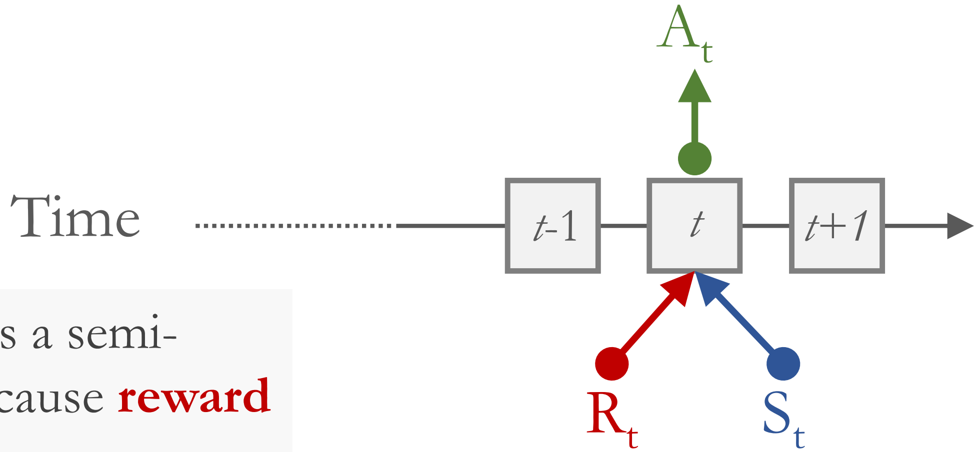
Reinforcement learning

Introduction to reinforcement learning and deep reinforcement learning

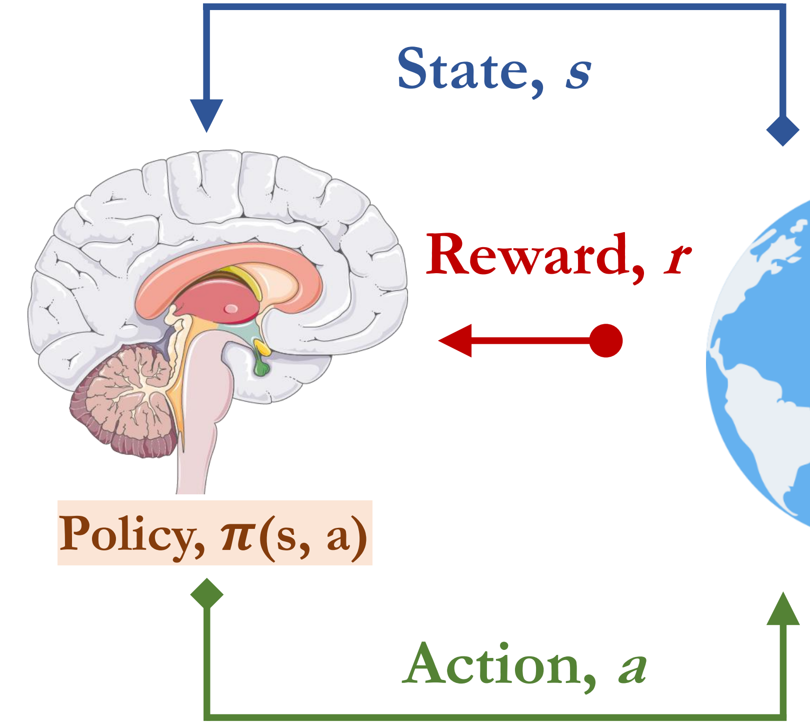
Markov Decision Process (MDP)

Reinforcement learning

Reinforcement learning is a framework for learning how to interact with the environment from experience.



