STOCKHOLM SCHOOL OF ECONOMICS
Department of Economics
5350 Master's thesis in economics
Academic year 2019-2020

Tacit collusion with deep multi-agent reinforcement learning

Filip Mellgren (23644)

Abstract. Automatic pricing now attracts the attention of competition authorities following recent machine learning developments. In particular, previous research shows that the Q-learning algorithm can reach collusive outcomes despite receiving only minimal human intervention. This thesis extends the current body of knowledge by considering the addition of neural networks to the Q-learning algorithm which enables learning in more complicated and close to reality environments. A simulation is conducted where two deep Q-learning agents play against each other in a sequentially repeated price game with payoffs configured to resemble a social dilemma. The agents start without any prior knowledge, but are deployed with the objective to maximise a discounted profit function which is learned about through ongoing exploration. After 3.5 million repeated interactions, the agents are evaluated in a test mode. Four specifications are tested, a basic scenario, an increased action space, random demand, and vertical differentiation. In the basic scenario, I find that both agents learn to associate high prices with large profits, resulting in profits reaching 95% of a hypothetical monopolist's profit. However, the agents do not learn reciprocity, meaning that they are exploitable. For the other specifications, the increased complexity means the agents do not learn stable behaviour during the training period under consideration. The findings add to previous research by showing that the addition of neural networks is possible, which opens up the door for more realistic future applications.

Keywords: deep multi-agent reinforcement learning, tacit collusion, pricing algorithms
JEL: C45, L13

Supervisor: Andreea Enache
Date submitted: 17/05/2020
Date examined: 27/05/2020
Discussant: Alessandro Festante
Examiner: Kelly Ragan

Acknowledgements
Thanks to Johannes Matt, Oscar Krumlinde, and Erik Senn for giving insightful comments.

Contents

1 Introduction **1**

2 Background **3**

2.1 Machine learning . . . 3

2.1.1 Reinforcement learning . . . 4

2.1.2 Q-learning . . . 6

2.1.3 Function approximation using neural networks . . . 7

2.1.4 Combining Q-learning with deep learning . . . 9

2.1.5 Multi-agent learning . . . 12

2.2 Social dilemmas and pricing . . . 14

2.2.1 Game theory and cooperation in social dilemmas . . . 14

2.2.2 Collusion . . . 15

2.2.3 Algorithmic pricing . . . 16

3 Literature review **18**

4 Method **19**

4.1 Reinforcement learning environment . . . 20

4.2 Action space . . . 21

4.3 Rewards . . . 22

4.4 State space . . . 23

4.5 Neural network . . . 23

4.6 Training . . . 24

5 Results – main specification **25**

5.1 Training dynamics . . . 26

5.2 Q-values . . . 27

5.3 Profit gain in a test setting . . . 28

6 Variations to the main specification **29**

6.1 Random demand . . . 29

6.2 Increased action space . . . 30

6.3 Varying vertical differentiation . . . 31

7 Discussion **32**

8 Conclusion **34**

A Additional figures **38**

1 Introduction

At a conference on competition in Berlin, the European competition commissioner Margrethe Vestager held a speech pointing towards the potency and troubles of algorithmic pricing:

> But when we look at the challenges for cartel enforcement in the future, one of the biggest things we need to deal with is the risk that automated systems could lead to more effective cartels. [...] The algorithms can also help to establish a cartel in the first place. Vestager (2017)

What Mrs Vestager refers to is primarily the ability of algorithms to monitor price levels. This ability is problematic because transparent pricing makes it easier to sustain a cartel, thereby reducing overall welfare. The possibility of algorithms monitoring prices also means that pricing online more closely resembles theoretical scenarios of complete information. An old result depending on complete information known as the *folk theorem* (see for instance Belleflamme & Peitz (2015, p.346-347)) asserts that with discounting close to unity in infinitely repeated games, there exists a subgame perfect Nash equilibrium arbitrarily close to any feasible outcome of the game, such as fully collusive outcomes in a pricing game.

In particular, the British Competition and Markets Authority points out that a 'more serious situation' occurs when an intermediary supplies two competitors with a common algorithm, as this may result in hub and spoke agreements without competitors ever having to reach collusion explicitly (The Competition and Markets Authority 2018, p. 26). Such collusion would fall under what is called tacit collusion which current regulation may be unable to prevent, as expressed in a report by the OECD:

> It is still unclear at this point whether any regulations can be created to prevent machine learning algorithms from autonomously reaching tacit co-ordination, at least not without harming the competitive process in other ways. To the best of the Secretariat's knowledge, no solutions have been proposed so far in the antitrust literature to tackle this conduct. Moreover, there are no competition cases or investigations providing supportive evidence of this "virtual" form of collusion, making it hard to justify the creation of regulations to prevent the negative impact of conducts that have not been observed yet. OECD (2017, p. 49)

The machine learning algorithms mentioned above fall under a paradigm called reinforcement learning (RL). Under RL, algorithmic agents are trained in a physical or a simulated environment which the agent interacts with in order to learn about the environment's system dynamics. The agents learn by relating state-action pairs to rewards and transitions to new states. In a price setting context, the environment can be thought of as the demand conditions firms face, rewards as profits, and the system's transition dynamics as something dependent on competition and demand.

In this thesis, investigations relate to the broader issue of algorithmic price collusion by focusing on a special class of algorithms called deep multi-agent reinforcement learning (DMARL) algorithms. These algorithms are of interest because they have the ability to learn about the dynamics of an unknown economic environment and can use this knowledge to optimise a reward function without further human intervention. In addition, the inclusion of neural networks makes them capable of handling more complex environments. As a consequence of the algorithm's benign objective to maximise profits, they become hard to prosecute even if algorithms were to collude with other similar algorithms since eventual collusion that stems from the sole objective to unilaterally maximise profits would be deemed as tacit collusion, which cannot be prosecuted under

current legislation. The question of interest thus becomes: are deep Q-learning algorithms able to find and sustain collusive outcomes and can the algorithm enable learning in more complex environments than previously considered?

The approach to the problem is empirical and results are obtained through a simulation. In the simulation, an RL agent is trained against a copy of itself before the agent's performance is measured and tested. Training starts with a random initialisation and the agent is subsequently left to explore the environment and gradually learns to associate actions with profits. The environment is characterised by a logistic demand specification and two firms. After training, the agent is evaluated by the profit it is able to collect in both the training environment.

I find evidence that DMARL agents favour collusive outcomes. This aligns with theoretical results and what previous researchers have found when investigating non-deep multi-agent reinforcement learning (MARL) algorithms. However, collusive outcomes are observed without an indication of the agent learning to punish deviating strategies, suggesting that a DMARL agent may be exploitable in the sense that pricing according to the best response is able to outperform the DMARL agent. Another important consideration is that collusive behaviour emerges after relatively many interactions in a sterile environment, leaving the question open to whether such algorithms are able to learn effectively in the field. The algorithm considered did not lead to stable learning in the more complex environments.

The main contribution of this thesis is to show that, despite a theoretical result known as the *deadly triad*, it is possible to extend tabular Q-learning algorithms with a function approximator, such as neural networks, without observing diverging learning and unbounded value estimates. The purpose of studying the addition of neural networks to Q-learning is to learn about the most promising class of algorithms, namely learning algorithms that do not need a model of the environment and simultaneously have the ability to use function approximation to learn about complex underlying relationships. Algorithms used in previous papers are limited by having to resort to a tabular discretisation of the environment and are consequently incapable of taking into account the topological structure of the environment. Therefore, the tabular algorithms have to estimate values for each possible combination found in the environment which means that the time requirement of adding more variables or finer discretisation is increasing combinatorically with the number of possible states. Therefore, tabular algorithms are ultimately infeasible when one wants to represent a continuous environment or an environment of several variables. Neural networks overcome this problem by learning an approximation to the underlying relationship which also means that they are able to generalise knowledge to previously unseen states of the environment. Whether the agent in this thesis is able to learn in more complex environments is tested but ultimately does not lead to strong results showing that the neural network can handle the more complex environments.

Nonetheless, the necessity of function approximation to construct reinforcement learning agents was expressed in Sutton & Barto (2018, p. 215) which states that '[...] function approximation most clearly cannot be given up. We need methods that scale to large problems and to great expressive power. [...] State aggregation or non-parametric methods whose complexity grows with data are too weak or too expensive'. The pricing problem is arguably a large problem, considering the number of possible prices, the variety of product differentiation and costs, and demand fluctuations. Consequently, a more potent algorithm than what has been previously considered in the literature may lead to meaningful advancements.

From a policy perspective, the results are relevant insofar that they show the possibility of collusion by DMARL agents. For instance, the results are applicable when two similar algorithms compete against each other. This could be true if a developer or consultant sells the same algorithm

to competing firms, enabling so called *hub and spoke* collusion. Furthermore, the results indicate that significantly more potent algorithms than previously seen may be just around the corner. Finally, the way in which the results are obtained may enhance understanding of the process that creates such algorithms, thereby demystifying the black box they are sometimes seen as.

The structure of the paper is as follows, in the next section, Section 2, I provide the necessary background knowledge by introducing concepts from machine learning, and game theory. Then, Section 3 provides an overview of previous research on pricing algorithms. Section 4 describes the firms and their algorithms (jointly called *agents*), the economic environment in which the agents interact, and the neural network specification that is used to train the agents. Results of the agents' behaviour are presented in section 5. I then provide a couple of variations to the main specification in Section 6, a discussion in Section 7, and Section 8 concludes.

2 Background

2.1 Machine learning

Machine learning (ML) is a field within AI that uses data to learn functional approximations of underlying relationships. The approach to 'intelligence' is not based on prior knowledge incorporated into algorithms by humans, but rather on letting the algorithm learn relations based on data. For example, an ML approach to classify hand written digits would be to feed the algorithm images of hand written digit coupled with a label variable of what digit each image represents, and then let the algorithm learn a probabilistic relation between the pixel input and the provided labels, whereas a non-ML AI approach might try to associate combinations of pixels (e.g. 'lines' and 'circles') with digits directly.

To formalise the definition of learning algorithms, an often cited definition is that by Mitchell et al. (1997, p. 2) which states that: 'A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E'. Where an instance of T might be to classify hand written digits, which could have as a suitable P the percent of digits correctly classified, and E would be the input data in the form of labelled images. Thus, increasing the amount of labelled images, E, fed into the learning algorithm is then supposed to increase the percent of correctly classified digits, P.

There are three main paradigms of ML algorithms: supervised learning, unsupervised learning, and reinforcement learning. The first paradigm, supervised learning, is characterised by the requirement of a data set that contains a labelled outcome variable (the digit classification task is thus an example of supervised learning) and the main objective of a supervised learning algorithm is to use data to predict the labelled outcome variable. Typical tasks include regression problems, where the outcome variable takes a numeric value, and classification problems, where the outcome variable belongs to a category, such as a digit in the example of classifying hand written digits.

In contrast to econometrics, where the main objective arguably is to obtain unbiased estimates of individual parameters (the focus is on $\hat{\beta}$), supervised learning is not concerned with estimating parameters consistently. Instead, the main focus of research has been to develop methods that enable prediction (focus on $\hat{y}$). For example, an economist would care about the marginal return of a year of education, β, whereas a machine learning practitioner would be satisfied with noting that education is a strong predictor of income, the y. However, the overlap between machine learning and econometrics is not negligible and a paper by Mullainathan & Spiess (2017) demonstrates, and gives examples of, how supervised learning algorithms can be applied in economics. Notable cat-

egories of potential applications include handling more complex data types such as images or text, using ML for implicit prediction tasks such as the first stage of an instrumental variable estimation, and policy applications where the effect of a policy needs to be predicted. This thesis deals with another category, namely investigating the impact of ML algorithms on economic outcomes.

Another distinguishing feature of ML is that it seeks to learn tasks in order to generalise the knowledge. For this reason, a model is evaluated on a test set of data which is held separate from data used to train, or fit, a model. This means that rather than optimising a loss function[1] on the train set, the aim is *risk minimisation*, minimising the expected loss function on the test set. Though, the way this is achieved is often by optimising on the train set, so called minimising *empirical risk*. By measuring performance on a previously unseen test set, it is possible to measure whether the algorithm is improving its ability to generalise knowledge learned using the train set.

In addition to supervised learning, machine learning algorithms may be unsupervised. The crucial difference between supervised and unsupervised learning is that unsupervised learning algorithms do not assume labelled training data. The purpose is thus different and more often takes the form exploring the data, identifying clusters, and reducing the dimensionality of data. Unsupervised learning is not applied in this thesis and is subsequently left out in what follows.

The third paradigm of ML algorithms, reinforcement learning (RL), is about algorithms that learn to optimise a reward function by taking actions in unknown, but fixed, environments by interacting in it. Such environments may be either simulated or a part of the physical world. It is this type of algorithm that lead to the breakthroughs mentioned in the beginning of the introduction. More technical details and lengthy discussions of this type of algorithm will be discussed in section (2.1.1) as RL is used extensively in this thesis. To be precise, the DMARL algorithm that is used in this thesis uses concepts from RL, but also from supervised learning as the 'D' in DMARL stands for 'deep' meaning a neural network is used for function approximation which is a supervised learning task. Furthermore, the 'MA' stands for 'multi-agent' which hints towards a connection with game theory.

2.1.1 Reinforcement learning

RL, at its essence, is about an agent interacting in an environment in order to learn to behave in order to maximise a reward function. The agent takes actions sequentially inside the environment and observes the quality of these actions by receiving rewards and reaching new states based on the transition dynamics of the environment. If a state-action pair is associated with favourable outcomes, the tendency of the agent to select the action in the corresponding state is reinforced and vice versa if the outcome of a state-action pair is poor. Eventually, the agent learns to behave better than random play based on its experience of the environment. In practice, this means the agent must interact several times with an environmental state so that it learns about the quality of all possible actions in that state which means the algorithm may require substantial computing resources to train. Therefore, the environment the agent interacts with is often simulated and highly stylized such as a game with a clear notion of possible states, actions, and transitions between the two.

