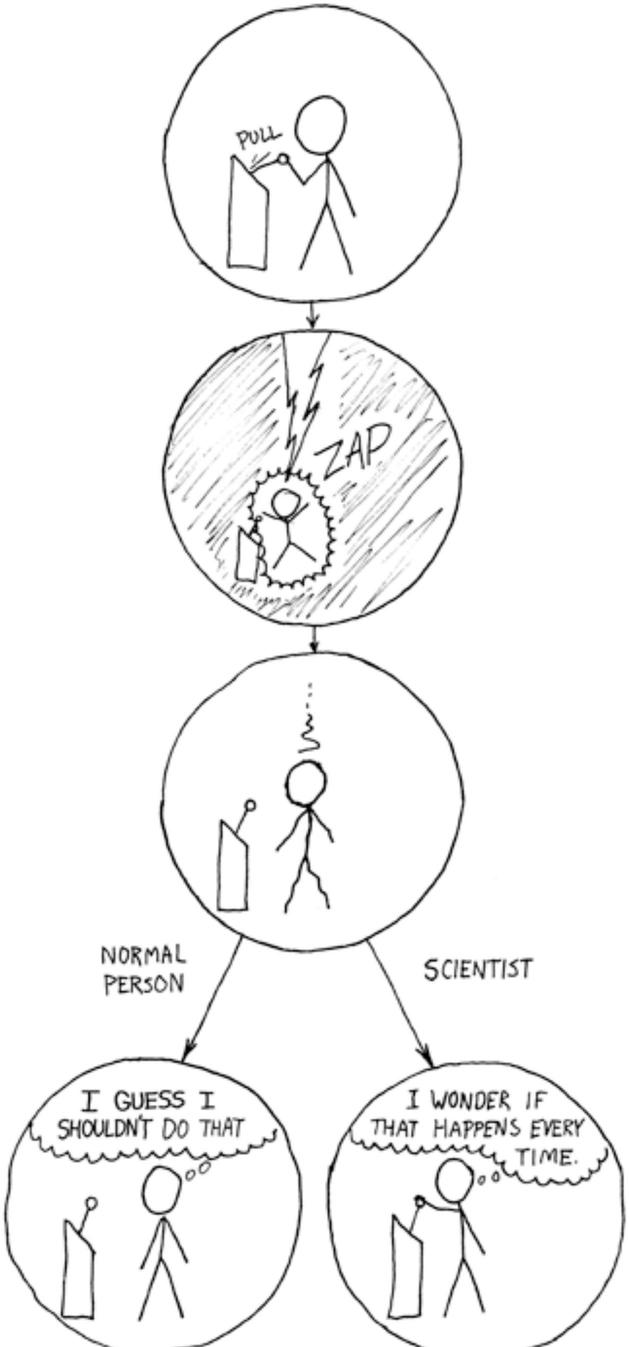


Reinforcement Learning

Slides by Bryan Pardo;
thanks in part to Bill Smart at WUSTL

Learning Types

- Supervised learning:
  - (Input, output) pairs of the function to be learned can be perceived or are given.

    Regression or classification

- Unsupervised Learning:
  - No information about desired outcomes given

    Clustering or expectation-maximization

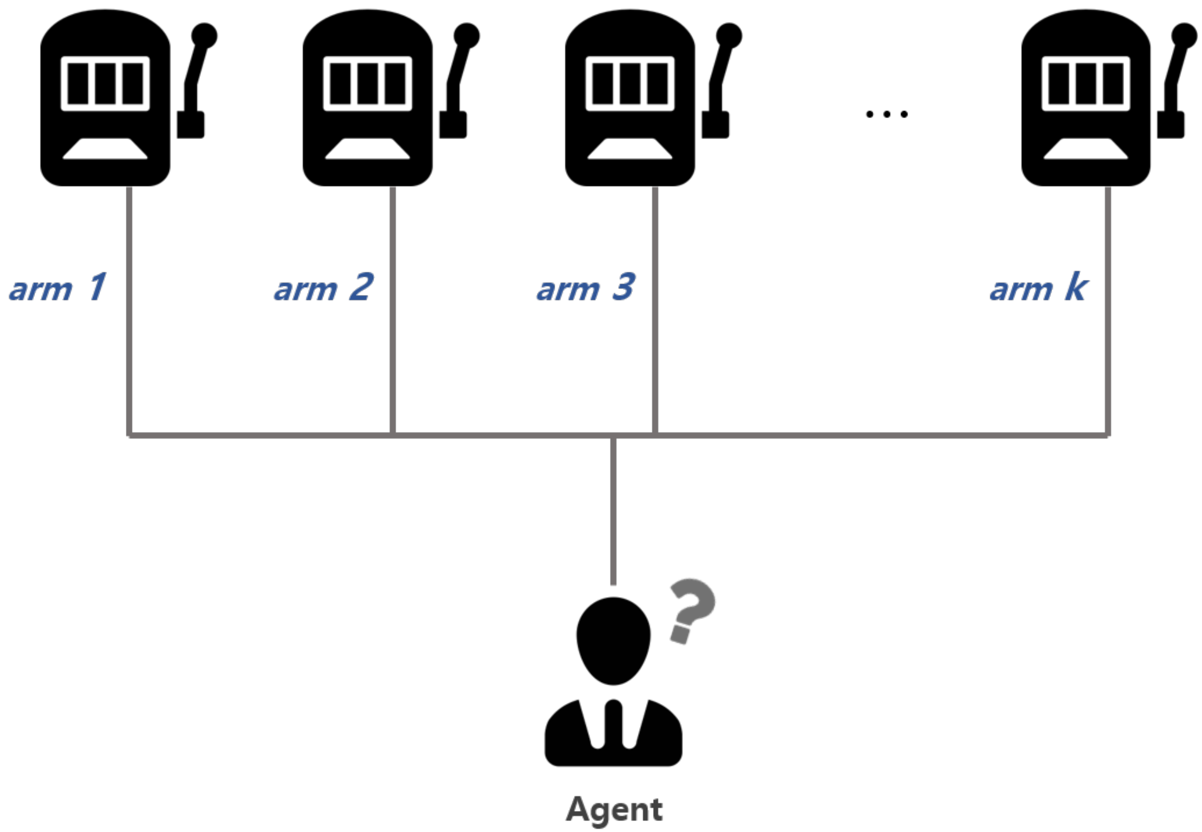
- Reinforcement learning:
  - Reward or punishment for actions

    Q-Learning

Reinforcement Learning

- Task
  - Learn how to behave to achieve a goal
  - Learn through experience from trial and error
- Examples
  - Game playing: The agent knows when it wins, but doesn’t know the appropriate action in each state along the way
  - Control: a traffic system can measure the delay of cars, but not know how to decrease it.

