A Study of Reinforcement Learning as an Inference Problem

Name	Zhu Mingren
Student No.	3120200445
Department	School of Mechanics
Email	3120200445@bit.edu.cn
Date	May 18, 2021

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

Contents

Abstract

The framework of reinforcement learning provides a mathematical formalization of intelligent decision making that is powerful and broadly applicable. While the general form of the reinforcement learning problem enables effective reasoning about uncertainty, the connection between reinforcement learning and inference in probabilistic models is not immediately obvious. However, such a connection has considerable value when it comes to algorithm design: formalizing a problem as probabilistic inference in principle allows us to bring to bear a wide array of approximate inference tools, extend the model in flexible and powerful ways, and reason about compositionality and partial observability. In this article, we will discuss how a generalization of the reinforcement learning or optimal control problem, which is sometimes termed maximum entropy reinforcement learning, is equivalent to exact probabilistic inference in the case of deterministic dynamics, and variational inference in the case of stochastic dynamics.

Keywords: Reinforcement Learning, Probabilistic Inference, Soft Q-Learning, Soft Actor-Critic

1. Introduction

The idea that we learn by interacting with the environment is probably the first to occur to us that we think about the nature of learning. When an infant plays, waves its arms, or looks about, it has no explicit teacher, but it does have a direct sensorimotor connection to its environment. Exercising this connection produces a wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals. Throughout our lives, such interactions are undoubtedly a major source of knowledge about our environment and ourselves. Whether we are learning to drive a car or to hold a conversation, we are acutely aware of how our environment responds to what we do, and we seek to influence what happens through our behavior. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence.

Reinforcement learning (RL) is a computational approach to learning from interaction. Rather than directly theorizing about how people or animals learn, the study about RL is to explore idealized learning situations and evaluate the effectiveness of various learning methods. Compared to other approaches in machine learning, RL is much more focused on goal-directed learning from interaction.

1.1 Elements fo RL

Beyond the agent and the environment, one can identify four main elements of a RL system: a policy, a reward signal, a value function, and a model of the environment.

A policy defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a RL agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic.

A reward signal defines the goal in a RL problem. On each time step, the environments sends to the RL agent a single number, a reward. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent.

Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. One thing to keep in mind is that value function is about policies, which means different policies can lead to different value functions.

The final element of some RL systems is a model of the environment. This is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. Methods for solving RL problems that use models and planning are called model-based methods, as opposed to simpler model-free methods that are explicit trial-and-error learners–viewed as almost the opposite of planning.

1.2 Examples of RL

A good way to understand RL is to consider some of the examples and and possible applications that have guided its development.

- A master chess player makes a move. The choice is informed both by planning, anticipating possible replies and counter replies, and by immediate, intuitive judgments of the desirability of particular positions and moves.
- An adaptive controller adjusts parameters of a petroleum refinery's operation in real time. The controller optimizes the yield/cost/quality trade-off on the basis of specified marginal costs without sticking strictly to the set points originally suggested by engineers.
- A gazelle calf struggles to its feet minutes after being born. Half an hour later it is running at 20 miles per hour.
- A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on the current charge level of its battery and how quickly and easily it has been able to find the recharger in the past.
- Phil prepares his breakfast. Closely examined, even this apparently mundane activity reveals a complex web of conditional behavior and interlocking goal–subgoal relationships: walking to the cupboard, opening it, selecting a cereal box, then reaching for, grasping, and retrieving the box. Other complex, tuned, interactive sequences of behavior are required to obtain a bowl, spoon, and milk jug. Each step involves a series of eye movements to obtain information and to guide reaching and locomotion. Rapid judgments are continually made about how to carry the objects or whether it is better to ferry some of them to the dining table before obtaining others. Each step is guided by goals, such as grasping a spoon or getting to the refrigerator, and is in service of other goals, such as having the spoon to eat with once the cereal is prepared and ultimately obtaining nourishment. Whether he is aware of it or not, Phil is accessing information about the state of his body that determines his nutritional needs, level of hunger, and food preferences.

In all of these examples the agent can use its experience to improve its performance over time. The knowledge the agent brings to the task at the start, either from previous experience with related tasks or built into it by design or evolution, influences what is useful or easy to learn, but interaction with the environment is essential for adjusting behavior to exploit specific features of the task.

