On-Policy Concurrent Reinforcement Learning

ELHAM FORUZAN, COLTON FRANCO

Outline

- Off- policy Q-learning
- On-policy Q-learning
- Experiments in Zero-sum game domain
- Experiments in general-sum domain
- Conclusions

Off-Policy

Q-learning for an individual learner

Inputs
S is a set of states
A is a set of actions
γ the discount
α is the step size
initialize $Q[S,A]$ arbitrarily
observe current state s
repeat
select action a
observe reward r and state s_{t+1}
$V(s_{t+1})=max_{a'} Q[s_{t+1},a']$
$Q[s,a] \leftarrow (1-\alpha)Q[s,a] + \alpha(r+ \gamma V(s_{t+1}))$

$s \leftarrow s_{t+1}$
until termination

Multi-agent Q-learning

Bimatrix game is a two-player normal form game where

player 1 has a finite strategy set S = {s1, s2, . . . , sm}

player 2 has a finite strategy set T = {t1, t2, . . . , tn}

when the pair of strategies (si , tj) is chosen, the payoff to the first player is aij = u1(si , tj) and the payoff to the second player is bij = u2(si , tj);

u1, u2 are called payoff functions.

Mixed-strategy Nash Equilibrium for a bimatrix game ($M1$, $M2$) is pair of probability vectors (π_1^*, π_2^*) such

$$\pi_1^{*T} M_1 \pi_2^* \geq \pi_1^T M_1 \pi_2^*, \quad \forall \pi_1 \in PD(A_1).$$

$$\pi_1^{*T} M_2 \pi_2^* \geq \pi_1^{*T} M_2 \pi_2, \quad \forall \pi_2 \in PD(A_2).$$

where PD(Ai) = set of probability-distributions over the ith agent's action space

[1] Multiagent Reinforcement Learning☐ Theoretical Framework and an Algorithm, Junling Hu and Michael P☐ Wellman

Multi-agent Q learning

Minimax-Q algorithm in multi-agent zero-sum games, Solving bimatrix game

(M(s),M(-s)).

Littman minimax-Q

$$v_1^t(s_{t+1}) = \max_{\pi \in PD(A)} \min_{o \in O} \pi^T Q_1^t(s_{t+1}, \cdot, o)$$

General-sum games, each agent observes the other agent's actions and rewards and, each agent should update the Q matrix of its opponents and itself. The value function for agent 1 is:

Where the vectors (π_1^*, π_2^*) is the Nash-equilibrium for agents.

$$v_1^t(s_{t+1}) = \pi_1^*(s_{t+1})^T Q_1^t(s_{t+1}, \cdot, \cdot)\pi_2^*(s_{t+1})$$

On-Policy

Q learning for individual agent learning

Inputs

S is a set of states
A is a set of actions
γ the discount
α is the step size

initialize $Q[S,A]$ arbitrarily
observe current state s
select action a using a policy based on Q
repeat
carry out an action a
observe reward r and state s'
select action a' using a policy based on Q
$Q[s,a] \leftarrow (1-\alpha)Q[s,a] + \alpha(r+ \gamma Q[s',a'])$
$s \leftarrow s'$
$a \leftarrow a'$
end-repeat

On-policy versus off-policy learning

Q-learning learns an optimal policy no matter what the agent does, as long as it explores enough. There may be cases where ignoring what the agent actually does is dangerous (there will be large negative rewards). An alternative is to learn the value of the policy the agent is actually carrying out so that it can be iteratively improved. As a result, the learner can take into account the costs associated with exploration.

An **off-policy learner** learns the value of the optimal policy independently of the agent's actions. Q-learning is an off-policy learner. An **on-policy learner** learns the value of the policy being carried out by the agent, including the exploration steps.

SARSA (on-policy method) converges to a stable Q value while the classic Q-learning diverges

[2] Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms, Machine Learning, 39, 287–308, 2000.

