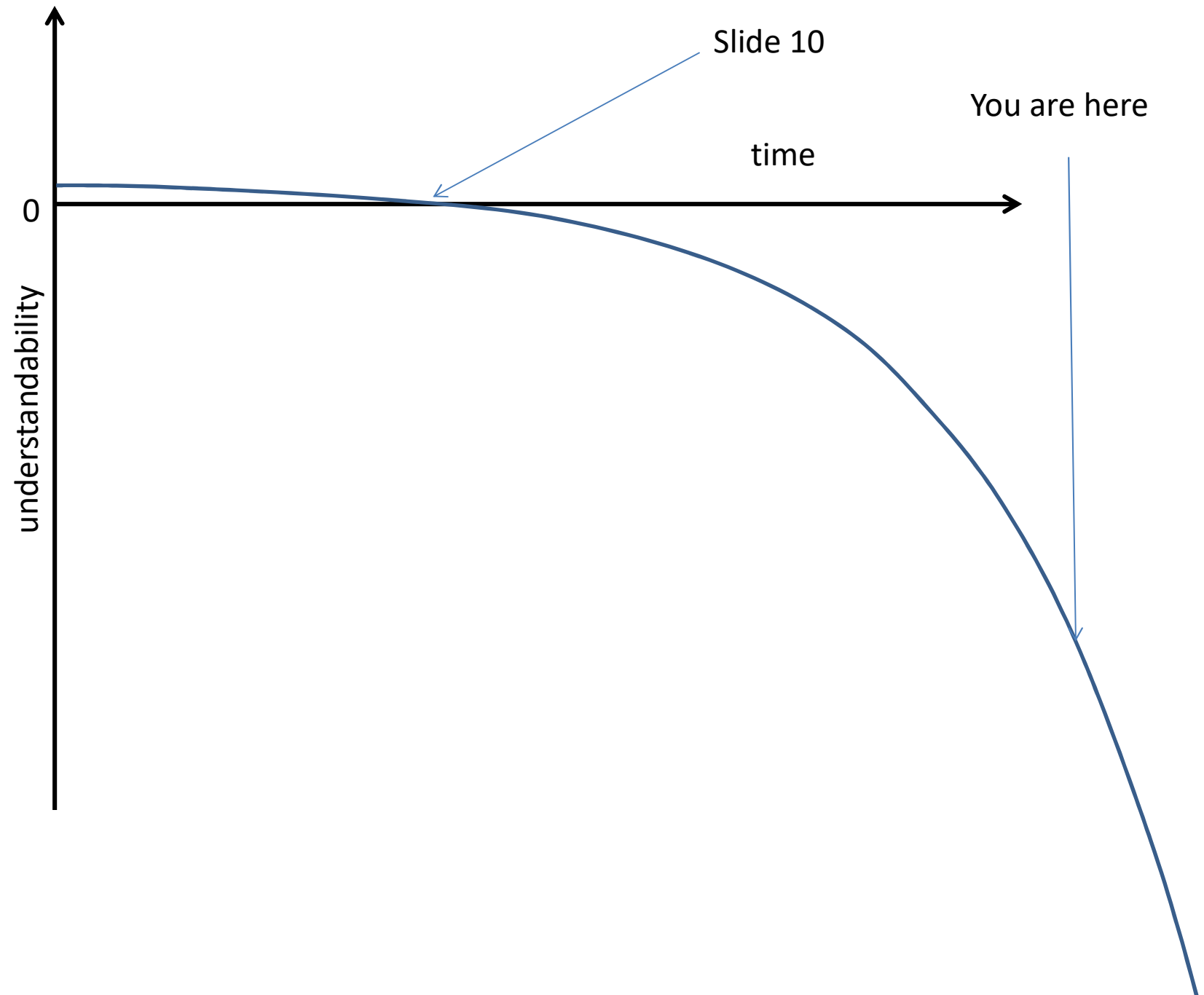


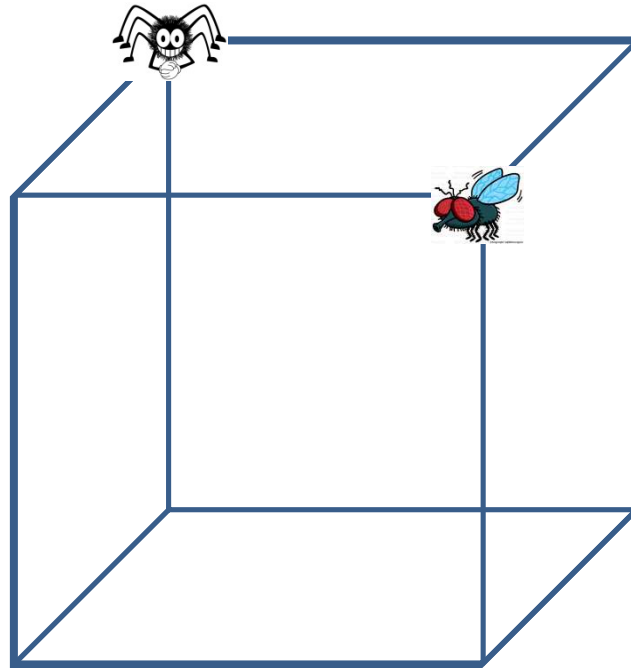
Reinforcement Learning

Spring 2019

Defining MDPs, Planning

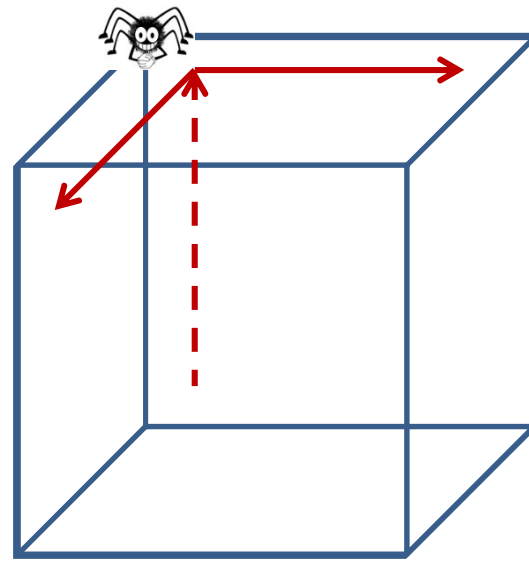


Markov Process



- Where you will go depends only on where you are

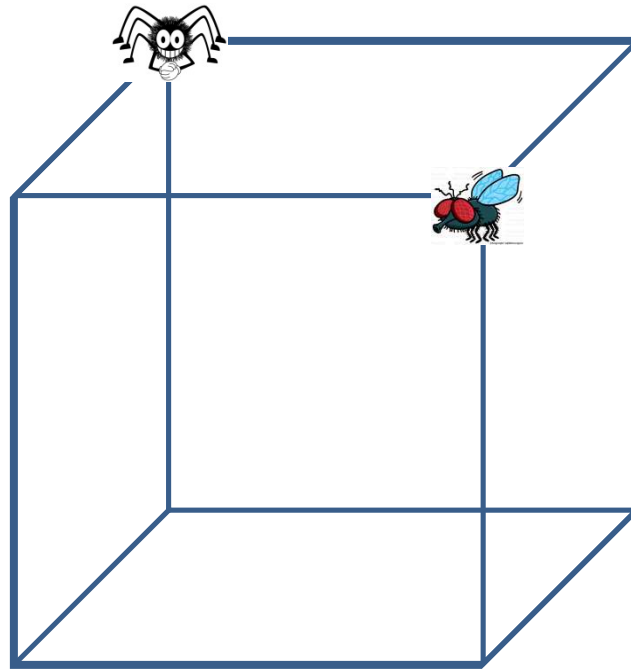
Markov Process: Information state



This spider doesn't like to turn back

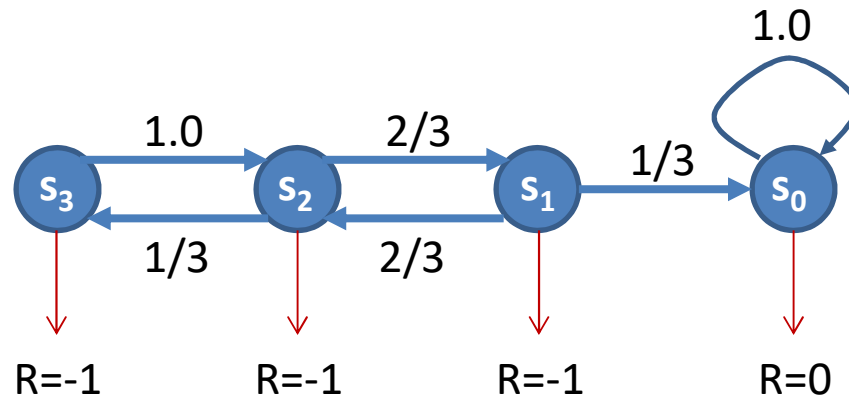
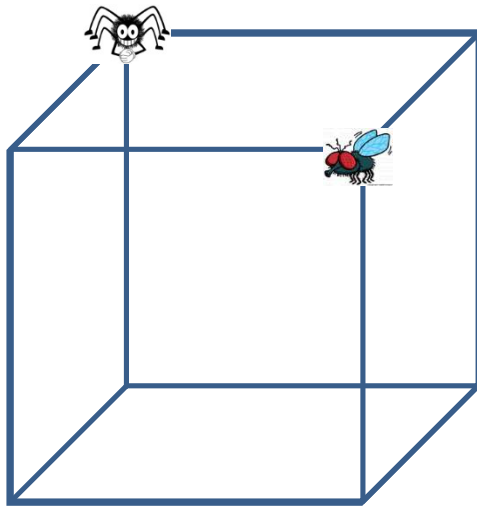
- The *information* state of a Markov process may be different from its physical state

