Reinforcement Learning

Markov decision process & Dynamic programming

value function, Bellman equation, optimality, Markov property, Markov decision process, dynamic programming, value iteration, policy iteration.

Vien Ngo
MLR, University of Stuttgart

Outline

- Reinforcement learning problem.
  - Element of reinforcement learning
  - Markov Process
  - Markov Reward Process
  - Markov decision process.

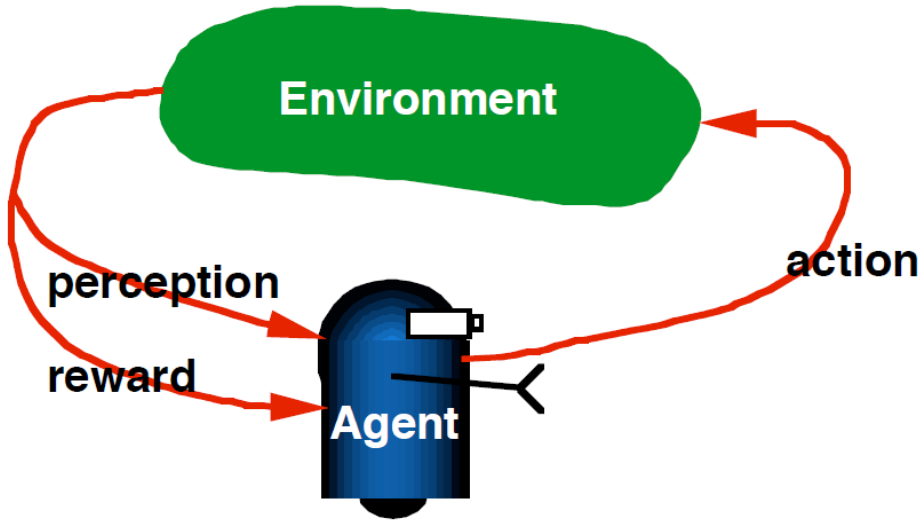
- Dynamic programming
  - Value iteration
  - Policy iteration

Reinforcement Learning Problem

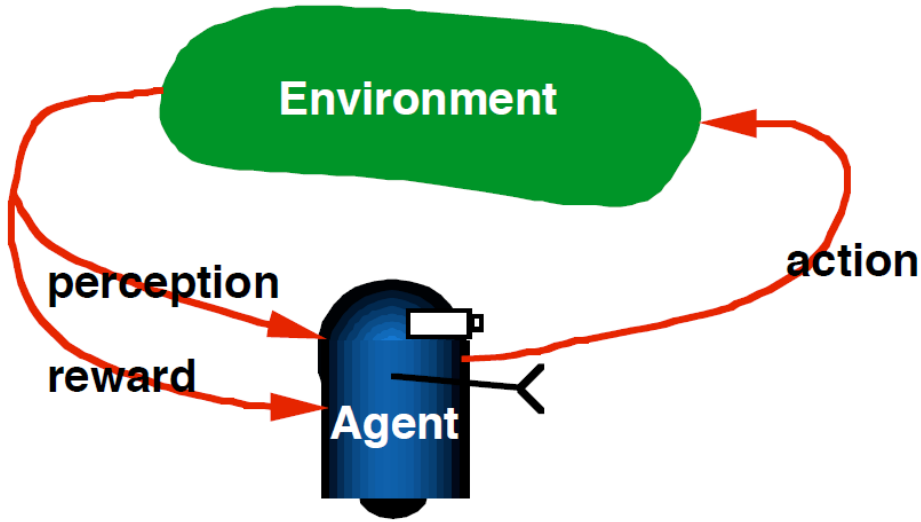
Elements of Reinforcement Learning Problem

- Agent vs. Environment.
- State, Action, Reward, Goal, Return.
- The Markov property.
- Markov decision process.
- Bellman equations.
- Optimality and Approximation.

Agent vs. Environment



Agent vs. Environment



- The learner and decision-maker is called the agent.
- The thing it interacts with, comprising everything outside the agent, is called the environment.
- The environment is formally formulated as a Markov Decision Process, which is a mathematically principled framework for sequential decision problems.

(from *Introduction to RL book*, Sutton & Barto)

The Markov property

> *A state that summarizes past sensations compactly yet in such a way that all relevant information is retained. This normally requires more than the immediate sensations, but never more than the complete history of all past sensations. A state that succeeds in retaining all relevant information is said to be Markov, or to have the Markov property.*
>
> (*Introduction to RL book*, Sutton & Barto)

The Markov property

> *A state that summarizes past sensations compactly yet in such a way that all relevant information is retained. This normally requires more than the immediate sensations, but never more than the complete history of all past sensations. A state that succeeds in retaining all relevant information is said to be Markov, or to have the Markov property.*
>
> (*Introduction to RL book*, Sutton & Barto)

- Formally,

$$Pr(s_{t+1}, r_{t+1} | s_t, a_t, r_t, \cdots, s_0, a_0, r_0) = Pr(s_{t+1}, r_{t+1} | s_t, a_t, r_t)$$

The Markov property

A state that summarizes past sensations compactly yet in such a way that all relevant information is retained. This normally requires more than the immediate sensations, but never more than the complete history of all past sensations. A state that succeeds in retaining all relevant information is said to be Markov, or to have the Markov property.