Minimax-SARSA learning

The updating rules in minimax-SARSA for multi-agent zero-sum game:

$$v_1^t(s_{t+1}) = Q_1^t(s_{t+1}, a_{t+1}, o_{t+1})$$

$$Q_1^{t+1}(s_t, a_t, o_t) = (1 - \alpha_t) Q_1^t(s_t, a_t, o_t) + \alpha_t [r_t^1 + \gamma Q_1^t(s_{t+1}, a_{t+1}, o_{t+1})]$$

The fix-point for the minmax-Q algorithm for agent 1 is:

$$Q_1^*(s_t, a_t, o_t) = R_1(s_t, a_t, o_t) + E_y[\max_{\pi_1} \min_{o} \pi_1^T Q_1^*(y, \cdot, o)]$$

Nash-SARSA learning

In general sum game the extended SARSA algorithm is:

$$Q_1^*(s_t, a_t, o_t) = R_1(s_t, a_t, o_t) + E_y[\pi_1^{*T} Q_1^*(y, \cdot, \cdot)\pi_2^*]$$

Where vectors (π_1^*, π_2^*) is the Nash-equilibrium for the game $\{Q_1^*, Q_2^*\}$

Minimax-Q(λ)

The Minimax-Q(λ) use the Time Difference (TD) estimator.

TD learns value function V(s) directly. TD is on-policy, the resulting value function depends on policy that is used.

[3] Incremental Multi-Step Q-Learning, Machine Learning, 22, 283-290 (1996)

Sutton's TD(A)

$$\mathbf{r}_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n} + \cdots$$

$$\mathbf{r}_t^{(n)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n \hat{V}_{t+n}^{\pi}(x_{t+n})$$

$$\begin{aligned} V_t &= \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \\ &= r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \cdots) \\ &= r_t + \gamma \sum_{\tau=t+1}^{\infty} \gamma^{\tau-(t+1)} r_\tau \\ &= r_t + \gamma V_{t+1} \end{aligned}$$

$$\mathbf{r}_t^{\lambda} = (1-\lambda)[\mathbf{r}_t^{(1)} + \lambda \mathbf{r}_t^{(2)} + \lambda^2 \mathbf{r}_t^{(3)} + \cdots] = r_t + \gamma(1-\lambda)\hat{V}_t^{\pi}(x_{t+1}) + \gamma\lambda \mathbf{r}_{t+1}^{\lambda}$$

λ represents the trace decay parameter. Higher λ means rewards farther in the future have more weight resulting in longer traces.

