# Reinforcement Learning: An Introduction - Notes

Scott Jeen

March 26, 2021

# 1 Introduction

## 1.1 Overview

- Supervised learning = learning with labels defined by human; Unsupervised learning = finding patterns in data. Reinforcement learning is a 3rd machine learning paradigm, in which the agent tries to maximise its reward signal.
- Exploration versus exploitation problem - agent wants to do what it has already done to maximise reward by exploitation, but there may be a bigger reward available if it were to explore.
- RL is based on the model of human learning, similar to that of the brain's reward system.

## 1.2 Elements of Reinforcement Learning

**Policy** Defines the agent's way of behaving at any given time. It is a mapping from the perceived states of the environment to actions to be taken when in those states.

**Reward Signal** The reward defines the goal of the reinforcement learning problem. At each time step, the environment sends the RL agent a single number, a *reward*. It is the agent's sole objective to maximise this reward. In a biological system, we might think of rewards as analogous to pain and pleasure. The reward sent at any time depends on the agent's current action and the agent's current state. If my state is hungry and I choose the action of eating, I receive positive reward.

**Value function** Reward functions indicate what is good in the immediately, but value functions specify what is good in the long run. The value function is the total expected reward an agent is likely to accumulate in the future, starting from a given state. E.g. a state might always yield a low immediate reward, but is normally followed by a string of states that yield high reward. Or the reverse. Rewards are, in a sense, primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no value. Nevertheless it is values with which we are most concerned when evaluating decisions. We seek actions that bring the highest value, not the highest reward, because they obtain the greatest amount of reward over the long run. Estimating values is not trivial, and efficiently and accurately estimating them is the core of RL.

**Model of environment (optionally)** Something that mimics the behaviour of the true environment, to allow inferences to be made about how the environment will behave. Given a state and action, the model might predict the resultant next state and next reward. They are used for *planning*, that is, deciding on a course of action by considering possible future situations before they are actually experienced.

# 2 Multi-arm Bandits

RL evaluates the actions taken rather than instructs correct actions like other forms of learning.

## 2.1 An n-Armed Bandit Problem

**The problem**:

- You are faced repeatedly with a choice of $n$ actions.
- After each choice, you receive a reward from a stationary probability distribution.
- Objective is to maximise total reward over some time period, say 100 time steps.
- Named after of slot machine (one-armed bandit problem), but $n$ levers instead of 1.
- Each action has an expected or mean reward based on it's prob distribution. We shall call that the *value* of the action. We do not know these values with certainty.
- Because of this uncertainty, there is always an exploration vs exploitation problem. We always have one action that we deem to be most valuable at any instant, but it is highly likely, at least initially, that there are actions we are yet to explore that are more valuable.

## 2.2 Action-Value Methods

**The estimated action value**

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{N_t(a)}}{N_t(a)} \tag{1}$$

The true value (mean reward) of an action is $q$, but the estimated value at the $t$th time-step is Q(a), given by Equation 1 (our estimate after $N$ selections of an action yielding $N$ rewards).

**The greedy action selection method:**

$$A_t = \underset{a}{\operatorname{argmax}}\, Q_t(a) \tag{2}$$

- Simplest action selection rule is to select the action with the highest estimated value.
- Argmax a means the value of $a$ for which $Q_t$ is maximised.
- $\epsilon$-greedy methods are where the agent selects the greedy option most of the time, and selects a random action with probability $\epsilon$.
- Three algorithms are tried: one with $e$=0 (pure greedy), one with $e$=0.01 and another with $e$=0.1
- Greedy method gets stuck performing sub-optimal actions.
- $e$=0.1 explores more and usually finds the optimal action earlier, but never selects it more that 91% of the time.
- $e$=0.01 method improves more slowly, but eventually performs better than the e=0.1 method on both performance measures.
- It is possible to reduce $e$ over time to try to get the best of both high and low values.

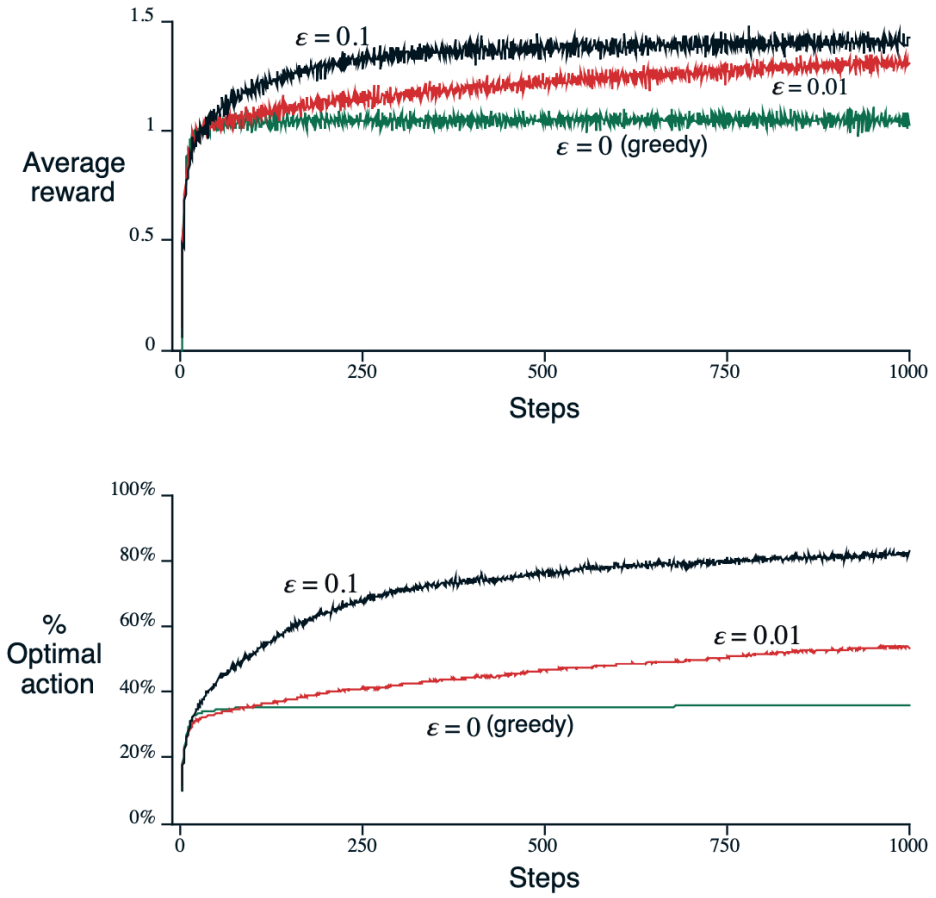



Figure 1: tbc

## 2.3 Incremental Implementation

The sample-average technique used to estimate action-values above has a problem: memory and computation requirements grow over time. This isn't necessary, we can devise an incremental solution instead:

$$\begin{aligned} Q_{k+1} &= \frac{1}{k}\sum_{i}^{k} R_i \\ &= \frac{1}{k}\left(R_k + \sum_{i=1}^{k-1} R_i\right) \\ &= \vdots \qquad (3) \\ &= Q_k + \frac{1}{k}\left[R_k - Q_k\right] \qquad (4) \\ & \qquad (5) \end{aligned}$$

We are updating our estimate of $Q_{k+1}$ by adding the discounted error between the reward

just received and our estimate for that reward $Q_k$.

$$NewEstimate \leftarrow OldEstimate + StepSize\,[Target - OldEstimate] \tag{6}$$

$\alpha$ is used to denote the stepsize ($\frac{1}{k}$) in the rest of this book.

