Introduction to Reinforcement Learning

Credits: RL Course Google DeepMind

Lars Quaedvlieg
Larsquaedvlieg@outlook.com

October 2020

Contents

Chapter 1

Intro to Reinforcement Learning

1.0.1 The Reinforcement Learning Problem

What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a **reward** signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Rewards are scalar feedback signals. Reinforcement Learning is based on the **Reward Hypothesis**, meaning all goals can be described by the maximization of expected cumulative reward. This can be hard, since actions can have long-term consequences and reward can be delayed.

A state is **Markov**, if and only if it holds the **Markov Property**, meaning $\mathbb{P}(S_t + 1|S_t) = \mathbb{P}(S_t + 1|S_1, ..., S_t)$. This means the probability of future states solely depends on the current state, and not on any previous states. I.e. the history is a sufficient statistic of the future.

Let S_t^a be the state of the agent at any time t and S_t^e be the state of the environment on any time t. If the environment is **fully observable**, then $S_t^a = S_t^e$. This means that the Markov property holds, so formally it is a Markov Decision Process.

However, when the environment is **partially observable**, the agent indirectly observes the environment. Now, $S_t^a \neq S_t^e$. Formally, this is called a partially observable Markov decision process (POMDP). The agent must construct it's own state representation S_t^a. For example:

- Complete history: $S_t^a = H_t$
- Beliefs of environment state: $S_t^a = (\mathbb{P}[S_t^e = s^1], ..., \mathbb{P}[S_t^e = s^n])$

- Recurrent Neural Network: $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o))$

1.0.2 Components of an RL Agent

An RL agent may include one or more of these components:

- **Policy**: agent's behaviour function
- **Value function**: how good is each state and/or action
- **Model**: agent's representation of the environment

A **policy** describes the agent's behavior. It maps states to actions. You can have deterministic ($a = \pi(s)$) and stochastic policies ($\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$). Often, π is used to denote a policy.

A **value function** is a prediction of future reward of a given state. You can use it to determine if a state is good or bad. This means you can use it to select actions. It can be computed by $v_\pi(s) = \mathbb{E}_\pi(G_t|S_t = s)$, where G_t is the **return** (or total reward). The return is defined as $G_t = R_1 + \gamma R_2 + \gamma^2 R_3 + ... = \sum_{i=t+1}^{\infty} \gamma^{i-t-1} R_i$ for some $\gamma \in [0, 1]$. This gamma is the **discount factor**, and it influences how much the future impacts return. This is useful, since it is not known if the representation of the environment is perfect. If it is not, it is not good to let the future influence the return as much as more local states. So, it is discounted.

Finally, a **model** predicts what the environment will do next. We let $P_{ss'}^a = \mathbb{P}(S_t + 1 = s'|S_t = s, A_t = a)$ and $R_s^a = \mathbb{P}(R_t + 1|S_t = s, A_t = a)$. P (**Transition model**) is the transition probability to a next state given an action, while R is the probability of obtaining a certain reward when taking an action in some state.

Category	Properties
Value based	No Policy (implicit), Value function
Policy based	Policy, No Value function
Actor Critic	Policy, Value function
Model Free	No Model
Model based	Model

Figure 1.1: Types of RL agents

RL Agents can be categorized into the categories that are listed in 1.1. These can require different approaches that will be discussed later.

There are two fundamental problems in **sequential decision making**.

- **Reinforcement Learning**
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- **Planning** (e.g. deliberation, reasoning, introspection, pondering, thought, search)
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy

It is important for an agent to make a trade-off between exploration and exploitation as well. Depending on the choice in this trade-off, agents will be more or less flexible and may or may not find better actions to perform.

- **Exploration** finds more information about the environment
- **Exploitation** exploits known information to maximize reward

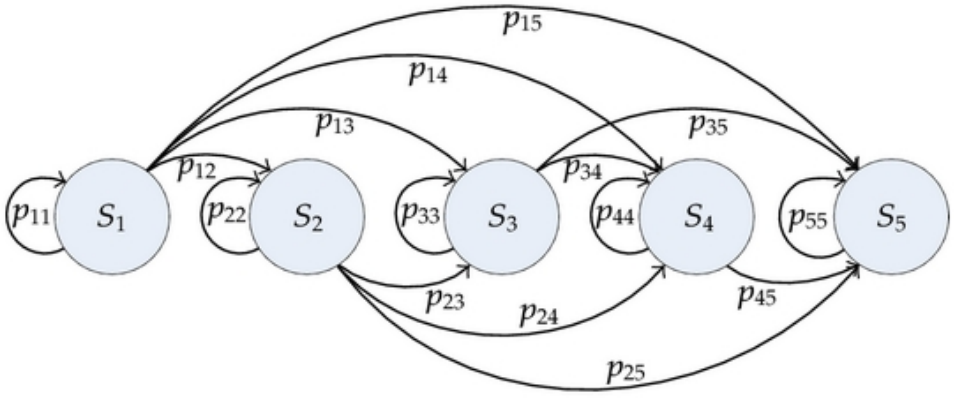
Finally, it is possible to differentiate between prediction and control. **Prediction** is about evaluating the future given a certain policy, while **control** is about finding the best policy to optimize the future.

