An Introduction to Reinforcement Learning

Fundamental and formalism

Pei Liu

May 20, 2019

Department of Computer Science @UESTC

Table of contents

Background

Background - challenges

In supervised learning, the agent does:

- learning from the labeled training set
- making their outputs mimic the labels y given in the training set.

But it is indeed stuck in some scenarios, like *decision making* and *control problems*. Luckily, Reinforcement Learning (RL) came to the stage and solved them! In essence, it can be briefly summarized as:

- **experience-driven** autonomous learning
- improving over time through **trial and error**

Background - gallery

RL scenarios A: *Flappy Bird*

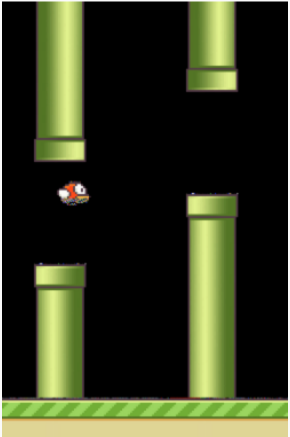

Figure 1: RL based Flippy Bird

Background - gallery

RL scenarios B: *Inverted Pendulum*

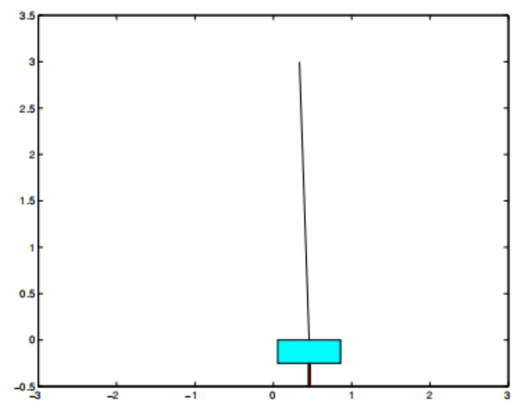

Figure 2: RL based balanced inverted pendulum

Background - question

In recent years, we have witnessed more examples like: AlphaGo, Self-driving, Robot, Control system.

Compared with previous generation RL, RL represented as above is more powerful in:

- directly recognizing *image* or *text* as the input
- more adaptive in the real world

It's called Deep Reinforcement Learning (DRL). DRL agents could be trained on raw, high-dimensional observations, solely based on a reward signal.

Background - question

Here our study of reinforcement learning will begin with a definition of the Markov decision processes (MDP), which provides the formalism in which RL problems are usually posed.

Formalism

Formalism - definition

A alertMarkov decision processes (MDP) is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$, where:

- S, A are set of **states** and **actions**, respectively
- P_{sa} are the state transition probabilities
- $\gamma \in [0, 1)$ is called the **discount factor**
- $R : S \times A \mapsto \mathbb{R}$ is the **reward function**. (or $R : S \mapsto \mathbb{R}$)

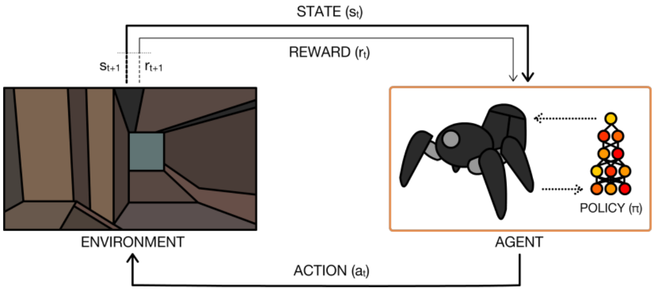

Figure 3: The perception-action-learning loop

Formalism - procedure

At time t, the procedure in the loop of perception-action-learning would be as follows:

- the agent receives state S_t from the environment
- the agent uses its policy π to choose an action A_t
- once the action is executed, the environment transitions a step
- the environment provides the next state S_{t+1} and reward R_{t+1}

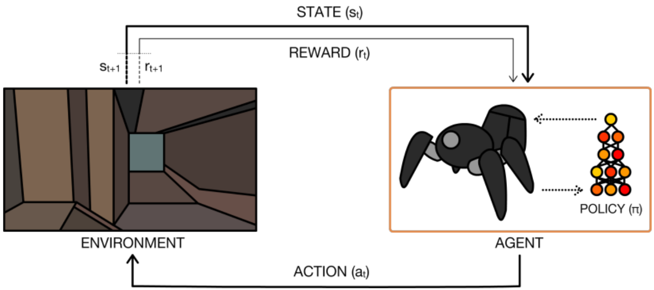

Figure 4: The perception-action-learning loop

Formalism - procedure of MDP

In MDP, the procedure could be simplified as:

- we starts in a state s_0
- we get to choose some action $a_0 \in A$ to take in the MDP
- once the action is executed, the state randomly transitions to some state $s_1 \sim P_{s_0 a_0}$
- we receive the reward $R(s_0)$ or $R(s_0, a_0)$
- $\cdots\cdots$

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots$$

Our total payoff is given by

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

.

Formalism - solution to MDP

In MDP, the procedure could be simplified as:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

Our goal in RL is to **choose actions** over time so as to maximize the expected value of the total payoff:

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

It is to say that we should find a **policy function** $\pi : S \mapsto A$ mapping from the states to the optimal actions. Under policy π, the total payoff we get is the **value function** as given by

$$V^{\pi}(s) = E\big[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi\big]$$

Formalism - solution to MDP

With the definition of the **value function**

$$V^{\pi}(s) = E\big[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi\big]$$

and a fixed **policy** function π. We can know that the value function $V^{\pi}(s)$ satisfies the **Bellman equations**:

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s^{'})V^{\pi}(s^{'})$$

which is also called as Dynamic Programming. Bellman's equations can be used to efficiently solve for V^{π}. (i.e. a set of $|S|$ linear equations in $|S|$ variables)

But the solution we want is the optimal value function $V^{*}(s)$.

Formalism - solution to MDP

We also define the **optimal value function** according to

$$V^*(s) = \max_\pi V^\pi(s). \tag{1}$$

In other words, this is the best possible expected sum of discounted rewards that can be attained using any policy. There is also a version of Bellman's equations for the optimal value function:

$$V^*(s) = R(s) + \max_{a\in A} \gamma \sum_{s'\in S} P_{sa}(s')V^*(s'). \tag{2}$$

The first term above is the immediate reward as before. The second term is the maximum over all actions a of the expected future sum of discounted rewards we'll get upon after action a. You should make sure you understand this equation and see why it makes sense.

We also define a policy $\pi^* : S \mapsto A$ as follows:

$$\pi^*(s) = \arg\max_{a\in A} \sum_{s'\in S} P_{sa}(s')V^*(s'). \tag{3}$$

Note that $\pi^*(s)$ gives the action a that attains the maximum in the "max" in Equation (2).

Algorithms

Algorithms - overview

How can we find the optimal policy so as to maximize the expected total payoff?

Here we describe two efficient algorithms for solving **finite-state MDPs**:

- value iteration
- policy iteration

The two methods are heuristic and solved by iterations.

Algorithms - value iteration

In this case, the algorithm can be viewed as implementing a ‘’Bellman backup operator‘’ that takes a current estimate of the value function, and maps it to a new estimate.

1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence {

   For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s')V(s')$.

   }

This algorithm can be thought of as repeatedly trying to update the estimated value function using Bellman Equations (2).

Figure 5: Algorithm - value iteration

NOTE: step (a) can be done via solving Bellman's equations as described earlier, which in the case of a fixed policy, is just a set of $|S|$ linear equations in $|S|$ variables.

1. Initialize π randomly.
2. Repeat until convergence {
   (a) Let $V := V^{\pi}$.
   (b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s')$.

   }

Figure 6: Algorithm - policy iteration

Extension

Extension

As we can see, MDP is a simple and ideal model. Even the transition probabilities P_{sa} is known!

Assuming that we don't know P_{sa}, and all we have is **just the reward signal** (actually, it's the situation), how can we learn a agent/model from experiences via Reinforcement Learning?

The **Inverted Pendulum** problem may be the best practice! See *here* for more details.

Extension

Next presentation, we will dive into **Flappy Bird** and reveal the principle **Q-learning** (a common method in RL) behind it!

Please be sure that you have known the solution of the problem **Inverted Pendulum**!

Questions?

References

References:

- Stanford CS229 - Reinforcement Learning
- cnblogs - An introduction to Reinforcement Learning
- A Brief Survey of Deep Reinforcement Learning