Reinforcement Learning

Robotica Probabilistica

Reinforcement Learning

Chapter 1, Barto

- **RL Task**
  - Learn how to behave successfully to achieve a goal while interacting with an external environment.

    Learn through experience from <u>trial and error</u>

- **Examples**
  - Game playing: The agent knows it has won or lost, but it doesn't know the appropriate action in each state.
  - Control: a traffic system can measure the delay of cars, but not know how to decrease it.

Reinforcement Learning

- No knowledge of environment
  - The agent can act in the world and observe states and reward

- Many factors make RL difficult:

  - No supervisor
  - Actions have non-deterministic effects
    - Which are initially unknown
  - Rewards / punishments are infrequent
    - Often at the end of long sequences of actions
      - Credit assignment: what actions are responsible for rewards or punishments
  - World is large and complex

- Learner must decide what actions to take
  - We will assume the world behaves as an MDP

Reinforcement Learning

- Learning and acting at the same time
- Scalability is a big issue
  - Zhang, W., Dietterich, T. G., (1995). **A Reinforcement Learning Approach to Job-shop Scheduling**
  - *G. Tesauro* (1994). "**TD-Gammon, A Self-Teaching Backgammon Program Achieves Master-level Play**" in Neural Computation
  - Reinforcement Learning for Vulnerability Assessment in Peer-to-Peer Networks, IAAI 2008
    - Policy Gradient Update
  - **An Application of Reinforcement Learning to Aerobatic Helicopter Flight**, Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. NIPS, 2007
  - **DeepQ Learing for Atari Games** 2015
  - **DeepQ Learning AlphaGo** (2015/2016)

Reinforcement Learning

- Learning and acting at the same time
- Scalability is a big issue
  - **An Application of Reinforcement Learning to Aerobatic Helicopter Flight**, Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. NIPS, 2007

Reinforcement Learning

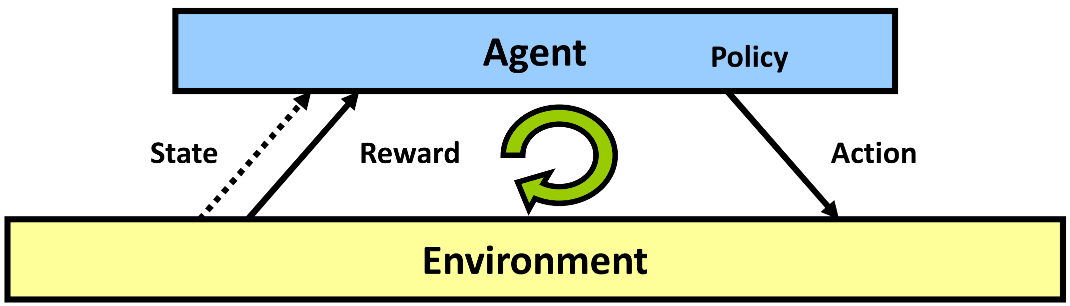
- Learning and acting at the same time
- Scalability is a big issue
  - **DeepQ Learing for Atari Games** 2015

Reinforcement Learning

- MDP model:
  - States, Actions, Reward, Transitions
- Goal:
  - Learn the policy as in MDP
- Knowledge:
  - State, Actions
- No knowledge:
  - Transitions, Reward
- Discover by acting:
  - Effects of Actions
  - Rewards

Sequential Decision Problem

Chapter 3, Barto



$$s_0 \xrightarrow{a_0 : r_0} s_1 \xrightarrow{a_1 : r_1} s_2 \xrightarrow{a_2 : r_2} \cdots$$

- **Transition model,** how action influence states
- **Reward R**, immediate value of state-action transition
- **History:** sequence of actions, rewards, observations/state
- **Policy** π**,** maps states to actions

