QD(λ) Learning: Towards Multi-agent Reinforcement Learning for Learning Communication Protocols

Nicola Mc Donnell
University of Galway
Galway, Ireland
n.mcdonnell9@universityofgalway.ie

Enda Howley
University of Galway
Galway, Ireland
enda.howley@universityofgalway.ie

Jim Duggan
University of Galway
Galway, Ireland
james.duggan@universityofgalway.ie

ABSTRACT

Multi-agent Reinforcement Learning (MARL) extends single-agent RL, studying multiple agents operating in a shared environment. One classification of MARL systems relates to whether the agents are controlled centrally or in a decentralised manner. Decentralised MARL algorithms with networked agents can exchange information with their neighbouring agents, to facilitate convergence. While these approaches can significantly outperform independent agents, they may learn slowly in the beginning. We propose the QD(λ) learning algorithm, a novel value-based MARL algorithm that leverages networked agents. It incorporates the advantages of transfer learning from the MARL QD learning algorithm and faster learning when rewards are delayed from the RL Q(λ) learning algorithm. The empirical results support the hypothesis that the QD(λ) learning algorithm can learn faster than three other RL algorithms for environments where the agent must transition through many states before it receives a reward. Thus, QD(λ) learning is suited to multi-agent systems learning a communication protocol, an emerging research area that has seen significant activity in the last few years.

KEYWORDS

Multi-agent systems, Transfer Learning, Multi-Agent Reinforcement Learning

1 INTRODUCTION

Reinforcement learning (RL) is a single-agent learning framework, where with no prior knowledge, the agent aims to learn what action to take next, given the state of the environment at that time, to achieve its goal [1]. It tries different actions, assesses their outcome, and learns the best actions to take. Typically, the agent meets its goal and realises the award at the end of a series of actions. Multi-agent Reinforcement Learning (MARL) extends single-agent RL, studying multiple agents operating in a shared environment [2, 3].

One classification of MARL systems relates to whether the agents are controlled centrally or in a decentralised manner. Figure 1 shows three multi-agent system learning schemes: centralised controller, fully decentralised and decentralised with networked agents [4].

With the centrally controlled scheme, the central controller receives the observations from all the agents and decides the actions to be taken by each agent, and receives a joint reward. It follows the *centralised-learning-decentralised-execution* paradigm and has recently been widely adopted with multi-agent deep reinforcement learning research [5]. However, this scheme is only possible when a central controller can exist, which is often not the case. A further challenge is that the state space expands with the number of agents.

A MARL system is said to be "fully decentralised" when the agents are fully autonomous, have no central controller and do not coordinate with one another. In the fully decentralised learning scheme, each agent individually makes observations, takes actions and receives rewards. The agents learn independently. Several fully decentralised schemes exist. Examples include classical Q-learning [6], Distributed Q Learning [7], Hysteretic Q learning [8] and two approaches proposed by Zeng, which combine Q-learning and actor-critic methods [9]. A shortcoming of this scheme is that it suffers from non-convergence [10].

Transfer learning is a machine learning technique which leverages previous knowledge to improve learning speed or performance [11]. When applied to MARL, this knowledge can come from another agent. Thus, a powerful extension to the fully decentralised learning scheme is where agents can communicate over a network. They exchange information with their neighbouring agents. This scheme facilitates convergence to an optimal decision policy where the performance of the multi-agent system stabilises.

However, research on MARL contends that while "agents engaging in partnership can significantly outperform independent agents … they may learn slowly in the beginning" [10]. Our research aims to develop a MARL algorithm that overcomes this limitation. Additionally, it aspires to design a MARL algorithm suited to a sizeable multi-agent system learning a communication protocol. Research into MARL systems learning communication protocols is an emerging area that has seen significant activity in recent years [12–18]. As such, the main contribution of this research is the QD(λ) learning algorithm, a value-based MARL algorithm that leverages networked agents.

This paper is structured as follows. The next section presents the foundational research for this novel MARL algorithm. In Section 3.1, we describe the QD(λ) learning algorithm. Section 3.2 presents the test reinforcement learning environments designed to emulate agents learning communication protocols. The experiments presented in Section 4 compare the QD(λ) learning algorithm against two other RL algorithms and one MARL algorithm assessing its convergence speed and quality. In Section 5, we propose the general hypothesis that the QD(λ) learning algorithm learns faster than the other RL algorithms for the environments and present several limitations of the algorithm and research. Finally, in the last section, Section 6, we consider the impact of the algorithm and research findings and propose future work.

ALA '23, May 29-30, 2023, London, UK, https://alaworkshop2023.github.io/
2023.

