International Journal of Artificial Intelligence,
ISSN 0974-0635; Int. J. Artif. Intell.
2018, Volume 16, Issue Number 2

Double Delayed Q-learning

Bilal H. Abed-alguni[1] **and Mohammad Ashraf Ottom**[2]

[1]Department of Computer Sciences
Yarmouk University
21163, University Street, Irbid, Jordan
Bilal.h@yu.edu.jo

[2]Department of Computer Information Systems
Yarmouk University
21163, University Street, Irbid, Jordan
Ottom.ma@yu.edu.jo

ABSTRACT

Delayed Q-learning is an efficient model-free reinforcement-learning algorithm. This algorithm is guaranteed to converge in polynomial time to near optimal policies in Markov decision processes. However, Delayed Q-learning performs very poorly in some stochastic environments because it overestimates action values. Overestimated action values are caused by a positive bias that is a result of using the maximum value function to update the maximum expected action value. This paper applies the double-estimator method to Delayed Q-learning to construct a new algorithm called Double Delayed Q-learning (2D Q-learning). The 2D Q-learning was tested using the gambling game of roulette. The experimental results showed that 2D Q-learning converges to an optimal policy and that it performs better than Delayed Q-learning in some settings where Delayed Q-learning has a poor performance because of its large overestimation. The source code of the experiments is publicly available at `https://www.dropbox.com/s/2essch6b52sjyni/2D%20Q-learning.zip?dl=0`

Keywords: Reinforcement Learning, Double Q-learning, Delayed Q-learning, Markov Decision Process, PAC-MDP

Computing Classification System (CCS): k.3.2

1 Introduction

In many forms of machine learning, the learner follows a certain set of instructions and actions, but in Reinforcement Learning (RL) the learner must discover a set of actions to solve a certain problem. RL problems involve learning what to do, how to map situations to actions; in order to maximize a numerical payout (Gilles and Peroumalnaik, 2008; Sutton and Barto, 2017; Abed-alguni, 2017a). The Delayed Q-learning algorithm is a RL algorithm that was proposed by Strehl et al., (Strehl, Li and Littman, 2009) and can be used to find near-optimal solutions for Markov Decision Processes (MDPs). Delayed Q-learning is the first RL algorithm that is

known to be PAC-MDP (Probably Approximately Correct in Markov Decision Processes) (Strehl et al., 2009; Strehl, Li, Wiewiora, Langford and Littman, 2006).

Although the Delayed Q-learning algorithm is guaranteed to converge in polynomial time to near-optimal policies, it may overestimate its action values in some stochastic environments (Capen, Clapp, Campbell et al., 1971; Hasselt, 2010; Thaler, 1988; Van den Steen, 2004). Actually, the overestimation of action values is not exclusive to Delayed Q-learning, it is a common problem with all the RL algorithms that use the maximum operator to estimate the value of the next state such as Q-learning (Sutton and Barto, 1998; Watkins, 1989) and Fitted Q-iteration (Ernst, Geurts and Wehenkel, 2005). This is because those algorithms use the maximum operator to find an approximation for the expected maximum action value. Also, the maximum operator uses the same action values for selecting and evaluating a certain action. As a result, the possibility of selecting overestimated action values becomes more likely.

It is worth pointing out that the overestimation problem is not specific to the RL algorithms. The overestimation problem may also occur with the optimization algorithms (Abed-alguni, 2017b; Abed-alguni, 2017a; Vrkalovic, Teban and Borlea, 2017; Ali, Awad and Duwairi, 2016; Precup, Angelov, Costa and Sayed-Mouchaweh, 2015; Vaščák, 2012) when estimating the maximum of the expected values of multiple random variables over multiple samples. In other words, the overestimation problem is a natural result when estimating the expected utility of a best choice using the maximum function. For example, the expected utility of a dice roll is 3.5, but if we throw the dice 100 times and then calculate the maximum over all throws, we will probably have a value more than 3.5.

Hasselt (Hasselt, 2010) proposed the Double Q-learning algorithm (DQL) to address the overestimation problem of Q-learning using a double-estimator method (Abdallah and Kaisers, 2016; Hasselt, 2010; Lee, Defourny and Powell, 2013). In the DQL algorithm, two separate value functions (two estimators) are used. Whenever an experience occurs, one of the value functions is randomly selected to be updated, such that there are two separate weight sets: a set for selecting actions and another set for evaluating actions. DQL has been proved theoretically and experimentally that it performs better than Q-learning. Furthermore, it does not overestimate its action values and in most cases DQL underestimates its action values.

According to a recent search on double-estimator method and Delayed Q-learning and to best of authors' knowledge; there have been no contribution or research study that apply the double-estimator method to Delayed Q-learning. This paper tries to contribute to the field of RL by applying the double-estimator method to Delayed Q-learning to construct a new approach called Double Delayed Q-learning (2D Q-learning). There are two goals for the current paper. The first goal is to show that Delayed Q-learning can suffer a performance penalty because it uses the maximum function to approximate the maximum expected value. The second goal is to show that 2D Q-learning can reduce the overestimation problem of Delayed Q-learning.

The remainder of the paper is organized as follows: Section 2 presents basic definitions, Section 3 discusses related work, Section 4 discusses Double Delayed Q-learning, Section 5 discusses simulation results using the roulette problem and Section 7 presents the conclusion and future work of this paper.

2 Background Information

2.1 Markov Decision Processes

RL tasks are commonly represented as Markov Decision Processes (MDP) (Sutton and Barto, 1998; Abed-alguni, Chalup, Henskens and Paul, 2015b). An MDP T is a six tuple $< I, S, A, T, R, \gamma >$, where I is an agent, $S = \{s_0, s_1, ..., s_{n-1}\}$ is a set of n states representing the environment, $A = \{a_0, a_1, ..., a_{m-1}\}$ is a set of m actions available to agent I. The transition function $T : S \times A \times S \rightarrow \mathbb{R}$ measures the probability of transferring to state s' upon executing action a at state s. The reward function $R : S \times A \rightarrow \mathbb{R}$ returns the expected reward that agent I receives after executing action a at state s. Finally, the discount factor $\gamma \in [0, 1]$ is used to determine the present value of future rewards. In RL, an agent applies action a from state s then receives reward r with expectation $R(s, a)$ and observes a new state s' that has probability $T(s'|s, a)$. The goal of solving MDPs using RL is to find a policy $\pi = S \rightarrow A$, which is a sequence of actions, that the agent can apply to maximize its total discounted reward $\gamma(r_0 + r_1, ... + r_{n-1})$.

2.2 Delayed Q-learning

Delayed Q-learning is classified as a model-free algorithm because its state complexity is $O(|S|^2|A|)$ and its computational complexity is $O(\ln |A|)$, where $|S|$ is the number of states of S and $|A|$ is the number of actions of A for a given MDP (Strehl et al., 2006; Zhang, Tang and Yao, 2015). These complexities are asymptotically equivalent to those of Q-learning. Strehl et al. (Strehl et al., 2006) have mathematically proven that Delayed Q-learning is a PAC-MDP by proving that the polynomial upper bound of Delayed Q-learning on its sample complexity using two dependencies (S and A) is a noteworthy improvement compared to the upper bound of any other previously known RL algorithm.

