBM calibrated on the last L days. The same discount rate of $\beta = e^{-0.05/252}$ was used. Finally, we let the reader refer to Table 3 to find the best hyperparameters found using Valid_HP for each algorithm.

Table 2 shows the performance of the 3 RL algorithms against the four benchmarks. We can see that both C51 and IQN outperform DDQN in the Valid_HP set, this is confirmed in the Valid_Model set which points to C51 as the best model to recommend for out-of-sample tests, although IQN holds a tight second place. The Test set demonstrates that the best IQN outperforms significantly the four benchmarks in terms of Expected Reward (and EROP). Indeed, it achieves on average a 2.91% relative option payout compared to exercising on the last day which achieves 2.17%, and the binomial tree model approach that achieves 2.53%. The table also presents Expected Option Return (EOR) which accounts approximately for the return on investment when implementing each policy assuming that the option is priced based on a GBM risk neutral measure calibrated on the recent history. Specifically, we see that C51 achieves a 22.0% return on average which is 8% higher than any of the competing classical benchmark.

We wish to emphasize that, throughout our extensive set of experiments, including unreported experiments with stocks which dynamics followed a more sophisticated Generalized AutoRegressive Conditionally Heteroscedastic (GARCH) stock model, we observed that IQN has the ability to rapidly fit the training data, although this can in some cases lead to overfitting. Also, during our experiments, we noted that DDQN was 1.2-1.5x faster than IQN and around 2-4x faster than C51 depending on machine configuration, the type of GPU and available memory. Finally, IQN consumes considerably more memory than DDQN. These are all characteristics that are worth taking into account when choosing the right RL approach.

Table 2: Performance of the 3 versions of deep RL vs Baselines for S&P 500 data

Data \ Method		DDQN	C51	IQN	Rand	Last	First	B.M.
Training (60 stocks)	ER/EROP	0.0290	0.0326	**0.0336**	0.0249	0.0281	0.0174	0.0270
Valid_HP (60 stocks)	ER/EROP	**0.0156**	0.0126	0.0147	0.0130	0.0118	0.0082	0.0123
Valid_Model (51 other stocks)	ER/EROP	0.0274	**0.0289**	0.0285	0.0245	0.0272	0.0166	0.0256
	CI	0.0003	0.0004	0.0003	0.0003	0.0004	0.0003	0.0004
Test (all stocks combined on a future period)	ER/EROP	0.0285	**0.0291**	0.0281	0.0265	0.0271	0.0175	0.0258
	CI	0.0003	0.0004	0.0003	0.0004	0.0004	0.0003	0.0004
	EOR	17.1%	**22.0%**	17.6%	8.8%	13.9%	-32.6%	5.3%
	CI	1.2%	1.6%	1.3%	1.4%	1.6%	1.2%	1.5%

Best values are marked in **bold**. CI refers to a 90% confidence interval. Out-of-sample performance of RL model selected with Valid_Model is underlined.

6 Concluding discussion

Solving the problem of optimal stopping in finance where data is known to have a high degree of randomness (unpredictable) is both a notoriously challenging and intriguing task. In this paper, we demonstrated the ability of three variants of deep reinforcement learning algorithms (DDQN, C51, and IQN) to learn simply from real historical stock price observations complex stopping time policies in the presence of uncertainty, volatility, and non-stationarities. Despite being more difficult to employ and requiring a more significant computional investment than traditional off-the-shelf methods, our experiments present empirical evidence that these deep RL algorithms are flexible enough to retrieve optimal policies in context where these can be computed exactly (option pricing under GBM dynamics), and to significantly out-perform off-the-shelf methods when the dynamics of the underlying stochastic system are both unknown and likely to violate simplifying Markovian assumptions. In particular, distributional IQN and C51 are able to learn the value distribution of option returns and rise up as the favoured algorithms to employ in practice, with a strong preference for C51 when computation time is less of an issue.