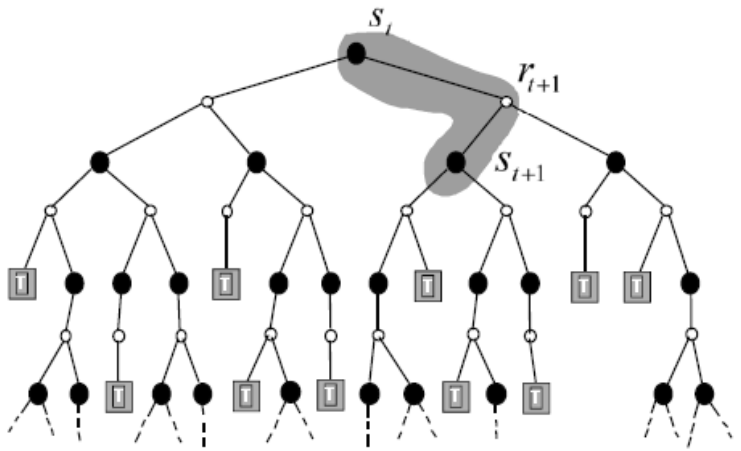
Q-learning versus Monte Carlo

TD(0) $\mathbf{r}_t^0 = r_t + \gamma \hat{V}_t^{\pi}(x_{t+1})$

TD(1) $\mathbf{r}_t^1 = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$

Temporal Difference backup

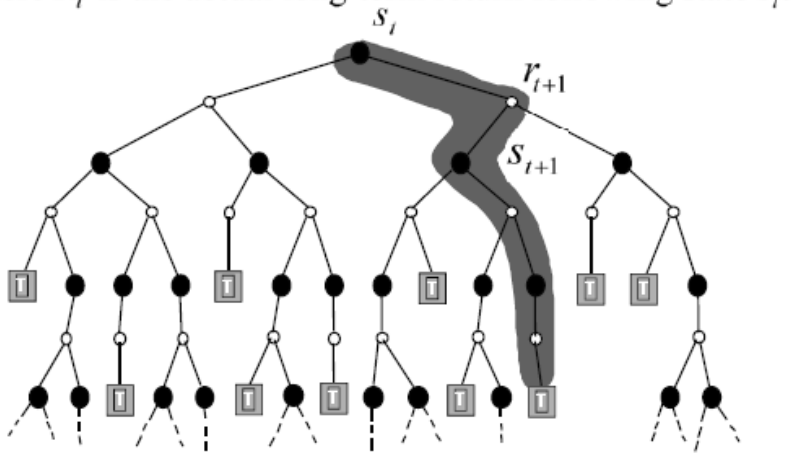
$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$



Monte Carlo

$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

where R_t is the actual long-term return following state s_t.



Experiments in Zero-sum domain

Soccer game : 10 samples each consisting of 10,000 iterations

Environment:

-4x5 grid, with

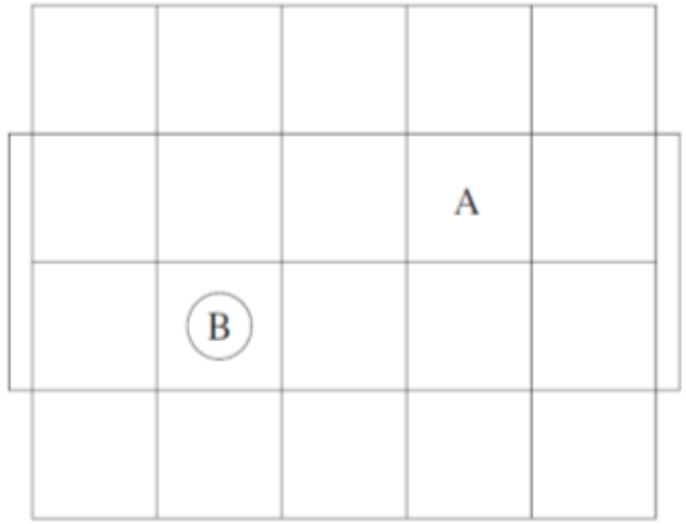
-2: 2x1 goals at each end

Rules:

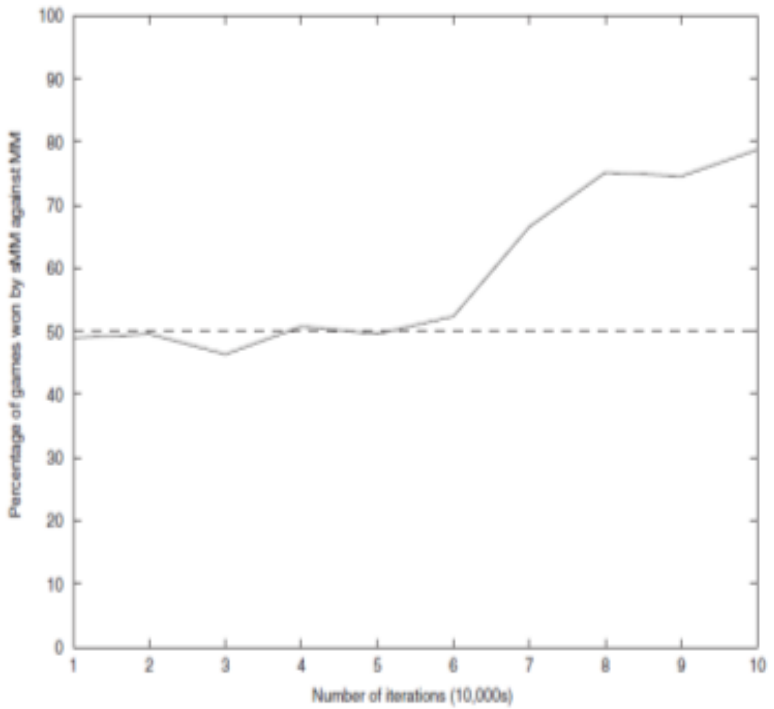
-If A Goes in B Goal or B goes in B Goal {1,-1}

