ALMA MATER STUDIORUM · UNIVERSITY OF BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MASTER'S DEGREE IN COMPUTER ENGINEERING

MASTER'S DEGREE THESIS
in
OPERATIONS RESEARCH

Autonomous Pricing using Policy-Gradient Reinforcement Learning

Advisors:
Prof. Silvano MARTELLO
Prof. Vincenzo DENICOLÒ
Prof. Sergio PASTORELLO

Candidate:
Kevin Michael FRICK

Graduation session: 20/07/2022
A.Y. 2021/2022

Abstract

Nella letteratura economica e di teoria dei giochi vi è un dibattito aperto sulla possibilità di emergenza di comportamenti anticompetitivi da parte di algoritmi di determinazione automatica dei prezzi di mercato. L'obiettivo di questa tesi è sviluppare un modello di reinforcement learning di tipo actor-critic con entropy regularization per impostare i prezzi in un gioco dinamico di competizione oligopolistica con prezzi continui. Il modello che propongo esibisce in modo coerente comportamenti cooperativi supportati da meccanismi di punizione che scoraggiano la deviazione dall'equilibrio raggiunto a convergenza. Il comportamento di questo modello durante l'apprendimento e a convergenza avvenuta aiuta inoltre a interpretare le azioni compiute da Q-learning tabellare e altri algoritmi di prezzo in condizioni simili. I risultati sono robusti alla variazione del numero di agenti in competizione e al tipo di deviazione dall'equilibrio ottenuto a convergenza, punendo anche deviazioni a prezzi più alti.

Abstract

In the economics, game theory, and competition policy literature, there is an open debate on whether autonomous pricing algorithms are able to exhibit cooperative behavior in reasonable timescales. I develop an entropy-regularized actor-critic deep reinforcement learning model able to price goods in an oligopolistic competition game with continuous prices. The model I propose reliably shows cooperative behavior that is supported by reward-punishment schemes that discourage deviations from the point of convergence. The behavior of this model during learning and at convergence also helps interpret the behavior of tabular Q-learning and other pricing algorithms under similar conditions. Results are robust to variations in the number of agents competing and the type of deviation from the convergence outcome, even punishing deviations to higher prices, even if introducing more firms in the market leads to slower learning.

Preface

I wrote no preface for my bachelor's thesis. It was a project that I developed in a much shorter timespan, with a very different feedback loop, in an entirely different field and during what probably was one of the most stressful periods of my life. I did not feel like it needed, or deserved, any introduction or preface. On the other hand, I started working on my master's thesis just a few months after starting my master's, and I am glad that I took the time to write it. I believe the process of developing this thesis closely matches actual research work, if anything because there's at least two other Git repositories and literature reviews that were very helpful in learning, understanding the problem, and formulating the final idea. They are not included here, but if you are reading this and feel like you should definitely know more about expert algorithms (also known as hedging), feel free to contact me.

Two years have gone by fast, and there are a lot of people to which I owe gratitude for this work. There are probably many more which I am forgetting as I write this preface.

I wish to thank Prof. Denicolò for accepting to supervise a student in computer science, for bearing with my incomplete knowledge of economics, and discussing with me every iteration, problem and result that surfaced during the development of this work.

I also wish to thank Prof. Pastorello for introducing me to reinforcement learning, for taking the time to explain his work to me, for always asking the right question when I discussed my work with him, and for letting me monopolize his workstation for months on end to run my experiments.

This work would not be here if it weren't for Prof. Bigoni, who introduced me to the world of research in economics during the second year of my bachelor's, supervised me in writing my first published paper, and suggested me to speak with Prof. Denicolò for my master's thesis.

Neither would any of this be here without Prof. Martello, who over the two years of my master's degree supervised me in writing this thesis and developing other projects. He also taught one of the most influential classes in my academic career,

and provided me with constant feedback, answers and support every time I worked with him.

Thanks to Massimiliano, for reading every chapter of this thesis probably more times than I had. Thank you for reading my code, for helping me understand it better than I ever could alone (yes, my own code!) and for replying to my infinite list of very dumb questions.

Thanks to Anna, for being there for the last ten years or so, throughout different fields, universities, jobs, ideas, throughout a thousand disagreements, cities, friends and stories. I could always speak with you about any doubt I had, obstacle I encountered, or event that made me happy or sad. You kept encouraging me, and I couldn't be more grateful to you for all this.

Thanks to my mother and father, for caring, for listening to to my complaints and my celebrations. Thank you for the values you taught me, both those I agree with and those I don't, and for never getting tired of debating them and always making sure we both learned something from it. Thank you for always striving to help me seize the best parts of life, and get through the roughest patches.

Thanks to everyone in Collegio Superiore, for making sure I will remember these five years living with them for the rest of my life. In alphabetical order, special thanks go to Cosmin, Federico, Francesca, Martina, Matteo, Nicolas, Paola, Simone, and Stefano. Thank you for taking planes like they were buses to come see me, for the years we shared in our lives and for all future moments we may live together.

Thanks to Violetta. Thank you for always being on my side, for believing in me, for being able to take away every worry or fear I had and for making feel right at home in any place, at any time. For helping me be the best version of me, no matter what.

Contents

Chapter 1

Introduction

Reinforcement learning is a research field in machine learning whose developments led to advancements in the control of complex systems such as robots, appliances, vehicles, and video game AI. One application of reinforcement learning that is heavily debated in both the computer science literature and other fields such as economics is how reinforcement learning algorithms behave in multi-agent environments, be they cooperative or competitive. Specifically, a recent research question in economics asks whether firms employing models able to automatically price goods and services may pose a threat to consumers by learning anticompetitive, cooperative behavior.

Most workhorse models of competition in economics resemble traditional social dilemmas such as the prisoner's dilemma. Firms have an incentive to keep prices high to increase profits, a practice known as collusion. However, demand for lower-priced goods is higher so firms also have the option of undercutting one another. This is what drives competition and what antitrust laws seek to encourage, as lower prices are beneficial to consumers in general. However, antitrust laws generally do not punish high prices per se, rather targeting collusion that results from an explicit agreement between firms' executives. Results in algorithmic game theory and economics prove that, even without communication, autonomous agents are able to reach tacit, robust, and sustained collusive behaviors in many social dilemmas. However, current results apply only to relatively simple social dilemmas that do not closely resemble competitive environments or require a very long time for collusion to emerge, enough that it is possible to argue that it would never surface in real situations once the model is taken out of the lab: the time it takes to observe the effect of a price change on demand is much longer than what it takes to perform computations.

The main contribution of this work is to develop a model that achieves collusive outcomes in a standard model of a competitive market in which firms can choose any price from a continuous bounded interval, and that reaches convergence very quickly compared to previous literature, with still notable room for further improvement.

The rest of this work is organized as follows. Chapter 2 aims at explaining the concepts required to understand the environment at hand and the model I develop. Chapter 3 outlines relevant previous work in the field of reinforcement learning, some of its applications to game theory and multi-agent systems, and the state of current research in the economics literature concerning algorithmic collusion. Chapter 4 describes the model I propose and the economic environment that defines the Markov decision process that agents are learning to solve. Chapter 5 details how agents behave and how their behavior changes during learning. It also dissects the strategies they employ to maintain cooperative behavior. Finally, chapter 6 sums up my contribution and provides directions for future work.

Chapter 2

Background

Most of this chapter is based on well-known textbooks in game theory and reinforcement learning, such as Tadelis (2013), Bertsekas (2019) and Sutton and Barto (2018). For ease of exposition, I do not cite these textbooks every time I reference them.

I use capital letters for random variables, bold capital letters for matrices, lower case letters for the values of random variables and for scalar functions, and calligraphic letters for sets. I write quantities that are required to be real-valued vectors in bold and in lower case, even if they are random variables. The only exceptions to this notation are the strategy sets S and the history sets H in a stage game. I denote them with a non-calligraphic capital letter to avoid confusion with the notation used later for the state set in a Markov decision process and for the Shannon entropy of a distribution, as well as to remain consistent with the notation in Tadelis (2013).

2.1 Normal-form games

A normal-form game is a formalization of one-shot decision-making that aims at framing a problem of multiple rational agents seeking to obtain the outcome that they prefer the most from a set of possible outcomes. In a normal-form game, *agents* choose *strategies* that are evaluated according to *payoffs*. Payoffs are numerical values the sum of which the agents want to maximize.

A normal-form game is defined by a set of agents, a set of payoff functions Π_i (one for each agent), and a set of pure strategies S_i (one for each agent). A pure strategy is a deterministic plan of action. Pure strategies may be numerical values, e.g. the quantity of a certain good to sell, or elements of some finite set of discrete choices, e.g. the card to play in a card game.

Definition 2.1.1. *The strategy set S_i of a player contains all possible strategies of* *that player. The strategy set S of a normal-form game is the Cartesian product of the pure strategy sets of all agents.*

Definition 2.1.2. *A payoff function $\Pi_i : S \to \mathbb{R}$ maps a combination of the strategies chosen by agents to a payoff value.*

Definition 2.1.3. *The strategy $s_i \in S_i$ is agent i's best response to its opponents' strategies $s_{-i} \in S_{-i}$ if $\Pi_i(s_i, s_{-i}) \geq \Pi_i(s_i', s_{-i}) \; \forall \; s_i' \in S_i$.*

A special case of normal-form games that have an intuitive graphical representation are bimatrix games (Littman and Stone, 2001). A bimatrix game is a normal form game with two agents defined by two matrices $\mathbf{M}_1, \mathbf{M}_2 \in \mathbb{R}^{m \times n}$. The two agents are denoted "row" and "column" agents since their actions correspond to a row index i and a column index j. The row agent receives payoff $\mathbf{M}_{1,ij}$ and the column agent receives payoff $\mathbf{M}_{2,ij}$. Therefore, the two matrices define two strategy sets $S_i = \{1, ..., m\}, S_{-i} = \{1, ..., n\}$ for the row and column agent respectively, and two payoff functions that map indices (strategies) to payoffs (elements of the matrices).

Definition 2.1.4. *A game is of complete information if all agents have knowledge of all players' payoff functions Π_i and strategy sets S_i.*

Note that, for a bimatrix game, complete information amounts to each agent knowing its own payoff matrix as well as that of the other agent.

For the purpose of analyzing agents' behavior in games and formally defining notions of cooperation, betrayal, competition, and so on, a fundamental concept is that of the Nash equilibrium.

Definition 2.1.5. *The pure-strategy profile $s^* \in S$ is a Nash equilibrium of a normal-form game if s_i^* is a best response to s_{-i}^* for all $i \in \{1..n\}$ where n is the number of players, i.e.*

$$\Pi(s_i^*, s_{-i}^*) \geq \Pi(s_i', s_{-i}^*) \; \forall \; s' \in S_i, i \in \{1..n\} \tag{2.1}$$

2.2 Repeated stage games

Normal-form games can only effectively describe a limited set of real-world events, namely those where the outcomes have consequences only in the short run and do not influence future events. A more general model of decision-making is a sequence of normal-form games, known as a multi-stage game.

Definition 2.2.1. *A multi-stage game is a finite or infinite sequence of independent, well-defined normal-form games.*

These normal-form games are played sequentially by the same players; the *total* *payoffs* from the multi-stage games are the sum of the payoffs from all past games. After each stage is completed, both agents observe the outcomes. Outcomes may or may not be common knowledge.

Multi-stage games can potentially be played an infinite number of times. If we assume that agents are maximizing their total payoffs, then the sum of payoffs may eventually be infinite, so the maximization problem would not be well-defined. For this reason, we assume agents are maximizing the *sum of discounted payoffs* instead, with a discount factor $\gamma \in [0, 1)$ that places a higher weight on payoffs received in the current timestep than in future steps. If $\Pi_{i,t}$ is agent i's payoff from the outcome of the stage game played in timestep t, then the total discounted payoff is

$$\sum_{t=0}^{\infty} \gamma^t \Pi_{i,t} \tag{2.2}$$

If $\gamma \in [0, 1)$ and $\Pi_{i,t} \in \mathbb{R}$ is bounded above, this sum is finite.

Repeated stage games are a special case of multi-stage games.

Definition 2.2.2. *A repeated stage game is a multi-stage game where the same normal-form game is being played at every stage.*

In repeated stage games, agents may condition strategies on their own past actions and that of other agents. Therefore, the definition of a strategy in the context of repeated stage games differs from the one for a normal-form game, as strategies may be conditioned on *histories*.

Definition 2.2.3. *A history h_t is an ordered set of all the actions played by all agents up to timestep t.*

The set H_t contains all possible histories $h_t \in H_t$ of length t. The set H contains all possible histories of any length.

Definition 2.2.4. *A pure strategy for agent i in a repeated stage game is a mapping $s_i : H \to S$ that maps histories into strategies of the repeated normal-form game.*

Definition 2.2.5. *A behavioral strategy for agent i is a probability distribution $\sigma_i : H \to \Delta S$ that maps histories into stochastic choices of actions of the repeated normal-form game.*

2.3 Markov decision processes

A Markov decision process is a formalization of sequential decision-making that aims at framing a problem of learning from interaction. In a Markov decision process, an *agent* interacts with an *environment* and periodically receives some *rewards*, numerical values analogue to payoffs in repeated stage games.

Definition 2.3.1. *An environment is defined by:*

- *a set of states* $\mathcal{S}$
- *a set of actions* $\mathcal{A}$
- *a probability distribution over transition to the next state given the current state and current action* $\Pr(S_{t+1} = s'|S_t = s, A_t = a)$ *with* $s, s' \in \mathcal{S}, a \in \mathcal{A}$
- *a probability distribution over numerical rewards given the current state and current action* $\Pr(R_{t+1} = r|S_t = s, A_t = a)$ *with* $s \in \mathcal{S}, a \in \mathcal{A}, r \in \mathcal{R} \subseteq \mathbb{R}$

Markov decision processes can be called as such if they respect the Markov assumption.

Definition 2.3.2. ***Markov assumption****: state transitions only depend on the current state and the actions taken by the agents, i.e.* $\Pr(S_{t+1} = s|S_t, S_{t-1}, ..., S_0) = \Pr(S_{t+1} = s|S_t)$

As a consequence of the Markov assumption, it is possible to define a function that expresses transition probabilities given the current state and actions. This function can be either a probability density function or a probability mass function, depending on whether the state and action spaces are discrete or continuous.

