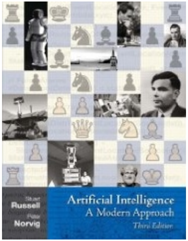
Reinforcement Learning 1

Oana Cocarascu & Helen Yannakoudakis

Department of Informatics
King's College London

Today

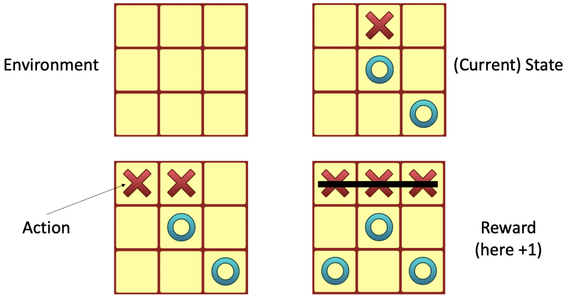
- Simple bandit problems.
- Markov Decision Processes.
  Mathematical and computational basis of reinforcement learning.
- Passive reinforcement learning.
- Today's lecture is from:



Different types of learning

- Supervised learning: learning from a training set of labelled examples.
- Unsupervised learning: finding structure hidden in unlabelled data.
- Reinforcement learning (RL): learning what to do (i.e. how to map situations to actions) to maximise a numerical reward signal.

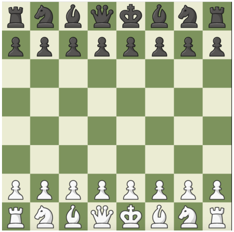
Reinforcement learning



- State space: an agent occupies a given state at a given time.
- An action moves the agent from one state to another.
- In RL, there is no involvement of any human interaction.
- An agent is placed in an environment, is given a goal and learns how to behave in this environment by trial and error using feedback from its own actions and experiences.

Reinforcement learning

- The agent needs to know that something good has happened or that something bad has happened.
- This kind of feedback is called a reward or reinforcement.
- In chess, the reinforcement is received only at the end of the game.
- In table tennis, each point scored can be considered a reward.



- RL in a nutshell: Imagine playing a new game whose rules you do not know; after a number of moves, your opponent announces whether you have lost or won.

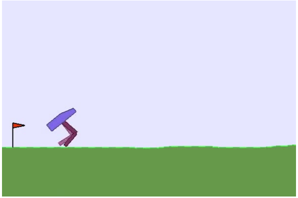
Reinforcement learning characteristics

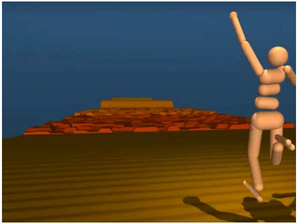
- Goal-oriented learning
- Interaction with an uncertain environment
- Learning how to map situations to actions to maximise a numerical reward signal
- Learner is not told which actions to take
- Trial and error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains.
- Exploration vs. exploitation
  - Exploration finds more information about the environment and might enable higher future reward.
  - Exploitation exploits known information to maximise reward, leading to an immediate reward.

Exploration vs. exploitation examples

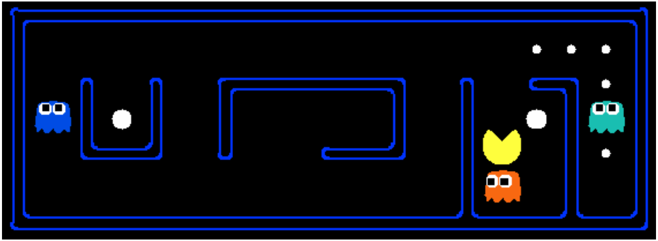
- Game playing:
  - Exploitation: Play the move you believe is best
  - Exploration: Play an experimental move
- Restaurant choice:
  - Exploitation: Go to your favourite restaurant
  - Exploration: Try a new restaurant
- Route home:
  - Exploitation: Take the normal route
  - Exploration: Try another route
- Hollywood studio:
  - Exploitation: Fast and the Furious (25th movie in the series)
  - Exploration: New movies

Reinforcement learning in practice





Reinforcement learning in person



- Coursework 2 will be using RL to play Pacman (and reasonably well).

What kind of problems?

- Evaluative vs instructive feedback
  - Evaluative feedback = how good was the action.
    Does not say whether that was the best thing to do.
  - Instructive feedback = what was the best action.
    Does not rate the action taken.
- Associative vs. non-associative learning
  - Assocative maps inputs to outputs and learns the best output for each input.
  - Non-associative learns the one best output.
- RL works with evaluative feedback.
- We will start with non-associative learning.

