# Note on functions and functions in Reinforcement Learning

(from Chapter 3 of *Sutton and Barto’s* book)

*written by D. Gueorguiev, Nov 26, 2023*

Table of Contents

[Note on functions and functions in Reinforcement Learning 1](#_Toc151904194)

[Notation and Definitions 1](#_Toc151904195)

[State-Value and State-Action Functions 1](#_Toc151904196)

[Estimation Methods for State-Value and State-Action Functions 3](#_Toc151904197)

[Bellman’s Equations for State-Value and State-Action Functions 3](#_Toc151904198)

[Bellman’s equation for state values 3](#_Toc151904199)

[Bellman’s equation for action values 4](#_Toc151904200)

[Expressing the current state values in terms of the next action values 5](#_Toc151904201)

[Expressing the current action values in terms of the next state values 5](#_Toc151904202)

[Optimal Policies and Optimal Value Functions 6](#_Toc151904203)

[Bellman’s Optimality Equations 6](#_Toc151904204)

[Bellman’s optimality equation for the optimal state-value function 6](#_Toc151904205)

[Bellman’s optimality equation for the optimal state-value function 7](#_Toc151904206)

[Determining the Optimal Policy from the Bellman Optimality Equations 7](#_Toc151904207)

## Notation and Definitions

**Distribution of the Dynamics of the MDP**: defined through the following 4 arguments function:

which is the probability to get from state to state with action and with reward .

**Distribution of the state-transition probabilities**: defined through the following 3 arguments function:

**Markov Decision Process** (abbrev *MDP*): a 4-tuple with

* is a set of states (finite or infinite, discrete, or continuous)
* is a set of actions (finite or infinite, discrete, or continuous)
* is the probability to get from state to state with action and with reward . This is the function describing the dynamics of the MDP.
* is the discount factor which determines to what extent the focus is on the most recent rewards. with there is no focus on the most recent rewards only.

Note: There is another equivalent definition of Markov process which uses the *state-transition probabilities distribution* represented by the three-argument function and *reward function* . With this definition the Markov Decision Process is defined as a 5-tuple , where:

* is a set of states (finite or infinite, discrete, or continuous)
* is a set of actions (finite or infinite, discrete, or continuous)
* is the probability to get from state to state with action .
* defines a *reward function*
* is the discount factor

Note 2: A more detailed definition of MDP involves specifying the initial state distribution and augments either of the MDP definitions as:

the Markov Decision Process is a 6-tuple , where:

* is a set of states (finite or infinite, discrete, or continuous)
* is a set of actions (finite or infinite, discrete, or continuous)
* is the probability to get from state to state with action .
* defines a *reward function*
* defines the initial state distribution
* is the discount factor

**Learning Policy** (or just *Policy*): function which represents mapping from states to probabilities of selecting each possible action.

If the agent is following policy at time , then is the probability that if . Note that is an ordinary function which defines a probability distribution over for each .

We would like to modify the policy with training or experience.

## State-Value and State-Action Functions

Let us assume that the current state is , and actions are selected according to a stochastic policy then we would like to derive an expression for the expectation of in terms of and .

Recall, the function defines the dynamics of the MDP and is given as:

for all (1)

Then we can write:

(2)

Here denotes the reward of going from state to state taking action is given by MDP’s function: .

**State-Value Function for Policy** (or simply *Value function;* aka *function*): the value function of a state under a policy , denoted with , is the expected return when starting in and following thereafter. For MDPs, we can define formally by

for all (3)

where denotes the expected value of a random variable given that the agent follows policy , and is any time step. Note that the value of the terminal state, if any, is always zero.

