# Examples using Policy and Value Functions

(from *Sutton and Barto*)

Compiled by D. Gueorguiev 1/21/2024

Table of Contents

[Examples using Policy and Value Functions 1](#_Toc160433224)

[5x5 Gridworld with special states 1](#_Toc160433225)

[Finding state value and action value functions from the Bellman’s equations 2](#_Toc160433226)

[Iterative Policy Evaluation 6](#_Toc160433227)

[Policy Improvement 6](#_Toc160433228)

[4x4 Gridworld with terminal state 6](#_Toc160433229)

[Appendix 7](#_Toc160433230)

[Notation and Definitions from Sutton and Barto’s RL book 7](#_Toc160433231)

[State-Value and Action-Value Functions 8](#_Toc160433232)

[Bellman’s Equations for State-Value and Action-Value Functions 9](#_Toc160433233)

[Bibliography 13](#_Toc160433234)

## 5x5 Gridworld with special states

The cells of the grid correspond to the states of the environment. At each cell, four actions are possible: ***north***, ***south***, ***east***, and ***west*** which deterministically cause the agent to move one cell in the respective direction on the grid. Actions that would take the agent off the grid leave its position unchanged, but also result in a reward of .

Other actions result in a reward of 0, except those that move the agent out of the special states and . From state , all four actions yield a reward of and take the agent to . From state , all actions yield a reward of and take the agent to .

A black background with a black square

Description automatically generated with medium confidence

**Figure 1**: 5x5 Gridworld Example with special states

Suppose the agent selects all four actions with equal probability in all states. Figure 2 below shows the value function, , for this policy, for the discounted reward case of .



**Figure 2**: State-value function for Gridworld

Now let us denote with and we set .

Then represents the state of the position in the grid. Thus .

### Finding state value and action value functions from the Bellman’s equations

Let us assume random (equiprobable) policy so that all are equally probable with probability . The reward and the new state for every pair of action and state as a tuple is given with the following lookup table :

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| action\state |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| action\state |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| action\state |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Table 1**: a tuple containing the reward and new state when taking an action from state .

The following pseudo code will do that computation:

# define the action set of Gridworld

from enum import Enum

class Action(Enum):

North=1

South=2

East=3

West=4

# generate the lookup table shown on Table 1

lookup\_table\_rsprime = dict()

for r in range(0,5): # current row

for c in range(0,5): # current column

for a in Action:

if (r,c) == (0,1):

lookup\_table\_rsprime[(5\*r+c,a)] = (10, (4,1))

elif (r,c) == (0,3):

lookup\_table\_rsprime[(5\*r+c,a)] = (5, (2,3))

else:

if a == Action.North and r == 0 or \

a == Action.South and r == 4 or \

a == Action.East and c == 4 or \

a == Action.West and c == 0:

lookup\_table\_rsprime[(5\*r+c,a)] = (-1, (r,c))

elif a == Action.North:

lookup\_table\_rsprime[(5\*r+c,a)] = (0, (r-1,c))

elif a == Action.South:

lookup\_table\_rsprime[(5\*r+c,a)] = (0, (r+1,c))

elif a == Action.East:

lookup\_table\_rsprime[(5\*r+c,a)] = (0, (r,c+1))

elif a == Action.West:

lookup\_table\_rsprime[(5\*r+c,a)] = (0, (r,c-1))

**Code Excerpt 1**: implementation of the lookup table

Then the function of the MDP dynamics is given with

and (1)

Here the function is defined as having value if and otherwise ().

The following pseudocode will compute the function of the MDP dynamics

def delta\_fun(tau\_1, tau\_2):

if tau\_1 == tau\_2:

return 1

else:

return 0

def state\_id(state):

return 5\*state[0]+state[1]

def mdp\_dynamics(state\_prime, reward, state, action):

if not (state\_id(state), action) in lookup\_table\_rsprime:

raise ValueError(f"Unrecognized state and action pair! state: {state}, action: {action}")

return delta\_fun((reward, state\_prime), lookup\_table\_rsprime[(state\_id(state), action)])

**Code Excerpt 2**: Implementation of the function of the MDP dynamics

Let us compute given the random (equiprobable) policy specified in the previous paragraph.

