DMQL

Deep Maximum Q-Learning: Combatting Relative Overgeneralisation in Deep Independent Learners using Optimism and Similarity

by

Erwin Dam

to obtain the degree of Master of Science
at the *Delft University of Technology*,
to be defended on
the 29th of August, 2022.

Student Number:		4704886
Master Programme:		Computer Science
Specialisation:		Artificial Intelligence Technology
Research Group:		Algorithmics Research Group Department of Software Technology Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS) Delft University of Technology Delft, The Netherlands
Daily supervisor and thesis committee member	:	Dr. Wendelin Böhmer
Thesis advisor and thesis committee member	:	Dr. Matthijs Spaan
Thesis committee member:		Dr. Frans Oliehoek

TU Delft

Preface

This work is the product of my thesis, carried out to obtain the degree of Master of Science in Computer Science at the Delft University of Technology. In this project, we have developed a new algorithm which enables independent learners in multi-agent reinforcement learning problems to avoid relative overgeneralisation.

First, I want to express my gratitude towards my daily supervisor, Dr. Wendelin Böhmer, for his guidance throughout the project and for providing the opportunity to work on this subject. His advice and sharing of knowledge during our frequent meetings provided me with a lot of guidance and opportunities to learn. Secondly, I would like to thank Dr. Matthijs Spaan for acting as my thesis advisor and for his support during the project. In addition, I would like to thank Dr. Frans Oliehoek for joining the thesis committee.
Finally, I would like to thank my family and others around me for supporting me during my studies. This work would not have been completed without the support of all of the above, thus thanks to all.

Erwin Dam
Delft, August 2022

Summary

Various pathologies can occur when independent learners are used in cooperative Multi-Agent Reinforcement Learning. One such pathology is Relative Overgeneralisation, which manifests when a suboptimal Nash Equilibrium in the joint action space of a problem is preferred over an optimal Equilibrium. Approaches exist to combat relative overgeneralisation in Q-Learning problems, yet many of these do not scale well with the state space or joint action space, are hard to adapt or configure, or are not applicable in partially observable environments.

In this work, we introduce Deep Maximum Q-Learning (DMQL), a methodology combining Deep Recurrent Q-Networks [Hausknecht & Stone, 2015] and the optimistic assumption which can be found in Distributed Q-Learning [Lauer & Riedmiller, 2000]. DMQL is a maximum-based learning technique which can be scheduled to transition to an average-based learner (or any other arbitrary type of learner), which can utilise independent learners without communication. DMQL is designed to be relatively intuitive and easy to adapt and configure and to be able to utilise notions of similarity to provide solutions in large and continuous state spaces, as well as in environments with partial observability.

DMQL clusters similar histories by mapping them to the same hash based on a subset of the information contained within them, such as the current observation, or other related available information sources, such as state information. Using these hashes, DMQL constructs a hash-action pseudo-maximum Q-value estimation dictionary, which is updated at every gradient update step. A dictionary value degradation technique ensures stability by preventing overestimations from being retained in the dictionary by decaying them after they have been encountered. This way, optimism *is* introduced, and relative overgeneralisation is prevented *without* using true maximums of past Q-value estimates directly, as these are not guaranteed to be indicative of the real optimal Q-values *and* can lead to divergent behaviours. Contrasting similar deep learning methodologies [Palmer et al., 2017], DMQL augments Deep Q-Network targets through value replacement instead of value discardment, potentially leading to improved efficiency. In addition, DMQL can be adapted such that it can be utilised as a maximisation-based step in the greater learning process of other deep learning algorithms.

Our experimental results indicate that DMQL is a successful extension of Distributed Q-learning, which can be used in small environments even without the usage of similarity. Using similarity, however, grants us the ability to learn in increasingly large and complex environments. Interestingly, various problems exist within the process of developing a suitable manner of incorporating similarity into hashes. We speculate on how these problems can be prevented or circumvented, and our experiments validate our circumvention methods. Lastly, our experiments show that DMQL can successfully be applied to combat relative overgeneralisation in partially observable environments as well.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
Dec-MDP	Decentralised Markov Decision Process
Dec-POMDP	Decentralised Partially Observable Markov Decision Process
DMQL	Deep Maximum Q-Learning
DQN	Deep Q-Network
DDQN	Double Deep Q-Network
DRQN	Deep Recurrent Q-Network
DDRQN	Double Deep Recurrent Q-Network
IQL	Independent Q-Learning
LL	Lenient Learning
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
QL	Q-Learning
RL	Reinforcement Learning
RNN	Recurrent Neural Network

Symbols

Symbol	Definition
D_I	Experience replay buffer at learning iteration (gradient update step) I
D_t	Dataset or 'experience replay buffer' at time step t
H	Episode length
I	Training iteration (gradient update step)
N	Number of agents
O^i	Observation function of agent i
P	Probability function
Q	Q-value function
$Q*$	Optimal Q-value function
T	Transition probability function
V	Value function
Y_t	Target at time step t
a	Action
a_t^i	Action by agent i at time step t
d_t	Degradation factor for DMQL Dictionary Degradation Mechanism
e	Experience
e_t	Experience at time step t
i	Agent
l	Leniency function
p	Punishment value (negative reward)
r	Reward value (positive reward) or reward returned by reward function
r_t	Collaborative reward at time step t
r_t^i	Reward for agent i at time step t
s	State

Symbol	Definition
s_t	State at time step t
s'	Next state
t	Time step (discrete)
u	Hash, string from language $\mathcal{U}^*$
u_s	Soft network update replacement factor
o_t^i	Observation of agent i at time step t
h_t^θ	Hidden state of recurrent neural network with parameterisation θ at time step t
q	Q-value
q^*	optimal Q-value
$\boldsymbol{a}$	Joint action
$\boldsymbol{a}_t$	Joint action at time t
$\boldsymbol{a}'$	Joint action from next state
$\mathcal{A}$	Joint action space
$\mathcal{A}_t^i$	Action space of agent i at time step t
$\mathcal{D}_I^i$	DMQL Dictionary Mechanism Dictionary of agent i at learning iteration (gradient update step) I
$\mathcal{H}$	Maximum episode length
$\mathcal{O}$	Joint observation space
$\mathcal{O}_t^i$	Observation space of agent i at time step t
$\mathcal{R}$	Immediate reward function (collaborative)
$\mathcal{S}$	State space
$\mathcal{T}$	Lenient Learning: Temperature
$\mathcal{U}$	Alphabet (e.g. Unicode)
$\mathcal{U}^*$	Language constructed from $\mathcal{U}$
α	Learning rate
γ	Discount factor
ϵ	Probability of random action selection in ϵ-greedy action selection
θ	Predictive network parameters
θ_I	Predictive network parameters at training iteration (gradient update step) I
θ'	Target network parameters
θ'_I	Target network parameters at training iteration (gradient update step) I
π	Policy
π^*	Optimal policy
ρ	Function generating initial state s_0
$\boldsymbol{\tau}_t$	Joint action-observation history at time step t
τ_t^i	Observation history of agent i at time step t
ϕ	Hash function
C	'Catch' action in Hunter-Prey environment
D	'Move down' action in Hunter-Prey environment
I	'Idle' action in Hunter-Prey environment
L	'Move left' action in Hunter-Prey environment
R	'Move right' action in Hunter-Prey environment
U	'Move up' action in Hunter-Prey environment

List of Figures

List of Tables

1

Introduction

Multi-Agent Reinforcement Learning (MARL) is a subfield of Reinforcement Learning (RL) which seeks to enable sets of agents to develop policies to achieve their respective goals. It offers methodologies to learn these policies, which are of particular use when the complexity of the tasks is substantial enough to prevent conventional methods, such as pre-programming agent behaviours, from being scalable. This work focuses on Cooperative MARL, which, as the name suggests, is the subset of cases in which the agents are rewarded collectively for achieving common goals, often requiring the development of a joint policy involving cooperation.

One of the main problems in MARL involves scalability as well: the combined decision space typically grows exponentially with the number of agents [OroojlooyJadid & Hajinezhad, 2019]. This property quickly makes the problem intractable when using straightforward centralised decision-making.
To circumvent the intractability introduced by these centralised decision-making processes, one has to introduce assumptions to allow for alternative approaches. One often-made assumption is that the decision-making - the control policy - can be decentralised.
Examples of state-of-the-art methodologies successfully making use of this assumption include ones factoring the joint value function into functions depending on the utility of single agents - such as Value Decomposition Networks and QMIX [Sunehag et al., 2017] - and ones factoring the joint value function into functions depending on pairwise utilities [Castellini et al., 2019].

One of the first and one of the most straightforward approaches made possible through the use of the assumption that one can decentralise the control policy is an extension of the Q-Learning (QL) algorithm [C. J. Watkins & Dayan, 1992; C. J. C. H. Watkins, 1989]: Independent Q-Learning (IQL) [Tan, 1993] is an extension that considers each agent in a MARL problem to be an independent learner. It works utilising an additional assumption: the assumption that one can treat the other agent and their behaviours as if they were part of the environment. It marginalises out the decisions made by other agents, maximising utility from the perspective of individual agents. Presupposing the assumption is valid, this still requires the expected values of viable actions to depend on the joint policy of all other agents. This dependency can lead to an interesting problem:

While even the single-agent-utility-based factorisation of the policy found in IQL can represent any deterministic policy, there is a noteworthy problem with regards to the learning processes used with the aim to learn optimal representations; *learning* (some/optimal) representations may be *utterly infeasible* when action-taking is not (sufficiently well-) coordinated, even in the tabular case. These independent learners lack the convergence guarantees they often offer in single-agent problems. Take, for example, the problem described in ‘Hunter-Prey Environment’.

Hunter-Prey Environment

An example of a problem which may induce learning problems can be found in a Predator-Prey Environment [Gupta et al., 2021]. To explore the problem, we will consider a simplified environmental state representative of a context in which two agents (referred to as 'agents'/'hunters' for the rest of this work) are adjacent to prey in a one-dimensional version of the environment (Figure 1.1).

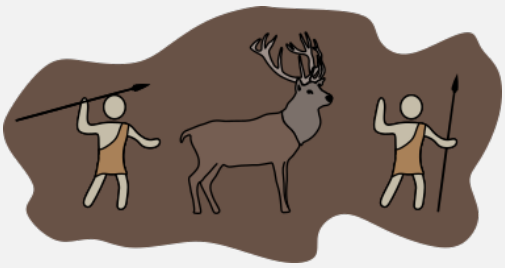

Figure 1.1: Illustration: Two hunters adjacent to their prey

In this case, we define the action space to consist of the actions 'move left' (**L**), 'move right' (**R**), 'idle' (**I**), and 'attempt catch' (**C**). Catching the prey is good for the agents and grants the collective reward r. It is impossible for a single agent to catch prey; it will simply fail to do so if attempted and will additionally result in a negative collective reward, i.e. a punishment $-p$.

The rewards for the possible combined actions of our agents are depicted in the reward matrix of Table 1.1. Agents do not know the matrix and need to explore it randomly. From this matrix, one can observe that the chance of attaining the value r when randomly and uniformly choosing actions for each agent only is $P(\text{reward} = r) = \frac{1}{16}$. When agent i chooses the catch action, the chance that agent j ($\neq i$) will choose an action which results in a reward will be $P(\text{reward} = r | a^i = \textbf{C}) = \frac{1}{4}$, whilst the chance of taking an action which results in a punishment will be $P(\text{reward} = -p | a^i = \textbf{C}) = \frac{3}{4}$.
This leads us to two main inferences: the value of taking capture action **C** depends greatly on the actions of the other agent, and the expected value for the catch action from experiences in which the other agent acts randomly alone will be $\frac{r-3p}{4}$, which is less than the expected values of the other actions ($\frac{-p}{4}$) when $p > \frac{1}{2}r$.

		Agent 1			
		L	**R**	**I**	**C**
Agent 2	**L**	0	0	0	-p
	R	0	0	0	-p
	I	0	0	0	-p
	C	-p	-p	-p	r

Table 1.1: Reward Matrix

In practice, the expected value of the catch action depends on the returns from previous experiences. If the other agent favours actions other than **C** often enough, i.e. if the other agent is not cooperative enough on average, this expectation will become lower, leaving the values of the other actions as the better alternatives from the perspective of this agent alone. This process, in turn, practically ensures that the other agents that encounter this agent often enough will develop a similarly pessimistic view on the catch action as the rate of cooperation is lowered.

After a while, the system will reach a relatively stable, locally optimal policy in which the catch action is avoided. This prevents the agents from developing a truly optimal policy. This is a manifestation of '*relative overgeneralisation*', a pathology from game-theory [Panait, Luke, & Wiegand, 2006].

The experiments in this work have been performed using a two-dimensional version of this environment. For completeness, we define actions 'move up' (**U**) and 'move down' (**D**) similarly to the aforementioned actions. Sets of hunters and prey are able to move around in a grid world. Due to the introduction of this extra dimension, the chance of an agent randomly selecting an action other than the catch action is increased, thereby increasing the odds of relative overgeneralisation occurring. The expected value of **C** will be lower than the ones of other actions sooner: $\frac{r-5p}{6} < \frac{-p}{6}$ if $p > \frac{1}{4}r$.

Rules of the environment can be found in Appendix B.

In the Hunter-Prey environment, relative overgeneralisation can completely inhibit convergence to an optimal joint policy if there is a lot of punishment. Relative overgeneralisation manifests when a sub-optimal Nash Equilibrium in the joint action space is preferred over an optimal Equilibrium. The three main factors which influence the phenomenon of relative overgeneralisation for independent reinforcement learners are the reward/punishment ratio, the ratio of action sets yielding rewards to ones yielding punishments, and the amount of exploration performed by agents. The reward/punishment ratio determines how many punishments - relative to rewards - are required to make the expected value for rewardable actions negative, and therefore determines the disruptive power of exploratory actions which increase the number of punishments. The ratio between sets of actions yielding rewards to ones yielding punishments influences the relative encounter rate from taking random actions (and thereby, once again, the disruptive power of exploratory actions).

To overcome relative overgeneralisation, various approaches have been proposed. These include the usage of a centralised training scheme and removal of structural factorisation constraints [Son et al., 2019], the usage of a centralised Q-value to change weightings [Rashid et al., 2020], and the usage of coordination graphs to enable the usage of higher-order factorisations [Böhmer et al., 2020; Castellini et al., 2019]. Most of these approaches aim to grant the representational capacity to allow agents to distinguish whether or not joint actions were coordinated or not, or aim to factorise the utility function in various manners other than through summed independent utilities.

An older work, *Distributed Q-Learning* [Lauer & Riedmiller, 2000], believed to be the earliest independent-learner algorithm specifically designed for cooperative multi-agent learning scenarios [Wei & Luke, 2016], introduces a way to counteract relative overgeneralisation in the discrete, fully-observable tabular state space case for cooperative multi-agent problems without asserting coordination [Lauer & Riedmiller, 2000]. It describes that the effects from relative overgeneralisation can be removed through the use of an 'optimistic assumption': the assumption that all other agents act optimally, or, more practically, that all other agents will choose actions that have yielded the highest reward thus far. This means that one essentially disregards the effects of exploration-based miscoordination for the calculation of action values. As *Distributed Q-Learning* maximises over both other agents' actions during value calculation and table entries and calculated values during policy updates, the approach has become known as a maximum-based learning approach which is highly optimistic. It has been shown to work in discrete fully-observable tabular state spaces.

Many works have followed since, augmenting various parts of RL approaches in order to introduce some kind of optimism, or to more effectively explore the joint search space. Examples include ones that opt to mainly change action selection [Kapetanakis et al., 2002; Matignon et al., 2009], and some that augment the update strategy. The latter kind is of particular interest for this work. Most methodologies which augment the update strategy can be classified as one of two approaches: the aforementioned maximum-based learning approaches, and approaches augmenting the learning process by introducing various learning rates, learning slower from experiences which are found to be negative in some way [Bowling & Veloso, 2001, 2002; Matignon et al., 2012; Matignon et al., 2007].

One of the most prominent subfields developing maximum-based reinforcement learning approaches, *Lenient Reinforcement Learning*, is one which contains approaches which we consider to be most similar to the approach introduced in this work. Approaches from this subfield introduce significant benefits in cooperative settings where multiple agents have to learn how to interact with each other [Panait, Sullivan, et al., 2006]. The main argument for lenience in reinforcement learning algorithms follows from the observation that agents tend to converge towards certain areas of the state space due to their learning process in concurrent learning [Panait & Luke, 2005; Panait, Sullivan, et al., 2006]. The agents' perception of the state space may benefit if multiple rewards are considered for policy development at early stages of learning, with diminishing impact towards the end of the learning process [Panait, Sullivan, et al., 2006]. The concept of lenience targets these observations: at early stages of learning, agents ignore low rewards, as they assume these low rewards are due to sub-optimal actions performed by other agents. Hence, agents are initially lenient towards their fellow agents and become gradually less lenient as learning progresses to ensure the information on truly lower rewards is eventually incorporated. Lenient Learning prevents relative overgeneralisation by preventing agents

from gravitating towards a robust yet sub-optimal joint policy induced by the influence of other agents' exploration strategies on each agent's learning updates [Palmer et al., 2017].

The field of Lenient Learning (LL) (applied to RL) has since progressed to include designs which target various pathologies, as well as increasingly stochastic cooperative games [Wei & Luke, 2016]. Recently, the concept of lenient learning has been ported to Multi-Agent Deep Reinforcement Learning, where it has been shown to facilitate cooperation in tabular fully-cooperative MARL problems and has been extended with auto-encoders to be able to work with high-dimensional or continuous state spaces [Palmer et al., 2017]. While lenient learning has been shown to provide promising results, many lenient methods receive criticism due to the complexity of their implementations, difficulties in selecting the correct hyperparameters, the overhead due to the need to store various extra values, and the time needed to converge [Palmer et al., 2017; Wei & Luke, 2016].

In this work, an approach will be introduced which aims to provide the benefits introduced by the optimistic assumption introduced by the work in *Distributed Q-Learning* [Lauer & Riedmiller, 2000] by applying an optimistic assumption directly to the learning process through the manipulation of target values, replacing some observed targets with more optimistic values derived from past experiences. This can be interpreted as a strategy of replacement of update values, where Lenient Multi-Agent Deep Reinforcement Learning employs a strategy of discarding updates. This difference in strategies forms an essential step in introducing optimism using agent action-observation histories to train recurrent deep Q-networks. Our approach targets various criticisms received by LL approaches, and should be easy to adapt, relatively easy to implement and configure, and should need less time to converge to a cooperative policy as updates are not discarded. In addition, our approach should be able to function under partial observability.

The main objectives of this work are to develop a viable extension of the iteration rule of *Distributed Q-Learning* to Deep Multi-Agent Reinforcement learning, to make the approach scalable and hence to identify problems and propose targeted solutions, and to assess how this method performs in increasingly complex environments, as well as under partial observability.

The remainder of this paper proceeds as follows: In Section 2, we elaborate on background information needed to understand the methodology of this work and introduce *Lenient Learning* in more detail in order to allow the reader to contrast the approaches. In Section 3, we introduce our approach to the prevention of relative overgeneralisation, main design choices, as well as clarifications and justifications for most approximations used. Section 4 presents the results of our experiments, starting from the most direct port from *Distributed Q-Learning* to Deep Learning, steadily scaling up to more complex environments, demonstrating various variants of our approach. Lastly, in Section 5, we discuss our results and answer our research questions, after which we introduce opportunities for future works and discuss various interesting insights.