The Multi Armed Bandit Problem



Which slot machine do I play?

image from https://velog.io/@taejinjeong/Reinforcement-Learning-Multi-Armed-Bandit-Problem

Multi-Armed Bandits

- What if we can’t observe the current state, or we assume there is only one state?

- Common examples:
  - Bidding for advertisement space on websites
  - Price setting in a grocery store
  - Playing slot machines

Multi-Armed Bandits

- The action value Q(*a*) is the expected reward when we take action *a*.
- Say we take action *a* N times, and observe rewards $r_1, r_2, \dots r_N$.

$$\begin{aligned} Q_{N+1}(a) &= E[r|a] \\ &\approx \frac{1}{N}\sum_{i=1}^{N} r_i \\ &= Q_N(a) + \frac{1}{N}[r_N - Q_N(a)] \end{aligned}$$

- Update based on the difference between expected and observed rewards

Picking Actions

- There are two common approaches.

- Greedy

  Pick the action *a* with the highest current Q(*a*) estimate.

- ε-greedy

  Pick the best action with with probability $1 - \varepsilon$

  Else, pick the action randomly with equal probability

Multi-Armed Bandits

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$$Q(a) \leftarrow 0$$
$$N(a) \leftarrow 0$$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$
$$R \leftarrow bandit(A)$$
$$N(A) \leftarrow N(A) + 1$$
$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$$

Image from Sutton & Barto book

Example multi-armed bandit rewards

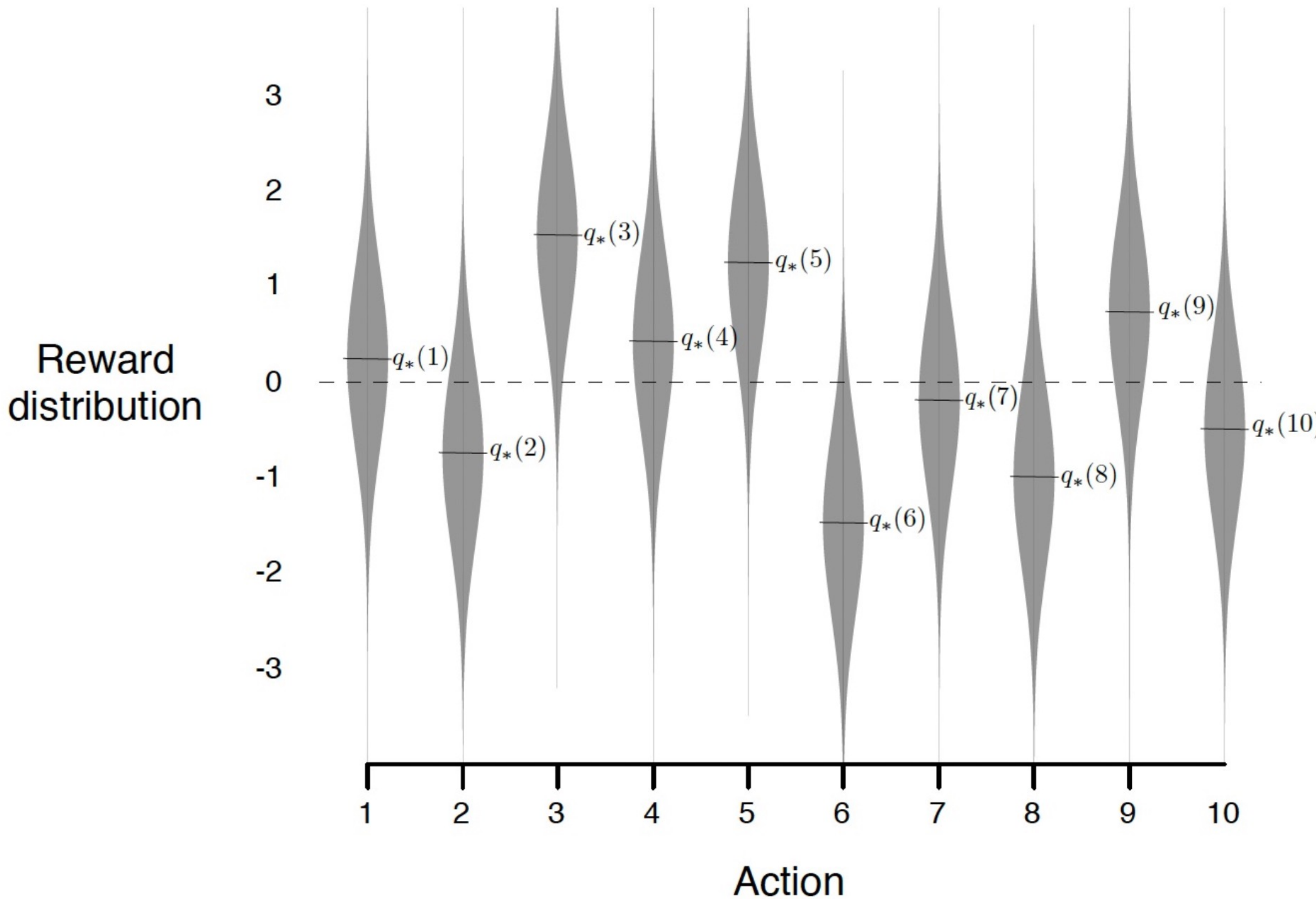
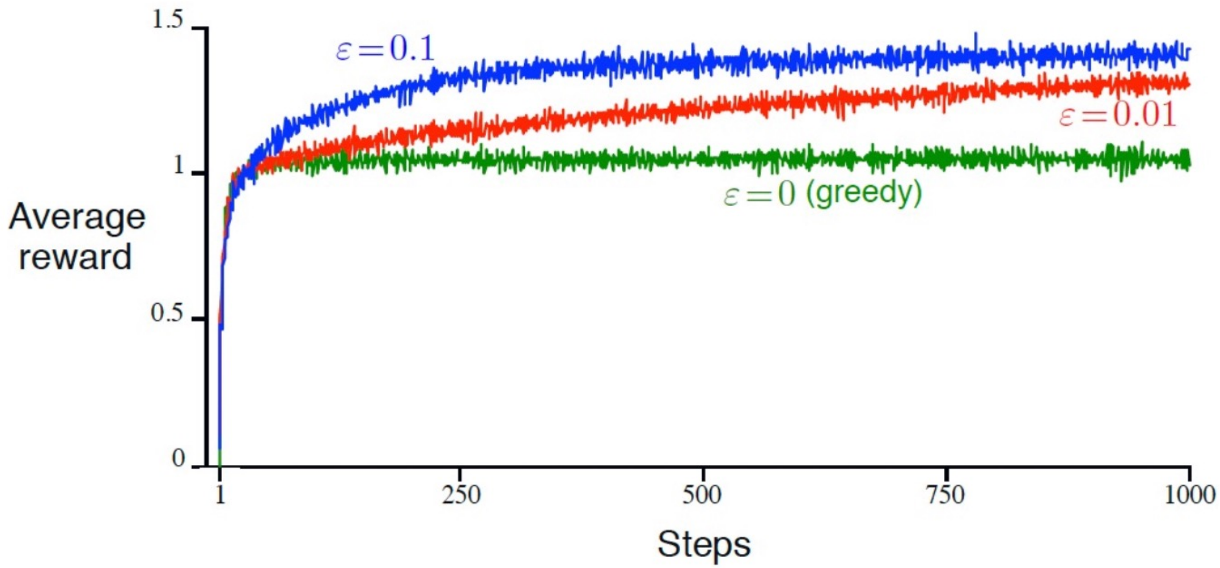