We use the derived in the Appendix expression for (A.18)-(A.24):

(A.18)

(A.19)

(A.20)

For convenience we abbreviate:

, , (A.21)

Using (A.16)-(A.21) in (3) leads to :

(A.22)

(A.22) represents a linear system of equations with respect to the unknowns .

Let us denote with the column vector of the unknowns .

We denote with the matrix formed by the elements and the vector as shown below:

, (A.23)

Then (A.22) in matrix form:

(A.24)

First, we would like to compute using (A.18):

(A.18)

(2)

In pseudo-code we write:

def random\_policy(action, state):

return 1.0/len(Action)

def state\_reward\_distr(state\_j, reward, state\_i, policy):

res = 0.0

for action in Action:

res += mdp\_dynamics(state\_j, reward, state\_i, action)\*policy(action, state\_i)

return res

**Code Excerpt 3**: implementation of

In the Gridworld problem we do not have more than one reward value when we transition from one state to another; This is true for any policy

Thus (2) becomes

(3)

where denotes the reward for transitioning from to which is given with the function defined with the following algorithm:

Given and find such that equals

If such does not exist return -math.inf for . Otherwise, return

In pseudo-code this is expressed as:

def r(j, i):

for a in Action:

lookup\_res = lookup\_table\_rsprime[(i, a)]

if state\_id(lookup\_res[1]) == j:

return lookup\_res[0]

return 0 # we assign zero for the reward of a missing state transition

**Code Excerpt 4**: Implementation of the state transition reward coefficients

Notice that (3) in fact is the quantity defined in (A.19)

(A.19)

as the summation over for given and always results in a single term which is entirely determined by specifying the indices and .

(3) computes the total probability to make a transition from state to given the equiprobable (random) policy . As an illustration let us compute the probability :

As another example let us compute :

For :

Clearly, since no combination of action and reward from state leads again into state we conclude that .

In pseudo-code this is expressed as:

def state\_distr(state\_j, state\_i, policy):

return state\_reward\_distr(state\_j, r(state\_id(state\_j),state\_id(state\_i)), state\_i, policy)

**Code Excerpt 5**: implementation of the distribution for transitioning from state to

Now using (A.20) we compute which represents the expected reward for the transition from to

(A.20)

In the Gridworld problem we do not have more than one reward value when we transition from one state to another; This is true for any policy Hence:

for any (4)

Next, we compute the coefficient given with (A.21)

(A.21)

The expected reward coefficients and the total reward from state are computed with the following pseudo-code:

def expected\_reward\_for\_state\_transition(state\_j, state\_i, policy):

return state\_distr(state\_j, state\_i, policy)\*r(state\_id(state\_j), state\_id(state\_i))

def total\_expected\_reward\_from\_state(state, policy):

res = 0

for r in range(0,5): # current row

for c in range(0,5): # current column

res += expected\_reward\_for\_state\_transition((r,c), state, policy)

return res

**Code Excerpt 6**: impl for the expected reward coefficients and the total reward from state

Now we can solve the system of equations for given with (A.23) and (A.24)

, (A.23)

Then (A.22) in matrix form:

(A.24)

//TODO: finish finding the state value and action value functions for Gridworld from Bellman’s equations given the MDP dynamics probabilities

### Iterative Policy Evaluation

We have

for all (A.12)

Let us compute

.

Clearly, is a fixed point in this update rule because the Bellman equation for assures us of equality in this case.

//TODO: finish applying iterative policy evaluation in order to find state value and action value functions for the Gridworld example given equiprobable policy and the MDP dynamics probabilities

### Policy Improvement

//TODO: finish policy improvement algorithm applied to the Gridworld example

## 4x4 Gridworld with terminal state

|  |  |  |  |
| --- | --- | --- | --- |
|  | **1** | **2** | **3** |
| **4** | **5** | **6** | **7** |
| **8** | **9** | **10** | **11** |
| **12** | **13** | **14** |  |

**Figure 3**: 4x4 Gridworld Example with terminal state. The terminal state is shown in gray

The non-terminal states are . There are four actions possible in each state, which deterministically cause the corresponding state transitions, except that actions that would take the agent off the grid in fact leave the state unchanged. This is an undiscounted, episodic task.

