Temporal Difference Learning

Rick Valenzano and Sheila McIlraith

Acknowledgements

- Based on textbook by Sutton and Barto
- Also used slides from Adam White

Outline

- TD updates insteads of MC or DP
- TD prediction
- Sarsa on-policy control
- Q-learning off-policy control

State-Value Updates

- Recall the update template

NewEstimate =
LastEstimate + StepSize · (Target – LastEstimate)

- Target is what we want
  - Or an estimate (*i.e.* sample) of what we want
- Taking a step toward that target

Policy Evaluation

- Consider the prediction problem
  - Specifically, trying to compute $v_\pi(s)$

Policy Evaluation

$v_\pi(s) = \mathrm{E}_\pi[G_t | S_t = s]$

Policy Evaluation

$$v_\pi(s) = \mathrm{E}_\pi[G_t | S_t = s]$$
$$= \mathrm{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right]$$

Policy Evaluation

$$v_\pi(s) = \mathrm{E}_\pi[G_t | S_t = s]$$
$$= \mathrm{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right]$$
$$= \mathrm{E}_\pi\left[R_{t+1} + \gamma \cdot \sum_{i=1}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right]$$

Policy Evaluation

$$\begin{aligned} v_\pi(s) &= \mathrm{E}_\pi[G_t | S_t = s] \\ &= \mathrm{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right] \\ &= \mathrm{E}_\pi\left[R_{t+1} + \gamma \cdot \sum_{i=1}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right] \\ &= \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s] \end{aligned}$$

Policy Evaluation

$$\begin{aligned} v_\pi(s) &= \mathrm{E}_\pi[G_t | S_t = s] \\ &= \mathrm{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right] \\ &= \mathrm{E}_\pi\left[R_{t+1} + \gamma \cdot \sum_{i=1}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right] \\ &= \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s] \end{aligned}$$

Monte Carlo State Update

$v_\pi(s) = \mathrm{E}_\pi[G_t | S_t = s]$

Leads to the following update rule:

$$V(s) = V(s) + \alpha \cdot (\mathrm{E}_\pi[G_t | S_t = s] \; - V(s))$$

where α is a constant step size

Monte Carlo State Update

$v_\pi(s) = \mathrm{E}_\pi[G_t | S_t = s]$

Leads to the following update rule:

$$V(s) = V(s) + \frac{1}{N(s)} \cdot (G_t - V(s))$$

G_t is being used as an estimate of $\mathrm{E}_\pi[G_t | S_t = s]$

Policy Evaluation

$$v_\pi(s) = \mathrm{E}_\pi[G_t|S_t = s]$$
$$= \mathrm{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right]$$
$$= \mathrm{E}_\pi\left[R_{t+1} + \gamma \cdot \sum_{i=1}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right]$$
$$= \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s]$$

Policy Evaluation

$$v_\pi(s) = \mathrm{E}_\pi[G_t|S_t = s]$$
$$= \mathrm{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right]$$
$$= \mathrm{E}_\pi\left[R_{t+1} + \gamma \cdot \sum_{i=1}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right]$$
$$= \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s] \quad \longleftarrow$$

Dynamic Programming Update

$$v_\pi(s) = \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s]$$

Leads to the following update rule

$$V(s) = V(s) + \alpha \cdot (\mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s] - V(s))$$

Dynamic Programming Update

$$v_\pi(s) = \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s]$$

Leads to the following update rule

$$V(s) = V(s) + (\mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s] - V(s))$$

If $\alpha = 1$

Dynamic Programming Update

$v_\pi(s) = \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) |\, S_t = s]$

Leads to the following update rule

$$V(s) = V(s) + (\mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) |\, S_t = s] - V(s))$$

If $\alpha = 1$

Dynamic Programming Update

$v_\pi(s) = \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) |\, S_t = s]$

Leads to the following update rule

$$V(s) = \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) |\, S_t = s]$$

If $\alpha = 1$

Dynamic Programming Update

$v_\pi(s) = \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) |\, S_t = s]$