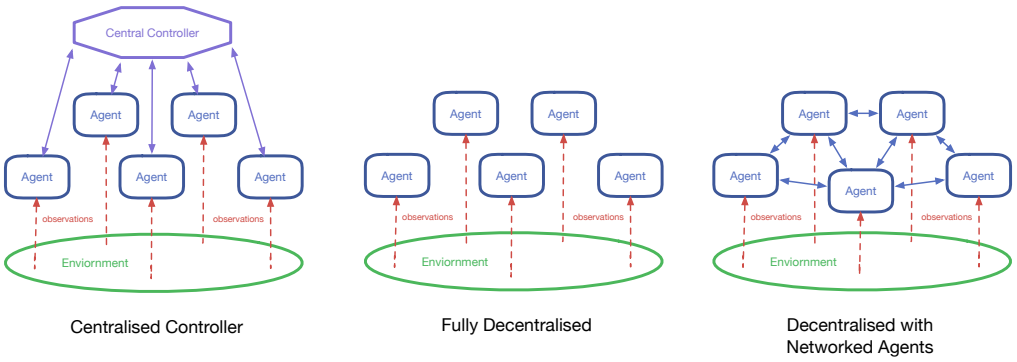


Figure 1: Three multi-agent system learning schemes: centralised controller, fully decentralised and decentralised with networked agents [4]

2 RELATED WORK

This section describes the foundational research for the QD(λ) learning algorithm. First, it covers the Watkins Q(λ) learning algorithm [19] on which QD(λ) builds. Then, it critiques other MARL algorithms that use networked agents, describing the QD learning algorithm [20] in detail, as it heavily inspires the QD(λ) learning algorithm.

2.1 Watkins Q(λ) Learning

Watkins Q(λ) Learning is an extension to his earlier Q Learning algorithm that incorporates the concept of eligibility traces [19]. An eligibility trace is a temporal record of taking an action in a state, or "passing through" a state-action. In Q(λ) Learning, when we update a state-action's Q value, we also update other recently visited state-actions. As a result, Q(λ) Learning may learn more efficiently. A parameter to the Q(λ) learning algorithm, the eligibility decay, λ, controls how quickly the update of the visited state-actions decays. The lower the eligibility decay, the faster the update diminishes.

Several variants of Q(λ) learning differ in how they accumulate the eligibility traces, though similar in how they decay the eligibility. *Accumulating traces* increments the state-action visited by one. This cause the eligibility trace for a repeatedly visited state-action to grow considerably over time. This is especially true for continuing tasks, where the agent will learn to constantly take action to remain in or return quickly to a state with a high reward. A preferred approach for this case is *replacing traces*, which resets the visited eligibility trace to one, as shown in Equation 1.

$$e(s, a) = 1 \tag{1}$$

A third approach is *dutch traces*, where the original trace value is scaled by $1 - \alpha$ before it is incremented by one.

2.2 MARL algorithms with networked agents

The earliest research into MARL with networked agents was conducted by Varshavskaya et al. in 2009 [21]. They developed DGAPS, an extension of the policy-based stochastic gradient descent algorithm, known as gradient ascent in policy space (GAPS), to the distributed realm by incorporating agreement algorithms. The agreement algorithm exchanges local rewards and experiences broadcasting to all the agents. One criticism the authors made of their work was that it sends a large amount of information at each exchange. A preferred approach is to send only the newly learned information and use a caching mechanism to record previously sent data.

The first value-based MARL algorithm with networked agents was the QD Learning algorithm [20]. In QD learning, the update rule for the Q function has two terms, an *innovation* term and a *consensus* term. The innovation term corresponds to the agent's local knowledge of the reward and best next state-action value. The consensus term corresponds to information from the agent's neighbours.

Equation 2 shows the Q function update rule for agent i, where s and a are the agent's previous state and action, respectively. R is the reward for the transition from state s to the current state, s', having taken action a. The discount factor, γ, reduces the relative importance of future rewards compared to present rewards. The learning rate or innovation step-size parameter, α, such that $\alpha > 0$, weights the newly learned value relative to its original value. The consensus term has a second learning rate, β, where $\beta > 0$. N is the set of agents in the agent i's neighbourhood. Given certain conditions on the step sizes, α and β, the algorithm is guaranteed to converge to the optimum Q-function.

$$Q^i(s,a) \leftarrow Q^i(s,a) + \underbrace{\alpha \left[R^i + \gamma \max_b Q^i(s',b) - Q^i(s,a) \right]}_{\text{innovation term}} \underbrace{- \beta \sum_{j \in N^i} \left[Q^i(s,a) - Q^j(s,a) \right]}_{\text{consensus term}} \tag{2}$$

In 2016, Mathkar et al. proposed another value-based distributed reinforcement learning algorithm [22]. It builds on the classical TD(0) algorithm where the networked agents incorporate updates they received from their neighbouring agents using a gossip-like mechanism, the simple averaging scheme from gossip. They proved that convergence is guaranteed.