The main difference between the original Q-learning algorithm that was proposed by Watkins (Watkins, 1989) and Delayed Q-learning is the frequency of Q-value update. A Q-value of a state-action pair is updated using Q-learning each time the state-action is visited, while a Q-value of a state-action pair is updated using Delayed Q-learning after every m visits to the state-action pair, where m is a predetermined parameter.

Figure 1 shows the flow of the Delayed Q-learning algorithm. This algorithm calculates the expected utilities, $Q(s, a)$ for each state-action pair (s, a). In the algorithm, $Q_t(s, a)$ denotes the Q-value for state-action pair (s, a) and $V_t(s)$ denotes $max_{a \in A} Q_t(s, a)$ at instant t. The algorithm follows a greedy action-selection method to choose its actions, which means that the next action to be chosen a' is one of the actions with the highest expected utilities $a' :=$ argmax$_{a \in A} Q(s, a)$.

In addition to the standard learning parameters of Q-learning, the Delayed Q-learning algorithm depends on two free parameters: $\epsilon 1 \in (0, 1)$ and m. The parameter $\epsilon 1$ was introduced to provide a constant exploration value that is added to each Q-value when it is updated. The parameter m is a positive integer number that represents the number of visits to a state-action pair before its Q-value is updated. Actually, the update of a Q-value in Delayed Q-learning is an

average of the target values for m sample updates. This averaging process helps in alleviating some of the effects of randomness and therefore it contributes in achieving optimism with high probability when it is combined with $\epsilon 1$.
Delayed Q-learning maintains three internal variables $U(s,a)$, $L(s,a)$ and $LEARN(s,a)$ for each (s,a). The first variable stores the sum of possible estimates that are used to update $Q(s,a)$ when (s,a) has been visited m times. The variable $L(s,a)$ stores the number of visits to (s,a). The last variable $LEARN(s,a)$ is a flag that indicates if it is allowed to collect samples for (s,a). Initially, $LEARN(s,a)$ is set to $true$ for each (s,a) or whenever any (s,a) is being updated in order to reflect the changes that may be required to update the estimate Q(s,a). The learning flag can only be set to $false$ during the period of m visits to (s, a) that finally ended with an attempted update of (s,a) that was not successful.

2.2.1 The Update Rule of Q-learning vs The Update Rule of Delayed Q-learning

The Q-learning algorithm updates its Q-value estimates on every time step using a function called the Q-function (Abed-alguni, Paul, Chalup and Henskens, 2016). Let us assume that action a was applied in state s at time t which resulted in reward r_t and a transition to next state s'. Then, the Q-value estimate for (s,a) is updated as follows:

$$Q_{t+1}(s,a) \quad \longleftarrow \quad (1 \; - \; \alpha_t)\, Q_t(s,a) \; + \; \alpha\, [r_t \; + \; \gamma\, max_{a_{t+1} \in A} Q(s',a') \; - \; Q_t(s,a)], \quad (2.1)$$

where s and $s' \in S$, a and $a' \in A, \alpha_t \in [0,1]$ is the learning rate at instant t and $\gamma \in [0,1]$ is the discount factor.
On the other hand, the update of Q-value estimate for state-action pair (s,a) in Delayed Q-learning occurs after each m attempted updates for (s,a). Let us assume that state-action pair (s,a) was executed m times, resulting in m transitions $s_{k_1}, s_{k_2}, ..., s_{k_m}$ and m associated rewards $r_{k_1}, r_{k_2}, ..., r_{k_m}$ respectively at times $k_1 < k_2 < ... < k_m$, where $k_m = t$. Using these information the Q-value estimate for (s,a) is updated as follows:

$$Q_{t+1}(s,a) \longleftarrow \frac{1}{m} \sum_{i=1}^{m} \big(r_{k_i} + \gamma V_{k_i}(s_{k_i})\big) + \epsilon_1 \quad (2.2)$$

Equation 2.2 can be performed as long as the difference between $Q_t(s,a)$ and $Q_{t+1}(s,a)$ is at least $\epsilon 1$. Formally, an update can be performed as long as the following equation is satisfied:

$$Q_t(s,a) - \Big(\frac{1}{m} \sum_{i=1}^{m} \big(r_{k_i} + \gamma V_{k_i}(s_{k_i})\big)\Big) \geq 2\epsilon_1 \quad (2.3)$$

Otherwise, the Q-value estimate cannot be updated and therefore $Q_{t+1}(s,a) = Q_t(s,a)$.

2.3 The Single Estimator vs The Double Estimator

Let X be a set of n random variables $X = X_1, ..., X_n$. In many learning problems, the main goal is to maximize the expected value of the variables:

$$\max_i E\{X_i\} \quad (2.4)$$

Inputs: γ, S, A, m, ϵ_1
for all (s, a) **do**
  $Q(s, a) \leftarrow 1/(1-\gamma)$ // Q-value estimates
  $U(s, a) \leftarrow 0$ // used for attempted updates
  $l(s, a) \leftarrow 0$ // counters
  $t(s, a) \leftarrow 0$ // time of last attempted update
  $LEARN(s, a) \leftarrow true$ // the LEARN flags
end for
$t^* \leftarrow 0$ // time of most recent Q-value change
for $t = 1, 2, 3, ...$ **do**
  Let s denote the state at time t.
  Choose action $a := \text{argmax}_{a' \in A} Q(s, a')$.
  Let r be the immediate reward and s' the next state after executing action a from state s.
  if $LEARN(s, a) = true$ **then**
    $U(s, a) \leftarrow U(s, a) + r + \gamma max_{a'} Q(s', a')$
    $l(s, a) \leftarrow l(s, a) + 1$
    if $l(s, a) = m$ **then**
      if $Q(s, a) - U(s, a)/m \geq 2\epsilon 1$ **then**
        $Q(s, a) \leftarrow U(s, a)/m + \epsilon 1$
        $t^* \leftarrow t$
      else if $t(s, a) \geq t^*$ **then**
        $LEARN(s, a) \leftarrow false$
      end if
      $t(s, a) \leftarrow t, U(s, a) \leftarrow 0, l(s, a) \leftarrow 0$
    end if
  else if $t(s, a) < t^*$ **then**
    $LEARN(s, a) \leftarrow true$
  end if
end for

Figure 1: Delayed Q-learning

Equation 2.4 is commonly approximated using the value of the maximum estimator as follows:

$$\max_i E\{X_i\} = \max_i E\{\mu_i\} \approx \max_i \mu_i(S) \tag{2.5}$$

This method is known as the single-estimator method (Hasselt, 2010). It is used in Delayed Q-learning as an approximation method that finds an approximated value for the next state by maximizing over the estimated Q-value estimates in that state.