However, the environment does not have to be simulated and notable applications include agents controlling cars based on their perception of the surrounding environment via sensors such as cameras. This information is then interpreted by the vehicle as a state. The vehicle then uses information about the state to take an action which leads to a new state and potentially a reward. In a price setting context, the algorithms might observe features of the economic environment, such

[1] An example of a loss function is the mean squared difference between predictions and actual values. This concept will become important later.

as previous price levels, rivals' price levels, demand for the product, and similar characteristics, and use this information to make decisions on what the price should be in the next time period (e.g. a second) which leads to the possibility of a reward (customer purchases the product), and a new state.

Irrespective of whether the environment is a simulated game or not, the environment of an RL problem is thought of as having a transition function which is a mapping between actions and outcomes that dictate the probability of reaching a state given that a certain action was taken in a given state. More specifically, the transitions are probabilities of reaching new states, $s_{t+1} \in \mathcal{S}$, in time $t+1$ given that the agent was in state $s_t \in S$ and took action $a_t = a \in \mathcal{A}$ at time t.[2] Both $\mathcal{A}$ and $\mathcal{S}$ denote *spaces* known as *action space*, and *state space* respectively. These spaces may be either continuous or discrete. Altogether, the transition function is defined as a probability density function: $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. This function can be governed by a simulation or the real world, depending on application, and the goal of an agent is to learn enough about the transition function in order to be able to approximate the reward maximising behaviour in a given state.

The way in which the RL agent learns to maximise its reward is by gathering experience through repeated interaction with the environment, i.e. learning. The learning model is loosely inspired by how animals learn and is akin to teaching a pet a desirable behaviour by rewarding it whenever desired behaviour is observed. See Sutton & Barto (2018, Chapter 14) for more about the link to psychology and animal learning. Each time the agent selects an action, and thereby performs a *step* in the environment, it observes the ensuing state space transition and receives a reward $r_t \in \mathbb{R}$. Depending on the algorithm, the agent updates its knowledge of the transition dynamics and, an estimate of a *value function* which estimates how good a state is to be in.

The value of a state depends on future streams of reward that the agent thinks it will obtain from that state based on the agent's current *policy*, π. A policy is formed as the agent experiences transitions in the environment and the ultimate goal of the agent is to form a policy that maximises the expected present value of a stream of rewards. More formally, according to Sutton & Barto (2018, p.45): 'a policy 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 the probability that $A_t = a$ if $S_t = s$'.[3] Note the similarity with a game theoretic *strategy*, which is defined as a complete mapping between the set of possible histories (states) to the set of possible actions for all stages of a game. In contrast to game theory though, the focus in RL is more on learning the optimal behaviour of repeated Markov decision processes (discussed in the next paragraph) than finding Nash equilibria of games, though there are many similarities which will be discussed below.

Because the transition dynamic function, $T(s_{t+1}|s_t, a_t)$, conditions only on information available at time t, the *Markov property* is fulfilled; which basically states that the current state, s_t, is as useful for predicting the future as the full history of states and actions. See Sutton & Barto (2018, p. 381) for a formal definition. Furthermore, because of the Markov property, the problem of maximising a discounted stream of rewards described above follows a *Markov decision process* (MDP). An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T(s_{t+1}|s_t, a_t)$ is the transition function, $R(s, a)$ is the reward function, and γ is an optional parameter that discounts future rewards. The MDP is considered solved when a policy, $\pi^*(s, a)$, has been found that maximises the cumulative discounted rewards.

[2] s_t, and a_t denote state and action at time t. Sometimes, the current state and action are denoted as s, and a in the literature whereas s', and a' denote the next time period state and action.

[3] Capital letters denote random variables.

2.1.2 Q-learning

There are many different RL algorithms and this thesis focuses on one main value iteration algorithm that was first introduced in Watkins (1989) and is known as one step Q-learning. The Q-learning algorithm seeks to solve an MDP by estimating the optimal state-action value function,

$$Q^*(s,a) := \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi], \tag{1}$$

where the optimal action-value function is the function that maximises the expected present value of a stream of rewards by following a policy, $\pi = P(a|s)$, given that a certain state, s, has been observed. Again, γ is a discounting factor, and r_t is the reward at time t.

The Q-learning algorithm learns the transition dynamics function by repeatedly visiting state-action pairs and updating the value of these pairs using the observed reward and the previously estimated value of the next state following an updating schedule defined in Sutton & Barto (2018, p.105) as follows:

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\Big). \tag{2}$$

Each time the agent takes an action at a given state, the agent observes and updates the value of this state based on what reward it receives and the current estimate of the highest obtainable value in the next state. However, only a fraction, α, called *learning rate* is internalised in each iteration. This hyper-parameter determines how much of previous information should be kept and thus regularises noise. With values close to one, the agent quickly overwrites past experiences, while values close to zero means the agent remains stubborn in its belief about the environment and only changes slowly.

A feature of Q-learning is that it belongs to a class of *off-policy* learning algorithms. By being able to learn off-policy, the Q-learning algorithm is able to generate behaviour that is unrelated to the policy that is evaluated and learned, the *target policy*. This has the advantage that Q-values are estimated as the sum of discounted future rewards without requiring the algorithm to ever follow such a policy. Instead, the algorithm is able to explore state-action pairs while the optimal policy is being updated which, according to Sutton & Barto (2018, p. 105), enabled early convergence proofs by simplifying analysis.

Importantly, the optimal action-value function $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ obeys the Bellman equation:

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}\Big[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})|s_t, a_t\Big], \forall t \in [0, T], \tag{3}$$

namely that the value of a state under an optimal policy is equal to the expected return from the best action of that state. In fact, Equation 3 is a system of equations with one equation for each state (Sutton & Barto 2018, p. 51) and can thus be solved using value function iteration assuming the dynamics of the environment are known, there is enough computational resources, and that the Markov property is fulfilled. In RL, it is often the case that at least one assumption is violated which is why approximate solutions such as those given by Q-learning are motivated.

A useful intuitive representation of the action-value function is to view the Q-function as an $|\mathcal{A}| \times |\mathcal{S}|$ matrix, Q where each pair of state and action has an associated Q-value. Furthermore, for an economist s_t can be thought of as the current stock of capital, k_t, and the choice, a_t, can be thought of as the choice of capital tomorrow, k_{t+1}, and it becomes evident that Q-learning may be applied on problems traditionally solved using value function iterations and dynamic program-

ming. In fact, Q-learning is a combination of dynamic programming and off-policy Monte Carlo simulations, which can both be used to solve MDPs. First, like dynamic programming, Q-learning bootstraps its estimates, meaning that the value of states are updated based on estimated values of successor states. For Q-learning, the bootstrapping manifests itself in the use of the max operator in Formula 2. Second, like Monte Carlo, Q-learning is able to learn about the transition function without a model of the environment's dynamics. It does so by averaging returns occurring after visits to each state as an estimate of the value of that state. After enough visits to each possible state, the estimates converge in probability to the expected value of the state. See Sutton & Barto (2018, Chapter 6) for details.

2.1.3 Function approximation using neural networks

Before discussing feedforward neural networks in the context of reinforcement learning, they are first presented in their most canonical form, namely as a flexible function approximators used for classification in supervised learning tasks. The presentation of neural networks is based on Chapters 6 ('Deep Feedforward Networks'), and 8 ('Optimization for Training Deep Models') in Goodfellow et al. (2016).

The goal of a feedforward neural network is to approximate an underlying function, f^*, such as a classifier, $f^*(\mathbf{x}) = y$, which takes independent variable data, $\mathbf{x}$, and predicts either a category, or a value of a dependent variable y. When predicting values, the problem is a regression problem and the underlying idea is the same as with linear regression, namely that the underlying function, $f^*(\mathbf{x}) = y$, can be approximated. What sets feedforward neural networks apart from linear regression and other classification and regression techniques is the networks' ability to approximate highly non-linear underlying functions, which makes them generally more widely applicable. For instance, the input to a neural network could potentially be non-traditional data such as sounds or images. A common task is to link such input to a prediction of a category, where an introductory level problem is to predict hand written digits following Ciregan et al. (2012).

A feedforward neural network refers to a neural network model where all information flows in one direction. The start of the flow happens with the input data, $\mathbf{x}$, which subsequently is transformed by computations that define the function being evaluated, f. The flow ends with an output value, $\hat{\mathbf{y}}$, that is potentially vector valued. Unlike recurrent neural networks (RNN), information does not flow from the output value to the input value. When the information does flow back from the output to the input, the network can be used to model time series as RNNs incorporate information about past values, such as lagged values. However, in this thesis, the focus is on feedforward neural networks and these will from here on be referred to as 'neural networks' or simply as 'networks'.

Neural networks can be used for supervised learning, where the goal is to minimise a loss function, $L(\theta)$, by fitting a vector valued parameter, θ. Going back to the analogy of linear regression, fitting the parameters is akin to estimating linear coefficients by the method of ordinary least squares (OLS), although the optimisation tool used is more involved than OLS, see below. If the neural network, $f(\mathbf{x}|\theta) = \hat{\mathbf{y}}$, is a good representation of the underlying function, $f^*(\mathbf{x})$, and we wish to predict $\mathbf{y}$ of the underlying function, then the output values $\hat{\mathbf{y}}$ should lie close to the true values, $\mathbf{y}$. How well the neural network represents the underlying function is determined by a loss function where a typical loss function is the sum of squared residuals in the test set: $L(\theta) = \sum_{i=1}^{n}(\hat{y}_i - y_i|\theta)^2$ that the optimiser tries to indirectly minimise by optimising on the loss function applied to the train set.

As briefly mentioned above, machine learning strives to minimise risk, namely out of sample

performance, by minimising an objective function based on training data. Consequently, the objective function is often a function of the observations in the training set and can be rewritten as an expectation, such as the expected mean squared error. Because computing expectations over the full training data set can be expensive, an often used approach is to sample data from the training set into a *minibatch* and compute expectations based on data in the minibatch instead. The reason this yields speed increases is that the standard error of the mean over n samples, $\sigma_{\bar{X}} = \frac{\sigma}{\sqrt{n}}$, decreases less than linearly with more data. At the extreme, some optimisation algorithms use what is so called *stochastic* optimisation, where the size of the minibatch used to calculate the expectation equals unity. In practice, the size of the minibatch tend to fall between one and the size of the training set.

Algorithms used for the optimisation task include *stochastic gradient descent* and versions of it such as *Adam* (Kingma & Ba 2014). The former algorithm, stochastic gradient descent, is an approximation to *gradient descent*, which minimises the loss function over a training set by calculating the gradient, which is a vector of partial derivatives with respect to the parameters that are being optimised over, of the loss with respect to the parameters over the full training set, $\nabla_\theta L(\theta)$ (Goodfellow et al. 2016, p.82). The idea behind gradient descent is to change the values of the parameters in the opposite direction of the gradient, in order to travel 'downhill' on the loss function, from one loss to another lower loss according to the gradient according to: $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$, where α denotes the learning rate. When gradient descent is made stochastic by estimating the gradient based on a random sample drawn from the training data, stochastic gradient descent is obtained. As the name suggests, it is an instance of a stochastic optimisation algorithm discussed above and therefore brings computational benefits in that it is faster to compute multiple estimates of the gradient, than to compute fewer exact gradients.

The other optimisation algorithm mentioned, Adam, builds upon stochastic gradient descent and incorporates *momentum* and *adaptive learning rates* for the individual parameters. Momentum means that previous gradient estimates exhibit some inertia and influences the direction of the change in the parameter estimates so that the information used to guide the direction of the update is based on historic gradients in addition to the most recent gradient using an exponentially decaying weighting scheme. This stabilises training as gradients estimated with a small minibatch tend to have a high variance. An adaptive learning rate means that the algorithm adjusts the learning rate according to the gradient which can speed up learning in gentle sloping areas of the loss function. In addition to these features, Adam also incorporates second derivatives to guide the parameter search. Fortunately, the algorithm is robust to hyper-parameters and has therefore become a popular algorithm (Goodfellow et al. 2016, pp. 309-310).

A challenge optimisation algorithms (such as Adam) are developed to overcome is that deep learning applications often involve non-convex optimisation.[4] In non-convex optimisation problems, the method of simple gradient descent can lead to stabilisation in a local minimum or at a saddle point because by definition, these are points where the gradient is the zero vector which dictate that the algorithm has nowhere to proceed 'downhill'.

Irrespective of the optimisation algorithm chosen, the algorithms tunes parameters, θ that transform the input data inside the neural network. It works as follows, input data, $\mathbf{x}$, enters the network and is subsequently transformed by an *activation function*, before it enters the next layer, typically a *hidden layer*, where the data is again transformed by another activation function. The data continues to enter hidden layers depending on the depth of the neural network until the data reaches the final layer, called the *output* layer containing predictions $\hat{\mathbf{y}}$ (such as probabilities of vari-

[4] 'Convexity' is a beneficial property of an optimisation problem and means that a global solution can be found in what is called 'polynomial time'.

ous categories or a predicted value in a regression). For example, with three layers, $f^{(1)}, f^{(2)}, f^{(3)}$, the network can be defined as a chain of vector valued functions $\hat{\mathbf{y}} = f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$.

Between each layer, data from all nodes in the previous layer (raw data in the first layer) enter into the next layer as seen in the chain above. Typically, a linear combination of transformed values in the previous layer is used to compute values for each of the nodes in the next layer according to $f(\mathbf{x}|\theta) = f(\mathbf{x}|\mathbf{w}, b) = \mathbf{x}^T\mathbf{w} + b$. and the loss function is minimised with respect to $\mathbf{w}$ and b. However, in order to be able to learn non-linear functions, a non-linear transformation of the data, is applied after linearly combining the data into the next layer, which is handled by the activation function. The current recommended default activation function is known as *rectified linear units*, ReLus, defined as $g(z) = \max\{0, z\}$, where z is a linear combination of values in the preceding layer and are usually an acceptable default choice for activation function (Goodfellow et al. 2016, p.191). To summarise, data flow from an input layer to a subsequent layer using a linear combination of all the data in the previous layer, an activation function is applied and the procedure continues onto the next layer until the output layer is reached and a prediction, $\hat{\mathbf{y}}$, is obtained.

