Exercise 3.1

Devise three example tasks of your own that fit into the reinforcement learning 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.

Combat Game

State - The (x, y) position of the agent. (x, y) position of identical opponent agent. - The health of both the agent and opponent.

Actions - At each time step agent may choose to move up, down, left, or right. - It may choose to light attack up down left or right. This attack deals a constant amount of damage but will allow the agent to perform an action again soon. - It may also choose to heavy attack up, down, left, or right. Its attack will be a large swing that covers a block of six tiles in the direction it swings. This attack leaves the agent unable to select an action for a significant amount of time and its damage is drawn from a normal distribution with a mean equal to the light attack's damage.

Rewards - For bringing the other agent's health down to 0, the agent will receive a reward of 1. - For dying, it shall receieve a reward of -1.

Maze Designer

State - A 30 x 30 grid of 1s and 0s. 1s denoting walls, 0s for spaces. The state initially starts off full of 1s. There is an exit block denoted with 2 in the top left corner.

Actions - The Maze Designer will be brought to each tile in the grid and be given the option to either select 0 or 1 for the tile.

Rewards - When the Maze Designer has finished the episodic task it will be given a reward. The reward will be equal to -900 if there is no clear path to the exit (using a hardcoded algorithm). If there is a path to the exit, the reward will equal how many time steps an RL Agent trained to explore mazes took to find it.

Stock N Broker

State - A set of N stock options and their current price + their entire price history. - A separate estimate of the expected value of a stock along with a value for how certain it is. - Total value of money in stocks and total value of money to spend. - Measure of available computational power + how much is in use.

Actions - Sell x amount of i stock from N options - Buy x amount of i stock from N options - Queue research for i stock option from N options. Researching i stock will increase the certainty and update its expected value. Research will be done by a seperate entity, could be a hardcoded algorithm that uses NLP or another reinforcement learning agent, but it updates the expected value function. - Stop reserach for i stock option from N options.

Rewards - At the end of each day, the reward will equal the cumulative difference in total stock amount + spending money from the beginning of the day.

Exercise 3.2

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

The RL framework is able to represent most goal-directed learning tasks that I can think of. I ignore tractability and lack of ability to represent a physical environment because those have nothing to do with the framework's ability to represent a task but more to do with physical issues. We use approximations to deal with lack of tractability and it has been shown that having non Markov environments are still okay.

Difficult tasks to represent would be tasks in which the actions available to be taken are constantly changing, or a task where the rewards are subjective.

A task where the actions are constantly changing must be a very complex and broad task. Perhaps, life itself could be the most complex and broad task available. However abstract this is, this can always be broken down into a set of actions that are constant. Although we have complex actions like "applying for a job" or "painting" or "conversing with someone", and the actions that we need to take can change on a daily basis, we can break these all down into the minute controls of our body. We can make the actions into what electrical signals we pass down our brain stem and what electrical signals we pass to our mouth and eyes.

Deciding the rewards for the task of life is subjective because things are more rewarding to different people. This may be a quandry. However, since the question is asking for a "goal-directed" task, then we can choose something like money, or dopamine, or serotonin as the direct factor for reward. We could also, given a person's philosophy, try to create a representation for it in the form of a set of weights that decide how much reward each of these things like "money", "dopamine", etc. should receive in contributing to a person's "philosophical fulfillment". Perhaps then, the most complicated goal-oriented task could be "living to a person's philosophy".

There's also the problem of representing "fulfillment" in a philosophy, which is a complicated goal and is oriented around a very specific achievement in a person's environment. A person's dream of having a family or an idealized career is a complicated fulfillment that is a part of the person's philosophy. I suppose this could be

simply something coded in their reward function to give them a massive reward of "fulfillment points" if they achieve it, and the difficulty of representing this may be pushed out of the RL framework and into the reward function.

A person's philosophy may also change based on their environment, but then we'd simply have to decide that the person's philosophy is something in the environment as well and is something that the agent has no control over. Their philosophy is a part of their reward function for general "philosophy fulfillment".

This is a very basic representation of "living" and an AI would definitely need a lot more components to be able to live at the capacity of a human but the RL framework can serve as a representation of the problem of "living to a specific set of goals AKA philosophy". It seems to me that if someone is creative enough in deciding a set of actions, states, and rewards to represent a problem then the RL framework can represent any goal-oriented learning task.

Exercise 3.3

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?

The problem of driving is a very broad task that can be broken up into many sub-goals, it could also be kept very abstract as "get from point A to point B as efficiently as possible without breaking any laws or injuring yourself + anyone else."

When the line is drawn at an abstract level, and the agent's goal is abstract, then the agent's distance of control between its actions and its goal is quite close. There is a shorter and more direct feedback loop between the actions chosen and the rewards received.