Zhang et al. proposed decentralised actor-critic algorithms with value function approximation in 2018. Each agent carried out its actor step, while for the critic step, the agents shared information over the network to reach a consensus. They evaluated a large

number of agents, 20, and a large number of states, 20, with two possible actions [23]. In a follow-up publication later that year, they proposed a multi-agent version of expected policy gradient method, which they applied to continuous spaces [24]. They proved that convergence is guaranteed if linear functions are used for value function approximation. But actor-critic algorithms have drawbacks; they tend to be more complicated, both conceptually and computationally, and have a lower sample efficiency.

In summary, MARL algorithms with networked agents use transfer learning, exchanging information with their neighbouring agents, to facilitate convergence. These approaches "can significantly outperform independent agents … they may learn slowly in the beginning" [10]. Several referenced MARL algorithms with networked agents leverage policy gradient or actor-critic algorithm approaches [21, 23, 24]; these approaches have a lower sample efficiency and are conceptually, and computationally more complicated. Additionally, research into MARL systems learning communication protocols is an emerging area that has seen significant activity in recent years [12–18].

Our research aims to address these gaps by developing a value-based MARL algorithm that leverages networked agents. It should be suitable for a sizeable multi-agent system to learn a communication protocol, and it should overcome the initial slow learning limitation reported by Tan.

3 METHODOLOGY

3.1 QD(λ) Learning

The QD(λ) learning algorithm is a hybrid of the RL Q(λ) learning [19] and the MARL QD learning [20] algorithms. It combines the eligibility traces of the Q(λ) learning algorithm with the consensus term — information from the agents' neighbourhoods — from QD learning, as shown in Figure 2.

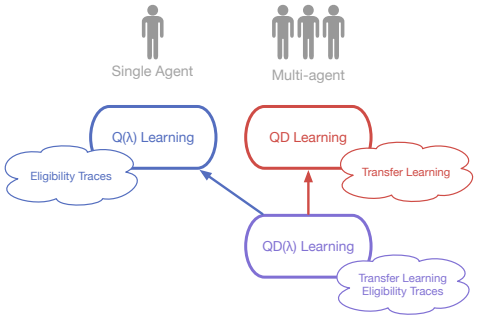


Figure 2: The relationship between the Q(λ) learning, QD learning and QD(λ) learning algorithms.

The parameters in QD(λ) learning, described in Table 1, are a combination of those from the Q(λ) learning and QD learning algorithms.

The QD(λ) learning algorithm is shown in Algorithm 1. At the initialisation of the learning process, the algorithm sets the AVF, $Q(s,a)$, and the eligibility traces, $e(s,a)$, to zero and randomly initialises the ϵ-greedy policy, π. Next, it sets the initial state, s, and the initial action, a. Then, it randomly connects each agent to N other agents. The agents use this network to transfer learning information, specifically the $Q(s,a)$ values that comprise the consensus term of the update rule. The body of the algorithm is repeated for each learning step.

Table 1: The parameters of QD(λ) learning algorithms.

Parameter		Description
Discount factor	γ	Present value of future reward
Eligibility decay	λ	Relative updates to recently visited state actions
Innovation step size	α	Rate at which an algorithm converges to a solution
Consensus step size	β	Weight of learning information received from the agent's neighbourhood
Number of neighbours	N	Number of neighbours from whom the agent receives learning information

First, the algorithm takes action a and transitions to state s'. It gets the reward, R, for the transition (s,a,s'). Then, it chooses the next action, a', from the policy, where the set of all possible actions for the state is $A(s')$. The algorithm will differ when it takes a non-optimal action — an exploratory action with a lower value in the AVF than the policy's greedy action. It records the policy's greedy action, a^*, to ascertain this later. However, when a' has the same values in the AVF as a^*, it sets a^* as the action to be taken.

Then, the algorithm sets the eligibility trace for the previous state action, $e(s,a)$, to one. The AVF update is a combination of the *innovation* term, $\alpha\left[R+\gamma Q(s',a^*)-Q(s,a)\right]$ and the *consensus* term, $\beta\sum_{j=N}\left[Q(s,a)-Q^j(s,a)\right]$. For every state action in the AVF with an eligibility trace greater than zero, the value is updated with the eligibility trace for the state actions scaling the update. Equation 3 shows the full update rule.

$$Q^i(s,a) \leftarrow Q^i(s,a) + e^i(s,a)\Bigg[\underbrace{\alpha\left[R+\gamma\max_b Q^i(s',b)-Q^i(s,a)\right]}_{\text{innovation term}} \underbrace{-\beta\sum_{j\in=N^i}\left[Q^i(s,a)-Q^j(s,a)\right]}_{\text{consensus term}}\Bigg] \tag{3}$$

After the algorithm updates the AVF, it updates the eligibilities. When it takes an exploratory step, it resets all the eligibility traces to zero. Otherwise, when $a' = a^*$, it decays the eligibilities for all state actions. It does this, as with the Q(λ) learning algorithm, because steps before the exploratory steps no longer have the required relationship to the greedy policy. Finally, the algorithm improves the greedy policy using the newly-updated action-value function. It sets the state-action, (s,a), to the new state-action, (s',a'), and continues to the next learning step.