Definition 2.3.3. *A finite Markov decision process is a Markov decision process where* $\mathcal{S}, \mathcal{A}, \mathcal{R}$ *are finite sets and* $p(s', r|s, a) = \Pr(S_{t+1} = s' \wedge R_{t+1} = r|S_t = s, A_t = a)$ *is a well-defined discrete joint probability distribution of rewards and state transition.*

Most textbooks on reinforcement learning, such as Sutton and Barto (2018), focus on finite Markov decision processes because of their better tractability and ease of intuition. However, I choose to expand on the textbook notions and concern this section with the more general case in which the state and action spaces are continuous, referencing an array of different sources that treat these cases, attempting to unify notation. All proofs and computations carry over from the continuous to the discrete case by replacing integrals with summations and probability density functions with probability mass functions; in cases where the parallel is not clear, I refer the reader to the relevant sections of Sutton and Barto (2018), Rao and Jelvis (2022) and van Hasselt (2012).

Definition 2.3.4. *A policy* $\pi : \mathcal{S} \to \mathbb{R}$ *is a mapping from states to probabilities of selecting each possible action. If the agent is following policy* π *at time* t*, then* $\pi(a|s)$ *is a probability density function, conditional on the current state* $S_t = s$*,* *whose support is the action space $\mathcal{A}$.*

Note that π is an ordinary function; the conditional probability symbol serves to note that it defines a probability distribution over $\mathcal{A}$ for each state $s \in \mathcal{S}$.

There is substantial overlap between the environment of a Markov decision process and one of a repeated stage game.

For example, let there be a decision process in which:

- the action set $\mathcal{A}$ coincides with the set of strategies S of a repeated stage game
- the state set $\mathcal{S}$ coincides with the set of histories H_n of length n the same repeated stage game
- the rewards are given by the payoff function $\Pi : S \to \mathcal{R} \subset \mathbb{R}$ of a given agent in the same repeated stage game
- the transition probability distribution p is degenerate, deterministically mapping the current history and agents' joint actions to a new history
- players of the repeated stage game condition their strategies only on current history h_t (of length n)

Then, from the point of view of a single player, the other players' policies define a probability distribution that is a function of the current state. A policy in this process represents the same concept as a behavioral strategy in the repeated game. If n is not equal to t at every t, this means that *memory is bounded*, i.e. agents do not have perfect recall of all past actions and therefore cannot condition their behavioral strategies on all of them. The process becomes a *k-th order Markov decision process*, that is the next state depends not only on the current state and actions but on k previous states that allow for reconstructing the entire history of actions and, therefore, the policy of the agents playing the game.

2.4 Reinforcement learning

Reinforcement learning algorithms are a class of algorithms concerned with learning a *policy* that maximizes total reward in a Markov decision process. As in repeated stage games, the time horizon may be infinite, so what is actually maximized in this case is the *total discounted reward.*

A fundamental step in defining effective reinforcement learning algorithms is designing a process to estimate a *value function*, that is a function that estimates how "desirable" it is for an agent to be in a specific state, in terms of the potential reward it can expect from being in that state.

Definition 2.4.1. *The value function $v_\pi : \mathcal{S} \to \mathbb{R}$ is the expected total reward that the agent receives starting in state s and following policy π from that point on.*

Formally,

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \quad \forall\, s \in \mathcal{S} \tag{2.3}$$

where $\mathbb{E}_\pi$ denotes the expectation of a random variable given that the agent follows policy π. Like in Sutton and Barto (2018), I denote the reward received by taking action A_t in state S_t as R_{t+1}, to emphasize the fact that the reward and the next state S_{t+1} are jointly determined.

Note that the value function may easily be written recursively. Assuming a deterministic reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and defining the *transition function* T as in van Hasselt (2012) so that

$$\int_{\mathcal{S}' \subset \mathcal{S}} T(s'|s,a)ds' = \Pr(S_{t+1} \in \mathcal{S}' | S_t = s, A_t = a) \tag{2.4}$$

(note that T is a probability density function), we can write:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \\
&= \int_{\mathcal{A}} \pi(a|s) \int_{\mathcal{S}} T(s'|s,a) \left[R(s,a) + \gamma \mathbb{E}_\pi\left[\sum_{k=1}^{\infty} \gamma^k R_{t+k+1} | S_{t+1} = s'\right]\right] ds' da \\
&= \int_{\mathcal{A}} \pi(a|s) \int_{\mathcal{S}} T(s'|s,a)[R(s,a) + \gamma v_\pi(s')] ds' da
\end{aligned} \tag{2.5}$$

With eq. (2.5) being called the *Bellman equation* for the value function. Solving it amounts to finding v^*, its unique solution. Existence and uniqueness of v^* are proven for finite Markov decision processes in Szepesvári (2010) and for continuous-space/action and discrete-time Markov decision processes in Bertsekas (2019).

The Bellman equation is often written more compactly with the help of the Bellman operator.

Definition 2.4.2. *The Bellman operator $\mathcal{T}$ is defined so that*

$$\mathcal{T}v_\pi(s) = \int_{\mathcal{A}} \pi(a|s) \int_{\mathcal{S}} T(s'|s,a)[R(s,a) + \gamma v_\pi(s')]ds'da \tag{2.6}$$

The Bellman equation can then be written as

$$v_\pi(s) = \mathcal{T}v_\pi(s) \tag{2.7}$$

Having defined the value function, it is immediate to define its precursor, the *action-value function*, that is the desirability of taking a certain action in a given state.

Definition 2.4.3. *The action-value function $q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the expected total reward that the agent receives starting in state s, taking action a, and following policy π from that point on.*

Formally,

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \quad \forall\ (s,a) \in \mathcal{S} \times \mathcal{A} \tag{2.8}$$

Note that the value function can be recovered from the action-value function by taking its expected value over actions weighted by policy π, so the Bellman equation may be defined for the value function by simple algebra on the equivalence

$$v_\pi(s) = \mathbb{E}_\pi[q(s, A_t)] = \int_{\mathcal{A}} \pi(a|s)q(s,a)da \tag{2.9}$$

The objective of reinforcement learning algorithms is to learn the *optimal policy* $\pi^*(s,a)$. The optimal policy is the policy that maximizes the value and action-value function *for all possible states.* The optimal policy need not be unique, but all optimal policies share the same *optimal value function* $v^*(s)$ and *optimal action-value function* $q^*(s,a)$.

It is important to notice that the value and action-value functions take a single state (and action) as their argument, but they consider all future rewards in their value. This means that a policy that simply selects the optimal action in any state, i.e. $\pi^*(a|s) = \max_{a\in\mathcal{A}} q^*(s,a)$ is the optimal policy.

If everything about the environment is known, the optimal action-value function may be found through dynamic programming (Bertsekas, 2012). In most cases, however, dynamic programming methods may be inapplicable for a variety of reasons, like the *curse of dimensionality*: the computational complexity of dynamic programming is exponential in the dimension of the state space, if the state space is discrete, and these methods become completely inapplicable in a problem with continuous state spaces. Another very common problem is that of *unknown environment dynamics*: in the majority of cases it is not trivial to have a perfect model of the system, and the only available information is *experience*, most commonly in the form of a sequence of tuples $(S_t, A_t, R_{t+1}, S_{t+1})$ that must be used to estimate environment dynamics.

In such cases, the objective becomes to *approximate* the optimal policy, a task that can be carried out either directly, via *policy gradient methods*, or by estimating the optimal value or action-value functions. This latter approach, while computationally easier to tackle, is conceptually more convoluted as it formulates the problem of reinforcement learning as a two-stage problem. There is a *prediction* aspect, estimating the optimal value or action-value function, and a *control* aspect, the usage of the estimated value or action-value function to decide upon a policy.

2.5 Temporal-difference methods

Generalized policy iteration refers to the interaction of the approximation processes for the policy and the value function. A generalized policy iteration process progressively drives the current policy estimation towards a policy that is greedy with respect to the current value function estimation, while it also drives the current value function estimation towards consistency with the current estimated policy. It can be proven (Bertsekas, 1987) that iterating over value functions in such a way that $v_k(s) \geq v_{k-1}(s) \; \forall \; s \in \mathcal{S}, k \in \mathbb{N}$ converges to the optimal value function. Therefore, driving the policy estimation towards being greedy with respect to a value function estimation that is being constantly improved eventually results in both the policy and the value function converging towards optimality.

A way to implement generalized policy iteration is to estimate the action-value function and to always play the greedy policy with respect to said function, that is $\pi_k(s) = \arg\max_{a \in \mathcal{A}} q_k(s, a)$. However, following a strictly greedy policy with respect to a bad action-value function estimation may prevent the latter from improving and thus impede convergence. To help the estimation effectively traverse the space of possible action-value function, it is necessary to maintain *exploration* of the action spaces, which means sometimes not choosing the greedy action. A simple method to maintain exploration is, at each timestep, to only take the greedy action with some probability $1-\varepsilon$, and otherwise, with probability ε, sample an action uniformly from the action space. This is known as an *ε-greedy policy*. To ensure that the policy eventually converges to the greedy policy, it is possible to *decay* the exploration probability ε in such a way that $\varepsilon_t \to 0$ as $t \to \infty$.

Definition 2.5.1. Temporal-difference methods *are a class of reinforcement learning algorithms that learn to estimate the optimal action-value function and use some form of generalized policy iteration to control the agent in its interaction with the environment.*

In its simplest form, a general temporal-difference estimation of the value function $\hat{v}(S_t)$ is updated as

$$\hat{v}_{k+1}(S_t) = \hat{v}_k(S_t) + \alpha[R_{t+1} + \gamma\hat{v}_k(S_{t+1}) - \hat{v}_k(S_t)] \tag{2.10}$$

where $\alpha \in (0,1)$ is a learning rate and γ is the discount factor. If the state space is finite, $\hat{v}(s)$ can be a vector whose cell corresponding to the current state is updated at each timestep with this rule. This algorithm is known as TD(0) and can be proven (Sutton, 1988) to converge to the same answer, independently of the parameter α, if the latter parameter is sufficiently small. TD(0) does not learn a policy nor an action-value function but is instrumental in understanding the two most commonly used and cited temporal-difference methods, that is Sarsa and Q-learning.

TD(0) considers transitions from state to state and learns the values of states; the case of transitions from state-action pair to state-action pair allows for learning the values of such pairs and is formally identical. The theorems assuring the convergence of state values under TD(0) also apply to the corresponding algorithm for action values:

$$\hat{q}_{k+1}(S_t, A_t) = \hat{q}_k(S_t, A_t) + \alpha[R_{t+1} + \gamma\hat{q}_k(S_{t+1}, A_{t+1}) - \hat{q}_k(S_t, A_t)] \tag{2.11}$$

which is commonly seen written as a convex combination of current and past experience, i.e.

$$\hat{q}_{k+1}(S_t, A_t) = (1-\alpha)\hat{q}_k(S_t, A_t) + \alpha[R_{t+1} + \gamma\hat{q}_k(S_{t+1}, A_{t+1})] \tag{2.12}$$

This algorithm, with $\hat{q}(s,a)$ being a matrix, was proposed in Rummery and Niranjan (1994) as Modified Connectionist Q-learning and is known as Sarsa, because it requires knowledge of the tuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

Sarsa is actually an evolution of Q-learning, an algorithm proposed earlier by Watkins (1989) that has a slightly different update rule:

$$\hat{q}_{k+1}(S_t, A_t) = (1-\alpha)\hat{q}_k(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a\in\mathcal{A}} \hat{q}_k(S_{t+1}, a)] \tag{2.13}$$

Note that, while Sarsa's updates depend on the policy that is being followed, those of Q-learning only depend on the action-value estimation, i.e. there is no need to know A_{t+1}. This allows for approximating the optimal value function independent of the policy being followed. For this reason, Sarsa is known as an *on-policy* algorithm, while Q-learning is an *off-policy* algorithm because the policy it uses for its updates is different from the policy that the agent is currently following.

Both algorithms converge to the optimal action-value function under an ε-greedy policy with decaying exploration, but Sarsa tends to enjoy a better online performance in general, at the cost of slower convergence.

Temporal-difference methods are said to be *bootstrapping*.

Definition 2.5.2. *A bootstrapping reinforcement learning algorithm uses update rules that include current estimates of value or action-value functions instead of relying exclusively on actual rewards.*

This is important because bootstrapping off-policy algorithms, when combined with function approximation approaches, often exhibit divergent and unstable behavior. The combination of bootstrapping, off-policy algorithms, and function approximation is known as the *deadly triad*. Methods such as using a separate, *target neural network* to perform bootstrapping (van Hasselt et al., 2018), estimating two different action-value function to reduce maximization bias (van Hasselt, 2010) and multi-step bootstrapping (Watkins, 1989) are effective countermeasures for the divergence induced by bootstrapping.

2.6 Function approximation

Temporal-difference methods have historically been defined as approximating the action-value function by holding current estimates in a matrix and updating one cell at a time through their update rules. However, this approach quickly becomes unfeasible as state and action spaces grow and are inapplicable to continuous state and/or action spaces. Moreover, it results in algorithms that by design learn very slowly, since they update only one cell at a time and it takes a large amount of experience to converge to the optimal policy. In general, if reinforcement learning is to be used for real-world control tasks such as driving vehicles or controlling complex systems, algorithms need to be able to learn policies that can generalize to previously-unseen states and do not require a large amount of experience to converge, since experience is in general costly to obtain. Tabular temporal-difference methods, while simple to implement, are not able to perform either task well.

Temporal-difference methods may be adapted to not use a table, but a learned model instead, such as a linear estimation, a decision tree, or an artificial neural network. Algorithms that estimate a functional form of the value function, action-value function, or policy are known as *function approximation approaches*. Besides enhancing the formulation of temporal-difference methods, function approximation allows for introducing a new paradigm of RL, that is *policy gradient methods* that optimize a functional form of the policy directly.

Before introducing policy gradient methods, it is worth noting that the ordering of policies is *partial* (Naik et al., 2019) under function approximation for infinite-horizon tasks. That is, if there is no natural "ending" to a task, the definition given above for the optimal policy (maximizing the sum of discounted returns) does not allow for comparison between any two pairs of policies: some policies may achieve higher rewards in some state and lower in others. In principle, this is not an issue with tabular methods because they can represent potentially any policy and it can be proven that there exists one that maximizes value at every state simultaneously.

However, this is not feasible under function approximation, and even more so given a continuous state variable. The space of possible policies is so large that in most cases the optimal policy is not representable, so the objective becomes to find the *best representable policy*.

Naik et al. (2019) show that in the case of methods requiring function approximation, *finding the best representative policy in terms of discounted reward is not a well-defined optimization problem*. They suggest that a metric that can be maximized with stochastic gradient ascent is *average reward*. This metric can be defined if the Markov decision process is *ergodic*.

