The λ Return

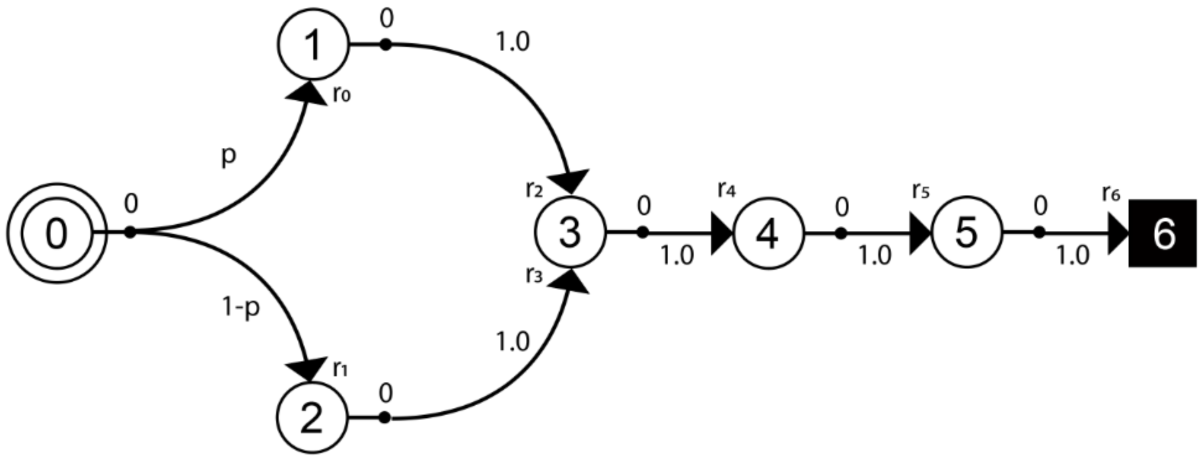
Description

Given an MDP and a particular time step t of a task (continuing or episodic), the λ-return, G_t^λ, $0 \leq \lambda \leq 1$, is a weighted combination of the n-step returns $G_{t:t+n}$, $n \geq 1$:

$$G_t^\lambda = \sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}G_{t:t+n}.$$

While the n-step return $G_{t:t+n}$ can be viewed as the target of an n-step TD update rule, the λ-return can be viewed as the target of the update rule for the TD(λ) prediction algorithm, which you will become familiar with in project 1.

Consider the Markov reward process described by the following state diagram and assume the agent is in state 0 at time t (also assume the discount rate is $\gamma = 1$). A Markov reward process can be thought of as an MDP with only one action possible from each state (denoted as action 0 in the figure below).



Procedure

- You will implement your solution using the `solve()` method in the code below.
- You will be given `p`, the probability of transitioning from state 0 to state 1, `V`, the estimate of the value function at time t, represented as a vector $[V(0), V(1), V(2), V(3), V(4), V(5), V(6)]$, and `rewards`, a vector of the rewards $[r_0, r_1, r_2, r_3, r_4, r_5, r_6]$ corresponding to the MDP.

- Your return value should be a value of λ, strictly less than 1, such that the expected value of the λ-return equals the expected Monte-Carlo return at time t.
- Your answer must be correct to 3 decimal places, truncated (e.g. 3.14159265 becomes 3.141).

Resources

The concepts explored in this homework are covered by:

- Lecture Lesson 3: TD and Friends
- Chapter 7 (7.1 n-step TD Prediction) and Chapter 12 (12.1 The λ-return) of http://incompleteideas.net/book/the-book-2nd.html (http://incompleteideas.net/book/the-book-2nd.html)
- 'Learning to Predict by the Method of Temporal Differences', R. Sutton, 1988

Submission

- The due date is indicated on the Canvas page for this assignment. Make sure you have your timezone in Canvas set to ensure the deadline is accurate.
- Submit your finished notebook on Gradescope. Your grade is based on a set of hidden test cases. You will have unlimited submissions - only the last score is kept.
- Use the template below to implement your code. We have also provided some test cases for you. If your code passes the given test cases, it will run (though possibly not pass all the tests) on Gradescope.
- Gradescope is using *python 3.6.x* and *numpy==1.18.0*, and you can use any core library (i.e., anything in the Python standard library). No other library can be used. Also, make sure the name of your notebook matches the name of the provided notebook. Gradescope times out after 10 minutes.

In [16]:
```
################
# DO NOT REMOVE
# Versions
# numpy==1.18.0
################
import numpy as np

class TDAgent(object):
    def __init__(self):
        pass

    def solve(self, p, V, rewards):
        """Implement the agent"""
        pass
        return True
```

Test cases

We have provided some test cases for you to help verify your implementation.

```
In [ ]: ## DO NOT MODIFY THIS CODE.  This code will ensure that your submissio
        ## will work probерly with the autograder

        import unittest

        class TestTDNotebook(unittest.TestCase):
            def test_case_1(self):
                agent = TDAgent()
                np.testing.assert_almost_equal(
                    agent.solve(
                        p=0.81,
                        V=[0.0, 4.0, 25.7, 0.0, 20.1, 12.2, 0.0],
                        rewards=[7.9, -5.1, 2.5, -7.2, 9.0, 0.0, 1.6]
                    ),
                    0.622,
                    decimal=3
                )

            def test_case_2(self):
                agent = TDAgent()
                np.testing.assert_almost_equal(
                    agent.solve(
                        p=0.22,
                        V=[12.3, -5.2, 0.0, 25.4, 10.6, 9.2, 0.0],
                        rewards=[-2.4, 0.8, 4.0, 2.5, 8.6, -6.4, 6.1]
                    ),
                    0.519,
                    decimal=3
                )

            def test_case_3(self):
                agent = TDAgent()

                np.testing.assert_almost_equal(
                    agent.solve(
                        p=0.64,
                        V=[-6.5, 4.9, 7.8, -2.3, 25.5, -10.2, 0.0],
                        rewards=[-2.4, 9.6, -7.8, 0.1, 3.4, -2.1, 7.9]
                    ),
                    0.207,
                    decimal=3
                )

        unittest.main(argv=[''], verbosity=2, exit=False)
```