Leads to the following update rule

$$V(s) = \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) |\, S_t = s]$$
$$= \sum_a \pi(a|s) \cdot \left[\sum_{s',r} p(s',r|s,a)[r + \gamma \cdot V(s')] \right]$$

where $V(s')$ is an estimate of $v_\pi(s')$

Dynamic Programming Update

$$V(s) = \sum_a \pi(a|s) \cdot \left[\sum_{s',r} p(s',r|s,a)[r + \gamma \cdot V(s')] \right]$$

- **Bootstrapping**: not just learning from outcomes, but on other value function estimates
- Explicitly uses knowledge of the reward function and the transition probabilities

Policy Evaluation

$$v_\pi(s) = \mathrm{E}_\pi[G_t | S_t = s]$$
$$= \mathrm{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right]$$
$$= \mathrm{E}_\pi\left[R_{t+1} + \gamma \cdot \sum_{i=1}^{\infty} \gamma^i \cdot R_{t+i+1} \mid S_t = s\right]$$
$$= \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s]$$

Temporal Difference Evaluation

$$v_\pi(s) = \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s]$$

- Don’t have explicitly defined model
- But when we take an action, we get a reward R_{t+1} and a new state s'

TD Target: $R_{t+1} + \gamma \cdot V\,(s')$

- Estimate of return we will get

TD(0) Update

$$v_\pi(s) = \mathrm{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | \, S_t = s]$$

Leads to the following update rule

$$V(s) = V(s) + \alpha \cdot (R_{t+1} + \gamma \cdot V(s') - V(s))$$

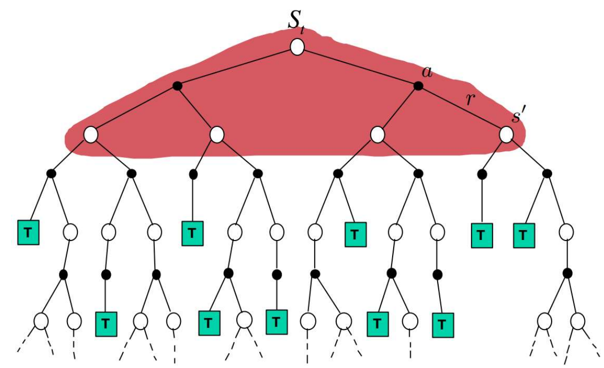
Temporal Difference Update

- Consider predicting the temperature on Saturday
  - Have radar info, wind info, air pressure, …
  - Defines the state
- Prediction says 18 degrees
- Could wait until Saturday, then adjust how we predict temperature in the Thursday state
  - Like an MC update

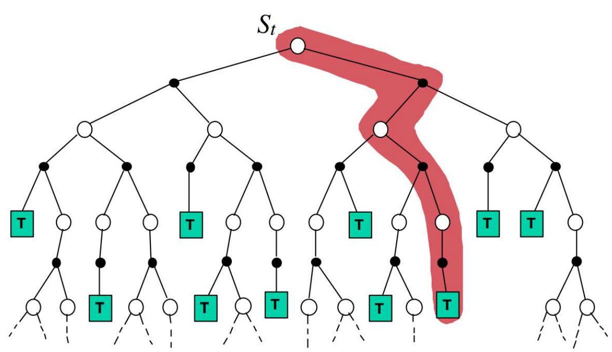
Temporal Difference Update

- Consider predicting the temperature on Saturday
  - Have radar info, wind info, air pressure, …
  - Defines the state
- Prediction says 18 degrees
- On Friday, can use our method to update the way the prediction was made on Thursday
  - Like a temporal difference update

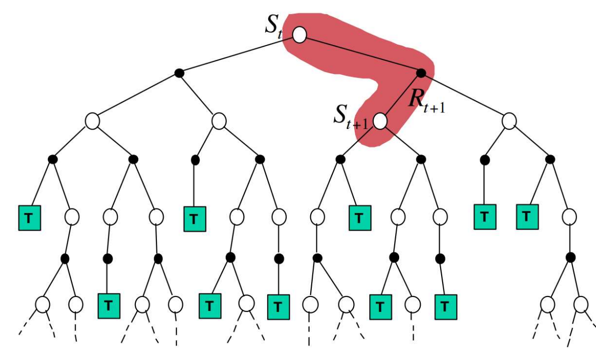
Dynamic Programming Update



Monte Carlo Update



TD(0) Update



TD(0) Policy Evaluation

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
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