Definition 2.6.1. *A Markov decision process is ergodic if for any policy $\pi(s|a)$ there exists a* steady-state distribution *of states $\mu_\pi(s) = \lim_{t\to\infty} \Pr[S_t = s|(A_0, ..., A_{t-1} \sim \pi)]$ that is independent of the initial state S_0.*

Definition 2.6.2. *Average reward $r(\pi)$ represents the average reward by state weighted by how much time the agent spends in that state upon convergence, i.e.*

$$\begin{aligned} r(\pi) &= \lim_{t\to\infty} \mathbb{E}\left[R_t|(A_0, ..., A_{t-1} \sim \pi(\boldsymbol{\theta})), S_0\right] && (2.14) \\ &= \int_{\mathcal{S}} \mu_\pi(s) \int_{\mathcal{A}} \pi(a|s) \int_{\mathcal{S}} R(s,a) T(s'|s,a) ds'\, da\, ds && (2.15) \end{aligned}$$

Under the average-reward maximization problem, value and action-value functions are redefined in their *differential* formulation:

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} (R_{t+k+1} - r(\pi)) \middle| S_t = s\right] \tag{2.16}$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}(R_{t+k+1} - r(\pi))\middle| S_t = s, A_t = a\right] \tag{2.17}$$

The average reward can be estimated by keeping a running average that is initially equal to 0 and is then updated as

$$\bar{r}_{t+1}(\pi) = (1-\lambda)\bar{r}_t(\pi) + \lambda(R_{t+1} + \hat{v}(s') - \hat{v}(s)) \tag{2.18}$$

or, using the action-value function instead,

$$\bar{r}_{t+1}(\pi) = (1-\lambda)\bar{r}_t(\pi) + \lambda(R_{t+1} + \hat{q}(s',a') - \hat{q}(s,a)) \tag{2.19}$$

where $\lambda \in (0,1)$ is a learning rate, $S_{t+1} = s', S_t = s, A_t = a, A_{t+1} = a'$, and $a' = \pi(s')$ is given by the current policy estimation.

Note that these formulas do not merely represent a running average of rewards but include the *TD error* $\delta = R_{t+1} - \bar{r}_t + \hat{q}(S_{t+1}, A_{t+1}) - \hat{q}(S_t, A_t)$. The reason for this is that it ensures convergence: by updating the estimate as $\bar{r}_{t+1} = \bar{r}_t + \beta\delta$, once action-value function estimation converges we get

$$\begin{aligned}
\delta &= R_{t+1} - \bar{r}_t + \hat{q}(S_{t+1}, A_{t+1}) - \hat{q}(S_t, A_t) \\
&= R_{t+1} - \bar{r}_t + (R_{t+2} - r(\pi^*) + R_{t+3} - r(\pi^*) + ...) - (R_{t+1} - r(\pi^*) + R_{t+2} - r(\pi^*) + ...) \\
&= R_{t+1} - \bar{r}_t - (R_{t+1} - r(\pi^*)) \\
&= r(\pi^*) - \bar{r}_t
\end{aligned}$$

Therefore, the estimation converges to the average reward of the optimal policy, since it is being updated as $\bar{r}_{t+1} = (1-\lambda)\bar{r}_t + \lambda r(\pi^*)$.

I refer to the policy maximizing the objective function $r(\pi)$, as the optimal policy, even though under function approximation it is technically the optimal *representable* policy.

2.7 Policy gradient methods

Policy gradient methods focus on learning a parameterized policy $\pi(a|s;\boldsymbol{\theta})$, where $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ is a parameter vector, and $\pi(a|s;\boldsymbol{\theta})$ is a conditional probability density function on the current state whose support is $\mathcal{A}$. Policy gradient methods involve performing stochastic gradient ascent on some performance measure $J(\boldsymbol{\theta})$, so that every update step has the form

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J(\boldsymbol{\theta_t})} \tag{2.20}$$

where α is a learning rate and $\widehat{\nabla J(\boldsymbol{\theta_t})}$ is a stochastic estimate that is equal to the gradient of J in expectation.

Value (or action-value) function estimation, with the value function parameterized by $\mathbf{w} \in \mathbb{R}^d$ as $v_\pi(s, \mathbf{w})$, might still take place, but v_π is not required to select actions and is only used in computing the performance measure J. Algorithms that estimate a value function are usually called *actor-critic methods*, where the "actor" is the policy and the "critic" is the value function.

Policy gradient methods have an important theoretical advantage. Action-value methods greedily (or ε-greedily) select the "best" action according to a learned action-value function, so a single update to the action-value function estimation discontinuously changes the learned policy. On the other hand, the updates to a parameterized policy make action probability change smoothly, so stronger convergence guarantees are available in the form of the *policy gradient theorem*. This theorem gives an explicit expression for the gradient of the parameterized policy that is easy to compute. It is usually stated for the discounted return formulation of reinforcement learning but holds for the average reward formulation too (Sutton et al., 1999).

Theorem 2.7.1. ***Policy gradient theorem****. Let $J(\boldsymbol{\theta})$ be a performance measure of a parameterized policy π that is defined as the discounted value function of the initial state, i.e.*

$$J(\boldsymbol{\theta}) = v_{\pi(\boldsymbol{\theta})}(S_0) = \mathbb{E}_{\pi(\boldsymbol{\theta})}[\sum_{t=0}^{\infty}\gamma^t R_{t+1}]$$

or as the average reward rate as in eq. (2.15), i.e.

$$J(\boldsymbol{\theta}) = r(\pi(\boldsymbol{\theta})) = \lim_{t\to\infty} \mathbb{E}\left[R_t | (A_0, ..., A_{t-1} \sim \pi(\boldsymbol{\theta})), S_0\right]$$

Then, its gradient $\nabla J(\boldsymbol{\theta})$ is equal to

$$\nabla J(\boldsymbol{\theta}) = \int_{\mathcal{S}} \mu_\pi(s) \int_{\mathcal{A}} q_\pi(s,a)\nabla\pi(a|s;\boldsymbol{\theta}) da\, ds \tag{2.21}$$

The policy gradient theorem makes it possible to perform gradient ascent on J by computing the gradient of the policy with respect to its parameter vector, which is a relatively easy computation, and estimating an action-value function, a task that can be carried out, for example, through function approximation.

Note that most policy gradient methods proposed in the literature mention the discounted return formulation of the policy gradient theorem in their formulation, but actually update their parameters using estimates that do not approximate the performance measure gradient (Nota and Thomas, 2020), possibly leading to sub-optimal results. This issue does not concern my work since I am optimizing in the average-reward formulation and I do not use a discount factor for the rewards or the state distribution.

Chapter 3

Related work

The majority of deep reinforcement learning algorithms in the literature make use of one or more among *stochastic policies* (if they concern continuous action spaces) *replay memory* (if they are off-policy) and *target networks*.

This section addresses these concepts and specifies the algorithm I am using, *soft actor-critic* (Haarnoja et al., 2018b,a). It is currently the state of the art in reinforcement learning for continuous control and employs *entropy regularization* to maintain exploration and ensure good generalization capabilities.

3.1 Actor-critic methods in continuous action spaces

Algorithms that estimate both a parameterized policy and an action-value function, using the latter estimate to learn the former, are actually among the earliest algorithms studied in the literature. However, work on *off-policy* actor-critic learning is more recent (Degris et al., 2019) and kicked off an entire strand of reinforcement learning literature (Schulman et al., 2015, 2017; Fujimoto et al., 2018; Haarnoja et al., 2018b) concerned with solving the control problem in discrete and continuous action and state spaces.

In general, actor-critic methods are a class of policy gradient methods that learn a policy $\pi(a|s)$ that maximizes its own evaluation according to an action-value function estimation $\hat{q}(s, a)$, that is itself a learned approximation to actual expected differential (or discounted) reward. That is, actor-critic methods learn to jointly optimize (under the discounted or average-reward formulation) the objectives

$$\max_{\theta} \mathbb{E}_{\pi(\boldsymbol{\theta})} \left[\hat{q}(s, a; \mathbf{w})\right] \tag{3.1}$$

$$\min_{\mathbf{w}} \mathbb{E}_{\pi(\boldsymbol{\theta})} \left[\frac{1}{2} (\hat{q}(s, a; \mathbf{w}) - q(s, a))^2 \right] \tag{3.2}$$

Maximizing the first objective while improving action-value function approximation amounts to maximizing the performance measure $J(\boldsymbol{\theta})$, defined above in the discounted or differential cases. Optimization, in the case of neural-network-based function approximation, is carried out through usual stochastic gradient descent and backpropagation.

Parameterizing the policy with a real-valued vector is a very important paradigm shift since it allows for using an actor-critic method in continuous action spaces. In discrete action spaces, learning a policy means learning a vector of probabilities whose dimension is the number of discrete actions. In continuous action spaces, the number of actions is uncountably infinite. While learning to approximate the probability density function might be feasible in principle, in practice it is much more reasonable to either learn a deterministic mapping of spaces to actions, the approach used in deep deterministic policy gradient (Lillicrap et al., 2021), or fix a probability distribution and learn its parameters. The latter is how soft actor-critic methods proceed. In particular, actions are given by sampling from a normal distribution whose mean and variance are state-dependent and parameterized by the parameter vector $\boldsymbol{\theta}$, i.e.

$$A_t \sim N(\mu(s; \boldsymbol{\theta}), \sigma(s; \boldsymbol{\theta})) \tag{3.3}$$

where $\mu, \sigma : \mathcal{S} \times \mathbb{R}^{d'} \to \mathcal{R}$ are parameterized function approximations. Most modern actor-critic methods, including soft actor-critic, usually approximate action statistics and value functions using neural networks.

In principle, actions sampled for a normal distribution are unbounded, as the density function of a normal distribution is never zero. In practice, the probability for very low or high values *is* numerically zero, but it is often desirable (like in our case) to force actions inside a chosen interval. To do so, Haarnoja et al. (2018b) propose applying a squashing function to the normally-distributed network output whose range is a bounded interval, such as the hyperbolic tangent function whose range is $[-1, 1]$. The formulation of the probability density function of the resulting distribution is shown and proved in the appendices of Haarnoja et al. (2018b).

3.2 Reducing estimation variance

In general, convergence proofs for stochastic gradient descent require that the samples in each batch be independent and identically distributed, so that the estimator obtained equals the actual function gradient in expectation. However, this is assumption is violated when optimizing on subsequent experiences such as what happens in temporal-difference learning, since any two subsequent tuples (s, a, r, s') and (s', a', r'', s'') are correlated with one another and share one element, the middle state s'.

Replay memory, also known as experience replay, was first introduced by Lin (1992) to solve this issue. This method stores the agent's experience at each timestep in a *replay buffer* containing tuples $(S_t, A_t, R_{t+1}, S_{t+1})$. A buffer usually has a maximum size β and older experiences are removed and substituted with newer experiences in a first-in-first-out fashion once the buffer is full.

At each timestep, having a replay memory allows for performing multiple gradient ascent steps based on a *batch* of experiences sampled uniformly at random from the replay buffer.

Instead of S_{t+1} becoming the new state for the next update as it would in the usual form of temporal-difference learning, a new unconnected experience is drawn from the replay memory to supply data for the next update. Off-policy algorithms do not need to be applied along connected trajectories, and the possibility to use each stored experience for many updates allows for more efficient learning from experience.

Experience replay *reduces the variance* of the updates because successive updates are not correlated with one another as they would be if gradient descent was performed on subsequent experience tuples where the last element of the first tuple is always the first element of the second tuple. By removing the dependence of successive experiences on the current weights, experience replay eliminates one source of instability.

Moreover, experience replay allows for more efficient parallel computation, for example by using graphic processing units (GPUs) as is standard practice in supervised machine learning.

It is important to note that replay buffers have been conceived with Markov decision processes in mind. In a multi-agent environment where all agents are learning, taking an action in a given state many periods ago probably did not give out the same reward as it does now because the agent is still learning, which makes the parameter β a proxy for the degree of "smoothing the non-stationarity" of the environment, at least theoretically, by reducing the variance of the updates while increasing their bias.

In addition to the content of the experiences themselves, another source of correlated output is bootstrapping itself. The objective function in eq. (3.7) is used to optimized action-value function estimation but includes the estimation itself. Again, this translates into a correlation between inputs and impedes convergence. To solve this other issue, Mnih et al. (2015) propose computing the TD error using a separate network parameterized by $\mathbf{u}$, with $\mathbf{u}_0 = \mathbf{w}_0$ at $t = 0$. $\mathbf{u}$ is then updated as an exponential moving average between $\mathbf{u}$ and $\mathbf{w}$, i.e. $\mathbf{u}_{t+1} = \lambda \mathbf{u}_t + (1 - \lambda)\mathbf{w}_{t+1}$, where λ is a learning rate. This detaches the optimization target from the current estimation, thus allowing for smoother convergence, while only introducing a very slight lag in action-value function estimation if λ is small enough. Usually, this hyperparameter is in the order of 0.001, but models are not overly sensitive to it. As an alternative, Mnih et al. (2015) also explore setting $\mathbf{u} = \mathbf{w}$ every few periods, observing very similar results.

Note that, in the average-reward formulation, the TD error has to be computed with the target network to make average reward estimation consistent (proof that eq. (2.19) converges to average reward is provided in the previous chapter).

3.3 Entropy regularization

Soft actor-critic methods (Haarnoja et al., 2018b) propose a different formulation of action-value function and policy estimation. They learn a stochastic policy $\pi(\boldsymbol{\theta})$ that maximizes rewards subject to a soft entropy constraint $\mathcal{H}\{\pi(\boldsymbol{\theta})\} = \bar{\mathcal{H}}$, where $\mathcal{H}$ is defined as the Shannon differential entropy of a continuous distribution

$$\mathcal{H}\{f(x)\} = -\int_{-\infty}^{\infty} f(x) \log x dx = -\mathbb{E}_f[\log x] \tag{3.4}$$

Intuitively, this number is 0 is the distribution is deterministic and has a maximum equal to $\log(b - a)$ at the uniform distribution with support $[a, b]$. To enforce the entropy constraint, the authors reformulate the action-value function as the *soft action-value function* (they use the discounted formulation, but the average-reward formulation is identical)

$$q_{\pi,soft}(s, a) = \mathbb{E}_\pi \left[\left(\sum_{k=0}^{\infty} R_{t+k+1} - r(\pi) + \alpha \log \pi(A_{t+k}|S_{t+k}) \right) \middle| S_t = s, A_t = a \right] \tag{3.5}$$

and reformulate the optimization problem to:

$$\max_\theta \mathbb{E}_{\pi(\boldsymbol{\theta})} \left[\hat{q}(s, a; \mathbf{w}) + \alpha \log \pi(a|s; \boldsymbol{\theta}) \right] \tag{3.6}$$

$$\min_{\mathbf{w}} \mathbb{E}_{\pi(\boldsymbol{\theta})} \left[\frac{1}{2} (\hat{q}(s, a; \mathbf{w}) - q_{\pi,soft}(s, a))^2 \right] \tag{3.7}$$

with α being called the *temperature* and determining the relative weights of the two terms of the new objective function. From here on, learning proceeds as usual with stochastic gradient descent and backpropagation.

