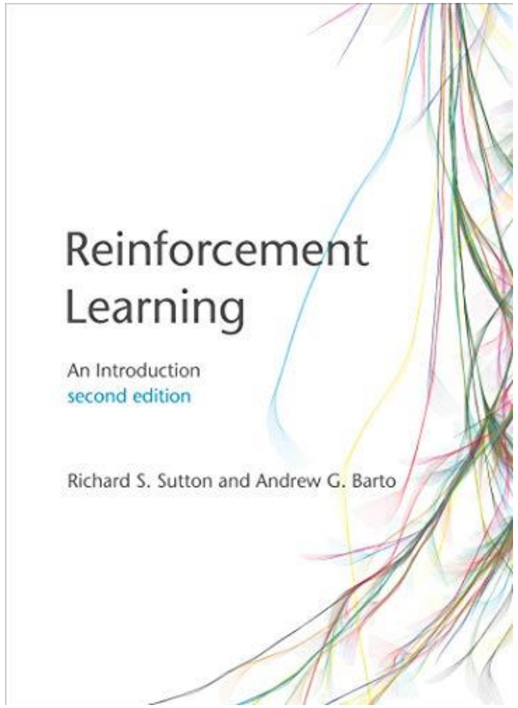
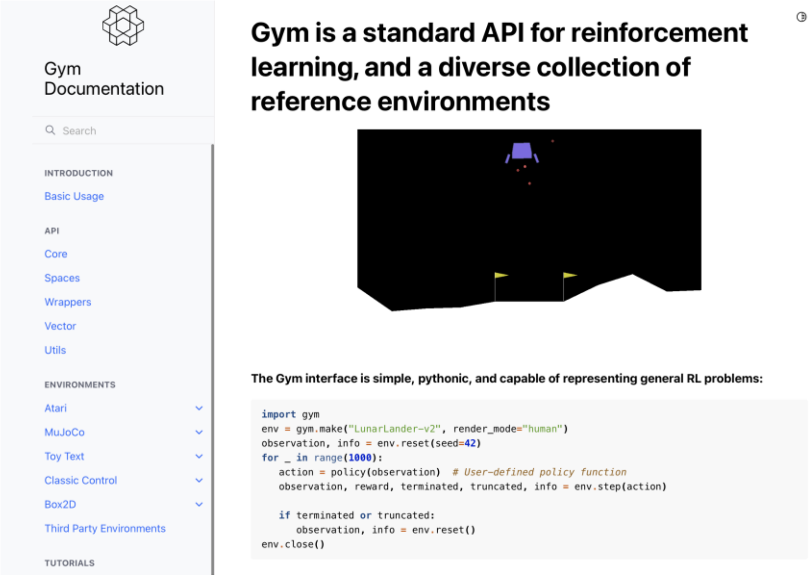
L2D
Reinforcement learning

Dr Neythen Treloar

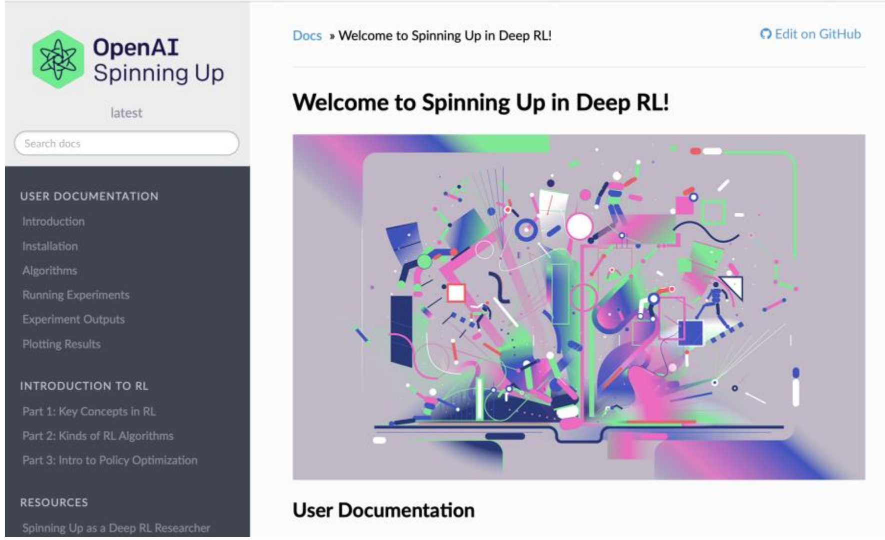
Resources



Available free here:
https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf



https://github.com/Farama-Foundation/Gymnasium



https://spinningup.openai.com/en/latest/

Deep reinforcement learning for the control of microbial co-cultures in bioreactors

