Machine Learning

Reinforcement Learning (1)

Dr. Harry Goldingay
goldihj1@aston.ac.uk

Learning Outcomes

- At the end of this lecture you should:
  - Know what a sequential decision making problem is and how to formulate it as a Markov Decision Process (MDP).
  - Understand solution policies for MDPs and know a simple algorithm for generating an optimal policy given sufficient information.
  - Understand the characteristics of problems addressed by reinforcement learning and know an algorithm for policy evaluation on these problems well enough to implement it.

Motivation

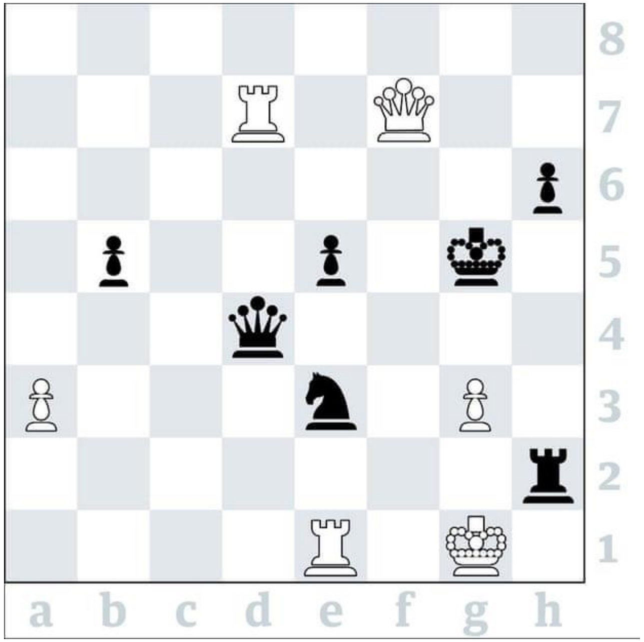
Game Playing

- Some of the biggest machine learning media stories in recent years concern “game playing”.

- How would you apply machine learning to playing chess?

Learning Chess

- Firstly, what do we need to learn?



Rh1+

Learning Chess using Supervised Learning

- We want to learn a mapping from **chess positions** to **best moves**.
  - Chess positions could be seen as a **feature vector**.
  - Single, categorical output needed.
- Should remind you of **classification**.
  - We could apply supervised learning…
  - …to a database of games from top players…
  - …to try to predict how they would move in the position.
- Issues:
  - How can we improve on top players by copying them?
  - Do we have enough (of the right) data?

Aston University
Birmingham

Sequential Decision Making

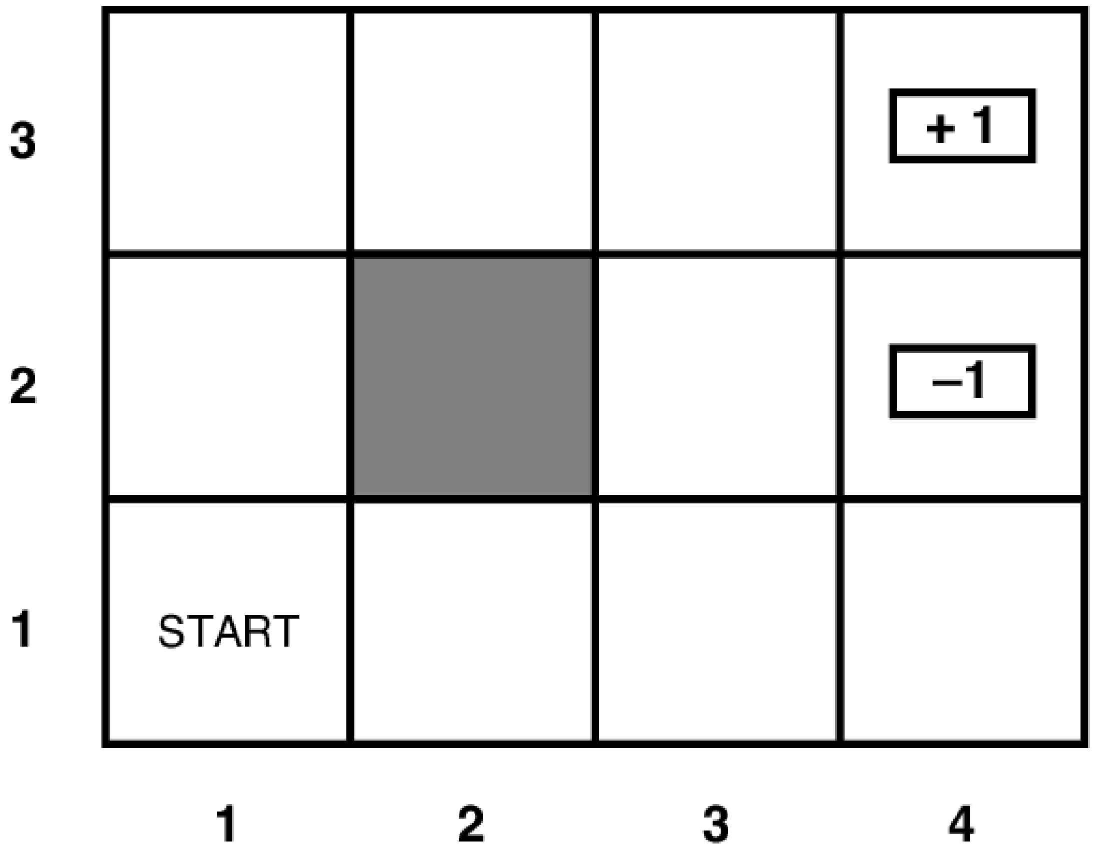
Sequential Decision Making

- A more useful way to think of this problem is as one of **sequential decision making**:
  - an agent takes an **action**,
  - it **observes** the effect of the action…
  - …and then chooses a subsequent action.

- This generates a sequence of actions and effects.

- The aim is to take actions which lead to the “best” sequence of effects.

Grid World



“Best” Sequence

- On each square in grid world, the agent will receive a **reward**:
  - +1 on square (4,3)
  - -1 on square (4,2)
  - -0.04 on any other square.
- It the agent reaches either of squares (4,3) or (4,2) (**terminal states**), the game ends.
- The best sequence will be the one which maximises **utility**: the sum of rewards.
  - Note: there are other ways of defining utility.

