Reinforcement Learning 2

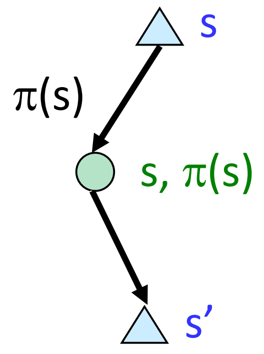
With many slides from Dan Klein and Pieter Abbeel and Percy Liang

Outline

- Q-learning recap + Example
- Exploit-Explore tradeoff (epsilon-greedy)
- Approximation
  - Least squares approximation
- Fun Example: AI Gym

Temporal Difference Learning (TD)

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often



- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

TD: Exponential Moving Average

- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1-\alpha)\cdot \bar{x}_{n-1} + \alpha \cdot x_n$
  - Makes recent samples more important:

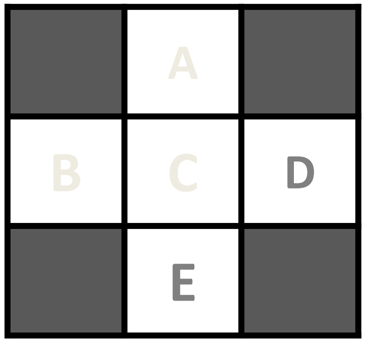
$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

  - Forgets about the past (distant past values were wrong anyway)

- Decreasing learning rate (alpha) can give converging averages

Example: Temporal Difference Learning (revised from last class)