Markov Reward Process



- Random wandering through states will occasionally win you a reward

The Fly Markov Reward Process



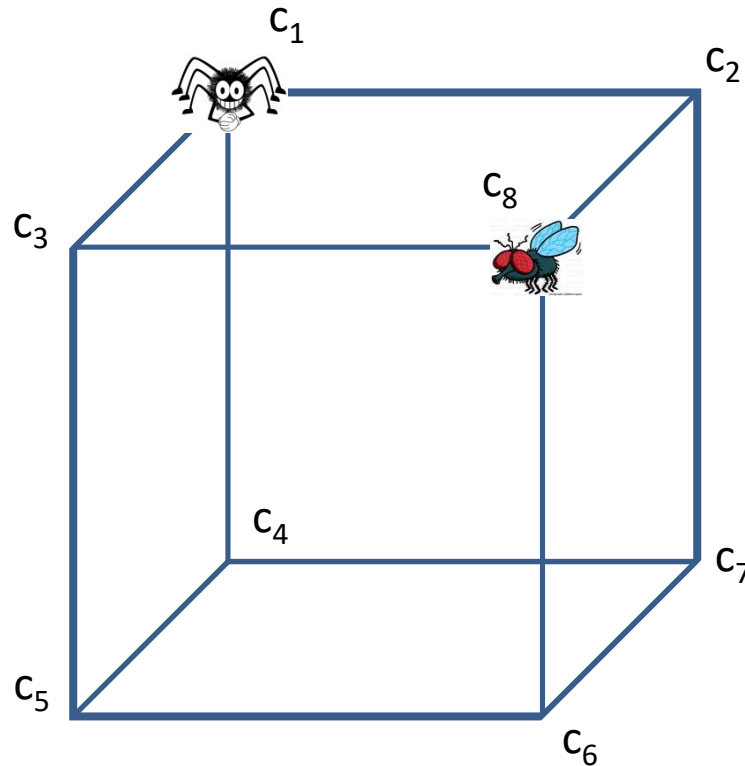
- There are, in fact, only four states, not eight
 - Manhattan distance between fly and spider = 0 (s_0)
 - Distance between fly and spider = 1 (s_1)
 - Distance between fly and spider = 2 (s_2)
 - Distance between fly and spider = 3 (s_3)
- Can, in fact, redefine the MRP entirely in terms of these 4 states

The discounted return

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

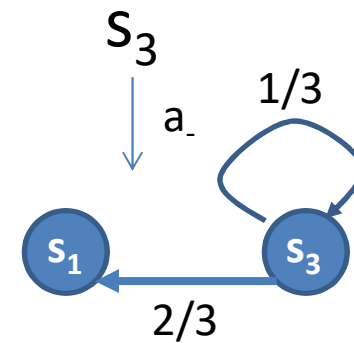
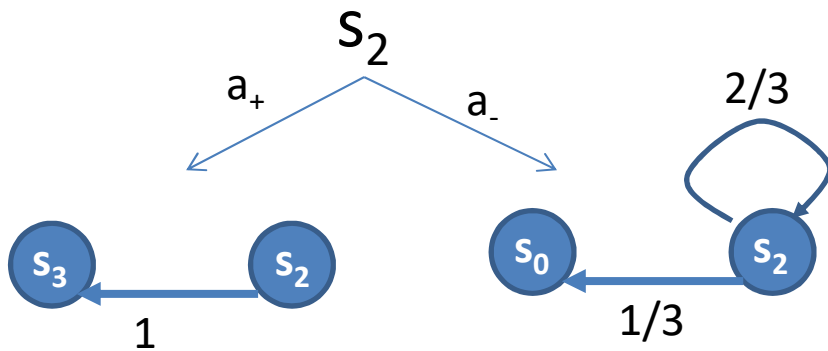
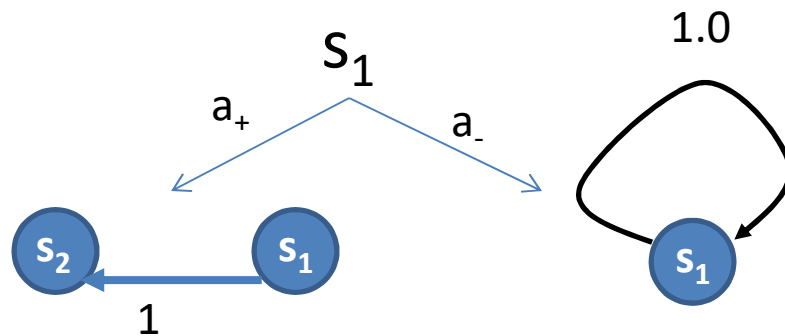
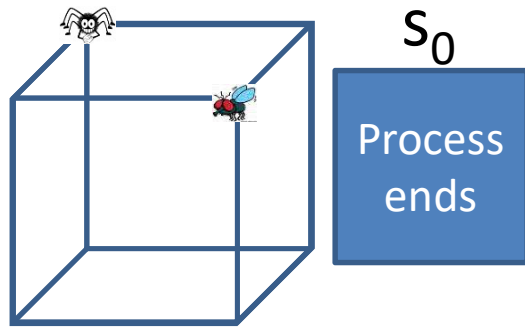
- Total *future* reward all the way to the end

Markov Decision Process

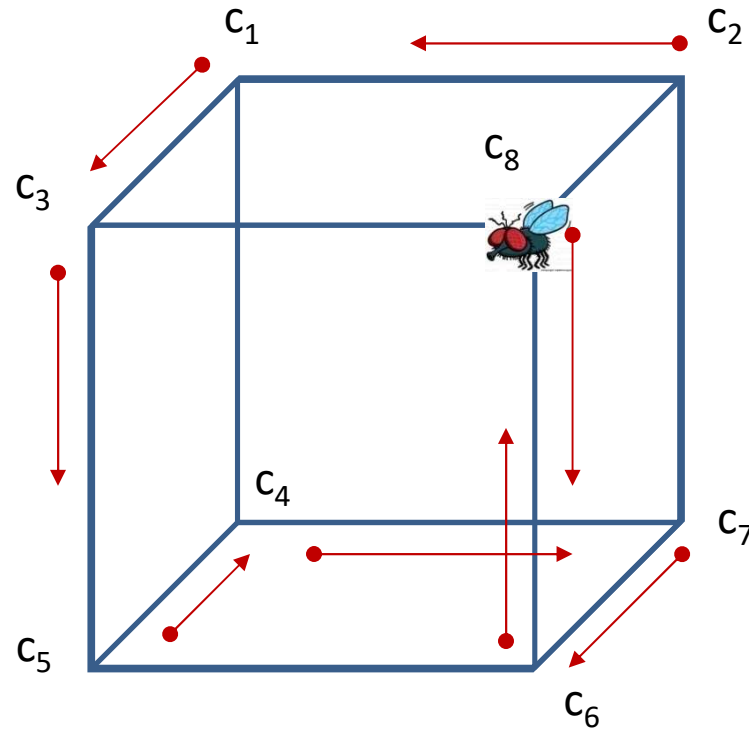


- Markov Reward Process with following change:
 - Agent has real agency
 - Agent's actions modify environment's behavior

The Fly Markov Decision Process



Policy



- The *policy* is the agent's choice of action in each state
 - May be stochastic

The Bellman Expectation Equations

- The Bellman expectation equation for state value function

$$v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s',s}^a v_{\pi}(s') \right)$$

- The Bellman expectation equation for action value function

$$q_{\pi}(s, a) = R_s^a + \gamma \sum_{s'} P_{s',s}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_{\pi}(s', a')$$

Optimal Policies

- The optimal policy is the policy that will maximize the expected total discounted reward at every state: $E[G_t | S_t = s]$

$$= E \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s \right]$$

- Optimal Policy Theorem:** For any MDP there exist optimal policies π_* that is better than or equal to every other policy:

$$\pi_* \geq \pi \quad \forall \pi$$

$$v_*(s) \geq v_\pi(s) \quad \forall s$$

$$q_*(s, a) \geq q_\pi(s, a) \quad \forall s, a$$

The optimal value function

$$\pi_*(a|s) = \begin{cases} 1 & \text{for } \operatorname{argmax}_{a'} q_*(s, a') \\ 0 & \text{otherwise} \end{cases}$$

$$v_*(s) = \max_a q_*(s, a)$$

Bellman *Optimality* Equations

- Optimal value function equation

$$v_*(s) = \max_a R_s^a + \gamma \sum_{s'} P_{s',s}^a v_*(s')$$

- Optimal action value equation

$$q_*(s, a) = R_s^a + \gamma \sum_{s'} P_{s',s}^a \max_{a'} q_*(s', a')$$

Planning with an MDP

- Problem:
 - **Given:** an MDP $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma \rangle$
 - **Find:** Optimal policy π_*
- Can either
 - **Value-based Solution:** Find optimal value (or action value) function, and derive policy from it OR
 - **Policy-based Solution:** Find optimal policy directly

Value-based Planning

- “Value”-based solution
- **Breakdown:**
 - **Prediction:** Given *any* policy π find value function $v_{\pi}(s)$
 - **Control:** Find the optimal policy

Prediction DP

- Iterate

$$v_{\pi}^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s',s}^a v_{\pi}^{(k)}(s') \right)$$

Policy Iteration

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0 \dots$ convergence):
 - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
 - Find the greedy policy

$$\pi^{(k+1)}(s) = \textit{greedy}\left(v_{\pi^{(k)}}(s)\right)$$

Value iteration

$$v_*^{(k)}(s) = \max_a R_s^a + \gamma \sum_{s'} P_{s',s}^a v_*^{(k-1)}(s')$$

- Each state simply inherits the cost of its best neighbour state
 - Cost of neighbour is the value of the neighbour plus cost of getting there

Problem so far

- *Given all details of the MDP*
 - Compute optimal value function
 - Compute optimal action value function
 - *Compute optimal policy*
- This is the problem of *planning*
- **Problem:** In real life, nobody gives you the MDP
 - How do we plan???