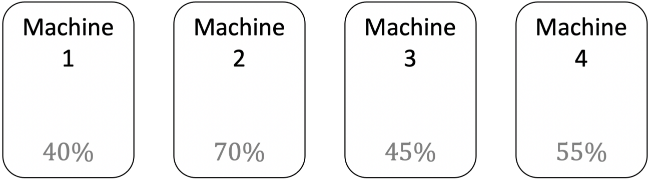
n-armed bandits

- Example: playing a pair of slot machines. Choice is between playing machine A and machine B.
- With many machines, you must choose which lever to play on each successive coin: the one that has paid off best, or maybe one that you have not tried?

- How can we formally analyse this kind of situation?
- n-armed bandit (n actions to choose from)

n-armed bandits

- Classic reinforcement learning example.
- The goal is to maximise the total reward in the long run.
- Which machine to select next?
- Reward probabilities are not known.



n-armed bandits

- Choose repeatedly from one of n actions. Each choice is called a play
- Action values: after each play a_t, you get a reward r_t, where:

$$E\langle r_t | a_t \rangle = Q^*(a)$$

- Distribution of r_t depends only on a_t.
- Objective is to maximise the reward in the long term.
  - Say over 1000 plays
- To solve the n-armed bandit problem, you must explore a variety of actions and then exploit the best of them.

Exploration vs. exploitation

- Suppose you form action value estimates, i.e. what each action is worth at each point in time:

$$Q_t(a) \text{ which estimates } Q^*(a)$$

- Have a $Q_t(a)$ estimate for every a.
- If you maintain estimates of the action values, then there is (at least) one action whose estimated value is greatest at any time step (i.e. greedy).
- The greedy action at t is:

$$a_t^* = \arg\max_a Q_t(a)$$

- Which action should you pick?

Exploration vs. exploitation

- The greedy action at t is

$$a_t^* = \arg\max_a Q_t(a)$$

- Picking a_t^* is exploitation.
- Picking $a_t \neq a_t^*$ is exploration.
- Exploitation maximises the expected reward on one step but exploration may produce the greater total reward in the long run.

    $\rightarrow$ You cannot exploit all the time.
    $\rightarrow$ You cannot explore all the time.

- You can never stop exploring but you should eventually reduce exploring.

Action-value methods

- "Sample-average" method: maintain a list of all rewards received and average them for each action.
- If by the t-th play, action a had been chosen k_a times prior to t, yielding rewards $r_1, r_2, \ldots, r_{k_a}$ then:

$$Q_t(a) = \frac{r_1 + r_2 + \ldots + r_{k_a}}{k_a}$$

- If $k_a = 0$, set $Q_t(a)$ to some default value, e.g. $Q_1(a) = 0$.
- We have:

$$\lim_{k_a \to \infty} Q_t(a) = Q^*(a)$$

- Thus, if we update often enough, the estimate of the value of each action $Q_t(a)$ will converge to the actual value of the action $Q^*(a)$.
- The "sample-average" method uses too much memory. Instead, use incremental implementation.

Incremental implementation

- If Q_k is the average of the first k rewards and r_{k+1} is the $k+1$th reward, then:

$$
\begin{aligned}
Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\
&= \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{k} r_i\right) \\
&= \frac{1}{k+1}\left(r_{k+1} + kQ_k\right) \\
&= \frac{1}{k+1}\left(r_{k+1} + kQ_k + Q_k - Q_k\right) \\
&= \frac{1}{k+1}\left(r_{k+1} + (k+1)Q_k - Q_k\right) \\
&= Q_k + \frac{1}{k+1}\left(r_{k+1} - Q_k\right)
\end{aligned}
$$

ϵ-greedy

- Given an estimate of a reward for each action, we then have to decide what to do.
- Greedy action selection:

$$\begin{aligned} a_t &= a_t^* \\ &= \arg\max_a Q_t(a) \end{aligned}$$

- Greedy action selection always exploits current knowledge to maximise immediate reward.
- But we do not want to just exploit.
- ϵ-greedy gives us a way to balance exploration-exploitation.