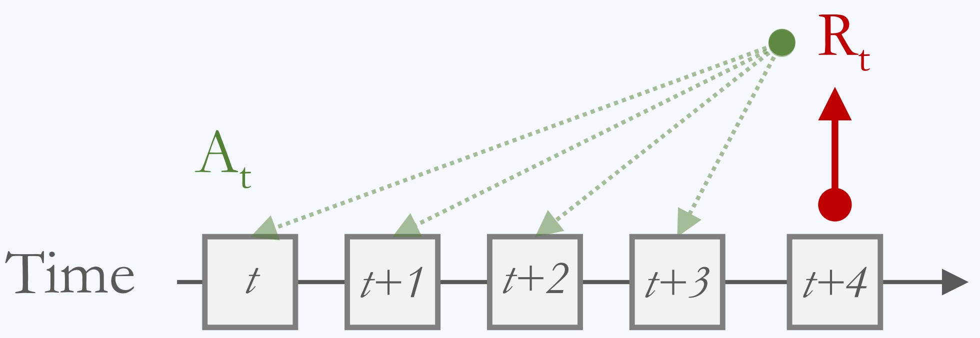
Most of the time, RL is a semi-supervised learning because **reward** is time-delayed



Environment is modelled as probabilistic (**Markov Decision Process, MDP**)

Exploration | Exploitation

Credit Assignment Problem



Source: https://www.youtube.com/watch?v=0MNVhXEX9to

• Nicolas Legrand • Postdoctoral fellow • Embodied Computation Group

Key concepts

Model: predict what the environment will do next.

$$p(s',r|s,a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)]$$

Value function: prediction of expected rewards.

$$\vartheta_\pi(s) = \mathbb{E}[R_t + \boxed{\gamma} R_{t+1} + \gamma^2 R_{t+2} + \cdots |S_t = s]$$

Discount rate

The value of a state **s** given a policy **π** is my expectation of how much reward I will get in the future if I start in that state and enact that policy.

Policy: how the agent pick its actions.

Deterministic	$\alpha = \pi(s)$
Stochastic	$\alpha \sim \pi(a|s)$

Q-learning

$$Q^\pi(s,a) = \text{quality of state/action pair}$$

$$Q(s,a) = Q^{old}(s_{t,}a_t) + \alpha(r_t + max_a Q(S_{t+1},a) - Q^{old}(s_t,a_t))$$

Given a state **s** and an action **a**, and assuming that I will do the best thing I can in the future, what is the quality of being in that state and taking that action.

Policy learning | Value learning

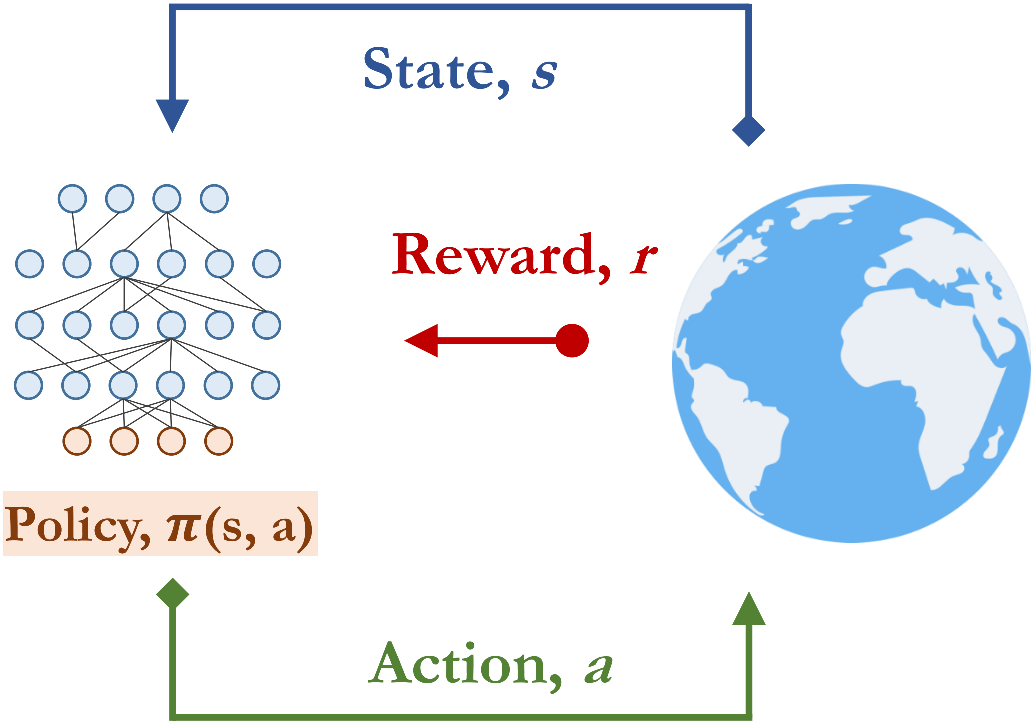
Source: https://www.youtube.com/watch?v=K67RJH3V7Yw&list=PLMsTLcO6ettgmyLVrcPvFLYi2Rs-R4JOE&index=4