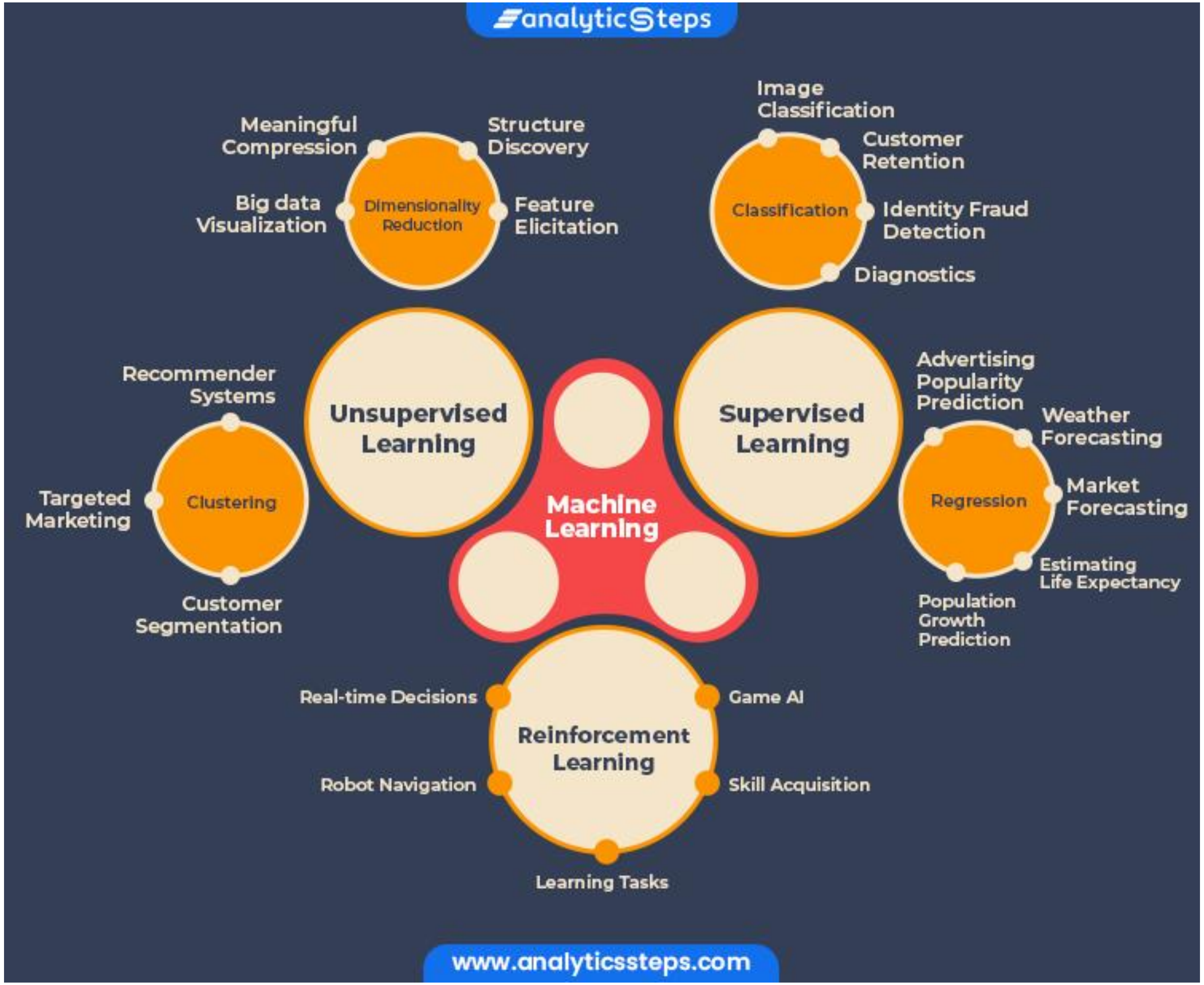
Neythen J. Treloar, Alex J. H. Fedorec, Brian Ingalls, Chris P. Barnes

https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1007783

Brief overview

- Session 1: RL intro and tabular methods
  - Monte Carlo techniques
  - Temporal difference
    - Q-learning (off policy)
    - SARSA (on policy)
- Session 2: Deep reinforcement learning
  - Difficulties using neural networks for RL and the solutions

Introduction to machine learning approaches



Reinforcement learning demos

Learning good behaviour through trial and error

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

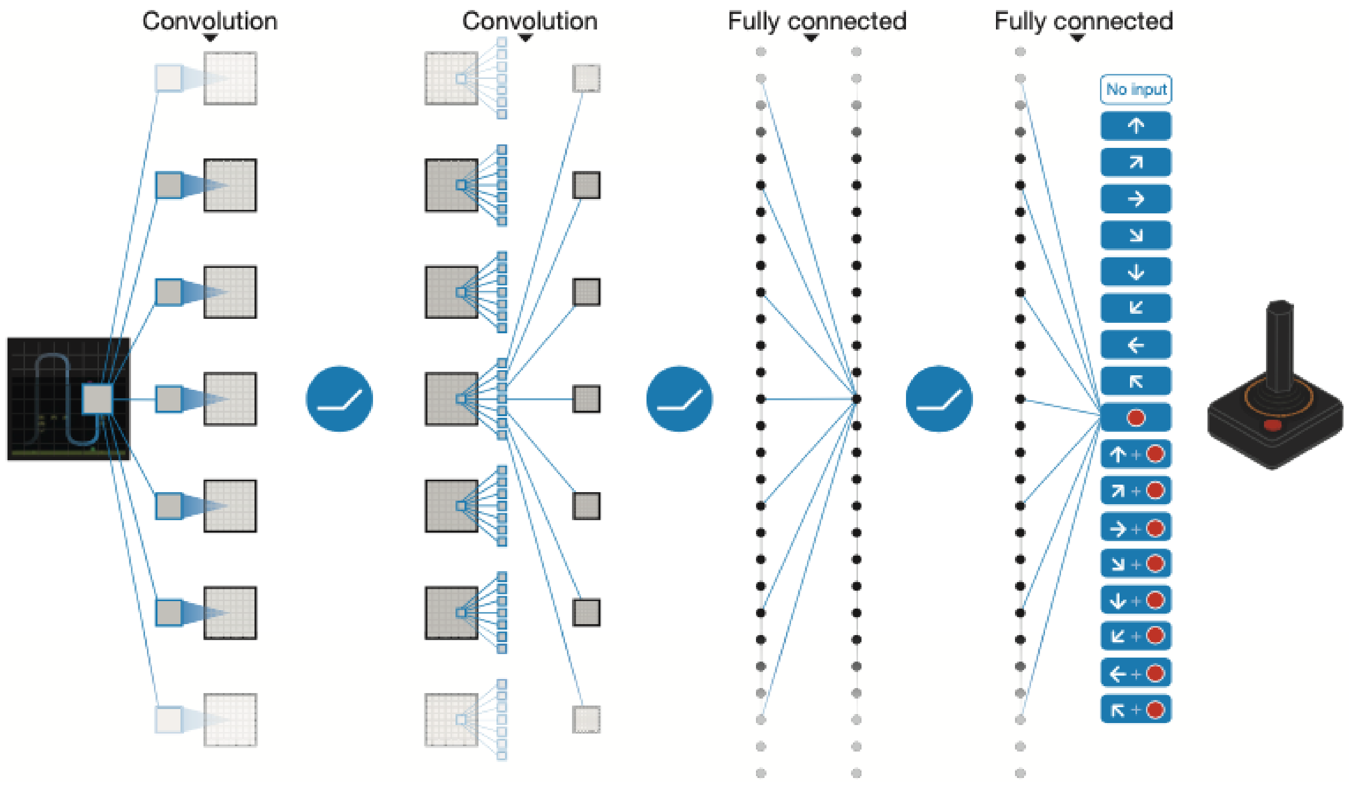
The ancient history

- Has roots in both psychology and control theory
- Control:
  - Bellman developed dynamic programming to solve the Bellman equations
  - The only feasible way to solve general stochastic optimal control problems
- Trial and error learning in animal behaviour:
  - How do animals learn behaviour that maximises positive effects
  - Pavlov etc.
- Development of Q-learning by Chris Watkins 1989 [1]
- TD-gammon, backgammon playing program 1994 [2]
- Page 13 in Sutton and Barto