## 2.4 Tracking a Nonstationary Problem

The averaging methods discussed above do not work if the bandit is changing over time. In such cases it makes sense to weight recent rewards higher than long-past ones. The popular way of doing this is by using a constant step-size parameter.

$$Q_{k+1} = Q_k + \alpha\,[R_k - Q_k] \tag{7}$$

where the step-size parameter $\alpha \in (0, 1]$ is constant. This results in $Q_{k+1}$ being a weighted average of the past rewards and the initial estimate $Q_1$:

$$\begin{aligned} Q_{k+1} &= Q_k + \alpha\,[R_k - Q_k] \\ &= \alpha R_k + (1-\alpha)Q_k \\ &= \alpha R_k + (1-\alpha)[\alpha R_{k-1} + (1-\alpha)Q_{k-1}] \\ &= \alpha R_k + (1-\alpha)\alpha R_{k-1} + (1-\alpha)^2 Q_{k-1} \\ &= \vdots \\ &= (1-\alpha)^k Q_1 + \sum_i^k \alpha(1-\alpha)^{k-i} R_i \end{aligned} \tag{8}$$

$$\tag{9}$$

- Because the weight given to each reward depends on how many rewards ago it was observed, we can see that more recent rewards are given more weight. Note the weights $\alpha$ sum to 1 here, ensuring it is indeed a weighted average where more weight is allocated to recent rewards.
- In fact, the weight given to each reward decays exponentially into the past. This sometimes called an *exponential* or *recency-weighted* average.

## 2.5 Optimistic Initial Values

- The methods discussed so far are dependent to some extent on the initial action-value estimate i.e. they are biased by their initial estimates.
- For methods with constant $\alpha$ this bias is permanent.
- In effect, the initial estimates become a set of parameters for the model that must be picked by the user.
- In the above problem, by setting initial values to +5 rather than 0 we encourage exploration, even in the greedy case. The agent will almost always be disappointed with it's samples because they are less than the initial estimate and so will explore elsewhere until the values converge.
- The above method of exploration is called *Optimistic Initial Values*.

## 2.6 Upper-confidence-bound Action Selection

$\epsilon$-greedy action selection forces the agent to explore new actions, but it does so indiscriminately. It would be better to select among non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainty in those estimates. One way to do this is to select actions as:

$$A_t = \underset{a}{\text{argmax}} \left[ Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{10}$$

where c $> 0$ controls the degree of exploration.

- The square root term is a measure of the uncertainity in our estimate. It is proportional to $t$ i.e. how many timesteps have passed and inversely proportional to $N_t(a)$ i.e. how many times that action has been visited. The more time has passed, and the less we have sampled an action, the higher our upper-confidence-bound.
- As the timesteps increases, the denominator dominates the numerator as the ln term flattens.
- Each time we select an action our uncertainty decreases because N is the denominator of this equation.
- UCB will often perform better than e-greedy methods

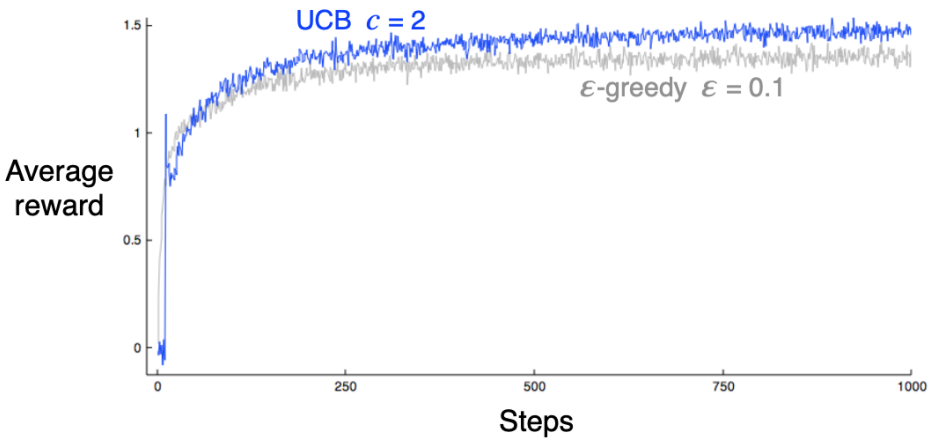



Figure 2: UCB performs better than $e$-greedy in the n-armed bandit problem

## 2.7 Associative Search (Contextual Bandits)

- Thus far we have been discussing the stationary k-armed bandit problem, where the value of each arm is unknown but nonetheless remains stationary. Now, we consider a problem where the task could change from step to step, but the value distributions of the arms in each task remain the same. This is called contextual bandits, and in the toy example we are usually given a hint that the task has changed e.g. the slot machine changes colour for each task.

- Now we want to learn the correct action to take in a particular setting, given the task colour observed. This is an intermediary between the stationary problem and the full reinforcement learning problem. See exercise 2.10 below.

## 2.8 Key Takeaways

- The value of an action can be summarised by $Q_t(a)$, the sample average return from an action
- When selecting an action, it is preferable to maintain exploration, rather than only selecting the action we believe to be most valuable at any given timestep, to ensure we continue to improve our best estimate of the optimal action. We do so using $\epsilon$)-greedy policies.
- If our problem is non-stationary, rather than taking a standard average of every return received after an action, we can take a weighted average that gives higher value to more recently acquired rewards. We call this an *exponential* or *recency-weighted* average.
- Optimistic initial values encourage lots of early exploration as our returns always decrease our estimate of $Q_t$ meaning the greedy actions remain exploratory. Only useful for stationary problems.
- $\epsilon$-greedy policies can be adapted to give more value to actions that have been selected less-often, i.e. actions where we have higher uncertainty in their value, using *upper-confidence-bound* action selection.
- Lastly, each of these techniques have varied performance on the n-armed bandit test dependent on their parametrisation. Their performance is plotted in Figure 3.