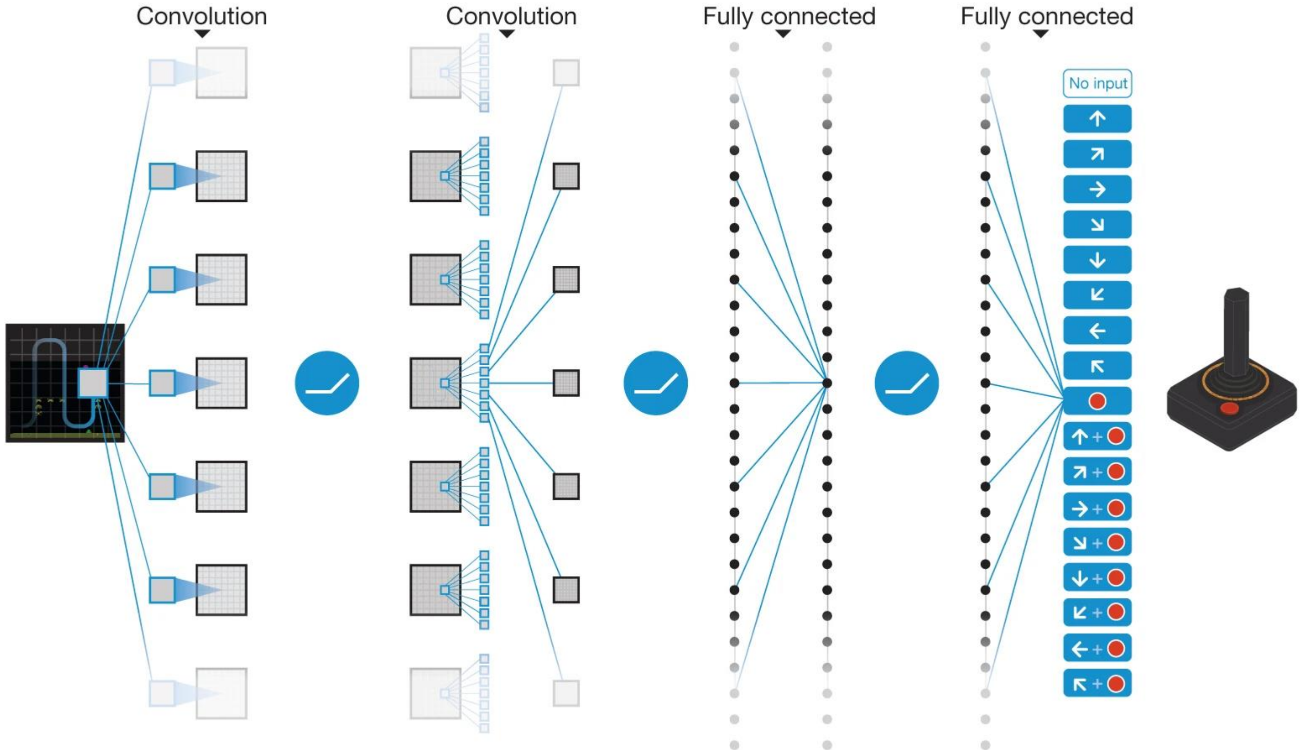
RL Algorithms

Hindsight Experience Replay
Save all behaviors and code reward for different goal.
https://www.youtube.com/watch?v=0Ey02HT_1Ho

- Reinforcement learning algorithms
  - Model-free
    - Policy optimization
    - Q-learning
  - Model-based
    - Learn model
    - Given model

Deep reinforcement learning





Mnih et al. (2015)

