Addressing the Policy-bias of Q-learning by Repeating Updates[*]

Sherief Abdallah
the British University in Dubai, United Arab Emirates
University of Edinburgh, United Kingdom
shario@ieee.org

Michael Kaisers
Maastricht University, Netherlands
Michael.Kaisers@maastrichtuniversity.nl

ABSTRACT

Q-learning is a very popular reinforcement learning algorithm being proven to converge to optimal policies in Markov decision processes. However, Q-learning shows artifacts in non-stationary environments, e.g., the probability of playing the optimal action may decrease if Q-values deviate significantly from the true values, a situation that may arise in the initial phase as well as after changes in the environment.These artifacts were resolved in literature by the variant Frequency Adjusted Q-learning (FAQL). However, FAQL also suffered from practical concerns that limited the policy subspace for which the behavior was improved. Here, we introduce the Repeated Update Q-learning (RUQL), a variant of Q-learning that resolves the undesirable artifacts of Q-learning without the practical concerns of FAQL. We show (both theoretically and experimentally) the similarities and differences between RUQL and FAQL (the closest state-of-the-art). Experimental results verify the theoretical insights and show how RUQL outperforms FAQL and QL in non-stationary environments.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*

Keywords

Q-learning, Non-stationary Environment, Dynamics

1. INTRODUCTION

Q-Learning is one of the most widely-used and widely-studied learning algorithms due to its ease of implementation, intuitiveness, and effectiveness over a wide-range of problems [7]. Although Q-learning was originally designed for single-agent domains, Q-learning was also used (with reasonable success) in multi-agent settings [3]. Even the gradient-based multi-agent learning algorithms still relied on Q-learning as an internal component to estimate the values of different actions [1, 2, 9].

Despite the popularity of Q-learning, advances in the formal analysis of Q-learning in strategic interactions have only been made recently [4, 8]. Theoretical analysis showed that Q-learning in some cases exhibits behavior in the policy space that (temporarily) decreased the probability of playing actions with higher expected payoffs. This counter-intuitive behavior is the result of updating only the selected action: highly promising actions are updated more often and are thus adjusted towards the correct value more quickly. For example, if all actions are optimistically overestimated then the more promising action loses its competitive edge and is played less rather than more. We refer to this artifact as the *policy-bias* of Q-learning. This policy-bias may be insignificant for single-agent learning, but it creates a cascading effect in multi-agent interactions, where the behavior of one agent influences the learning of other agents [4, 8]. In addition, in non-stationary environments this updating scheme may take many iterations to update the estimators of non-optimal actions to their true value. This delayed response to changes in the environment results in an inertia effect whenever the optimal action changes.

The policy-bias problem occurs because an agent can execute and get feedback for only one action at any time step. If, hypothetically, the reward information for all actions (at every state) were available at every time step, then the update rule would be applied to all actions and the mismatch between theoretical and actual behavior would disappear. Recently, a simple yet effective modification to the Q-learning update equation was proposed: the Frequency-Adjusted Q-learning (FAQL) [4]. FAQL addressed the policy-bias of Q-learning by normalizing the update value of an action with respect to the probability of choosing that action. In other words, the less frequently-selected an action is, the more boost FAQL will give to the update of the action. The previously reported results (which we reproduce here in Section 5 for benchmarking) show more stable and consistent dynamics for FAQL when compared to QL [4]. However, FAQL also suffered from practical concerns that limited the policy subspace for which the behavior was improved. FAQL is discussed in detail in Section 2.

The algorithm we propose here, RUQL, addresses both the bias of Q-learning and the practical concerns faced by FAQL. RUQL repeats the update of an action inversely proportional to the probability of choosing that action. For example, if an action is to be chosen for execution only 10% of the time, then the update of that action (when finally chosen) shall be repeated 1/0.1 or 10 times. As a result, every action will, in expectation, be updated an equal number of times. We show in Section 3 that RUQL has a nice closed-form expression that removes the need to actually repeat the updates. We show both theoretically and experimentally that the dynamics of RUQL are consistent and very comparable to the dynamics of FAQL (which are the idealized dynamics of QL). Furthermore, our proposed algorithm outperforms both QL and FAQL in non-stationary settings (over wide-range of parameter values).

[*]The work reported in this paper was supported in part by Emirates Foundation grant #2010/107 , Program: Science and Engineering Research Grants and by the British University in Dubai

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

In summary, the main contributions of this paper are:

- A new reinforcement learning algorithm, RUQL, which builds on the Q-learning algorithm but does not suffer from the policy-bias problem of QL nor does it suffer from the practical concerns of FAQL.
- A mathematical derivation of a closed-form expression for RUQL that makes the RUQL algorithm computationally feasible.
- Theoretical analysis that shows the similarities and differences between RUQL and the most-related state-of-the-art algorithm (FAQL).
- Experimental analysis that confirms the theoretical analysis and compares the performance of RUQL to QL and FAQL.