Unfortunately, the single-estimator method may negatively affect the performance of some well-known RL algorithms such as Q-learning and Delayed Q-learning (Capen et al., 1971; Hasselt, 2010; Thaler, 1988; Van den Steen, 2004). This is due to the use of the maximum operator as

an approximation of a set of estimators which may cause overestimation of Q-value estimates ($\max_i E\{X_i\} < max\ \mu_i(S)$). Therefore, Hasselt (Hasselt, 2010) proposed the double-estimator method that attempts to address the overestimation problem of the single-estimator method. The double-estimator method maintains two sets of estimators: $\mu^X = \{\mu_1^X, ..., \mu_M^X\}$ and $\mu^Y = \{\mu_1^Y, ..., \mu_M^Y\}$. Both μ^X and μ^Y are updated using a subset of samples, such that $S = S^X \bigcup S^Y$ and $S^X \bigcap S^Y = \phi$ and $\mu_i^X = \frac{1}{|S_i^X|} \sum_{s \in S_i^X} s$ and $\mu_i^Y = \frac{1}{|S_i^Y|} \sum_{s \in S_i^Y} s$.

2.4 Double Q-learning

The Double Q-learning algorithm (DQL) that was proposed by Hasselt (Hasselt, 2010) is an interesting variation of Q-learning. DQL is specifically designed to address how Q-learning may largely overestimate Q-value estimates when it is learning in some stochastic environments (Abdallah and Kaisers, 2016). The overestimation problem may be caused by the use of max operator in the Q-function of Q-learning (Section 2.2.1). DQL attempts to solve the overestimation problem by using two separate Q-value estimates for each state-action pair: Q^X and Q^Y. DQL updates the Q-value of state-action pair (s, a) by uniformly selecting either $Q^X(s, a)$ or $Q^Y(s, a)$ then updating it. The update is performed by using a modified Q-function that applies the max operator to the Q-values of the other table (Figure 2).

Initialize Q^X, Q^Y, s.
repeat
  Choose a, based on $Q^X(s, .)$ or $Q^Y(s, .)$, observe r, s'
  Choose uniformly either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $x^* :=$ argmax$_{a \in A} Q^X(s', a)$.
    $Q^X(s, a) \leftarrow Q^X(s, a) + \alpha(s, a)\big(r + \gamma Q^Y(s', x^*) - Q^X(s, a)\big)$
  else if UPDATE(B) **then**
    Define $y^* :=$ argmax$_{a \in A} Q^Y(s', a)$.
    $Q^Y(s, a) \leftarrow Q^Y(s, a) + \alpha(s, a)\big(r + \gamma Q^X(s', y^*) - Q^Y(s, a)\big)$
  end if
  $s \leftarrow s'$
until end

Figure 2: Double Q-learning

The mathematical analysis of DQL (Hasselt, 2010) proves that DQL does not suffer from an overestimation bias but it may underestimate the Q-values.

3 Related Work

This section is divided into two subsections. Section 3.1 discusses famous variations of Q-learning that attempt to address the policy bias of Q-learning, while Section 3.2 discusses famous variations of Q-learning that have been proven to be PAC-MDP.

3.1 Addressing the Policy-bias of Q-learning

Repeated Update Q-learning (RUQL) is a variation of the Q-learning algorithm that addresses the problem of policy bias of Q-learning(Abdallah and Kaisers, 2016). RUQL is based on the idea of repeating the update of the actions that are chosen with low probabilities. Suppose that action a was chosen with low probability $P(s, a)$, then according to RUQL that update of that action will be repeated $\frac{1}{P(s,a)}$ times. A main advantage of RUQL is that it is guaranteed to converge to optimality in non-stationary environment. However, Q-learning may outperform RUQL in stationary environments that have stochastic outcomes. This is because RUQL may increase the learning rates of the actions that are chosen with low probabilities.

Some variations of Q-learning have been proposed recently in order to minimize the estimation errors of Q-learning in non-stationary environments using methods to adjust the learning rate of Q-learning (Noda, 2009; George and Powell, 2006). For example, Noda (Noda, 2009) proposed a variation of the Q-learning algorithm that dynamically changes the value of its learning rate using the gradient descent optimization algorithm. This algorithm achieves better performance than Q-learning in non-stationary environments. However, it can be only applied to stateless problems. George et al., (George and Powell, 2006) also introduced an algorithm called optimal step size algorithm (OSA) to update the learning rate of Q-learning in dynamic environments. OSA attempts to reduce the noise factors in noisy non-stationary environments by considering the relationships among the noise variance, value of learning rate and changes in learning values. Unlike the algorithm proposed by Noda (Noda, 2009), the OSA algorithm focuses on minimizing the estimation errors without investigating its effect on the learning performance.

Q-learning does not act in accordance to its prediction because it updates its action estimates at different frequencies. A possible solution to this problem was proposed by Kaisers and Tuyls (Kaisers and Tuyls, 2010). This research introduced a RL algorithm called Frequency Adjusted Q-learning (FAQ-learning) that combines Q-learning with the prediction of the evolutionary model. FAQ-learning recompenses the difference in frequencies by independently controlling the influence of the learning rate for each action. This process removes the initialization dependencies and accelerates converging to more rational policy trajectories. An advantage of FAQ-learning is that it outperforms Q-learning in single-state environments, but it requires more computations than Q-learning. It would be interesting to verify the performance and computational complexity of FAQ-learning in multi-state environments.

Peng and Willlams (Peng and Williams, 1996) proposed a variation of Q-learning by the name incremental multi-step Q-learning (Q(λ)-learning) that attempts to accelerate the convergence rate of Q-learning. This algorithm is a combination of Q-learning and the returns of the estimation process of TD(λ). The returns of TD(λ) is used in Q(λ)-learning to incrementally correct the predictions of Q-learning, which helps in propagating information quickly to where it is needed. Q(λ)-learning performs much better than Q-learning on a number of problems[1]. A disadvantage of Q(λ)-learning is that it does not converge to an optimal policy under arbitrary action-selection policy that tests all the possible actions in every state.

Several cooperative Q-learning algorithms have been proposed recently that allow indepen-

[1] Details of the experiments were not mentioned in (Peng and Williams, 1996)

dent Q-learners to share their Q-values without using the maximum function to update their expected Q-values (Abed-alguni et al., 2015b; Abed-alguni, Chalup, Henskens and Paul, 2015a; Abed-alguni, 2017b; Abed-alguni et al., 2016). Q-learning with aggregation (Abed-alguni et al., 2015b; Abed-alguni et al., 2015a) is a cooperative Q-learning algorithm that utilizes three types of agents (workers, tutors and consultants) to decompose the search space into smaller problems. Average aggregation Q-learning (Abed-alguni et al., 2016) is another cooperative Q-learning algorithm that calculates its Q-table by averaging the Q-tables of several cooperative algorithms (BEST-Q , AVE-Q, PSO-Q, and WSS).