ϵ-greedy

- Pick $\epsilon \ll 1$
- ϵ-greedy action selection:

$$a_t = \begin{cases} a_t^* & \text{with probability} \quad 1 - \epsilon \\ \textit{random action} & \text{with probability} \quad \epsilon \end{cases}$$

- Most of the time be greedy, but explore sometimes.
- This is the simplest way to try to balance exploration and exploitation.

KING'S College LONDON

10-armed testbed

- Bandit with $n = 10$ possible actions.
- Each $Q^*(a)$ is chosen randomly from a normal distribution:

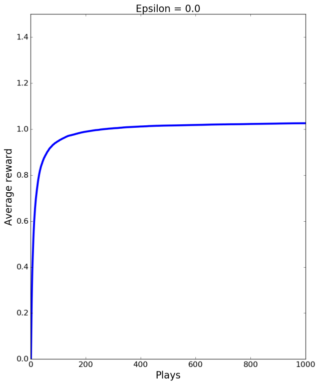
$$\mathcal{N}(\mu = 0, \sigma = 1)$$

- Each r_t is also chosen from a normal distribution:

$$\mathcal{N}(\mu = Q^*(a_t), \sigma = 1)$$

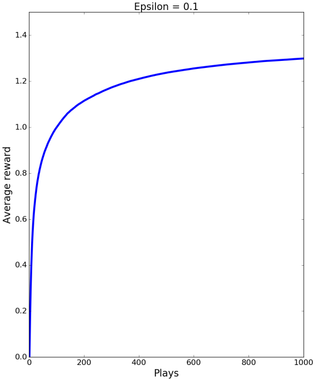
- The 10-armed testbed: 1000 plays and 2000 randomly generated 10-armed bandit tasks.

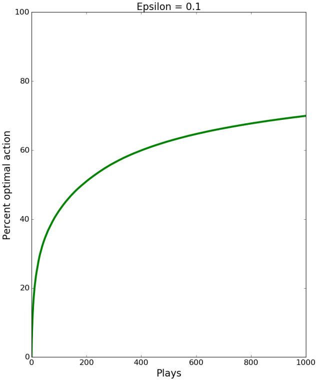
10-armed testbed



- Greedy agent.
  - $\longrightarrow$ levels off at an average reward of about 1
  - $\longrightarrow$ finds the optimal action in approximately 1/3 of the tasks

10-armed testbed



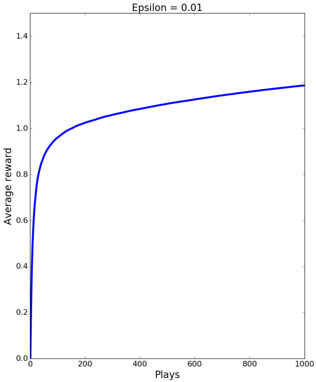


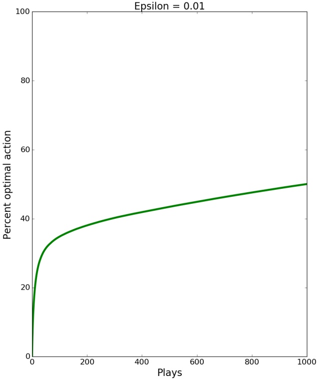
- $\epsilon = 0.1$

  $\rightarrow$ explores more

  $\rightarrow$ finds the optimal action earlier

10-armed testbed



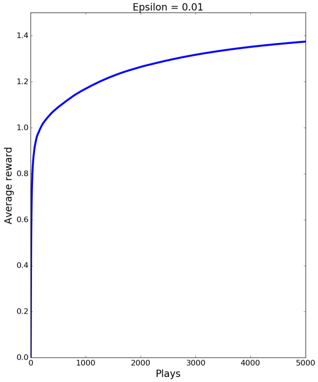


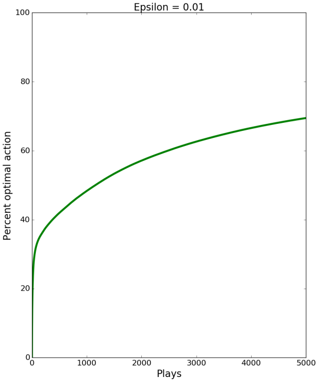
- $\epsilon = 0.01$

  $\rightarrow$ improves more slowly

  $\rightarrow$ eventually performs better than the $\epsilon = 0.1$

10-armed testbed





- Note that $\epsilon = 0.01$ does better in the long run.

