Indian Institute of Technology, Delhi

Mid-Semester Project Report

MTD-854, II Semester, 2020-21

Reinforcement Learning

Author:
Aman Arora

Supervisor:
Prof. Vikas Vikram
Singh

March 24, 2021

Reinforcement Learning
Mid Semester Project Report
MTD854 - II Semester 2020-21

Aman Arora

Contents

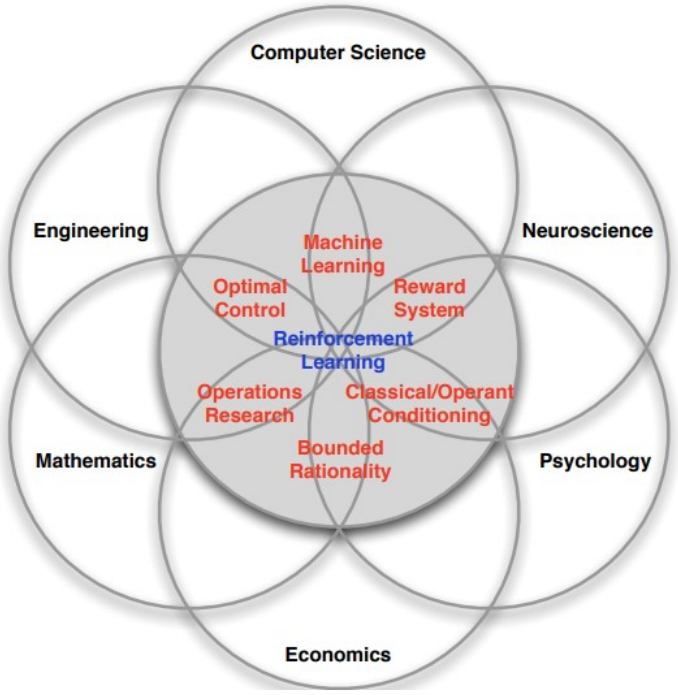
1 Introduction

1.1 What is Reinforcement Learning?

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward [1]

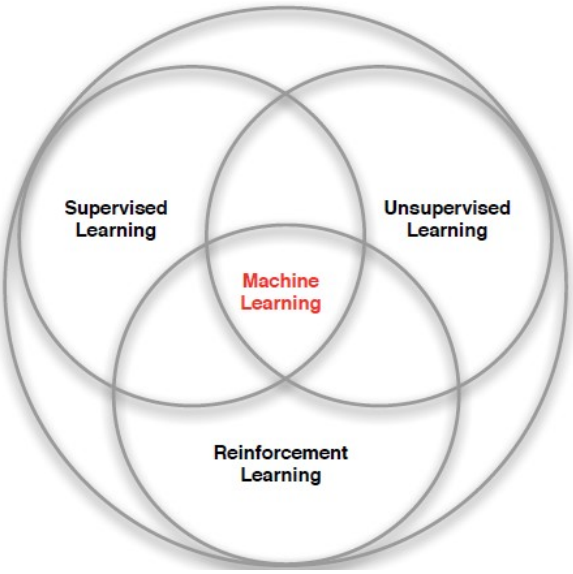
Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics—**trial-and-error search** and **delayed reward**—are the two most important distinguishing features of reinforcement learning.

Figure 1: Many faces of Reinforcement Learning



1.2 RL as a third paradigm of Machine Learning

Figure 2: Branches of Machine Learning



1.2.1 RL versus Supervised learning

Reinforcement learning is different from *supervised learning*, the kind of learning studied in most current research in the field of machine learning. Supervised learning is learning from a training set of labeled examples provided by a knowledgeable external supervisor. Each example is a description of a situation together with a specification—the label—of the correct action the system should take to that situation, which is often to identify a category to which the situation belongs. The object of this kind of learning is for the system to extrapolate, or generalize, its responses so that it acts correctly in situations not present in the training set. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act. In uncharted territory—where one would expect learning to be most beneficial—an agent must be able to learn from its own experience.

