CQE Center for Quantitative Economics

Algorithmic Collusion: Insights from Deep Learning

Matthias Hettich [†]

94/2021

[†] Einstein Center Digital Future, TU Berlin, Germany

wissen.leben
WWU Münster

Algorithmic Collusion: Insights from Deep Learning

Matthias Hettich
Einstein Center Digital Future
TU Berlin, Germany
matthias.hettich@tu-berlin.de

November 24, 2021

Abstract

Increasingly, firms use algorithms powered by artificial intelligence to set prices. Previous research simulated interactions among Q-learning algorithms in an oligopoly model of price competition. The algorithms learn collusive strategies but require a long time that corresponds to several years to do so. We show that pricing algorithms using deep learning (DQN) can collude significantly faster. The availability of these more powerful pricing algorithms enables simulations in larger markets. Collusion disappears in wide oligopolies with up to 10 firms. However, incorporating knowledge of the learning behavior by reformulating the state representation increases the ability to collude effectively.

Keywords Algorithmic Pricing, Collusion, Artificial Intelligence, Reinforcement Learning, DQN

JEL Classification D21, D43, D83, L12, L13

1 Introduction

Firms increasingly deploy algorithms to set prices for their products and services. Enhancements in machine learning and artificial intelligence enable algorithms to learn pricing strategies without any previous knowledge by active experimentation. This autonomous nature fuels concerns that the algorithms of competing firms learn to collude without communicating with each other and without any instruction to do so (Mehra 2016; Ezrachi and Stucke 2016, 2017). Current antitrust regulation focuses on communication leading to a conscious commitment, rather than collusion as such. Thus, the current regulatory approach becomes insufficient when algorithms learn collusive strategies by themselves and without communication (Harrington Jr. 2019; Calvano et al. 2020a).

The risk of collusion by algorithms is hard to assess empirically, except for cases where rich market data is available (e.g. Assad et al. 2020). Thus, current research focuses mainly on theoretical and experimental approaches. First theoretical studies yield interesting insights (e.g. Salcedo 2015; Brown and MacKay 2021; Harrington Jr. 2020; Hansen et al. 2021), but often rely on simplifying assumptions because interactions among firms in pricing games generate complex stochastic dynamic systems. Experimental approaches observe the complex interaction of pricing algorithms in computer-simulated marketplaces. By analyzing a repeated Bertrand competition, Calvano et al. (2020b) provide the most comprehensive analysis so far. Not only do they find that algorithms systematically set supra-competitive prices, but they verify that the algorithms facilitate collusion with reward-punishment

schemes.[1] Based on this fundamental work, further studies investigate the influence of imperfect monitoring (Calvano et al. 2021), platform design (Johnson et al. 2020) and how markets could be disciplined to hinder collusion (Abada and Lambin 2020).

Previous simulations model the pricing algorithms with Q-learning.[2] Even though Q-learning is suited to provide a proof-of-concept of algorithmic collusion, it is only partly representative of algorithms used in practice. The inability to use powerful approximation methods makes Q-learning too slow to be applied in real-world scenarios.[3] Additionally, Q-learning's limited computational resources restrict the previous studies to markets with two to four firms.

This paper contributes to the experimental research branch by approaching a more realistic pricing algorithm. We demonstrate how deep learning can be incorporated in pricing algorithms, thereby paving the way for further research on even more realistic algorithms or market environments. We let Deep Q-Network (DQN) algorithms compete against each other in markets with up to 10 firms. This algorithm is the straightforward enhancement of Q-learning to function approximation with the help of Deep Neuronal Networks (DNNs). The simulated marketplace builds on the repeated Bertrand competition of Calvano et al. (2020b) which can go on infinitely. DQN is designed to tackle episodic tasks and, thus, we restated DQN's optimization problem with an average reward formulation, as proposed by Sutton and Barto (2018).

Our results indicate that DQNs consistently learn collusive strategies in a duopoly. They learn significantly faster than previously used algorithms and start to increase prices after 20,000 time-steps. With an increasing number of market participants, the level of collusion decreases, and with seven or more firms, at the latest, collusion entirely disappears. However, incorporating knowledge about the algorithm's learning behavior by reformulating the state representation seems to make collusion even in wide oligopolies possible.

The paper continues with a description of the simulated economic environment. Section 3 explains the pricing algorithms. The simple Q-learning illustrates the concept of Reinforcement Learning (RL) and makes the understanding of the more complex DQN easily possible. Section 4 demonstrates the learning pace of DQN in a duopoly. Section 5 focuses on wider oligopolies and presents two interesting extensions. The paper concludes with a short discussion of the findings.

2 Economic Environment

Following Calvano et al. (2020b), we model the oligopolistic competition between pricing algorithms by a repeated Bertrand model. Each firm i, $i = 1, 2, .., n$, uses an independent pricing algorithm and produces a single product with quality g_i and marginal costs c_i. We assume no fixed costs. The Bertrand model describes interactions among firms that set price p_i for their product and consumers that choose quantities at this price, the demand d_i. This one-

1. Klein (2021) uses a market model of staggered prices and finds that self-learning pricing algorithms frequently set supra-competitive prices as well.

2. One exception is Meylahn and Boer (2021).

3. In the study of Calvano et al. (2020b), the Q-learning algorithms have, on average, 850,000 time-steps for learning. Even If we assume 30 updates per hour (the maximum update frequency of the Amazon (2020) API), this translates to more than three years.

shot game repeats each period. At the end of each period t, firm i earns profit $r_{i,t}(p_{j\in n,t}) \doteq (p_{i,t} - c_i) \times d_{i,t}(p_{j\in n,t})$. The well-known multinomial logit model describes the demand side of the simulated oligopolistic market.[4] In each period, the demand $d_{i,t}$ for firm i's product is:

$$d_{i,t}(p_{j\in n,t}) \doteq \frac{e^{\frac{g_i - p_{i,t}}{\mu}}}{1 + \sum_{j=1}^{n} e^{\frac{g_j - p_{j,t}}{\mu}}}. \tag{1}$$

Thus, a firm's profit and demand depends not only on the price p_i that the firm itself sets, but on the prices $p_{j\in n}$ that the competing firms set.

To compare the degree of collusion, we need a consistent measure across all simulations. In line with Calvano et al. (2020b), we use the average profit gain Δ_i, defined as:

$$\Delta_i \doteq \frac{\bar{r_i} - r_i^N}{r_i^M - r_i^N}, \tag{2}$$

where $\bar{r_i}$ denotes the average profit of firm i over a given number of periods. r_i^N is the profit of firm i in the static Bertrand-Nash equilibrium, and r_i^M is the monopoly profit. With collusion, firms can obtain supra-competitive profits per definition. The Nash profit r^N of the one-shot game is a reasonable estimation of the competitive outcome and the monopoly profit r^M normalizes the collusion measure between 0 and 1. Thus, for $\Delta = 0$, the average profit corresponds to the competitive outcome, and for $\Delta = 1$, to the outcome under full collusion (Calvano et al. 2020b).