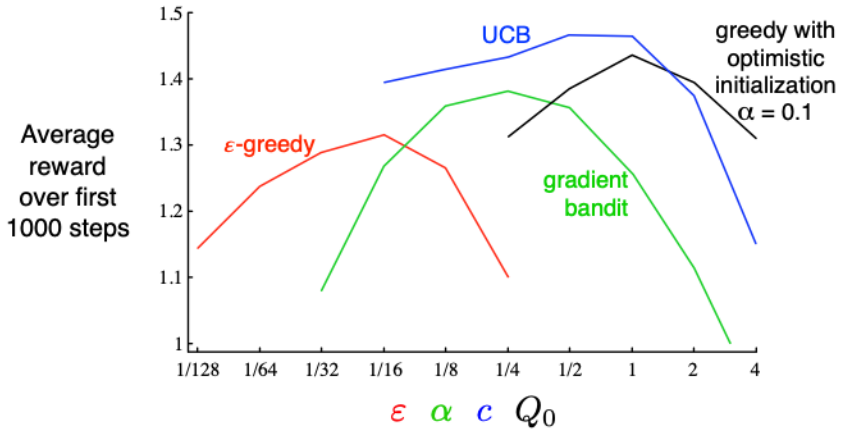



Figure 3: Performance of each of the bandit algorithms explored in this chapter

| Timestep | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | Greedy action at timestep | Action selected |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | - | $A_1$ |
| 1 | 1 | 0 | 0 | 0 | $A_1$ | $A_2$ |
| 2 | 1 | 1 | 0 | 0 | $A_1/A_2$ | $A_2$ |
| 3 | 1 | 1.5 | 0 | 0 | $A_2$ | $A_2$ |
| 4 | 1 | 1.66 | 0 | 0 | $A_2$ | $A_5$ |
| 5 | 1 | 1.66 | 0 | 0 | $A_2$ | end |

## 2.9 Exercises

### 2.9.1 Exercise 2.1

**Q**
In $e$-greedy action selection, for the case of two actions and $e = 0.5$, what is the probability that the greedy action is selected?

**A**

- Greedy action selected with $p = 0.5$

### 2.9.2 Exercise 2.2

**Q**
Bandit example Consider a k-armed bandit problem with k = 4 actions, denoted 1, 2, 3, and 4. Consider applying to this problem a bandit algorithm using $e$-greedy action selection, sample-average action-value estimates, and initial estimates of $Q_1(a) = 0$, for all $a$. Suppose the initial sequence of actions and rewards is A1 = 1, R1 = 1, A2 = 2, R2 = 1, A3 = 2, R3 = 2, A4 = 2, R4 = 2, A5 = 3, R5 = 0. On some of these time steps the $e$ case may have occurred, causing an action to be selected at random. On which time steps did this definitely occur? On which time steps could this possibly have occurred?

**A**

- Action selection at timesteps 0, 1 and 4 are random as they are not reward maximising - see Table **??**.
- Action selection at timestep 2 could be random as $A_1$ is also reward maximising - see Table **??**.

### 2.9.3 Exercise 2.3

**Q**
In the comparison shown in Figure 2, which method will perform best in the long run in terms of cumulative reward and probability of selecting the best action? How much better will it be? Express your answer quantitatively.

**A**

- In the limit of $t \to \infty$, both non-zero $e$-greedy policies will learn the optimal action abd value function $q*$. The policy with $e = 0.01$ will select the optimal action 10x more regularly than the policy with $e = 0.1$.

### 2.9.4 Exercise 2.4

**Q**
If the step-size parameters, $\alpha_n$, are not constant, then the estimate $Q_n$ is a weighted average of previously received rewards with a weighting different from that given by (2.6). What is the weighting on each prior reward for the general case, analogous to (2.6), in terms of the sequence of step-size parameters?.

**A** For a n-dependent $\alpha$ we have:

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha_n \left[R_n - Q_n\right] \\
&= \alpha_n R_n + (1-\alpha_n) Q_n \\
&= \alpha_n R_n + (1-\alpha_n) \left[Q_{n-1} + \alpha_{n-1} \left[R_{n-1} - Q_{n-1}\right]\right] \\
&= \alpha_n R_n + \alpha_{n-1} R_{n-1} - \alpha_n \alpha_{n-1} R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1}) Q_{n-1} \\
&= \vdots \\
&= \sum_i^n \alpha_n R_n - R_{n-1} \prod_i^n \alpha_n + Q_1 \prod_i^n (1-\alpha_n) \qquad (11)
\end{aligned}$$

$$(12)$$

### 2.9.5 Exercise 2.5

**Q**
Design and conduct an experiment to demonstrate the difficulties that sample-average methods have for nonstationary problems. Use a modified version of the 10-armed testbed in which all the $q_*(a)$ start out equal and then take independent random walks (say by adding a normally distributed increment with mean 0 and standard deviation 0.01 to all the $q_*(a)$ on each step). Prepare plots like Figure 2.2 for an action-value method using sample averages, incrementally computed, and another action-value method using a constant step-size parameter, $\alpha = 0.1$. Use $\alpha = 0.1$ and longer runs, say of 10,000 steps.

**A**
This is a programming exercise, the relevent code can be found on my GitHub.

### 2.9.6 Exercise 2.6

**Q**
*Mysterious Spikes*: The results shown in Figure 2.3 should be quite reliable because they are averages over 2000 individual, randomly chosen 10-armed bandit tasks. Why, then, are there oscillations and spikes in the early part of the curve for the optimistic method? In other words, what might make this method perform particularly better or worse, on average, on particular early steps?

**A**
The optimistic greedy policy with explore on every initial step as all value estimates are greater than their true value. It is possible, therefore, that it randomly selects the optimal action and then immediately forgets it in favour of yet-to-be-explored actions. This explains the spike at timestep $\approx 10$.

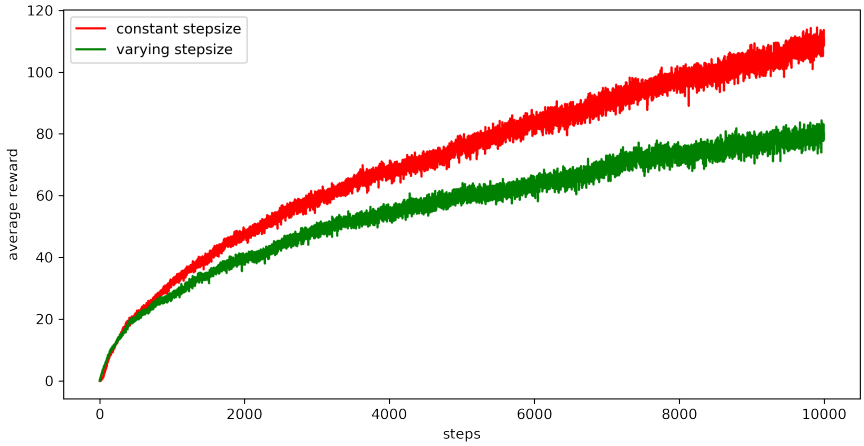



Figure 4: Stationary versus varying stepsize *alpha* in a 10-armed bandit problem. We see that the constant stepsize parameter (exponential decay) performs better than the varying stepsize parameter as we place more weight on the recently observed (moving) values.

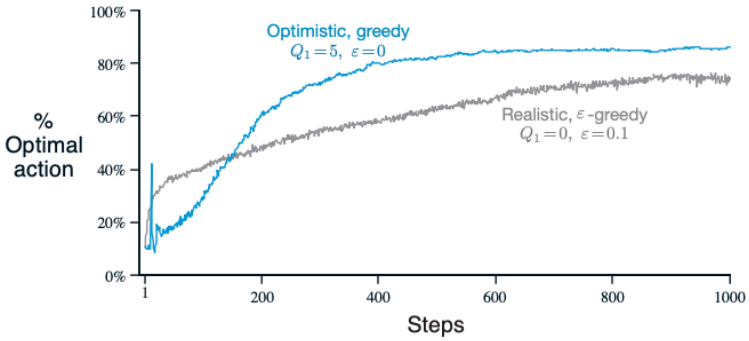



**Figure 2.3:** The effect of optimistic initial action-value estimates on the 10-armed testbed. Both methods used a constant step-size parameter, $\alpha = 0.1$.