Source: https://www.youtube.com/watch?v=IUiKAD6cuTA

Examples

Hide and seek

https://www.youtube.com/watch?v=Lu56xVlZ40M

AlphaGo Move 37

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

Flexible muscle-based locomotion for bipedal creatures

https://vimeo.com/79098420

Cart-Pole

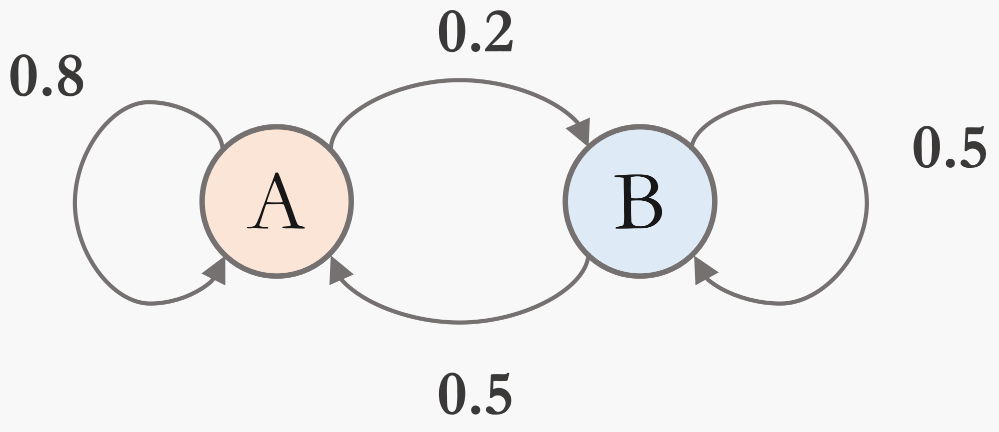
https://www.youtube.com/watch?v=XiigTGKZfks

Atari video games

https://www.youtube.com/watch?v=TmPfTpjtdgg&t=43s

Markov Decision Process

Markov chains

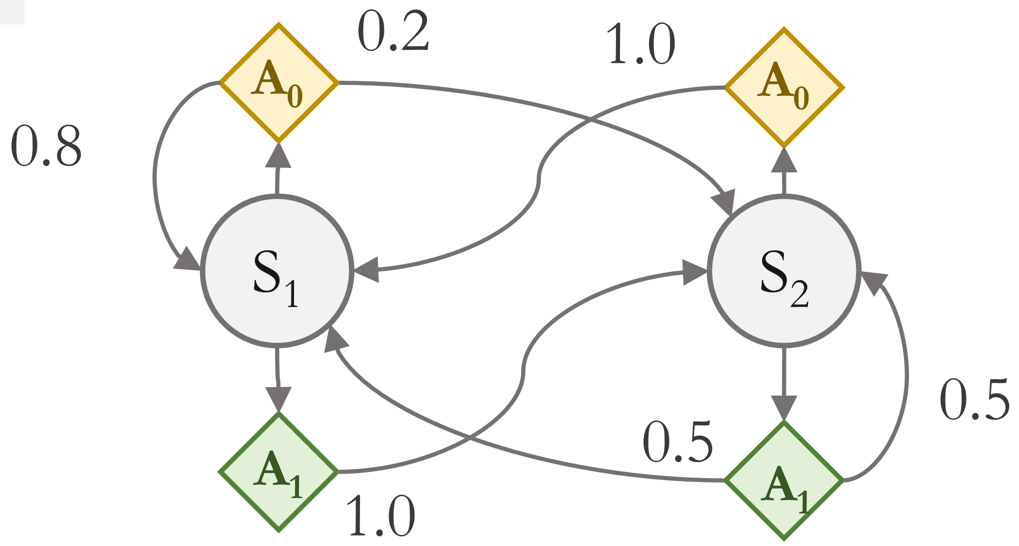


Markov property

$$P(R_{n+1}\,|R_1\,,R_2\,...\,R_n) = P(R_{n+1}\,|\,R_n)$$

Almost all reinforcement learning problems can be modeled as MDP.