Softmax

- ϵ-greedy makes a random choice among non-optimal actions.
- Sometimes, it is good not to pick actions with poor outcomes.
- Softmax picks non-optimal actions based on their reward.
- Common to use the Gibbs distribution to pick the action.
- Choose action a on the r-th play with probability:

$$\frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_{b=1}^{n} e^{\frac{Q_t(b)}{\tau}}}$$

where τ is the temperature.

Softmax

- Choose action *a* on the *r*-th play with probability:

$$\frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_{b=1}^{n} e^{\frac{Q_t(b)}{\tau}}}$$

- When temperature is high, all actions are approximately equally likely.
- As temperature tends to 0, action selection tends to greedy selection.
- Can vary τ over time — high to start, low as the agent thinks it is converging.

More complex scenarios

- The bandit model makes a key simplifying assumption: the agent is always in the same state. So we only have to learn about one action.
- In general, agents can be in multiple states, and the best action varies with state.
- To handle this, we need a model of this kind of scenario.

How an agent might decide what to do

- Consider an agent with a set of possible actions A.
- Each $a \in A$ has a set of possible outcomes s_a.
- Each state s_a has a value associated to it called utility $u(s_a)$.
- Which action should the agent pick?

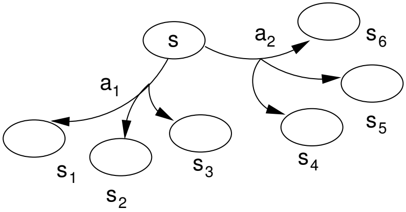
How an agent might decide what to do

- The action a^* which a rational agent should choose is that which maximises the agent's utility:

$$a^* = arg \max_{a \in A} u(s_a)$$

- The problem is that in any realistic situation, we do not know which s_a will result from a given a, so we do not know the utility of a given action.
- Instead we have to calculate the expected utility of each action and make the choice on the basis of that.

How an agent might decide what to do



- Given an action a_i with a set of possible outcomes s_{a_i}, the expected utility is the sum of the utility of each state in s_{a_i} weighted by the probability of getting there by doing a_i:

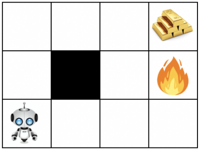
$$EU(a_i) = \sum_{s' \in s_{a_i}} u(s') \Pr(s_{a_i} = s')$$

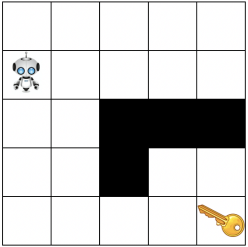
- Select a_i with the highest expected utility.

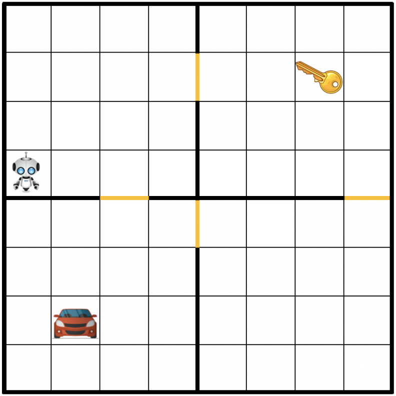
Sequential decision problems

- We have a method we can apply to individual decisions by agents. A bit more general than the *n*-armed bandit case. However, it is not enough.
- Agents are not usually in the business of taking single decisions. The best overall result is not necessarily obtained by a greedy approach to a series of decisions.
- The current best option is not the best thing in the long-run.

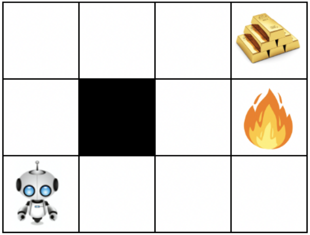
Some grid examples

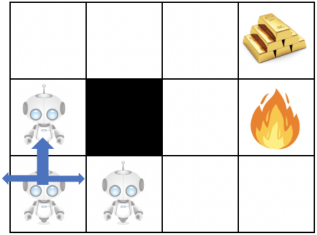


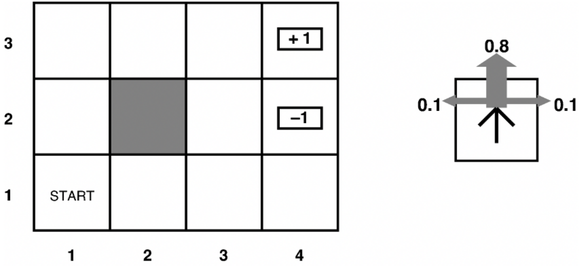




Grid example

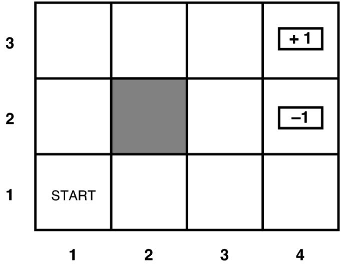




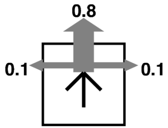


(a) (b)