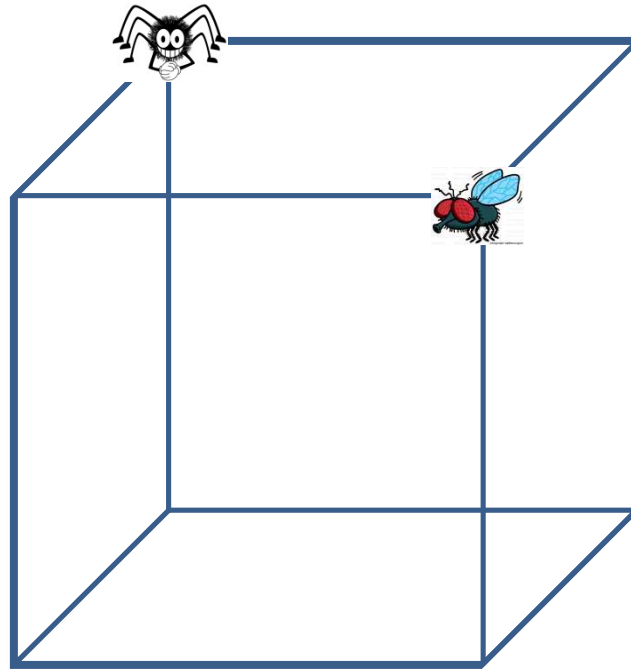
Model-Free Methods

- AKA model-free **reinforcement learning**
- How do you find the value of a policy, without knowing the underlying MDP?
 - Model-free *prediction*
- How do you find the optimal policy, without knowing the underlying MDP?
 - Model-free *control*

Model-Free Methods

- AKA model-free **reinforcement learning**
- How do you find the value of a policy, without knowing the underlying MDP?
 - Model-free *prediction*
- How do you find the optimal policy, without knowing the underlying MDP?
 - Model-free *control*
- **Assumption:** We can identify the states, know the *actions*, and measure rewards, but have no knowledge of the system dynamics
 - The key knowledge required to “solve” for the best policy
 - A reasonable assumption in many **discrete-state** scenarios
 - Can be generalized to other scenarios with infinite or unknowable state

Model-Free Assumption



- Can see the fly
- Know the distance to the fly
- Know possible actions (get closer/farther)
- But have no idea of how the fly will respond
 - Will it move, and if so, to what corner

Model-Free Methods

- AKA model-free **reinforcement learning**

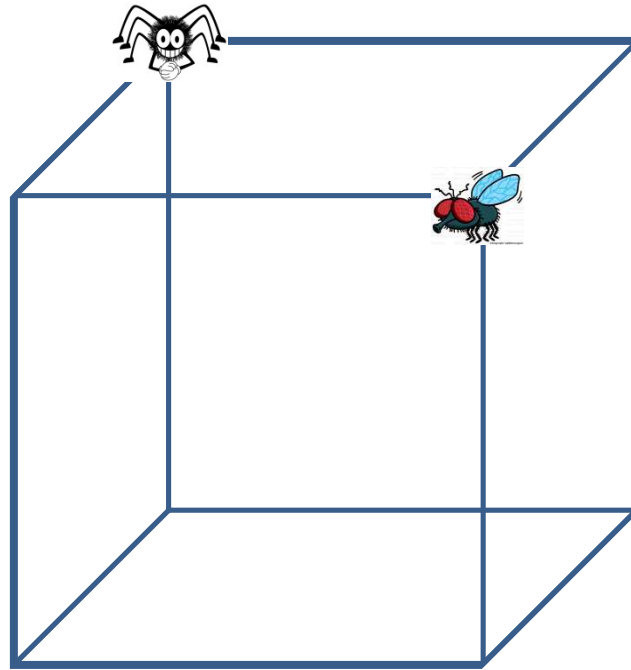
- How do you find the value of a policy, without knowing the underlying MDP?

- Model-free *prediction*

- How do you find the optimal policy, without knowing the underlying MDP?

- Model-free *control*

Model-Free Assumption



- Can see the fly and distance to the fly
- But have no idea of how the fly will respond to actions
 - Will it move, and if so, to what corner
- *But will always try to reduce distance to fly (have a known, fixed, policy)*
- ***What is the value of being a distance D from the fly?***

Methods

- *Monte-Carlo* Learning
- *Temporal-Difference* Learning
 - $TD(1)$
 - $TD(K)$
 - $TD(\lambda)$

Monte-Carlo learning to learn the value of a policy π

- Just “let the system run” while following the policy π and learn the value of different states
- Procedure: Record several *episodes* of the following
 - Take actions according to policy π
 - Note states visited and rewards obtained as a result
 - Record entire sequence:
 - $S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_T$
 - **Assumption:** Each “episode” ends at some time
- Estimate value functions based on observations by counting

Monte-Carlo Value Estimation

- Objective: Estimate value function $v_{\pi}(s)$ for every state s , given recordings of the kind:

$$S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_T$$

- Recall, the value function is the expected return:

$$\begin{aligned} v_{\pi}(s) &= E[G_t | S_t = s] \\ &= E[R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T | S_t = s] \end{aligned}$$

- To estimate this, we replace the *statistical* expectation $E[G_t | S_t = s]$ by the *empirical* average $avg[G_t | S_t = s]$

A bit of notation

- We actually record *many* episodes
 - $episode(1) = S_{11}, A_{11}, R_{12}, S_{12}, A_{12}, R_{13}, \dots, S_{1T_1}$
 - $episode(2) = S_{21}, A_{21}, R_{22}, S_{22}, A_{22}, R_{23}, \dots, S_{2T_2}$
 - ...
 - Different episodes may be different lengths

Counting Returns

- For each episode, we count the returns at all times:

– $S_{11}, A_{11}, R_{12}, S_{12}, A_{12}, R_{13}, S_{13}, A_{13}, R_{14}, \dots, S_{1T_1}$

$G_{1,1}$


- Return at time t

– $G_{1,1} = R_{12} + \gamma R_{13} + \dots + \gamma^{T_1-2} R_{1T_1}$

Counting Returns

- For each episode, we count the returns at all times:

– $S_{11}, A_{11}, R_{12}, S_{12}, A_{12}, R_{13}, S_{13}, A_{13}, R_{14}, \dots, S_{1T_1}$

$G_{1,2}$ 

- Return at time t

– $G_{1,1} = R_{12} + \gamma R_{13} + \dots + \gamma^{T_1-2} R_{1T_1}$

– $G_{1,2} = R_{13} + \gamma R_{14} + \dots + \gamma^{T_1-3} R_{1T_1}$

Counting Returns

- For each episode, we count the returns at all times:
 - $S_{11}, A_{11}, R_{12}, S_{12}, A_{12}, R_{13}, S_{13}, A_{13}, R_{14}, \dots, S_{1T_1}$
- Return at time t
 - $G_{1,1} = R_{12} + \gamma R_{13} + \dots + \gamma^{T_1-2} R_{1T_1}$
 - $G_{1,2} = R_{13} + \gamma R_{14} + \dots + \gamma^{T_1-3} R_{1T_1}$
 - \dots
 - $G_{1,t} = R_{1,t+1} + \gamma R_{1,t+2} + \dots + \gamma^{T_1-t-1} R_{1T_1}$

Estimating the Value of a State

- To estimate the value of any state, identify the instances of that state in the episodes:

– $\underbrace{S_{11}}_{s_a}, A_{11}, \underbrace{R_{12}}_{s_b}, S_{12}, A_{12}, \underbrace{R_{13}}_{s_a}, \underbrace{S_{13}}_{s_a}, A_{13}, R_{14}, \dots, S_{1T_1}$

- Compute the average return from those instances

$$v_{\pi}(s_a) = avg(G_{1,1}, G_{1,3}, \dots)$$

Estimating the Value of a State

- For every state s
 - Initialize: Count $N(s) = 0$, Total return $v_{\pi}(s) = 0$
 - For every episode e
 - For every time $t = 1 \dots T_e$
 - Compute G_t
 - If $(S_t == s)$
 - » $N(s) = N(s) + 1$
 - » $v_{\pi}(s) = v_{\pi}(s) + G_t$
 - $v_{\pi}(s) = v_{\pi}(s) / N(s)$- Can be done more efficiently..

Online Version

- For all s Initialize: Count $N(s) = 0$, Total return $totv_{\pi}(s) = 0$
- For every episode e
 - For every time $t = 1 \dots T_e$
 - Compute G_t
 - $N(S_t) = N(S_t) + 1$
 - $totv_{\pi}(S_t) = totv_{\pi}(S_t) + G_t$
 - For every state $s : v_{\pi}(s) = totv_{\pi}(s)/N(s)$
- Updating values at the end of each episode
- Can be done more efficiently..

