Temporal-Difference Learning

TD Prediction

- Both TD and Monte Carlo methods use experience to solve the prediction problem
  - TD methods learn from current predictions, and don't wait for actual returns
- TD(0) - One-step TD
  - Updates value function immediately after receiving state-value pair
- TD methods combine the sampling of Monte Carlo with the value bootstrapping of DP
- TD Error
  - Difference between the estimated value of S_t and the better estimate of $R_{t+1} + \gamma V(S_{t+1})$
- **a** is the learning rate

Monte Carlo Value Estimation

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[G_t - V(S_t) \Big]$$

TD Value Estimation

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \Big]$$

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

Advantages of TD Prediction Methods

- TD methods update their estimates based in parts on other estimates (they bootstrap)
  - They learn a guess from a guess
- In practice, TD methods converge faster than MC methods
  - TD only needs to wait one time step to update its values - not an entire episode
  - TD methods are much more feasible than Monte Carlo methods for approximating solutions for problems with large state spaces

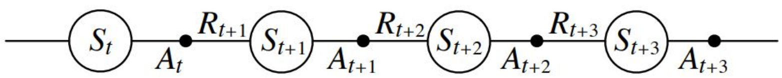
Optimality of TD(0)

- Batch Updating
  - Value function is only updated after processing a batch of training data (episode)
  - Monte Carlo methods find the estimates that minimize MSE on the training set
  - TD(0) finds the estimates that would be correct for the maximum-likelihood model (the model most likely to generate the data) of the Markov Process
- Certainty-Equivalence Estimate
  - Estimate of the value function is computed as if the model (observed from experience) was exactly correct - i.e. known with certainty
  - What TD(0) converges to, and why TD(0) converges faster than Monte Carlo

Sarsa: On-Policy TD Control

- Learns action-value function for the policy (epsilon-greedy) it is following
    - Learns from quintuples of: state-action reward → state-action (Sarsa)
- TD(0) is used to estimate V, then epsilon-greedy policy improvement is used
- Sarsa methods learn *during the episode*, and therefore move out of poor policies quickly

Improvement Process of Sarsa



Sarsa Action-Value Update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  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)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning: Off-Policy TD Control

- The learned action-value function directly approximates the optimal action-value function, independent of the policy being followed
    - Updates action-value function from the greedy policy
    - Chooses action from an epsilon-greedy policy
    - Converges slower than Sarsa, but can continue learning while changing policies

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

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

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (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

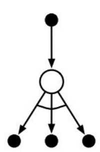
Expected Sarsa

- Like Q-Learning, but uses the expected value of state-action pairs, rather than the maximum of state-action pairs
- Moves deterministically in the same direction as Sarsa moves in expectation
- Expected Sarsa is more complex computationally than Sarsa, but it eliminates the variance due to the random selection of A_{t+1}
- Unlike Sarsa, Expected Sarsa can set **a = 1** without suffering any consequences, and can therefore obtain better short-term results
- If Expected Sarsa is is used as an off-policy algorithm (exploratory behavior policy and greedy decision policy) than Expected Sarsa is exactly like Q-Learning
    - Expected Sarsa subsumes and generalizes Q-Learning while reliably improving over Sarsa - at the cost of a small additional computational cost

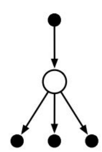
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

Backup Diagrams for Q-Learning and Expected Sarsa

Q-Learning takes the maximum of action-value pairs (represented by the arc)
Expected Sarsa uses the expected value of action-value pairs



Q-learning



Expected Sarsa

Maximization Bias and Double Learning

- Maximization Bias
    - Both Q-Learning and Sarsa use maximization to construct their target policies
    - This can lead to overestimating the Q value for actions with random rewards
- Double Learning
    - Two Q functions - Q_1 and Q_2 0 are independently learned
    - One function is used to determine the maximizing action, and the second to estimate its value - this removes maximization bias
    - Either Q_1 or Q_2 is updated randomly with the following equation

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \operatorname*{argmax}_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big]$$

Double Q-learning

Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
    else:
      $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Games and Afterstates

In games like tic-tac-toe, afterstate value functions would assess identical positions the same regardless of how the positions was reached, and the learning would transfer