Grid example



(a)



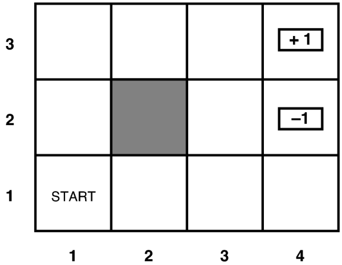
(b)

- The agent has to pick a sequence of actions

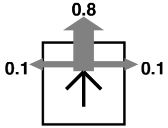
$$A(s) = \{Up, Down, Left, Right\}$$

  to get from START to one of the terminal states.
- They are called terminal states because the agent stops when it gets there. The terminal states are in the top right with values $+1$ or -1.

Grid example



(a)



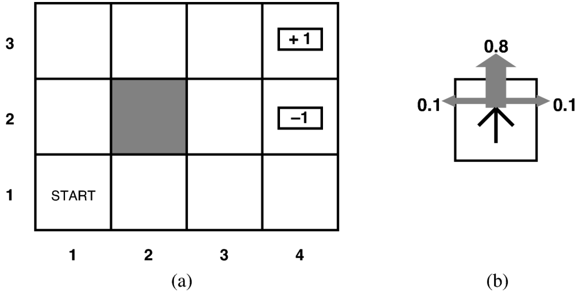
(b)

- If the world was deterministic, the choice of actions would be easy: the optimal action in each state

$$Up, Up, Right, Right, Right$$

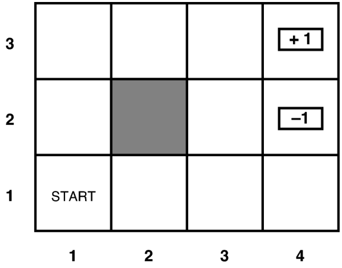
- But actions are stochastic. There is a probability distribution that tells you the probability of each new state.

Grid example

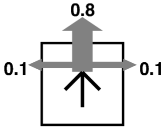


- 80% of the time the agent moves as intended, but 20% of the time the agent moves perpendicular to the intended direction: half the time to the left, half the time to the right.
- The agent does not move if it hits a wall.

Grid example



(a)



(b)

- Thus *Up*, *Up*, *Right*, *Right*, *Right* succeeds with probability:

$$0.8^5 = 0.32768$$

- Also a small chance of going around the obstacle the other way.

Grid example

- We can write a transition model to describe these actions. A transition model tells the agent the new state given a current state and an action.
- Since the actions are stochastic, the model looks like:

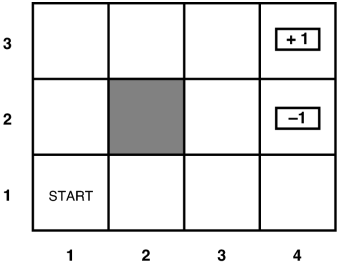
$$P(s'|s,a)$$

  where a is the action that takes the agent from s to s'.
- Transitions are assumed to be Markovian. They only depend on the current and next states.
- We could write a large set of probability tables that would describe all the possible actions executed in all the possible states. This would completely specify the actions.

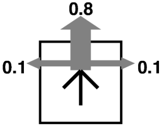
Grid example

- The full description of the problem also has to include the utility function.
- The utility function is defined over sequences of states (an environment history) rather than on a single state.
- We will assume that in each state s the agent receives a reward. This may be positive or negative.

Grid example



(a)



(b)

- The reward for non-terminal states is -0.04.
- We will assume that the utility of a run is the sum of the utilities of states, so -0.04 is an incentive to take fewer steps to get to the terminal state.