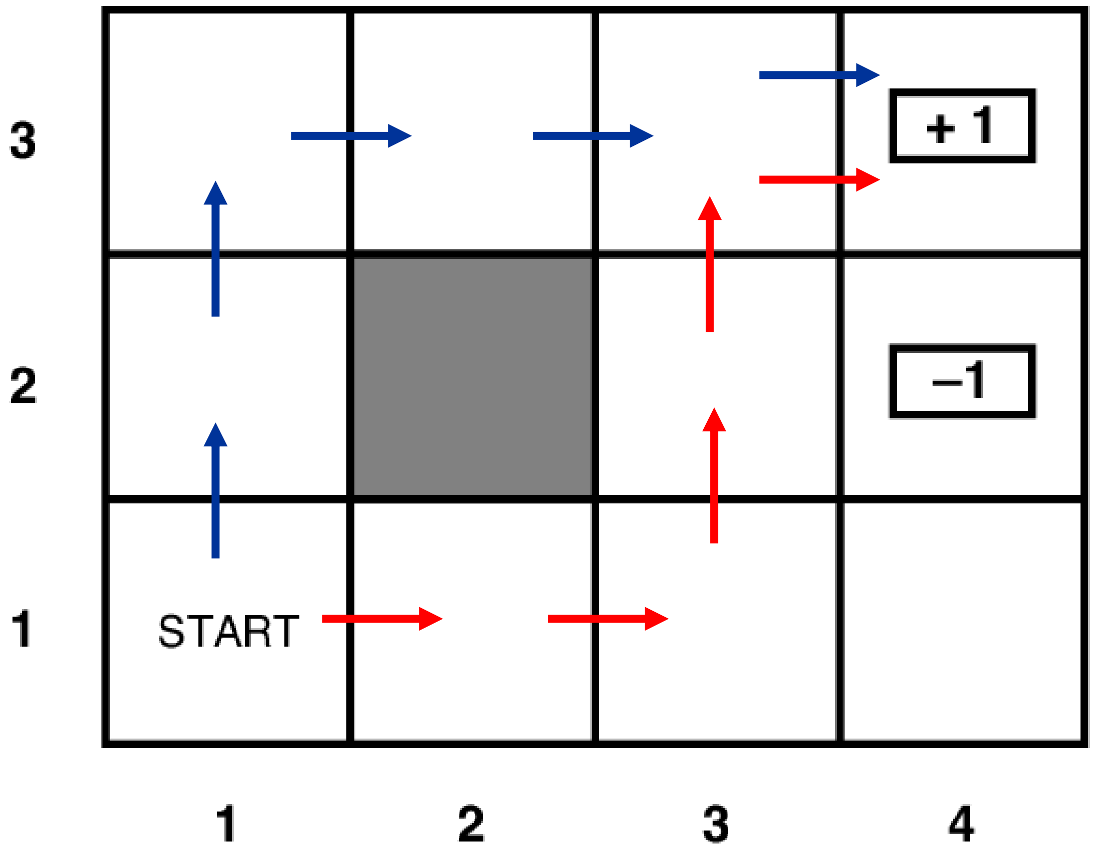
Solving Grid World

- A “solution” is just a sequence of actions.

- The **rewards** an agent receives in grid world…
  - +1 or -1 for a transition to an exit (terminal state),
  - -0.04 for a transition to any other square,
- …mean that it should find its way to the terminal at (4,3) as quickly as possible.

- The optimal solution is the sequence which achieves the above.

Grid World: Optimal Solution(s)



Aston University
Birmingham

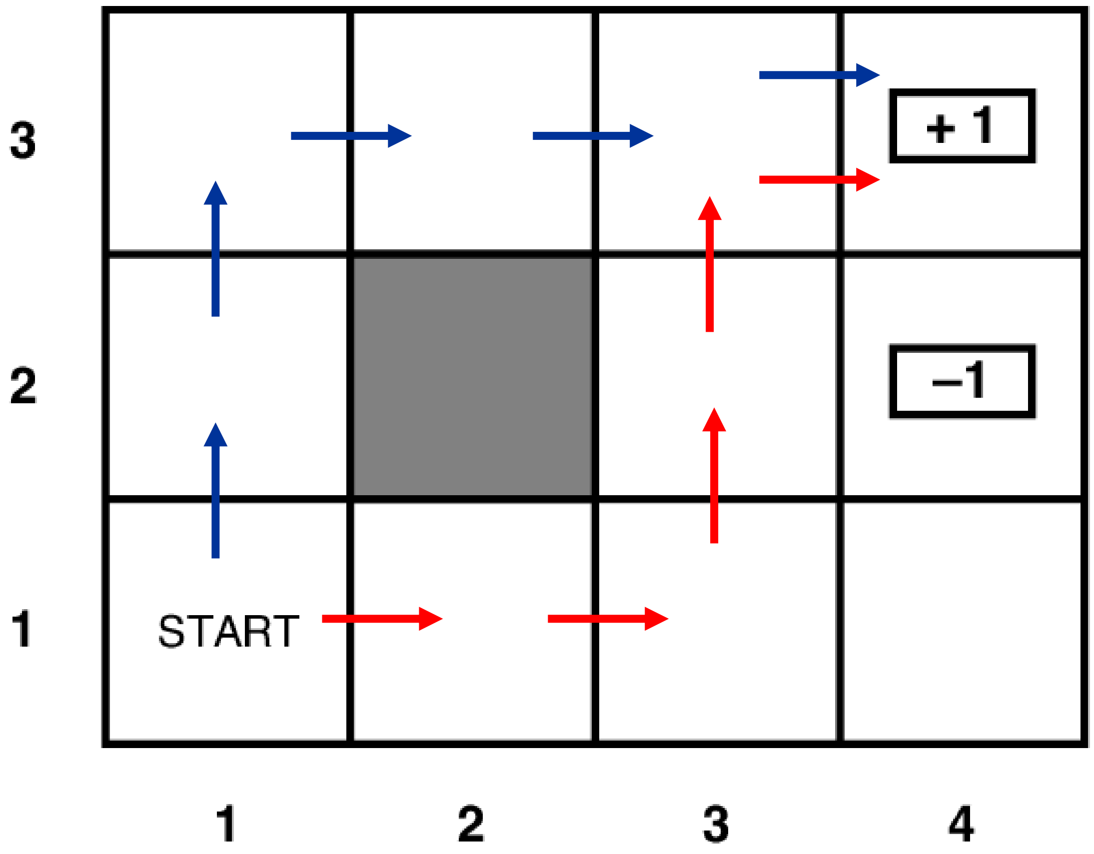
Decision Making with Uncertainty

- That was easy but, unfortunately, unrealistic.
  - Many interesting decision problems involve uncertainty.
  - Our actions effect outcomes, NOT determine them.

- We can incorporate uncertainty into our problem, by making transitions probabilistic:
  - Make the intended move with probability 0.8,
  - Make a perpendicular move with probability 0.2 (each direction equally likely),
  - If the agent hits a wall, it stays where it is.

- How does this affect our solutions?

Stochastic Grid World: Optimal Solutions?



