Eligibility Traces

Rick Valenzano and Sheila McIlraith

Acknowledgements

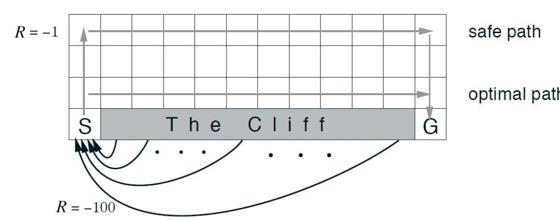
- Based on textbook by Sutton and Barto
- Also used slides from Adam White

Outline

- 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

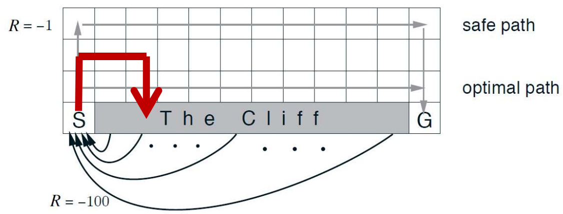
Cliff World

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



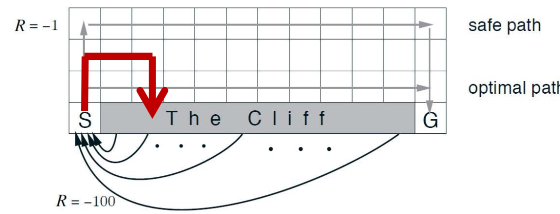
Cliff World

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



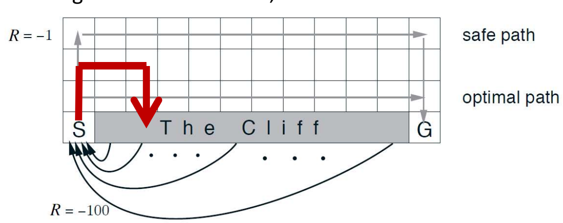
Cliff World

- Consider grid world, -1 per step, -1000 if fall off cliff
  - 4-connected, deterministic actions
  - MC gives all states on path -1000 + -g



Cliff World

- Consider grid world, -1 per step, -1000 if fall off cliff
  - 4-connected, deterministic actions
  - TD gives last state -1000, others -1



$\boldsymbol{n}$-Step Returns

- TD only looks at immediate outcome
  - Reward and value function of resulting state

- Could look a couple of steps along the episode

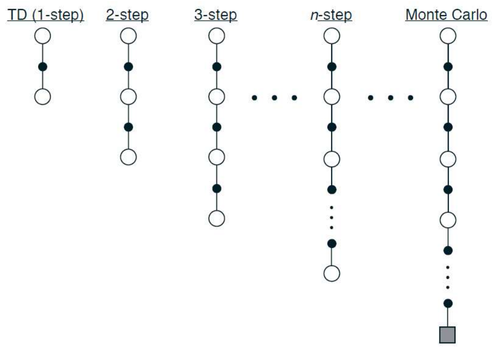
TD(0) uses $R_{t+1} + \gamma \cdot V(S_{t+1})$ as the target

could use

$R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot V(S_{t+2})$ or

$R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \gamma^3 \cdot V(S_{t+3})$

$\boldsymbol{n}$-Step Returns



$\boldsymbol{n}$-Step Returns

$$G_t^{(n)} = R_{t+1} + \gamma \cdot R_{t+2} + \cdots + \gamma^n \cdot V(S_{t+n})$$

- n-step TD uses $G_t^{(n)}$ as the target for update
  - Previous update was $G_t^{(1)}$ (one-step return)
- $G_t^{(n)}$ approximates the return G_t
  - Actual return for first n steps, $V(S_{t+n})$ approx. the rest

$\boldsymbol{n}$-Step TD

- n-step TD uses $G_t^{(n)}$ as the target for update
  - All converge as we want
- Can't update $V(S_t)$ for n steps
  - Nothing happens in the meantime
  - Even though we get feedback in that time
- Also ignores the value functions along the way
  - Have a sequence of V values might tell you more than 1

Complex TD Backups

- Turns out other can use combinations as well

$$\frac{1}{3}G_t^{(1)} + \frac{1}{3}G_t^{(2)} + \frac{1}{3}G_t^{(3)}$$

Or

$$\frac{9}{12}G_t^{(1)} + \frac{2}{12}G_t^{(2)} + \frac{1}{12}G_t^{(3)}$$

- Converge as long as coefficients sum to 1

TD(λ) Updates

- The TD(λ) update is a particular kind

$$G_t^\lambda = \left[(1-\lambda) \cdot \sum_{n=1}^{T-t-1} \lambda^{n-1} \cdot G_t^{(n)}\right] + \lambda^{T-t-1} \cdot G_t$$