To be computationally feasible, the pricing algorithms must choose from a discrete price range. A reasonable price range includes the static Nash equilibrium prices $p_{i\in n}^N$ and the monopoly prices $p_{i\in n}^M$ of the one shot-game. Then, the price range $\mathcal{A}$ is given by m equally spaced points in the interval $\mathcal{A} \doteq \left[\min(p_{i\in n}^N) - \xi, \max(p_{i\in n}^M) + \xi\right]$, where we set $\xi \doteq 0.1[\max(p_{i\in n}^M) - \min(p_{i\in n}^N)]$. This markup increases the price range by 10% in both directions.[5]

3 Pricing Algorithms

Repeated games, like the repeated Bertrand competition, can be modeled as a Markov Decision Process (MDP). In a MDP, the decision-maker, called the agent, interacts continually with the environment, that is, everything outside of the agent. In each period t, $t = 1, 2, ..., \infty$, the agent observes the environment's state, $S_t \in \mathcal{S}$. Based on the observed state, the agent selects an action, $A_t \in \mathcal{A}$. As a consequence of the action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and the system moves on to the next state, S_{t+1}, according to the state-transition probabilities F. They describe a probability distribution for each combination of $a \in \mathcal{A}$ and $s \in \mathcal{S}$:[6]

$$F(s', r|s, a) \doteq Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}. \tag{3}$$

4. Other demand models are conceivable. Calvano et al. (2020b) show that the choice of the demand model does not influence the collusive behavior of self-learning algorithms.

5. Please refer to Calvano et al. (2020b) for a more in-depth description of the economic environment.

6. Capital letters denote random variables, whereas lower case letters are used for values of random variables. s' is shorthand for the next period's state.

The agent's goal is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the present value of future rewards $G_t \doteq \sum_{k=0}^{\infty} \gamma^k \, R_{t+k+1}$, where γ is a discount factor, $0 \leq \gamma < 1$.[7] RL algorithms encode policies by estimating value v or action-value q functions that reflect how good it is for the agent to be in a given state or state-action pair.[8] The action-value function defines the value of taking action a in state s and then following a policy π:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \, R_{t+k+1} | S_t = s, A_t = a \right]. \tag{4}$$

As stated above, we try to find an optimal policy π^* that maximizes the agent's goal. Optimal policies share the same optimal action-value function that is defined as $q_*(s,a) \doteq \max_\pi q_\pi(s,a)$. We can rewrite the optimal action-value function without reference to a policy by using the Bellmann equation. The value of a state-action pair under an optimal policy equals the reward expected from transition to the next period plus the discounted action-value of the next state. The next state's action-value assumes that the agent will behave optimally, and again reflects all future rewards under an optimal policy. With this recursive property, the optimal action-value function becomes:

$$q_*(s,a) = \mathbb{E} \left[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_*(S_{t+1}, a') | S_t = s, A_t = a \right]. \tag{5}$$

The Bellmann optimality equation (5) reflects all long-term returns as a locally available value. If the optimal action-value function is known, an optimal policy π^* can easily be derived by $\pi^*(a \mid s) = \arg\max_a q_*(s,a)$. The optimal policy describes which action a to take in state s. The agent simply observes the current state s and chooses action a for which $q_*(s,a)$ is maximal (Sutton and Barto 2018).

If we apply the MDP framework to the repeated Bertrand competition, one independent agent represents each firm. The state of the environment at period t, S_t, are the prices that the agents, or firms, played in the previous period $t-1$. The set of possible actions $\mathcal{A}$ are the prices that an agent can set for her product, i.e., the price range. After each agent played her action A_t, the environment moves to the next state S_{t+1}. The environment determines the demand for the products, and each agent receives her reward signal R_{t+1}, i.e., the profit.

The Bellmann optimality equation(5) describes a system of $|\mathcal{S}| \times |\mathcal{A}|$ nonlinear equations, one for each state-action pair, in the same number of unknowns. If the state-transition probabilities (3) are known, standard methods can solve the system of equations. In the repeated Bertrand competition, the state-transition probabilities are unknown because the transition from one state to the other state depends not only on the own action but also on all other agents' actions.[9] Thus, the agents must estimate the optimal action-value function $q_*(s,a)$ by trial-and-error interactions with the environment, i.e., with RL. Even though RL algorithms are initially designed to tackle MDPs with a single agent and time-invariant state-transition probabilities, they can be applied to environments with multiple agents (Sutton and Barto 2018; Leibo et al. 2017).

7. The name policy originates from the RL literature's terminology. It describes the agent's way of acting and corresponds to the term strategy from the game theory domain. In the following, we will use both terms synonymously.

8. The action-value function is related to the value function by the identity $v(s) \doteq \max_{a \in \mathcal{A}} q(s,a)$.

9. Besides, the agents do not know the demand function.

3.1 Q-Learning

One of the simplest and best understood RL algorithms is Q-learning. It learns the Q-values $Q(s,a)$ that directly approximate the optimal action-value function $q_*(s,a)$ by the following updating rule:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t) \right]. \tag{6}$$

After initializing $Q(s,a)$ with arbitrary values for each state-action pair, the value of a state-action pair is updated every time it is visited according to equation (6). The quantity in the square brackets constitutes an error term. It measures the difference between the current estimate of the action-value and an updated estimate, the target value $Y_t = R_{t+1} + \gamma \max_a Q(S_{t+1},a)$. This target value comprises the observed reward and the discounted value of being in the next state $Q(S_{t+1},a)$. The latter assumes that the agent will behave optimally in the next state and includes all expected future rewards.[10] The learning rate α, $0 \leq \alpha \leq 1$, specifies how much weight the agent gives to new knowledge compared to the current estimation (Sutton and Barto 2018).

Algorithm 1 Q-Learning

1: For all $s \in \mathcal{S} and a \in \mathcal{A}$, initialize $Q(s,a)$ arbitrarily
2: Initialize S_1 arbitrarily
3: **for** $t = 1, T$ **do**
4: With probability ε_t play a random action A_t
Otherwise play $A_t = \arg\max_a Q(S_t,a)$
5: Observe R_{t+1}, S_{t+1}
6: $Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\big]$

The Q-value updates only for state-action pairs who are visited. Therefore, all state-action pairs have to be visited sufficiently often in order to approximate the optimal action-value function $q_*(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Instructing the agent to experiment, i.e., randomly choose an action that may appear sub-optimal based on her current knowledge, assures sufficient exploration. A straightforward procedure to ensure exploration is a ε-greedy policy. With probability ε, $0 \leq \varepsilon \leq 1$, the agent will randomly select any of the possible actions. In all other cases, the agent behaves optimal or greedy according to her current knowledge:

$$a = \begin{cases} \arg\max_a Q(s,a) & \text{with probability } 1-\varepsilon \\ U\{\mathcal{A}\} & \text{with probability } \varepsilon \end{cases}, \tag{7}$$

where $U\{\mathcal{A}\}$ denotes a sample from the discrete uniform distribution over the set of actions $\mathcal{A}$. Experimentation is especially helpful at the beginning when the agents have little knowledge about the environment or the behavior of the competitors. However, experimentation comes at increasing costs of not exploiting the current knowledge. Therefore, the agents follow a policy with exponentially decreasing probability of experimentation $\varepsilon_t = \left(0.015^{\frac{2}{T}}\right)^t$.[11] See algorithm 1 for the pseudocode of Q-learning.

10. It is interesting to note that Q-learning estimates $Q(S_t,A_t)$ based on another estimate $\max_a Q(S_{t+1},a)$: it uses bootstrapping.
11. The probability of experimentation starts around 1 in the first period to 0.015 halfway the run until 0.000225 in the last period $t = T$ (Klein 2021).