The following section presents necessary background about Q-learning and the related algorithms. We then introduce the algorithm, RUQL, followed by theoretical and experimental analysis. Finally, a summary and a discussion of the contributions conclude the article.

2. Q-LEARNING

Q-learning, as we mentioned earlier, is one of the most widely-used learning algorithms [7]. The algorithm is simple and relies primarily on the following update equation:

$$Q^{t+1}(s,a) = Q^t(s,a) + \alpha\left(r + \gamma\, maxQ^t(s,a') - Q^t(s,a)\right) \tag{1}$$

The function $Q^t(s,a)$ represents the expected discounted reward at time t that the agent believes it would get if it executes action a at state s (or intuitively, what the agent believes, at time t, to be the worth of action a at state s). The parameters α and γ are tunable learning parameters (α is called the learning rate and γ is called the discount factor). The variables r and s' refer to the immediate reward and the next state (respectively) after executing action a at state s. Algorithm 1 illustrates how the Q-learning update equation is typically used in practice.

Algorithm 1: Q-learning Algorithm

```
1.1  begin
1.2      Initialize function $Q$ arbitrarily.
1.3      Observe the current state $s$.
1.4      repeat
1.5          Compute the policy $\pi(s,a)$ using $Q(s,a)$.
1.6          Choose an action $a$ according to agent policy $\pi(s,a)$.
1.7          Execute action $a$ and observe the resulting reward $r$
             and the next state $s'$.
1.8          Update $Q(s,a)$ using Equation 1.
1.9          Set $s \leftarrow s'$.
1.10     until done
1.11 end
```

The function Q^t in itself does not specify explicitly which action should an agent choose in a given state. The policy of an agent, $\pi(s,a)$, is a function that specifies the probability of choosing action a at state s. A (naive) definition of $\pi^t(s,a) = argmax_{a'} Q^t(s,a')$ can result in arbitrarily bad policy, because the optimal action may (initially) have low value of Q. Instead, several definitions of π were proposed that ensured all actions (including actions that may appear sub-optimal) are selected with non-zero probability. These definitions of π are often called *exploration* strategies [7]. The two most common exploration strategies are the ϵ-greedy exploration and the Boltzmann exploration. We use in our experiments the Boltzmann exploration which defines the policy $\pi(s,a)$ as follows:

$$\pi^t(s,a) = \frac{e^{\frac{Q^t(s,a)}{\tau}}}{\sum_{a'} e^{\frac{Q^t(s,a')}{\tau}}}$$

where the tunable parameter τ is called the temperature. Ideally, an agent should update the value of each action at every state (i.e. $Q(s,a)\forall s,a$) at every time step. However, since an agent can only execute one action at a time and the agent will only receive feedback (reward) for the action that was actually executed, this ideal scenario can not be realized.[1] As a result, in Q-learning, the rate of updating an action relies on the probability of choosing that action. The theoretical analysis of this limitation showed mismatch between the theoretical dynamics of Q-learning and the obtained actual dynamics [4, 8].

The Frequency Adjusted Q-learning algorithm (FAQL) was recently proposed to overcome this limitation by scaling the learning rate inversely proportional to the policy. More formally, FAQL modified the Q-learning update rule to be [4]:

$$\begin{aligned} Q^{t+1}(s,a) &= Q^t(s,a) + \frac{1}{\pi(s,a)}\alpha\left(r + \gamma\, max_{a'} Q^t(s',a')\right. \\ &\quad \left. -Q^t(s,a)\right) \end{aligned}$$

This nice and simple modification to the Q-learning update rule suffered from a practical concern: the update rate becomes unbounded (approaches ∞) as $\pi(s,a)$ approaches zero. Therefore a safe-guard condition had to be added [4]:

$$\begin{aligned} Q^{t+1}(s,a) &= Q^t(s,a) + min(1, \frac{\beta}{\pi(s,a)})\alpha\,(r+ \\ &\quad \gamma\, max_{a'} Q^t(s',a') - Q^t(s,a)) \end{aligned} \tag{2}$$

where β is a tuning parameter that safeguards against the cases where $\pi(s,a)$ is close to zero. The resulting algorithm is similar to Algorithm 1 but with Line 1.8 modified to use Equation 2 instead of Equation 1.

While introducing parameter β does make FAQL practically implementable, the introduction of β also results in two undesirable properties. The first undesirable property is reducing the effective learning rate (instead of α it is now $\alpha\beta$) and therefore reducing the responsiveness of FAQL.[2] The second and more important undesirable property is what we refer to as the β-limitation. Once the probability of choosing an action, $\pi(s,a)$, goes below β, FAQL becomes identical to the original QL. In other words, FAQL suffers from the same policy-bias problem when the policy π becomes lower than β. This is problematic because the lower the probability of choosing an action is, the more important it is to adjust the

[1]One possible approximation to having this full reward information is to keep track of the most recent reward received for every state-action pair. A learning algorithm can then update the value of every state-action pair at every time-step using the most recent reward. This approximation, however, quickly becomes intractable as the number of states and actions increase (since all state-action pairs need to be updated at every time step, not just the pair that was recently encountered).