When the line is drawn at control over the agent's limbs, and the agent's goal is still abstract, then the distance between its actions and the goals achieved is much larger. The agent would have to explore over a much larger space of possibilities before receiving any positive feedback. The agent may learn the correct ways to hit the gas but still go in the wrong direction with its arms, receiving no positive feedback for a partially correct action.

The best place to draw the line really depends on what the target goal is. If you draw the line far from the goal

you'll end up with a larger search space. On the other hand, if the agent is having problems with tire torques and putting rubber to the road, and this is out of its control, then that is also problematic. Where to draw the line also depends on what controls we want to learn.

A good compromise would be to train a separate RL agent for each sub goal- one for learning how to push the gas pedal, one for learning how to properly drive straight and turn, and one for learning how to get from point A to point B, and place their lines respectively according to these sub goals to achieve a larger abstract goal. For training, it would be a good idea to train bottom up since there is a directional reliance with these agents.

Exercise 3.4

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?

For the episodic case, the value estimates would lie between 0 and -1 depending on how many time steps away expected failure is. The closer failure is expected, the closer the value estimate would be to -1.

In the discounted and continuing formulation, the value estimates could be between 0 and any negative number depending on the discounting factor.

Exercise 3.5

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 (3.1). 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?

With no time bound on when to reach a terminal state, the agent will always eventually find its way out of the maze. It always has an expected total reward of 1 no matter what policy it follows. Thus, it has no incentive to solve the maze any more efficiently.

A way to solve this would be give it a small penalty for every time step so that in order to maximize total expected reward it must solve the maze in minimal time steps.

Exercise 3.6:

Broken Vision System Imagine that you are a vision system. When you are first turned on for the day, an image floods into your camera. You can see lots of things, but not all things. You can't see objects that are occluded, and of course you can't see objects that are behind you. After seeing that first scene, do you have access to the Markov state of the environment? Suppose your camera was broken that day and you received no images at all, all day. Would you have access to the Markov state then?

Yes, when you are first turned on you have access to the Markov state because you have not been given access to any information prior to this image. "You can't be penalized for not knowing what you have not seen."

If your camera was broken that day and your received no images at all, you would no longer have access to the Markov state. The system is expected to be watching and to be able to keep track of the state somehow but it is now missing this information. Something could have happened while the camera was broken, a ticking time bomb could have come on screen, and then placed in an occluded area. The robot should be able to use this information to tell that the room is about to look like an explosion but since it malfunctioned it can't know that and thus does not have access to the Markov state.

Exercise 3.7

There is no exercise 3.7.

There is no answer for exercise 3.7..

Exercise 3.8

What is the Bellman equation for action values, that is, for qπ? It must give the action value qπ(s, a) in terms of the action values, qπ(s0, a0), of possible successors to the state–action pair (s, a). As a hint, the backup diagram corresponding to this equation is given in Figure 3.4b. Show the sequence of equations analogous to (3.12), but for action values.

$$q\pi(s,a) = \sum_{s',r} p(s',r|s,a) * [r + \gamma \sum_{a'} q\pi(s',a')]$$

Exercise 3.9

The Bellman equation (3.12) must hold for each state for the value function vπ shown in Figure 3.5b. As an example, show numerically that this equation holds for the center state, valued at +0.7, with respect to its four neighboring states, valued at +2.3, +0.4, –0.4, and +0.7. (These numbers are accurate only to one decimal place.)

Since all the states have reward 0 and equal probability:

$v(centerstate) = \sum_{s'} 0.25 * 1 * [0 + 0.9 * v(s')]$

$= 0.225 * \sum_{s'} v(s')$

$= 0.225 * [2.3 + 0.4 - 0.4 + 0.7]$

$= 0.225 * [3.0] = 0.675 = 0.7*$

Exercise 3.10

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 (3.2), that adding a constant c to all the rewards adds a constant, vc, to the values of all states, and thus does not affect the relative values of any states under any policies. What is vc in terms of c and γ?

$v(s) = E_\pi[\sum_{k=0}^{\infty} \gamma^k (R_{t+k+1} + c) | S_t = s]$

$= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} + \sum_{k=0}^{\infty} \gamma^k c | S_t = s]$

$= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] + \sum_{k=0}^{\infty} \gamma^k c$

Thus all the value functions for states simply have term v_c added to them:

$v_c = \sum_{k=0}^{\infty} \gamma^k c$

Exercise 3.11

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.