[1] Watkins, C.J.C.H., 1989. Learning from delayed rewards
[2] Tesauro, G., 1994. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*

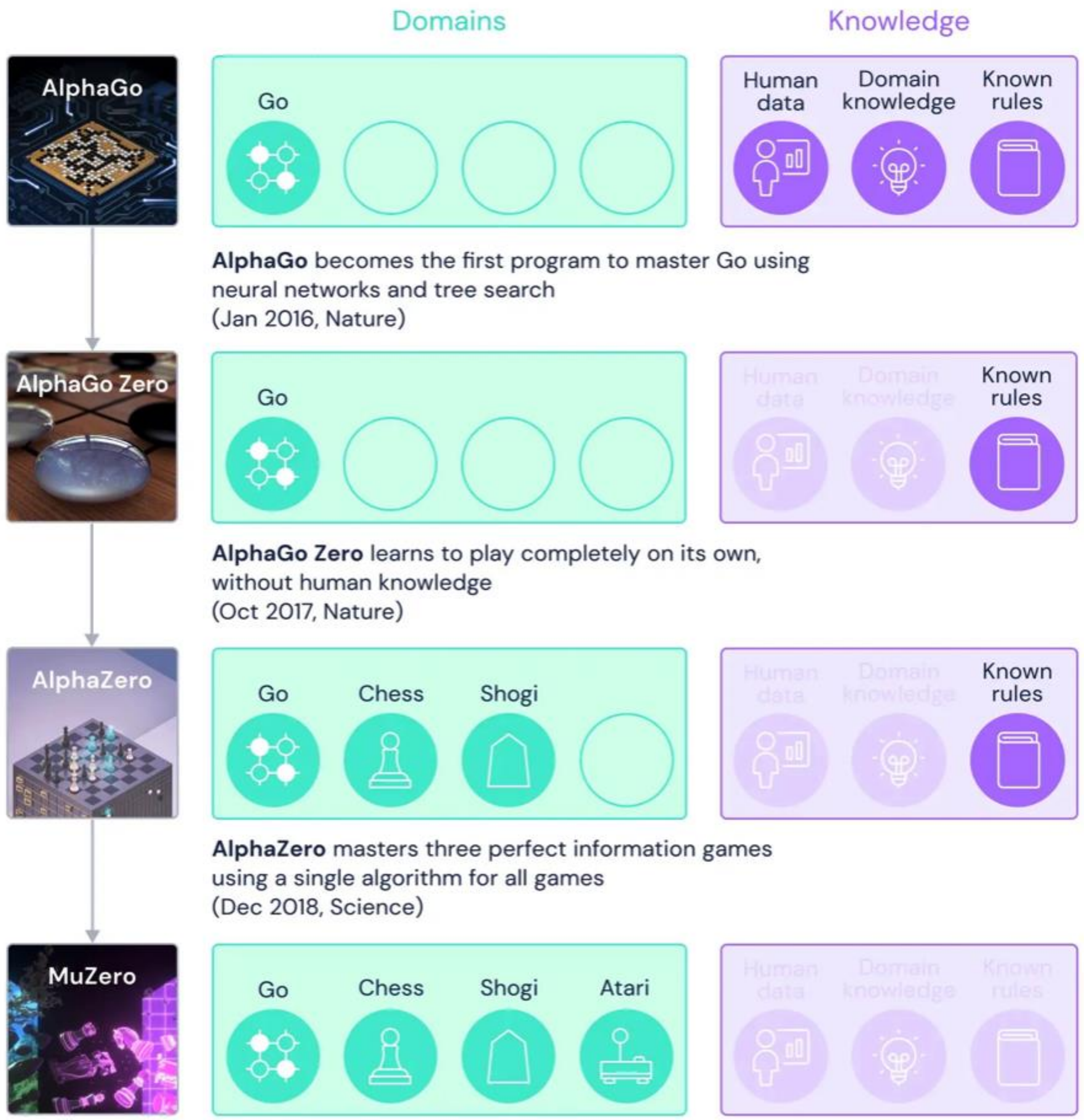
Modern history

- Development of DQN, Q learning with neural networks (2015)



Mnih, V et al. 2015. Human-level control through deep reinforcement learning. *nature*

- AlphaGo [1]
- AlphaGo Zero [2]
- Alpha Zero [3]
- Mu Zero [4]



https://www.deepmind.com/blog/muzero-mastering-go-chess-shogi-and-atari-without-rules

[1] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *nature*
[2] Silver, David, et al. "Mastering the game of go without human knowledge." *nature*
[3] Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science*
[4] Schrittwieser, Julian, et al. "Mastering atari, go, chess and shogi by planning with a learned model." *Nature*

Useful applications

Deep reinforcement learning for the control of microbial co-cultures in bioreactors