2. Traditional RL

The traditional framework of RL is based on the markov decision process (MDP), which defines the field of RL: any method that is suited to solving this problem is considered to be a RL method.

2.1 Elements of MDP

The RL problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision-maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time. A complete specification of an environment defines a task, one instance of the RL problem.

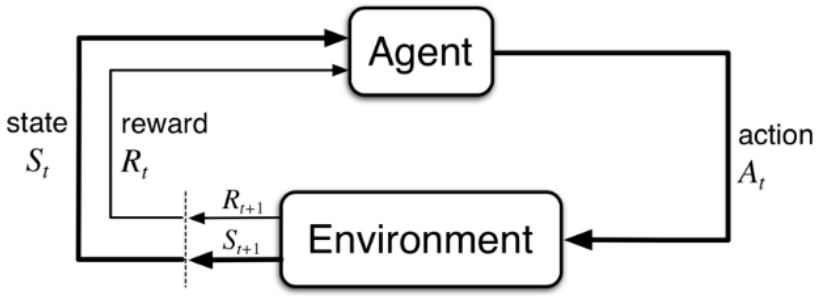

Figure 2.1: Interactions between an agent and the environment.

We have said that the agent's goal is to maximize the cumulative reward it receives in the long run. In the simplest case the return is the sum of the rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{2.1}$$

where R_t is the reward in time step t and T is a final time step.

The additional concept is discounting, which makes the later reward worth less than the earlier. The discounted return is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-1} \gamma^k R_{t+k+1} \tag{2.2}$$

where $0 \leq \gamma \leq 1$ is called the discount rate.

A particular MDP is defined by its state and action sets and by the one-step dynamics of the environment. Given any state and action s and a, the probability of each possible pair of next state and reward, s' and r, is donated:

$$p(s', r|s, a) = Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a) \tag{2.3}$$

These quantities completely specify the dynamics of a MDP. Given the dynamics as specified by Eq.(2.3), one can compute anything else one might want to know about the environment, such as the expected rewards for state-action pairs,

$$r(s, a) = \mathbb{E}\{R_{t+1}|S_t = s, A_t = a\} = \sum_{r \in \mathscr{R}} r \sum_{s' \in \mathscr{S}} p(s', r|s, a) \tag{2.4}$$

and the state-transition probabilities,

$$p(s'|s, a) = Pr(S_{t+1} = s'|S_t = s, A_t = a) = \sum_{r \in \mathscr{R}} p(s', r|s, a) \tag{2.5}$$

Almost all RL algorithms involve estimating value functions, which is determined by both the environment and the policy of agent. Recall that a policy, π, is a mapping from each state $s \in \mathscr{S}$, and action, $a \in \mathscr{A}(s)$, to the probability $\pi(a|s)$ of taking action a when in state s. Informally, the value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter:

$$V_\pi(s) = \mathbb{E}\{G_t|S_t = s\} = \mathbb{E}_\pi\{\sum_{k=0}^{T-1} \gamma^k R_{t+k+1}|S_t = s\} \tag{2.6}$$

where $\mathbb{E}_\pi$ denotes the expected value of a random variable given that the agent follows policy π. Note that the value of the terminal state, if any, is always zero. V_π is called the state-value function for policy π.

Similarly, we define 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:

$$Q_\pi(s, a) = \mathbb{E}_\pi\{G_t|S_t = s, A_t = a\} = \mathbb{E}_\pi\{\sum_{k=0}^{T-1} \gamma^k R_{t+k+1}|S_t = s, A_t = a\} \tag{2.7}$$

Q_π is called the action-value function for policy π.