Monte Carlo estimation

- Learning from experience explicitly
- After a sufficiently large number of episodes, in which all states have been visited a sufficiently large number of times, we will obtain good estimates of the value functions of all states
- Easily extended to evaluating *action value functions*

Estimating the Action Value function

- To estimate the value of any state-action pair, identify the instances of that state-action pair in the episodes:

$$- S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$

$s_a \ a_x$

$s_b \ a_y$

$s_a \ a_y \dots$

- Compute the average return from those instances

$$q_{\pi}(s_a, a_x) = avg(G_{1,1}, \dots)$$

Online Version

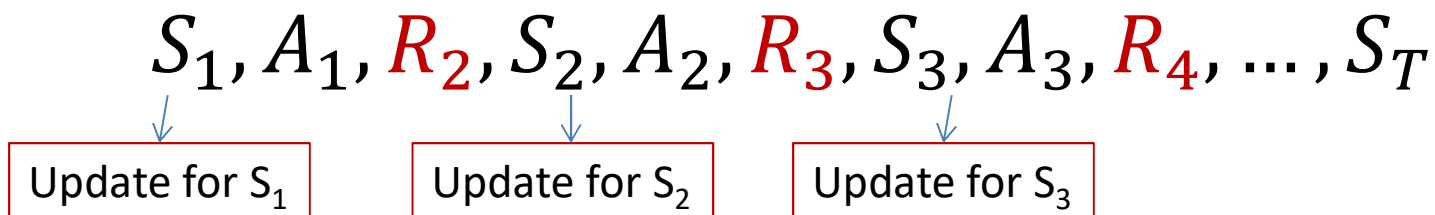
- For all s, a Initialize: Count $N(s, a) = 0$, Total value $totq_{\pi}(s, a) = 0$
- For every episode e
 - For every time $t = 1 \dots T_e$
 - Compute G_t
 - $N(S_t, A_t) = N(S_t, A_t) + 1$
 - $totq_{\pi}(S_t, A_t) = totq_{\pi}(S_t, A_t) + G_t$
 - For every $s, a : q(s, a) = totq_{\pi}(s, a) / N(s, a)$
- Updating values at the end of each episode

Monte Carlo: Good and Bad

- Good:
 - Will eventually get to the right answer
 - *Unbiased* estimate
- Bad:
 - Cannot update anything until the end of an episode
 - Which may last for ever
 - High variance! Each return adds many random values
 - Slow to converge

Online methods for estimating the value of a policy: **Temporal Difference Learning (TD)**

- Idea: Update your value estimates after every observation



- Do not actually wait until the end of the episode

Incremental Update of Averages

- Given a sequence x_1, x_2, x_3, \dots a running estimate of their average can be computed as

$$\mu_k = \frac{1}{k} \sum_{i=1}^k x_i$$

- This can be rewritten as:

$$\mu_k = \frac{(k-1)\mu_{k-1} + x_k}{k}$$

- And further refined to

$$\mu_k = \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})$$

Incremental Update of Averages

- Given a sequence x_1, x_2, x_3, \dots a running estimate of their average can be computed as

$$\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

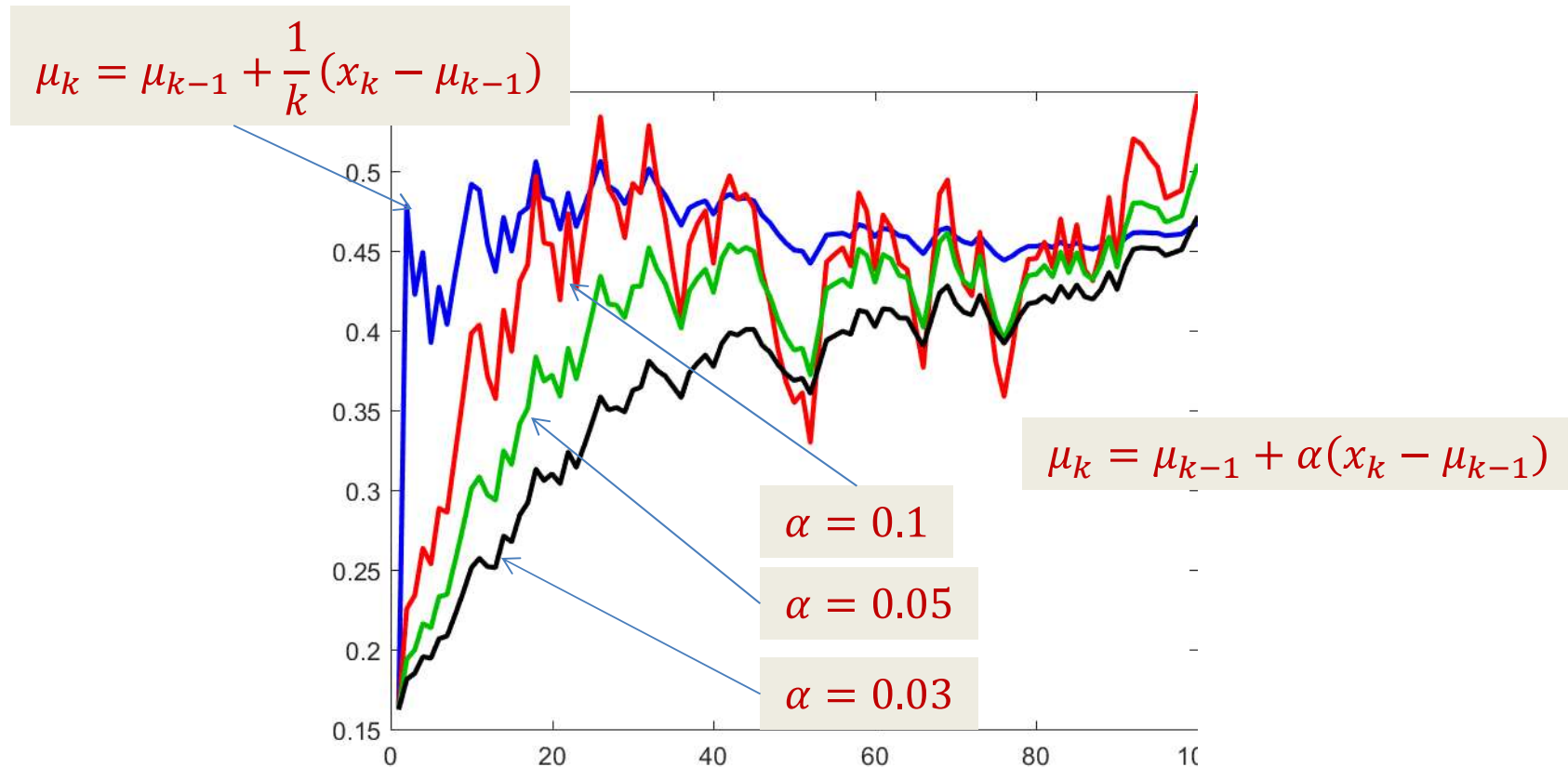
- Or more generally as

$$\mu_k = \mu_{k-1} + \alpha(x_k - \mu_{k-1})$$

- The latter is particularly useful for non-stationary environments
- For stationary environments α must shrink with iterations, but not too fast

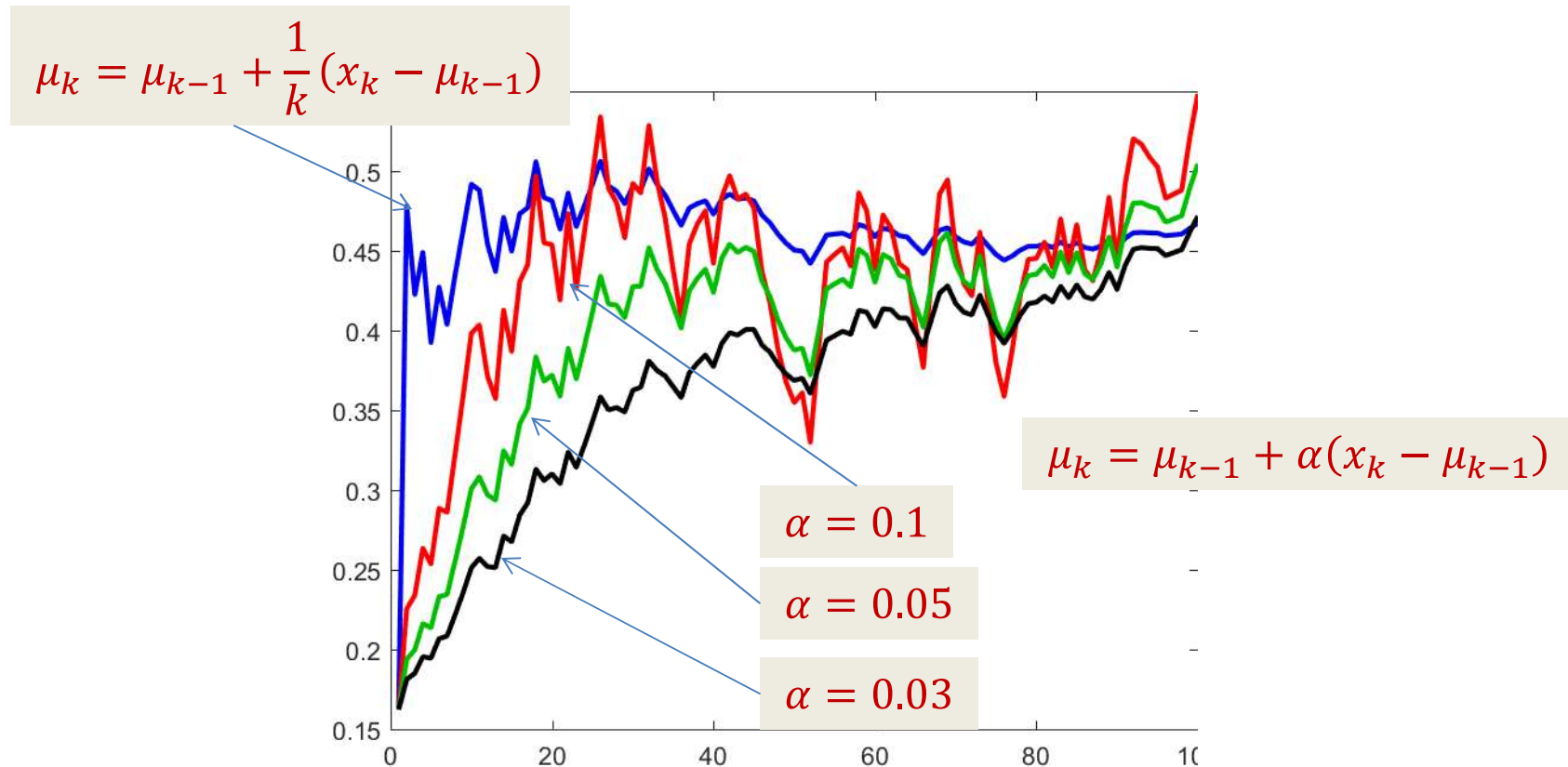
$$- \sum_k \alpha_k^2 < C, \quad \sum_k \alpha_k = \infty, \quad \alpha_k \geq 0$$