Neythen J. Treloar, Alex J. H. Fedorec, Brian Ingalls, Chris P. Barnes

Deep reinforcement learning for optimal experimental design in biology

Neythen J. Treloar, Nathan Braniff, Brian Ingalls, Chris P. Barnes

Article | Published: 09 June 2021

A graph placement methodology for fast chip design

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter & Jeff Dean

Nature **594**, 207–212 (2021) | Cite this article

Article | Open Access | Published: 16 February 2022

Magnetic control of tokamak plasmas through deep reinforcement learning

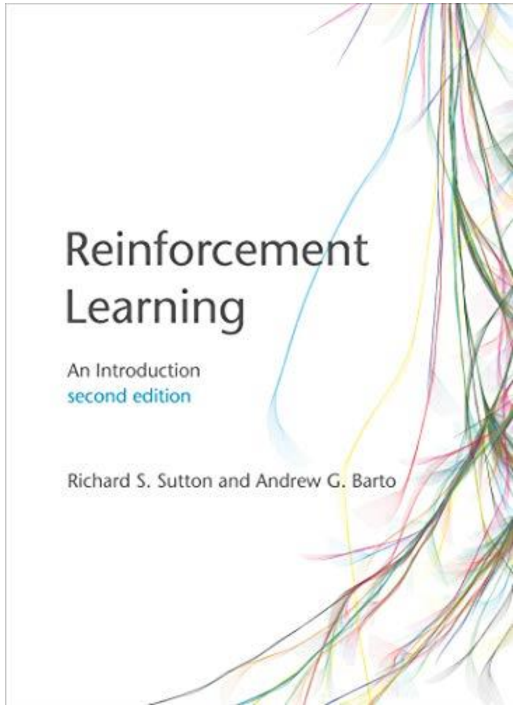
Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis & Martin Riedmiller

— Show fewer authors

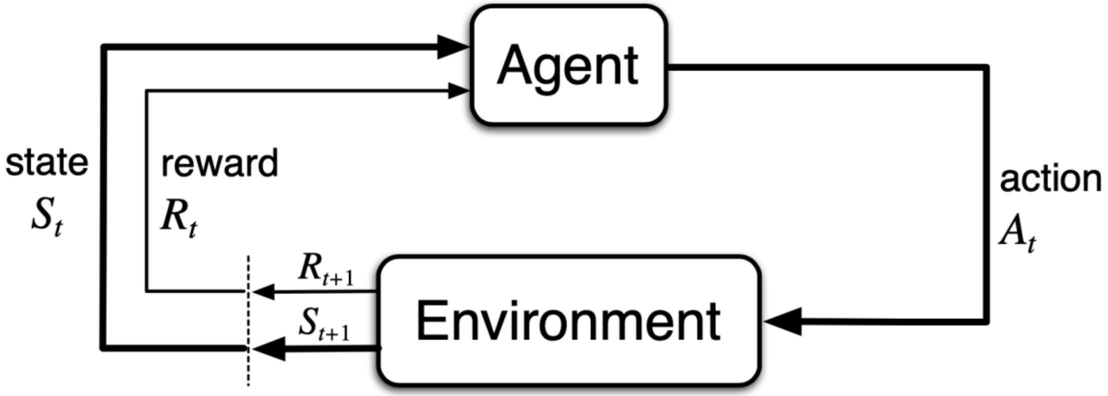
Nature **602**, 414–419 (2022) | Cite this article

Starting at the end

1. Multi armed bandits
2. Finite Markov decision processes
3. Dynamic programming
4. Reinforcement learning



Markov decision process



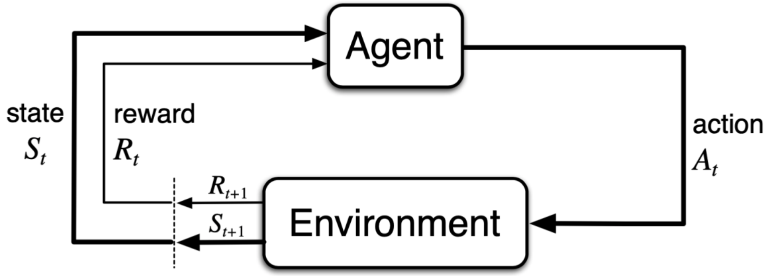
We work with discrete timesteps: $t_0, t_1, t_2, t_3, \ldots$

In a fully observable MDP all information is encoded in the state i.e. the past is irrelevant:
- Most RL theory is based on this assumption

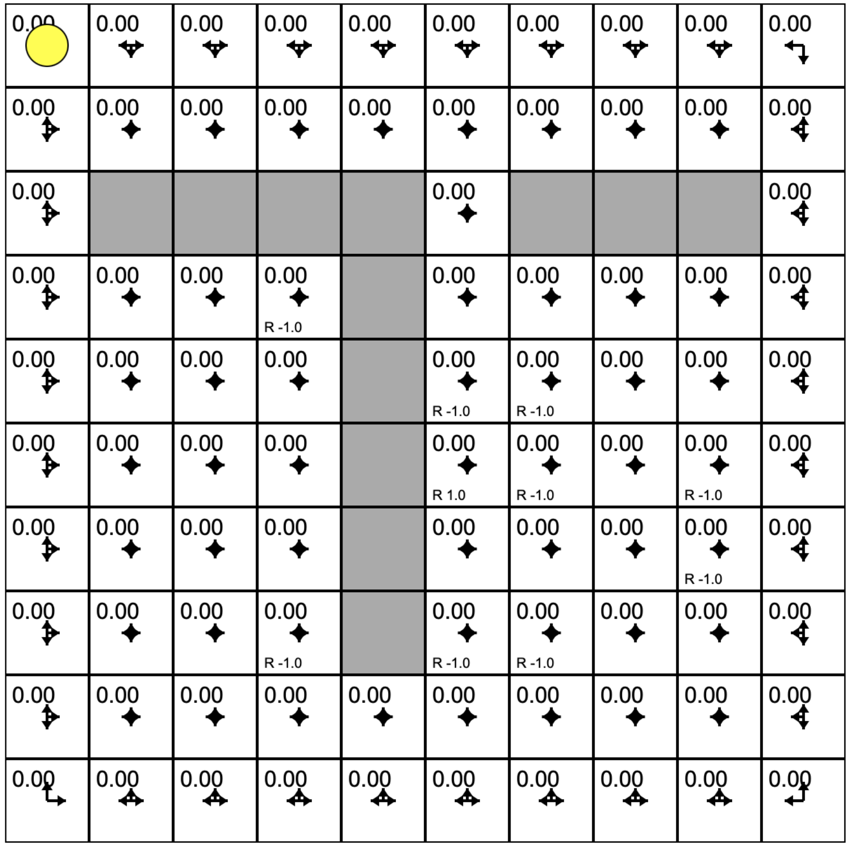
In a partially observable MPD (POMDP) only partial information can be observed and therefore history is important:
- We can include history using e.g. recurrent neural networks