(*Introduction to RL book*, Sutton & Barto)

- Formally,

$$Pr(s_{t+1}, r_{t+1} | s_t, a_t, r_t, \cdots, s_0, a_0, r_0) = Pr(s_{t+1}, r_{t+1} | s_t, a_t, r_t)$$

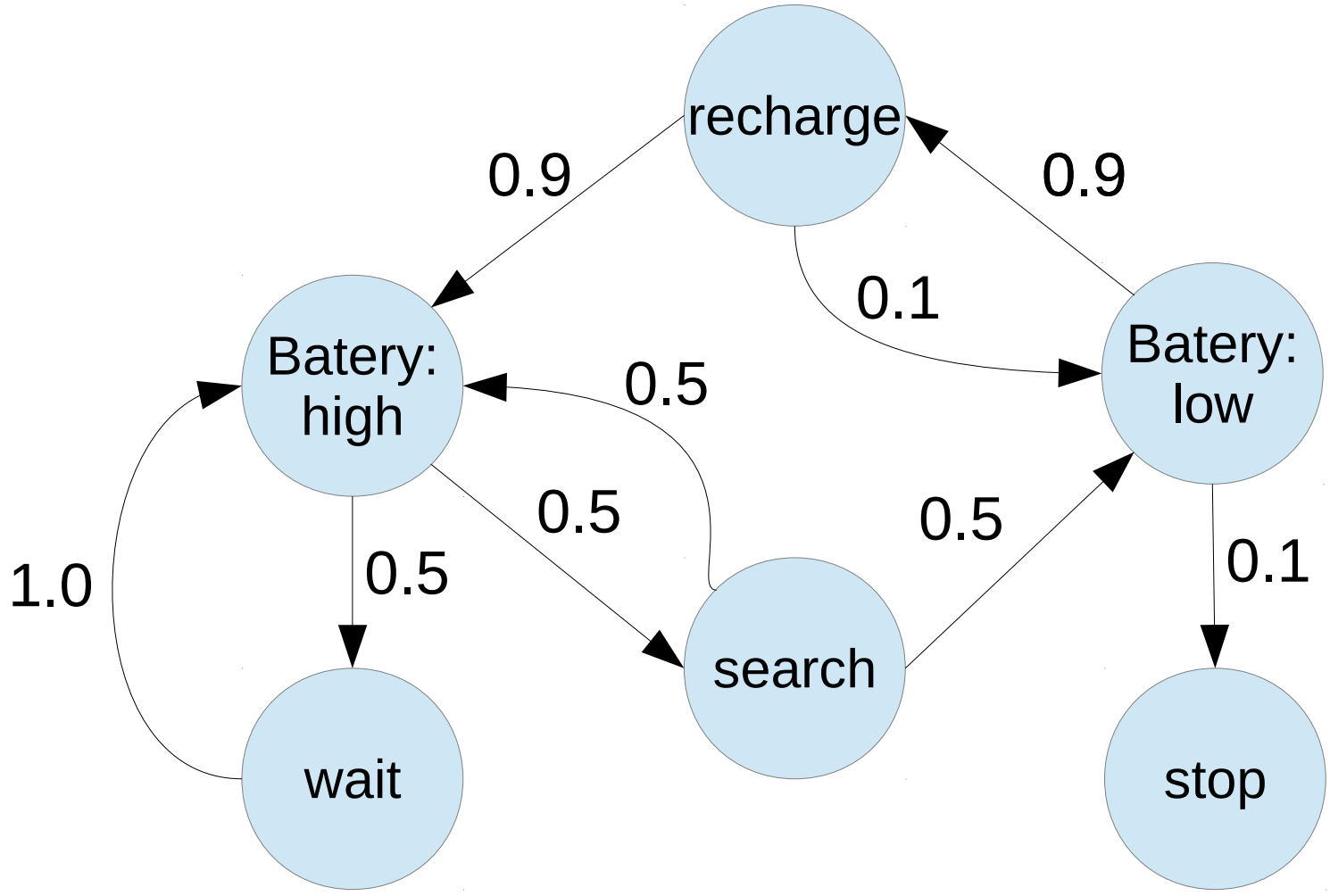
- Example: the current configuration of the chess board for predicting the next steps, the position, velocity of the cart, the angle and its changing rate of the pole in cart-pole domain.

Markov Process

- A Markov Process (Markov Chain) is defined as 2-tuple $(\mathcal{S}, \mathcal{P})$.
  – $\mathcal{S}$ is a state space.
  – $\mathcal{P}$ is a state transition probability matrix: $P_{ss'} = P(s_{t+1} = s' | s_t = s)$

