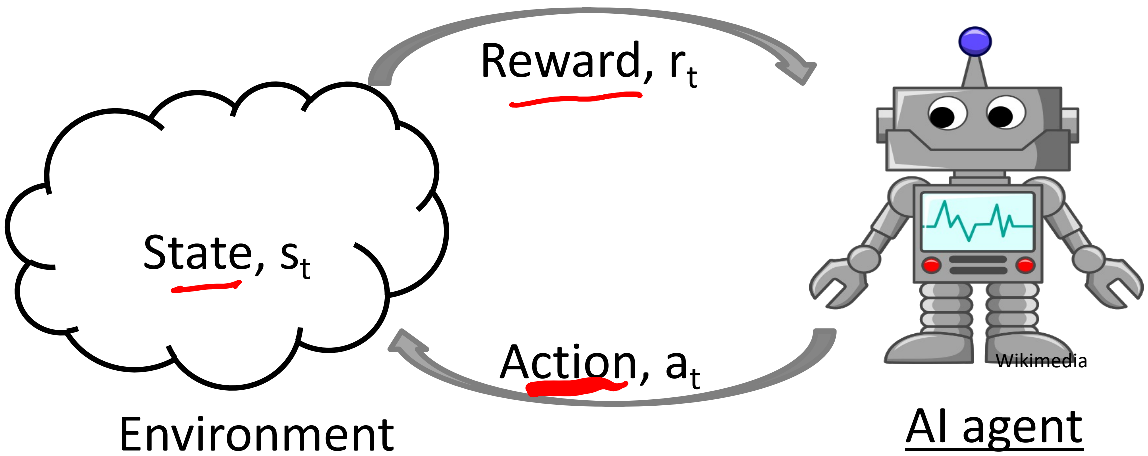
Reinforcement Learning II

Aarti Singh

Machine Learning 10-701
Apr 5, 2023

Slides courtesy: Henry Chai, Eric Xing

RL setup



MDP:

1. Start in some initial state s_0
2. For time step t:
   a. Agent observes state s_t
   b. Agent takes action $a_t = \pi(s_t)$ π- policy
   c. Agent receives reward $r_t \sim p(r \mid s_t, a_t)$
   d. Agent transitions to state $s_{t+1} \sim p(s' \mid s_t, a_t)$

RL setup

- Policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$
  - Specifies an action to take in *every* state
- Value function, $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$
  - $V^\pi(s) = \mathbb{E}[$*discounted* total reward of starting in state s and executing policy π forever$]$

$$= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}\left[R\left(s_t, \pi(s_t)\right)\right]$$

$0 \le \gamma < 1$

R – deterministic reward

- **Goal:** Find policy that maximizes expected discounted total reward

$$\pi^* = \operatorname*{argmax}_{\pi} \; V^\pi(s) \;\; \forall \, s \in \mathcal{S}$$

Bellman Equation

Value function satisfies the set of recursive equations:

$$\boxed{V^{\pi}(s) = R\big(s,\pi(s)\big) + \gamma \sum_{s_1 \in \mathcal{S}} p\big(s_1 \mid s,\pi(s)\big) V^{\pi}(s_1)}$$

- Optimal value function:

$$V^*(s) = \max_{a \in \mathcal{A}} \Big[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')\Big]$$

  - System of $|\mathcal{S}|$ equations and $|\mathcal{S}|$ variables – nonlinear!

- Optimal policy:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')$$

Value iteration

known

- Inputs: $R(s,a)$, $p(s' \mid s,a)$, $0 < \gamma < 1$
- Initialize $V^{(0)}(s) = 0\ \forall\ s \in \mathcal{S}$ (or randomly) and set $t = 0$
- While not converged, do:

$\overset{\pi}{V}(s)$ - value of a state

  - For $s \in \mathcal{S}$

$$V^{(t+1)}(s) \leftarrow \max_{a \in \mathcal{A}} \underbrace{\left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(t)}(s')\right]}_{Q(s,a)}$$

Q-function

value of state - action pair

  - $t = t + 1$

- For $s \in \mathcal{S}$

$$\pi^*(s) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left[R(s,a) + \gamma \textstyle\sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(t)}(s')\right]$$

- Return π^*

Value iteration

- Inputs: $R(s,a)$, $p(s' \mid s,a)$, $0 < \gamma < 1$ known
- Initialize $V^{(0)}(s) = 0 \; \forall \, s \in \mathcal{S}$ (or randomly) and set $t = 0$
- While not converged, do:
  - For $s \in \mathcal{S}$
    - For $a \in \mathcal{A}$
      $$Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(t)}(s')$$
    - $V^{(t+1)}(s) \leftarrow \max_{a \in \mathcal{A}} Q(s,a)$
  - $t = t + 1$
- For $s \in \mathcal{S}$
  $$\pi^*(s) \leftarrow \operatorname*{argmax}_{a \in \mathcal{A}} Q(s,a)$$
- Return π^*

Value iteration: convergence

Theorem 1: Value function convergence

V will converge to V^* if each state is "visited" infinitely often (Bertsekas, 1989)