1.2.2 RL versus Unsupervised learning

Reinforcement learning is also different from what machine learning researchers call *unsupervised learning*, which is typically about finding structure hidden in collections of unlabeled data. The terms supervised learning and unsupervised learning would seem to exhaustively classify machine learning paradigms, but they do not. Although one might be tempted to think of reinforcement learning as a kind of unsupervised learning because it does not rely on examples of correct behavior, reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure. Uncovering structure in an agent's experience can certainly be useful in reinforcement learning, but by itself does not address the reinforcement learning problem of maximizing a reward signal.

We therefore consider reinforcement learning to be a **third machine learning paradigm**, alongside supervised learning and unsupervised learning and perhaps other paradigms.

1.3 Exploration vs exploitation

One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between **exploration** and **exploitation**. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it has already experienced in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward. The exploration–exploitation dilemma has been intensively studied by mathematicians for many decades, yet remains unresolved. For now, we simply note that the entire issue of balancing exploration and exploitation does not even arise in supervised and unsupervised learning, at least in the purest forms of these paradigms.

For the rest of introduction and more examples, we request the reader to refer the book *Reinforcement Learning: an Introduction* by Sutton, Barto [2]

2 Elements of Reinforcement Learning

There are four main sub-elements of a reinforcement learning system: a *policy*, a *reward signal*, a *value function*, and, optionally, a *model* of the environment.

A *reward* R_t is a scalar feedback signal that indicates how well and agent is doing at step t. The agent's job is to maximise the cumulative reward.

Definition 2.1 (Reward Hypothesis)**.** All goals can be described by the maximisation of expected cumulative reward.

Sequential Decision Making: Our *goal* in reinforcement learning is to select actions so as to maximise total future reward. Note that actions at any time step t may have long term consequences. Also, the reward may be delayed. Sometimes it may very well be the case that sacrificing immediate reward is better to gain more long-term reward. Some examples could be:

- A financial investment (may take months to mature)
- Refuelling a helicopter (might prevent a crash in several hours)
- Blocking opponent moves (might help winning chances many moves from now)

2.1 Agent and Environment

At each step t the **agent** executes action A_t, receives observation O_t and receives scalar reward R_t.
The **environment** receives action A_t, emits observation O_{t+1} and emits scalar reward R_{t+1}.
The time step t increments at the environment step.

The **history** is the sequence of observations, actions, rewards:

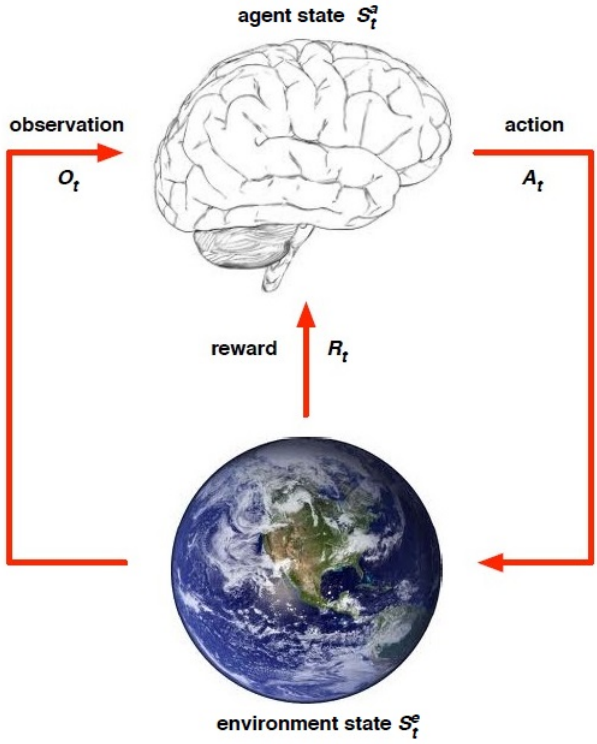
$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$$

i.e., all observable variables up to time t. What happens next will depend on the history, because based on this history, the agent selects actions and the environment selects observations and rewards.

State is the information used to determine what happens next. Formally, state is a function of the history:

$$S_t = f(H_t)$$

Figure 3: Agent-Environment interaction



The **environment state** S_t^e is the environment's private representation i.e. whatever data the environment uses to pick the next observation/reward. The environment state is usually not visible to the agent. Even if S_e is visible, it may contain irrelevant information for the agent.

