CMPUT 655
Assignment 5

September 23, 2024

Dikshant
1877098

---

1. **Temporal Difference**

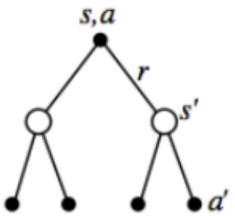
   (a) Write the Q-function updates of iterative PI, MC, Q-Learning, and SARSA.

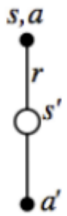
   (b) In two sentences, describe one advantage of TD over DP and one over MC.

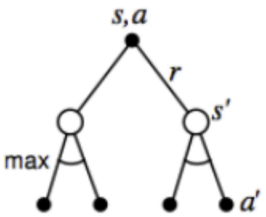
   (c) Is TD more or less susceptible to variance than MC? Explain your answer in one sentence.

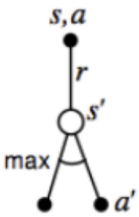
   (d) Q-Learning suffers from overestimation (or maximization) bias. Describe it in one sentence and point out where it comes from in the Q-Learning update.
   • Double Q-Learning prevents overestimation bias. In at most two sentences, explain how it works.
   • Does SARSA also suffers from the overestimation bias? Explain your answer in one sentence.

   (e) What do these backup diagrams represent? Label them with the update they represent.









Ans:

(a) **Q-function updates**:

1. **Iterative Policy Improvement (PI)**:

$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V(s')\right]$$

2. **Monte Carlo (MC)**:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(G_t - Q(s,a)\right)$$

where G_t is the return following the action a in state s.

3. **Q-Learning**:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

4. **SARSA**:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma Q(s',a') - Q(s,a)\right)$$

(b) Advantages of TD:

• TD methods can learn online and make updates after each step, allowing them to be more efficient in terms of computation compared to DP methods, which require full knowledge of the environment and complete value iterations.

• TD can learn from incomplete episodes and does not require the entire return to compute updates, unlike Monte Carlo methods, which depend on complete episodes.

(c) TD methods are generally less susceptible to variance than MC methods because they update estimates based on the current value estimate and a bootstrapped target, rather than relying on the potentially high variance of complete episode returns.

(d) Q-Learning suffers from overestimation bias because it consistently uses the maximum Q-value across actions in its update rule ($\max_{a\prime} Q(s\prime, a\prime)$), leading to an inflated estimate of the value of the chosen action.

• **Double Q-Learning** addresses overestimation bias by maintaining two separate Q-value estimates and using one to select actions while the other is used to evaluate the selected action, effectively decoupling the action selection from the value estimation. This approach reduces the likelihood of

overestimating the value of actions by averaging the two Q-value estimates, leading to more accurate updates.

• **SARSA** does not suffer from the same overestimation bias as Q-Learning because it uses the action actually taken (policy-based) in the update, rather than always selecting the maximum Q-value, which can help mitigate the bias introduced by overestimation.

(e) V_π(top-left) — SARSA (top-right) — Q_*(bottom-left) — Q-learning (bottom-right)

2. *If V changes during the episode, then (6.6) only holds approximately; what would the difference be between the two sides? Let V_t denote the array of state values used at time t in the TD error (6.5) and in the TD update (6.2). Redo the derivation above to determine the additional amount that must be added to the sum of TD errors in order to equal the Monte Carlo error.*

Ans: equation 6.2 becomes: $V_{t+1}(S_t) = V_t(S_t) + \alpha\left[R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)\right]$ and equation 6.5 becomes: $\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$

$$\begin{aligned}
G_t - V_t(S_t) &= R_{t+1} + \gamma G_{t+1} - V_t(S_t) + \gamma V_t(S_{t+1}) - \gamma V_t(S_{t+1}) \\
&= \delta_t + \gamma(G_{t+1} - V_t(S_{t+1})) \\
&= \delta_t + \gamma\left(G_{t+1} - V_{t+1}(S_{t+1})\right) + \gamma\theta_{t+1} \\
&\text{where, } \theta_t = \alpha\left[R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)\right] \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\left(G_{t+2} - V_{t+2}(S_{t+2})\right) + \gamma\theta_{t+1} + \gamma^2\theta_{t+2} \\
&= \sum_{k=t}^{T-1}[\gamma^{k-t}\delta_k + \gamma^{k-t+1}\theta_{k+1}]
\end{aligned}$$

