Reinforcement Learning

MInDS @ Mines

Reinforcement learning is the last primary type of machine learning to cover. The idea here is to place an agent in an environment and reward or punish their actions. Over time, this results in an agent that acts in a way to maximize the reward we specified.

Reinforcement Learning

Reinforcement learning is the third primary type of machine learning other than supervised learning and unsupervised learning. Both supervised and unsupervised learning shared a general data representation and formulation. With reinforcement learning the input to our models is slightly different. The general idea of reinforcement learning is to reinforce good behavior and stop supporting bad behavior. We place a model in an environment that it can observe and have it learn a policy to determine the actions it should make to maximize a value function that can calculate.

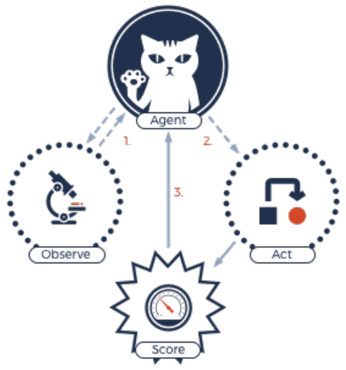

Figure 1: An illustration of the general reinforcement learning process.

General components of Reinforcement Learning

The following are the general components of reinforcement learning methods and applications,

- Agent
  The agent is the model that acts and learns within our environment.
- Environment
  The environment is the limited space that the agent exists within that defines the rules of what is possible.
- State
  The state is the relevant information about the environment with respect to the agent's actions and results. In theory, we distinguish between observations and the state. The state is the full true reality of the world whereas observations are the details that the agent has access to. The state at a particular time point is a function of the previous state and the agent's action,
  $$\mathsf{s}_{t+1} = f(\mathsf{s}_t, \mathsf{a}_t). \tag{1}$$

When an agent's observations represent the full state, we say that the state is *fully observed*, otherwise it is *partially observed*.

- Actions
  Actions are taken by the agent in order to move the state from one time period to the next. The action space consists of all possible actions that the agent can make. Similar to supervised learning, whether the space is discrete or continuous has a significant effect on the available methods we can use.

A sequence of states and actions is sometimes called a trajectory.

- Policy
  A policy is the set of rules an agent follows to determine the action it should take given its observations about the environment.

- Returns: Rewards & Punishment
  The environment provides a returned value to the agent after it takes an action. This value is in the form of a reward or punishment of some sort. The goal of the agent is to maximize the total rewards it receives from its actions in the environment. We can choose a finite horizon or infinite horizon approach to determine the return of a scenario at time τ,

$$\underset{finite}{R(\tau)} = \sum_{t=0}^{T} r_t, \quad \underset{infinite}{R(\tau)} = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{2}$$

  where r_t is the return received at time t in the simulation and γ is a discount factor. We can use the infinite horizon sum to weight earlier returns higher and also consider the future returns beyond time τ instead of simply adding up all the returns we've received.

- Exploration vs Exploitation
  Exploration and exploitation are usually hyperparameters to our policy learning where we try to balance between learning new things (exploration) and utilizing the knowledge we already have (exploitation). In general, we want the model to achieve the best result but it can sometimes only do so if it tries new things in order to learn more about the consequences of its actions. Once it has learned enough, it should minimize trying new things in order to make the most of the knowledge it has accumulated.

The objective of reinforcement learning methods is to select the optimal policy that results in the highest total reward. This can be represented as,

$$\arg\max_{\pi} \underset{\tau\sim\pi}{\mathrm{E}}[R(\tau)], \tag{3}$$

where π represents the policy we wish to learn in order to maximize our expected return after τ iterations. We also define two functions, the value function V and the action-value Q of a policy π at time τ given a starting state s as,

$$V^{\pi}(s) = \underset{\tau\sim\pi}{\mathrm{E}}[R(\tau)|s_0 = s], \tag{4}$$

$$Q^{\pi}(s,a) = \underset{\tau\sim\pi}{\mathrm{E}}[R(\tau)|s_0 = s, a_0 = a]. \tag{5}$$

The optimal value and action-value functions are those that determine the policy that maximizes the respective function.

We can also reformulate these equations with the idea of calculating the value not only based on the current state but the value of the future state given our policy. This means that we think of the value of our action as not only that which gives us a reward in the next time step but also places us in a state where we can continue to get rewards following our policy. The Bellman equations are this reformulation,

$$V^\pi(s_t) = \mathsf{E}\left[r(s_t, a_t) + \gamma V^\pi\left(s_{t+1}\right)\right], \tag{6}$$

$$Q^\pi(s_t, a_t) = \mathsf{E}\left[r(s_t, a_t) + \gamma \mathsf{E}\left[Q^\pi\left(s_{t+1}, a_{t+1}\right)\right]\right]. \tag{7}$$

Considerations with Reinforcement Learning

There are a couple things to consider with reinforcement learning that are somewhat intuitive and yet vital to understanding how it all works. One key idea is understanding the significance of a good reward function. Given a simulation of a city that is an ”open world” environment, we can train two very different models using the exact same method based on the reward function we specify. For example, the reward function for a driver in the game of Grand Theft Auto would be different from the reward function for a self-driving car that we hope to deploy in the real world. Another key idea is with over fitting in reinforcement learning; our model can focus on one key area in an environment based on the reward it expects to get there without exploring other options. A dire case of this is when a model ends up focusing on just minimizing the punishment it receives and ends up in a state of learned helplessness.