Recently, different research studies have attempted to use or integrate the optimization technique (e.g., Bat algorithm (Yang, 2011), Cuckoo search algorithm (Yang and Deb, 2009)) with Q-learning (Abed-alguni, 2017b; Abed-alguni, 2017a). This is because the problem model of Q-learning can be modeled as an optimization problem (maximization or minimization problem), where the Q-values represent a population of candidate solutions and the Q-function represents the fitness function. For example, the Bat Q-learning algorithm (Abed-alguni, 2017b) is a cooperative Q-learning algorithm that uses the Bat algorithm as a sharing strategy of Q-values between independent learners. Abed-alguni (Abed-alguni, 2017a) proposed an action selection technique based on the Cuckoo search algorithm in order to reduce the effects of both the overestimation and underestimation problems of Q-values.

3.2 PAC Algorithms

PAC (probably approximately correct) is a mathematical analysis model that is commonly used to analyze RL algorithms. In this model, the reinforcement learner collects a set of samples with known probability distributions, then it should select a generalization function from a set of possible functions, such that there is a high probability that the selected function has a low generalization error (Even-Dar, Mannor and Mansour, 2002). A generalization error is a statistical measure that evaluates the accuracy of a RL algorithm based on a random set of samples.

Various RL algorithms have been proven to be PAC, such as Delayed Q-learning (Section 2.2), Speedy Q-learning (Azar, Munos, Ghavamzadeh and Kappen, 2011) and Batch Q-learning (Murphy, 2005). These algorithms are guaranteed to converge to optimality on a set of samples with polynomially bounded sample sizes.

Speedy Q-learning (SQL) is a mathematically proven PAC algorithm that was proposed by Azar et al., (Azar et al., 2011). At instant t when state-action pair (s, a) is performed, SQL updates two Q-value estimates $Q_t(s, a)$ and $Q_{t-1}(s, a)$ then use them to update $Q_{t+1}(s, a)$. This means that the implementation of SQL requires double the state complexity of Q-learning in order to perform the update of Q-values. However, this process allows SQL to use larger learning rates in its Q-function. Consequently, SQL converges faster to optimal solutions than Q-learning. A drawback of SQL is that it was not empirically compared with other PAC-MDP algorithms, which is necessary to prove the efficiency of it. (Abdallah and Kaisers, 2016; Strehl et al., 2009; Szita and Szepesvári, 2010).

Batch Q-learning is a version of Q-learning that was specially proposed for non-stationary, non-Markovian domains. The purpose of Batch Q-learning is to learn a policy using a single

training set of finite horizon trajectories and a big observation space (Murphy, 2005). The term Batch in Batch Q-learning is used here to indicate that the learning process takes place after gathering the training set. The whole training set is used to calculate the approximations of the Q-functions. In Batch Q-learning, a generalization error is calculated at instant t for observation O_t and policy π that is used to generate an independent training set. An advantage of Batch Q-learning is that it is a mathematically proven PAC algorithm that converges faster to a solution than Q-learning.

4 Double Delayed Q-learning (2D Q-learning)

This section describes a new RL algorithm, Double Delayed Q-learning (2D Q-learning).
Figure 3 shows the flow of the 2D Q-learning algorithm. The algorithm maintains two Q-value estimates, $Q^X(s,a)$ and $Q^Y(s,a)$, for each state-action pair (s,a). As in the original Delayed Q-learning algorithm, each Q-value estimate is evaluated following a greedy policy according to its current values, but the value of the greedy policy is evaluated using the Q-table of the other Q-value estimate. At instant $t = 1, 2, ...$, let $Q^X(s,a)$ and $Q^Y(s,a)$ denote two Q-value estimates for state-action pair (s,a) and let $V_t^X(s)$ denote the Q-value of the next state $Q^X(s,a^*)$ for Q^X, where $a^* :=$ argmax$_{a' \in A} Q^Y(s,a')$ and let $V_t^Y(s)$ denote the Q-value of the next state $Q^Y(s,b^*)$ for Q^Y, where $b^* :=$ argmax$_{a' \in A} Q^X(s,a')$.
Besides having two Q-value estimates for each state-action pair (s,a), the 2D Q-learning maintains a single learning flag LEARN$(s,a) \in \{true, false\}$, for each (s,a). At time t, let LEARN$_t(s,a)$ be the value of the learning flag before performing the tth action. LEARN$_t(s,a)$ indicates whether it is allowed to collect sample updates for the Q-value estimates $Q^X(s,a)$ and $Q^Y(s,a)$ at instant t.
As in Delayed Q-learning, 2D Q-learning also depends on the following internal variables and free parameters:

- ϵ_1: a free parameter that provides a constant additional exploration value that is added to each Q-value when it is updated.
- m: the number of attempted updates to $Q(s,a)$ of state action pair (s,a) before an actual update is allowed.
- $l(s,a)$: a counter that represents the number of collected sample updates for (s,a)

2D Q-learning maintains two temporal variables $U^X(s,a)$ and $U^Y(s,a)$ that store the sum of possible estimates for $Q^X(s,a)$ and $Q^Y(s,a)$, receptively. After each m attempted updates to state-action pair (s,a), either $Q^X(s,a)$ or $Q^Y(s,a)$ is selected randomly to be updated using its corresponding temporal storage.

4.1 Initialization of 2D Q-learning

The Q-value estimates $Q^X(s,a)$ and $Q^Y(s,a)$ are both initialized to $1/(1-\gamma)$, the temporal storages for attempted updates $U^X(s,a)$ and $U^Y(s,a)$ are both set to 0, the counter $L(s,a)$ is set to 0 and finally the learning flag $LEARN(s,a)$ is set to true for all $(s,a) \in S \times A$.

4.2 The Update Rules of 2D Q-learning

Suppose that at instant $t \geq 0$, the state-action pair (s, a) is performed, producing an attempted update. Based on this, let $s_{w_1}, s_{w_2}, ..., s_{w_m}$ be the last m perceived next-states and let $r_{w_1}, r_{w_2}, ..., r_{w_m}$ be the last m received rewards at times $w_1 < w_2 < ... < w_m$, respectively, such that $w_m = t$.
In 2D Q-learning, the update of Q-value estimate for state-action pair (s, a) occurs either for $Q^X(s, a)$ or $Q^Y(s, a)$ after each m attempted updates. Therefore, at time t, action a was performed from state s, resulting in a transition to state s_{w_m} and an immediate reward r_{w_m}. After the tth action, either $Q^X(s, a)$ or $Q^Y(s, a)$ is uniformly selected at random then updated.
In case $Q^X(s, a)$ is selected, it is updated using a Q-function that is similar to the Q-function of Delayed Q-learning (Equation 2.2) with only one modification: the highest rewarded action a^* is determined by applying the maximum operator to the action estimates of table Y.

$$\begin{aligned} Q^X_{t+1}(s, a) &\longleftarrow \frac{1}{m}\sum_{i=1}^{m}\left(r_{w_i} + \gamma V^X_{w_i}(s_{w_i})\right) + \epsilon_1 \\ V^X_t(s) &\leftarrow Q^X(s, a^*), \text{ where } a^* := \text{argmax}_{a' \in A} Q^Y(s, a') \end{aligned} \tag{4.1}$$

In case $Q^Y(s, a)$ is selected randomly, it is also updated using a modified Q-function. In this function, the best action b^* is calculated by applying the maximum operator to the action estimates of table X.