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*

Markov decision process

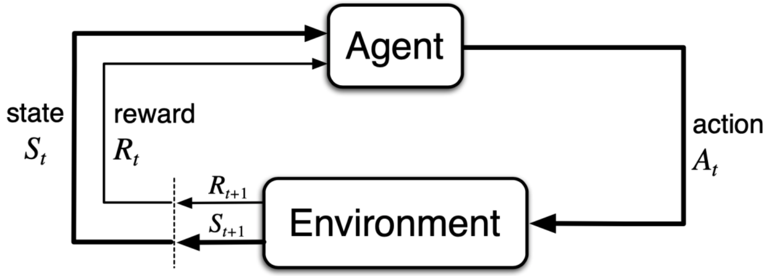


- Agent: yellow ball
- Environment: grid
- Action: move (U,D,L,R)
- State: current position
- Reward: reach the goal position and avoid penalty positions

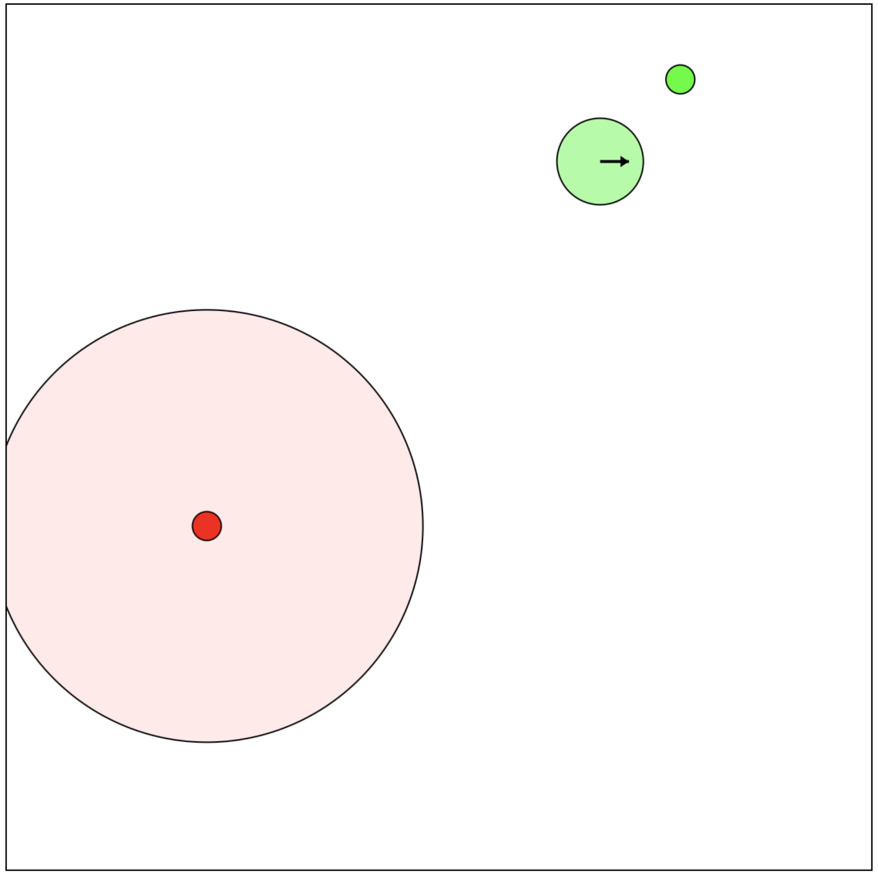


Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*
https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

Markov decision process



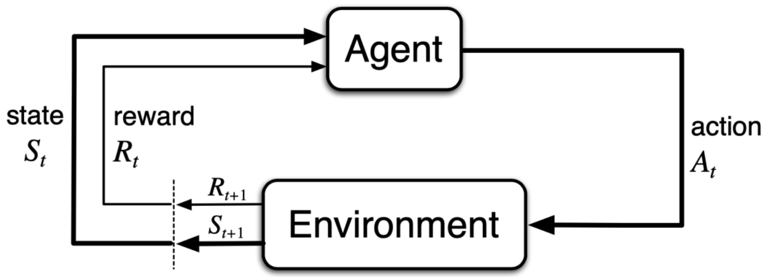
- Agent: large green ball
- Environment: area
- Action: apply thrusters (U,D,L,R)
- State: position of agent, green and red ball
- Reward: +ve close to the small green ball, -ve close to the red ball