Markov Process: Example

Rycycling Robot's Markov Chain

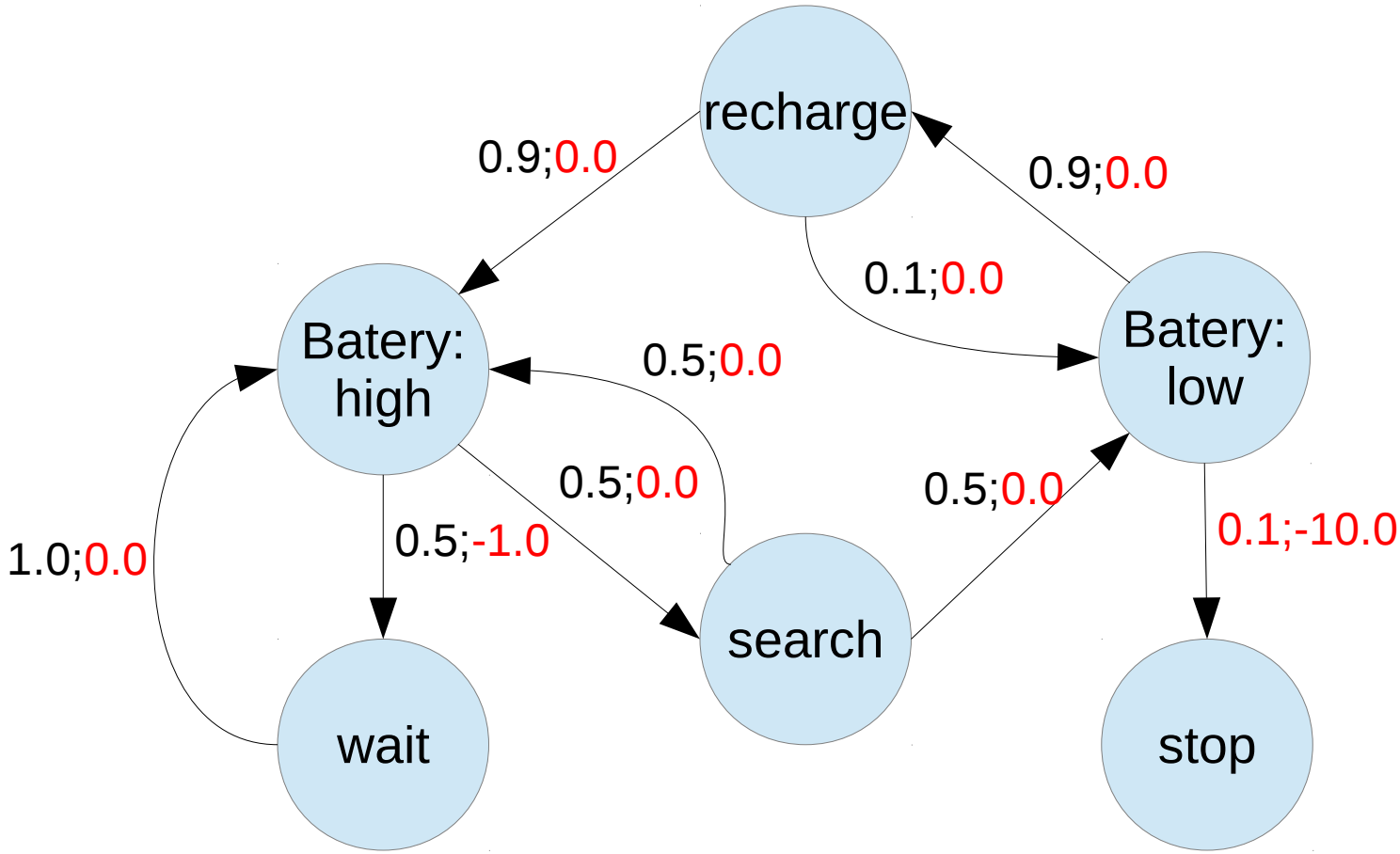


Markov Reward Process

- A Markov Reward Process is defined as 4-tuple $(\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma)$.
  - $\mathcal{S}$ is a state space of n states.
  - $\mathcal{P}$ is a state transition probability matrix: $P_{ss'} = P(s_{t+1} = s' | s_t = s)$
  - $\mathcal{R}$ is a reward matrix of R_s.
  - γ is a discount factor, $\gamma \in [0, 1]$.
- The total return is

$$\rho_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots$$

Markov Reward Process: Example



Markov Reward Process: Bellman Equations

- The value function $V(s)$

$$\begin{aligned} V(s) &= \mathbb{E}\big[\rho_t | s_t = s\big] \\ &= \mathbb{E}\big[R_t + \gamma V(s_{t+1}) | s_t = s\big] \end{aligned}$$

- $V = R + \gamma P V$, hence $V = (I - \gamma P)^{-1} R$

We will visit again in MDP.

Markov Reward Process: Discount Factor?

Many meanings:

- Weighing the importance of differently timed rewards, higher importance of more recent rewards.
- Representing uncertainty over the presence of next rewards, i.e geometric distributions.
- Representing human/animal's preference over ordering of received rewards.

Markov decision process

Markov decision process

- A reinforcement learning problem that satisfies the Markov property is called a Markov decision process, or MDP.