Figure 5:

### 2.9.7 Exercise 2.7

**Q**

Unbiased Constant-Step-Size Trick In most of this chapter we have used sample averages to estimate action values because sample averages do not produce the initial bias that constant step sizes do (see the analysis leading to (2.6)). However, sample averages are not a completely satisfactory solution because they may perform poorly on nonstationary problems. Is it possible to avoid the bias of constant step sizes while retaining their advantages on nonstationary problems? One way is to use a step size of

$$\beta_n = \alpha / \bar{o_n} \tag{13}$$

to process the $n$th reward for a particular action, where $\alpha$ ¿ 0 is a conventional constant step size, and ¯on is a trace of one that starts at 0:

$$\bar{o_n} = \bar{o_{n-1}} + \alpha(1 - \bar{o_{n-1}}), for n > 0 and \bar{o_0} = 0. \tag{14}$$

Carry out an analysis like that in (2.6) to show that Qn is an exponential recency-weighted average without initial bias.

**A**
If we recall our answer for Exercise 2.4 for varying stepsize, we see that $Q_1$ is weighted by $w = \prod_{i=1}^{\infty}(1 - \alpha_i)$. When $i = 1$, $\beta_n = \alpha$, thus $w \to 0 \forall i$ and $Q_1$ no longer affects our estimate of $Q_{n+1}$.

### 2.9.8 Exercise 2.8

**Q**
*UCB Spikes* In Figure 2 the UCB algorithm shows a distinct spike in performance on the 11th step. Why is this? Note that for your answer to be fully satisfactory it must explain both why the reward increases on the 11th step and why it decreases on the subsequent steps. Hint: If c = 1, then the spike is less prominent.

**A**
After 10 timesteps the UCB algorithm has explored all 10 actions as, until they are selected, their upper confidence bound is infinite (as $N_t(a) = 0$) as so it guarenteed to be selected once in the first 10 actions. At this point the agent has one sample to assess the expected value of each arm and the same confidence/uncertainty in each action. With c ¡ 0 it is likely to pick the action with highest return from first sample, which will likely give it an similarly large reward, creating the spike. Now, the upper confidence bound for that action will decrease and the agent will select another, less valuable action, causing the decrease in performance at the next timestep.

### 2.9.9 Exercise 2.9

**Q**
Show that in the case of two actions, the soft-max distribution is the same as that given by the logistic, or sigmoid, function often used in statistics and artificial neural networks.

**A**
Soft-max distribution is defined as:

$$Pr A_t = a \approx \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \approx \pi_t(a) \tag{15}$$

For two actions 1 and 2 this becomes:

### 2.9.10 Exercise 2.10

**Q**
Suppose you face a 2-armed bandit task whose true action values change randomly from time step to time step. Specifically, suppose that, for any time step, the true values of actions 1 and 2 are respectively 10 and 20 with probability 0.5 (case A), and 90 and 80 with probability 0.5 (case B). If you are not able to tell which case you face at any step, what is the best expected reward you can achieve and how should you behave to achieve it? Now suppose that on each step you are told whether you are facing case A or case

B (although you still don't know the true action values). This is an associative search task. What is the best expected reward you can achieve in this task, and how should you behave to achieve it?

**A**

Part 1): If we do not know whether we are in task A or B we could decide pick the same action each time to maximise expected reward. Selecting either action 1 or action 2 every time would provide an expected reward of 50. Picking actions randomly would also provide an expected reward of 50 in this example. Part 2): If we know we are in task A or B we can learn the optimal action for each (A(a) = 2 and B(a) = 1). Doing so would provide us a higher expected reward than the non-contextual case of 55.

# 3 Finite Markov Decision Processes

## 3.1 The Agent-Environment Interface

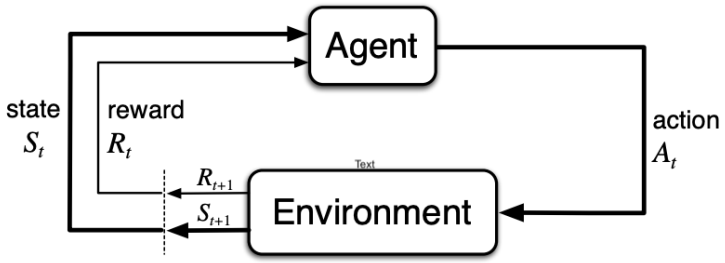



Figure 6: The agent-environment interface in reinforcement learning

- At each timestep the agent implements a mapping from states to probabilities of selecting a possible action. The mapping is called the agents *policy*, denoted $\pi$, where $\pi(a|s)$ is the probability of the agent selecting actions a in states.
- In general, actions can be any decision we want to learn how to make, and states can be any interpretation of the world that might inform those actions.
- The boundary between agent and environment is much closer to the agent than is first intuitive. E.g. if we are controlling a robot, the voltages or stresses in its structure are part of the environment, not the agent. Indeed reward signals are part of the environment, despite very possibly being produced by the agent e.g. dopamine.

## 3.2 Goals and Rewards

The *reward hypothesis*:

> All we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

The reward signal is our way of communicating to the agent what we want to achieve not how we want to achieve it.

## 3.3 Returns and Episodes

The return $G_t$ is the sum of future rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_t \tag{16}$$

- This approach makes sense in applications that finish, or are periodic. That is, the agent-environment interaction breaks into *episodes*.
- We call these systems *episodic tasks*. e.g playing a board game, trips through a maze etc.
- Notation for state space in an episodic task varies from the conventional case ($s \in \mathcal{S}$) to ($s \in \mathcal{S}^+$)

- The opposite, continuous applications are called *continuing tasks*.
- For these tasks we use *discounted returns* to avoid a sum of returns going to infinity.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{17}$$

If the reward is a constant + 1 at each timestep, cumulative discounted reward $G_t$ becomes:
$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$
*Discounting* is a crucial topic in RL. It allows us to store a finite value of any state (summarised by its expected cumulative reward) for continuous tasks, where the non-discounted value would run to infinity.

## 3.4 Unified Notation for Episodic and Continuing Tasks

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \tag{18}$$

## 3.5 The Markov Property

A state signal that succeeds in retaining all relevant information about the past is *Markov*. Examples include:

- A cannonball with known position, velocity and acceleration
- All positions of chess pieces on a chess board.

In normal causal processes, we would think that our expectation of the state and reward at the next timestep is a function of all previous states, rewards and actions, as follows:

$$Pr\{R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \tag{19}$$

If the state is Markov, however, then the state and reward right now completely characterizes the history, and the above can be reduced to:

$$p(s', r|s, a) = Pr\{R_{r+1} = r, S_{t+1} = s' | S_t, A_t\} \tag{20}$$

- Even for non-Markov states, it is appropriate to think of all states as at least an approximation of a Markov state.
- Markov property is important in RL because decisions and values are assumed to be a function only of the current state.
- Most real scenarios are unlikely to be Markov. In the example of controlling HVAC, the HVAC motor might heat up which affects cooling power and we may not be tracking that temperature. It is hard for a process to be Markov without sensing all possible variables.