Algorithm 1: QD(λ) learning

Algorithm Parameter: discount $\gamma \in [0, 1]$
Algorithm Parameter: innovation step size $\alpha > 0$
Algorithm Parameter: consensus step size $\beta > 0$
Algorithm Parameter: eligibity trace decay rate $\lambda \in [0, 1]$

Initialise: $Q(s, a) = 0$, $e(s, a) = 0$ for all $s \in S+$, $a \in A(s)$
Initialise: ϵ-greedy policy π randomly
Initialise: s to initial state
Initialise: a to initial action
Initialise: connect each agent to N other agents randomly

foreach *learning step* **do**
  Take action a and transition to state s'
  $R :=$ get reward for transition (s, a, s')
  Choose a' from $A(s')$ using π
  $a^* := argmax_b\ Q(s', b)$ if a' ties for max then $a^* \leftarrow a'$
  $\delta_i := \alpha\left[R + \gamma Q(s', a^*) - Q(s, a)\right]$ `// Innovation Term`
  $\delta_c := \beta \sum_{j=N} \left[Q(s, a) - Q^j(s, a)\right]$ `// Consensus Term`
  $e(s, a) := 1$
  for *all* (s, a) *where* $e(s, a) > 0$ **do**
    $Q(s, a) := Q(s, a) + e(s, a)\left[\delta_i - \delta_c\right]$
  end
  for *all* (s, a) **do**
    if $a' = a^*$ **then**
      $e(s, a) := \gamma\lambda e(s, a)$ `// Decay eligibilities`
    else
      $e(s, a) := 0$ `// Took exploratory action`
    end
  end
  Improve π greedily with respect to Q
  $s := s'$ and $a := a'$
end

3.2 Reinforcement learning Test Environments

The experiments use randomly generated reinforcement learning environments to evaluate the QD(λ) learning algorithm. They were specially-designed environments that mimic a multi-agent system learning a communication protocol. In them, the agents must transition through many states before they receive a reward. This is equivalent to an environment with sparse rewards [25] and limited routes through the state space. These transitions emulate an agent sending and receiving messages as part of a protocol. These test environments are more complex than those used in the other cited research into MARL algorithms with networked agents. Kar evaluated the QD learning algorithm with a simple binary-valued state-action space and a randomly sampled transition function [20]. Zhang used a more complex environment with 20 states; again, it had only two actions. They did not describe how they developed the transition functions [23].

The experiments use three sets of environments that vary in the number of states; they have 8, 16 and 32 states, each having two actions. There are no terminal states, so the experiments are continuing tasks. The reward for all states is zero, except for one, selected randomly, which returns a reward of one. From any state action, the probability of transitioning to any other state is non-zero. These state transitions are generated randomly, but the transition probability is increased fifty-fold for one state, and then the probabilities are normalised. This fifty-fold increase creates an environment where the agent has to pass through a series of states to get a reward.

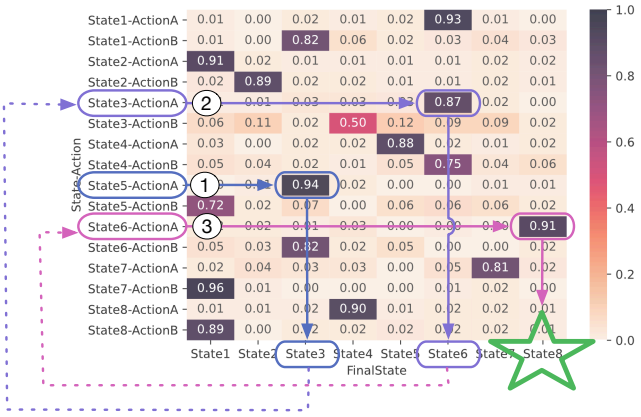


Figure 3: Heatmap of the transition function probabilities for the environment with eight states and seed set to zero.

Figure 3 illustrates this process. It shows a heatmap of the transition probabilities for an environment with eight states and two actions in each state. It is the transition function generated for $seed = 0$. The reward state was *State 8*, as indicated with the green star. Overlaid on the heatmap is an example of a route from the starting state, *State 5*. The agent takes *Action A*, which transitions the environment to *State 3*. Then, the agent retakes *Action A*, which transitions the environment to *State 6*. Lastly, the agent takes *Action A* for a third time, and the environment transitions to *State 8*, and the agent receives the reward. As this is a continuous task, as opposed to an episodic task, the agent is never reset to an initial state. After receiving the reward, it takes actions, either *Action A* or *Action B*, until the experiment ends. It receives the reward every time it transitions to *State 8*.