2.2 The Q-learning Algorithm

One can figure out the relationships between V_π and Q_π:

$$V_\pi(s) = \mathbb{E}_\pi\{Q_\pi(s, a)\} = \sum_{a \in \mathscr{A}(s)} \pi(a|s) Q_\pi(s, a) \tag{2.8}$$

and:

$$Q_\pi(s,a) = \mathbb{E}_\pi\{r(s,a) + \gamma V_\pi(s')\} = r(s,a) + \gamma \sum_{s' \in \mathscr{S}} p(s'|s,a) V_\pi(s') \tag{2.9}$$

Eq.(2.8) and Eq.(2.9) can lead to the famous Bellman equations for both v_π:

$$V_\pi(s) = \sum_{a \in \mathscr{A}} \pi(a|s) \sum_{s' \in \mathscr{S}} p(s'|s,a)[r(s,a) + \gamma V_\pi(s')] \tag{2.10}$$

and q_π:

$$Q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathscr{S}} p(s'|s,a) \sum_{a' \in \mathscr{A}} \pi(a'|s') Q_\pi(s',a') \tag{2.11}$$

Given the optimal policy π^*, the optimal value functions are $V^*(s)$ and $Q^*(s,a)$, where the "optimal" means there is no other policy can make the value functions greater than these two,

$$V^*(s) \geq V_\pi(s), \forall s \in \mathscr{S}, \forall \pi \tag{2.12}$$

$$Q^*(s,a) \geq Q_\pi(s,a), \forall (s,a) \in \mathscr{S} \times \mathscr{A}, \forall \pi \tag{2.13}$$

where π^* can be derived by $\pi^*(a|s) = \delta(a = arg\max_a Q^*(s,a))$, and V^* can be derived by $V^*(s) = \max_a Q^*(s,a)$.

The Bellman equation for Q^*:

$$Q^*(s,a) = \mathbb{E}_{s' \sim p(s'|s,a)}\{r(s,a) + \gamma \max_{a'} Q^*(s',a')\} \tag{2.14}$$

It can be used to update $Q_\pi(s,a)$ even though the policy is not optimal:

$$Q_{new}(s,a) = \mathbb{E}_{s' \sim p(s'|s,a)}\{r(s,a) + \gamma \max_{a'} Q_{old}(s',a')\} \tag{2.15}$$

This updating can lead Q to be greater and going to converge:

$$Q_{old}(s,a) \leq Q_{new}(s,a) \rightarrow Q^*(s,a) \tag{2.16}$$

This is an algorithm called Q-learning. It is the best-known algorithm in RL and it is also the basic of many other more complex algorithms.

If the state s is in a high dimensional space such as pictures, the Q function can be parametrized to a neural network as $Q_\theta(s,a)$. Then calculating the Q_{target} value:

$$Q_{target} = \mathbb{E}_{s' \sim p(s'|s,a)}\{r(s,a) + \gamma \max_{a'} Q_\theta(s',a')\} \tag{2.17}$$

Finally using gradient descent methods to minimize the mean-square-error loss:

$$\theta \leftarrow \theta - \varepsilon \nabla_\theta ||Q_{target} - Q_\theta(s,a)||_2^2 \tag{2.18}$$

This is another well-known algorithm called deep Q-learning or deep Q-network (DQN). DQN is not guaranteed to converge, but it works well in practice.

3. Inference-version of RL

There are some issues about traditional RL:

- Can RL be used to explain human behavior?
- Does RL provide a reasonable model of human behavior?
- Is there a better explanation of human behavior?

Actually, there is a sub-field in RL called the inverse RL, it can be used to model the agent behavior according to the environment and some observations of the agent traces. But the data suited for RL needs to be optimal, while human behavior is non-optimal and stochastic. However, good behavior is still the most likely, and can be modeled in a probabilistic graph.

3.1 PGM of inference-version RL

The probabilistic graphical model (PGM) for decision making process is shown in fig.3.1.

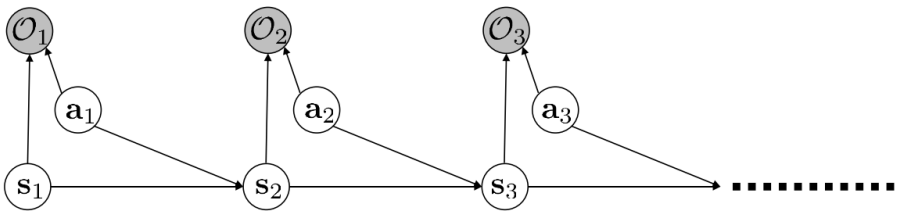

Figure 3.1: PGM for decision making

The system dynamic is still $p(s'|s,a)$, and it should not change no matter how the agent behaves. There are some new binary variables notated as O_t, $p(O=1|s,a) \propto \exp\{r(s,a)\}$ means the probability of action a is optimal for state s. $r(s,a)$ plays a role similar to "reward function".

