Reinforcement Learning (Part 1)

Machine Learning

Michael Wand, TA: Dylan Ashley

{michael.wand, dylan.ashley}@idsia.ch

Dalle Molle Institute for Artificial Intelligence Studies (IDSIA) USI - SUPSI

Fall semester 2024

Introduction

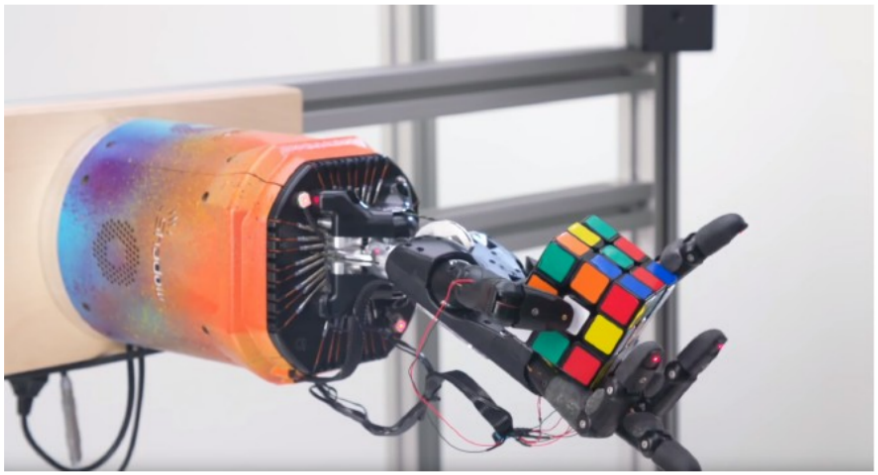
- Reinforcement Learning deals with an agent taking actions in some kind of environment.
- The agent collects scalar rewards, its goal is to maximize the cumulative reward (the return) over time.
- This concept differs from supervised learning in several ways:
  - ➔ learn actions instead of predictions
  - ➔ no fixed data set, data is generated by interacting with the environment
  - ➔ rewards do not explicitly give error correction
  - ➔ rewards are *asynchronous*, not clear which action led to which reward
  - ➔ inherently sequential, focus on online capabilities.
- Often only partial information is available (one does not know the full state of the environment).

Examples

Game playing: Racing [Koutnik et al. 2013],
diverse ATARI games [Mnih et al. 2015],
Dota 2 [Brockman et al. 2019],
Chess [Silver et al. 2018]

Examples

Robotics: Rubic's Cube Manipulation [Brockman et al. 2019]



References

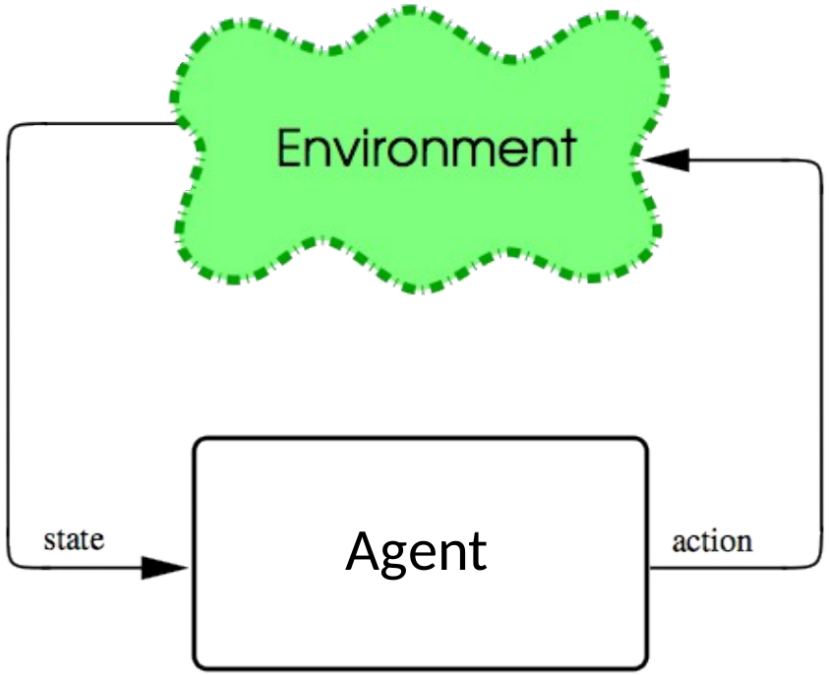
- Standard reference: Sutton/Barto, *Reinforcement Learning - An Introduction* (available online, recommended reading)
- Good blog: `https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html`
- Recommended resource, in particular for advanced concepts: The deep RL bootcamp `https://sites.google.com/view/deep-rl-bootcamp/lectures`