2

Background

In this work, the cooperative multi-agent task is represented by a Decentralised Partially Observable Markov Decision Process (Dec-POMDP) [Oliehoek & Amato, 2016] defined as the tuple which can be found in Equation 2.1.

$$\langle \mathcal{S}, \{\mathcal{A}^i\}_{i=1}^N, T, \{\mathcal{O}^i\}_{i=1}^N, \{O^i\}_{i=1}^N, \mathcal{R}, \rho, N \rangle \tag{2.1}$$

Where

- $\mathcal{S}$ represents the state space of the environment, where
  - $s_t \in \mathcal{S}$ denotes the state of the environment at time step t.
- $\mathcal{A}$ represents the joint action space, where
  - $\mathcal{A}^i$ represents the action space for each agent $1 \leq i \leq N$
  - $\boldsymbol{a}_t \in \mathcal{A} := \mathcal{A}^1 \times \ldots \times \mathcal{A}^N$ denotes joint actions at time step t
  - $a_t^i \in \mathcal{A}_t^i$ represents actions by each agent $1 \leq i \leq N$ at time step t
- $T(s_t, \boldsymbol{a}_t, s_{t+1}) = P(s_{t+1}|s_t, \boldsymbol{a}_t)$ of $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ represents the probability of transitioning from state s_t to s_{t+1} using joint action $\boldsymbol{a}_t$.
- $\mathcal{O}^i$ represents the observation space for each agent $1 \leq i \leq N$, where
  - $o_t^i \in \mathcal{O}$ represents an observation for each agent $1 \leq i \leq N$ at time step t
- $O^i(s_t)$ of $\mathcal{S} \rightarrow \mathcal{O}$ represents the observation function for each agent $1 \leq i \leq N$.
- $\mathcal{R}(s_t, \boldsymbol{a}_t)$ of $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ represents a collaborative immediate reward function yielding r_t.
- $\rho(\cdot)$ of $\emptyset \rightarrow \mathcal{S}$ represents the function yielding initial state s_0.
- N represents the number of agents.

The Dec-POMDP is a generalisation of a Partially Observable Markov Decision Process (POMDP) which allows us to model multiple decentralised agents. Depending on our requirements, it can be used to create a model which takes into account various sources of uncertainty. In this work, this is primarily useful as it allows us to consider nonexistent communication. In addition to this, it allows us to consider the incomplete knowledge of the trajectories of 'the other agents' and thus their behaviours.

States are either discrete or continuous, with this work opting to experiment with discrete environmental states (the findings from this work should, however, also apply to continuous environmental states). The set of actions available to agent i ($\mathcal{A}^i$) is, however, assumed to be discrete. Moreover, passage of time in the environment is assumed to be discrete, happening at discrete time steps. At any such time step t, a next state $s_{t+1} \in \mathcal{S}$ is drawn from T using current state $s_t \in \mathcal{S}$ and joint action $\boldsymbol{a}_t \in \mathcal{A}$ ($s_{t+1} \sim P(\cdot|s_t, \boldsymbol{a}_t)$), the transition between which yields collaborative reward $r_t := \mathcal{R}(s_t, \boldsymbol{a}_t)$ [Böhmer et al., 2020].

The state space is not directly observable in this work. Agents instead are provided with $o_t^i \sim O^i(\cdot|s_t)$, an observation drawn using the agent's observation function. Whilst the output of this observation

function and its relation to the state can take many forms, the experiments in this work use observations which are agent-centred representations of the state s_t, from which some information is always hidden (actor identities: we opt to only represent actor 'types' in order to improve convergence speed, as actors are homogeneous when they are of the same type). We will primarily evaluate fully observable environments where these observations encompass the entirety of the state, yet will also evaluate versions of the experiment environment which are partially observable, in which case an observation radius applies, centred upon each agent, outside of which the respective agents cannot observe state information.

In the environments considered in this work, the initial state $s_0 \sim \rho(\cdot)$ is drawn randomly from the set of possible initial states (i.e. having the appropriate number of actors). The environment states themselves represent a grid world in which actors cannot occupy the same space. In addition, this work considers episodic tasks which yield episodes $\{s_0, \{o_0^i\}_{i=1}^N, \boldsymbol{a}_0, r_0, \ldots, s_h, \{o_h^i\}_{i=1}^N, \boldsymbol{a}_H\}$. Episodes have finite length $H \leq \mathcal{H}$ (some pre-defined maximum episode length or fewer time steps). Episodes can be terminated before $\mathcal{H}$ if a terminal state is reached. In the environments used in this work, episodes are terminated if it is no longer possible to change the amount of reward obtained in the episode (insufficient number of actors).

What makes Dec-POMDPs decentralised where a multi-agent POMDP is not is that in a Dec-POMDP, agents only know their own individual action a_t^i and observation o_t^i at any time step t. Each agent selects an action, which form a joint action when taken together, which in turn leads to a state transition [Oliehoek & Amato, 2016]. In addition to this, agents are assumed to act based on their individual observations alone during execution, hence assuming no (explicit) communication [Oliehoek & Amato, 2016].

The states of MDPs normally have an important property, the Markov property: one is able to deduce all relevant information about the environment from the current state itself. To keep track of this information, it is not required to keep track of the history of transitions which has led up to the current state. Policies for MDPs can therefore be a function which maps a state s to an action a. If reinforcement learning agents are placed in a domain with the Markov property, convergence to an optimal policy is guaranteed at $t = \infty$. By contrast, in (Dec-)POMDPs, the observations often do not have the Markov property, nor do agents have access to the states upon execution. Hence, the learned policy cannot condition on the state [Oliehoek & Amato, 2016].

As a consequence, one has to use information sources other than the current state s_t or observation o_t^i alone to condition our learned policy upon. Without the need for us to send individual agents more information than they already possess, they are able to keep track of two histories: their observation history $\{o_0^i, o_1^i, \ldots, o_{t-1}^i, o_t^i\}$, and their action history $\{a_0^i, a_1^i, \ldots, a_{t-1}^i\}$. Combining these yields an action-observation history $\tau_t^i = \{o_0^i, a_0^i, o_1^i, a_1^i, \ldots, o_{t-1}^i, a_{t-1}^i, o_t^i\}$. This also allows us to define a joint history $\boldsymbol{\tau}_t := [\tau_t^1, \ldots, \tau_t^N]$.

Interestingly, Multi-agent POMDPs can condition on this joint history $\boldsymbol{\tau}_t$, which can be a sufficient statistic over a belief distribution over the state space, which in turn follows the Markov property. They are able to construct a belief over states. This allows them to solve an induced Multi-agent Markov Decision Process where the histories take the role of Markov states. Dec-POMDPs sadly do not have access to the joint history $\boldsymbol{\tau}_t$, which prevents them from conditioning their policies upon it.

For Dec-POMDPs, a belief over states alone does not suffice; beliefs also have to take the behaviour of other agents into account, as it is unknown what information is available to them, preventing agents from predicting what actions the other agents will take. Hence, problem optimisers need to make assumptions about the policies of other agents when only having individual beliefs based on τ_t^i available to them to solve the Dec-POMDP [Oliehoek & Amato, 2016]. *"In a Dec-POMDP, agents do not have access to a Markovian signal during execution"* [Oliehoek & Amato, 2016].

2.1. Q-Learning, (Double) Deep Q-Learning, DQNs, and DRQNs

As mentioned before, the goal of MARL is to enable sets of agents to develop policies to achieve their respective goals. In other words, the goal is to find an optimal joint policy π^* (or optimal policies for each agent).

For MDPs, this policy can be defined as a function $\pi^*(s_t, \boldsymbol{a}_t) : \mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\}$ [Montague, 1999] which allows agents to maximise the expected discounted sum of rewards through (joint) action $\boldsymbol{a}_t$ from any state s_t.

Q-Learning is a methodology which allows us to converge towards an optimal policy by greedily choosing (joint) action $\boldsymbol{a} \in \mathcal{A}$, which maximises the corresponding optimal Q-value. This optimal Q-value, in turn, is the estimated sum of discounted rewards starting from some state-action pair using the optimal policy: $Q^*(s, \boldsymbol{a}) := \mathbb{E}_{\pi^*}[\sum_{t=0}^{h-1} \gamma^t r_t |_{\boldsymbol{a}_0 = \boldsymbol{a}}^{s_0 = s}] = r(s, \boldsymbol{a}) + \gamma \int T(s, \boldsymbol{a}, s') \max_{\boldsymbol{a}' \in \mathcal{A}'} Q^*(s', \boldsymbol{a}') ds'$ [Böhmer et al., 2020]. In Markovian domains with discrete state and action spaces, the optimal Q-value (q^*) can be learned, in the limit, from interactions with the environment, through *Q-Learning* [C. J. Watkins & Dayan, 1992]. For large and continuous state spaces, estimation of the (state-action) Q-table can, however, be intractable: large state spaces can cause computation to become infeasible due to exceedingly large Q-tables, whereas continuous state spaces yield infinitely large tables. Hence, for large or continuous state spaces, we need to approximate q^*.

One of the ways in which q^* can be approximated is through the use of a deep neural network. One such deep neural network is the Deep Q-Network (DQN) [Mnih et al., 2015]. When learning with DQNs, the Q-value function - also known as the action-value function - is replaced by approximate Q-value function $Q(s, a; \theta_I)$. This approximate Q-value function is parameterised on the weights of a deep convolutional neural network at learning iteration (gradient descent update step) I [Mnih et al., 2015], a network which takes any s and $\boldsymbol{a}$ as input. In Q-Learning, one updates entries of a state-action matrix (Q-table). In Deep Q-Learning, one updates deep neural network parameters θ. Equation 2.2 (update rule) and 2.3 ('target') represent the direct port of the update rule for DQNs, where we assume, like in standard Q-learning, the network is updated after taking action $\boldsymbol{a}_t$ from state s_t, resulting in the observation of r_{t+1} and s_{t+1} [Van Hasselt et al., 2016].

$$\theta_{t+1} = \theta_t + \alpha(Y_t - Q(s_t, \boldsymbol{a}_t; \theta_t)) \nabla_{\theta_t} Q(s_t, \boldsymbol{a}_t; \theta_t) \tag{2.2}$$

$$Y_t = r_t + \gamma \max_{\boldsymbol{a}} Q(s_{t+1}, \boldsymbol{a}; \theta_t) \tag{2.3}$$

During learning, various stabilisation techniques are, however, often applied to prevent instabilities and divergent behaviours which are otherwise introduced by representing the Q-value function through the use of a non-linear function approximator [Mnih et al., 2015; Tsitsiklis & Van Roy, 1996]. The two main stabilisation techniques applied are *experience replay* [Lin, 1992; Mnih et al., 2015; O'Neill et al., 2010] and *Double (Deep) Q-Learning* [Hasselt, 2010; Mnih et al., 2015; Van Hasselt et al., 2016].

Experience replay is one of the most-used techniques for stabilising the learning process of deep learning agents. It allows one to separate the learning phase from the process of gaining experience. Its most important characteristic is that it allows one to reduce correlations in the observation sequence by sampling (random) past experiences. Additionally, it allows one to draw a (mini-)batch of experiences from a pool of samples, which allows one to learn from multiple experiences in one learning iteration, smoothing out changes. In DQNs, experiences most often are defined as $e_t = \{s_t, \boldsymbol{a}_t, r_t, s_{t+1}\}$ [Mnih et al., 2015; Van Hasselt et al., 2016]. These are then stored in dataset $D_t = \{e_1, \ldots, e_t\}$, a type of 'experience replay buffer'.

Double Q-Learning is a technique which allows one to prevent large overestimations of action (Q-)values which are typically introduced due to Q-Learning using the maximum action value as an approximation for the maximum expected action value [Hasselt, 2010]. Q-Learning updates would typically be defined as $Q_{t+1}(s_t, \boldsymbol{a}_t) = Q_t(s_t, \boldsymbol{a}_t) + \alpha_t(s_t, a_t)(r_t + \gamma \sum_{s_{t+1}} T(s_t, \boldsymbol{a_t}, s_{t+1})(\max_{\boldsymbol{a} \in \mathcal{A}} Q_t(s_{t+1}, \boldsymbol{a}) - Q_t(s_t, \boldsymbol{a_t})))$ where α_t is the learning rate at time step t. Henceforth, as state transitions in this work are generally deterministic, we shall omit the transition probability from equations. For clarity, we

shall rewrite the equation as $Q_{t+1} = Q_t(s_t, \boldsymbol{a}_t) + \alpha(Y_t - Q_t(s_t, \boldsymbol{a_t}))$, where Y_t is defined as $Y_t = r_t + \gamma \max_{\boldsymbol{a} \in \mathcal{A}} Q_t(s_{t+1}, \boldsymbol{a})$.
Looking at the target used in method to update Q-values, we can see that a maximum is taken over a future Q-value estimator for the maximum action value. This results in a significant positive bias, and consequently overestimations [Hasselt, 2010]. Double Q-learning uses a double estimator method with which one randomly updates one of the estimators using a target computed with the other estimator, instead of using the same [Hasselt, 2010]. The maximisation operation in the target is decomposed into parts corresponding to action selection and action evaluation. This significantly reduces the chance one bootstraps from overestimations, preventing them from becoming large. Given two estimators x and y and some action $\boldsymbol{a}$ based on $Q^x(s, \cdot)$ and $Q^y(s, \cdot)$, yielding r and s', we randomly update estimator $z \in \{x, y\}$ using estimator $w \neq z \in \{x, y\}$ using update rule $Q^z(s, \boldsymbol{a}) \leftarrow Q^z(s, \boldsymbol{a}) + \alpha(Y_t - Q^z(s, \boldsymbol{a}))$ with $Y_t = r + \gamma Q^w(s', \arg\max_{\boldsymbol{a}'} Q^z(s', \boldsymbol{a}'))$.

In Double Deep Q-Learning, Double Q-Learning principles are ported to DQN updates. The argument for the need of Double Q-Learning still holds for DQNs: using the same predictor to select and evaluate an action makes it very likely to select already overestimated values, resulting in overoptimistic value estimates [Van Hasselt et al., 2016].
Instead of using two alternatingly updating i.i.d estimators which bootstrap of each other for target computations, the DQN algorithm utilises a target network with parameters θ' [Mnih et al., 2015]. The parameters of this target network lag behind the ones of the online network, being updated periodically such that $\theta' \leftarrow \theta$ [Mnih et al., 2015]. It should be noted that it is also possible to alternatingly use two seperate independent deep neural networks, or to symmetrically updating the online (henceforth 'predictive') and target networks by switching their roles periodically. However, whilst the Double Deep Q-Network (DDQN) approach may - strictly speaking - not be utilising truly independent estimators as both networks are not completely decoupled, utilising a periodically updated target network grants one most of the benefit of Double Q-Learning with minimal computational overhead [Van Hasselt et al., 2016].

Utilising both stabilisation techniques allows one to reduce the correlations between action values and both the observation sequence and the target. We redefine our target (from Equation 2.3) to include double Q-learning (Equation 2.4), and define L2 loss (Equation 2.5) between the predictive network and a target value [Mnih et al., 2015]. This is used to enable us to generate network updates from (mini-)batches of experiences from our experience replay buffer, instead of singular ones, by applying gradient descent to minimise this loss.

$$Y_t = r_t + \gamma Q(s_{t+1}, \arg\max_{\boldsymbol{a}} Q(s_{t+1}, \boldsymbol{a}; \theta_I); \theta'_I) \tag{2.4}$$

$$L_I(\theta_I) = \mathbb{E}_{(s_t, \boldsymbol{a}_t, r_t, s_{t+1}) \sim D_I}[(Y_t - Q(s_t, \boldsymbol{a}_t; \theta_I))^2] \tag{2.5}$$

It is important to note that conventional (D)DQNs most often are not suitable when considering a (Dec-)POMDP: conventional DQN agents condition their policies on $\boldsymbol{a}_t$, which, as mentioned before, Dec-POMDP agents' policies likely cannot condition on. In addition, they condition on s_t and s_{t+1}, which (Dec-)POMDP agents likely cannot condition on either. Instead, for partially observable decision processes, an approximate Q-value function should be defined which conditions on another information source which definitely *is* available to the agents. As mentioned before, action-observation histories fit this description.

Deep recurrent Q-learning for partially observable mdps [Hausknecht & Stone, 2015] introduces the (Double) Deep Recurrent Q-Network (D)DRQN architecture, and aims to provide a methodology for deep reinforcement learning which is suitable for POMDPs. In Deep Recurrent Q-Networks, the approximate Q-value function is conditioned on agents' history $\boldsymbol{\tau}_t$. To achieve this, a Recurrent Neural Network (RNN) is utilised, or to be more precise, the first set of DQN layers are replaced with recurrent layers. This network does not have states as inputs, but is provided with (joint) observations and actions (up to $\boldsymbol{o}_t$ and $\boldsymbol{a}_{t-1}$ at time t). The approximate Q-value function conditions on the RNN's hidden state h. Evaluations show that the usage of recurrency allows POMDP agents to better estimate the underlying system state, thereby narrowing the gap between $Q(\boldsymbol{o}, \boldsymbol{a}|\theta)$ and $Q(s, \boldsymbol{a}|\theta)$ [Hausknecht & Stone, 2015].

When sampling from the experience replay buffer, the networks described up until now all sampled batches of individual experiences. For RNNs, the usage of these batches is rather inefficient, as utilising these i.i.d. sample experiences would require one to process the transitions of the corresponding episodes up to corresponding time steps t to compute the Q-values which depend on action-observation histories. One of the ways in which the idea of experience replay buffers can efficiently be applied to the DRQNs is to sample entire action-observation and corresponding reward histories (together representing episodes) or subsections thereof. *Bootstrapped Sequential Updates*, in which episodes are selected randomly from replay memory and the RNNs hidden state is carried forward throughout the episode, these longer sequences can allow the network to gain more knowledge on the environmental state, yet violate the random sampling policy which was meant to prevent correlations from the observation sequence [Hausknecht & Stone, 2015]. Experiments, however, indicate that the method is viable and competitive with alternatives [Hausknecht & Stone, 2015]. To limit complexity, this approach shall be used.

We rewrite our loss (Equation 2.7) and target (Equation 2.6) functions one last time to account for the changes introduced by usage of the DDRQN architecture.

$$Y_t = r_t + \gamma Q(\boldsymbol{\tau}_{t+1}, \arg\max_{\boldsymbol{a}} Q(\tau_{t+1}, \boldsymbol{a}; \theta_I, h_t^{\theta}); \theta_I', h_{t+1}^{\theta'}) \tag{2.6}$$

$$L_I(\theta_I) = \mathbb{E}_{\boldsymbol{\tau}_H, r_0, \ldots, r_H \sim D_I}[\sum_{\tau_t \in \tau_H} (Y_t - Q(\boldsymbol{\tau}_t, \boldsymbol{a}_t; \theta_I, h_t))^2] \tag{2.7}$$

Where h^{θ} and $h^{\theta'}$ denote the hidden states of the predictive and target networks respectively.

2.2. Joint- and Independent Learners

Provided with a Q-value function conditioning on $\boldsymbol{\tau}_t$ and $\boldsymbol{a}_t$ (such as the approximate Q-value function for DRQN, or even an augmented version of conventional Q-Learning for smaller cases), one can start to learn an optimal policy π^* for a POMDP by maximising the Q-value function from Equation 2.8 with respect to joint policy π (breaking arg max ties through some deterministic procedure).

$$\begin{gathered} Q^*_{\mathsf{JQL}}(\boldsymbol{\tau}_t, \boldsymbol{a}_t) := \max_{\pi} Q^{\pi}(\boldsymbol{\tau}_t, \boldsymbol{a}_t) = r(\boldsymbol{\tau}_t, \boldsymbol{a}_t) + \gamma \int T(\boldsymbol{\tau}_t, \boldsymbol{a}_t, d\boldsymbol{\tau}_{t+1}) V^*_{\mathsf{JQL}}(\boldsymbol{\tau}_{t+1}) \\ \pi^*(\boldsymbol{\tau}_t, \arg\max_{\boldsymbol{a} \in \mathcal{A}_t} Q^*_{\mathsf{JQL}}(\boldsymbol{\tau}_t, \boldsymbol{a}_t)) = 1 \\ V^*_{\mathsf{JQL}}(\boldsymbol{\tau}_t) := \max_{\boldsymbol{a}_t \in \mathcal{A}_t} Q^*_{\mathsf{JQL}}(\boldsymbol{\tau}_t, \boldsymbol{a}_t) \end{gathered} \tag{2.8}$$

The main problem with this is, however, that if joint Q-learning is applied to multi-agent tasks, the combinatorial growth of joint spaces quickly makes the problem intractable. In addition, joint Q-learning conditions on $\boldsymbol{\tau}_t$ and $\boldsymbol{a}_t$, which one can condition on in POMDPs, but not in Dec-POMDPs.