Image from Reinforcement Learning: An Introduction 2nd Ed by Sutton & Barto

Greedy vs ε−Greedy



Assumes a stationary world

This update rule:

$$Q_{N+1}(a) = Q_N(a) + \frac{1}{N}[r_N - Q_N(a)]$$

…assumes a stationary world, where the rewards never change.

What if things change over time?

A new update rule

This update rule:

$$Q_{N+1}(a) = Q_N(a) + \alpha[r_N - Q_N(a)]$$

…assumes a world where change can happen. Let's rearrange the terms….

$$\begin{aligned} Q_{N+1}(a) &= Q_N(a) + \alpha[r_N - Q_N(a)] \\ &= Q_N(a) + \alpha r_N - Q_N(a) \\ &= (1-\alpha)Q_N(a) + \alpha r_N \end{aligned}$$

Now, it should be clear we're balancing our existing knowledge Q(a) vs our new information *r.*

Non-stationary Multi-armed Bandit

A simple bandit algorithm

Initialize, for $a = 1$ to k:
$\quad Q(a) \leftarrow 0$

Loop forever:
$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$
$\quad R \leftarrow bandit(A)$

$$Q_{N+1}(a) = Q_N(a) + \alpha[r_N - Q_N(a)]$$

Note: this formulation is from Sutton & Barto's "Reinforcement Learning"
See equation 2.5 on page 32.

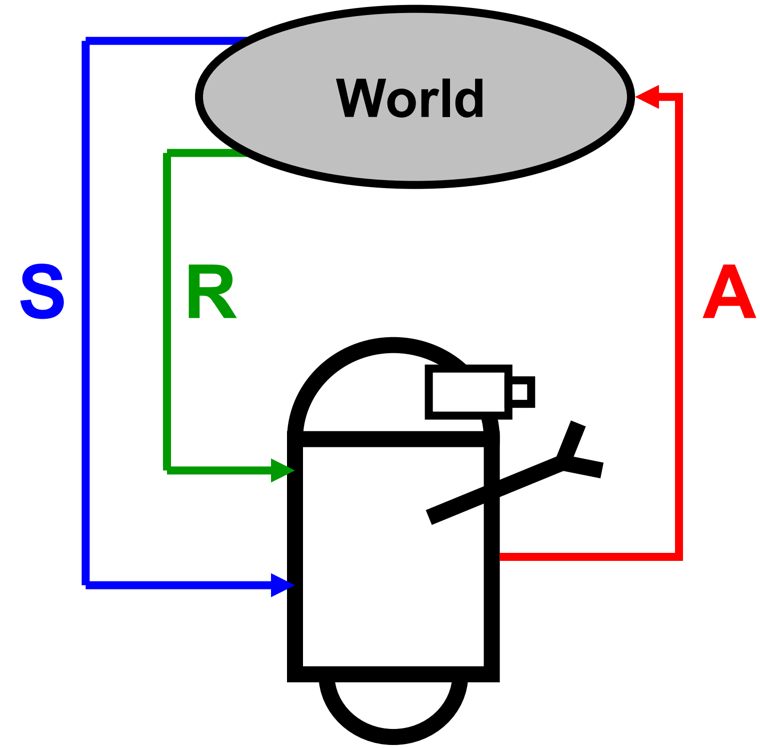
Actions have consequences

- What if taking an action changes the state of the world?

- This is the full reinforcement learning problem.

Basic RL Model

1. Observe state, s_t
2. Decide on an action, a_t
3. Perform action
4. Observe new state, s_{t+1}
5. Observe reward, r_{t+1}
6. Learn from experience
7. Repeat



•Goal: Find a control policy that will maximize the observed rewards over the lifetime of the agent

An Example: Gridworld

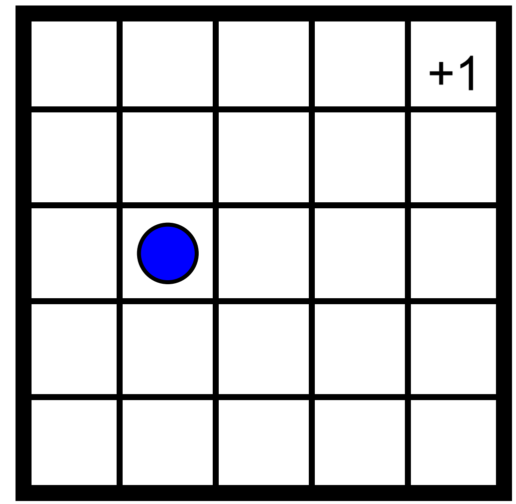
- Canonical RL domain

  States are grid cells

  4 actions: N, S, E, W

  Reward for entering top right cell

  -0.01 for every other move

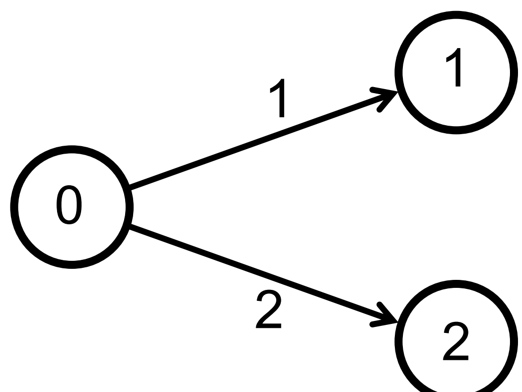


Mathematics of RL

- Before we talk about RL, we need to cover some background material
  - Simple decision theory
  - Markov Decision Processes
  - Value functions
  - Dynamic programming