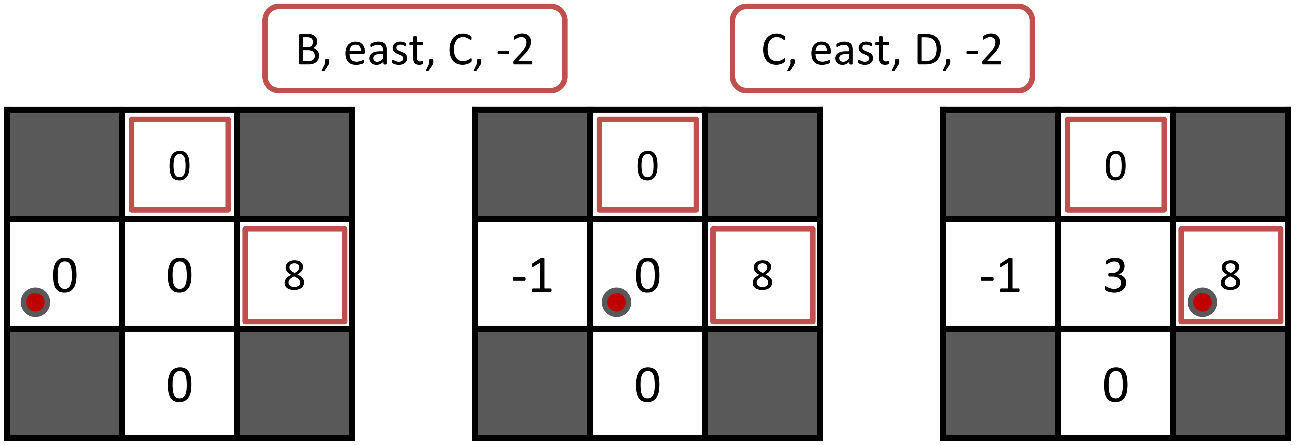
States



Assume:`

γ = 1,
α = 1/2

Observed Transitions

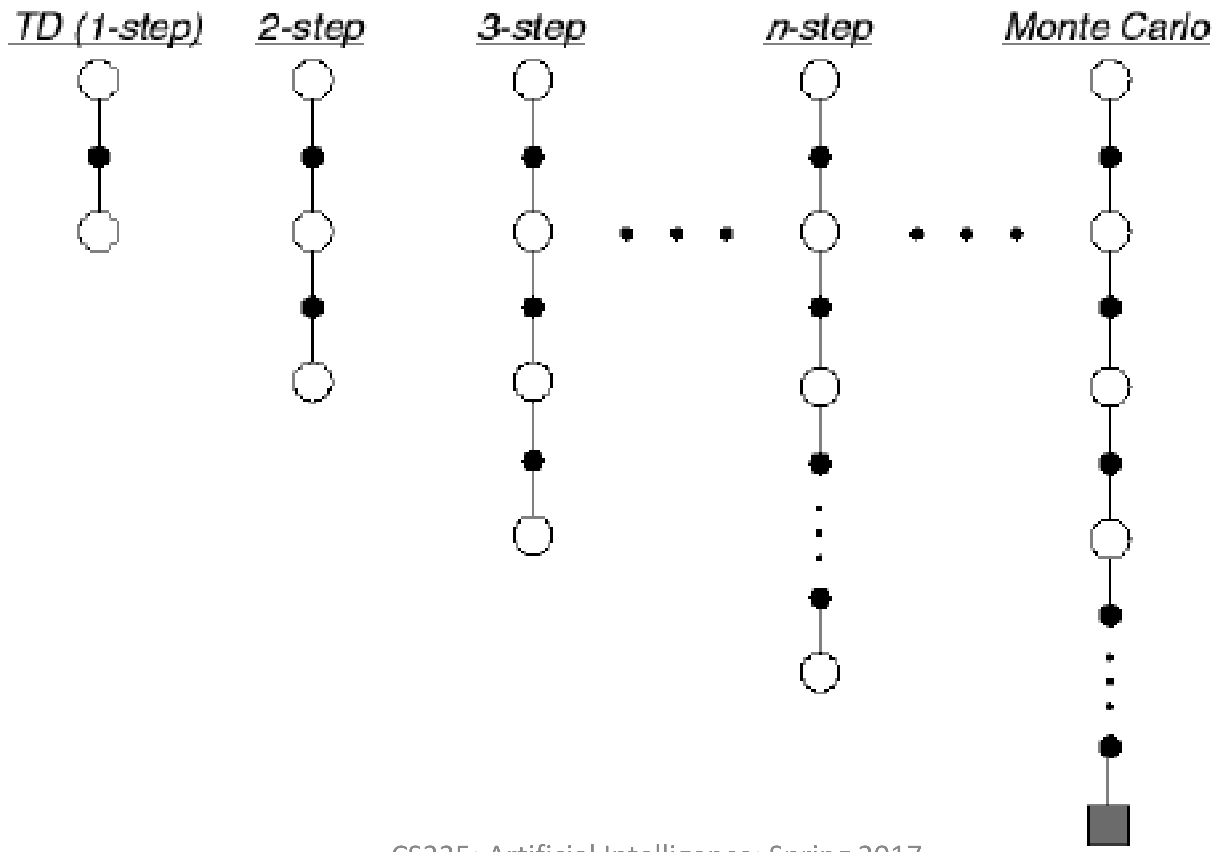


$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

0.5*0 + 0.5*[-2 + 1 * 8]

N-step TD Prediction

- Idea: Look farther into the future when you do TD backup (1, 2, 3, …, n steps)



R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

Mathematics of N-step TD Prediction

- Monte Carlo: $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$

- TD: $R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$
  - Use V to estimate remaining return

- n-step TD:
  - 2 step return: $R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$
  - n-step return: $R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$

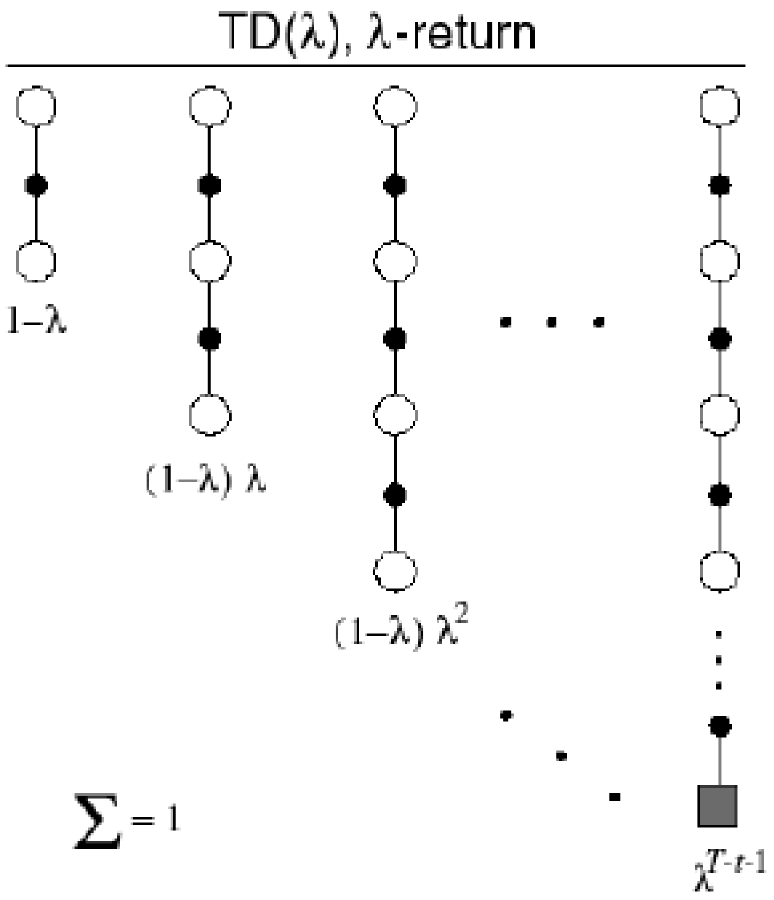
Forward View of TD(λ)

- TD(λ) is a method for averaging all n-step backups
  - weight by λ^{n-1} (time since visitation)
  - λ-return:

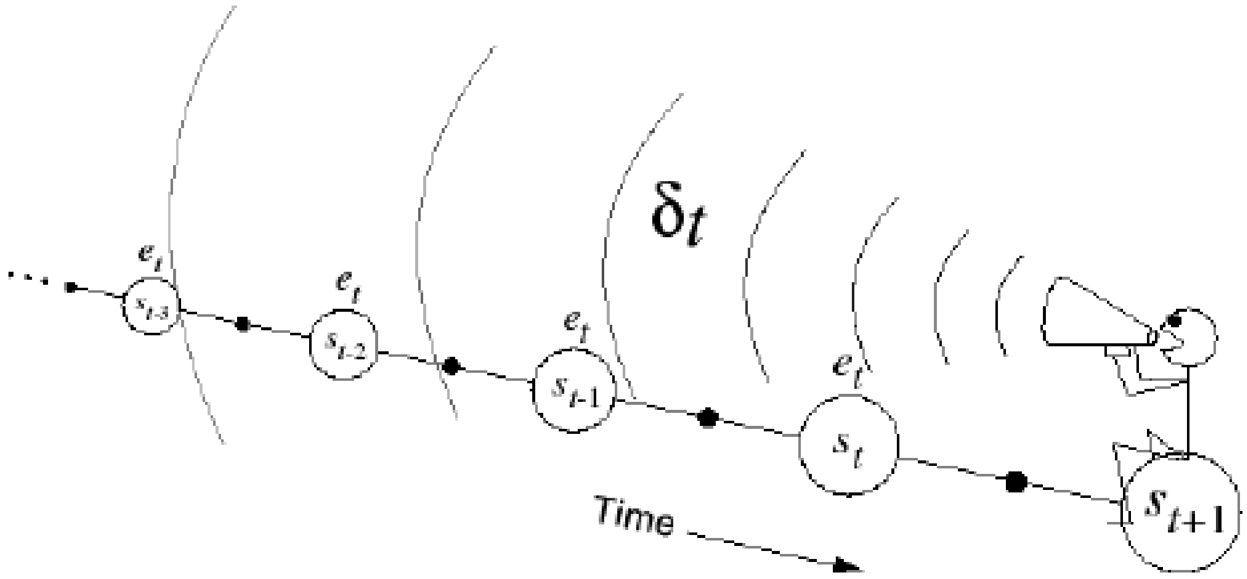
$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$$

- Backup using λ-return:

$$\Delta V_t(s_t) = \alpha\left[R_t^\lambda - V_t(s_t)\right]$$



Backward View



$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

- Shout δ_t backwards over time
- The strength of your voice decreases with temporal distance by $\gamma\lambda$

Advantages of TD Learning

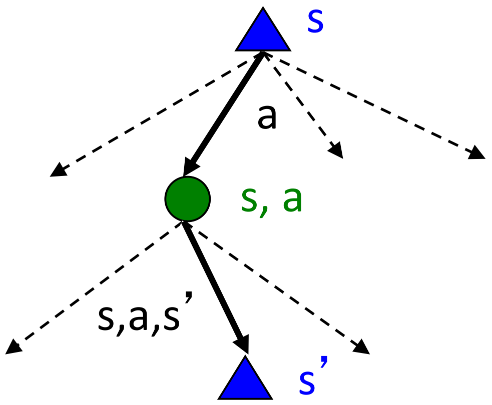
- Combines the "bootstrapping" (1-step self-consistency) idea of DP with the "sampling" idea; maybe the best of both worlds
- Doesn't need a model of the environment, only experience
- TD can be fully incremental
  - you can learn before knowing the final outcome
  - you can learn without the final outcome (from incomplete sequences)

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, if we want to turn values into a (new) policy, we're stuck:

$$\pi(s) = \arg\max_a Q(s,a)$$

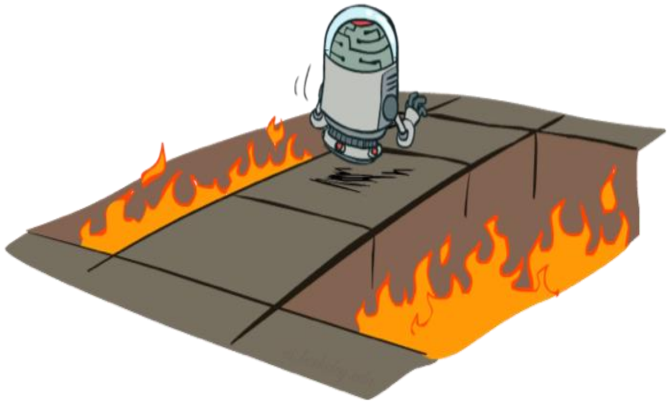
$$Q(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right]$$



- Idea: learn Q-values, not values
- Makes action selection model-free too!

Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values



- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Review: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right (why?)
  - Given V_k, calculate the depth k+1 values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right (why?)
  - Given Q_k, calculate the depth k+1 q-values for all q-states:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

Q-Learning Recap

- Q-Learning: sample-based Q-value iteration

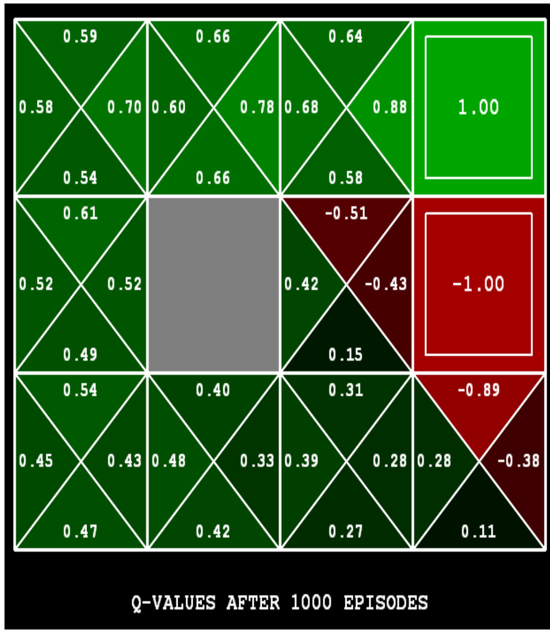
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