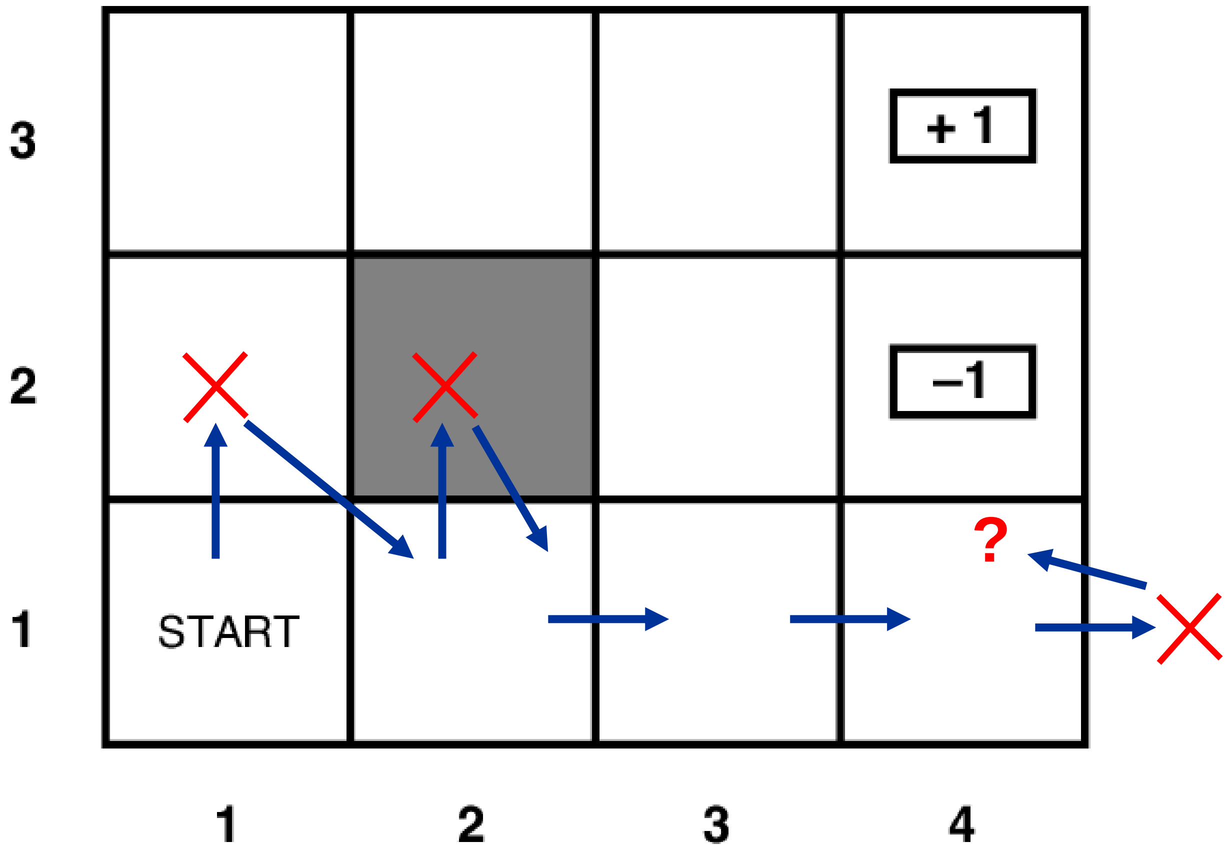
Aston University
Birmingham

Policies

- In a stochastic setting, we prefer the path **Up, Up, Right, Right, Right** to the other optimal deterministic path.

- Is this a good solution to the problem?
  - No!
  - What if one of the steps fails (e.g. we go Right instead of Up) at step 1?

Stochastic Grid World: Optimal Solutions?



Aston University
Birmingham

Policies

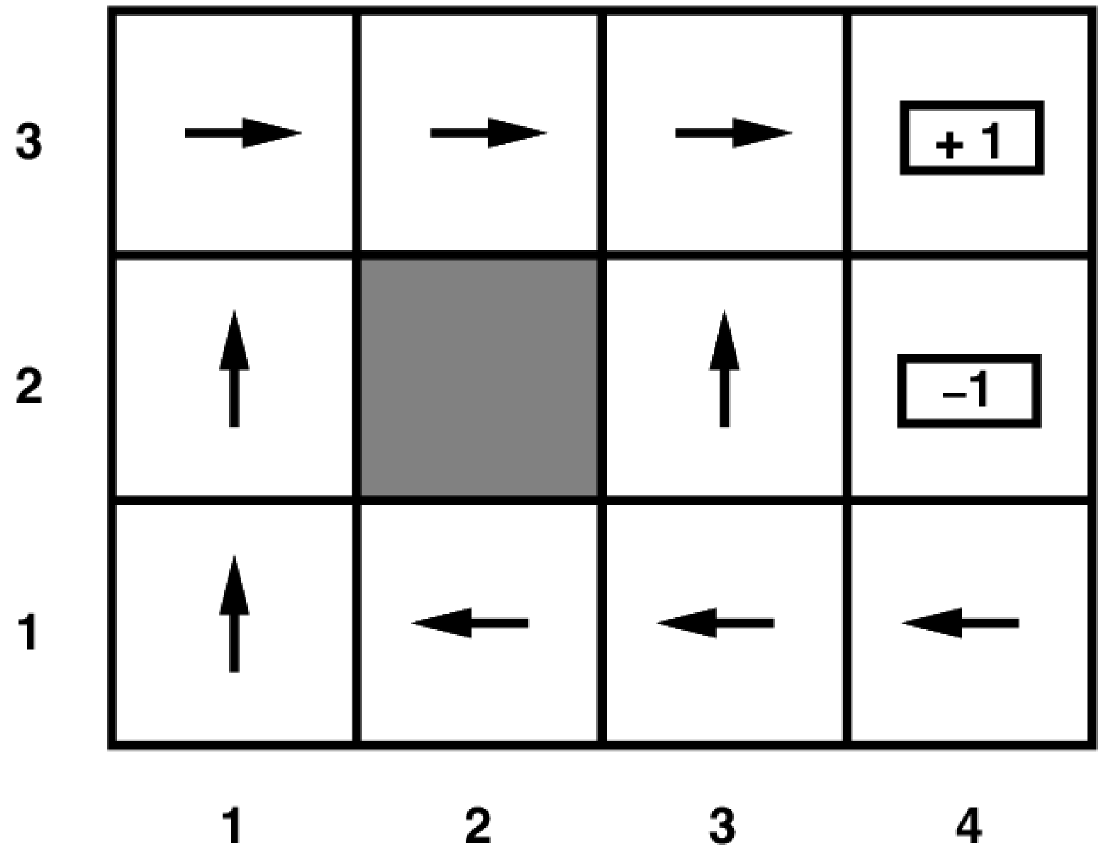
- In a stochastic setting, we prefer the path **Up, Up, Right, Right, Right** to the other optimal deterministic path.

- Is this a good solution to the problem?
  - No!
  - What if one of the steps fails (e.g. we go Right instead of Up) at step 1?

- A sequence of actions is not a good solution if we don't know the results of the actions.
  - Better to view each location as a **state**…
  - …and formulate a solution by choosing an action per state.
  - Such a solution is called a **policy**.

Aston University
Birmingham

Stochastic Grid World: Optimal Policy

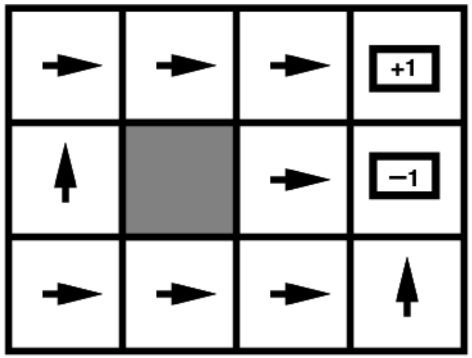


Markov Decision Processes

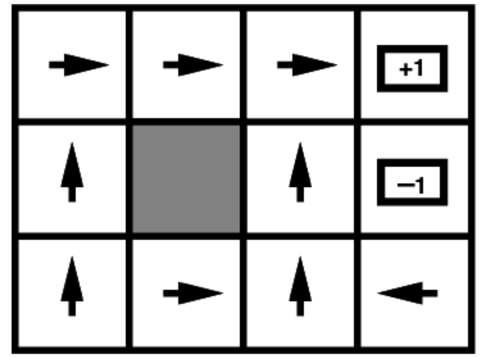
- Sequential decision problems are often formalised as **Markov Decision Processes (MDPs)** which require:
  - A set of **states**, S, including a start state, s_0
  - For each state, a set of **actions**, A_s
  - A **transition model**, $P(s'|s,a)$
    - Gives the probability of reaching state s' from state s if taking action a.
  - A **reward function**, $R(s)$
    - Sometimes also dependent on action and outcome: $R(s,a,s')$

- The goal is to find an optimal policy, π^*, which maximises **expected utility**.

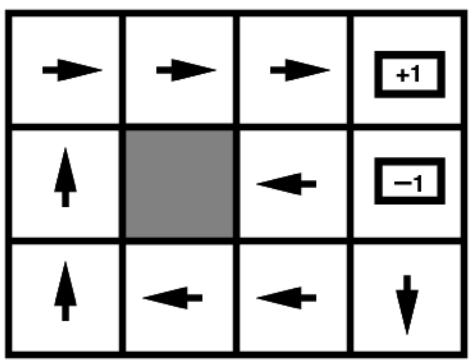
Rewards and Optimal Policy



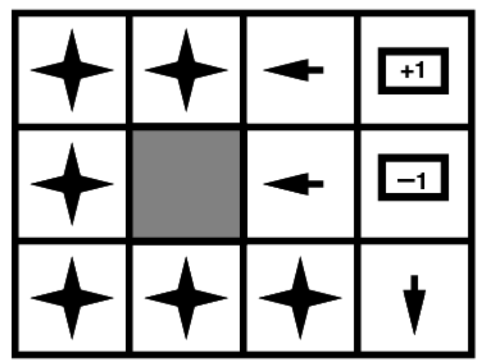
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$