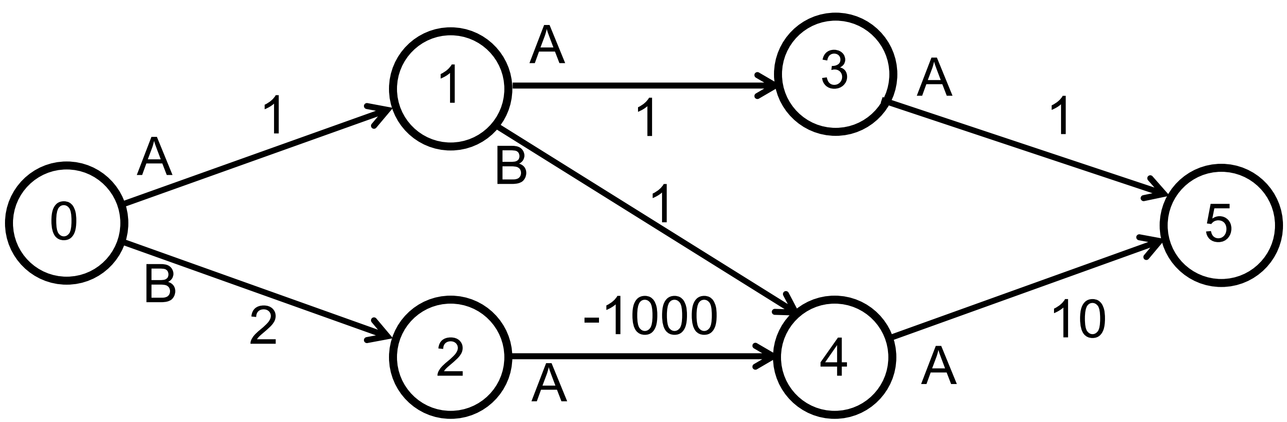
Making Single Decisions

- Single decision to be made
  - Multiple discrete actions
  - Each action has an associated reward
- Goal is to maximize reward
  - Just pick the action with the largest reward
- State 0 has a value of 2
  - Reward from taking the best action



Markov Decision Processes

- We can generalize the previous example to multiple sequential decisions
  - Each decision affects subsequent decisions
- This is formally modeled by a Markov Decision Process (MDP)



Markov Decision Processes

- Formally, a MDP is
  - A set of states, $S = \{s_1, s_2, \ldots, s_n\}$
  - A set of actions, $A = \{a_1, a_2, \ldots, a_m\}$
  - A reward function, $R: S \times A \times S \rightarrow \Re$
  - A transition function, $P_{ij}^{a} = P(s_{t+1} = j \mid s_t = i, a_t = a)$
    - Sometimes $T: S \times A \rightarrow S$
- We want to learn a policy, $\pi: S \rightarrow A$
  - Maximize sum of rewards we see over our lifetime

Policies

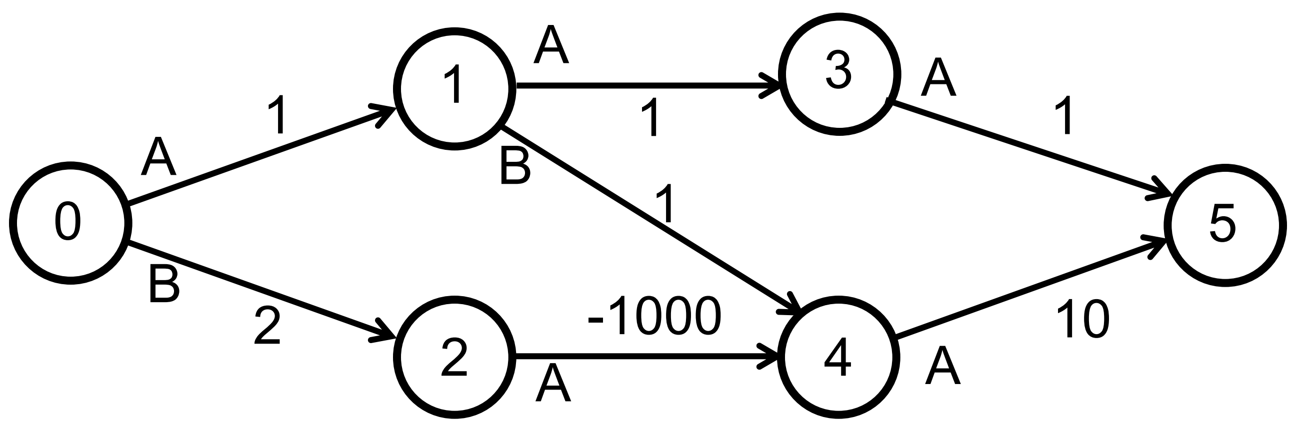
- A policy $\pi(s)$ returns the action to take in state s.