Note that here and in table B.1 I define the target entropy as a positive number since I employ the usual definition of Shannon entropy with a minus sign in front and redefine eq. (3.6) and eq. (3.7) to be consistent. This is the only difference between my definition and that of Haarnoja et al. (2018b), but signs are reversed in the code to match theirs for ease of understanding when compared with other implementations.

Entropy does not represent a sort of "variety" of the policy and has no meaning in the case of a deterministic policy, like those learned by DDPG (Lillicrap et al., 2021) or PPO (Schulman et al., 2017). Rather, it is linked to the exploration-by-distribution-learning inherent to algorithms that learn to estimate a probability distribution of actions given state instead of a deterministic function. Confusingly, the term "distributional RL" in the literature does not refer to such a class of algorithms but rather to algorithms such as Rainbow (Hessel et al., 2018) that learn to estimate a distribution of *rewards* rather than actions, that is the non-determinism lies in $q(\cdot)$ and not $\pi(\cdot)$.

3.4 Machine learning and cooperative behavior

Autonomous pricing is increasingly used across industries, powered by the rise in online shopping (Chen et al., 2016). Most likely, many of the algorithms that firms actually employ are not based on model-free reinforcement learning, since it is reasonable to assume that firms are able to estimate market characteristics and, therefore, desire to inform their pricing strategies with as much information as possible. Moreover, classical (i.e. not based on machine learning) algorithms from control theory are able to achieve impressive feats in fields such as robotics (Tassa et al., 2012; Kuindersma et al., 2015) without suffering from the brittleness and non-determinism inherent to machine learning and especially to (unsupervised) reinforcement learning.

Nevertheless, the reinforcement learning setting is interesting because it provides an environment in which there is almost no information and, importantly, no conception of an adversary, thus it is very easy for firms to justify their use and label them as non-collusive. However, collusion seems to emerge even in such an informationstarved environment. Calvano et al. (2020) develop a tabular Q-learning model applied to repeated competition in a Bertrand model of the market with logit demand and constant marginal costs, showing that collusive behavior with reward-punishment schemes emerges after around one million timesteps. Their results are robust to linear and stochastic demand systems, variations of market parameters, multiple firms, and different formulations of the action set, but the time scale of convergence is so long (in the order of millions of timesteps) that it is unreasonable to expect similar strategies to be deployed in real markets. Recently, Furlan (2022) showed that even if agents do not have perfect recall of past actions they are nevertheless able to coordinate on collusive prices, and the only way to ensure collusion does not emerge is by making the opponent's price completely unobservable, which makes agents unable to punish deviations and leads to Nash equilibrium pricing. His work supports the view of algorithmic collusion as a behavior that does not require much information and that is a natural results of optimizing to maximize profit in a price competition environment. Klein (2021) extends the results of Calvano et al. (2020) to a model of sequential pricing, showing that in such a setting agents converge on either stable or cyclical supra-competitive pricing. It is interesting to note that the cyclical behavior emerges if the set of prices is large, but the cycles are of the Edgeworth kind from Maskin and Tirole (1988) and not symmetrical. A hypothesis for the emergence of symmetrical cycles is that they are the results of algorithms "wanting" to play a price that is in the middle of the cycle, but this does not explain the emergence of asymmetric and finite-duration Edgeworth-type cycles. Hettich (2021) shows that applying discrete-action/state function approximation approaches, specifically DQN (Mnih et al., 2015), to the simultaneous-play case leads to faster convergence, even though the improvement is still not sufficient to justify the emergence of collusion in a reasonable time in most real markets, and DQN still shares the limitations of tabular Q-learning, such as a discrete action set.

There are further studies on cooperative outcomes obtained by reinforcement learning in games, even from an economic perspective. For example, Siallagan et al. (2013) shows that agents using aspiration learning, a form of reinforcement learning that chooses actions by comparing them to a learned measure of "payoff to aspire to", leads to collusive outcomes in some Cournot oligopolies. However, in this case, there is no memory of rivals' past actions, which means that cooperation is brittle and only emerges because algorithms are matched and learn to never deviate. A human player or a smarter algorithm, knowing this, may exploit the cooperating associate. A similar model applied to the prisoner's dilemma, Karandikar et al. (1998), showed cooperative behavior in a setting in which agents could only remember their own actions, which however still does not allow for detecting deviations.

Concerning model-based learning, Asker et al. (2021) argue that endowing Q-learning with information about market structure makes it significantly more sensitive to structure itself. In their analysis, they propose a different formulation of the tabular Q-learning update rule that updates an entire row of their Q-matrix at once, by knowing the demand function and computing exact counterfactual profits, a process they call "synchronous update". They show that in this case, myopic agents ($\gamma = 0$) converge to Nash equilibrium prices very rapidly, which is expected since by updating all cells in a row, prices are necessarily skewed downwards since for any price above the Nash equilibrium, undercutting is profitable in the immediate. Agents ignore future consequences if $\gamma = 0$, and the update rule in eq. (2.13) reduces to a moving average that converges to the one-shot payoff. In the case of non-zero discount factors, their modified Q-learning achieves behavior similar to standard tabular Q-learning, but with lower profit gains. This behavior remains even if the all-actions update is merely informed by the fact that demand curves are downward sloping, and not by an exact knowledge of the profit functions. Again, this behavior is arguably expected since it makes agents more myopic to the evolution of the rival's strategy, reducing the variance in the action-value function estimation (because in expectation updates effectively have a learned baseline, such as in Williams 1992) while introducing bias towards the current policy.

Still on the topic of model-based cooperative learning, but from a non-economic perspective, Crandall et al. (2018) show that an aspiration-based meta-algorithm that manages an ensemble of models, coming from both classical game theory and machine learning, is able to reach cooperative outcomes when associated with a human player or a similar algorithm. However, the games they use as benchmarks are in general very small, with complete information, discrete actions, and little to no concept of state, and therefore are quite different from how a competitive market is modeled in economics. More complex algorithms such as counterfactual regret minimization have successfully been applied to poker (Moravčík et al., 2017), a game that has state, complete but imperfect information, stochastic behavior, and a much more complex structure. In general, however, the model-based literature on game theory seems to focus on discrete action spaces rather than continuous ones, where reinforcement learning shows suboptimal results compared to model-based control laws (Mania et al., 2018), and a vast literature coming from control theory is available. This is reasonable since many well-known game theory problems and applications, from the classic prisoner's dilemma to video games, effectively have discrete action spaces, but is arguably a bad fit for simulating competitive markets.

On the empirical side, Assad et al. (2020) provide evidence of the effect of Autonomous pricing on market prices for gasoline pumps in Germany, showing that they consistently lead to higher prices when matched with one another. To this date, to the best of my knowledge, there are no empirical studies on the behavior of deployed pricing algorithms when faced with an exogenous price cut that is not due to a demand shock, i.e. responses to deviations from a collusive equilibrium. Both Assad et al. (2020) and Byrne and de Roos (2019) show a timescale of collusion in the order of a few years, which is interesting since, as shown in the following sections, allowing for a few daily fluctuations in price makes it possible to get a comparable timescale of collusion with policy gradient methods. This realistic timescale as well as other behavior analyzed in the following section, such as convergence with persistent exploration, also match some of the gaps that critics of the idea of the possibility of algorithmic collusion, such as Schwalbe (2018), have pointed out and that have not been yet taken care of by tabular Q-learning and may call for a re-evaluation of the issue of algorithmic collusion, especially since model-free deep reinforcement learning does not benefit from any of the improvements that sensible model-based learning would bring.

Chapter 4

Methodologies

Having specified soft actor-critic, in this chapter I specify and argue in favor of several architectural specifications that simplify the model and allow for more robust behavior in a multi-agent context. I also describe the economic model underlying the environment in which agents are learning and the characteristics that make policy gradient methods a fitting choice.

Concerning technical details, I developed this model in Python 3.9.12 using PyTorch 1.11.0. I ran experiments on Ubuntu 20.04 on Windows Subsystem for Linux 2 running on a system equipped with an Intel Xeon Silver 4210 CPU and two NVIDIA GeForce RTX 2080 Ti GPUs. This setup was kindly provided by the Department of Economics at Bologna University. I release the code[1] under the GNU Affero General Public License v3.0.

4.1 Economic environment

I follow the economic environment of Calvano et al. (2020) and call *pricing game* a repeated stage game in which firms set market prices for various heterogeneous goods and observe their profits in discrete timesteps. The action space is defined by the prices set by the n agents, the state space is defined by the past actions of all agents, and the reward is given by profits under a model of price competition with logit demand and constant marginal costs. In economics, a model of the market with firms competing on prices is known as a Bertrand competition model.

In a given timestep t, prices are defined as a positive real vector $\mathbf{p}_t$. Each of the n firms produces a distinct product i. Demand for product i is given by

[1]`https://github.com/kmfrick/Algorithmic_Pricing_ActorCritic`

$$q_{i,t} = \frac{e^{\frac{a_i - p_{i,t}}{\mu}}}{\sum_{j=1}^{n} e^{\frac{a_j - p_{j,t}}{\mu}} + e^{\frac{a_0}{\mu}}} \tag{4.1}$$

Parameters a_i capture vertical differentiation, that is the difference in demand for products that comes from some asymmetry in their inherent "quality". Conversely, parameter μ quantifies horizontal differentiation, that is differences between products that cannot be measured objectively and come down to individual tastes.

and profits, which correspond to the reward received by the agent, are given by

$$R_{i,t+1} = (p_{i,t} - c_i)q_{i,t} \tag{4.2}$$

where c_i is the *marginal cost* of producing a single unit of a good. Note that the *reward for actions taken in timestep t is accrued in timestep $t+1$*, following the notation in Sutton and Barto (2018). This notation, while it might be regarded as confusing, is standard in most of the reinforcement learning literature.

When firm are symmetrical, i.e. $a_i = a_j = a \; \forall \; i, j \in \{1..n\}$, and the game is treated as as being one-shot, the Nash equilibrium is the unique fixed point (Anderson et al., 1992) of the equation

$$p_N = c + \frac{\mu}{1 - (n + e^{\frac{a_0 - a + p}{\mu}})^{-1}} \tag{4.3}$$

and it is possible to define the full collusion (monopoly) price vector as

$$\mathbf{p}_M = \max_{\mathbf{p}} \sum_i (p_i - c_i)q_i \tag{4.4}$$

In the symmetric firm case, this reduces to a single-function unconstrained maximization problem

$$p_M = \max_p n(p - c)\frac{e^{\frac{a-p}{\mu}}}{ne^{\frac{a-p}{\mu}} + e^{\frac{a_0}{\mu}}} \tag{4.5}$$

Two issues require attention in defining a Markov decision process from the pricing game.

The first issue is the size of the state space. If n agents remembered all prices played by themselves and all of their opponents in all discrete timesteps up to t, the state space would be a vector of dimension n^{2t}, making any computation completely unfeasible after a handful of timesteps. To solve this issue, I define the state space as the set of histories of length $p < t$, i.e. agents have a *bounded memory*, resulting in a state space of dimension n^p. Unless otherwise specified, I focus on the case where $p = 1$, that is, agents only remember the last price that their and their associates played.

Note that as soon as memory is bounded, the process does not respect the Markov assumption as usually stated, therefore applying reinforcement learning algorithms to play repeated stage games may be considered improper. However, models used in applied science usually have some degree of fuzziness that is allowed in the validity of their assumptions, and reinforcement learning is no exception. This is true, for example, in linear regression, where diagnostic plots serve the purpose of evaluating *the degree of correctness* of the assumptions on the data set that is being considered. To the same extent, applying reinforcement learning algorithms intended for Markov processes may not be soundly theoretically grounded, they may be empirically be found to be effective and provide incentive for further theoretical research.

The second issue is that, in a competitive market, firms are free to set their prices to any positive real number. From the point of view of a single reinforcement learning agent that sets a firm's prices after having observed the others', this means that the action and state space are potentially equal to $\mathbb{R}$ and $\mathbb{R}^n$ respectively. However, temporal-difference algorithms assume discrete action and state spaces. Discretizing the state and action space as in Calvano et al. (2020) can be an effective workaround for this issue, and allows to treat the game as any other bimatrix game, where results from the preceding literature, like Littman and Stone (2001), may still apply. However, discretization entails a loss of information. Temporal-difference methods applied to a discretized price grid have no way of knowing the ordering of prices: the topological structure of the state and action spaces is lost. The use of policy gradient algorithms removes this requirement, as the policy estimation can be parameterized with any parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$.

To characterize emerging collusion under autonomous pricing as "tacit", I want to ensure that there is no possibility of communication between the two agents, or any other information that could potentially be described as nudging the algorithms towards collusive behavior. For this reason, I do not make any assumptions about complete information and let agents learn in a completely unsupervised and model-free context. I do not make the agents share any parameters, do not inject any knowledge about the environment, and only let agents observe their own profits and their and their associates' market prices. The only objective of the algorithms is to maximize the expected reward, subject to the maximum entropy constraint. This leads us to the issue of *sample complexity*, which is also debated, conversely, as "sample efficiency" and is the root of the very long timescale of collusion observed by Calvano et al. (2020).

Definition 4.1.1. *Sample complexity is the number of samples (state-action-reward-next state tuples) that an algorithm requires to achieve convergence or satisfactory performance.*

It is the RL analogue of computational complexity for classical algorithms, and does a better job of highlighting the main bottleneck in their execution, which in general is not the learning itself but the collection of experience and feedback. In this regard, the pricing game serves as a very good illustrative example. Rainbow (Hessel et al., 2018) is a model for Markov decision processes with discrete state and action spaces that achieves human performance at about forty million timesteps, defined as frames in a video game. In Calvano et al. (2020), convergence takes hundreds of thousands to millions of timesteps. However, video game frames are cheap to generate, while demand for goods is not. Once a firm sets its price for a good, it takes time for it to observe demand and, therefore, profit. While this time span may vary between industries and markets, cases where it is comparable to the execution time of a gradient descent step are extremely rare and arguably limited to high-frequency trading in financial markets. This means that whatever algorithm the firm may be using, a timestep does not only last for the length of time required for the algorithm to run, but there is a sizable delay in observing the reward. If this delay is in the order of one hour, the million timesteps necessary for convergence in the work of Calvano et al. (2020) amount to more than one hundred years of real-life time. The main contribution I bring is showing that it is possible to significantly improve sample efficiency on the pricing game without changing the formulation of the pricing game in a manner that requires more information, or assumptions. In addition, I discuss possibilities for achieving even higher sample efficiency through model-based learning in the conclusions.