-If B Goes in A Goal or A goes in A Goal {-1,1}

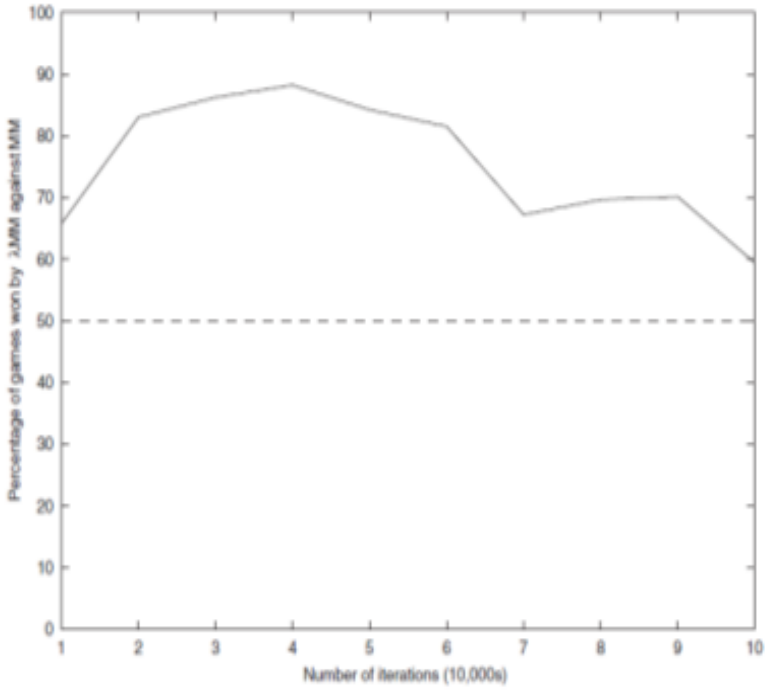
-If a player bumps into another, the stationary player gets the ball



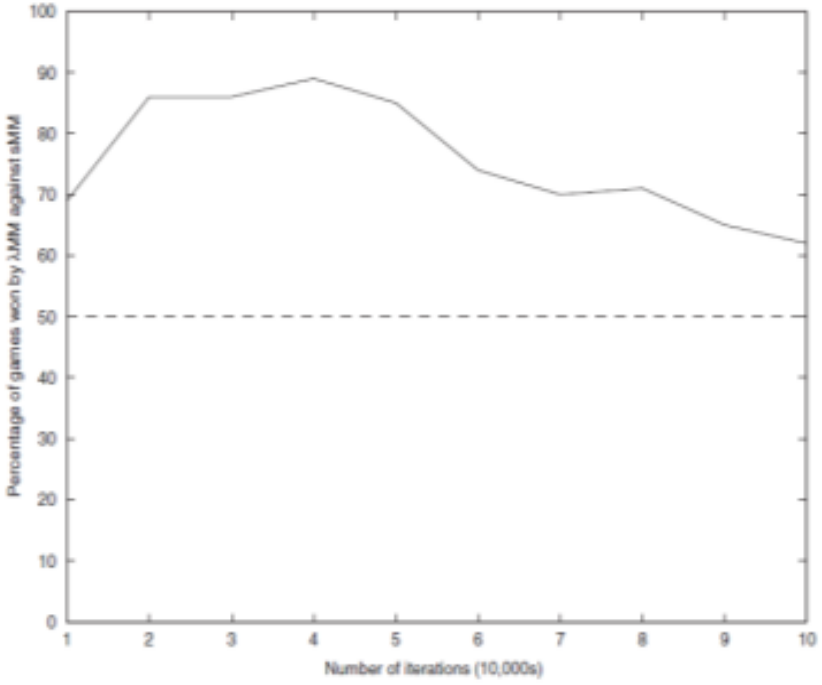
Experiment results



Minimax-SARSA vs. Ordinary Minimax



Minimax-Q(λ) vs. Ordinary Minimax



Minimax-Q(λ) vs. Minimax-SARSA

Results

The basic minimax-Q algorithm initially dominates, but SARSA gradually ends up outperforming the other in the long run.