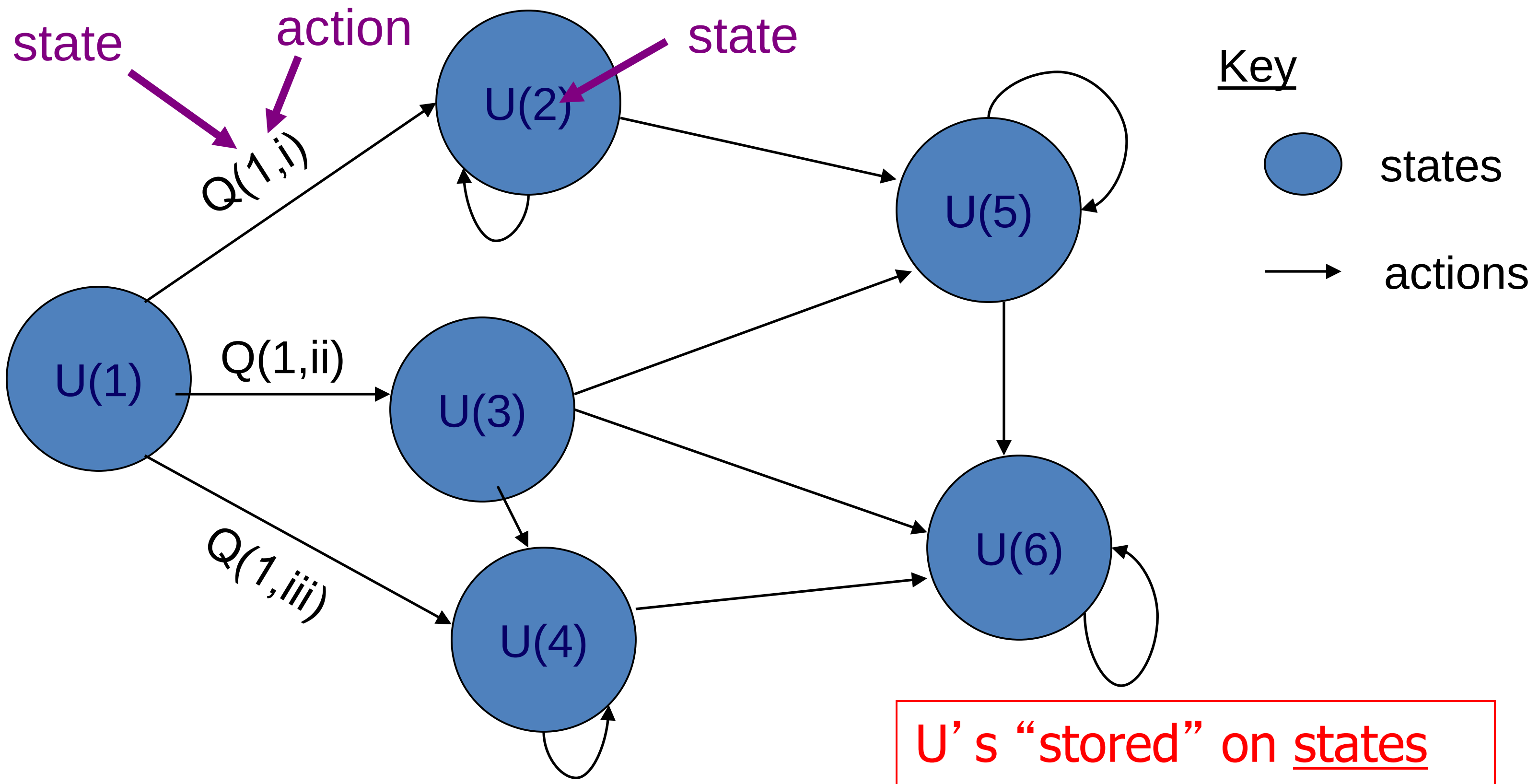
$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate new estimate in running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$



Q vs. *U* Visually



U’s “stored” on <u>states</u>

Q’s “stored” on <u>arcs</u>

Q-Learning (Watkins PhD, 1989)

Let Q_t be our current estimate of the correct Q

Our current policy is

$$\Pi_t(s) = a, \qquad Q_t(s,a) = \max_{\substack{b \in known \\ actions}} [Q_t(s,b)]$$

Our current utility-function estimate is

$$U_t(s) = Q_t(s, \Pi_t(s))$$

- hence, the U table is embedded in the Q table and we don't need to store both

Q-Learning (cont.)

Assume we are in state S_t

"Run the program"[(1)] for awhile (*n* steps)

1. Determine actual reward and compare to predicted reward
2. Adjust prediction to reduce error

(1) I.e., follow the current policy

One-Step Q-Learning Algorithm (Deterministic version)

```
0.  S ← initial state

While true: #or exceed number of training episodes
    1.  if random r ≤ P
            then a = random legal action from S
            else a = $\Pi_t$(S)

    2.  $S_{next}$ ← V(S, a)         } Act on world and get reward
        $R_{immed}$ ← R($S_{next}$)  }

    3.  Q(S, a) ← $R_{immed}$ + $\gamma$ $Max_{a'}$ Q($S_{next}$, a')
         #update Q as max of possible Q values from $S_{next}$
    4.  S ← $S_{next}$
```

In Stochastic World, Don't Trash Current Q Entirely… Update.