For an episodic task, the constant would add varying constants to the value function depending on how close it is to the terminal state. In the case without discouning:

$v(s_t) = E_\pi[\sum_{k=0}^{T-t} (R_{t+k+1} + c) | S_t = s]$

$= E_\pi[\sum_{k=0}^{T-t} R_{t+k+1} + \sum_{k=0}^{T-t} c | S_t = s]$

$= E_\pi[\sum_{k=0}^{T-t} R_{t+k+1} | S_t = s] + (T - t) * c$

$v_c = (T - t) * c$

And in the case with discounting:

$v(s_t) = E_\pi[\sum_{k=0}^{T-t} \gamma^k (R_{t+k+1} + c) | S_t = s]$

$= E_\pi[\sum_{k=0}^{T-t} \gamma^k R_{t+k+1} + \sum_{k=0}^{T-t} \gamma^k c | S_t = s]$

$= E_\pi[\sum_{k=0}^{T-t} \gamma^k R_{t+k+1} | S_t = s] + \sum_{k=0}^{T-t} \gamma^k c$

$v_c = \sum_{k=0}^{T-t} \gamma^k c$

In both cases we see that the further we are from the terminal state, the greater the effect the constant has on the value function. Having larger rewards will result in a more dynamic range of values from states farther and closer to the reward.

If are positive rewards at each time step it will incentivize the agent to prolong the episode to maximize total expected rewards.

Example: Maze runner. Let's consider a maze running problem where there are various cherrys around the maze to be picked up that give the agent a reward of 1 point. Exiting the maze gives the agent 10 points and ends the game. In a continuous model the agent would be more likely to desire completing the maze as fast as possible in order to keep getting 10 points continuously. However, in an episodic model the agent would want to collect as many points as possible before finishing the maze.

Thus, the relative directions of the rewards matter in an episodic model. Agents are looking at *total* reward per episode so if there are no negative rewards it would want to collect all rewards before finishing. If there are random "points" that can be gained intermittently it may even end up never finishing the game because then it can maximize T-t. So for an episodic model with points like this, there must be balance from negative points like a penalty over each time step.

Exercise 3.12

The value of a state depends on the 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, vπ(s), in terms of the value at the expected leaf node, qπ(s, a), given St = s. This expectation depends on the policy, π. Then give a second equation in which the expected value is written out explicitly in terms of π(als) such that no expected value notation appears in the equation.

$v_\pi(s) = E_\pi[\gamma q_\pi(s, A_{t+1}) | S_t = s]$

$v_\pi(s) = \sum_a \pi(a|s) * \gamma q_\pi(s, a)$

Exercise 3.13

The value of an action, qπ(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, qπ(s, a), in terms of the expected next reward, Rt+1, and the expected next state value, vπ(St+1), given that St = s and At = a. Then give a second equation, writing out the expected value explicitly in terms of p(s0, rls, a) defined by (3.6), such that no expected value notation appears in the equation.

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

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

Exercise 3.14

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

The optimal state-value function for the golf example would equal to the maximum expected return value from state s depending on which action is selected. For the golf problem state is described as the negative of how many strokes away the ball is from the hole in an optimal case. However, because there is an accuracy distribution we have to take into account the probabilites of getting to a state s+1 (in this case +1 is good) given a certain action. The optimal state-value function will equal to the largest expected value of taking an action a.

Where p, d = putter, driver

$v*(s) = max_{a \in (p,d)}$
{p(s-1,-1ls,p) * [-1 + γv*(s-1)],

p(s,-1ls,p) * [-1 + γv*(s)],

p(s-1,-1ls,d) * [-1 + γv*(s-1)],

p(s,-1ls,d) * [-1 + γv*(s)]}

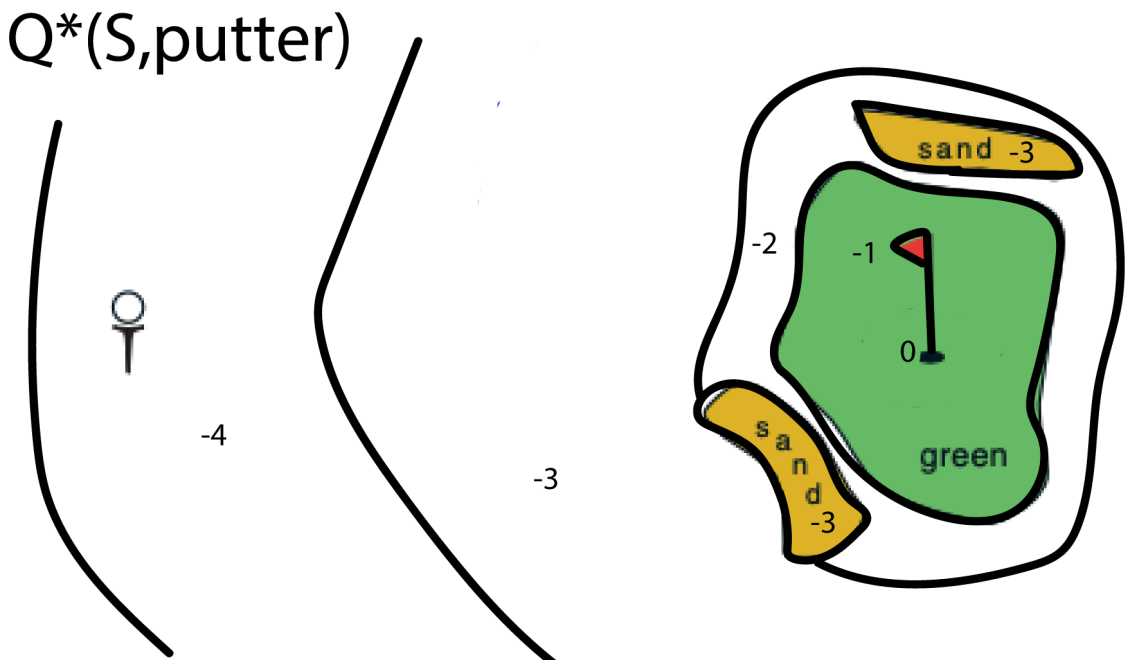
However, since we are assuming we can putt/drive with optimal accuracy then p(s-1, -1ls,p) is either equal to 1 or 0 and since the driver has the longest distance possible the optimal choice is to just use the driver each time.