- λ is a parameter from 0 to 1
- T is the length of the episode

TD(λ) Updates

- The TD(λ) update is a particular kind

$$G_t^\lambda = \left[(1-\lambda) \cdot \sum_{n=1}^{T-t-1} \lambda^{n-1} \cdot G_t^{(n)}\right] + \lambda^{T-t-1} \cdot G_t$$

- If $\lambda = 1$ …

TD(λ) Updates

- The TD(λ) update is a particular kind

$$G_t^\lambda = \left[(1-\lambda) \cdot \sum_{n=1}^{T-t-1} \lambda^{n-1} \cdot G_t^{(n)}\right] + \lambda^{T-t-1} \cdot G_t$$

0 1

- If $\lambda = 1$ …

TD(λ) Updates

- The TD(λ) update is a particular kind

$$G_t^\lambda = G_t$$

- If $\lambda = 1$ …

TD(λ) Updates

- The TD(λ) update is a particular kind

$$G_t^\lambda = G_t$$

- If $\lambda = 1$ … becomes MC

TD(λ) Updates

- The TD(λ) update is a particular kind

$$G_t^\lambda = \left[(1-\lambda)\cdot\sum_{n=1}^{T-t-1}\lambda^{n-1}\cdot G_t^{(n)}\right] + \lambda^{T-t-1}\cdot G_t$$

- If $\lambda = 0$ …

TD(λ) Updates

- The TD(λ) update is a particular kind

$$G_t^\lambda = \left[(1-\lambda)\cdot\sum_{n=1}^{T-t-1}\lambda^{n-1}\cdot G_t^{(n)}\right] + \lambda^{T-t-1}\cdot G_t$$

$(1-\lambda) \to 1$, $\lambda^{T-t-1} \to 0$

- If $\lambda = 0$ …

TD(λ) Updates

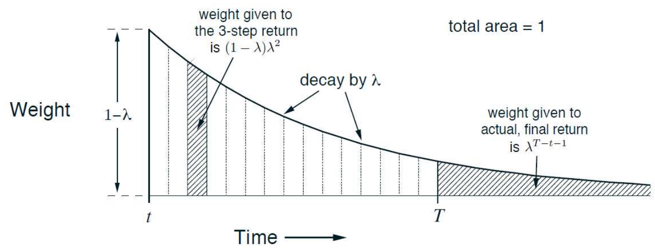
- The TD(λ) update is a particular kind

$$G_t^\lambda = \lambda^0 G_t^{(1)} + \lambda^1 G_t^{(2)} + \cdots$$

- If $\lambda = 0$ … $0^0 = 1$ 0

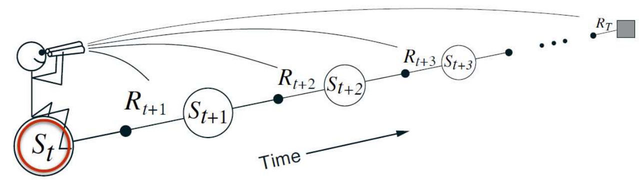
λ-Returns

- For non-zero or non-one values, decaying weighting
  - But always sums to 1



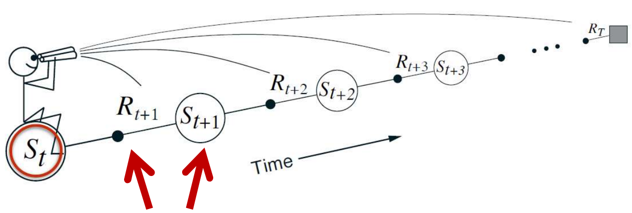
TD(λ) Forward View

- TD(λ) is a theoretical algorithm
  - Need all the returns to properly do the update
  - "Looking into the future"



TD(λ) Forward View

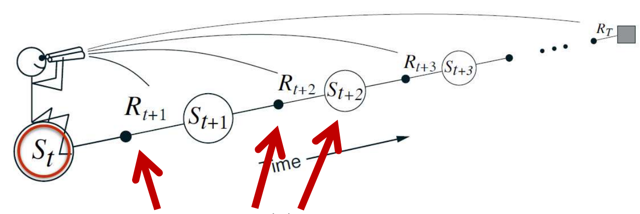
- TD(λ) is a theoretical algorithm
  - Need all the returns to properly do the update
  - "Looking into the future"