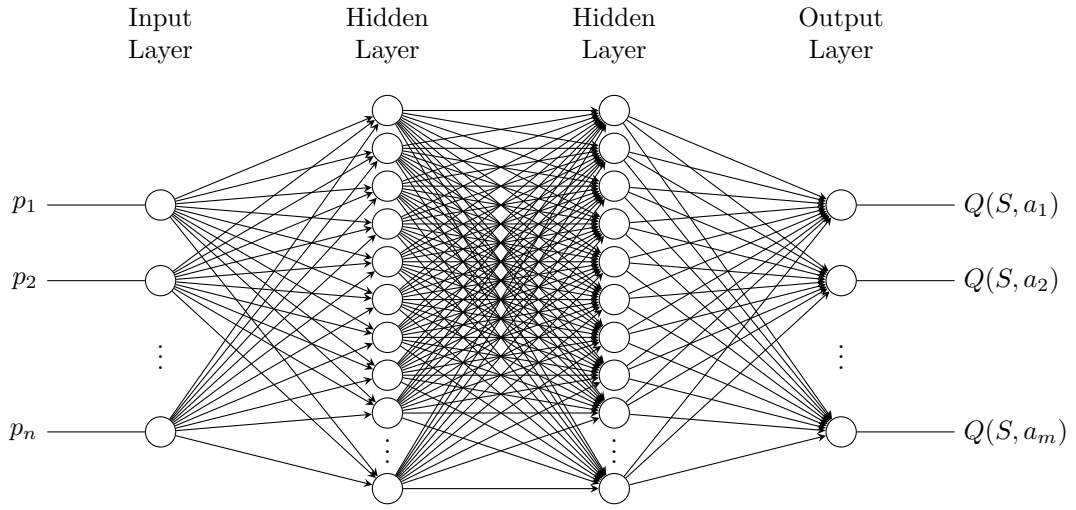

Figure 1: Schematic illustration of the Q-network.

3.2 Deep Q-Network

Q-learning estimates the optimal action-values for each state-action pair separately. The learning capabilities of this approach are not efficient enough for real-world applications. Thus, it is common to use a function approximation with weights θ to estimate the optimal action-value function $Q(s, a, \theta) \approx q_*(s, a)$. Since the pioneering work of Mnih et al. (2015), nonlinear function approximation based on DNNs, called Q-network, are widely used.

Using function approximation has two advantages: First, the experience from one state-action pair improves the estimate for nearby state-action pairs by generalization. Second, it frees us from the curse of dimensionality. The tabular representation of action-values used by Q-learning exponentially increases in the number of agents, and memory consumption quickly reaches its limit.

Figure 1 shows the structure of the Q-network. Input is the agents' previously played prices, the state of the environment. Therefore, the number of input nodes corresponds to the number of agents in the market. The Q-network has two fully-connected hidden layers with 32 nodes each, followed by rectified linear activation functions and one output unit for each possible action. The output layer is fully-connected as well and uses a linear activation function. Thus, the output of the DNN corresponds to the Q-values of all actions at the input state.[12] The main advantage of this approach is that only one forward pass is required to compute the Q-values of all actions in a given state (Mnih et al. 2015).[13]

With function approximation, we improve weights θ instead of direct estimates of the action-values. There are more state-action pairs than weights by definition.[14] In contrast to Q-learning, we cannot get each action-value exactly correct: making one state-action pair's estimate more accurate means making others' less accurate. Thus, the Mean

12. E.g., if an agent wants to play the greedy action, she chooses the action for which the Q-Network predicts the highest value, $A_t = \arg\max_a Q(S_t, a, \theta)$.

13. The size of the network is a somewhat arbitrary decision. Theoretically, one large hidden layer can approximate any function. In practice, deep networks with more than one hidden layer proved to perform better than shallow networks (Goodfellow et al. 2016).

14. In the duopoly setting and with baseline parameters for the Q-networks we have $|\mathcal{S}| = 2$ input nodes, two times 32 hidden nodes, and $|\mathcal{A}| = 15$ output nodes . This translates to $2 \times 32 + 32 \times 32 + 32 \times 15 = 1,568$ node weights and $32 + 32 + 15 = 79$ bias weights. The number of state-action pairs is $15 \times 15 \times 15 = 3,375$. For larger markets the difference further increase, e.g. for markets with three firms there are 1,679 weights and 50,625 state-action pairs.

Squared Error (MSE) evaluates the measure-of-fit of our approximation. At each iteration, we adjust the weights θ to reduce the MSE in the Bellman optimality equation (5). Again, the true target value, the optimal action-value function $R_{t+1} + \gamma \max_a q_*(S_{t+1}, a)$, is not known. Thus, we approximate it by $Y_t = R_{t+1} + \gamma \max_a \hat{Q}(S_{t+1}, a, \hat{\theta})$ using the Q-network with weights $\hat{\theta}$ from a previous iteration.[15] Then, the loss function we want to minimize is:

$$L(\theta) \doteq \left[R_{t+1} + \gamma \max_a \hat{Q}(S_{t+1}, a, \hat{\theta}) - Q(S_t, A_t, \theta)\right]^2. \tag{8}$$

As described above, we need to balance the fit between action-values. At each learning step, we train the Q-network with only a small sample. Instead of reducing the error for this sample completely, the weights should move only a small step in the direction where the error falls most rapidly. We use the popular Adam optimizer to implement this iterative fitting procedure. This gradient descent method considers estimations of first-order and second-order moments to guide the search to areas where the loss declines fastest (Goodfellow et al. 2016).[16]

Algorithm 2 Deep Q-Network

Initialize local network $Q(s, a, \theta)$ with arbitrary weights θ
Initialize target network $\hat{Q}(s, a, \hat{\theta})$ with weights $\hat{\theta} = \theta$
Initialize average reward estimate $\bar{R}$ arbitrarily
Initialize S_1 arbitrarily
for $t = 1, T$ **do**
    With probability ε_t play a random action A_t
    Otherwise play $A_t = \arg\max_a Q(S_t, a; \theta)$
    Observe R_{t+1}, S_{t+1}
    Store transition S_t, A_t, R_{t+1}, S_{t+1} in B
    Sample random minibatch of transitions (S_j, A_j, R_{j+1}, S_{j+1}) from B
    Set $Y_j = R_{j+1} - \bar{R} + \max_a \hat{Q}(S_{j+1}, a, \hat{\theta})$
    Perform a gradient descent step on $[Y_j - Q(S_j, A_j, \theta)]^2$ with respect to θ
    $\bar{R} \leftarrow \bar{R} + \lambda \left[R_{t+1} - \bar{R} + \max_a \hat{Q}(S_{t+1}, a, \hat{\theta}) - \hat{Q}(S_t, A_t, \hat{\theta})\right]$
    Every C steps reset $\hat{Q} = Q$

Algorithm 2 shows the pseudo-code for DQN. Like Q-learning, the algorithm estimates $q_*(s, a)$ and, thus, the optimal policy $\pi_*(a \mid s)$ by following the non-optimal ε-greedy policy. Algorithms that follow another policy while learning the optimal policy, as DQN does, are said to learn off-policy. Off-policy learning combined with function approximation leads to a danger of instability and divergence (Sutton and Barto 2018). As mentioned above, a separate network with older weights, the target network $\hat{Q}(s, a, \hat{\theta})$, calculates the approximated target values Y_t. The target network $\hat{Q}$ has the same structure as the local network Q. The gradient descent step, i.e., the learning, is performed on the local network $Q(s, a, \theta)$ and the agent uses this network to determine her actions. Each C periods, the weights of local network θ replace the weights of target network $\hat{\theta}$. This delay between learning and the effect in the target Y_t considerably reduces the risk of oscillation or divergence in the learning process (Mnih et al. 2015).