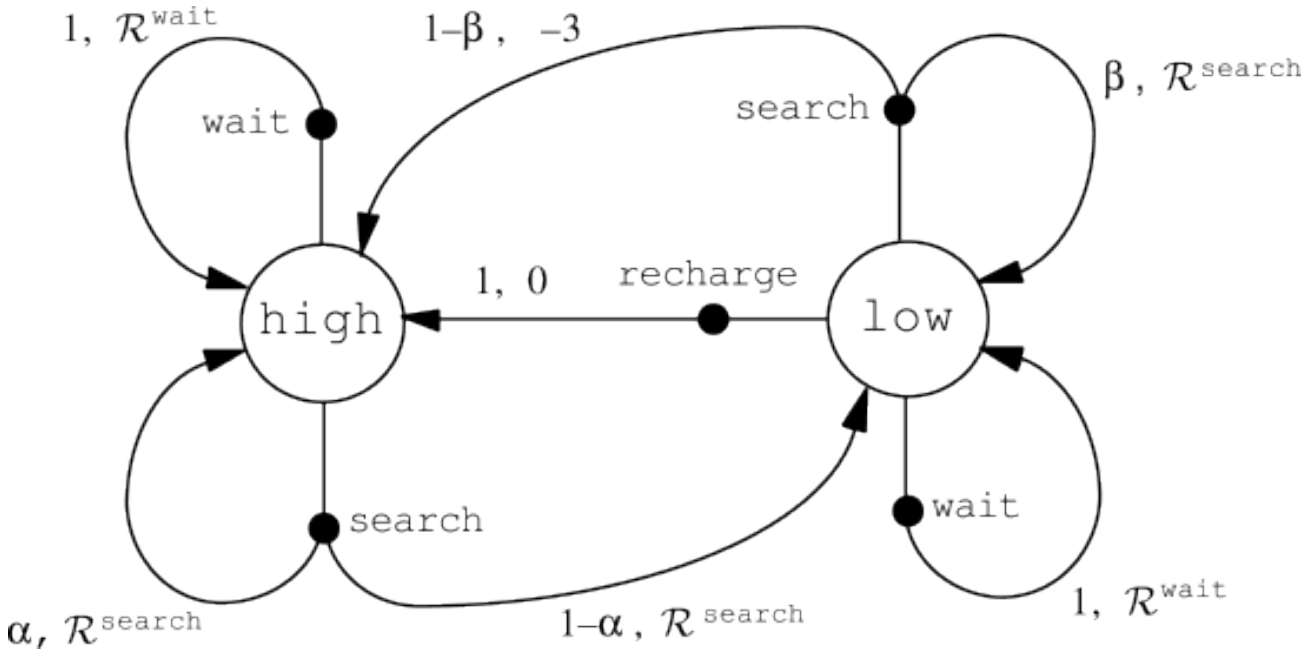
Markov decision process

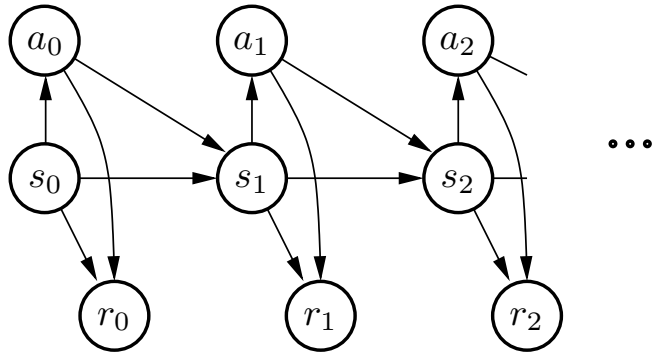
- A reinforcement learning problem that satisfies the Markov property is called a Markov decision process, or MDP.
- MDP = $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{P}_0, \gamma\}$.

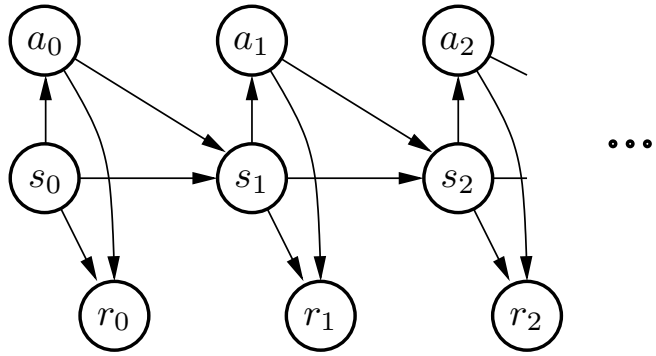
Markov decision process

- A reinforcement learning problem that satisfies the Markov property is called a Markov decision process, or MDP.
- MDP = $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{P}_0, \gamma\}$.
  – $\mathcal{S}$: consists of all possible states.
  – $\mathcal{A}$: consists of all possible actions.
  – $\mathcal{T}$: is a transition function which defines the probability $\mathcal{T}(s', s, a) = Pr(s'|s, a)$.
  – $\mathcal{R}$: is a reward function which defines the reward $\mathcal{R}(s, a)$.
  – $\mathcal{P}_0$: is the probability distribution over initial states.
  – $\gamma \in [0, 1]$: is a discount factor.

Example: Recycling Robot MDP

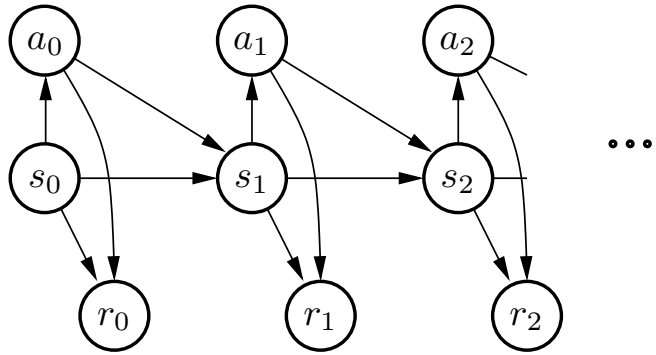



a_0
a_1
a_2
s_0
s_1
s_2
r_0
r_1
r_2
...