3. *Suppose action selection is greedy. Is Q-learning then exactly the same algorithm as Sarsa? Will they make exactly the same action selections and weight updates?*

Ans: When using greedy action selection, Q-Learning and SARSA may appear to behave similarly, as both choose the action that currently seems optimal based on their respective action-value functions. However, their updates differ: SARSA updates its values based on the actual action taken, which might not be greedy if exploration occurs. In contrast, Q-Learning always updates its values assuming

the best possible next action, as though following a fully greedy policy without considering exploration.

4. *(a)* ***Part 1***

Implement Q-Learning, SARSA, and Expected SARSA where exploration is maintained using an ϵ-greedy policy. For Expected SARSA, the policy used for the expectation in the TD target is the same ϵ-greedy used to explore.

- *γ = 0.9*
- *max_steps = 10,000*
- *Episode horizon: 10*
- *ϵ starts at 1.0 and decays after every environment steps:*

$\epsilon = max(\epsilon - 1.0/max\_steps, 0.01)$.

- *Use a learning rate α that starts at 0.1 and decays after every environment steps: $\alpha = max(\alpha - 0.1/max\_steps, 0.001)$*
- *Log the following stats:*
  - *At every step, log the TD error of the current sample. Note that the TD error depends on the algorithm used, e.g., $|r_t + \gamma max_a Q(s_{t+1}, a - Q(s_t, a_t)|$ for Q-Learning.*
  - *Every 100 steps, log the Bellman error over the whole Q-function (not just one state-action pair). This depends on your target policy for SARSA. Also, compute the **mean** error this time, not the sum as in the previous assignments (it's easier to compare the mean Bellman error against the TD error of a single sample).*
  - *Use the provided bellman_q() to compute the true Q-function. It computes Q iteratively and it's much faster than the exact method (only because $\gamma = 0.9$ and the iterative method converges quickly).*
  - *Every 100 steps, log the expected return that the agent would get following the current greedy policy (use expected_return()).*
  - *Remember to pass env_eval to the function and not env because the evaluation resets the environment and starts new episodes.*

Plot the stats next to each other at the end with the provided snipped, averaged over 50 seeds. You'll have a total of 9 curves (3 algorithms and 3 different Q-function initialization (-10, 0, 10)).

Discuss your results:

- *Is one initialization always better for all algorithm? Or is one better for some but worse for others?*
- *Is there a discrepancy between the TD error and the Bellman error plots? Why?*
- *What about the expected return?*
- *Can the policy act optimally (i.e., achieve maximum expected return) even when*

it hasn't learned the optimal Q-function yet? Why?
• Is there a best algorithm, i.e., one that always "performs" best regardless of the initialization?
• What criterion would you use to define "performance"? TD error, Bellman error, or expected return? Why?
Be concise and answer each bullet point in 1-2 sentences

(b) ***Part 2***
Repeat the experiment but this time have stochastic rewards. In your experiment loop, make the environment with the argument reward_noise_std=3.0 to add Gaussian noise with 0 mean and 3.0 standard deviation, and run it again.
• Note that the matrix $\boldsymbol{R}$ *used to compute the true Q-function is still correct, because it denotes the expectation over rewards (and the Gaussian noise has 0 mean).*
• For the same reason, env_eval is made without noise.
Discuss your results:
• How do plots compare to Part 1? Is the trend different now? If yes, why?
Be concise and answer each bullet point in 1-2 sentences.