In order to allow for efficient maximisation for Multi-Agent Q-Learning, and to allow for the development of decentralised policies, various algorithms have been developed. One of the earliest and most straightforward approaches is to model each decentralised agent as an independent learner, which in turn estimates the state-action (or in our case own history-action) value function by assuming all other agents follow a stationary decentralised policy [Tan, 1993]. In other words, this methodology, Independent Q-Learning (IQL), causes its agents to consider the other agents part of the environment.

Each agent trains its own decentralised policy π_i using a DRQN with $Q^i(\tau_t^i, a^i; \theta)$. Due to the assumption that other agents are part of the environment, value functions can be trained without knowledge on other agents (Equation 2.9).

$$\begin{aligned} Q^i_{\mathsf{IQL}}(\tau_t^i, a_t^i; \pi^{-i}) :&= \mathbb{E}[Q^{\pi}(\boldsymbol{\tau}_t, \boldsymbol{a}_t) | a_t^i, \tau_t^i] \\ &= \mathbb{E}[r(\boldsymbol{\tau}_t, \boldsymbol{a}_t) | a^i, \tau_t^i] + \gamma \int \mathbb{E}[T(\boldsymbol{\tau}_t, \boldsymbol{a}_t, \boldsymbol{\tau}_{t+1}) | a^i, \tau_t^i] \max_{a' \in \mathcal{A}^i} Q_{\mathsf{IQL}}(\tau_{t+1}^i, a'; \pi^{-i}) \end{aligned} \tag{2.9}$$

Each agent trains its own (Dec-)POMDP $\langle \mathcal{S}, \mathcal{A}^i, T^i, \mathcal{O}^i, O^i, r^i, \rho \rangle$ induced by the joint policy over all *other* agents, π^{-i}. Assuming that other agents are part of the environment implies that one also assumes that these policies are stationary. In case of learning processes in which agents learn at the same time, self-play, these policies will however be stochastic. This makes the Dec-POMDP induced by π^{-i} non-stationary, potentially making it instable due to the violation of the stationarity assumption for the transition function [Foerster et al., 2017]. Whilst this possible instability can be partially counteracted through more involved replay memory sampling techniques [Foerster et al., 2017], these are often not applied, and hence we shall not assume these sampling techniques either.

Despite this possible source of instability, IQL is widely used in practice due to its simplicity, adaptability, and readily available efficiency-improving methodologies. One such sample efficiency-improving methodologies is parameter sharing, a methodology from the family of *centralised training for decentralised execution* methods where the parameters the networks of homogeneous agents are shared [Kaushik, Krishna, et al., 2018; Terry et al., 2020]. As the agents in this work will be homogeneous, this technique *will* be applied.

2.3. Stochastic and Greedy Action Selection

For any reinforcement learning algorithm, deciding when to explore the environment and when to exploit the current policy is an important choice. Generally speaking, one would like to explore a lot when agents need to broaden their knowledge about taking actions in the environment, whilst one would like to greedily exploit current policies to steer towards what agents perceive to be the path to bring it closer to the best attainable reward.

Various methodologies exist for balancing the amount of exploration and exploitation in the process of gathering experiences. One of the simplest thereof is ϵ-greedy action selection (Equation 2.10). With ϵ-greedy action selection the action to be taken is determined through a stochastic process when training, allowing the agent to either randomly explore with probability ϵ, or to exploit its current policy.

$$a_t^i = \begin{cases} \text{arg max}_{a \in a_t^i} Q(\tau_t^i, a; \theta_t) & \text{with probability } 1 - \epsilon \\ \text{random action } a_t^i \in \mathcal{A}^i & \text{with probability } \epsilon \end{cases} \tag{2.10}$$

Exploration might be especially important in the beginning of our learning process, with sufficient exploitation becoming increasingly important in later stages of learning, where the agent has learned about the value of taking actions given some information from the environment. To account for this, ϵ is often scheduled, which allows one to decrease ϵ over time.

$$\epsilon = \text{min}(\epsilon_{\text{start}}, \text{max}(\epsilon_{\text{end}}, \epsilon_{\text{start}} - (\frac{\epsilon_{\text{start}} - \epsilon_{\text{end}}}{t_{\text{decay}}}) * (t - t_{\text{burn in}}))) \tag{2.11}$$

Equation 2.11 shows a linear example schedule for ϵ, where ϵ_{start} and ϵ_{end} denote the start and end values for ϵ respectively, and t_{decay} and $t_{\text{burn in}}$ denote the time span over which ϵ should decay and the time span over which ϵ should initially be kept high respectively to allow for a prolonged period of highly random action-taking (i.e. a 'burn-in time').

2.4. Relative Overgeneralisation

Independent Q-Learning has been shown to work when each agent is able to accomplish a task by itself [Tan, 1993]. When this is not the case, it depends:

Stochastic exploration by other agents introduces a problem for factorisation approaches [Böhmer et al., 2020; Rashid et al., 2018; Son et al., 2019; Sunehag et al., 2017]. The expected value of the action which is optimal can become sub-optimal from the perspective of agents if the other agents do not cooperate enough on average.

This phenomenon, a pathology called *relative overgeneralisation*, occurs when there exists an action for which the expected Q-value exceeds the the expected Q-value of the optimal action assuming optimal actions from other agents. In other words, agents are able to gravitate towards a relatively stable but sub-optimal joint policy due to noise in learning updates induced by 'other agents" exploration strategies [Wiegand, 2004]. Relative overgeneralisation draws independent learners to sub-optimal wide peaks in the reward space due to the perceived likelihood of 'collaborating' there [Panait, Sullivan, et al., 2006].

$$\exists a' \in \mathcal{A}^i : Q^i_{\mathsf{IQL}}(\tau^i_t, a'; \pi^{-i}) > Q^i_{\mathsf{IQL}}(\tau^i_t, a^*; \pi^{-i}) \tag{2.12}$$

Equation 2.12 shows the condition for relative overgeneralisation occurring in IQL, with π^{-i} once again denoting the joint policy of all other agents, and $a^* = \arg\max_{a \in a^i} \max_{\pi^{-1}} Q^i_{\mathsf{IQL}}(\tau^i_t, a; \pi^{-i})$.

The 2D Hunter-Prey environment, as introduced in Section 1, exemplifies the impact of punishments from miscoordinated actions due to exploration. It should be noted that relative overgeneralisation is a stochastic phenomenon, meaning that performance in practice is heavily dependent on experiences.

If we consider a version of the 2D (which is henceforth omitted) Hunter-Prey environment with two hunters and one prey, in which the two agents with policies π^1 and π^2 respectively and the prey are located in a configuration such that the agents could successfully catch the prey if their actions are coordinated properly (i.e. the agents both are adjacent to the prey). Assuming an initially uniform exploration policy, we can predict the impact of punishment p for miscoordinated actions on the performance of IQL with Equation 2.13. This gives $p > \frac{1}{4}r$ for our condition at which the optimal action **C** is relatively consistently considered sub-optimal by agent 1, preventing optimal cooperation.

$$Q^1_{\mathsf{IQL}}(\{\mathbf{L}/\mathbf{R}/\mathbf{U}/\mathbf{D}/\mathbf{I}\}) > Q^1_{\mathsf{IQL}}(\mathbf{C}) \text{ when } \pi^2(\mathbf{C})r - (1 - \pi^2(\mathbf{C}))p < -\pi^2(\mathbf{C})p \tag{2.13}$$

2.5. Distributed Q-Learning

Distributed Q-Learning, used as a shorthand for "an algorithm for distributed reinforcement learning on the basis of Q-learning" [Lauer & Riedmiller, 2000], is an exploration into the possibilities of performing 'distributed' (decentralised) reinforcement learning given coordination problems and the question of how to project large state-(actions for each agent) tables to suitable state-action tables for each agent.

The last remark is a comment on the fact that it is not trivial to determine how to construct state-action tables for each agent when taking them from a large centralised state-'joint action' table [Lauer & Riedmiller, 2000]. independent learners cannot compute Q-tables of the form $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, as they only know their own action $a_i \in \mathcal{A}$. They could, however, compute Q-tables of the form $Q : \mathcal{S} \times \mathcal{A}^i \to \mathbb{R}$. Projecting from the larger tables to the smaller tables inherently requires making assumptions on the behaviours of other agents [Lauer & Riedmiller, 2000]. A straightforward approach would be to assume that all other agents' actions are equally likely to occur, in which case one can take an average of the relevant entries of the larger table. Another straightforward approach would be to keep track of occurrence rates of other agents' actions, in which case we can take a weighted average instead (which is somewhat analogous to the assumption introduced in Section 2.2, independent learners: the assumption that the other agents are part of the environment).

The authors find that projections like these lead to results which are highly dependent on the learning algorithm and its configuration (explicitly mentioning the exploration strategy), and that these approaches do not yield (nearly-)optimal policies in some explored examples [Lauer & Riedmiller, 2000]. They introduce alternative projections which reduce this effect: projections using a 'pessimistic assumption' and a 'optimistic assumption' respectively.

The pessimistic assumption is quickly disregarded due to it resulting in excessively cautious policies, but the optimistic assumption yields interesting results. By setting the Q-value in table $Q^i(s, a^i)$ to the maximum value occurring in table Q where the action of agent i in joint action $\boldsymbol{a}$ corresponds with a^i,

one implicitly assumes that all other agents act in a way which optimises the cumulative reward [Lauer & Riedmiller, 2000]. This causes the agents to act as if their teammates act in an optimal manner (provided that through experiences table entries have been calculated which indicate there exist such ways to act). This way of acting prevents relative overgeneralisation from occurring (as often), as negative experiences from taking action a^i are no longer being represented in the Q^i individual Q-tables of agents, only the maximum persists. The optimistic assumption is able to induce optimistic behaviours without adding additional communication between the agents, and can be fully decentralised through the use the iteration rule defined in Equation 2.14 (executed after every transition, for every agent independently) [Lauer & Riedmiller, 2000]. Interestingly, where conventional Q-learning uses a learning rate $\alpha < 1.0$, Distributed Q-Learning uses $\alpha = 1.0$. Current Q-values are effectively completely replaced by observed rewards and follow-on utility estimates when they are found to result in a greater Q-value.

$$\begin{aligned} Q_0^i(s_t, a_t) &= 0 \\ Q_{t+1}^i(s_t, a_t) &= \max\{Q_{t+1}^i(s_t, a_t), r(s_t, \boldsymbol{a}_t) + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q_t^i(s_{t+1}, \boldsymbol{a}')\} \end{aligned} \tag{2.14}$$

In addition to this iteration rule, Distributed Q-Learning applies another trick to prevent coordination problems other than relative overgeneralisation, following from the Pareto selection problem: policies are updated if, and only if, a higher Q-value has been stored in the agents Q-table Q^i since the last update (Equation 2.15). This breaks ties by effectively allowing only the first-found best action to be incorporated as such in the policies.

$$\pi_{t+1}^i(s) \leftarrow \begin{cases} \pi_t^i(s) & \text{if } s \neq s_t \text{ or } \max_{\boldsymbol{a} \in \mathcal{A}} Q_t^i(s, \boldsymbol{a}) = \max_{\boldsymbol{a} \in \mathcal{A}} Q_{t+1}^i(s, \boldsymbol{a}) \\ a_t^i & \text{otherwise} \end{cases} \tag{2.15}$$

Distributed Q-Learning, based on Q-tables, has the same scaling issues as other conventional tabular Q-learning techniques, meaning that it does not scale well with the state space, making solving problems with it intractable for large or continuous state spaces. It additionally should be noted that stochasticity other than stochasticity from agent exploration could not not be handled in a way which ensured the development of an optimal policy: whilst stochasticity induced by the behaviours of other agents can be maximised over, other random influences have to be taken into account by leveraging their expected values. As no way was found to distinguish between the kinds of stochasticity, they also could not be handled separately [Lauer & Riedmiller, 2000].

2.6. Lenient Learning

Lenient Learning, like Distributed Q-Learning, is a maximum-based learning approach. Leniency as a concept for concurrent learning algorithms was introduced in a field related to reinforcement learning, cooperative co-evolution (evolutionary algorithms), to help concurrently evolving agents to converge to an optimal joint policy [Potter & Jong, 1994] and especially to prevent relative overgeneralisation [Wiegand, 2004]. *Theoretical Advantages of Lenient Learners: An Evolutionary Game Theoretic Perspective* [Panait et al., 2008] highlights that the fields of co-evolution and Multi-Agent Q-Learning do not only share the influence of pathologies in cases where no lenience is applied, but also respond similarly when leniency is introduced. Subsequently, leniency was applied to Multi-Agent Q-Learning and developed into a new branch of maximum-based approaches.

Lenient learners are learners which are somewhat lenient towards other agents. That is, they can implicitly forgive sub-optimal actions by teammates by ignoring corresponding table. Typically, lenient learners transition from being an optimistic learner to an average-based learner by reducing the amount of 'leniency' over time. This allows these learners to effectively explore large parts of the state space which would otherwise already be considered to be fruitless.

In order to exemplify how lenient approaches work, we summarise the steps taken by the Lenient Multi-Agent Q-Learning algorithm introduced in *Lenient Learning in Independent-Learner Stochastic*

Cooperative Games [Wei & Luke, 2016] (for brevity we omit moderation factors):

1. $\forall$ agent $1 \leq i \leq N \; \forall s \in \mathcal{S} \; \forall a^i \in \mathcal{A}^i : Q^i(s, a^i) \leftarrow \infty$.
   - Initialise all state-action tables for all agents to infinity to signal that the value should be replaced by the first-encountered reward.
2. $\forall$ agent $1 \leq i \leq N \; \forall s \in \mathcal{S} \; \forall a^i \in \mathcal{A}^i : \mathcal{T}^i(s, a^i) \leftarrow$ maxTemp.
   - Temperatures are defined to be real numbers between minTemp and maxTemp. Each state-action pair has its own temperature, which affects two mechanisms in the algorithm: action selection and lenience (see later steps).
3. $s \leftarrow s_0$.
4. Repeat ad infinitum (or end state if defined):
   (a) $\bar{\mathcal{T}}^i(s) \leftarrow \text{mean}_{a \in \mathcal{A}} T^i(s, a)$
      - Calculate the mean temperature in current state s. This average temperature affects action selection:
   (b) if $\bar{\mathcal{T}}^i(s) < \text{minTemp}$ or $\max_{a \in \mathcal{A}} Q^i(s, a) = \infty$ use conventional Q-Learning greedy action selection. Else $a^i \in \mathcal{A}^i$ and $a^i \sim P^i$ with $\forall a : P^i_a \leftarrow \frac{W^i_a}{\sum_j W_j}$ with $\forall a : W^i_a \leftarrow e^{\frac{Q^i(s,a)}{\bar{\mathcal{T}}^i(s)}}$.
      - If the temperature has been lowered to below the minimum or if no maximum Q-value has yet been calculated for state s, we effectively use the action selection of a conventional Q-learner. Otherwise, Boltzmann Selection, a stochastic action selection methodology, is used.
   (c) *from state s, perform action a^i. Transition to new state s' after receiving reward r.*
   (d) $rand \leftarrow$ uniform random value in range $[0, 1]$.
   (e) $Y^i_t \leftarrow \begin{cases} r & \text{if } max_{a' \in \mathcal{A}^i_t} Q(s', a') = \infty \\ r + \gamma \max_{a' \in \mathcal{A}^i_t} Q(s', a') & \text{otherwise} \end{cases}$
      - Calculate a target, only include future if a Q-value has already been calculated for that state.
   (f) $Q^i(s, a^i) \leftarrow \begin{cases} Y_t & \text{if } Q^i(s, a^i) = \infty \\ \alpha Y^i_t + (1 - \alpha) Q^i(s, a^i) & \text{otherwise, if } Q^i(s, a^i) \leq Y^i_t \text{ or } rand < 1 - e^{\frac{-1}{\mathcal{T}(s, a^i)}} \\ Q^i(s, a^i) & \text{otherwise} \end{cases}$
      - Set the Q-value of this state action pair to the first target encountered after initialisation. If a value was already defined, only update the value if it would increase (positive TD-Error) or with some random chance inversely depending on the state-action temperature. Discard the update otherwise.
   (g) $\mathcal{T}^i(s, a^i) \leftarrow \delta \times \begin{cases} (1 - \Xi)\mathcal{T}^i(s, a^i) + \bar{\mathcal{T}}^i(s') & \text{if } s' \text{ is not an end state (if these exist)} \\ \mathcal{T}^i(s, a^i) & \text{otherwise} \end{cases}$
      - Lower the temperature of the state-action pair. This will cause action selection to be less random and more greedy upon the next encounter, and will decrease the leniency upon next encounter as well, as the chance of randomly accepting a 'negative' table update increases. Here δ and Ξ are pre-defined temperature decay and temperature diffusion coefficients respectively.
   (h) $s \leftarrow s'$

In short, this lenient learner is an augmented Q-learner which uses a temperature-based stochastic action selection procedure and table updates. It only updates Q-table values if a greater value is found or if, by chance, the agent does not apply leniency. This procedure allows agents to transition from being optimistic to being average reward learners for frequently encountered state-action pairs, allowing the agents to outperform optimistic and maximum-based learners (such as optimistic Distributed Q-Learners) in environments with misleading stochastic rewards [Palmer et al., 2017; Wei & Luke,

2016].

As this kind of Lenient learning approach is still tabular, it suffers from the same scaling problems we discussed before. Hence, a Lenient Deep Reinforcement learning approach which combines Deep Reinforcement Learning and Lenient Learning was developed: Lenient Multi-Agent Deep Reinforcement Learning [Palmer et al., 2017]. Lenient Deep Q-Networks (LDQNs) include the concepts of 'temperatures' and accompanying 'lenience' from tabular Lenient Learning into deep learning by augmenting the DQN's replay buffer by redefining an experience to $e = (s_{t-1}, a^i_{t-1}, r_t, s_t, l(s_t, a^i_t)_t)$, where $l(s_t, a^i_t) = 1 - e^{-k \times \mathcal{T}^i(\phi(s_t), a^i_t)}$, and where ϕ is a hashing function [Palmer et al., 2017], and subsequently using leniency values from samples in the DQN loss computation.

Due to the fact that for large or continuous state paces, keeping track of temperature $\mathcal{T}^i$, previously defined as a state-action temperature-value table, would be equally impossible as it were for Q-values, this temperature is redefined to $\mathcal{T}^i(\phi(s), a^i)$. In LDQNs, temperatures are not stored in a state-action table, but in a hash-action dictionary. As a consequence, the algorithm does no longer keep track of the temperature for individual states, instead keeping track of the temperature for groups of 'similar' states (where similarity is determined by hash function $\phi(s)$). Hash-action temperatures are decayed slightly with each encounter [Palmer et al., 2017].

Having solved the problem caused by the intractability caused by the temperature table, LDQNs can apply this temperature to experiences in the replay memory (as mentioned above) and to action selection (similar to non-deep Lenient Learning). For action selection, the Boltzmann action selection strategy is, however, replaced by a temperature-based version of ϵ-greedy action selection: the normalised average temperature value for state s_t simply replaces ϵ [Palmer et al., 2017].

The way in which leniency is actually applied in the learning process, is through the use of the leniency values stored with experiences in the replay buffer. These are used to augment the loss function used in DQN updates. The modification to the loss (Equation 2.5) is that for each sample from the replay buffer in the batch, the leniency conditions from Step 4.f of the Lenient Learning approach above determine whether or not the sample is treated as if it were not part of the batch [Palmer et al., 2017].