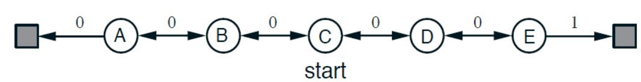
TD(0) Policy Evaluation

- Consider predicting the temperature on Saturday
  - Have radar info, wind info, air pressure, …
  - Defines the state
- Prediction says 18 degrees
- On Friday, can use our method to update the way the prediction was made on Thursday
  - Like a temporal difference update

TD(0) vs Monte Carlo

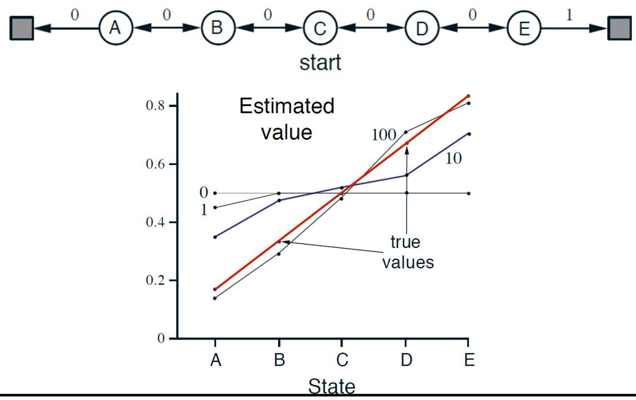
- Both converge to v_π in the limit
- TD does bootstrapping, MC does not
- MC must wait until the end of episodes
  - Episodes can be very long
  - Can't handle continuing domains
- TD updates occur after every action
  - Can be used on continuing domains
  - Can be implemented in an online fashion

TD(0) vs MC on RW Domain

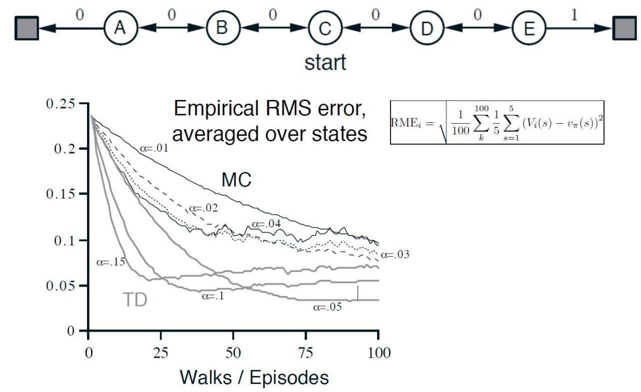


- C is the start state
  - Termination on either end
- Go left or right with equal probability
- $v_\pi = [\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}]$

TD(0) vs MC on RW Domain

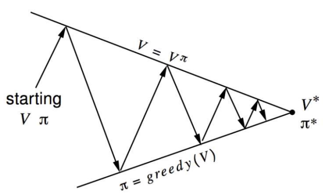


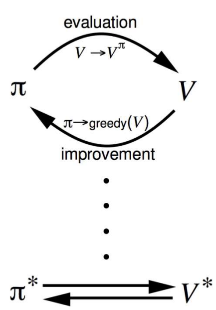
TD(0) vs MC on RW Domain



TD Control

- Will once again do Generalized Policy Iteration





Policy evaluation Estimate v_π
e.g. Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
e.g. Greedy policy improvement

TD Q-Value Updates

- Will use TD(0) updates on Q-values
  - Recall that we want estimates to help choose actions

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

TD Q-Value Updates

- Will use TD(0) updates on Q-values
  - Recall that we want estimates to help choose actions