Chapter 2

Markov Decision Processes

As said in chapter 1, fully observable environments in Reinforcement Learning have the Markov property. This means the environment can be represented by a **Markov Decision Process** (MDP). This means that the current state completely characterizes the process. MDP's are very important in RL, since they can represent almost every problem.

2.0.1 From Markov Chains to MDP's



$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} & p_{15} \\ 0 & p_{22} & p_{23} & p_{24} & p_{25} \\ 0 & 0 & p_{33} & p_{34} & p_{35} \\ 0 & 0 & 0 & p_{44} & p_{45} \end{bmatrix} \text{ where } p_{ij} \geq 0 \text{ and } \sum_j p_{ij} = 1$$

Figure 2.1: Example of a Markov Chain

Here, P is the **Transition Matrix**. It contains all transition probabilities of the model. A **Markov Chain** (or Markov Process), is a tuple (S, P), where S is a set of states and P is the transition matrix. A sequence $S_0, ..., S_n$

is called an **episode**.

From this Markov Chain, we can create a **Markov Reward Process**. This is defined as an extension of the tuple, with the elements (S, P, R, γ). S and P mean the same thing as before. However, now we also have a reward function (recall that $R_s = \mathbb{E}[R_{t+1}|S_t = s]$) and a discount factor $\gamma \in [0, 1]$. How to then turn 2.1 into a MRP? Simply add reward values to each state in the Markov Chain. The rewards of each state can then be computed with the reward function.

Now let's introduce the **Bellman Equation** for MRP's. It is possible to rewrite the formula of the value function in the following way.

$$\begin{aligned} v(s) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2}, \gamma R_{t+3}, ...)|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \\ &(= R_s + \gamma \sum_{s' \in S} P_{ss'} v(s')) \end{aligned} \tag{2.1}$$

Figure 2.2: Bellman equation of the value function in a MRP

Now it is possible to see that the value of a state is dependent on the immediate reward, and the return of neighboring states. This can then be computed using the transition matrix ($P_{ss'} > 0 \iff$ you can go from s to s'). The bellman equation can then be written in matrix form, namely $v = R + \gamma P v$. This can be rewritten as $v = (I - \gamma P)^{-1} R$. This equation can be solved in $O(n^3)$, since a matrix inverse is necessary. For large MRP's, there are several iterative methods to solve this equation. For example,

- Dynamic programming
- Monte-Carlo evaluation
- Temporal-Difference learning

These methods will be explained in later chapters.

2.0.2 Markov Decision Processes

Finally, let's talk about the **Markov Decision Process**. This is an extension of the MRP, with properties (S, A, P, R, γ). It introduces a new set, defining the actions that can be taken. The transition matrix P^a and R^a now also conditionally depend on the action.

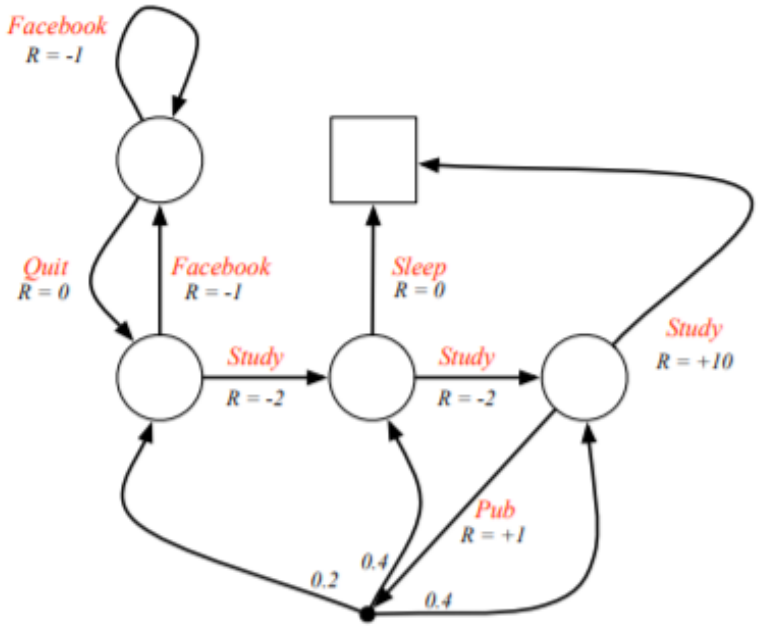


Figure 2.3: Example of a Markov Decision Process

2.3 is an example of a Markov Decision Process. In the image you see that the rewards are now dependent on the actions, as previously explained. Now it is possible to talk about the behavior of an agent. In chapter 1, policies were mentioned. These determine the behavior of agents (like which action to take in which state).

