Q-learning Tutorial

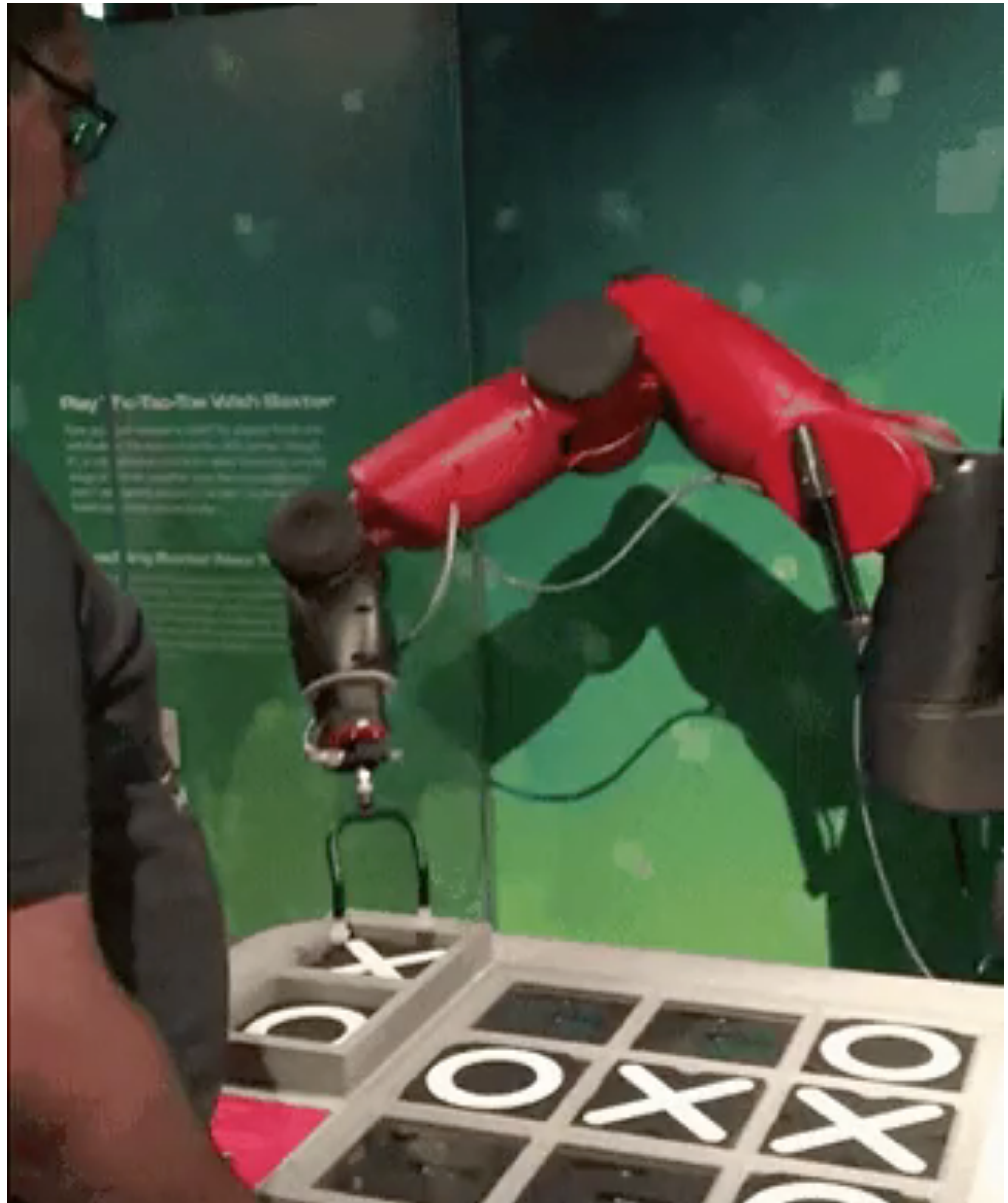
CSC411
Geoffrey Roeder

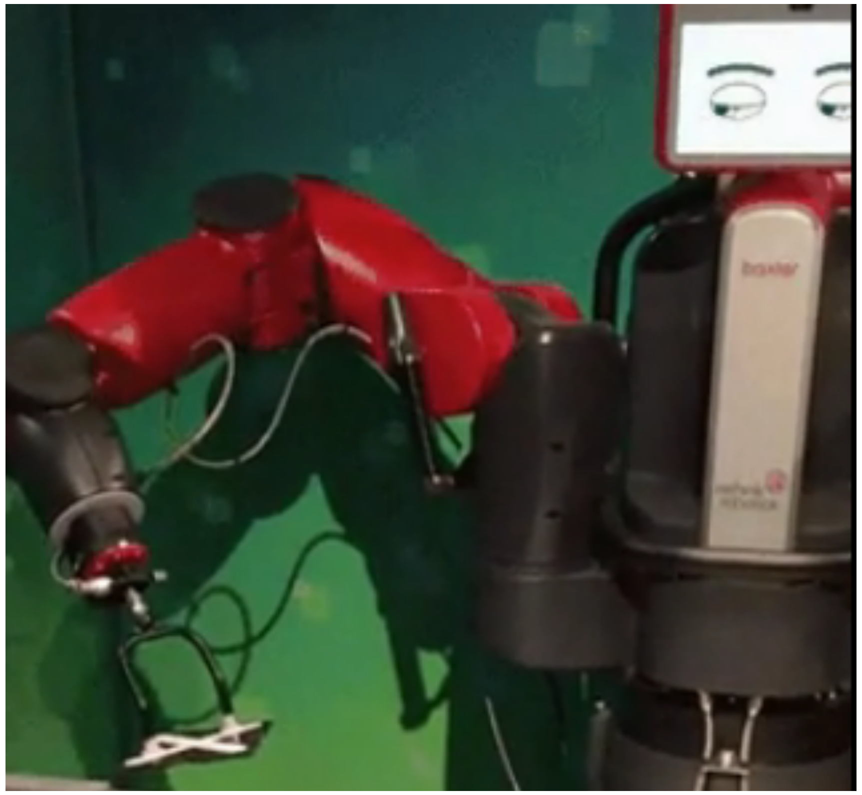
Slides Adapted from lecture: Rich Zemel, Raquel Urtasun, Sanja Fidler, Nitish Srivastava