Reinforcement Learning

Learner — passive / active

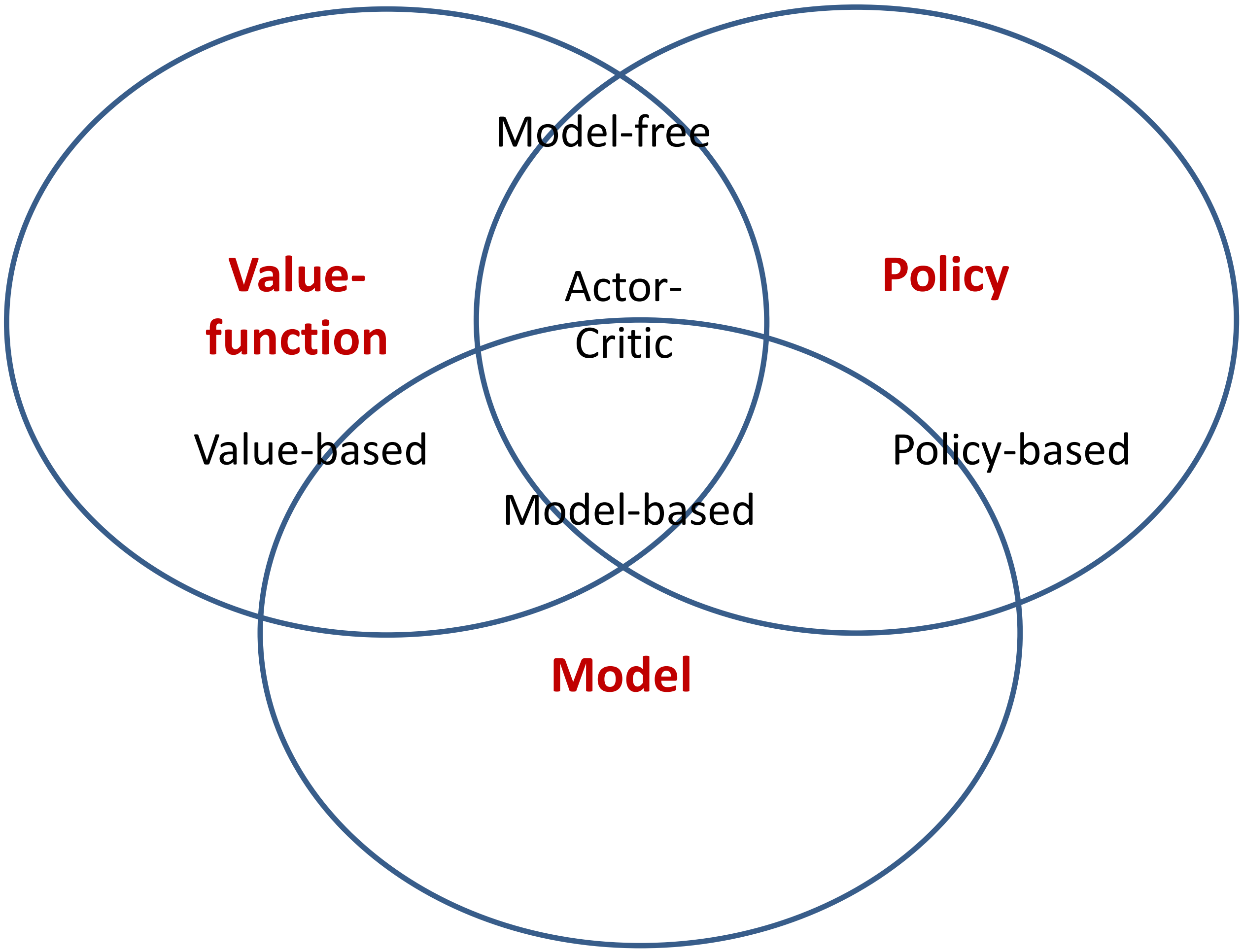
Sequential decision problems

- Approaches:
  - Learn values of states (or state histories) and try to maximize utility of their outcomes (Model-based).
    - Need a model of the environment: what ops and what states they lead to
  - Learn values of state-action pairs (Model free)
    - Does not require a model of the environment (except legal moves)
    - Cannot look ahead

Key Aspects in RL

- How do we update value function or policy:
  - How do we acquire training data
  - Sequence of (s,a,r)….
- How do we explore and act:
  - Exploit or Exploration dilemma

Taxonomy: Reinforcement Learning



Category of Reinforcement Learning

- Model-based RL
  - Constructs domain transition model, MDP
    - E^3 – Kearns and Singh
- Model-free RL
  - Only concerns policy
    - Q-Learning - Watkins
- Active Learning (Off-Policy Learning)
  - Q-Learning
- Passive Learning (On-Policy learning)
  - Sarsa - Sutton