Q(λ) significantly outperforms minimax in the beginning, however the degree to which it wins over minimax decreases with more iterations.

As in the last example Q(λ) outperforms SARSA, but wins to a lesser degree as more iterations occur.

SARSA outperforms minimax as a result of it's procedures in updating the Q table. SARSA updates its table, according to the actual state is going to travel to and minimax uses the max/min q valued next state to update its table.

Experiments in general-sum

Environment:

-3x4 Grid

-Each cell has 2 rewards 1 for each agent

Rewards:

-Lower left: Agent 1 reward

-Upper right: Agent 2 reward

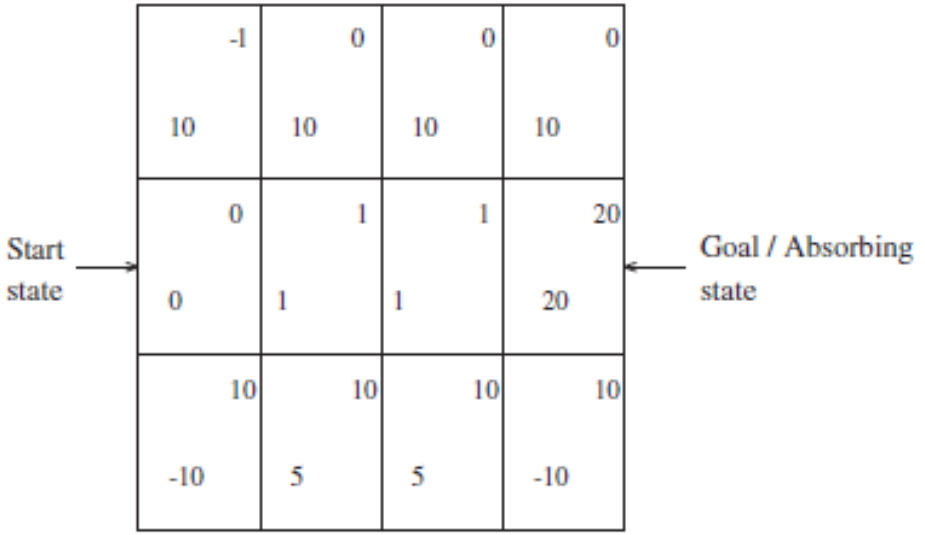
Rules:

-Both agents start in the same cell

-The agents can only transition if they both move in the same direction

Objective:

-Reach the goal state in cell 2x4



General-Sum Experiment

Minimax-Q vs. Minimax-SARSA

Set the exploration probabilities for the agents to 0.2

Analogy: 2 people moving a couch.

They tested 3 different reward generation probability's

- The value 0, means the agents receive nothing until they reach the goal
- The value 1, means the agents receive a reward each time they move

Goal: investigate the effect of infrequent rewards on the covergence of the algorithm.

The average RMS deviation of the learning action were plotted at a sampling rate of 1/1000 iterations.