## 3.6 Markov Decision Process (MDP)

Given any state and action s and a, the probability of each possible pair of next state and reward, s', r is denoted:

$$p(s', r|s, a) = Pr\{R_{r+1} = r, S_{t+1} = s'|S_t, A_t\} \tag{21}$$

We can think of $p(s', r|s, a)$ as the dynamics of our MDP, often called the *transition function*–it defines how we move from state to state given actions.

## 3.7 Policies and Value Functions

- Value functions are functions of states or functions of state-value pairs.
- They estimate how good it is to be in a given state, or how good it is to perform a given action in a given state.
- Given future rewards are dependent on future actions, value functions are defined with respect to particular policies as the value of a state depends on the action an agent takes in said state.
- A *policy* is a mapping from states to probabilities of selecting each possible action.
- RL methods specify how the agent's policy changes as a result of its experience.
- For MDPs, we can define nu-pi(s) formally as:

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t|S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{22}$$

i.e. the expected future rewards, given state $S_t$, and policy $\pi$. We call $v_\pi(s)$the **state value function for policy** *pi*. Similarly, we can define the value of taking action $a$ in state $s$ under policy $\pi$ as:

$$q_\pi(s, a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{23}$$

i.e. the expected value, taking action $a$ in state $s$ then following policy $\pi$.

- We call $q_\pi$ the **action-value function for policy** $\pi$
- Both value functions are estimated from experience.

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy recursive relationships similar to that which we have already established for the return. This recursive relationship is characterised by the **Bellman Equation**:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right] \tag{24}$$

This recursion looks from one state through to all possible next states given our policy and the dynamics as suggested by 7:

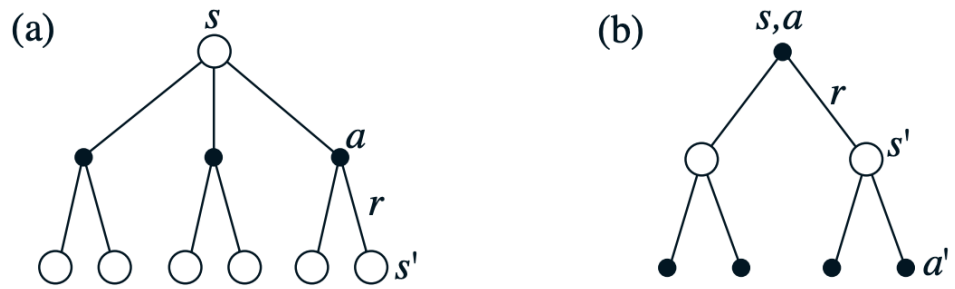



Figure 7: Backup diagrams for $v_\pi$ and $q_\pi$

## 3.8 Optimal Policies and Value Functions

- A policy $\pi'$ is defined as better than policy pi if its expected return is higher for all states.
- There is always AT LEAST one policy that is better than or equal to all other policies - this is the *optimal policy.*
- Optimal policies are denoted $\pi*$
- Optimal state-value functions are denoted $v*$
- Optimal action-value functions are denoted $q*$
- We can write $q*$ in terms of $v*$:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] \tag{25}$$

We can adapt the Bellman equation to achieve the Bellman optimality equation, which takes two forms. Firstly for $v_*$:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_*(s')\right] \tag{26}$$

and secondly for $q_*$:

$$q_*(s) = \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \tag{27}$$

- Using $v*$ the optimal expected long term return is turned into a quantity that is immediately available for each state. Hence a one-step-ahead search, acting greedily, yield the optimal long-term actions.
- Fully solving the Bellman optimality equations can be hugely expensive, especially if the number of states is huge, as is the case with most interesting problems.

## 3.9 Optimality and Approximation

- We must approximate because calculation of optimality is too computationally intense.
- A nice way of doing this, is allowing the agent to make sub-optimal decisions in scenarios it has low probability of encountering. This is a trade off for being optimal in situations that occur frequently.

## 3.10 Key Takeaways

## 3.11 Exercises

### 3.11.1 Exercise 3.1

**Q**
Devise three example tasks of your own that fit into the MDP framework, identifying for each its states, actions, and rewards. Make the three examples as different from each other as possible. The framework is abstract and flexible and can be applied in many different ways. Stretch its limits in some way in at least one of your examples.

**A**

1. *Golf: Putting*

    - State: Coordinates of ball; coordinates of hole; x,y,z contour plot of green; grass length; grass type; wind speed; wind direction; ball type; putter type.
    - Actions: (X, Y) direction of aim; length of stroke
    - Rewards: -1 for each unit of distance from hole

2. *Optimizing investment portfolio*

    - State: Investment in each company; cash balance; % return over last minute/hour/day/week/month/year
    - Actions: For each investment: buy (discretized by some cash interval), sell (discretized by some cash interval), stick
    - Rewards: +1 for each £ return per day

3. *Shoe-tying robot*

    - State: Coordinates of laces; coordinates of robot arms/joints
    - Actions: Grip pressure on laces; adjust position of arms to coordinates x,y,z
    - Rewards: -1 for unit of shoe displacement from foot once tied.

### 3.11.2 Exercise 3.2

**Q**
Is the MDP framework adequate to usefully represent all goal-directed learning tasks? Can you think of any clear exceptions?

**A**
The MDP framework fails when the state cannot be fully observed. For example, one may try to control the temperature of a house using a state space represented by one thermometer in one room. Our agent would control the temperature observed by that thermometer well, but would be blind to temperatures that aren't measured by said thermometer elsewhere in the house.

| $s$ | $a$ | $s'$ | $r$ | p$(s, r, \|s, a)$ |
|---|---|---|---|---|
| *high* | *search* | *high* | $r_{search}$ | $\alpha$ |
| *high* | *search* | *low* | $r_{search}$ | $(1-\alpha)$ |
| *low* | *search* | *high* | -3 | $(1-\beta)$ |
| *low* | *search* | *low* | $r_{search}$ | $\beta$ |
| *high* | *wait* | *high* | $r_{wait}$ | 1 |
| *low* | *wait* | *low* | $r_{wait}$ | 1 |
| *low* | *recharge* | *high* | 0 | 1 |

### 3.11.3 Exercise 3.3

**Q**
Consider the problem of driving. You could define the actions in terms of the accelerator, steering wheel, and brake, that is, where your body meets the machine. Or you could define them farther out—say, where the rubber meets the road, considering your actions to be tire torques. Or you could define them farther in—say, where your brain meets your body, the actions being muscle twitches to control your limbs. Or you could go to a really high level and say that your actions are your choices of where to drive. What is the right level, the right place to draw the line between agent and environment? On what basis is one location of the line to be preferred over another? Is there any fundamental reason for preferring one location over another, or is it a free choice?