Tutorial Agenda

- Refresh RL terminology through Tic Tac Toe
- Deterministic Q-Learning: what and how
- Q-learning Matlab demo: Gridworld
- Extensions: non-deterministic reward, next state
- More cool demos

Tic Tac Toe Redux





Tic Tac Toe Redux

$R =$

	Lose	Tie	Win
Reward	-1	0	+1

$s_t =$

	x	o
x		o

$$\pi : S \to A$$

$\pi\Big($

	x	o
x		o

$\Big) \mapsto a$

$$\mathcal{V}^{\pi} : S \to R$$

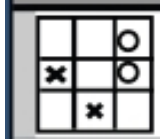
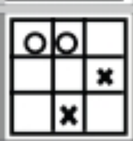
$\mathcal{V}^{\pi}($

	x	o
x		o

$) \mapsto r_{\text{future}}$

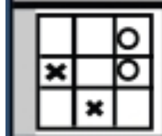
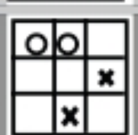
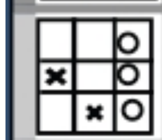
RL & Tic-Tac-Toe

- Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

RL & Tic-Tac-Toe

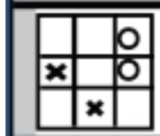
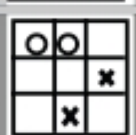
- Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

- Simple learning process:

RL & Tic-Tac-Toe

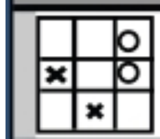
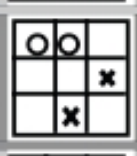
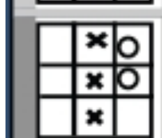
- Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

- Simple learning process:
  - start with all values = 0.5

RL & Tic-Tac-Toe

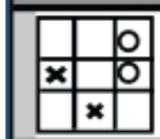
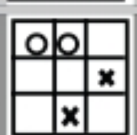
- Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

- Simple learning process:
  - start with all values = 0.5
  - policy: choose move with highest probability of winning given current legal moves from current state

RL & Tic-Tac-Toe

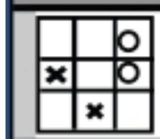
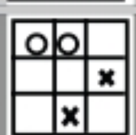
- Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

- Simple learning process:
  - start with all values = 0.5
  - policy: choose move with highest probability of winning given current legal moves from current state
  - update entries in table based on outcome of each game

RL & Tic-Tac-Toe

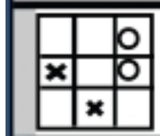
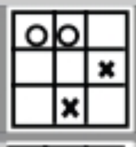
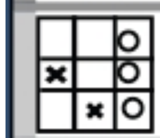
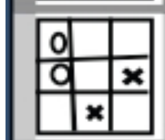
- Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

- Simple learning process:
  - start with all values = 0.5
  - policy: choose move with highest probability of winning given current legal moves from current state
  - update entries in table based on outcome of each game
  - After many games value function will represent true probability of winning from each state

RL & Tic-Tac-Toe

- Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

- Simple learning process:
  - start with all values $= 0.5$
  - policy: choose move with highest probability of winning given current legal moves from current state
  - update entries in table based on outcome of each game
  - After many games value function will represent true probability of winning from each state
- Can try alternative policy: sometimes select moves randomly (exploration)

MDP Refresher

Familiar? Skip?

MDP Formulation

- Goal: find policy π that maximizes expected accumulated future rewards $V^{\pi}(s_t)$, obtained by following π from state s_t:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

MDP Formulation

- Goal: find policy π that maximizes expected accumulated future rewards $V^\pi(s_t)$, obtained by following π from state s_t:

$$\begin{aligned} V^\pi(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned}$$

- Game show example:

MDP Formulation

- Goal: find policy π that maximizes expected accumulated future rewards $V^\pi(s_t)$, obtained by following π from state s_t:

$$\begin{aligned} V^\pi(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned}$$

- Game show example:
  - assume series of questions, increasingly difficult, but increasing payoff

MDP Formulation

- Goal: find policy π that maximizes expected accumulated future rewards $V^{\pi}(s_t)$, obtained by following π from state s_t:

$$\begin{aligned} V^{\pi}(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned}$$

