Report: Assignment 2

Erik Kuiper[1], Tommy Maaiveld[2], and Whitney Mok[3]

[1] {VU University Amsterdam; 2521056; e2.kuiper@student.vu.nl}
[2] {VU University Amsterdam; 2528586; t.m.maaiveld@student.vu.nl}
[3] {VU University Amsterdam; 2546975; w.mok@student.vu.nl}

Introduction This report describes the presents the results of applying various temporal difference (TD) methods to the ice-world environment. The implementations include SARSA, both Q-learning and Double-Q Learning with ϵ-greedy and soft-max exploration mechanisms, Q-learning using Eligibility Traces and Q-learning with Experience Replay. Output is benchmarked against the results from Value Iteration as reported in our previous work along with a description of the environment.

1 Implementation

The parameters used for all implementations described in this report were a learning rate of $\alpha = 0.01$, a discount factor of $\gamma = 0.9$, an exploration rate of $\epsilon = 0.12$ for all ϵ-greedy algorithms, an eligibility trace discount factor of $\lambda = 0.1$ and a number of planning steps of $N = 10$ for the Dyna-Q algorithm.

Q-learning In Q-Learning, state-action values are incrementally updated in a way that the discounted reward of the next state is obtained with the greedy action taken from that next state. ϵ-greedy exploration was implemented, which chooses an exploratory action with a probability of ϵ, as well as the soft-max exploration strategy, which maps state-action values to a probability distribution for action selection via the soft-max function.

Q-Learning was also extended with Experience Replay and Eligibility Traces. For Experience Replay, we created a Dyna-Q algorithm, which models the environment by sampling from observed experience in order to learn more from the environment (update the value function) after each step of real experience. Our Dyna-Q implementation with a small Experience Buffer ensures that the reward/next state values are updated frequently to account for the stochastic element of the game.

Eligibility Traces were also implemented, which extend upon temporal difference methods by conducting a multiple-state look-ahead. Instead of updating the value of a state based on the reward and a Bellman backup, eligibility traces enable a state to be updated based on a sequence of following states when taking greedy actions (in the case of Q-learning). By discounting these states by a factor of λ and adding the discounted value of any subsequent states via a standard Bellman backup, a more accurate prediction may be made than with a one-state look ahead (such as for TD(0)), where only the following state is evaluated.

SARSA SARSA is a TD method similar to Q-learning, but is on-policy rather than off-policy. In its Q-update, the discounted reward of the next state is not always determined based on the greedy next-action: a soft policy is used. Here we employed epsilon-greedy exploration.

Double Q-learning In the implementation presented here, action selection was based on the sum of the values of the state-action pair in the two Q-tables, for both the version that employed an ϵ-greedy and the one that employed a soft-max exploration strategy. State-values were determined with the averages rather than the sum of the matching values in the two Q-tables.

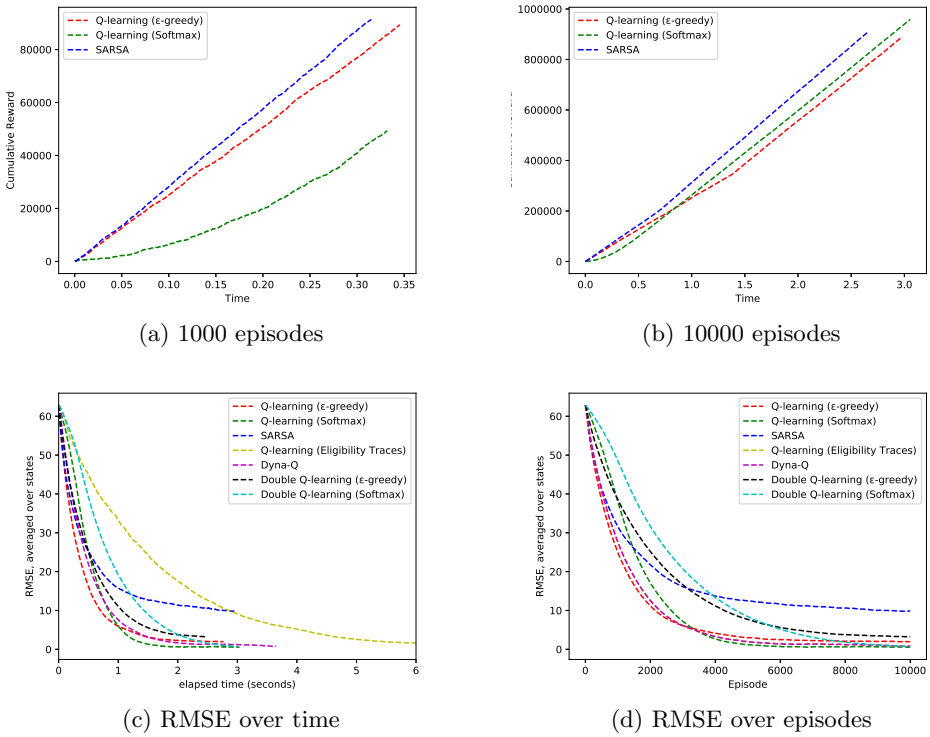
2 Evaluation

MH-8 Q-learning learns the same optimal policy as value iteration. Since Q-learning is a sample-based reinforcement learning method, it will approach the values found by value iteration but never exactly converge on V^{π}. Table 1 gives both the values found by value iteration and the values found by Q-learning after 10000 learning episodes.