LDQNs depend heavily on an appropriate hash function $\phi(s)$ to cluster similar states by mapping them to the same hash. Lenient Multi-Agent Deep Reinforcement Learning depends on a combination of (properly trained) auto-encoders and SimHash to calculate these hashes $\phi(s) \in \{0,1\}^k$. SimHash is a locality-sensitive hashing function which recently has been successfully applied in a wide range of applications [Charikar, 2002; Tang et al., 2017]. The notion of utilising state similarity is centred around random encodings of inputs, in case of both count-based exploration and LDQNs being the outputs of auto-encoders. SimHash is defined in Equation 2.16.

$$\phi(s) = (A * g(s) \geq 0) \in \{0,1\}^k \tag{2.16}$$

Here, $A \in \mathbb{R}^{k \times D}$ is a matrix of i.i.d. entries drawn from $\mathcal{N}(0,1)$, where k determines the length of the hash generated (and therefore consequently ultimately granularity/collision rates), with higher values leading to fewer collisions. D is an input vector, and $g : \mathcal{S} \to \mathbb{R}^{|D|}$ is an (optional) pre-processing function [Tang et al., 2017]. SimHash maps our state s to the same hash as states which result in an input vector D with a small angular distance to the one resulting from state s. In both methods mentioned above, the *optional* pre-processing step was filled using an auto-encoder (taking D as the vector of output neuron values). How optional this pre-processing step is in the applied context remains to be seen.

An important note on the applicability of LDQNs is that they can be applied for Decentralised Markov Decision Processes (Dec-MDPs) but not Dec-POMDPs, due to the fact that they condition on the state. Efficacy when conditioning on observations instead has, to our knowledge, not been evaluated yet.

2.7. Soft Network Updates

In *Continuous Control with Deep Reinforcement Learning* [Lillicrap et al., 2015], *soft target network updates* are introduced. Instead of directly copying the weights of the predictive network parameterised θ periodically such that $\theta' \leftarrow \theta$, they update the target network every time the predictive network is updated ($\theta' \leftarrow u_s\theta + (1 - u_s)\theta'$). This is to ensure that the target does not move excessively from a single update to the target network, which could cause instability. Whilst the approach can slow down learning, this was found to offset by the approach resulting in exceptional relative stability in the learning process [Kobayashi & Ilboudo, 2020; Lillicrap et al., 2015].

3

Methodology

We introduce an alternative approach to optimism-based prevention of relative overgeneralisation, *Deep Maximum Q-Learning* (DMQL). DMQL aims to apply the optimistic assumption introduced in Distributed QL [Lauer & Riedmiller, 2000] to the domain of Deep Reinforcement Learning, thereby granting agents the ability to learn to be cooperative.

DMQL differs from the existing deep maximum-based learning approaches by the way it introduces optimism: where LL approaches introduce optimism by omitting negative updates, i.e. leniency, DMQL introduces optimism by using maximum values encountered in past and current similar histories. One could interpret this as granting the agents the ability to learn from encountered positive experiences when a worse experience is encountered, as opposed to not learning from negative experiences at all. In the context of the Hunter-Prey environment introduced in Section 1, it grants the agents the ability to recall that they were rewarded by performing an action in a given similar state before, allowing them to overwrite the negative experience under the assumption that the negative reward is due to exploration of another agent.

Whilst exploratory experiments with stateless tabular cases indicate that using this replacement strategy instead of LL's discardment strategy only leads to marginal improvements in convergence speeds as a consequence of the increased number of updates actually performed, differences should be substantial for Dec-POMDPs, as well as for Dec-MDPs with large or continuous state spaces.
This intuition is based on two points related to learning rates and utilisation of similarity (of which the meaning is introduced later on) respectively. The former point, on learning rates, is based on the fact that after having encountered a high Q-value for a state-action pair, Distributed Q-learning regards this value the new maximum value which replaces the current value completely. Deep LL, by contrast, has a learning rate $\alpha < 1$, meaning that the maximum value found up to this point is not the new output of the state/hash-action function. The fact that updates are subsequently omitted if they are lower than the new output, and the fact that values encountered later on may be higher than the new output but lower than the encountered maximum, mean that we likely neither do truly converge to the maximum value encountered, nor with the maximum speed possible through gradient update steps with the specified learning rate. By contrast, DMQL most importantly *does* utilise every update step, whilst allowing the retention of the maximum value encountered to be configured. The latter point, based on what we can learn for various histories based on the occurrence of a positive TD-error, can best be introduced through a simplified example: given an unspecified number of histories which end up in the same critical state s (a state in which we can be rewarded or punished), the same performed action a, and the fact that one of these histories yields a positive TD-error, LL and DMQL learn quite differently: even if through some arbitrary methodology we determine all the considered histories to be similar, LL approaches will only learn that the history which yields rewards and the histories which led up to it are good. Similarity is only used to retrieve temperature values (determining leniency). By contrast, provided this similarity and the positive TD-error, DMQL will learn that all these similar histories are good, leading to better generalisation and higher sample efficiency.

In the following sections, we will introduce the design choices and related components of DMQL. Firstly, we discuss the concept of similarity between histories, problems one can encounter, and decisions we made to counter these problems. Secondly, we introduce DMQL's maximisation approach, building upon the design choices made to accommodate the usage of histories, and introducing how the stability of the learning process of DMQL agents can be improved. Subsequently, we show DMQL fits into the DRQN architecture, and how one can apply DMQL to an existing DRQN IQL implementation. Lastly, we build off of the concept of similarity between histories to explore how one can encode similarity into a hash function to further allow us to exploit similarities with DMQL, discussing pay-offs between various optional design choices, as well as various problems which one can encounter when developing a hashing strategy.

3.1. Similar Histories

In order to find a maximum value provided some history τ^i and access to all past encountered histories, one might consider maximising over exactly the same histories encountered before, with current network parameterisations. This, however, is likely quite infeasible to work, mostly due to the fact that the exact same history might not occur often enough, or even twice. In this case, DMQL would, de facto, behave like IQL due to the effective lack of a maximisation step, causing no values to be overwritten.

As a consequence, one might want to consider not only exactly the same histories, but also histories that are 'similar', where similarity is determined by some similarity metric. In this case, however, it is important to consider the fact that there exists no consensus on what makes a good similarity measure for multivariate time series, and that similarity calculations used are often quite computationally expensive [Kale et al., 2014]. In practice, this means that one likely has to investigate what parts of arbitrary agent histories would be important factors for similarity calculations for each problem.

In this work, we opt to start with a strategy employing centralised training and decentralised execution: by not limiting ourselves to decentralised learning, we are able to introduce a similarity metric based upon the states histories cause our agents to end up in (in addition to allowing us to use often-used optimisation strategies such as parameter sharing, mentioned in Section 2.2). In Section 3.4.1, we will elaborate on the meaning of our notions of similarity, but for now it suffices to know that our most strict notion of similarity is defined by two situations having the same optimal target values for a given agent. As long as the evolution of the Markov process in the future depends only on the present state and does not depend on past history, the state configuration is an information source we can use to reliably provide similarity metrics. The environment used to evaluate DMQL in this work has this property as rewards obtained are, for the most part, a result of the joint actions performed from a given state configuration. Other environments might require more insights due to more complex requirements or the unavailability of a sufficiently informative state representation. In these cases one could opt to include information from more time steps, yet as this does increase the number of perceived dissimilarities between histories, it *might* decrease efficiency.

3.2. Maximisation Strategy

As mentioned before, DMQL introduces a maximum into the target computation of a DRQN. In effect, we therefore aim to augment Equation 2.6 to establish Equation 3.1.

$$Y_t = \max(\mathcal{D}_I^i(\phi(s_t), a_t^i), r_t + \gamma Q(\tau_{t+1}, \arg\max_a Q(\tau_{t+1}, a; \theta_I, h_t^\theta); \theta'_I, h_{t+1}^{\theta'})) \tag{3.1}$$

Where $\mathcal{D}_I^i$ is some hash-action dictionary mechanism for agent i at learning iteration I, and where function $\phi(s_t)$ represents some hashing function which is able to encode a chosen notion of similarity (discussed in Section 3.4 below).

Whilst Equation 3.1 illustrates that the DMQL target is defined as a maximum over the target calculated for the current trajectory and some target retrieved using past experiences, it does not show how the target retrieved using past experiences is calculated. As the process used is not trivial, we first introduce used approximations and why they are needed.

3.2.1. Finding a Maximum from Past Experiences

Ideally, we would have liked to be able to determine the greatest target value from all past similar histories, using current network parameterisations. As parameterisations are updated each learning iteration, we do not know the values yielded by experiencing these similar histories without re-calculating them using current parameterisations. Re-calculating these values would require us to iteratively process each time step for each similar history up to the relevant trajectory length for a single target computation. Running the model using current parameters over every history to consider for every target calculation for each new time step in the main RNN observation processing loop does result in a quadratic time complexity assuming trajectories of similar length, hence requiring substantial time resources. This, in addition to the requirement to actually store all similar history tuples, makes for rather unrealistic resource requirements. Given resource constraints, we consequently have to use some approximations.

To prevent one from having to store and re-run similar histories for new network parameterisations, one might consider to simply store the similar history which has yielded the best values thus far. Whilst this does make much sense at first sight, requiring us to store and process only this 'best history' upon target computation when new network parameterisations are present, there are caveats once again: what is considered the 'best history' from all similar histories encountered in the past may not be considered the best among them when using future parameterisations.

Given that storing and processing all similar past histories is infeasible and that the concept of a 'best history' is not consistent between parameterisations, we push our approximation even further: instead of storing some 'best history' thus far, we look back to *Distributed Q-Learning* and store the *best estimated target value* thus far. Using this maximum value obtainable using past parameterisations has the same problems as storing the 'best history', on top of the fact that we do not account for changes in parameterisations anymore. Using this approach, however, we only need to calculate estimate target values once for each encountered history, and we only need to store a small set of values for each hash, instead of entire histories. We can additionally make many adjustments in the storage and retention, collection, and usage of stored values, including the fact that we can additionally maximise over values encountered in the same minibatch (re-defining our target to be $Y_t = \mathcal{D}_I^i(\phi(s_t), a_t^i)$ in the process).
We do, however, lose information. We no longer are able to ascertain that the stored values are the best value we could get using our current parameterisations. We are limited to a notion of 'the best estimate calculated using past parameterisations', which now only acts as an estimation of our maximum target value estimate. Additionally, due to the fact that we are in effect iteratively maximising over estimations, we are at great risk of overestimations being stored in our dictionaries, which will subsequently keep being used to overwrite current target values. This consequence is a factor which we deal with with a separate mechanism, introduced in Section 3.2.2.

3.2.2. DMQL's Dictionary Mechanism

Having established that DMQL's Dictionary Mechanism will store target estimations encountered using past and current network parameterisations, we establish that our dictionaries take the form $\mathcal{D}^i(\phi(s), a^i) = Y_{\mathsf{max}} : \mathcal{U}^* \times \mathcal{A}^i \to \mathbb{R}$ for each agent i, with $\phi(s)$ being a function $\mathcal{S} \to \mathcal{U}^*$, where $\mathcal{U}^*$ is a language constructed from alphabet $\mathcal{U}$ (e.g. Unicode).

At any learning iteration I (i.e. gradient update step), a conventional DDRN IQL implementation learns by gathering Q-value outputs from the predictive and target networks, parameterised by θ_I and θ'_I respectively, ran a minibatch of past experiences in the form of past episodes (containing action-observation histories and state information from the replay buffer). The outputs of the networks are subsequently used to calculate the loss needed to update our networks. The maximisation part of DMQL's Dictionary Mechanism takes place after Q-values have been calculated by the networks. Just like with IQL, we calculate our current target $Y_{t,\mathsf{current}} = r_t + \gamma Q(s_{t+1}, \mathsf{arg\,max}_{\boldsymbol{a}}\, Q(s_{t+1}, \boldsymbol{a}; \theta_I); \theta'_I)$ for each agent, episode, and time step. When this target is calculated, however, we do not immediately use it for the loss computation. Instead, current target values are used to update the dictionaries of the Dictionary Mechanism, using the update rule represented in Equation 3.2.

$$\mathcal{D}_I^i(\phi(s_t), a_t^i) = \begin{cases} Y_{t,\text{current}} & \text{if } Y_{t,\text{current}} > \mathcal{D}_I^i(\phi(s_t), a_t^i) \\ \mathcal{D}_I^i(\phi(s_t), a_t^i) & \text{otherwise} \end{cases} \tag{3.2}$$

Where subscripts related to minibatch episode indices are omitted for brevity.

Using the update rule represented in Equation 3.2 guarantees that after every time step of every minibatch episode, the dictionary contains values which either equal the maximum value encountered for every respective hash-action pair encountered during target calculations for the entire batch, or equal some higher value retained from a previous learning iteration. This means we can subsequently gather DMQL's target values from the dictionary ($Y_t = \mathcal{D}_I^i(\phi(s_t), a_t^i)$), where minibatch episode-related subscripts have once again been omitted), and can calculate a loss (Equation 2.7) using these gathered *maximum* estimated target values.

As mentioned before, this strategy, which maximises over past and current values, is likely to experience divergent behaviours due to overestimations being favoured by this maximisation step. To combat this, we augment our strategy. We speculate that the main problems caused by the storage of overestimations are caused by the fact that following learning iterations bootstrap from these overestimations, causing a feedback loop. Whilst various techniques exists to reduce the impact of (summed) errors (elaborated upon in Section 3.5.3), the usage of a direct maximum over estimations is inherently problematic. Hence, we introduce one simple augmentation to our strategy: instead of storing a *maximum of past estimated target values*, we opt to store a *pseudo-maximum value* derived from maximums of past and current estimated target values.

We introduce an additional operation in our dictionary mechanism, triggered between learning iterations. This operation is represented in Equation 3.3

$$\mathcal{D}_{I+1}^i(u,a).q = \begin{cases} d_t \mathcal{D}_I^i(u,a).q - (1-d_t)x & \text{if hash-action pair } (u,a) \text{ occurred in } I \\ \mathcal{D}_I^i(u,a).q & \text{otherwise} \end{cases} \tag{3.3}$$

Where $u \in \mathcal{U}^*$, $a \in \mathcal{A}^i$, where d_t is a *degradation factor*, and where x is some positive value which is negated to function as a value for $\mathcal{D}_I^i(u,a).q$ (the target network estimate component of the target value) to decay towards (which can be kept constant, but in this work is assumed to be the maximum target value found this training iteration).

The introduction of this operation, dubbed the *degradation mechanism*, another kind of update rule, has some important consequences and properties:

- *If we did not encounter a hash-action pair, we do not degrade its value*. This is important to enable us to remain optimistic for hash-action pairs which do not occur often.
- *If an excessively high value is stored in the dictionary, it will be degraded repeatedly until it is in-line with (maximums of) network returns*. If we presume that our networks do not consistently output similarly high overestimations for the hash-action pair, these lower estimations will not overwrite the stored value. Hence, the excessively high value is degraded every learning iteration for which it is higher than network returns; we are able to *'forget'* overestimations. (Note that this does assume networks are updated sufficiently slowly, as degradation does take some time when overestimations are great. Otherwise the networks could still successfully bootstrap off of these overestimations successfully.)
- *If a value was not excessively high, the degraded value will be 'restored'.* If the value which we degraded was not excessively high, we retain a slightly lower value - most likely still able to introduce optimism - which allows us to overwrite low values from situations in which exploration-based miscoordination occurs. If miscoordination does not occur, and our value was not excessively high, the network which produced the value should grant us a replacement value to replace our degraded one with (even though it has been updated of course).

In practice, this means that the degradation mechanism should not inhibit our ability to introduce optimism whilst allowing us to prevent the divergence caused by repeatedly bootstrapping off of overestimations stored. When the algorithm is in the initial learning stages (up to the point that estimate values

Figure 3.1: The impact of maximisation and value decay (degradation mechanism) when (nearly) converged. Dotted line $y = 10$, No maximisation step ■, maximisation without decay ■, maximisation with decay ■.

have nearly converged to the optimal ones), where overestimations are relatively benign, we are able to introduce optimism by overwriting low values from exploration-based overestimations with either not or slightly degraded pseudo-maximum values from the dictionary. Whilst this causes us to converge slightly slower (due to targets potentially being slightly lower than the theoretical maximum we could infer), it allows us to retain stability in later stages, where overestimations are more likely to be impactful. Here, the overestimations should be corrected by the application of our degradation mechanism between learning iterations.

Figure 3.1 exemplifies the impact of our maximisation strategy and the degradation mechanism. It is representative of a sequence of encounters of a specific configuration for which the target estimate has already converged to the value it should converge to in order to establish an optimal policy. Without maximisation, various over- and underestimations cause our actual returns to fluctuate a few percent. With our maximisation strategy without maximum value decay, we see that underestimations no longer influence our results. Overestimations, however, are no longer balanced with underestimations, and bootstrapping of off these overestimations causes results to continuously increase. This can lead to divergent behaviour in our agents. Using our maximisation strategy in combination with decaying maximum values (like in our degradation mechanism), we are able to incorporate slightly discounted past maximums in the initial learning stages in which we converge to 'correct' (near-optimal) values, whilst preventing feedback and therefore instabilities and subsequent divergent behaviours when (nearly) converged.

3.3. Applying DMQL

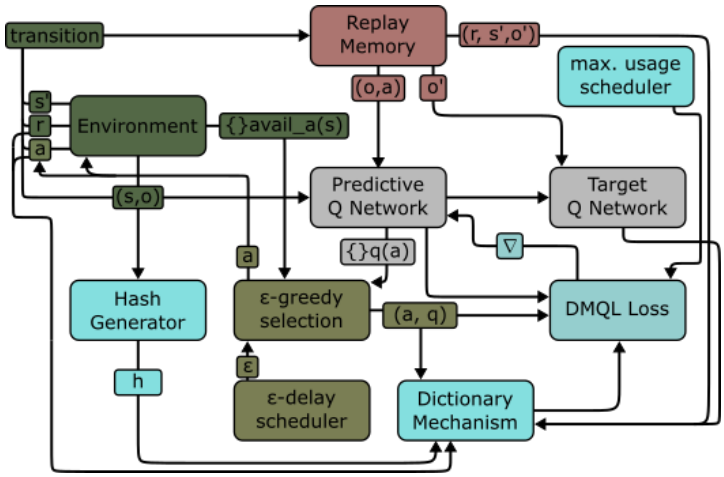

Figure 3.2: DMQL Overview

In Figure 3.2, one can observe an overview of interactions between DMQL components and the environment. The elements related to the environment, replay memory, action selection, and the Q-Networks, are exactly the same as a possible DDRQN IQL implementation. The contributions from this work can be found in the target value computation which is now routed through the dictionary mechanism, and the generation of hashes. One should note that there are multiple possible locations to implement the hash generator: one could store state- or observation-hash mappings in the dictionary mechanism (which *can* be quite heavy on space resources but *can* also be faster), or one could calculate hashes

on the fly from information retrieved from the replay memory (which allows us to omit storing mappings, and for which speed depends on the efficiency of the hashing algorithm). In our implementation we have opted for the latter.

In Algorithm 1, one can see that adding DMQL functionalities to an algorithm such as DDRQN IQL can be quite straightforward.

Algorithm 1 DDRQN IQL / DMQL training (additions in blue, omitting details (e.g. masking))

```
dict_mech ← init_dictionary_mechanism(config)

procedure train(batch, predictive_net, target_net)
    predictive_net_returns, max_actions ← predictive_net(batch)
    target_net_returns ← target_net(batch)
    target_net_returns ← select_action_values(target_net_returns, max_actions)
    targets ← calculate_targets(batch, target_returns, max_actions)
    loss ← MSE(predictive_net_returns, targets.detach())
    optimise(loss)
    update_target_net_if_applicable()
end procedure

procedure calculate_targets(batch, target_net_returns, actions)
    rewards ← batch.rewards
    states ← batch.states
    targets ← rewards + γ ∗ target_net_returns
    # calculate hashes for states, update dictionary entries accordingly
    # (Equation 3.2, optionally 3.3 before that)
    agent_hashes ← dict_mech.update(states, rewards, target_net_returns, actions)
    # fetch targets from dictionary
    targets ← dict_mech.apply_maximisation(agent_hashes, actions)
    return targets
end procedure
```

3.4. Developing a Hashing Strategy

As mentioned in the sections above, DMQL uses a hash function in order to generate a hash, which represents the state, in order to retrieve pseudo-maximum target value estimate values. This function takes state configuration information (or a subset thereof) as an input, and generates a hash which represents the state or some representation thereof.