[2]In our experiments we had to take the effective learning rate into account when we compared FAQL to our algorithm RUQL, setting $\alpha_{RUQL} = \alpha_{FAQL}\beta$.

learning rate (to account for the infrequency of choosing that action). Our proposed algorithm RUQL does not suffer from these limitations and we show in the experiments how this can improve performance in non-stationary settings.

3. THE REPEATED-UPDATE Q-LEARNING, RUQL

Our proposed algorithm is based on a simple intuitive and idea. If an action is chosen with low probability $\pi(s,a)$ then instead of updating the corresponding action value $Q(s,a)$ once, we repeat the update $\frac{1}{\pi(s,a)}$ times. Algorithm 2 shows the naive implementation of this idea. Line 2.8 is the only difference between Algorithm 2 and 1.

Algorithm 2: RUQL (Impractical Implementation

```
2.1  begin
2.2      Initialize function $Q$ arbitrarily.
2.3      Observe the current state $s$.
2.4      repeat
2.5          Compute the policy $\pi$ using $Q$.
2.6          Choose an action $a$ according to agent policy $\pi$.
2.7          Execute action $a$ and observe the resulting reward $r$
             and the next state $s'$.
2.8          for $\lfloor \frac{1}{\pi(s,a)} \rfloor$ times do
2.9              Update $Q(s,a)$ using Equation 1.
2.10         end
2.11         Set $s \leftarrow s'$.
2.12     until done
2.13 end
```

This simple modification to the QL algorithm addresses the policy-bias but has a serious drawback. As $\pi(s,a)$ gets lower the number of repetitions increases and quickly becomes unbounded as $\pi(s,a)$ approaches 0. Unlike FAQL, here the computation *time* (not the *value* of the update) is what becomes unbounded as $\pi(s,a)$ approaches zero. In the remainder of this section we will derive a closed-form expression for the RUQL algorithm. The closed-form expression removes the need for actually repeating any update and therefore makes RUQL computationally feasible.

First, we expand the recursion in the Q-learning equation according to Steps 2.8-2.10 in Algorithm 2:

$$\begin{aligned}
Q^{t+1}(s,a) &= [1-\alpha]Q^t(s,a) + \alpha[r + \gamma\, max_{a'} Q^t(s',a')] \\
&= [1-\alpha]\left([1-\alpha]Q^{t-1}(s,a) + \alpha[r+ \right. \\
&\quad \left. \gamma\, max_{a'} Q^{t-1}(s',a')]\right) + \alpha[r + \gamma \\
&\quad max_{a'} Q^t(s',a')] \\
&= [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor} Q^{t-\lfloor \frac{1}{\pi(s,a)} \rfloor}(s,a) + \Big(\\
&\quad \alpha[r + \gamma\, max_{a'} Q^t(s',a')] \\
&\quad + [1-\alpha]\alpha[r + \gamma\, max_{a'} Q^{t-1}(s',a')] \\
&\quad ... \\
&\quad + [1-\alpha]^{t-\lfloor \frac{1}{\pi(s,a)} \rfloor} \alpha[r + \gamma\, max_{a'} \\
&\quad Q^{t-\lfloor \frac{1}{\pi(s,a)} \rfloor}(s',a')]\Big)
\end{aligned}$$

To simplify the above equation, we assume that the value $max_{a'}$ $Q^{t-i}(s',a')$ is resolved once for $\lfloor 1/\pi(s,a) \rfloor \geq i \geq 1$. In other words, we assume the maximum value at a given state, $max_{a'} Q(s,a')$, does not significantly change between repeated updates. Such an assumption is reasonable with a sufficiently small learning rate α.[3] With this assumption we get:

$$\begin{aligned}
Q^{t+1}(s,a) &= [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor} Q^t(s,a) + \alpha[r + \\
&\quad \gamma Q^t(s',a_{max})]\left(1 + (1-\alpha) + \right. \\
&\quad \left. (1-\alpha)^2 + ... + (1-\alpha)^{\lfloor \frac{1}{\pi(s,a)} \rfloor}\right) \\
&= [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor} Q^t(s,a) + \alpha[r + \\
&\quad \gamma Q^t(s',a_{max})] \frac{[1-(1-\alpha)^{\lfloor \frac{1}{\pi(s,a)} \rfloor}]}{1-(1-\alpha)} \\
&= [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor} Q^t(s,a) + [1 - \\
&\quad (1-\alpha)^{\lfloor \frac{1}{\pi(s,a)} \rfloor}][r + \gamma Q^t(s',a_{max})]
\end{aligned}$$

We can also remove the floor notation for a better generalization, and thus we end up with RUQL's main update rule:

$$\begin{aligned}
Q^{t+1}(s,a) &= [1-\alpha]^{\frac{1}{\pi(s,a)}} Q^t(s,a) + \\
&\quad [1-(1-\alpha)^{\frac{1}{\pi(s,a)}}][r + \gamma Q^t(s',a_{max})] \quad (3)
\end{aligned}$$

This equation can then replace Lines 2.8-2.10 in Algorithm 2 to produce the efficient implementation of RUQL. The equation of RUQL, as Section 5.1 verifies and the following section discusses, changes the dynamics of learning by giving more weight to more recent reward samples for actions with *low probability* of being chosen.