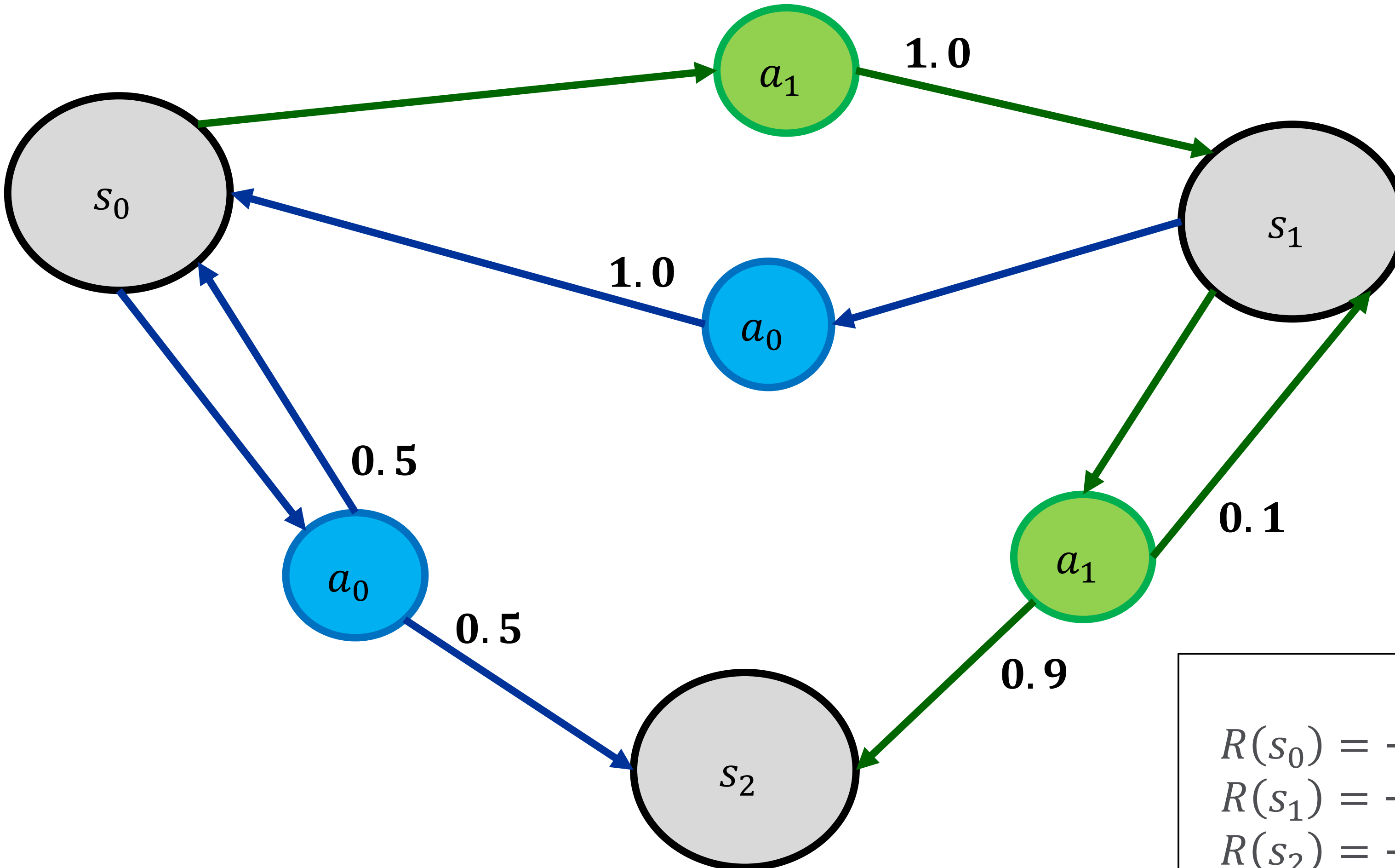


$R(s) > 0$

MDPs: Expected Utility

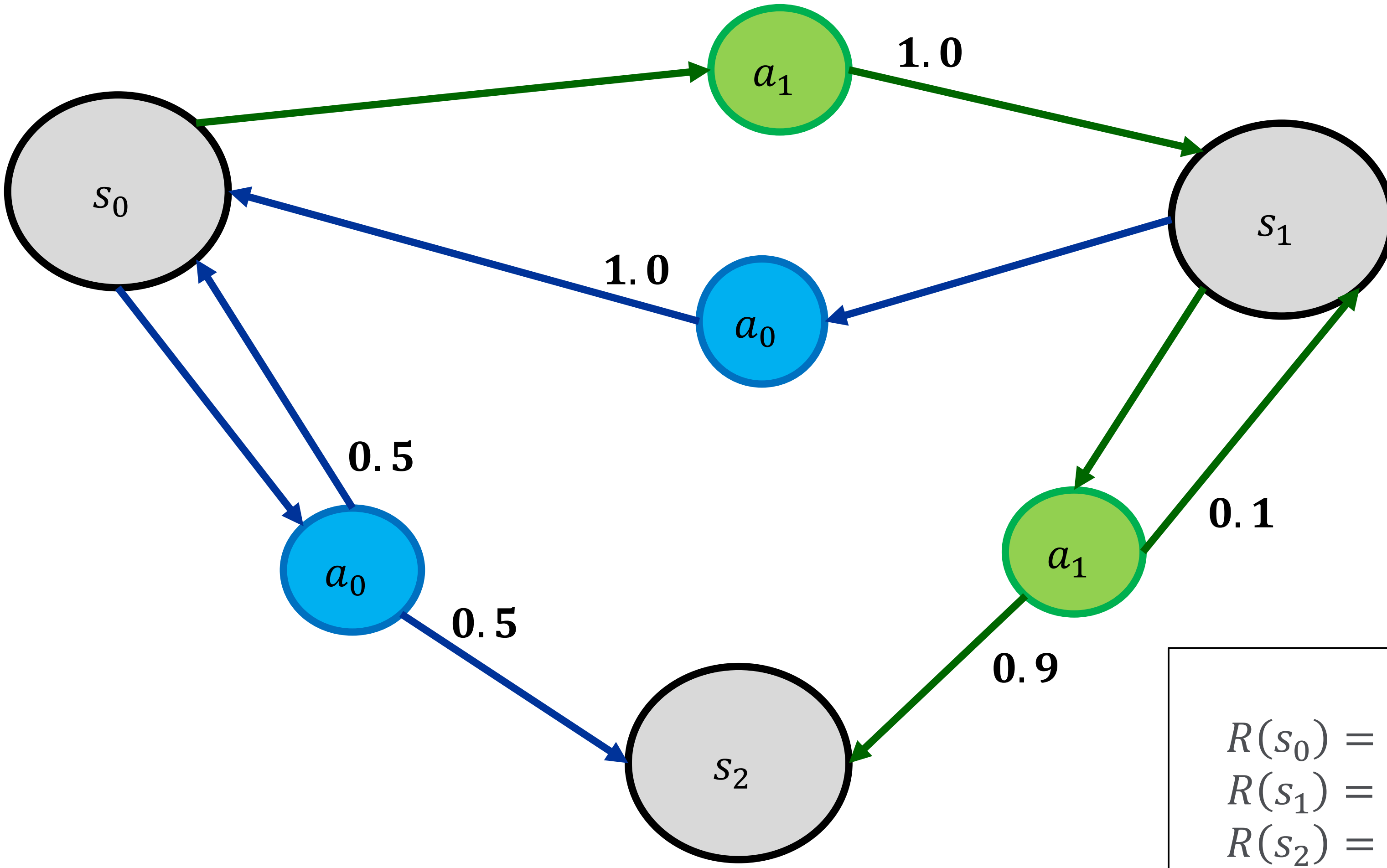
Aston University
Birmingham

Playing an MDP (1)