(c) ***Part 3***
Modify your code to implement Double Q-Learning, SARSA, and Expected SARSA.
• We suggest to copy-paste your TD function and modify the TD target rather than having more "if ... else" in the same td() function.
• The book does not have pseudocode for Double SARSA and Expected SARSA. You need to write them on your own, but just know that they follow the same principle of Double Q-Learning, i.e., the TD target is computed using both Q-functions. That is, when you update Q1 you select the next action according to Q1 but the next state-action value comes from Q2.
• Log the Bellman error with respect to the mean of the two Q-functions.
• Run experiments with deterministic reward only.
Discuss your results:
• How do plots compare to Part 1? Is there an initialization that benefits more/less from the double estimator?
Be concise and answer each bullet point in 1-2 sentences.

Ans:

(a) **Part 1**
• **Dependence on Initialization**: No, one initialization is not always better for all algorithms. SARSA and expected SARSA performed good with $Q_o = -10$, while Q-learning performed best with With $Q_o = 0$.

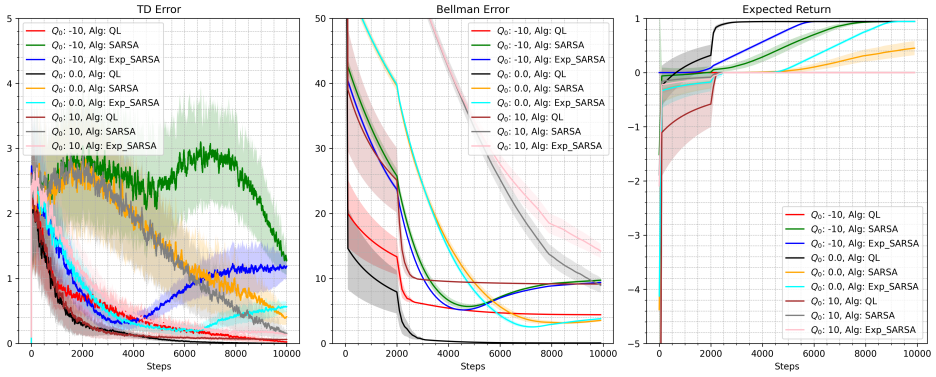


Figure 1: TD algorithms

• **TD error vs Bellman error**: Yes, there is a discrepancy between TD error and Bellman error plots. TD error measures the temporal difference at each step, reflecting how well the immediate update matches the expectation, while Bellman error compares the Q-function globally to the optimal values. In some cases, TD error may reduce locally, while Bellman error remains high due to larger, uncorrected errors in the overall Q-function.

• **Expected return**: The policy can act near-optimally before the Q-function is fully learned because it relies on the relative ordering of Q-values rather than their exact values. As long as the best actions are consistently chosen, the agent can achieve good returns, even if the Q-values themselves are still inaccurate.

• **Best algorithm**: Expected SARSA performs best among all three algorithms across different initializations. While it doesn't seem to achieve the expected reward for $Q_o = 10$ within 10000 steps, it reaches optimal behavior after 50000 steps. I have attached the plots with 50000 steps where we can see that expected SARSA got the required expected return fastest among all.

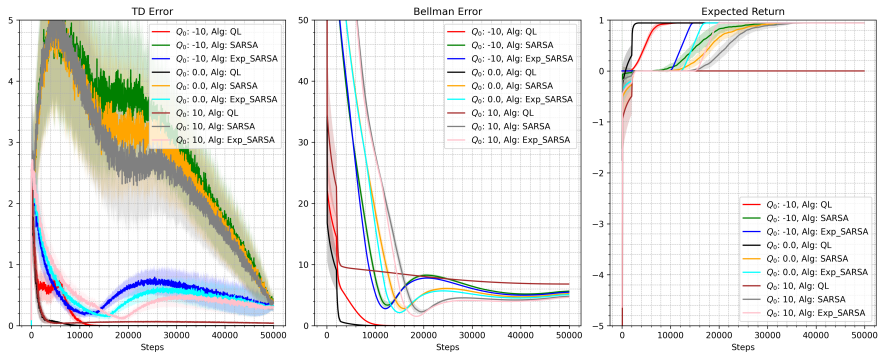


Figure 2: TD algorithms with 50000 steps

• **Performance measure**: The expected return is the best criterion for defining performance, as it directly reflects how well the agent is maximizing cumulative rewards, which is the ultimate goal.

