CS234 Notes - Lecture 1
Introduction to Reinforcement Learning

Michael Painter, Emma Brunskill

March 20, 2018

1 Introduction

In **Reinforcement Learning** we consider the problem of learning how to act, through experience and without an explicit teacher. A reinforcement learning agent must interact with its world and from that learn how to maximize some cumulative reward over time.

Reinforcement learning has become increasingly more popular over recent years, likely due to large advances in the subject, such as Deep Q-Networks [1]. Moreover, other areas of Artificial Intelligence are seeing plenty of success stories by borrowing and utilizing concepts from Reinforcement Learning. For example, in game playing AlphaGo used Reinforcement Learning methods to reach superhuman performance in Go [3], and Reinforcement Learning concepts are also often borrowed in the training of Generative Adversarial Networks [2].

Many people often wonder how Reinforcement Learning differs from other types of learning. In **Supervised Learning** we are given a dataset, which consists of examples and labels. In the Supervised Learning setting, we are given a training set where for each example, we are provided the correct label (classification problems) / correct output (regression problems). In contrast, when no labels are provided in a dataset, **Unsupervised Learning** refers to methods that find underlying, latent structure in some data, when there is not labels for each example. However, in a Reinforcement Learning setting, we are dealing with making decisions and comparing actions that could be taken, rather than making predictions. A Reinforcement Learning agent may interact with the world, and receive some immediate, partial feedback signal — commonly called a **Reward** — for each interaction. However, the agent is given little indication if the action it took was actually the 'best' it could have chosen, and the agent must somehow learn to pick actions that will maximize a long term cumulative reward. Therefore, because of the weak/incomplete feedback provided by the reward signal we could consider Reinforcement Learning to lie somewhere between Supervised Learning, which gives strong feedback with labeled data, and Unsupervised Learning, with no feedback or labels.

The Reinforcement Learning setting introduces a number of challenges that we need to overcome, and potentially make trade-offs between. The agent must be able to *optimize* its actions to maximize the reward signal it receives. However, as the agent needs to learn by interacting with its environment, *exploration* is required. This leads to a natural trade-off between exploration and exploitation, where the agent needs to decide between potentially finding new, better strategies at the risk of a receiving a lower reward, or, if it should exploit what it already knows. Another question we face is, can the agent *generalize* its experience? That is, can it learn whether some actions are good/bad in previously unseen states? And finally, we also need to consider *delayed consequences* of the agents actions, that is, if it receive a high reward, was it because of an action it just took, or because of an action taken much earlier?

2 Overview of reinforcement learning

2.1 Sequential Decision Making

Commonly we will consider the problem of making a sequence of good decisions. To formalize this, in a discrete setting, an agent will make sequence of **actions** $\{a_t\}$, observe a sequence of **observations** $\{o_t\}$ and receive a sequence of **rewards** $\{r_t\}$. We define the **history** at time t to be $h_t = (a_1, o_1, r_1, ..., a_t, o_t, r_t)$. The agent's choice of what action to take next can be viewed as a function of the history, that is, $a_{t+1} = f(h_t)$, and the problem of sequential decision making can be thought of as defining and computing the function f appropriately. objective

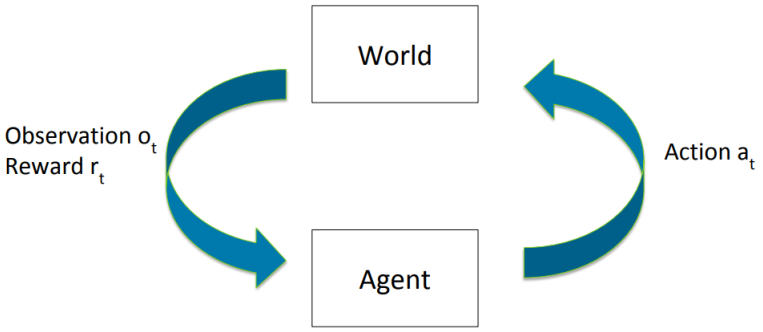

Figure 1: Overview of how an agent interacts with its world.

Finite states When the model of the sequential decision making process is known, in some deterministic and finite settings AI techniques like *A* search* and *minimax* can be used to find an optimal sequence of actions.

Infinite states (very large) However, when we have a very large number of possible states (potentially infinite), or introduce stochasticity into our model of the world, brute-force search becomes infeasible. In such a setting, it becomes *necessary* to incorporate generalization to make the task feasible, as in the Atari example seen in class. Moreover, without the ability to exhaustively search, the agent must decide on how to strategically make actions to balance short and long term rewards.

2.2 Modeling the world

Let S be the set of possible states that our world may be in, let $\{s_t\}$ be a sequence of states observed, indexed by time and let A be the set of possible actions. Often we will want to consider the **transition dynamics** of the world $P(s_{t+1}|s_t, a_t, ..., s_1, a_1)$, a probability distribution over S, and a function of previous states and actions. In reinforcement learning, we often assume the **Markov Property** that

$$P(s_{t+1}|s_t, a_t, ..., s_1, a_1) = P(s_{t+1}|s_t, a_t), \tag{1}$$

which in practice is flexible enough for our needs. A useful trick to make sure that the Markov property holds is to use the history h_t *as* our state.