4. THEORETICAL ANALYSIS

Before presenting the theoretical aspects of RUQL, it is worthwhile to first clarify, intuitively, why RUQL's update rule makes sense. If the probability of choosing an action a approaches 1 (the action is selected almost always), then RUQL's update equation reduces to the original Q-learning update equation (Equation 1). This reduction makes sense because for an action that is selected all the time, Q-learning update equation works perfectly fine. On the other hand, if $\pi(s,a)$ approaches 0, then the term $(1-\alpha)^{\frac{1}{\pi(s,a)}}$ approaches zero as well (because $(1-\alpha) < 1$). In this case RUQL's update equation reduces to $r + \gamma Q^t(s',a_{max})$. In other words, as $\pi(s,a)$ decreases, RUQL places more weight on the sample obtained $[r + \gamma Q^t(s',a_{max})]$ rather than the current expectation $Q(s,a)$. Giving more weight to the sample makes sense, because the smaller $\pi(s,a)$ gets the more out-of-date $Q(s,a)$ becomes (due to the infrequent updates).

Both RUQL and FAQL provide an algorithmic implementation to address the policy-bias of Q-learning. However, two algorithms resolve this problem in different ways: FAQL normalized the learning rate of the Q-learning update while RUQL repeats Q-learning update rule. In the remainder of this section we analyze the con-

[3]The assumption simplifies the update and the dynamics for the case $s' = s$ and $a' = a$ (otherwise it does not make a difference). If $s' = s$ and $a' = a$, then $Q(s',a')$ changes from a repeated update to the next. In fact, without this assumption we also need to consider the switch between the cases during the repeated updates. For example, when an action that is being updated is not the best action initially but becomes the best action during the repeated updates.

nection between RUQL and FAQL (the closest state-of-the-art) to highlight their similarities and differences.

Both FAQL and RUQL yield equivalent behavior in the limit of an infinitesimal learning rate α. Consider the term $(1-\alpha)^{\frac{1}{\pi(s,a)}}$ in RUQL's update equation. This term can be expanded using the Taylor series expansion at $\alpha = 0$ and we get $1 - \frac{\alpha}{\pi(s,a)} - O(\alpha^2)$. For small values of α, the higher order terms are negligible and RUQL's update equation reduces to the ideal (without the safeguard) equation of FAQL.

The advantage of RUQL is that the update rule does not need the additional β parameter and therefore RUQL does not suffer from the two limitations that were described in Section 2 (particularly the β-limitation). FAQL on the other hand has the advantage that it behaves in expectation according to the idealized dynamical model of Q-learning (without the policy-bias), even for learning rates that are not infinitesimal (as long as the probability of choosing an action is larger than β). This is not the case for RUQL. However, the deviations of RUQL are bounded by an $O(\alpha^2)$ term, which for realistic small learning rates becomes negligible. More importantly, the deviations of RUQL are *independent* of the policy, unlike FAQL.

5. EXPERIMENTAL ANALYSIS

Our experimental analysis consists of three parts. The first two parts focus on verifying the theoretical arguments we have made in the previous section using simple and small-scale domains (one or two agents), while the the third part provides evaluation in a large-scale domain. The first part (Section 5.1) aims at verifying the equivalence of RUQL and FAQL for sufficiently small values of the learning rate α. The second part of the experiments (Section 5.2) uses a variation of the multi-armed-bandit problem to expose the effect of the β-limitation and how it separates RUQL from FAQL (which in turn results in RUQL outperforming both FAQL and QL). In the third part we evaluate the three algorithms using the social learning domain with hundreds of agents. In all three scenarios each agent learns Q values for a single state (i.e. we do not consider multi-state learning).

5.1 Prisoner's Dilemma Game

This section compares the (experimental) dynamics of Q-learning, FAQL, and RUQL in the Prisoner's Dilemma game (PD). Figure 1 shows the dynamics of the three algorithms for three different initial value levels of Q. For all three algorithms the temperature (for the Boltzmann exploration) was set to 0.1. The learning rate α was set at 10^{-6} for both Q-learning and RUQL, while for FAQL both α and β were set to 10^{-3}. Table 1 shows the payoff values for the PD game. The above setting was used before in the literature [4] and we use the same setting here for benchmarking.

	c	d
c	5,5	0,6
d	6,0	1,1

Table 1: The prisoner's dilemma game.

From Figure 1, we can see that the dynamics of Q-learning and FAQL are consistent with the results reported before [4]. We can also see that the dynamics of both FAQ and RUQL are very similar and close to the expected (theoretical) Q-learning dynamics, despite the conceptually different update equations. RUQL update rule indeed change the dynamics of learning.