Some slides are based on lectures given by Faustino Gomez and Jan Koutnik

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

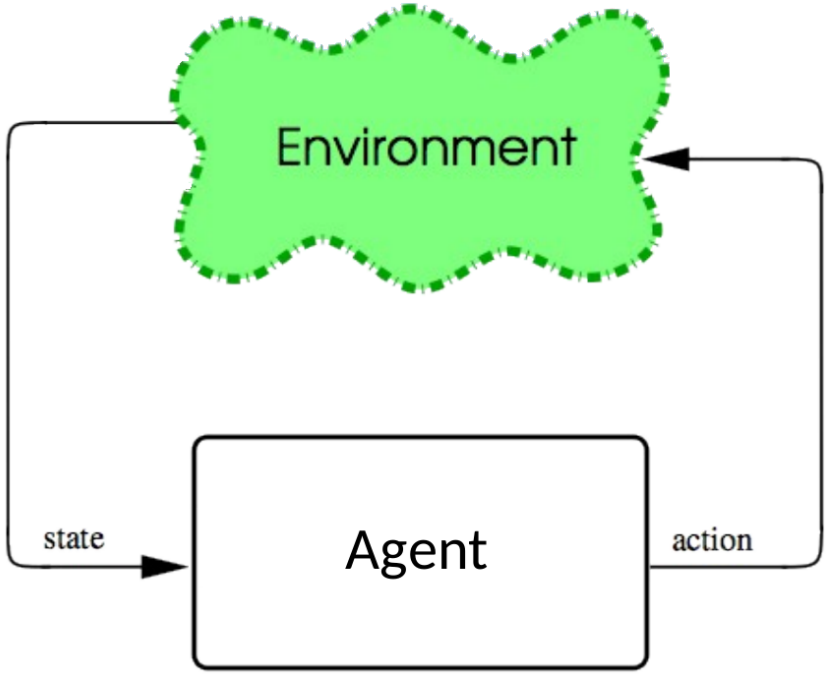
Basics of Reinforcement Learning

Overview



- The agent "sees" the state of the environment at each time step and must select best action to achieve a goal.
- Problem: We do not know what the correct actions are, so we cannot learn from examples!

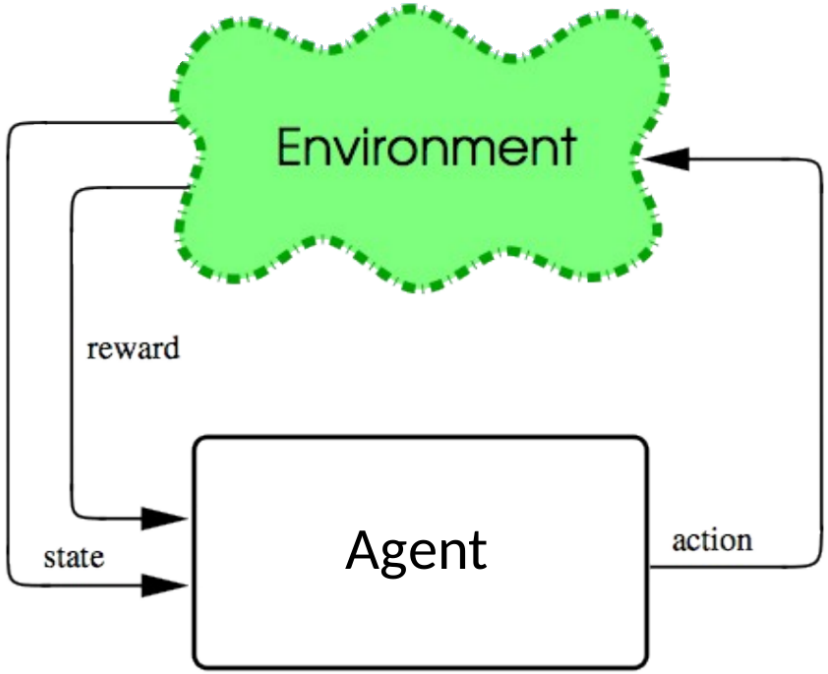
Overview



- Define S_t as the state of the environment at time t, and A_t the action taken at time t, after seeing S_t.
- Then we have a sequence

$$\ldots \xrightarrow{A_{t-1}} S_t \xrightarrow{A_t} S_{t+1} \xrightarrow{A_{t+1}} S_{t+2} \xrightarrow{A_{t+2}} \ldots$$

