# 8a: Reinforcement Learning

## Week 8: Overview

The topic for this week is Reinforcement Learning (RL). We will see how reinforcement learning tasks can be formally defined, compare different models of optimality, and discuss the need for exploration and the problem of delayed rewards. We will introduce the classic Temporal Difference learning (TD-learning) and Q-learning algorithms, and see how they can be applied to games like backgammon.

## Weekly learning outcomes

By the end of this module, you will be able to:

- explain the difference between supervised learning, reinforcement learning, and unsupervised learning
- describe the formal definition of a reinforcement learning task
- identify different models of optimality
- explain the need for exploration in reinforcement learning
- describe the TD-learning and Q-learning algorithms
- apply Q-learning to simple tasks.

## Reinforcement Learning

We have previously discussed supervised learning, where pairs of inputs and target outputs are provided and the system must learn to predict the correct output for each input.

There are many situations where we instead want to train a system to perform certain actions in an environment, in order to maximise a reward function. These situations include, for example, playing a video game, allocating mobile phone channels or other resources dynamically, driving a car or flying a helicopter.

Supervised learning can sometimes be used for this purpose, if we construct a training set of situation-action pairs (for example, a database of game positions and the move chosen by a human expert, or sensor readings from a motor car and the steering direction chosen by a human driver). This process is sometimes called Behavioral Cloning.

However, it is often better if the system can learn by purely by self-play, without the need for training data from a human expert.

## Reinforcement Learning Framework

Reinforcement Learning (RL) can be formalised in terms of an Agent interacting with its Environment.

The **Environment** includes a set $\mathrm{S}$ of states and a set $\mathrm{A}$ of actions. At each time step $t$, the agent is in some state $s_t$. It must choose an action $a_t$ , whereupon it goes into state $s_{t+1} = \delta(s_t, a_t)$ and receives reward $r_t = R(s_t, a_t)$

The **Agent** chooses its actions according to some **policy** $\pi : \mathrm{S} \to \mathrm{A}$.

The aim of Reinforcement Learning is to find an **optimal** policy $\pi^*$ which maximises the cumulative reward.

In some cases, the Environment may be probabilistic or **stochastic** $-$ meaning that the transitions and/or rewards are not fully determined, but instead occur randomly according to some probability distribution:

$$\delta(s_{t+1} = s \mid s_t, a_t), \quad R(r_t = r \mid s_t, a_t)$$

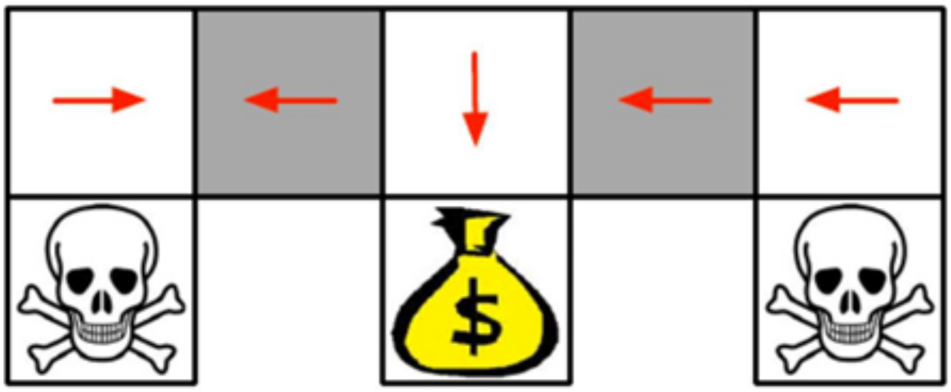
The Agent can also employ a probabilistic policy, with its actions chosen according to a probability distribution:

$$\pi(a_t = a \mid s_t)$$

## Probabilistic Policies

The main benefit of a probabilistic policy is that it forces the agent to explore its environment more comprehensively while it is learning.

However, probabilistic policies may have additional benefits in other contexts, such as partially observable or multi-player environments.



Consider, for example, a situation in which the Agent is able to observe that it is in one of the grey squares, but is not able to determine which grey square it is in. In this case, the policy of moving left or right with equal probability from the grey square will perform better than any deterministic policy.

In two-player games like rock-paper-scissors, a random strategy is also required in order to make agent choices unpredictable to the opponent.

## Models of Optimality

What exactly do we mean when we say that an RL Agent should choose a policy which maximises its future rewards?

The old saying "a fast nickel is worth a slow dime" reminds us that people often prefer to receive a smaller reward sooner instead of a larger reward later. Economists use surveys to gauge people's preferences in this regard: "Would you prefer ten dollars today, or 15 dollars next week? How about 50 dollars today compared to 100 dollars next year?" Responses to these surveys can be used to estimate a number $\gamma$ between $0$ and $1$ that is chosen such that one dollar in the current timestep is considered equally desirable with $\gamma$ dollars in the next timestep. In Reinforcement Learning, this number $\gamma$ is called the **discount factor** and is used to define the infinite discounted reward, which can be compared with average reward and finite horizon reward.

$$\text{Finite horizon reward: } \sum_{i=0}^{h-1} r_{t+i}$$

$$\text{Infinite discounted reward: } \sum_{i=0}^{\infty} \gamma^i r_{t+i}, \quad 0 \le \gamma < 1$$

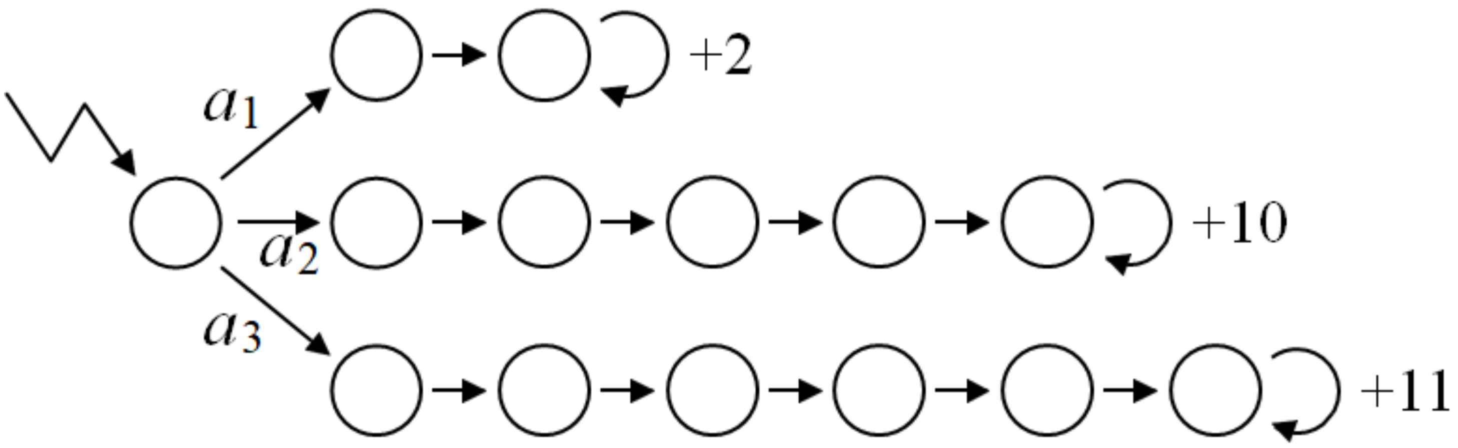
$$\text{Average reward: } \lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h-1} r_{t+i}$$

We normally try to maximise the infinite discounted reward, because it is easier to analyse theoretically and also works well in practice.

The finite horizon reward is easy to compute, but may lead to bad decisions by failing to take into account rewards which are just over the horizon.

Average reward is hard to deal with because we cannot sensibly choose between a small reward soon and a large reward very far in the future — for example, 100 dollars today compared with a million dollars, 50 years from now.

## Comparing Models of Optimality



This environment illustrates how the choice of action may depend on the model of optimality. An agent trying to maximise the finite horizon reword with $h$=4 would prefer action $a_1$ because it is the only action which will gain any reward within the first four time steps. An agent maximising average reward will prefer action $a_3$ because, after the first few time steps, it will be getting a reward of +11 every timestep which is larger than +10 for an agent who chose action $a_2$. An agent maximising infinite discounted reward with $\gamma = 0.9$ will prefer action $a_2$. One way to see this is as follows: Consider Agent A$_2$ choosing action $a_2$, compared to Agent A$_3$ choosing action $a_3$. The reward of 11 received by A$_3$ in timestep 7 would be equivalent to 9.9 in timestep 6 and therefore slightly less than the $10 received by A$_2$ in timestep 6. By the same logic, the $10 received by A$_2$ in timestep $n$ is always slightly more valuable than the $11 received by A$_3$ in timestep $n$+1.

## Value Function

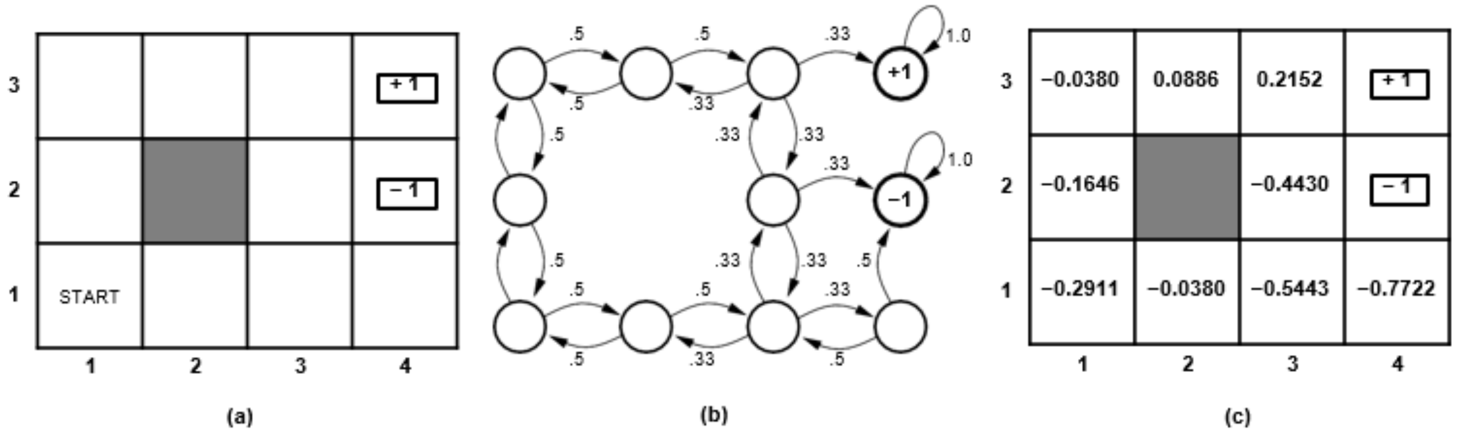
Every policy $\pi$ determines a Value Function $V^\pi : S \to \mathrm{R}$ where $V^\pi(s)$ is the average discounted reward received by an agent who begins in state $s$ and chooses its actions according to policy $\pi$.

For small environments, we can compute the value function using simultaneous equations. For larger environments, we need a way of learning it, through experience.

If $\gamma = 0.9$, the value of the last node in the middle row of the above diagram is $\frac{10}{1-\gamma} = 100$. The value of the last node in the lowest row is $\frac{11}{1-\gamma} = 110$. The value of the second last node in the lower row is $99$, which can be obtained from the value of the last node either by subtracting $11$ or multiplying by $\gamma$.

## Value Function Example

This image shows the value function $V_\pi$ in a grid world environment, where $\pi$ is the policy of choosing between available actions uniformly randomly.



## Exploration and Delayed Reinforcement

> I was born to try ...
> But you've got to make choices
> Be wrong or right
> Sometimes you've got to sacrifice the things you like.
> — Delta Goodrem

### Multi-Armed Bandit Problem

The special case of a stochastic reinforcement learning environment with only one state is called the **Multi-Armed Bandit Problem**, because it is like being in a room with several (friendly) slot machines (also called "one-armed bandits") for a limited time, and trying to collect as much money as possible. We assume that each slot machine provides rewards chosen from its own (stationary) distribution, and that in each time step we are able to pull the lever on only one machine.

### Exploration / Exploitation Tradeoff

After pulling the levers a few times and observing the rewards received, it makes sense that most of the time we should choose the lever (action) which we think will give the highest expected reward, based on previous observations.

However, in order to ensure convergence to the optimal strategy, we must occasionally choose something different from our preferred action. Perhaps the simplest way to achieve this is known as an **epsilon-greedy strategy** , where with probability $1-\varepsilon$ we choose what we think is the best action, and with probability $\varepsilon$ (typically, 5%) we choose a random action. More sophisticated strategies also exist such as Thompson Sampling, Upper Confidence Bound (UCB) algorithms, or choosing from a Softmax (Boltzmann) distribution based on the average reward so far observed for each action.

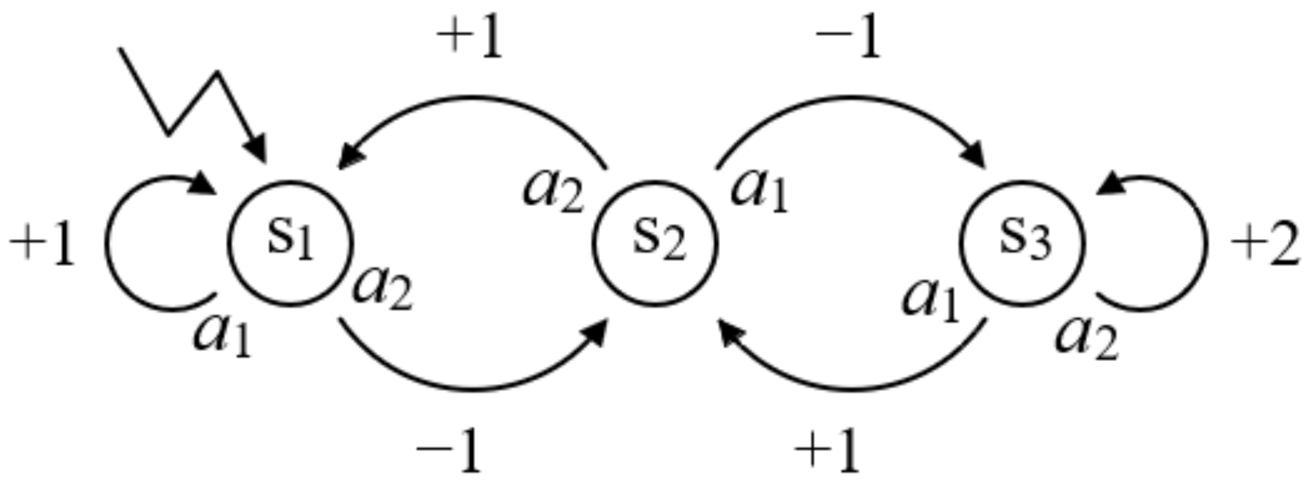
$$P(a) = \frac{e^{R(a)/T}}{\sum_{b \in A} e^{R(b)/T}}$$

You may have noticed that we make the same kind of judgements about exploration versus exploitation in everyday situations. Should you eat at the old restaurant you have visited many times, or should you try the newly opened restaurant across the street which might be worse but might be much better?

---

## Delayed Reinforcement

The need for exploration also applies in the general case where there are multiple states, and where we may need to take a whole sequence of actions in order to get to the state from which we can obtain a high reward.



Recall that every policy $\pi$ determines a Value Function $V^\pi : S \to \mathrm{R}$ where $V^\pi(s)$ is the expected discounted reward received by an agent who begins in state $s$ and chooses its actions according to policy $\pi$.

If $\pi = \pi^*$ is optimal, then $V^*(s) = V^{\pi^*}(s)$ is the maximum (expected) discounted reward obtainable from state $s$. Knowing this optimal value function can help to determine the optimal policy.

## Computing the Value Function

Let $\gamma = 0.9$ and consider the policy $\pi : S_1 \mapsto a_2, S_2 \mapsto a_1, S_3 \mapsto a_2$. We have

$$\begin{aligned} V^\pi(S_3) &= \frac{2}{1-\gamma} = 20 \\ V^\pi(S_2) &= -1 + \gamma V(S_3) = -1 + 18 = 17 \\ V^\pi(S_1) &= -1 + \gamma V(S_2) = -1 + 15.3 = 14.3 \end{aligned}$$

For this example, the policy $\pi$ shown above must be the **optimal** policy, because the value function for states $S_1$ and $S_2$ under any other policy would not be more than $10$. Therefore, the optimal value function is

$$V^* : S_1 \mapsto 14.3, S_2 \mapsto 17, S_3 \mapsto 20$$

If we were given this value function $V^*$, we could use it to determine the optimal policy $\pi^*$ provided we also know the reward function $R : S \times A \to \mathbf{R}$ and the transfer function $\delta : S \times A \to S$. This information is sometimes called the "World Model".

## Q-Function

The Q-function is a more sophisticated version of the value function which enables an agent to act optimally without needing to know the World Model.

For any policy $\pi$, the **Q-function** $Q^\pi(s, a)$ is the expected discounted reward received by an agent who begins in state $s$, first performs action $a$ and then follows policy $\pi$ for all subsequent timesteps.

If $\pi = \pi^*$ is optimal, then $Q^*(s, a) = Q^{\pi^*}(s, a)$ is the maximum (expected) discounted reward obtainable from $s$, if the agent is forced to take action $a$ in the first timestep but can act optimally thereafter.

If the optimal Q-function $Q^*$ is known, then the optimal policy is given by

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

The Q-function for the environment shown above can be computed as follows:

$$\begin{aligned}
Q(S_1, a_1) &= 1 + \gamma V(S_1) = 1 + 0.9 \times 14.3 = 13.87 \\
Q(S_1, a_2) &= V(S_1) = 14.3 \\
Q(S_2, a_1) &= V(S_2) = 17 \\
Q(S_2, a_2) &= 1 + \gamma V(S_1) = 13.87 \\
Q(S_3, a_1) &= 1 + \gamma V(S_2) = 1 + 0.9 \times 17 = 16.3 \\
Q(S_3, a_2) &= V(S_3) = 20
\end{aligned}$$

## Week 8 Wednesday video