Reinforcement Learning in the Grid World: Q-Learning, Double Q-Learning, SARSA(λ)

Samuel Gerstein
College of Engineering and Computer Science
Florida Atlantic University
Boca Raton, USA
sgerstein2019@fau.edu

***Abstract*—This paper explores multiple reinforcement learning algorithms that can guide an agent through a grid world simulation, specifically Q-learning, double Q-learning, and SARSA(λ). Each of these algorithms were tested on a 3x3 grid world using the OpenAI FrozenLake-v0 gym. The speed of convergence to the optimal policy and the error of the predicted state value for each of these algorithms are evaluated.**

I. Introduction

Temporal difference (TD) control algorithms have shown to be effective in finding the optimal path to the goal in a grid world environment. Although each of these algorithms use a TD step, they differ in how they update their action-value estimates, their emphasis on off-policy versus on-policy learning, and the depth of the TD steps.

Q-learning is an off-policy control algorithm that updates action-value estimates (represented by Q) using a one-step TD. It considers the maximum Q value at the next state (s') and updates the current Q value accordingly. As it is off-policy, the current policy does not affect the greedy action of the agent.

Double Q-Learning is a variant of Q-learning that uses two separate Q-tables–Q_A and Q_B. An action is chosen using a policy for both tables, then the Q-value at that state is updated stochastically. 50% of the time the Q_A table is updated using a greedy action from Q_B, while 50% of the time the Q_B table is updated using a greedy action from Q_A. This variant of Q-Learning is particularly helpful to prevent maximization bias, where an agent chooses a sub-optimal action due to previous sampling. In this case, less bias is introduced as Q-estimates for one table are updated with the Q-estimates of the other.

SARSA(λ) is a TD(λ) variant of SARSA. SARSA is an on-policy control algorithm that uses the current policy to guide future actions. It observes reward r from the current state-action pair (s,a), then uses the policy to choose (s',a'). It updates the Q(s,a) with Q(s',a'), then restricts (s,a) to (s',a') in the next episode. SARSA allows the agent to explore non-greedy actions with respect to the next state. This is superior in on-policy scenarios where the agent has to act conservative, like in cliff walking, where the optimal path is not necessarily the safest path. SARSA(λ) builds off of this algorithm by averaging n-step TD, combining the benefits of both TD and Monte Carlo learning. The inclusion of eligibility traces quickly converges the policy as the agent now has a forward and backwards view of its environment.

II. Algorithm

A. Pseudocode

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Fig. 1. Q-Learning Pseudocode [1]

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Fig. 2. Double Q-Learning Pseudocode [1]

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S, A
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \leftarrow E(S, A) + 1$
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$
      $E(s,a) \leftarrow \gamma \lambda E(s,a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

Fig. 3. SARSA(λ) Pseudocode [1]

B. Parameters

The parameters that can be tuned in the TD Learning algorithms include:

- Episodes: Number of times the agent can reach a terminating state.
- Epsilon (ϵ): Probability that the agent chooses a random action, $\epsilon = 0.01$ in this implementation.
- Discount Factor (γ): Rate that future steps will be discounted, $\gamma = 0.8$ in this implementation.
- Learning Rate (α): Rate that the agent learns new Q-values, $\alpha = 1$ in this implementation.

In addition to these parameters, TD(λ) can be tuned for:

- Lambda (λ): Weight for time since visitation, λ = 0.9 in this implementation.
- Eligibility Traces ($E(s,a)$): Increased when (s,a) is visited using accumulating traces, dutch traces, or by replacing traces. Accumulating traces are used in this implementation.

III. Experiment

A. Setup

This experiment modeled each of these algorithms using a standard 3x3 grid world. The grid world was created using OpenAI's FrozenLake-v0 gym with deterministic actions.

Start		
		Goal (R = 1)

TABLE I
Reward Table

The theoretical state value ($V^*(s)$) table for this grid world was calculated.

0.512	0.640	0.512
0.640	0.800	0.640
0.800	1.00	**R**

TABLE II
V_* Table

Each of these algorithms ran for 100 episodes, and their respective results were averaged over 100 separate trials.

