Introduction to reinforcement learning

Erik Pärlstrand

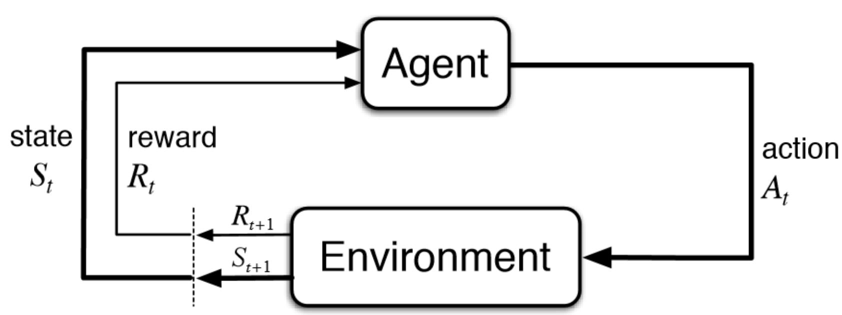
April 2018

Introduction

In supervised learning, we are try to find a good mapping from the feature-space to the target (x to y). In unsupervised learning, we are try to transform the features, do clustering etc. They are "fairly" similar. However, reinforcement learning is different from these two. Common terminology in reinforcement learning:

- **Agent** - sense the environment and operates in it.
- **Environment** - real/simulated world where the agent lives in.
- **States** - different configurations of the environment that the agent can act in. In this course we will only consider finite number of states.
- **Rewards** - rewards that the agent receives. It tries to maximize immediate and long-term rewards. The rewards are real numbers.
- **Actions** - what the agent does in it's environment.

Below is an example of the setup:



Episode: represents one run of the game. Our rl-agent learn across multiple episodes. Note that the number of episodes is a hyper-parameter.
There are two types of episodes: **episodic task** have a terminal state and we play it over and over again. **Continuous task** never ends (no terminal state).

A state is called terminal if no action can be taken from that state.

Credit assignment problem: what did I do in the past that led to the rewards I'm receiving now?

Exploration vs exploitation dilemma

The exploration versus exploitation dilemma refers to the problems where we want to acquire new knowledge and maximize the reward at the same time. Think of a row of slot machines in a casino. Each machine has its own probability of winning.
How do we figure out which of the machine has the best odds, while at the same time maximizing the profit? If you constantly played one of the machines you would never learn anything about the other machines. If you always picked a machine at random you would learn a lot about each machine but would not make as much money as you could have always playing the "best" machine.

MDP

The (first) order Markov property is defined as:

$$p(x_t|x_{t-1},...,x_1) = p(x_t|x_{t-1})$$

In the reinforcement learning we have:

$$p(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a, ...,) = p(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a) = p(s', r|s, a)$$

where s' represent the state we end up in at time $t+1$, r is the reward we receive in time $t+1$, s is the current state and a is the action we took.

Any reinforcement learning task with a set of states, actions and rewards that follow the Markov property is a MDP (Markov Decision Process). A MDP is defined by:

- A set of states.
- A set of actions.
- A set of rewards.
- A set of state-transition probabilities.

The policy (π) is not part of MDP, but along with the value function form the solution. Think of π as a shorthand for the algorithm the agent is using to navigate the environment.

Value function

The total reward is defined as:

$$G(t) = \sum_{\tau=1}^{\infty} R(t+\tau)$$

We usually want to weight the rewards long into the future less then the close ones. Therefore we can introduce a discount factor γ:

$$G(t) = \sum_{\tau=0}^{\infty} \gamma^{\tau} R(t+\tau+1)$$

If $\gamma = 1$ then we weight all rewards equally. If $\gamma = 0$ then we have a truly greedy method. Usually one chose $\gamma \approx 0.9$.

The value function is defined as the expected future reward from a state s. Estimating the value function is a central task in reinforcement learning. It is defined as:

$$V_{\pi}(s) = E_{\pi}[G(t)|S_t = s]$$

We can also define the value-action function as:

$$Q_{\pi}(s,a) = E_{\pi}[G(t)|S_t = s, A_t = a]$$

Optimal policy

In reinforcement learning we are typically interested in two aspects:

- Finding $V(s)$ given a policy. This is called the prediction problem.
- Finding the optimal policy π. This is called the control problem.