4.2 Specifications of the algorithm

Constructing a reinforcement learning model implies tuning many implicit hyper-parameters in the form of architectural choices, and since the release of the first paper on soft actor-critic (Haarnoja et al., 2018b) various improvements have been proposed that are both specific to soft-actor critic and generally applicable to machine learning architectures. Machine learning is a very active research field, but models in deep reinforcement learning do not necessarily benefit from architectural improvements that have been developed for supervised learning. For example, batch normalization has been found to induce gradient explosion (Yang et al., 2019), a catastrophic phenomenon in the context of reinforcement learning. Another example is the activation function for the hidden layers. While in supervised learning

ReLUs usually result in faster training as they do not suffer from vanishing gradients, Andrychowicz et al. (2020) show that using tanh activations in reinforcement learning stabilizes training and improves convergence, and this behavior probably stems *exactly from the behavior of gradients induced by tanh activations*. On neurons with tanh activations, gradients decrease for extremely high or low values of neuron outputs, thus providing an alternative to the gradient clipping that is usually performed (see for example Mnih et al. 2015).

It is worth noting that temporal-difference methods are not exempt from this "architecture tuning", as they do not only require tuning of the learning and exploration rates but entail a choice of an exploration strategy, as well as between on- or off-policy learning and whether or not to implement bias/variance reduction techniques such as n-step return or double Q-learning.

When constructing the model I used, I followed the principle of Occam's razor and preferred removing components to adding them. For example, adding a separate value function estimation like in Haarnoja et al. (2018b) introduces an additional layer of complexity whose benefit is unclear. Since the TD error may be formulated in terms of an action-value function estimation, I decided not to employ a separate value network and compute the TD error as $R_{t+1} - \tilde{r}_t(\pi) + \hat{q}(s', a') - \hat{q}(s, a)$ (Haarnoja et al., 2018a).

I train the actor network using gradient ascent on the entropy-regularized policy objective, but remove the entropy regularization term from the critic network loss, a modification that has been found to lead to more stable convergence (Yu et al., 2022). I also employ an actor network whose layers are one-eighth of the size of those in the critic network and use the default initialization for weights and biases, i.e. with zero bias and random weights sampled from $\mathcal{U}\left(-\sqrt{\frac{1}{d}}, \sqrt{\frac{1}{d}}\right)$, d being the number of input features.

I use neural networks with two hidden layers with tanh nonlinearities and employ two Q-networks to reduce overestimation of the action-value function (van Hasselt, 2010). The estimate $\hat{q}(s, a)$ is thus given by $\min_{\widehat{q_1}, \widehat{q_2}} \{\widehat{q_1}(s, a), \widehat{q_2}(s, a)\}$

These architectural choices lead to a relatively simple model and have been empirically found to improve convergence, even though at the moment there is no formal explanation for their effectiveness. In the literature, a hypothesis that has been put forward, also mentioned above, is that smaller networks and saturating activations reduce variance in neural network weights, outputs and gradients. This is undesirable in supervised learning as it slows down training but helps achieve convergence and stabilize training curves in the heavily non-stationary environment of reinforcement learning (Andrychowicz et al., 2020).

Agents do not discount rewards, and they keep a running estimate of average reward as in eq. (2.19). The action-value estimations required are computed using the target network.

To ensure prices are above marginal cost, I scale the actor network output $x \in (-1, 1)$ as $a = p_N - \xi + \frac{x+1}{2}(p_M - p_N + 2\xi)$, which means that the action space is $\mathcal{A} = (p_N - \xi, p_M + \xi)$ and the state space is $\mathcal{S} = \mathcal{A}^n$, with n being the number of agents. I enforce an entropy constraint equal to $\bar{\mathcal{H}} = \dim(\mathcal{A}) = 1$, following the heuristic in Haarnoja et al. (2018a).

The critic network has an input layer of size $\dim(\mathcal{A}) + \dim(\mathcal{S}) = n + 1$. The input layer is followed by two hidden layers of size M with tanh activation functions and an output layer of size 1. The actor network has the same structure, but the input layer is of size $\dim(\mathcal{S}) = n$, the output layer is of size 2 (mean and variance of a normal distribution) and the hidden layers are of size $M/8$.

I use the Adam stochastic optimizer (Kingma, D.P. et al., 2015) to perform gradient descent, with a constant step size λ_A for the actor and λ_C for the critic and entropy. I perform updates at every step, sampling batches of size B from a replay buffer of maximum length β, implemented in C++ for performance (Yamada, 2019). For the first B steps, I let agents play entirely random actions.

This configuration translates into multiple hyperparameters, all of which have to be tuned. However, the model being simple means that in general, once the order of magnitude has been found, results are robust to changes to hyperparameters that remain in the same order of magnitude. Since an exhaustive grid search is not feasible due to the runtime of a single experiment, which takes about one hour on an RTX 2080 GPU, I ran a sequential model-based optimization search using Optuna (Akiba et al., 2019). Even if the effectiveness of such a search is debated, there is no reason why it should perform worse than random search, which has been shown to perform better than grid search (Bergstra and Bengio, 2012) in finding good hyperparameters for neural networks.

To ensure reproducibility, I fix the random seeds used in the simulations and include them in the code.

Chapter 5

Results

In this chapter, I describe the evolution of payoffs obtained in the environment at hand by the algorithms specified in the previous chapter. I also analyze what kind of policies they end up learning, whether they can be described as collusive policies and the reasons for their emergence.

Collusive outcomes may be the result of simply failing to learn the optimal strategy. That is, agents may simply observe that, in general, higher prices tend to correspond to higher profits and drive their policies towards blindly setting high prices regardless of what the opponent is doing. If all agents behave like this, collusion may seem to emerge. However, this is not an equilibrium since collusion is not supported by any punishment and such a strategy could be easily exploited by an opposing firm by undercutting the blind-pricing agent. The aim of this section is therefore to show that collusive outcomes are the result of learning the optimal policy for the given environment and that agents respond optimally if the associate deviates in any way.

Reported results have been obtained using the hyperparameters specified in table B.1, but they are robust to changes in these hyperparameters that do not exceed an order of magnitude. In terms of economic characteristics, to enable comparisons with the tabular Q-learning of Calvano et al. (2020), I fix $a_0 = 0, \mu = 0.25, a_1 = a_2 = 2$. I use a slightly larger action space than they do, setting $\xi = 0.1$.

Note that it is difficult to define convergence as in Calvano et al. (2020) when using policy gradient algorithms, for a variety of reasons. First, since prices are continuous and agents' policies are required to have a constant, non-zero entropy, it is not trivial to define when they have "stabilized": soft actor-critic, by definition, converges to a stochastic policy that does not maximize average profit only but also factors in the entropy of the learned policy, therefore it can never converge to a stationary policy, or to one with arbitrarily low variance. Second, algorithms using replay buffers and neural networks are prone to catastrophic forgetting if learning continues indefinitely (and it usually does so since there is no learning rate decay), which means that in general, they always diverge given enough time. Both factors essentially stem from the fact that exploration is not decaying as it often does when using tabular methods. Usually, the reinforcement learning literature deems convergence as achieved if reward exceeds some predefined threshold for a certain number of timesteps, usually an index of how well humans do in this task. Afterward, exploration is turned off and final performance is evaluated. However, setting market prices so as to collude with no information or communication is not a task that humans usually perform. We can note that, robustly across sessions, average profit gains surpass 50% (the profit gains obtained by random pricing) after a handful of iterations, and in general remain higher than 75% throughout simulations. However, algorithms may still be optimizing their strategies from the replay buffer while profit gains are stable, so evaluating performance as soon as profit exceeds some threshold would not be representative of the actual behavior of the algorithms. In general, it would be up to the firm to decide when the algorithm has achieved satisfactory performance and can be deployed without exploration. Therefore, I decide to define convergence on a data-driven basis, finding "chaotic" sessions by applying techniques for outlier detections to the sample variance of end-of-learning profits. Moreover, plots of impulse-response behavior and visualizations of the learned policies during learning allow us to understand agents' training process.

5.1 Training curves

To ease exposition and abstract from market characteristics, I define *profit gains* in a similar way to Calvano et al. (2020):

$$\Delta = \frac{r - \Pi_N}{\Pi_M - \Pi_N} \tag{5.1}$$

where r is the average reward obtained at convergence, Π_N is the Bertrand-Nash static equilibrium profit, and Π_M is profit under full collusion, or monopoly profit. Δ is 1 in case of perfect collusion (monopoly pricing) and 0 in case of Bertrand-Nash play. Since my state space is slightly larger than Calvano et al. (2020), profit gains have a wider range. Note that *profits* themselves cannot be negative because the lowest price agents can set is higher than marginal cost; however, if profits are lower than Nash equilibrium profits, Δ is negative.

The training curve of the representative experiment, showing profit gains over 40000 time steps, is plotted in fig. 5.1.

We can see that both agents reach profit gains of around 0.8 after about 5000 timesteps and remain more or less stable afterward. Interestingly, as soon as agents

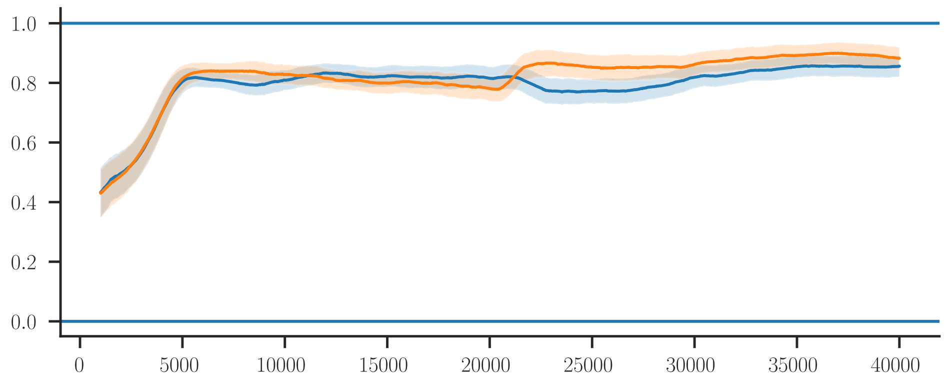


Figure 5.1: Profit gains over time. Moving average over 1000 steps. Shaded area represents standard deviation.

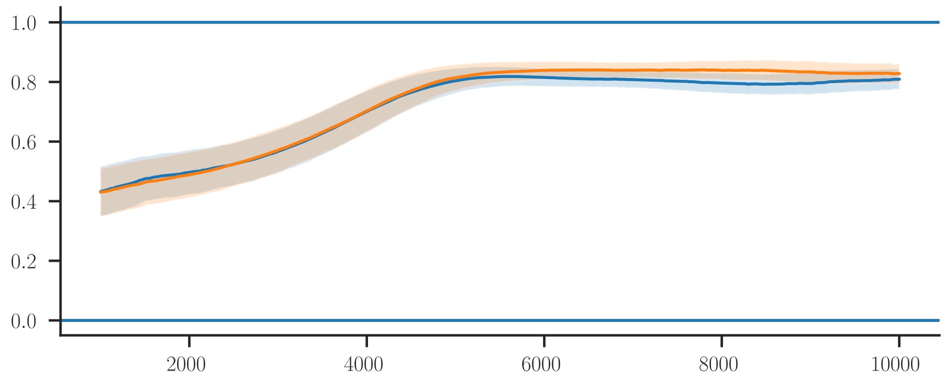


Figure 5.2: Profit gains over the first 10000 time steps. Moving average over 1000 steps. Shaded area represents standard deviation.

are allowed to use their policy network (remember that, for the first 512 timesteps, they play entirely random prices) they start to raise prices. This behavior is not observed with tabular Q-learning, with agents initially undercutting themselves and rising prices gradually. Figure 5.2 zooms in on the first 10000 timesteps, and fig. 5.3 shows mean profit gains between the two agents. Both figures confirm that indeed profit gains rise very rapidly and reliably.

Keeping in mind the discussion of convergence from above, and that the standard deviation of prices seems low in general, we could indeed argue that algorithms have converged to collusive strategies after about 10000 timesteps, two orders of magnitude faster than what Calvano et al. (2020) observed with tabular Q-learning. Moreover, as compared to tabular Q-learning, price cycles almost never surface. Cycles occurred only five times across more than 100 experiments, and those cases were characterized by high variance in prices throughout learning, so they could easily be characterized as a lack of convergence (see the discussion on chaotic sessions

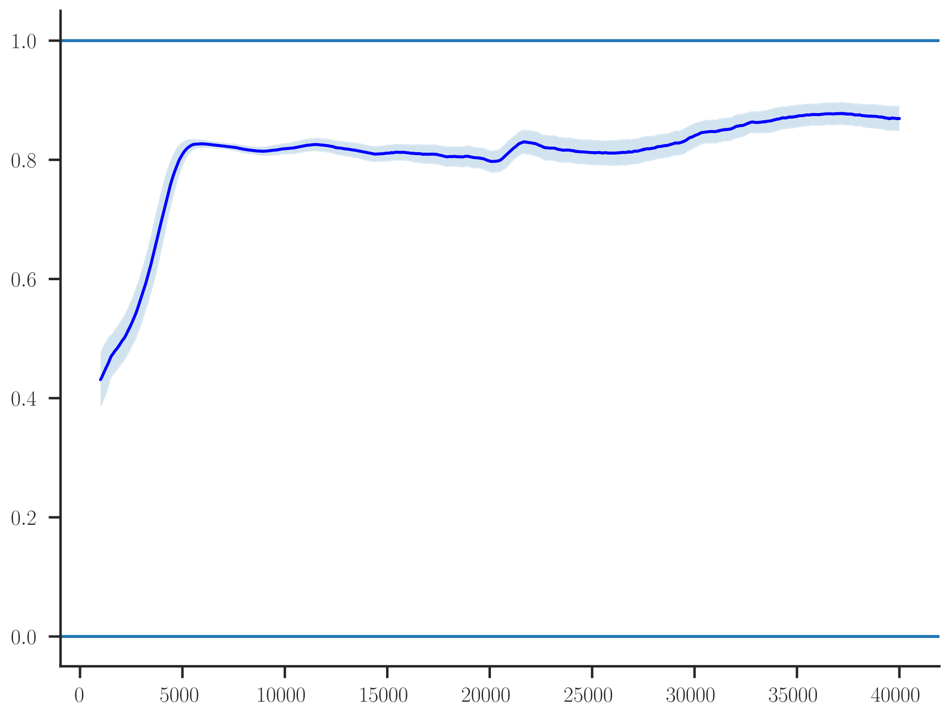


Figure 5.3: Mean profit gains over time. Moving average over 1000 steps. Shaded area represents standard deviation.

below). This supports the view that price cycles are an artifact of discretization and disappear when treating the problem in continuous action spaces.