In closing, it is worth mentioning that our experience of hyper-parameters tuning taught us that it is demanding and fragile, often requiring us to re-align the search grid the moment that problems are slightly modified. We also observed in our experiments with real stock data, that it could be beneficial to avoid shuffling the episodes during training with the effect of improving the out-of-sample performance in periods that are chronologically close to the last episodes that were trained on. This idea could potentially be useful in online learning, when the underlying process is non-stationary, since it implicitly fine-tunes the algorithm according to the most recent data. We believe these constitute two important directions of future investigation.

A Pseudo-code for DDQN and C51 algorithms

A.1 Customized double deep Q-Learning algorithm

Algorithm 1 Customized double deep Q-Learning algorithm, a.k.a. DDQN

1: **Inputs:** A set of M episodes $\{\boldsymbol{s}^i_{0:T}\}_{i=1}^M$
2: **Initialize:** replay memory D to capacity C of episodes
3: **Initialize:** action-value function Q with random weights θ
4: **Initialize:** target action-value function $\widehat{Q}$ with weights $\phi = \theta$
5: **for** episode $i = 1$ **to** M **do**
6:   **Initialize:** episode buffer B to capacity T
7:   With probability ϵ episode is random and a uniformly random day to stop t_{stop} is selected
8:   **for** $t = 0$ **to** T **do**
9:     **if** episode is random **then**
10:       Set $a_t = \begin{cases} \text{continue} & \text{if } t < t_{stop} \\ \text{stop} & \text{otherwise.} \end{cases}$
11:     **else**
12:       Set $a_t = \arg\max_a Q(s^i_t, a; \theta)$
13:     **end if**
14:     Execute action a_t and observe reward r_t, store transition $(s^i_t, a_t, r_t, s^i_{t+1})$ in B
15:     **if** ($a_t =$ stop or $t = t_{stop}$) **then**
16:       Exit **for** loop
17:     **end if**
18:   **end for**
19:   Store episode buffer B in replay memory D. If D full, drop the oldest episode
20:   Sample a random mini-batch of buffers B of sequence transitions (s_j, a_j, r_j, s_{j+1}) from D
21:   Set $y_j = \begin{cases} r_j & \text{if } a_j = \text{stop} \\ r_j + \gamma \max_{a'} \widehat{Q}(s_{j+1}, a'; \phi) & \text{otherwise.} \end{cases}$
22:   Perform gradient descent on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to network parameters θ
23:   Every U episodes reset target network $\widehat{Q} = Q$
24: **end for**

A.2 Customized categorical distributional RL algorithm

Algorithm 2 Customized categorical distributional RL algorithm, a.k.a. C51 when $N = 51$

Inputs: A set of M episodes $\{\boldsymbol{s}^i_{0:T}\}_{i=1}^M$, V_{max} and V_{min} are the maximum and minimum values of possible returns, N is the number of atom probabilities
Initialize: replay memory D to capacity C of episodes
Initialize: discrete support $z_k = V_{min} + k\Delta z$ for $k = 0$ to $N - 1$, with $\Delta z = \frac{V_{max} - V_{min}}{N-1}$
Initialize: value distribution P with random weights θ, and target distribution $\widehat{P}$'s with weights $\phi = \theta$
for episode $i = 1$ **to** M **do**
  Perform lines 6–19 from Algorithm 1 where line 12 uses $Q(s^i_t, a) = \sum_k z_k P_k(s^i_t, a; \theta)$
  Sample a random mini-batch of buffers B of sequence transitions (s_j, a_j, r_j, s_{j+1}) from D
  Set $m_k = 0$, for all $k \in 1, \ldots, N - 1$
  for $k = 1$ **to** $N - 1$ **do**
    Set $v_k = \max(V_{min}, \min(V_{max}, r_j + \gamma z_k))$ #Project distributional Bellman update onto the support $\{z_k\}_{k=0}^{N-1}$
    Set $b_k = (v_k - V_{min})/\Delta z$ # Identify support index of projection
    Set $l = \lfloor b_k \rfloor, u = \lceil b_k \rceil, a' = \arg\max_a \sum_k z_k P_k(s_{j+1}, a; \phi)$
    Set $m_l = m_l + \widehat{P}_k(s_{j+1}, a'; \phi)(u - b_k)$, set $m_u = m_u + \widehat{P}_k(s_{j+1}, a'; \phi)(b_k - l)$ #Distribute probability of sample
  end for
  Perform a gradient step on cross-entropy loss $-\sum_k m_k \log P_k(s_j, a_j; \theta)$ with respect to θ
  Every U episodes update target distribution $\widehat{P} = P$, i.e. $\phi = \theta$
end for