**A**
There's a trade-off between state-action space complexity, computational expense and accuracy. If we draw the boundary at the brain we would create a state-action space contingent on the number of neurons in the brain and their interplay with physical actions like turning the steering wheel; too large to be stored or computed efficiently. Equally, if we draw the boundary at the *journey* level, then we miss the detail required to act on second-by-second state changes on the road that could lead to a crash. The fundamental limiting factor in this selection is whether the goal can be achieved safely at your chosen layer of abstraction, indeed this feels like one of the tasks of engineering more widely.

### 3.11.4 Exercise 3.4

**Q**
Give a table analogous to that in Example 3.3, but for $p(s0, r|s, a)$. It should have columns for $s, a, s', r$, and $p(s0, r|s, a)$, and a row for every 4-tuple for which $p(s0, r|s, a) > 0$.

**A**

As there is no probability distribution over the rewards (i.e. for each state-action pair, the agent receives some reward with $p(r) = 1)$), $p(s', r|s, a) = p(s'|s, a)$.

### 3.11.5 Exercise 3.5

**Q**
The equations in Section 3.1 are for the continuing case and need to be modified (very slightly) to apply to episodic tasks. Show that you know the modifications needed by giving the modified version of (3.3).

**A**
Equation 3.3 from section 3.1 is:

$$\sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) = 1, \forall s \in S, a \in A(s) \tag{28}$$

for episodic tasks this becomes:

$$\sum_{s' \in S^+} \sum_{r \in R} p(s', r|s, a) = 1, \forall s \in S, a \in A(s) \tag{29}$$

### 3.11.6 Exercise 3.6

**Q**
Suppose you treated pole-balancing as an episodic task but also used discounting, with all rewards zero except for 1 upon failure. What then would the return be at each time? How does this return differ from that in the discounted, continuing formulation of this task?

**A**
In the discounted, episodic version of the pole-balancing task, the return after each episode is:

$$G_t = -\gamma^{T-t} \tag{30}$$

where $T$ is the timestep at which the episode ends i.e. the task is failed. In the continuining case the cumulative return is:

$$G_t = -\sum_{K \in \mathcal{K}} \gamma^{K-1} \tag{31}$$

where $\mathcal{K}$ is the set of times where the task is failed. Note this reward will increase in the long run, irrespective of improved performance. Designing this as a continuous task does not therefore make sense here.

### 3.11.7 Exercise 3.7

**Q**
Imagine that you are designing a robot to run a maze. You decide to give it a reward of +1 for escaping from the maze and a reward of zero at all other times. The task seems to break down naturally into episodes—the successive runs through the maze—so you decide to treat it as an episodic task, where the goal is to maximize expected total reward (3.7). After running the learning agent for a while, you find that it is showing no improvement in escaping from the maze. What is going wrong? Have you effectively communicated to the agent what you want it to achieve?

**A**
We have designed the reward such that it receives +1 only once it has exited the maze, and are therefore not incentivising it to learn how to exit the maze faster. To do so, we would need to provide a negative reward proportional to time in the maze e.g. -1 per timestep.

### 3.11.8 Exercise 3.8

**Q**
Suppose $\gamma = 0.5$ and the following sequence of rewards is received R1 = -1, R2 = 2, R3 = 6, R4 = 3, and R5 = 2, with T = 5. What are $G_0, G_1, \ldots, G_5$? Hint: Work backwards.

**A**
We know: $G_t = R_{t+1} + \gamma G_{t+1}$ Therefore, for the terminal state: $G_5 = 0$ then: $G_4 = 2 + (0.5 \times 0) = 2$
$G_3 = 3 + (0.5 \times 2) = 4$
$G_2 = 6 + (0.5 \times 4) = 8$
$G_1 = 2 + (0.5 \times 8) = 6$
$G_0 = -1 + (0.5 \times 6) = 2$
Note our expected cumulative reward $G_t$ depends greatly on our instant reward $G_{t+1}$ because it is not discounted.

### 3.11.9 Exercise 3.9

**Q**
Suppose $\gamma = 0.9$ and the reward sequence is $R_1 = 2$ followed by an infinite sequence of 7s. What are $G_1$ and $G_0$?

**A**
We know that if the reward is an infinite series of 1s, $G_t$ is: $G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$
So for an infinite series of 7s this becomes: $G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{7}{0.1}$
Therefore:

$$G_0 = 2 + \frac{7}{0.1} \tag{32}$$
$$= 72 \tag{33}$$

and

$$G_1 = 7 + \frac{7}{0.1} \tag{34}$$
$$= 77 \tag{35}$$

### 3.11.10 Exercise 3.10

**Q**
Prove the second equality in (3.10).

**A**
Skipped.

### 3.11.11 Exercise 3.11

**Q**
If the current state is $S_t$, and actions are selected according to a stochastic policy $\pi$, then what is the expectation of $R_{t+1}$ in terms of $\pi$ and the four-argument function $p$ (3.2)

**A**
$\mathbb{E}_\pi[R_{t+1}|s_t] = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r]$

### 3.11.12 Exercise 3.12

**Q**
Give an equation for $v_\pi$ in terms of $q_\pi$ and $\pi$.

**A**
The state value function $v_\pi$ is equal to the expected cumulative return from that state given a distribution of actions. The state-action value function $q_\pi$ is the value of being in a state and taking a deterministic action. Therefore the state value function is the weighted sum of the state action value function, with the weights equal to the probabilities of selecting each action: $v_\pi = \sum_a \pi(a|s) q_\pi(s, a)$

### 3.11.13 Exercise 3.13

**Q**
Give an equation for $q_\pi$ in terms of $v_\pi$ and the four-argument $p$.

**A**
Given an action $a$, the state-action value function is the probability distributions over the possible next states and rewards from that action, times the one-step reward and the discounted state value function at the next timestep: $q_\pi = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)[r + \gamma v_\pi(s_{t+1})]$

### 3.11.14 Exercise 3.14


**Figure 3.2:** Gridworld example: exceptional reward dynamics (left) and state-value function for the equiprobable random policy (right).

Figure 8: Gridworld example for ex. 3.14

**Q**
The Bellman equation (3.14) must hold for each state for the value function $v_\pi$ shown in Figure 3.2 (right) of Example 3.5. Show numerically that this equation holds for the center state, valued at +0.7, with respect to its four neighbouring states, valued at +2.3, +0.4, 0.4, and +0.7. (These numbers are accurate only to one decimal place.) .

**A**
Bellman equation for $v_\pi$ is:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{36}$$

for the centre square in Figure **??** we get the following:

$$v_\pi(s) = 0.25\,[0.9 \times 2.3] + 0.25\,[0.9 \times 0.7] + 0.25\,[0.9 \times 0.4] + 0.25\,[0.9 \times -0.4] \tag{37}$$
$$= 0.68 \approx 0.7 \tag{38}$$

### 3.11.15 Exercise 3.15

**Q**
In the gridworld example, rewards are positive for goals, negative for running into the edge of the world, and zero the rest of the time. Are the signs of these rewards important, or only the intervals between them? Prove, using (3.8), that adding a constant $c$ to all the rewards adds a constant, $v_c$, to the values of all states, and thus does not affect the relative values of any states under any policies. What is $v_c$ in terms of $c$ and $\gamma$?