- Game show example:
  - assume series of questions, increasingly difficult, but increasing payoff
  - choice: accept accumulated earnings and quit; or continue and risk losing everything
- Notice that:

$$V^{\pi}(s_t) = r_t + \gamma V^{\pi}(s_{t+1})$$

What to Learn

- We might try to learn the function V (which we write as V^*)

$$V^*(s) = \max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

- Here $\delta(s,a)$ gives the next state, if we perform action a in current state s

What to Learn

- We might try to learn the function V (which we write as V^*)

$$V^*(s) = \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- Here $\delta(s, a)$ gives the next state, if we perform action a in current state s
- We could then do a lookahead search to choose best action from any state s:

$$\pi^*(s) = arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

What to Learn

- We might try to learn the function V (which we write as V^*)

$$V^*(s) = \max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

- Here $\delta(s,a)$ gives the next state, if we perform action a in current state s
- We could then do a lookahead search to choose best action from any state s:

$$\pi^*(s) = arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

- But there's a problem:

What to Learn

- We might try to learn the function V (which we write as V^*)

$$V^*(s) = \max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

- Here $\delta(s,a)$ gives the next state, if we perform action a in current state s
- We could then do a lookahead search to choose best action from any state s:

$$\pi^*(s) = arg \max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

- But there's a problem:
  - This works well if we know $\delta()$ and $r()$

What to Learn

- We might try to learn the function V (which we write as V^*)

$$V^*(s) = \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- Here $\delta(s, a)$ gives the next state, if we perform action a in current state s
- We could then do a lookahead search to choose best action from any state s:

$$\pi^*(s) = arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- But there's a problem:
  - This works well if we know $\delta()$ and $r()$
  - But when we don't, we cannot choose actions this way

Q Learning

Deterministic rewards and actions

Q Learning

- Define a new function very similar to V^*

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

Q Learning

- Define a new function very similar to V^*

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

- If we learn Q, we can choose the optimal action even without knowing δ!

$$\pi^*(s) \quad = \quad arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

Q Learning

- Define a new function very similar to V^*

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

- If we learn Q, we can choose the optimal action even without knowing δ!

$$\begin{aligned}\pi^*(s) &= arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))] \\ &= arg \max_a Q(s, a)\end{aligned}$$

Q Learning

- Define a new function very similar to V^*

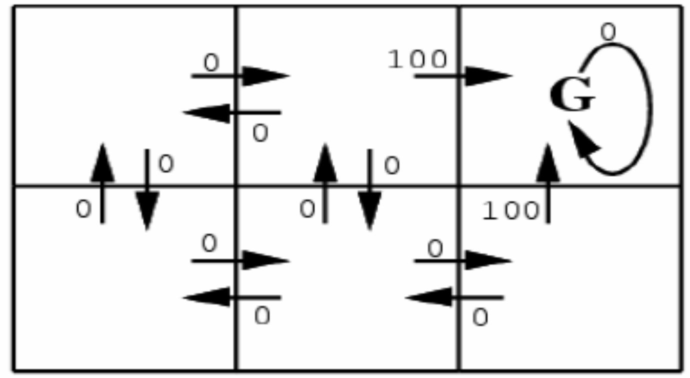
$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$

- If we learn Q, we can choose the optimal action even without knowing δ!

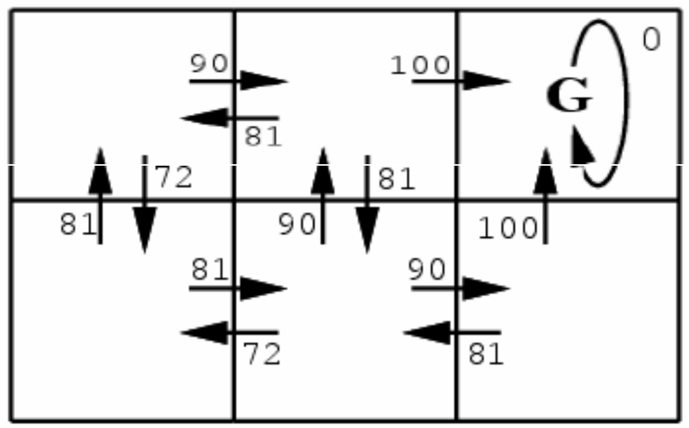
$$\begin{aligned}\pi^*(s) &= arg \max_a [r(s,a) + \gamma V^*(\delta(s,a))] \\ &= arg \max_a Q(s,a)\end{aligned}$$