At this point, initial parameter values have been assumed and at the beginning of training, a random initialisation of the parameters is required. However, after the data has flown to the output layer and produced an output, the loss function can be computed which can then be used to update the parameters θ according to the optimisation algorithm. The procedure of shifting data back from the output layer is known as *backpropagation*. Backpropagation is an algorithm developed in Rumelhart et al. (1986) that computes the gradient $\nabla_\theta L(\theta)$ which contains the information the optimiser uses to learn parameter values, namely an approximation to the gradient of the loss function with respect to the parameters being tuned. Once the parameters have been updated, a new iteration can begin from which the algorithm can improve upon again.

Before proceeding, it should be mentioned that the choice of network architecture is up to the researcher. Choices include number of hidden layers, number of hidden units in each hidden layer, activation function, form of the model between layers, and cost function; but can be substantially more involved as neural networks can be flexibly applied. What the neural network learns by itself is the parameter values of the form of the model between the layers. For a visual representation of the network used in this thesis, refer to Figure 2. Additionally, to complete the parable with linear regression, refer to Figure 11 in Appendix A where linear regression is characterised as a simple neural network without activation functions and hidden layers.

2.1.4 Combining Q-learning with deep learning

When Q-values for the entire state-action space, $\mathcal{S} \times \mathcal{A}$, cannot be represented by a table, such as when observations are continuous and the number of possible states grows to infinity, $|\mathcal{S} \times \mathcal{A}| \to \infty$, an approximation to the Q-function may be necessary. In deep Q-learning, the idea is to represent the Q-function with a deep neural network, $f(\mathbf{x}|\theta) = Q(s, a|\theta)$. The objective of the neural network is to fit a vector valued parameter, θ, of the network such that an objective loss function is minimised. A typical loss function is the mean squared error, which sums over the n iterations, i, that have previously been experienced:[5][6]

[5] Although, it is more common to use a minibatch of recent transitions as discussed previously.
[6] Here, I make use of the notation where a', and s' denote values of the next time period.

$$L(\theta_i) = \frac{1}{n}\sum_{i=1}^{n}(y_i^{DQN} - Q(s,a|\theta_i))^2 \tag{4}$$

$$\text{where } y_i^{DQN} := r + \gamma \max_{a'} Q(s',a'|\theta_i), \tag{5}$$

although other loss functions exists for different problems. Since the actual target value is unknown, which is the reward maximising Q-function we intend to find, the current best estimate of the Q-value is used instead, y_i^{DQN}. The regression problem is thus to predict values of an estimated Q-function using the reward and the discounted estimated Q-value for the next step according to the model relation in Equation 6:

$$r + \gamma \max_{a'} Q(s',a'|\theta_i) = Q(s,a|\theta_i). \tag{6}$$

The data used in the regression is data generated by the algorithm as it interacts with the environment and experiences transitions and thereby obtain rewards. Therefore, the algorithm has access to more data the longer the agent has been trained. Additionally, note that the regression problem is to predict an estimate that may be both wrong and moving since we do not know the true outcome. The full naïve deep Q-learning algorithm, which is called naïve for reasons explained below, is given in Algorithm 1:

Algorithm 1: Naïve deep Q-learning

Initialization;
Initialise $Q(s,a|\theta_0)$ with random parameter θ
Initialise random parameter for target network: θ^-
for *episode* $= 1, M$ **do**
  for $t = 1, T$ **do**
    With probability ε select a random action: a
    Otherwise select $a \leftarrow \arg\max_a Q(s,a|\theta)$
    Obtain tuple: $(s,a,r,s') \leftarrow \text{envstep}(a)$
    Calculate loss: $L(\theta) = \frac{1}{n}\sum_1^n \left((r + \gamma \max_{a' \in A} Q(s',a'|\theta^-)) - Q(s,a|\theta)\right)^2$ or
    $L(\theta) = (Q(s,a|\theta) - r)^2$ if episode has ended.
    Update θ using an optimisation algorithm by minimising $L(\theta)$) with respect to θ
    $\theta^- \leftarrow \theta$
  end
end

Unfortunately, the direct combination of neural networks and Q-learning in Algorithm 1 has no convergence guarantees. This result is known as the *deadly triad*. (Sutton & Barto 2018, p.215)

The deadly triad refers to the instability that occurs when three elements are combined. These elements are function approximation, such as neural networks; bootstrapping, i.e. extrapolating the value of an action instead of estimating it using Monte Carlo techniques until the end of the episode; and off-policy training, the training on a distribution of transitions rather than data following an actual policy trajectory. Q-learning relies on the last two elements and neural networks adds the first element.

However, a seminal paper by Mnih et al. (2015) overcomes the deadly triad by applying techniques that address the main issues of non-convergence, showing there is a gap in the understanding of this issue. The challenge in the paper is to achieve super human performance in classic Atari

games such as *pong*, where a ball bounces between two paddles until one paddle misses the ball and the player controlling the other paddle makes a score; *Video Pinball*, where a ball bounces between a ceiling of bricks and a player controlled paddle; and many more classic Atari games. For most of the games tested, DQN is shown to perform well enough to beat any human after sufficient training. As in Algorithm 1, the Q-function is represented by a neural network parameterised by θ such that θ minimise a loss function. Crucially, the authors exploit two techniques, *experience replay* and *target network*, and apply these to Algorithm 1.

The first technique, experience replay, addresses the issue that the data used to solve the regression problem in Equation 6 is correlated and therefore not independent and identically distributed. The dependence in the data occurs because observations close in time share common characteristics such as the policy trajectory being followed by the agent and the states considered. Because the dependence in the data is mainly temporal, the problem can be overcome by sampling transitions randomly from a fixed number of previous iterations into a minibatch that is used as a sample with which parameters are fit and used to update preexisting parameters. In addition uncorrelating the data, this technique also emphasize recent data, as the minibatch is drawn from recently experienced transitions stored in a replay buffer, D_t. The data set has a time subscript, t, because it adds experience after each time step. Additionally, data may be discarded over time by introducing a *replay size*, which functions as a sliding window which keeps the most recent experiences. Overall, the introduction of experience replay changes the expectation of the loss we are maximising in Equation 4 and instead of summing over all iterations, the expectation is computed using the minibatch.

The second technique, target network, addresses the issue that the learning algorithm has to 'chase a moving target' by introducing a second neural network, the target network, which is regularly synchronised with the training network. This again changes equation 4 and the loss function at iteration i ultimately becomes:

$$L(\theta_i) = \frac{1}{K} \sum_{(s,a,r,s') \sim U(D_t)} \Big(r + \gamma \max_{a'} Q(s, a|\theta_i^-) - Q(s, a|\theta_i) \Big)^2, \tag{7}$$

where the main difference is that the predicted value does not update after every iteration as the target network parameters θ^- is held fixed for a predefined period of length C, before being synchronised with θ, which continues to update after each episode. Additionally, the summation is now over K experiences drawn uniformly from the data, D_t, available at time t where K is used to denote the size of the minibatch.

After the publishing of Mnih et al. (2015), several authors made contributions which individually showed improvements on the Atari games studied in Mnih et al. (2015). A paper testing the combination of these techniques where published in Hessel et al. (2018) and shows that performance can be greatly enhanced by combining these ideas. Notably, double deep Q-learning (DDQN) shows a large gain in performance while being easy to implement. For brevity, the discussion that follows is limited to DDQN, even though there are many more promising techniques presented in Hessel et al. (2018).

DDQN, first proposed by Van Hasselt et al. (2016), addresses an issue of overestimation of Q-values that occurs when an agent is selecting a maximising action. The problem of overestimation occurs because any Q-value estimate is a noisy estimate of the actual Q-value. This means that an attempt to choose a value maximising action will often select actions based on random noise,

which will tend to overestimate the value of the maximum Q-value.[7]

Furthermore, since the maximum Q-value is used to estimate the target network, the target network will also tend to be overestimated. The problem with this is that the same noise go into both the Q-values, and the target network, see Equation 8 for a reminder of the maximisation over the action space used to estimate the target values:

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'|\theta_i^-) = r + \gamma Q(s', \arg\max_{a'} Q(s', a'|\theta_i^-)|\theta_i^-) \tag{8}$$

$$y_i^{DDQN} = r + \gamma \max_{a^*} Q(s', a^*|\theta_i^-) = r + \gamma Q(s', \arg\max_{a'} Q(s', a'|\theta_i)|\theta_i^-). \tag{9}$$

In practice, the problem of overestimation diminishes over time as estimates become more precise. However, faster convergence is possible by using the solution of DDQN proposed in Van Hasselt et al. (2016), where the main idea is to decouple the target update from the action selection by introducing a second Q-network. By decoupling the two, noise present in the selection of an action will be decorrelated from the noise of the target value.

Specifically, the idea presented in Van Hasselt et al. (2016) is to use the current Q-network to evaluate the qualities of the actions according to the argmax operator, and the target network used in Mnih et al. (2015) to evaluate the quality of an already chosen action. Because the target network is readily available, implementing DDQN is an easy way to improve performance. Note that in Equation 9, the only difference is that the action is selected from a Q-function evaluated using the current network, $Q(s', a', \theta_i)$, rather than that of the target network, $Q(s', a', \theta_i^-)$.

2.1.5 Multi-agent learning

The previous discussion of RL centers around the case of a single learning agent. While interesting and useful in its own right, it is necessary to go beyond that and introduce the concept of multiple simultaneous agents in order to be able to study social dilemmas. While most of the concepts relating to a single learning agent introduced still applies, it is also the case that the addition of one or more agents introduces fundamentally new challenges. Furthermore, the dynamics between the agents provides a link to game theory, and the field of multi-agent reinforcement learning can be said to lie in the intersection of reinforcement learning and game theory. The discussion below is based on a survey article by Zhang et al. (2019) which goes deeper than this thesis and provides further references to more specialised topics.

As in the single agent case, the goal of each individual MARL agent is to select actions such that a future stream of rewards is maximised. The environment is learned about by sequentially interacting with it, and therefore also interacting with other agents contained in the environment. The mere presence of other learning agents causes the environment to become *non-stationary* meaning that the environment facing each agent is changing over time. This effect is exacerbated when actions taken by the agents have a strong impact on rewards and state transitions for each of the other agents, rendering learning difficult as old information becomes invalidated as the other agents develop their behaviour. The induced non-stationarity invalidates the earlier proofs of convergence for single agent learners as the Markov property no longer holds as the next state and reward depend on more influences than the current state and action.

An *independent learner* that ignores the issue of non-stationarity and strives to maximise its own long run discounted stream of rewards regardless of the other learners may thus fail to

[7] In the case of two random variables, X_1, and X_2, we have that $E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$, which is the reason an overestimation bias occurs in the case of selecting actions by maximising over random Q-value estimates.

converge (Tan 1993). However, the lack of theoretical guarantees of convergence does not exclude the possibility of convergence; empirically, it is known that independent learners may nonetheless converge towards some policy. Notably, even MARL problems with large state-action spaces such as *StarCraft II*, a real time strategy computer game considered one of the most challenging games to solve, have been solved to the extent that an agent is able to beat even the best human player using raw data fed into a deep neural network (Vinyals et al. 2019).

Apart from the issues associated with non-stationarity, there are further issues involved in multi-agent learning. One such issues is the *combinatorial nature* of MARL which refers to the combinatorial increase of the joint action space as more agents are added to the environment. This makes learning harder since more possible joint actions means more transitions need to be observed before the dynamics of any state-action combination can be estimated accurately. Another issue is the issue of modelling the *information structure*, which refers to the agents' awareness about each other, which opens up the possibility of endless chains of knowledge. For instance, an agent might be aware of another agent's policy, but not being aware to the fact that the other agent is aware of its awareness, and so on up until a degree of awareness. With independent learning, this problem is ignored and there is no explicit awareness of other agents.

Because of the presence of more than one agent, a MARL algorithm no longer aims to solve an MDP, which is an insufficient description of any MARL problem as an MDP does not allow for multiple agents. Instead, there are two main generalisations to MDP used for the study of MARL problems. These are *Markov Games* (MG), and *Extensive form games*. However, for the purpose of this thesis, it is enough to discuss the case of MG. As with an MDP, the MG is a tuple that contains the already discussed components $\mathcal{S}$, $\mathcal{A}$, P, R, and γ with the addition of $\mathcal{N}$, which is the set of agents. To allow for differences among the agents, the action spaces and reward spaces may be individual to each agent. However, the state space, the transition function remain shared because the agents operate in the same environment. An MG is thus defined as a tuple $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{N}}, P, \{R^i\}_{i\in N}, \gamma)$ where the action space is defined as the cross product of all the individual action spaces, $\mathcal{A} := \mathcal{A}^1 \times \mathcal{A}^2 \times ... \times \mathcal{A}^{|\mathcal{N}|}$, and a *joint action*, $a \in \mathcal{A}$, is what together with the state space and transition function determines the environmental transitions. That is, the environment takes a joint action, a_t, and the current state, s_t, to determine s_{t+1}, and r_t. In contrast to single agent learning, the action of one agent is thus not the same as the action in the environment. This means that both agents may unilaterally change outcomes at environmental steps, and that no agent has full control in the sense that one action in a given state may lead to different transition probabilities depending on the other agent's action.

There are three main settings into which an MG can be classified. These depend on the reward process and are either fully cooperative, fully rivalrous, or a mixture between the two. In the cooperative setting, the agents' reward functions correlate positively, so that what benefits one agent necessarily benefits the other agents as well. Usually, the reward function is common among the agents and in the game theory, such a game is known as a *team Markov game*. Not all cooperative games need the reward function to be shared, and heterogeneous agents is possible and typically require a way of communicating between the agents. The other extreme setting, that of a rivalrous game can be thought of as a zero sum game, such as chess, where the rewards of the agents are perfectly negatively correlated which makes such games directly competitive. In between cooperative and rivalrous games are general sum games, where agents can enhance each others payoff but have no intrinsic motivation to do so unilaterally. This means that social dilemma, such as the prisoner dilemma, belong to this category of games.