**A**
Part 1): The sign of the reward is of no consequence, it is indeed the interval between each reward that drives behaviour.
Part 2): Equation 3.8 is as follows:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{39}$$

when we add a constant $c$ to all rewards this becomes: $G_t = \sum_{k=0}^{\infty} \gamma^k [R_{t+k+1} + c] G_t = \frac{R_{t+k+1}}{1-\gamma} + \frac{c}{1-\gamma}$
Expected cumulative reward from every state receives a constant additive term. $v_c$ in terms of $c$ and $\gamma$ is:

$$v_c = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k c\right] \tag{40}$$
$$= \frac{c}{1-\gamma} \tag{41}$$

### 3.11.16 Exercise 3.16

**Q**
Now consider adding a constant $c$ to all the rewards in an episodic task, such as maze running. Would this have any effect, or would it leave the task unchanged as in the continuing task above? Why or why not? Give an example.

**A**
In the episodic task adding a constant to all rewards does affect the agent. Since the cumulative reward now depends on the length of the episode ($G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$), timesteps that incur positive rewards act to lengthen the episode and vice versa. In the maze running example, we may have chosen to give the agent -1 reward at each timestep to ensure it completes the task quickly. If we add $c = 2$ to every reward such that the reward at each timestep is now positive, the agent is now incentivised to not find the exit, and continue collecting intermediate rewards indefinitely.

### 3.11.17 Exercise 3.17

**Q**
What is the Bellman equation for action values, that is, for $q_\pi$? It must give the action value $q_\pi(s, a)$ in terms of the action values, $q_\pi(s0, a0)$, of possible successors to the

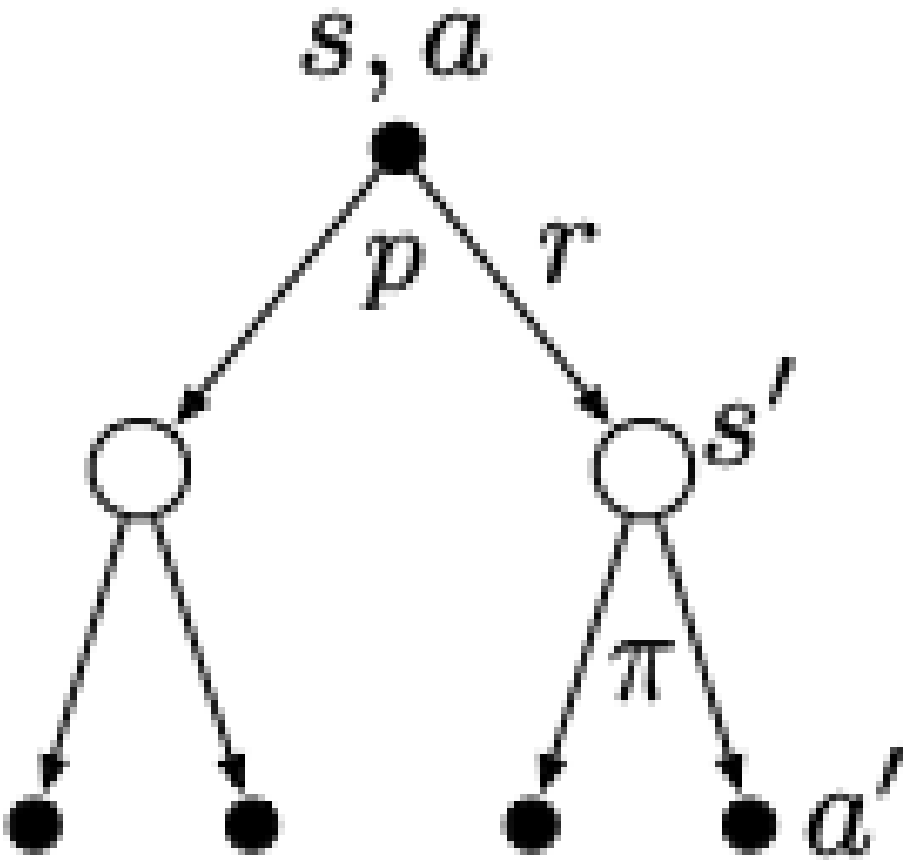



Figure 9: Backup diagram for $q_\pi$

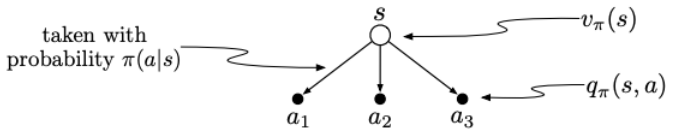
state–action pair $(s, a)$. Hint: The backup diagram to the right corresponds to this equation. Show the sequence of equations analogous to (3.14), but for action values.
**A**

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] && (42)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] && (43)\\
&= \sum_{s',r} p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|s', a']] && (44)\\
&= \sum_{s',r} p(s', r|s, a)[r + \gamma q_\pi(s', a')] && (45)\\
& && (46)
\end{aligned}
$$

### 3.11.18 Exercise 3.18

**Q**

*Exercise 3.18* 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. We can think of this in terms of a small backup diagram rooted at the state and considering each possible action:




Give the equation corresponding to this intuition and diagram for the value at the root node, $v_\pi(s)$, in terms of the value at the expected leaf node, $q_\pi(s,a)$, given $S_t = s$. This equation should include an expectation conditioned on following the policy, $\pi$. Then give a second equation in which the expected value is written out explicitly in terms of $\pi(a|s)$ such that no expected value notation appears in the equation. □

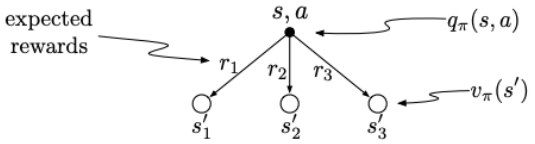
**A**

$$v_\pi(s) = \mathbb{E}_\pi \left[ q_\pi(s,a) \right] v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) \tag{47}$$

### 3.11.19 Exercise 3.19

**Q**

*Exercise 3.19* The value of an action, $q_\pi(s,a)$, depends on the expected next reward and the expected sum of the remaining rewards. Again we can think of this in terms of a small backup diagram, this one rooted at an action (state–action pair) and branching to the possible next states:




Give the equation corresponding to this intuition and diagram for the action value, $q_\pi(s,a)$, in terms of the expected next reward, $R_{t+1}$, and the expected next state value, $v_\pi(S_{t+1})$, given that $S_t = s$ and $A_t = a$. This equation should include an expectation but *not* one conditioned on following the policy. Then give a second equation, writing out the expected value explicitly in terms of $p(s', r \,|\, s, a)$ defined by (3.2), such that no expected value notation appears in the equation. □