The **agent state** S_t^a is the agent's internal representation i.e. whatever information the agent uses to pick the next action. It is the information used by reinforcement learning algorithms. It can be any function of history:

$$S_t^a = f(H_t)$$

An **information state** (a.k.a. **Markov state**) contains all useful information from the history.

Definition 2.2. A state S_t is **Markov** if and only if:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t]$$

This means that "the future is independent of the past given the present". So, once the state is known, the history may be thrown away.

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

i.e., the state is a sufficient statistic of the future.

2.1.1 Fully Observable Environments

In the case of full observability, the agent directly observes the environment state, i.e.

$$O_t = S_t^a = S_t^e$$

Formally, this is a **Markov Decision Process** which we will discuss more the upcoming sections.

2.1.2 Partially Observable Environments

In this case the agent *indirectly* observes the environment. Some examples are:

- A robot with camera vision isn't told its absolute location
- A trading agent only observes current prices
- A poker playing agent only observes public cards

Here we can see that the agent state is not same as the environment state.

$$S_t^a \neq S_t^e$$

Formally, this a **partially observable Markov decision process. (POMDP)**. In the case of partially observable environments, the agent must construct its 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: $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$

2.2 Major 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

2.2.1 Policy

A policy defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus–response rules or associations. In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic, specifying probabilities for each action.

- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$

2.2.2 Value function

Whereas the reward signal indicates what is good in an immediate sense, a *value function* specifies what is good in the long run. Roughly speaking, the *value* of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the *long-term* desirability of states after taking into account the states that are likely to follow and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or the reverse could be true.

Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, and the only

purpose of estimating values is to achieve more reward. Nevertheless, it is values with which we are most concerned when making and evaluating decisions. Action choices are made based on value judgments. We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run. Unfortunately, it is much harder to determine values than it is to determine rewards. Rewards are basically given directly by the environment, but values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. In fact, the most important component of almost all reinforcement learning algorithms we consider is a method for efficiently estimating values.

2.2.3 Model

A *model* of the environment is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. Methods for solving reinforcement learning problems that use models and planning are called model-based methods, as opposed to simpler model-free methods that are explicitly trial-and-error learners—viewed as almost the opposite of planning.

- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

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

2.3 RL Agent Taxonomy

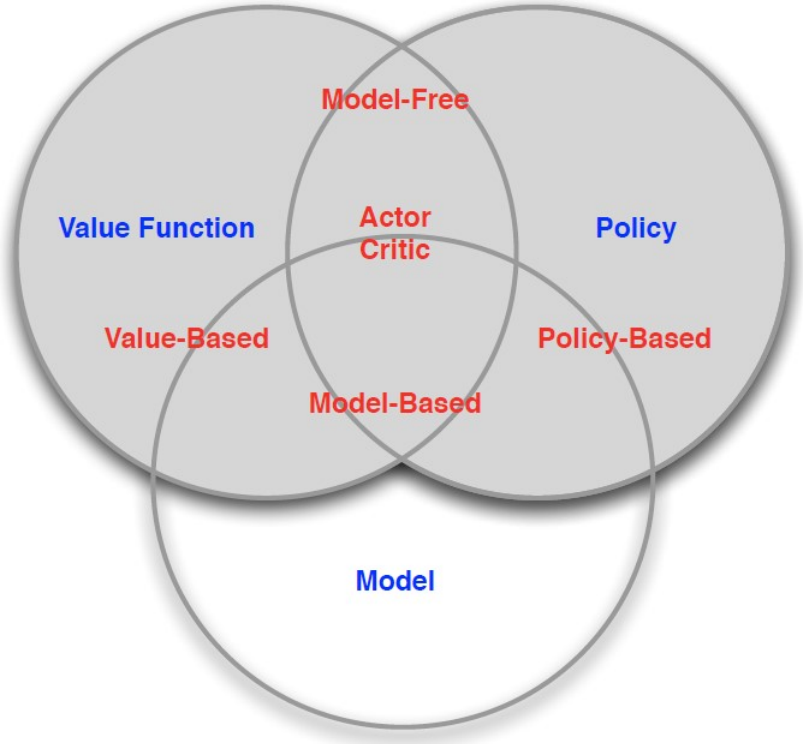
2.3.1 Categorizing RL Agents

Any RL agent can essentially be put into one of the following categories:

- Value Based

  - *No Policy (Implicit)*
  - Value Funtion
- Policy Based
  - Policy
  - *No Value Function*
- Actor Critic
  - Policy
  - Value Function

Figure 4: RL Agent Taxonomy



A fundamental distinction between RL agents is:

- Model Free

- Model Based

2.3.2 Learning and Planning

There are two fundamental problems in sequential decision making:

- Reinforcement Learning:
  Here the environment is initially unknown. The agent interacts with the environment and improves its policy. Methods for solving reinforcement learning problems that use models and planning are called model-based methods, as opposed to simpler model-free methods that are explicitly trial-and-error learners—viewed as almost the opposite of planning.

- Planning:
  A model of the environment is known and the agent performs computations with its model (without any external interaction). Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced.

3 Finite Markov Decision Processes

MDPs are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards. Thus MDPs involve delayed reward and the need to trade-off immediate and delayed reward.

MDPs are a mathematically idealized form of the reinforcement learning problem for which precise theoretical statements can be made, where the environment is *fully observable* and the current state completely characterises the process.

3.1 Markov Processes

Markov Property

"The future is independent of the past given the present"

As we had earlier discussed:

Definition 3.1. A state S_t is **Markov** if and only if:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t]$$

The current state captures all relevant information from the history. Therefore, once the state is known, the history may be thrown away, i.e. the state is a sufficient statistic of the future.

State Transition Matrix

For a Markov state s and the successor state s', the *state transition probability* is defined by:

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

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & \ddots & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

where $\mathcal{P}_{ij}$ represents transition probability from state i to state j and each row of the matrix sums to 1.

Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, \dots$ with the Markov property.

Definition 3.2 (Markov Process)**.** A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,

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

3.2 Markov Reward Process

A Markov reward process is Markov chain with values.

Definition 3.3 (Markov Reward Process)**.** A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,

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

- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

Definition 3.4 (Return)**.** The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time steps is $\gamma^k R$
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

This is important to note that most of the Markov reward and decision processes are discounted. There are a few reasons for this that we need to discuss.

Usually it is mathematically convenient to discount rewards as this avoids having infinite returns in cyclic Markov processes. Moreover, the uncertainty about the future may not be fully represented. As an example, consider the reward to be financial after investing in a stock. In this case, immmediate rewards may earn more interest than delayed rewards.

In general, animal and human behaviour shows preference for immediate reward. Sometimes, it is possible to use *undiscounted* Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.

Returns at successive time steps are related to each other in a way that is important for the theory and algorithms of reinforcement learning:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Note that although the *return* is a sum of an infinite number of terms, it is still finite if the reward is nonzero and constant (if $\gamma < 1$). For example, if the reward is a constant $+1$, then the return is

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

3.2.1 Bellman Equation for MRPs

The value function $v(s)$ gives the long-term value of state s

Definition 3.5 (Value Function)**.** The *state value function* $v(s)$ of an MRP is the expected return starting from state s

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

The value function can be decomposed into two parts:

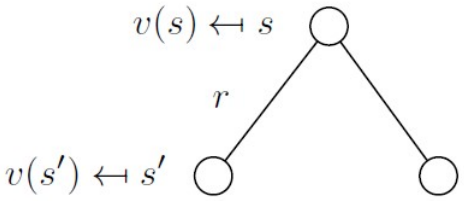
- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

$$\begin{aligned} v(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \end{aligned}$$

We now use the linearity of expectation and the *law of iterated expectation* to get:

$$\begin{aligned} v(s) &= \mathbb{E}[R_{t+1}|S_t = s] + \mathbb{E}[\gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[\mathbb{E}[G_{t+1}|S_{t+1}]|S_t = s] \\ &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[v(S_{t+1})|S_t = s] \\ &= \mathbb{E}[R_{t+1}|S_t = s] + \mathbb{E}[\gamma v(S_{t+1})|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \end{aligned} \tag{1}$$

which is an important result and is called the Bellman equation for a Markov Reward Process. We will build up from this equation when we further discuss Markov Decision Processes with a policy π