The algorithm that arise when combining Q-learning with deep learning and allowing for two agents is given in Algorithm 2. Apart from stabilising Algorithm 1 using a replay buffer D, target

networks, and double deep Q-learning, it also allows for several agents. In this algorithm, the agents are assumed to be similar, sharing the same network and replay buffer. In a price setting context, the number of episodes, M, can be set to 1. However, when updating the firms by, for instance, changing their degree of vertical differentiation, it is necessary to introduce several episodes since otherwise the agents might erroneously relate transitions to new levels of vertical differentiation based on its actions. The notion of an infinitely repeated game can be preserved by terminating episodes with a small probability.

Algorithm 2: Multi-agent double deep Q-learning

Initialization;
Initialise the memory of D to capacity N
Initialise $Q(s, a|\theta)$ with random parameter θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
Initialize a random state, s'
for *episode* $= 1, M$ **do**
  for $t = 1, T$ **do**
    $s \leftarrow s'$
    if $t \mod 2 = 0$ **then**
      $i \leftarrow 0$
    end
    else
      $i \leftarrow 1$
    end
    With probability ε, $agent_i$ selects a random action: a_i
    Otherwise select $a_i \leftarrow \arg\max_{a_i} Q(s, a_i|\theta)$
    Form joint action: $a = (a_0, a_1)$
    Obtain tuple: $(s, a, r, s') \leftarrow \text{envstep}(a)$
    Store transition in D
    Sample minibatch of K transitions (s, a, r, s') from D # Experience replay
    Calculate loss, $L(\theta))$, on the minibatch using mean squared error.
    Update θ using an optimisation algorithm by minimising $L(\theta))$ with respect to θ
    Every C steps, update $\theta^- \leftarrow \theta$ # Update target network
  end
end

2.2 Social dilemmas and pricing

Beside the machine learning literature on multi-agent reinforcement learning, important contributions come from game theory and this subsection provides a background on relevant parts of game theory.

2.2.1 Game theory and cooperation in social dilemmas

In the most canonical form of a social dilemma, a *Prisoner's Dilemma* (PD), the game theoretic prediction is that both prisoners defect, leaving both worse off than had they cooperated. The reason the prisoners are predicted to defect is because it is individually rational to defect independent of the other player's action. Since the PD is a symmetric game, both players find that defecting is the rational choice. Despite this prediction, people seem to behave cooperatively in many social interactions.

Therefore, a long standing issue in evolutionary theory and social science has been to understand how cooperation can arise in an asocial world. Perhaps the most important paper in the area is Axelrod & Hamilton (1981), which addresses the issue by using the concept of *evolutionary stable strategy* to show that cooperation can be stable, even in PD styled games, when players are using strategies based on punishing asocial behaviour. In economics, another explanation of cooperation has been reached through the theory of repeated games. For instance, the *folk theorem*, see for instance Fudenberg & Tirole (1991, pp. 150-160), asserts that with discounting sufficiently close to unity in infinitely repeated games, there exists a Nash equilibrium arbitrarily close to any feasible outcome of the game, such as a fully cooperative outcome. The intuition behind that fully egoistic and rational players may chose to not deviate from cooperation is that any finite one period gain from deviation will be outweighed by small losses in subsequent periods, given that players are patient enough.

The crucial component behind the folk theorem is the condition of an infinitely repeated game, or at least that the game has no known end. Without this assumption, rational players solve the game using backward induction and reason that in the last PD subgame it is rational to defect. This means that the players in the penultimate subgame know that no matter their actions in the current subgame, the next subgame will see both players defect making it rational to defect also in the second to last game. The chain continues until the first subgame and results in both players always defecting. With infinitely repeated games, there is no end from which backward induction can begin which makes cooperative strategies possible. There are several strategies that can lead to cooperative behaviour, these are discussed in the next section.

A refinement of the theory of repeated games for dynamic oligopolies where price is taken to be the strategic variable is developed in Maskin & Tirole (1988). A model is developed where identical duopolists set prices sequentially and commit to a price in the short run. The game is modelled as an MG which means that the firms condition their prices on the observed price of the other firm, which remain fixed until the next period due to the short run commitment. Both firms aim to maximise the present value of future profits for an indefinite amount of time. It is shown that in the game described above, the equilibrium must either be an *Edgeworth cycle* or of the *kinked demand type*. In an *Edgeworth cycle* (Edgeworth 1925) the duopolists undercut each other's prices until a higher price level is restored by one of the firms and the process of undercutting resets. It is therefore a cyclical equilibrium in which the market price never settles down, but it is always at least above the marginal cost, which is the static equilibrium prediction of a standard Bertrand duopoly. In contrast, if the equilibrium is of the *kinked demand type* (Sweezy 1939), prices settle at a focal point after a finite time. Maskin & Tirole (1988) shows that 'there exists a multiplicity of kinked demand curve equilibria', and that these equilibria lie on a range including the monopolist price, all equilibria being well above competitive levels. Furthermore, only the monopolist price equilibrium is said to be *renegotiation proof*, which simply means that neither firm finds it advantageous to move to another equilibrium.

2.2.2 Collusion

Collusion refers to a situation where firms avoid competition by agreeing on how to conduct themselves on the market. It may involve market splitting or agreements to charge higher than competitive prices. Because it involves agreements to cooperate, collusion is an instance of cooperative behaviour and can be seen as a social dilemma. Collusion may be either explicit, in which case the collusion is an illegally formed cartel, or tacit, which refers to a situation where firms charge high prices without an explicit agreement. According to Belleflamme & Peitz (2015, p.335),

the consensus among economists is that collusion is a harmful practice as it harms consumers more than it benefits colluding firms. However, in most jurisdictions, only explicit collusion is illegal as firms colluding tacitly merely exercise a right to set prices freely.

Fortunately, collusion is often inherently unstable owing to the fact that there are strong incentives to deviate from collusive prices. First, competition authorities often work with leniency programs, which benefit whistle-blowing firms while harming its competitors. Second, because prospective collusion can often be described as social dilemma, the best response for any given firm is typically to deviate by setting a lower price. Furthermore, for some industries monitoring is difficult which means that the probability of being detected when cheating is low, which creates incentives to cheat as detection is typically associated with some form of punishment.

However, there are cases where collusion is stable and even a Nash equilibrium. To be stable, the cartel must be incentive compatible, namely that the expected present value of staying in the cartel exceeds the expected present value of defecting for all firms involved, $V^C > V^D$. To make the cartel more stable could then involve reducing the value of defecting, V^D, by punishing firms that are observed to defect by introducing a period of lower prices. Strategies that include punishment of defectors include strategies known as *grim trigger* which means all firms resort to equilibrium pricing once a defection has been detected, and *tit for tat*, under which strategy players reciprocate the behaviour of the other player's behaviour in the last period.

Another important component to a stable collusive agreements is that future profits are valued highly enough in order to make the immediate reward of defecting less lucrative in comparison to the delayed reward of the going cartel. Therefore, the rate at which future profits are discounted, the discount factor, play an important role. As a simple example to illustrate the trade off that each colluding firm must make when other firms are following a grim trigger strategy, consider the following:

$$V^C = \sum_{t=0}^{\infty} \gamma^t \pi^C = \frac{\pi^C}{1-\gamma} \tag{10}$$

$$V^D = \pi^D + \sum_{t=1}^{\infty} \gamma^t \pi^N = \pi^D + \gamma \frac{\pi^N}{1-\gamma} \tag{11}$$

$$V^C > V^D \iff \gamma > \frac{\pi^D - \pi^C}{\pi^D - \pi^N} \tag{12}$$

where π^C is the collusive profit, π^D is the profit of defecting, and π^N is the competitive profit received during the punishment phase that is triggered after a deviation has been observed. Note that the example includes an infinite sum, which is a requirement of a cartel as any game of finite duration can be solved using backward induction to the only equilibrium which is to play competitively. The inequality 12 shows that the discount factor γ needs to be sufficiently high to make cooperation incentive compatible, i.e. that $V^C > V^D$ for the given example.

2.2.3 Algorithmic pricing

The advent and current use of pricing algorithms have been thoroughly reviewed by The Competition and Markets Authority (2018). The authors of the report find that the use of pricing algorithms is widespread, where 'pricing algorithm' is defined broadly to include any software used to set prices. The authors additionally highlight the concern that pricing algorithms can be used to coordinate outcomes towards higher prices without explicit agreements but note that most studies leave the choice of algorithm exogenous. Finally, a conclusion regarding whether pricing algorithms

can lead to tacit collusion is that the algorithms may be 'the last piece of the puzzle' leading to tacit collusion for markets already prone to collusion (The Competition and Markets Authority 2018, p.31). Markets deemed prone to collusion include those that share some characteristics that facilitate collusion such as transparent outcomes, a low and fixed number of firms, and multi market contract.

The use of pricing algorithms is known to be prevalent in online market places such as Amazon The Competition and Markets Authority (2018, p.17), where third party sellers can deploy pricing algorithms developed by firms specialising in algorithmic pricing. Chen et al. (2016) analyse the the top 1641 best selling products on Amazon and by doing so about 30 000 sellers and find that about 500 sellers made use of some form of automatic pricing algorithms, which may be as simple as rule based algorithms based on the price of rival firms. In the extreme, such pricing can lead to bizarre prices, such as a textbook about flies that was priced at USD 24 million on Amazon (Solon 2011).

There are four main ways in which algorithms may form collusive prices (Ezrachi & Stucke 2017, p.1782), *messenger*, *hub and spoke*, *predictable agent*, and *digital eye*. The first, *messenger*, works on behalf of a human request to collude and restrict competition, it is therefore a method to implement explicit collusion as the algorithms merely convey an illicit agreement made by humans.

The second way, hub and spoke, refers to a situation where prices for all firms are set by a common algorithm, often provided by a common supplier, the 'hub'. In itself, vertical agreements between a firm and a company providing access to pricing technology is not illegal, but it could potentially be used as a tool for forming a cartel and in particular, the developer could help with coordinating an industry wide cartel in which case the vertical agreements may be deemed to have intent to change market dynamics and may thus be prosecuted.

The third category, *predictable agent*, considers agents that behave in a predictable way. Such agents are well understood by rival firms which may then set prices to accommodate the pricing agent and expect a cooperative retaliation. Adopted widely and predictable agents could lead to industry wide collusion made possible by the transparency brought by the pricing agents. While each individual agent's price increases may themselves be rational responses, the deployment of such algorithms may be possible to prosecute if an intention to change market dynamics can be proven, according to Ezrachi & Stucke (2017, p.1783).

Under the fourth and final category, *digital eye*, a unilaterally deployed algorithm learns how to optimise profit by experimenting with prices. According to Ezrachi & Stucke (2017, p.1783): 'The machines, through self learning and experiment, independently determine the means to optimise profit'. This category represents the most tricky category of algorithms to prosecute according to Ezrachi & Stucke (2017, p.1783) as the algorithms are merely implemented by humans to price with the sole objective of profit maximisation, which is not illegal *per se*.

The classification of pricing algorithms described above presents a hierarchy where the categories line up according to how difficult they are to prosecute. Of the three categories that can lead to collusive outcomes—hub and spoke, predictable agent, and digital eye—hub and spoke poses the most immediate risk according to The Competition and Markets Authority (2018, p. 31) because of the simple requirement that firms adopt the same pricing technology along with the presence of a natural hub, namely the provider of the technology. The other two categories of pricing algorithms could also lead to tacit collusion 'if the pricing algorithms were sufficiently technologically advanced and in widespread use. It is unclear how likely they are to materialise at this point' (The Competition and Markets Authority 2018, p.31) and adds that these two models of harm could become important in the future.

3 Literature review

An early paper where multi-agent Q-learning was applied on a repeated PD was Sandholm & Crites (1996). The paper investigates the PD, where the payoff structure of the agents is not entirely collaborative (totally positively correlated payoffs), nor a zero sum game (totally negatively correlated payoffs). Instead, the agents must learn a strategy that promotes collaboration, such as tit for tat, in order to achieve high payoffs. The paper investigates tabular Q-learning agents playing each other and fixed strategy opponents. It also considers a recurrent neural network, a type of neural network that can store an arbitrarily long memory. It was found that agents playing against other learning agents fare worse than those facing fixed strategy opponents, that a long exploration phase promotes collaboration, and that agents equipped with neural networks failed to learn collaboration. The somewhat surprising last result, that neural network players failed to learn to collaborate, was hypothesised to depend on the relatively short learning horizon over which the agents were trained. Deep learners generalise the knowledge, and over a small observation space, such generalisation takes a relatively long time to learn compared to algorithms that do not intend to generalise the knowledge beyond observed transitions.

A paper which examines sequential social dilemmas in a deep multi-agent reinforcement learning setting and does find emerging complex behaviour is Leibo et al. (2017). The authors consider two games, Gathering and Wolfpack, with subgames having PD-type payoffs repeated over multiple interactions. In Gatherer, the goal of two concurrent agents is to maximise their respective number of gathered apples in a given time. The agents have the option to use a laser beam to freeze the other player for a short time and thus gains the opportunity to gather apples undisturbed by its opponent who may otherwise have made it first to the apple. The second game, Wolfpack, is described as a game where 'two players (wolves) [...] chase a third player (the prey). When either wolf touches the prey, all wolves within the capture radius receive a reward. The reward received by the capturing wolves is proportional to the number of wolves in the capture radius' (Leibo et al. 2017, sec. 5.2). A key difference between the two games is whether cooperation is the relatively more complex behaviour, which is the case in Wolfpack but not in Gatherer. The authors analyse how different types of behaviour can emerge in sequential social games after environmental parameters have been changed exogenously and argue that the sequential social dilemma (SSD) is a comparably realistic model of real world social dilemmas.