$R(s_0) = -0.50$
$R(s_1) = -0.75$
$R(s_2) = -0.10$

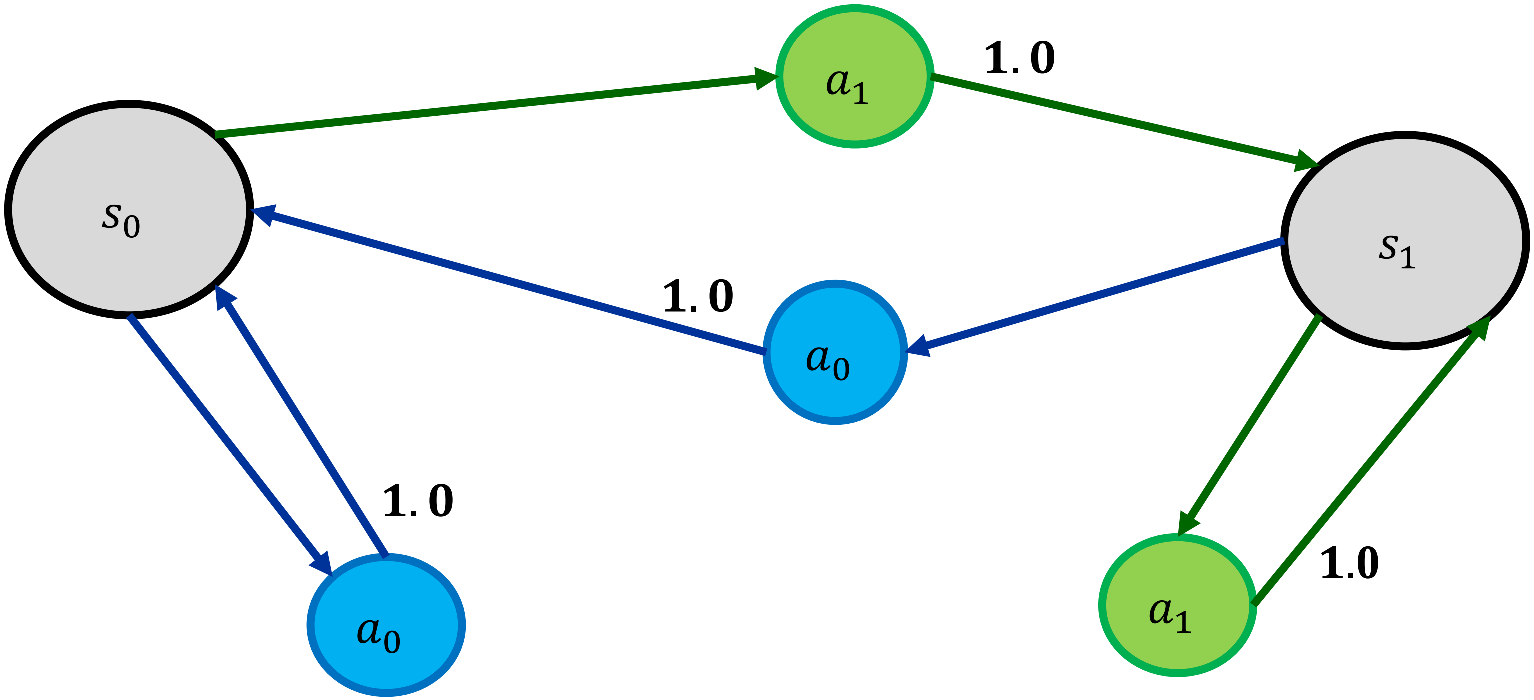
Playing an MDP (2)



$R(s_0) = 0.50$
$R(s_1) = 0.25$
$R(s_2) = 0.90$

Aston University
Birmingham

Playing an MDP (3)



$R(s_0) = -0.50$
$R(s_1) = -0.75$

Utilities Over Time

- So far, we have been assigning utility on the basis of **additive rewards**:

$$U([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$$

- It is common to use discounted rewards:

$$U([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

- Where $0 < \gamma < 1$ is called a **discount factor**. Advantages:
  - Utility converges even for an infinite sequence of actions,
  - Seems to be a good model for human preferences.

Utilities of States

- We can define the utility of a state under policy π as:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

- where $S_0 = s$.

- The optimal policy for a given state is:

$$\pi_s^* = \underset{\pi}{\operatorname{argmax}}\, U^{\pi}(S)$$

- We can show that, for any state, $\pi_s^* = \pi^*$. Therefore, we can denote the (expected) utility of a state as:

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

Grid World: Utilities of States

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

For $\gamma = 1$

Aston University
Birmingham

Policies and Utilities of States

- If we know $U(s)$ for each state, we can use it to infer the optimal policy.
  - Choose the action which results in the highest expected utility of the next state.
- What action should an agent take in state (3,1) in the previous example?

Grid World: Utilities of States

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

For $\gamma = 1$

Policies and Utilities of States

- If we know $U(s)$ for each state, we can use it to infer the optimal policy.
  - Choose the action which results in the highest expected utility of the next state.

- What action should an agent take in state (3,1) in the previous example?
  - Up: $0.8 \times 0.66 + 0.1 \times (0.655 + 0.388) \approx 0.63$
  - Left: $0.8 \times 0.655 + 0.1 \times (0.66 + 0.611) \approx 0.65$
- The agent should choose Left.

- **Key takeaway**: π^* can be inferred directly from utility.

Calculating Expected Utility

- Does this give us a solution methodology for MDPs?

- Not yet…
  - We can calculate π^* from $U(s)$
  - …but recall that $U(s) = U^{\pi^*}(s)$

- …but there iterative are algorithms for doing so:
  - Value iteration
  - Policy iteration

Value Iteration

- A consequence of our definition of utility is that the utility of states can be expressed as follows (**Bellman equation**):

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$$

- This gives us a system of equations to solve (one for each state) but the $\max$ component makes this problematic.

- Try an iterative approach.

Value Iteration

- We can turn the Bellman equations into update equations:

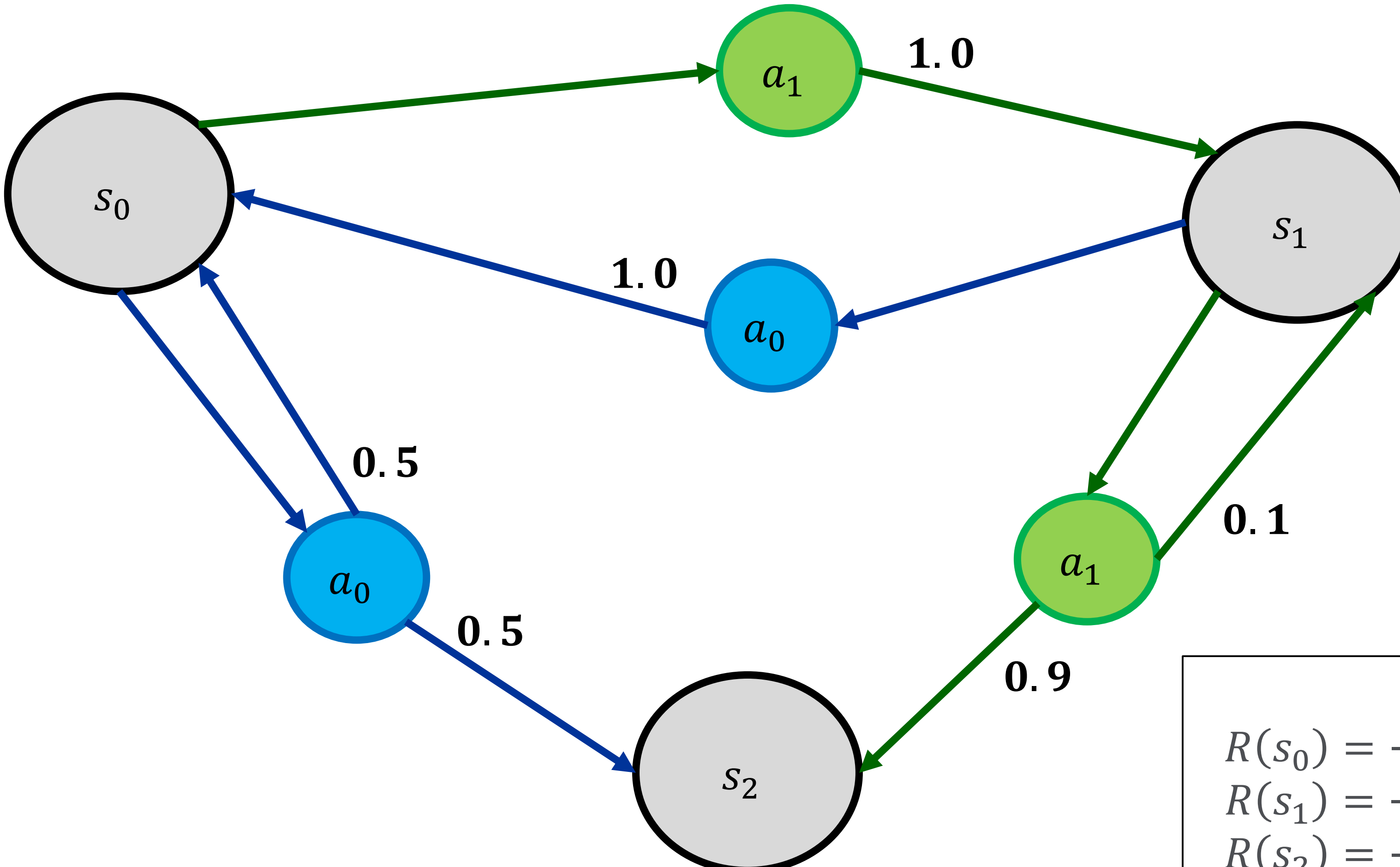
$$U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

- If $\gamma < 1$, then this is guaranteed to converge to the (unique) solution to the Bellman equations.

- In practice:
  - Pick arbitrary initial values for the $U_0(s)$
  - Run until the difference between estimates at different timesteps become small

- This is called **value iteration**.

Value Iteration Illustrated



$$R(s_0) = -0.50$$
$$R(s_1) = -0.75$$
$$R(s_2) = -0.10$$

Value Iteration Illustrated

Transition Model

$$P(s_0|s_0, a_0) = 0.5$$
$$P(s_2|s_0, a_0) = 0.5$$
$$P(s_1|s_0, a_1) = 1.0$$
$$P(s_0|s_1, a_0) = 1.0$$
$$P(s_1|s_1, a_1) = 0.1$$
$$P(s_2|s_1, a_1) = 0.9$$

Reward Function

$$R(s_0) = -0.50$$
$$R(s_1) = -0.75$$
$$R(s_2) = -0.10$$

Utility Estimates ($i = 0$)

$$U_i(s_0) = 0$$
$$U_i(s_1) = 0$$
$$U_i(s_2) = 0$$

Update

Initial choice of action unimportant. All states have estimated utility 0.

$$U_{i+1}(s_0) = R(s_0) + 0 = -0.50$$
$$U_{i+1}(s_1) = R(s_1) + 0 = -0.75$$
$$U_{i+1}(s_2) = R(s_2) + 0 = -0.10$$

Value Iteration Illustrated

Transition Model

$$P(s_0|s_0,a_0) = 0.5$$
$$P(s_2|s_0,a_0) = 0.5$$
$$P(s_1|s_0,a_1) = 1.0$$
$$P(s_0|s_1,a_0) = 1.0$$
$$P(s_1|s_1,a_1) = 0.1$$
$$P(s_2|s_1,a_1) = 0.9$$

Reward Function

$$R(s_0) = -0.50$$
$$R(s_1) = -0.75$$
$$R(s_2) = -0.10$$

Utility Estimates ($i = 1$)

$$U_i(s_0) = -0.50$$
$$U_i(s_1) = -0.75$$
$$U_i(s_2) = -0.10$$

Update Example

Expected value of next state from s_0 given a.
a_0: $0.5 \times -0.5 + 0.5 \times -0.1 = -0.3$
a_1: $1.0 \times -0.75 = -0.75$

Best action: a_0.

$$U_{i+1}(s_0) = R(s_0) - 0.3 = -0.8$$

Value Iteration Illustrated

Transition Model

$P(s_0|s_0, a_0) = 0.5$
$P(s_2|s_0, a_0) = 0.5$
$P(s_1|s_0, a_1) = 1.0$
$P(s_0|s_1, a_0) = 1.0$
$P(s_1|s_1, a_1) = 0.1$
$P(s_2|s_1, a_1) = 0.9$

Reward Function

$R(s_0) = -0.50$
$R(s_1) = -0.75$
$R(s_2) = -0.10$

Utility Estimates ($i = 2$)

$U_i(s_0) = -0.8$
$U_i(s_1) = -0.915$
$U_i(s_2) = -0.10$

Update Example

Expected value of next state from s_0 given a.
a_0: $0.5 \times -0.8 + 0.5 \times -0.1 = -0.45$
a_1: $1.0 \times -0.915 = -0.915$

Best action: a_0.
$U_{i+1}(s_0) = R(s_0) - 0.45 = -0.95$

Aston University
Birmingham

Value Iteration Illustrated

Transition Model

$$P(s_0|s_0, a_0) = 0.5$$
$$P(s_2|s_0, a_0) = 0.5$$
$$P(s_1|s_0, a_1) = 1.0$$
$$P(s_0|s_1, a_0) = 1.0$$
$$P(s_1|s_1, a_1) = 0.1$$
$$P(s_2|s_1, a_1) = 0.9$$

Reward Function

$$R(s_0) = -0.50$$
$$R(s_1) = -0.75$$
$$R(s_2) = -0.10$$

Utility Estimates ($i = 3$)

$$U_i(s_0) \approx -0.95$$
$$U_i(s_1) \approx -0.93$$
$$U_i(s_2) \approx -0.10$$

Utility Estimates ($i = 4$)

$$U_i(s_0) \approx -1.03$$
$$U_i(s_1) \approx -0.93$$
$$U_i(s_2) \approx -0.10$$

Utility Estimates ($i = 5$)

$$U_i(s_0) \approx -1.06$$
$$U_i(s_1) \approx -0.93$$
$$U_i(s_2) \approx -0.10$$

Change < 0.05

Introduction to Reinforcement Learning

Aston University
Birmingham

Planning and Reinforcement Learning

- So far, the problems we have seen can be solved through **offline planning**:
  - Consider the information available…
  - …and infer the optimal policy.
  - No need to take any actions!
- A reinforcement learning problem is an MDP in which we don't know:
  - the **transition model**,
  - The **reward function**.
- Not enough information to solve offline. Need to experiment!

Naïve Approach

- We know how to solve MDPs when we do know the transition model and reward function.
  - Try to learn them from the environment!
  - Pick policy/policies.
  - Count state transitions that we observe and average to infer transition probabilities.
  - Observe rewards directly.
- Assume that our model is correct, and solve using value iteration.

- How does this scale with the number of actions/states?

Types of Reinforcement Learning

- Solving an MDP without prior knowledge of transitions and rewards is challenging.

- We will leave it to one side for now and look at approaches to **policy evaluation**.
  - Given a policy, what are the expected utilities of the states?