RL Task

- Execute actions in environment, observe results.
- Learn action policy $\pi : state \rightarrow action$ that maximizes expected discounted reward

$$\boldsymbol{E}\,[\boldsymbol{r}(\boldsymbol{t}) + \gamma \boldsymbol{r}(\boldsymbol{t} + 1) + \gamma^2 \boldsymbol{r}(\boldsymbol{t} + 2) + \ldots]$$

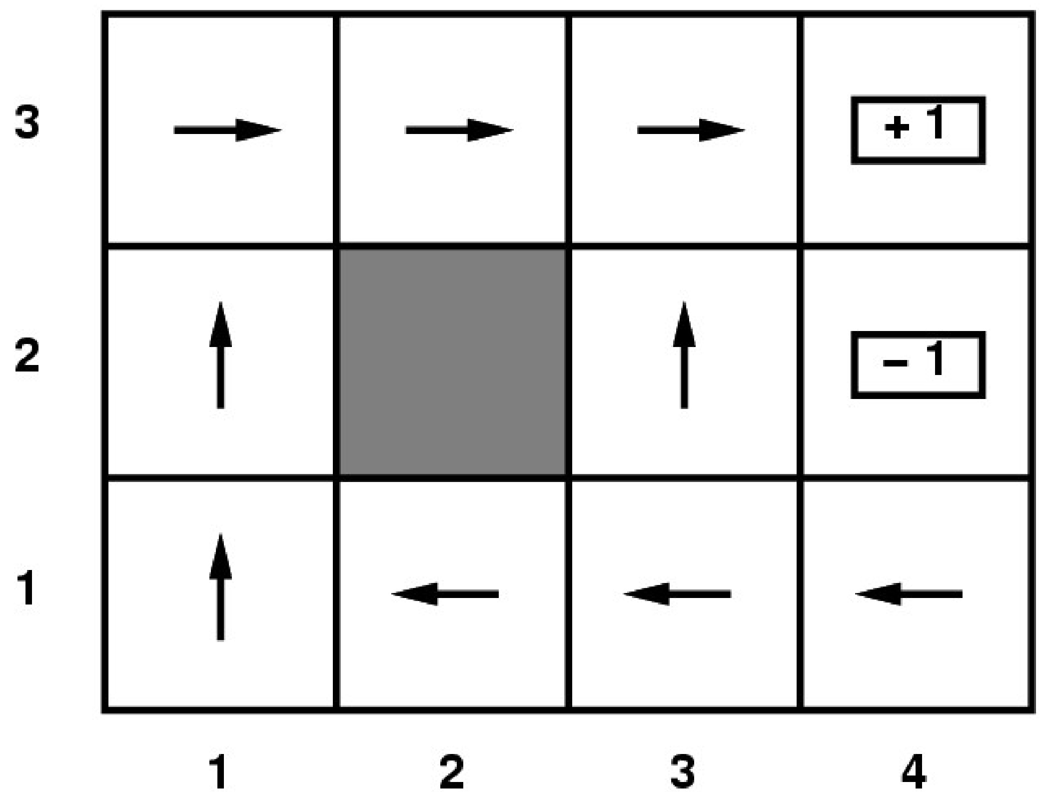
from any starting state in S

Reinforcement Learning

- Target function is π : *state* $\rightarrow$ *action*
- However…
  - We have no training examples of form <*state*, *action*>
  - Training examples are of form <<*state*, *action*>, *reward*>