A Q-learning paper related to pricing algorithms is Tesauro & Kephart (2002). Using tabular Q-learning, the authors study a two firm sequential Bertrand competition in three economic environments. The first variant considered is 'Price-Quality', where the two firms are vertically differentiated leading to an asymmetry in seller profit functions. Second, the authors study 'Information-Filtering', namely the introduction of horizontal differentiation. The last case considered is that with 'shopbot', where some consumers shop at random, and other consumers have full information of the price of two otherwise identical sellers (using 'shopbots'), meaning that it can be profitable to abandon competition over bargain hunting consumers. For the first two scenarios considered, convergence to a stationary fixed point was found. In the case of vertical differentiation, higher prices were upheld by an expectation of price war when undercutting the opponent by pricing according to the best response. In the case of horizontal differentiation, convergence was obtained dependent on the discount parameter γ and lead to higher profits when convergence was found.

Further papers on algorithmic pricing and tabular Q-learning are papers by Calvano et al. (2018) and Klein (2018). As in this thesis, these papers investigate and describe whether Q-learning agents are able to learn to tacitly collude with other Q-learning agents. The papers differ

in that Calvano et al. (2018) uses a simultaneous pricing decision framework whereas Klein (2018) focuses on a sequential pricing duopoly, thereby adapting the sequential move framework of Maskin & Tirole (1988), where firms *commit* to a price for a time duration during which the other duopolist is able to act.

In both papers by Calvano et al. (2018) and Klein (2018), Q-learning agents are trained against each other in a simulated market environment where the agents interact with each other until they have trained for long enough for the Q-values, and thereby the policy, to stabilise. This results in the finding that tabular Q-learning algorithms trained in simulated environments eventually reach collusive prices. In Calvano et al. (2018), the Q-learning algorithms develop the ability to punish another Q-learning algorithm that deviates from collusion and the algorithms generally tend to collude, albeit not at monopoly levels. Meanwhile, in the paper by Klein (2018), where a sequential Q-learning algorithm is assessed, it is found that prices follow Edgeworth cycles. That is, prices follow a cycle that resets the price upwards after having gradually declined due to an incentive to always slightly undercut the opponent until the price is low enough. In both papers, collusion occurs despite the algorithms having no other objective than to maximise profits.

Another paper on algorithmic pricing is Brown & MacKay (2019). This paper considers another aspect of pricing algorithms where the authors develop and calibrate a model where firms compete sequentially after selecting a price algorithm that dictate the frequency at which the firms are able to update their prices. The authors show that it is not an equilibrium outcome for all firms to choose algorithms such that the algorithms behaves according to the price setting best response function according to the Bertrand equilibrium. In particular, the authors find that it is the algorithms' ability to set prices *frequently*, and to *commit* to prices in the short run that turns the game into a sequential pricing game in which prices are higher than in the Bertrand outcome. Hence, asymmetries in the algorithms play a crucial role in establishing the results. However, the authors also establish that even symmetric short run commitments can lead to higher prices when the firms' strategic variable is selecting a pricing algorithm. A policy recommendation made by the paper is that it should be forbidden for algorithms to condition the price on rivals' prices.

The main distinction between this thesis and the previous work by Tesauro & Kephart (2002), Calvano et al. (2018), and Klein (2018) is that this thesis incorporates function approximation in the form of deep neural networks which allows more complex environments to be studied. The addition of deep neural networks is a relevant object to study because neural networks allow the algorithms to learn how to act in a continuous state space. In tabular approaches, the agents have no notion of how to react to states not previously seen, whereas neural networks can accept any input values. In addition, a neural network representation is in principle more capable of taking into account more features (such as observable differentiation) without suffering from the curse of dimensionality that tabular approaches eventually stumble upon. The choice of algorithm is left exogenous, but the discussion relates back to the concept of endogenous pricing algorithms and under what circumstances they may be applied.

4 Method

In this thesis, I test whether reinforcement learning algorithms equipped with neural networks are able to find collusive outcomes in a market competition game of repeated interaction.

The investigation is based on the following logic. A reinforcement learning agent is initiated with a neural network and the network is subsequently trained in a simulated economic environment, where profits and transition dynamics are generated based on the prices of two firms and a predefined demand function. The two firms set prices sequentially using access to the same

reinforcement learning agent, which takes data available to the firm setting its price in period t to set a price. Because the game is set up as a dynamic duopoly with price as strategic variable and alternating moves, it largely follows the setup of Maskin & Tirole (1988, Section 2).

The implementation is thus similar to that in Klein (2018) and this paper in conjunction with Calvano et al. (2018) provide the benchmark on which the investigation is based. However, because of the addition of neural networks to Calvano et al. (2018) and Klein (2018), the space of possible hyper-parameters is substantial and requires a thorough motivation. Consequentially, choices with respect to the neural network are largely based upon Mnih et al. (2015) and Leibo et al. (2017), although deviations are made where deemed justifiable or necessary. With this in mind, the remainder of the section describes the method in further detail, clarifies important decisions, and details the set of parameters chosen.

4.1 Reinforcement learning environment

The reinforcement learning agents interact in a simulated economic environment characterised by a duopolistic competitive market. In principle, the agent could be deployed in any fully observable environment, including the real world. However, since interaction in the real world is costly, I simulate an environment by defining a simple demand function. Following Calvano et al. (2018), consumer demand facing each firm follows a logistic demand curve. This choice of demand curve allows for Bertrand competition, and is flexible enough to allow for firm asymmetries in the form of cost differences and product differentiation. Specifically, the demand for firm i in time t, is defined as follows:

$$q(p_{i,t}, p_{j,t}) := \frac{e^{\frac{a_i - p_i}{\mu}}}{\sum_{j=1}^{n} e^{\frac{a_j - p_j}{\mu}} + e^{\frac{a_0}{\mu}}}. \tag{13}$$

Using this specification, demand is affected by both firms' prices, $p_{i,t}$; the degree of their horizontal differentiation, μ; the quality of both firms' goods measured by a_i, and an outside good of quality a_0. The case of perfect substitutability between the duopolists' products is obtained as $\mu \to 0$. Consumers demand a higher quantity when quality, a_i, is high, and price, $p_{i,t}$ is low.

For the most basic specifications, all parameter values are set in accordance with the values used in Calvano et al. (2018). To resemble a duopoly, the number of firms is set to $n = 2$. The firms are horizontally differentiated to degree μ, but not perfectly so. There is an outside good of moderate quality, $a_0 = 1$, and marginal costs are set to $c_1 = c_2 = 1$. Table 1 summarises the choices of parameters made for the economic environment. In Section 6, some variations to these choices are explored.

Parameter	Value
n	2
μ	1/2
a_0	1
$a_1 = a_2$	2
$c_1 = c_2$	1

Table 1: Economic parameters

The setup just described gives rise to a sequential social dilemma, where each period payoff

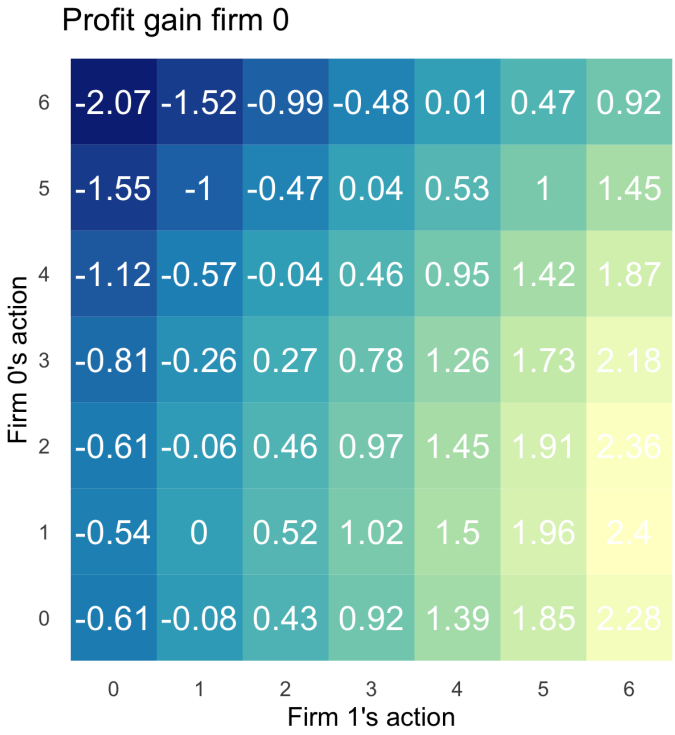


Figure 1: Payoff matrix

can be described as a normal form game. The payoff matrix of the subgame is presented in Figure 1. This figure represents a measure of profit (presented in Section 4.3) obtained at each step in the environment, which depends on the actions of the two players. Since the firms are symmetric, the profit metric displays the payoff of $firm_0$. Based on the figure, the action pair $(1, 1)$ is a Nash equilibrium of the subgame which gives rise to a profit gain of 0 to both firms. However, it is possible for the firms to cooperate and achieve a higher profit, thus giving rise to a PD styled social dilemma. Whether the agents learn to cooperate or converge to Nash equilibrium depends on whether the agents learn to play a strategy that rewards cooperation, such as tit for tat.

Another important aspect of the environment is that consumers are not strategic. Instead, consumers purchase quantities of the two goods available to them at each period depending on what gives them the higher utility. Furthermore, the demand curve remains static over all steps in the environment.

4.2 Action space

Once during each step in the environment, one agent acts by selecting an action, $a_{i,t} \in \mathcal{A}^i$, that is then transformed into a price, $p_{i,t}$. The space from which actions are drawn is discrete, $\mathcal{A}^i = \{1, ...K\}$, with the number of legal actions $|\mathcal{A}^i|$. Because of the increasing cost of computation, the number of actions in the basic set up is limited to, $|\mathcal{A}^i| = 7$, as is represented in Figure 1. Having a discrete action space is necessary for the deep Q-learning algorithm being used which means results are not exact and that the actual theoretical Nash, monopoly, and collaborative actions may not be feasible actions.

The transformation from $\mathcal{A}^i$ to prices is defined as:

$$\mathcal{A}^i \to \mathbb{R}, \text{by } p(a) = \frac{a(p_{max} - p_{min})}{K - 1} + p_{min}, \tag{14}$$

where the agent selects an action enumerated by an integer, $a \in [0, K-1]$. p_{min} and p_{max} are set such that the resulting set of price pairs correspond to 'reasonable' prices. In Calvano et al. (2018), the range is set such that $p_{min} = 0.9p_n$, and $p_{max} = 1.1p_m$, where n denotes Nash, and m denotes monopoly. However, for $|\mathcal{A}^i| = 7$, the range is too wide and instead, prices are set such that $p_{min} = 0.98p_n$, and $p_{max} = 1.02p_m$. This results in the normal form game shown in Figure 1, which allows for an action pair strictly below the Nash action pair, and another action pair strictly above the monopoly action pair, along with a range of possible intermediate values.

Because of the sequential dynamics, only one firm, i, updates its price at time step t in the environment whilst the other firm, j, remains passive during that step. The firms alternate every move such that in time $t+1$, firm j sets a new price while firm i is stuck with the price it set in period t.

Together with the previous action of the passive firm, a joint action is defined as the pair of actions $a_t = (a_{i,t}, a_{j,t-1}) = (a_{i,t}, a_{j,t}) \in \mathcal{A}^i \times \mathcal{A}^j = \mathcal{A}$. It is the joint action, a_t, which, converted into two prices, determines the periodic reward, the next state and therefore the transition dynamics of the environment by calculating demand and profit for each firm. An action is set either randomly or based on the neural network's current estimate of the Q-values. Which method is used depends on the exploration scheme. If the action is to be set randomly, it is sampled from a discrete uniform distribution, $a_{i,t} \sim U[0, K-1]$. Otherwise, the action is based on the current estimates of the Q-values,[8] where the action associated with the highest Q-value, given all the information in the state space, is chosen.

The exploration scheme follows Mnih et al. (2015), where the authors employ a linear decay schedule of the probability to choose an action at random. The probability, ε_t, is set such that $\varepsilon_0 = 1$, and is then decayed linearly over 500000 steps in the environment until it remains constant at $\varepsilon_{t \geq 500000} = 0.01$. The value used in Leibo et al. (2017) and Mnih et al. (2015), 0.1 is deemed too high for this application as each randomisation risks breaking cooperation entirely which is not the case in Leibo et al. (2017).

4.3 Rewards

After each step, both agents are rewarded an instantaneous profit based on the price pair at time t. For the inner mechanics of the environment, the Nash profit is subtracted from the profit of the firm to constitute the reward at each step. It is the cumulative discounted stream of such rewards the agent strives to maximise by selecting actions.

The profit of respective firm is determined by the setup described in Section 4.1. Concretely, together with the price of the other firm, $p_{-i,t}$, the profit in each period for both firms is calculated using $\Pi_{i,t} = q_{i,t}(p_{i,t} - c)$, where the demand, $q_{i,t}$, for each firm is calculated using the definition of the demand specified in Equation 13.

For ease of evaluating the results, rewards are standardised by using an average profit gain index as in both Calvano et al. (2018) and Klein (2018). The index used in these papers is defined as:

$$\Delta_t := \frac{\pi_t - \pi_n}{\pi_m - \pi_n}, \tag{15}$$

where π_n is the profit under Nash equilibrium play, π_m is the 'monopoly profit' which arise under fully collusive pricing when profits are split evenly between the two firms, and π_t is the reward in

[8] Note that actions are not based on the target network, but that because of Double Deep Q-Learning, the target network is used to evaluate the quality of the actions.

Parameter	Value
Discount factor, γ	0.95
Minibatch size, K	2^9
Replay size, N	$5 * 10^5$
Replay start size	10^5
Learning rate, α,	10^{-3}
sync_target_frames, C	10^5
ε decay last frame	$5 * 10^5$
ε_{start}	1
ε_{final}	10^{-2}
Number of actions, $\lvert\mathcal{A}\rvert$	7
State space dimensionality, $\lvert\mathcal{S}\rvert$	2
Steps	$3.5 * 10^6$
Network nodes	8

Table 2: Hyper-parameters

step t. When $\Delta_t = 0$, the firms set prices according to the one shot equilibrium prediction and do not collude. In contrast, $\Delta_t = 1$ means the firms are pricing as a monopolist.

