Emergent Software Systems

Summer School

Barry Porter & Roberto Rodrigues Filho
School of Computing and Communications
Lancaster University

Funded by The Royal Society Newton Fund

Reinforcement Learning

- General concepts
  - Actions, rewards, states, and environments
- Multi-armed bandits
  - Concepts and the UCB1 algorithm

Reinforcement Learning

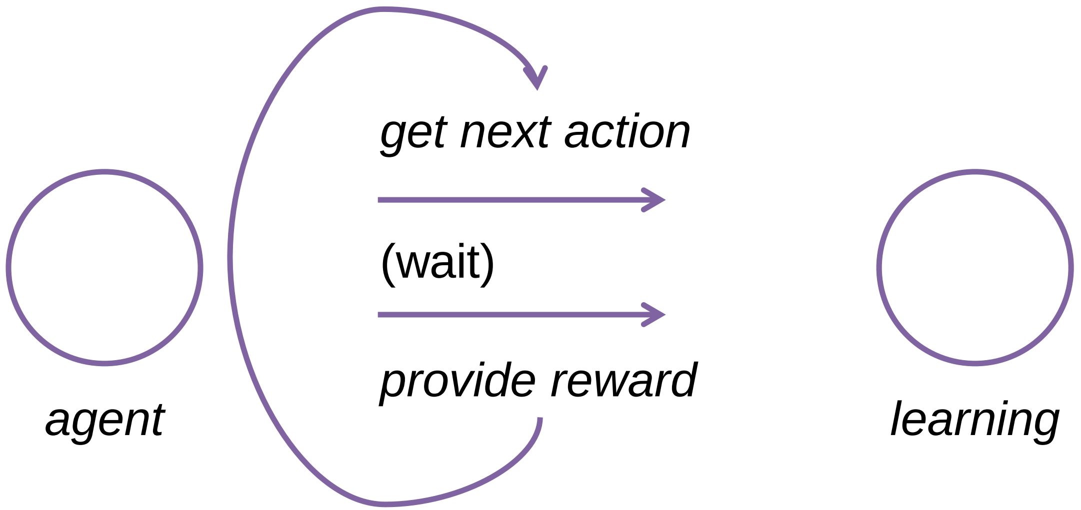
- This is a real-time form of learning which considers a set of **actions** and an associated **reward** for each action
  - We can also consider a set of *states* and an *environment* which can modify the set of available actions and their rewards

- Reinforcement learning focuses on balancing *exploration* with *exploitation* in trying to maximise overall reward – often by trading off short-term pain for long-term gain

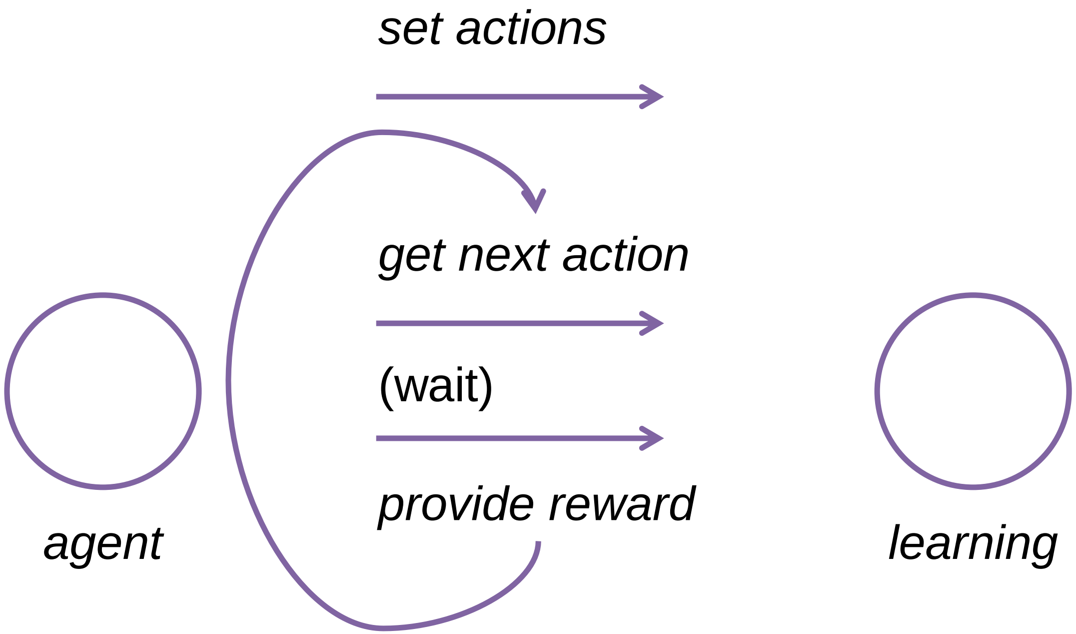
Reinforcement Learning

- This kind of learning is a good fit for emergent software
  - It starts from no information, learning everything from experience
  - It requires no human supervision or training data
  - It is based only on the ability to take actions and observe rewards