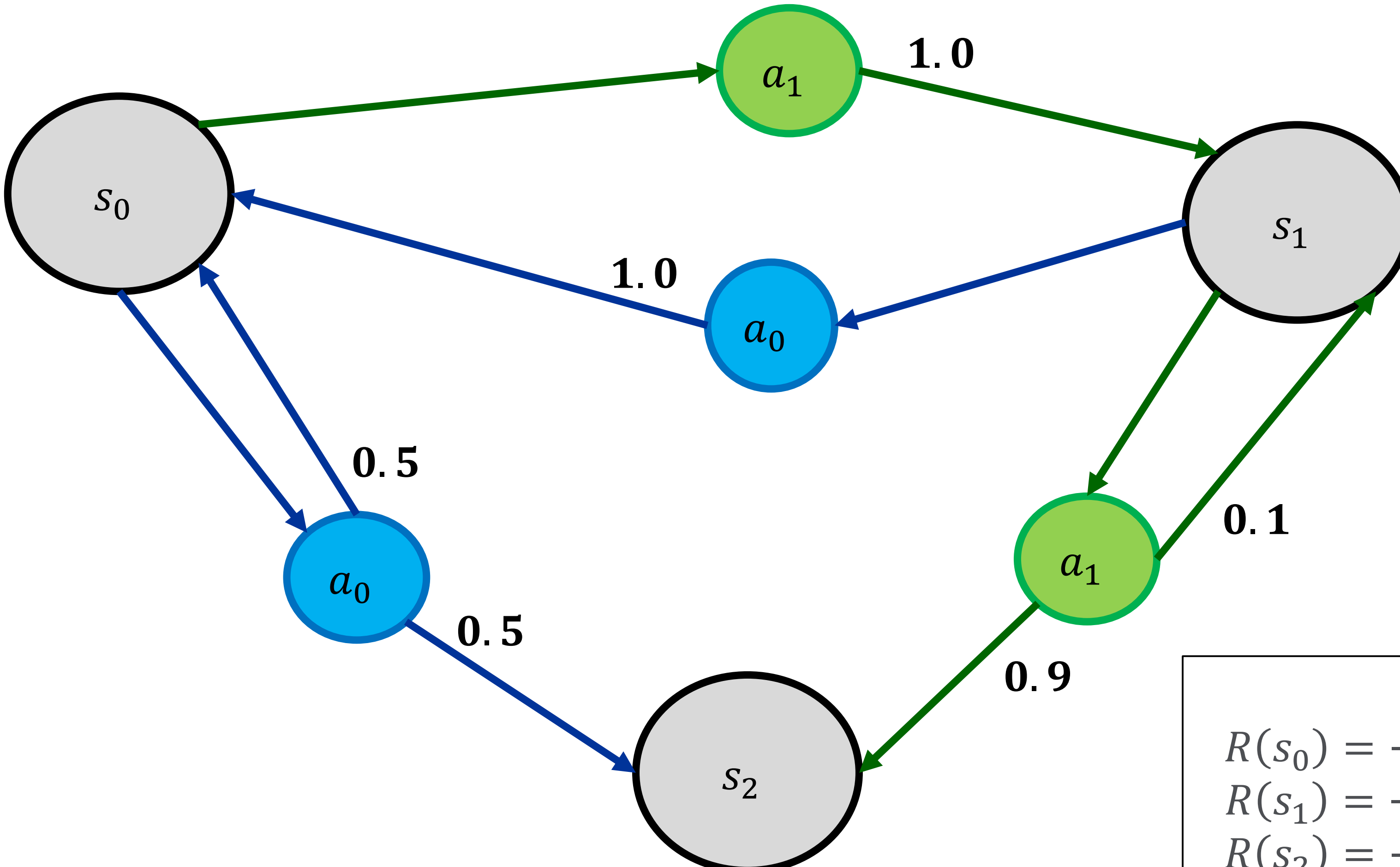
- This type of approach, with a fixed policy, is called **passive learning**.

- This is in contrast to **active learning**, in which an agent must also decide which actions to take.

Direct Evaluation

- We can take a similar approach to policy evaluation to our initial suggestion for solving RL problems:
- Repeatedly run trials of a policy from random starting states.
- Whenever we encounter a state, record the utility which follows in the trial.
- Average these records, to estimate the utility of the state under the policy.

Direct Evaluation Illustrated



$R(s_0) = -0.50$
$R(s_1) = -0.75$
$R(s_2) = -0.10$

Aston University
Birmingham

Direct Evaluation Illustrated

- Running four random trials with the optimal policy gives me:

- $s_2(-0.1)$
- $s_0(-0.5) \to s_2(-0.1)$
- $s_1(-0.75) \to s_2(-0.1)$
- $s_0(-0.5) \to s_0(-0.5) \to s_0(-0.5) \to s_2(-0.1)$

- We have seen s_0 four times, with an average utility of -0.975:
$$\frac{-0.6 - 1.6 - 1.1 - 0.6}{4}$$
- We have seen s_1 once, with a total utility of -0.85
- We have seen s_2 four times, with an average utility of -0.1

Aston University
Birmingham

Direct Evaluation

- Direct evaluation will eventually converge to the correct utility values.
- We don't need to learn the full transition model and reward function
  - don't have the same problem with number of state-action pairs as we saw before.
- However, the Bellman equations give us relationships between states.
  - Direct evaluation doesn't make use of these relationships…
  - …so is very inefficient.

Temporal Difference Learning

Temporal Difference Learning

- Rather than trying to build a model of transitions directly, **temporal difference learning** operates directly on relationships between states.

- Whenever an agent transitions from state s to state s' (in which it received rewards $R(s)$ and $R(s')$ respectively), it updates its estimate of utility as follows:

$$U(s') = R(s') \quad if\ s' has\ not\ been\ visited\ previously$$

$$U(s) = U(s) + \alpha\big(N_s(s)\big)\big(R(s) + \gamma U(s') - U(s)\big)$$

- where $\alpha(N_s(s))$ is a "step-size", decreasing in the number of times the state s is encountered (e.g. $\frac{1}{1+N_s(s)}$).