4.4 State space

When the active agent is not exploring, actions taken are based on the agent's ability to perceive two values that constitute the current state, s_t, of the environment observable to the agent at time t. First, the agent observes the price it set in the previous time period, $p_{i,t-1}$. Second, it observes the passive firm's price set in the previous period, $p_{j,t-1} = p_{j,t}$. This makes the state the pair $s_t = (p_{i,t-1}, p_{j,t-1})$.

Remembering both actions of the last period is akin to endowing the agent with a memory capacity of length one and the current state is also everything that matters for future payoffs, making the game a sequential Markov game.

Together, the observable variables constitute the input layer, or the regressors, $\mathbf{x}$, which are taken as input by the neural network in order to predict the values $Q_a(x)$ for each of the actions, $a \in \mathcal{A}$. Note that all states that can be observed by the agents lie in the Cartesian plane of the cross product of the action space by itself, such that $s \in \mathcal{S} = \mathcal{A}^i \times \mathcal{A}^j = \mathcal{A}$. Nonetheless, because of the function approximation of neural networks, exposing the agent to states of any real valued pair, $s \in \mathbb{R}^2$, is feasible.

4.5 Neural network

The purpose of the neural network is to associate the observable variables in s_t with Q-values of actions in order to learn how to pick actions that maximises cumulative discounted profits. There is substantial room for flexibility in the network's design and to characterise the network, this subsection describes architectural choices that was made for the neural network. Choices of hyper-parameters and the network size are guided by Mnih et al. (2015), and Leibo et al. (2017) and decided upon using an informal search. The full set of a default parameters is summarised in Table 2 and results follow this set unless deviations are reported, as is the case in the alternative specifications.

Beginning with the discount factor, γ, this parameter is set to 0.95. It takes the value 0.99 in Leibo et al. (2017) and according to the economic literature summarised in 2.2.2, the discount factor must be sufficiently high in order to enable collusion. However, values close to 1 makes

training less stable as it increases the Q-value estimates' dependency of the bootstrapped estimate of the future relative to observed rewards.

Building upon Mnih et al. (2015), the ideas of a replay buffer and a target network are implemented. The size of the replay buffer from which the agent samples data is set to contain 500000 tuples of the form $(s_t, a_t, r_t, done, s_{t+1})$. In general, a larger replay buffer improves stability owing to the effect of decorrelating observations (Levine 2019, 01:19:30), however too large buffers put less emphasis on recent experience which might be more relevant. To ensure there is training data when the agent begins to learn, the algorithm takes 100000 steps in the environment to populate the replay buffer, following the approach taken in Mnih et al. (2015).

The target network is set to synchronise with the main network every 10000 steps, which is the same as in Mnih et al. (2015) and is also used to further stabilise training using the idea of DDQN (Van Hasselt et al. 2016). That is, when the agent does not randomise, it draws an action based on the current Q-network and updates the current Q-network using the target Q-network, thereby decorrelating errors that would have accrued had the same network been used to select action and to update the network.

The neural network used for the main setup is depicted in Figure 2. The network is given two hidden layers that connect with the input layer and the output layer. The first layer, the input layer, is composed of all the observations the agent can observe from the environment which are the two actions of the last round, denoted s_1 and s_2 in the figure. Next, the network is configured with 2 hidden layers with 8 nodes in each. Leibo et al. (2017) uses 2 hidden layers with 32 nodes each and find that complex coordination strategies are learned more frequently with more hidden units which resembles the greater 'cognitive capacity' of more hidden units. In the case of learning reciprocity, a larger number of hidden units are motivated as the relatively simple action is to defect and undercut the opponents price, thereby driving prices towards the Nash equilibrium. However, because the environment in this thesis is relatively simple compared to Leibo et al. (2017), 8 is deemed sufficient. The final layer, the output layer is defined to have a separate output unit for each action in the output layer following Mnih et al. (2015). All output units correspond to a predicted Q-value covering all actions in the action space denoted as $\hat{Q}_{s,i}$ for action i. Having a Q-value for each action makes it possible to select actions greedily using the Q-values given a state. Finally, between all the layers are rectified linear units which is a standard choice (Goodfellow et al. 2016, p.174). To summarise the network, the input layer assumes two values that are fed into a layer of eight nodes, that connect to another layer of eight nodes and outputs seven values, one for each action. Note that an additional input unit is present, a so called *bias* unit that serves as a form of intercept.

Another idea implemented inspired by Mnih et al. (2015) is that of gradient clipping. At every pass through the network, gradients are limited to the range $[-1, 1]$. That is, if gradients exceed this range, they are limited to the outer boundary of the interval, and otherwise left untouched. The purpose of gradient clipping is to prevent the gradients from exploding which makes learning unstable.

4.6 Training

As in Tesauro & Kephart (2002), Calvano et al. (2018), and Klein (2018) the agents are allowed to train and interact with the environment. While training, the agents learn about the transition dynamics by updating the current estimate of the Q-function according to Algorithm 2.

Setting a limit for the length of the training is a somewhat arbitrary choice based on available computational resources. Ultimately, this thesis uses 3.5 million environmental steps and therefore

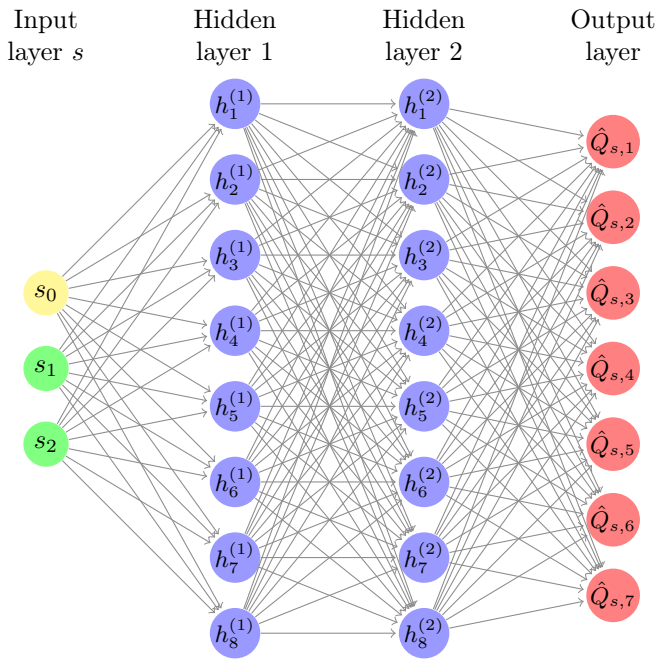


Figure 2: Network used for the main specification

equally many estimations of gradients and back-propagation steps. This decision was determined by computational resources and would ideally be even longer as truly complex behaviour arise after a substantial interaction with the environment and a low learning rate. For example, in Leibo et al. (2017), results are measured after 40 million environmental steps, which is made possible by access to Google Deepmind's computing infrastructure. The time requirement to run the algorithm with 3.5 million iterations is approximately 36 hours on a virtual computer equipped with a standard CPU.

The number of necessary iterations significantly exceeds the number of iterations in the papers by Tesauro & Kephart (2002), Calvano et al. (2018), and Klein (2018). This can be explained by the fact that the DMARL algorithm learns about the quality of significantly more states when it interpolates between observed states. Without a tabular approach, the observation space is allowed to be continuous and the Q-values can no longer be represented by a look up table. Instead, the Q-values are represented by a function which requires more experience to be learned fully, as noted in Sandholm & Crites (1996) where two neural networks with long memory behave less cooperatively in the iterated prisoner dilemma than two discrete players.

The algorithm starts training without any prior knowledge by a random initialisation of the network weights. By the exploration scheme, actions are selected at random and successively become determined by the network. During training, the profit gain and loss is measured at every step and a mean profit gain per 1000 steps is reported along with the loss.

5 Results – main specification

This section shows results for training and the learned Q-values after 3.5 million iterations according to the main specification outlined in Section 4.

5.1 Training dynamics

The method outlined above in conjunction with the parameters summarised in Tables 1 and 2 sometimes leads to stable behaviour over the specified time horizon as shown in Figure 3. The figure shows a measure of average profit gain, Δ_m, obtained by one of the agents over an episode, m, lasting 1000 steps. That is, the value recorded for episode $m \in [1, M]$ indicate $\frac{1}{1000}\sum_{i=1}^{1000} \Delta_{1000m+i}$, where Δ_t is defined as in Formula 15 and evaluated for each environmental step indicated by t. While results are reported as if there are episodes, the change of episode is not noticeable for the agents.

Additionally, the figure contains the trajectory of three simulations that differ in random seed. The random seed influences the actions taken by the agents when they randomise and the starting values of the network parameters. Additionally, Q-learning is known to be inconsistent between different runs because of the randomisation of the actions (Levine 2019, 01:21:06). Consequently, the algorithm was run three times to account for this inconsistency.

The inconsistency of different seeds is observed over the three runs. However, two of the runs are characterised by a relatively similar and stable behaviour. These runs show that a Q-learning agent learns to set high prices, with average profit gains, Δ_m, close to, but distinct from, unity. The third run shows signs of not finding convergence, at least not during the considered time horizon. Instead, it is characterised by large fluctuations in average profit gain.

When considering both agents at the same time, it appears as their respective rewards are closely related, as shown in Figure 4, which focuses on a single seed. See Appendix A for corresponding graphs for the two other random seeds, which give a similar result except for episodes marked by abnormally high and low average profit gains. Instead, these periods are characterised by one agent being able to profit on the other agent until this is corrected for by the exploited agent and the average profit gains again become synchronised.

In terms of convergence, the lower part of Figure 4 shows the mean squared error of the regression target network on the estimated network after each iteration. The loss measures the network's ability to predict the target network Q-values using the current network. As can be seen in the figure, the number of steps required for the loss to stabilise around 10^{-6}, roughly a million, is similar to the number of steps after which the profit gain begins to stabilise between 0.8 and 1. It is important that this value does not diverge towards infinity with increased training time. Given that this is not the case, the most important information is given in the upper part of the training dynamics figures.

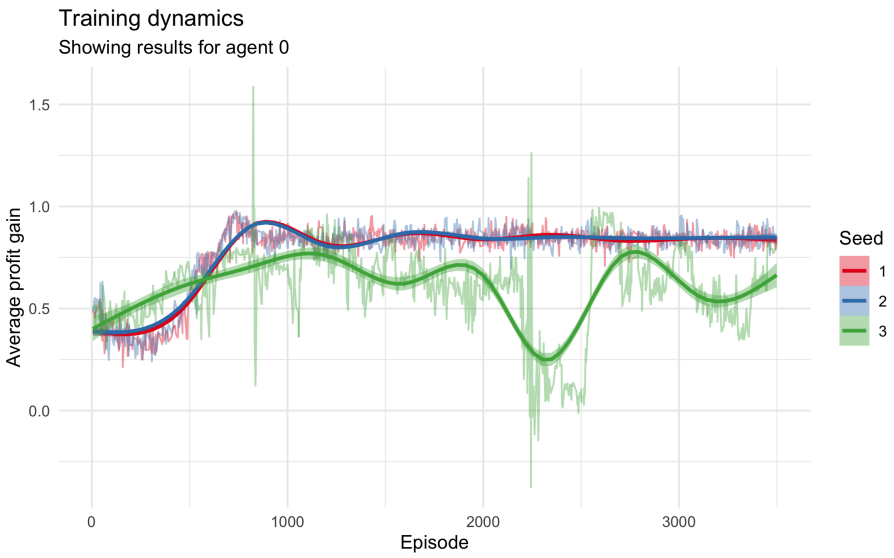


Figure 3: Training dynamics for varying seeds

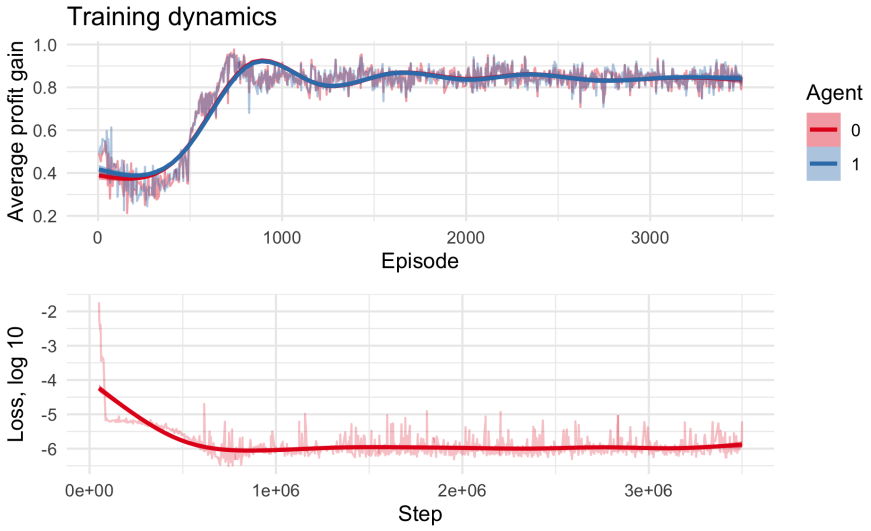


Figure 4: Training dynamics for the two simultaneous agents

5.2 Q-values

Figure 5 plots Q-values associated with each action given previous states. In total, four states are displayed, although there are a total of $|\mathcal{A}^i||\mathcal{A}^j| = 49$ possible states. First, a state deemed 'cooperative' is plotted where both firms price according to action '4' such that $s = (4, 4)$. The action '4' represents a high price that lies just below the monopoly price and when both firms price according to this action, they both receive a profit gain $\Delta_t = 0.95$ for the step which is almost perfectly collusive, refer to Figure 1 for the value.

For all three random seeds, the state $s = (4, 4)$ represents a fixed point when the exploration scheme is turned off, the network is no longer updating, and the agents chose actions according to the highest Q-value because actions chosen according to the highest Q-value results in the same state again. It is necessary to assume that the network is no longer updating to say that it is a fixed point because an updating Q-network might lead to new evaluations of the Q-values and

thereby making the state no longer a fixed point.