Usually, we consider the reward r_t to be received on the transition between states, $s_t \xrightarrow{a_t} s_{t+1}$. A **reward function** is used to predict rewards, $R(s, a, s') = \mathbb{E}[r_t|s_t = s, a_t = a, s_{t+1} = s']$. We will often consider the reward function to be of the form $R(s) = \mathbb{E}[r_t|s_t = s]$ or $R(s, a) = \mathbb{E}[r_t|s_t = s, a_t = a]$. It will also often be the case that $r_t|s_t = s$ is degenerate, that is, r_t has a fixed value, given $s_t = s$.

A **model** consists of the above transition dynamics and reward function.

a model is { $P_{sa}(s')$; $R(s,a)$ or $R(s)$

An agent is { $V^{\pi}(s)$, $\pi(a|s_t^a)$, model } all 3 are optional.

2.3 Components of an reinforcement learning agent

Recall action model is $a_{t+1} = f(h_t)$

First, let the **agent state** be a function of the history, $s_t^a = g(h_t)$. A reinforcement learning agent typically has an explicit representation of one or more of the following three things: a policy, a value function and optionally a model. A **policy** π is a mapping from the agent state to an action, $\pi(s_t^a) \in A$, or, sometimes it is a stochastic distribution over actions $\pi(a_t|s_t^a)$. When the agent wants to take an action and π is stochastic, it picks action $a \in A$ with probability $P(a_t = a) = \pi(a|s_t^a)$. Given a policy π and discount factor $\gamma \in [0, 1]$, a **value function** V^{π} is an expected sum of discounted rewards,

$$V^{\pi}(s) = \mathbb{E}_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s]. \quad (2)$$

↑ expected value following policy π

Where $\mathbb{E}_{\pi}$ denotes that the expectation is taken over states encountered by following policy π, and, the **discount factor** γ is used to weigh immediate rewards versus delayed rewards. Lastly, a reinforcement learning agent may have a model, which is defined in section 2.2. If the agent has a model, we would call it a **model-based** agent, and if it doesn't incorporate a model, we would call it a **model-free** agent.

no known $P_{sa}(s')$ and $R(s,a)$

So far, we have consider definitions in a very general setting, and we have not made any assumptions about the relationship between o_t and s_t. We call the case where $o_t \neq s_t$ **partially observable**, and it is common in partially observable settings for RL algorithms to maintain a probability distribution over the true world state to define s_t^a, which is known as a **belief state.**

However, for the majority of the class, we will consider the **fully observable** case, where $o_t = s_t$, and will assume that $s_t^a = s_t$.

2.4 Taxonomy of reinforcement learning agents

We can classify our agents in a number of ways, as can be seen in table 1, and each type of agent isn't necessarily unique. For example, an actor critic agent could also be a model free agent. An overview of ways that we can classify agents can also be seen in figure 2.

Agent type	Policy	Value Function	Model
Value Based	Implicit	✓	?
Policy Based	✓	X	?
Actor Critic	✓	✓	?
Model Based	?	?	✓
Model Free	?	?	X

Table 1: An overview of properties of different types of reinforcement learning agent. Where a check-mark indicates that the agent has the component, a cross indicates that it must not have the component, and a question mark indicates that the agent may have that component, but isn't required to have it.

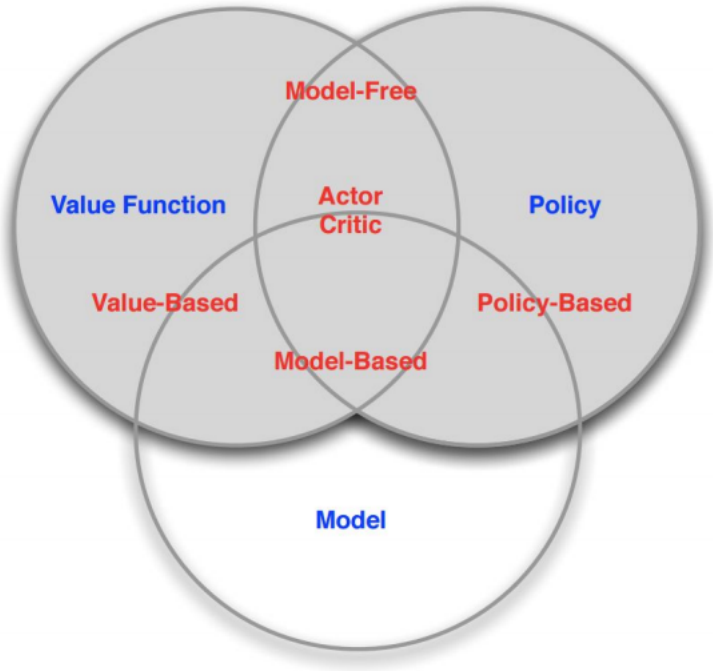

Figure 2: Classification of different reinforcement learning agents. Figure from David Silver. [4]

2.5 Continuous domains

For simplicity, we have focused our discussion on only discrete state and action spaces, with discrete time-steps. However, there are many applications — especially within Robotics and Control — that is most appropriately modeled with continuous state and action spaces, with continuous time. The discussion above can be generalized to incorporate these continuous settings.

References

[1] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

[2] Pfau, David, and Oriol Vinyals. "Connecting generative adversarial networks and actor-critic methods." arXiv preprint arXiv:1610.01945 (2016).

[3] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

[4] Silver, David. "Reinforcement Learning." 15 Jan. 2016. Reinforcement Learning, UCL.