Overview



- At each step, the agent receives a scalar *reward* R_t (may be 0).
- Now we have a sequence

$$\ldots \xrightarrow{A_{t-1}} S_t \xrightarrow{A_t} S_{t+1} \xrightarrow{A_{t+1}} S_{t+2} \xrightarrow{A_{t+2}} \ldots$$

R_{t-1} R_t R_{t+1}

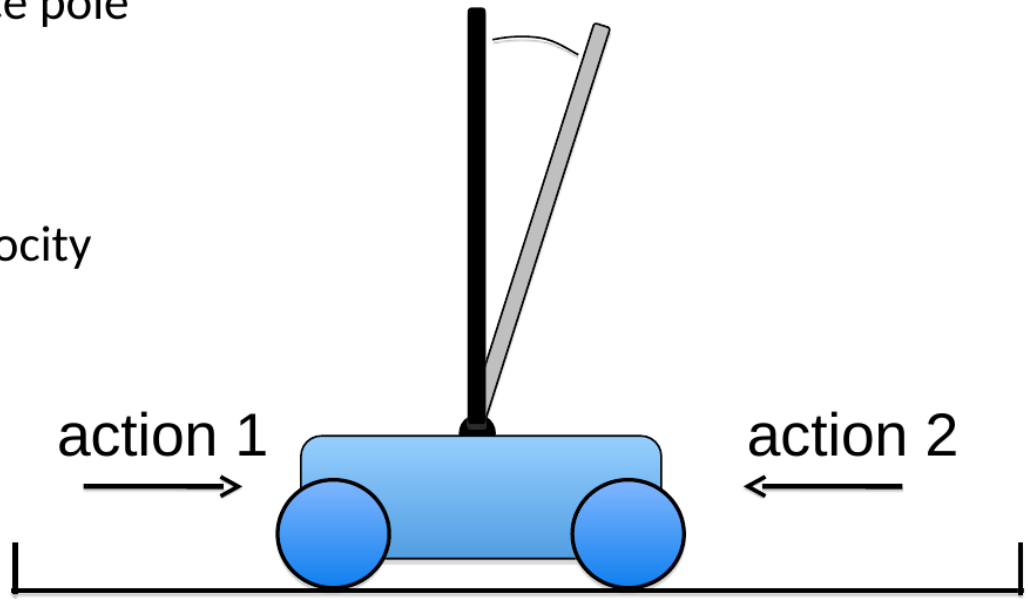
Example: Cartpole

Apply force to cart to balance pole

4 state variables

- position + velocity
- pole angle + angular velocity

negative reward only
when pole falls



RL benchmark for over two decades!

some concrete implementations: Kumar, Balancing a CartPole System with Reinforcement Learning - A Tutorial. arXiv 2006.04938, 2020

Reward and Return

- The cumulative future reward is called the return G:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{H} \gamma^k R_{t+k+1}.$$

  where $0 < \gamma \leq 1$ is the discount factor.
- Future rewards are taken into account up to the horizon H, which may be finite or infinite.
  - ➔ $\gamma = 1$: all rewards matter equally, the agent is farsighted
  - ➔ $\gamma < 1$: rewards further in the future are less important than rewards obtained sooner, the agent is shortsighted.
- In this lecture we assume $H = \infty$ (for mathematical convenience). A finite horizon is approximated by setting $\gamma < 1$.
- Objective of RL: *choose actions to maximize the (expected) return!*

The Policy

- The policy π defines the agent's behavior:
- For any state s the agent encounters, the policy tells it what action a to take.
- The policy may be stochastic, i.e. given by a probability distribution $\pi(s, a)$, or deterministic, i.e. $a = \pi(s)$.
- We can reformulate the objective of RL using this definition: *learn a policy which maximizes the (expected) return!*

Observations

- Why are we talking about the *expected* return?
  - ➔ the environment is often stochastic (i.e. the same action may lead to different successor states and rewards)
  - ➔ as we have seen, also the policy may be stochastic
  - ➔ this means that also the return may be stochastic. We are interested in the expected value of the return.
- Actions should be chosen to maximize the return, i.e. the cumulative future reward. *Always keep in mind that choosing an action now may lead to a (positive or negative) reward only much later in the future!*
  - ➔ Credit assigment problem: how much credit should each action in the sequence of actions get for the outcome?
  - ➔ No explicit error correction: reward valuates actions, but does not give instructions
- We assume (for now) the environment is stationary: its properties do not change over time.