A.3 Implicit quantile networks distributional RL algorithm

Algorithm 3 Customized implicit quantile networks RL Algorithm, a.k.a. IQN

Inputs: A set of M episodes $\{\boldsymbol{s}^i_{0:T}\}^M_{i=1}$. N, N' are the number of samples of $\tau, \tau' \sim U([0,1])$ respectively, with $N = N' = 8$. $K = 32$ is the number of samples $\tilde{\tau} \sim U([0,1])$.
Initialize: replay memory D to capacity C of episodes
Initialize: β is a distorsion risk measure, and $\kappa = 1$ is the threshold for the Huber quantile regression loss
Initialize: state-action quantile function Z with random weights θ, and target quantile function $\widehat{Z}$'s with weights $\phi = \theta$
for episode $i = 1$ **to** M **do**
    Perform lines 6–19 from Algorithm 1 where line 12 uses $Q(s^i_t, a) = \sum_{k=1}^{K} Z_{\beta(\tilde{\tau}_k)}(s^i_t, a; \theta)$, $\tilde{\tau} \sim U([0,1])$
    Sample a random mini-batch of buffers B of sequence transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set $a' = \arg\max_a Q(s^i_j, a; \theta)$
    \# Sample quantile thresholds
    $\tau_l, \tau'_u \sim U([0,1]), \qquad 1 \le l \le N, 1 \le u \le N'$
    \# Compute distributional temporal differences
    $\delta_j^{\tau_l, \tau'_u} = r_j + \gamma Z_{\tau'_u}(s^i_{j+1}, a'; \phi) - Z_{\tau_l}(s^i_j, a_j; \theta)$, $\forall l, u$
    Perform a gradient step on Huber quantile loss $\frac{1}{N'} \sum_{l=1}^{N} \sum_{u=1}^{N'} \rho^{\kappa}_{\tau_l}(\delta_j^{\tau_l, \tau'_u})$ with respect to θ, where

$$\rho^{\kappa}_{\tau_l}(\delta_j^{\tau_l, \tau'_u}) = \left|\tau - \mathbb{1}\{\delta_j^{\tau_l, \tau'_u} < 0\}\right| \frac{\mathcal{L}_\kappa(\delta_j^{\tau_l, \tau'_u})}{\kappa} \text{ with } \mathcal{L}_\kappa(\delta_j^{\tau_l, \tau'_u}) = \begin{cases} \frac{1}{2} {\delta_j^{\tau_l, \tau'_u}}^2 & \text{if } \left|\delta_j^{\tau_l, \tau'_u}\right| \le \kappa \\ \kappa(\left|\delta_j^{\tau_l, \tau'_u}\right| - \frac{1}{2}\kappa) & \text{otherwise.} \end{cases}$$

    Every U episodes update target quantile function $\widehat{Z} = Z$, i.e. $\phi = \theta$
end for

B Final choice of hyper-parameters

Table 3: Hyperparameters of the different RL versions

Task	Algorithm	Hyperparameters
GBM	DDQN	learning-rate=0.0001 + batch-size=128 + C=10000 + U=300
	C51	learning-rate=0.0025 + batch-size=64 + C=3000 + U=30
	IQN	learning-rate=0.00005 + batch-size=128 + C=3000 + U=1000
S&P500	DDQN	learning-rate=0.005 + batch-size=64 + C=10000 + U=300
	C51	learning-rate=0.0025 + batch-size=64 + C=3000 + U=30
	IQN	learning-rate=0.0025 + batch-size=64 + C=3000 + U=100

References

[1] Ali Ajdari, Maximilian Niyazi, Nils Henrik Nicolay, Christian Thieke, Robert Jeraj, and Thomas Bortfeld. 2019. Towards optimal stopping in radiation therapy. Radiotherapy and Oncology 134 (2019), 96–100.