One of the most important factors to keep in mind when developing a hashing strategy, is that these hashes will be used to cluster similar states together. In other words, we have to think of what *similarity* means for us, and how we can best represent it in our hashes. Many ways to incorporate similarity in our hashing strategy can be conceived, but for illustrative purposes, we shall, through this section, work our way towards ways which use our insights into the problem used to evaluate the algorithm, and expose problems which can occur when using approximations to cluster similar states together this way.

3.4.1. Unique Hash

The most straightforward hash function is one which generates a unique hash for all possible configurations; take all state information available, and directly map each unique entry to a different hash.

In Figure 3.3, we illustrate how this strategy works. On the left, we see two configurations the state can be in for a three-agent, one-prey variant of the problem. In the middle, we see what representation of the state is used to generate unique hashes for. In this case, the entirety of the state configuration is taken into account. On the right, we see a representation of the hash generated, which will be used as a partial key of our hash-action pseudo-maximum target estimate value dictionary. As can be seen,

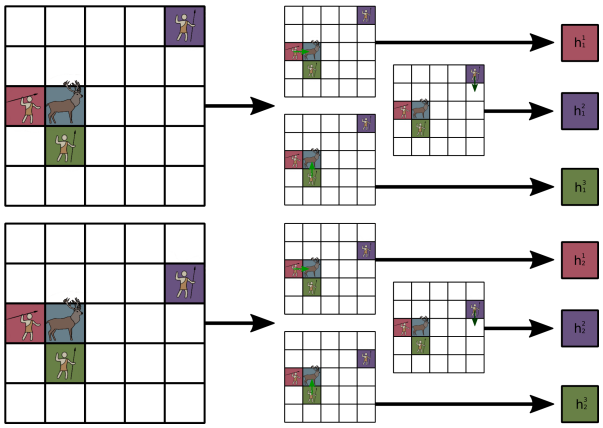

Figure 3.3: Example: Unique hash for all configurations. Left: two state configurations. Centre: state representation used to generate unique hashes for. Right: generated hash for respective agents.

agents (represented by respective colours) will generate a different hash for the two state configurations. If h_z^i is a hash for agent i, where z is used to denote 'some hash', we have $h_1^1 \neq h_2^1$, $h_1^2 \neq h_2^2$, and $h_1^3 \neq h_2^3$. There are differences between the state configurations, so the hashes will also be different.

For small non-complex environments this should suffice in a similar manner as the tabular projection from Distributed Q-Learning [Lauer & Riedmiller, 2000] would. This restriction hints at the most significant drawback of this hashing strategy, however: it is not scalable, as highlighted in 'Hash Problem 1' below.

Hash Problem 1: Similar configurations not yielding the same hash

Some configurations, from the perspective of an agent, are quite similar. One could define a strict notion of similarity between two configurations by the optimal target values for the agent being equal, which implies that the policy for handling this subset of configurations is the same as well. This same implication can, however, be reached through a weaker notion of similarity as well, in which optimal target values are not 'too different', or wherein ratios between Q-values should at least be approximately the same.
If two configurations are similar, exploring one of these configurations ideally should provide information the algorithm can use to handle similar configurations. If similar configurations do not yield the same hash, similarities between configurations cannot be exploited. DMQL defaults to IQL when a hash is discovered for the first time, thus, if for a given configuration in which relative overgeneralisation occurs the reward 'outcome' is not encountered at least once *for its hash*, DMQL's maximisation step does not come to fruition. As exploration is often reduced with the number of timesteps passed, exploitation of similarity could yield exceptional benefits in the learning process. It can be of increased benefit in environments with relative overgeneralisation as it can help prevent a stable non-cooperative policy from forming in early stages, which is exceedingly hard to overcome as a result of negative Q-values for actions which move the agent closer to rewardable state configurations, and the often significantly lowered exploration rate at later stages of the learning process.
An approach compatible with our strict notion of similarity could be to exploit *symmetries*. As many environments are not directional, policies and optimal Q-values can be symmetric. In the Hunter-Prey environment, for example, rotating the grid world by $90°$ should result in optimal Q-values for this new configuration being equal to ones from the non-rotated configuration, only with the movement actions being rotated $90°$ as well (e.g. up $\rightarrow$ right). For our weaker notion, we can use additional insights.

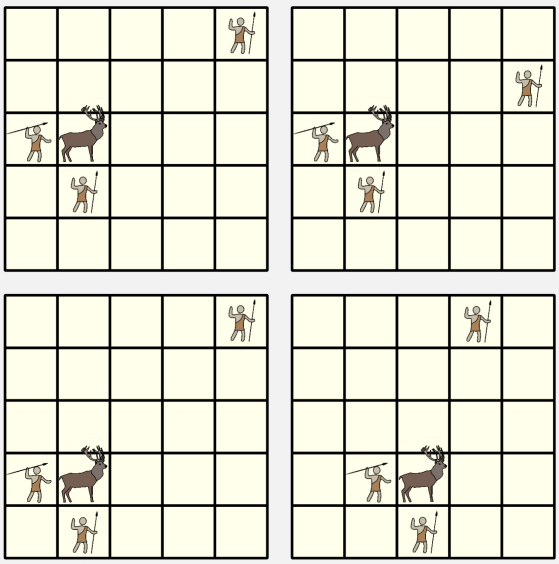

Figure 3.4: Similar configuration example

Figure 3.4 depicts various state configurations for which the value of taking actions is roughly the same for (two of) the agents. From the perspective of these agents, the configurations should therefore be quite similar. Including all the information in these configurations causes us to expect to need multiple times as much exploration in order to successfully employ our maximisation strategy. For the two upper configurations, disregarding the information related to the rightmost agent in this process for the two leftmost agents allows us to combine information gathered through learning from experiences for both configurations, likely with few drawbacks.

Given that providing each and every unique state configuration a unique hash does not allow us to map

similar configurations to the same hash, we can try to augment our hash function. As discussed, we could try to exploit symmetries in order to adhere to our strict notion of similarity. For grid world environments such as Predator-Prey, one could find not only rotational, but also line symmetries. Using these symmetries can divide the number of unique hashes we have to account for by a constant factor. In addition, depending on the environment, one could remove unnecessary features; in the hunter-prey environment for example, both hunters and prey do not need to be identified individually, but can be identified solely by what type of actor they are. This once again reduces the number of unique hashes we can encounter by a constant factor depending on the number of actors of the same types.

Whilst this already improves scalability somewhat, we would ideally like to not only divide the number of unique hashes the algorithm can encounter by a constant factor, but to completely decouple the size of the state from the number of hashes we have to separately explore for. Approaches adhering to our strict notion of similarity additionally would be quite useless for continuous state spaces where it is exceedingly unlikely to visit a state configuration or its symmetries multiple times.

Our weaker notion of similarity could provide us with a solution which takes inspiration from the kind of problem we try to find policies for. These problems often involve partial observability, i.e. inability to observe parts of the environments. This forces one to base their decision-making process on available information which is often relatively local to the agent (in addition to historical information, of which the usefulness can degrade with time, depending on the environment).
If an algorithm utilising individual learners is able to solve problems through usage of this subset of the information available, the information provided to each agent should be sufficient to make good decisions. As discussed, the set of information available to the agent during execution (when using histories) includes information from preceding parts of the trajectory. An approximation utilising subsets of information from the state configuration, however, is possible, as we only need to determine similarity during training.

3.4.2. SimHash

As discussed in Section 2.16, SimHash is a locally sensitive hash which measures similarity by angular distance through the use of sign random projections [Tang et al., 2017]. Whilst it is therefore likely to be able to determine *'similarity'* between input vectors which are 'close' to each other, thereby reducing the number of hashes we have to take into account, the notion of similarity used by SimHash is different from the ones we introduced. We can interpret the notion of similarity used by SimHash using the Hunter-Prey environment described in Section 1. Here, the measure of similarity (between two inputs, i.e. state configurations) provided by SimHash should decrease with the cumulative actor steps they are apart from each other.

Whilst this might be a great measure for other problems, it likely is not ideal for solving the problem of the cumulatively rewarded Hunter-Prey environment and environments like it (without first encoding state configurations through other means, such as auto-encoders): Configurations which are close to each other, step-wise, do not necessarily share optimal policy target values or even ratios thereof: whilst Q-values likely are related, rewards for taking actions do not have to be. In other words, whilst the *value* of nearby configurations is likely quite similar, the *advantages* of actions therefrom are likely quite different. This *advantage* is, however, the more important value to take info account when making decisions.
For the Hunter-Prey environment, Q-values and especially their ratios are primarily determined by the configuration of the state space surrounding an agent; this determines whether or not a hunter can attempt to catch a prey and whether or not this is a good idea, or which direction a hunter should move in to get closer to a configuration from which it is wise to attempt to catch. Mapping configurations together with the different notion of similarity introduced by SimHash yields a situation in which we map configurations together which are dissimilar according to our own notions of similarity, of which consequences are discussed in 'Hash Problem 2'.

Hash Problem 2: Dissimilar configurations yielding the same hash

Dissimilar configurations yielding the same hash could prohibit learning by overwriting values for state configurations by values from more dominant, yet dissimilar, high-Q-value state configurations. Within the Hunter-Prey environment, the configuration of space near the agent is most important in determining what action is the optimal one (given some trajectory, or in general), hence, if two dissimilar local configurations of space map to the same hash, too high targets will be retrieved, which will lead to wrong Q-values. These Q-values will in turn be propagated further. Whether or not two state configurations are 'similar enough' (i.e. map to optimal values which are close enough) to be represented by the same hash is not trivial to determine, and will therefore have to be explored.

3.4.3. Contextual Hashes

As we expect that simply applying SimHash to full representations of our state configurations will only be successful in reducing the number of unique hashes the algorithm can encounter, not in actually clustering states (or therefore histories) together which are similar from the perspective of our notions, we need to develop hashing strategies based on these notions directly. As mentioned in Section 3.4.1, one option is to look to our weaker notion of similarity, and focus our development on the part of the state configuration which is relatively local to the agents, as these parts of the configurations most heavily influence the outcome of our (also local) interactions with the environment. The main aim would therefore be to reduce the amount of information from our state representation (used to calculate hashes for) to a set of information which at least functions as a predictor for target (ideally advantage) ratios. We once again start by using single-step state configurations as a practical approximation of the information present from following a trajectory to a given time step.

As described, our weak notion of similarity is based on information which can be taken from an agent-centric representation of the state configuration. One could interpret this as representing the state as some 'context' the agent is in. Hence, we shall dub these kind of hashes '*contextual hashes*'.

Given that context are unique to agents for any given state configuration, their respective state representations should also be. The first step for contextual hashing strategies then, is to convert information from our state configuration to information relative to the respective agents. This first step can be seen as a way to map all spatial translations of the same (used part of the) state configuration to the same hash. Figure 3.5 helps to exemplify why this makes sense: for every spatial translation of the same configuration (excluding empty space in this case), the actions required of each agent to act optimally are exactly the same.

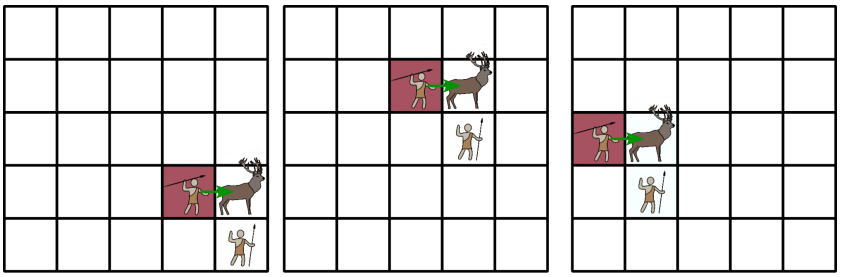

Figure 3.5: Example: Translations of configurations often do not change the optimal actions to take. For all three depicted state configurations, the optimal action to take for the red agent is 'catch'.

An easy follow-up step for any contextual hashing strategy is to remove information which can straightforwardly be reasoned to be irrelevant. As described in Section 3.4.1, homogeneity within groups of actors we have to represent is something to consider in a way quite similar to how one would do for the seemingly unrelated *observation function* (if one is aiming to improve convergence speed by augmenting this observation function, that is). Assuming homogeneity between actor groups (hunters/prey), Figure 3.6 exemplifies why using actor groups instead of actors themselves in our state representa-

tions makes sense: if actors truly are homogeneous with the rest of their group (and assuming no large differences between agent networks, as can be achieved using parameter sharing), swapping the places of any two actors of the same type, other than the agent for which we are constructing a state representation, would result in a context for which the optimal actions to take are once again exactly the same.

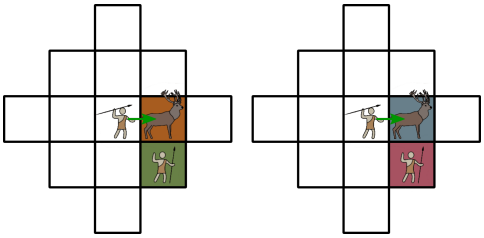

Figure 3.6: Example: If other actors are homogeneous, using 'types' or 'groups' makes more sense. Irrespective of the identities of the other actors, the agent at the centre of these contexts has the same optimal action 'catch'.

If we assume or assert that our *agents* are homogeneous, we can take an additional performance-improving design choice: if we use the same hashing method (and the same seed if applicable) for all agents, we can use a *shared* hash-action pseudo-maximum target estimate value dictionary $\mathcal{D}_I$ for all agents as well. This allows us to introduce optimism in agent-hash-action pairs if *any* agent has recently encountered an outcome without miscoordination in a similar *context*. One could interpret this as *'if any agent has performed this action from a context similar to the current one and achieved a great outcome, we assume we could achieve that outcome as well, and that low values are a result of exploration-based miscoordinations'*.

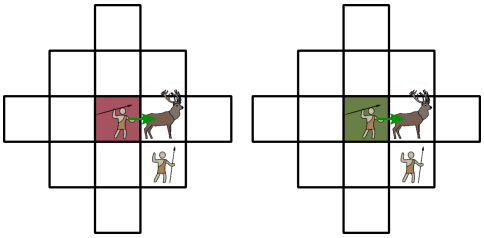

Figure 3.7: Example: If agents are homogeneous, using 'types' or 'groups' makes more sense. Irrespective of whether the green or the red agent is in the context depicted, they have the same optimal action 'catch'.

Depending on the level of knowledge we have on our environment, we can go further (*possibly better*) than mapping all translations of state configurations and all permutations within actor groups to the same hash: Whilst the introduction of contextual hashes as defined above already reduces the number of possible unique hashes the algorithm can encounter by a factor related to the environment size, we are still including information which potentially is quite irrelevant, like explained in 'Hash Problem 1'. Now that we are able to construct state representations which include information in a form relative to the agents, we are able to exclude information which we deem irrelevant to the agent based on distance alone. The previous definition can be seen as us contructing a 'view' of the environment which completely encompassed it. If we, however, start to decrease the radius of this 'view', we can limit our representations of the state to include only the local information most relevant to immediate decision-making. We even choose views which include information which is a subset of the information contained in observations, making the approach compatible with decentralised training (if parameter sharing and dictionary sharing are also disabled).

Figure 3.8 illustrates how limiting the range of included information helps us map similar configurations together, and illustrates the dictionary sharing technique: as can be seen, local configurations for which the same optimal actions should be taken are clustered together now using the former, and this is extended to other agent permutations using the shared dictionary. This means that using this approach,

$h_1^3 = h_1 \neq h_2^1 = h_2 = h_2^2 \neq h_3^1 = h_3 = h_3^2$.

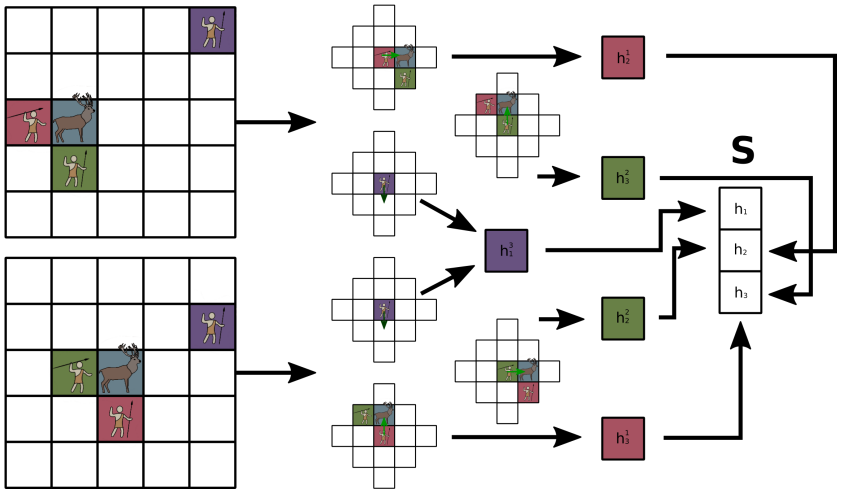

Figure 3.8: Example: Contextual Hashes. Left: two different state configurations. Center left: state representations used to generate a hash. Center right: hashes generated for respective agents. Right: when using a shared dictionary, we can use the same hash as before to retrieve a pseudo-maximum target estimate calculated using the collective's experiences.

The choice of what information to include is a problem of balancing Hash Problem 1 and Hash Problem 2: utilising too much unnecessary information can lead us to a version of Hash Problem 1, and vice versa. We explore these potential problems in 'Hash Problem 1a' through 'Hash Problem 2b'.

Hash Problem 1a: Too much information in context.

Like the main Problem 1, this problem reduces efficiency. In these cases where hashes do not include enough information to uniquely identify state configurations (i.e. include subsets of the full information), we do have one major windfall: DMQL only needs to be able to gather enough information for contexts in which relative overgeneralisation can occur. Other context could be handled by conventional IQL without us losing the ability to reach collaborative policies, hence it would not hinder us if DMQL defaults to it). If DMQL removes the impact of exploration-based miscoordinations, IQL should have no problem optimally manoeuvring agents to configurations in which relative overgeneralisation was previously a problem, as it would encounter an apparent lack of relative overgeneralisation itself. This is something we can effectively exploit for contextual hashes (see Section 3.5.1).

The problem of efficiently clustering state configurations in which relative overgeneralisation occurs does, however, remain. In addition, we might not have sufficient insights into the workings of the environment to apply the technique presented in Section 3.5.1 to its greatest potential. Hence, in many cases, how much information of lesser relevancy we can prevent from influencing the output of the hash function still directly determines the chances DMQL has to prevent relative overgeneralisation and to establish a cooperative policy.

If we once again observe Figure 3.4 from Hash Problem 1, we see that using contexts centered around agents can help to apply similarity between translations of subsets of the state configuration. This does, however, require us to balance the range of information to include. Include too little, and Hash Problem 2 occurs due to the lack of local context, include too much, and the exploration needed increases substantially. For our purposes, it would be ideal to include just enough information to identify situations in which relative overgeneralisation could occur (for now ignoring the impact this has on other situations or contexts in which little information is available locally).

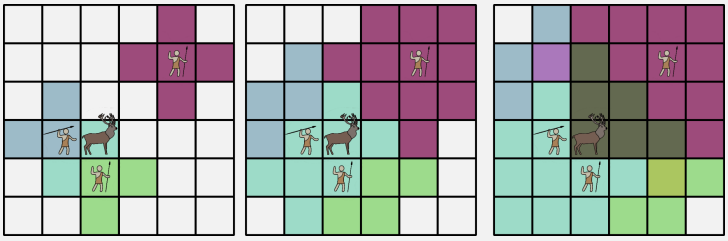

Figure 3.9: Influence of range of inclusion. Left: 1-step. Centre: 2-step. Right: 3-step.