Incremental Updates



- Example of running average of a uniform random variable

Incremental Updates



- Correct equation is *unbiased* and converges to true value
- Equation with α is *biased* (early estimates can be expected to be wrong) but *converges* to true value

Updating Value Function Incrementally

- Actual update

$$v_{\pi}(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_{t(i)}$$

- $N(s)$ is the total number of visits to state s across all episodes
- $G_{t(i)}$ is the discounted return at the time instant of the i -th visit to state s

Online update

- Given any episode

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$

- Update the value of each state visited

$$N(S_t) = N(S_t) + 1$$

$$v_{\pi}(S_t) = v_{\pi}(S_t) + \frac{1}{N(S_t)} (G_t - v_{\pi}(S_t))$$

- Incremental version

$$v_{\pi}(S_t) = v_{\pi}(S_t) + \alpha (G_t - v_{\pi}(S_t))$$

- Still an unrealistic rule

- Requires the entire track until the end of the episode to compute G_t

Online update

- Given any episode

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$

- Update the value of each state visited

$$N(S_t) = N(S_t) + 1$$

$$v_{\pi}(S_t) = v_{\pi}(S_t) + \frac{1}{N(S_t)} (G_t - v_{\pi}(S_t))$$

Problem

- Incremental version

$$v_{\pi}(S_t) = v_{\pi}(S_t) + \alpha (G_t - v_{\pi}(S_t))$$

- Still an unrealistic rule

- Requires the entire track until the end of the episode to compute G_t

TD solution

$$v_{\pi}(S_t) = v_{\pi}(S_t) + \alpha(\underbrace{G_t}_{\text{Problem}} - v_{\pi}(S_t))$$

- But

$$G_t = R_{t+1} + \gamma G_{t+1}$$

- We can approximate G_{t+1} by the *expected* return at the next state S_{t+1}

Counting Returns

- For each episode, we count the returns at all times:
 - $S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$
- Return at time t
 - $G_1 = R_2 + \gamma R_3 + \dots + \gamma^{T-2} R_T$
 - $G_2 = R_3 + \gamma R_4 + \dots + \gamma^{T-3} R_T$
 - ...
 - $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-2} R_T$
- **Can rewrite as**
 - $G_1 = R_2 + \gamma G_2$
- Or
 - $G_1 = R_2 + \gamma R_3 + \gamma^2 G_3$
 - ...
 - $G_t = R_{t+1} + \sum_{i=1}^N \gamma^i R_{t+1+i} + \gamma^{N+1} G_{t+1+N}$

TD solution

$$v_{\pi}(S_t) = v_{\pi}(S_t) + \alpha(\underbrace{G_t - v_{\pi}(S_t)}_{\text{Problem}})$$

- But

$$G_t = R_{t+1} + \gamma G_{t+1}$$

- We can approximate G_{t+1} by the *expected* return at the next state $S_{t+1} \approx v_{\pi}(S_{t+1})$

$$G_t \approx R_{t+1} + \gamma v_{\pi}(S_{t+1})$$

- We don't know the real value of $v_{\pi}(S_{t+1})$ but we can “bootstrap” it by its current estimate

TD(1) true online update

$$v_{\pi}(S_t) = v_{\pi}(S_t) + \alpha(G_t - v_{\pi}(S_t))$$

- Where

$$G_t \approx R_{t+1} + \gamma v_{\pi}(S_{t+1})$$

- Giving us

$$- v_{\pi}(S_t) = v_{\pi}(S_t) + \alpha(R_{t+1} + \gamma v_{\pi}(S_{t+1}) - v_{\pi}(S_t))$$

TD(1) true online update

$$v_{\pi}(S_t) = v_{\pi}(S_t) + \alpha \delta_t$$

- Where

$$\delta_t = R_{t+1} + \gamma v_{\pi}(S_{t+1}) - v_{\pi}(S_t)$$

- δ_t is the TD *error*
 - The error between an (estimated) *observation* of G_t and the current estimate $v_{\pi}(S_t)$

TD(1) true online update

- For all s Initialize: $v_{\pi}(s) = 0$
- For every episode e
 - For every time $t = 1 \dots T_e$
 - $v_{\pi}(S_t) = v_{\pi}(S_t) + \alpha(R_{t+1} + \gamma v_{\pi}(S_{t+1}) - v_{\pi}(S_t))$
- There's a “lookahead” of one state, to know which state the process arrives at at the next time
- But is otherwise online, with continuous updates

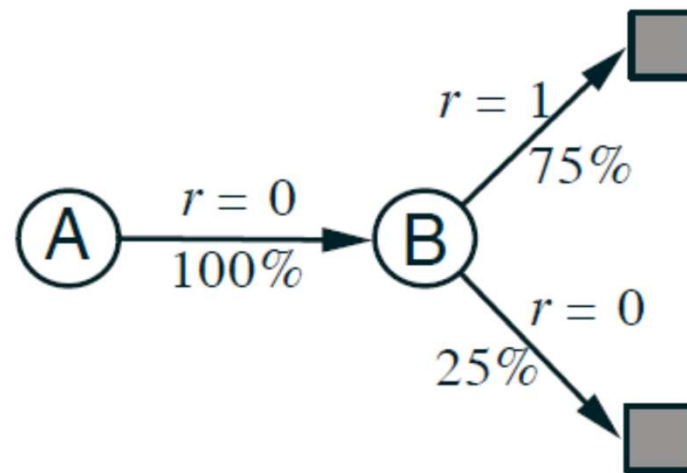
TD(1)

- Updates continuously – improve estimates as soon as you observe a state (and its successor)
- Can work even with *infinitely long* processes that never terminate
- Guaranteed to converge to the true values eventually
 - Although initial values will be biased as seen before
 - Is actually lower variance than MC!!
 - Only incorporates one RV at any time
- TD can give correct answers when MC goes wrong
 - Particularly when TD is allowed to *loop* over all learning episodes

TD vs MC

A, 0, B, 0
B, 1
B, 1
B, 1

B, 1
B, 1
B, 1
B, 0



- What are $v(A)$ and $v(B)$
 - Using MC
 - Using TD(1), where you are allowed to repeatedly go over the data

TD – look ahead further?

- TD(1) has a look ahead of 1 time step

$$G_t \approx R_{t+1} + \gamma v_{\pi}(S_{t+1})$$

- But we can look ahead further out

- $G_t(2) = R_{t+1} + \gamma R_{t+2} + \gamma^2 v_{\pi}(S_{t+2})$

- ...

- $G_t(N) = R_{t+1} \sum_{i=1}^N \gamma^i R_{t+1+i} + \gamma^{N+1} v_{\pi}(S_{t+N})$

TD(N) with lookahead

$$v_{\pi}(S_t) = v_{\pi}(S_t) + \alpha \delta_t(N)$$

- Where

$$\delta_t(N) = R_{t+1} + \sum_{i=1}^N \gamma^i R_{t+1+i} + \gamma^{N+1} v_{\pi}(S_{t+N}) - v_{\pi}(S_t)$$