$$\begin{aligned} Q^Y_{t+1}(s, a) &\longleftarrow \frac{1}{m}\sum_{i=1}^{m}\left(r_{w_i} + \gamma V^Y_{w_i}(s_{w_i})\right) - \epsilon_1 \\ V^Y_t(s) &\leftarrow Q^Y(s, b^*), \text{ where } b^* := \text{argmax}_{a' \in A} Q^X(s, a') \end{aligned} \tag{4.2}$$

Whether $Q^X(s, a)$ or $Q^Y(s, a)$ is chosen to be updated, the difference between $Q_t(s, a)$ and $Q_{t+1}(s, a)$ must be at least $\epsilon 1$. In other words, the value of the following condition should be $true$.

$$Q_t(s, a) - \left(\frac{1}{m}\sum_{i=1}^{m}\left(r_{w_i} + \gamma V_{w_i}(s_{w_i})\right)\right) \geq 2\epsilon_1 \tag{4.3}$$

Otherwise the Q-value estimates are nor changed.

4.3 2D Q-learning Discussion

2D Q-learning is similar in many aspects to Delayed Q-learning and Double Q-learning. The main goal behind the design of 2D Q-learning is to construct a PAC-MDP algorithm that does not overestimate its Q-value estimates.
As in Double Q-learning, 2D Q-learning has two Q-functions: Q^X and Q^Y. Both Q-functions learn from separate sets of experiences, but both Q-functions can be used to select an action. Each Q-function depends on the other Q-function to determine the value of the next state.

Inputs: γ, S, A, m, ϵ_1
for all (s, a) **do**
  $Q^X(s, a) \leftarrow 1/(1-\gamma)$ // Q-value estimates of Q-table X
  $Q^Y(s, a) \leftarrow 1/(1-\gamma)$ // Q-value estimates of Q-table Y
  $U^X(s, a) \leftarrow 0$ // used for attempted updates on Q-table X
  $U^Y(s, a) \leftarrow 0$ // used for attempted updates on Q-table Y
  $l(s, a) \leftarrow 0$ // counters
  $t(s, a) \leftarrow 0$ // time of last attempted update
  $LEARN(s, a) \leftarrow true$ // the LEARN flags
end for
Choose (e.g. random) either UPDATE(X) or UPDATE(Y)
$t^* \leftarrow 0$ // time of most recent Q-value change
for $t = 1, 2, 3, ...$ **do**
  Choose a, based on $Q^X(s, .)$ and $Q^Y(s, .)$, observe r, s'
  if LEARN$(s, a) = true$ And UPDATE$(X) = true$ **then**
    Define $a^* :=$ argmax$_{a' \in A} Q^Y(s, a')$.
    $U^X(s, a) \leftarrow U^X(s, a) + r + \gamma Q^X(s, a^*)$
    $l(s, a) \leftarrow l(s, a) + 1$
    if $l(s, a) = m$ **then**
      if $Q^X(s, a) - U^X(s, a)/m \geq 2\epsilon 1$ **then**
        $Q^X(s, a) \leftarrow U^X(s, a)/m + \epsilon 1$
        $t^* \leftarrow t$
      else if $t(s, a) \geq t^*$ **then**
        $LEARN(s, a) \leftarrow false$
      end if
      $t(s, a) \leftarrow t, U^X(s, a) \leftarrow 0, l(s, a) \leftarrow 0$
    end if
  else if LEARN$(s, a) = true$ And UPDATE$(Y) = true$ **then**
    Define $b^* :=$ argmax$_{a' \in A} Q^X(s, a')$.
    $U^Y(s, a) \leftarrow U^Y(s, a) + r + \gamma Q^Y(s, b^*)$
    $l(s, a) \leftarrow l(s, a) + 1$
    if $l(s, a) = m$ **then**
      if $Q^Y(s, a) - U^Y(s, a)/m \geq 2\epsilon 1$ **then**
        $Q^Y(s, a) \leftarrow U^Y(s, a)/m - \epsilon 1$
        $t^* \leftarrow t$
      else if $t(s, a) \geq t^*$ **then**
        $LEARN(s, a) \leftarrow false$
      end if
      $t(s, a) \leftarrow t, U^Y(s, a) \leftarrow 0, l(s, a) \leftarrow 0$
    end if
  else if $t(s, a) < t^*$ **then**
    $LEARN(s, a) \leftarrow true$
  end if
end for

Figure 3: Delayed Q-learning with A Double Estimator.

In equation 4.1, $V^X(s) = Q^X(s, a^*)$ represents the Q-value of the best action in next-state s' according to Q_X. However, Q^Y is used here to select the best action $a^* := \text{argmax}_{a' \in A} Q^Y(s, a')$ instead of using Q^X to select the best action $\text{argmax}_{a' \in A}\ Q^X(s, a')$ as in Delayed Q-learning. This update is considered unbiased because Q^X was updated using two sets of samples. Similarly, Q^Y uses b^* and Q^X to update its Q-value estimates. According to Hasselt (Hasselt, 2010), using a double-estimator approach may produce underestimated Q-values, but will not produce overestimated Q-values.

Like Delayed Q-learning, 2D Q-learning updates the Q-value of state-action pair (s, a) after it has been experienced m times by averaging the action values for the m most recent attempted updates to (s, a) and adding an exploration bonus ϵ_1 to it. This process reduces some of the effects of randomness and contributes in achieving optimism with high probability.

An implementation of 2D Q-learning can be accomplished with $O(|S||A|)$ space complexity and $O(\ln |A|)$ computational complexity as with Delayed Q-learning and Double Q-learning.

5 Experiments

Q-learning and Delayed Q-learning normally overestimate the expected profit of playing the game of roulette (Hasselt, 2010) because both algorithms use the single-estimator method (Section 2.2.1) to evaluate the expected profit. In this section, the gambling game of roulette was used to illustrate the bias problem of Q-learning and Delayed Q-learning and was also used to compare the performance of 2D Q-learning with Q-learning, Delayed Q-learning and Double Q-learning. The main goal of the experiments is to test whether the double-estimator method of 2D Q-learning (Section 4) can reduce the overestimation problem of Delayed Q-learning (Section 2.2).

Roulette is a well known casino game. In Roulette, a player may choose among 170 possible betting actions that includes betting on numbers, betting on black or red colors or betting on the number whether it is an even or odd number. Assuming that the player places a $\$1$ bet every play, the expected reward for any possible bet is $\$0.947$ per dollar which means that the expected loss is $-\$0.053$ per play (Hasselt, 2010). The roulette game can be represented in machine learning as a single-state MDP (Tokic, 2010) with 171 possible actions (170 possible betting actions and one stopping action). The game takes place in rounds where the player places a $\$1$ bet each round regardless of the available fund with him.