The "optimal" behavior sequence here is a stochastic process with distribution which can be derived by the bayes theorem:

$$\begin{aligned} p(\tau = (s_t,a_t)_{t=1}^T | O_{1:T}=1) &\propto p(\tau)p(O_{1:T}|\tau) = p(\tau)\prod_{t=1}^{T} p(O_t=1|s_t,a_t) \\ &\propto [p(s_1)\prod_{t=1}^{T} p(s_{t+1}|s_t,a_t)]\exp\{\sum_{t=1}^{T} r(s_t,a_t)\} \end{aligned} \tag{3.1}$$

It can be used to model suboptimal and stochastic behavior. The posterior $p(a_t|s_t, O_{1:T})$ can be used as the optimal policy only when the system dynamic is $p(s_{t+1}|s_t, a_t, O_{t:T})$.

However, as it is said before, the system dynamic should not change no matter how the agent behaves, obviously $p(s_{t+1}|s_t, a_t, O_{t:T}) \neq p(s_{t+1}|s_t, a_t)$ in an ordinary way. Even so, it can still be used as a policy, just not the optimal one under $p(s'|s, a)$, because it can not generate the "optimal" behavior sequence as $p(\tau|O_{1:T})$.

Then how to get the optimal policy under $p(s'|s, a)$? What policy can generate the same "optimal" behavior sequence as $p(\tau|O_{1:T})$? Luckily, there is a method called variational inference (VI) to help us with it.

3.2 Soft Q-learning Algorithm

VI uses another policy $q(a|s)$ to approximate $p(\tau|O_{1:T})$ under the original system dynamic $p(s'|s, a)$, as shown in fig.3.2.

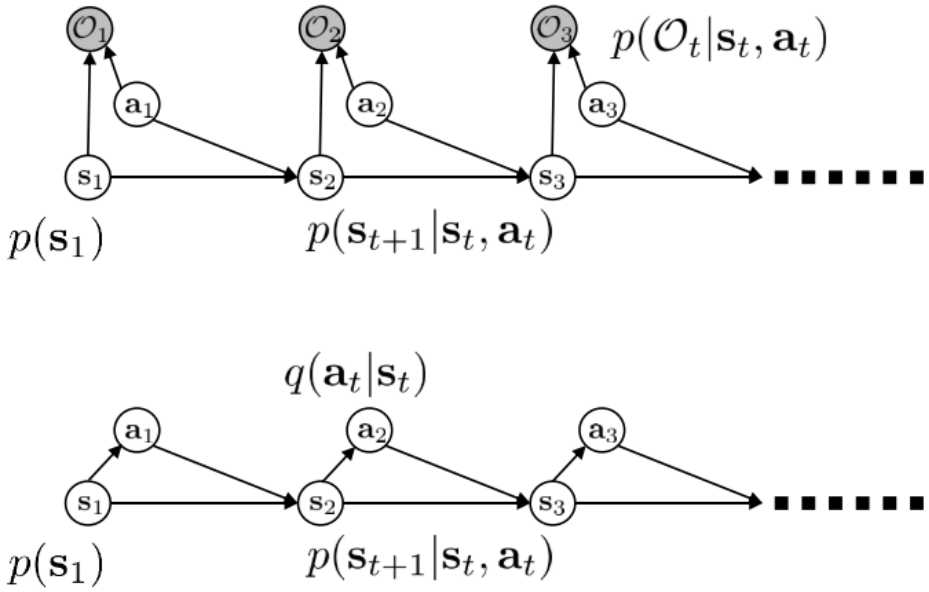

Figure 3.2: VI approximation

In VI, what we want is making $p(\tau) = p(s_1)\prod_{t=1}^{T} q(a_t|s_t)p(s_{t+1}|s_t, a_t)$ be a good approximation to $p(\tau|O_{1:T})$, which means we should maximize the ELBO:

$$\begin{aligned}
\mathbb{ELBO} &= \mathbb{E}_{\tau\sim q(\tau)}\{\log p(\tau, O_{1:T}) - \log q(\tau)\} \\
&= \mathbb{E}_{\tau\sim q(\tau)}\{\sum_{t=1}^{T}[r(s_t, a_t) - \log q(a_t|s_t)]\} \\
&= \sum_{t=1}^{T}\mathbb{E}_{(s_t,a_t)\sim q}\{r(s_t, a_t) - \log q(a_t|s_t)\}
\end{aligned} \tag{3.2}$$