Dynamic programming (DP) can generate estimates of the true AVF when an environment's state transition function and reward function are known. The DP algorithm has two parameters: a discount factor, γ, and an allowed difference threshold, θ. Figure 4 shows the AVF calculated using DP for an environment with eight states and $seed = 0$. The experiments set these to 0.9 and 0.001, respectively. The evaluation of an RL algorithm compares the algorithm's AVF with the estimate of the AVF generated using DP.

The root-mean-squared error (RMSE) is the root of the mean of the squared errors of each state-action value for each agent. A second measure, the accumulative root-mean-squared error (ARMSE), measures the convergence speed of an RL algorithm. It is the accumulation of the RMSE at each episode.

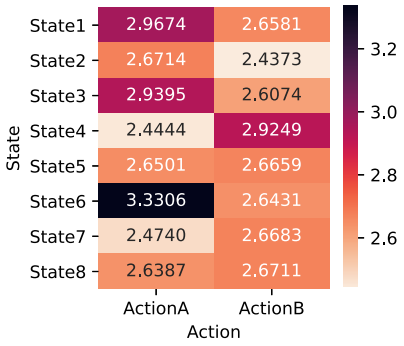


Figure 4: Heatmap of the action value function calculated using dynamic programming for the environment with eight states and seed set to zero.

Table 2: The parameters used for each learning algorithm.

Algorithm	Parameter	Values
	Discount factor, γ	0.9
Q Learning	Step size, α	0.05, 0.025, 0.01, 0.005
Q(λ) Learning	Step size, α	0.05, 0.025, 0.01, 0.005
	Eligibility decay, λ	0.3, 0.6, 0.9
QD Learning	Innovation step size, α	0.05, 0.025, 0.01, 0.005
	Consensus step size, β	0.005, 0.0025, 0.00125
QD(λ) Learning	Innovation step size, α	0.05, 0.025, 0.01, 0.005
	Consensus step size, β	0.005, 0.0025, 0.00125
	Eligibility decay, λ	0.3, 0.6, 0.9

4 EXPERIMENTAL RESULTS

The experiments evaluate the performance of the QD(λ) learning algorithm against three well-known value-based learning algorithms. Two of these algorithms, Q learning and Q(λ) learning, employ the fully decentralised scheme, while one, QD learning, uses the networked agent's scheme.

One or more parameters configure the RL algorithms. Changes to the parameters will impact the performance of a learning algorithm in an environment. There is no correct configuration of these parameters. However, there is an optimal configuration of the parameters for a given environment. Thus, the experiments evaluated the RL algorithms across several different configurations, as shown in Table 2. Note that every combination of the parameters listed is evaluated when there is more than one parameter. For example, for the Q(λ) learning, there were 12 different parameter combinations evaluated.

The experiments had 50 agents in the multi-agent system. The agents had a soft policy with the probability of taking a random, rather than greedy, action, ϵ, set to 0.1. Each agent takes an action once per cycle. For every episode, 10,000 cycles, the experiments recorded the state action values and their error from the estimated AVF. Note the problem was not reset to an initial state at the start of an episode; hence these experiments represent continuing tasks. Each experiment comprised 200 episodes. Table 3 summarises this experimental configuration.

Table 3: The configuration used for the experiments.

Parameter	Value
Number of Agents, N	50
Probability of taking a random action, ϵ	0.1
Learning cycles per episode	10,000
Number of episodes	200

The experiments use three different environment types with 8, 16 and 32 states. Each was repeated 40 times with different random seeds. A Shapiro–Wilk test of normality [26] showed that the results did not follow a normal distribution; thus, non-parametric statistical measures are used. For example, the results of two different RL algorithms were compared using a two-sample Wilcoxon Signed test [27].

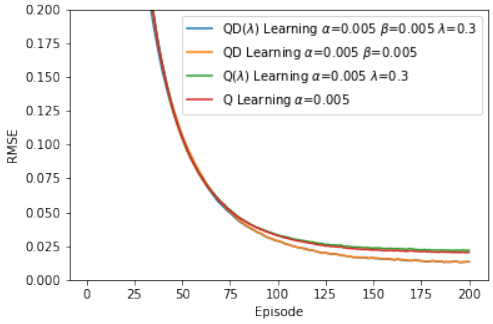
Figure 5 shows the AVF RMSE and ARMSE for the best configurations of each of the four learning algorithms for the three different types of environments. Table 4 shows the AVF median RMSE and ARMSE across 40 random seeds for episodes 50 and 200 for the best configurations of the four learning algorithms for three different types of environments. The p-values indicate the statistical significance of the difference between an algorithm and the best algorithm.