**A**

$$q_\pi(s,a) = \mathbb{E}\left[ R_{t+1} + v_\pi(s') | S_t = s \right] \tag{48}$$

$$= \sum_{s',r} p(s', r | s, a)[r + v_\pi(s')] \tag{49}$$

$$\tag{50}$$

### 3.11.20 Exercise 3.20

**Q**
**Q**

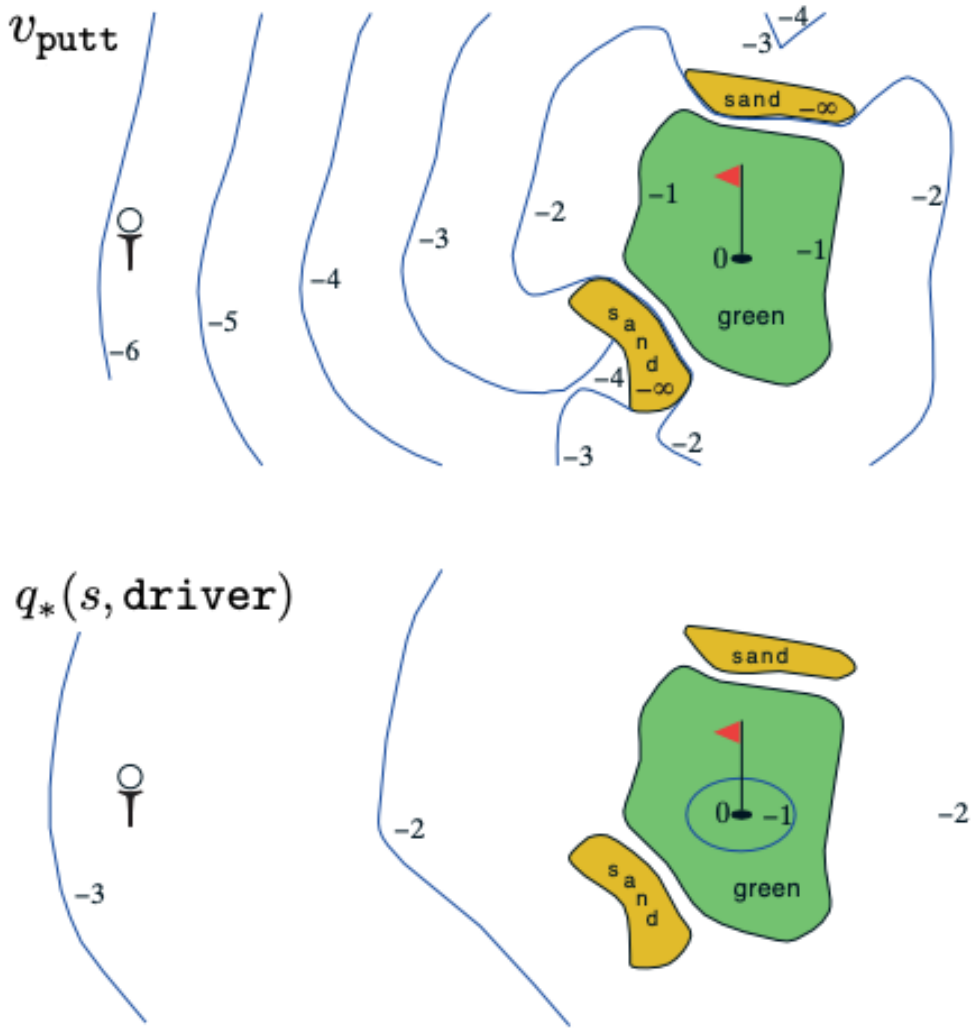



Figure 10: Value functions for golf.

Draw or describe the optimal state-value function for the golf example.

**A**
Qualitatively, the optimal state-value function for golf (outlined by Figure **??**) would be derived from a policy that selected the driver for the first two shots (off-the green), then selected the putter for the final shot (on the green).

### 3.11.21 Exercise 3.21

**Q**
Draw or describe the contours of the optimal action-value function for putting, $q_*(s, putter)$, for the golf example.

**A**
The optimal action-value function is given by the following:

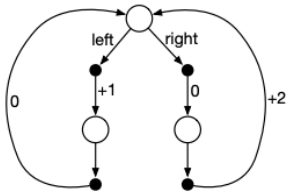
$$q_*(s,a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \operatorname*{argmax}_a q_*(s, a)] \tag{51}$$

Since we have only one action(*putter*), the argmax collapses to $v_*(s')$, and $q_*(s, a) = v_*(s)$ - illustrated in Figure **??**.

### 3.11.22 Exercise 3.22

**Q**

*Exercise 3.22* Consider the continuing MDP shown to the right. The only decision to be made is that in the top state, where two actions are available, left and right. The numbers show the rewards that are received deterministically after each action. There are exactly two deterministic policies, $\pi_{\text{left}}$ and $\pi_{\text{right}}$. What policy is optimal if $\gamma = 0$? If $\gamma = 0.9$? If $\gamma = 0.5$? □




**A**
Let's first evaluate the simplest case of $\gamma = 0$. Recall that:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \tag{52}$$

For $\pi_{left}$ we get:

$$v_{\pi left} = 1 + \mathbb{E}_{\pi left}\left[0 \times \gamma G_{t+1}\right] \tag{53}$$
$$v_{\pi left} = 1 \tag{54}$$
$$\tag{55}$$

And for $\pi_{right}$ we get:

$$v_{\pi left} = 0 + \mathbb{E}_{\pi left}\left[0 \times \gamma G_{t+1}\right] \tag{56}$$
$$v_{\pi left} = 0 \tag{57}$$
$$\tag{58}$$

So the optimal policy for $\gamma = 0$ is $v_{\pi left}$. If instead $\gamma = 0.9$, we get for $\pi_{left}$:

$$v_{\pi left} = 1 + \mathbb{E}_{\pi left}\left[0.9 \times G_{t+1}\right] \tag{59}$$
$$v_{\pi left} = 1 + \mathbb{E}_{\pi left}\left[0.9 \times [r_{t+1} + \gamma G_{t+2}]\right], \tag{60}$$
$$\tag{61}$$

where $G_{t+2}$ is our expected reward back at our current state. We can negate it and just look at the first loop, as max reward over first loop will create max reward in the limit. Therefore:

$$v_{\pi left} = 1 + 0.9 \times 0 = 1, v_{\pi right} \qquad = 0 + 0.9 \times 2 = 1.8 \tag{62}$$

$$\tag{63}$$

So the optimal policy for $\gamma = 0$ is $v_{\pi right}$. If, finally, $\gamma = 0.5$, we get:

$$v_{\pi left} = 1 + 0.5 \times 0 = 1, v_{\pi right} \qquad = 0 + 0.5 \times 2 = 1 \tag{64}$$

$$\tag{65}$$

So both polices are optimal.

#### 3.11.23 Exercise 3.23

**Q**
Give the Bellman equation for $q_*$ for the recycling robot.

**A**
$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r+\gamma \operatorname{argmax}_{a \in \mathcal{A}} q_*(s',a')]$ where $\mathcal{A} = \{search, wait, recharge\}$

#### 3.11.24 Exercise 3.24

**Q**
Figure 3.5 gives the optimal value of the best state of the gridworld as 24.4, to one decimal place. Use your knowledge of the optimal policy and (3.8) to express this value symbolically, and then to compute it to three decimal places.

**A**
Equation 3.8 was:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{66}$$