To solve this optimal problem, let us think about $q(a_T|s_T)$ first:

$$q(a_T|s_T) = arg \max_{q(a_T|s_T)} \mathbb{E}_{(s_T,a_T)\sim q}\{r(s_T,a_T) - \log q(a_t|s_t)\}$$
$$\propto \exp\{r(s_T,a_T)\} \tag{3.3}$$

Let $Q(s_T,a_T) = \exp\{r(s_T,a_T)\}$ and $V(s_T) = \log\sum_{a\in\mathscr{A}} \exp\{r(s_T,a)\}$:

$$q(a_T|s_T) = \exp\{Q(s_T,a_T) - V(s_T)\} \tag{3.4}$$

Then we solve rest $q(a_t|s_t)$ by recursion. Let $Q(s_t,a_t) = r(s_t,a_t) + \mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}\{V(s_{t+1})\}$:

$$q(a_t|s_t) = arg \max_{q(a_t|s_t)} \mathbb{E}_{(s_t,a_t)\sim q}\{Q(s_t,a_t) - \log q(a_t|s_t)\}$$
$$\propto \exp\{Q(s_t,a_t)\} \tag{3.5}$$

Let $V(s_t) = \log\sum_{a\in\mathscr{A}} Q(s_t,a)$:

$$q(a_t|s_t) = \exp\{Q(s_t,a_t) - V(s_t)\} \tag{3.6}$$

This is an algorithm for inference-version RL called soft Q-learning. There is some common sense between the standard Q-learning and soft Q-learning:

Table 3.1: Standard & Soft Q-learning

	Standard Q-learning	**Soft Q-learning**
Action-Value Function	$Q(s,a) = r(s,a) + \mathbb{E}_{s'\sim p(s'|s,a)}\{V(s')\}$	$Q(s,a) = r(s,a) + \mathbb{E}_{s'\sim p(s'|s,a)}\{V(s')\}$
State Function	$V(s) = arg\max_a Q(s,a)$	$V(s) = \log\sum_a \exp\{Q(s,a)\}$
policy	$\pi(a|s) = \delta(a = arg\max_a Q(s,a))$	$\pi(a|s) = \exp\{Q(s,a) - V(s)\}$

We can see these two algorithms are very similar to each other. Actually, if $r(s,a)$ are made to be large enough, then the soft Q-learning will behaves just like standard Q-learning.

3.3 A Toy Experiment

We now use a toy experiment called "cliff walking" (fig.3.3) to test the performance of standard Q-learning and soft Q-learning. The goal of the agent is to walk from the beginning to the end, as short a path as possible and to avoid falling off the cliff (or returning to the beginning). For the first experiment (fig.3.4), soft Q-learning is not as good as standard Q-learning in the case because the dynamic of the system is deterministic with only one optimal policy. However, if rewards are magnified 10 times during training with Soft-Q, it turns out that Soft-Q can be very similar to the standard Q-learning (fig.3.5) as we mentioned before.