Aston University
Birmingham

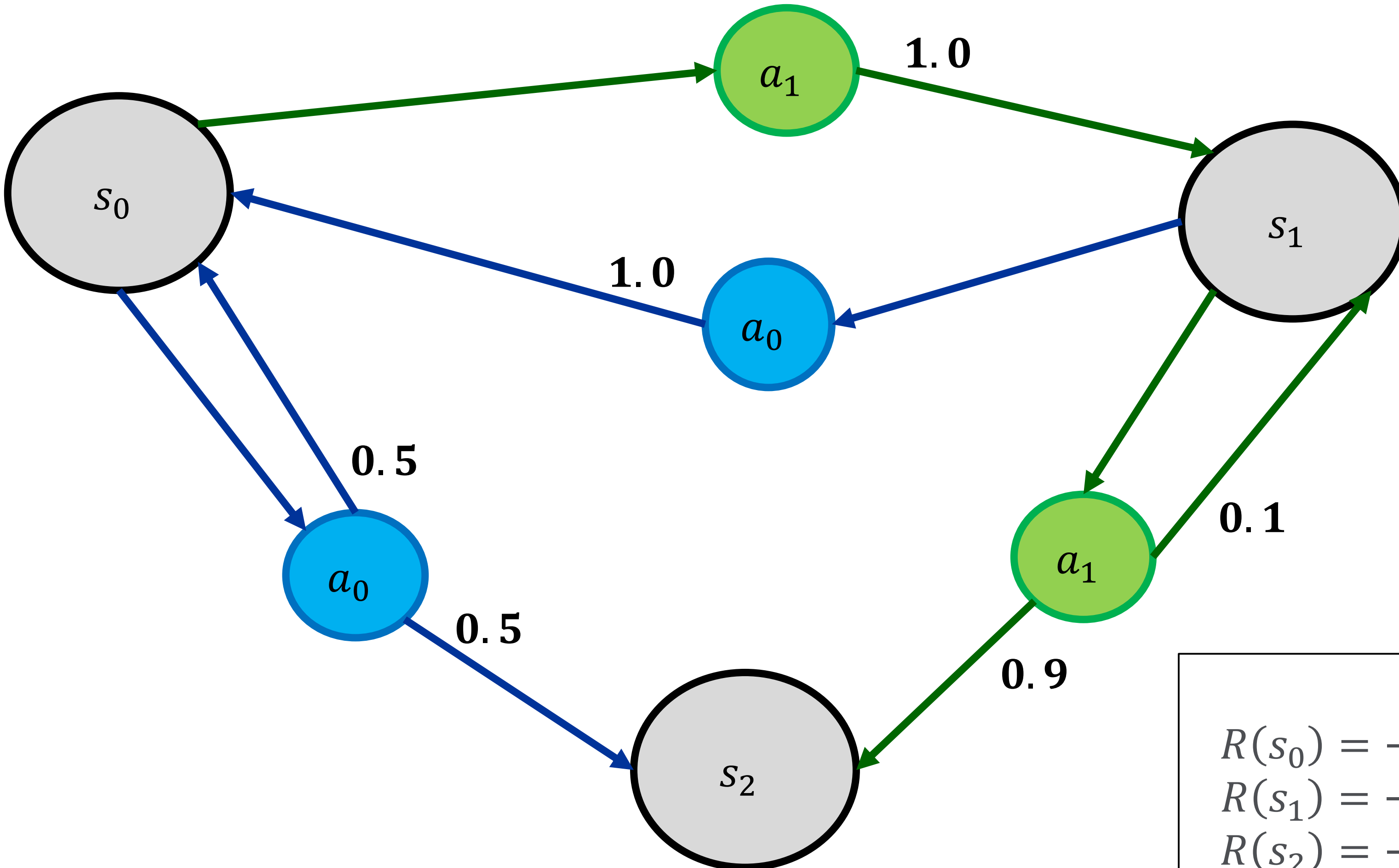
TDL Breakdown

- Breaking down the equation:

$$U(s) + \alpha\big(N_s(s)\big)\big(\underbrace{R(s) + \gamma U(s')}_{\text{Estimate of utility of } s \text{ based on transition in current episode}} - \underbrace{U(s)}_{\text{Previous estimate of utility of } s}\big),$$

  - This difference $(R(s) + \gamma U(s') - U(s))$ will be positive if the estimate of utility based on the current episode is higher than our previous estimate.
  - A positive difference would be evidence that our estimate of utility $(U(s))$ was too low and should be increased.
  - $\alpha\big(N_s(s)\big)$ decreases as we encounter state s more. Each new observation is a smaller fraction of our total observations.

Temporal Difference Learning Illustrated



$$R(s_0) = -0.50$$
$$R(s_1) = -0.75$$
$$R(s_2) = -0.10$$

Aston University
Birmingham

Temporal Difference Learning Illustrated

- Imagine that the previous MDP was encountered twice by an agent whose policy was to play a_1 in both states, generating the following sequences of states and rewards:

- $s_0(-0.5) \rightarrow s_1(-0.75) \rightarrow s_2(-0.1)$
- $s_1(-0.75) \rightarrow s_1(-0.75) \rightarrow s_2(-0.1)$

- How to apply TDL?

Temporal Difference Learning Illustrated

Transitions

$s_0(-0.5)$
$s_0(-0.5) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

$s_1(-0.75)$
$s_1(-0.75) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

Counts

$N_s(s_0) = 0$
$N_s(s_1) = 0$
$N_s(s_2) = 0$

Utility Estimates

$U(s_0) = ?$
$U(s_1) = ?$
$U(s_2) = ?$

Update

State s_0 encountered for the first time.

$$U(s_0) = -0.5$$

Temporal Difference Learning Illustrated

Transitions

$s_0(-0.5)$
$s_0(-0.5) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

$s_1(-0.75)$
$s_1(-0.75) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

Counts

$$N_s(s_0) = 0$$
$$N_s(s_1) = 0$$
$$N_s(s_2) = 0$$

Utility Estimates

$$U(s_0) = -0.5$$
$$U(s_1) = ?$$
$$U(s_2) = ?$$

Update

State s_1 encountered for the first time.

$$U(s_1) = -0.75$$

State s_0 updated based on observed transition to s_1.

$$N_s(s_0) = 1$$

$$U(s_0) = \overbrace{-0.5}^{U(s)} + \frac{1}{\underbrace{1}_{N_s(s)} + 1}(\underbrace{-0.5}_{R(s)} \overbrace{-\,0.75}^{U(s')} + \underbrace{0.5}_{U(s)}) = -0.875$$

Temporal Difference Learning Illustrated

Transitions

$s_0(-0.5)$
$s_0(-0.5) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

$s_1(-0.75)$
$s_1(-0.75) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

Counts

$$N_s(s_0) = 1$$
$$N_s(s_1) = 0$$
$$N_s(s_2) = 0$$

Utility Estimates

$$U(s_0) = -0.875$$
$$U(s_1) = -0.75$$
$$U(s_2) = ?$$

Update

State s_2 encountered for the first time.

$$U(s_2) = -0.1$$

State s_1 updated based on observed transition to s_2.

$$N_s(s_1) = 1$$

$$U(s_1) = \overbrace{-0.75}^{U(s)} + \frac{1}{\underbrace{1+1}_{N_s(s)}}\left(\underbrace{-0.75}_{R(s)} - \overbrace{0.1}^{U(s')} + \underbrace{0.75}_{U(s)}\right) = -0.8$$

Temporal Difference Learning Illustrated

Transitions

$s_0(-0.5)$
$s_0(-0.5) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

$s_1(-0.75)$
$s_1(-0.75) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

Counts

$N_s(s_0) = 1$
$N_s(s_1) = 1$
$N_s(s_2) = 0$

Utility Estimates

$U(s_0) = -0.875$
$U(s_1) = -0.8$
$U(s_2) = -0.1$

Update

State s_1 encountered previously. No change.

Temporal Difference Learning Illustrated

Transitions

$s_0(-0.5)$
$s_0(-0.5) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

$s_1(-0.75)$
$s_1(-0.75) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

Counts

$N_s(s_0) = 1$
$N_s(s_1) = 1$
$N_s(s_2) = 0$

Utility Estimates

$U(s_0) = -0.875$
$U(s_1) = -0.8$
$U(s_2) = -0.1$

Update

State s_1 encountered previously. No change.

State s_1 updated based on observed transition to s_1.

$$N_s(s_1) = 2$$

$$U(s_1) = \underbrace{-0.8}_{U(s)} + \frac{1}{\underbrace{1+2}_{N_s(s)}}(\underbrace{-0.75}_{R(s)} - \underbrace{0.8}_{U(s')} + \underbrace{0.8}_{U(s)}) = -1.05$$

Temporal Difference Learning Illustrated

Transitions

$s_0(-0.5)$
$s_0(-0.5) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

$s_1(-0.75)$
$s_1(-0.75) \rightarrow s_1(-0.75)$
$s_1(-0.75) \rightarrow s_2(-0.1)$

Counts

$N_s(s_0) = 1$
$N_s(s_1) = 2$
$N_s(s_2) = 0$

Utility Estimates

$U(s_0) = -0.875$
$U(s_1) = -1.05$
$U(s_2) = -0.1$

Update

State s_1 encountered previously. No change.

State s_1 updated based on observed transition to s_2.

$$N_s(s_1) = 3$$

$$U(s_1) = \overbrace{-1.05}^{U(s)} + \frac{1}{1 + \underbrace{3}_{N_s(s)}}(\underbrace{-0.75}_{R(s)} - \overbrace{0.1}^{U(s')} + \underbrace{1.05}_{U(s)}) = -1$$

Conclusion

- You should know:
  - Know what a sequential decision making problem is and how to formulate it as a Markov Decision Process (MDP).
  - Understand solution policies for MDPs and know a simple algorithm for generating an optimal policy given sufficient information.
  - Understand the characteristics of problems addressed by reinforcement learning and know an algorithm for policy evaluation on these problems well enough to implement it.