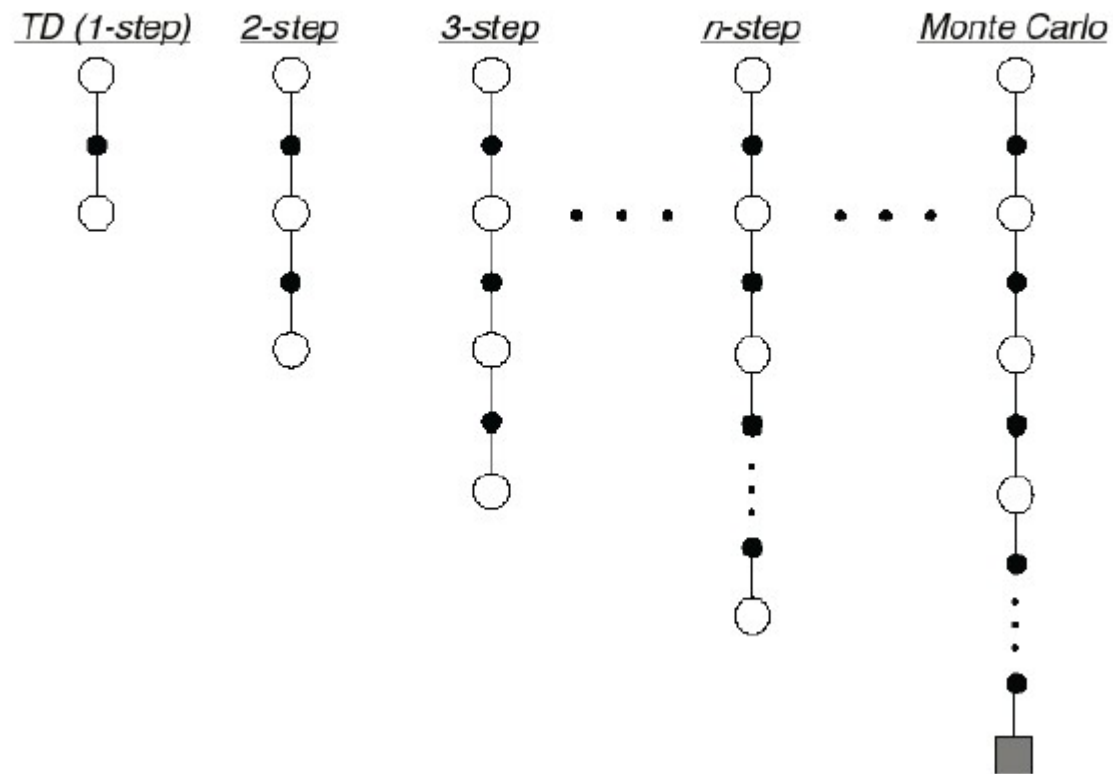
- $\delta_t(N)$ is the TD *error* with N step lookahead

Lookahead is good

- Good: The further you look ahead, the better your estimates get
- Problems:
 - But you also get more variance
 - At infinite lookahead, you're back at MC
- Also, you have to wait to update your estimates
 - A lag between observation and estimate
- So how much lookahead must you use

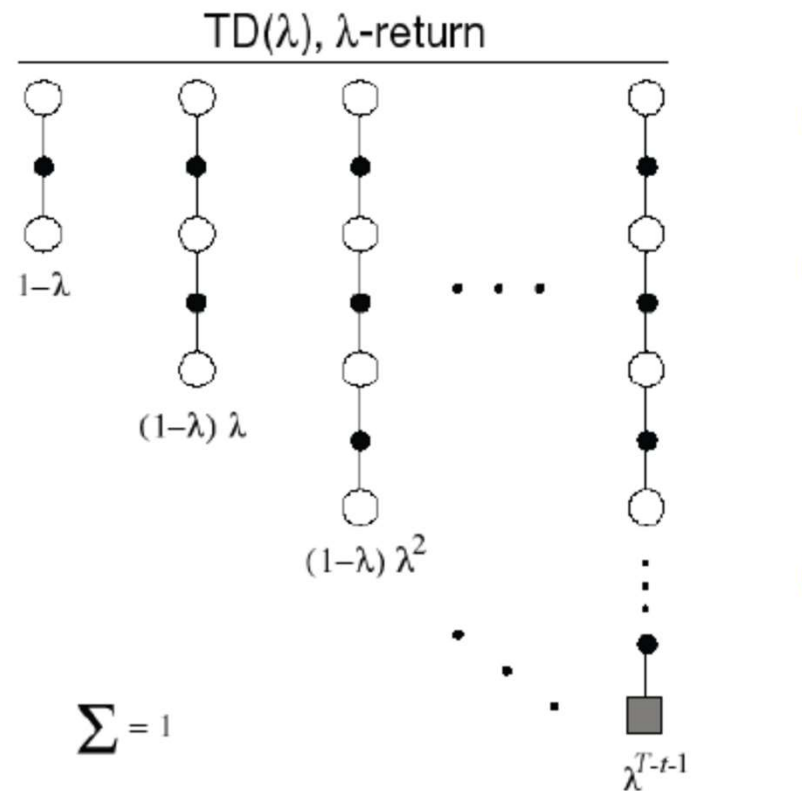
Looking Into The Future

- Let TD target look n steps into the future



- How much various TDs look into the future
- Which do we use?

Solution: Why choose?



- Each lookahead provides an estimate of G_t
- Why not just combine the lot with discounting?

TD(λ)

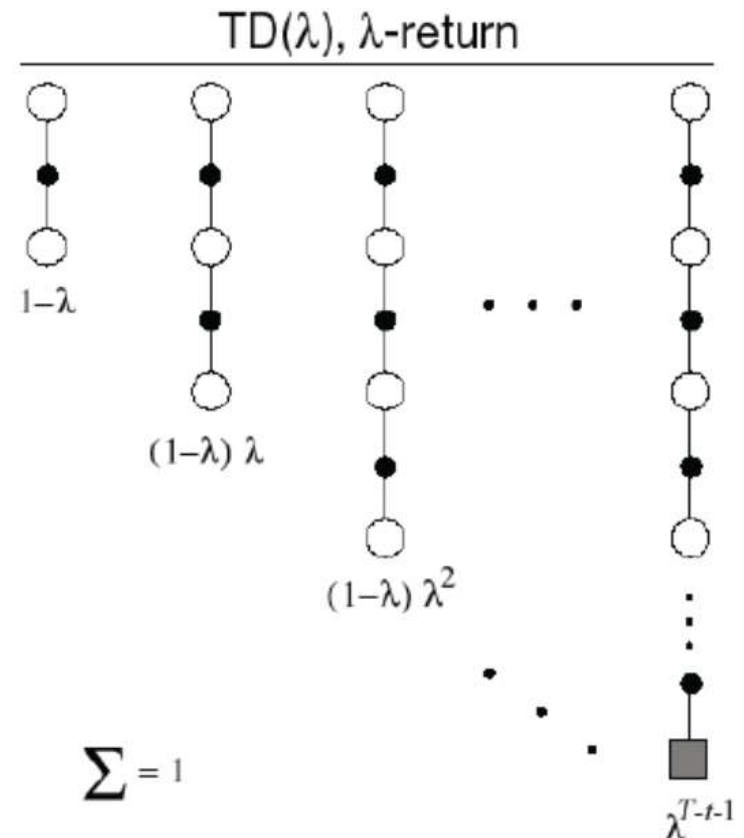
$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t(n)$$

- Combine the predictions from all lookaheads with an exponentially falling weight
 - Weights sum to 1.0

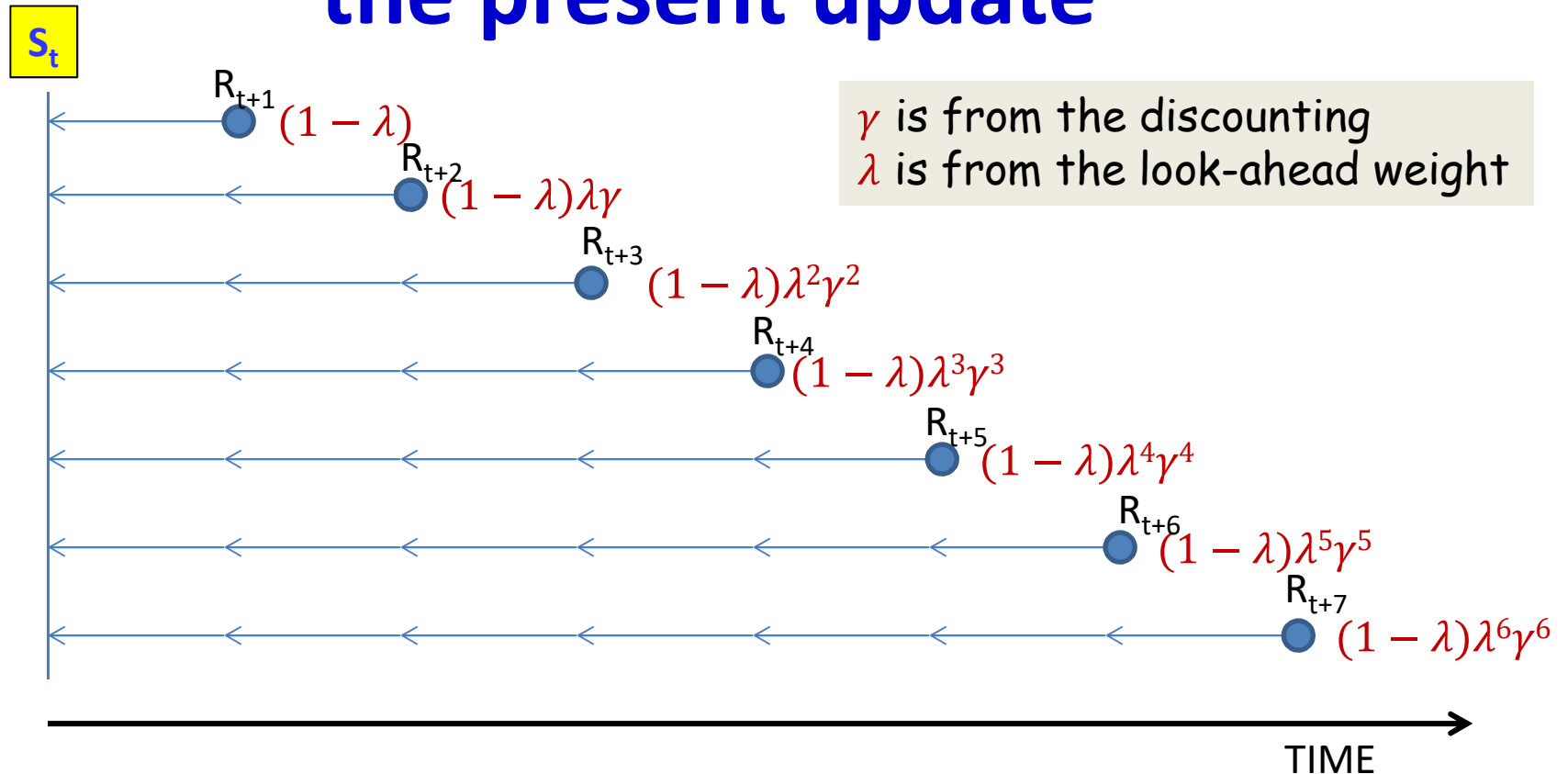
$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^\lambda - V(S_t) \right)$$

Something magical just happened

- $\text{TD}(\lambda)$ looks into the infinite future
 - I.e. we must have all the rewards of the future to compute our updates
 - How does that help?