Given an MDP and a policy π. The state sequence $S_0, S_1, ...$ is a Markov Process (S, P^π) and the state and reward sequence $S_0, R_1, S_1, R_2, ...$ is a Markov Reward Process $(S, P^\pi, R^\pi, \gamma)$ where $P^\pi_{s,s'} = \sum_{a \in A} \pi(a|s) P^a_{s,s'}$ and $R^\pi_s = \sum_{a \in A} \pi(a|s) R^a_s$ (Law of Total Probability).

Similarly to the state-value function $v_\pi(s)$, let's now define the **action-value** function $q_\pi(s, a) = \mathbb{E}_\pi[G_t | S = s, A = a]$. This is the expected return starting from state s, taking action a, and then following policy π. In a RL

problem, this is what we care about. The goal is to select the best action in a given state, in order to maximize the reward. If we have access to the q-values, it is the solution to the problem.

After deriving the Bellman equations for the MDP's, we end up with the following formulas.

$$\begin{aligned} v(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \\ &(= \sum_{a\in A} \pi(a|s) q_\pi(s,a)) \\ &(= \sum_{a\in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_\pi(s') \right)) \end{aligned} \tag{2.2}$$

Figure 2.4: Bellman equation of the state value function in a MDP

$$\begin{aligned} q(s,a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \\ &(= R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_\pi(s')) \\ &(= R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a \sum_{a'\in A} \pi(a'|s') q_\pi(s',a')) \end{aligned} \tag{2.3}$$

Figure 2.5: Bellman equation of the action value function in a MDP

The second-last form of formula (2.5) can be intuitively thought of as the following. Imagine we are taking a certain action in a state. Then, the environment might bring us into different successor states even if we take the action. So, we have to sum over all these possible successor states and use the law of total probability again to compute the action-values.

Now we can talk about optimality. The **optimal state-value** and **optimal action-value** functions are defined as $v_*(s) = \max_\pi v_\pi(s)$ and $q_*(s,a) = \max_\pi q_\pi(s,a)$. We can say $\pi \geq \pi'$ if $\forall s : v_\pi(s) \geq v_{\pi'}(s)$. The optimal value function specifies the best possible performance in the MDP. An MDP is "solved" when we know the optimal value. For any MDP, there always exist an optimal policy that is better or equal to all policies. This policy achieves

both the optimal value function and the optimal action-value function.

An **optimal policy** can be found by maximizing over $q_*(s, a)$.

$$\pi_*(a|s) = \begin{cases} 1, & \text{if a} = \text{argmax}_{a \in A}\, q_*(s, a) \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

Figure 2.6: Equation for an optimal policy

Thee optimal value functions are recursively related to the **Bellman Optimality Equations**. $v_*(s) = \max_a q_*(s, a)$ and $q_*(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$. Solving for these equations is difficult, since the Bellman Optimality Equation is non-linear. Generally, there is no closed form solution. However, there exist iterative methods to approximate the solution. Example of this are

- Value Iteration
- Policy Iteration
- Q-learning
- Sarsa

2.0.3 Advanced: Extensions to MDPs

Infinite and continuous MDPs

- Countably infinite state and/or action spaces
  - Straightforward
- Continuous state and/or action spaces
  - Closed form for linear quadratic model (LQR)
- Continuous time
  - Requires partial differential equations
  - Hamilton-Jacobi-Bellman (HJB) equation
  - Limiting case of Bellman equation as time-step approaches 0

Partially observable MDPs

A Partially Observable Markov Decision Process is an MDP with hidden states. It is a **hidden Markov model** with actions. A POMDP is a tuple $(S, A, O, P, R, Z, \gamma)$. We now also have O representing a finite state of observations and Z being an observation function $Z^a_{s'o} = \mathbb{P}[O_{t+1} = o | S_{t+1} = s', A_t = a]$.