Needed for $G_t^{(1)}$

TD(λ) Forward View

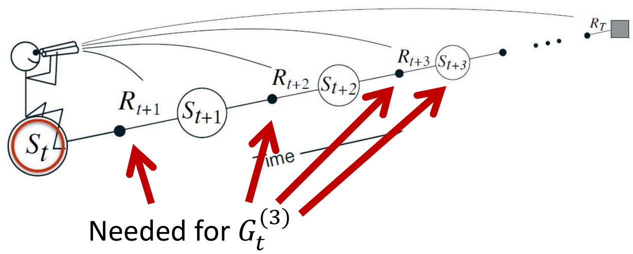
- TD(λ) is a theoretical algorithm
  - Need all the returns to properly do the update
  - "Looking into the future"



Needed for $G_t^{(2)}$

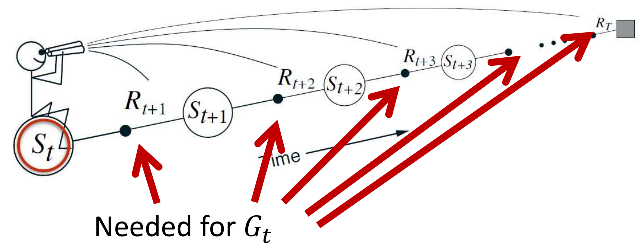
TD(λ) Forward View

- TD(λ) is a theoretical algorithm
  - Need all the returns to properly do the update
  - “Looking into the future”



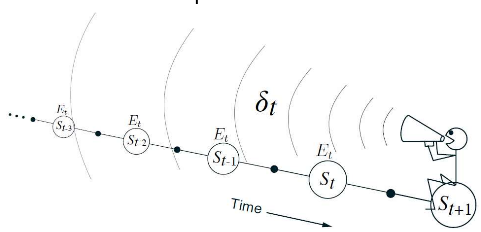
TD(λ) Forward View

- TD(λ) is a theoretical algorithm
  - Need all the returns to properly do the update
  - “Looking into the future”



TD(λ) Backward View

- Can’t look into the future to implement fwd view
- Instead, update as we go
  - Use latest info to update states visited earlier in episode



TD(λ) Updates

- But what should we send back?
- Could use latest reward and state transition to calculate $G_t^{(i)}$ for S_t, i steps later
  - Calculate $(1-\lambda)\cdot\sum_{1\le n\le i-1}\lambda^{n-1}\cdot G_t^{(n)} + \lambda\cdot G_t^{(i)}$
  - Use it as target
- Requires a lot of book-keeping, and is expensive
  - Not exactly forward view anyways
  - Makes decisions on incomplete intermediate values
  - Think about cases where an episode loops

Approximate TD($\boldsymbol{\lambda}$) Updates

- Will use a simpler approximation
- On time step t', every S_t where $t < t'$ is updated
- Will compute some $\Delta(t', S_t)$
- On time step t', update to $V(S_t)$ is as follows:

$$V(S_t) \leftarrow V(S_t) + \Delta(t', S_t)$$

Approximate TD($\boldsymbol{\lambda}$) Updates

- On time step t', update to $V(S_t)$ is as follows:

$$V(S_t) \leftarrow V(S_t) + \Delta(t', S_t)$$

- To calculate $\Delta(t', S_t)$, first consider **TD Error:**

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

- Going to define $\Delta(t', S_t) = \alpha \cdot \delta_{t'} \cdot E(S_t)$
  - Intuitively, $E(S_t)$ will be smaller for t farther in past
  - "Latest TD-error is less relevant farther back in time"

Eligibility Traces

$$\Delta(t', S_t) = \alpha \cdot \delta_{t'} \cdot E(S_t)$$

- **Eligibility traces** implement this intuition
  - "Latest TD-error is less relevant farther back in time"
- Eligibility traces keep track of how recent each state was visited
  - Determine how "eligible" a state is for newest learning update (using the TD-error)

Accumulating Traces

- **Accumulating traces** are a type of eligibility trace
- Let $E_t(s)$ be the value of $E(s)$ after t steps
  - Start with $E_t(s) = 0, \forall s$ at beginning of each episode
- Update is as follows:

$$E_t(s) = \begin{cases} \gamma \cdot \lambda \cdot E_{t-1}(s) + 1 & \text{, if } s = S_t \\ \gamma \cdot \lambda \cdot E_{t-1}(s) & \text{, otherwise} \end{cases}$$