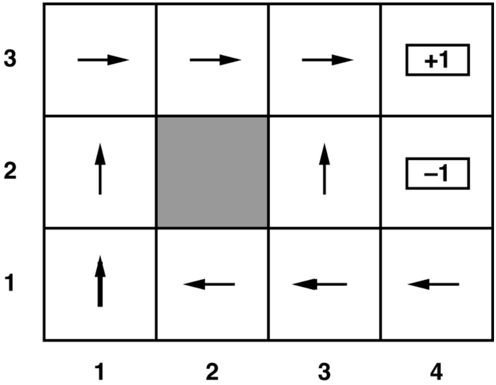
Markov decision process

- The overall problem the agent faces here is a Markov decision process (MDP).
- Mathematically we have
  - a set of states $s \in S$ with an initial state s_0
  - a set of actions $A(s)$ in each state
  - a transition model $P(s' | s, a)$
  - a reward function $R(s)$.
- Captures any fully observable non-deterministic environment with a Markovian transition model and additive rewards.
- What does a solution to an MDP look like?

Markov decision process

- A solution is a policy, which we write as π. It represents a mapping from perceived states of the environment to actions to be taken when in those states.
- A choice of action for every state. That way, if we get off track, we still know what to do.
- In any state s, $\pi(s)$ identifies what action to take.

Markov decision process

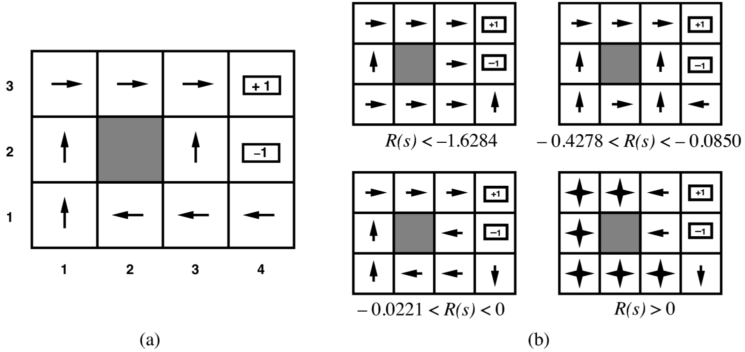


- This is a policy.
- The arrows are shorthand for *Up*, *Right*, *Left*, *Down*.

Markov decision process

- We would prefer not just any policy but the optimal policy.
  - But how to find it?
- Need to compare policies by the reward they generate.
- Since actions are stochastic, policies will not give the same reward every time.
  - So compare the expected value.
- The optimal policy π^* is the policy with the highest expected value.
- At every stage the agent should do $\pi^*(s)$.

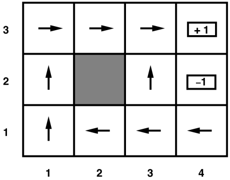
Grid example



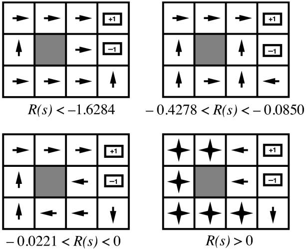
(a) Optimal policy for the original problem.

(b) Optimal policies for different values of $R(s)$.

Grid example



(a)



(b)

- $R(s) < -1.6284$: The agent heads for exit (even for -1).
- $-0.4278 < R(s) < -0.0850$: The agent heads for the $+1$ state and is prepared to risk falling into the -1 state.
- $-0.0221 < R(s) < 0$: The agent does not take any risks.
- $R(s) > 0$: The agent does not want to leave.

Optimal policies

- How do we get the optimal policy?
- There are several ways.
- The only way we will consider here is by calculating the utility for each state.

Optimal policies

3	0.812	0.868	0.918	$\boxed{+1}$
2	0.762		0.660	$\boxed{-1}$
1	0.705	0.655	0.611	0.388
	1	2	3	4

- Here we have the values of states if the agent executes an optimal policy.
- We write these values for every state s as:

$$U^{\pi^*}(s)$$

Optimal policies

- If we have these values, the agent has a simple decision process.
- It just picks the action *a* that maximises the expected utility of the next state:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a) U^{\pi^*}(s')$$

- Only have to consider the next step.