Markov decision process

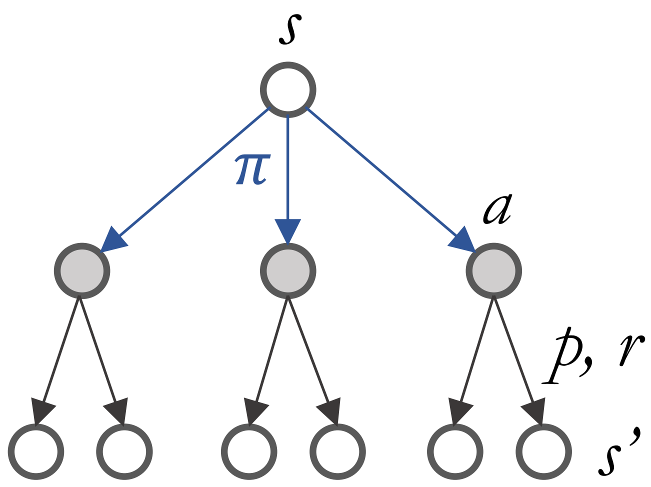


s	a	s'	p(s' | s, a)	r(s, a, s')
S1	A0	S1	0.8	1
S1	A0	S2	0.2	1
S1	A1	S2	1.0	1
S2	A0	S1	1.0	1
S2	A1	S2	0.5	1
S2	A0	S1	1.0	1

The dynamic function

$$p(s', r|s, a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$$

The probability of going from state *s* to state *s'*, getting the reward *r*, only depend on state *s* and the action *a* initiated by the agent.

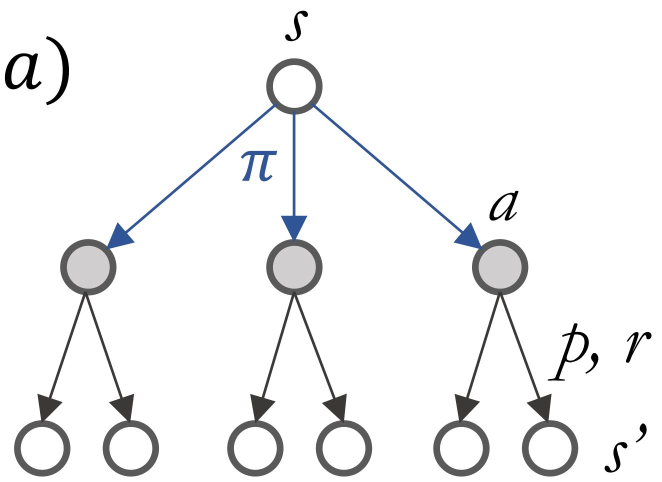


$$p(s'|s, a) = P(S_t = s'|S_{t-1} = s, A_{t-1} = a) = \sum_{r \epsilon \mathcal{R}} p(s', r|s, a)$$

A corollary of this is that the probability of going from state *s* to state *s'*, only depend on state *s* and the action *a* initiated by the agent, considering all the possible rewards *r*.

The reward function

$$r(s,a) = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in S} p(s', r | s, a)$$



The reward value that we can expect when making action *a* while in state *s* is the weighted sum of possible rewards and their probabilities.

$$r(s,a,s') = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s' | s, a)}$$

The reward value that we can expect when making action a while in state s and going to state s' is the weighted sum of possible rewards by the ratio of their probabilities.

Goals and return

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

The return is the sum of rewards. An agent tries to maximize the expected return.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

This works well for *episodic task* that have a finite number of states. For *continuing tasks*, we use a **discounting factor**.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) \\ &\boxed{= R_{t+1} + \gamma G_{t+1}} \end{aligned}$$

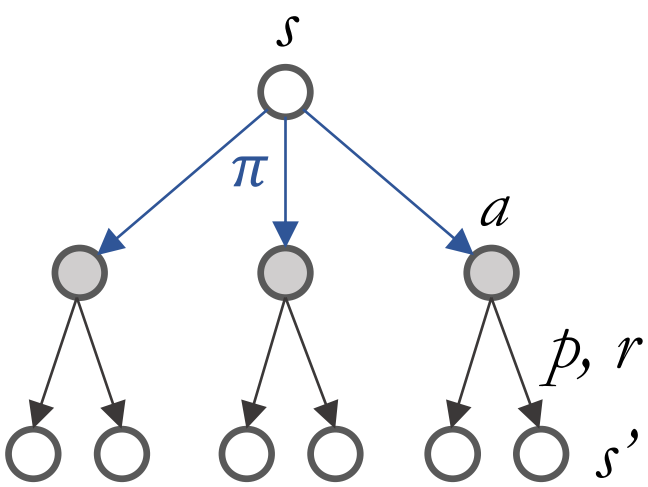
Consistency condition

Returns at successive time steps are related to each other in a way that is important for the theory and algorithms of reinforcement learning.