15. In contrast to supervised learning, the loss function's target values are not fixed before training but depend on previous weights. At each learning step, the weights $\hat{\theta}$ are held constant. Thus, the optimization problem is well-defined (Mnih et al. 2015).

16. There are many different implementations of gradient descent. For each of the most common optimization methods, we tested how well the DNN learns several pricing strategies of increasing complexity, and Adam performed best.

The use of experience replay further improves the algorithm's stability. Each period t, the agent does not learn with the newest experience $E_t = S_t, A_t, R_{t+1}, S_{t+1}$. Instead, she stores the experience in the replay buffer $B = E_1, ..., E_\beta$ and randomly samples ω experiences from the replay buffer. This minibatch is used to perform a gradient descent step. The replay buffer is limited to store β experiences. If the replay buffer is full, the oldest experience is deleted to free space for the new experience.

The last necessary modification to improve stability replaces the average reward formulation with the discounted formulation. The discounted formulation is incompatible with function approximation in continuing tasks. The definition of the optimal action-value function, $q_*(s,a) \doteq \max_\pi q_\pi(s,a)$ implies that a policy is optimal if, for every state-action pair, following this policy leads to a higher discounted sum of future rewards than any other policy. The optimal policy satisfies the following inequality:

$$q_{\pi_*}(a,s) \geq q_\pi(a,s) \text{ for all } a, s \text{ and } \pi. \tag{9}$$

These inequalities define a partial order on the set of possible policies. Tabular methods store a separate estimate for each state-action pair. Thus, they can represent any possible policy. E.g., Q-learning updates one state-action pair at a time, thus, slowly improving its policy in the direction of the optimal policy. Q-learning can tackle this problem formulation and find the policy that is at least as good as all other policies for each state-action pair (Sutton and Barto 2018).

With function approximation, not each possible policy can be represented since there are fewer weights than state-action pairs by definition. Thus, the optimal policy defined in (9) usually cannot be represented. Instead, the agent tries to learn the best representable policy. However, there is typically no representable policy that is universally better than all other representable policies. For some state-action pairs, one representable policy will be better, and for other state-action pairs, another representable policy. The partial order described in (9) is not transferable to representable policies and the problem formulation is not well-defined (Singh et al. 1994; Naik et al. 2019).

Instead of the partial order of policies, we should use an explicit optimization objective to compare any two policies. The repeated Bertrand competition has no natural end and will potentially go on infinitely. The agent tries to maximize the rewards not just in the present but over the whole lifetime and should choose actions that cause her to visit states with high rewards more frequently (Naik et al. 2019). Since the time horizon is infinitely in continuing tasks, the average reward $r(\pi)$ is a good candidate for the optimization objective:

$$r(\pi) \doteq \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t | A_{0:t-1} \sim \pi], \tag{10}$$

where the expectation is conditioned on the prior actions $A_0, A_1, ..., A_{t-1}$ taken according to the policy π. This measure can order the policies by a single number, their average reward $r(\pi)$. Each policy that attains the maximum value of $r(\pi)$ is optimal (Sutton and Barto 2018).

In the average-reward setting, the agent's goal must be restated as differences between rewards and average rewards:

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots . \tag{11}$$

The optimal action-value function (5) can easily be adjusted to the average formulation by replacing the discount factor γ with the difference between reward and the true average reward:

$$q_*(s,a) = \mathbb{E}[R_{t+1} - r(\pi_*) + \max_{a' \in \mathcal{A}} q_*(S_{t+1}, a') | S_t = s, A_t = a]. \tag{12}$$

The loss function (8) must reflect these changes as well:

$$L(\theta) = \left[R_{t+1} - \bar{R} + \max_a \hat{Q}(S_{t+1}, a, \hat{\theta}) - Q(S_t, A_t, \theta) \right]^2, \tag{13}$$

where $\bar{R}$ is an estimate of the true average reward $r(\pi_*)$ at time t. This estimate is updated at the end of every period. It is pushed a small step, defined by λ, towards the true value as the learned policy converges to the optimal policy (Sutton and Barto 2018).

Figure 2 shows the average profit gain Δ of a DQN agent with discounted reward formulation compared to an agent with average reward formulation. They play against an agent with a hard-coded Tit-for-Tat policy.[17] Both agents can increase their rewards over time. However, the oscillating behavior of the agent with discounted reward formulation is a sign that the optimization problem is not well-defined. In contrast, the agent with average reward formulation learns smoothly and behaves almost optimally at the end of the simulation.[18]

4 Fast Collusion by Deep Learning

Q-learning agents frequently learn to collude, as shown by Calvano et al. (2020b). However, the algorithms learn on average 850,000 periods, which, at best, translates to more than three years on Amazon. It is unlikely that markets stay unaltered that long, especially on e-commerce platforms. This section shows that agents based on DQN learn to collude significantly faster in duopolies.

Each firm or agent i has marginal costs $c_i = 1.0$ and product quality $g_i = 2.0$. The price sensitivity is set to $\mu = 0.25$, the agents can choose from $m = 15$ prices, and one simulation run lasts $T = 100,000$ periods. The superior performance of DQN is achieved at the cost of higher complexity and, thus, more parameters. It is

17. The Tit-for-Tat agent will play the lowest possible price if the opponent undercut her in the previous round. Otherwise, she will replicate the opponent's previous action. This policy encourages cooperation.

18. We implemented the simulated marketplace in Python. Its object-oriented approach allows a free combination of different agents, demand models, or policies. New agents or more sophisticated demand models integrate straightforwardly, and the environment is freely scalable to markets with two or more firms. The complete source code can be found on GitHub (https://github.com/mesjou/price_simulator). Deployment on the HPC cluster PALMA II of the WWU Münster allowed simulations on up to 8 GPUs in parallel.

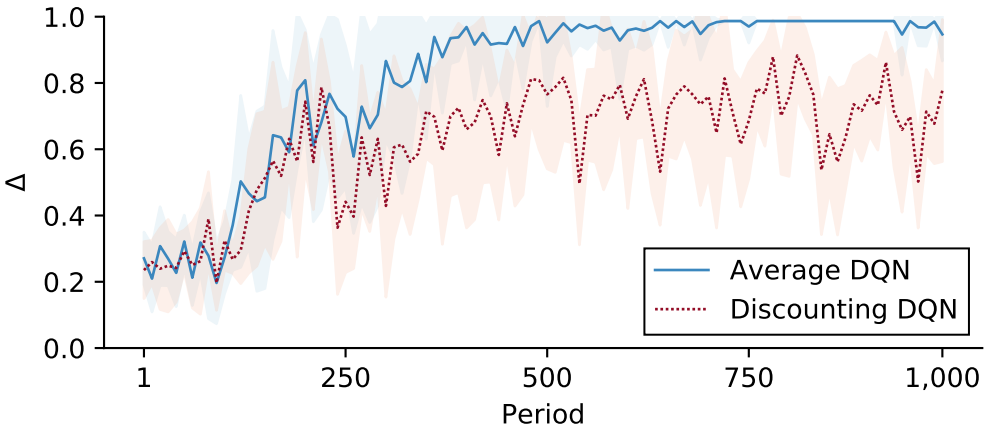

Figure 2: Learning behavior of DQN with discounted and average reward formulation against a Tit-for-Tat agent. The lines show average values and the shaded areas show standard deviations over 10 simulation runs.