Example: Passive RL

- Assume a given policy
- We want to determine how good it is

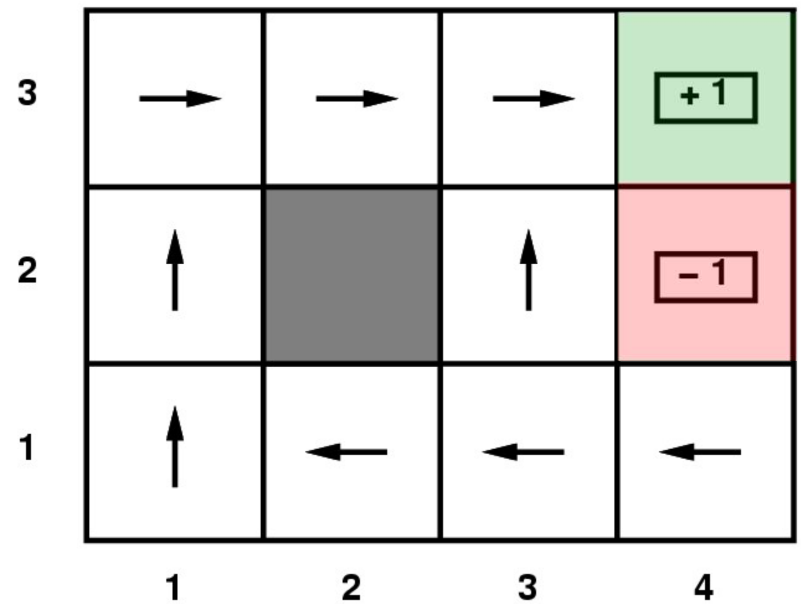


Objective: Value Function

3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Passive RL

- Given policy π,
  - estimate $V^{\pi}(s)$
- Not given
  - transition matrix, nor
  - reward function!
- Simply follow the policy for many epochs
- Epochs: training sequences



Chapter 5, Sutton Barto, Section 5.1-5.3

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1

Each epoch should end

Direct Estimation

- Direct estimation (model free)
  - Estimate $V^{\pi}(s)$ as average total reward of epochs containing s (calculating from s to end of epoch)

- ***Reward to go*** of a state s

  the sum of the (discounted) rewards from that state until a terminal state is reached

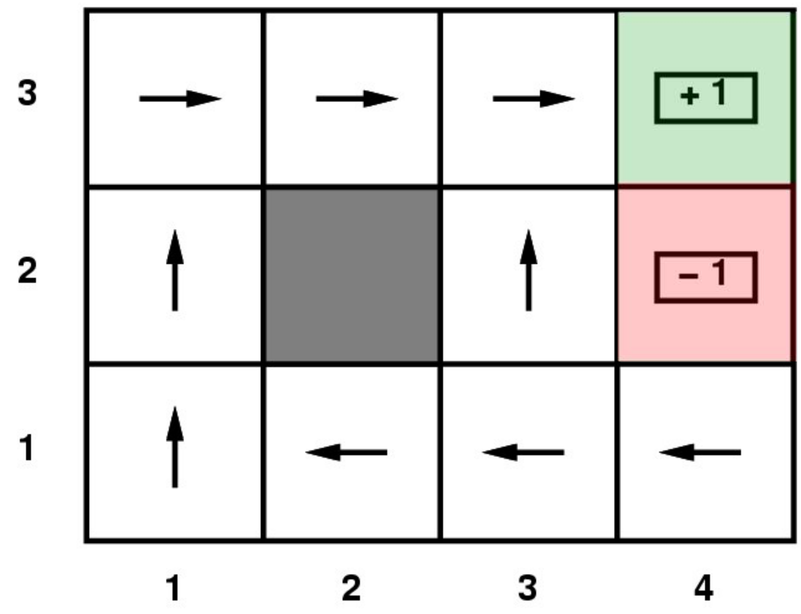
- Key: use observed ***reward to go*** of the state as the direct evidence of the actual expected utility of that state

- Averaging the reward to go samples will converge to true value at a state (empirical mean)

- Mont-Carlo Policy Evaluation

Passive RL

- Given policy π,
  - estimate $V^{\pi}(s)$
- Not given
  - transition matrix, nor
  - reward function!
- Simply follow the policy for many epochs
- Epochs: training sequences



(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1

0.57 0.64 0.72 0.81 0.9

(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1

(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1

Direct Estimation

- Converge very slowly to correct utilities values (requires a lot of sequences)

- Does not exploit Bellman on policy values

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

How can we incorporate constraints?

Adaptive Dynamic Programming (ADP)

- ADP is a model based approach
  - Follow the policy for a while
  - Estimate transition model based on observations
  - Learn reward function
  - Use estimated model to compute utility of policy

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s, a, s') V^{\pi}(s')$$

learned

- How can we estimate transition model T(s,a,s')?
  - Statistics: the fraction of times we see s' after taking a in state s.