In Figure 3.9, the dilemma of information selection is exemplified. The leftmost choice (1 step) is insufficient as agents cannot identify differences between situations in which relative overgeneralisation occurs and ones in which it does not, due to the fact that the agents cannot see each other. The second choice is suitable for the identification of situations with relative overgeneralisation, yet there may be a lack of information in hashes generated from some local contexts (this problem scales with environment size and in some sense with actor density, see 'Hash Problem 2a'). The last choice depicted shows all agents including some information in their hashes. The information for the rightmost agent still might not be sufficient. With this size, the number of unique hashes which can be generated is substantially higher, which does not help in situations with relative overgeneralisation in this case (number of possible hashes are 39, 883, and 6949 respectively, of which 0, 132, and 276 are uniquely for situations with relative overgeneralisation respectively).

Hash Problem 2a: Not all contexts are equally informative.

When hashes only take into account the subset of state configuration information in the vicinity of the agent, it is possible to encounter local subsets of state configuration information of varying informativeness; in the case of the Hunter-Prey environment it is possible to encounter subsets containing information on various other actors, or none at all. When contexts with little information are encountered, using their information to map to a hash will yield collisions which might not be ideal. One example would be an information subset containing no information on other actors: from this information it is not possible to uniquely identify suitable actions to take, just as it is impossible to do so from an similarly uninformative observation alone in partially observable problems. As the proposed maximisation strategy does not include information on histories, it might therefore be useful to require a certain amount of information to be included in a hash before the maximisation strategy is used; this way, Q-values estimated by the RNN - which does use histories - can be used. This will allow the algorithm to include maximised values from information-rich local configurations, and the information provided by the RNN which is relatively rich in situations where local configurations do not contain enough information. If relatively uninformative hashes are not accounted for, the subset of local contexts represented by such a hash may retrieve seemingly arbitrary maximum future values as seen from the perspective of being in any specific local contexts. As these hashes are able to represent less information than the overwritten network could return, which takes the entire trajectory up to timestep t into account, information could be wasted; it may not be reasonable to assume the way in which we ended up in this context is irrelevant for maximisation in these local contexts.

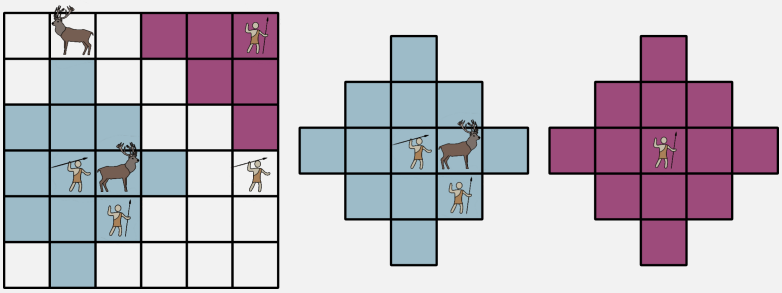

Figure 3.10: Informativeness example: contexts of varying informativeness within the same configuration.

In a given state configuration (example Figure 3.10 left), not all possible contexts necessarily are equally informative for our maximisation. For example, the subset depicted in the middle of Figure 3.10 can likely be used for maximisation due to the fact that it can inform an actor on what action to take to get rewarded (assuming optimal behaviour from other agents), whilst the subset depicted on the right of Figure 3.10 cannot inform the agent on what direction to move in to get to a higher-value configuration: the value of any action here is dependent on information outside the subset, hence neither of our notions of similarity hold here. Figure 3.11 shows that the same uninformative hash can follow from wildly different situations, even within the same state configuration. The fact that this hash cannot be used to inform what actions to take makes it uninformative. Maximising here can lead to behaviour which can prevent attaining rewards.

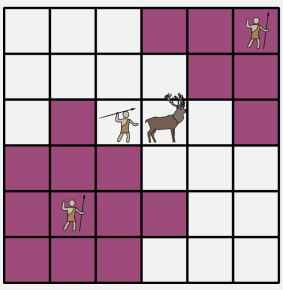

Figure 3.11: Informativeness example: Equally uninformative contexts within the same configuration, for which the optimal actions are quite different.

Hash Problem 2b: Reward causality is not included in the hash

When relatively local contexts are considered once again, our definition of cumulative rewards might be a hindrance: when a cumulative reward is encountered due to events which follow from information not present in a hash, the dictionary entries for some hash-action pairs can be filled with seemingly arbitrary rewards. Reward causality might not be related to hashes. If this becomes a significant problem, one can attempt to decay stored rewards, but this will not fully rectify the problem due to the retained value. Full decay, however, would by contrast cause the opposite problem by also removing information from hashes which represent the configurations with relative overgeneralisation, thereby preventing the maximisation strategy from being used.

3.4.4. Alternative Hashing Strategies

It should be noted that the selection of information from the state configuration, as well as its representation, can be altered greatly. The methodology described above wherein we take a context or 'view' centered around agents is not necessarily ideal for every given usecase. The choice for the representation used in this work was made due to it being intuitive for grid worlds, as well as due to it being quite capable of illustrating problems which plague various other options, in addition to the fact that this strategy could be applied in truly 'decentralised' settings (needed information can be derived from observations instead of the true complete state configuration). For the Hunter-Prey environment alone, many alternatives exist which address various Hash Problems. One example of a family of representations is the set which uses distance indications (representing the state by 'distances' of a subset of the elements within the environment relative to the agent considered). Depending on the implementation, this could prevent Hash Problem 2a (if appropriate actors and 'measures of distance' are selected to be represented, otherwise incurring Hash Problem 1a).
SimHash, whist not expected to be particularly effective when used without its optional pre-processing steps, could also be made quite effective; there is a precedent of successful applications when combined with auto-encoders [Palmer et al., 2017; Tang et al., 2017].

3.5. Additional Techniques

Whilst the aforementioned techniques form the basis for the functionality of various DMQL variants, we employ a few more techniques as well. The first we shall highlight is mostly unique to DMQL, whereas the second takes inspiration from the scheduling approaches used by the works described in Section 2. The last set of techniques we discuss in this section are techniques focused on limiting the impact of errors upon an algorithm's learning process.

3.5.1. Selective Utilisation

As described in Section 3.4.3, it is possible for various problems to occur when using contextual hashes. Problem 1a occurs when too much needless information is used to determine a hash, reducing efficiency. By contrast, removing information from the hash, especially by restricting the range from the agent up to which information included, can lead to problems 2a and 2b. Selectively using the maximisation strategy of DMQL instead of using it all the time could combat the latter set of problems.

Problem 2a stems from the existence of contexts which hold insufficient information inform decision-making. These will lead DMQL to disregard useful information, as its history holds disproportionally more value for decision making in such contexts; if the contexts contain insufficient information to identify differences to inform decision-making, the information gathered in DMQL's dictionaries would also not be applicable in a way which could inform decision-making successfully. One example of this would be a context which contains no information on other actors: given this context, we would not know what direction to move in, whereas histories *may* hold information which could provide useful insights.

Problem 2b stems from the fact that the causality of rewards is often not observed (even when the rewards themselves are 'observed'), and that rewards therefore seemingly arbitrarily can start to be represented in dictionary entries where they are not quite relevant, thereby reducing efficacy of the decision making process in affiliated contexts.

Both problems can be combatted by introducing a function to allow for the selective usage of DMQL's maximisation. Depending on ones knowledge on the workings of the environment, one can specify where DMQL's functionality is applied, and therefore can target situations with relative overgeneralisation more specifically. One example would be to simply exclude usage of maximisation for information subsets in which no other actors are represented, which would consequently exclude the most uninformative contexts. More specifically, one could specify that all actors/features needed for rewards are present (one other hunter, and one prey, for the Hunter-Prey environment). Most specifically, and using the most information on our problem, we can specifically target information subsets which represent situations in which relative overgeneralisation can occur (at least one adjacent prey, which has at least one other agent adjacent to it, in case of the Hunter-Prey environment).

The more specific our selection function, the less Problem 2a should occur. Whilst Problem 2b still remains a problem even when usage is restricted, its influence will be limited to selected states only, potentially decreasing its influence greatly. In case of our implementation, excluded contexts will be handled by IQL, which should be sufficiently capable of learning how to navigate towards state configurations which can result in cooperative rewards through utilisation of the maximisation strategy. Figure 3.13 depicts various contexts, for which various selection functions, or 'selectors', would cause DMQL to either include or exclude the context for its maximisation step.

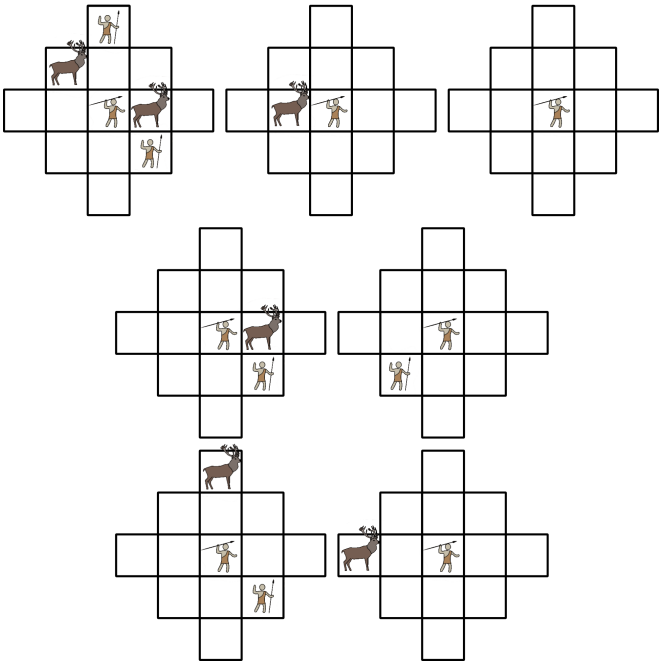

Figure 3.12: Without a selection function, all these contexts would be maximised over. A 'catchable' selector would, for example, only include the top left and centre left contexts. 'Adjacent stag', in turn, would include the top left and centre contexts, as well as the centre left context. No reasonable selector would include the top left context.

3.5.2. Usage Decay

As DMQL uses a lot of approximations, it might not be able to distinguish all nuances available through information present in histories. Solely using DMQL's maximisation strategy during the entirety of the learning process could lead to suboptimal policies, due to our agents utilising only approximations/subsets of the information available to them. Once cooperative actions have been established as being beneficial, and a cooperative policy has been established, one could decay the usage of our maximisation strategy in order to allow values to converge to realistic values based on the entirety of the trajectories leading to rewards (or punishments).

Alternatively, one can decay usage in a manner similar to what can be seen in Deep LL methodologies:

given that we already have the capability to calculate a hash given some state-configuration and agent, we can decay the usage depending on the number of uses by keeping a usage dictionary and using a schedule similar to the one discussed for ϵ (Section 2.3) for each hash. Whilst this approach has the benefit of allowing us to use maximisation longer for any 'exceedingly rare' configurations while removing the approximation errors induced for 'more common' configurations, it is harder to configure.

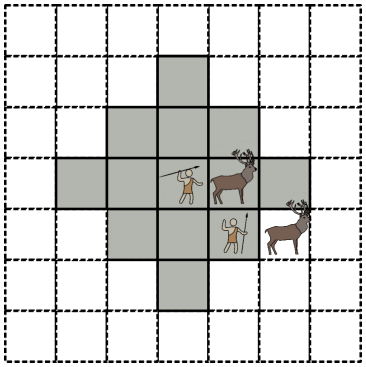

Figure 3.13: In situations like this, the expected reward for attempting to catch is $\frac{1}{2}r - \frac{1}{2} * (2p + r)$: the environment randomly selects *which* adjacent prey an agent will attempt to catch *first*. If both agents attempt to catch, they will, on average, be rewarded with that lower expected value, not r, which DMQL would suggest. In this case, it would be better to use information on where the other prey is to determine what actions to take.

3.5.3. Limiting the Impact of Errors

In addition to the degradation strategy discussed in Section 3.2.2, there are other techniques which can limit the impact of errors introduced by overestimations.

The first technique we will employ is to simply lower the learning rate γ. This will decrease summed errors over when future values are discounted. This can, however, lead to a strong bias towards immediate rewards with respect to rewards further in the future if lowered too much.

The second technique employs soft target network updates. If target networks are updated slowly enough, the magnitude of overestimations following from mistakes in the predictive network may be limited for the target network. Hard updates will cause the target network to take over the parameters of the predictive network at once, along with all overestimations it may cause for a given set of inputs. Slow soft updates can prevent the weights and biases from being copied all at once, thereby only letting the target network inherit overestimation-causing ones if they persist in the predictive network for a long time.

4

Results

In this section, we evaluate various DMQL configurations in order to explore the limits of DMQL. Once we have explored these limits, we compare DMQL to various state-of-the-art algorithms, taking into account varying observation capabilities for agents.
All plots in this section are constructed using the methodology described in Appendix A. For clarity, where relevant, each plot contains a dotted line to denote the x-axis and a striped line to depict optimal values (wherever the optimal value is not zero). Experimental configurations are defined alongside plots for clarity and replicability. Unless stated otherwise, average test episode rewards are depicted through the use of a continuous line.

4.1. Porting Distributed Q-Learning to Deep Q-Learning

As described in Section 1, Distributed Q-Learning [Lauer & Riedmiller, 2000] has been shown to overcome relative overgeneralisation in tabular Q-Learning. To be able to exploit a similar optimistic assumption for Deep Q-Learning, it is crucial to validate that our extension from tabular to DQN-based learning algorithms can overcome relative overgeneralisation in an environment which realistically should be solvable by tabular learners in the same timeframe.
The tabular assumption resembles a projection from a greater state-(action given other agent actions) table to the smaller state-action table, which yields the highest value given other agent actions. This differs from the optimistic assumption in this work, which uses a pseudo-maximum target estimate value. Thus far, this work has assumed that the usage of these values would be a valid analogue to directly obtained Q-values from tabular algorithms. This experiment aims to verify the assumption.

Experiment: Extending Distributed Q-Learning to Deep Q-Learning

Evaluated Algorithms	IQL, DMQL
Environment	
Observation	Observing Full State
Punishment	$100\%r$ with $r = 10$
World Size	3×3
Agents / Prey	2 / 1
Shared	
γ	0.99
ϵ	$1 \rightarrow 0.05$ over $(10, 300)k$T
DMQL	
Hashing strategy	Full State Information
Degradation factor	0.9

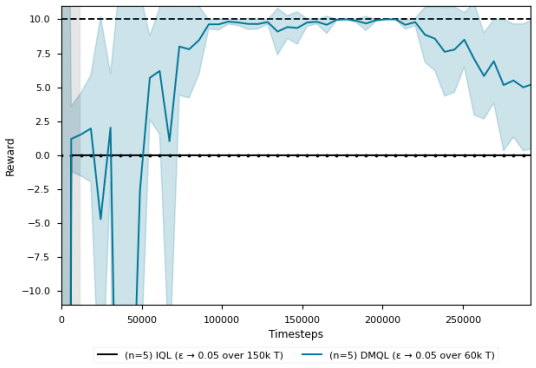

Figure 4.1: Porting experiment exploratory results.

Figure 4.1 shows the average performance for both IQL and DMQL, using the parameters from within the specified ranges which yielded the best results on a per-algorithm basis. The results of this exploratory experiment show that this most straightforward implementation of DMQL is able to learn in heavily-punishing environments despite only using independent learners, whereas IQL is unable to do so even when granted significantly more time to explore. This observation validates the assumption that maximums over usage-discounted estimated target values can be used to form optimistic assumptions in a way similar to the projections used in Distributed Q-Learning [Lauer & Riedmiller, 2000].
As shown in Figure 4.1, near-perfect returns are achieved soon after t_ϵ has passed. Rewards do, however, fluctuate after that, indicating slight instabilities in the learning process, specifically when converged.

4.2. Stability

As discussed in 3.2.2, one of the problems which can occur with DMQL is that it could be more affected by overestimations passed to the target network than other techniques, as target value estimates are maximised over. DMQL should be able to successfully leverage various stabilisation techniques from literature, as well as one specific to itself, in order to account for the possible loss of stability due to its maximisation step. As discussed in Sections 3.2.2 and 3.5.3, some of the techniques we can use are soft target network updates, lowered discount factors, and lowered dictionary degradation factors (usage-based discounts to dictionary entries), each with their own benefits and downsides. The influence of these techniques is evaluated in the following experiment.

Experiment: Stability

Evaluated Algorithms	IQL, DMQL
Environment	
Observation	Observing Full State
Punishment	$\{0, 100\}\%r$ with $r = 10$
World Size	3×3
Agents / Prey	2 / 1
Shared	
γ	$\{1, 0.99, 0.95\}$
ϵ	$1 \rightarrow 0.05$ over $60k$T
DMQL	
Hashing strategy	Full State Information
Degradation factor d_t	$\{1.00, 0.99, 0.90,$ $0.50, 0.00\}$
Soft update factor u_s	None or $(0.001, 0.1)$

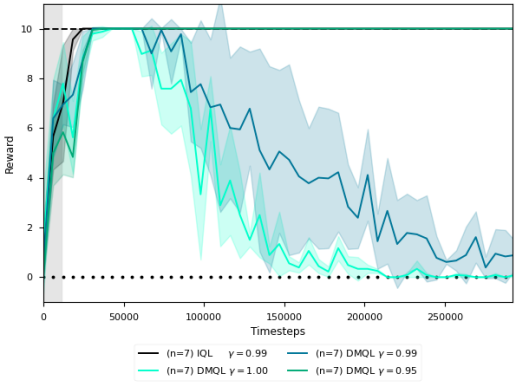

Figure 4.2: Stability experiment results: varying γ ($p = 0$, $d_t = 0.9$)

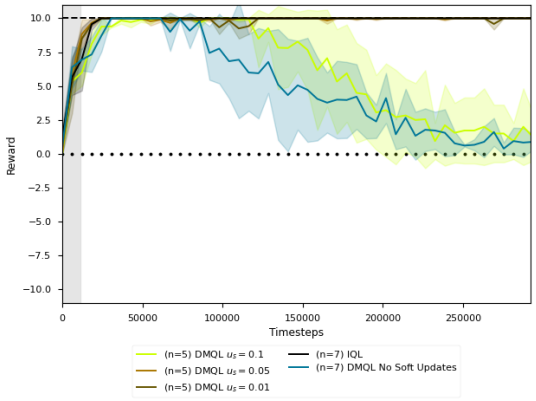

Figure 4.3: Stability experiment results: varying u_s ($p = 0$, $\gamma = 0.99$)

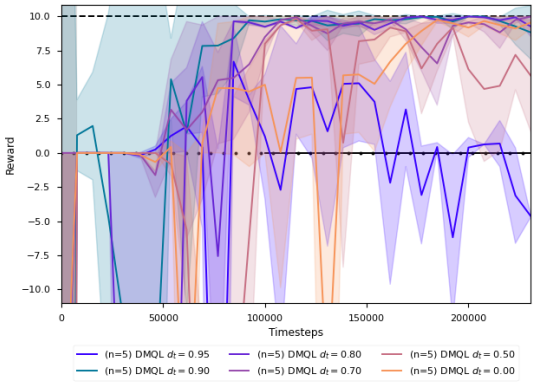

Figure 4.4: Stability experiment results: varying d_t ($p = -10$, $\gamma = 0.99$)

In Figures 4.2 through 4.4, averages over runs using parameter configuration within the specified ranges are shown. The averages depicted are from a subset of configurations which we deem representative. To evaluate varying γ and u_s we use an environment without punishment, as this allows us to establish that degrading performance is due to policy divergence (likely due to overestimations), as opposed issues due to non-stationary environments. Additionally, it is an environment which likely affects DMQL a lot due to rapid learning and great gradients.

Figure 4.2 shows the results from varying γ on a small, punishment free environment. IQL is able to reach an optimal policy quite quickly, and manages to retain performance using $\gamma = 0.99$. Using the same configuration with DMQL yields significantly worse results: degradation of performance occurs. This shows us that overestimations are able to disrupt DMQL more easily. Results from running DMQL with $\gamma = 0.95$ shows us that summed errors can be reduced sufficiently by choosing a lower discount factor. It should be noted that lowering γ influences the policies we will settle on: discounting future values inherently causes us to favour (near) immediate rewards over ones further in the future. Depending solely on using increasingly lowered γ values might be insufficient.

