Reinforcement Learning

COMP2002

Lauren Ansell

Introduction

Today's topics:

- Reinforcement learning
- Markov decision process

Session learning outcomes - by the end of today's lecture you will be able to:

- Explain what reinforcement learning is.
- Explain what a Markov decision process is.

Reinforcement Learning

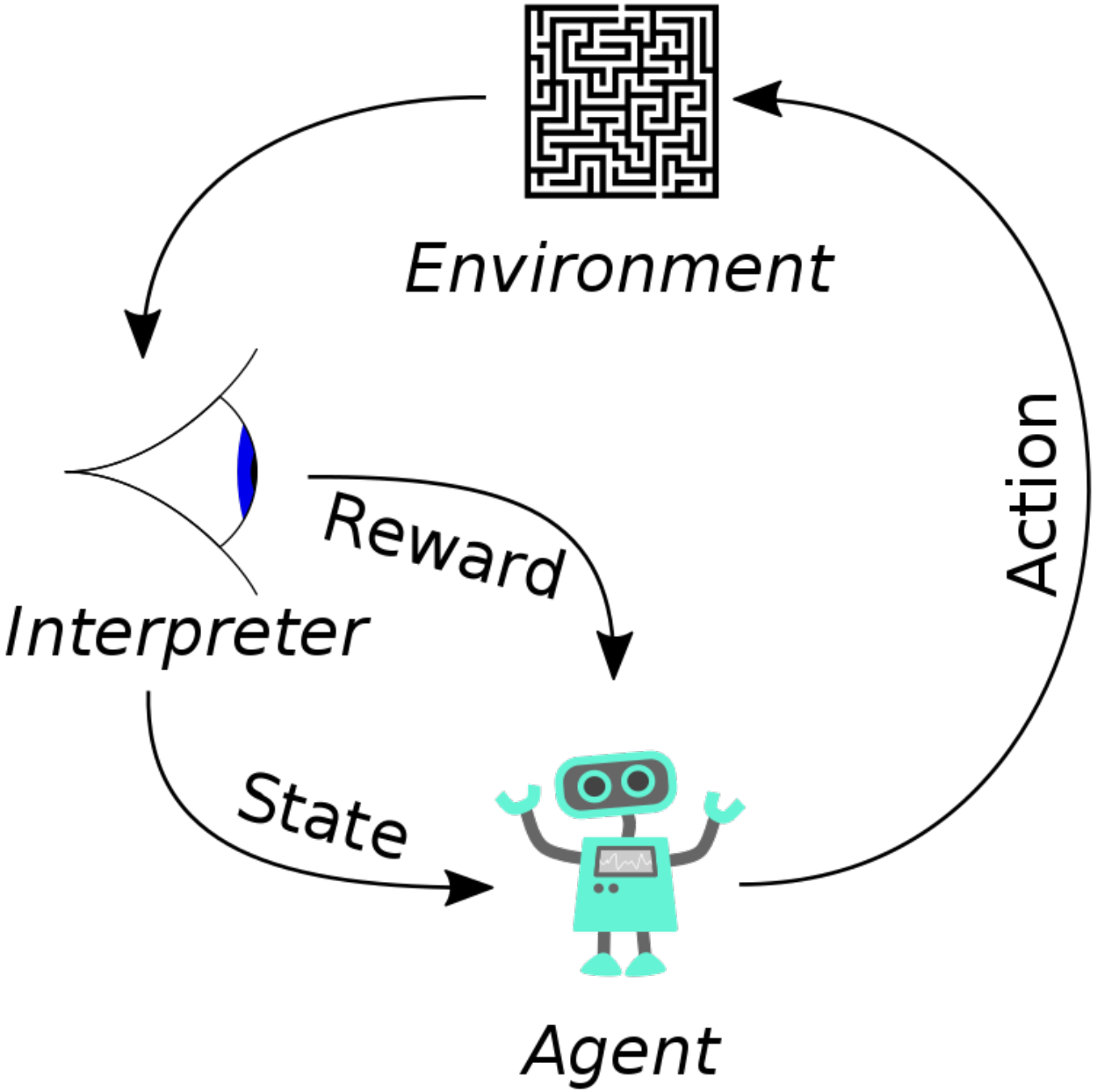
Reinforcement learning (RL) is concerned with how an intelligent agent ought to take actions in a dynamic environment in order to maximize the cumulative reward.

Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

Reinforcement learning does not require labelled input/output pairs.

We also do not need to correct sub-optimal actions.

The focus is on finding a balance between exploration and exploitation with the goal of maximizing the long term reward.


Environment
Reward
Action
Interpreter
State
Agent

Elements Of Reinformcement Learning

A basic reinforcement learning agent AI interacts with its environment in discrete time steps.

At each time t, the agent receives the current state S_t and reward R_t.

It then chooses an action A_t from the set of available actions, which is subsequently sent to the environment.

The environment moves to a new state S_{t+1} and the reward R_{t+1} associated with the transition (S_t, A_t, S_{t+1}) is determined.

The goal of a reinforcement learning agent is to learn a policy: $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, $\pi(s, a) = \Pr(A_t = a \mid S_t = s)$ that maximizes the expected cumulative reward.

The problem is formulated as an Markov decision process and assumes the agent directly observes the current environmental state.

If the agent only has access to a subset of states, or if the observed states are corrupted by noise, the agent is said to have partial observability.

In both cases, the set of actions available to the agent can be restricted.

Markov Decision Process

Basic reinforcement learning is modeled as a Markov decision process:

- a set of environment and agent states, $\mathcal{S}$
- a set of actions, $\mathcal{A}$, of the agent;
- $P_a(s, s') = \Pr(S_{t+1} = s' | S_t = s, A_t = a)$, the probability of transition (at time t) from state s to state s' under action a.
- $R_a(s, s')$, the immediate reward after transition from s to s' with action a.

Uses Of Reinforcement Learning

Reinforcement learning is particularly well-suited to problems that include a long-term versus short-term reward trade-off.

It has been applied successfully to various problems:

- energy storage operation
- robot control
- photovoltaic generators dispatch
- boardgames
- autonomous driving systems

Why Reinforcement Learning?

Two elements make reinforcement learning powerful:

- the use of samples to optimize performance
- the use of function approximation to deal with large environments.

Thanks to these two key components, reinforcement learning can be used in large environments in the following situations:

- A model of the environment is known, but an analytic solution is not available
- Only a simulation model of the environment is given (the subject of simulation-based optimization)
- The only way to collect information about the environment is to interact with it

Policy

The agent's action selection is modeled as a map called policy:

$$\pi : \mathcal{A} \times \mathcal{S} \to [0, 1] \tag{1}$$

$$\pi(a, s) = \Pr(A_t = a \mid S_t = s) \tag{2}$$

The policy map gives the probability of taking action a when in state s.

There are also deterministic policies.

State-value Function

The state-value function $V_\pi(s)$ is defined as, expected discounted return starting with state s, i.e. $S_0 = s$, and successively following policy π.

$$V_\pi(s) = \mathbb{E}[G \mid S_0 = s] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s\right], \tag{3}$$

where the random variable G denotes the discounted return, and is defined as the sum of future discounted rewards:

$$G = \sum_{t=0}^{\infty} \gamma^t R_{t+1} = R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots, \tag{4}$$

where R_{t+1} is the reward for transitioning from state S_t to S_{t+1}, $0 \leq \gamma < 1$ is the discount rate.

Value Function

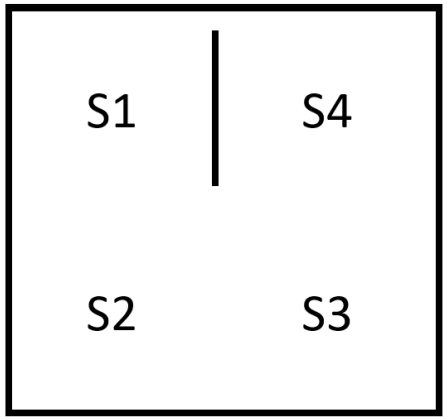
Value function approaches attempt to find a policy that maximizes the discounted return by maintaining a set of estimates of expected discounted returns $\mathbb{E}[G]$ for some policy.

