Reinforcement Learning for Autonomous Systems Design

Introduction to Reinforcement Learning

Mirco Musolesi

mircomusolesi@acm.org

Introduction to Reinforcement Learning

- Key idea: a natural way of thinking about learning is learning through interaction with the external world.

- Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence.

- Reinforcement learning is learning what to do - how to map situations to actions - so as to maximise a numerical reward.

  - *Goal-directed* learning from interaction.

- The learner is not told which actions to take, but instead it must discover which actions yield the most reward by trying them.

Examples of Problems

Credit: DepositPhoto

Examples of Problems

Finite Markov Decision Processes

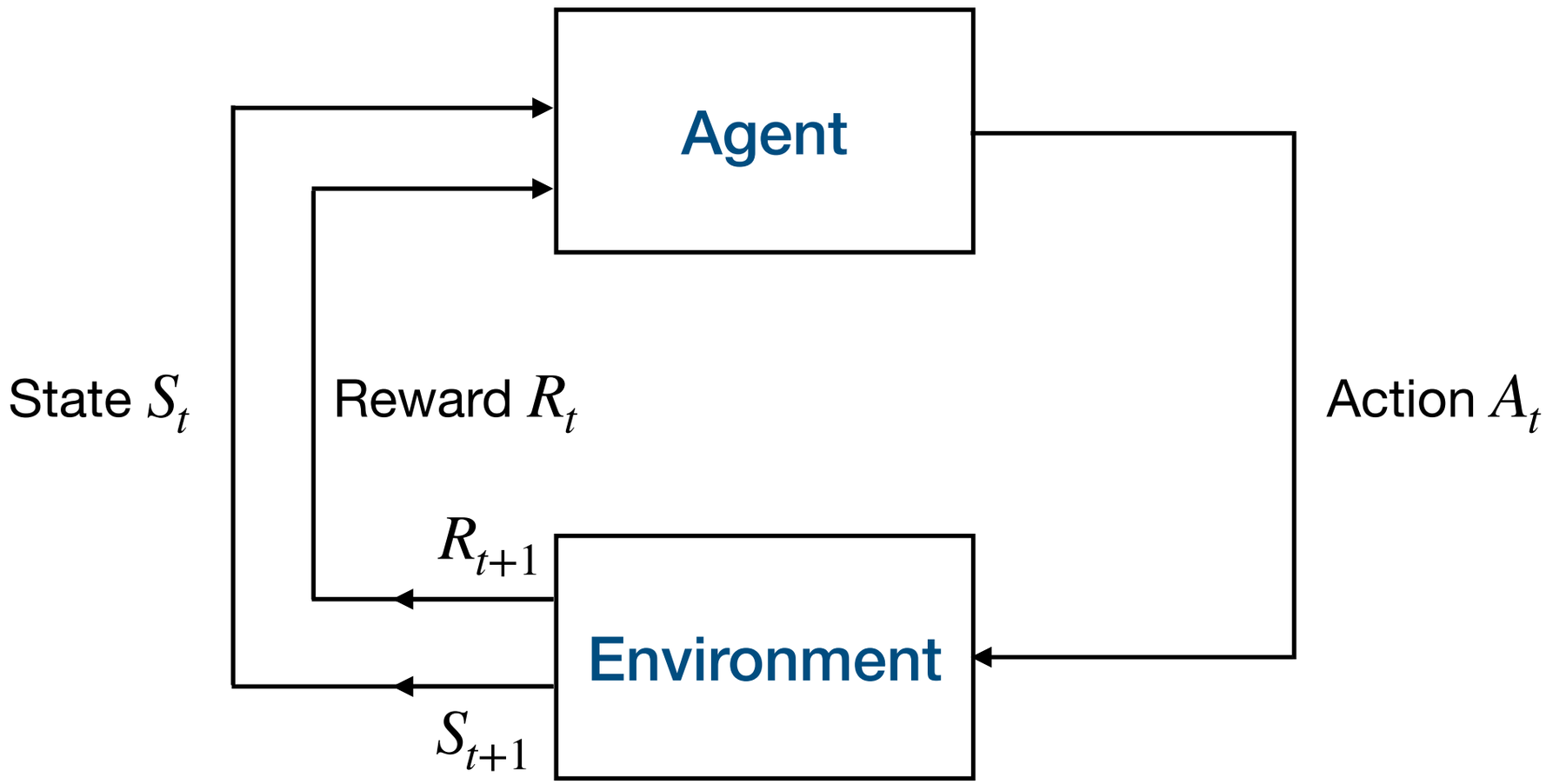
- Markov Decision Processes (MDPs) are a mathematically idealised formulation of Reinforcement Learning for which precise theoretical statements can be made.