Theorem 2: Convergence criterion

if $\max_{s \in \mathcal{S}} \left| V^{(t+1)}(s) - V^{(t)}(s) \right| < \epsilon$,

then $\max_{s \in \mathcal{S}} \left| V^{(t+1)}(s) - V^*(s) \right| < \frac{2\epsilon\gamma}{1-\gamma}$ (Williams & Baird, 1993)

Theorem 3: Policy convergence

The "greedy" policy, $\pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q(s,a)$, converges to the optimal π^* in a finite number of iterations, often before the value function has converged! (Bertsekas, 1987)

Policy iteration

➢ Can we learn the policy directly, instead of first learning the value function?

known

- Inputs: $R(s, a)$, $p(s' \mid s, a)$, $0 < \gamma < 1$
- Initialize π randomly

$V^{*\pi}(s) = \max\limits_{a \in A} R + \ldots$

- While not converged, do:
  - Solve the Bellman equations defined by policy π

Now linear!

$$V^{\pi}(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s' \in \mathcal{S}} p\big(s' \mid s, \pi(s)\big) V^{\pi}(s')$$

solve for $V^{\pi}(s)$

  - Update π

$$\pi(s) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}} \; R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) V^{\pi}(s')$$

- Return π

Policy iteration: convergence

$|A| = 2$ 2 2 2

$|S| = 3$ 2^3

- Number of policies: $|A|^{|S|}$
- Policy improves each iteration
- Thus, the number of iterations needed to converge is bounded!

- Empirically, policy iteration requires fewer iterations than value iteration.

Next Questions

- How to handle unknown state transition and reward functions? ✓

- How to handle continuous states and actions?

Optimal Q function and policy

- Deterministic rewards
- $Q^*(s,a) = \mathbb{E}[$total discounted reward of taking action a in state s, assuming all future actions are optimal$]$

$$= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')$$

$$V^*(s') = \max_{a' \in \mathcal{A}} Q^*(s',a')$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \left[\max_{a' \in \mathcal{A}} Q^*(s',a') \right]$$

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

- Insight: if we know Q^*, we can compute an optimal policy π^*!

Optimal Q function and policy

- Deterministic rewards and state transitions
- $Q^*(s,a) = \mathbb{E}[$total discounted reward of taking action a in state s, assuming all future actions are optimal$]$

$$= R(s,a) + \gamma V^*\big(\delta(s,a)\big)$$

- $V^*\big(\delta(s,a)\big) = \max\limits_{a' \in \mathcal{A}} Q^*(\delta(s,a), a')$

$$Q^*(s,a) = \underbrace{R(s,a)}_{r} + \gamma \max_{a' \in \mathcal{A}} Q^*(\underbrace{\delta(s,a)}_{s'}, a')$$

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

- Insight: if we know Q^*, we can compute an optimal policy π^*!

Online Q-learning

- Inputs: discount factor γ, an initial state s

- Initialize $Q(s, a) = 0 \; \forall \; s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - Take a random action a

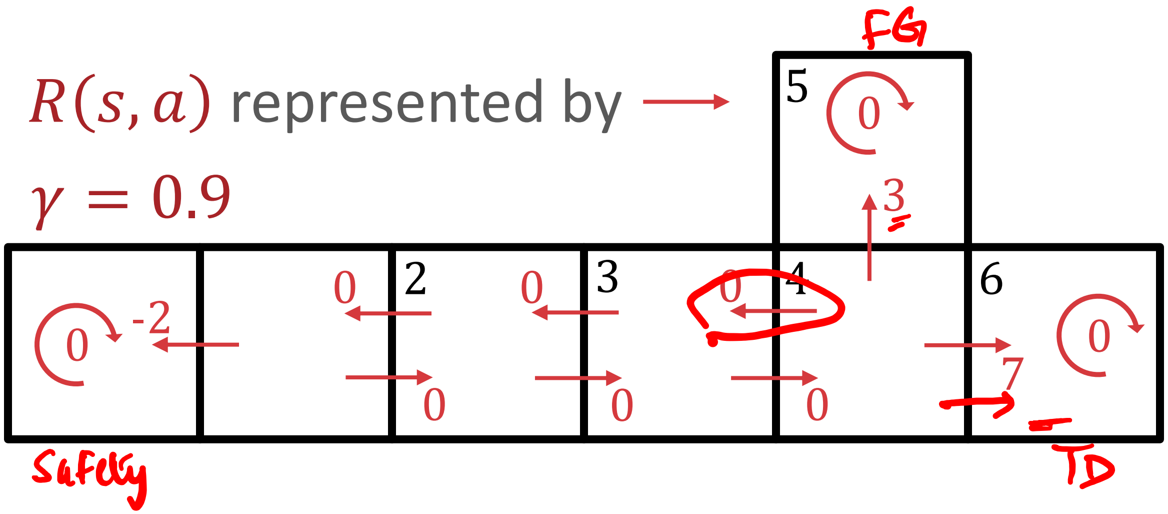
  - Receive reward $r = R(s, a)$
  - Update the state: $s \leftarrow s'$ where $s' = \delta(s, a)$
  - Update $Q(s, a)$:

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

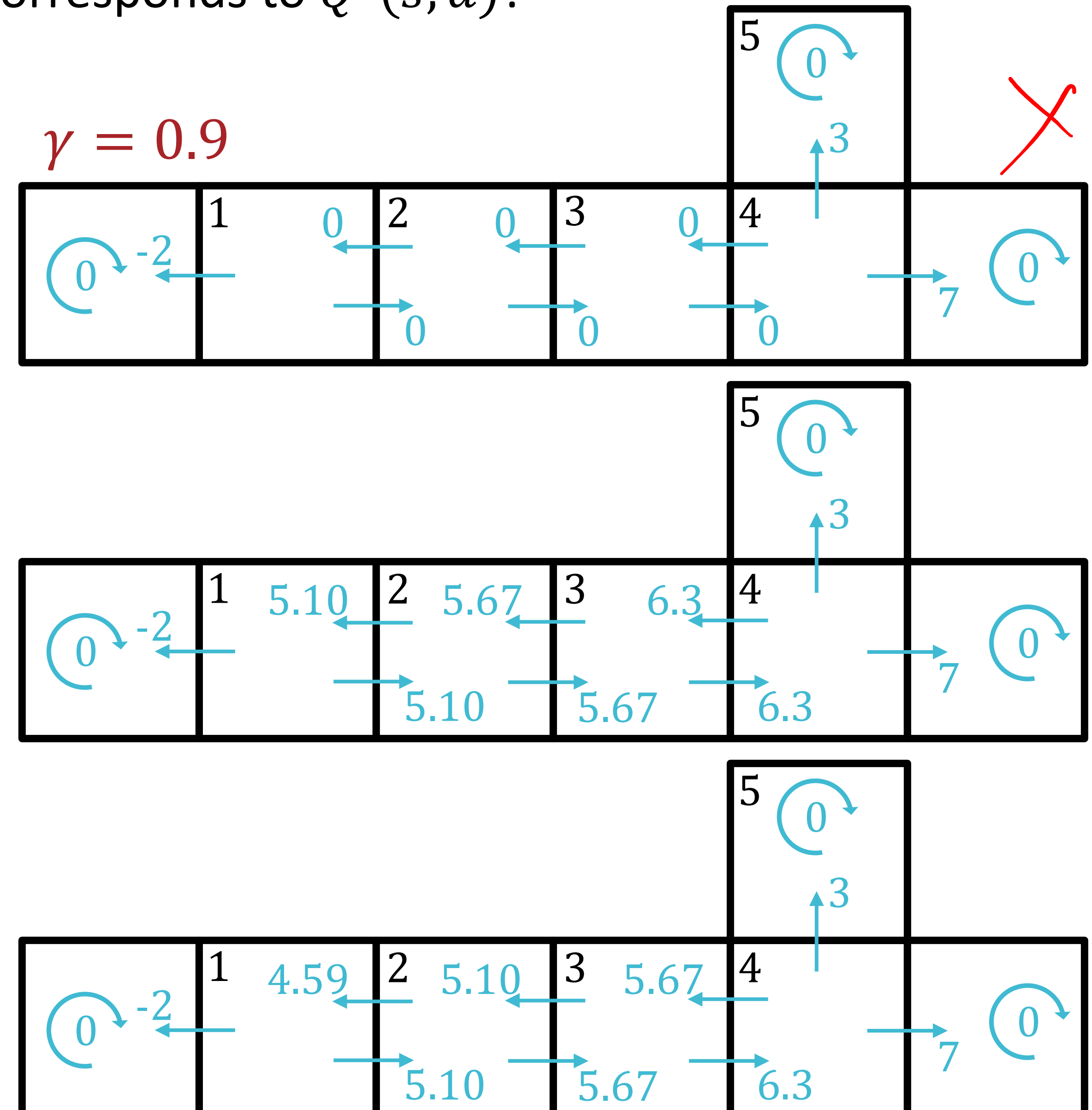
Q-learning example

$R(s, a)$ represented by $\longrightarrow$

$\gamma = 0.9$



Which set of blue arrows (roughly) corresponds to $Q^*(s, a)$?