All this being said, it is important to remember that collusion may not be determined by actual collusive strategy but by failure to learn an optimal policy and mere blind pricing. Therefore, the next two sections are devoted to quantifying *what* have algorithms learned.

5.2 Visualizing learned policies

Using neural networks to represent policies and action-value function means positing that there exist functional forms of the two, since we know that neural networks are in fact universal function approximators (Hornik et al., 1989). Using continuous pricing in a game with two agents and one-period memory allows for visualizing this functional form through a heatmap, with observed prices p_1, p_2 as the axes, and the color being proportional to the price to be set. This is possible since, in this environment, a policy is effectively a function $\pi : \mathbb{R}^2 \to \mathbb{R}$.

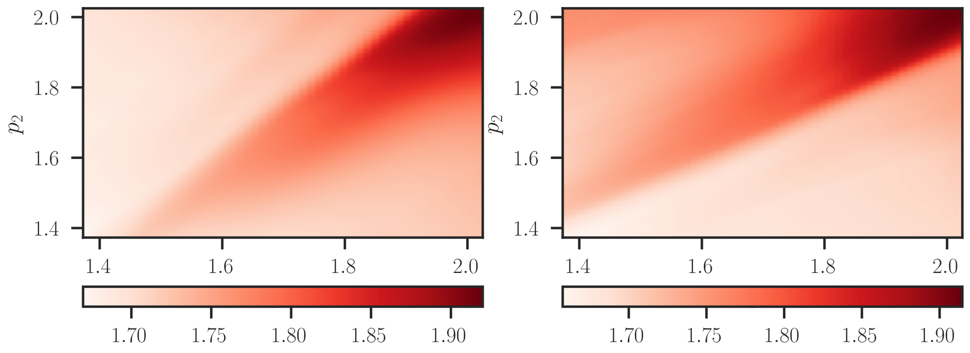


Figure 5.4: Heatmap of the two agents' learned policies after 40000 timesteps.

Figure 5.4 shows a heatmap corresponding to the policies learned by two agents after 40000 timesteps. We can see that agents exhibit symmetrical learning behavior that leads to high but not perfectly collusive prices, as well as the emergence of reward-and-punishment schemes. Both agents learn to play high prices if their associate is doing so, and there appears to be a zone of very high prices that corresponds to the point where profit gains stabilize, with resulting profit gains of around 0.8. Agents learn that it is profitable to undercut the associate when it is playing very high prices, which explains why they do not reach full collusion. However, they also learn not to undercut unless the other player is doing the same. Heatmaps are symmetrical, with the "collusive zone" lying on opposite sides of a line that is at an angle of slightly less than 45 degrees with respect to the axis of the opposing player so that the zone with the highest prices overlaps in the two plots. Punishment gets harsher with more learning, as is shown by fig. 5.5 and the "collusive zone" becomes smaller and with a steeper descent. Evolution appears to stop after 40000 periods, which is reasonable since by this point a replay buffer of 20000 periods is filled with experiences from policies that are almost stationary, and agents are merely reinforcing their behavior by learning from very similar experiences.

5.3 Response to deviations

Having verified learning behavior, one key question in reinforcement learning is how the learned policy generalizes to unseen behaviors. This same question arises in economics, where a collusive strategy is sustained only if it is not optimal to deviate from it, i.e. if the non-deviating agent acts in such a way as to punish deviations and render them unprofitable. As stated earlier, if agents learned to merely play high prices, this would not be collusion but a failure to learn the optimal strategy and firms may expect rivals to take advantage of their algorithms' blindness and undercut them. To understand whether agents learned optimal behavior even off

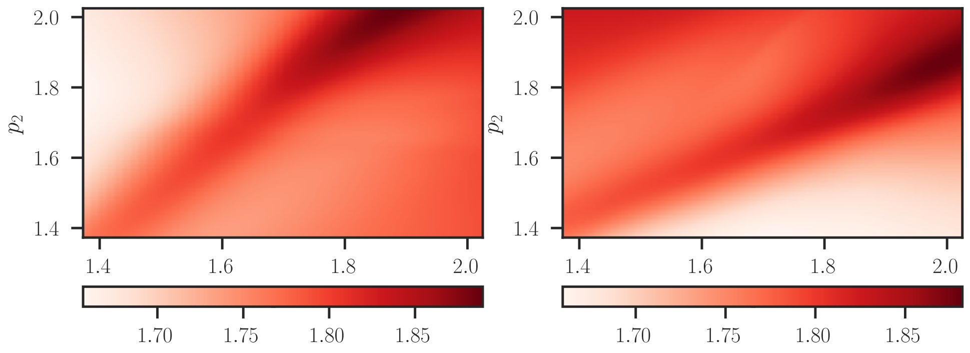


Figure 5.5: Heatmap of the two agents' learned policies after 10000 timesteps.

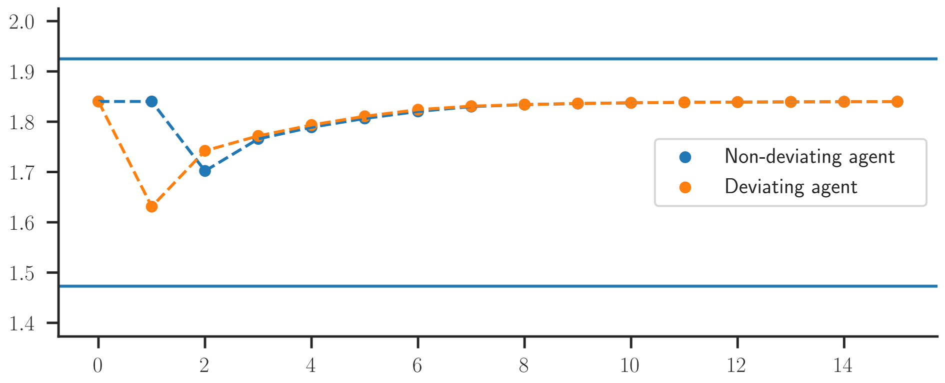


Figure 5.6: Response to a unilateral deviation to the static best response to the rival's last price after 40000 timesteps.

the path of equilibrium, I plot agents' responses to deviations and see how they evolve during learning.

The figures and tables that follow have been constructed by letting agents play with exploration and learning disabled for 1000 periods, then making one of the two agents defect to the static best-response to the last price played by the rival. Section 5.3 shows this impulse response.

We can see that after 40000 timesteps, a reward-and-punishment scheme emerges that is harsh enough to effectively deter deviation in the majority of cases, as shown by table 5.1. Moreover, as shown by section 5.3, this scheme emerges very early during learning, even though punishments are not harsh enough.

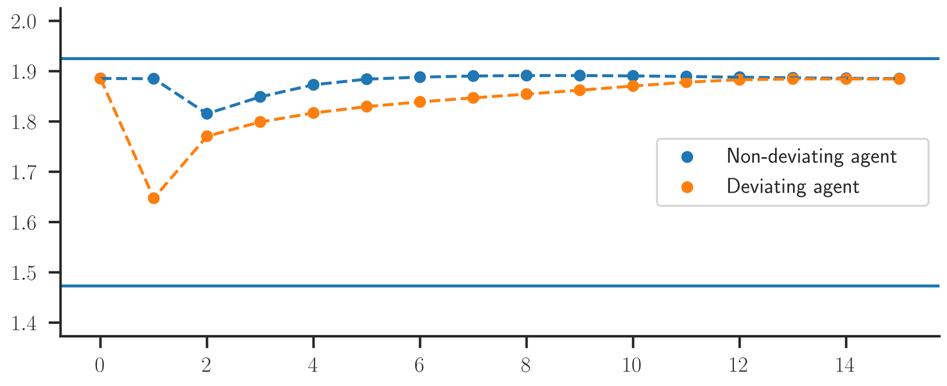


Figure 5.7: Response to a unilateral deviation to the static best response to the rival's last price after 10000 timesteps.

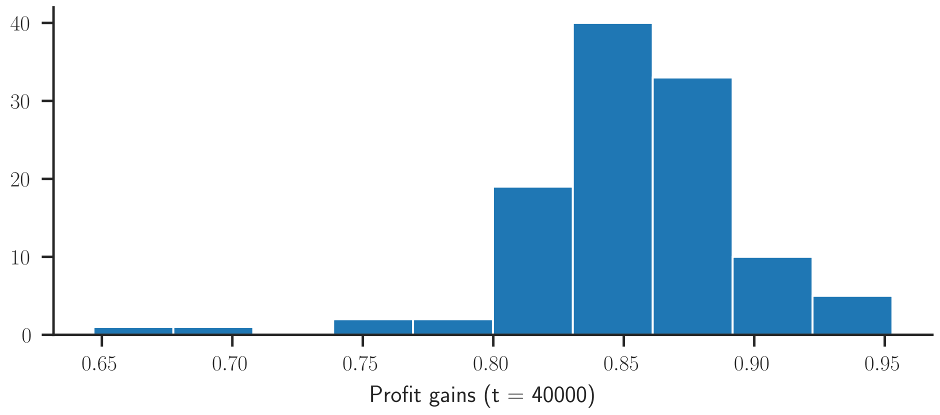


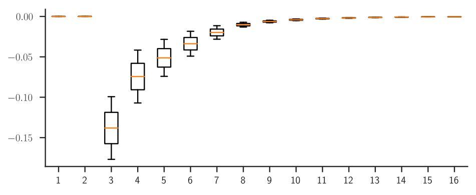
Figure 5.8: Distribution of mean profit gains after 40000 timesteps.

5.4 Performance across random seeds

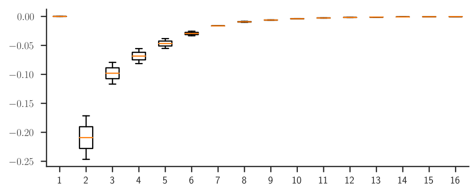
Usually, deep reinforcement learning algorithms tend to exhibit notable brittleness and high variance, which is usually masked by averaging across random seeds but emerges when the standard deviation is visualized. To understand the degree of brittleness, as well as what and how algorithms are learning, we can look at the distribution of profit gains, deviation behavior, and deviation profits.

Section 5.4 shows that indeed, in the vast majority of sessions profit gains are over 80%, with very few outliers and no session seeing profit gains below 65%. Keep in mind that, with the price range agents are allowed to employ, random pricing leads to 50% profit gains.

To discern whether the averaging of impulse responses is masking more complex



(a) Non-deviating agent



(b) Deviating agent

Figure 5.9: Box plot of the prices played during and after a deviation from the equilibrium reached after 40000 timesteps.

behavior, we can inspect the distribution of the difference between the price played just before the deviation and the price played in the subsequent timesteps, shown in fig. 5.9.

These box plots confirm that the reward-punishment scheme is actually emerging on the vast majority of runs and is not just a result of averaging chaotic impulse responses. They also show that reward and punishment are much more stable and with lower variance than what is obtained by tabular Q-learning. Prices rarely surpass the starting equilibrium level and exhibit much lower variance overall; when they do surpass the equilibrium level, the difference is slim.

Analyzing the variance of prices if exploration is turned off also brings to light long-term behavior during learning. Variance is slightly higher overall towards the beginning of training (after 10000 timesteps) which is expected. Variance is also higher towards the end (after 70000 timesteps), which may be an early symptom of catastrophic forgetting - again, an expected behavior since there is no learning rate decay. However, this forgetting is not so catastrophic after all: it only results in a slight deviation from the equilibrium price right before the deviation, and an imperfect return to cooperation afterward, that nevertheless does not impact collusion in a quantitatively relevant manner.

5.5 Discounted and differential return on deviation

Finally, we can look at the distribution of discounted and differential profits originating from a deviation, shown by the histogram in fig. 5.10 and fig. 5.11 I also report mean values and quartiles for profit gains and shares of profitable deviations in the discounted and differential case in table 5.1.

To enable comparisons with the tabular Q-learning of Calvano et al. (2020), I use a discount factor $\gamma = 0.95$ in evaluating discounted profit. However, this is an entirely exogenous choice and clearly more deviations become profitable when the discount factor falls since learning does not take into account the discount factor as it does

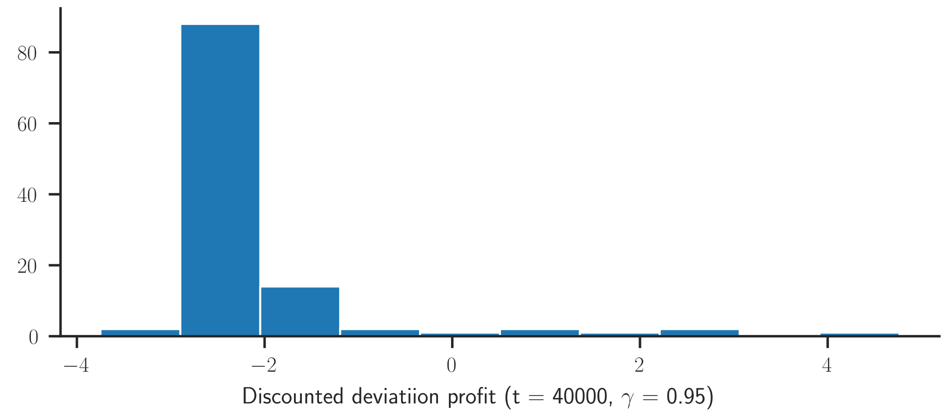


Figure 5.10: Distribution of discounted deviation profits after 40000 timesteps.

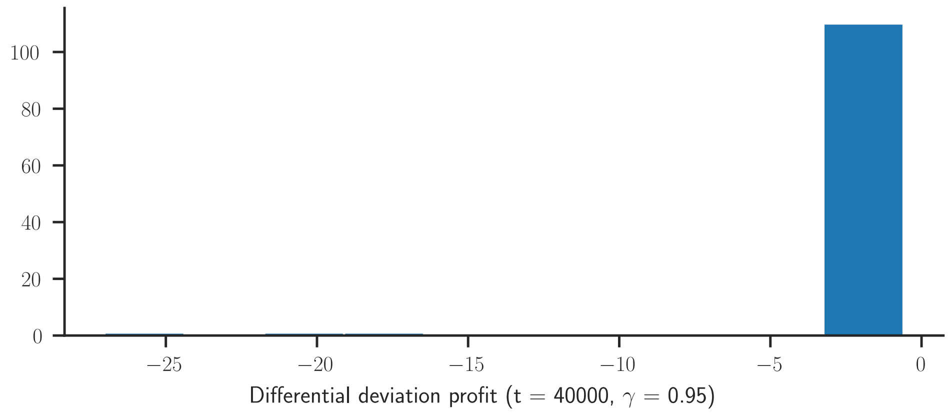


Figure 5.11: Distribution of differential deviation profits after 40000 timesteps.

in Calvano et al. (2020). In the limit where $\gamma = 0$, any deviation is regarded as profitable, so no collusion is possible; when instead $\gamma = 1$, discounted profits are analogous to differential profits since the quantity $R_t - r(\pi)$ becomes 0 as $t \to \infty$ if policy estimation converged and exploration is disabled.