computationally not feasible to perform a systematic grid search to select values for all parameters. Besides, we would like to observe if such behavior occurs with as little tuning as possible. Except for the learning rate α and the size of the replay buffer β, we adopted standard values or performed an informal search.[19]

Learning rate α is the most important parameter for training a DNN. Together with the size of the replay buffer, it determines the pace of learning. If the learning rate is too large, gradient descent will overshoot and oscillate without finding local or global minima. If the learning rate is too low, the weights could shift too little, and it is likely to end up in bad local minima (Goodfellow et al. 2016). Replay buffer size β determines how quickly the training data represents changes in the competitor's strategy. With small replay buffers, the learning targets change too fast, and the DQNs do not have enough time to learn the opponent's strategy. They play prices near the one-shot game's optimal price, which results in average profit gains near the static Nash equilibrium. If the replay buffer is too large, the learning target is stable, but the algorithm will take a long time to adapt to changes in the competitor's strategy.

The significant influence of the two parameters on learning requires a systematic grid search for optimal values. We considered replay buffer sizes β from 500 to 10,000 samples and learning rates α from 0.00025 to 0.005. For replay buffer sizes above 5,000 and learning rates around 0.001, the average profit gains of DQN are comparable to the ones of Q-learning. In the following, we keep the replay buffer size fixed at $\beta = 5,000$ and the learning rate at $\alpha = 0.001$. However, the results are robust to reasonable changes in these parameters.

Figure 3 shows the average profit gain Δ over time. At the beginning of the simulation, both agents experiment frequently. It is not possible to answer to a stable strategy of the opponent. Thus, playing a low price to undercut the opponent is the best an algorithm can do: prices decrease, and Δ gears towards the static Nash equilibrium level. As the experimentation probability ε decreases, the agents' strategies become more stable. Already after

19. For an overview of all parameters, see appendix A.

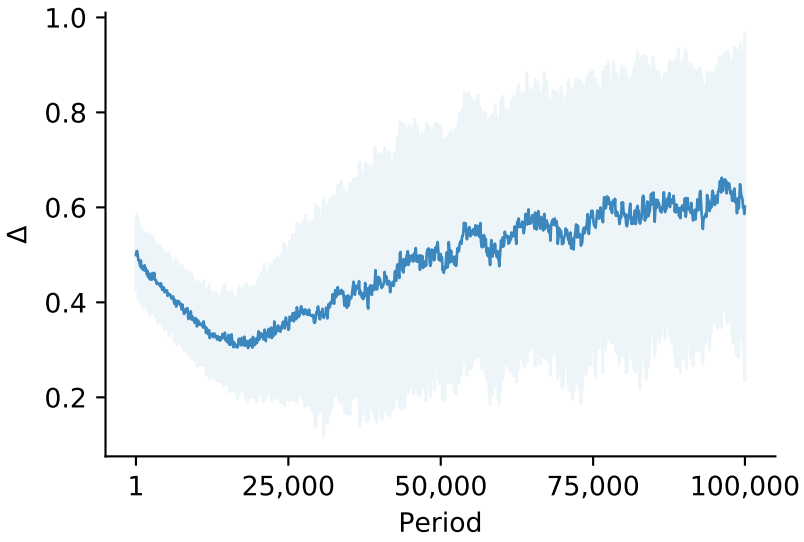

Figure 3: Average profit gain of DQN in a duopoly. The line shows average values and the shaded area shows standard deviation over 50 simulation runs.

20,000 periods, the agents start to raise profits by increasing prices. After this turning point, collusion steadily increases. In the last 5,000 periods of the simulation, the agents obtain average profit gains of $\Delta = 63\%$.[20]

Besides looking at market outcomes, it is crucial to examine how collusion occurs. Supra-competitive prices could arise because either i) the agents fail to learn how to compete effectively or ii) the agents learned effective reward-punishment schemes. Following Calvano et al. (2020b), we verify that DQN learns a reward-punishment scheme by analyzing the response to a price cut of the opponent. The agents play for $T = 100,000$ periods against each other. Then learning is disabled and the agents play for 50 periods with fixed strategies. In $t = 0$, we artificially force the defecting agent to play the static Nash price, which causes the price cut in figure 4.

The non-defecting agent punishes the other agent by lowering her price in $t = 1$, the period after defection. The defecting agent expects this punishment and sets her price near the punishment price as well. Then both agents simultaneously start to increase their prices. Already after five periods, the prices are near the pre-defection level. On average, the prices stay slightly below the long-run price after the punishment phase, making defection unprofitable for both agents.

In addition, it is possible to show the average learned strategy for the whole state-space in markets with two agents. Figure 5 shows the optimal action of agent 1 in each of the $15 \times 15 = 225$ states.[21] The first thing that catches the eye is the symmetry of the surface. The DQN agent reacts similarly to previous prices no matter which agent played which price. The agent will not just punish the opponent's defection, but also anticipates the punishment if she defects herself. For prices in the medium range, the agent holds the price approximately

20. We replicated the results of Calvano et al. (2020b). Q-learning has discount parameter $\gamma = 0.95$, learning rate $\alpha = 0.125$ and one simulation run lasts $T = 1,000,000$ periods. In a duopoly, the agents consistently reach $\Delta = 67\%$. Calvano et al. (2020b) document Δ from 70% to 90%. The reason for the higher values is the initialization of the Q-values. Instead of setting the Q-values to the discounted return that would result if the opponent plays random prices, we initialized it to 0. We refrain from this procedure to make a comparison to DQN possible, and it is not clear what pendant to use for initialization of the Q-network weights θ.