Model-based Reinforcement Learning

- In Reinforcement Learning, the term model refers to
  - ➔ the set of states,
  - ➔ the ways in which actions lead to state transitions
  - ➔ the rewards which are obtained in each state, or state-action pair.
- If this information is perfectly known, we can do model-based reinforcement learning:
  - ➔ Create a full map of optimal actions in each state
  - ➔ Plan actions which lead to high return
- Requirement: set of states (and actions) is finite and observable.

Model-free Reinforcement Learning

- What if any of the following is true?
  - ➔ There is an very large or unlimited number of states
  - ➔ The state cannot be completely observed
  - ➔ Even if we know about the states, we do not know about expected rewards
- In this case, we cannot rely on a full map of states to actions.
  - ➔ Apply model-free reinforcement learning
  - ➔ Focus on learning to derive good action from a (partial) observation of a state
  - ➔ Use diverse Machine Learning techniques (e.g. neural networks)
  - ➔ Generalize across states or observations of states!

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

Model-based Reinforcement Learning

Overview

- In this section, we cover basic reinforcement learning methods in relatively simple scenarios:
  - There is a finite number of states $\{s_1, \ldots, s_N\}$, and we can exactly observe the current state s_n at a time step t: $S_t = s_n$.
  - Likewise, the set of possible actions $\{a_1, \ldots, a_M\}$ finite.
  - Each action causes a change of state, which may or may not be deterministic.
  - We know the distribution of rewards in a particular state, or for a particular state-action pair. (Also rewards may be probabilistic!)
  - In particular, the whole model has a finite description.

Markov Decision Process

- This setup is formalized as a Markov Decision Process (MDP).
- At any time t, we are in a state S_t.
- In each time step, we reach a new state.
- The state transitions have the Markov property:

$$P(S_{t+1} = s' | S_t, S_{t-1}, \ldots, S_1) = P(S_{t+1} = s' | S_t);$$

  i.e. the transition probabilities to the next state depend only on the current state, not on the past.
- The reward likewise only depends on the current state and the current action, not on the past.
- Frequently assume a stationary environment.

Markov Decision Process

- Formally, an MDP is a tuple $(\mathcal{S}, \mathcal{A}, P^a_{ss'}, R^a_{ss'})$, where
  - $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states
  - $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions
  - $P^a_{ss'}$ are the state transition probabilities:

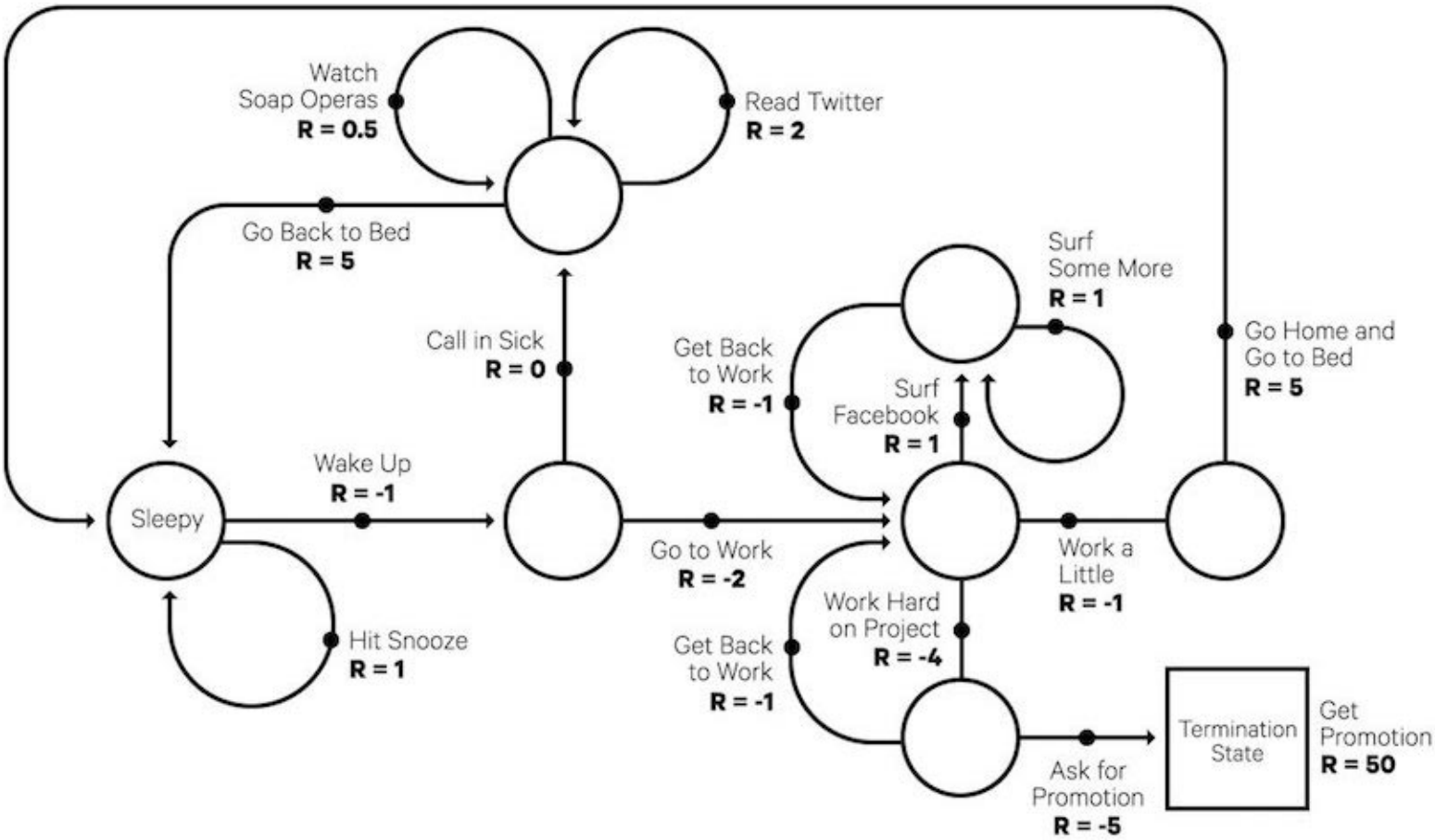
$$P^a_{ss'} = P(S_{t+1} = s' | S_t = s, A_t = a)$$

  - $R^a_{ss'}$ is the reward function:

$$R^a_{ss'} = E(R_{t+1} | S_{t+1} = s', S_t = s, A_t = a)$$

- We also include the discount factor γ in this definition.

Markov Day



Source: https://randomant.net/reinforcement-learning-concepts

Value/Action-Value Function

- The value function indicates how "good" it is to be in a given state, given a policy π.

$$V^{\pi}(s) = E(G_t|S_t = s) \qquad \text{where} \qquad G_t = \sum_{k=0}^{H} \gamma^k R_{t+k+1}.$$

- It is defined as the reward the agent can expect to receive in the future if it continues with its policy from state s.
- Note that we assume a stationary environment (definition does not depend on t).
- We similarly define the action-value function:

$$Q^{\pi}(s, a) = E(G_t|S_t = s, A_t = a)$$

which is the expected return if the agent chooses action a in state s, and then continues to follow policy π.

Value/Action-Value Function

- Clearly, the value function can be computed from the action-value function:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) Q^{\pi}(s,a)$$

- Likewise, the action-value function can be derived from the value function by looking one step into the future:

$$\begin{aligned} Q^{\pi}(s,a) &= E\left[\sum_{k=0}^{H} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \\ &= \sum_{s' \in \mathcal{S}} P^a_{ss'} \left[R^a_{ss'} + \gamma E\left(\sum_{k=0}^{H} \gamma^k R_{t+k+2} \middle| S_{t+1} = s'\right)\right] \\ &= \sum_{s' \in \mathcal{S}} P^a_{ss'} \left[R^a_{ss'} + \gamma V^{\pi}(s')\right] \end{aligned}$$

- Note that this is only exact for infinite horizon ($H = \infty$).