- There are 3 policies for this MDP

Policy 1: $0 \rightarrow 1 \rightarrow 3 \rightarrow 5$

Policy 2: $0 \rightarrow 1 \rightarrow 4 \rightarrow 5$

Policy 3: $0 \rightarrow 2 \rightarrow 4 \rightarrow 5$



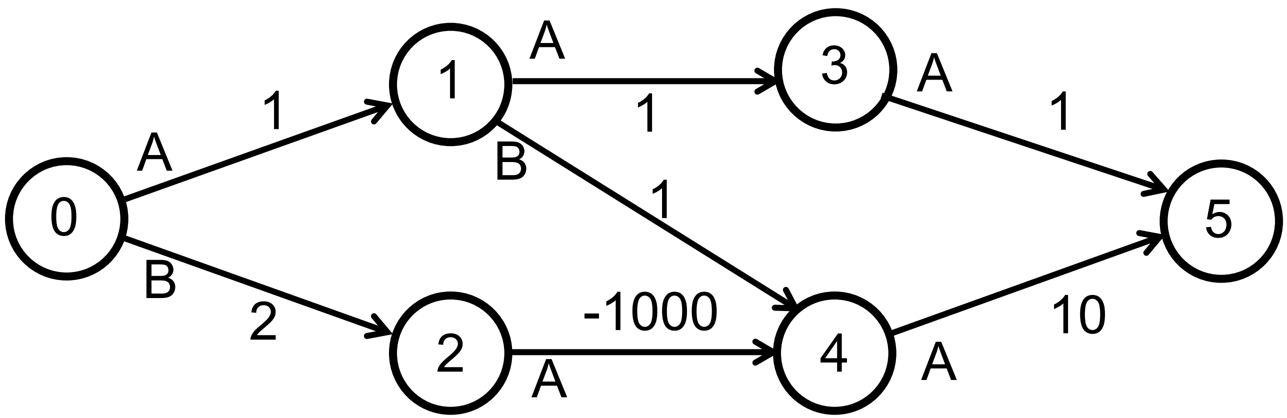
Comparing Policies

- Which policy is best?
- Order them by how much reward they see

Policy 1: $0 \rightarrow 1 \rightarrow 3 \rightarrow 5 = 1 + 1 + 1 = 3$

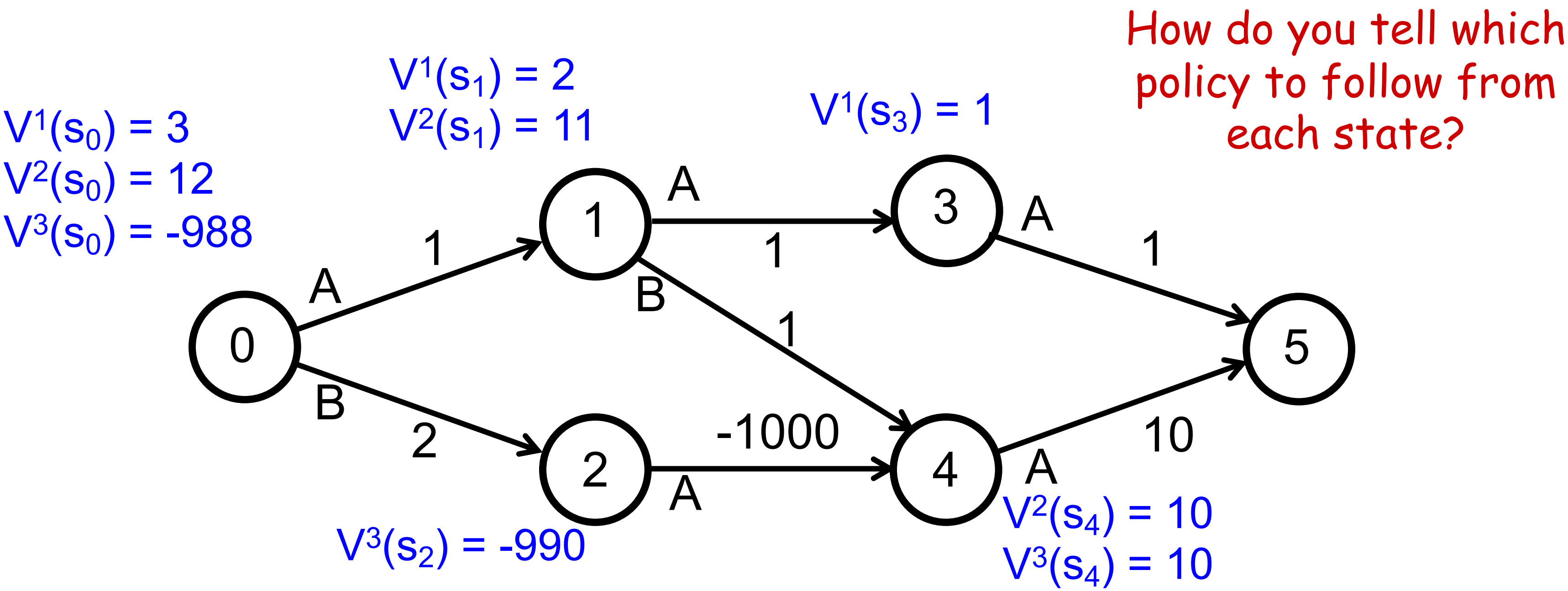
Policy 2: $0 \rightarrow 1 \rightarrow 4 \rightarrow 5 = 1 + 1 + 10 = 12$

Policy 3: $0 \rightarrow 2 \rightarrow 4 \rightarrow 5 = 2 - 1000 + 10 = -988$



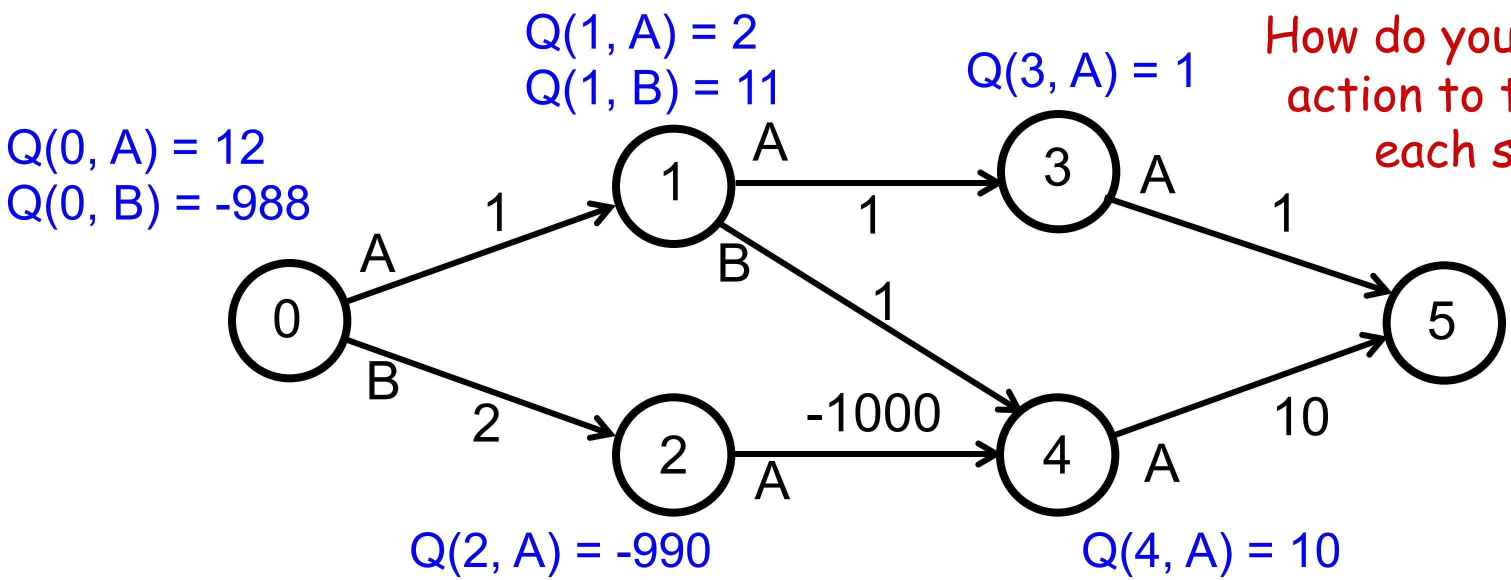
Value Functions

- We can associate a value with each state
  - For a fixed policy
  - How good is it to run policy π from that state s
  - This is the state value function, V



Q Functions

- Define value without specifying the policy
  - Specify the value of taking action A from state S and then performing optimally, thereafter



How do you tell which action to take from each state?