$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

3.2.2 Bellman equation in Matrix form

The Bellman equation and be expressed concisely using matrices,

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

where v is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & \ddots & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

3.2.3 Solving the Bellman equation

The Bellman equation is a linear equation and can be solved directly as:

$$\begin{aligned} v &= \mathcal{R} + \gamma \mathcal{P} v \\ (I - \gamma \mathcal{P}) v &= \mathcal{R} \\ v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R} \end{aligned}$$

The computational complexity for n states is $O(n^3)$ and therefore the direct solution is only feasible for small MRPs. For large MRPs, we use one of many available iterative methods, for example,

- Dynamic Programming
- Monte-Carlo evaluation
- Temporal-Difference Learning

3.3 Markov Decision Process

A **Markov decision process (MDP)** is a Markov reward process with *decisions*. It is an *environment* in which all states are Markov.

Definition 3.6 (Markov Decision Process)**.** A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions item $\mathcal{P}$ is a state transition probability matrix,
$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$

3.3.1 Policies and Value Functions in MDPs

Almost all reinforcement learning algorithms involve estimating *value functions*—functions of states (or of state–action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The notion of "how good" here is defined in terms of future rewards that can be expected, or, to be precise, in terms of expected return. Of course the rewards the agent can expect to receive in the future depend on what actions it will take. Accordingly, value functions are defined with respect to particular ways of acting, called policies.

Definition 3.7 (Policy)**.** A *policy* π is a distribution over actions given states,

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

Formally, a *policy* is a mapping from states to probabilities of selecting each possible action. If the agent is following policy π at time t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$. Reinforcement learning methods specify how the agent's policy is changed as a result of its experience.

Stationary and non-stationary policy

A policy fully defines the behaviour of an agent and MDP policies depend only on the current state and not the history, i.e. the policies are *stationary* (time-independent).

A *stationary policy*, π_t, is a policy that does not change over time, that is, $\pi_t = \pi, \forall t \geq 0$, where π can either be a function, $\pi : \mathcal{S} \to \mathcal{A}$ (a deterministic policy), or a conditional density, $\pi(a|s)$ (a stochastic policy). A *non-stationary policy* is a policy that is not stationary. More precisely, π_i may not be equal to π_j, for $i \neq j \geq 0$, where i and j are thus two different time steps.

There are problems where a stationary optimal policy is guaranteed to exist. For example, in the case of a stochastic (there is a probability density that models the dynamics of the environment, that is, the transition function and the reward function) and discrete-time Markov decision process (MDP) with finite numbers of states and actions, and bounded rewards, where the objective is the long-run average reward, a stationary optimal policy exists. The proof of this fact is in the book Markov Decision Processes: Discrete Stochastic Dynamic Programming (1994), by Martin L. Puterman [3]

Definition 3.8 (Value function)**.** The *value function* of a state s under a policy π, denoted by $v_\pi(s)$, is the expected return when starting in s and following π thereafter. For MDPs, we can define v_π formally by:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \text{for all } s \in S$$

where $\mathbb{E}_\pi[.]$ denotes the expected value of a random variable given that the agent follows policy π and t is any time step.

Note that the value of the terminal state, if any, is always zero. We call the function v_π the *state-value function for policy* π.

Similarly, we define the value of taking action a in state s under a policy π, denoted by $q_\pi(s, a)$, as the expected return starting from s, taking the action a, and thereafter following policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]$$

We call q_π the *action-value function* for policy π. This is the key quantity that we use to help us optimize our MDP and pick the best actions.

3.3.2 Bellman Expectation equation

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

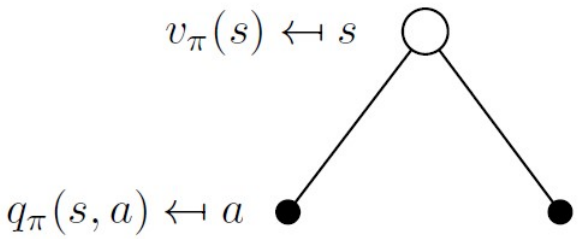
The action-value function can similarly be decomposed,

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

Let us now understand these concept pictorially with look-ahead search diagrams. First we need to understand how v_π and q_π relate to each other.