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

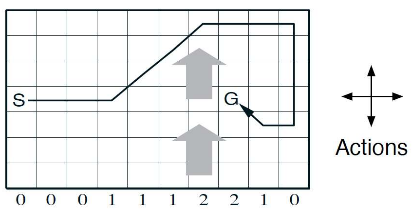
- Where the name Sarsa comes from
  - On-policy TD control algorithm

Sarsa On-Policy Control

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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)
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

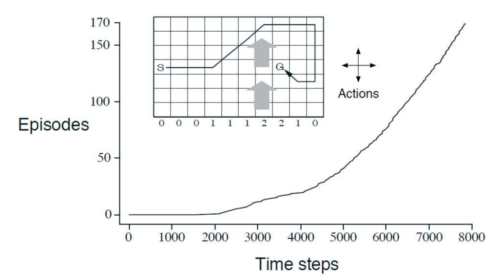
Sarsa on Windy Gridworld

- 4-connected grid, but wind pushes the agent up
  - Reward of -1 on every time step before goal is reached



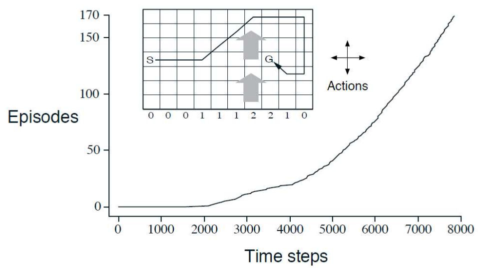
Sarsa on Windy Gridworld

- Use $\epsilon = 0.1, \gamma = 1, \alpha = 0.5$, initial $Q(s, a) = 0$



Sarsa on Windy Gridworld

- Use $\epsilon = 0.1, \gamma = 1, \alpha = 0.5$, initial $Q(s,a) = 0$



- MC would really struggle due to episode lengths

Sarsa Properties

- Sarsa converges to the best ϵ-greedy policy
- Can also get it to converge to optimal policy
  - Each state-action pair visited infinite number of times
  - ϵ converges to 0 over time (*i.e.* $\epsilon = 1/t$)

Off-Policy TD Learning

- Sarsa uses the action it will select for bootstrapping

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

This is the action that will be chosen by ϵ-greedy

- It is not the action it "should" have chosen

Q-Learning Update

- Q-learning uses the following update

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

This is the action that should have been chosen

- ϵ-greedy may pick something else

Q-Learning Update

- Q-learning uses the following update

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

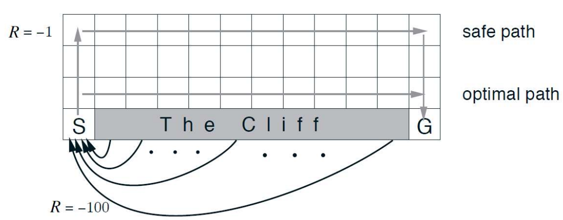
- Directly approximates q^*, regardless of policy used
  - Allows for proof of convergence to q^* if the followed policy guarantees all state-action pairs are seen
  - This is why it is **off-policy**

Q-Learning

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  until S is terminal

Sarsa vs. Q-Learning

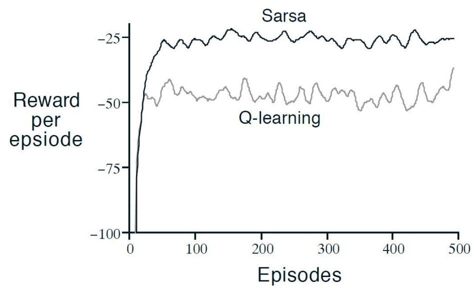
- Consider grid world, -1 per step, -1000 if fall off cliff
  - 4-connected, deterministic actions



Sarsa vs. Q-Learning

- Both converge to “their optimal” policy
  - So what is happening?

Sarsa vs. Q-Learning



- Both converge to “their optimal” policy
  - So what is happening? Sarsa takes ϵ into account

Summary

- TD update online based on one-step returns
  - Can be used for episodic and continuing tasks
- TD prediction using TD updates
  - Often faster than MC
- Sarsa on-policy control
- Q-learning off-policy control