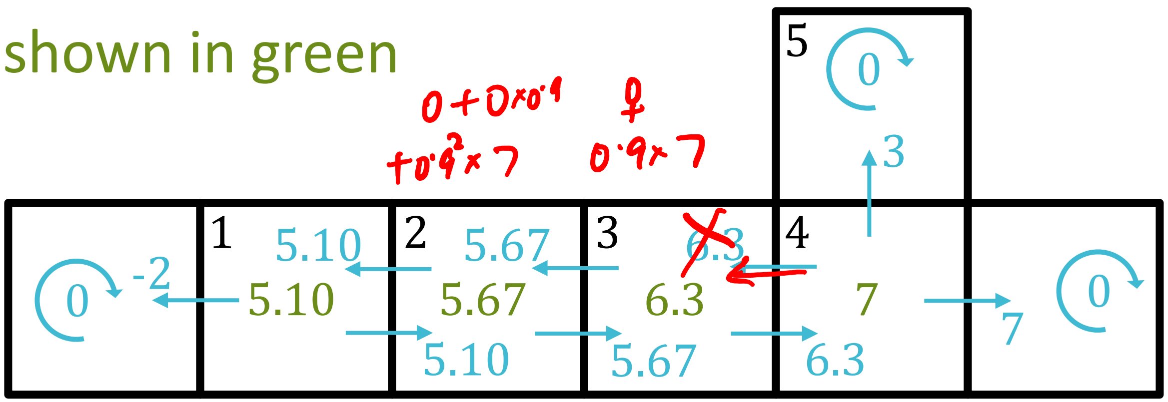


Which set of blue arrows
(roughly) corresponds to $Q^*(s,a)$?

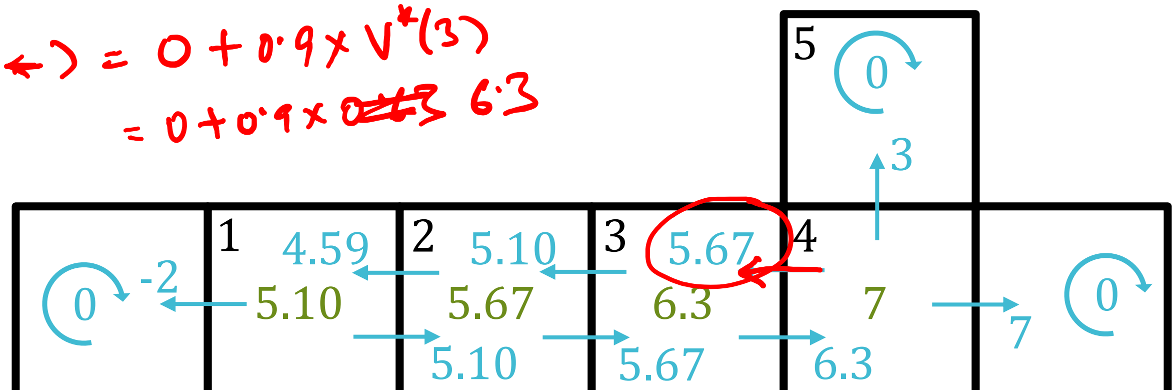
$$Q^*(s,a) = \underline{R(s,a)} + \gamma \underline{V^*(\delta(s,a))}$$

$V^*(s)$ shown in green

$0 + 0 \times 0.9 + 0.9^2 \times 7$ 0.9×7

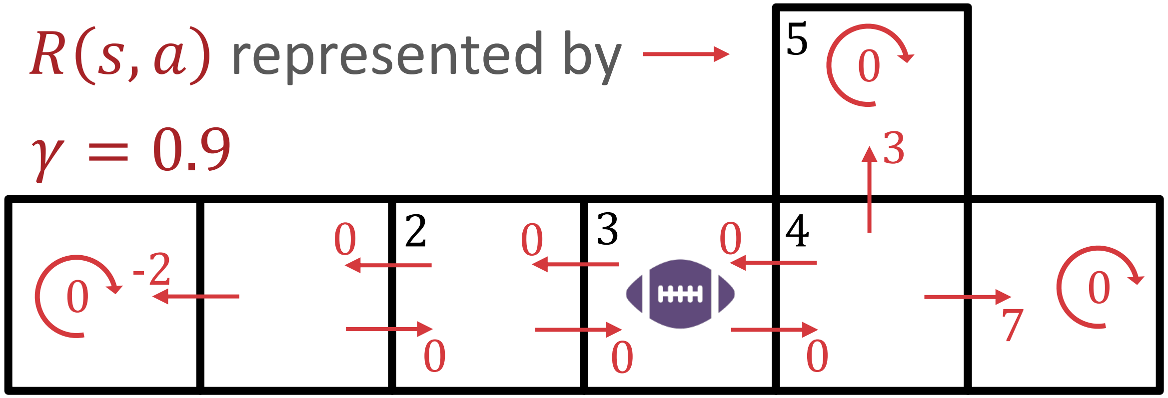


$Q^*(4, \leftarrow) = 0 + 0.9 \times V^*(3)$
$= 0 + 0.9 \times \cancel{0.63}\ 6.3$