- Q is then the evaluation function we will learn

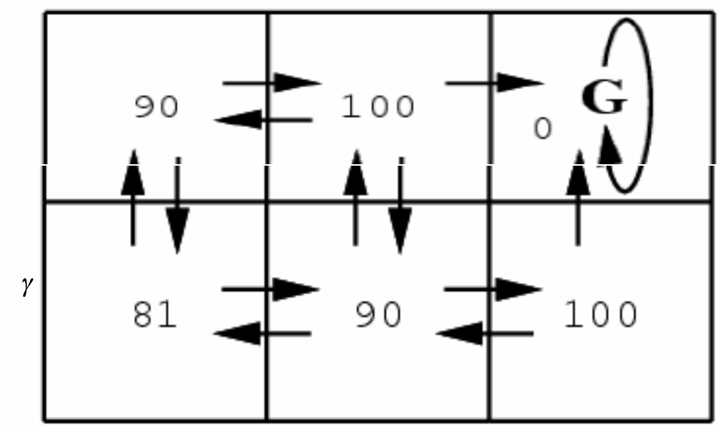
$\gamma = 0.9$



$r(s,a)$ (immediate reward) values

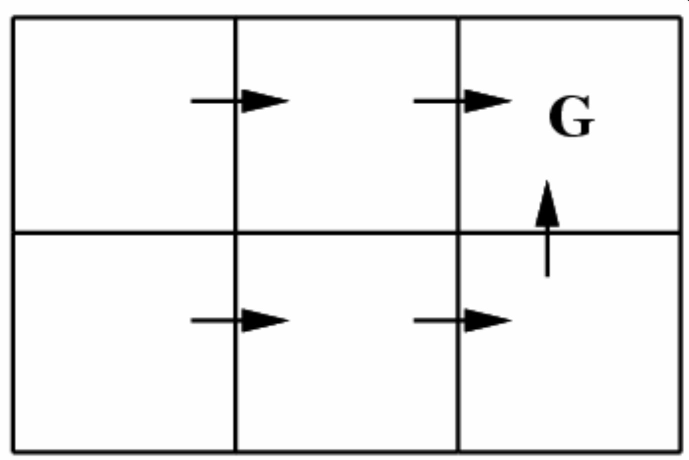


$Q(s,a)$ values



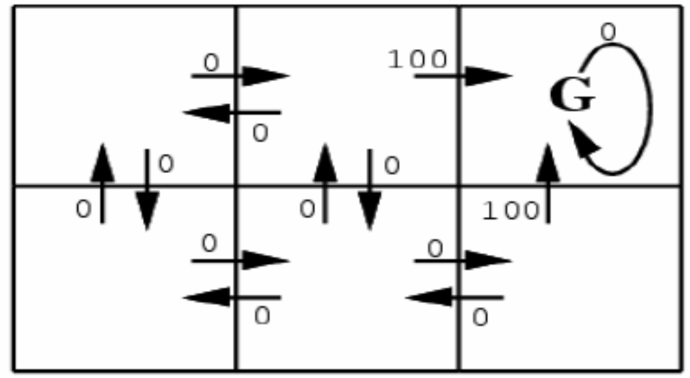
$V^*(s)$ values

$$V^*(s_5) = 0 + \gamma 100 + \gamma^2 0 + \ldots = 90$$

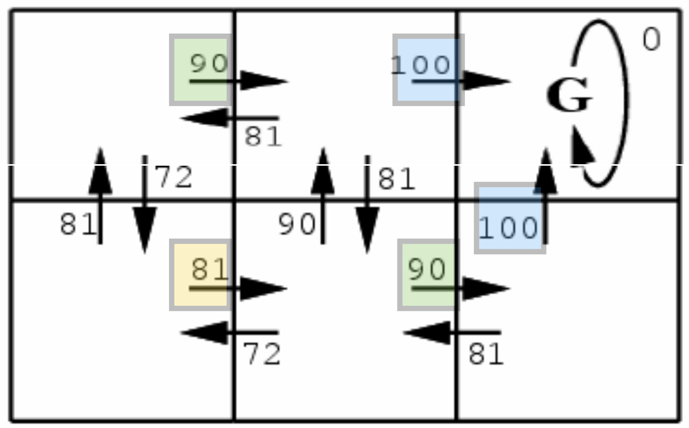


One optimal policy

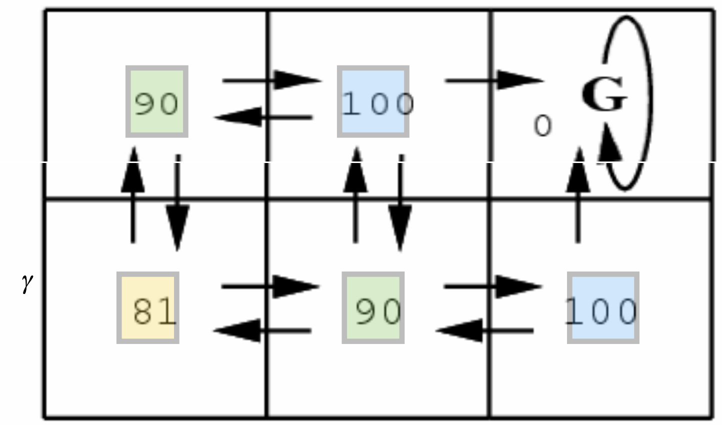
$\gamma = 0.9$



$r(s,a)$ (immediate reward) values

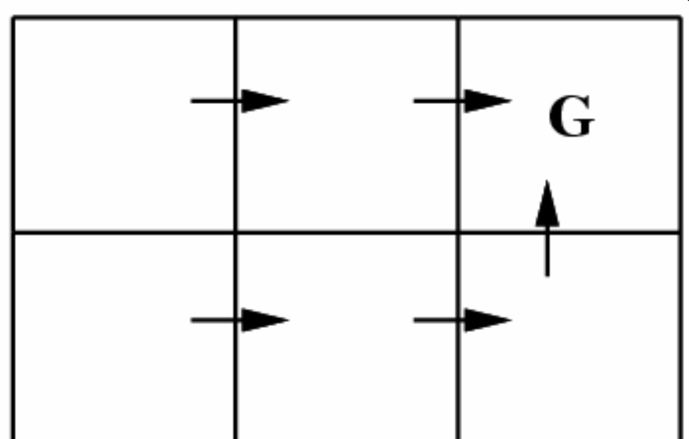


$Q(s,a)$ values



$V^*(s)$ values

$$V^*(s_5) = 0 + \gamma 100 + \gamma^2 0 + \ldots = 90$$



One optimal policy

Training Rule to Learn Q

- Q and V^* are closely related:

$$V^*(s) = \max_a Q(s, a)$$

Training Rule to Learn Q

- Q and V^* are closely related:

$$V^*(s) = \max_a Q(s, a)$$

- So we can write Q recursively:

$$Q(s_t, a_t) \quad = \quad r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t))$$

Training Rule to Learn Q

- Q and V^* are closely related:

$$V^*(s) = \max_a Q(s, a)$$

- So we can write Q recursively:

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

Training Rule to Learn Q

- Q and V^* are closely related:

$$V^*(s) = \max_a Q(s,a)$$

- So we can write Q recursively:

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

- Let $\hat{Q}$ denote the learner's current approximation to Q

Training Rule to Learn Q

- Q and V^* are closely related:

$$V^*(s) = \max_a Q(s, a)$$

- So we can write Q recursively:

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

- Let $\hat{Q}$ denote the learner's current approximation to Q
- Consider training rule

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

where s' is state resulting from applying action a in state s

Q Learning for Deterministic World

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Q Learning for Deterministic World

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state s

Q Learning for Deterministic World

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state s
- Do forever:

Q Learning for Deterministic World

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state s
- Do forever:
  - Select an action a and execute it

Q Learning for Deterministic World

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state s
- Do forever:
  - Select an action a and execute it
  - Receive immediate reward r

Q Learning for Deterministic World

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state s
- Do forever:
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s'

Q Learning for Deterministic World

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state s
- Do forever:
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s'
  - Update the table entry for $\hat{Q}(s, a)$ using Q learning rule:

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

Q Learning for Deterministic World

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state s
- Do forever:
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s'
  - Update the table entry for $\hat{Q}(s, a)$ using *Q* learning rule:

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

  - $s \leftarrow s'$

Q Learning for Deterministic World

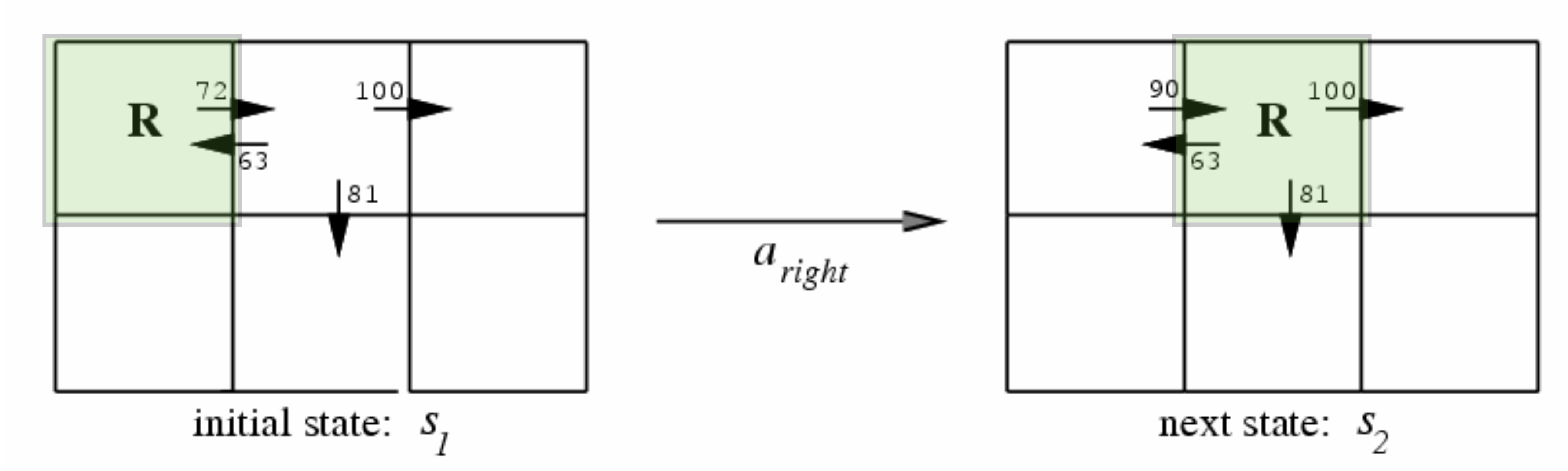
- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state s
- Do forever:
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s'
  - Update the table entry for $\hat{Q}(s, a)$ using Q learning rule:

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

  - $s \leftarrow s'$
- If we get to absorbing state, restart to initial state, and run thru "Do forever" loop until reach absorbing state