5.1 Setup

In all of the experiments, the discount factor γ was 0.95 and the learning rate was $\alpha_t(s, a) = 1/n_t(s, a)$ (Even-Dar and Mansour, 2003). The value of the exploration parameter ϵ_1 was 0.7 and the number of sample updates m before the actual update was 5 for Delayed Q-learning and 2D Q-learning. The parameter $n_t(s, a)$ denotes the number of updates for state-action pair (s, a) using Q-learning. For Delayed Q-learning, $n_t(s, a) = n_t^X(s, a)$ when Q^X is selected to be updated, while $n_t(s, a) = n_t^Y(s, a)$ when Q^Y is selected to be updated. All the 171 actions are concurrently updated each episode. It is important to note that the values of parameters were chosen based on extensive simulations.

As in any betting system, the average action values for all betting actions in roulette will approach 0 because the geometric series that represents the roulette game with expected profit of 0.95 and expected loss of -0.053 will certainly tend to zero with time (Hasselt, 2010; Orselli and Dunne, 1996). Therefore, all the average action values in the following section were compared to 0.

6 Results and Discussion

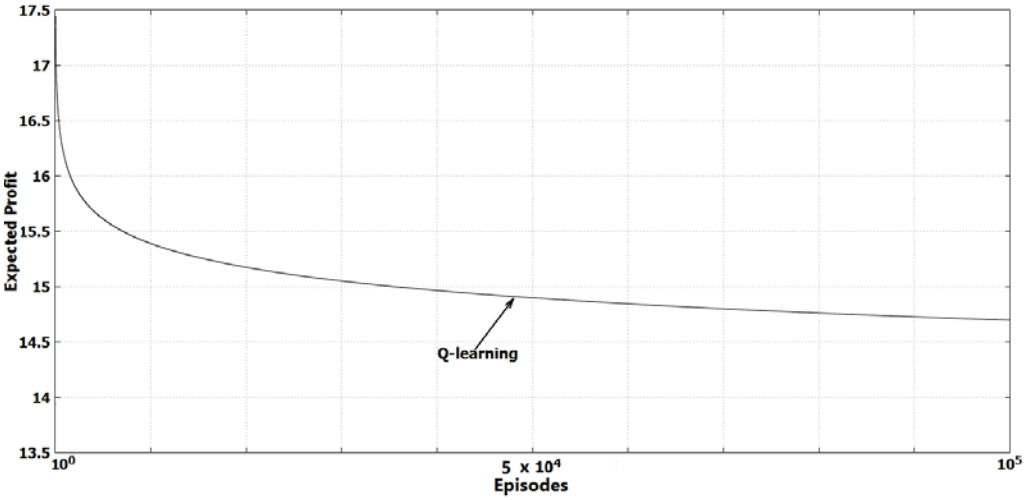

Figure 4: Average action values for all bets over $100,000$ episodes as found by Q-learning.

Figure 4 shows the average action values for all betting actions over 100,000 episodes according to Q-learning. After 100,000 episodes of learning, Q-learning estimated the expected average profits for all actions between $14.6 and $14.68. Obviously, any value within this range is far from approaching zero. These results indicate that Q-learning overestimates the action values in the Roulette game.

Figure 5 shows the average action values over all possible actions according to Delayed Q-learning, Double Q-learning and 2D Q-learning. After $100,000$ episodes of learning, Delayed Q-learning estimated the expected average profits for all actions between $1.026 and $1.073 while Double Q-learning estimated the expected values of the actions between $0.860 and $0.4667. These results suggest that Double Q-learning performs even better than Delayed Q-learning.

Moreover, the above results suggest that the overestimation bias of Delayed Q-learning and Double Q-learning are not heavy as the overestimation bias of Q-learning shown in Figure 4 ($14.6 and $14.68). This is because Q-learning updates the Q-value of a state-action pair once it has been experienced, while Delayed Q-learning delays the update of the Q-value of a state-action pair until it has been experienced m times and Double Q-learning uses two estimation functions for each Q-value estimation. These results are not surprising because the update process of Delayed Q-learning minimizes the randomness of the Q-values which contributes

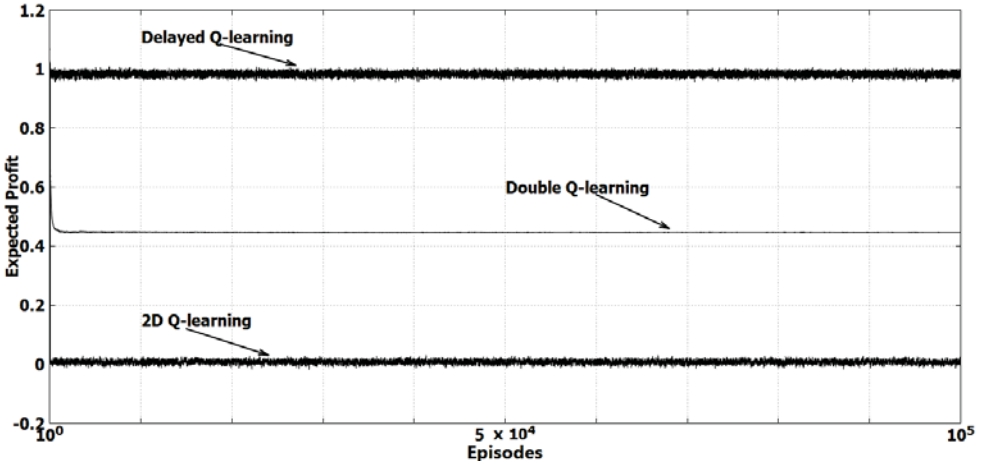

Figure 5: Average action values for all bets over $100,000$ episodes as found by Delayed Q-learning, Double Q-learning and 2D Q-learning.

with other factors in achieving optimism with high probability (Section 2.2).
Figure 5 also shows that by the end of the learning process; 2D Q-learning estimated the average expected profits between $0.035 and $-0.014. These numbers are closer to zero than the results of Delayed Q-learning ($1.026 and $1.073) and Double Q-learning ($0.860 and $0.4667). These results suggest that although 2D Q-learning may slightly underestimate the action values, but it does not suffer from the overestimation problem as Delayed Q-learning and Q-learning do.

6.1 Convergence of the Algorithms

Figures 6 to 8 show the average action values for all betting actions over 1,000 episodes according to Delayed Q-learning, Double Q-learning and 2D Q-learning. The purpose of these figures is to demonstrate the convergence of the algorithms under different values of ϵ_1 (ϵ_1=0.7 in Figure 6, ϵ_1=-1 in Figure 7, ϵ_1=5 in Figure 8). In these figures, an algorithm is said to have converged when its average expected profit is 0 ± 0.05 (assuming 0 is the optimal solution) for 50 consecutive episodes.
It can be seen from Figure 6 (ϵ_1=0.7) that 2D Q-learning is the only algorithm that converged to a solution (expected profit between -0.001550492 and +0.008252326) after 120 episodes of learning. The other algorithms, Delayed Q-learning and Double Q-learning, did not converge to solutions within 1,000 episodes of learning.
In Figure 7, Delayed Q-learning underestimated the expected profit and did not converge to a solution when ϵ_1 was set to -1, while 2D Q-learning slightly underestimated the expected profit (expected profit between -0.042225065 to 0.0422300) and converged to a solution after 184 episodes.
Figure 8 shows that Delayed Q-learning overestimated the expected profit (around 6.127885) when ϵ_1 was set to 5 On the other hand, 2D Q-learning did not converge to a solution and its

expected profit fluctuated between 1.9863 and -1.53 after 160 episodes of learning.
To sum up, the results in Figure 6 (ϵ_1=0.7) indicate that 2D Q-learning converges faster to a solution than the other algorithms. The results in Figure 7 (ϵ_1=-1) indicate that a negative value of ϵ_1 makes Delayed Q-learning underestimates its expected profit and slightly affects the performance of 2D Q-learning. The results in Figure 8 (ϵ_1=5) suggest that a large positive value of ϵ_1 makes Delayed Q-learning underestimates its expected profit. Figure 8 also indicates that the performance of 2D Q-learning fluctuates with large values of ϵ_1. It is important to note that the performance of Double Q-learning (Figures 6 to 8) is not affected by ϵ_1 because it is not one of its parameters.