- A policy is a mapping from state space to action space

$$\mu : \mathcal{S} \mapsto \mathcal{A}$$



- A policy is a mapping from state space to action space

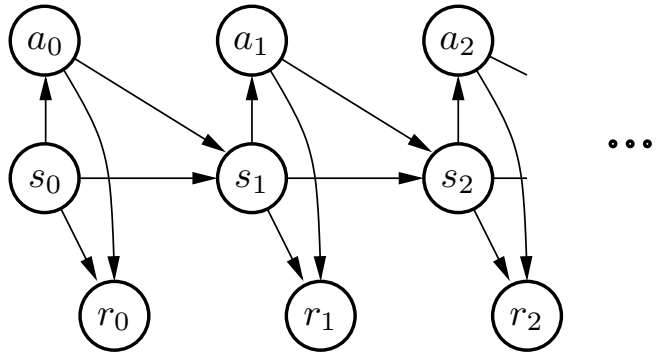
$$\mu : \mathcal{S} \mapsto \mathcal{A}$$

- Objective function:
  – Expected average reward.

$$\eta = \lim_{T \to \infty} \frac{1}{T} E\Big[\sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1}) \Big]$$

  – Expected discounted reward.

$$\eta_\gamma = E\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \Big]$$



...

- A policy is a mapping from state space to action space

$$\mu : \mathcal{S} \mapsto \mathcal{A}$$

- Objective function:
  – Expected average reward.

$$\eta = \lim_{T \to \infty} \frac{1}{T} E\Big[\sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1}) \Big]$$

  – Expected discounted reward.

$$\eta_\gamma = E\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \Big]$$

- Singh et. al. 1994:

$$\eta_\gamma = \frac{1}{1-\gamma} \eta$$

Dynamic Programming

Dynamic Programming

- State Value Functions
- Bellman Equations
- Value Iteration
- Policy Iteration

State value function

- The **value** (*expected* discounted return) of policy π when started in state s:

$$V^{\pi}(s) = \mathrm{E}_{\pi}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s\} \tag{1}$$

discounting factor $\gamma \in [0, 1]$

State value function

- The **value** (*expected* discounted return) of policy π when started in state s:

$$V^{\pi}(s) = \mathrm{E}_{\pi}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s\} \tag{1}$$

  discounting factor $\gamma \in [0, 1]$

- definition of **optimality**: *behavior* π^* *is optimal iff*

$$\forall_s : \; V^{\pi^*}(s) = V^*(s) \qquad \text{where} \;\; V^*(s) = \max_{\pi} V^{\pi}(s)$$

  (simultaneously maximising the value in all states)

  (In MDPs there always exists (at least one) optimal deterministic policy.)

Bellman optimality equation

$$
\begin{aligned}
V^{\pi}(s) &= \mathrm{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s; \pi\} \\
&= \mathrm{E}\{r_0 \mid s_0 = s; \pi\} + \gamma \mathrm{E}\{r_1 + \gamma r_2 + \cdots \mid s_0 = s; \pi\} \\
&= R(\pi(s), s) + \gamma \textstyle\sum_{s'} P(s' \mid \pi(s), s)\, \mathrm{E}\{r_1 + \gamma r_2 + \cdots \mid s_1 = s'; \pi\} \\
&= R(\pi(s), s) + \gamma \textstyle\sum_{s'} P(s' \mid \pi(s), s)\, V^{\pi}(s')
\end{aligned}
$$

Bellman optimality equation

$$
\begin{aligned}
V^\pi(s) &= \mathrm{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s; \pi\} \\
&= \mathrm{E}\{r_0 \mid s_0 = s; \pi\} + \gamma \mathrm{E}\{r_1 + \gamma r_2 + \cdots \mid s_0 = s; \pi\} \\
&= R(\pi(s), s) + \gamma \textstyle\sum_{s'} P(s' \mid \pi(s), s)\ \mathrm{E}\{r_1 + \gamma r_2 + \cdots \mid s_1 = s'; \pi\} \\
&= R(\pi(s), s) + \gamma \textstyle\sum_{s'} P(s' \mid \pi(s), s)\ V^\pi(s')
\end{aligned}
$$

- We can write this in vector notation $\boldsymbol{V}^\pi = \boldsymbol{R}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{V}^\pi$ with vectors $\boldsymbol{V}^\pi_s = V^\pi(s)$, $\boldsymbol{R}^\pi_s = R(\pi(s), s)$ and matrix $\boldsymbol{P}^\pi_{s's} = P(s' \mid \pi(s), s)$
- For stochastic $\pi(a|s)$: $V^\pi(s) = \sum_a \pi(a|s) R(a, s) + \gamma \sum_{s',a} \pi(a|s) P(s' \mid a, s)\ V^\pi(s')$

Bellman optimality equation

$$
\begin{aligned}
V^\pi(s) &= \mathrm{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s; \pi\} \\
&= \mathrm{E}\{r_0 \mid s_0 = s; \pi\} + \gamma \mathrm{E}\{r_1 + \gamma r_2 + \cdots \mid s_0 = s; \pi\} \\
&= R(\pi(s), s) + \gamma \textstyle\sum_{s'} P(s' \mid \pi(s), s)\ \mathrm{E}\{r_1 + \gamma r_2 + \cdots \mid s_1 = s'; \pi\} \\
&= R(\pi(s), s) + \gamma \textstyle\sum_{s'} P(s' \mid \pi(s), s)\ V^\pi(s')
\end{aligned}
$$