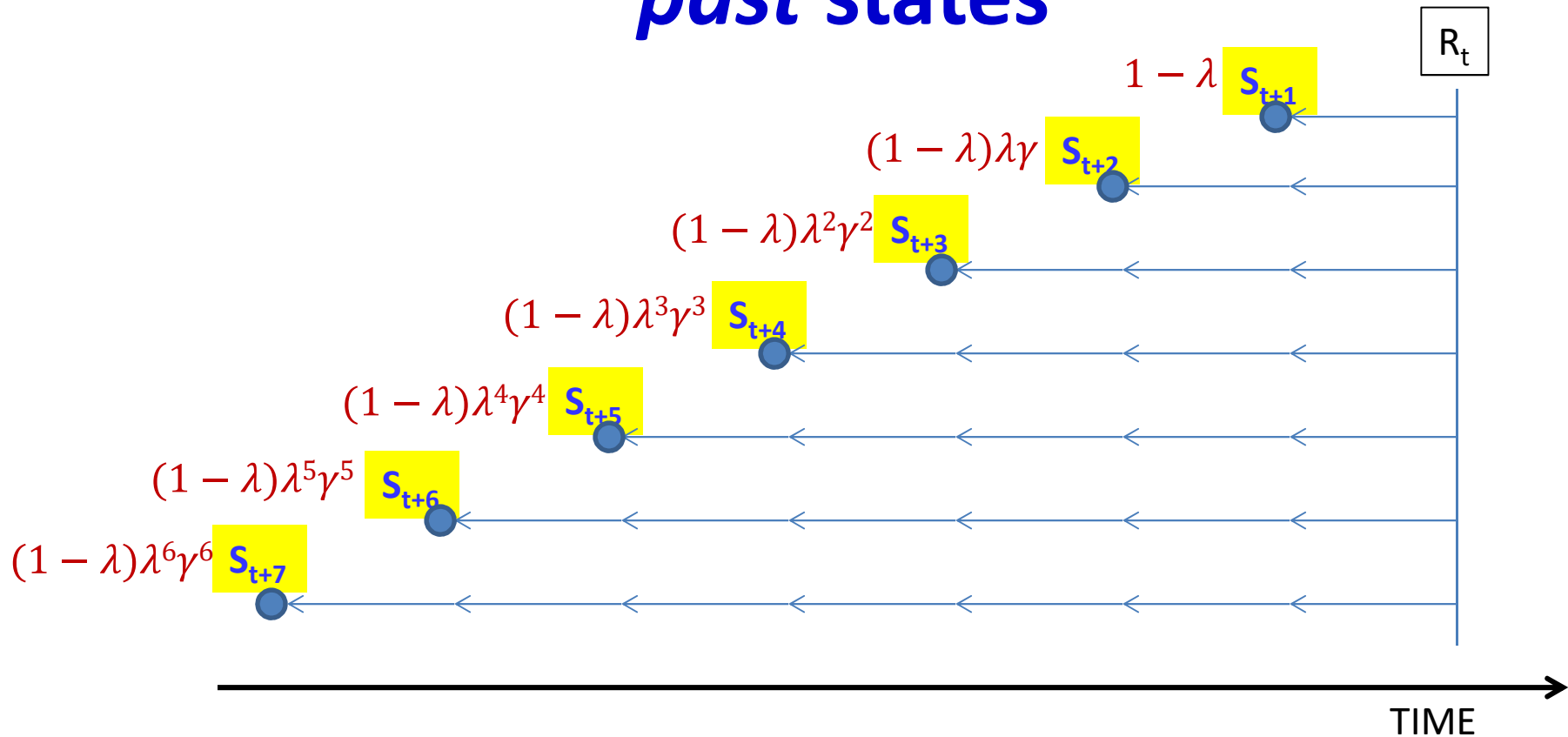


The contribution of future rewards to the present update



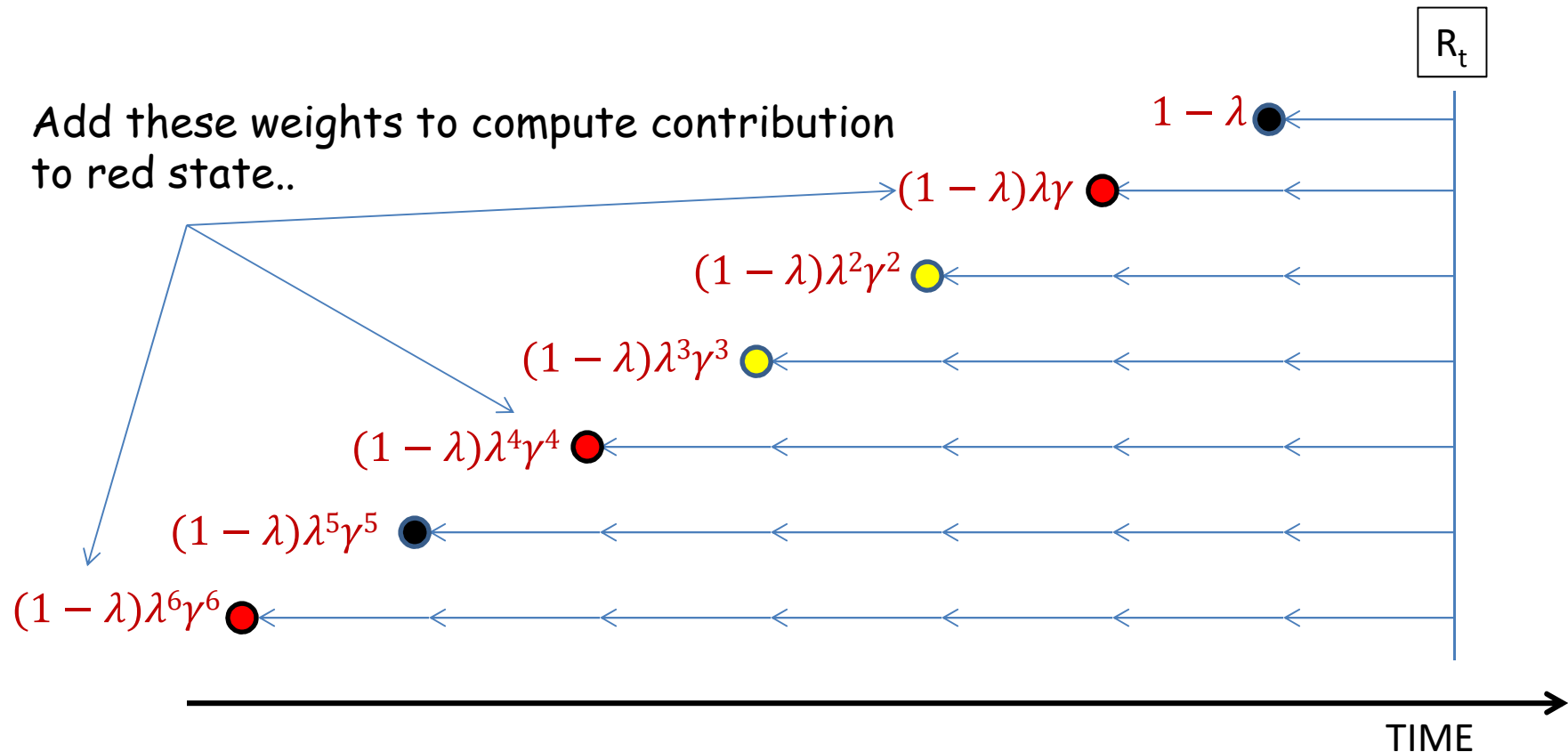
- All future rewards contribute to the update of the value of the current state

The contribution of current reward to *past* states



- All current reward contributes to the update of the value of all past states!