An even more cooperative state than $s = (4, 4)$, deemed the monopoly state as it corresponds to monopoly profits where $\Delta_t = 1$, is the state $(5, 5)$. This state is shown in the upper right corner of Figure 5. From this state, both agents show a tendency to deviate and play according to action 4. A reasonable explanation for this behaviour is that the somewhat lower price serves as a protective hedge for times when the other agent (randomly) sets a low price. Had the price been higher, the profit loss of a deviation would be greater. On the other hand, the agents seem to anticipate cooperative actions most of the time and therefore prefer the relatively cooperative value.

The two lower parts of Figure 5 show the asymmetric cases where a firm is either cheated on, or has cheated itself. These states correspond to $(1, 5)$ and $(5, 1)$ respectively. In both cases, the firms prefer to return to action 4. This is evidence that the agents have not learned to punish deviating behaviour and are instead taught an association between action 4 and future high rewards. Perhaps owing to the fact that it is common to observe deviations based on random chance that are followed by a return to normal, collusive, pricing.

Overall, the three seeds lead to different evaluations of the Q-values of the various actions, meaning that the networks disagree on the value of future discounted profits.

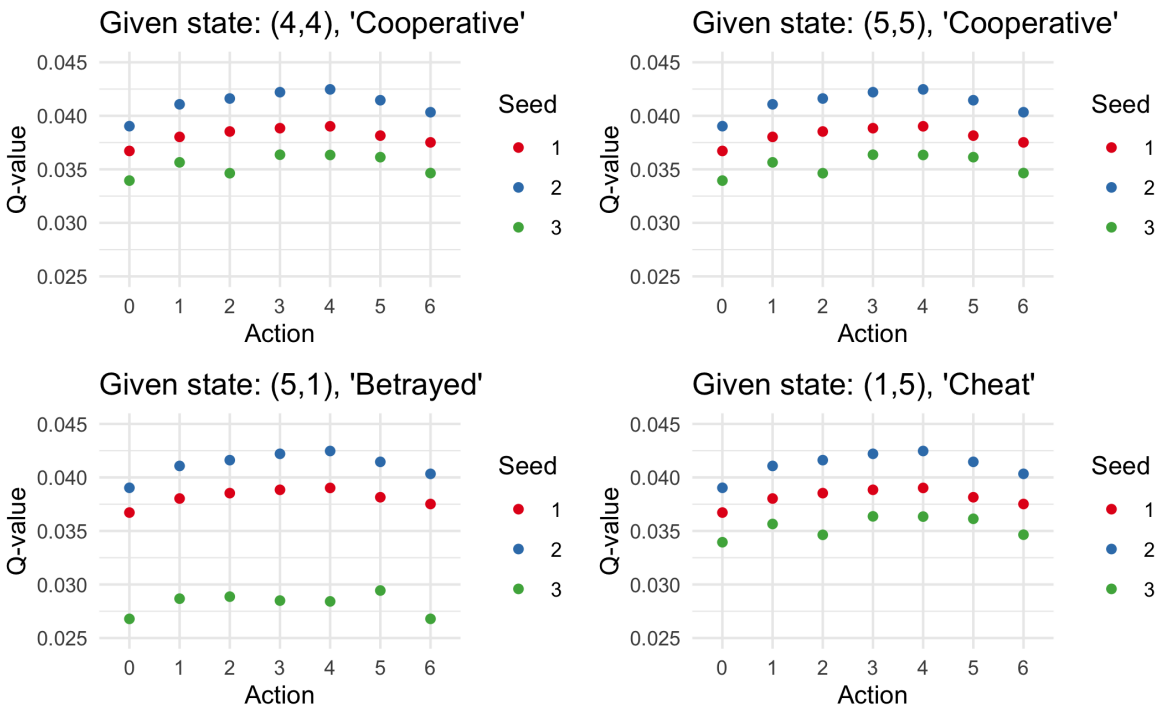


Figure 5: Q-value plot. Plotted Q-values corresponds to the agent's perception of the Q-values of each action after observing a given state. Action 4 is the preferred action in most cases.

5.3 Profit gain in a test setting

Table 3 was generated in a test setting where the probability to select an action at random is set to 0 meaning that the results in the table show the profit gain when the algorithm always selects actions according to its current estimate of the optimal policy after 3.5 million iterations. Each observation on which the data is estimated is the average profit gain observed in an episode lasting

Table 3: Average profit gain, Δ

Specification	Mean	SD	Median
Seed = 1	0.950	0	0.950
Seed = 2	0.950	0	0.950
Seed = 3	0.780	0.010	0.780
Random	-0.120	1.350	-0.990
21 actions	-0.210	1.370	-1.160
Varying diff	0.890	0.110	0.910

[a] $N = 1000$, each observation is average Δ over 100 steps.
[b] For the varying differentiation specification, firms of a vertical differentiation of $[1.9, 2, 2.1]$ were tested in specifications of all possible pairs with 1000 repetitions for each pair.

100 steps that was initialised in a random state. The specifications discussed so far are represented in the first three rows of the table. The remaining rows will discussed under Section 6.

The results in the table show that there is no variation in the average profit gain for the two relatively more stable seed runs, meaning that no matter what the starting state is, both agents play the same action and earn the profit gain reported in the table, corresponding to action 4. There is some variation for the specification where the seed is set to 3, which means that the optimal action depend on the starting state, likely an artefact of the specification not converging.

6 Variations to the main specification

In addition to the specification outlined in Section 4, a few variations are investigated. First, the influence of random demand introduced by letting the outside good exhibit random quality is presented, making the demand stochastic from the firms' perspectives. Second, the action space is expanded from 7 to 21 actions. Third, vertical differentiation is introduced to the model and the agents are able to observe their own and the other firm's degree of vertical differentiation. Overall, these specifications increase the complexity of the learning task and consequently show less signs of convergence.

6.1 Random demand

In this variation, the demand facing the firms is random as the quality of the outside good, a_0 varies and assumes values according to the random variable $A_0 \sim N(a_0, 0.1^2)$. This means the demand function facing each firm is also a random variable as it now depends on a random quantity:

$$q(p_{i,t}, p_{j,t}, A_0) := \frac{e^{\frac{a_i - p_i}{\mu}}}{\sum_{j=1}^{n} e^{\frac{a_j - p_j}{\mu}} + e^{\frac{A_0}{\mu}}} \tag{16}$$

The quality of the outside good is set to update once every environmental step, thereby not allowing the agent to learn intermediate values of the randomness and not giving the network enough time to learn about the new demand. Instead, it has to form a policy that takes the uncertainty into account. Because $a_0 = 1$, the standard deviation of the random variable is 10% of the mean value.

The results of this specification are presented in Figure 6, along with values in Table 3.

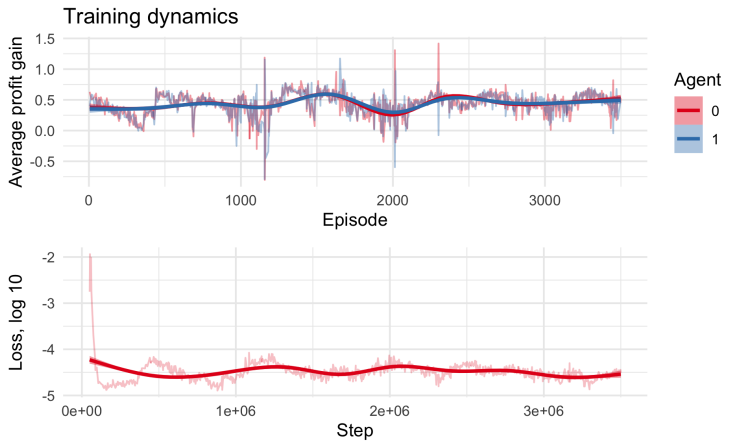


Figure 6: Training dynamics when a_0 varies randomly

While the lower part of Figure 6 shows an initially decreasing loss function towards a low value, the upper part of the figure, which shows profit gain, is seemingly unstable. However, it does show signs of reduced volatility towards the end of the training session. Nonetheless, Table 3 indicates that no learning happened as the algorithms even obtain negative profit gains on average in the testing setting, which indicate a failure to identify even the best response action and thereby the Nash equilibrium. Furthermore, the distribution of average profit gains is skewed and exhibits a large variation.

6.2 Increased action space

In the second variation, the size of the action space is increased to $|\mathcal{A}| = 21$, effectively expanding the output layer to 21 nodes. The range remains the same as in the case of $|\mathcal{A}| = 7$. Under this specification, training is also unstable and needs more time to stabilise. The intermediate results indicate that behaviour is similar to that of random demand in the sense that the mean average profit gain is negative and also skewed downwards.

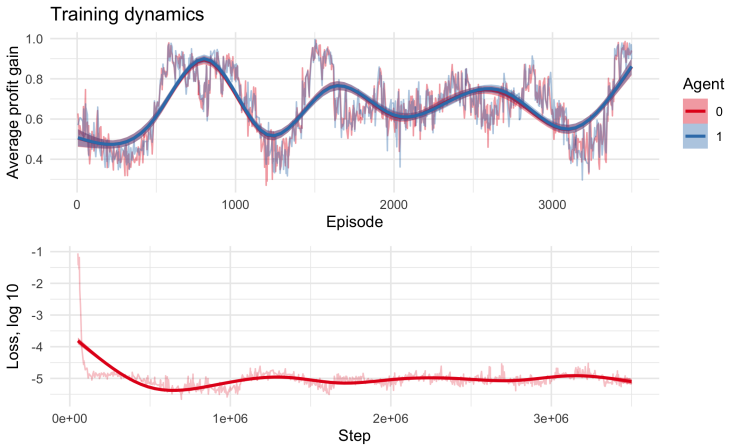


Figure 7: Training dynamics when the agent has 21 actions in the same range

6.3 Varying vertical differentiation

In this final variation to the main specification, the degree of vertical differentiation changes between different episodes. First, an episode lasts until it randomly terminates, which happens with probability 0.001. During an episode, the level of vertical differentiation remains fixed for the two firms, but when an episode changes, both firms are randomly endowed with a new level of vertical differentiation. The length of an episode is thus short enough such that the network is unable to adapt itself during training.

The new demand specification becomes:

$$q(p_{i,t}, p_{j,t}, a_{i,m}, a_{j,m}) := \frac{e^{\frac{a_{i,m}-p_i}{\mu}}}{\sum_{j=1}^{n} e^{\frac{a_{j,m}-p_j}{\mu}} + e^{\frac{a_0}{\mu}}} \tag{17}$$

Where the quality indices, $a_{i,m}$, have an episode subscript, m meaning that the quality of the firms' products are constant within an episode, i.e. they are not dynamic. However, they do vary between episodes, which means the neural network needs to accommodate further complexity to handle variations of product quality for both firms. The values can assume any value in $[1.8, 1.9, 2, 1.1, 1.2]$ and is drawn from this grid randomly using the uniform distribution.

The degree of vertical differentiation is assumed to be observable and both firms know their own degree of vertical differentiation and that of the other firm. Therefore, the dimensionality of the state space is increased to 4, making use of the neural network's ability to approximate the underlying relationship.

Because the firms changes in this specification, the subgame in Figure 1 is no longer valid as the profit gain will vary depending on the firms, as will the Nash and monopoly profits which vary depending on the current configuration and the Nash and monopolist profits were computed using a brute force algorithm. Note that the updated monopoly and Nash profits means that the profit gain metric may no longer be valid for some specifications, as the numerator in Formula 15 might become negative for a firm if the monopolist's profit maximisation implies that one firm earns less than it would under Nash competition.

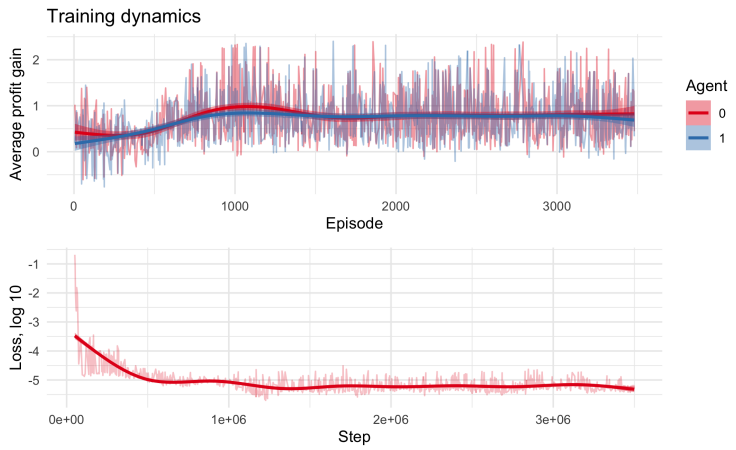


Figure 8: Training dynamics when the agents change vertical differentiation and observes changes.

Figure 8 shows the training dynamics for this specification. What sets this apart is the instability of the training profit gain.

In Table 3, results are reported for the cross product pair of firms where the degree of vertical

differentiation is concentrated about the mean, plus minus 5 percentage points, guaranteeing that the profit metric remains well defined. The average profit gain is relatively high, but less than under the specification of no change in vertical differentiation and there is variation in the metric which reflects that the algorithm cannot perfectly adjust its pricing to new opponents.

7 Discussion

As in other empirical investigations of the Q-learning algorithm by Tesauro & Kephart (2002), Calvano et al. (2018), Klein (2018), this thesis finds that the Q-learning algorithm coupled with a neural network leads to higher than competitive prices. Specifically, the profit gain metric in the main specification of Section 5 indicates that the resulting profits are highly collusive at 95% of the monopolist level and that the direct deployment of the simulation trained algorithm will lead to high prices. In that sense, the addition of a neural network does not fundamentally change the empirical results, while opening up the possibility to expand the potential state space towards a more realistic one.

The fact that one of the three random seeds did not lead to stable behaviour is an artefact of the Q-learning algorithm being unstable. In practice, this is not an issue as the algorithm is meant to train in a simulation before being deployed in a real world environment and any application could simply select a trained algorithm that did converge, or train it during a greater number of iterations before deploying the algorithm. Additionally, the property of being unstable does mean that it would be risky to deploy the algorithm directly in the real world in order to make it learn from real experience as results depend on a random seed initialisation.