3. $Q(S, a) \leftarrow R_{immed} + g\,\max_{a'} Q(S_{next}, a')$

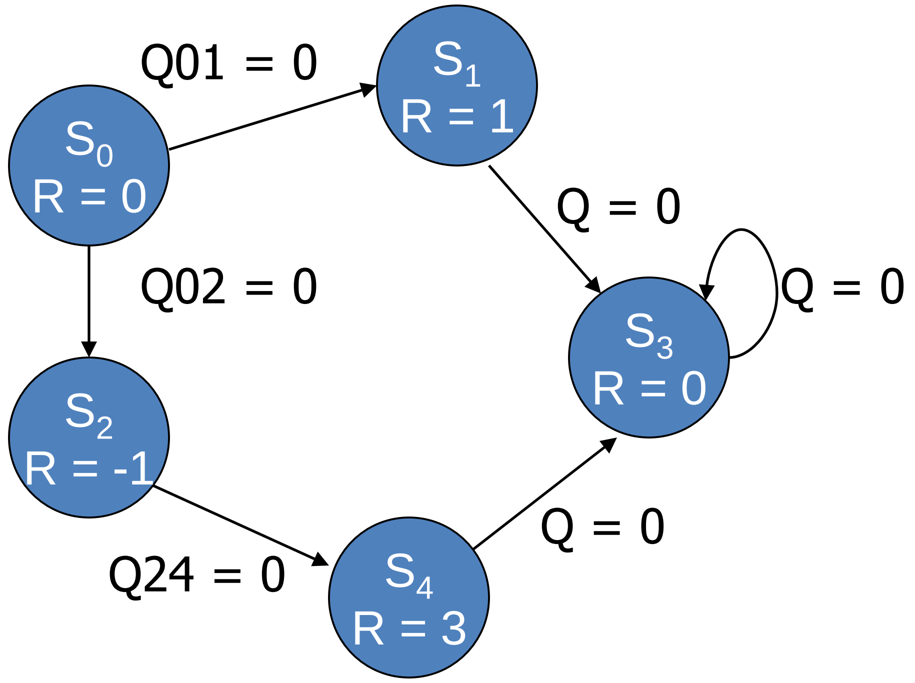
To:

Learning rate

3. $Q(S, a) \leftarrow \alpha\, [R_{immed} + g\,\max_{a'} Q(S_{next}, a')] + (1-\alpha)\, Q(S, a)$

Q-Learning Example

(with updates after each step, *N* =1)



Let γ = 2/3

<u>Algo</u>: Pick State +Action

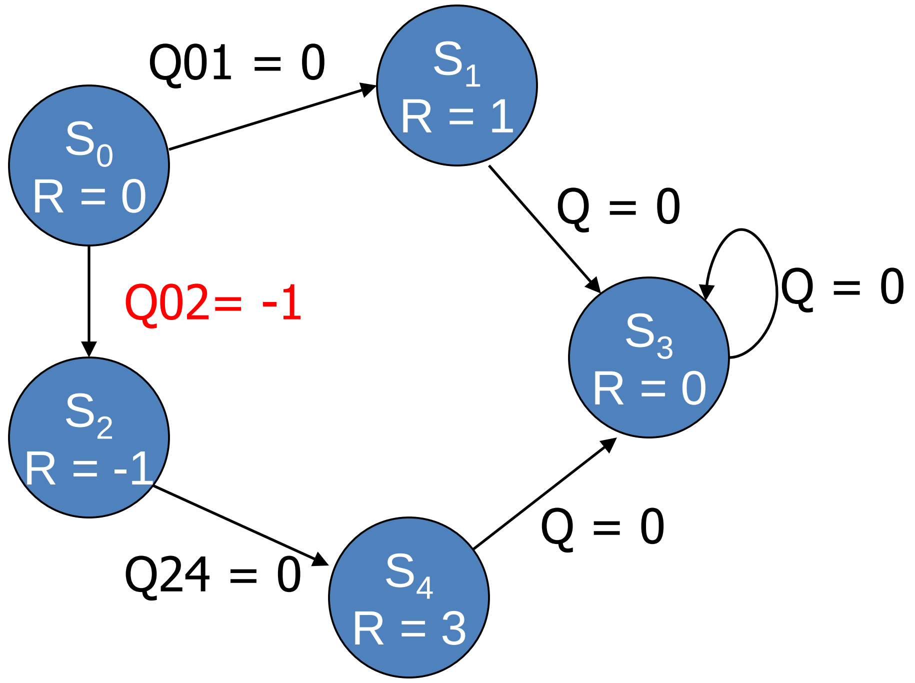
$$Q_{new} = R + \gamma_{\max} Q_{next\ state}$$

Repeat

(deterministic world, so α=1)

Example (Step 1)