Bellman Equations

- Plugging these equations together yields the following recursive relation:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

- This is called the Bellman equation for the value function.
- For a given policy, *the value function is the unique solution to the Bellman equation.*
- A similar recurrence can be found for the action-value function:

$$Q^{\pi}(s,a) = \sum_{s' \in \mathcal{S}} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma \sum_{a \in \mathcal{A}} \pi(s',a) Q^{\pi}(s',a) \right]$$

- The agent can now quantify the properties of the environment, and the quality of its policy.
- Importantly, it can also use V and/or Q to *improve* the policy!

Deriving a Policy

- Assume the Q (or V) are given.
- Then we can derive a policy as follows: in each state, choose the action with the highest value, i.e.

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

- (Homework: How can we do this if V instead of Q is given?)
- This policy is called the greedy policy.
- More generally, we can define
  - ➔ ϵ-greedy policy: select greedy action $1 - \epsilon$ of the time and some other, random action ϵ of the time
  - ➔ stochastic policy: select actions probabilistically (e.g. following the relative values of Q)
- Clearly, this does not solve the RL objective (why?)

The Optimal Value Function

- We have seen that we can derive a value/action-value function from the policy, and a policy from the value or action-value function.
- How can we use this ideas developed so far to actually determine optimal policies, or equivalently the value / action-value function related to the optimal policy?
- We define the optimal value function and optimal action-value function as

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

The Optimal Value Function

- Since $V^*(s)$ is a value function of a policy, it follows the usual recurrence condition for value functions.
- But since it is the value function of the best policy, it can also be written as:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

  (this can easily be derived along the lines of deriving the Bellman equation).
- Note that this recurrence relation does not depend on any policy - it only depends on the model!
- Consequently, if we can determine $V^*(s)$, we can derive the optimal policy from it!

The Optimal Value Function

- The "beauty" (Sutton) of $V^*(s)$:
  - ➔ our problem so far: we can easily select the action which gives the highest immediate reward (greedy policy)
  - ➔ but we want long-term cumulative reward (i.e. the return)!
  - ➔ The values of the optimal value function $V^*(s)$ account for the long-term reward in each state!
  - ➔ The greedy policy is now optimal.
- Of course, we still need to estimate/approximate $V^*(s)$ ...

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

Dynamic Programming

Solving finite RL Problems

Two related algorithms for practical optimization:

- Policy Iteration
  - ➔ Start with random policy
  - ➔ Iteratively repeat two steps:
  - ➔ Policy Evaluation: compute value function of current policy
  - ➔ Policy Improvement: Improve policy with respect to current value function
  - ➔ can be shown to converge to optimal policy
- Value Iteration
  - ➔ Start with random value function
  - ➔ Iteratively improve value function (greedily), converges to $V^*(s)$
  - ➔ Compute optimal policy from $V^*(s)$

Policy Evaluation

- Assume a policy is given.
- How to compute the value function for this policy?
- Iterative approach:
  - ➔ initialize $V_0(s)$ randomly
  - ➔ iterate using the Bellman equation as update rule:

$$V_k(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P^a_{ss'} \left[R^a_{ss'} + \gamma V_{k-1}(s') \right]$$

- After enough iterations, $V_k(s)$ converges to the true value function of the policy. (Why? Remember that $V^\pi(s)$ is the only fixed point of the Bellman equation.)

Policy Improvement

- Assume we have determined the value function for a given policy. How can we improve this policy?
- For instance, assume we are in state s. Should be choose an action a which is different from the one suggested by the current policy?
- Idea: Try it out once and then follow the old policy.
- If the new action a is better, we should assume it is better *all the time*.

Policy Improvement Theorem

- Let π and π' be any pair of deterministic policies such that

$$Q^{\pi}(s, pi'(s)) \geq V^{\pi}(s)$$

- Then the policy π' is better than π: For all states,

$$V^{\pi'}(s) \geq V^{\pi}(s)$$

- Furthermore, if the first inequality is strict for any state, then also the second inequality is strict.

Policy Improvement

- We now naturally extend the idea to all states.
- At each state, select the action which appears best, taking the current reward and the value of the next state into account:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

- Due to the policy improvement theorem, the new policy is better than the old one.
- In fact, it is strictly better, unless the optimal policy has already been found.

Policy Iteration

- Policy iteration consists of iterative policy evaluation and policy improvement.
- In a finite MDP, there are only finitely many deterministic policies, so the algorithm is guaranteed to converge.
- However, the algorithm can be very slow for large state spaces.
- Finally, note that all the ideas so far also extend to stochastic policies.