The results show that QD learning has the lowest AVF median RMSE across all the environments in the 200th episode. However, QD(λ) learning has the lowest RMSE across all environments in the 50th episode. Furthermore, QD(λ) learning has the lowest AVF median ARMSE across all the environments in the 200th episode. All these differences are statistically significant. These results support the hypothesis that the QD(λ) learning algorithm learns faster than the other three RL algorithms.

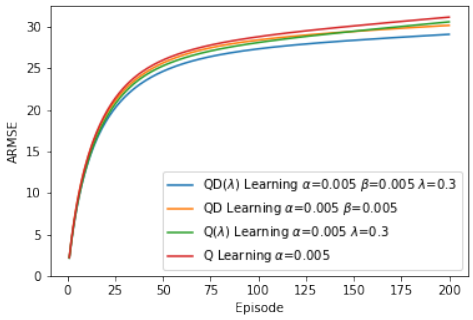
5 DISCUSSION

The main contribution of this research is the design of a new value-based MARL algorithm that leverages networked agents, the QD(λ) learning algorithm. The experiments did not show that the QD(λ) learning algorithm has the lowest overall median AVF root-mean-squared error (RMSE). Indeed, they showed that QD learning has a lower RMSE and that the difference is statistically significant. However, the results did show that the QD(λ) learning algorithm gets the lowest median AVF accumulative root-mean-squared error (ARMSE).

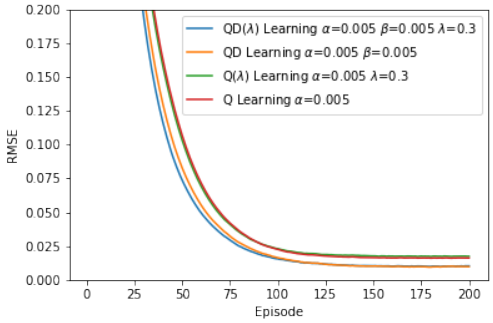
Based on this experimental evidence, we propose a general hypothesis that the QD(λ) learning algorithm can learn faster than the other RL algorithms for environments where the agent must transition through many states before it receives a reward. While the experimentation supports the theory, it has not been proven. A mathematical proof of the hypothesis is outside the scope of this research. Although there is no extra network traffic for the QD(λ) algorithm over the QD learning algorithm, it does require additional computation. It does learn faster than the other RL algorithms for the test environments, but the difference is slight. Is



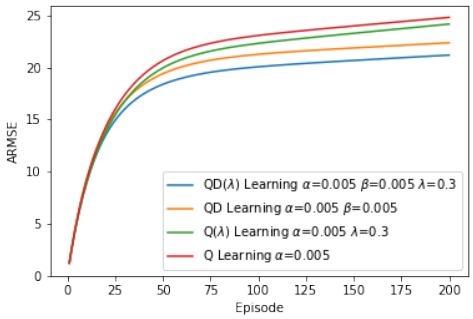
(a) RMSE Number of states = 8



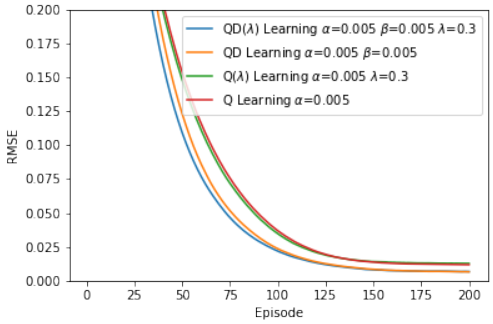
(b) ARMSE Number of states = 8



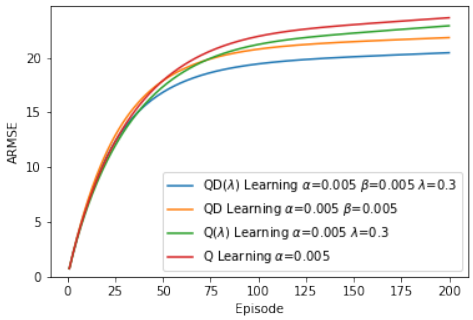
(c) RMSE Number of states = 16



(d) ARMSE Number of states = 16



(e) RMSE Number of states = 32



(f) ARMSE Number of states = 32

Figure 5: The action-value function root-mean-squared error (RMSE) and accumulative root-mean-squared error (ARMSE) for the best configurations of each of the four learning algorithms for four different types of environments.

the extra computation worth the effort for a minor improvement in learning speed?

This research and experimentation focused on developing a MARL algorithm to facilitate communication protocol learning. The approach assumes the agents in the MAS are identical and their learned behaviour is interchangeable. However, the agents' incentives are not aligned in many decentralised systems. For these cases, the transfer learning within QD(λ) would be prohibitive rather than beneficial to the learning process.