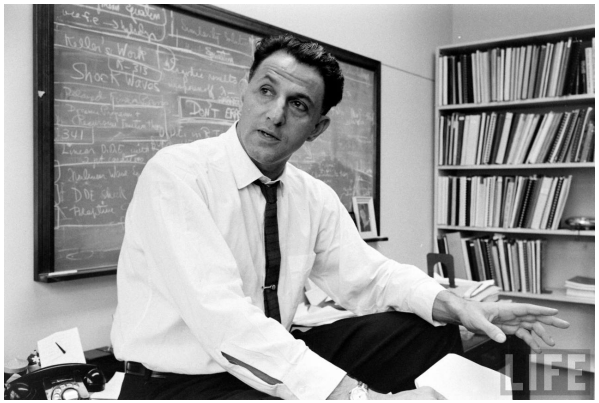
- We can write this in vector notation $\boldsymbol{V}^\pi = \boldsymbol{R}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{V}^\pi$
  with vectors $\boldsymbol{V}^\pi_s = V^\pi(s)$, $\boldsymbol{R}^\pi_s = R(\pi(s), s)$ and matrix $\boldsymbol{P}^\pi_{s's} = P(s' \mid \pi(s), s)$
- For stochastic $\pi(a|s)$: $V^\pi(s) = \sum_a \pi(a|s) R(a, s) + \gamma \sum_{s',a} \pi(a|s) P(s' \mid a, s)\ V^\pi(s')$

- Bellman optimality equation

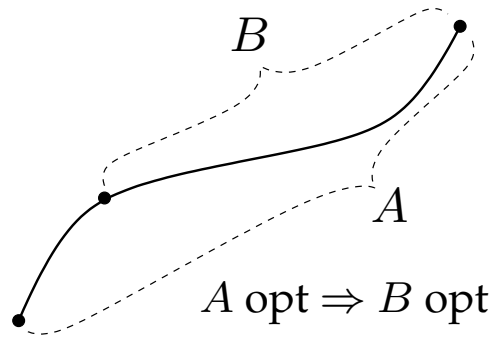
$$
\begin{aligned}
V^*(s) &= \max_a \Big[R(a, s) + \gamma \textstyle\sum_{s'} P(s' \mid a, s)\ V^*(s') \Big] \\
\pi^*(s) &= \operatorname*{argmax}_a \Big[R(a, s) + \gamma \textstyle\sum_{s'} P(s' \mid a, s)\ V^*(s') \Big]
\end{aligned}
$$

  (Sketch of proof: If π would select another action than $\operatorname{argmax}_a[\cdot]$, then π' which $= \pi$ everywhere except $\pi'(s) = \operatorname{argmax}_a[\cdot]$ would be better.)
- This is the **principle of optimality** in the stochastic case (related to Viterbi, max-product algorithm)

Richard E. Bellman (1920-1984)



Bellman's principle of optimality



$$V^*(s) = \max_a \left[R(a,s) + \gamma \sum_{s'} P(s' \mid a, s)\, V^*(s') \right]$$

$$\pi^*(s) = \operatorname*{argmax}_a \left[R(a,s) + \gamma \sum_{s'} P(s' \mid a, s)\, V^*(s') \right]$$

Value Iteration

- Given the Bellman equation

$$V^*(s) = \max_a \left[R(a,s) + \gamma \sum_{s'} P(s' \,|\, a, s)\; V^*(s') \right]$$

  $\rightarrow$ iterate

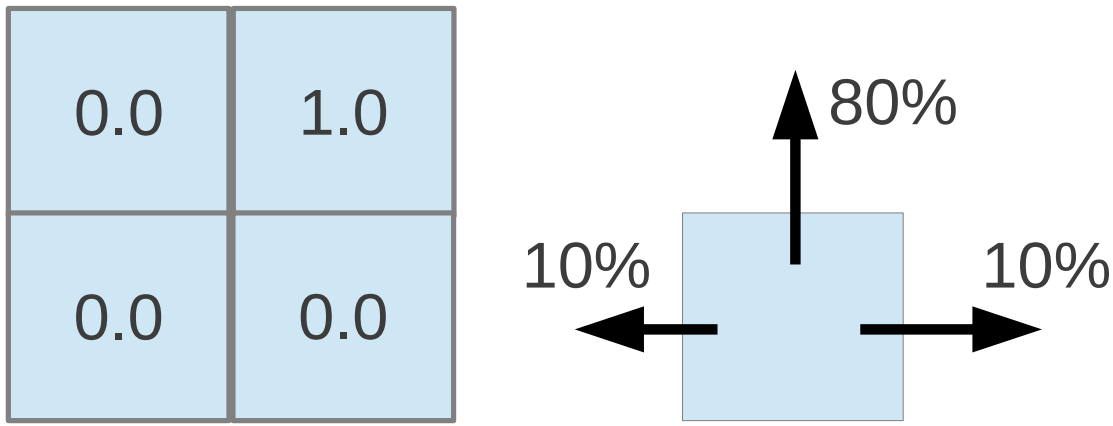
$$\forall_s :\; V_{k+1}(s) = \max_a \left[R(a,s) + \gamma \sum_{s'} P(s'|\pi(s), s)\; V_k(s') \right]$$

  stopping criterion:

$$\max_s |V_{k+1}(s) - V_k(s)| \leq \epsilon$$

- Value Iteration converges to the optimal value function V^* (proof below)

2x2 Maze



manually solving.