(b) **Part 2 • Adding stochastic rewards**: With stochastic rewards, the bellman error tends to fluctuate more due to the variability in rewards, leading to noisier Q-value estimates. This slows down convergence and makes the learning process less stable, especially in early stages.
The trend is different because the stochastic noise in rewards introduces additional uncertainty. Even though the environment's true Q-function remains the same (since the noise has a 0 mean), the agent's estimates are affected by the randomness in rewards, causing deviations from the true Q-function during updates.

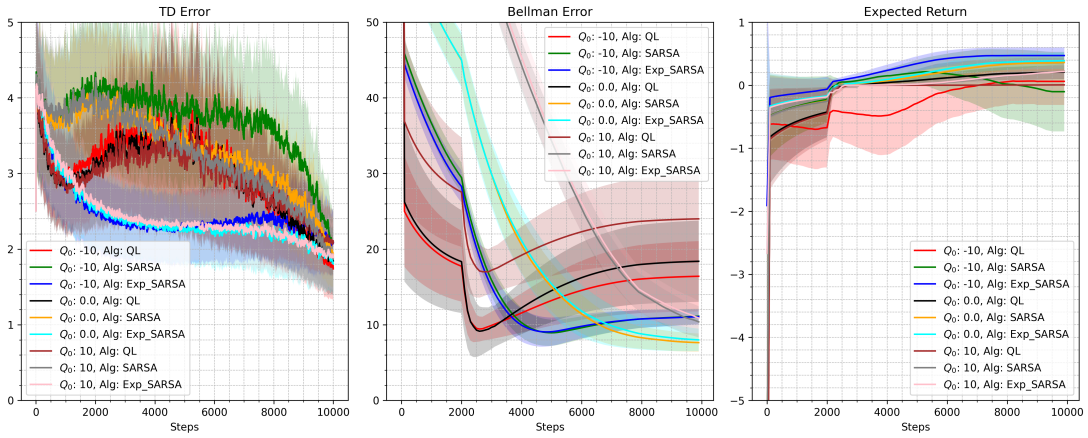


Figure 3: TD algorithms with stochastic rewards

(c) **Part 3**
• **Comparison to Part 1**: Using two Q values helps in decoupling the updates. By decoupling action selection from evaluation, the double estimators mitigate overestimation bias, leading to more stable Bellman error curves.
• **Initialization impact**: Expected rewards doesn't change much even after decoupling so I don't think it changes too much compared to part 1.

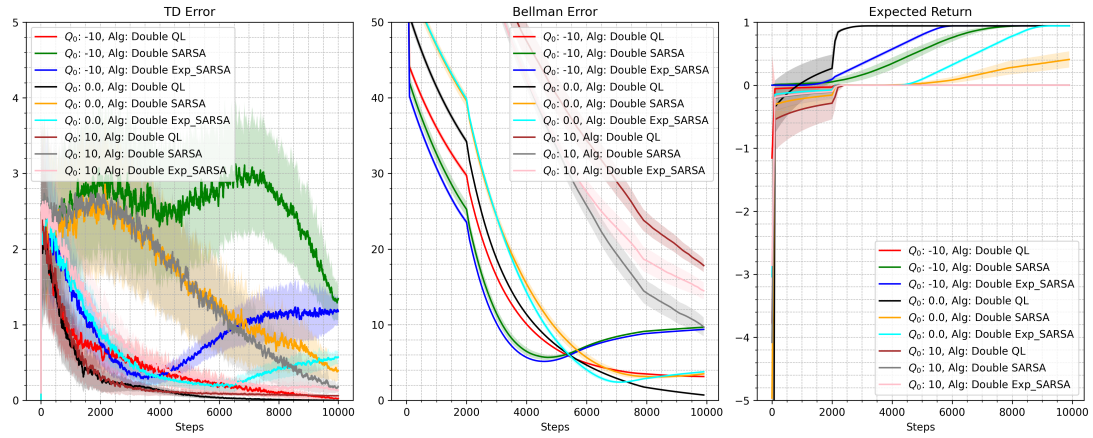


Figure 4: Double TD algorithms

Code can be accessed on github from here: TD algorithms