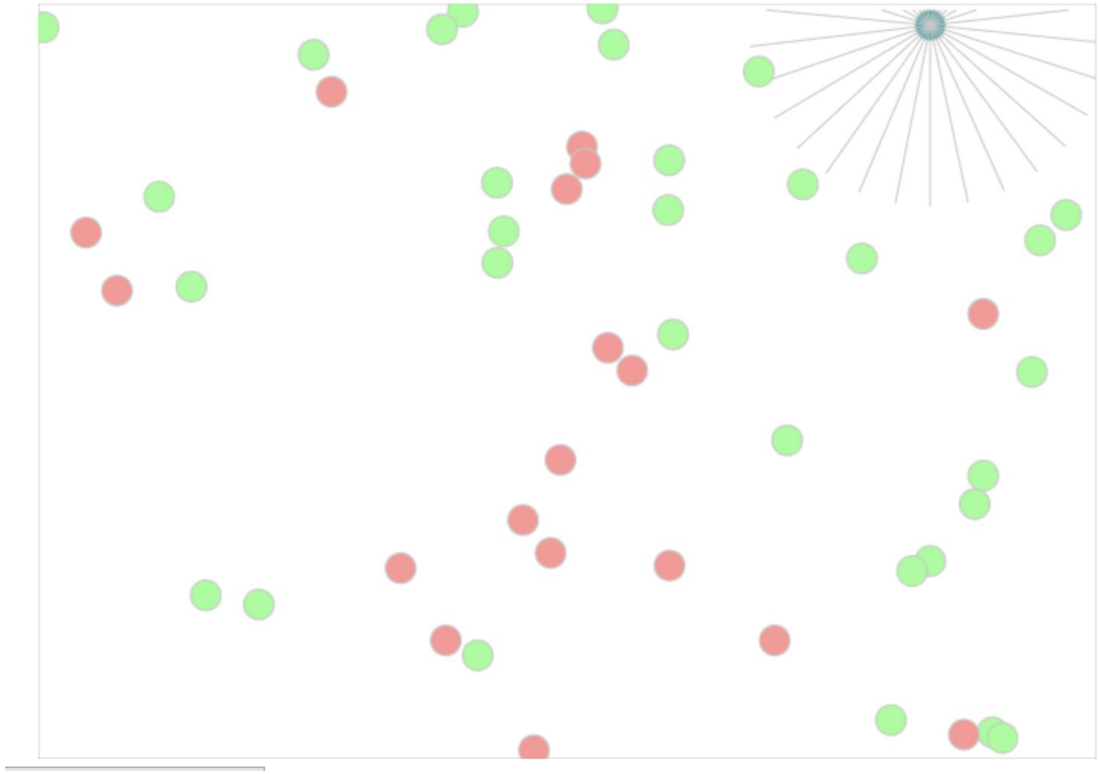


Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*

https://cs.stanford.edu/people/karpathy/reinforcejs/puckworld.html

Markov decision process



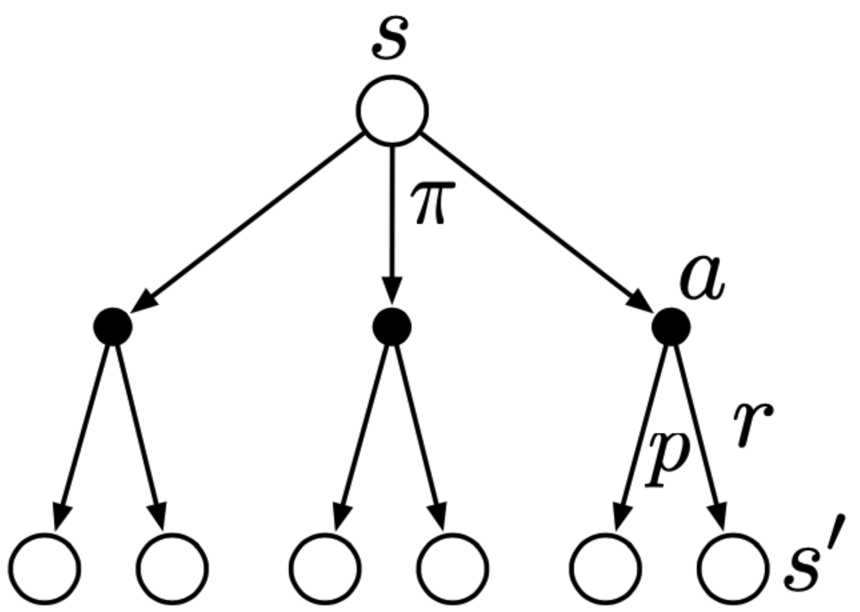


- Agent: blue ball
- Environment: area
- Action: apply thrusters (U,D,L,R)
- State: Agent has 30 eye sensors which it uses to measure the range and velocity of sensed object. Agent also knows its own velocity
- Reward: +ve eat the red apples, -ve eat the green poison apples

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*

https://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html

Backup diagrams



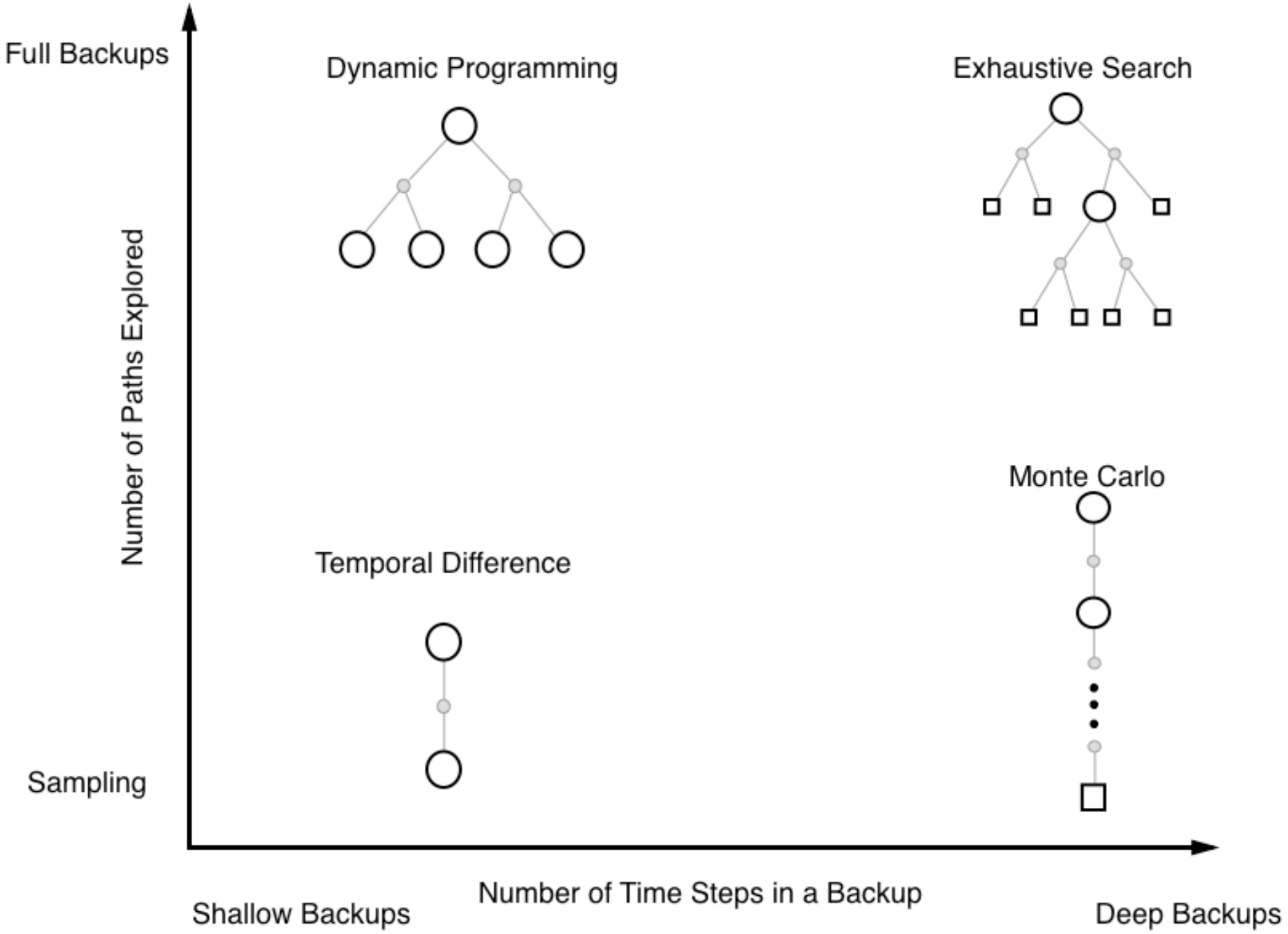
Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*