State-action value function (Q-function)

- The *state-action value function* (or **Q-function**) is the *expected* discounted return when starting in state s and taking first action a:

$$\begin{aligned} Q^\pi(a,s) &= \mathrm{E}_\pi\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, a_0 = a\} \\ &= R(a,s) + \gamma \sum_{s'} P(s' \mid a, s)\, Q^\pi(\pi(s'), s') \end{aligned}$$

  (Note: $V^\pi(s) = Q^\pi(\pi(s), s)$.)

- Bellman optimality equation for the Q-function

$$\begin{aligned} Q^*(a,s) &= R(a,s) + \gamma \sum_{s'} P(s' \mid a, s) \max_{a'} Q^*(a', s') \\ \pi^*(s) &= \operatorname*{argmax}_a Q^*(a,s) \end{aligned}$$

Q-Iteration

- Given the Bellman equation

$$Q^*(a,s) = R(a,s) + \gamma \sum_{s'} P(s' \,|\, a,s) \;\max_{a'} Q^*(a',s')$$

$\rightarrow$ iterate

$$\forall_{a,s} :\; Q_{k+1}(a,s) = R(a,s) + \gamma \sum_{s'} P(s'|a,s) \;\max_{a'} Q_k(a',s')$$

stopping criterion:

$$\max_{a,s} |Q_{k+1}(a,s) - Q_k(a,s)| \leq \epsilon$$

- Q-Iteration converges to the optimal state-action value function Q^*

Proof of convergence

- Let $\Delta_k = ||Q^* - Q_k||_\infty = \max_{a,s} |Q^*(a,s) - Q_k(a,s)|$

$$
\begin{aligned}
Q_{k+1}(a,s) &= R(a,s) + \gamma \sum_{s'} P(s'|a,s) \ \max_{a'} Q_k(a',s') \\
&\leq R(a,s) + \gamma \sum_{s'} P(s'|a,s) \ \max_{a'} \Big[Q^*(a',s') + \Delta_k \Big] \\
&= \Big[R(a,s) + \gamma \sum_{s'} P(s'|a,s) \ \max_{a'} Q^*(a',s') \Big] + \gamma \Delta_k \\
&= Q^*(a,s) + \gamma \Delta_k
\end{aligned}
$$

similarly: $Q_k \geq Q^* - \Delta_k \ \Rightarrow \ Q_{k+1} \geq Q^* - \gamma \Delta_k$

Convergence

- Contraction property: $||U_{k+1} - V_{k+1}|| \leq \gamma ||U_k - V_k||$
  which guarantees convergence with different initial values U_0, V_0 of two approximations.

$$||U_{k+1} - V_{k+1}|| \leq \gamma ||U_k - V_k|| \leq \ldots \leq \gamma^{k+1} ||U_0 - V_0||$$

Convergence

- Contraction property: $||U_{k+1} - V_{k+1}|| \leq \gamma ||U_k - V_k||$
  which guarantees convergence with different initial values U_0, V_0 of two approximations.

$$||U_{k+1} - V_{k+1}|| \leq \gamma ||U_k - V_k|| \leq \ldots \leq \gamma^{k+1} ||U_0 - V_0||$$

- Stopping condition: $||V_{k+1} - V_k|| \leq \epsilon \Rightarrow ||V_{k+1} - V^*|| \leq \epsilon\gamma/(1-\gamma)$

Convergence

- Contraction property: $||U_{k+1} - V_{k+1}|| \leq \gamma ||U_k - V_k||$
  which guarantees convergence with different initial values U_0, V_0 of two approximations.

$$||U_{k+1} - V_{k+1}|| \leq \gamma ||U_k - V_k|| \leq \ldots \leq \gamma^{k+1} ||U_0 - V_0||$$

- Stopping condition: $||V_{k+1} - V_k|| \leq \epsilon \Rightarrow ||V_{k+1} - V^*|| \leq \epsilon\gamma/(1-\gamma)$
  Proof:

$$\frac{|\, V_{k+1} - V^* \,|}{\gamma} \leq |\, V_k - V^* \,| \leq |\, V_{k+1} - V_k \,| + |\, V_{k+1} - V^* \,|$$

$$\frac{|\, V_{k+1} - V^* \,|}{\gamma} \leq \epsilon + |\, V_{k+1} - V^* \,|$$

Policy Evaluation

Value Iteration and Q-Iteration compute directly V^* and Q^*

If we want to evaluate a given policy π, we want to compute V^π or Q^π:

Policy Evaluation

Value Iteration and Q-Iteration compute directly V^* and Q^*
If we want to evaluate a given policy π, we want to compute V^π or Q^π:

- Iterate using π instead of $\max_a$:

$$\forall_s : \; V_{k+1}(s) = R(\pi(s), s) + \gamma \sum_{s'} P(s'|\pi(s), s) \; V_k(s')$$

$$\forall_{a,s} : \; Q_{k+1}(a, s) = R(a, s) + \gamma \sum_{s'} P(s'|a, s) \; Q_k(\pi(s'), s')$$

Policy Evaluation

Value Iteration and Q-Iteration compute directly V^* and Q^*
If we want to evaluate a given policy π, we want to compute V^π or Q^π:

- Iterate using π instead of $\max_a$:

$$\forall_s : \; V_{k+1}(s) = R(\pi(s), s) + \gamma \sum_{s'} P(s'|\pi(s), s) \; V_k(s')$$
$$\forall_{a,s} : \; Q_{k+1}(a, s) = R(a, s) + \gamma \sum_{s'} P(s'|a, s) \; Q_k(\pi(s'), s')$$

- Or, invert the matrix equation

$$\begin{aligned} \boldsymbol{V}^\pi &= \boldsymbol{R}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{V}^\pi \\ \boldsymbol{V}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{V}^\pi &= \boldsymbol{R}^\pi \\ (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi) \boldsymbol{V}^\pi &= \boldsymbol{R}^\pi \\ \boldsymbol{V}^\pi &= (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi)^{-1} \boldsymbol{R}^\pi \end{aligned}$$

requires inversion of $n \times n$ matrix for $|S| = n$, $O(n^3)$

Policy Iteration

- What does it help to just compute V^π or Q^π to find the optimal policy?