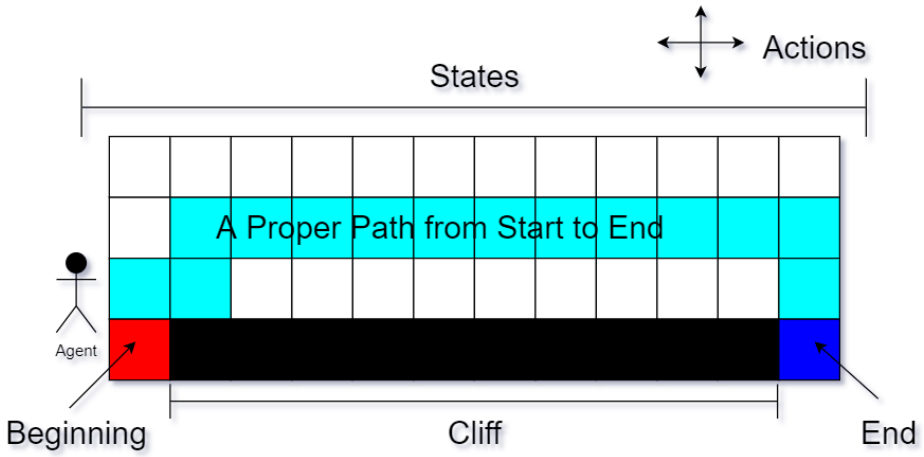

Figure 3.3: Cliff Walking

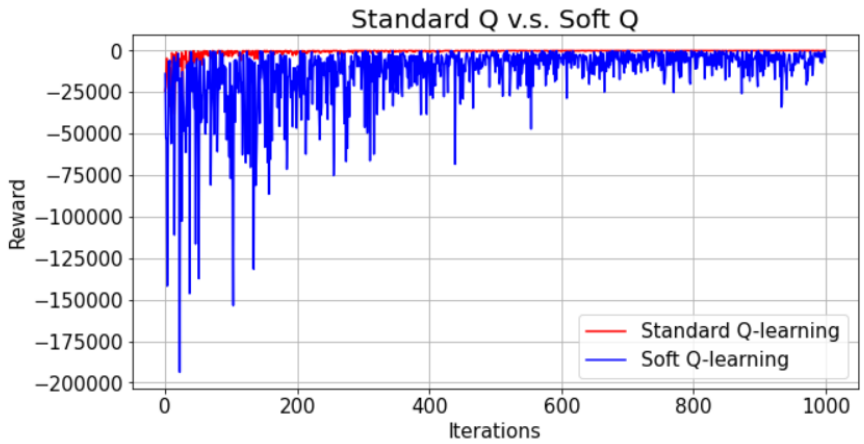

Figure 3.4: First Experiment

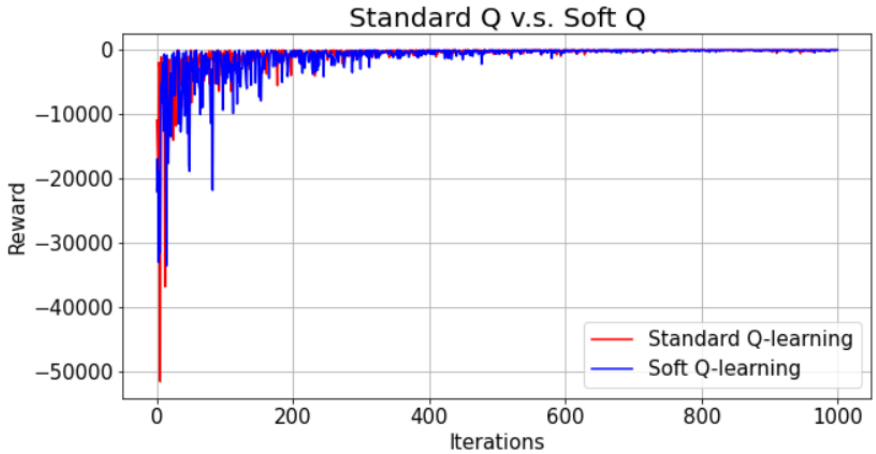

Figure 3.5: Second Experiment

4. Summary

Inference-version RL has many fancy applications such as to explore multi-goal/multi-mode environments (fig.4.1),