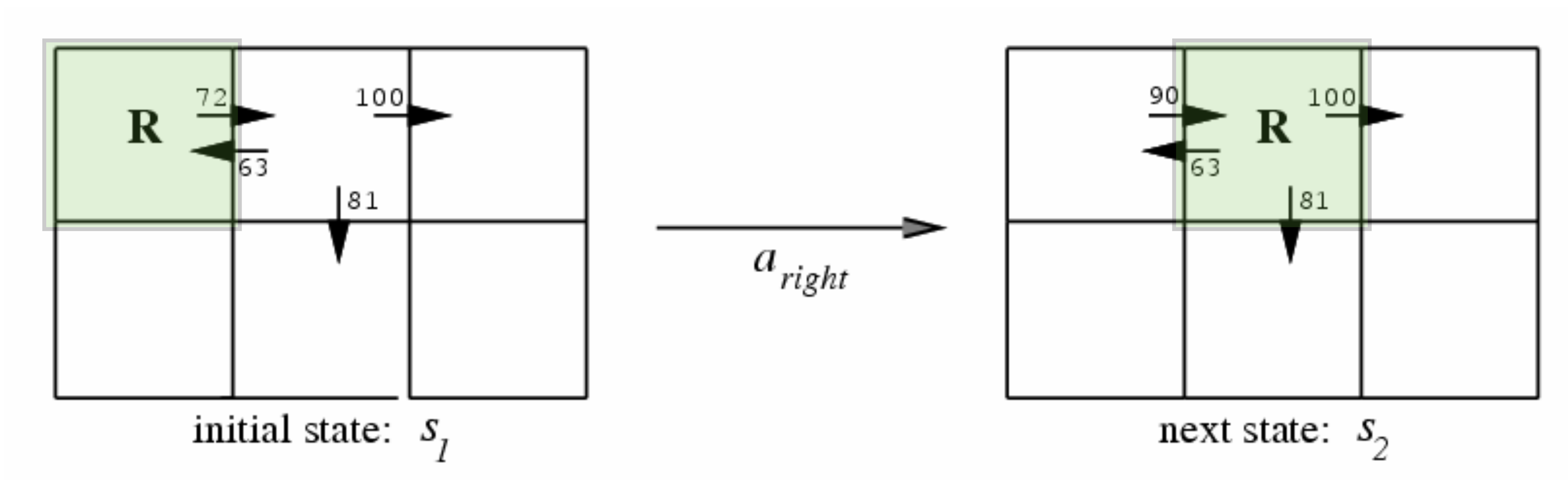
Updating Estimated Q

- Assume the robot is in state s_1; some of its current estimates of Q are as shown; executes rightward move



Updating Estimated Q

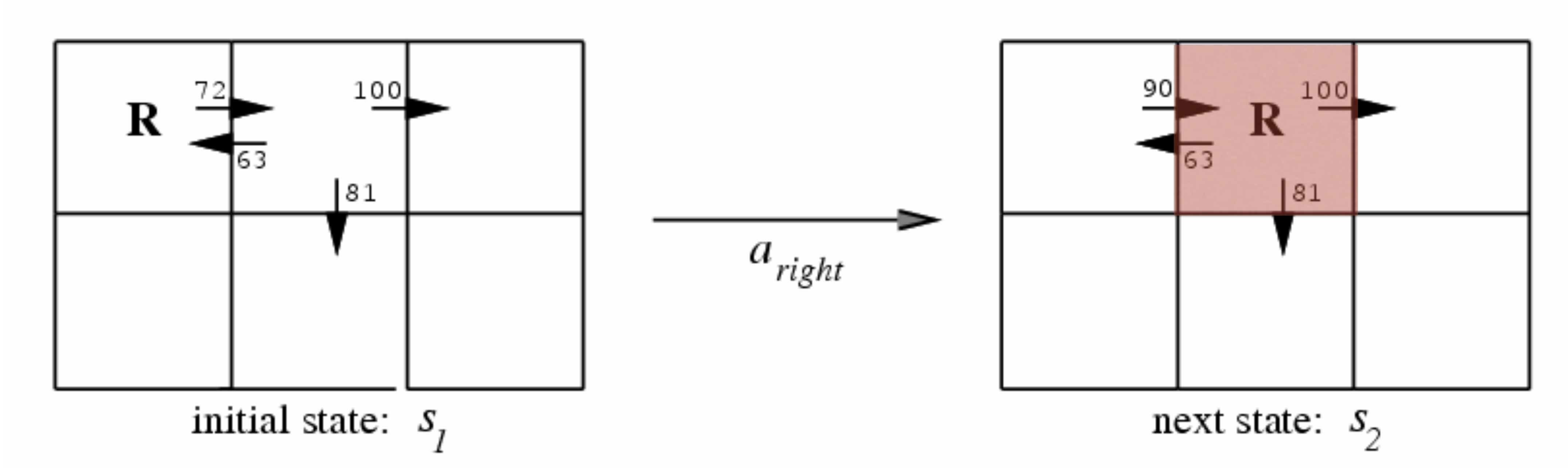
- Assume the robot is in state s_1; some of its current estimates of Q are as shown; executes rightward move



$$\hat{Q}(s_1, a_{right}) \quad \leftarrow \quad r + \gamma \max_{a'} \hat{Q}(s_2, a')$$