The experiments used parameter sweeps to evaluate the model across different configurations. In total, the evaluations used 64 configurations across the four RL algorithms. The best configuration for QD(λ) learning was $\alpha = 0.005$, $\beta = 0.005$ and $\lambda = 0.3$. However, we should not extrapolate this finding to other environments with different transition functions, rewards or discounts (γ).

Ideally, the parameter sweep would ensure that the parameter values of the best configurations were not at the limits of the sweep. For example, the parameter sweep of step size, α, ranges from 0.05 to 0.005, with the best configuration for all four RL algorithms being $\alpha = 0.005$; this is at the limit of the parameter sweep. Would $\alpha = 0.0025$ have gotten better results for any RL algorithm? While this is likely the case, as a smaller step size results in a lower RMSE,

Table 4: The action-value function median root-mean-squared error (RMSE) and accumulated root-mean-squared error (ARMSE) across 40 random seeds for episodes 50 and 200 for the best configurations of the four learning algorithms for three different types of environments.

		Episode = 50		Episode = 200		Episode = 50		Episode = 200	
Number of states = 8	Configuration	RMSE	p-value	RMSE	p-value	ARMSE	p-value	ARMSE	p-value
QD(λ) Learning	α=0.005 β=0.005 λ=0.3	0.0446	$< 10^{-4}$	0.0074	0.0139	19.9183	-	22.3943	-
QD Learning	α=0.005 β=0.005	0.0461	-	0.0070	-	20.3954	$< 10^{-4}$	23.6829	0.0001
Q(λ) Learning	α=0.005 λ=0.3	0.0534	$< 10^{-4}$	0.0173	$< 10^{-4}$	20.9529	0.0201	25.5197	0.0075
Q Learning	α=0.005	0.0524	$< 10^{-4}$	0.0159	$< 10^{-4}$	21.8806	0.0738	26.5480	0.0007
Number of states = 16	Configuration	RMSE	p-value	RMSE	p-value	ARMSE	p-value	ARMSE	p-value
QD(λ) Learning	α=0.005 β=0.005 λ=0.3	0.0553	$< 10^{-4}$	0.0064	0.0011	18.2678	-	20.3043	-
QD Learning	α=0.005 β=0.005	0.0594	-	0.0059	-	19.2181	$< 10^{-4}$	20.7923	$< 10^{-4}$
Q(λ) Learning	α=0.005 λ=0.3	0.0808	$< 10^{-4}$	0.0151	$< 10^{-4}$	20.8282	0.0294	23.9693	$< 10^{-4}$
Q Learning	α=0.005	0.0956	$< 10^{-4}$	0.0135	$< 10^{-4}$	22.3586	0.5453	25.9672	$< 10^{-4}$
Number of states = 32	Configuration	RMSE	p-value	RMSE	p-value	ARMSE	p-value	ARMSE	p-value
QD(λ) Learning	α=0.005 β=0.005 λ=0.3	0.0996	$< 10^{-4}$	0.0049	0.0315	15.8704	-	18.9138	-
QD Learning	α=0.005 β=0.005	0.1111	-	0.0048	-	16.5807	$< 10^{-4}$	19.6555	$< 10^{-4}$
Q(λ) Learning	α=0.005 λ=0.3	0.1398	$< 10^{-4}$	0.0116	$< 10^{-4}$	16.4933	$< 10^{-4}$	21.3227	$< 10^{-4}$
Q Learning	α=0.005	0.1498	$< 10^{-4}$	0.0110	$< 10^{-4}$	17.1567	$< 10^{-4}$	22.3191	$< 10^{-4}$

it may not have achieved better results within 200 episodes, as there was insufficient time for the algorithm to converge. But extra episodes may result in the configuration of $\alpha = 0.0025$ being the best. As the best step size is again at the limit of the parameter sweep, the earlier question remains valid — would an even smaller step size get even better results for one of the RL algorithms? A similar cyclical argument applies to the eligibility decay parameter, λ and the consensus step size, β, which are also at the limit of the parameter sweep.

Two observations from the experimental results are consistent with previous findings in the literature. The first concerns Q(λ) learning where Sutton et al. reported *"methods using eligibility traces… offer significantly faster learning, particularly when rewards are delayed by many steps"* [28]. The second concerns Q(λ) learning where Tan stated that "agents engaging in partnership … may learn slowly in the beginning" [10]. Our experimental results are consistent with Sutton and Tan's assertions.

6 CONCLUSION & FURTHER WORK

This research aimed to develop a MARL algorithm which overcomes the limitation reported by Tan [10], that MARL algorithms with networked agents "may learn slowly in the beginning". Additionally, we aspire to design a MARL algorithm suited to a sizeable multi-agent system learning a communication protocol.