In addition to aligning with previous empirical work, the results also align with theoretical research by Maskin & Tirole (1988). The fact that the results in the basic specification of Section 5 seem to lead to a unique focal price (in the sense that once the price combination is set, firms continue to charge those prices forever) aligns with Maskin & Tirole (1988, Section 4), where it is established that 'any equilibrium must be either of the kinked demand type (where the market price converges in finite time to a unique focal price) or the Edgeworth cycle variety (in which the market price never settles down)'. Furthermore, this means that the focal point discovered in this thesis may not be unique as developed in Maskin & Tirole (1988, Section 5). Because the economics literature offers a theoretical understanding of these types of games, it would potentially be fruitful to bridge the gap with the reinforcement learning literature; in particular for multi-agent problems related to social dilemmas or infinitely repeated games.

Concerning the three alterations to the main specification, the added complexity means that the configuration in Section 4 is insufficient for finding stable behaviour. Potentially, a greater number of steps will be required which would give the algorithm more experience to learn from. Alternatively, the network needs a different specification, the hyper-parameters are not configured optimally, or the random seed starts are unfortunate. The fact that there are many potential sources of error and that investigating the sources of error is made difficult by first, a long computation time, and second, the large set of possible hyper-parameters (summarised in Table 2) is a pitfall of neural networks. However, the parameter most able to speed up training is the learning rate, which may plausibly be set higher when used in conjunction with gradient clipping which limits the step size associated with larger learning rates (Goodfellow et al. 2016, p.238).

In terms of applicability of the results, the DMARL algorithm explored in this thesis can be said to fall under the category which Ezrachi & Stucke (2017) refer to as 'hub and spoke'. This means that if a 'hub', or a central provider, of the algorithm distributes the algorithm to several competitors, the result could be collusive. As is pointed out in The Competition and Markets

Authority (2018, p. 26), a scenario where competitors chose to outsource pricing decisions to an intermediary is potentially serious as it is harder to prosecute than outright agreements. However, the deep Q-learning is not unique in leading to collusive prices when played against itself and there may be simpler algorithms which could achieve similar outcomes.

A shortcoming of the DMARL algorithm trained in this thesis is that even after a substantial period of training, it is unable to develop punishing tactics of deviating behaviour, rendering the algorithm exploitable in the sense that a firm pricing according to the best response action would profit from facing the algorithm which naïvely prices cooperatively. On the other hand, the algorithm exhibits more complex behaviour in other papers of social dilemma (Leibo et al. 2017) and may in principle also learn more complex behaviour in a pricing scenario given more environmental experience, that is a longer training time. An algorithm with a developed ability to punish deviations would correspond to the category of 'predictable agents' according to the classification in Ezrachi & Stucke (2017). The reason it is predictable is that punishment of deviations is reciprocal behaviour, hence predictable. A predictable agent type of algorithm raises more serious concerns from a competition authority's point of view than the hub and spoke category as a predictable and non-exploitable agent means the algorithm plays according to tit for tat, which is one viable way to sustain tacit collusion.

On the other hand, it seems unlikely that the deep Q-learning algorithm of this thesis will develop into the category 'digital eye', under which category a unilaterally deployed agent learns to optimise profits by interaction with the environment. The reason is that the data requirements for a versatile agent are high and that learning may currently be too slow. For example, while the advantage of Q-learning is that it is a model free algorithm that can learn unknown environments, such as the real world where the true demand function is unknown, the amount of data required in even the sterile setting presented in this thesis would require a potentially unfeasible amount of interaction with the environment in order to learn. In this thesis, about a million steps were required to associate high prices with high profits. To translate this into a real world environment would require sales to be near continuous as to reach a million steps in a year would require the algorithm to receive feedback more than twice every minute. In addition, the algorithm will need more interaction to learn punishing and even more interaction when the environment is not as sterile as in the main scenario, see the specification variations in Section 6.

Consequently, the deep Q-learning algorithm needs improvements in order to become a digital eye type of algorithm. Allowing for training in a simulation before deployment potentially solves the data requirement if a rich and accurate enough model of the environment can be developed. Given the model, solving it could potentially be done by a deep Q-learning algorithm as scaling up to complex relationships efficiently is the main strength of a neural network. However, modelling a rich enough environment might not be trivial as such an environment should account for strategic agents, entry and exits, investment dynamics, customer reputation, and other important features that characterise the real world environment. Another option is to use real world data from several products and impose structure on the problem, e.g. assuming similar demand dynamics across a menu of similar products. A problem with this approach was pointed out in Schwalbe (2019, p.591) as each time a new product is introduced, a new firm enters, or someone makes an innovation, the algorithm has to adapt itself to the new information.

An important consideration that is excluded from this thesis is that the choice of algorithm is exogenous to the firms. A more complete description would make the choice of algorithm endogenous, as is the case in Brown & MacKay (2019). As pointed out earlier, the algorithm developed in this thesis is exploitable and thus there is an incentive to exploit it as long as the firm using the algorithm is committed to the pricing technology. Another crucial aspect left out of this

thesis is that of competition with more than two firms. In this setup, it is left to the researcher to specify the ordering of the sequential moves, or whether moves should be simultaneous.

8 Conclusion

This thesis described, developed, and deployed a deep reinforcement learning algorithm known as Q-learning in a simulated multi-agent economic environment in order to investigate whether deep Q-learning algorithms are able to find and sustain collusive outcomes. While the algorithm learned to associate high prices with high profits after a period of substantial training, it remained exploitable in the sense that it during testing was unable to reciprocate non-cooperative behaviour. Instead, the agent learned to anticipate that deviations are random exploration conducted by the other agent which otherwise reverted to cooperation in the time horizon considered in this thesis.

Most importantly and despite the 'deadly triad', learning did show evidence of not diverging and obtaining unbounded value estimates. This indicates that the addition of neural networks when coupled with experience replay, target networks and double learning may not make learning fundamentally different from the tabular Q-learning algorithms developed in previous research. Furthermore, this hints that there may soon be algorithms able to navigate more complex environments than previously considered.

As mentioned in the introduction, some form of function approximation is required in order to make reinforcement learning applicable on 'large problems', such as real world pricing characterised by varying demand, product qualities, and a near continuous grid of possible prices. It is therefore necessary to deepen the understanding of the DMARL algorithm applied to pricing problems since a research laboratory with the knowledge and resources to develop such algorithms could potentially create an agent that prices collusively and reaches the third or fourth level of difficulty to prosecute on the hierarchy in Ezrachi & Stucke (2017).

References

Axelrod, R. & Hamilton, W. D. (1981), 'The evolution of cooperation', *Science* **211**(4489), 1390–1396.

Belleflamme, P. & Peitz, M. (2015), *Industrial organization: markets and strategies*, Cambridge University Press.

Brown, Z. & MacKay, A. (2019), 'Competition in pricing algorithms', *Harvard Business School Working Paper* **20**(67).

Calvano, E., Calzolari, G., Denicolò, V. & Pastorello, S. (2018), 'Artificial intelligence, algorithmic pricing and collusion', *CEPR Discussion Paper No. DP13405* .

Chen, L., Mislove, A. & Wilson, C. (2016), An empirical analysis of algorithmic pricing on amazon marketplace, *in* 'Proceedings of the 25th International Conference on World Wide Web', pp. 1339–1349.

Ciregan, D., Meier, U. & Schmidhuber, J. (2012), Multi-column deep neural networks for image classification, *in* '2012 IEEE conference on computer vision and pattern recognition', IEEE, pp. 3642–3649.

Edgeworth, F. Y. (1925), 'The pure theory of monopoly', *Papers Relating to Political Economy* **1**, 111–142.

Ezrachi, A. & Stucke, M. E. (2017), 'Artificial intelligence & collusion: When computers inhibit competition', *University of Illinois Law Review* p. 1775.

Fudenberg, D. & Tirole, J. (1991), *Game theory*, MIT Press.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep learning*, MIT Press.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. & Silver, D. (2018), Rainbow: Combining improvements in deep reinforcement learning, *in* 'Thirty-Second AAAI Conference on Artificial Intelligence'.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Klein, T. (2018), Assessing autonomous algorithmic collusion: Q-learning under short-run price commitments, Technical report, Tinbergen Institute Discussion Paper.

Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J. & Graepel, T. (2017), Multi-agent reinforcement learning in sequential social dilemmas, *in* 'Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems', International Foundation for Autonomous Agents and Multiagent Systems, pp. 464–473.

Levine, S. (2019), 'Deep RL with Q-Functions', youtu.be/7Lwf-BoIu3M? Accessed: 07/03/2020.

Maskin, E. & Tirole, J. (1988), 'A theory of dynamic oligopoly, ii: Price competition, kinked demand curves, and edgeworth cycles', *Econometrica: Journal of the Econometric Society* **56**(3), 571–599.

Mitchell, T. M. et al. (1997), *Machine learning*, McGraw-hill New York.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al. (2015), 'Human-level control through deep reinforcement learning', *Nature* **518**(7540), 529.

Mullainathan, S. & Spiess, J. (2017), 'Machine learning: an applied econometric approach', *Journal of Economic Perspectives* **31**(2), 87–106.

OECD (2017), 'Algorithms and collusion: Competition policy in the digital age', http://www.oecd.org/competition/algorithms-collusion-competition-policy-in-the-digital-age.htm. Accessed: 06/03/2020.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), 'Learning representations by back-propagating errors', *Nature* **323**(6088), 533–536.

Sandholm, T. W. & Crites, R. H. (1996), 'Multiagent reinforcement learning in the iterated prisoner's dilemma', *Biosystems* **37**(1-2), 147–166.

Schwalbe, U. (2019), 'Algorithms, machine learning, and collusion', *Journal of Competition Law & Economics* **14**(4), 568–607.

Solon, O. (2011), 'How a book about flies came to be priced $24 million on amazon', Wired, https://www.wired.com/2011/04/amazon-flies-24-million/. Accessed: 11/05/2020.

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement learning: An introduction*, MIT Press.

Sweezy, P. M. (1939), 'Demand under conditions of oligopoly', *Journal of Political Economy* **47**(4), 568–573.

Tan, M. (1993), Multi-agent reinforcement learning: Independent vs. cooperative agents, *in* 'Proceedings of the tenth international conference on machine learning', pp. 330–337.

Tesauro, G. & Kephart, J. O. (2002), 'Pricing in agent economies using multi-agent q-learning', *Autonomous Agents and Multi-Agent Systems* **5**(3), 289–304.

The Competition and Markets Authority (2018), 'Pricing algorithms: Economic working paper on the use of algorithms to facilitate collusion and personalised pricing', https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/746353/Algorithms_econ_report.pdf. Accessed: 04/04/2020.

Van Hasselt, H., Guez, A. & Silver, D. (2016), Deep reinforcement learning with double q-learning, *in* 'Thirtieth AAAI conference on artificial intelligence'.

Vestager, M. (2017), 'Algorithms and competition', https://wayback.archive-it.org/12090/20191129221651/https://ec.europa.eu/commission/commissioners/2014-2019/vestager/announcements/bundeskartellamt-18th-conference-competition-berlin-16-march-2017_en. Accessed: 06/03/2020.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul,

T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D. & Silver, D. (2019), 'AlphaStar: Mastering the Real-Time Strategy Game StarCraft II', https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/. Accessed: 28/03/2020.

Watkins, C. J. C. H. (1989), Learning from delayed rewards, PhD thesis, King's College, University of Cambridge.

Zhang, K., Yang, Z. & Başar, T. (2019), 'Multi-agent reinforcement learning: A selective overview of theories and algorithms', *arXiv preprint arXiv:1911.10635* .

A Additional figures

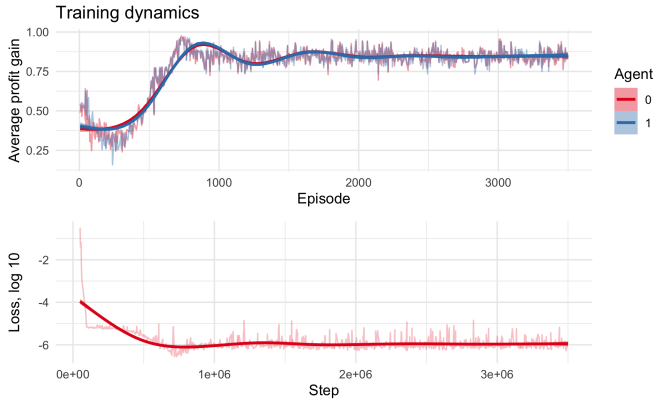


Figure 9: Training dynamics, seed = 2

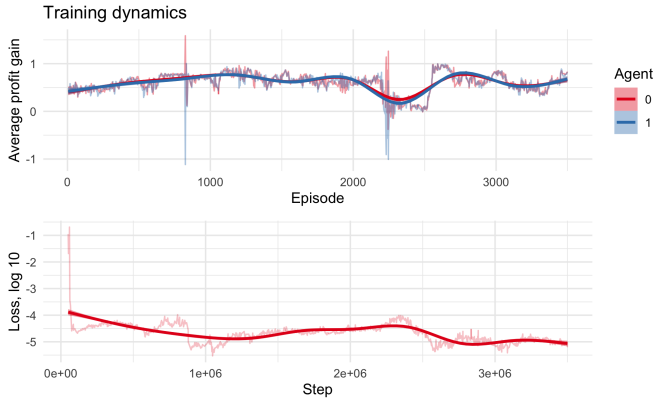


Figure 10: Training dynamics, seed = 3

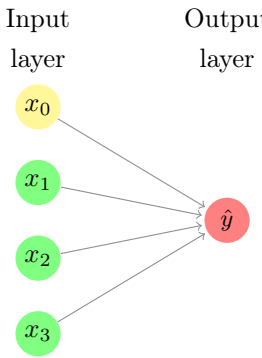


Figure 11: Linear regression of three variables and an intercept represented as a neural network. The 'weights' that connect the input layer (independent variables) to the output layer (dependent variable) in the network correspond to variable coefficients. The intercept term, $x_0 = \mathbf{1}$, is known as 'bias unit'.