21. The strategy for agent 2 has the same shape.

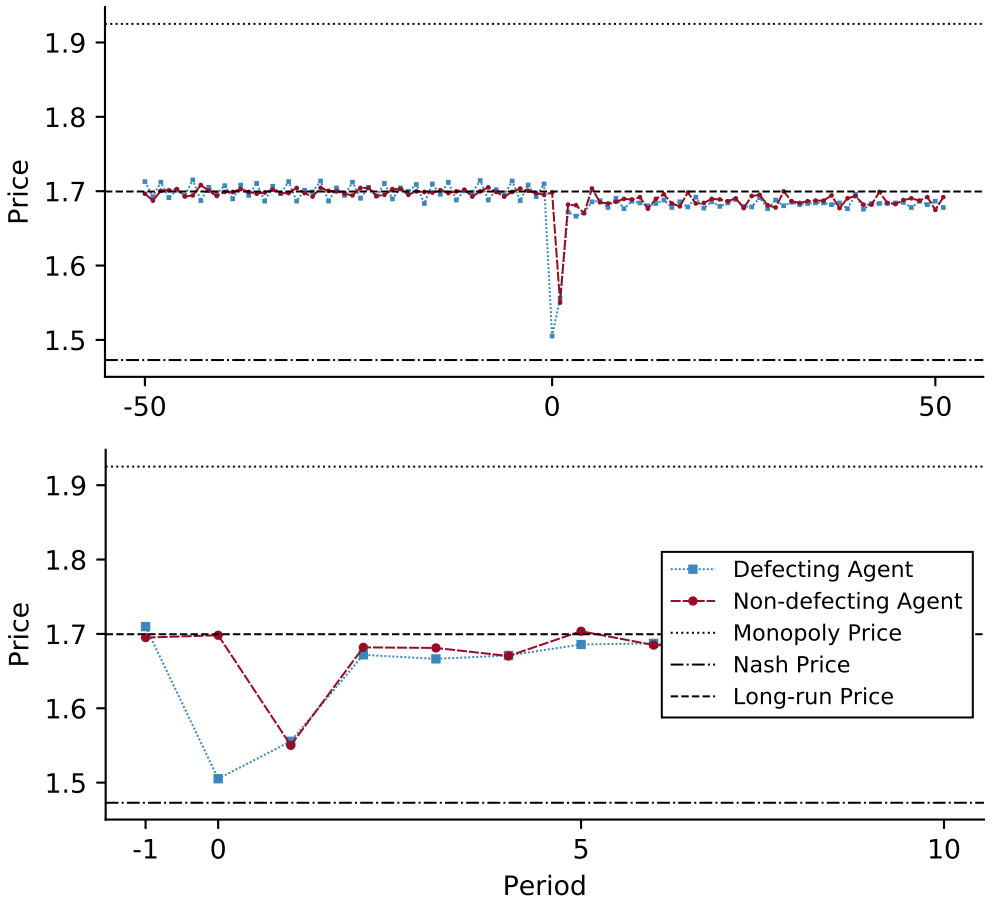

Figure 4: Price reaction after a defection of one agent to the static Nash equilibrium price. The lower figure zooms in to the post-defection phase. The lines represent average values over 50 simulation runs (Own representation following Calvano et al. 2020b).

constant. The concave shape of the surface at the top shows that the agent starts to undercut as prices approach the monopoly price.

Another remarkable behavior is the price increase when both agents play prices near the Nash price. This mutual understanding enables the agents to return to higher prices after a punishment. As long as both cooperate, prices stay at a collusive outcome. If one of the agents defects and unilaterally decreases her price, both agents stop cooperating and lower their prices. If both agents defect and play prices near the Nash equilibrium, both mutually return to cooperation and again play prices at a collusive level. The learned strategy is reminiscent of the Pavlov (or win-stay-loose-shift) strategy described by Kraines and Kraines (1989).

This section shows that the reinforcement learning algorithms based on deep learning do not simply fail to learn competitive strategies but learn effective reward-punishment schemes. Equipped with this strategy, DQNs are able to collude significantly faster than the simple Q-learning of previous studies. The algorithm description of section 3.2 shows that the higher speed comes at the cost of greater complexity. Deep learning methods tend to diverge to sub-optimal solutions and, thus, modifications of the Q-learning algorithm are necessary to ensure the stability of learning.

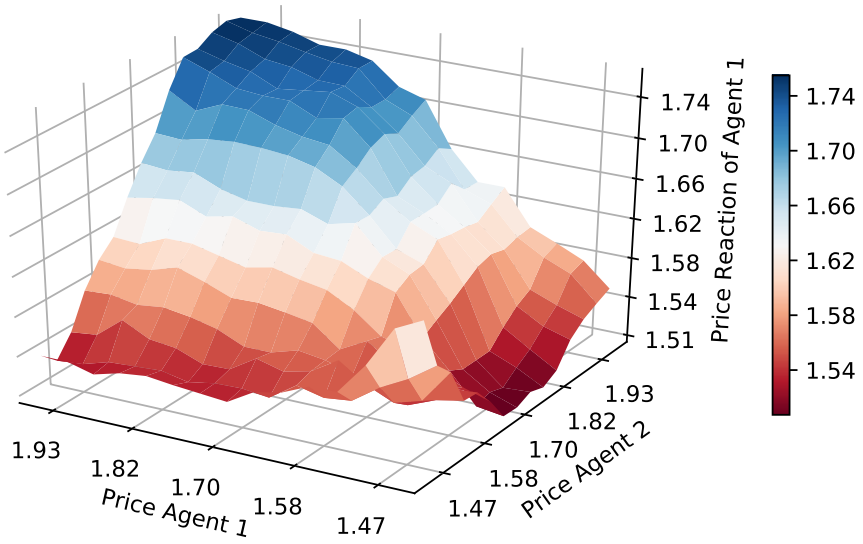

Figure 5: Learned strategy of DQN in a duopoly. The figure displays the optimal action of agent 1 in each possible state according to $a = \arg\max_a Q(s, a, \theta)$. The state s are the previously played prices of the two agents, the values on the x-axis and y-axis. The 3-d surface shows the average optimal action over 50 simulation runs.

5 Collusion in Wide Oligopolies

Economic experiments show that collusion by human price-setters is strongly affected by the number of competitors. Implicit coordination of prices above the competitive level is rarely observed in markets with four firms (Potters and Suetens 2013). Calvano et al. (2020b) show that collusion among algorithms decreases but is still present in markets with three or even four firms. The availability of DQN makes experiments in larger markets possible. To answer if DQN can collude even in wide oligopolies, we repeat the previous section's simulation for markets with up to ten firms. Figure 6 shows the average profit gain Δ In the last 5,000 periods for different market sizes. Similar to Q-learning, collusion decreases in the number of market participants. In markets with three firms, the average profit gain is $\Delta = 30\%$, and thus significantly higher than the static Nash equilibrium level. With more market participants, Δ falls below 20% and quickly approaches zero.[22]

We tested for various factors that could have hindered the algorithms from coordinating their prices above competitive levels in wide oligopolies. Neither a higher approximation capacity with larger DNN (up to three hidden layers with 128 nodes each), more time to learn (simulation runs with up to $T = 500,000$ periods), nor straightforward enhancements of DQN (i.e., DDQN) increases Δ significantly.

Alternatively, we tested the influence of the state representation on the learning of strategies. The state determines how an agent perceives the environment and, therefore, the other agents' behavior. The examination of strategies

22. In our benchmark simulations with Q-learning, the average profit gains are similar: $\Delta = 30\%$ in markets with three agents and $\Delta = 27\%$ with four agents. Q-learning, DQN, and the economic environment have the same parameter compared to section 4.

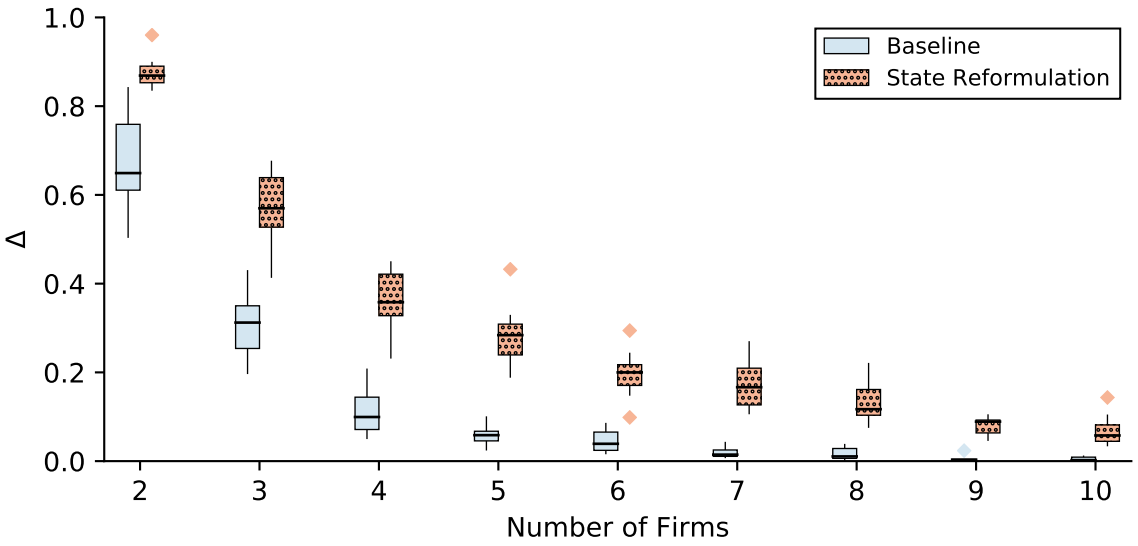

Figure 6: Box-plot of the average profit gain of DQN in wide oligopolies over 10 simulation runs per setting.