$R(s,a)$ represented by →

$\gamma = 0.9$



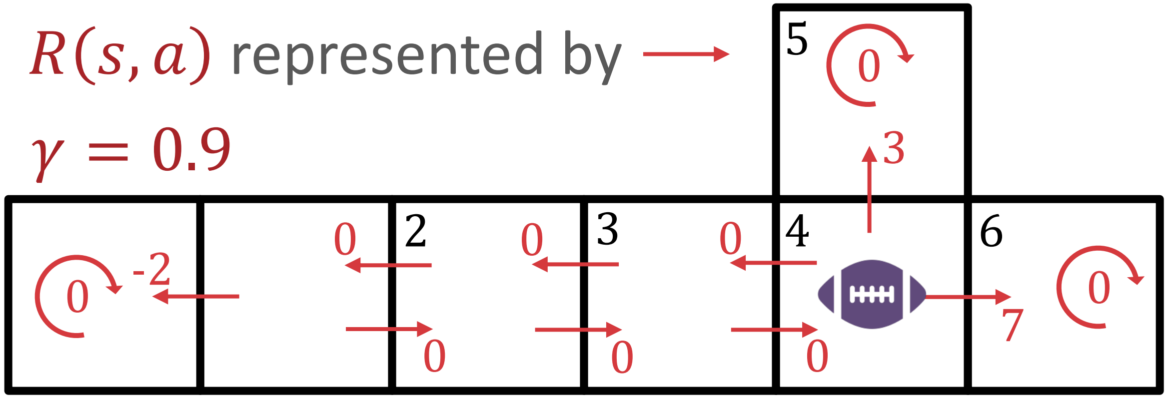
a →

s ↓

$Q(s,a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

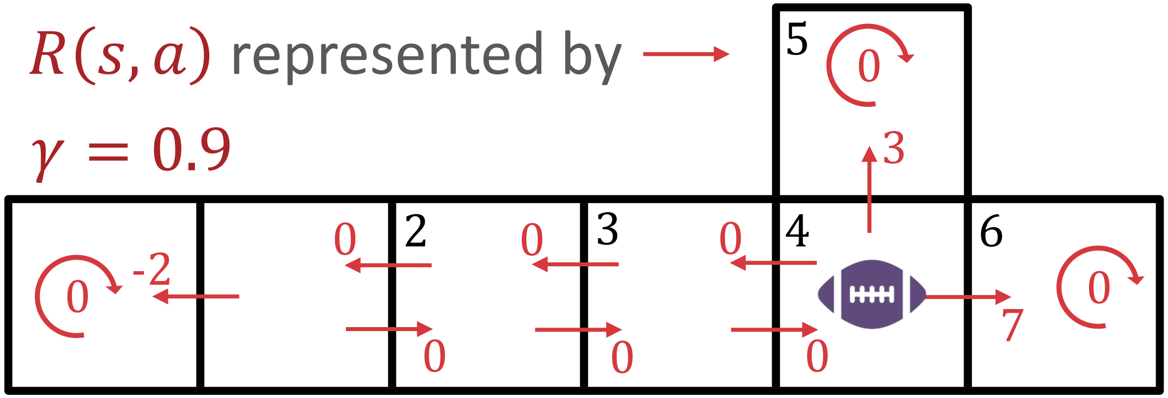
$R(s, a)$ represented by →

$\gamma = 0.9$



$$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \circlearrowright\}} Q(4, a') = 0$$

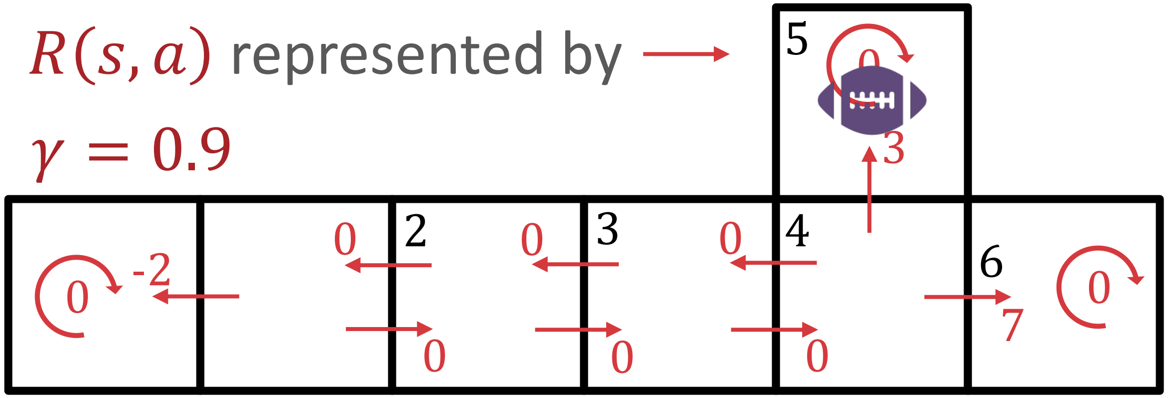
$Q(s,a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

