Credit assignment problem

Imagine playing a game such as chess:

- Players make their moves with only partial knowledge of their consequences.
- Target value is assigned only when the game is finished.

Imagine a cleaning robot moving in a space that needs to

- systematically clean the space,
- occasionally recharge batteries.

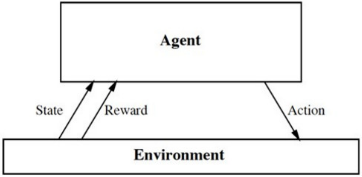
Issues:

- How does the robot evaluate quality of its moves?
- How does it decide where to go?

Reinforcement learning: learn gradually from experience

Reinforcement learning – overview

- Supervised learning: Immediate feedback
- Unsupervised learning: No feedback
- **Reinforcement learning**: Delayed feedback



The model:

- Agents that sense & act upon their environment

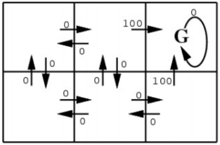
Applications:

- Cleaning robots
- Investments strategies
- Game playing
- Scheduling
- Verification
- ...

A concrete example

Robot grid world

- 6 states, arrows are possible actions
- Robot gets a reward for performing actions
- ... so eventually gets a sequence of rewards $r_1, r_2, \ldots$
  ... maximizes the discounted reward $r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots$
  here $0 < \gamma < 1$ is a *discount factor*



The goal is to find an optimal *policy* which chooses an appropriate action in each state.

Deterministic Markov decision processes

A *deterministic Markov decision process (DMDP)* consists of

- a set of states S
- a set of actions A
- a transition function $\delta : S \times A \to S$
- a reward function $r : S \times A \to \mathbb{R}$

Semantics: Assuming that the current state is $s \in S$, the agent chooses an action a, receives the reward $r(s, a)$, and then moves on to the state $\delta(s, a)$.

A *policy* is a function $\pi : S \to A$ which prescribes how to choose actions in states.

Following a given policy π starting in a state s, the agent collects a sequence of rewards $r^{\pi,s} = r_1^{\pi,s}, r_2^{\pi,s}, \ldots$.
Here r_i^{π} is the reward collected in the i-th step.

Deterministic Markov decision processes

How to specify the overall quality of a policy?

Given a policy π and a state s, denote by $V^{\pi}(s)$

the *discounted reward* $r_1^{\pi,s} + \gamma r_2^{\pi,s} + \gamma^2 r_3^{\pi,s} + \cdots$

Here $0 < \gamma < 1$ is a *discount factor* that specifies how the agent "sees" the future.

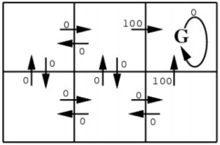
Our goal: Find π which belongs to $\arg\max_{\pi'} V^{\pi'}(s)$ for all $s \in S$.

For the rest we **fix a discount factor** $0 < \gamma < 1$.

Example

Consider the policies:

- π_1: always go down and right
- π_2: whiteboard ...



Let s be the left-most down-most state.

What is $V^{\pi_1}(s)$?

What is $V^{\pi_2}(s)$?

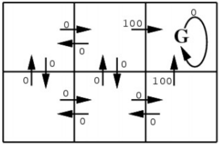
In both cases consider $\gamma = 0.5$.

The Value

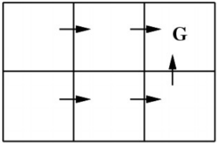
Define the *optimal policy* π^* by

$$\pi^* \in \arg\max_{\pi} V^{\pi}(s) \text{ for all } s \in S$$

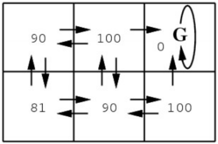
We use $V^*(s)$ to denote $V^{\pi^*}(s)$.



Compute $V^*(s)$ for all six states if the discount factor $\gamma = 0.9$.