in section 4 shows that it does not matter who played which price in the previous round. More important is the overall price level and whether an agent defected by either playing a higher or lower price. Thus, a reasonable state representation should contain the average of the last period's prices. Defection can be measured by including the minimum and maximum value of last period's prices.

Figure 6 shows the results for simulations where each agent perceives the environment's state according to this formulation. The average profit gains are higher for all simulated market sizes. For three firms, the profit gain is $\Delta = 57\%$, for four $\Delta = 36\%$ and for five still $\Delta = 27\%$. With more firms, it falls below 20% and approaches zero, similar to the baseline setting. Somewhat surprising is the fact that the reshaped state representation also helped collusion in duopolies.

A possible explanation could be the reduction of detrimental variation by aggregating the previous period's prices. That frees the DNNs from distinguishing between important price defection and small price fluctuations. In real-world applications, more advanced methods than the described pre-processing are available. Possible candidates are a direct measure of the opponent's willingness to cooperate, some economic structure, or a strategy model. The simulation results indicate that such expert knowledge could facilitate collusion even in wide oligopolies.

The results so far have shown that wide oligopolies tend to make collusion more difficult. However, this finding should be viewed in light of the fact that pricing algorithms with a rule-based or adaptive strategy are still in use today. The managers or programmers define more or less exactly how the algorithm should set the price for their product. Self-learning algorithms are on the rise, but there will be a transition phase where both types of algorithms compete with one another. The question arises of which affect such simple algorithms have on collusion.

The left figure 7 shows the average profit gain in a market with two DQN agents and one rule-based agent. The latter plays a strategy that Chen et al. (2016) found to be popular on Amazon: targeting the lowest price. The

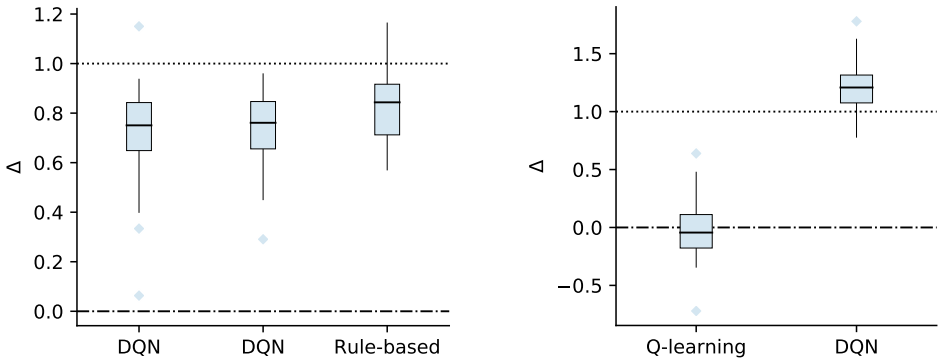

Figure 7: Effect of a rule-based agent and different learning pace on average profit gains. The box-plot shows the results for 50 simulation runs.

rule-based agent plays the lowest price of the competitor's last period's prices. The DQN agents reach average profit gains of $\Delta = 73\%$ which is comparable to the duopoly case with two DQNs.

This stylized experiment illustrates an essential property of a self-learning algorithm: the algorithm perceives the other firms as part of the environment. By interacting with the environment, the algorithm learns the demand model plus the opponent's strategy because both influence the state-transition probabilities (3). This task is daunting with self-learning opponents because their strategy changes during the learning process and makes the state-transition probabilities non-stationary. Contrary, a rule-based opponent behaves according to a time-invariant strategy. Even though this increases the complexity of the state-transition probabilities, it does not decrease their stationarity.

With these insights, it becomes apparent that the coordination problem's complexity between two self-learning algorithms will not significantly increase with an additional rule-based agent. Larger markets will decrease the likelihood of algorithmic collusion. However, this seems only valid if the additional agents are self-learning. Simple rule-based pricing algorithms or human-price setters that do not frequently change prices do not necessarily hinder collusion.

Another fact about real marketplaces can be addressed with a stylized experiment. Firms develop their pricing algorithms independently and will end up with algorithms that differ in many dimensions, e.g., state representation or learning pace. Section 4 shows that algorithmic collusion relies upon the mutual understanding to increase prices after defection. Up until now, the self-learning algorithms in each simulation are identical. One might presume that this similarity facilitates collusion. Thus, we test what happens if two algorithms play against each other that, in principle, can learn a similar level of collusion but learn at a different pace. We let one DQN agent play against one Q-learning agent, and each simulation run lasts $T = 100,000$ periods. The algorithms' parameter remain unchanged compared to section 4.

The right figure 7 reveals that the faster DQN exploits the slower Q-learning. DQN reaches profits above the monopoly level, whereas Q-learning earns profits near the static Nash equilibrium. Inspection of prices, quantities, and profits reveal that DQN plays a highly competitive strategy. DQN takes advantage of the fact that Q-learning

cannot optimize her behavior fast enough and lowers her price. In this way, she drives Q-learning out of the market, increases the demand for her product, and gains high rewards.[23] The high profits of DQN are not a sign of collusion, but a sign of full exploitation. Again, this shows the importance of joint learning for collusion. If both agents learn effective reward-punishment schemes, collusion emerges. However, cooperation is not an integral part of self-learning algorithms. Instead, the algorithms fully take advantage of the chance to exploit the opponent and selfishly increase their profit.

6 Conclusion

The simulations have shown that DQN algorithms systematically set supra-competitive prices in a duopoly. The learned strategies contain a reward-punishment scheme that enforces collusion and, thus, coincide with those of Q-learning. However, DQNs collude significantly faster than Q-learning. The algorithms start to increase prices after approximately 20,000 periods. From a regulatory point of view, the results seem alarming, and algorithms that learn even more efficiently than DQN are already available today, e.g., through self-play or meta-learning.

The more efficient DQN algorithm enabled experiments in wide oligopolies. With an increasing number of market participants, the level of collusion decreases, and with seven or more firms, at the latest, collusion completely vanishes. Remarkably, a slight reformulation of the state representation increases the ability to collude across all market sizes. Future research should identify which state representation or incorporation of which expert knowledge further facilitates collusion.

In contrast to additional self-learning algorithms, firms that use rule-based algorithms do not necessarily complicate collusion. Thus, algorithmic collusion could emerge even in wide oligopolies that seem to bear a low risk of collusion when two or three self-learning algorithms compete with simple rule-based algorithms or human price-setters with low pricing frequency.