The terminal state is shown in gray on Figure 3 above. The reward is for all transitions until terminal state is reached. Thus, the expected reward function

.

For example here are few values of the function of the MDP dynamics of this Example:

, ,

### Iterative Policy Evaluation

We have

for all (A.12)

Let us compute

.

Clearly, is a fixed point in this update rule because the Bellman equation for assures us of equality in this case.

//TODO: finish applying iterative policy evaluation in order to find state value and action value functions for the Gridworld example given equiprobable policy and the MDP dynamics probabilities

### Policy Improvement

//TODO: finish policy improvement algorithm applied to the 4x4 Gridworld example

## Jack’s Car Rental

Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited $10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for renting the day after they are returned. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of $2 per car moved. We assume that the number of cars requested and returned at each location are Poisson random variables with probability for the respective number equal to , where is the expected number. Suppose is 3 and 4 for rental requests at the first and the second locations and 3 and 2 for returns. That is, if we denote with the number of rental requests and with the number of rental returns for location we will have:

; ; (1)

Let us denote with the number of cars at location . Obviously, we require .

We will impose the additional constraint:

(2)

That is, we require no more than 20 cars in each location. Any additional cars are returned to the nationwide company location which will be modeled with the surplus variable . Thus, instead of (2), we have:

(3)

## Appendix

### Notation and Definitions from Sutton and Barto’s RL book

**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 5-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 function which describes the MDP dynamics i.e. probability to get from state to state with action and with reward .
* defines a *reward function*
* 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 . 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 distribution of the *state-transition probabilities* i.e. 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:

Markov Decision Process with specified *initial state* 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 Action-Value 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 (A.1)

Then we can write:

(A.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 (A.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** (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 (A.4)

Let us express in terms of and . Given a state s, the state value function , given with (A.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 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:

(A.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:

(A.6)

It is important to recognize that

. (A.7)

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

. (A.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 (A.8) can be expressed as:

. (A.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 (A.8) and (A.9) into (A.7) gives us:

. (A.10)

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

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

#### Bellman’s equation for state values

The value functions satisfy recursive relationships, and this property of the former will prove quite useful.

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

(A.11)

Using (5) the last equation becomes:

for all (A.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 (A.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. (A.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 A.1: Backup diagram for

This relationship is expressed by the *Backup diagram* shown on Figure A.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 (A.12) averages over all of the possibilities weighing each possibility represented by a path from the root of the Backup diagram on Figure A.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.

#### Derivation of action value system of equations for discrete finite state

Let us assume that we are dealing with *discrete finite state* – that is we have . Here we have denoted .

Then from (A.3) we have:

for all (A.13)

We also use (A.12) rewritten as:

for all (A.14)

hence

(A.15)

The left-hand side of (A2) can be rewritten as:

(A.16)

The right-hand side of (A.15) are rearranged as :

(A.17)

we denote with and the following expressions:

(A.18)

(A.19)

(A.20)

For convenience we abbreviate:

, , (A.21)

Using (A.16)-(A.21) in (3) leads to :

(A.22)

(A.22) represents a linear system of equations with respect to the unknowns .

Let us denote with the column vector of the unknowns .

We denote with the matrix formed by the elements and the vector as shown below:

, (A.23)

Then (A.22) in matrix form:

(A.24)

#### Bellman’s equation for 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 A.2 below.

Figure A.2: Backup diagram for

From (A.7) we can write:

(A.25)

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

(A.26)

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

Thus we get the Bellman’s equation with respect :

(A.27)

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

It is instructive to compare Eq (A.5) which we derived earlier with Eq (A.12) and Eq (A.27).

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

(A.5)

Figure A.3: Backup diagram for relation between and

Eq (A.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 A.4 shown below.

Figure A.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 (A.7) we have

Clearly, where

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

Combining the last two results we obtain:

(A.28)

## Bibliography

[Reinforcement Learning, Richard S. Sutton, Andrew G. Barto, second edition, 2020](https://github.com/dimitarpg13/reinforcement_learning_and_game_theory/blob/main/books/ReinforcementLearningSuttonSecondEdition2020.pdf)