[2] Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. 2019. Deep Optimal Stopping. Journal of Machine Learning Research 20, 74 (2019), 1–25.

[3] Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 449–458.

[4] Dimitri P Bertsekas and John N Tsitsiklis. 1996. Neuro-dynamic programming. Vol. 5. Athena Scientific Belmont, MA.

[5] David B. Brown, James E. Smith, and Peng Sun. 2010. Information Relaxations and Duality in Stochastic Dynamic Programs. Operations Research 58, 4-part-1 (2010), 785–801.

[6] Jacques F. Carriére. 1996. Valuation of the Early-Exercise Price for Derivative Securities using Simulations and Splines.

[7] Shuhang Chen, Adithya M Devraj, Ana Bušić, and Sean P Meyn. 2019. Zap Q-Learning for Optimal Stopping Time Problems. arXiv preprint arXiv:1904.11538 (2019).

[8] Dragos Florin Ciocan and Velibor V. Misic. 2018. Interpretable Optimal Stopping. Computing Research Repository (CoRR) abs/1812.07211 (2018).

[9] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. 2018. Implicit quantile networks for distributional reinforcement learning. arXiv preprint arXiv:1806.06923 (2018).

[10] Zhongxiang Dai, Haibin Yu, Bryan Kian Hsiang Low, and Patrick Jaillet. 2019. Bayesian Optimization Meets Bayesian Optimal Stopping. In International Conference on Machine Learning. 1496–1506.

[11] Kristopher De Asis, J Fernando Hernandez-Garcia, G Zacharias Holland, and Richard S Sutton. 2018. Multi-step reinforcement learning: A unifying algorithm. In Thirty-Second AAAI Conference on Artificial Intelligence.

[12] Karan Goel, Christoph Dann, and Emma Brunskill. 2017. Sample efficient policy search for optimal stopping domains. arXiv preprint arXiv:1702.06238 (2017).

[13] David A. Goldberg and Yilun Chen. 2018. Beating the curse of dimensionality in options pricing and optimal stopping. arXiv preprint arXiv:1807.02227 (2018).

[14] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (Phoenix, Arizona) (AAAI'16). AAAI Press, 2094–2100.

[15] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In Thirty-Second AAAI Conference on Artificial Intelligence.

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation 9, 8 (1997), 1735–1780.

[17] Ruimeng Hu. 2019. Deep Learning for Ranking Response Surfaces with Applications to Optimal Stopping Problems. arXiv preprint arXiv:1901.03478 (2019).

[18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).

[19] Yuxi Li, Csaba Szepesvari, and Dale Schuurmans. 2009. Learning exercise policies for American options. In Artificial Intelligence and Statistics. 352–359.

[20] Yasitha Warahena Liyanage, Daphney-Stavroula Zois, Charalampos Chelmis, and Mengfan Yao. 2019. Automating the Classification of Urban Issue Reports: an Optimal Stopping Approach. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 3137–3141.

[21] Francis A. Longstaff and Eduardo S. Schwartz. 2001. Valuing American options by simulation: A simple least-squares approach. Review of Financial Studies (2001), 113–147.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. Computing Research Repository (CoRR) abs/1312.5602 (2013). arXiv:1312.5602

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. Nature 518, 7540 (Feb. 2015), 529–533.

[24] John N. Tsitsiklis and Benjamin Van Roy. 2001. Regression methods for pricing complex American-style options. IEEE Transactions on Neural Networks 12, 4 (2001), 694–703.

[25] Vijay V. Desai, Vivek F. Farias, and Ciamac C. Moallemi. 2012. Pathwise Optimization for Optimal Stopping Problems. Management Science 58 (12 2012).

[26] Hado van Hasselt. 2010. Double Q-learning. In Advances in Neural Information Processing Systems 23, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Eds.). 2613–2621.

[27] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling Network Architectures for Deep Reinforcement Learning. In Proceedings of The 33rd International Conference on Machine Learning, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 1995–2003.

[28] Huizhen Yu and Dimitri P Bertsekas. 2007. Q-learning algorithms for optimal stopping based on least squares. In 2007 European Control Conference (ECC). 2368–2375.