Accumulating Traces

- **Accumulating traces** are a type of eligibility trace
- Let $E_t(s)$ be the value of $E(s)$ after t steps
  - Start with $E_t(s) = 0, \forall s$ at beginning of each episode
- Update is as follows:

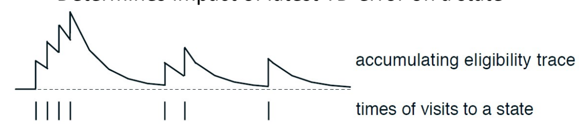
$$E_t(s) = \begin{cases} \gamma \cdot \lambda \cdot E_{t-1}(s) + 1 & \text{, if } s = S_t \\ \gamma \cdot \lambda \cdot E_{t-1}(s) & \text{, otherwise} \end{cases}$$

Accumulating Traces

- Accumulating trace update is as follows:

$$E_t(s) = \begin{cases} \gamma \cdot \lambda \cdot E_{t-1}(s) + 1 & \text{, if } s = S_t \\ \gamma \cdot \lambda \cdot E_{t-1}(s) & \text{, otherwise} \end{cases}$$

- “Eligibility” of a state decays when not visited
  - Determines impact of latest TD-error on a state



Accumulating Traces

- Accumulating trace update is as follows:

$$E_t(s) = \begin{cases} \gamma \cdot \lambda \cdot E_{t-1}(s) + 1 & \text{, if } s = S_t \\ \gamma \cdot \lambda \cdot E_{t-1}(s) & \text{, otherwise} \end{cases}$$

- If $\lambda = 0$, $E_t(s) = 1$ for $s = S_t$, else $E_t(s) = 0$
  - Only updates the last state visited
  - Still equivalent to TD(0) update
- If $\lambda = 1$, still equivalent to TD(1) update as well

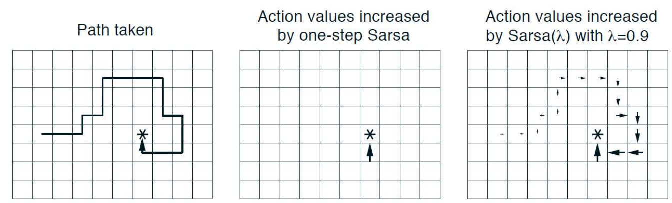
Eligibility Traces

- Accumulating traces are simple
  - But have some issues
- Other types of traces as well
  - Replacing traces
  - Dutch traces
- See textbook for more details
- But can now consider accumulating traces in Sarsa

Sarsa(λ) with Accumulating Traces

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

One-Step vs Multi-Step Sarsa



Off-Policy Control with Eligibility Traces

- When using off-policy methods, need to be more careful when using eligibility traces
- Consider $G_t^{(2)} = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot V(S_{t+2})$
  - This is a two-step estimate of expected return when using the current policy for two-steps
  - But is only an estimate for that specific policy
- In off-policy methods, those two actions might have been selected according to some other policy
  - Can't necessarily use them as estimate of target policy

Off-Policy Control with Eligibility Traces

- Consider $G_t^{(2)} = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot V(S_{t+2})$
- Can only use $G_t^{(2)}$, if actions chosen would have been selected by the target policy
  - Can't "backpropagate" current TD-error to previous states past points where target and behaviour policy don't coincide
- Implement this by resetting all eligibility traces to 0 whenever action selected is not what the target policy would have selected

Q(λ) with Accumulating Traces

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)
    $A^* \leftarrow \arg\max_a Q(S',a)$ (if A' ties for the max, then $A^* \leftarrow A'$)
    $\delta \leftarrow R + \gamma Q(S',A^*) - Q(S,A)$
    $E(S,A) \leftarrow E(S,A) + 1$
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$
      If $A' = A^*$, then $E(s,a) \leftarrow \gamma\lambda E(s,a)$
        else $E(s,a) \leftarrow 0$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

Off-Policy vs On-Policy TD(λ)

- Off-policy methods are more complicated
- Often must reset eligibility traces in off-policy
  - Decreases “how much is learned” per step

Efficient Eligibility Traces

- Earlier descriptions are naïve
- If have parallel machine, can quickly do eligibility trace updates
  - If not, will be expensive
- But eligibility of most states will be 0, many others will be close to 0
  - Can usually get effective behaviour by only updating a few steps in the past (instead of all steps)

Summary

- Eligibility traces allow for middle ground between TD(0) and Monte Carlo updates
- Realized using eligibility traces
  - Used accumulating traces as an example
- Introduced Sarsa(λ) and Q(λ)