Episodes

- We often train over repeated **episodes**
- An episode is a sequence of states, actions and rewards that continues until a terminal state is reached
- For example:
  - A level in a video game
  - An experiment
  - A match of a board game
- It is also possible to train on one infinitely long episode:
  - The puckworld and waterworld examples

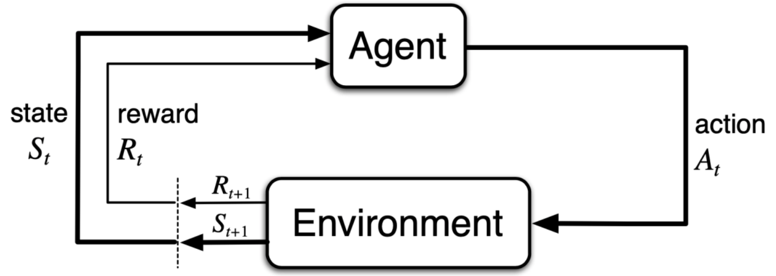
Have to be model based

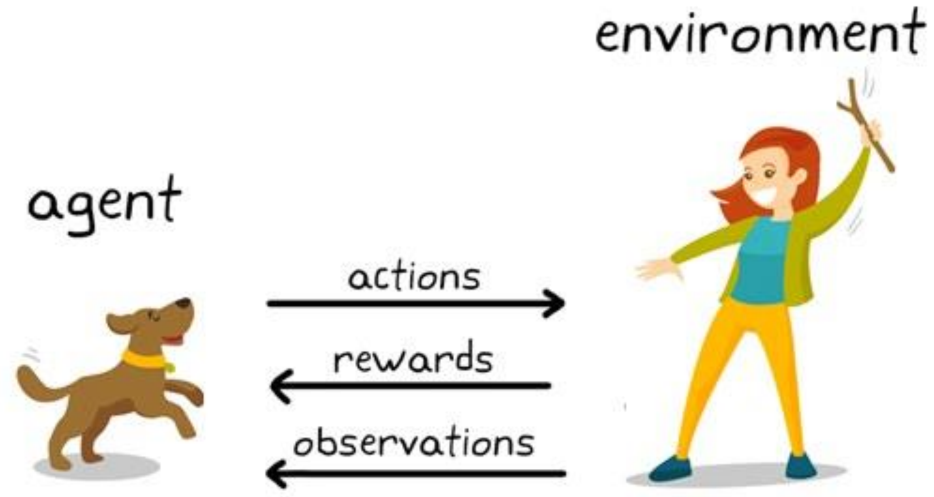
Can learn from experience
(these are RL methods)

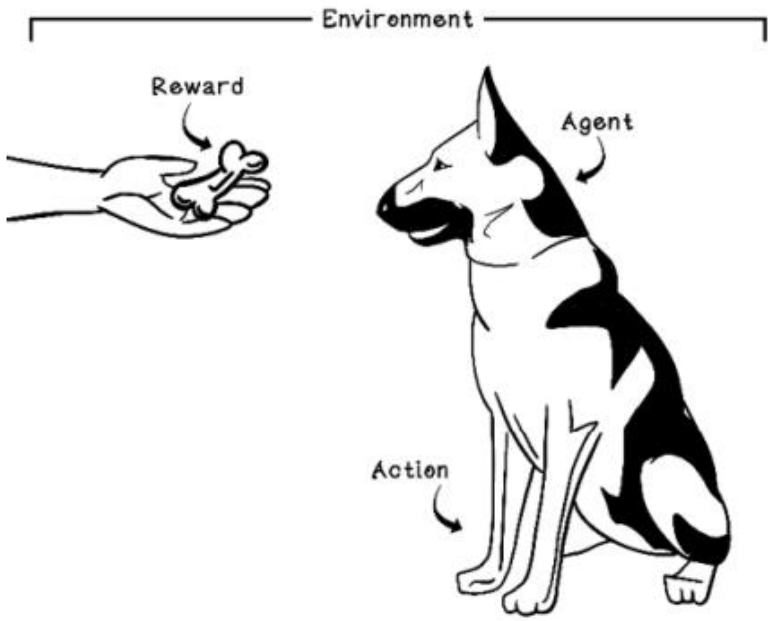


Reinforcement learning

Agent learns an optimal behaviour **policy** by interacting with its **environment** to maximise its total **reward**, the **return**