Bellman expectation equation for $v_\pi(s)$

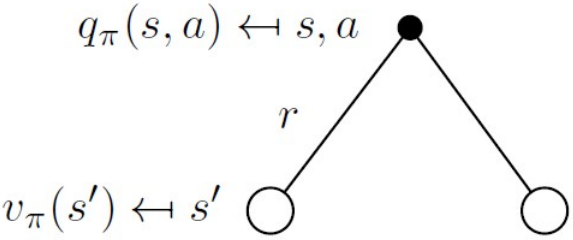
Let the black dots represent *actions* and open circles represent the *states* in the upcoming diagrams.



Consider we are at a state s whose *state value* is $v_\pi(s)$. There is some probability that we take either the right or left *action* defined by a policy π. For each of these *actions* that we might take, there's an *action value* telling us how good it is to take that action from that state. Concretely, we do a one step look-ahead, look at the action values, average them together and that tells us the value of being in that state s. So, we can write

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

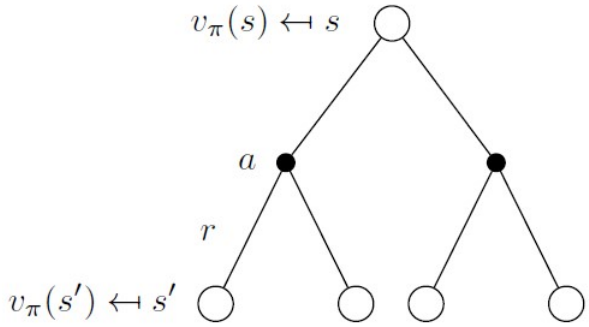
Now consider the converse part, i.e. what happens when we start-off taking some action a. Thus the root of the tree is now a state s and we are considering a specific action a that we take from this state.



We now have to average over the dynamics of the MDP, i.e. over all the situations that the *environment* can send us to, using the probabilities from our *state transition matrix*. This would give us the *action-value function* at the root.

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

Therefore, we can say the $v_\pi(s)$ tells us how good it is to be in a particular state s and $q_\pi(s,a)$ tells us how good it is to take a particular action.



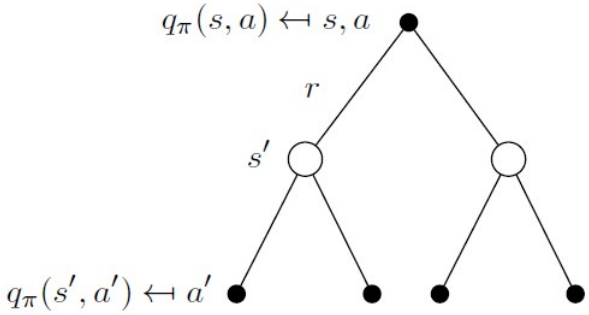
Stitching the two trees discussed above, we get a recursion which helps us understand $v_\pi(s)$ in terms of itself

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

The above equation is the **Bellman expectation equation** for $v_\pi(s)$

Bellman expectation equation for $q_\pi(s, a)$

Following the similar idea of stitching the two trees together, we get



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

The above equation is the **Bellman expectation equation** for $q_\pi(s, a)$

The basic idea utilized in deriving the above two recursive equations is that the value function at the current time step is equal to the immediate reward plus the value function of where we end up in the next time step.