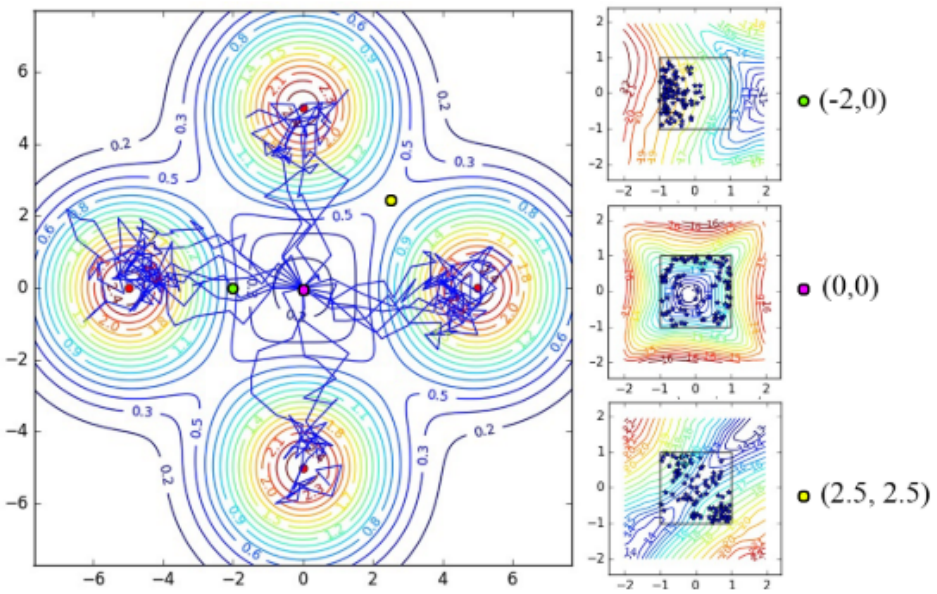

Figure 4.1: To explore multi-goal/multi-mode environments.

and to generate pre-trained policies (fig.4.2).

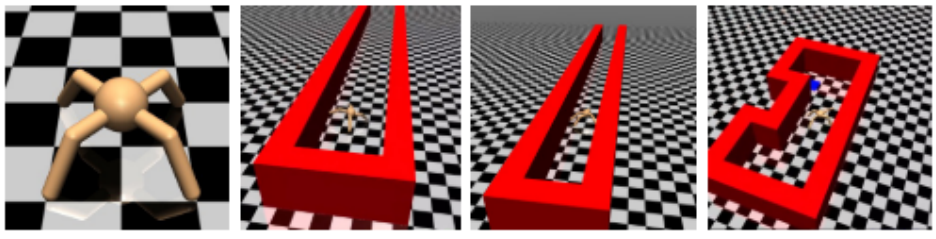

Figure 4.2: To generate pre-trained policies.

Of course, there are many inference-version RL algorithms besides soft Q-learning, for example another famous algorithm Actor-Critic also has its soft version called soft Actor-Critic. And there are many other applications that inference-version RL can be used. In conclusion, following are the benefits of soft optimality:

- Improving exploration and prevents entropy collapse;
- Empirically, policies are easier to fine-tune for more specific tasks;
- Better robustness (due to wider coverage of states);
- Reducing to hard optimality (by increasing the magnitude of the rewards);
- Good model for human behavior.

Finally, the relationship between probabilistic inference and control can shed some light on the design of reward functions and objectives in reinforcement learning. This is an often-neglected topic that has tremendous practical implications: reinforcement learning algorithms typically assume that the reward function is an extrinsic and unchanging signal that is provided as part of the problem definition. However, in practice, the design of the reward function requires considerable care, and the success of a reinforcement learning application is in large part determined by the ability of the user to design a suitable reward function. The control as inference framework suggests a probabilistic interpretation of rewards as log probability of some discrete event variable O_t, and exploring how this interpretation can lead to more interpretable, more effective, and easier to specify reward functions could lead to substantially more practical reinforcement learning methods in the future.

References

[1] Richard S.Sutton and Andrew G.Barto, Reinforcement Learning: An Introduction (Second edition), 2015, The MIT Press

[2] Sergey Levine, Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review, 2018, arXiv

[3] John Schulman, Equivalence Between Policy Gradients and Soft Q-Learning, 2017, arXiv

[4] Tuomas Haarnoja, Reinforcement Learning with Deep Energy-Based Policies, 2017, arXiv

[5] Tuomas Haarnoja, Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, 2018, arXiv

[6] Anjukan Kathirgamanathan, A Centralised Soft Actor Critic Deep Reinforcement Learning Approach to District Demand Side Management through CityLearn, 2020, arXiv

[7] Ruihan Yang, Multi-Task Reinforcement Learning with Soft Modularization, 2020, arXiv

[8] Kyungjae Lee, Generalized Tsallis Entropy Reinforcement Learning and Its Application to Soft Mobile Robots, 2020, Robotics: Science and Systems 2020