Reinforcement Learning

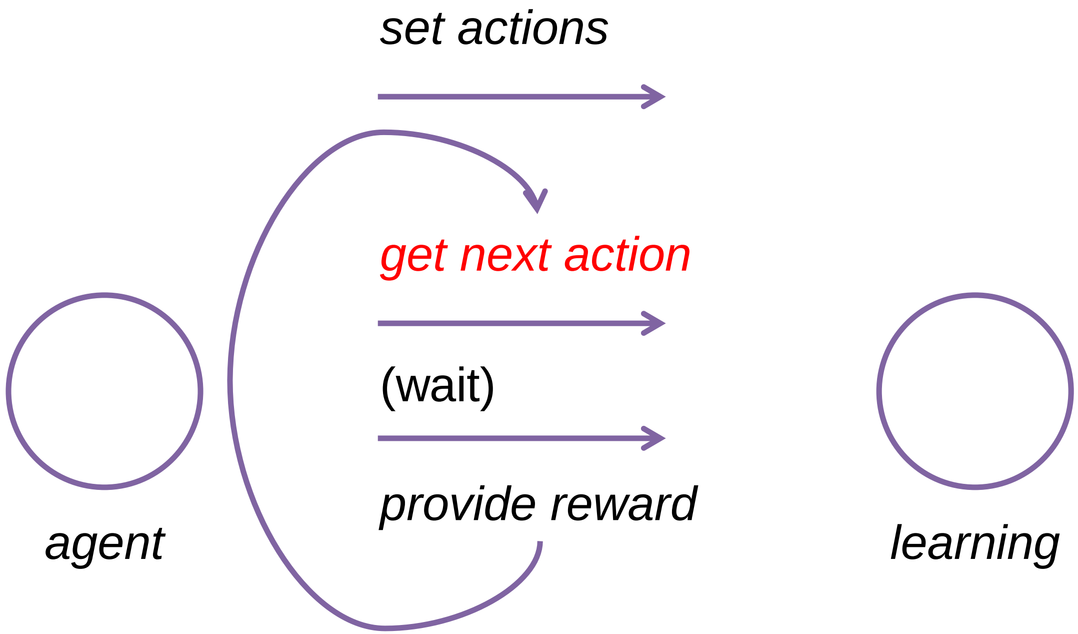


Reinforcement Learning



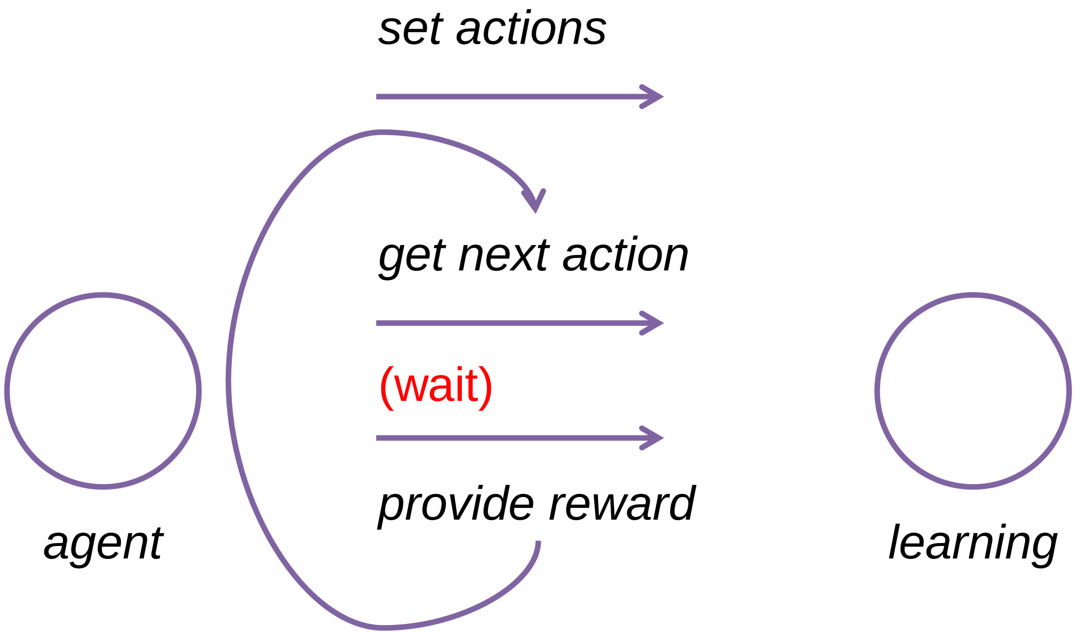
Actions	Rewards
a	Ø
b	Ø
c	Ø
d	Ø
e	Ø

Reinforcement Learning



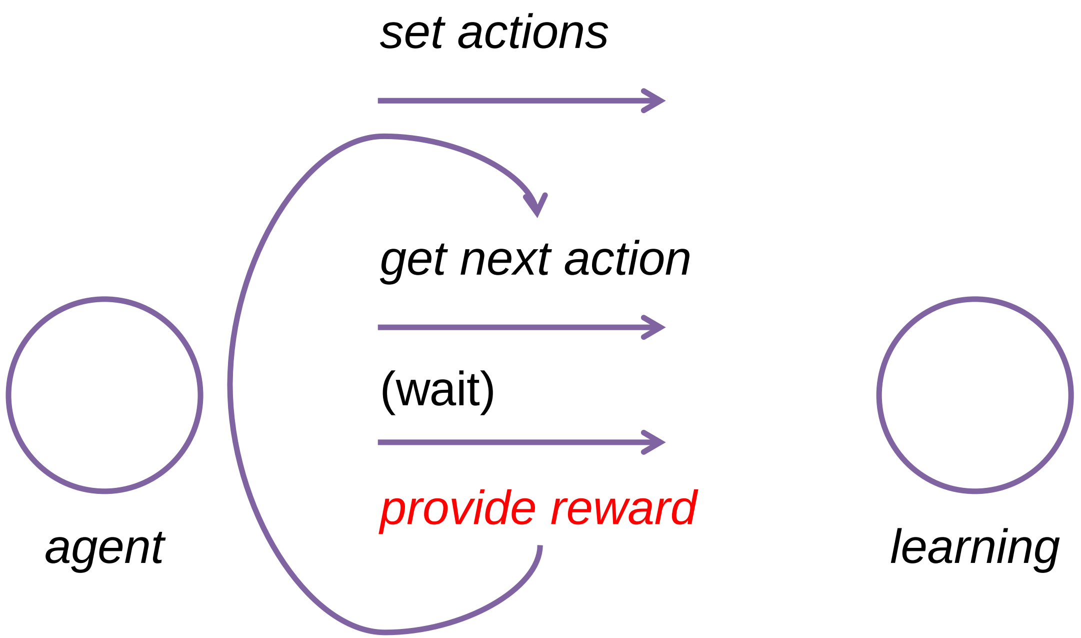
Actions	Rewards
a	**Ø**
b	Ø
c	Ø
d	Ø
e	Ø

Reinforcement Learning



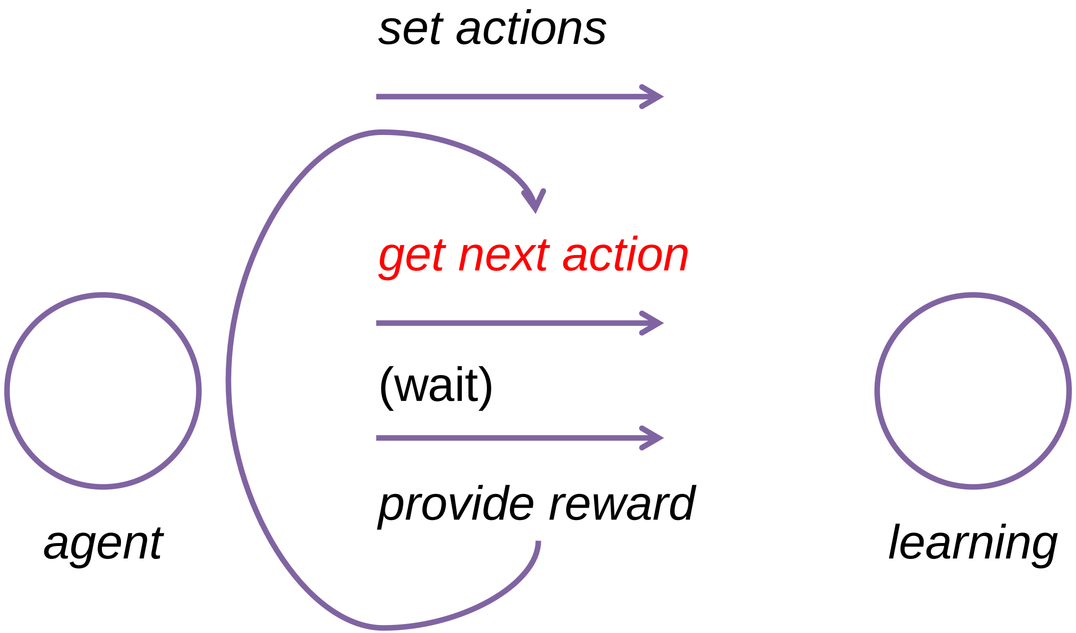
Actions	Rewards
a	**Ø**
b	Ø
c	Ø
d	Ø
e	Ø

Reinforcement Learning



Actions	Rewards
a	**0.72**
b	Ø
c	Ø
d	Ø
e	Ø

Reinforcement Learning



Actions	Rewards
a	0.72
b	Ø
c	**Ø**
d	Ø
e	Ø

Other concerns

- Are rewards deterministic? Or is there some level of (stochastic?) variance from the environment?
- How large is the space of possible actions? How do we navigate that space quickly?
- How do we observe a reward after an action is taken? Do we need to wait some time to get a stable reward signal?
- What if the environment changes?

Multi-Armed Bandits

Multi-Armed Bandits

- This is a field of probability theory from statistics, and is one of the most heavily studied models for balancing exploration and exploitation in reinforcement learning
- It is equivalent to a single-state Markov decision process