As discussed in Section C, using soft target network updates can be a viable way to reduce the magnitude of overestimations from the target network. Figure 4.3 shows us that this stabilisation technique works with DMQL, and that the impact is dependent on u_s. Greater values lead to more rapid replacement of target network parameters with predictive network parameters. These greater values can lead to increased learning speed, yet also diminish the reduction of the overestimations passed to the target network. In the runs depicted in Figure 4.3, we can see that using the largest value considered indeed does not significantly improve stability. Lowering the value yields significantly more stable results, although lowering it too much might substantially inhibit the functionality of the target network, thereby reducing training speed. The experiment shows us that using soft target network updates can substantially improve the stability of DMQL.

Lastly, varying d_t shows us the impact of our dictionary value degradation strategy, as discussed in Section 3.2.2. Figure 4.4 grants us various insights. The first insight is that dictionary value degradation in some form does indeed seem necessary: Any configuration with $d_t \geq 0.95$ was completely unstable, showing divergent behaviour. Configurations were found to be most stable around $d_t = 0.80$. $d_t = 0.00$, which is analogous to not retaining any value in the dictionary at all, was able to learn a near optimal policy, albeit relatively slowly, solely by maximising over hashing within each training batch. This shows us that the dictionary could be omitted altogether when using a sufficiently large batch size. This is, however, not likely to scale well to larger, more complex environments, as the maximisation step still requires sufficiently many similar configurations to maximise over.

4.3. Scaling Up the Environments

As expected in Section 3.4.1, the most straightforward implementation, using unique hashes for each state configuration, works with simple/small environments when provided a suitable learning parameter configuration. The other side of this expectation is, however, that the results we have seen in Section 4.1 are likely not attainable within a 'reasonable timeframe' for larger and more complex environments due to scalability issues. To explore whether or not these scalability issues truly exist, we perform an experiment in which we scale up the environment, thereby increasing the number of unique hashes and consequently likely the burn-in time / exploration required to gather enough information from configuration encounters to ensure the maximisation step comes to fruition.

Experiment: Scaling Up the Environment

Evaluated Algorithms	IQL, DMQL
Environment	
Observation	Observing Full State
Punishment	$100\%r$ with $r = 10$
World Size	$x \times x$ with $x \in \{4, 5, 6\}$
Agents / Prey	2 / 1
Shared	
γ	0.95
ϵ	$1 \rightarrow 0.05$ over $[10, 1000]k$T after $[0, 900]k$T delay
DMQL	
Hashing strategy	Full State Information
Degradation factor d_t	0.8
Soft update factor u_s	0.005

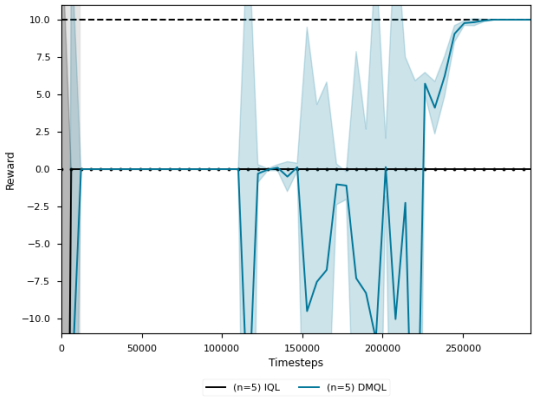

Figure 4.5: Scaling Experiment, 4×4 world. (ϵ $1 \rightarrow 0.05$ over $60k$ T after 200k T delay)

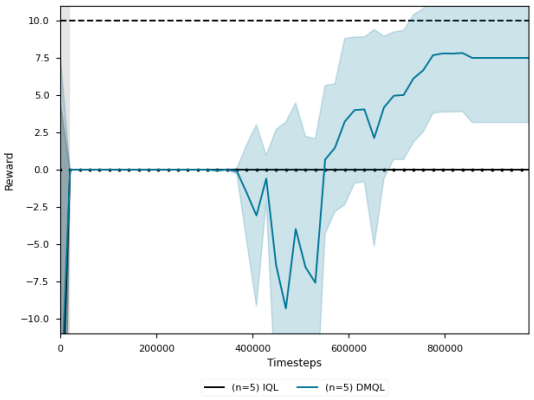

Figure 4.6: Scaling Experiment, 5×5 world. (ϵ $1 \rightarrow 0.05$ over $60k$ T after 500k T delay)

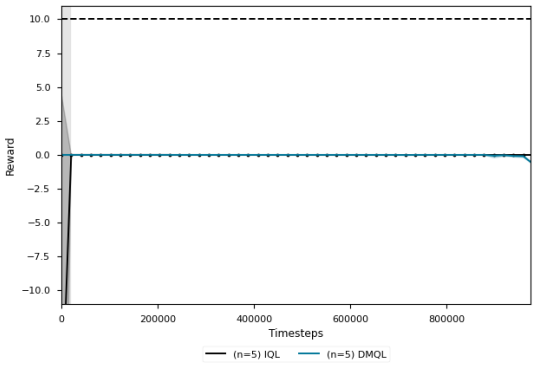

Figure 4.7: Scaling Experiment, 6×6 world. (ϵ $1 \rightarrow 0.05$ over $60k$ T after 700k T delay)

For our environments of increasing sizes, the numbers of unique hashes to consider are 1680, 6900, and 21420 respectively. As can be seen in Figures 4.5 through 4.7, which once again show the averages over the runs for the best encountered configuration for a given environment configuration, the burn-in time required before DMQL is able to learn a cooperative policy increases substantially with the size of the environment. At an environment size of 6×6 we observe no clear difference from IQL.
This validates our expectation that this most straightforward variant of DMQL does not scale well with an increasing number of unique hashes to consider.

4.4. Exploiting Similarity: SimHash (DMQL-R)

As discussed in Section 2.16, SimHash is a methodology for determining similarity between states which is used a lot in related literature. Whilst it is true that SimHash provides a measure of similarity between state configurations, its definition of similarity might not align with what is required for DMQL to function. To determine whether using SimHash alone is sufficient to improve performance compared to the most straightforward DMQL implementation, as it was for many other works, we perform an experiment in which we compare its performance to that implementation directly. We vary the parameters of SimHash, where greater values of k prevent collisions and therefore exploitation of SimHash's notion of similarity, and vice versa. The experiment is performed on an environment configuration found to work in Section 4.3.

Experiment: SimHash-based Similarity

Evaluated Algorithms	IQL, DMQL, DMQL-R
Environment	
Observation	Observing Full State
Punishment	$100\%r$ with $r = 10$
World Size	5×5
Agents / Prey	2 / 1
Shared	
γ	0.95
ϵ	$1 \rightarrow 0.05$ over $60k$T after $500k$T delay
DMQL(-R)	
Hashing strategy	SimHash on State Information
Degradation factor d_t	0.8
Soft update factor u_s	0.005
SimHash k	$[10, 200]$

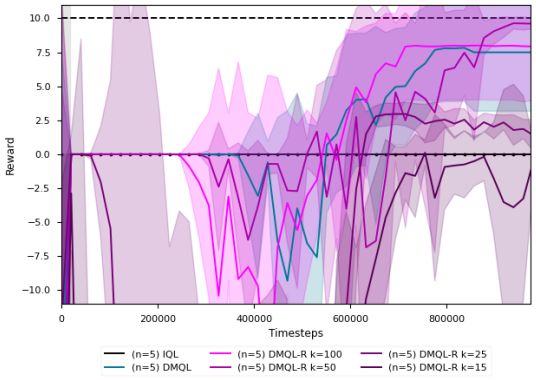

Figure 4.8: SimHash Experiment.

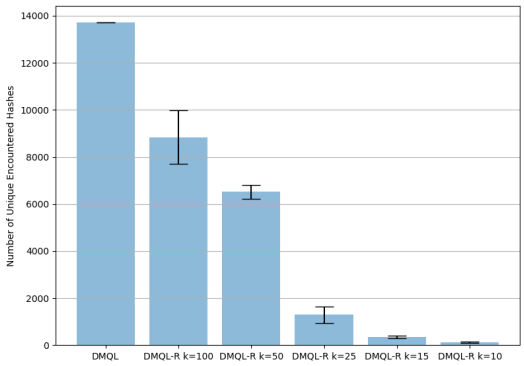

Figure 4.9: SimHash Experiment, number of unique encountered hashes.

The usage of SimHash reduces the theoretical maximum number of encounterable hashes to 2^k. In practice, however, this difference is even greater due to the fact that not all these hashes will be used: if two state configurations have sufficiently little angular distance between them they likely will be represented by the same hash.

As can be seen in Figure 4.9, the number of encountered hashes was indeed reduced significantly through the use of SimHash. As can be seen in Figure 4.8, this reduction in the number of hashes to explore for does not translate into performance increases. For larger values of k, DMQL-R performs worse than the previous implementation, yet not significantly so. Lowering k further prevents learning, likely due to too many instances of Hash Problem 2 occurring (Section 3.4.3). The fact that usage of SimHash does not degrade performance significantly when greater values of k are used shows that we can have collisions in our hashing methodology, likely even some which exemplify Hash Problem 2. It should however be noted that a subset of runs was unable to provide a cooperative policy, even with $k = 100$.

This experiment therefore shows that SimHash is not a suitable solution to achieve increased performance by reducing the number of hashes to consider, when one omits a pre-processing step such as an auto-encoder. Whether or not the inclusion of such as pre-processing step will allow SimHash to cluster states together in a suitable manner remains to be explored.

4.5. Exploiting Similarity: Environmental Insights

As discussed in Section 3, we have established that there exists a 'weak' notion of similarity. Crucially, this notion of similarity depends on roughly similar ratios of Q-values in contexts.

4.5.1. Contextual Hashes (DMQL-C / DMQL-CS)

As the notion of similarity we use is centered around agents, we experiment with contextual hashes, hashes which are based on subsets of the information contained in the state configuration which look similar to agent observations. In this experiment, the first major difference between this and previous hashing strategies is that the coordinate system is no longer absolute, but relative to the agents. The second major difference in this strategy is identity indifference: if there should be no functional difference between actors in the environment, we do not represent them either. This strategy collides all translations of configurations with the same relative entity positions to the same hash, as well as all permutations of actor locations within type groups. For the Hunter-Prey environment, identities of entities in the environment should be mostly irrelevant, with the exception of actor type identification, i.e. what type of actor it is (e.g. a hunter or a prey). This experiment therefore also explores whether it is reasonable to disregard actor identities for determining similarity when it is reasonable to do so in observations.

As we assume homogeneity between agents for this hash, we can augment the dictionary system to allow multiple agents to share the same values. This allows us to share information from learning from experiences between agents, and aids us in reducing the required amount of exploration before the maximisation strategy is able to achieve its goals. This experiment therefore also explores whether performance increases can be gained from sharing our dictionary mechanism between agents, if the agents are homogeneous.

Experiment: Similarity from Insights: Contextual Hashing (DMQL-C / DMQL-CS) Entire Environment Size Included

Evaluated Algorithms	DMQL, DMQL-C, DMQL-CS
Environment	
Observation	Observing Full State
Punishment	$100\%r$ with $r = 10$
World Size	5×5
Agents / Prey	2 / 2
Shared	
γ	0.95
ϵ	$1 \rightarrow 0.05$ over $[250, 500]k$T after $[0, 500]k$T delay
DMQL(-C(S))	
Hashing strategy	(Agent-Centred) Full State Information (Disregarding Identities)
Degradation factor d_t	0.8
Soft update factor u_s	0.005

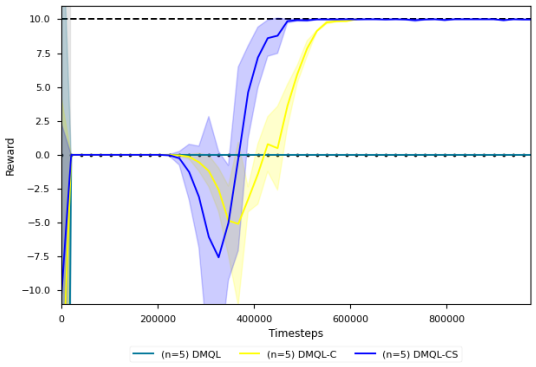

Figure 4.10: Disregarding Identities Experiment. (DMQL: $\epsilon\ 1 \rightarrow 0.05$ over $200k$ T after 700k T delay) (DMQL-C: $\epsilon\ 1 \rightarrow 0.05$ over $200k$ T after 300k T delay) (DMQL-CS: $\epsilon\ 1 \rightarrow 0.05$ over $200k$ T after 200k T delay)

Figure 4.10 shows that introducing an additional actor into the environment prevents the first DMQL implementation from learning within a reasonable timeframe given this environment configuration. The experiment shows that a significant improvement in performance can indeed be gained by removing unneeded information from the hash calculation (DMQL-C). The results for DMQL-CS, a variant disregarding identities and sharing dictionaries between agents, shows even more substantial performance benefits.

It can be argued that the strategy employed for this contextual hashing strategy still includes unnecessary information in the form of information about actors which are not near to the agent in question. The

next experiments shall evaluate whether we can safely disregard this information, or that Hash Problems 2a and 2b prove to be significant hindrances, and whether or not this can lead to performance increases.

In order to provide initial insights into the impact of Hash Problem 2a, we perform an experiment on an environment with two hunters and two prey. Experiments are run with 2-step and 3-step windows of information inclusion. As this environment has few actors, the number of hashes to consider for the larger window does not exceed realistic bounds. This setup allows us to ascertain the impact of disregarding information further than n steps removed from the agents, and more specifically Hash Problem 2a, as Hash Problem 2b cannot be encountered due to there being only one possible pair of hunters. If Hash Problem 2a occurs often enough, we expect the 3-step variant to converge significantly closer to the theoretically optimal mean reward. We additionally expect degression of mean rewards after the initial stage of rapid mean reward growth, due to the increased impact of our maximisation strategy when high values are stored for the 'uninformative contexts' (which takes time to pass down from rewardable configurations).

Experiment: Exploring Hash Problem 2a (1/2)

Evaluated Algorithms	DMQL-CS
Environment	
Observation	Observing Full State
Punishment	$100\%r$ with $r = 10$
World Size	6×6
Agents / Prey	2 / 2
Shared	
γ	0.95
ϵ	$1 \rightarrow 0.05$ over $200k$T after $200k$T delay
DMQL-CS	
Hashing strategy	Agent-Centred State Information Subsets Disregarding Identities
Degradation factor d_t	0.8
Soft update factor u_s	0.005

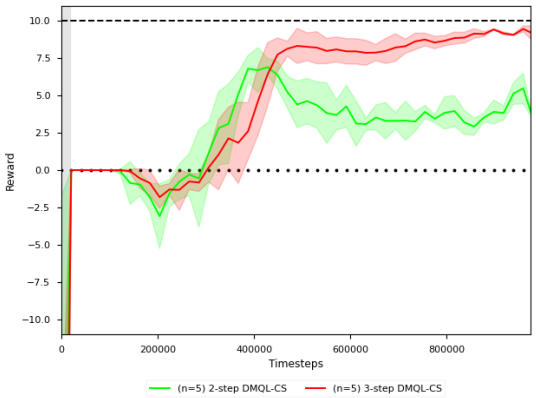

Figure 4.11: Exploring Hash Problem 2a.

Figure 4.11 shows a significant difference between DMQL variants using 2- and 3-step windows. The 2-step windows should be sufficient to establish a collaborative policy when only considering rewardable configurations. Whilst the 3-step variant initially seems to converge slower due to the increased amount of irrelevant information available to it, it degresses far less than the 2-step variant.

If one takes all possible state configurations and generates views for both variants, we see that views without any actor information constitute 39.5% and 18.5% of the total possible configurations for the 2-step and 3-step variants respectively. Whilst this type of view arguably is the least informative for our maximisation step, other views might also not provide sufficient information. Views containing both an agent and at least one prey, which should be relatively informative for this setup, constitute 13.1% and 32.2% respectively. It should be noted that these ratios are not necessarily representative of actual encounter rates, yet might be indicative. These insights and the results observed in Figure 4.11 give rise to interesting questions: what views are informative enough, and can results be improved when the maximisation step is no longer applied to uninformative hashes?

4.6. Exploiting Similarity: Selective Utilisation

In order to explore what views are informative and whether excluding uninformative views from the maximisation step yields better results, we introduce selectors based on properties of the state configuration within the view. This allows us to prevent maximisation from being utilised when insufficient information is present. This has the benefit of allowing our maximisation step to be used solely where it is able to provide benefits, with other situations being handled by IQL, using all available information from histories.

In order to provide further insights on Hash Problem 2a, we utilise selectors for the 2-step variant of the previous experiment. The various selectors used reflect various levels of insights into the Hunter-Prey problem, with more specific selectors targeting situations in which relative overgeneralisation can occur more specifically, letting conventional IQL handle all other situations.

Our most specific selector, 'catchable' selects only views in which the hunter in question can attempt to catch a prey, with another hunter present in the view which could do the same for this same prey. 'see both' selects views in which the other actors required to successfully get rewarded are present (at least one other hunter and at least one prey). 'adjacent prey' selects views in which the agent can attempt to catch a prey (even when there is no other agent present, therefore yielding punishments), and 'see agent' and 'see prey' are selectors which select views in which at least one agent or at least one prey are present respectively.

We expect the 'catchable' selector to yield the best results, as it most specifically targets relative overgeneralisation, and therefore only includes the most informative views. We additionally expect the other selectors to yield results between the results for this selector and the earlier results using no selector, with better results indicating that a selector was more informative and vice versa. We expect 'see both' to be the second-most effective selector due to it excluding most completely uninformative contexts as well. In addition, we expect the 'see agent' selector to be the least effective, as contexts in which 'at least one other agent is present' are likely only marginally more informative for decision making than the baseline on average.

Experiment: Exploring Hash Problem 2a (2/2)

Evaluated Algorithms	DMQL-CS
Environment	
Observation	Observing Full State
Punishment	$100\%r$ with $r = 10$
World Size	6×6
Agents / Prey	2 / 2
Shared	
γ	0.95
ϵ	$1 \rightarrow 0.05$ over $200k$T after $200k$T delay
DMQL-CS	
Hashing strategy	Agent-Centred 2-Step State Information Subsets Disregarding Identities Using Selectors
Degradation factor d_t	0.8
Soft update factor u_s	0.005

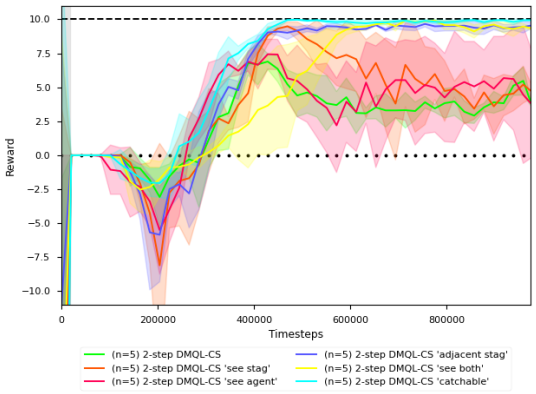

Figure 4.12: Exploring Hash Problem 2a (using selectors).

Figure 4.12 shows us the performance of our various selectors. The environment used in this experiment is the same as the one from the previous experiment, allowing us to compare the results for these 2-step context strategies directly.

It can be observed that, as expected, the most specific selector, 'catchable', was most effective. By limiting DMQL's maximisation step to only contexts in which relative overgeneralisation can occur, Hash

Problem 2a was fully prevented. As expected, the 'see both' selector was also quite effective, yet performed marginally worse than the 'catchable' selector, which was caused by Hash Problem 2a, as Hash Problem 2b could not have occurred in this environment with only two agents.

Unexpectedly, the 'adjacent stag' selector also performed quite well, on par with the 'see both' selector even. We speculate this is due to the fact that the number of contexts in which stags are adjacent to the agent are quite limited, whilst this adjacency also is a requirement for encountering relative overgeneralisation.