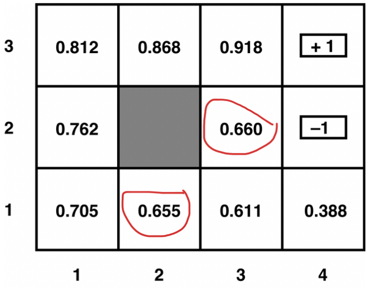
Optimal policies

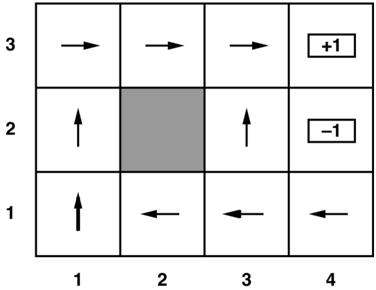
3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

3	→	→	→	+1
2	↑		↑	–1
1	↑	←	←	←
	1	2	3	4

- Here we have the utilities under the optimal policy (left) and the corresponding policy (right).

Optimal policies





- This is not the same as picking the action that takes you to the state with the highest value!
- Calculate: $EU(Up_{(3,1)})$, $EU(Left_{(3,1)})$, $EU(Down_{(3,1)})$, $EU(Right_{(3,1)})$. You should obtain that the action with greatest expected utility in $(3, 1)$ is *Left*.

Optimal policies

- The big question is how to compute $U^{\pi^*}(s)$.
- For this module we just have to know the method to calculate $U^{\pi^*}(s)$.
- To compute these values, we use value iteration.

Value iteration

- Execute the following procedure:
  1. Assign an arbitrary utility $U_0(s)$ to every state.
  2. For every state, carry out the following update:

     $$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

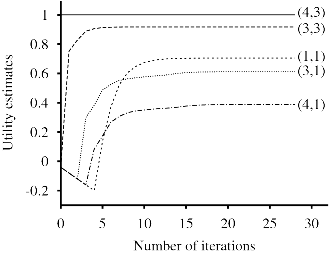
     where $0 < \gamma \leqslant 1$ is the discount rate.
     This update computes the utility of each state by doing a maximum expected utility calculation on the current utility of the states around it.
  3. Continue updating until the utilities of states do not change.
- Note that in computing $U_{i+1}(s)$ for each state we use the $U_i(s')$ for the states around it.

Value iteration

- After an infinite number of applications, the utilities are guaranteed to converge on the optimal values.
- In practice we need $n \ll \infty$ updates.

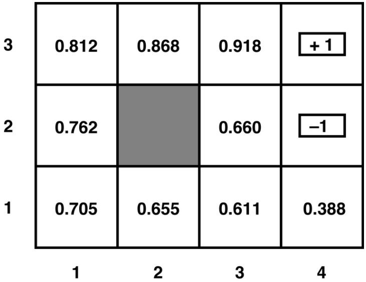
Value iteration

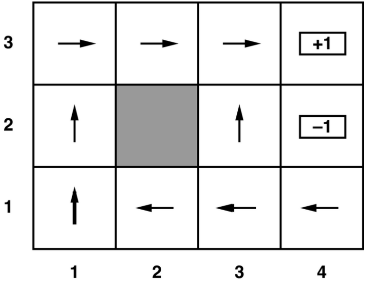
	1	2	3	4
3	0.812	0.868	0.918	$\boxed{+1}$
2	0.762		0.660	$\boxed{-1}$
1	0.705	0.655	0.611	0.388



- How the values of states change as updates occur.
- $U_0(s)$ was set to 0 for all states.
- States at different distances from (4,3) accumulate negative reward until a path is found to (4,3); then the utilities start to increase.
- $U(4,3)$ is pinned to 1.
- $U(3,3)$ quickly settles to a value close to 1.
- $U(1,1)$ becomes negative and then grows as positive utility as the goal feeds back to it.

Back to reinforcement learning





- We can solve this as an MDP.
- But what about learning?

Back to reinforcement learning

- What if the agent does not know the transition model:

$$P(s'|s, a)$$

  and it does not know the reward function

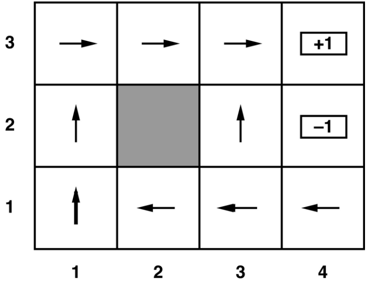
$$R(s)$$

- How can it decide what to do?
- Needs to learn the transition model and reward.