$R(s,a)$ represented by → 

$\gamma = 0.9$

$Q(s,a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

$R(s,a)$ represented by →



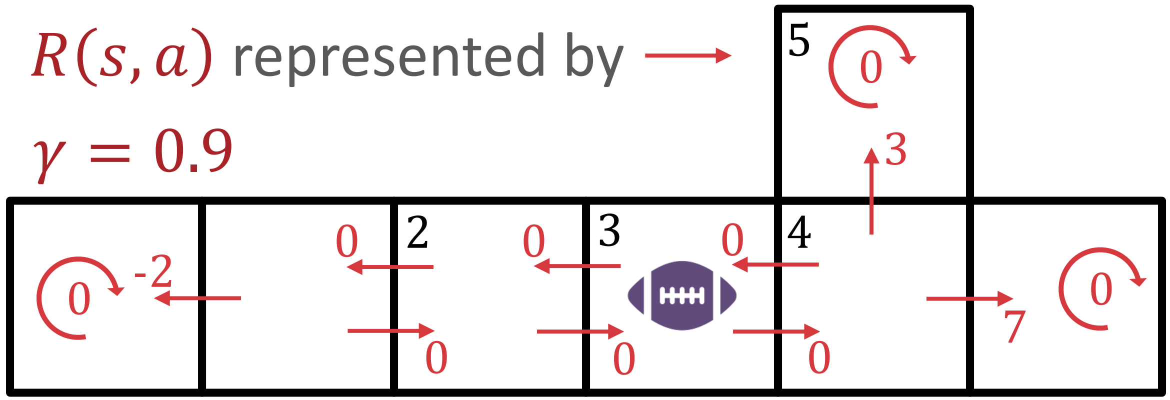
$\gamma = 0.9$

$$Q(4,\uparrow) \leftarrow 3 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(5,a') = 3$$

$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

$R(s,a)$ represented by $\longrightarrow$

$\gamma = 0.9$

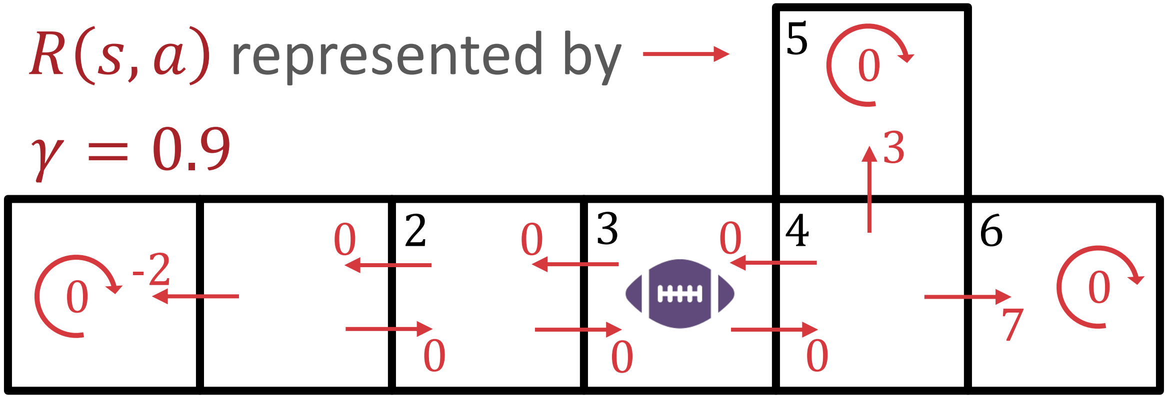


$$Q(3,\rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(4,a') = 2.7$$

$Q(s,a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	3	0
5	0	0	0	0
6	0	0	0	0

$R(s, a)$ represented by $\longrightarrow$

$\gamma = 0.9$



$$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \circlearrowright\}} Q(4, a') = 2.7$$

$Q(s,a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	2.7	0	0	0
4	0	0	3	0
5	0	0	0	0
6	0	0	0	0

Online Q-learning

- Inputs: discount factor γ, an initial state s

- Initialize $Q(s,a) = 0 \; \forall \; s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - Take a random action a
  - Receive reward $r = R(s,a)$
  - Update the state: $s \leftarrow s'$ where $s' = \delta(s,a)$
  - Update $Q(s,a)$:

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

ε-greedy Online Q-learning

- Inputs: discount factor γ, an initial state s,
  greediness parameter $\epsilon \in [0, 1]$

- Initialize $Q(s, a) = 0 \; \forall \, s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - With probability ϵ, take the greedy action

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

    Otherwise, with probability $1 - \epsilon$, take a random action a explore
  - Receive reward $r = R(s, a)$
  - Update the state: $s \leftarrow s'$ where $s' = \delta(s, a)$
  - Update $Q(s, a)$:

    $$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

Stochastic Transitions

$s, a \to s'$

$p(s' | s, a)$

$p(s' | s, a)$

- Inputs: discount factor γ, an initial state s,
  greediness parameter $\epsilon \in [0, 1]$,
  learning rate $\alpha \in [0, 1]$ ("trust parameter")
- Initialize $Q(s, a) = 0 \; \forall \, s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - With probability ϵ, take the greedy action

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

    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s, a)$
  - Update the state: $s \leftarrow s'$ where $s' \sim \underline{p(s' \mid s, a)}$
  - Update $Q(s, a)$:

$$Q(s, a) \leftarrow (1 - \alpha) \underbrace{Q(s, a)}_{\text{Current value}} + \alpha \underbrace{\left(r + \gamma \max_{a'} Q(s', a') \right)}_{\text{Update w/ deterministic transitions}}$$