Looking at discounted profits is particularly interesting since, as far as the agent is concerned, future payoffs are *not* discounted in the average reward formulation. As Naik et al. (2019) show, it is not a well-defined optimization problem to maximize discounted payoffs under function approximation. For this reason, evaluating average-reward reinforcement learning agents based on how they minimize discounted loss from a deviation is not entirely fair, as they were not optimized for this objective. Remarkably, they perform well nevertheless, with about 80% of the deviations being unprofitable in the discounted case and the median deviation being unprofitable after only 40000 timesteps. As shown by fig. 5.10, the distribution of deviation gains exhibits a large mass near 0 profits (most of it on the left side), but a few outliers are enough to make the mean a very unreliable statistic.

5.5.1 Different deviations

The static best-response is not the only deviation possible. In principle, agents could deviate to any other price, and all such deviations should be punished by a truly collusive strategy. Some examples of deviations are: deviation to monopoly price, an upwards deviation that disrupts equilibrium and should therefore be punished, however "advantageous" for the other agent; deviation to marginal cost, similar to the optimal punishment in Abreu (1986); deviation to the Nash equilibrium price, the harshest reasonable punishment in a Bertrand setting. Responses to these deviations are reported in table 5.2, with the static BR reported for comparison.

5.6 Chaotic sessions

To give a more quantitative and applicable definition of convergence, I employ the usual definition of "outliers" in profit gains and deviation profits as sessions whose value is outside the range $[Q1 - \frac{3}{2}IQR, Q3 + \frac{3}{2}IQR]$, where $Q1, Q3$ are the 25th and 75th percentile and IQR is the interquartile range. These are sessions that exhibit chaotic behavior, wildly fluctuating prices, and so on. I show the fraction of these sessions, as well as the condition of the 75th percentile, median and 25th percentile in table 5.3.

We can see that the percentage of these values falls under 1% for profit gains and under 5% for deviation profits after around 40000 timesteps, while from table 5.1 we can see that this amount of timesteps is sufficient to make at least 75% of deviations unprofitable from the differential point of view, while 10000 more timesteps are needed to get the same number unprofitable discounted deviations. It is worth noting, however, that excluding these outliers from the analysis of deviation profits makes the central mass of the distribution more apparent and moves the mean significantly closer to the median, and that these outliers usually lie on the right side of the distribution and not the left. Therefore, the mean profit obtained by deviating is an unreliable statistic that is skewed towards profitable deviations. This is expected behavior, since forcing one agent to respond optimally in just one period brings a substantial improvement in a chaotic session with heavily suboptimal pricing by both agents. Therefore, 40000 timesteps may be regarded as a reasonable threshold to define a convergent session: policy gradient methods, in this context, converge to collusive outcomes two orders of magnitude faster than the tabular Q-learning described in Calvano et al. (2020). If a timestep lasts one hour in real life, we get a timescale of collusion that is about a year long, which matches results in the empirical literature such as Byrne and de Roos (2019); Assad et al. (2020).

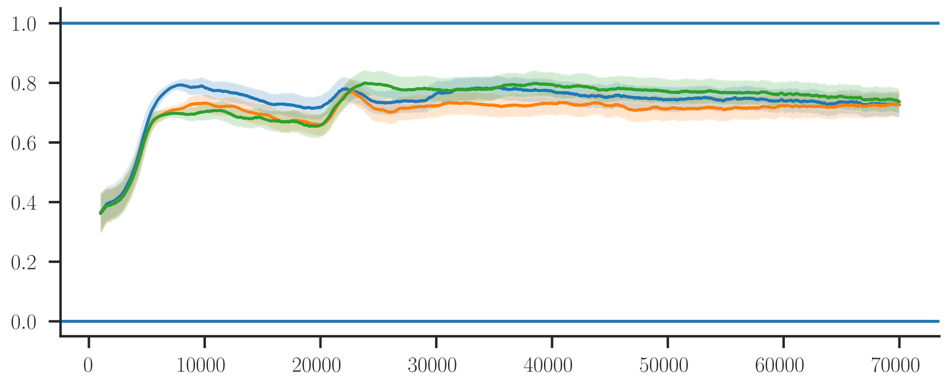


Figure 5.12: Profit gains over time in a game with three agents. Shaded area represents standard deviation.

5.7 Number of agents

The competition model at hand leads to the conclusion that markets with more firms are harder to collude in, because collusion profits are lower and therefore it is more tempting to deviate as punishment is weaker. Indeed, profit gain decreases to about 75% when three agents are competing, as shown by fig. 5.12. This result is consistent with what is obtained by tabular Q-learning but requires no adjustments to the exploration strategy.

However, the general behavior remains the same, albeit with slower learning. Reward-punishment schemes take slightly more time to emerge and there is a higher variance in impulse responses, as shown by figures 5.13 and 5.14. Moreover, a larger fraction of deviations is profitable, although the other results lead to believe that this is merely an artifact of slower learning and as learning progresses, more would become unprofitable.

With more than three agents, e.g. $n = 4$, profit gains are lower (mean profit gains of about 70%) and learning is slower still, with reward-punishment schemes emerging but taking an even longer time to make the majority of deviations to the static best-response unprofitable.

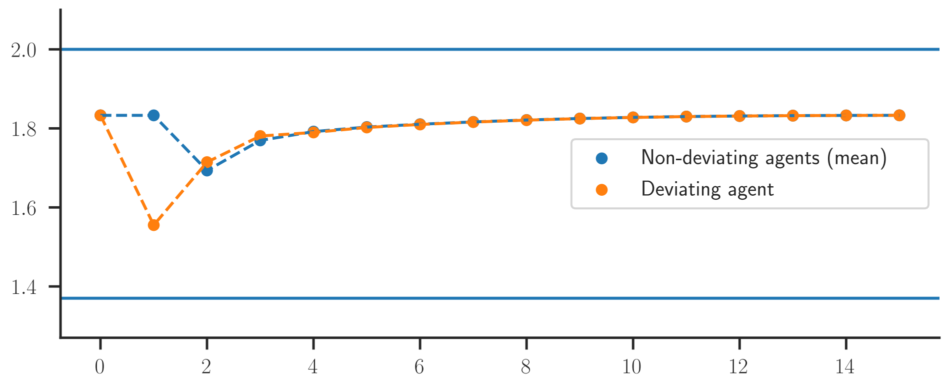
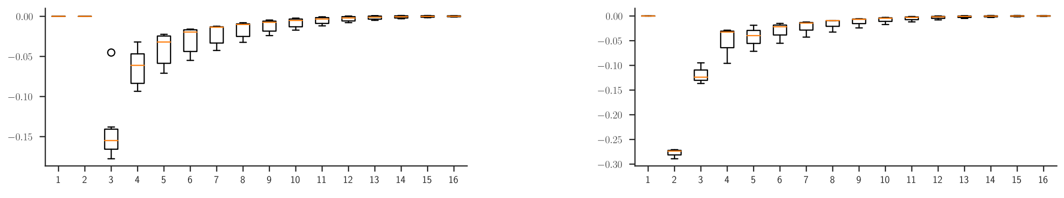


Figure 5.13: Responses to a unilateral deviation to the static best response to the rivals' last prices, with 3 agents, after 70000 timesteps. For non-deviating agents, I show mean prices.



(a) Non-deviating agents (mean) (b) Deviating agent

Figure 5.14: Box plot of the prices played during and after a deviation from the equilibrium reached after 70000 timesteps, with 3 agents.

Table 5.1: Responses to deviation during learning.

Statistic	Mean	Pctl(75)	Median	Pctl(25)
t = 10000				
Deviation gain (discounted, %)	4.697	5.434	2.180	1.113
Deviation gain (differential)	7.046	0.425	0.176	0.074
Unprofitable deviation (differential)	0.035	0	0	0
Unprofitable deviation (discounted)	0.009	0	0	0
t = 20000				
Deviation gain (discounted, %)	1.233	1.468	0.523	0.106
Deviation gain (differential)	1.147	0.114	0.021	−0.010
Unprofitable deviation (differential)	0.336	1	0	0
Unprofitable deviation (discounted)	0.186	0	0	0
t = 30000				
Deviation gain (discounted, %)	0.278	0.393	−0.069	−0.294
Deviation gain (differential)	0.593	0.014	−0.018	−0.043
Unprofitable deviation (differential)	0.664	1	1	0
Unprofitable deviation (discounted)	0.540	1	1	0
t = 40000				
Deviation gain (discounted, %)	0.277	0.132	−0.116	−0.302
Deviation gain (differential)	−0.215	−0.000	−0.024	−0.044
Unprofitable deviation (differential)	0.752	1	1	1
Unprofitable deviation (discounted)	0.637	1	1	0
t = 50000				
Deviation gain (discounted, %)	0.448	−0.023	−0.208	−0.396
Deviation gain (differential)	0.601	−0.017	−0.043	−0.066
Unprofitable deviation (differential)	0.876	1	1	1
Unprofitable deviation (discounted)	0.788	1	1	1
t = 60000				
Deviation gain (discounted, %)	−0.019	−0.077	−0.246	−0.428
Deviation gain (differential)	−0.550	−0.024	−0.052	−0.077
Unprofitable deviation (differential)	0.903	1	1	1
Unprofitable deviation (discounted)	0.814	1	1	1
t = 70000				
Deviation gain (discounted, %)	0.311	−0.063	−0.255	−0.390
Deviation gain (differential)	−0.257	−0.032	−0.053	−0.077
Unprofitable deviation (differential)	0.867	1	1	1
Unprofitable deviation (discounted)	0.832	1	1	1

Table 5.2: Responses to various deviations.

Statistic	Mean	Pctl(75)	Median	Pctl(25)
Deviation type = Static BR				
Deviation gain (discounted, %)	0.311	-0.063	-0.255	-0.39
Deviation gain (differential)	-0.257	-0.032	-0.053	-0.077
Unprofitable deviation (differiential)	0.867	1	1	1
Unprofitable deviation (discounted)	0.832	1	1	1
Deviation type = Bertrand-Nash				
Deviation gain (discounted, %)	-0.591	-0.582	-0.695	-0.904
Deviation gain (differential)	-1.302	-0.068	-0.087	-0.119
Unprofitable deviation (differiential)	0.92	1	1	1
Unprofitable deviation (discounted)	0.929	1	1	1
Deviation type = Marginal cost				
Deviation gain (discounted, %)	-5.863	-5.934	-6.178	-6.508
Deviation gain (differential)	-0.734	-0.426	-0.454	-0.489
Unprofitable deviation (differiential)	0.965	1	1	1
Unprofitable deviation (discounted)	0.982	1	1	1
Deviation type = Monopoly				
Deviation gain (discounted, %)	-0.66	-0.267	-0.676	-1.258
Deviation gain (differential)	-0.66	-0.042	-0.073	-0.119
Unprofitable deviation (differiential)	0.858	1	1	1
Unprofitable deviation (discounted)	0.85	1	1	1

Table 5.3: Share of chaotic sessions during learning.

Statistic	Mean
t = 10000	
Outlier in profit gains	0.106
Outlier in deviation profits (differential)	0.201
Outlier in deviation profits (discounted)	0.162
t = 20000	
Outlier in profit gains	0.142
Outlier in deviation profits (differential)	0.228
Outlier in deviation profits (discounted)	0.053
t = 30000	
Outlier in profit gains	0.053
Outlier in deviation profits (differential)	0.111
Outlier in deviation profits (discounted)	0.031
t = 40000	
Outlier in profit gains	0.009
Outlier in deviation profits (differential)	0.024
Outlier in deviation profits (discounted)	0.022
t = 50000	
Outlier in profit gains	0.009
Outlier in deviation profits (differential)	0.06
Outlier in deviation profits (discounted)	0.02
t = 60000	
Outlier in profit gains	0.009
Outlier in deviation profits (differential)	0.042
Outlier in deviation profits (discounted)	0.033
t = 70000	
Outlier in profit gains	0.009
Outlier in deviation profits (differential)	0.058
Outlier in deviation profits (discounted)	0.018

Chapter 6

Conclusions and future work

In this work, I show that policy gradient maximum-entropy reinforcement learning algorithms using neural networks as function approximators may autonomously learn collusive strategies in a repeated pricing game significantly faster than tabular strategies. Agents begin to raise prices as soon as they are allowed to, a behavior that is not exhibited by tabular Q-learning. Learned policies are able to sustain collusion through reward-punishment schemes that make it unprofitable to deviate from the collusive equilibrium.

From an engineering perspective, I conclude with potential directions for future research in the field of reinforcement learning applied to multi-agent continuous-state/action repeated games.

The basic formulation even of complex reinforcement learning algorithms often leaves room for tweaks that improve performance, within the same class of algorithms. Examples of such enhancements are Munchausen reinforcement learning (Vieillard et al., 2020), improved experience replay buffers (Wang and Ross, 2019), and deeper networks (Sinha et al., 2020). In addition, there is a strand of literature that is concerned precisely with the applications of reinforcement learning in non-stationary (Xie et al., 2022; Paternain et al., 2020) and competitive and cooperative environments (Kim et al., 2021; Yang and Wang, 2021). Integrating newer developments in entropy-regularized actor-critic methods or introducing more complexity and information in the game by employing algorithms that are aware of the non-stationarity of agents' policies may be a promising direction for future research on model-free pricing algorithms.

In addition, it is worth noting that, as with tabular temporal-difference methods, there are no known proofs of convergence for policy-gradient methods in non-stationary environments, except for a handful of special cases. However, as the pricing game is relatively simple, it may constitute a special case where reinforcement learning can be proven to converge, and a deeper theoretical analysis of this context is an interesting direction for future research that may be of interest to both computer scientists and economists.

From an economic perspective, instead, I conclude by noting that reinforcement learning algorithms are, in general and as of the time of writing, suboptimal choices for control problems, especially non-stationary ones, compared to model-based optimal control.

Research in the field is often based on flawed assumptions (Nota and Thomas, 2020; Naik et al., 2019) and most problems that are studied in the literature are non-Markovian, or at least non-first-order-Markovian, just like the one I studied. When a model of the environment is available, trivial random search performs better than policy gradient methods, which can be shown to be equivalent to random search in this context (Mania et al., 2018). Therefore, the fact that reinforcement learning algorithms are able to reach collusive agreements quickly is not to be taken to mean that firms may collude by actually using reinforcement learning. On the contrary, firms that employ pricing algorithms are probably *not* using reinforcement learning precisely because better algorithms are available that may achieve even higher profits in a shorter amount of time, for example by estimating demand functions and imposing the desired level of profit gains over the competitive outcome.

