Introduction to Reinforcement Learning

Uncharted Technologies

Abstract

This document contains an introduction to reinforcement learning and its applications. It is intended for an audience with a scientific background, but does not assume previous knowledge of the field. It starts with a theoretical overview of reinforcement learning, and then discusses some applications in various fields.

Reinforcement learning (RL) is a branch of artificial intelligence that aims to build algorithms that autonomously learn to achieve a given objective through interactions with an environment. Reinforcement learning is a rapidly expanding field that has gained prominence since around 2015, with the creation of algorithms that achieve super-human performance at video games [1, 2] and complex games such as Go and chess [3]. Real-world applications of RL now include data center cooling, battery use optimization in smartphones, and online recommendation systems. As RL algorithms improve, we expect that they will play an important role in fields such as robotics and finance.

The following section is a short and relatively technical introduction to the theory of reinforcement learning. The objective of this section is to provide the reader with some theoretical foundations that can accelerate further learning.

1 Theory of Reinforcement Learning

Reinforcement learning applies to problems that can be modelled as a Markov Decision Process (MDP), in which an agent interacts with an environment during episodes consisting of a finite number of time steps. At every time step t the agent observes the current state s_t of the environment, chooses an action a_t to perform in this environment from a set of possible actions, and then receives both a reward r_t and an observation of the new state s_{t+1}. **The goal of reinforcement learning is to find a *policy* for choosing actions, given the states, that yields the maximum expected sum of rewards over an episode.**

Autonomous learning is usually achieved using a trial and error approach, in which the agent first has to explore the environment to understand its dynamics and find the rewards. The rewards it finds are used as feedback that will modify the agent's behavior in subsequent episodes. Two prominent methods used to modify the agent's behavior based on the rewards it finds are known as value-based methods and policy-based methods.

1.1 Value-Based methods

We define the *Q-value* at time t as the expected sum of future rewards from time t following a given policy, knowing that action a_t was taken at state s_t:

$$Q(a_t|s_t) = \mathbb{E}\left[\sum_{t'>t} r_{t'}\right] \tag{1}$$

This value is implicitly a function of the chosen policy, since the rewards that are received at future time steps depend on what actions were chosen at those time steps.

This Q-value plays an important role when it comes to finding the optimal policy. Assume that we know the optimal policy. It can easily be seen that the Q-value for this optimal policy (which we designate as Q^*) then verifies the following equation:

$$Q^*(a_t|s_t) = r_t + \max_{a_{t+1}}(Q^*(a_{t+1}|s_{t+1})) \tag{2}$$

since the action that is taken at step $t+1$ has to be the one that maximises the sum of future returns (otherwise the policy would not be optimal). Conversely, it can be shown that any function that verifies equation 2 is the Q-value for the optimal policy. If we can find this Q-value, then the optimal policy consists simply of always choosing the action that maximises that value.

Value-based methods attempt to find a function that verifies equation 2 for all relevant states and actions. This is typically done by randomly initialising some function Q^θ with some parameters θ that describe the function, and then iteratively updating θ so as to minimise the loss

$$L(\theta) = (Q^\theta(a_t|s_t) - (r_t + \max_{a_{t+1}} Q_t^\theta(a_{t+1}|s_{t+1}))^2 \tag{3}$$

over all states and actions. This can be done by gradient descent for example. At the end of this update process, we should have $Q^\theta \approx Q^*$.

1.1.1 The DQN algorithm

A value-based method has for example achieved super-human performance in video games, using the Deep Q Networks (DQN) algorithm [1]. This algorithm minimizes $L(\theta)$ in 3, where θ are the parameters of a neural network that aims to approximate the Q-function.

In order to balance exploration and exploitation, this algorithm follows an ϵ-greedy policy, whereby with probability ϵ the agent takes a random action in the environment, and otherwise the agent takes the action that maximizes its current estimate of the Q-value. ϵ is gradually annealed over time from 1 to a smaller value such as 0.1 so that the agent transitions from exploring its environment to exploiting the information it acquired. DQN also alternates between taking steps in the environment and updating its neural network with the loss in equation 3.

Another key component of DQN is that of a target network. Empirically, updating the neural network using $\text{Ł}(\theta)$ causes the learning to be unstable. Instead, DQN uses a fixed target network to approximate $Q_t^\theta(a_{t+1}|s_{t+1})$ in equation 3. This target network is only periodically updated, which allows more stable learning.

1.2 Policy-based methods

In policy-based methods, we directly define a policy as some function mapping states to actions, and we seek to optimize this function to maximize the sum of rewards over an episode. For example, we can choose to define a *stochastic* policy π that maps a state to probabilities over actions, such that at every time step t we choose action a_t with probability $\pi(a_t|s_t)$. π is a function of both the state and the action, and we choose to parameterise this function using a set of parameters θ. We now call it π^θ.

We pointed out in the previous section that the expected sum of future rewards is implicitly a function of the policy, and hence of θ; we make use of this in the following. Specifically, we can take the gradient of the expected sum of future rewards from time t with respect to θ, and with a bit of maths we can prove the following relationship:

$$\Delta_\theta \left(\sum_{t'>t} r_{t'} \right) = \mathbb{E}\left[Q(a_t|s_t) \Delta_\theta \log \pi^\theta(a_t|s_t) \right] \tag{4}$$

where the expectation is over the possible actions. Intuitively, equation 4 is telling us to increase the probability of actions that yield greater rewards. We note that given a single trajectory of the agent interacting with the environment, the (a priori unknown) Q-value can be approximated using the empirical sum of rewards.