Reinforcement Learning

- What happens if we don' t have the whole MDP?
  - We know the states and actions
  - We don't have the system model (transition function) or reward function
- We're only allowed to sample from the MDP
  - Can observe experiences (s, a, r, s')
  - Need to perform actions to generate new experiences
- This is Reinforcement Learning (RL)
  - Sometimes called Approximate Dynamic Programming (ADP)

Exploration vs. Exploitation

- We want to pick good actions most of the time, but also do some exploration
- Exploring means we can learn better policies
- But we want to balance known good actions with exploratory ones
- This is the **exploration / exploitation** problem

Picking Actions

- ε-greedy
  – Pick best (greedy) action with probability 1 - ε
  – Otherwise, pick a random action
- Softmax function
  – Pick an action randomly based on its Q-value

$$p(A = a \mid S = s) = \frac{\exp(Q(s, a))}{\sum_{a'} \exp(Q(s, a'))}$$

Q-Learning

[Watkins & Dayan, 92]

- Q-learning iteratively approximates the state-action value function, Q
  - We won’t estimate the MDP directly
  - Learns the value function and policy simultaneously
- Keep an estimate of Q(s, a) in a table
  - Update these estimates as we gather more experience
  - Estimates do not depend on exploration policy
  - Q-learning is an off-policy method

Q-Learning Algorithm

1. Initialize $Q(s, a)$ to 0, $\forall s,a$
   (Often initialize randomly or with prior knowledge)
2. Observe state, s
3. ε-greedy pick an action, a
4. Observe next state, s', and reward, r
5. $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
6. $s \leftarrow s'$
7. Go to 2

$0 \leq \alpha \leq 1$ is the learning rate (which we might decay α)

Note: this formulation is from Sutton & Barto's "Reinforcement Learning"

Breaking apart that update formula

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

This can be written another way…

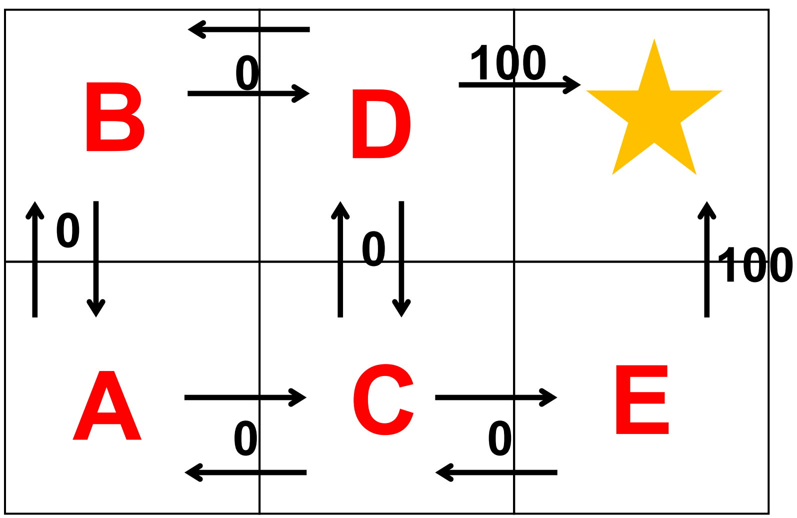
$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$$

Looked at this way, it is more obvious that α controls whether we value past experience more or new experience more.

Q-learning

- Q-learning, learns the expected utility of taking a particular action **a** in state **s**

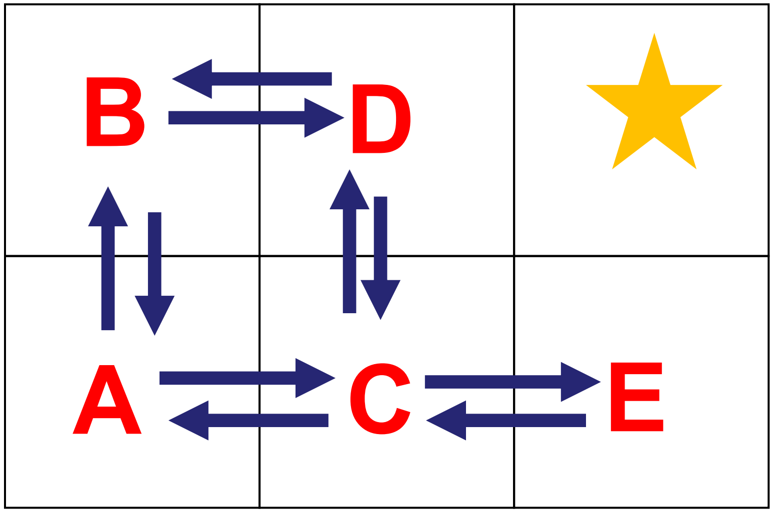
$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