5.2 Multi-Armed Bandit With Switching Means

In order to tease out the differences between RUQL and FAQL, we propose a simple variation of the multi-armed bandit (MAB) problem. The original MAB is a single-agent problem where each agent can choose between two actions. Each action i follows a normal distribution with a mean μ_i and standard deviation σ_i, where $\mu_i \in \{0, 1\}$ and $\sigma_i \in \{0, 0.1, 0.2, 0.3, 0.4\}$. The modification we do here is *switching* the value of μ for the two actions every duration D. In other words, having $D = 10000$ means that from time 0 to time 9999 $\mu_0 = 0$ and $\mu_1 = 1$ then from time 10000 to time 19999 $\mu_0 = 1$ and $\mu_1 = 0$ and so on.

This simple modified domain allows us to study two desired properties of learning algorithms: responding (quickly) to genuine change in the payoff *mean* of a particular action (e.g. when an opponent changes its strategy) while resisting hasty response to stochastic payoff samples (e.g if the mean is not changing but the payoff has high variance).

For different values of D and σ we collect the average payoff over 100000 consecutive time steps, and then compute both the mean and the standard deviation (of this average payoff) across 20 independent simulation runs. Table 2 summarizes the results of the MAB domain (with switching μ) for QL, FAQL, and RUQL and with the following settings. The temperature τ is set to 0.2. The learning rate α was set to 0.0004 for both QL and RUQL, while it was set to 0.02 for FAQL (to account for the $\beta = 0.02$). The discount factor γ was set to 0.9.

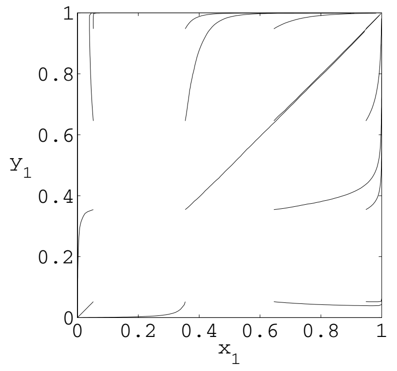
We can observe how RUQL compromises some stochastic stability for adaptability. Even when there is no switching in μ, the average payoff for $RUQL$ is 0.99 as opposed to the average payoff of 1 that was attained by both QL and $FAQL$. However, as the switching of μ takes place, the adaptability of $RUQL$ results in superior performance. For example, when the switching duration D is set to 10000, both QL and FAQL can not achieve more than 0.5 average payoff, while RUQL achieves 0.8 average payoff that slightly goes down with higher stochasticity (higher σ). It is also important to note the identical performance of FAQL to QL in this setting. The reason is the β-limitation, which was described earlier in Section 2.

Figure 2 illustrates dynamics in more detail by plotting the achieved payoff for the three algorithms over time (averaged over 1000 consecutive time steps and across 20 independent runs). Notice here that both QL and FAQL adapt so slowly to the switch in μ that they simply continue choosing the same action across the whole simulation run. On the other hand, RUQL adapts quickly to the change in μ.

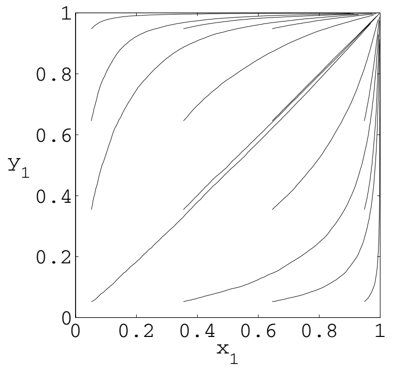
While the above settings of the learning rate ($\alpha = 0.0004$) and the discount factor ($\gamma = 0.9$) are typical, one has to wonder how sensitive the algorithms (QL, FAQL, and RUQL) are for different values of α and γ. Table 3 shows the average payoff for different values of α and for different values of D.[4] The deviation σ was set to 0.4 and $\gamma = 0.9$. The results for $D = 0$ were omitted because the results were very similar across different α values (similar to the first row of Table 2, but with RUQL slightly improving to reach 1). As we can see from the table, RUQL still achieves better performance across different values of α (particularly for $D = 10000$), but the gap in performance shrinks as α increases. Table 4 shows the average payoff for different values of γ and for different values of D. The deviation σ was kept at 0.4 while α was set to 0.0004.[5]

[4]The values of α in the table refer to the values used with QL and RUQL. For FAQL, and as we have mentioned before, we use $\alpha_{FAQL} = \sqrt{\alpha}$.

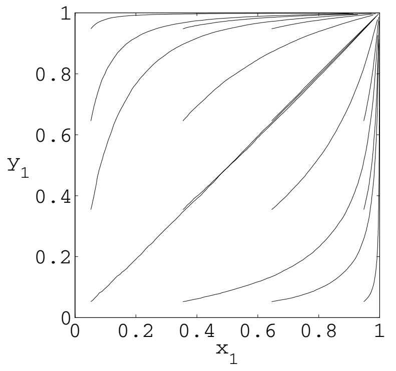
[5]Note that α_{FAQL} was set to 0.02.