The fact that deep reinforcement learning is effective here works to show that price competition is a relatively simple control task. Even brittle algorithms that have a very limited set of information are able to solve it quickly in a cooperative manner without being swayed towards collusion. Therefore, regulation limiting algorithmic complexity or information availability may be argued to not be on the right track to actually limit collusion.

Usually, algorithms that achieve sample-efficient control are model-based and use regret minimization, tree search, and similar techniques that require complete information. These algorithms are able to converge in a handful of timesteps, and some have also been applied to games of cooperation. Some examples are Crandall et al. (2018); Crandall (2014); Moravčík et al. (2017). Developing similar algorithms to be applied to the pricing game is an interesting direction for future work. In addition, once such studies are available, another interesting direction is to compare the behavior of model-free and model-based algorithms with panel data on prices, to understand which kind of technology is actually deployed when firms employ algorithmic pricing and orient future research in competition policy.

Bibliography

Abreu, Dilip, “Extremal equilibria of oligopolistic supergames,” *Journal of Economic Theory*, June 1986, *39* (1), 191–225.

Akiba, Takuya, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework,” Technical Report arXiv:1907.10902, arXiv July 2019. arXiv:1907.10902 [cs, stat] type: article.

Anderson, Simon P., Andre de Palma, and Jacques-Francois Thisse, *Discrete Choice Theory of Product Differentiation*, MIT Press, October 1992.

Andrychowicz, Marcin, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem, “What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study,” June 2020. Number: arXiv:2006.05990 arXiv:2006.05990 [cs, stat].

Asker, John, Chaim Fershtman, and Ariel Pakes, “Artificial Intelligence and Pricing: The Impact of Algorithm Design,” March 2021.

Assad, Stephanie, Robert Clark, Daniel Ershov, and Lei Xu, “Algorithmic Pricing and Competition: Empirical Evidence from the German Retail Gasoline Market,” 2020.

Bergstra, James and Yoshua Bengio, “Random search for hyper-parameter optimization,” *The Journal of Machine Learning Research*, February 2012, *13* (null), 281–305.

Bertsekas, Dimitri P., *Dynamic programming: deterministic and stochastic models*, USA: Prentice-Hall, Inc., 1987.

_ , *Dynamic programming and optimal control*, fourth edition ed., Belmont, Mass: Athena Scientific, 2012. OCLC: 863688177.

_ , *Reinforcement Learning and Optimal Control*, Athena Scientific, July 2019.

Byrne, David P. and Nicolas de Roos, "Learning to Coordinate: A Study in Retail Gasoline," *American Economic Review*, February 2019, *109* (2), 591–619.

Calvano, Emilio, Giacomo Calzolari, Vincenzo Denicolò, and Sergio Pastorello, "Artificial Intelligence, Algorithmic Pricing, and Collusion," *American Economic Review*, October 2020, *110* (10), 3267–3297.

Chen, Le, A. Mislove, and Christo Wilson, "An Empirical Analysis of Algorithmic Pricing on Amazon Marketplace," *WWW*, 2016.

Crandall, Jacob W, "Towards minimizing disappointment in repeated games," *Journal of Artificial Intelligence Research*, 2014, *49*, 111–142.

_ , Mayada Oudah, Fatimah Ishowo-Oloko, Sherief Abdallah, Jean-François Bonnefon, Manuel Cebrian, Azim Shariff, Michael A Goodrich, Iyad Rahwan, and others, "Cooperating with machines," *Nature communications*, 2018, *9* (1), 1–12. Publisher: Nature Publishing Group.

Degris, Thomas, Martha White, and Richard S. Sutton, "Linear Off-Policy Actor-Critic," in "ICML" July 2019.

Fujimoto, Scott, Herke Hoof, and David Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in "Proceedings of the 35th International Conference on Machine Learning" PMLR July 2018, pp. 1587–1596. ISSN: 2640-3498.

Furlan, Massimiliano, "Algorithmic Collusion with Coarse Memory," Master's thesis, Bologna University, Bologna, Italy July 2022.

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine, "Soft Actor-Critic Algorithms and Applications.," *CoRR*, 2018, *abs/1812.05905.*

_ , _ , Pieter Abbeel, and Sergey Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in "Proceedings of the 35th International Conference on Machine Learning" PMLR July 2018, pp. 1861–1870. ISSN: 2640-3498.

Hessel, Matteo, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning," in "Proceedings of the Thirty-Second AAAI Conference on Artificial

Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence" AAAI'18/IAAI'18/EAAI'18 AAAI Press 2018. event-place: New Orleans, Louisiana, USA.

Hettich, Matthias, "Algorithmic Collusion: Insights from Deep Learning," SSRN Scholarly Paper 3785966, Social Science Research Network, Rochester, NY November 2021.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White, "Multilayer feedforward networks are universal approximators," *Neural networks*, 1989, *2* (5), 359–366. Publisher: Elsevier.

Karandikar, Rajeeva, Dilip Mookherjee, Debraj Ray, and Fernando Vega-Redondo, "Evolving Aspirations and Cooperation," *Journal of Economic Theory*, June 1998, *80* (2), 292–331.

Kim, Dong Ki, Miao Liu, Matthew D. Riemer, Chuangchuang Sun, Marwa Abdulhai, Golnaz Habibi, Sebastian Lopez-Cot, Gerald Tesauro, and Jonathan How, "A Policy Gradient Algorithm for Learning to Learn in Multiagent Reinforcement Learning," in "Proceedings of the 38th International Conference on Machine Learning" PMLR July 2021, pp. 5541–5550. ISSN: 2640-3498.

Kingma, D.P., Ba, L.J., and Amsterdam Machine Learning lab (IVI, FNWI), "Adam: A Method for Stochastic Optimization," in "International Conference on Learning Representations (ICLR)" arXiv.org 2015.

Klein, Timo, "Autonomous algorithmic collusion: Q-learning under sequential pricing," *The RAND Journal of Economics*, 2021, *52* (3), 538–558. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1756-2171.12383.

Kuindersma, Scott, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *MIT web domain*, July 2015. Accepted: 2017-07-07T15:29:29Z Publisher: Springer-Verlag.

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, "Continuous control with deep reinforcement learning.," in "ICLR (Poster)" January 2021.

Lin, Long-Ji, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, May 1992, *8* (3), 293–321.

Littman, Michael L. and Peter Stone, "Leading Best-Response Strategies in Repeated Games," in "In Seventeenth Annual International Joint Conference on Artificial Intelligence Workshop on Economic Agents, Models, and Mechanisms" 2001.

Mania, Horia, Aurelia Guy, and Benjamin Recht, "Simple random search of static linear policies is competitive for reinforcement learning," in "Advances in Neural Information Processing Systems," Vol. 31 Curran Associates, Inc. 2018.

Maskin, Eric and Jean Tirole, "A Theory of Dynamic Oligopoly, II: Price Competition, Kinked Demand Curves, and Edgeworth Cycles," *Econometrica*, 1988, *56* (3), 571–599. Publisher: [Wiley, Econometric Society].

Mnih, Volodymyr, 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, "Human-level control through deep reinforcement learning," *Nature*, February 2015, *518* (7540), 529–533. Number: 7540 Publisher: Nature Publishing Group.

Moravčík, Matej, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling, "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, May 2017, *356* (6337), 508–513. Publisher: American Association for the Advancement of Science.

Naik, Abhishek, Roshan Shariff, Niko Yasui, and Richard S. Sutton, "Discounted Reinforcement Learning is Not an Optimization Problem," 2019. arXiv: 1910.02140.

Nota, Chris and P. Thomas, "Is the Policy Gradient a Gradient?," in "AAMAS" 2020.

Paternain, Santiago, Juan Andres Bazerque, and Alejandro Ribeiro, "Policy Gradient for Continuing Tasks in Non-stationary Markov Decision Processes," Technical Report arXiv:2010.08443, arXiv October 2020. arXiv:2010.08443 [cs, eess] type: article.

Rao, Ashwin and Tikhon Jelvis, *Foundations of Reinforcement Learning with Applications in Finance*, Unpublished, 2022.

Rummery, G. and Mahesan Niranjan, "On-Line Q-Learning Using Connectionist Systems," *Technical Report CUED/F-INFENG/TR 166*, November 1994.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, "Proximal Policy Optimization Algorithms," August 2017. arXiv:1707.06347 [cs].

_ **, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz**, "Trust Region Policy Optimization," in "Proceedings of the 32nd International Conference on Machine Learning" PMLR June 2015, pp. 1889–1897. ISSN: 1938-7228.

Schwalbe, Ulrich, "Algorithms, Machine Learning and Collusion," *Journal of Competition Law & Economics*, December 2018, *14* (4), 568–607.

Siallagan, Manahan, Hiroshi Deguchi, and Manabu Ichikawa, "Aspiration-Based Learning in a Cournot Duopoly Model," *Evolutionary and Institutional Economics Review*, December 2013, *10* (2), 295–314.

Sinha, Samarth, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg, "D2RL: Deep Dense Architectures in Reinforcement Learning," November 2020. arXiv:2010.09163 [cs].

Sutton, Richard S., "Learning to predict by the methods of temporal differences," *Machine Learning*, August 1988, *3* (1), 9–44.

_ **and Andrew G. Barto**, *Reinforcement Learning: An Introduction*, MIT Press, November 2018. Google-Books-ID: uWV0DwAAQBAJ.

_ **, David McAllester, Satinder Singh, and Yishay Mansour**, "Policy gradient methods for reinforcement learning with function approximation," in "Proceedings of the 12th International Conference on Neural Information Processing Systems" NIPS'99 MIT Press Cambridge, MA, USA November 1999, pp. 1057–1063.

Szepesvári, Csaba, *Algorithms for Reinforcement Learning*, Morgan & Claypool Publishers, 2010.

Tadelis, Steven, *Game Theory: An Introduction*, Princeton University Press, January 2013. Google-Books-ID: _4OqAaITAWMC.

Tassa, Yuval, Tom Erez, and Emanuel Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in "2012 IEEE/RSJ International Conference on Intelligent Robots and Systems" October 2012, pp. 4906–4913. ISSN: 2153-0866.

van Hasselt, Hado, "Double Q-learning," in "Advances in Neural Information Processing Systems," Vol. 23 Curran Associates, Inc. 2010.

_ , "Reinforcement Learning in Continuous State and Action Spaces," in Marco Wiering and Martijn van Otterlo, eds., *Reinforcement Learning: State-of-the-Art*, Adaptation, Learning, and Optimization, Berlin, Heidelberg: Springer, 2012, pp. 207–251.

_ , Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil, "Deep Reinforcement Learning and the Deadly Triad," *CoRR*, 2018, *abs/1812.02648.*

Vieillard, Nino, Olivier Pietquin, and Matthieu Geist, "Munchausen Reinforcement Learning," in "Proceedings of the 34th International Conference on Neural Information Processing Systems" NIPS'20 Curran Associates Inc. Red Hook, NY, USA 2020. event-place: Vancouver, BC, Canada.

Wang, Che and Keith Ross, "Boosting Soft Actor-Critic: Emphasizing Recent Experience without Forgetting the Past," *CoRR*, 2019, *abs/1906.04009.*

Watkins, Christopher, "Learning From Delayed Rewards." PhD dissertation, University of Cambridge, Cambridge January 1989.

Williams, Ronald J., "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, May 1992, *8* (3), 229–256.

Xie, Annie, James Harrison, and Chelsea Finn, "Deep Reinforcement Learning amidst Lifelong Non-Stationarity," in "4th Lifelong Machine Learning Workshop at ICML 2020" June 2022.

Yamada, Hiroyuki, "cpprb," January 2019.

Yang, Greg, Jascha Sohl-dickstein, Jeffrey Pennington, Sam Schoenholz, and Vinay Rao, "A Mean Field Theory of Batch Normalization," in "Google Research" 2019.

Yang, Yaodong and Jun Wang, "An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective," March 2021. arXiv: 2011.00583.

Yu, Haonan, Haichao Zhang, and Wei Xu, "Do You Need the Entropy Reward (in Practice)?," January 2022. Number: arXiv:2201.12434 arXiv:2201.12434 [cs] version: 1.

Appendix A

Formulation of the policy gradient theorem

The formulation of the policy gradient theorem used by most of the literature is derived from eq. (2.21) by noting that, in a *stationary* Markov decision process, the steady-state distribution μ_π is induced by the policy π. Dropping the explicit dependence of π on $\boldsymbol{\theta}$ for clarity, we have that

$$\int_{\mathcal{S}} \mu_\pi(s) f(s) ds = \int_{\mathcal{S}} T(s'|s,a) \int_{\mathcal{A}} \pi(a|s) f(s') da ds' = \mathbb{E}_\pi[f(S_t)] \tag{A.1}$$

and we can therefore express the integral as an expected value on the random variable S_t, as in

$$\nabla J(\boldsymbol{\theta}) = \int_{\mathcal{S}} \mu_\pi(s) \int_{\mathcal{A}} q_\pi(s,a) \nabla\pi(a|s) da\, ds = \mathbb{E}_\pi \left[\int_{\mathcal{A}} q_\pi(S_t,a) \nabla\pi(a|S_t) da \right] \tag{A.2}$$

Then we can multiply and divide the equation by $\pi(a|S_t)$ to get

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[\int_{\mathcal{A}} q_\pi(S_t,a) \pi(a|S_t) \frac{\nabla\pi(a|S_t)}{\pi(a|S_t)} da \right] \tag{A.3}$$

This integral now also represents an expected value weighted by π:

$$\int_{\mathcal{A}} q_\pi(S_t,a) \pi(a|S_t) \frac{\nabla\pi(a|S_t)}{\pi(a|S_t)} da = \mathbb{E}_\pi \left[q_\pi(S_t,A_t) \frac{\nabla\pi(A_t|S_t)}{\pi(A_t|S_t)} \right] \tag{A.4}$$

Therefore we can write it out as an expectation, drop the double expected value and, since since $\nabla x/x = \nabla \log x$, we can write:

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t)}{\pi(A_t|S_t)} \right] = \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \nabla \log \pi(A_t|S_t) \right] \tag{A.5}$$

Appendix B

Hyperparameters for the representative experiment

Symbol	**Hyperparameter**	**Value**
$\bar{\mathcal{H}}$	Target entropy	1
M	Hidden layer size (critic network)	2048
B	Batch size	512
β	Replay buffer size	20000
λ_A	Actor optimizer learning rate	$1 \cdot 10^{-3}$
λ_C	Critic optimizer learning rate	$3 \cdot 10^{-5}$
λ_R	Average reward estimation learning rate	0.03

Table B.1: Hyperparameters of the representative experiment

.