- Tension between breadth of applicability and mathematical tractability.

- MDPs provide a way for framing the problem of learning from experience, and, more specifically, from interacting with an environment.

Markov Decision Processes: Definitions

- Two entities:
  - **Agent**: learner and decision maker.
  - **Environment**: everything else outside the agent.
- The agent interacts with the environment selecting **actions**.
- The environment changes following actions of the agent.


Agent
State S_t
Reward R_t
Action A_t
R_{t+1}
Environment
S_{t+1}

Markov Decision Processes: Definitions

- The agent and the environment interact at each of a sequence of discrete time steps $t = 0,1,2,3,...$

- At each time step t, the agent receives some representation of the environment **state** $S_t \in \mathcal{S}$ where $\mathcal{S}$ is the set of the states.

- On that basis, an agent selects an **action** $A_t \in \mathcal{A}(S_t)$ where $\mathcal{A}(S_t)$ is the set of the actions that can be taken in state S_t.

- At time $t+1$ as a consequence of its action the agent receives a **reward** $R_{t+1} \in \mathcal{R}$, where $\mathcal{R}$ is the set of rewards (expressed as real numbers).

Goals and Rewards

- The goal of the agent is formalised in terms of the reward it receives.
- At each time step, the reward is a simple number $R_t \in \mathbb{R}$.
- Informally, the agent's goal is to maximise the total amount it receives.
- The agent should not maximise the immediate reward, but the *cumulative reward*.

The “Reward Hypothesis”

- We can formalise the goal of an agent by stating the “reward hypothesis”:

All of what we mean by goals and purposes can be well thought of as the maximisation of the expected value of the cumulative sum of a received scalar signal (reward).

Expected Returns

- We will now try to conceptualise the idea of **cumulative rewards** more formally.

- An agent receives a sequence of rewards $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$

- In order to define cumulative rewards, we introduce the concept of **expected return** G_t, which is a function of the reward sequence.

Episodic Tasks and Continuing Tasks

- Typically, we identify two cases: episodic tasks and continuing tasks.

- An **episodic task** is one in which we can identify a final step of the sequence of rewards , i.e., in which the interaction between the agent and the environment can be broken into sub-sequences that we call **episodes** (such a play of a game, repeated tasks, etc.).

  - Each episode ends in terminal state after T steps, followed by a reset to a standard starting state or to a sample of a distribution of starting states.

  - The next episode is completely independent from the previous one.

- A **continuing task** is one in which it is not possible to identify a final state (e.g., on-going process control or robots with a long-lifespan).

Expected Return for Episodic Tasks and Continuing Tasks

- In the case of *episodic tasks* the expected return associated to the selection of an action A_t is the sum of rewards defined as follows:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$$

- In the case of *continuing tasks* the expected return associated to the selection of an action A_t is defined as follows:

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

where γ is the discount rate, with $0 \leq \gamma \leq 1$.

Why Discounting?

- The definition of expected return that we used for episodic tasks would be problematic for continuing tasks: the expected return of time of termination T would be equal to ∞ in some cases, such as when the reward is equal to 1 at each time step.

- The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth γ^{k-1} what it would be worth if it were received immediately.

Relation between Returns at Successive Time Steps

- Returns at successive time steps are related to each others as follows:

$$
\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\
&= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Policies and Value Functions

- Almost all reinforcement learning algorithms involve estimating value functions, i.e., functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state).

- A policy is used to to model how the behaviour of the agent based on the previous experience and the rewards (and consequently the expected returns) an agent received in the past.

Definition of Policy

- Formally, a *policy* is a mapping from states to probabilities of each possible action.