[9] Qisong Yang, WCSAC: Worst-Case Soft Actor Critic for Safety-Constrained Reinforcement Learning, 2021, AAAI

[10] Yuan Pu, Decomposed Soft Actor-Critic Method for Cooperative Multi-Agent Reinforcement Learning, 2021, arXiv

[11] Anjukan Kathirgamanathan, Development of a Soft Actor Critic Deep Reinforcement Learning Approach for Harnessing Energy Flexibility in a Large Office Building, 2021, arXiv

Appendix

Cliff Walking Code

```
import numpy as np
import matplotlib.pyplot as plt
import gym
from tqdm import tqdm

env = gym.make('CliffWalking-v0')
env.render()

def q_learning(env, alpha = 1., gamma = 0.99, episodes = 200, max_epsilon = 1., min_epsilon = 0.1, descent_ratio = 0.01):
    q_table = np.zeros((env.observation_space.n, env.action_space.n))
    acc_rewards = []
    for e in tqdm(range(episodes)):
        epsilon = max(max_epsilon - e*descent_ratio, min_epsilon)
        state = env.reset()
        acc_rewards.append(0.)
        t = 0
        while True:
            if np.random.uniform() < epsilon:
                action = np.random.choice(env.action_space.n)
            else:
                action = np.argmax(q_table[state])
            next_state, reward, done, _ = env.step(action)
            acc_rewards[-1] += reward
            v = np.max(q_table[next_state])
            q_table[state][action] += alpha*(reward+gamma*v-q_table[state][action])
            state = next_state
            t += 1
            if done:
                break
    return q_table, acc_rewards

```

```
32 def soft_q_learning(env, alpha = 1., gamma = 0.99, episodes = 200,
       max_epsilon = 1., min_epsilon = 0.1, descent_ratio = 0.01, beta = 1):
33     q_table = np.zeros((env.observation_space.n, env.action_space.n))
34     acc_rewards = []
35     for e in tqdm(range(episodes)):
36         epsilon = max(max_epsilon - e*descent_ratio, min_epsilon)
37         state = env.reset()
38         acc_rewards.append(0.)
39         t = 0
40         while True:
41             if np.random.uniform() < epsilon:
42                 action = np.random.choice(env.action_space.n)
43             else:
44                 v = np.log(np.sum(np.exp(q_table[state])))
45                 action = np.random.choice(env.action_space.n, p = np.exp(
                       q_table[state]-v))
46             next_state, reward, done, _ = env.step(action)
47             acc_rewards[-1] += reward
48             v = np.log(np.sum(np.exp(q_table[next_state])))
49             q_table[state][action] += alpha*(beta*reward+gamma*v-q_table[
                   state][action])
50             state = next_state
51             t += 1
52             if done:
53                 break
54     return q_table, acc_rewards
55
56 alpha = 0.8
57 gamma = 0.95
58 max_epsilon = 0.9
59 min_epsilon = 0.1
60 descent_ratio = 0.001
61 episodes = 1000
62
63 q_table, acc_rewards = q_learning(
64     env, alpha, gamma, episodes,
65     max_epsilon, min_epsilon, descent_ratio
66 )
67 soft_q_table, soft_acc_rewards = soft_q_learning(
68     env, alpha, gamma, episodes,
69     max_epsilon, min_epsilon, descent_ratio,
70     beta = 1
71 )
72
73 win = 1
74 weights = np.ones(win)
```

```
75  weights = weights/win
76  plt.figure( figsize = (10, 5))
77  plt.plot(np.convolve(acc_rewards, weights, "valid"), "r-", label = "
        Standard␣Q-learning")
78  plt.plot(np.convolve(soft_acc_rewards, weights, "valid"), "b-", label = "
        Soft␣Q-learning")
79  plt.ylabel("Reward", fontsize = 15)
80  plt.xlabel("Iterations", fontsize = 15)
81  plt.legend( fontsize = 15)
82  plt.xticks( fontsize = 15)
83  plt.yticks( fontsize = 15)
84  plt. title ("Standard␣Q␣v.s.␣Soft␣Q", fontsize = 20)
85  plt.grid()
86  plt.show()
```