OpenAI has shown a new approach to reinforcement learning that focuses on curiosity driven learning. More information is available at `https://blog.openai.com/reinforcement-learning-with-prediction-based-rew`

Types of Reinforcement Learning Methods

There are many different types of reinforcement learning methods and approaches that we could learn about, however that would take us out of the scope of this class. We will only focus on a tiny subset of these but it is important to understand some common patterns in reinforcement learning approaches. First, there are methods that use or learn a state transition function for the environment and can use that in determining the best actions to take. These are called model based methods. Methods that do not utilize a state transition function are called model free methods. Model based methods fit in to one of two categories; those that learn the state transition function and those that are provided that function. Model free methods fit into one of two approaches; policy optimization and Q learning.

Policy optimization methods directly optimize the parameters that govern the policy and they usually only consider what happens in the environment when following the policy[1]. Q learning methods learn an approximation of the action-value function, Q and can do so by considering what happens in the environment regardless of the followed policy[2].

[1] Only considering what happens when following a policy is called *on-policy*.

[2] Considering what happens in the environment regardless of the policy used to perform the action is called *off-policy*

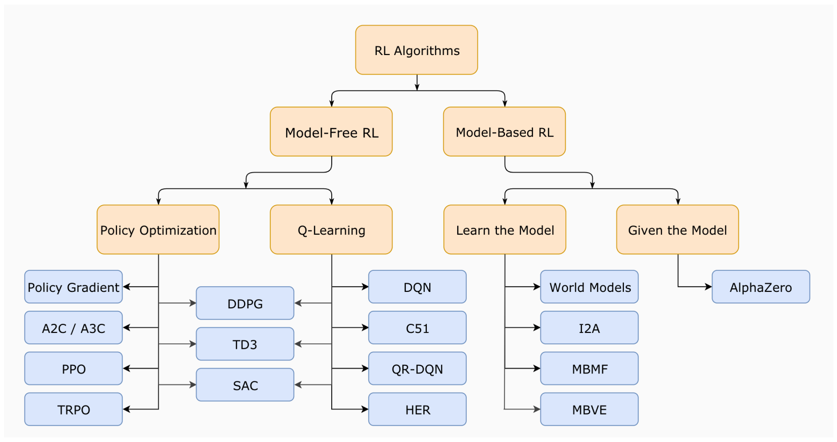

Figure 2: A useful hierarchy of a subset of reinforcement learning methods. Source

Q Learning

We will now focus on one particular method in reinforcement learning that defines a whole category of related methods that learn an approximation of the action-value function; Q learning. The first thing for Q learning is to develop a Q table or matrix. A Q table is a table of all the possible states and actions with the value for each cell being an estimate of $Q(s,a)$ as outlined above. We start by initializing the state usually as all set to 0 and then starting the simulation. After the model is trained, we would select the action that maximizes the Bellman equation. In training however, we can use a variety of different approaches for action selection. A common approach for action selection during training is the ϵ-greedy selection method which determines an action at time t,

One of the core premises here is that the current state and action are the only determining factors in the reward received.

$$a_t = \begin{cases} \arg\max_a Q(s_t, a) & P(1-\epsilon) \\ \text{random action} & P(\epsilon) \end{cases} \tag{8}$$

where ϵ is our exploration rate.

With our action selection approach, we take new actions in the simulated environment and update the Q table using,

$$Q_{\text{new}}(s,a) = (1-\alpha)Q(s,a) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a)), \tag{9}$$

where α is the learning rate and γ is our discount rate. Here the learning rate functions as another proxy for exploration where it defines how much we learn based on our new experience versus the old experiences we've had.

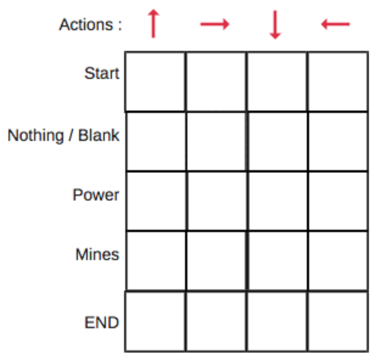

Figure 3: An example Q table with states and actions. Source

After training our model over many iterations, we should end up with a good Q table that can be used in practice. We can determine how well our model performs based on the total reward it achieves as discussed previously.

This lecture was heavily influenced by "Spinning up in Deep RL".