Examination of the learned strategies shows that collusion heavily depends on mutual understanding. Heterogeneity in the pace of learning hinders collusion, and it could be suspected that this holds for differences in other characteristics of the algorithms as well. Thus, competition policy should prevent firms from using the same or similar algorithms. The implementation of DQN requires modifications to ensure stability of learning. Real-world algorithms will require even more modifications, and it is unlikely that independently developed algorithms will be exactly similar. Therefore, especially pricing algorithms from third-party providers may pose a high risk if widely used in one market.

The presented work is the first step towards realistic pricing algorithms by demonstrating the deep learning implementation of standard RL algorithms. However, in real-world applications, pricing algorithms have to deal with two aggravating conditions. First, they have to choose prices from a continuous price spectrum or at least

23. We repeated the simulation with a longer time horizon of $T = 1,000,000$. With more time, Q-learning can learn a reward-punishment scheme. Both agents start to cooperate and end up at similar collusive outcomes compared to duopolies with two Q-learning or two DQN agents.

from a significantly wider price range than the one used in this study. Thus, future research should focus on policy gradient algorithms.

Second, future simulations should take the temporary character of real-world markets into account. E.g., on Amazon, the average product lifespan is between 15 and 30 days, and most price updates occur after more than 30 minutes (Chen et al. 2016). Even though the update frequency will likely increase in the future because it seems to translate into competitive advantages, real-world algorithms must learn more sample-efficient than the algorithms studied so far. A promising candidate for such algorithms is meta-learning that is expected to generalize learned behavior to new environments (see, e.g. Wang et al. 2018). The meta-learning algorithm could learn general pricing strategies across various market constellations. Based on this knowledge, the algorithm quickly adapts to a new market constellation and learn proficient pricing strategies after a few periods. In order to more accurately assess the risk of algorithmic collusion in real-world markets, it is essential to test whether such advances in RL enable algorithms to quickly restart collusion in fast-changing markets.

References

Abada, Ibrahim, and Xavier Lambin. 2020. "Artificial Intelligence: Can Seemingly Collusive Outcomes Be Avoided?" http://dx.doi.org/10.2139/ssrn.3559308.

Amazon. 2020. *Selling Partner API Documentation.* Accessed September 29, 2020. http://docs.developer.amazonservices.com/en_DE/dev_guide/index.html.

Assad, Stephanie, Robert Clark, Daniel Ershov, and Lei Xu. 2020. "Algorithmic Pricing and Competition : Empirical Evidence from the German Retail Gasoline Market." https://ssrn.com/abstract=3682021.

Brown, Zach, and Alexander MacKay. 2021. "Competition in Pricing Algorithms." http://dx.doi.org/10.2139/ssrn.3485024.

Calvano, Emilio, Giacomo Calzolari, Vincenzo Denicolò, Joseph E. Harrington Jr., and Sergio Pastorello. 2020a. "Protecting consumers from collusive prices due to AI." *Science* 370 (6520): 1040–1042.

Calvano, Emilio, Giacomo Calzolari, Vincenzo Denicoló, and Sergio Pastorello. 2021. "Algorithmic collusion with imperfect monitoring." *International Journal of Industrial Organization,* 102712.

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

Chen, Le, Alan Mislove, and Christo Wilson. 2016. "An Empirical Analysis of Algorithmic Pricing on Amazon Marketplace." In *25th International World Wide Web Conference.*

Ezrachi, Ariel, and Maurice E. Stucke. 2016. "Virtual Competition." *Journal of European Competition Law & Practice* 7 (9): 585–586.

———. 2017. "Artificial Intelligence & Collusion: When Computers Inhibit Competition." *University of Illinois Law Review,* 1775–1810.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* Cambridge: MIT Press.

Hansen, Karsten T., Kanishka Misra, and Mallesh M. Pai. 2021. "Frontiers: Algorithmic Collusion: Supra-competitive Prices via Independent Algorithms." *Marketing Science* 40 (1): 1–12.

Harrington Jr., Joseph E. 2019. "Developing Competition Law for Collusion by Autonomous Artificial Agents." *Journal of Competition Law & Economics* 14 (3): 331–363.

———. 2020. "Third Party Pricing Algorithms and the Intensity of Competition." http://dx.doi.org/10.2139/ssrn.3723997.

Johnson, Justin, Andrew Rhodes, and Matthijs R. Wildenbeest. 2020. "Platform Design When Sellers Use Pricing Algorithms." http://dx.doi.org/10.2139/ssrn.3691621.

Klein, Timo. 2021. "Autonomous Algorithmic Collusion: Q-Learning Under Sequential Pricing." *RAND Journal of Economics* 52 (3): 538–558.

Kraines, David, and Vivian Kraines. 1989. "Pavlov and the Prisoner's Dilemma." *Theory and Decision* 26:47–79.

Leibo, Joel Z, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. "Multi-Agent Reinforcement Learning in Sequential Social Dilemmas." In *Proceedings of the 16th International Joint Conference on Autonomous Agents and Multiagent Systems.*

Mehra, Salil K. 2016. "Antitrust and the Robo-Seller: Competition in the Time of Algorithms." *Minnesota Law Review* 100 (4): 1323–1375.

Meylahn, Janusz M., and Arnoud den Boer. 2021. "Learning to Collude in a Pricing Duopoly." http://dx.doi.org/10.2139/ssrn.3741385.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, et al. 2015. "Human-Level Control through Deep Reinforcement Learning." *Nature* 518 (7540): 529–533.

Naik, Abhishek, Roshan Shariff, Niko Yasui, Hengshuai Yao, and Richard S. Sutton. 2019. "Discounted Reinforcement Learning Is Not an Optimization Problem." https://arxiv.org/abs/1910.02140.

Potters, Jan, and Sigrid Suetens. 2013. "Oligopoly Experiments in the Current Millennium." *Journal of Economic Surveys* 27 (3): 439–460.

Salcedo, Bruno. 2015. "Pricing Algorithms and Tacit Collusion."

Singh, Satinder P., Tommi Jaakkola, and Michael I. Jordan. 1994. "Learning Without State-Estimation in Partially Observable Markovian Decision Processes." In *Machine Learning Proceedings 1994.*

Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* 2. Edition. Cambridge: MIT Press.

Wang, Jane X., Zeb Kurth-Nelson, Dharshan Kumaran, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Demis Hassabis, and Matthew Botvinick. 2018. "Prefrontal Cortex as a Meta-Reinforcement Learning System." *Nature Neuroscience* 21 (6): 860–868.

A Simulation Parameter

Parameter	Symbol	Default Value
Economic Environment		
Marginal costs	c	1.0
Quality	g	2.0
Price sensitivity	μ	0.25
Number of prices	m	15
Q-learning		
Learning rate	α	0.125
Discount factor	γ	0.95
Deep Q-Network		
Learning rate	α	0.001
Reward step size	λ	0.01
Periods between target network updates	C	100
Size of the replay buffer	β	5,000
Number of hidden layers		2
Number of nodes per hidden layer		32
Minibatch size	ω	32
The exponential decay rate for the 1st moment estimates of Adam	β_1	0.9
The exponential decay rate for the 2nd moment estimates of Adam	β_2	0.999
A small constant for numerical stability of Adam	ϵ	1e-7

Table 1: Default parameter of the economic environment and the algorithms.