- If the agent is following policy π at time t, then $\pi(a \mid s)$ is the probability that $A_t = a$ if $S_t = s$.

Definition of State-Value Function

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

- For MDPs, we can define the *state-value function* v_π for policy π formally as:

$$v_s \doteq E_\pi[G_t \,|\, S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a]$$

for all $s \in \mathcal{S}$

where $E_\pi[\,.\,]$ denotes the expected value of a random variable given that the agent follows π and t is any time step. The value of the terminal state is 0.

Definition of Action-Value Function

- Similarly, we define the *action-value function*, i.e., the value of taking action a in state s under a policy π, denoted $q_\pi(s, a)$, as the expected return starting from s, taking the action a, and thereafter following policy π:

$$q_\pi(s, a) \doteq E_\pi[G_t \,|\, S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a]$$

Choosing the Rewards

- When we model a real system as a Reinforcement Learning problem, the hardest problem is to select the right rewards.

- Typically, we use negative values for actions that do not help us in reaching our goal and positive if they do (and sometimes we set the values to 0 if they do not help us in reaching the goal).

- An alternative is to set the values of rewards to a negative number until we reach our goal (and we set the value to 0 when we reach our goal).

Choosing the Rewards

- It is very important to keep in mind that we should not “reward” the intermediate steps or the single actions.

- We are not “teaching” the agent how to execute an intermediate step, but how to reach the final goal. If we do so, the agent will learn how to reach the intermediate step, e.g., how to execute a sub-task.

- The reward should tell the agent if the current step is a step forward towards the final goal or not.

Example of Rewards

Credit: Shutterstock

Maze -> Rewards: -1 for no exit 0 for exit

Examples of Rewards

Chess -> Rewards: 1 for victory, -1 for defeat

Choosing the Rewards

- Sometimes it's not possible to know the reward until the end of an episode. The typical example is a board game (chess, go, etc.).

- This is usually called *credit assignment problem,* i.e., the problem of assigning a reward to each step.

- In that case the reward might be assigned at the end of a Montecarlo rollout for example (stochastic estimate of the reward).

- For example if the game is successful we can use +1 as reward for all the steps that leads to the victory (or -1 otherwise).

Steps Toward Artificial Intelligence*

MARVIN MINSKY†, MEMBER, IRE

The work toward attaining "artificial intelligence" is the center of considerable computer research, design, and application. The field is in its starting transient, characterized by many varied and independent efforts. Marvin Minsky has been requested to draw this work together into a coherent summary, supplement it with appropriate explanatory or theoretical noncomputer information, and introduce his assessment of the state-of-the-art. This paper emphasizes the class of activities in which a general purpose computer, complete with a library of basic programs, is further programmed to perform operations leading to ever higher-level information processing functions such as learning and problem solving. This informative article will be of real interest to both the general PROCEEDINGS reader and the computer specialist.—*The Guest Editor*

***Summary*—The problems of heuristic programming—of making computers solve really difficult problems—are divided into five main areas: Search, Pattern-Recognition, Learning, Planning, and Induction.**

find only a few machines (mostly "general-purpose" computers, programmed for the moment to behave according to some specification) doing things that might

Marvin Minsky. Steps Toward Artificial Intelligence. Proceedings of the IRE. Volume 49. Issue 1. January 1961.

Example of Rewards

- In Go or Chess, the reward will be 1 for winning or -1 losing for the terminal state (i.e., the state at time T), but we will know the result of the game only at the end.

- Therefore, the reward can be assigned only at the end of an episode.

- In Go or Chess, we can for example assign 1 or -1 to each step in case of victory or loss at the end of the episode after a Montecarlo playout/rollout.

How to Estimate the State-Value (Action-Value) Functions

- The state-value v_π and the action-value function q_π can be estimated for experience.

- If the behaviour of the MDP is known (i.e., the transitions probabilities between all the states are knows), the state function or the action-state function can be estimated by considering all the possible moves.

- This is not possible when:

  - The transitions probabilities are not know.

  - The system is very complex (for example a board game has a very large number of potential game configurations).