Policy Iteration

- What does it help to just compute V^π or Q^π to find the optimal policy?

- **Policy Iteration**
  1. Initialise π_0 somehow (e.g. randomly)
  2. Iterate
     – **Policy Evaluation:** compute V^{π_k} or Q^{π_k}
     – **Policy Improvement:** $\pi_{k+1}(s) \leftarrow \mathrm{argmax}_a \, Q^{\pi_k}(a, s)$

`demo: 2x2 maze`

Convergence proof

The fact is that:

- After policy improvement: $V^{\pi_k} \leq V^{\pi_{k+1}}$ (with a sketch proof from Rich Sutton's book)
- The policy space is finite, $|\mathcal{A}|^{|\mathcal{S}|}$.
- The Bellman operator has a unique fixed point (due to the **strict** contraction property ($0 < \gamma < 1$) on a Banach space). This condition is also used to prove the fixed point for the VI algorithm.

VI vs. PI

- VI is PI with one step of policy evaluation.
- PI converges surprisingly rapildy, however with expensive compution, i.e. the policy evaluation step (wait for convergence of V^{π}).
- PI is prefered if the action set is large.

Asynchronous Dynamic Programming

- The value function table is updated asynchronously.
- Computation is significantly reduced.
- If all states are updated infinitely, convergence is still guaranteed.
- Three simple algorithms:
  - Gauss-Seidel Value Iteration
  - Real-time dynamic programming
  - Prioritised sweeping

Gauss-Seidel Value Iteration

- Standard VI algorithm updates all states at next iteration using **old** values at previous iteration (each iteration finishes when all states get updated).

Algorithm 1 Standard Value Iteration Algorithm

1: **while** (!converged) **do**
2: $V_{old} = V$
3: **for** (each $s \in \mathcal{S}$) **do**
4: $V(s) = \max_a \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_{old}(s')\}$

- Gauss-Seidel VI updates each state using values from previous computation.

Algorithm 2 Gauss-Seidel Value Iteration Algorithm

1: **while** (!converged) **do**
2: **for** (each $s \in \mathcal{S}$) **do**
3: $V(s) = \max_a \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')\}$

Prioritised Sweeping

- Similar to Gauss-Seidel VI, but the sequence of states in each iteration is proportional to their update magnitudes (Bellman errors).
- Define Bellman error as $E(s; V_t) = |V_{t+1}(s) - V_t(s)|$ that is the change of s's value after the most recent update.

Algorithm 3 Prioritised Sweeping VI Algorithm

1: Initialize $V_0(s)$ and priority values $H_0(s)$, $\forall s \in \mathcal{S}$.
2: **for** $k = 0, 1, 2, 3, \ldots$ **do**
3: pick a state to update (with the highest priortiy): $s_k \in \arg\max_{s \in \mathcal{S}} H_k(s)$
4: value update: $V_{k+1}(s_k) = \max_{a \in \mathcal{A}} \left[R(s_k, a_k) + \gamma \sum_{s'} P(s'|s_k, a_k) V_k(s') \right]$
5: for $s \neq s_k$: $V_{k+1}(s) = V_k(s)$
6: update priority values: $\forall s \in \mathcal{S}, H_{k+1}(s) \leftarrow E(s; V_{k+1})$ (Note: the error is w.r.t the future update).

Real-Time Dynamic Programming

- Similar to Gauss-Seidel VI, but the sequence of states in each iteration is generated by **simulating** the transitions.

Algorithm 4 Real-Time Value Iteration Algorithm

1: start at an arbitray s_0, and initialize $V_0(s), \forall s \in calS$.
2: **for** $k = 0, 1, 2, 3, \ldots$ **do**
3: action selection:

$$a_k \in \arg\max_{a \in \mathcal{A}} \left\{ R(s_k, a) + \gamma \sum_{s'} P(s'|s_k, a) V_k(s') \right\}$$

4: value update: $V_{k+1}(s_k) = R(s_k, a_k) + \gamma \sum_{s'} P(s'|s_k, a_k) V_k(s')$
5: For $s \neq s_k$: $V_{k+1}(s) = V_k(s)$
6: simulate the next state: $s_{k+1} \sim P(s'|s_k, a_k)$

- So far, we introduce basic notions of an MDP and value functions and methods to compute optimal policies **assuming that we know the world** (know $P(s'|a,s)$ and $R(a,s)$):

  – Value Iteration/Q-Iteration $\rightarrow V^*, Q^*, \pi^*$
  – Policy Evaluation $\rightarrow V^\pi, Q^\pi$
  – Policy Improvement $\pi(s) \leftarrow \operatorname{argmax}_a Q^{\pi_k}(a,s)$
  – Policy Iteration (iterate Policy Evaluation and Policy Improvement)

- *Reinforcement Learning?*