Value function

Value functions estimate how good it is for the agent to be in a given state.

Value functions are defined with respect to particular ways of acting, called policies.



π(a|s) is the probability of performing action *a* in state *s*.

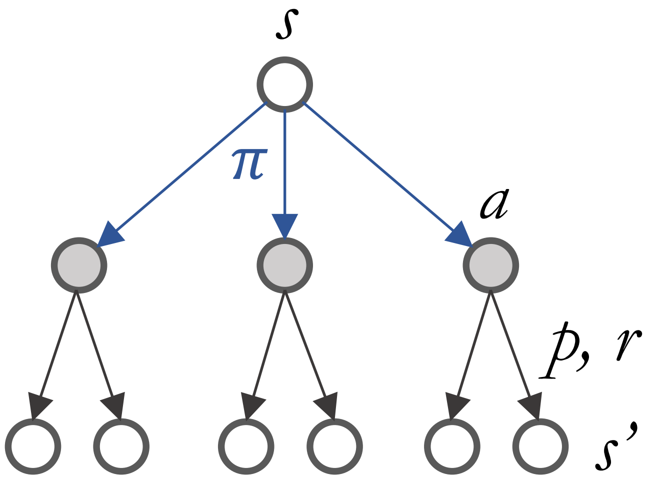
$$\vartheta_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

The value function of a state *s* under a policy **π**, denoted $\vartheta_\pi(s)$, is the expected return when starting in *s* and following **π** thereafter.

Action-value function

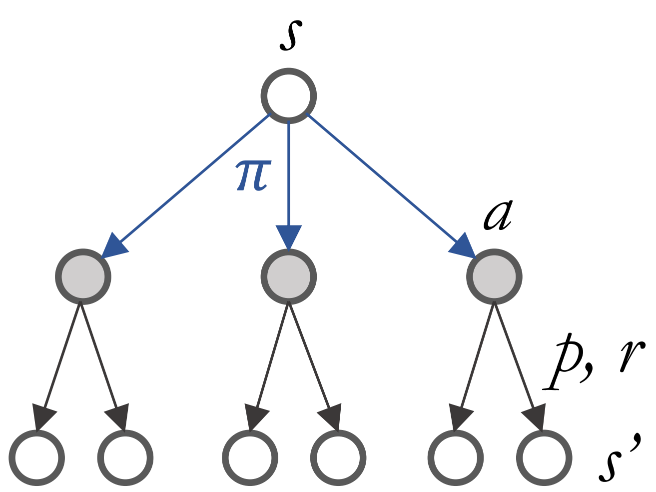
$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s,\, A_t = a\right]$$



The value function of taking action *a* in state *s* under a policy π, denoted $q_\pi(s,a)$, is the expected return when starting from *s*, taking action *a*, and following π thereafter.

Bellman expectation equation



$$\vartheta_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right]$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma \vartheta_\pi(s')]$$

The Bellman equation averages over all the possibilities, weighting each by its probability of occurring.

Optimal policies and optimal value functions

Optimal state-value function

$$\vartheta_*(s) = max_\pi \, \vartheta_\pi(s)$$

There is always at least one policy that is better than or equal to all other policies. This is an optimal policy, we denote by π_*.