Equation 4, combined with some form of gradient descent on the parameters θ, gives us a method for improving a given policy in the direction of greater rewards. We can start with a randomly defined policy, observe its behavior on an episode, and then update its parameters using the observed rewards and gradient descent. This can be done repeatedly until we achieve convergence of the policy towards some local optimum.

Policy-based methods have been used to achieve super-human performance in video games [2] and in the game of Go [3], for example. Once again, a wide range of policy-based algorithms have been developed for many different applications.

1.3 Deep Reinforcement Learning

Both methods used in reinforcement learning require the use of functions that can transform some (potentially complicated) description of the environmental state into a number, which is the Q-value in value-based methods or the probability of taking an action $\pi(a_t|s_t)$ in policy-based methods. Furthermore, the functions chosen must allow for some simple method of gradient descent. A lot of initial research into reinforcement learning used linear functions.

Neural networks are better suited for this task. They can capture complex non-linear relationships between inputs and outputs, and the backpropagation method can be used to easily perform gradient descent on network parameters. The most impressive empirical results in reinforcement learning have thus been achieved using neural networks.

2 Reinforcement Learning in Practice

Reinforcement learning has had many successes in recent years, in particular with the demonstration of algorithms that autonomously learn to achieve super-human performance in complex domains such as video games and Go.

In this section, we describe the type of problem that reinforcement learning can be successfully applied to, and provide examples of successful use cases.

2.1 When to use reinforcement learning

Reinforcement learning has been particularly successful at solving complex sequential decision-making problems with the following characteristics:

- A large search space. Typically, for an agent interacting with an environment over the course of T time steps, where at each step the agent can choose between N different actions, the search space will be of size about N^T.
- A well-defined reward for the agent to maximize. In the case of Go, this can be +1 if the agent wins or -1 if it loses. For temperature control in a data center, the reward can be minus any deviation from the target temperature.
- A simulation and/or test environment. Since learning is typically done using a trial and error approach, having access to a simulator considerably accelerates the learning process. After the agent has successfully learned within the simulator, access to a test environment is preferable to fine-tune the learned policy in the real world.

2.2 Successes of Reinforcement Learning

Reinforcement learning has been shown to be able to outperform humans or human-specified systems in complex decision-making tasks.

2.2.1 Game playing

Games such as Go and chess are good examples of domains ideally suited to reinforcement learning. These games have too many different possibilities over too many time steps for tree search techniques to be entirely successful on their own. Chess-playing algorithms typically use tree search combined with a set of human-specified rules or heuristics. These human-defined rules impose limitations on the performance of chess-playing algorithms. Reinforcement learning algorithms are not subject to these limitations; recently, a RL algorithm with no initial knowledge apart from the basic rules of the game defeated the world's best non-RL chess-playing algorithm 28-0 [3].

2.2.2 Industry

Although RL is a young field, industrial applications have been developed in the following settings:

- Data center cooling. Maintaining a constant temperature in a complex environment with many variable heat sources and cooling systems is a difficult task. By using reinforcement learning, Google was able to achieve the same performance as its previous control system with less energy expenditure [4].
- Recommendation systems. Maintaining user interest over the course of multiple interactions with a social network or e-commerce website is challenging. Facebook successfully used RL to select which notifications to show users, and achieved a significant increase in click-through rate [5].
- Optimizing battery usage in smartphones.

Note that these systems typically combine RL with a wide range of other machine learning techniques to be successful.

2.3 Future Applications

As the leading technology for autonomously learning to perform complex decision-making, we expect reinforcement learning to play an increasing role in many fields.

The field of robotics illustrates both the promise of reinforcement learning and the challenges it faces. Current robots are typically built for a single purpose and programmed by several experts. Reinforcement learning opens the prospect of creating self-learning robots that can autonomously adapt to a wide range of possible tasks. Such robots will be invaluable for applications as diverse as manufacturing, drone delivery, and domestic care. Challenges remain, however, in the form of the complexity of many robotics domains, simulation capacities, and the definition of suitable rewards.

3 Educational Resources

To learn more about reinforcement learning, we recommend the following resources. In addition to these resources, there are many blog posts (of varying quality) online on just about every conceivable machine learning topic, as well as a huge amount of freely available python code on github.

- The reference textbook in the field (easily available online and very readable) is: Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction.* MIT press, 2018.
- A helpful blog with posts explaining many of the key concepts in reinforcement learning: https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] OpenAI, "OpenAI five." https://blog.openai.com/openai-five/.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[4] N. Lazic, C. Boutilier, T. Lu, E. Wong, B. Roy, M. Ryu, and G. Imwalle, "Data center cooling using model-predictive control," in *Advances in Neural Information Processing Systems*, pp. 3818–3827, 2018.

[5] J. Gauci, E. Conti, Y. Liang, K. Virochsiri, Y. He, Z. Kaden, V. Narayanan, and X. Ye, "Horizon: Facebook's open source applied reinforcement learning platform," *arXiv preprint arXiv:1811.00260*, 2018.