Temporal Difference Learning (TD)

- Can we avoid the computational expense of full DP policy evaluation?
- Temporal Difference Learning
  - Model Free Method
  - Learns from incomplete episodes by bootstrapping
  - Approximate guesses from guesses
  - Do local updates of utility/value function on a per-action basis
  - Don't try to estimate entire transition function!
  - For each transition from s to s', we perform the following update:

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(R(s) + \beta V^{\pi}(s') - V^{\pi}(s))$$

learning rate (α)

TD target: $R(s) + \beta V^{\pi}(s')$

discount factor (β)

TD Error: $R(s) + \beta V^{\pi}(s') - V^{\pi}(s)$

Temporal Difference Learning (TD)

- Can we avoid the computational expense of full DP policy evaluation?
- Temporal Difference Learning
  - Do local updates of utility/value function on a per-action basis
  - Don't try to estimate entire transition function!
  - For each transition from s to s', we perform the following update:

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(R(s) + \beta V^{\pi}(s') - V^{\pi}(s))$$

learning rate

TD target

discount factor

TD Error

- Intutively, moves us closer to satisfying Bellman constraint

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s, a, s') V^{\pi}(s')$$

Temporal Difference Learning (TD)

- TD update for transition from s to s':

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(R(s) + \beta V^{\pi}(s') - V^{\pi}(s))$$

learning rate

(noisy) sample of utility based on next state

- So the update is maintaining a "mean" of the (noisy) utility samples
- If the learning rate decreases with the number of samples (e.g. 1/n) then the utility estimates will converge to true values

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s, a, s') V^{\pi}(s')$$

Temporal Difference Learning (TD)

- TD update for transition from s to s':

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(R(s) + \beta V^{\pi}(s') - V^{\pi}(s))$$

learning rate

(noisy) sample of utility based on next state

- When V satisfies Bellman constraints then **expected** update is 0.

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

N-step prediction

- TD(0):

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(R(s) + \beta V^{\pi}(s') - V^{\pi}(s))$$

learning rate

(noisy) sample of utility based on next state

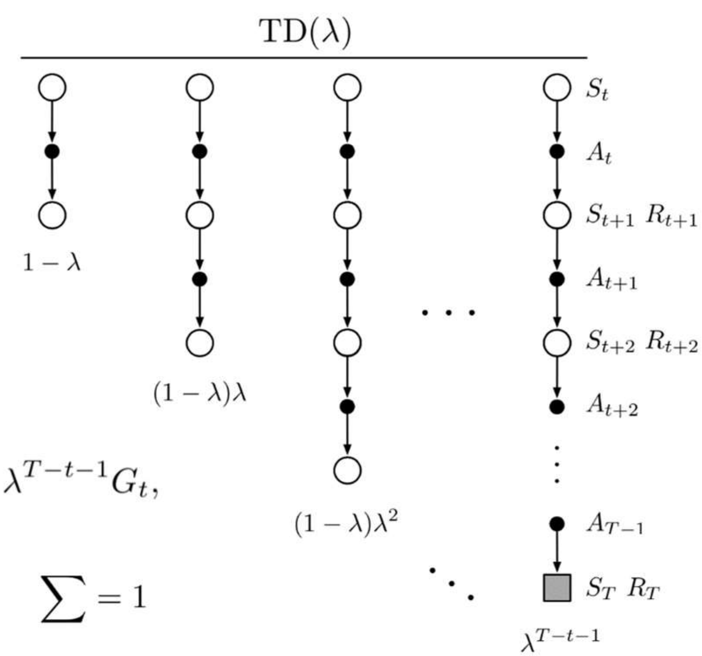
- update with 1-step prediction
- update with 2-steps
- update n+1-steps
  - $G^n = R(s) + \beta\, R(s_1) + \beta^2 R(s_2) + \ \dots + V^{\pi}(s_n)$
  - $V^{\pi}(s) = V^{\pi}(s) + \alpha(G^n - V^{\pi}(s))$
- Monte-Carlo: full evaluation

TD(λ)

- λ return:
  - $G^{\lambda} = (1-\lambda)\sum_{1}^{\infty} \lambda^{n-1} G^{n}$

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \sum_{n=0}^{\infty} \lambda^{n} = \frac{1}{1-\lambda}$$

- Forward-view TD(λ):
  - $V^{\pi}(s) = V^{\pi}(s) + \alpha(G^{\lambda} - V^{\pi}(s))$