Optimal policies also share the same optimal action-value function, denoted q_*

$$q_*(s,a) = max_\pi \, q_\pi(s,a)$$

We can write q_* in term of ϑ_*

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \vartheta_*(S_{t+1}) \mid S_t = s, A_t = a]$$

Bellman optimality equation

$$\vartheta_*(s) = max_{a \in A(s)}\ q_{\pi*}(s,a)$$

$$= max_{a \in A}\ q_{\pi*}(s,a)$$

$$= max_a \mathbb{E}_{\pi*}[G_t \mid S_t = s, A_t = a]$$

$$= max_a \mathbb{E}_{\pi*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a]$$

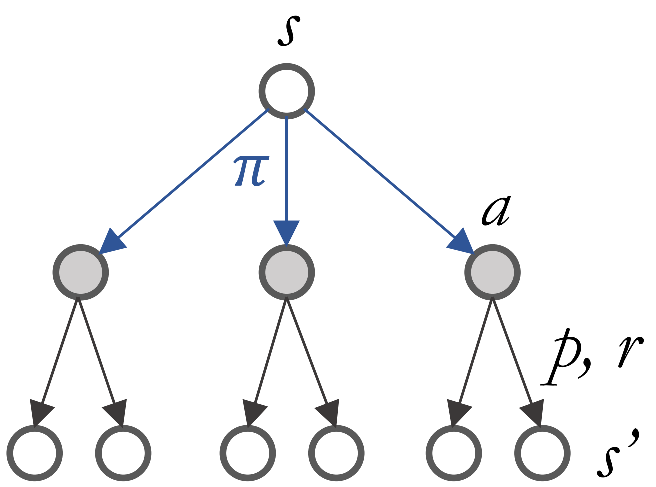
$$\boxed{= max_a \mathbb{E}_{\pi*}[R_{t+1} + \gamma \vartheta_*(S_{t+1}) \mid S_t = s, A_t = a] \qquad = max_a \sum_{s',r} p(s',r \mid s,a)[r + \gamma \vartheta_*(s')]}$$

The Bellman equation averages over all the possibilities, weighting each by its probability of occurring.

The value of a state under an optimal policy must equal the expected return for the best action from that state.

The Bellman optimality equation is actually a system of equations, one for each state, so if there are *n* states, then there are *n* equations in *n* unknowns.

Bellman optimality equation



The Bellman optimality equation for q_* is

$$q_*(s,a) = \mathbb{E}_{\pi*}[R_{t+1} + \gamma max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a]$$

$$= max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma max_{a'} q_*(s', a)]$$

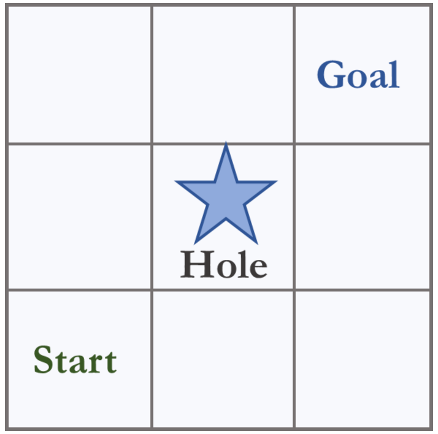
Dynamic programming

Iterative policy evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

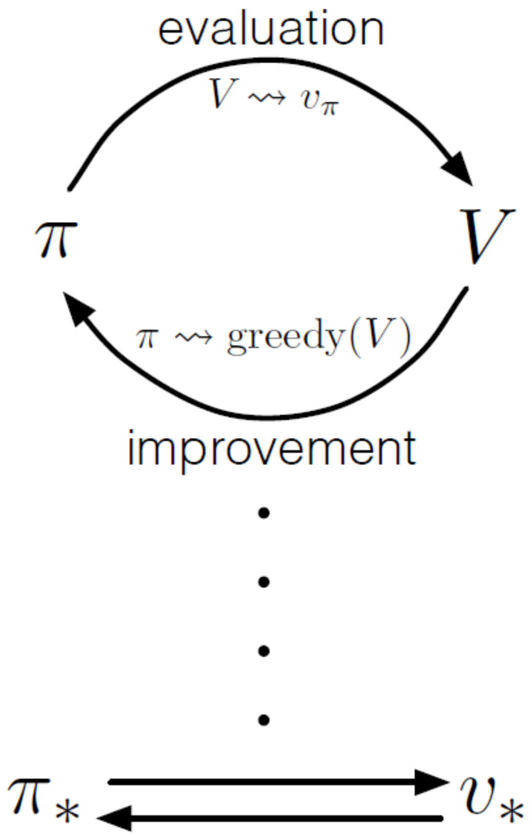
Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$