$S_0 \rightarrow S_2$



Let γ = 2/3

<u>Algo</u>: Pick State +Action

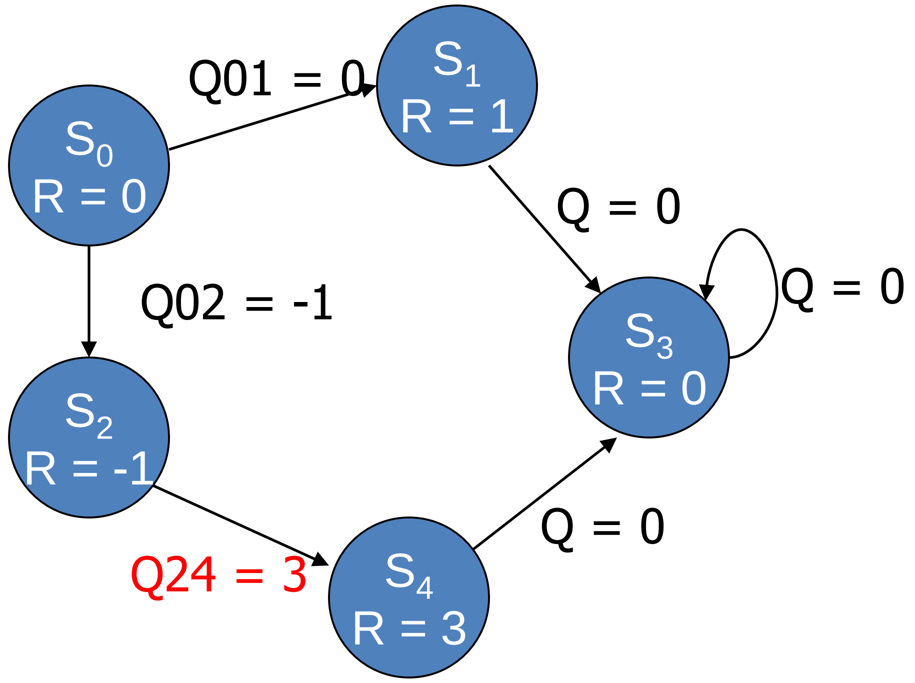
$$Q_{new} = R + \gamma_{\max} Q_{next\ state}$$

Repeat

(deterministic world, so α=1)

A Simple Example (Step 2)

$S_2 \rightarrow S_4$



Let γ = 2/3

Algo: Pick State +Action

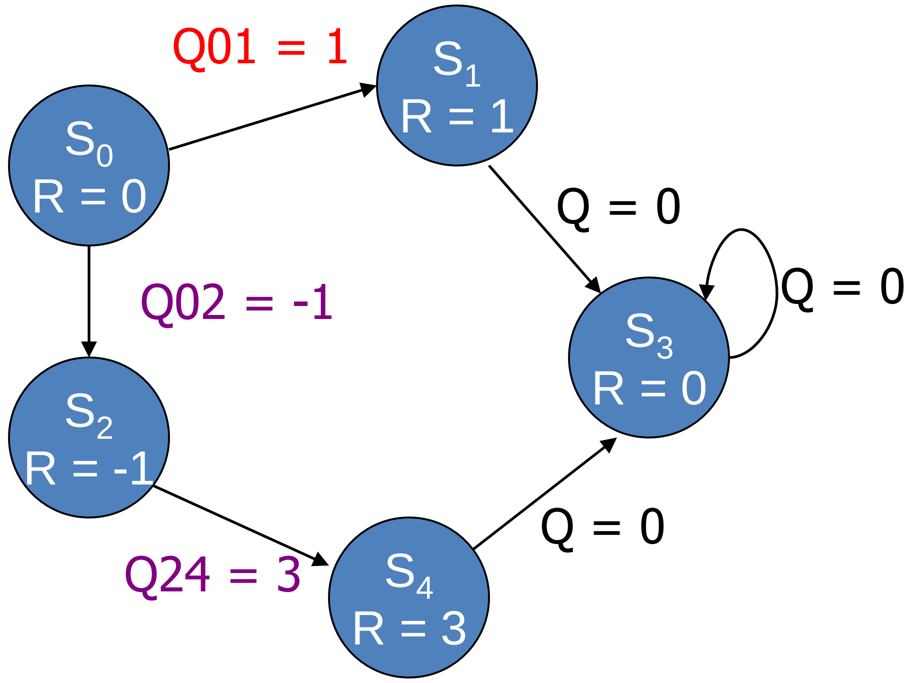
$$Q_{new} = R + \gamma_{\max} Q_{next\ state}$$

Repeat

(deterministic world, so α=1)

Example (Step 3)