Passive learning

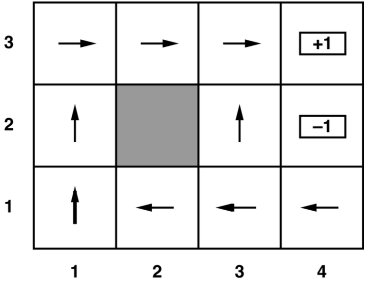
- In passive reinforcement learning the agent's policy is fixed.
- Agent learns utility $U^{\pi}(s)$ by carrying out runs through the environment, following some policy π.
- In state s, it always executes the action $\pi(s)$.

Passive learning



- Agent does not make a choice about how to act.
- That is, it does not choose how to act *while learning*.

Passive learning



- A *run* is a sequence of states and actions that continues until the agent reaches the terminal state:
$(1,1)_{-0.04} \xrightarrow{Up} (1,2)_{-0.04} \xrightarrow{Up} (1,3)_{-0.04} \xrightarrow{Right} (1,2)_{-0.04} \xrightarrow{Up}$
$(1,3)_{-0.04} \xrightarrow{Right} (2,3)_{-0.04} \xrightarrow{Right} (3,3)_{-0.04} \xrightarrow{Right} (4,3)_{+1}$
- Note that we have reward as well.

Direct utility estimation

- The utility of a state is a function of the rewards of all the states that are visited later.
- We can estimate the utility of a state by the rewards generated along the run from that state.
- Direct utility estimation.
- Each run gives us one or more samples for the reward from a state.

Direct utility estimation

- Given the run:

$$(1,1)_{-0.04} \stackrel{Up}{\rightarrow} (1,2)_{-0.04} \stackrel{Up}{\rightarrow} (1,3)_{-0.04} \stackrel{Right}{\rightarrow} (1,2)_{-0.04} \stackrel{Up}{\rightarrow}$$
$$(1,3)_{-0.04} \stackrel{Right}{\rightarrow} (2,3)_{-0.04} \stackrel{Right}{\rightarrow} (3,3)_{-0.04} \stackrel{Right}{\rightarrow} (4,3)_{+1}$$

  a sample reward for $(1,1)$ from the run above is the sum of the rewards all the way to the terminal state.
    - 0.72 in this case.
- The same run will produce two samples for $(1,2)$ and $(1,3)$.
    - 0.76 and 0.84
    - 0.80 and 0.88
- (Here we set the discount to 1).

Passive learning

- As the agent moves, it can calculate a sample estimate of $P(s'|s, \pi(s))$
- Each time it moves it creates a new sample for one state.
- Given:

$$(1,1)_{-0.04} \stackrel{Up}{\rightarrow} (1,2)_{-0.04} \stackrel{Up}{\rightarrow} (1,3)_{-0.04} \stackrel{Right}{\rightarrow} (1,2)_{-0.04} \stackrel{Up}{\rightarrow}$$
$$(1,3)_{-0.04} \stackrel{Right}{\rightarrow} (2,3)_{-0.04} \stackrel{Right}{\rightarrow} (3,3)_{-0.04} \stackrel{Right}{\rightarrow} (4,3)_{+1}$$

  we get:

$$\begin{aligned}
P((1,2)|(1,1), Up) &= 1 \\
P((1,2)|(1,3), Right) &= 0.5 \\
P((2,3)|(1,3), Right) &= 0.5 \\
&\vdots
\end{aligned}$$

Passive learning

- Over time, the agent builds up estimates of:

3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

and $P(s'|s, \pi(s))$, for every s, s' for the given $\pi(s)$.

Passive reinforcement learning: solution

- A list of states $s_1, \ldots s_n$.
- Each state has a utility estimate associated with it: $U(s)$.
- Each state has a set of actions associated with it: $a_1, \ldots a_m$.
- Each state/action pair has a probability distribution:

$$P(S'|s, \pi(s))$$

  over the states S' that it gets to from s by doing $\pi(s)$.
- May not encounter every state.

Passive learning

- How does an agent decide what to do?
- The agent just computes each step using one-step lookahead on the expected value of actions.
- Picks the action a with the greatest expected utility.
- Its data on actions will be limited because it has only been trying $\pi(s)$.

Passive learning

- Has to vary π if it wants to learn the full space.
- But is this worth it?
- After all, once we have an idea of how to act to get to the goal, is more learning justified?

Summary

- We started on reinforcement learning.
- n-armed bandits.
- Markov decision processes.
- Passive reinforcement learning.