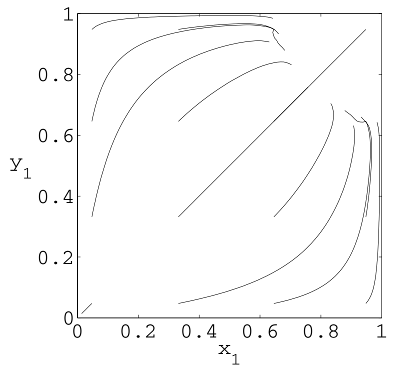
(a) Q-learning



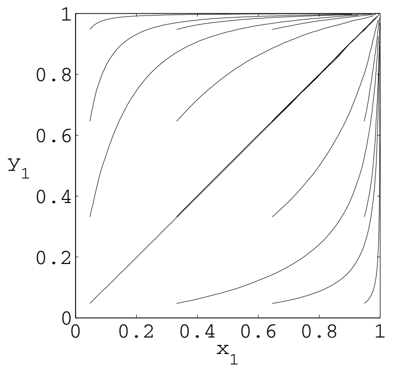
(b) FAQL



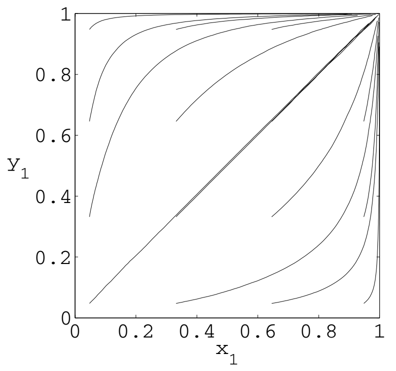
(c) RUQL



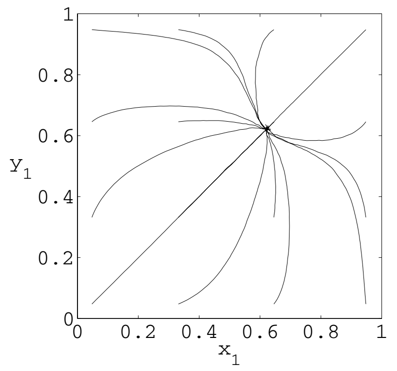
(d) Q-learning



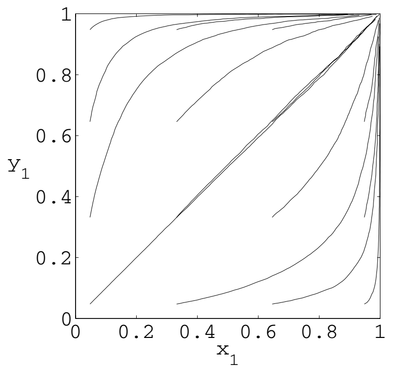
(e) FAQL



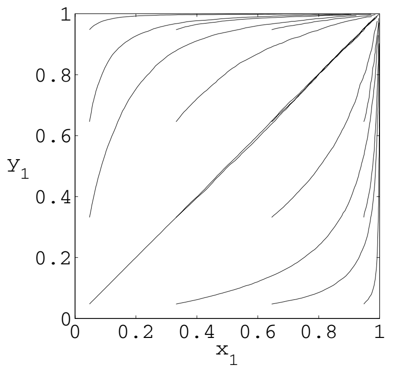
(f) RUQL



(g) Q-learning



(h) FAQL



(i) RUQL

Figure 1: The learning dynamics of Q-learning (left), FAQL (middle), and RUQL (right) for the PD game. Each figure plots the probabilty of cooperating for Player 1 (x-axis) against the probability of cooperation for Player 2 (y-axis). The top row shows the dynamics when the initial Q-values are around 0, the middle row shows the dynamics when the initial Q-values are around 2.5, and the bottom row shows the dynamics when the initial Q-values are around 5. For all figures, the algorithms were allowed to run for 500,000 time steps.