MH-9 Plots (a) and (b) in Figure 1 show the online performance of epsilon greedy Q-learning, compared with softmax Q-learning and SARSA. Softmax seems to underperform in the first 1000-2000 iterations, but eventually overtakes the less consistent ϵ-greedy approach. Apparently, it begins with a rather erratic exploration strategy, but quickly converges upon the best balance. The ϵ-greedy algorithm seems to struggle to maximize its reward, as the slope of its reward curve is less steep at first. Presumably, it is unable to find the optimal path past the treasure until many episodes have passed, because its highly stochastic exploration mechanism and slip chance is likely to end the game via this path. It will take a while for it to get proper estimates of the states on this path.

MH-10 Plots (a) and (b) in Figure 1 show that SARSA scores higher in online performance than Q-learning. This is likely because SARSA tries to maximize its score in testing by playing safe (by looking at its next action) instead of stochastic exploration (by looking at the optimal next action and trying to find a more optimal policy) like Q-learning does. This is why its cumulative reward while learning will be higher, but its eventual policy will yield a lower score.

O-11 The RMSE values of all our algorithms can be found in Figure 1 plots (c) and (d). Plot (d) shows the RMSE over episodes, showing that Dyna-Q passes standard Q-learning in performance after around 3000 episodes. Running an experiment to check for an RMSE value of less than 1 indicating convergence shows that regular Q-learning converges after 5000 - 10000 episodes, while Dyna-Q converges after 2500 - 7500 episodes. Because the amount of episodes is highly volatile between runs, a large large boundary. Nevertheless the ranges are quite different, showing that Dyna-Q converges earlier.



(a) 1000 episodes

(b) 10000 episodes

(c) RMSE over time

(d) RMSE over episodes

Fig. 1: Plots of the cumulative reward over time for a given run (a & b), and progressive root mean square error, averaged over 100 runs (c & d).

O-12 Plots (c) and (d) in Figure 1 show that our Q-learning extended with Eligibility Traces takes a lot longer to converge in time compared to regular Q-learning, though it does converge sooner in its amount of episodes. The episodes take a lot longer to complete.

Combining Eligibility Traces and Experience Replay in their standard form is not possible. Since Experience Replay is a model-based method that initializes its update of the model of the environment from a random state, states are not processed in the order they are visited, as TD(λ) can only update eligibility-traced states when visiting subsequent ones. This is necessary for ET, as traced states must be saved in a buffer and are updated as the agent progresses through the environment. A possibility for some of the advantages of Eligibility Traces to be applied in Experience Replay would be to have the agent traverse the virtual environment via some exploration/exploitation mechanism that travels through the environment via its MDP state transition rules. Then, the eligibility trace of a state would be meaningful to states subsequently explored. Multiple agents could roam the environment with different exploration parameters to allow for a full model to be built.

O-13 Though not a major concern in our problem, in real-world reinforcement learning problems, the response signal provided by the environment can be extremely sparse - for example, if a robot has to find an object in a given position, the reward signal is naturally represented by a binary signal indicating success or failure, where all other locations have a reward of 0 and the goal state of searching in the correct location has a reward of 1. This exacerbates the sample-efficiency problem.

One method that could improve the sample-efficiency of Q-learning is the use of Upper-Confidence Bound (UCB) action selection. Unlike ϵ-greedy or soft-max, UCB does take the uncertainty of estimates into consideration in the assignment of action probabilities. As shown by the formula, actions that have been taken a fewer number of times (N) have a larger upper bound and thereby a larger probability of being selected, whilst under unchanged N and progression of time, the choice probability of an action increases: $A_t = argmax_a[Q_t(a) + c\sqrt{\frac{ln(t)}{N_t(a}}]$. This way, it is guaranteed that relatively uncertain estimates, which need more updates still, are more likely to be chosen.

O-14c Double Q-learning is relatively slow to converge (Figure 1), which is typical as two independent estimates need to be updated whilst only one can be updated per step. The lines that represent Double Q-learning do appear relatively smooth, demonstrating how Double Q-Learning has a lower tendency to overestimate state values while learning.

82.38	90.5	100	0
74.14	0	90	0
74.86	88.14	81.45	0
67.71	0	0	0

(a) Value Iteration

82.63	90.62	100	0
66.08	0	90	0
73.1	86.97	81.56	0
66.77	0	0	0

(b) QL (ϵ-greedy)

82.51	90.87	100	0
74.14	0	90	0
76.1	87.92	81.57	0
67	0	0	0

(c) QL (softmax)

76.81	88.39	100	0
65.13	0	87.48	0
51.88	76.7	71.23	0
44.86	0	0	0

(d) SARSA

82.56	90.77	100.31	0
74.1	0	90.21	0
73.98	88.59	81.9	0
64.89	0	0	0

(e) QL(λ)

81.9	90.1	100	0
73.18	0	90	0
66.18	59.9	81	0
59.88	0	0	0

(f) QL with ER

Table 1: State-value tables for all TD methods. Double Q-Learning methods performed similarly to the other Q-Learning algorithms upon convergence.