The main contribution is the QD(λ) learning algorithm — a novel value-based MARL algorithm that leverages networked agents. It incorporates the advantages of transfer learning from the MARL QD learning [20] algorithm and the faster learning when rewards are delayed from the RL Q(λ) Learning algorithm [19]. We evaluated the QD(λ) learning algorithm in specially designed RL environments that require agents to transition through several states before receiving a reward. The QD(λ) learning algorithm achieved the lowest accumulative root-mean-squared error, thus overcoming the finding by Tan that "agents engaging in partnership … may learn slowly in the beginning" [10].

The empirical results support the hypothesis that the QD(λ) learning algorithm can learn faster than the other RL algorithms for environments where the agent must transition through many states before it receives a reward. Thus, QD(λ) learning is suited to a sizeable multi-agent system learning a communication protocol. This emerging research area has seen significant activity in the last few years [12–16, 18].

A natural progression of this work is to evaluate the QD(λ) learning algorithm's effectiveness at learning a communication protocol. In previous research, the authors developed Gossip Contracts, a communication protocol for networked multi-agent systems to facilitate decentralised cooperation strategies [29]. The QD(λ) learning algorithm could augment the Gossip Contracts protocol and continually learn the best protocol to address a specific problem. This use of QD(λ) learning is akin to how the Evolved Gossip Contracts framework leverages genetic programming to search for the implementation of the GC methods to tailor it to a problem [30].

7 ACKNOWLEDGEMENTS

The authors wish to acknowledge the DJEI/DES/SFI/HEA Irish Centre for High-End Computing (ICHEC) for the provision of computational facilities and support.

REFERENCES

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
[2] Yoav Shoham, Rob Powers, and Trond Grenager. Multi-agent reinforcement learning: a critical survey. Technical report, Technical report, Stanford University, 2003.
[3] Jiarui Zhang, Gang Wang, and Yafei Song. Task assignment of the improved contract net protocol under a multi-agent system. *Algorithms*, 12(4):70, 2019.

[4] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

[5] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

[6] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[7] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer, 2000.

[8] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69. IEEE, 2007.

[9] Siliang Zeng, Xingfei Xu, and Yi Chen. Multi-agent reinforcement learning for adaptive routing: A hybrid method using eligibility traces. In *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, pages 1332–1339. IEEE, 2020.

[10] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[11] Matthew E Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(9), 2007.

[12] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[13] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29, 2016.

[14] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *Advances in neural information processing systems*, 31, 2018.

[15] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, pages 1538–1546. PMLR, 2019.

[16] Hangyu Mao, Zhengchao Zhang, Zhen Xiao, Zhibo Gong, and Yan Ni. Learning multi-agent communication with double attentional deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 34:1–34, 2020.

[17] Simon Vanneste, Astrid Vanneste, Kevin Mets, Tom De Schepper, Ali Anwar, Siegfried Mercelis, Steven Latré, and Peter Hellinckx. Learning to communicate using counterfactual reasoning. *arXiv preprint arXiv:2006.07200*, 2020.

[18] Tze-Yang Tung, Joan Roig Pujol, Szymon Kobus, and D Gunduz. A joint learning and communication framework for multi-agent reinforcement learning over noisy channels. *arXiv preprint arXiv:2101.10369*, 2021.

[19] C J C H Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[20] Soummya Kar, José MF Moura, and H Vincent Poor. Qd-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus. *arXiv preprint arXiv:1205.0047*, 2012.

[21] Paulina Varshavskaya, Leslie Pack Kaelbling, and Daniela Rus. Efficient distributed reinforcement learning through agreement. In *Distributed Autonomous Robotic Systems 8*, pages 367–378. Springer, 2009.

[22] Adwaitvedant Mathkar and Vivek S Borkar. Distributed reinforcement learning via gossip. *IEEE Transactions on Automatic Control*, 62(3):1465–1470, 2016.

[23] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In *International Conference on Machine Learning*, pages 5872–5881. PMLR, 2018.

[24] Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. Networked multi-agent reinforcement learning in continuous spaces. In *2018 IEEE conference on decision and control (CDC)*, pages 2771–2776. IEEE, 2018.

[25] Joshua Hare. Dealing with sparse rewards in reinforcement learning. *arXiv preprint arXiv:1910.09281*, 2019.

[26] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[27] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[28] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[29] Nicola Mc Donnell, Enda Howley, and Jim Duggan. Dynamic virtual machine consolidation using a multi-agent system to optimise energy efficiency in cloud computing. *Future Generation Computer Systems*, 108:288–301, 2020.

[30] Nicola Mc Donnell, Enda Howley, and Jim Duggan. Evolved gossip contracts-a framework for designing multi-agent systems. In *International Conference on Parallel Problem Solving from Nature*, pages 637–649. Springer, 2020.