D \ σ	0	0.1	0.2	0.3	0.4
∞: QL	1 ± 10^{-5}	$1 \pm 3 * 10^{-4}$	$1 \pm 6 * 10^{-4}$	$1 \pm 8 * 10^{-4}$	1 ± 10^{-3}
FAQL	$1 \pm 2 * 10^{-5}$	$1 \pm 3 * 10^{-4}$	$1 \pm 6 * 10^{-4}$	$1 \pm 8 * 10^{-4}$	1 ± 10^{-3}
RUQL	$0.99 \pm 2 * 10^{-3}$	$0.99 \pm 2 * 10^{-3}$	$0.99 \pm 2 * 10^{-3}$	$0.99 \pm 2 * 10^{-3}$	$0.99 \pm 3 * 10^{-3}$
20000: QL	$0.6 \pm 2 * 10^{-4}$	$0.6 \pm 3 * 10^{-4}$	$0.6 \pm 6 * 10^{-4}$	$0.6 \pm 8 * 10^{-4}$	0.6 ± 10^{-3}
FAQL	$0.6 \pm 3 * 10^{-4}$	$0.6 \pm 3 * 10^{-4}$	$0.6 \pm 6 * 10^{-4}$	$0.6 \pm 8 * 10^{-4}$	0.6 ± 10^{-3}
RUQL	0.89 ± 0.05	0.88 ± 0.056	0.88 ± 0.056	0.88 ± 0.056	0.87 ± 0.06
10000: QL	$0.5 \pm 2 * 10^{-4}$	$0.5 \pm 3 * 10^{-4}$	$0.5 \pm 6 * 10^{-4}$	$0.5 \pm 8 * 10^{-4}$	0.5 ± 10^{-3}
FAQL	$0.5 \pm 2 * 10^{-4}$	$0.5 \pm 3 * 10^{-4}$	$0.5 \pm 6 * 10^{-4}$	$0.5 \pm 8 * 10^{-4}$	0.5 ± 10^{-3}
RUQL	0.80 ± 0.056	0.79 ± 0.06	0.79 ± 0.06	0.79 ± 0.06	0.78 ± 0.07

Table 2: The mean payoff (and standard deviation) for the MAB-switch game, for different values of D and σ and for the three algorithms QL, FAQL, and RUQL. RUQL achieves superior performance when $D < \infty$, with small deterioration as σ increases. The first and second columns from the left are consistent with the results reported earlier in the literature.

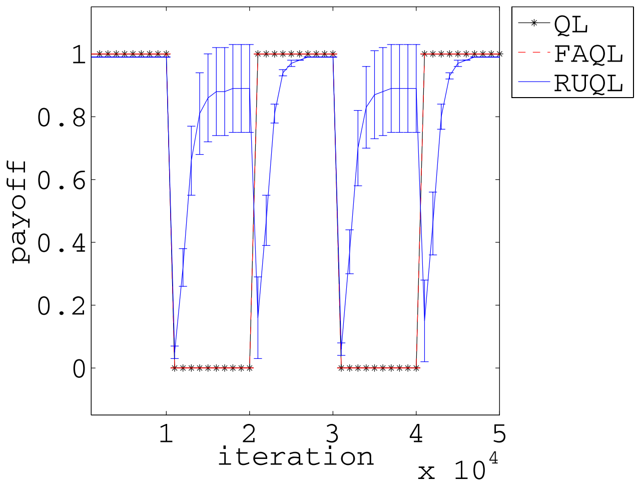


Figure 2: Average payoff of QL, FAQL, and RUQL for the MAB-switch domain, with $\alpha = 0.0004, \tau = 0.2, \sigma = 0.1, \gamma = 0.9$, and $D = 10000$. RUQL adapts reasonably well with the switching of μ while QL and FAQL have identical behavior due to the β-limitation.

Similar behavior can be observed: RUQL is superior for different values of γ but the gap shrinks as γ decreases.

5.3 Social Learning

In this section we compare RUQL to QL and FAQL in a social learning setting. Social learning studies how a group of agents interact and learn from one another to reach a *norm* [5]. A norm or a convention is an unwritten law that a society of agents agree on. In a multi-agent setting, a convention may refer to a dominant coordination strategy, a common communication language, or the right of way among a group of robots. Upon establishing a norm, the overhead of coordination drops and the reliability of the multi-agent system increases [6].The social learning process works as follows: a pair of agents are selected randomly from the population to interact with one another. The pair then learn from this interaction. The process repeats until the population reach convergence. The coordination game is the most widely used game for studying social learning as it presents an agent population with two equally plausible norms to choose from (i.e. two Nash equilibriums). Table 5 shows the coordination game that was used before as a benchmark [5].

	a	b
a	4,4	-1,-1
b	-1,-1	4,4

Table 5: The coordination game.

Here we evaluate social learning in a population of 225 agents playing the coordination game. Each agent was randomly initialized to one of the two conventions (i.e. the Q for one action was initialized to 1 while the Q of the other action was initialized to 0). Table 6 shows the average payoff that is collected from time step 5000 to time step 20000. The average is computed over 20 independent runs for different values of α and γ. The temperature τ was set to 0.2. Again we see here how RUQL performs generally better than QL and FAQL, but the gap in performance vanishes as γ approaches 0 or α increases.

α \ γ	0.9	0.6	0
0.0001: QL	2.13 ± 0.21	2.12 ± 0.22	2.27 ± 0.42
FAQL	3.67 ± 0.31	3.75 ± 0.24	3.78 ± 0.21
RUQL	3.80 ± 0.19	3.80 ± 0.19	3.79 ± 0.21
0.0009: QL	2.61 ± 0.79	3.81 ± 0.4	4 ± 0
FAQL	3.79 ± 0.58	4 ± 0	4 ± 0
RUQL	4 ± 0	4 ± 0	4 ± 0

Table 6: The mean payoff (and standard deviation) for the social learning domain, for different values of α and γ and for the three algorithms QL, FAQL, and RUQL. RUQL achieves superior performance but the gap with QL and FAQL shrinks as α increases or γ decreases.