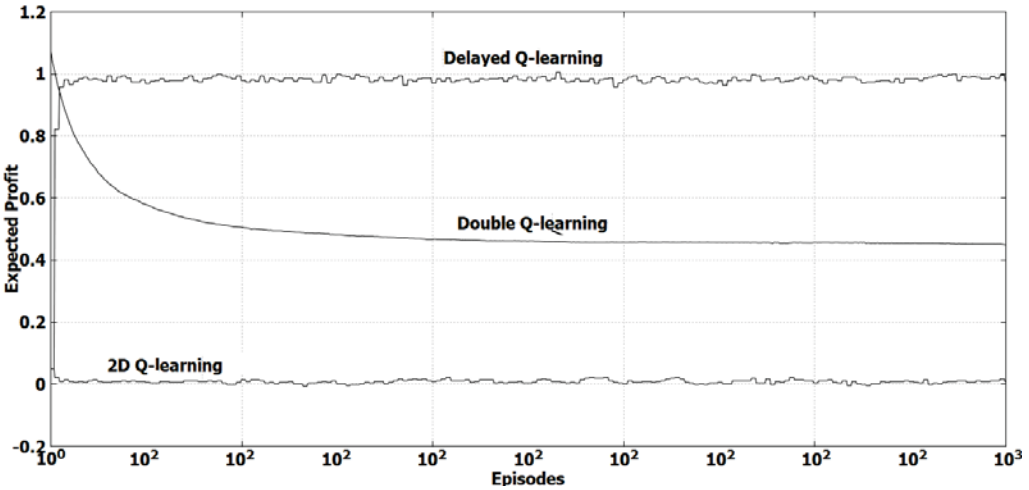

Figure 6: Average action values for all bets found by Delayed Q-learning, Double Q-learning and 2D Q-learning where ϵ_1=0.7.

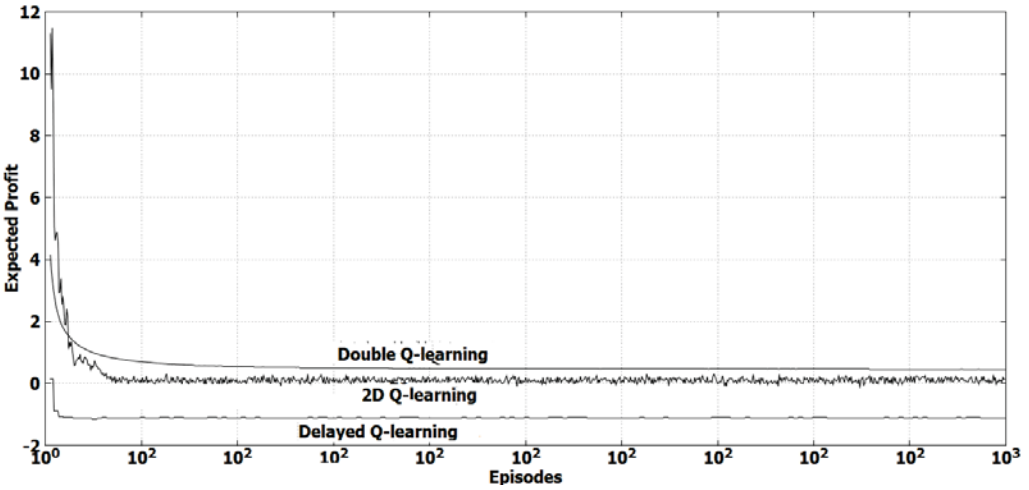

Figure 7: Average action values for all bets found by Delayed Q-learning, Double Q-learning and 2D Q-learning where ϵ_1=-1.

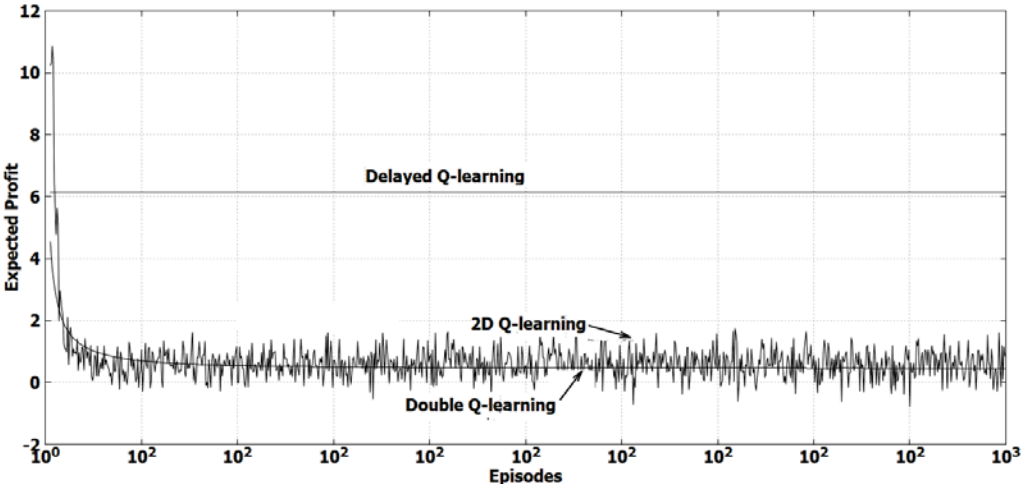

Figure 8: Average action values for all bets found by Delayed Q-learning, Double Q-learning and 2D Q-learning, where ϵ_1=5

7 Conclusion and Future Work

Overestimation of action values is a problem associated with Delayed Q-learning because of the use of a single-estimator method. This paper presented the 2D Q-learning algorithm which is a variation of the Delayed Q-learning algorithm that uses a double-estimator method to estimate the value of the next state. The results of pilot experiments in the roulette game suggest that 2D Q-learning does not overestimate the action values, but actually it may slightly underestimate the action values. The experimental results also suggest that 2D Q-learning performs much better than Q-learning, Delayed Q-learning and Double Q-learning.
Future work includes proving that 2D Q-learning is a PAC-MDP algorithm, and implementing the hybrid Cuckoo search with Simulated annealing algorithm (Alkhateeb and Abed-alguni, 2017) or the hybrid Cuckoo search with Boltzmann Selection algorithm (Abed-alguni and Alkhateeb, 2017) to Q-learning to improve the estimation of its Q-values. Future work also includes applying the double-estimator method to Fitted Q-iteration, which is another variation of Q-learning that suffers from the overestimation problem.