Let's define a **history** H_t being a sequence of actions, observations and rewards ($H_t = A_0, O_1, R_1, ..., A_{t-1}, O_t, R_t$). A **belief state** $b(h)$ is a probability distribution over states conditioned on the history h. $b(h) = (\mathbb{P}[S_t = s^1 | H_t = h], ..., \mathbb{P}[S_t = s^n | H_t = h])$.The history and belief state both satisfy the Markov property. A POMDP can be reduced to an (infinite) history tree and an (infinite) belief state tree.

Chapter 3

Planning - Dynamic Programming

Dynamic Programming (DP) breaks one problem down into smaller sub-problems in order to solve them. Then, you can combine the solutions of the sub-problem to answer the problem.

DP is used for planning within an MDP. This means the method assumes there is full knowledge of the MDP. It is technically not Reinforcement Learning yet, since we don't discover an initially unknown environment. It can be used for both *prediction* (the input is the MDP and policy, returns value function) and *control* (input only the MDP, returns optimal policy and value function).

3.0.1 Prediction

Iterative policy evaluation can be used to perform prediction in an MDP (evaluate how good a policy is). On a high level, it works by backing up the Bellman expectation from the states in which rewards are observed. The pseudocode below shows how the algorithm works.

Algorithm 1 Iterative policy evaluation

Require: the MDP, policy π
  $i \Leftarrow 0$, $v_0(s) \Leftarrow 0$ for each state s
  while not converged **do**
    for each state s **do**
      $v_{i+1}(s) \Leftarrow \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_i(s') \right)$
    end for
    $i \Leftarrow i + 1$
  end while
  return v_i

When this algorithm converges, we have obtained the value function v_π.

3.0.2 Control

Now that we have a way to evaluate how good a certain policy is, we can start improving it. There are two main ways of performing control using DP. The first method that will be discusses is called **Policy Iteration**.

At each iteration, this method consists of two components: **policy evaluation** and **policy improvement**. The whole algorithm works in the following way

Algorithm 2 Iterative policy evaluation

Require: the MDP, policy π
  while not converged **do**
    $v^\pi \Leftarrow$ iterative policy evaluation with π
    $\pi' \Leftarrow greedy(v_\pi)$
    $\pi = \pi'$
  end while
  return v_π, π

The image below shows how this algorithm works on a higher level.

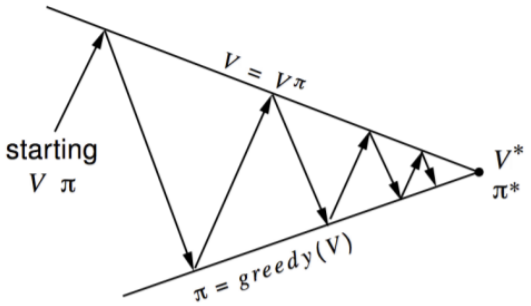


Figure 3.1: Policy Iteration

Why does acting greedily with respect to the obtained v_π improve the policy? There is a relatively simple proof for this. Say we choose a deterministic policy $\pi(s)$. Acting greedily would mean we create a new policy $\pi'(s) = \arg\max_{a \in A} q_\pi(s, a)$. This means that $q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$. So we know $q_\pi(s, \pi'(s)) \geq v_\pi(s)$. Therefore, $v_{\pi'}(s) \geq v_\pi(s)$.

This means if the improvement stops, we must have satisfied the Bellman optimality equation.

Policy iteration can be generalized. Before we performed policy evaluation until we obtained v_π. However, this is not necessary. The next DP control algorithm, **Value Iteration**, is a special variant of policy iteration, updating the policy after every step (so not after it converges to v_π). You can basically use **any** policy evaluation and policy improvement algorithm to perform policy iteration.

Lets now start talking about Value Iteration. This method works because of the *Principle of Optimality*, which states the following: An optimal policy can be subdivided into two components

- An optimal first action A_*
- An optimal policy from successor state S'

This means that if we know the solution to $v_*(s')$, we can find $v_*(s)$ by performing the following one-step lookahead. This is possible due to the Bellman optimality equations.

$$v^*(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \right)$$

The intuition is that you can start with the final rewards and work your way backwards.

Algorithm 3 Value iteration

Require: the MDP
  $i \Leftarrow 0$, $v_0(s) \Leftarrow 0$ for each state s
  while not converged **do**
    for each state s **do**
      $v_{i+1}(s) \Leftarrow \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_i(s') \right)$
    end for
    $i \Leftarrow i + 1$
  end while
  $v_* \Leftarrow v_i, \pi_* \Leftarrow \arg\max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$
  return v_*, π_*

Observe that the policy is extracted by acting greedily with respect to the computed q-values for any state. For any intermediate value functions, this might not be true. These in-between value function might not correspond to any policy.