These methods rely on the theory of Markov decision processes, where optimality is defined in a sense stronger than the one above: A policy is optimal if it achieves the best-expected discounted return from any initial state.

An optimal policy can always be found among stationary policies.

Reinforcement Learning Example - Gridworld

The aim is for the agent to find its way from a starting position to the final position.



The agent can take one of four possible actions.

If the agent reaches the goal position, it earns a reward of 10.

Routes that are not the shortest path have a reward of -1.

Creating The States, Actions And Environment

First, the states and actions need to be defined:

```
states <- c("s1", "s2", "s3", "s4")
actions <- c("up", "down", "left", "right")
```

The environment is then defined by a function with two inputs: state and action.

Generated 1000 random state tuples $(s_i, a_i, r_{i+1}, s_{i+1})$

State	Action	Reward	NextState
s4	right	-1	s4
s1	left	-1	s1
s1	up	-1	s1
s2	right	-1	s3
s3	left	-1	s2
s4	left	-1	s4

This observation sequence can now be used to learn the optimal behaviour for the agent.

We first customize the learning behavior of the agent by defining a control object.

We set the learning rate alpha to 0.1, the discount factor gamma to 0.5 and the exploration greediness epsilon to 0.1.

```
control <- list(alpha = 0.1, gamma = 0.5,
epsilon = 0.1)
```

Evaluating Policy Learning

We can view the policy that defines the best possible action in each state.

```
computePolicy(model)
 s1     s2       s3       s4
 "down" "right" "up" "right"
```

Alternatively, we can view the entire state-action table to the screen, i.e. the Q-value of each state-action pair.

```
> print(model)
State-Action function Q
          right           up         down         left
s1 -0.7206538 -0.7164033  0.6969987 -0.7116979
s2  3.5492453 -0.7923964  0.7113680  0.7204828
s3  3.5623270  9.1165862  3.5473654  0.7322279
s4 -1.8641853 -1.8967686 -1.8735248 -1.8902794

Policy
      s1       s2       s3       s4
 "down" "right"     "up" "right"

Reward (last iteration)
[1] -230
```

We can output additional diagnostics regarding the model.

This allows us to analyze the distribution of rewards.

```
Model details
Learning rule:  experienceReplay
Learning iterations:  1
Number of states:  4
Number of actions:  4
Total Reward:  -230

Reward details (per iteration
Min:  -230
Max:  -230
Average:  -230
Median:  -230
Standard deviation:  NA
```

Applying A Policy To Unseen Data

We create a new sequence of states to feed into our model.

```
data_unseen <- data.frame(State = c("s1",
"s2", "s1"),stringsAsFactors = FALSE)

data_unseen$OptimalAction <- predict(model,
data_unseen$State)
```

State	OptimalAction
s1	down
s2	right
s1	down

The we can update an existing policy with new observational data.

This is beneficial when, for instance, additional data points become available or when one wants to plot the reward as a function of the number of training samples.