B. Results

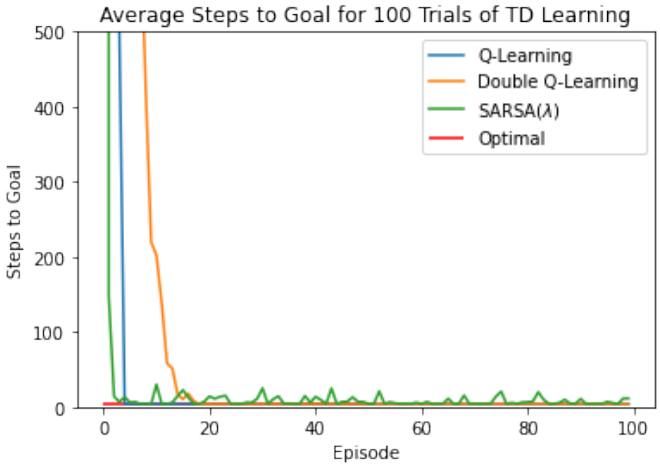

Fig. 4. Comparing each algorithm's ability to find the optimal amount of steps

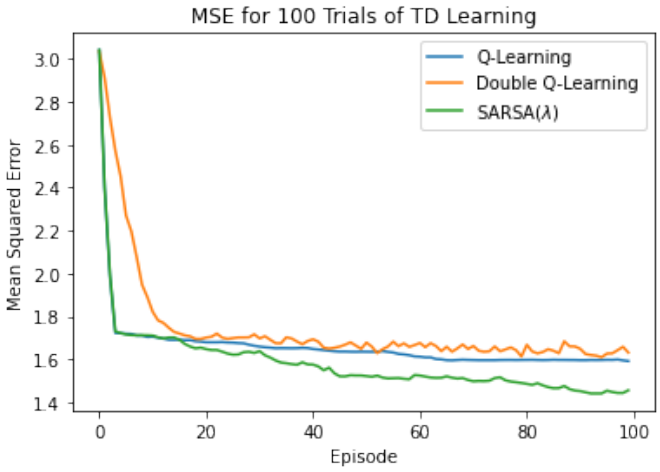

Fig. 5. Comparing each algorithm's error at $V^*(s)$

0.500	0	0
0.64	0.119	0.01
0.800	1.00	0

TABLE III
V^{π} TABLE FOR 100 TRIALS OF Q-LEARNING

0.512	0	0
0.64	0	0
0.800	1.00	0

TABLE IV
V^{π} TABLE FOR 100 TRIALS OF DOUBLE Q-LEARNING

0.489	0.178	0.053
0.505	0.316	0.106
0.680	0.967	0

TABLE V
V^{π} TABLE FOR 100 TRIALS OF SARSA(λ)

C. Discussion

These graphs show the comparison between various TD learning algorithms, both their ability to converge to the optimal policy in their environment and the accuracy of their $V(s)$ estimates. The data shows that all three algorithms relatively converged to the optimal policy with varying speeds and strengths.

The Q-learning algorithm had the worst estimation of $V(s)$ values. This is due to the fact that some sub-optimal states go unexplored by Q-Learning, preventing the agent from discovering greedy actions that are used to calculate action-value estimates. Despite this, the agent was able to still derive the optimal policy for the grid world, as lower action-value accuracy does not necessarily correlate to sub-optimal policy decisions.

The double Q-learning algorithm had the slowest convergence to the optimal policy and the worst mean squared error. Despite performing well in situations where maximization bias is present, this technique can also introduce underestimation bias. This result was validated in Pentaliotis et al., where double Q-learning performed worse than Q-learning on a 10x10 grid world. [2]

The SARSA(λ) algorithm converged the fastest, although it oscillated in its optimal policy. This behavior is expected and somewhat encouraged in on-policy algorithms, as the non-greedy Q-estimates encourage the algorithm to sometimes take a safer path compared to the optimal policy. SARSA(λ) also calculated the most accurate $V(s)$ table. This could be due to the forward/backward view of the eligibility traces, improving the calculation of Q-estimates.

These results display how the subtle differences in TD learning algorithms can equate to major changes in the accuracy and performance of reinforcement learning simulations. Although there was variation in speed of convergence and state value estimations, each algorithm relatively converged to the optimal policy.

Additionally, one interesting causality caused by the exploration versus exploitation dilemma is a high V^{π} error. Many states go unexplored by the agent in Q-learning and double Q-learning, leading to a gross underestimation of state values. This is due to the fact that once the agent converges to a policy, the greedy action step encourages the agent to continue the same policy. This error is less present in SARSA(λ) as its on-policy behavior encourages further exploration. In order to present a fairer comparison between TD methods, future researchers should explore the TD(λ) variation of Q-learning such as Watkin's Q(λ). It remains to be seen whether the increased performance of SARSA(λ) was due to SARSA itself or the n-step averaging of TD(λ). The results of this experiment could also vary depending on the stochasticity and complexity of the simulation environment.

IV. CONCLUSION

This paper explored a 3x3 grid world environment with three temporal difference (TD) algorithms–Q-learning, double Q-learning, and SARSA(λ). The differences in action-value estimation, depth of backups, and on/off-policy evaluations were also explored. The pseudocode for each of these algorithms was provided, along with the parameters defined in the scope of these algorithms. The experiment environment, theoretical state values, and method of algorithm analysis were also presented.

The mean squared error and steps to goal for each of these algorithms showed that SARSA(λ) performed the best with regards to speed of convergence and state value estimations, but had periods of divergence from the optimal policy due to its on-policy nature. Q-learning converged quicker and had better state value accuracy than double Q-learning, which was validated by previous research in the pitfalls of double Q-learning.

These results show the benefits and trade-offs for each of these algorithms. To present a fairer picture, future researchers should explore how these algorithms behave in different environments and with more sophisticated tools like a deep Q network (DQN).

REFERENCES

[1] R. Sutton and A. Barto, Reinforcement learning. The MIT Press.

[2] A. Pentaliotis and M. Wiering, "Variation-resistant Q-learning: Controlling and utilizing estimation bias in reinforcement learning for better performance," Proceedings of the 13th International Conference on Agents and Artificial Intelligence, 2021.

Code can be accessed at: https://tinyurl.com/rlqlearning