TD(λ) backward view



- The *Eligibility* trace:
 - Keeps track of *total* weight for any state
 - Which may have occurred at multiple times in the past

TD(λ)

- Maintain an eligibility trace for *every* state

$$E_0(s) = 0$$

$$E_t(s) = \lambda\gamma E_{t-1}(s) + 1(S_t = s)$$

- Computes total weight for the state until the present time

TD(λ)

- At every time, update the value of *every state* according to its eligibility trace

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

- Any state that was visited will be updated
 - Those that were not will not be, though

The magic of TD(λ)

- Managed to get the effect of infinite lookahead, by performing infinite *lookbehind*
 - Or at least look behind to the beginning
- Every reward updates the value of *all states* leading to the reward!
 - E.g., in a chess game, if we win, we want to increase the value of all game states we visited, not just the final move
 - But early states/moves must gain much less than later moves
- When $\lambda = 1$ this is exactly equivalent to MC

Story so far

- Want to compute the *values* of all states, given a policy, but no knowledge of dynamics
- Have seen monte-carlo and temporal difference solutions
 - TD is quicker to update, and in many situations the better solution
 - TD(λ) actually emulates an infinite lookahead
 - But we must choose good values of α and λ

Optimal Policy: Control

- We learned how to estimate the state value functions for an MDP whose transition probabilities are unknown *for a given policy*
- *How do we find the optimal policy?*

Value vs. Action Value

- The solution we saw so far only computes the *value functions* of states
- Not sufficient – to compute the optimal policy from value functions alone, we will need extra information, namely transition probabilities
 - Which we do not have
- Instead, we can use the same method to compute *action value* functions
 - Optimal policy in any state : Choose the action that has the largest *optimal* action value

Value vs. Action value

- Given only value functions, the optimal policy must be estimated as:

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Needs knowledge of transition probabilities

- Given action value functions, we can find it as:

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$$

- This is *model free* (no need for knowledge of model parameters)

Problem of optimal control

- From a series of episodes of the kind:
 $S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$
- Find the optimal action value function $q_*(s, a)$
 - The optimal policy can be found from it
- Ideally do this *online*
 - So that we can continuously improve our policy from *ongoing experience*

Exploration vs. Exploitation

- Optimal policy search happens while gathering experience *while following a policy*
- For fastest learning, we will follow an estimate of the optimal policy
- Risk: We run the risk of positive feedback
 - Only learn to evaluate our current policy
 - Will never learn about alternate policies that may turn out to be better
- Solution: We will follow our current optimal policy $1 - \epsilon$ of the time
 - But choose a random action ϵ of the time
 - The “epsilon-greedy” policy

GLIE Monte Carlo

- **Greedy in the limit with infinite exploration**
- Start with some random initial policy π
- Start the process at the initial state, and follow an action according to initial policy π
- Produce the episode

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$

- Process the episode using the following online update rules:

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Compute the ϵ -greedy policy for each state

$$\pi(a|s) = \begin{cases} 1 - \epsilon & \text{for } a = \operatorname{argmax}_{a'} Q(s, a') \\ \frac{\epsilon}{N_a - 1} & \text{otherwise} \end{cases}$$

- Repeat

GLIE Monte Carlo

- **Greedy in the limit with infinite exploration**
- Start with some random initial policy π
- Start the process at the initial state, and follow an action according to initial policy π
- Produce the episode

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$

- Process the episode using the following online update rules:

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

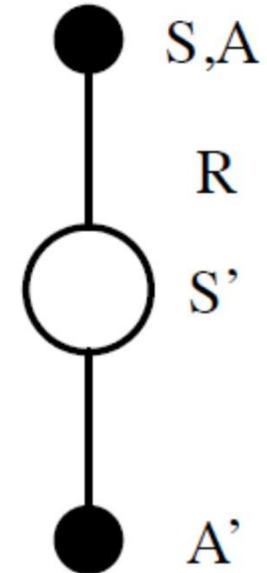
- Compute the ϵ -greedy policy for each state

$$\pi(a|s) = \begin{cases} 1 - \epsilon & \text{for } a = \operatorname{argmax}_{a'} Q(s, a') \\ \frac{\epsilon}{N_a - 1} & \text{otherwise} \end{cases}$$

- Repeat

On-line version of GLIE: SARSA

- Replace G_t with an estimate
- TD(1) or TD(λ)
 - Just as in the prediction problem
- **TD(1) \rightarrow SARSA**



$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

SARSA

- Initialize $Q(s, a)$ for all s, a
- Start at initial state S_1
- Select an initial action A_1
- For $t = 1..$ Terminate
 - Get reward R_t
 - Let system transition to new state S_{t+1}
 - Draw A_{t+1} according to ϵ -greedy policy

$$\pi(a|s) = \begin{cases} 1 - \epsilon & \text{for } a = \operatorname{argmax}_{a'} Q(s, a') \\ \frac{\epsilon}{N_a - 1} & \text{otherwise} \end{cases}$$

- Update

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

SARSA(λ)

- Again, the TD(1) estimate can be replaced by a TD(λ) estimate
- Maintain an eligibility trace for every state-action pair:

$$E_0(s, a) = 0$$
$$E_t(s, a) = \lambda\gamma E_{t-1}(s, a) + 1(S_t = s, A_t = a)$$

- Update every state-action pair visited so far

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E_t(s, a)$$

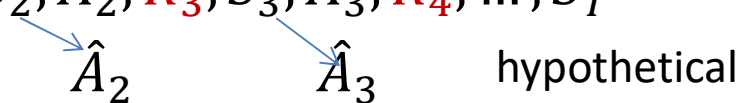
SARSA(λ)

- For all s, a initialize $Q(s, a)$
- For each episode e
 - For all s, a initialize $E(s, a) = 0$
 - Initialize S_1, A_1
 - For $t = 1 \dots Termination$
 - Observe R_{t+1}, S_{t+1}
 - Choose action A_{t+1} using policy obtained from Q
 - $\delta = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$
 - $E(S_t, A_t) += 1$
 - For all s, a
 - $Q(s, a) = Q(s, a) + \alpha \delta E(s, a)$
 - $E(s, a) = \gamma \lambda E(s, a)$

On-policy vs. Off-policy

- SARSA assumes you're following the same policy that you're learning
- Its possible to follow one policy, while learning from others
 - E.g. learning by observation
- The policy for learning is the whatif policy

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$


 $\hat{A}_2 \quad \hat{A}_3$ hypothetical

- Modifies learning rule

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

- to

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, \hat{A}_{t+1}) - Q(S_t, A_t))$$

- Q will actually represent the action value function of the *hypothetical policy*

SARSA: Suboptimality

- SARSA: From any state-action (S, A) , accept reward (R) , transition to next state (S') , choose next action (A')

- Use TD rules to update:

$$\delta = R + \gamma Q(S', A') - Q(S, A)$$

- Problem: which policy do we use to choose A'

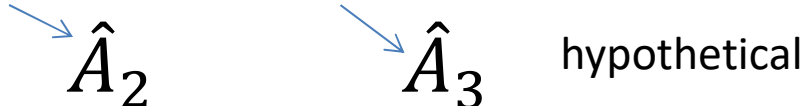
SARSA: Suboptimality

- SARSA: From any state-action (S, A) , accept reward (R) , transition to next state (S') , choose next action (A')
- Problem: which policy do we use to choose A'
- If we choose the *current judgment of the best action* at S' we will become too greedy
 - Never explore
- If we choose a *sub-optimal* policy to follow, we will never find the best policy

Solution: Off-policy learning

- The policy for learning is the whatif policy

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$



- Use the *best* action for S_{t+1} as your hypothetical off-policy action
- But actually follow an *epsilon-greedy* action
 - The hypothetical action is guaranteed to be better than the one you actually took
 - But you still explore (non-greedy)

Q-Learning

- From any state-action pair S, A
 - Accept reward R
 - Transition to S'
 - Find the *best action* A' for S'
 - Use it to update $Q(S, A)$
 - But then *actually perform an epsilon-greedy action* A'' from S'

Q-Learning (TD(1) version)

- For all s, a initialize $Q(s, a)$
- For each episode e
 - Initialize S_1, A_1
 - For $t = 1 \dots Termination$
 - Observe R_{t+1}, S_{t+1}
 - Choose action A_{t+1} at S_{t+1} using epsilon-greedy policy obtained from Q
 - Choose action \hat{A}_{t+1} at S_{t+1} as $\hat{A}_{t+1} = \underset{a}{\operatorname{argmax}} Q(S_{t+1}, a)$
 - $\delta = R_{t+1} + \gamma Q(S_{t+1}, \hat{A}_{t+1}) - Q(S_t, A_t)$
 - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha \delta$

Q-Learning (TD(λ) version)

- For all s, a initialize $Q(s, a)$
- For each episode e
 - For all s, a initialize $E(s, a) = 0$
 - Initialize S_1, A_1
 - For $t = 1 \dots Termination$
 - Observe R_{t+1}, S_{t+1}
 - Choose action A_{t+1} at S_{t+1} using epsilon-greedy policy obtained from Q
 - Choose action \hat{A}_{t+1} at S_{t+1} as $\hat{A}_{t+1} = \underset{a}{\operatorname{argmax}} Q(S_{t+1}, a)$
 - $\delta = R_{t+1} + \gamma Q(S_{t+1}, \hat{A}_{t+1}) - Q(S_t, A_t)$
 - $E(S_t, A_t) += 1$
 - For all s, a
 - $Q(s, a) = Q(s, a) + \alpha \delta E(s, a)$
 - $E(s, a) = \gamma \lambda E(s, a)$

What about the actual policy?

- Optimal greedy policy:

$$\pi(a|s) = \begin{cases} 1 & \text{for } a = \operatorname{argmax}_{a'} Q(s, a') \\ 0 & \text{otherwise} \end{cases}$$

- Exploration policy

$$\pi(a|s) = \begin{cases} 1 - \epsilon & \text{for } a = \operatorname{argmax}_{a'} Q(s, a') \\ \frac{\epsilon}{N_a - 1} & \text{otherwise} \end{cases}$$

- Ideally ϵ should decrease with time

Q-Learning

- Currently most-popular RL algorithm
- Topics not covered:
 - Value function approximation
 - Continuous state spaces
 - Deep-Q learning
 - Action replay
 - Application to real problem..