The fact that the number of contexts in which stags are adjacent to the agent are quite limited is the main factor which differentiates it from the 'see stag' selector. Whilst having a stag in the context is a requirement for encountering relative overgeneralisation, this selector does not exclude nearly as many contexts as the previous one, allowing Hash Problem 2a to once again take place.

Lastly, the 'see agent' selector was the least informative of the ones selected for this environment. Similarly to the previous selector, this selector does not seem to have excluded nearly enough contexts to prevent Hash Problem 2a from taking place, causing performance to be nearly as bad as when we used no selector at all.

The results of this experiment show that DMQL agents using contextual hashes can converge to cooperative policies consistently when one prevents Hash Problems 2a and 2b.

4.7. Partial Observability

One of the main benefits we stated during the introduction of DMQL was that it *should* work for partially observable environments. In addition, we would still like to validate whether DMQL works for even larger environments than considered until now, and with more than one agent pair, which was not feasible to do in any other experiments due to resource constraints. Hence, we shall experiment with partially observable environments to both evaluate the performance of DMQL when partial observability is introduced, and evaluate whether DMQL can successfully operate on larger and more complex environments without requiring significantly more time resources.

In order to explore both of these aspects, we shall approximately double the environment size (from a 6×6 world to 8×8 and 9×9 worlds), double the number of actors in the environment, and restrict observability to a square of size 7×7 (which occupies $25.0\% - 76.6\%$ and $19.8\% - 60.5\%$ of the environments respectively). In addition, we shall perform an experiment applying DMQL usage decay, as it could potentially yield great benefits in partially observable environments such as these, where agents can actually benefit from taking the information in their histories into account (with full observability, agents had access to almost all state information, mostly preventing the need).

Experiment: Partial Observability

Evaluated Algorithms	IQL, DMQL-CS
Environment	
Observation	Observing Full State, or 7×7 square ($o_r = 3$)
Punishment	$100\%r$ with $r = 10$
World Size	$x \times x$ with $x \in \{8, 9\}$
Agents / Prey	4 / 4
Shared	
γ	0.95
ϵ	$1 \rightarrow 0.05$ over $200k$T after $200k$T delay
DMQL-CS	
Hashing strategy	'Catchable' only 2-Step
Degradation factor d_t	0.8
Soft update factor u_s	0.005
Usage schedule	$1 \rightarrow 0$ over $200k$T after $400k$T delay when 'decay' else 1 throughout.

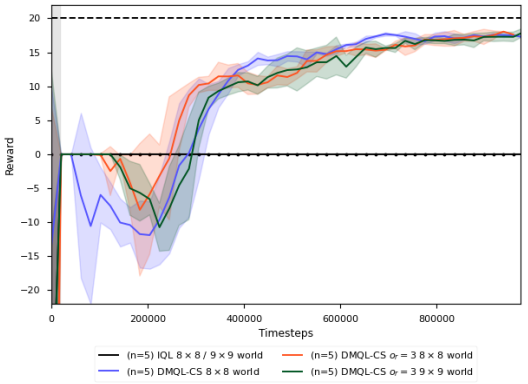

Figure 4.13: Partial Observability Experiment

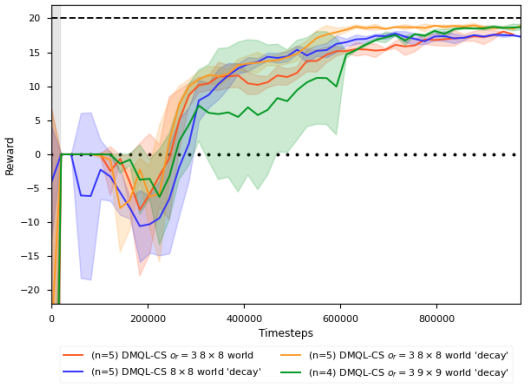

Figure 4.14: Partial Observability Usage Decay Experiment

The first thing Figure 4.13 shows us that DMQL agents are able to develop cooperative policies in environments which are approximately twice as big as the ones considered up to this experiment, and which contain double the agents of the previous experiments. These results were achieved within the same timeframe, and with the same DMQL configurations, as the results of the previous experiment. This signifies that this version of DMQL scales relatively well.

In addition to this, the methods which prevent Hash Problem 2a seem to be sufficiently successful at limiting the impact of Hash Problem 2b, which had become a concern as multiple agent pairs are present in the environments of this experiment.

Most interestingly, DMQL was able to develop cooperative policies in partially observable environments without any configuration changes. In fact, DMQL was able to perform *just as well* in environments with partial observability as it could in environments without it.
When DMQL was run in an environment of size 9×9, it was able to find equally good policies as it could in the same timeframe for environments of size 8×8, despite the larger environments being 27% larger. This, in combination with our earlier findings, suggests that we have successfully been able to remove environment size as the primary limit on which environments we can learn cooperative policies for.

Lastly, looking at Figure 4.14, we are able to see the impact of introducing usage decay, which decreased utilisation of DMQL's maximisation step from 100% to 0% linearly between time steps 400.000 and 600.000. As expected, we see no significant change between the results for fully-observable environments. For partially-observable environments, we are able to observe a slight, yet significant, increase in performance.

5

Discussion & Conclusion

In this work, we introduced DMQL, a methodology which aims to prevent relative overgeneralisation in Deep Learning processes employing independent learners. Without exploiting similarity, this methodology was able to match results expected from Distributed Q-Learning. As expected, however, exploitation of similarity was required to scale well. After defining practical similarity heuristics and deriving hashing strategies to cluster similar states and, therefore, histories, DMQL scales significantly better. The development process of these hashing strategies can, however, be plagued by various problems. Various strategies have been devised which be employed to combat these problems, including selective utilisation of DMQL's core maximisation step.

DMQL is able to prevent relative overgeneralisation from inhibiting the development of cooperative policies in a scalable manner, and is able to function in partially observable environments. Through the use of hash-action pseudo-maximum target estimation value dictionaries, we were able to cluster similar states and, therefore, histories together, whilst preventing overestimations from causing divergent behaviours. Clustering histories was achieved by incorporating notions of similarity into hash generation strategies, of which the output formed partial keys to our dictionaries. Divergent behaviours were prevented by using pseudo-maximum values derived from maximums taken over target network estimates, instead of taking maximum values from outdated network parameterisations directly.

5.1. Discussion and Future work

In retrospect, Hash Problem 2b, which was introduced in this work and was entirely caused by the fact that reward causality could often not be included in hashes, could have been entirely prevented by augmenting the reward function defined in Section 2. Instead of defining this collaborative reward function such that rewards would be received by all agents irrespective of if they caused the reward to happen, or if they could observe the cause at all, we could have opted to only assign rewards to the actors which directly caused it to be granted, i.e. we could have calculated the collective reward as the sum of individual rewards instead of the approach used in this work. This would have caused our agents to only 'observe' the reward if they had been responsible for it, hence preventing lack of clarity about causality in our dictionary entries.

Interestingly, we believe to have accidentally prevented another common pathology 'miscoordination' from manifesting when our value degradation strategy is employed:
The pathology known as 'miscoordination' happens in the special case of two Nash Equilibria having exactly the same value. To exemplify this, one can imagine the Hunter-Prey environment used in this work requiring the coordinated action of two agents through two different actions, e.g. 'distract' and 'stab'. Normally, as both actions should have the same value for two homogeneous agents, one would need to device a strategy to break the ties in the Nash Equilibrium formed, as our optimistic learners could otherwise greedily select joint actions such as ('distract', 'distract'), which are not effective. For DMQL, the establishment of induced Nash Equilibria with the same exact value like these is prevented automatically: if both actions have different values given some context, agents are exceptionally unlikely to

'reverse roles' through random exploration sufficiently often to bring dictionary values for these actions to equality. If miscoordination occurs due to exploration, the value is degraded in equal proportions for both agents. Whilst this could lead to both agents 'preferring' the same action, this is combatted by the degradation mechanism: if both agents choose the same value, the respective dictionary entries are degraded due to the lack of a reward. This process can continue until one of the agents performs an exploratory action, or when the highest values for the two catch actions becomes the other catch action after degradation, once again yielding joint action ('distract', 'stab'). In short, when the degradation mechanism is used, the situations in which these two actions would have the same dictionary values would be unstable, and vice versa.

We suggest various directions for future works, and some opportunities to further improve efficiency:

5.1.1. 'Transfer Learning'

In this work, we construct dictionaries which map hash-action pairs to our pseudo-maximum target estimate values, the hashes used can be based on agent-centric representations of the state which can take information within any specified range of the agent. Additionally, we can store dictionaries to re-use them at a later time. As contexts, and thus the hashes derived from them, do not have to be run- or even environment configuration-specific, it would be interesting to explore whether or not we can train DMQL once on one environment, filling the dictionary with realistic targets given some contexts, and then use the data again in another, possibly much larger environment with the same rules.
As the dictionary should already contain pseudo-maximum target estimate values for one environment with the same rules, agents would not even need to encounter 'good' outcomes of situations in which relative overgeneralisation can occur before they could be granted a means to introduce optimism. This, in turn, could help to significantly improve convergence speed. In addition, we expect issues relating to Q-values being lower on average for the larger environments to be mostly prevented by the dictionary value degradation strategy. Whether or not these expectations are valid, remains to be explored.

5.1.2. Advantage-based Approaches

As the keen reader might have picked up on, this work exploits a weak notion of similarity which is described to aim to cluster histories with similar action Q-value ratios together, yet clusters contexts together which should have similar *advantage* ratios, not Q-value ratios. Whilst this work shows that it is certainly possible to achieve collaborative policies this way, the Q-values estimated by our Q-networks may end up relatively high. As advantages tell us what action would be the best one to take from where in the environment the agent happens to be at any given moment, they are the deciding factor in (non-stochastic) decision making, and hence ideally should be the values we work with instead.

5.1.3. Fully Decentralised Learning

We expect DMQL to be able to function as a fully decentralised learning approach if parameter sharing and shared dictionaries are disabled, and if one designs hash functions based on representations of the current observation or a subsection of the action-observation history.
Whilst disabling parameter sharing and shared dictionaries would most definitely impact convergence speed, it should, for some environments, most definitely be possible to develop such a hashing strategy (in this work, the 2-step contexts consistently contains information which is a subset of the information contained in the respective observations for the same agents).

5.1.4. Exploiting Symmetries

Whilst we do mention spatial symmetries in this work, as a way to adhere to our strict notion of similarity whilst still reducing the number of hashes to consider, we do not exploit these symmetries when we are working with our weak notion of similarity, due to the added layer of complexity. Utilising these symmetries would, however, still provide similar benefits when employing the weak notion of similarity.

5.1.5. Retroactive Selection Procedures

Whilst we use 'selectors', or 'selection functions', in this work, these could be completely omitted in a practical manner by retroactively selecting contexts which have yielded coordinated joint actions and adding them to a 'whitelist', as opposed to 'blacklisting' all contexts which do not meet some requirements determined beforehand. This way, the equivalent of a 'catchable' selector could be implemented without requiring anywhere near as much knowledge about the environment.

References

Böhmer, W., Kurin, V., & Whiteson, S. (2020). Deep coordination graphs. *International Conference on Machine Learning*, 980–991.

Bowling, M., & Veloso, M. (2001). Rational and convergent learning in stochastic games. *International joint conference on artificial intelligence*, *17*(1), 1021–1026.

Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, *136*(2), 215–250.

Castellini, J., Oliehoek, F. A., Savani, R., & Whiteson, S. (2019). The representational capacity of action-value networks for multi-agent reinforcement learning. *arXiv preprint arXiv:1902.07497*.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 380–388.

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H., Kohli, P., & Whiteson, S. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. *International conference on machine learning*, 1146–1155.

Gupta, T., Mahajan, A., Peng, B., Böhmer, W., & Whiteson, S. (2021). Uneven: Universal value exploration for multi-agent reinforcement learning. *International Conference on Machine Learning*, 3930–3941.

Hasselt, H. (2010). Double q-learning. *Advances in neural information processing systems*, *23*.

Hausknecht, M., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *2015 aaai fall symposium series*.

Kale, D. C., Gong, D., Che, Z., Liu, Y., Medioni, G., Wetzel, R., & Ross, P. (2014). An examination of multivariate time series hashing with applications to health care. *2014 IEEE international conference on data mining*, 260–269.

Kapetanakis, S., Kudenko, D., & Strens, M. J. (2002). Reinforcement learning approaches to coordination in cooperative multi-agent systems. *Adaptive agents and multi-agent systems* (pp. 18–32). Springer.

Kaushik, M., Krishna, K. M. et al. (2018). Parameter sharing reinforcement learning architecture for multi agent driving behaviors. *arXiv preprint arXiv:1811.07214*.

Kobayashi, T., & Ilboudo, W. E. L. (2020). T-soft update of target network for deep reinforcement learning. *CoRR*, *abs/2008.10861*. https://arxiv.org/abs/2008.10861

Lauer, M., & Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. *In Proceedings of the Seventeenth International Conference on Machine Learning*, 535–542.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, L.-J. (1992). *Reinforcement learning for robots using neural networks*. Carnegie Mellon University.

Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2012). Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems. *The Knowledge Engineering Review*, *27*(1), 1–31.

Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2007). Hysteretic q-learning: An algorithm for decentralized reinforcement learning in cooperative multi-agent teams. *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 64–69.

Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2009). Coordination of independent learners in cooperative markov games.

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), 529–533.

Montague, P. R. (1999). Reinforcement learning: An introduction, by sutton, rs and barto, ag. *Trends in cognitive sciences*, *3*(9), 360.

Oliehoek, F. A., & Amato, C. (2016). *A concise introduction to decentralized pomdps*. Springer.

O'Neill, J., Pleydell-Bouverie, B., Dupret, D., & Csicsvari, J. (2010). Play it again: Reactivation of waking experience and memory. *Trends in neurosciences*, *33*(5), 220–229.

OroojlooyJadid, A., & Hajinezhad, D. (2019). A review of cooperative multi-agent deep reinforcement learning. *arXiv preprint arXiv:1908.03963*.

Palmer, G., Tuyls, K., Bloembergen, D., & Savani, R. (2017). Lenient multi-agent deep reinforcement learning. *arXiv preprint arXiv:1707.04402*.

Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, *11*(3), 387–434.

Panait, L., Luke, S., & Wiegand, R. P. (2006). Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation*, *10*(6), 629–645.

Panait, L., Sullivan, K., & Luke, S. (2006). Lenience towards teammates helps in cooperative multiagent learning. *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems–AAMAS-2006*.

Panait, L., Tuyls, K., & Luke, S. (2008). Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *The Journal of Machine Learning Research*, *9*, 423–457.

Potter, M. A., & Jong, K. A. D. (1994). A cooperative coevolutionary approach to function optimization. *International conference on parallel problem solving from nature*, 249–257.

Rashid, T., Farquhar, G., Peng, B., & Whiteson, S. (2020). Weighted qmix: Expanding monotonic value function factorisation. *arXiv e-prints*, arXiv–2006.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., & Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *International Conference on Machine Learning*, 4295–4304.

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., & Yi, Y. (2019). Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *International Conference on Machine Learning*, 5887–5896.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. (2017). Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.

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

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., De Turck, F., & Abbeel, P. (2017). # Exploration: A study of count-based exploration for deep reinforcement learning. *31st Conference on Neural Information Processing Systems (NIPS)*, *30*, 1–18.

Terry, J. K., Grammel, N., Hari, A., Santos, L., & Black, B. (2020). Revisiting parameter sharing in multi-agent deep reinforcement learning. *arXiv preprint arXiv:2005.13625*.

Tsitsiklis, J., & Van Roy, B. (1996). Analysis of temporal-diffference learning with function approximation. *Advances in neural information processing systems*, *9*.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the AAAI conference on artificial intelligence*, *30*(1).

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292.

Watkins, C. J. C. H. (1989). Learning from delayed rewards.

Wei, E., & Luke, S. (2016). Lenient learning in independent-learner stochastic cooperative games. *The Journal of Machine Learning Research*, *17*(1), 2914–2955.

Wiegand, R. P. (2004). *An analysis of cooperative coevolutionary algorithms*. George Mason University.

A

Plotting Methodology

In order to be able to plot results for various runs of the same algorithm as one line despite the datapoints of these runs not lining up with each other on the x-axis, we apply histogram smoothing to each seed with bins containing equal sections of the integer number line between 0 and the maximum number of time steps we train the algorithm for. For each seed, we determine a mean for each bin. Subsequently, we use these means per seed to calculate a mean and standard deviation between seeds.

In order to provide a clear representation of the results, the means for differing configurations are depicted as solid lines of various colours, with shaded areas with the same color (but different opacity) representing the standard deviation between seeds. IQL runs are consistently depicted with the color black. Zero is represented by a dotted line, and if applicable the theoretical maximum value attainable by the algorithm is depicted using a dashed line. The range of values depicted is chosen to be the entirety x-axis occupied by results, whereas the range of the y-axis is limited to ranges between -105% and 105% of the maximum attainable value if applicable.

B

Hunter-Prey Environment Rules

The environment is initialised with all actors at random positions in a grid world, none of them overlapping.

At any time step t, the environment shall provide agents with their respective observations and available actions, after which each agent will provide the environment with the action they chose from the set of provided available actions. The 'Idle' action is always available to agents, movement options are only available if they would not move the agent into a wall, and if an agent has not been removed from the grid. Catch actions are available if a prey is adjacent. Prey follow the same rules for action availability as agents/hunters, but always move randomly and do never have catch actions available to them.

After actions have been provided to the environment, hunters' actions are processed first. Attempted catch actions are the first to be processed. For each prey in the environment, in random order, we check if adjacent hunters are performing a catch action. If only only one hunter is performing a catch action, a collective punishment is issued. If multiple hunters perform a catch action, a collective reward is issued, and all catching hunters and the prey are removed from the grid. If no prey or no hunters remain, the game ends.

Agent movement actions are processed after catch actions. The order in which agents are allowed to move is random. If the location to which an agent is attempting to move is already occupied, the movement action is aborted. Agent movements yield neither punishments nor rewards.

After agent actions have been processed, all prey move randomly, in random order, with the same restrictions on movement actions being completed.

An interesting consequence of the rule that catch actions are considered from the perspective of prey, and in a random order, is that the outcome of the same joint ('catch', 'catch') action *can be different*, *if* at least one of the hunters has an additional prey adjacent to it.

C

Exploratory Experiment

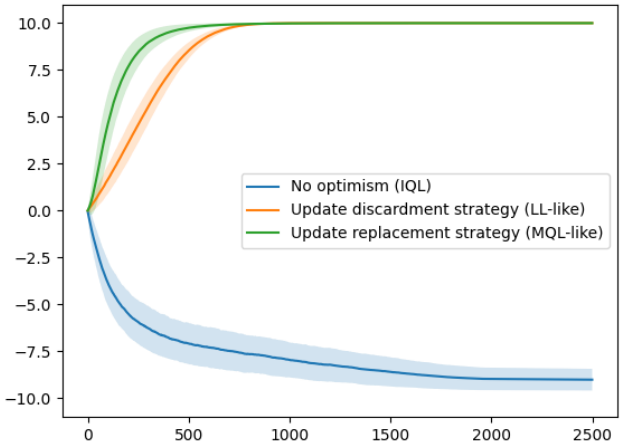

Figure C.1: Exploratory experiment comparing stateless tabular cases of the Hunter-Prey environment for a strategy without optimism, one with a discardment-based strategy, and one with a replacement-based strategy.

In Figure C.1, runs for various strategies are depicted. In this experiment, there is no state, the agents are repeatedly asked to perform one of the actions of the Hunter-Prey environment. For this experiment, we used simple tabular Q-learning and the following parameters:

Punishment	10
Reward	10
time steps	2500
ϵ_{start}	1
ϵ_{end}	0.01
ϵ-decay time steps	2000
Learning rate	0.05
runs	1000

Table C.1: Exploratory experiment configuration

Here one can see that the usage of optimism enables the agents to learn that collaborating is good, and results in the optimal policy. Apart from a slight increase in convergence speed, there is no significant difference between the outcomes for both optimistic strategies.