$S_0 \rightarrow S_1$



Let γ = 2/3

<u>Algo</u>: Pick State +Action

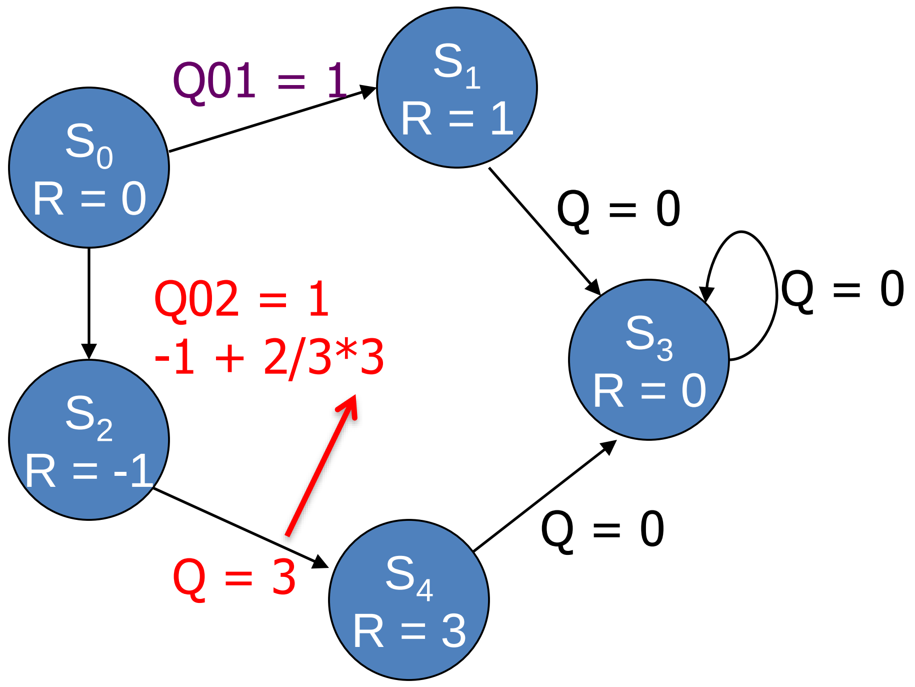
$$Q_{new} = R + \gamma \max Q_{next\ state}$$

Repeat

(deterministic world, so $\alpha=1$)

Example (Step 3), cont

$S_0 \rightarrow S_1$, update Q02 using Max(Q2*)



Let γ = 2/3

Algo: Pick State +Action

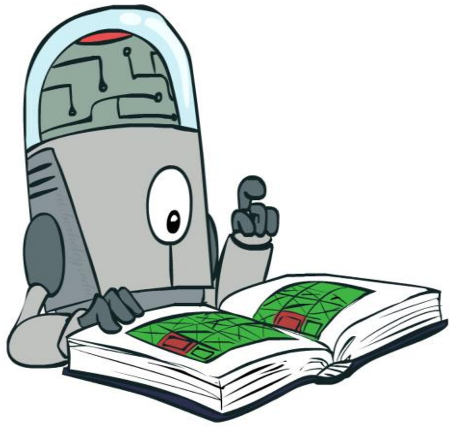
$$Q_{new} = R + \gamma_{\max} Q_{next\ state}$$

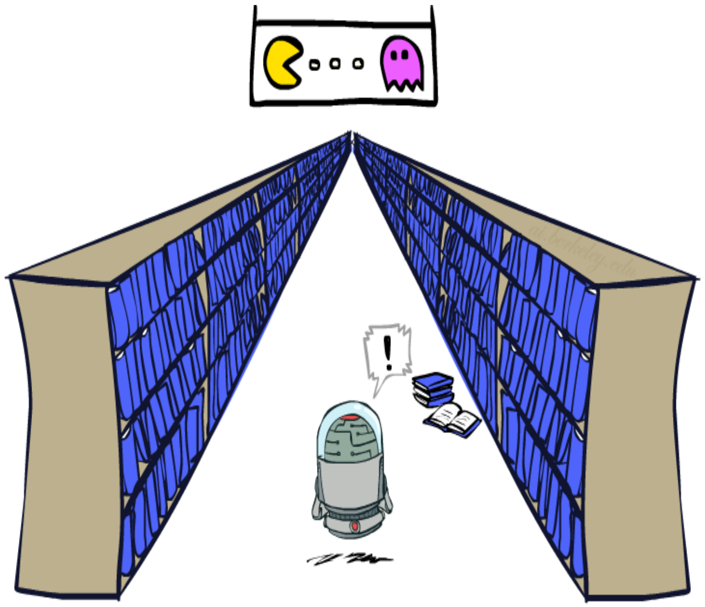
Repeat

(deterministic world, so α=1)

Generalizing Across States

- Basic Q-Learning keeps a **table** of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again





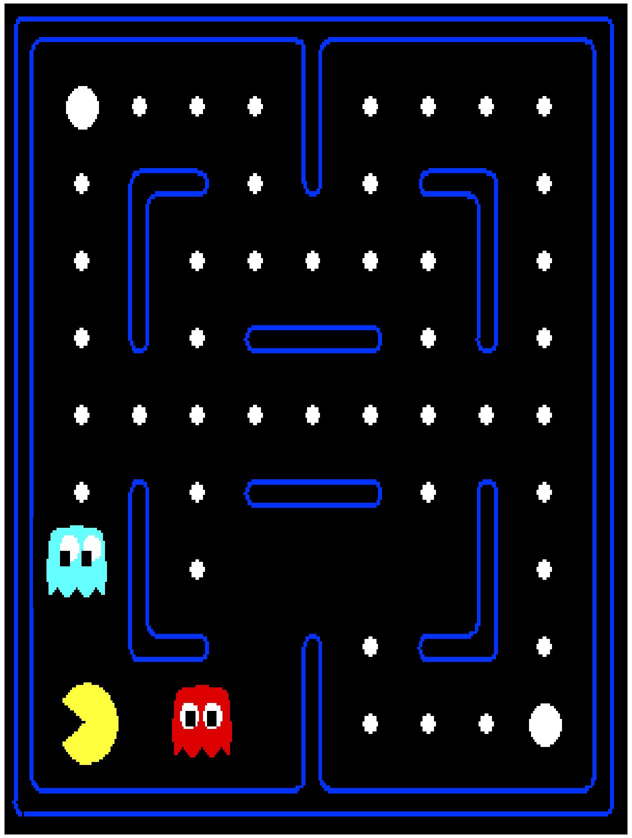
[demo – RL]