Result of experiment

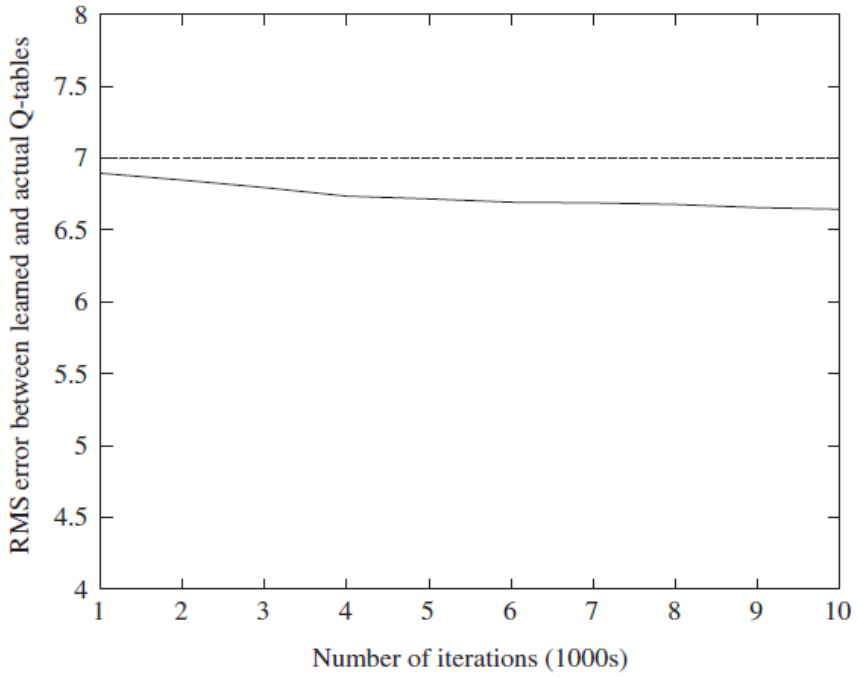


Figure 7. Mean RMS deviation plots of minimax-SARSA (solid) and ordinary minimax-Q for probability of reward-generation = 0.

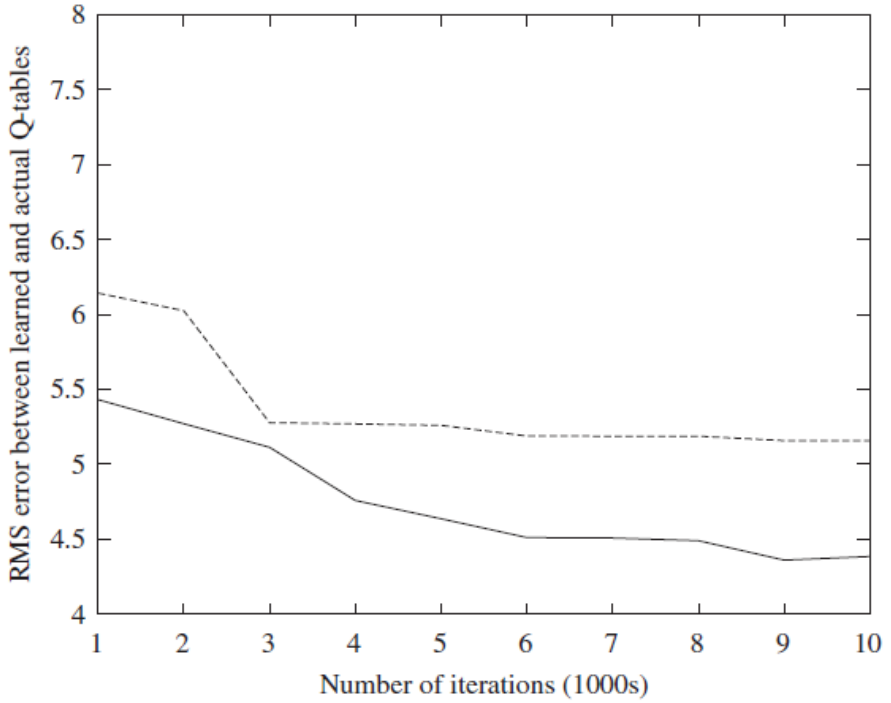


Figure 9. Mean RMS deviation plots of minimax-SARSA (solid) and ordinary minimax-Q for probability of reward-generation = 1.0.

Result analysis

The minimax-SARSA algorithm always approaches minimax values faster than the ordinary minimax-Q algorithm .

The error in all 3 test cases is decreasing monotonically, suggesting that both algorithms will eventually converge.

As expected the error-levels fall with increasing probability of reward-generation as seen in the second graph

Conclusion

Both the SARSA and Q(λ) versions of minimax-Q learn better policies early on than Littman's minimax-Q algorithm.

A combination of minimax-SARSA and Q(λ), minimax-SARSA(λ), would probably be more efficient than either of the two, by naturally combining their disjoint areas of expediency

Reference

[1] On-Policy Concurrent Reinforcement Learning

[2] Multiagent Reinforcement Learning Theoretical Framework and an Algorithm, Junling Hu and Michael P Wellman

[3] Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms, Machine Learning, 39, 287–308, 2000.

[4] Incremental Multi-Step Q-Learning, Machine Learning, 22, 283-290 (1996)