Updating Estimated Q

- Assume the robot is in state s_1; some of its current estimates of Q are as shown; executes rightward move

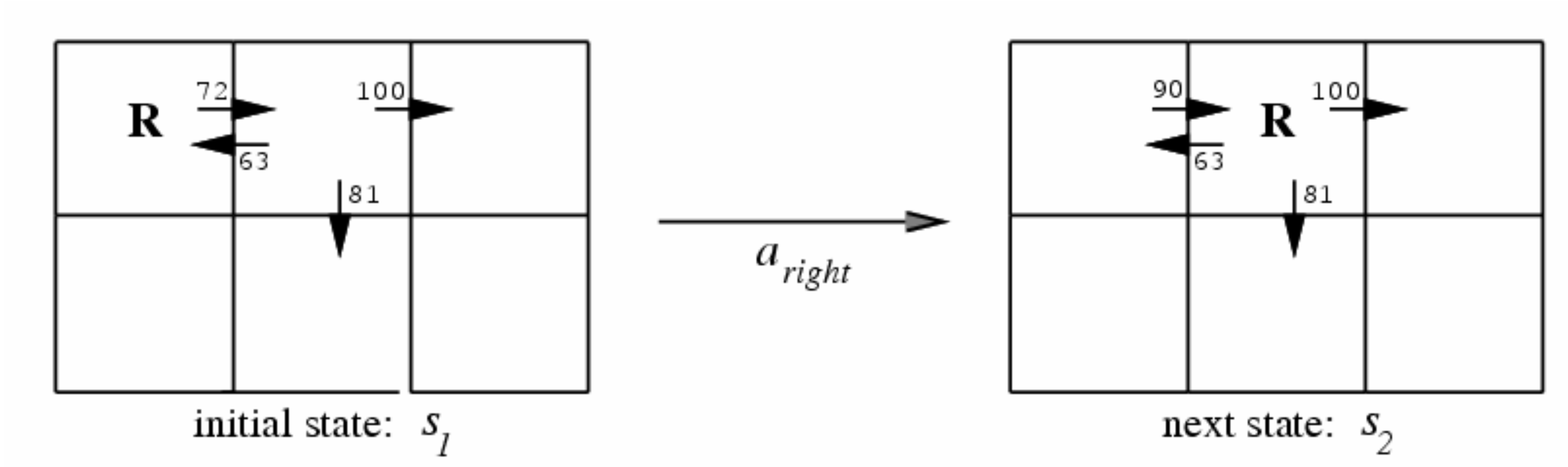


$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$

$$\leftarrow r + 0.9 \max_{a}\{63, 81, 100\} \leftarrow 90$$

Updating Estimated Q

- Assume the robot is in state s_1; some of its current estimates of Q are as shown; executes rightward move

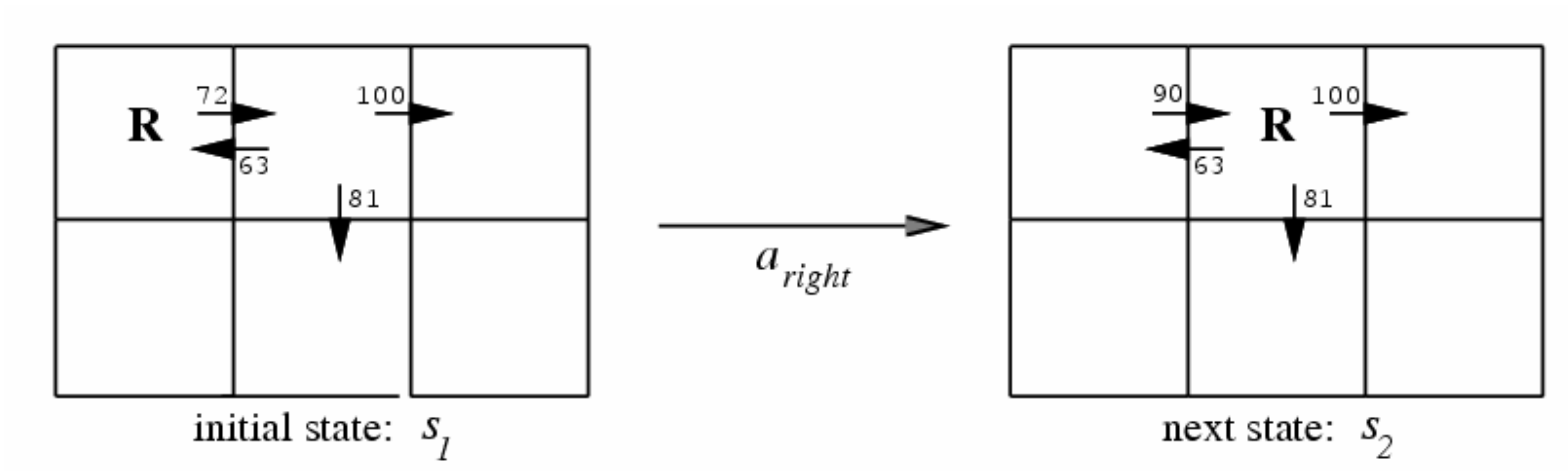


$$\begin{aligned}\hat{Q}(s_1, a_{right}) \quad &\leftarrow \quad r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow \quad r + 0.9 \max_{a} \{63, 81, 100\} \leftarrow 90\end{aligned}$$