Frozen Lake

Policy improvement

Interactive demo:
https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html



Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s))\left[r + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r | s, a)\left[r + \gamma V(s')\right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Value iteration

Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

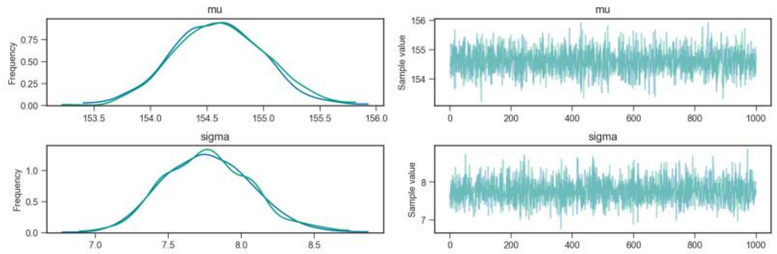
Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$

Monte Carlo Methods

***Temporal Difference Learning** and and **Monte Carlo methods** use experience to solve the prediction problem.*

Randomly sampling and averaging the return



e.g. Makov Chains Monte Carlo

Monte Carlo Methods use empirical means instead of expected return.

Update

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \mu_{k-1} + \boxed{\frac{1}{k}(x_k - \mu_{k-1})}$$

The mean is computed incrementally by adding observed values at each iteration

Greedy at the limit with infinite exploration

- Explore everything
- The policy converges on a greedy policy

1. Sample episodes using policy π

2. Increase a visit counter for each state visited:

$$N(S_t) = N(S_t) + 1$$

3. Update the value function at the end of each episodes using:

$$V(S_t) = V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

Temporal Difference Learning

Monte Carlo Methods

Does not need to explore the whole environment

+

Dynamic Programming

Update the estimate without getting the actual outcome

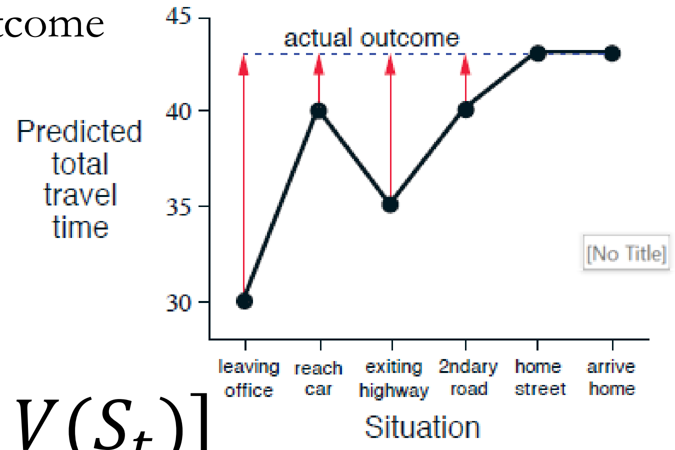
Update the value function towards an estimate of the return

TD target

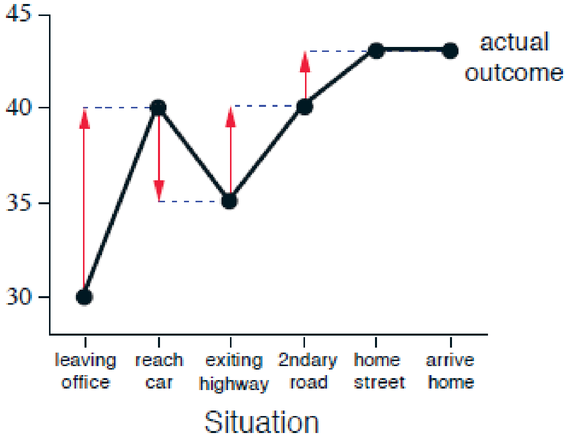
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Learning rate

MC Methods



TD learning



On-policy

Evaluates the policy that is currently being followed

Off-policy

Evaluates policies different from the one being currently followed

Q-learning

Learning rate
Reward
Discount factor
Estimate of optimal future value

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\underbrace{R_{t+1} + \gamma max_a Q(S_{t+1}, a)}_{\text{Update}} - Q(S_t, A_t)]$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal