Reinforcement Learning

Kristian Wichmann

February 2, 2018

1 Basics of reinforcement learning

Reinforcement learning deals with an *agent* making *decisions* over time in dealing with some *environment*, to maximize some cumulative, scalar *reward*.

An example would be playing a video game. Here, the player is the agent. The control inputs are the decisions. The game and its internal logic is the environment. And the score is the reward.

Sometimes, a reward will be greater if it is postponed - it is not always advantagegous to reap any immediate reward. In other words, reinforcement learning algorithms will not benefit from being greedy.

1.1 The reward hypothesis

The basis of all reinforcement learning is the *reward hypothesis*, which states:

> All goals can be described by the maximization of some cumulative, expected reward.

1.2 Observation, action, and reward

At at given timestep t, a reinforcement learning agent gets some input; an *observation* O_t about the environment. It then takes an *action* A_t. And finally it gets a scalar *reward* R_t.

1.3 History and state

At any given time step t, the agent has a *history* H_t, which simply consist of all the observations, actions, and rewards that have happened so far:

$$H_t = O_1, A_1, R_1, O_2, A_2, R_2, \cdots, O_t, A_t, R_t, \tag{1.1}$$

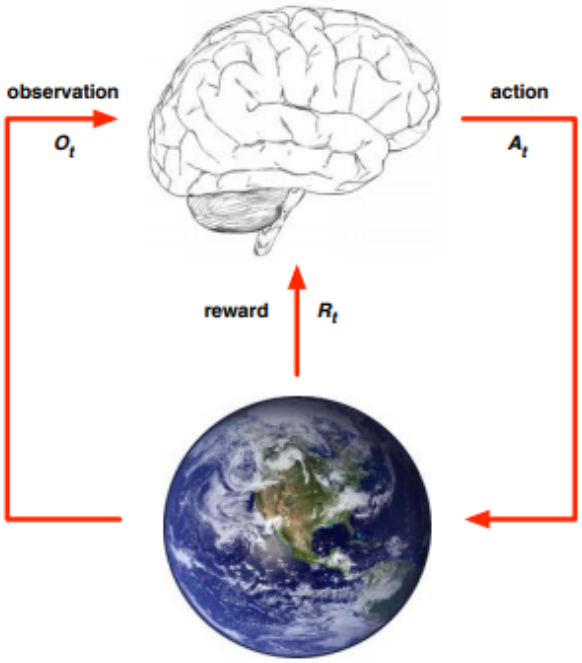

Figure 1: The interplay between agent and environment.

A *state* is a way for to parse this history into a more meaningful form. Formally, the state representation is simply a function of the history:

$$S_t = f(H_t) \tag{1.2}$$

It is very important to note, that this is generally distinct from the *environment state* H_t^e. The environment state contains complete information about the environment, including data and mechanisms that may be hidden to the agent. Hence H_t^e can depend on other things than just the historySuch information is *private* to the environment.

The *agent state* S_t^a on the other hand is the internal representation the agent uses to decide on which actions to take. Once again, it can be any function of history:

$$S_t^a = f(H_t) \tag{1.3}$$

1.4 Markov states

A state S_{t+1} is called a *Markov state* or an *information state* if it only depends on the state of the previous time step. Expressed probabilistically:

$$\mathbb{P}[S_{t+1}|S_1, S_2, \cdots S_t] = \mathbb{P}[S_{t+1}|S_t] \tag{1.4}$$

In other words, we don't need the entire history to decide on an action: Knowing the present is enough. Or put another way:

> The future is independent of the past, given the present.

Or:

> The state is a *sufficient statistic* of the future.

The environment state is always Markov, as by definition it contains all information about what can happen next. Similarly, the state consisting of the entire history is trivially Markov as well.

1.5 Full observability

This is the case where, in fact, we can observe everything about the environment and its inner workings. So that:

$$O_t = S_t^a = S_t^e \tag{1.5}$$

Sometimes this is reasonable. Sometimes not. But it will be a useful theoretical situation. This is known as a *Markov decision process* or MDP for short.

When this condition is not fulfilled, we speak about *partial observability* or a *partially observable evironment*. Here the agent only indirectly observes the invironment. This situation is known as a *partially observable Markov decision process* or POMDP for short.

1.6 State example: Bayesian beliefs

This state representation can be seen as a current best bet at what the actual environment state is. In other words, it is represented by Bayesian probabilities, which may then be updated over time using Bayes' rule:

$$S_t^a = (\mathbb{P}[S_t^e = s_1], \mathbb{P}[S_t^e = s_2], \cdots, \mathbb{P}[S_t^e = s_n]) \tag{1.6}$$

1.7 State example: Recurrent neural net

Here, the state S_t^a is a linear combination of the observation O_t and the state of the last time step S_{t-1}^a, followed by a non-linear *activation function* σ:

$$S_t^a = \sigma(W_O O_t + W_S S_{t-1}^a) \tag{1.7}$$

Here, the W's are weight matrices with sizes corresponding to the dimenstionality of observations and state vectors. Typical choices for σ are sigmoid, tanh, or rectified linear unit.

2 Components of an agent

A reinforcement learning agent may contain one or more of the following components:

- *Policy*: The agent's behaviour function. Shows how the agent gets from its state to deciding on an action.
- *Value function*: A measure of how desirable it is to be in a given state, or perform a given action.
- *Model*: The agent's representation of the environment.

We'll examine each of these in greater detail below:

2.1 Policy

A policy π is a mapping from state to action. For a *deterministic* policy, this is an ordinary function:

$$\pi(s) = a \tag{2.1}$$

So the state s is always mapped to the action a. We should ideal choose π so that the reward is maximized.

But a policy can also be *stochastic*, i.e. probabilistic. In this case π takes the form of conditional probabilities:

$$\pi(a|s) = \mathbb{P}[A = a|S = s] \tag{2.2}$$

2.2 Value function

A value function V_π is a prediction of the future reward for state s under a given policy π:

$$V_\pi(s) = \mathbb{E}_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | S_t = s] \tag{2.3}$$

Here γ is a *discounting factor*, between 0 and 1. 0 means that we only care about the immediate reward, while values approaching 1 means that we care progressively more about long run rewards.

2.3 Model

In this context, a model tries to predict the evolution of the environment. These can take two different forms:

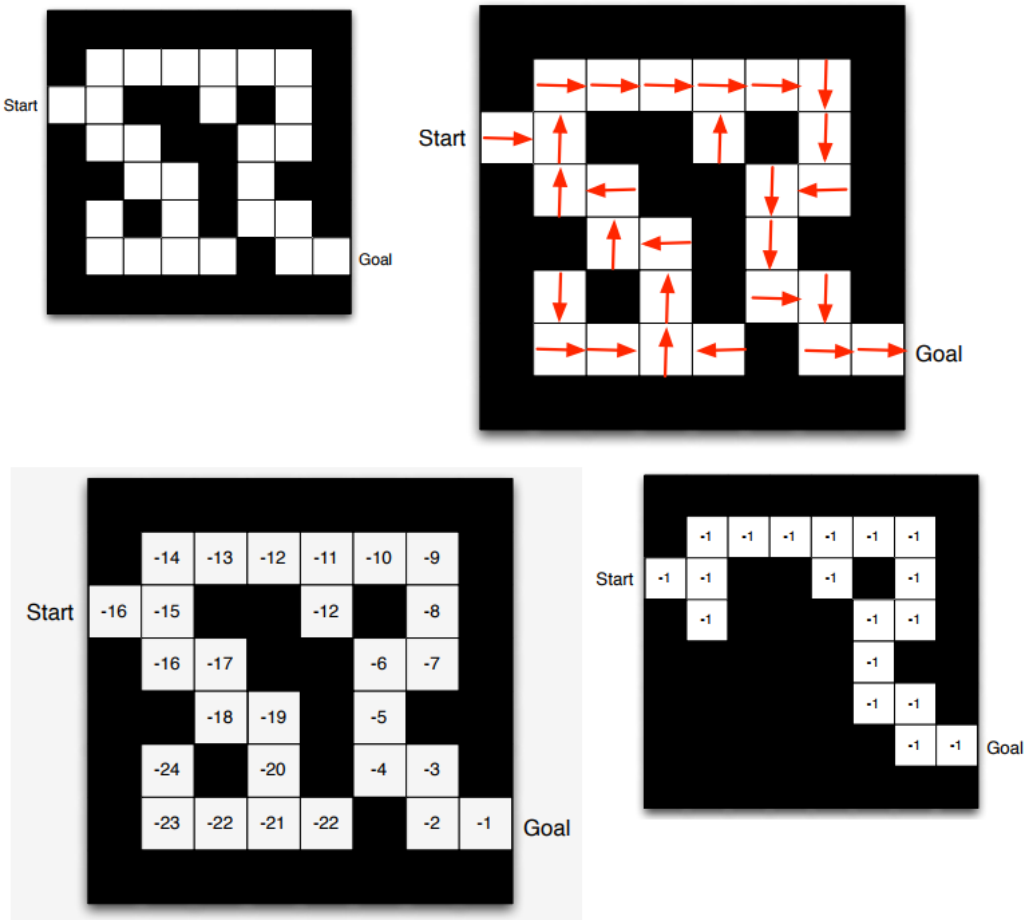

Figure 2: Upper left: Maze example. Upper right: Policy for maze. Lower left: Value function for maze. Lower right: Reward prediction for maze.

- *Transition prediction* $\mathcal{P}$ models the change of state of the environment:

$$\mathcal{P}^a_{ss'} = \mathbb{E}[S_{t+1} = s' | S_t = s, A_t = a] \tag{2.4}$$

- *Reward prediction* $\mathcal{R}$ models the change in (immediate) reward:

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] \tag{2.5}$$

2.4 Example: Maze

As an example, consider an agent that has to find its way though the maze in the upper left of figure 2. The problem has the following qualities:

- Rewards: We want the agent to get through the maze as quickly as possible, so each time step has a reward of -1.
- Actions: The agent can move up, down, left, or right in each time step.
- State: The state is the agents current position square in the maze.

A good policy for solving the problem is shown in the upper right. Here the arrows dictates the action given the state (position).

The lower left shows the value function corresponding to the policy: The future reward is negative the number of steps needed to get out of the maze from the current state (position). An optimal action can be chosen greedily, by picking the possibility with the lowest value of the function.

The lower right shows a reward prediction model for the maze. This one technically depends on both state and action, but as we no it is -1 no matter what.

2.5 Categorization of agents

Based on the above, we can broadly categorize agents into several categories:

- *Value based agents* are based on a value function - with the policy being implicit. I.e. in each step we can greedily pick the action with the highest future reward, based directly on the value function.
- *Policy based agents* on the other hand has the policy as its key ingredient. Here, the policy mapping directly shows what action to take in each state.
- *Actor critic agents* takes both a poicy and a value function into account. Ideally the "best of both worlds".
- *Model free agents* may by policy and/or value function based, but has no model! I.e. it doesn't try to make predictions about the environment.
- *Model agents* may by policy and/or value function based, but includes modelling.

3 Reinforcement learning and planning

There's two fundamental types of problems in sequential decision making: *Reinforcement learning* and *planning*.

Reinforcement learning can be viewed as dumping the agent into an alien environment it initially knows nothing about. The agent then begins to interact with the environment, eventually improving its policy for doing so.

Planning is the situation where the agent starts with a perfect model of the environment. The agent then uses this model to make calculations without any external interaction, which then informs policy.

So in planning, we can in principle look many steps ahead, determining states and rewards (or expectation values thereof) and use this to pick optimal actions. In other words, a tree search.

3.1 Exploration versus exploitation

So reinforcement learning is very much a trial-and-error process. The agent should hopefully learn an effective policy from interacting with the environment. This requires *exploration*. Ideally without losing too much reward along the way.

Exploration finds more information about the environment.

Planning, on the other hand relies on *exploitation* of the complete knowledge of the environment.

Exploitation uses known information to maximize reward.

Usually, a combination of exploration and exploitation is necessary for achieving success. So there's a balance between the two, sometimes knows as *exploration-exploitation tradeoff*. This consideration is unique to reinforcement learning as opposed to machine learning as a whole.

3.1.1 Examples

Situation: You're going out to eat.

- Exploitation: Going to your favorite restaurant.
- Exploration: Trying a new restaurant.

Situation: Online banner advertisement.

- Exploitation: Show the most succesful advert so far.
- Exploration: Display a different advert.

Situation: Drilling for oil.

- Exploitation: Drill at the best known location.
- Exploration: Drill at a new location.

3.2 Prediction and control

Prediction is trying out a given policy in practice and evaluating the results.

Control on the other hand is optimizing the future, i.e. finding the best strategy.

Usually in reinforcement learning, we have to do prediction in order to control.

4 Markov reward processes

Recall that a Markov decision process (or MDP for short) is a case where everything about the environment is known - the agent state is the same as the environment state. This also means that it has the Markov property.

It turns out that almost all reinforcement learning problems can be restated as MDP's. For instance:

- Optimal control primarily deals with continuous MDP's.
- Partially observable problems can be conversted into MDP's.
- *Bandits* are MDP's with one state.

In this section, we will deal with a simpler case - known as Markov reward processes - and the tackle the general problem later.

4.1 Transition probability matrix

Consider for a moment an agent with a state having the Markov property, which simply takes the same (or no) action in every time step. Then given the initial state s there is a probability it will end up in state s' in the next time step - it can depend only on s and s' per the Markov property:

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s] \tag{4.1}$$

These propabilities form a matrix; the *transition probability matrix* $\mathcal{P}$:

$$\mathcal{P} = \begin{pmatrix} \mathcal{P}_{11} & \mathcal{P}_{12} & \cdots & \mathcal{P}_{1n} \\ \mathcal{P}_{21} & \mathcal{P}_{22} & \cdots & \mathcal{P}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \mathcal{P}_{n2} & \cdots & \mathcal{P}_{nn} \end{pmatrix} \tag{4.2}$$

This form assumes a finite number of states n, but in principle this can be infinite.

Because the entries are probabilites, each row of $\mathbb{P}$ must sum to 1:

$$\forall i \in \{1, 2, \ldots, n\} : \sum_{j=1}^{n} \mathcal{P}_{ij} = 1 \tag{4.3}$$

Such a system - represented by the tuple $\langle S, \mathcal{P} \rangle$ - is known as a *Markov process* or *Markov chain*.

4.2 Markov reward processes

A *Markov reward process* (or MRP) is a Markov process with a finite set of states and a *reward function* $\mathcal{R}$ as well as a discount factor $\gamma \in [0, 1]$. The reward function is defined as the expectation value of the next reward given the current state:

$$\mathcal{R} : s \mapsto \mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s] \tag{4.4}$$

So a Markov reward process is the tuple $\langle S, \mathcal{P}, \mathcal{R}, \gamma \rangle$

4.3 Return and value function

The *return* of a given outcome of a Markov reward process[1] is:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{n=1}^{\infty} \gamma^{n-1} R_{t+n} \tag{4.5}$$

In other words, the return is the discounted future reward. This is the quantity we wish to maximize.

The value function, as we've already discussed above, is the expectation value of the return:

$$v(s) = \mathbb{E}[G_t | S_t = s] \tag{4.6}$$

[1] Or of any system with rewards and a discounting factor γ, really.

4.4 The Bellman equation for MRP's

We can now use equations 4.6, 4.5, and 4.4 to get a recursive formula for the value function:

$$
\begin{aligned}
v(s) =& \mathbb{E}[G_t|S_t = s] = && (4.7)\\
& \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s] = && (4.8)\\
& \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots)|S = s] = && (4.9)\\
& \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] = && (4.10)\\
& \mathbb{E}[R_{t+1}|S_t = s] + \gamma \mathbb{E}[G_{t+1}|S_t = s] = && (4.11)\\
& \mathcal{R}_s + \gamma \sum_{s'=1}^{n} \mathbb{P}[G_{t+1}|S_{t+1} = s']\mathbb{P}[S_{t+1} = s'] = && (4.12)\\
& \mathcal{R}_s + \gamma \sum_{s'=1}^{n} \mathcal{P}_{ss'} v(s') && (4.13)
\end{aligned}
$$

Here, $\mathcal{R}_s$ is the reward for leaving state s - which might seem somewhat backwards. Summing it up:

$$v(s) = \mathcal{R}_s + \gamma \sum_{s'=1}^{n} \mathcal{P}_{ss'} v(s') \tag{4.14}$$

Treating the value and reward functions as vectors v and $\mathcal{R}$, this can be rewritten in matrix form:

$$v = \mathcal{R} + \gamma \mathcal{P} v \tag{4.15}$$

This equation can be solved using the usual methods:

$$v = \mathcal{R} + \gamma \mathcal{P} v \Leftrightarrow (I_n - \gamma \mathcal{P})v = \mathcal{R} \Leftrightarrow v = (I_n - \gamma \mathcal{P})^{-1}\mathcal{R} \tag{4.16}$$

This direct solution however, is only tractable for small n, as the run time for matrix inversion is $O(n^3)$. Instead, there is a number of other, faster possibilities, including:

- Dynamic programming.
- Monte-Carlo evaluation.
- Temporal difference learning.

5 Markov decision processes

Now for the full MDP treatment. This case is very similar to the MRP case, but now there is several different actions to choose from, collectively written as $\mathcal{A}$. To each action $a \in \mathcal{A}$, corresponds a transition probability matrix $\mathcal{P}^a$. So formally, a Markov decision process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}^a$ is a probability transition matrix corresponding to the action $a \in \mathcal{A}$.
- $\mathcal{R}_s^a$ is the expected reward for taking action $a \in \mathcal{A}$ when in state $s \in \mathcal{S}$.
- γ is the discounting factor.

5.1 Policy

Recall that a policy π is a way to choose an action $a \in \mathcal{A}$ given the state $s \in \mathcal{A}$:

$$\pi(a|s) = \mathcal{P}[A_t = a | S_t = s] \tag{5.1}$$

Theorem 5.1. *Given a Markov decision process $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π for it, then:*

- *The state sequence $S_1, S_2, \ldots$ is a Markov process with probability transition matrix:*
$$\mathcal{P}_{ss'}^{\pi} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}_{ss'}^a \tag{5.2}$$
- *The combined state/reward sequence $S_1, R_1, S_2, R_2, \ldots$ is a Markov reward process with reward function:*
$$\mathcal{R}_s^{\pi} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}_s^a \tag{5.3}$$

Proof. This follows from the law of total probability. □

This means that any MDP with a policy can always be reduced to a MRP, should we so wish.

5.2 State- and action-value function

The *state-value function* $v_\pi(s)$ of a MDP is the expected return starting from state s and then following policy π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{5.4}$$

The *action-value function* $q_\pi(s, a)$ is the expected return starting in state s, taking action a and then following policy π, i.e.:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{5.5}$$

5.3 Bellman's expectation equation

It turns out that state- and action-value functions obey recursive Bellman equations as well. First the state-value function, where we explicitly write out the sum over the possibilities of the first action of the π policy:

$$\begin{aligned} v_\pi(s) =& \mathbb{E}_\pi[G_t|S_t = s] = & (5.6)\\ & \sum_{a\in\mathcal{A}} \mathbb{P}[S_t = s, A_t = a]\mathbb{E}_\pi[G_t|S_t = s, A_t = a] = & (5.7)\\ & \sum_{a\in\mathcal{A}} \pi(a|s)q_\pi(a, s) & (5.8)\end{aligned}$$

So the state-value function involved the action-value function. Let's examine the latter:

$$\begin{aligned} q_\pi(a, s) =& \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = & (5.9)\\ & \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s, A_t = a] = & (5.10)\\ & \mathbb{E}_\pi[R_{t+1}|S_t = s, A_t = a]+ & (5.11)\\ & \quad \gamma\mathbb{E}_\pi[R_{t+2} + \gamma R_{t+3} + \cdots |S_t = s, A_t = a] = & (5.12)\\ & \mathcal{R}_s^a + \gamma\mathbb{E}_\pi[G_{t+1}|S_t = s, A_t = a] & (5.13)\end{aligned}$$

Now, write out the sum over that state at $t + 1$:

$$\begin{aligned} q_\pi(a, s) =& \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathbb{P}[S_{t+1} = s', S_t = s, A_t = a]\mathbb{E}_\pi[G_{t+1}|S_{t+1} = s'] = & (5.14)\\ & \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') & (5.15)\end{aligned}$$

This means that we have a set of coupled equations:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(a, s), \quad q_\pi(a, s) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \tag{5.16}$$

Inserting the latter into the former we get an equation for state-value function only:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right] \tag{5.17}$$

Similarly, we can get an equation for action-value function by inserting the former into the latter:

$$q_\pi(a, s) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \left[\sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(a', s') \right] \tag{5.18}$$

Do note the renaming of indices, though.