References

Abdallah, S. and Kaisers, M. (2016). Addressing environment non-stationarity by repeating Q-learning updates, *Journal of Machine Learning Research* **17**(46): 1–31.

Abed-alguni, B. H. (2017a). Action-selection method for reinforcement learning based on cuckoo search algorithm, *Arabian Journal for Science and Engineering* pp. 1–15.

Abed-alguni, B. H. (2017b). Bat Q-learning algorithm, *Jordanian Journal of Computers and Information Technology (JJCIT)* **3**(1): 56–77.

Abed-alguni, B. H. and Alkhateeb, F. (2017). Novel selection schemes for cuckoo search, *Arabian Journal for Science and Engineering* **42**(8): 3635–3654.

Abed-alguni, B. H., Chalup, S. K., Henskens, F. A. and Paul, D. J. (2015a). Erratum to: A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers, *Vietnam Journal of Computer Science* **2**(4): 227–227.

Abed-alguni, B. H., Chalup, S. K., Henskens, F. A. and Paul, D. J. (2015b). A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers, *Vietnam Journal of Computer Science* **2**(4): 213–226.

Abed-alguni, B. H., Paul, D. J., Chalup, S. K. and Henskens, F. A. (2016). A comparison study of cooperative Q-learning algorithms for independent learners, *International Journal of Artificial Intelligence* **14**(1): 71–93.

Ali, M. Z., Awad, N. H. and Duwairi, R. M. (2016). Multi-objective differential evolution algorithm with a new improved mutation strategy, *International Journal of Artificial Intelligence* **14**(2): 23–41.

Alkhateeb, F. and Abed-alguni, B. H. (2017). A hybrid cuckoo search and simulated annealing algorithm, *Journal of Intelligent Systems* **0**(0): 1–28.

Azar, M. G., Munos, R., Ghavamzadeh, M. and Kappen, H. J. (2011). Speedy Q-learning, *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, Curran Associates Inc, USA, pp. 2411–2419.

Capen, E. C., Clapp, R. V., Campbell, W. M. et al. (1971). Competitive bidding in high-risk situations, *Journal of petroleum technology* **23**(06): 641–653.

Ernst, D., Geurts, P. and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning, *Journal of Machine Learning Research* **6**(Apr): 503–556.

Even-Dar, E., Mannor, S. and Mansour, Y. (2002). PAC bounds for multi-armed bandit and Markov decision processes, *in* J. Kivinen and R. H. Sloan (eds), *International Conference on Computational Learning Theory*, Springer, Berlin, Heidelberg, pp. 255–270.

Even-Dar, E. and Mansour, Y. (2003). Learning rates for Q-learning, *Journal of Machine Learning Research* **5**(Dec): 1–25.

George, A. P. and Powell, W. B. (2006). Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming, *Machine learning* **65**(1): 167–198.

Gilles, E. and Peroumalnaik, M. (2008). Adapted pittsburgh classifier system: Applying reinforcement learning techniques to meteorological forecasting, *International Journal of Artificial Intelligence* **1**(A08): 96–110.

Hasselt, H. V. (2010). Double Q-learning, *in* J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel and A. Culotta (eds), *Advances in Neural Information Processing Systems 23*, Curran Associates, Inc., pp. 2613–2621.

Kaisers, M. and Tuyls, K. (2010). Frequency adjusted multi-agent Q-learning, *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 309–316.

Lee, D., Defourny, B. and Powell, W. B. (2013). Bias-corrected Q-learning to control max-operator bias in Q-learning, *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, IEEE, pp. 93–99.

Murphy, S. A. (2005). A generalization error for Q-learning, *Journal of Machine Learning Research* **6**(Jul): 1073–1097.

Noda, I. (2009). Recursive adaptation of stepsize parameter for non-stationary environments, *International Workshop on Adaptive and Learning Agents*, Springer, pp. 74–90.

Orselli, T. S. and Dunne, J. (1996). Roulette game apparatus and method with additional betting opportunity. US Patent 5,540,442.

Peng, J. and Williams, R. J. (1996). Incremental multi-step Q-learning, *Machine Learning* **22**(1-3): 283–290.

Precup, R.-E., Angelov, P., Costa, B. S. J. and Sayed-Mouchaweh, M. (2015). An overview on fault diagnosis and nature-inspired optimal control of industrial process applications, *Computers in Industry* **74**: 75–94.

Strehl, A. L., Li, L. and Littman, M. L. (2009). Reinforcement learning in finite MDPs: PAC analysis, *Journal of Machine Learning Research* **10**(Nov): 2413–2444.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J. and Littman, M. L. (2006). PAC model-free reinforcement learning, *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, ACM, New York, NY, USA, pp. 881–888.

Sutton, R.-S. and Barto, A.-G. (1998). *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, USA, 1st Edition.

Sutton, R. S. and Barto, A. G. (2017). *Reinforcement learning: An introduction,*, MIT press Cambridge, 2nd Edition, draft, in progress.

Szita, I. and Szepesvári, C. (2010). Model-based reinforcement learning with nearly tight exploration complexity bounds, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, Omnipress, pp. 1031–1038.

Thaler, R. H. (1988). Anomalies: The winner's curse, *The Journal of Economic Perspectives* **2**(1): 191–202.

Tokic, M. (2010). Adaptive ε-greedy exploration in reinforcement learning based on value differences, *Proceedings of the 33rd Annual German Conference on Advances in Artificial Intelligence*, KI'10, Springer-Verlag, Berlin, Heidelberg, pp. 203–210.

Van den Steen, E. (2004). Rational overoptimism (and other biases), *The American Economic Review* **94**(4): 1141–1151.

Vaščák, J. (2012). Adaptation of fuzzy cognitive maps by migration algorithms, *Kybernetes* **41**(3/4): 429–443.

Vrkalovic, S., Teban, T.-A. and Borlea, I.-D. (2017). Stable Takagi-Sugeno fuzzy control designed by optimization, *International Journal of Artificial Intelligence* **15**: 17–29.

Watkins, C. (1989). *Learning from Delayed Rewards*, PhD thesis, Cambridge University, Cambridge, England.

Yang, X.-S. (2011). Bat algorithm for multi-objective optimisation, *International Journal of Bio-Inspired Computation* **3**(5): 267–274.

Yang, X.-S. and Deb, S. (2009). Cuckoo search via Lévy flights, *World Congress on Nature & Biologically Inspired Computing, 2009. NaBIC 2009.*, IEEE, pp. 210–214.

Zhang, L., Tang, K. and Yao, X. (2015). Increasingly cautious optimism for practical PAC-MDP exploration, *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, AAAI Press, pp. 4033–4040.