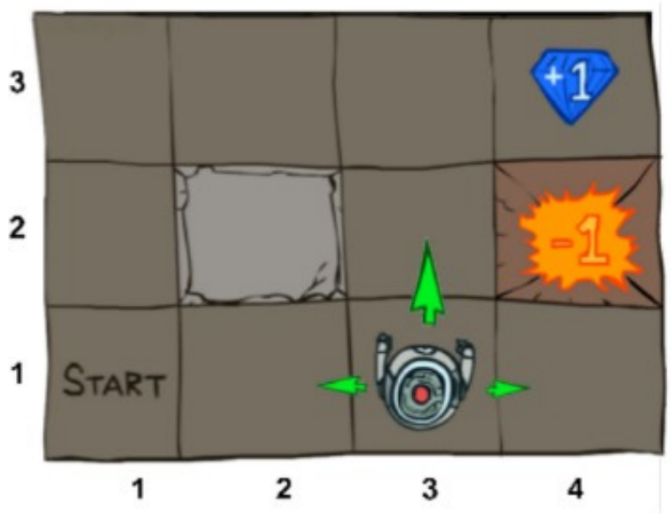
Value Iteration

- Value iteration directly optimizes the value function, using the Bellman optimality equation as an update rule!
- It can be seen as a special case of policy iteration, where the policy evaluation step is truncated.
- The algorithm:
  - initialize $V_0(s) = 0$ for all s
  - for $k = 1, \ldots, K$ or until convergence

$$V_k(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P^a_{ss'} \left[R^a_{ss'} + \gamma V_{k-1}(s') \right]$$

Value Iteration Example

- An example for value iteration! (taken from the Deep RL bootcamp, lecture 1)
- robot should reach diamond (reward $+1$) and avoid bomb (reward -1), states with reward are final
- 12 states (really 11 and the wall), 4 actions
- assume discounting, $\gamma = 0.9$



Value Iteration Example

- robot should reach diamond (reward +1) and avoid bomb (reward -1), states with reward are final
- 12 states (really 11 and the wall), 4 actions, discount γ=0.9

initialization

0.0	0.0	0.0	0.0
0.0	XXX	0.0	0.0
0.0	0.0	0.0	0.0

first iteration

0.0	0.0	0.0	1.0
0.0	XXX	0.0	-1.0
0.0	0.0	0.0	0.0

Value Iteration Example

second iteration

0.0	0.0	0.9	1.0
0.0	XXX	0.0	-1.0
0.0	0.0	0.0	0.0

third iteration

0.0	0.81	0.9	1.0
0.0	XXX	0.81	-1.0
0.0	0.0	0.0	0.0

fifth iteration

0.73	0.81	0.9	1.0
0.66	XXX	0.81	-1.0
0.0	0.66	0.73	0.66

finished

0.73	0.81	0.9	1.0
0.66	XXX	0.81	-1.0
0.59	0.66	0.73	0.66

Value Iteration Example

From the optimal value function, we can compute the optimal policy:

optimal value function

0.73	0.81	0.9	1.0
0.66	XXX	0.81	-1.0
0.59	0.66	0.73	0.66

associated policy

→	→	→	ok
↑	XXX	↑	bad
→ ↑	→	↑	←

Value Iteration Example

- same as before, but now state transitions are *stochastic*
- … i.e. noisy
- with probability 0.2, robot goes to an *arbitrary* adjacent state instead of performing the action

first iteration

0.0	0.0	0.0	1.0
0.0	XXX	0.0	-1.0
0.0	0.0	0.0	0.0

second iteration

0.0	0.0	0.77	1.0
0.0	XXX	-0.05	-1.0
0.0	0.0	0.0	-0.05

third iteration

0.0	0.59	0.76	1.0
0.0	XXX	0.54	-1.0
0.0	0.0	$-5e^{-3}$	-0.05

- rest left as exercise

Policy Iteration Algorithm

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 $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
   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

Source: Sutton/Barto

Value Iteration Algorithm

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

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

Source: Sutton/Barto

Conclusion/Outlook

- DP methods have their limitations, in particular:
  - ➔ state and action space must be finite
  - ➔ we need a complete model of the environment.
- However, in practical cases,
  - ➔ the model may be intractable or inexistent
  - ➔ the agent can only observe state transitions caused by its own actions (so the state space may be unknown, and one cannot easily reach or evaluate every state)
  - ➔ the model may only be partically observable.
- Solution: learn stochastically, approximate the action space, using ML techniques!