**Action-Value Function for Policy** (or simply *Value function;* aka *function*):

We define the value of taking action in state under a policy , denoted , as expected return starting from , taking the action , and thereafter following policy :

for all and (4)

Let us express in terms of and . Given a state s, the state value function , given with (3), is equal to the expected cumulative return from that state given a distribution of actions. The action value function is the expectation of the return given state , and taking action a as a starting point, and following policy thereafter. Therefore, given a state the action-value function is the weighted sum of the action-values over all relevant actions weighted by the policy weight:

(5)

Given a state and an action let us express the action-value function in terms of the state value function and the function defining the MDP dynamics . Recall, given a state and an action , the action value function is given by the mathematical expectation of the discounted future rewards i.e. return . The return is the discounted sequence of rewards after the time step and it can be written as:

(6)

It is important to recognize that

. (7)

The first term on the right-hand side of (7) can be expressed as:

. (8)

As before, denotes the reward of going from state to state taking action is given by MDP’s function: .

The expectation in the second term on the right-hand side of (8) can be expressed as:

. (9)

This is the expectation of the return starting at the next time step following the policy given the current state and the action , chosen according to .

Substituting (8) and (9) into (7) gives us:

. (10)

Thus, the action-value function given state s and action following policy is expressed as the sum of the next reward and discounted action-value weighted by probability distribution over the possible next states and next rewards from that action and state .

### Estimation Methods for State-Value and State-Action Functions

The value functions and can be estimated from experience. For example, if an agent follows policy and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state value , as the number of times that state is encountered approaches infinity. If separate averages are kept for each action taken in each state, then these averages will similarly converge to the action values, . We call estimation methods of this kind *Monte Carlo methods* because they involve averaging over many random samples of actual returns. Of course, in case there is a large number of states then it would not be feasible to manage separate averages for each state. Instead the agent would have to maintain and as parametrized functions (with fewer parameters than states) and adjust the parameters to better match the observed returns. This approach can produce accurate estimates, although much depends on the nature of the parametrized function approximator.

## Bellman’s Equations for State-Value and State-Action Functions

### Bellman’s equation for state values

The value functions satisfy recursive relationships this property of value functions will prove quite useful.

For any policy and for any state s , the following consistency condition holds between the value of s and the value of its successor states. Starting with (6) applied to the definition of :

(11)

Using (5) the last equation becomes:

for all (12)

where it is implicit that the actions, , are taken from the set , that the next states, , are taken from the set , and that the rewards, , are taken from the set . Here the reward of going from state to state taking action is given by MDP’s function: . Note that the right-hand side of (12) is interpreted as an expected value obtained as a sum over the values of the triplet , , and . For each triplet the quantity is weighed by its probability, .

Eq. (12) is known as the *Bellman equation* for . It expresses a relationship between the value of a state and the values of its successor states.

Figure 1: Backup diagram for

This relationship is expressed by the *Backup diagram* shown on Figure 1. Each open circle, which will be denoted as *reward-state node* so forth, colored in red represents a state and the reward, which is associated with this state For instance, the root node shown on Figure 1 has associated reward and state . Each solid circle, colored in blue represents a state-action pair and will be denoted as *state-action node* so forth. The specific state on the rightmost state-action node is shown as . Each directed blue edge connects state node with state-action node and represents application of the policy to the root reward-state node . Each directed red edge emanating from a state-action node ends in a possible reward-state node corresponding to specific probable pair of reward and new state . Thus, each directed red edge represents the application of the function of the MDP dynamics. The Bellman equation (12) averages over all of the possibilities weighing each possibility represented by a path from the root of the Backup diagram on Figure 1 to a leaf by its probability of occurring. It states that the value of the start state must equal the discounted value of the expected next state plus the reward expected along the way. The value function is the unique solution to its Bellman equation. Various methods exist to compute exactly, approximate, or learn the value function.

### Bellman’s equation for state-action values

Let us derive a similar recursive relation with respect to the state-action value function. That is, we will find out what is the relation between the action value and that for the possible successors to the state-action pair - . The derivation follows from the Backup diagram shown on Figure 2 below.

Figure 2: Backup diagram for

From (7) we can write:

(13)

The expectation of the reward on the right-hand side can be rewritten as:

(14)

Here using (7) again we denote with the expression for .