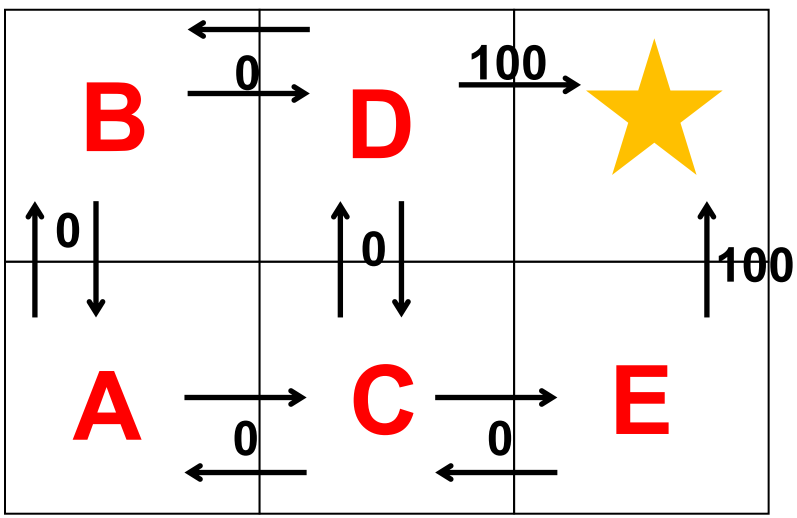


r (state, action)
immediate reward values

State	North	South	East	West
A				
B				
C				
D				
E				

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

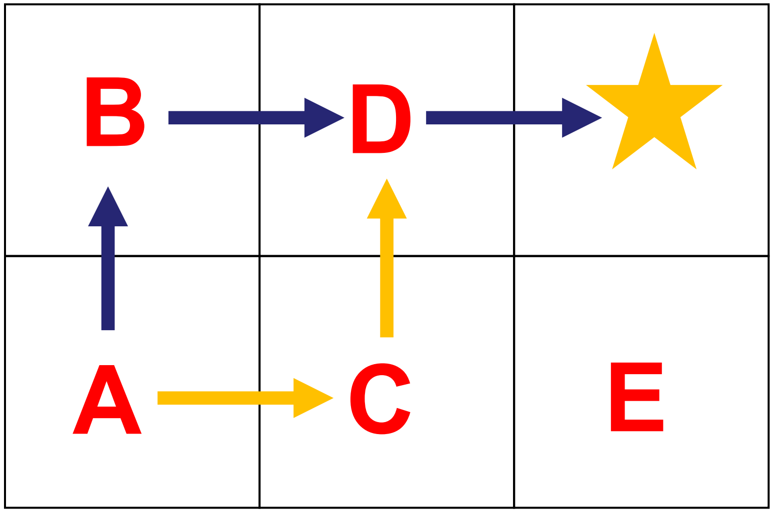




r (state, action)
immediate reward values

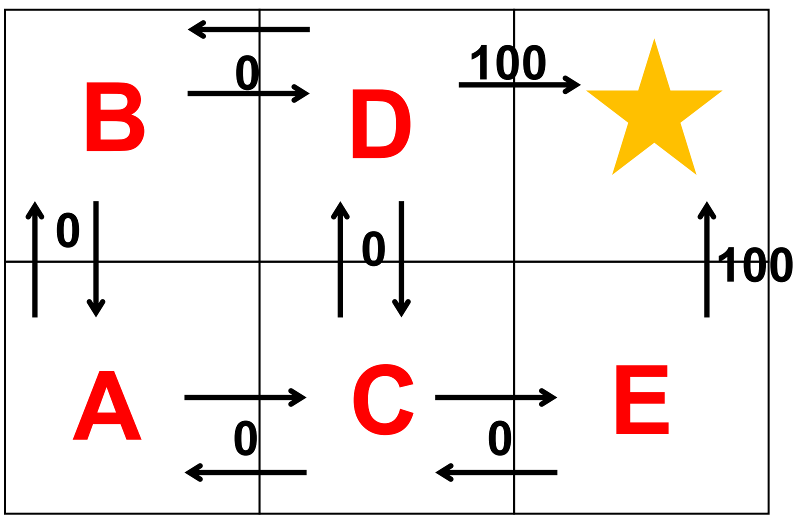
State	North	South	East	West
A	0		0	
B		0	0	
C	0		0	0
D		0	0	0
E	0			0

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



$$Q(D, \text{East}) = \alpha(100 + \gamma 0 - 0)$$

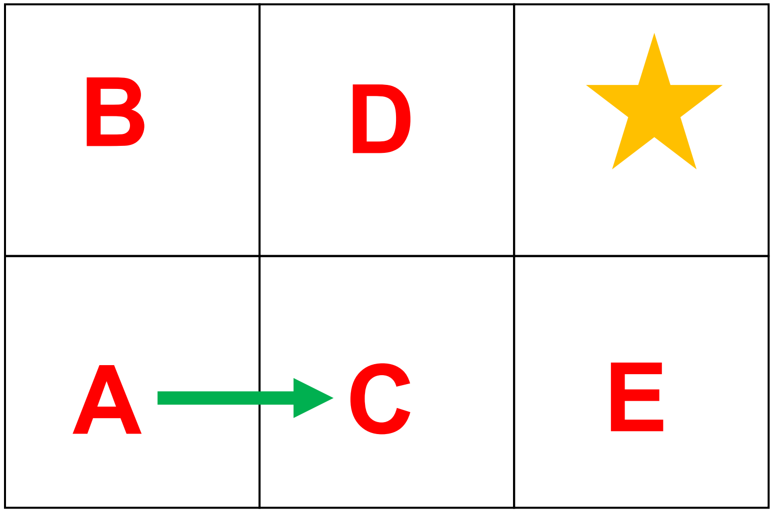
$$Q(C, \text{North}) = \alpha(0 + \gamma(\alpha 100) - 0)$$



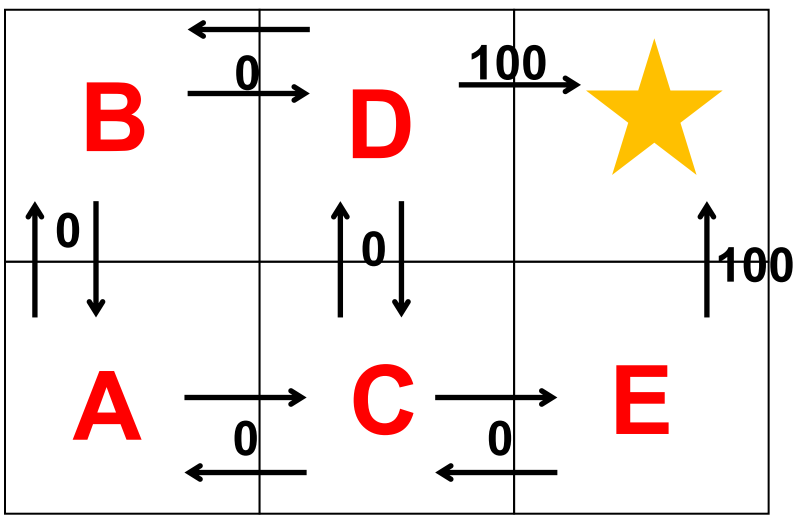
r (state, action)
immediate reward values

State	**North**	**South**	**East**	**West**
A	$\boxed{0}$		0	
B		0	$\boxed{0}$	
C	$\boxed{\alpha^2 \gamma 100}$		0	0
D		0	$\boxed{\alpha 100}$	0
E	0			0

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



$$Q(A, \text{East}) = \alpha(0 + \gamma(\alpha^2 \gamma\, 100) - 0)$$
$$= \alpha^3 \gamma^2\, 100$$

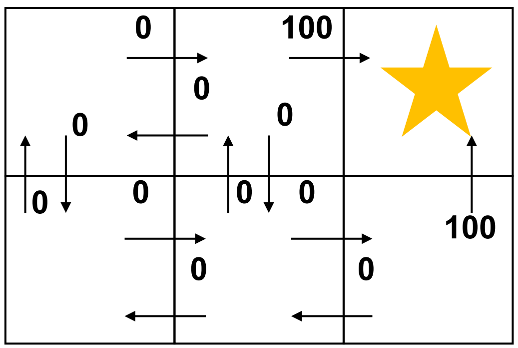


r (state, action)
immediate reward values

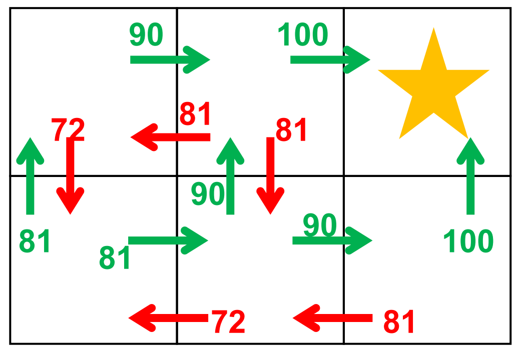
State	North	South	East	West
A	0		$\boxed{0}$	
B		0	0	
C	$\alpha^2 \gamma\, 100$		0	0
D		0	$\alpha 100$	0
E	0			0

Q-learning

- Q-learning, learns the expected utility of taking a particular action **a** in state **s**



r(state, action)
immediate reward values



Asymptotic
Q(state, action) values

RoboCup: An RL proving ground

robocub "reinforcement learning"

Scholar About 6,540 results (**0.07** sec)

[PDF] Scaling **reinforcement learning** toward **RoboCup** soccer
P Stone, RS Sutton - Icml, 2001 - academia.edu
… **RoboCup** soccer will answer this question. In this paper we begin to scale **reinforcement learning** up to **RoboCup** … we build on prior work in **RoboCup** soccer to formulate this problem at …
Save Cite Cited by 282 Related articles All 18 versions