- Important observation: at each time step (making an action a in state s **only one** entry of $\hat{Q}$ will change (the entry $\hat{Q}(s, a)$)

Updating Estimated Q

- Assume the robot is in state s_1; some of its current estimates of Q are as shown; executes rightward move



$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow r + 0.9 \max_{a}\{63, 81, 100\} \leftarrow 90\end{aligned}$$

- Important observation: at each time step (making an action a in state s **only one** entry of $\hat{Q}$ will change (the entry $\hat{Q}(s, a)$)
- Notice that if rewards are non-negative, then $\hat{Q}$ values only increase from 0, approach true Q

Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state

Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action a results in transition from state s_i to s_j; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule

Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action a results in transition from state s_i to s_j; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule
- Intuition for simple grid world, reward only upon entering goal state $\rightarrow Q$ estimates improve from goal state back

Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action a results in transition from state s_i to s_j; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule
- Intuition for simple grid world, reward only upon entering goal state $\rightarrow$ Q estimates improve from goal state back
    1. All $\hat{Q}(s, a)$ start at 0

Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action a results in transition from state s_i to s_j; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule
- Intuition for simple grid world, reward only upon entering goal state $\rightarrow Q$ estimates improve from goal state back
    1. All $\hat{Q}(s, a)$ start at 0
    2. First episode – only update $\hat{Q}(s, a)$ for transition leading to goal state

Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action a results in transition from state s_i to s_j; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule
- Intuition for simple grid world, reward only upon entering goal state $\rightarrow Q$ estimates improve from goal state back
  1. All $\hat{Q}(s, a)$ start at 0
  2. First episode – only update $\hat{Q}(s, a)$ for transition leading to goal state
  3. Next episode – if go thru this next-to-last transition, will update $\hat{Q}(s, a)$ another step back

Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action a results in transition from state s_i to s_j; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule
- Intuition for simple grid world, reward only upon entering goal state $\rightarrow Q$ estimates improve from goal state back
  1. All $\hat{Q}(s, a)$ start at 0
  2. First episode – only update $\hat{Q}(s, a)$ for transition leading to goal state
  3. Next episode – if go thru this next-to-last transition, will update $\hat{Q}(s, a)$ another step back
  4. Eventually propagate information from transitions with non-zero reward throughout state-action space

Gridworld Demo

Extensions

Non-deterministic reward and actions

Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)

Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s, a)$

Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s, a)$
- Good idea?

Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s, a)$
- Good idea?
- Can instead employ stochastic action selection (policy):

$$P(a_i|s) = \frac{\exp(k\hat{Q}(s, a_i))}{\sum_j \exp(k\hat{Q}(s, a_j))}$$

Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s, a)$
- Good idea?
- Can instead employ stochastic action selection (policy):

$$P(a_i|s) = \frac{\exp(k\hat{Q}(s, a_i))}{\sum_j \exp(k\hat{Q}(s, a_j))}$$

- Can vary k during learning

Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s, a)$
- Good idea?
- Can instead employ stochastic action selection (policy):

$$P(a_i|s) = \frac{\exp(k\hat{Q}(s, a_i))}{\sum_j \exp(k\hat{Q}(s, a_j))}$$

- Can vary k during learning
  - more exploration early on, shift towards exploitation

Non-deterministic Case

- What if reward and next state are non-deterministic?

Non-deterministic Case

- What if reward and next state are non-deterministic?
- We redefine V, Q based on probabilistic estimates, expected values of them:

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots] \\
&= E_{\pi}[\sum_{i=0}^{\infty} \gamma^i r_{t+i}]
\end{aligned}
$$

Non-deterministic Case

- What if reward and next state are non-deterministic?
- We redefine V, Q based on probabilistic estimates, expected values of them:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots] \\ &= E_{\pi}[\sum_{i=0}^{\infty} \gamma^i r_{t+i}] \end{aligned}$$

and

$$\begin{aligned} Q(s,a) &= E[r(s,a) + \gamma V^*(\delta(s,a))] \\ &= E[r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q(s',a')] \end{aligned}$$

Non-deterministic Case: Learning Q

- Training rule does not converge (can keep changing $\hat{Q}$ even if initialized to true Q values)

Non-deterministic Case: Learning Q

- Training rule does not converge (can keep changing $\hat{Q}$ even if initialized to true Q values)
- So modify training rule to change more slowly

$$\hat{Q}(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')]$$

where s' is the state land in after s, and a' indexes the actions that can be taken in state s'

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s,a)}$$

where visits is the number of times action a is taken in state s

More Cool Demos

Other Examples:

Super Mario World

https://www.youtube.com/watch?v=L4KBBAwF_bE

Model-based RL: Pole Balancing

https://www.youtube.com/watch?v=XiigTGKZfks

Learn how to fly a Helicopter

- http://heli.stanford.edu/
- Formulate as an RL problem
    - State - Position, orientation, velocity, angular velocity
    - Actions - Front-back pitch, left-right pitch, tail rotor pitch, blade angle
    - Dynamics - Map actions to states. Difficult!
    - Rewards - Don’t crash, Do interesting things.

Slide credit: Nitish Srivastava