One optimal policy



$V^*(s)$ values

Maximizing the value

The value V^* can be expressed using the following recurrent equations:

$$
\begin{aligned}
V^*(s) &= \max_\pi r^{\pi,s} \\
&= \max_\pi \left(r_1^{\pi,s} + \gamma r_2^{\pi,s} + \gamma^2 r_3^{\pi,s} \ldots\right) \\
&= \max_\pi \max_{\pi'} \left(r_1^{\pi,s} + \gamma r_2^{\pi',s} + \gamma^2 r_3^{\pi',s} \ldots\right) \\
&= \max_a \max_{\pi'} \left(r(s,a) + \gamma r^{\pi',\delta(s,a)}\right) \\
&= \max_a \left(r(s,a) + \gamma \max_{\pi'} r^{\pi',\delta(s,a)}\right) \\
&= \max_a \left(r(s,a) + \gamma V^*(\delta(s,a))\right)
\end{aligned}
$$

Value iteration algorithm:

- Initialize $V_0^*(s) = 0$ for every s.
- Compute $V_{i+1}^*(s)$ from V_i^* by

$$V_{i+1}^* = \max_a \left(r(s,a) + \gamma V_i^*(\delta(s,a))\right)$$

Generalization: Markov Decision Processes

Often the transitions function δ as well as rewards r are not deterministic.

If we can estimate the probability of outcomes, we may use $\delta : S \times A \to \mathcal{D}(S)$ where $\mathcal{D}(S)$ is the set of all probability distributions on S.

For example: $\delta(s, a)(s') = 1/2$ and $\delta(s, a)(s'') = 1/2$ means that if the agent chooses a in s, then it proceeds randomly either to s' or to s''.

Similarly, $r : S \times A \to \mathcal{D}(R)$ where R is a "reasonable" subset of $\mathbb{R}$.

Now the sequence of rewards is not determined by a policy, only a *distribution* on sequences of rewards.

The goal is to maximize the *expected discounted reward*.
The previous recursive equations can be generalized to MDPs.

Q-learning

The recursive equations can be used to compute optimal policies if δ and r are known to the agent. But what if they are not?

Assume that the agent only observes:

- The current state
- The current reward
- The set of available actions

The idea: Try to learn an approximation of the function $Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$

Apparently, when we have $Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$, then $V^*(s) = \max_{a'} Q(s, a')$ and thus we have $V^*(s)$.

But how to approximate $Q(s, a)$ when the agent observes only a local state and reward?

Q-learning algorithm

Denote by $\hat{Q}$ the successive approximations of Q.

- Initialize $\hat{Q}(s, a) = 0$

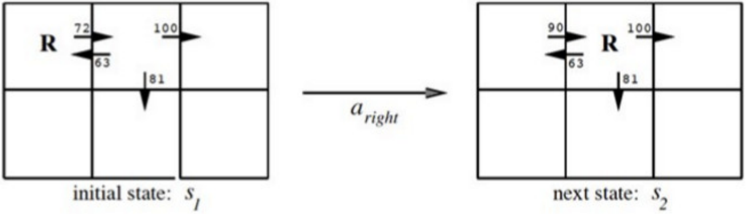
  (i.e. whenever a new state-action pair is encountered, the algorithm assumes that $\hat{Q}(s, a) = 0$)
- Observe the current state s
- Do forever:
  - Select an action a and execute it
  - Receive an immediate reward r
  - Observe the new state s'
  - Update the value of $\hat{Q}(s, a)$ by

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

  - $s = s'$

Under some technical conditions, $\hat{Q}$ converges to Q.

Q-learning example



$$
\begin{aligned}
\hat{Q}(s_1, a_{right}) &= r + \gamma \max_{a'} \hat{Q}(s_2, a') \\
&= 0 + 0.9 \max\{63, 81, 100\} \\
&= 90
\end{aligned}
$$

Exploration vs exploitation

In the Q-learning algorithm, we have not specified how to choose an action in each iteration.

Possible approaches:

1. maximize $\hat{Q}(s, a)$
2. give each action equal opportunity
3. choose randomly with a probability $k^{\hat{Q}(s,a)} / \sum_{a'} k^{\hat{Q}(s,a')}$ where $k > 1$

ad 1. exploits the values computes so far and thus improves the policy currently expressed by $\hat{Q}$

ad 2. explores the space while ignoring $\hat{Q}$

ad 3. combines exploration & exploitation: if $\hat{Q}(s, a) >> \max_{a' \neq a} \hat{Q}(s, a')$, the action a is chosen most of the time but *not always*

Representation of $\hat{Q}$

The $\hat{Q}$ can be represented by various means

- neural networks – Deep Q-learning
- decision trees
- SVM ...