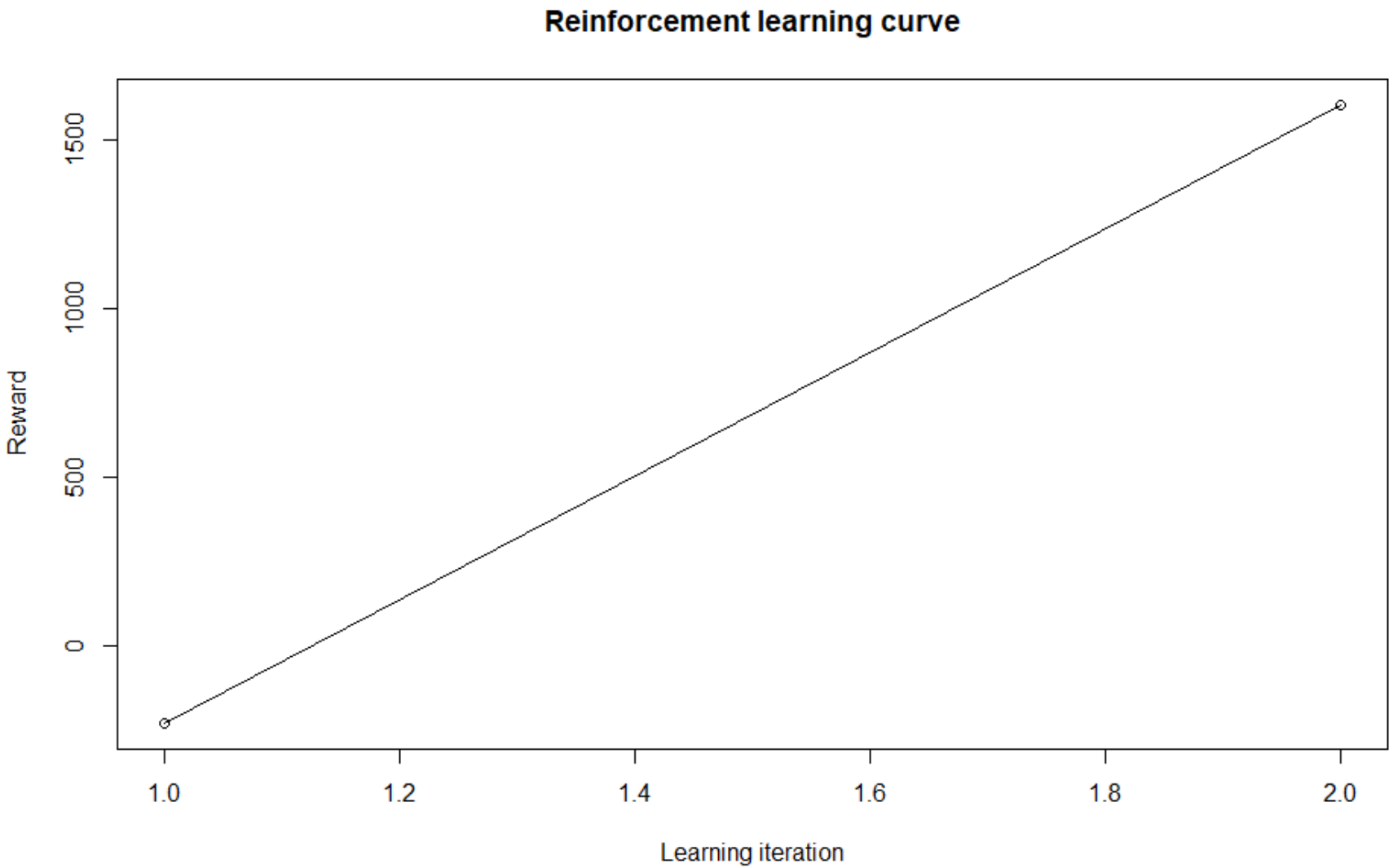
We can now re-evaluate the model, using the original model as an input parameter.

This time we will also define a selection mode, namely ϵ -greedy, thereby following the best action with probability 1- ϵ and a random one with ϵ.

```
State-Action function Q
          right           up         down         left
s1 -0.7011335 -0.6523225  0.7617176 -0.6936998
s2  3.5233217 -0.7326453  0.7294779  0.7355295
s3  3.5424044  9.0465074  3.5373335  0.7429292
s4 -1.9534772 -1.9152825 -1.9070291 -1.9320969

Policy
     s1       s2       s3       s4
 "down" "right"     "up"   "down"

Reward (last iteration)
[1] 1607
```


Reinforcement learning curve
Reward
0
500
1000
1500
1.0
1.2
1.4
1.6
1.8
2.0
Learning iteration

Summary

Reinforcement learning

- RL tells us how an agent takes actions in a dynamic environment to maximize the cumulative reward.
- Do no require labelled inputs or outputs.
- Uses Markov decision processes to calculate the return of taking a particular set of actions.