Temporal Difference Learning

- Inputs: discount factor γ, an initial state s,
  greediness parameter $\epsilon \in [0, 1]$,
  learning rate $\alpha \in [0, 1]$ ("trust parameter")
- Initialize $Q(s, a) = 0\ \forall\ s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - With probability ϵ, take the greedy action
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\ Q(s, a')$$
    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s, a)$
  - Update the state: $s \leftarrow s'$ where $s' \sim p(s' \mid s, a)$
  - Update $Q(s, a)$:

$$Q(s,a) \leftarrow \underbrace{Q(s,a)}_{\text{Current value}} + \alpha \left(\overbrace{\underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{Temporal difference target}} - Q(s,a)}^{\text{Temporal difference}} \right)$$

Q – learning: convergence

- For Algorithms 1 & 2 (deterministic transitions), Q converges to Q^* if
  1. Every valid state-action pair is visited infinitely often
     - Q-learning is exploration-insensitive: any visitation strategy that satisfies this property will work!
  2. $0 \leq \gamma < 1$
  3. $\exists\, \beta$ s.t. $|R(s, a)| < \beta \;\forall\, s \in \mathcal{S}, a \in \mathcal{A}$
  4. Initial Q values are finite

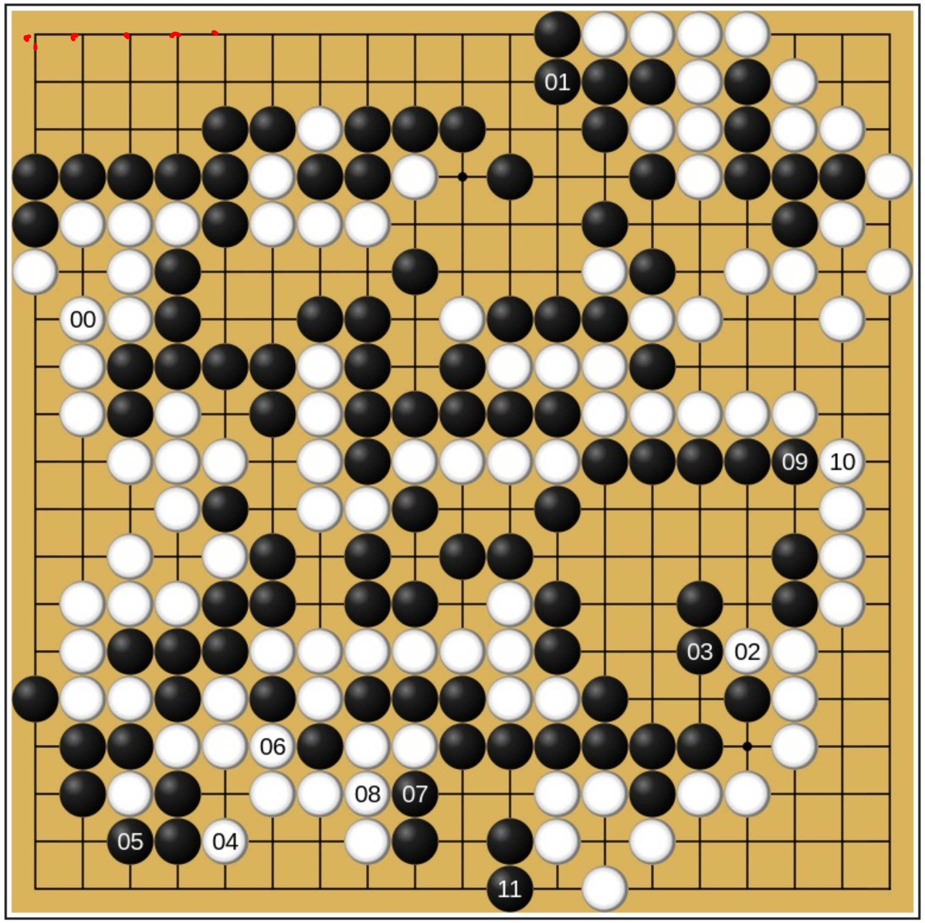
Q – learning: convergence

- For Algorithm 3 (temporal difference learning), Q converges to Q^* if
  1. Every valid state-action pair is visited infinitely often
     - Q-learning is exploration-insensitive: any visitation strategy that satisfies this property will work!
  2. $0 \leq \gamma < 1$
  3. $\exists\, \beta$ s.t. $|R(s, a)| < \beta\ \forall\, s \in \mathcal{S}, a \in \mathcal{A}$
  4. Initial Q values are finite
  5. Learning rate α_t follows some "schedule" s.t. $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$ e.g., $\alpha_t = {}^{1}/_{t+1}$

Deep Q-learning

- What if state-action spaces are continuous?
- Use a parametric function, $Q(s, a; \Theta)$, to approximate $Q^*(s, a)$
  - Learn the parameters using SGD
  - Training data $(\boldsymbol{s}_t, a_t, r_t, \boldsymbol{s}_{t+1})$ gathered online by the agent/learning algorithm
- If the approximator is a deep neural network => deep Q-learning

AlphaGo (Black) vs. Lee Sedol (White)
Game 2 final position (AlphaGo wins)



Playing Go

19-by-19 board

Players alternate placing black and white stones

The goal is claim more territory than the opponent

There are ~10^{170} legal Go board states!