3.3.3 Optimal Policies and Optimal Value Functions

Definition 3.9. The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

Definition 3.10. The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Solving a reinforcement learning task means, roughly, finding a policy that achieves a lot of reward over the long run. For finite MDPs, we can precisely define an optimal policy in the following way. Value functions define a partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states, i.e.

$$\pi \geq \pi' \quad \text{if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem. *For any Markov Decision Process*

- *There exists an optimal policy* π_* *that is better than or equal to all other policies,* $\pi_* \geq \pi$, $\forall \pi$
- *All optimal policies achieve the optimal state-value function,* $v_{\pi_*}(s) = v_*(s)$
- *All optimal policies achieve the optimal action-value function,* $q_{\pi_*}(s, a) = q_*(s, a)$

Finding an Optimal Policy

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

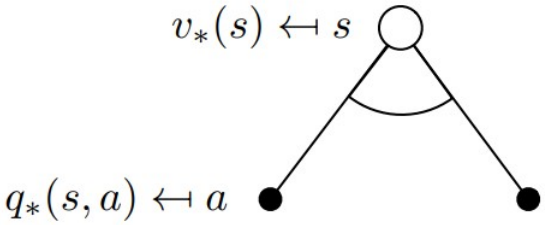
$$\pi_*(a|s) = \begin{cases} 1 & if \quad a = \arg\max_{x \in A} q_*(s, a) \\ 0 & otherwise \end{cases}$$

i.e., if we are in a state s, we pick the action a with probability 1 that maximizes $q_*(s, a)$ and that is the action that will give us the maximum possible reward. Also, we know that there is always an optimal policy. Therefore, if we know $q_*(s, a)$, we immediately have the optimal policy

3.3.4 Bellman Optimality Equations

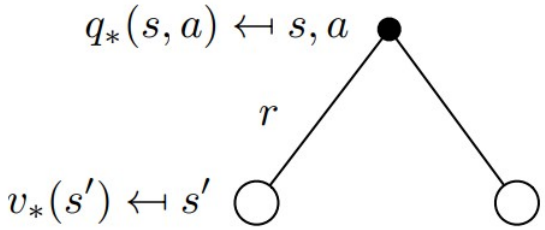
Earlier we looked at Bellman Expectation equations, now we will look at similar equations for optimal value functions. The optimal value functions are recursively related by the Bellman optimality equations as discussed below.

- What we essentially do is a one step look-ahead again. So, to find the optimal value of being in a state s, we look at each of the action value "nodes" and instead of taking the average over them like in the expectation equations, we take the maximum over values of each of the actions.



$$v_*(s) = \max_a q_*(s, a)$$

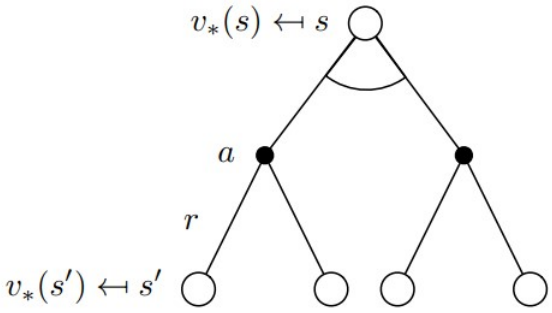
- Similar to what we did earlier in section 3.3.2, we do the other part now, i.e. what the environment can do to us,



Here we have to average over all the things that the environment can do, since we do not get to choose from either of the states we might end up in.

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

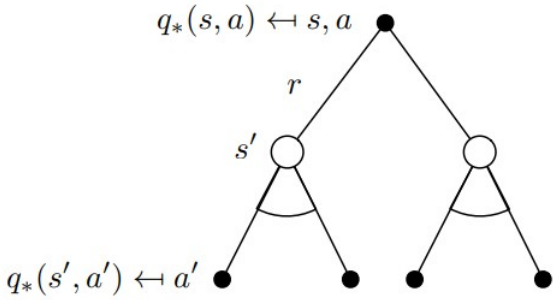
- Again, we put the two pieces together to get the recursive relationship that relates v_* to itself. So, we are essentially doing a two step look-ahead, looking over the actions we can take and what the environment can do to us taking those actions.



$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

This $v_*(s)$ tells us how good it is to be in the state s and is the **Bellman optimality equation** for v_*

- We can do the same thing starting with the action values to arrive at a recursive relationship that relates q_* values to itself.



First we consider the states that the environment can take us in, and then we get to decide and pick an action after that, maximizing over the decisions we take.

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a')$$

is the **Bellman optimality equation** for q_*

We strongly suggest the reader to watch Lecture 2 of Reinforcement Learning course by David Silver [4] to get a clear understanding where an example of a student MDP is used to explain the concepts.