6. CONCLUSIONS

We presented in this paper the Repeated Update Q-learning, a novel extension to the Q-learning algorithm that addresses the problem of infrequent action updates in Q-learning. Unlike the closest state-of-the-art algorithm, our algorithm behavior remains consistent even for arbitrarily small policy. We show both theoretically and experimentally that, despite the clear difference in the underlying update equation, our algorithm reduces to FAQL (the closest state-of-the-art algorithm) for small α. Furthermore, our algorithm does not have the practical concerns of FAQL and we show ex-

α D	0.0004	0.0016	0.0064	0.0256
20000: QL	0.6 ± 10^{-3}	$0.56 \pm 2 * 10^{-3}$	$0.87 \pm 2 * 10^{-3}$	0.97 ± 10^{-3}
FAQL	0.6 ± 10^{-3}	$0.7 \pm 7 * 10^{-3}$	$0.91 \pm 3 * 10^{-3}$	0.97 ± 10^{-3}
RUQL	0.87 ± 0.06	0.87 ± 0.04	0.93 ± 0.01	$0.98 \pm 2 * 10^{-3}$
10000: QL	0.5 ± 10^{-3}	0.5 ± 10^{-3}	$0.77 \pm 2 * 10^{-3}$	0.94 ± 10^{-3}
FAQL	0.5 ± 10^{-3}	0.5 ± 0.01	$0.85 \pm 7 * 10^{-3}$	$0.96 \pm 2 * 10^{-3}$
RUQL	0.78 ± 0.07	0.79 ± 0.1	0.91 ± 0.01	$0.97 \pm 4 * 10^{-3}$

Table 3: The mean payoff (and standard deviation) for the MAB-switch game, for different values of D and α and for the three algorithms QL, FAQL, and RUQL. RUQL achieves superior performance but the gap with QL and FAQL shrinks as α increases.

γ D	0.9	0.6	0
20000: QL	0.6 ± 10^{-3}	0.68 ± 10^{-3}	0.85 ± 10^{-3}
FAQL	0.6 ± 10^{-3}	$0.86 \pm 3 * 10^{-3}$	$0.91 \pm 2 * 10^{-3}$
RUQL	0.87 ± 0.06	$0.92 \pm 2 * 10^{-3}$	$0.92 \pm 2 * 10^{-3}$
10000: QL	0.5 ± 10^{-3}	$0.58 \pm 2 * 10^{-3}$	0.72 ± 10^{-3}
FAQL	0.5 ± 10^{-3}	$0.78 \pm 4 * 10^{-3}$	$0.82 \pm 3 * 10^{-3}$
RUQL	0.78 ± 0.07	$0.83 \pm 3 * 10^{-3}$	$0.833 * 10^{-3}$

Table 4: The mean payoff (and standard deviation) for the MAB-switch game, for different values of D and γ and for the three algorithms QL, FAQL, and RUQL. RUQL achieves superior performance but the gap with QL and FAQL shrinks as α increases.

perimentally how this can result in superior performance in non-stationary environments.

As we mentioned earlier, there is a class of multi-agent learning algorithms that attempt to learn the interaction policy explicitly [2, 1, 9]. These algorithms use some variation of gradient ascent optimization, where the policy gradient follows the action values. Consequently, most of these algorithms still use Q-learning as an internal component for estimating the values of different action. One interesting future direction that we are pursuing is the effect of replacing QL with RUQL as an internal component in gradient learners. Preliminary results show considerable improvements in the benchmark domains we have tried and more thorough analysis is currently taking place.

7. REFERENCES

[1] S. Abdallah and V. Lesser. A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research*, 33:521–549, 2008.

[2] M. Bowling. Convergence and no-regret in multiagent learning. In *Proceedings of the Annual Conference on Advances in Neural Information Processing Systems*, pages 209–216, 2005.

[3] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the National Conference on Artificial intelligence/Innovative Applications of Artificial Intelligence*, pages 746–752, 1998.

[4] M. Kaisers and K. Tuyls. Frequency adjusted multi-agent q-learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 309–316, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[5] S. Sen and S. Airiau. Emergence of norms through social learning. In *Proceedings of the international joint conference on Artifical intelligence*, IJCAI'07, pages 1507–1512, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[6] T. Sugawara. Emergence and stability of social conventions in conflict situations. In *Proceedings of the international joint conference on Artifical intelligence*, pages 371–378, 2011.

[7] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1999.

[8] M. Wunder, M. L. Littman, and M. Babes. Classes of multiagent q-learning dynamics with ϵ-greedy exploration. In *International Conference on Machine Learning(ICML)*, pages 1167–1174, 2010.

[9] C. Zhang and V. Lesser. Multi-Agent Learning with Policy Prediction. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pages 927–934, Atlanta, 2010.