RL and Function Approximation

- Exact Q-learning infeasible for many real applications due to curse of dimensionality: |S*A| table too big.
- Function Approximation (FA) is a way to "lift the curse:"
  - complexity D of FA needed to capture regularity in environment may be << |S|.
  - no need to sweep thru entire state space: train on N "plausible" samples and then generalize to similar samples drawn from the same distribution.

Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!

Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

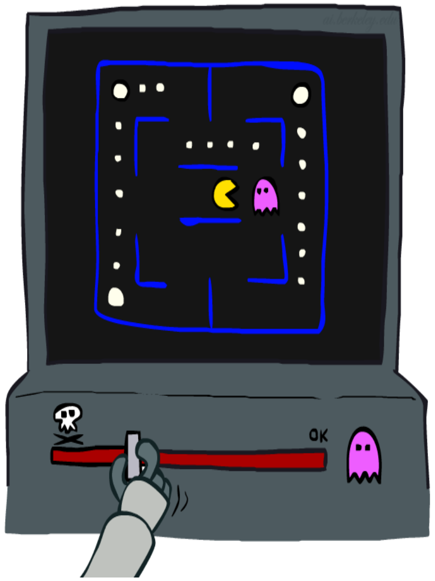
- Q-learning with linear Q-functions:

$$\text{transition } = (s, a, r, s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}] \qquad \text{Exact Q's}$$

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a) \qquad \text{Approximate Q's}$$

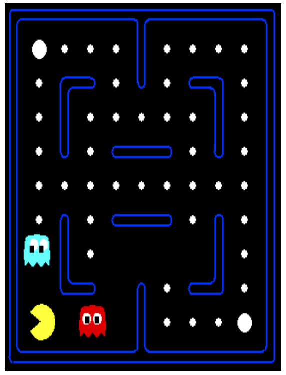


- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, *blame the features that were on*: dislike all states with that state's features

Example: Q-Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$

s



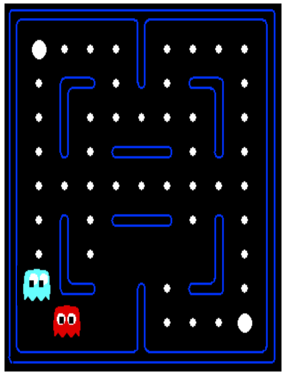
$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

$a = \text{NORTH}$

$r = -500$

s'



$Q(s, \text{NORTH}) = +1$

$Q(s', \cdot) = 0$

$r + \gamma \max_{a'} Q(s', a') = -500 + 0$

difference $= -501$

$w_{DOT} \leftarrow 4.0 + \alpha [-501] \, 0.5$

$w_{GST} \leftarrow -1.0 + \alpha [-501] \, 1.0$

$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Linear Combination of Features (Proj 3)

- Estimate Q(S,a) as weighted sum of features (e.g., for pacman, can use exactly same features as in Proj 2):

  Q(S,a) = a1*f1 + a2*f2 + …. + ak*fk
  Q(S,b) = b1*f1 + b2*f2 + …. + bk*fk

- Use linear regression to estimate w's:
- For each update of Q(S,a):
  – Update a1…ak s.t. min(MSE)

Exploration vs. Exploitation

In order to learn about better alternatives, we can’t always follow
the current policy (“exploitation”)

Sometimes, need to try
“random” moves (“exploration”)

Exploration vs. Exploitation (cont)

Approaches

1) *p* percent of the time, make a random move; could let

$$p = \frac{1}{\sqrt{\#moves\_made}}$$

2) Prob(picking action *A* in state *S*)

$$= \frac{const^{Q(S,A)}}{\sum_{i \in actions} const^{Q(S,i)}}$$

Exponentia-ting removes negative values

How to Explore?

- Several schemes for forcing exploration
  - Simple, effective: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy
  - Problems with random actions?
    - You do eventually explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

Idea: Exploration Functions

- When to explore?
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

- Exploration function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g.

$$f(u,n) = u + k/n$$

  - Note: this propagates the "bonus" back to states that lead to unknown states as well!

Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

Fun Examples From Denis: AI Gym: CartPole, FrozenLake

- Sites to experiment with Reinforcement Learning:
  - https://www.microsoft.com/en-us/research/project/project-malmo/
  - http://allenai.org/
  - https://deepmind.com/
  - https://universe.openai.com/
  - ➢ https://gym.openai.com/

- AIGym examples: CartPole and FrozenLake:
  - Need to **discretize** action space
  - Details: in zip posted in Piazza resources