How to Estimate the State-Value (Action-Value) Functions: Monte-Carlo Methods

- Alternatively, the state-value function v_π and the action-value function q_π can be estimated for experience.

- One possibility is to keep average values of the actual returns that have followed a certain state (or a certain action) while following a policy π. These values will converge to the actual state-value function v_π and the action-value function q_π asymptotically.

- These methods based on averaging sample returns are referred to as *Monte Carlo methods*.

How to Estimate the State-Value (Action-Value) Functions: Monte-Carlo Methods

- Monte Carlo methods are not appropriate in case the number of states is very large.

- In this case, it is not practical to keep separate averages for each state individually.

- Instead, v_π and q_π are maintained as parametrised functions with the number of parameters << number of states.

- Various function approximators of different complexity are possible.

  - Artificial neural networks are a possible option as function approximators -> Deep Reinforcement Learning

Bellman Equation

- For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$
\begin{aligned}
v_\pi(s) &\doteq E_\pi[G_t \,|\, S_t = s] \\
&= \sum_a \pi(a \,|\, s) \sum_{s'} \sum_r p(s', r \,|\, s, a)[r + \gamma E_\pi[G_{t+1} \,|\, S_{t+1} = s']] \\
&= \sum_a \pi(a \,|\, s) \sum_{s', r} p(s', r \,|\, s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

with actions $a \in \mathcal{A}(s)$, the next states $s' \in \mathcal{S}$ and the rewards $r \in \mathcal{R}$.

- This equation is called the Bellman equation for v_π. It expresses a relationship between the value of a state and the values of its successor states.

Optimal Policies and Optimal Value Functions

- Solving a reinforcement learning is roughly equivalent to finding a policy that maximise the amount of reward over the long run.

- In finite MDPs there is always at least one policy that is better or equal to all the other policies: this is called the *optimal policy*.

- Although there may be more than one, we denote all the optimal policies with π_*. They are characterised by the same value function v_* defined as

$$v_*(s) \doteq \max_{\pi} v_\pi(s)$$

for all $s \in \mathcal{S}$.

Optimal Policies and Optimal Value Functions

- Optimal policies also shares the same optimal action-value function q_*, which is defined as

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

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

- We can write q_* in terms of v_* as follows:

$$q_*(s, a) = E[R_{t+1} + \gamma v_*(S_{t+1}) \,|\, S_t = s, A_t = a].$$

Bellman Optimality Equation

- We can re-write the Bellman equation under the optimal policy, which is called the Bellman optimality equation.

- Intuitively, the Bellman optimality equation must equal the expected return for the best action from that state.

- Once we have v_* (or q_*), the actions that select the highest value for v_* (or q_*) at each step (state) are the optimal actions.

- Another way of saying that is that any policy that is *greedy* with respect to v_* (or q_*) is an optimal policy.

- This is very efficient: if we have v_* (or q_*) we just need to check the next step (local choice/short-term consequence) for taking into account long-term ones.

Optimality and Approximation

- Explicitly solving the Bellman optimality equation is not possible.
- You need to completely know the environment and you need to solve the equation for each state.
- Also simple games like backgammon have very large number of states (backgammon has 10^{20} states for example).
- There is a problem of memory. Considering a tabular form, you need a row for each state!
- In general, in MDPs we have *incomplete knowledge* of the environment.

Difference between Reinforcement Learning and Supervised Learning

- Supervised learning is learning from a set of labeled examples.

- In interactive problems, it is hard to obtain labels in the first place.

- In “unknown” situations, agent have to learn from their experience. In these situations, reinforcement learning is most beneficial.

Difference between Reinforcement Learning and Unsupervised Learning

- Unsupervised learning is learning from datasets containing unlabelled data.

- You might think that reinforcement learning is a type of unsupervised learning, because it does not rely on examples (labels) of correct behaviour and instead explores and learns it. However, in reinforcement learning the goal is to maximise a reward signal instead of trying to find a hidden structure.

- For this reason, reinforcement learning is usually considered a third paradigm in addition to supervised and unsupervised learning.

References

- The notation and definitions of these slides are taken (with small variations) from:

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning. An Introduction. Second Edition. MIT Press. 2018.