$O(10^{170}, -)$

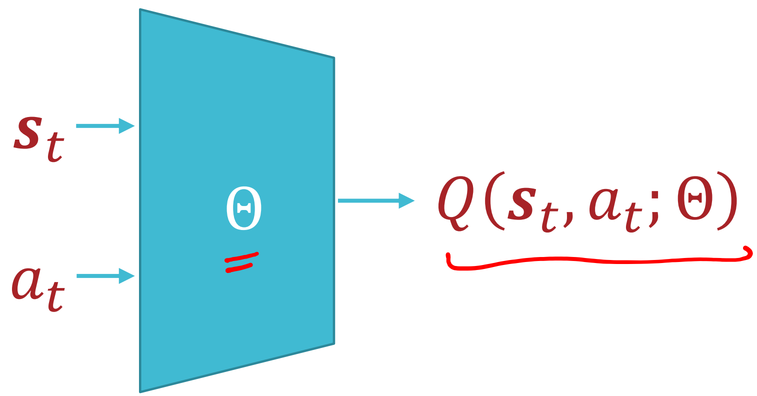
Source: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol

Source: https://en.wikipedia.org/wiki/Go_and_mathematics

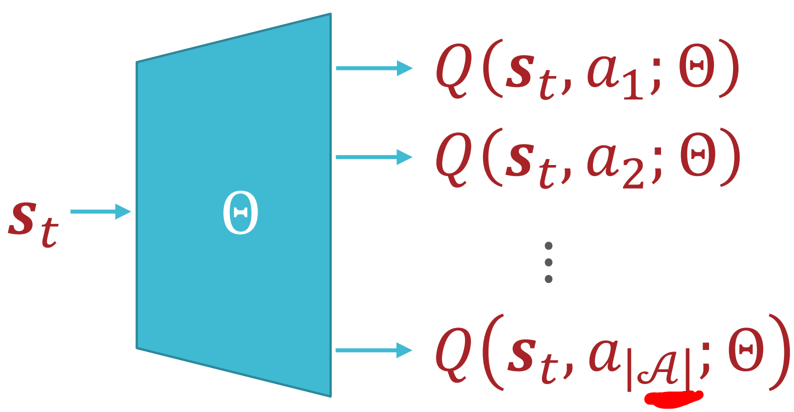
Deep Q-learning: Model

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$
  e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a neural network architecture

Model 1:



Model 2:



|A| finito

Deep Q-learning: Loss function

$\sum_{i=1}^{N} (y_i - f(x_i; w))^2$

- "True" loss

2. Don't know Q^*

$$\ell(\Theta) = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \left(Q^*(s,a) - Q(s,a;\Theta) \right)^2$$

1. $\mathcal{S}$ too big to compute this sum

1. Use stochastic gradient descent: just consider one state-action pair in each iteration
2. Use temporal difference learning:
   - Given current parameters $\Theta^{(t)}$ the temporal difference target is

$$Q^*(s,a) \approx r + \gamma \max_{a'} Q\left(s', a'; \Theta^{(t)}\right) := y$$

   - Set the parameters in the next iteration $\Theta^{(t+1)}$ such that $Q\left(s,a;\Theta^{(t+1)}\right) \approx y$

$$\ell\left(\Theta^{(t)}, \Theta^{(t+1)}\right) = \left(y - Q\left(s,a;\Theta^{(t+1)}\right)\right)^2$$

Deep Q-learning: parametric online learning

- Inputs: discount factor γ, an initial state s_0,

  learning rate α

- Initialize parameters $\Theta^{(0)}$
- For $t = 0, 1, 2, \ \dots$
  - Gather training sample $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$, compute y
  - Update $\Theta^{(t)}$ by taking a step opposite the gradient

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \alpha \nabla_{\Theta^{(t+1)}} \ell\left(\Theta^{(t)}, \Theta^{(t+1)}\right)$$

where

$$\nabla_{\Theta^{(t+1)}} \ell\left(\Theta^{(t)}, \Theta^{(t+1)}\right)$$
$$= 2\left(y - Q\left(s, a; \Theta^{(t+1)}\right)\right) \nabla_{\Theta^{(t+1)}} Q\left(s, a; \Theta^{(t+1)}\right)$$

Deep Q-learning: Experience replay

- Issue: SGD assumes i.i.d. training samples but in RL, samples are *highly* correlated
- Idea: keep a "replay memory" $\mathcal{D} = \{e_1, e_2, \dots, e_N\}$ of the N most recent experiences $e_t = (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ (Lin, 1992)
  - Also keeps the agent from "forgetting" about recent experiences
- Alternate between:
  1. Sampling some e_i uniformly at random from $\mathcal{D}$ and applying a Q-learning update (repeat T times)
  2. Adding a new experience to $\mathcal{D}$
- Can also sample experiences from $\mathcal{D}$ according to some distribution that prioritizes experiences with high error (Schaul et al., 2016)

RL summary

- States, actions, rewards
- Policy
- Value function, Q function
- Finding optimal policy:
  - value iteration
  - policy iteration
- Unknown reward and transition function:
  - Q learning (including temporal difference)
- Continuous states and actions:
  - parametric models, deep Q learning
  - Experience replay