Thus we get the Bellman’s equation with respect :

(15)

### Expressing the current state values in terms of the next action values

It is instructive to compare Eq (5) which we derived earlier with Eq (12) and Eq (15).

Eq (5) deserves its own Backup diagram shown on Figure 3:

(5)

Figure 3: Backup diagram for relation between and

Eq (5) tells us how the value of a state depends on the values of the actions possible in that state and on how likely each action is to be taken under the current policy. The state value which corresponds to the state-value node at the root is obviously and the action values which corresponds to its children are . The probability with which each action is taken is given by the policy i.e. .

### Expressing the current action values in terms of the next state values

The value of an action, , depends on the expected next reward and the expected sum of the remaining rewards. Expressed as a Backup diagram we arrive at Figure 4 shown below.

Figure 4: Backup diagram expressing the dependence of the current action value on the expected next reward-state values.

Formally expressed this relation becomes:

From Eq (7) we have

Clearly, where

Eq. (9) states that the expectation in the second term of (7) can be written as:

Combining the last two results we obtain:

(16)

## Optimal Policies and Optimal Value Functions

We want to find a policy which maximizes the reward over long enough run that is, maximizes the return.

For finite MDPs we define an optimal policy in the following way. Value functions define a partial ordering over policies. A policy is defined to be better than or equal to a policy if its expected return is greater than or equal to that of for all states. We write:

(17)

For finite MDPs it can be shown that there is always one policy that is better or equal to all other policies. This is an *optimal policy*. We denote with any one of the optimal policies which have the same *state-value function*, denoted with , and defined as:

(18)

Optimal policies also share the same *optimal action-value function*, denoted with and defined similarly:

(19)

From Eq (16) it follows that for the state-action pair , the function gives the expected return for taking action in state and thereafter following the optimal policy. So we can rewrite (16) in terms of the optimal policy as:

(20)

## Bellman’s Optimality Equations

### Bellman’s optimality equation for the optimal state-value function

Because is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values Eq. (12). Because it is the optimal value function, however, ’s consistency condition can be written in a special form without reference to any specific policy. The result will be the Bellman equation for or the *Bellman optimality equation*. Intuitively, the Bellman optimality equation expressed the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:

by Eq. (7)

(21)

(22)

Eq. (21) and Eq (22) are two forms of the Bellman optimality equation for .

Figure 5: Backup diagram for optimal state value function

### Bellman’s optimality equation for the optimal state-value function

The Bellman optimality equation for is

(23)

Figure 6: Backup diagram for optimal state value function

The Bellman optimality equation is actually a system of equations, one for each state, so if there are states, then there would be equations with unknowns. If the dynamics of the environment is known, then in principle one can solve this system of equations for as well as the set of equations for .

### Determining the Optimal Policy from the Bellman Optimality Equations

Once we have determined it is straightforward to the determine an optimal policy . For each state , there will be one or more actions at which the maximum is obtained in the Bellman optimality equation. Any policy that assigns nonzero probability only to these actions is an optimal policy. One can devise an algorithm based on one-step-ahead search. With the found optimal value function, , then the actions which appear best after one-step-ahead search will be optimal actions . In other words, any policy that is greedy with respect to the optimal evaluation function is an optimal policy. Thus, for any optimal policy the actions can be selected based only on the short-term consequences. The found greedy policy is actually optimal in a non-local sense – this is true because the equations for already account for the reward consequences of all possible future behaviors – that is sequential choices of actions.

Having makes choosing optimal actions even easier. With we do not need to perform one-step-ahead search: for any state , we need to find any action that maximizes . The action-value function effectively caches the results of all one-step-ahead searches and it provides a **non-local optimal return** as a value that is locally and immediately available for each state-action pair! Hence, at the cost of representing a function of state-action pairs, instead of just states, the optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values , that is, without having to know anything about the environmental dynamics ! This in turn may represent significant computational advantage compared to using for the construction of optimal policy.