- MABs have been studied for decades, and have many variations which try to find an optimal solution

Multi-Armed Bandits

- The idea is that we have a set of **arms** that we can pull (equivalent to actions) and the reward for each arm follows a certain (unknown) probability distribution

- Imagine we have 3 arms:



Multi-Armed Bandits

- Imagine we have 3 arms:

- To begin with we know nothing about any of the rewards yielded by each arm, so it doesn't matter which one we try

Multi-Armed Bandits

- Imagine we have 3 arms:

- Trying an arm gives us *one reward sample* from that arm
- A naïve implementation may assume that this (very high) reward is always true and so keep playing it forever…

Multi-Armed Bandits

- Imagine we have 3 arms:

- Trying an arm gives us *one reward sample* from that arm
- A naïve implementation may assume that this (very high) reward is always true and so keep playing it forever…

Multi-Armed Bandits

- Imagine we have 3 arms:

0.9
0.1
0.2

- Trying an arm gives us *one reward sample* from that arm
- A naïve implementation may assume that this (very high) reward is always true and so keep playing it forever…

Multi-Armed Bandits

- Imagine we have 3 arms:

0.9
0.1
0.2

- The less often we see a high reward from an arm, the more we might start to believe it’s a bad long-term choice

Multi-Armed Bandits

- Imagine we have 3 arms:

0.9	0.5	0.1
0.1	0.5	0.3
0.2	0.6	0.2

- Exploring other arms tells us more about the probability distributions of their rewards
- We can say that we have both a *reward distribution* and a *certainty level* about that distribution for each arm

Multi-Armed Bandits

- One of the simplest implementations of a MAB is called **Upper Confidence Bound 1** (UCB1)
- This is a deterministic algorithm which eventually settles on the best long-term choice *on average* while in theory allowing every action to be tried infinitely often

Multi-Armed Bandits

- We model our confidence bound as a logarithmic function over the total number of actions taken

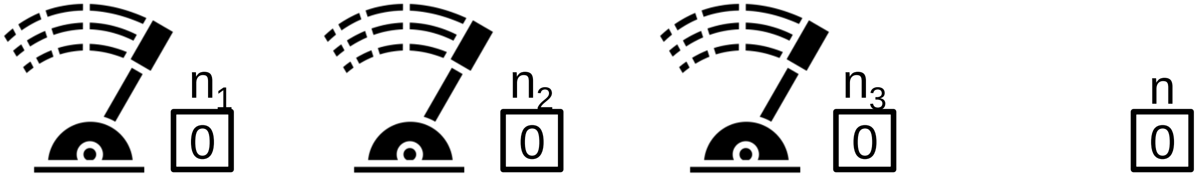
- This works by choosing our next action *j* to maximise:

$$\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

- $\bar{x}_j$ is the total average reward seen for action *j*
- n_j is the number of times we've tried action j
- n is the total number of actions we've ever taken