3.3.5 Solving the Bellman Optimality Equation

Previously we had the Bellman Expectation equations in matrix form that we could simply solve by doing matrix inversions. Unfortunately, that does not work for Bellman Optimality equations because our equations are **non-linear** now.

There is no closed form solution (in general) and we have to use one of the many available *iterative solution methods*:

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

3.4 Prediction vs Control: A note

It is important to understand the key difference between *prediction* and *control* methods in Reinforcement Learning.

The difference between prediction and control is to do with goals regarding the policy. The policy describes the way of acting depending on current state, and in the literature is often noted as $\pi(a|s)$, the probability of taking action a when in state s.

A prediction task in RL is where the policy is supplied, and the goal is to measure how well it performs. That is, to predict the expected total reward from any given state assuming the function $\pi(a|s)$ is fixed.

A control task in RL is where the policy is not fixed, and the goal is to find the optimal policy. That is, to find the policy $\pi(a|s)$ that maximises the expected total reward from any given state.

4 Future work

We are going to work on the `LunarLander-v2` [5] problem on Open-AI Gym and try to maximize our average reward over multiple episodes using Deep Q-Learning and Deep Q-Network (DQN) that is the first deep reinforcement learning method proposed by DeepMind in 2015 [6].

4.1 Q-learning: Off-Policy TD Control

If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning. TD methods can learn directly from raw experience without a model of the environment's dynamics. TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap). The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of reinforcement learning.

One of the early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning [7], defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right].$$

Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations.

For any finite Markov decision process (FMDP), Q-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state [7]. Q-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy[7]. "Q" refers to the function that the algorithm computes - the expected rewards for an action taken in a given state.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}^{\text{old value}} \Big)}^{\text{temporal difference}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

We shall leave it here and discuss more in our final end semester report with our code and work on `LunarLander-v2` problem. In next section we simply introduce the problem statement for the same.

4.2 LunarLander-v2, Problem Statement

In this task the agent has to land a lander on a landing pad. *Observation space.* The state of the environment consists of 8 variables:

- Horizontal position: p_x
- Vertical position: p_y
- Horizontal velocity: v_x
- Vertical velocity: v_y
- Angle w.r.t the verical axis: θ
- Angular velocity: ω
- Left leg contact: c_l

- Right leg contact: c_r

Action space. The agent can perform 4 actions:

- All engines disabled: *nop*
- Enable left engine: *left*
- Enable maine engine: *main*
- Enable right engine: *right*

Rewards. The reward for moving from the initial point to the landing pad with final velocity of zero varies between 100 and 140 points. If the lander crashes it receives a reward of -100 points. If the lander lands correctly it receives a reward of +100 points. For each leg contact the agent receives a reward of +10 points. Firing the main engine in a timestep gives a reward of -0.3 points, while firing a side-engine gives a reward of -0.03.

Termination criterion. The simulator ends if either 1000 timesteps are passed, the lander crashes or it passes the bounds of the environment.

Resolution criterion. This task is considered as solved if the mean total reward $R \geq 200$ on 100 episodic runs.

References

[1] Hu, J.; Niu, H.; Carrasco, J.; Lennox, B.; Arvin, F. (2020). *"Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning"*.. IEEE Transactions on Vehicular Technology. 69 (12): 14413-14423.

[2] Richard S. Sutton and Andrew G. Barto. (2018) *Reinforcement Learning: An Introduction, second edition.* Adaptive Computation and Machine Learning, The MIT Press, Cambridge, Massachusetts, London, England.

[3] Martin L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley Series in Probability and Statistics, Wiley-Interscience, 2005

[4] Introduction to Reinforcement Learning with David Silver, DeepMind and University College, London.
`https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver`

[5] LunarLander-v2, OpenAI-Gym.
`https://gym.openai.com/envs/LunarLander-v2/`

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, *"Playing Atari with Deep Reinforcement Learning"*, NIPS Deep Learning Workshop 2013

[7] Christopher J.C.H. Watkins, Peter Dayan, *"Technical Note,Q-Learning"*, Machine Learning, 8, 279-292 (1992), Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.