Reinforcement learning for **robocup** soccer keepaway
P Stone, RS Sutton, G Kuhlmann - Adaptive Behavior, 2005 - journals.sagepub.com
… **RoboCup** soccer will answer this question. In this article we begin to scale **reinforcement learning** up to **RoboCup** … the limits of what **reinforcement learning** methods can tractably handle …
Save Cite Cited by 570 Related articles All 20 versions

Heuristic **reinforcement learning** applied to **robocup** simulation agents
LA Celiberto, CHC Ribeiro, AHR Costa… - Robot Soccer World …, 2007 - Springer
… **RoboCup** Simulation 2D category that learns using a recently proposed Heuristic **Reinforcement Learning** … to speed up the well-known **Reinforcement Learning** algorithm Q–Learning. A …
Save Cite Cited by 48 Related articles All 14 versions

Half field offense in **RoboCup** soccer: A multiagent **reinforcement learning** case study

RoboCup: combining many tasks

https://youtu.be/xkoXeF9oVH4

On-Policy vs. Off Policy

- On-policy algorithms
  - Final policy is influenced by the exploration policy
  - Generally, the exploration policy needs to be “close” to the final policy
  - Can get stuck in local maxima

Given enough experience

- Off-policy algorithms
  - Final policy is independent of exploration policy
  - Can use arbitrary exploration policies
  - Will not get stuck in local maxima

Convergence Guarantees

- The convergence guarantees for RL are "in the limit"
  - The word "infinite" crops up several times

- Don't let this put you off
  - Value convergence is different than policy convergence
  - We're more interested in policy convergence
  - If one action is significantly better than the others, policy convergence will happen relatively quickly

Rewards

- Rewards measure how well the policy is doing
  - Often correspond to events in the world
    - Current load on a machine
    - Reaching the coffee machine
    - Program crashing
  - Everything else gets a 0 reward

These are sparse rewards

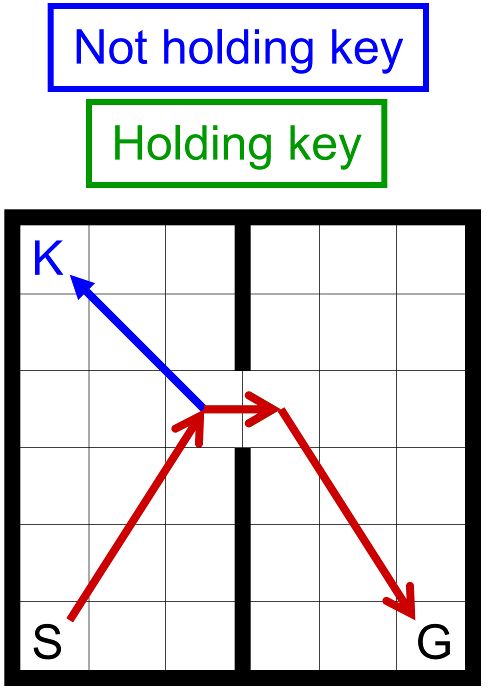
- Things work better if the rewards are incremental
  - For example, distance to goal at each step
  - These reward functions are often hard to design

These are dense rewards

The Markov Property

- RL needs a set of states that are Markov
  - Everything you need to know to make a decision is included in the state
  - Not allowed to consult the past

- Rule-of-thumb
  - If you can calculate the reward function from the state without any additional information, you're OK



But, What's the Catch?

- RL will solve all of your problems, but
  - We need lots of experience to train from
  - Taking random actions can be dangerous
  - It can take a long time to learn
  - Not all problems fit into the MDP framework

Learning Policies Directly

- An alternative approach to RL is to reward whole policies, rather than individual actions
  - Run whole policy, then receive a single reward
  - Reward measures success of the whole policy
- If there are a small number of policies, we can exhaustively try them all
  - However, this is not possible in most interesting problems

Policy Gradient Methods

- Assume that our policy, p, has a set of n real-valued parameters, $q = \{q_1, q_2, q_3, \ldots, q_n\}$
  - Running the policy with a particular q results in a reward, r_q
  - Estimate the reward gradient, $\frac{\partial R}{\partial \theta_i}$, for each q_i

$$\theta_i \leftarrow \theta_i + \alpha \frac{\partial R}{\partial \theta_i}$$

This is another learning rate

Policy Gradient Methods

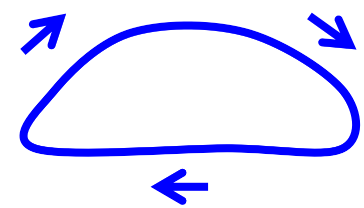
- This results in hill-climbing in policy space
  - So, it’s subject to all the problems of hill-climbing
  - But, we can also use tricks from search, like random restarts and momentum terms
- This is a good approach if you have a parameterized policy
  - Typically faster than value-based methods
  - “Safe” exploration, if you have a good policy
  - Learns locally-best parameters *for that policy*

An Example: Learning to Walk

[Kohl & Stone, 04]

- RoboCup legged league
  - Walking quickly is a *big* advantage

- Robots have a parameterized gait controller
  - 11 parameters
  - Controls step length, height, etc.



- Robots walk across soccer pitch and are timed
  - Reward is a function of the time taken

An Example: Learning to Walk

- Basic idea
  1. Pick an initial $\theta = \{\theta_1, \theta_2, \dots, \theta_{11}\}$
  2. Generate N testing parameter settings by perturbing θ

     $\theta^j = \{\theta_1 + \delta_1, \theta_2 + \delta_2, \dots, \theta_{11} + \delta_{11}\}, \quad \delta_i \in \{-\varepsilon, 0, \varepsilon\}$
  3. Test each setting, and observe rewards

     $\theta^j \to r_j$
  4. For each $\theta_i \in \theta$

     Calculate θ_1^+, θ_1^0, θ_1^- and set $\theta'_i \leftarrow \theta_i + \begin{cases} \delta & \text{if } \theta_i^+ \text{ largest} \\ 0 & \text{if } \theta_i^0 \text{ largest} \\ -\delta & \text{if } \theta_i^- \text{ largest} \end{cases}$
  5. Set $\theta \leftarrow \theta'$, and go to 2

Average reward when $q^n_i = q_i - d_i$

An Example: Learning to Walk

Initial

Final

http://utopia.utexas.edu/media/features/av.qtl

Video: Nate Kohl & Peter Stone, UT Austin

Value Function or Policy Gradient?

- When should I use policy gradient?
  - When there's a parameterized policy
  - When there's a high-dimensional state space
  - When we expect the gradient to be smooth

- When should I use a value-based method?
  - When there is no parameterized policy
  - When we have no idea how to solve the problem