Optimal policy and optimal value function are interdependent. We can measure relative goodness between two policies:

$$\pi_1 \geq \pi_2 \text{ if } V_{\pi_1}(s) \geq V_{\pi_2}(s)\ \forall s \in S$$

Therefore the optimal policy is the best policy:

$$V_*(s) = \max_{\pi} V_{\pi}(s)\ \forall s \in S$$

Policy evaluation

In the previous section we got the definition of the value function for a given policy. It can be shown (using the definition of $G(t)$ and conditional probability) that the value function can be expressed as:

$$V_\pi(s) = E_\pi[G(t)|S_t = s] = ... = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_\pi(s')]$$

This is called the Bellman equation. Note that the expected future reward is only dependent on the next state's value function $V_\pi(s')$. In a similar fashion we can define the value-action function as:

$$Q_\pi(s, a) = E_\pi[G(t)|S_t = s, A_t = a] = ... = \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_\pi(s')]$$

Policy improvement/iteration

Given a current policy π, find π' s.t. $V_\pi(s) \leq V_{\pi'}(s)$. If we have Q:

$$\pi'(s) = \text{argmax}_a Q_\pi(s, a)$$

If we have V:

$$\pi'(s) = \text{argmax}_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_\pi(s')]$$

Notice that it's greedy. Continue looping this until the policy doesn't change.

For policy iteration, alternate between policy evaluation and policy improvement. Keep doing this until the policy doesn't change.

Value iteration

Value iteration is an alternative technique for solving the control problem. One disadvantage of policy iteration is that we use nested "loops".

Value iteration combines policy evaluation and policy improvements into one step:

$$V_{k+1}(s) = \max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_k(s')]$$

Monte Carlo methods

Dynamic programming required us to known the probability distribution $p(s', r|s, a)$. However, this is usually not the case in real applications.

The Monte Carlo method is model free, i.e. we don't need to know/specify $p(s', r|s, a)$. Instead of calculating the true expected value of G as we did in dynamic programming, we calculate the sample mean instead.

Monte Carlo policy evaluation

Recall that the value function is given by:

$$V_\pi(s) = E_\pi[G(t)|S_t = s]$$

This expectation can be approximated with Monte Carlo, i.e:

$$\bar{V}_\pi(s) = \frac{1}{N}\sum_{i=1}^{N} G_{i,s}$$

where i is the episode index. For a given episode we can generate tuples of states and rewards, i.e. $(s_1, r_1), ..., (s_T, r_T)$. Using these, we can calculate G (backwards) using the definition of G, i.e.:

$$G(t) = r_{t+1} + \gamma G(t+1)$$

Note that we need to wait until the episode is completed before we can update the sample mean.

Exploring starts

If the policy (π) is deterministic, some/many (s, a)-pairs will never be visited. One approach to solve this is the exploring starts method. For an episodic task, choose a random initial state and action. Thereafter follow the policy π.

One disadvantage of exploring starts is that we need to know all states and actions for the environment beforehand. However, in many applications this is not possible.

One alternative approach to exploring starts is to use epsilon greedy, i.e. change the policy such that the actions are random with a probability ϵ.

Monte Carlo policy improvement/iteration

In Monte Carlo with a given policy we only have the value function $V(s)$ and don't know which actions leads to better $V(s)$, since we can't do a look-ahead search (as in dynamic programming). We can only play episodes and get the rewards/states. However, this can be solved by using the value-action function Q. Recall that:

$$Q_\pi(s,a) = E_\pi[G(t)|S_t = s, A_t = a]$$

As for policy evaluation, this expectation can be approximated with Monte Carlo, i.e:

$$\bar{Q}_\pi(s,a) = \sum_{i=1}^{N} G_{i,s,a}$$

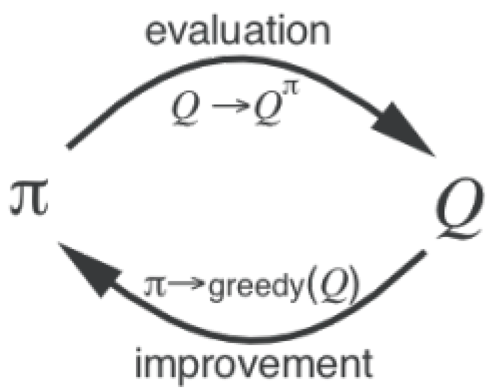
where i is the episode index. For a given episode we generate tuples of states, actions and rewards, i.e. $(s_1, a_1, r_1), ..., (s_T, a_T, r_T)$. Then we get policy by:

$$\pi(s) = \text{argmax}_a \bar{Q}_{\pi'}(s,a)$$

and the value function by:

$$V_\pi(s) = max_a \bar{Q}_\pi(s,a)$$

Now, policy iteration can be performed by alternate between policy evaluation and policy improvement, as described in the figure below:



However, this approach would imply that we need to estimate the value-action function from scratch for each policy. However, this would require a lot of iterations.

One solution to this problem is to not start with a fresh Monte Carlo evaluation each round but instead keep updating the same Q.