$v*(s) = -1 + \gamma v*(s-1)$

Exercise 3.15

Draw or describe the contours of the optimal action-value function for putting, q∗(s, putter), for the golf

example.



From anywhere in the -4 area the agent can putt once, then use the driver 3 times to get it into the hole.

From anywhere in the -3 area the agent can putt once, then use the driver twice to get into the hole. This includes the sand areas because when it putts once, it goes nowhere, then they would use the driver to get it in the green and can either putt or use the driver again to get it into the hole.

From anywhere in the -2 area the agent can putt once, get it in the green, then use the putter to get it into the hole.

From anywhere on the green it only takes one putt to get it into the hole.

Exercise 3.16

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

$$q_*(l, re) = \gamma * max_a\{q_*(h, s), q_*(h, w)\}$$

$$q_*(l, s) = (1 - \beta) * [-3 + \gamma * max_a\{q_*(h, s), q_*(h, w)\}] + \beta * [r_s + \gamma * max_a\{q_*(l, s), q_*(l, w), q_*(l, re)\}]$$

$$q_*(l, w) = r_w + \gamma * max_a\{q_*(l, s), q_*(l, w), q_*(l, re)\}$$

$$q_*(h, w) = r_w + \gamma * max_a\{q_*(h, s), q_*(h, w)\}$$

$$q_*(h, s) = (1 - \alpha) * [r_s + \gamma * max_a\{q_*(l, s), q_*(l, w), q_*(l, re)\}] + \alpha * [r_s + \gamma * max_a\{q_*(h, s), q_*(h, w)\}]$$

$q_*(n,s) = (1-\alpha) * [r_s + \gamma * max_a\{q_*(i,s), q_*(i,w), q_*(i,e)\}] + \alpha * [r_s + \gamma * max_a\{q_*(n,s), q_*(n,w)\}]$

Exercise 3.17

Figure 3.8 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 (3.2) to express this value symbolically, and then to compute it to three decimal places.

$v*(s) = \max_a\{p(s', r|s, a) * [r + \gamma * v*(s')]\}$
Since there is only one action to be done and one state to result in:

$v*(0, 1) = 1 * [10 + \gamma * v*(s')] = 1 * [10 + 0.9 * 16.0] = 24.4$
But this is based on our 1 decimal accuracy so we get a 1 decimal answer...

We must calculate v*(4,1) more accurately. I took the liberty of coding a dynamic programming approach to compute the value. The value I arrived at calculating up to 3 decimal point accuracy was: 24.419

Exercise 3.18

Give a definition of v$_*$ in terms of q$_*$.

$v_*(s) = \sum_a \pi_*(a|s) * \gamma q_*(s, a)$

Exercise 3.19

Give a definition of q$_*$ in terms of v$_*$.

$q_*(s, a) = \sum_{s'} p(s', r|s, a) * [r + \gamma v_*(s')]$

Exercise 3.20

Give a definition of π$_*$ in terms of q$_*$.

$\pi_*(a|s) = (1 if a \in max_{a'}\{q(s, a')\} else 0)/(total number of max_{a'}\{q(s, a')\})$

Exercise 3.21

Give a definition of π$_*$ in terms of v$_*$.

$\pi_*(a|s) = (1 if a \in max_{a'} \sum_{s', r} p(s', r|s, a') * [r + \gamma * v*(s')])$

$$\pi * (a|s) = (1 if\ a \in max_{a'} \sum_{s'} p(s', r|s, a') * [r + v * (s')])$$
$$/(totalnumberofmax_{a'} \sum_{s'} p(s', r|s, a') * [r + v * (s')]$$