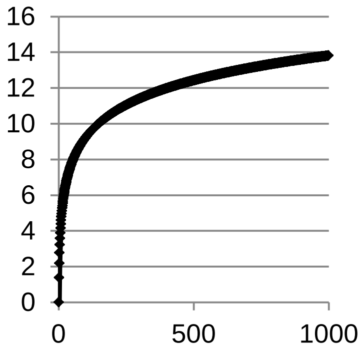
Multi-Armed Bandits

- In practice:



getAction()

putReward(r)



2 ln *n*

$$\sqrt{\frac{2 \ln n}{n_j}}$$

$$\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

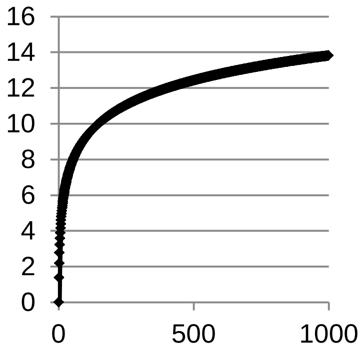
Multi-Armed Bandits

- In practice:

getAction()

putReward(r)

n_1	n_2	n_3	n
$\boxed{1}$	$\boxed{1}$	$\boxed{1}$	$\boxed{3}$
0.0 0.9	0.0 0.2	0.0 0.7	
1.5	1.5	1.5	$\sqrt{\frac{2\ln n}{n_j}}$
3.4	1.7	2.2	$\bar{x}_j + \sqrt{\frac{2\ln n}{n_j}}$



2 ln *n*

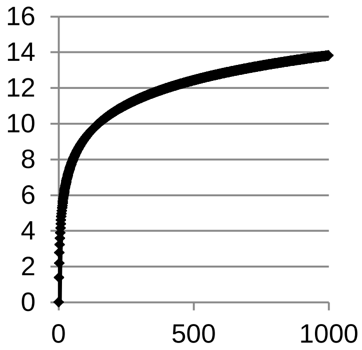
Multi-Armed Bandits

- In practice:

getAction()

putReward(r)

			
n_1 $\boxed{2}$	n_2 $\boxed{1}$	n_3 $\boxed{1}$	n $\boxed{4}$
0.0 0.9 0.3	0.0 0.2	0.0 0.7	
0.9	1.7	1.7	$\sqrt{\frac{2 \ln n}{n_j}}$
1.5	1.9	2.14	$\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$



2 ln *n*

Multi-Armed Bandits

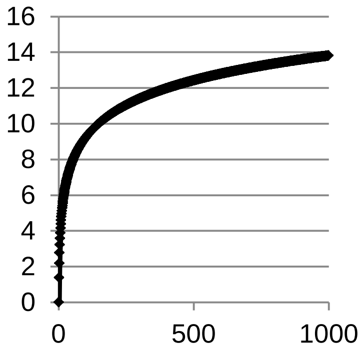
- In practice:

getAction()

putReward(r)



	n_1	n_2	n_3	n
	$\boxed{2}$	$\boxed{1}$	$\boxed{2}$	$\boxed{5}$
	0.0 0.9 0.3	0.0 0.2	0.0 0.7 0.8	
$\sqrt{\frac{2 \ln n}{n_j}}$	0.9	1.8	0.9	
$\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$	1.5	2.0	1.65	



2 ln *n*

Multi-Armed Bandits

- Adding *environment*

env = Ø

act a = lrn[env].getAction()

env = getEnvironment()

lrn[env].putReward(a, r)

Env. A

getAction()

putReward(a, r)

Env. B

getAction()

putReward(a, r)

If new environment, add new instance

Summary

- Reinforcement learning algorithms attempt to balance periods of exploration versus exploitation, such that the total reward over time is maximised (when compared against a perfect oracle)
- Multi-armed bandit theory has studied this balance extensively, with many variations
- UCB1 is one of the simplest to implement

Practical

- We’ll implement a version of UCB1 and experiment with its properties, then attach it to our emergent software