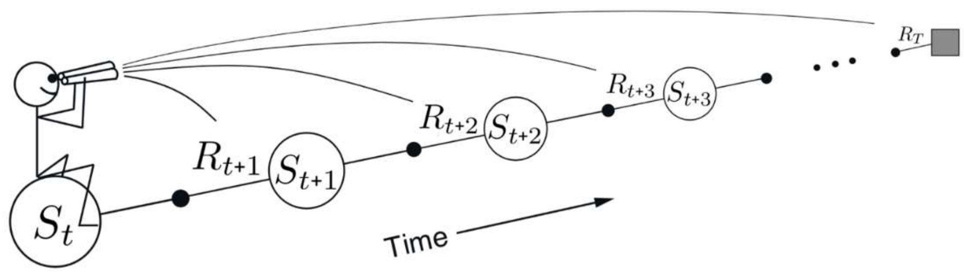


$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t,$$

$$\sum = 1$$

Forward view TD(λ)

- λ return:
  - $G^{\lambda} = (1-\lambda)\sum_{1}^{\infty} \lambda^{n-1} G^{n}$
- Forward-view TD(λ):
  - $V^{\pi}(s) = V^{\pi}(s) + \alpha(G^{\lambda} - V^{\pi}(s))$
- TD(0) is TD
- TD(1) is MC
- TD(λ) used for TD-Gammon



Backward view TD(λ)

- On-line version of TD($\boldsymbol{\lambda}$)
- Traces are collected backward, not forward
- Eligibility traces $E(s)$ that holds the decaying values of $V(s)$

instead of waiting for what is going to happen *next*, we remember what happened in the *past* and use current information to update the state-values for every state seen so far

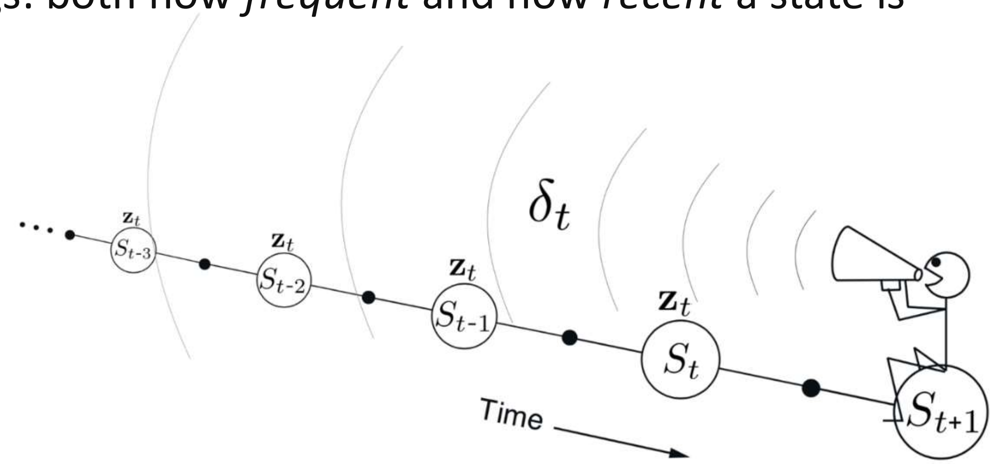
The eligibility traces combine two things: both how *frequent* and how *recent* a state is

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

$$E_0(s) = 0$$

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

$$V(S) = V(S) + \alpha\delta_t E_t(S)$$



Backward view TD(λ)

- On-line version of TD($\boldsymbol{\lambda}$)
- Traces are collected backward, not forward
- Eligibity traces $E(s)$ that holds the decaying values of $V(s)$

On-line Tabular TD(λ)

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in S$
Repeat (for each episode) :
  Initialize s
  Repeat (for each step of episode) :
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s :
      $V(s) \leftarrow V(s) + \alpha \delta e(s)$
      $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$
  Until s is terminal

Comparisons

- MC Estimation (model free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Adaptive Dynamic Programming (model based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Temporal Difference Learning (model free)
  - Update speed and implementation similar to direct estimation
  - Partially exploits Bellman constraints---adjusts state to 'agree' with observed successor
    - Not ***all*** possible successors
  - Convergence in between direct estimation and ADP