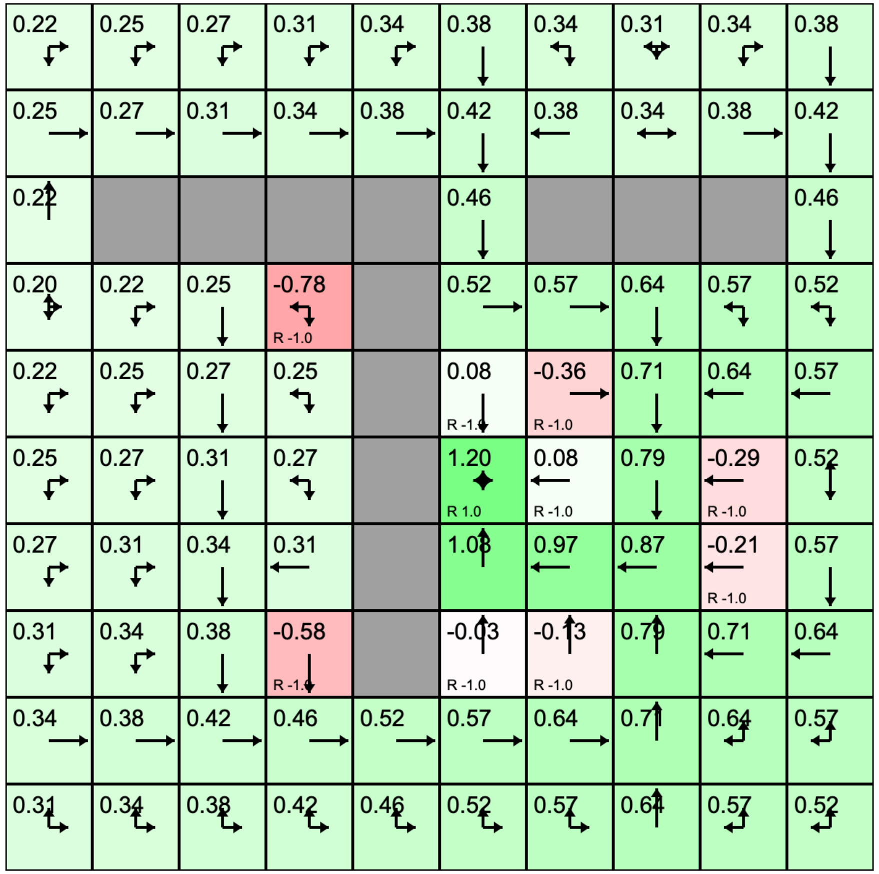


The value function

- Many RL methods aim to learn a **value function**
- This is an estimate of the **expected discounted future return** the agent will receive and is learned through trial and error
- This can be learned for different states (a **state value function**) or different combinations of state and action (a **state-action value function**)

The state, value function $V(s)$

- In the grid world example, the state is the position on the grid
- For each state a **value** is learned, which is an estimate of the expected discounted return for an agent in that state.
- The table is a state value function $V(s)$
- What is the value of being in the top left corner? $s = (0,0)$
  - 0.22
- What are the highest and lowest value states?
  - (5,5) and (3,3)



https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

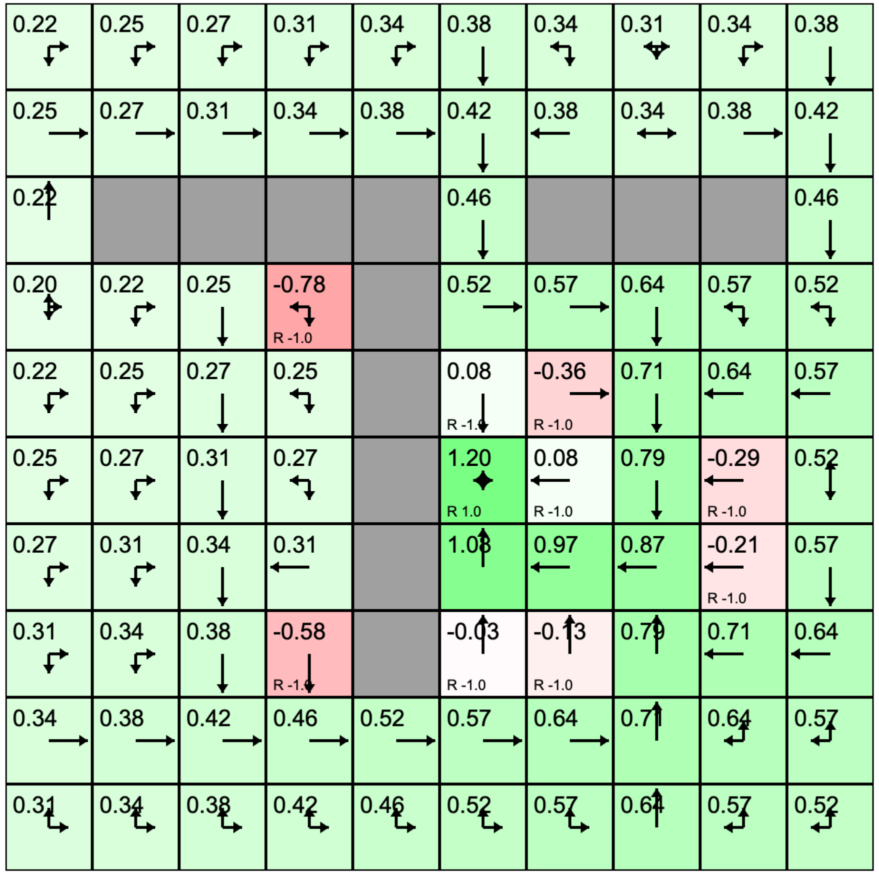
The state-action, value function $\mathrm{Q}(s, a)$

- The state-action value function $Q(s, a)$, this is the value obtained by starting in state s and taking action a

- In grid world we can choose from up to four actions, U,D,L,R

- E.g. for the top left corner $s = (0,0)$, we have two available actions, R or D. $Q\big((0,0), R\big) = Q\big((0,0), D\big) = 0.25$. Because starting from (0,0) and moving either R or D we end up in states with value 0.25

- What is $Q\big((3,2), R\big)$?
  - -0.78
- What is $Q\big((6,5), U\big)$?
  - 1.2
- Learning this is the basis of many basic and advanced RL techniques



https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Discounting

Agent learns from experience to maximise its total reward, the **return**. During learning this is discounted by a factor γ between 0 and 1

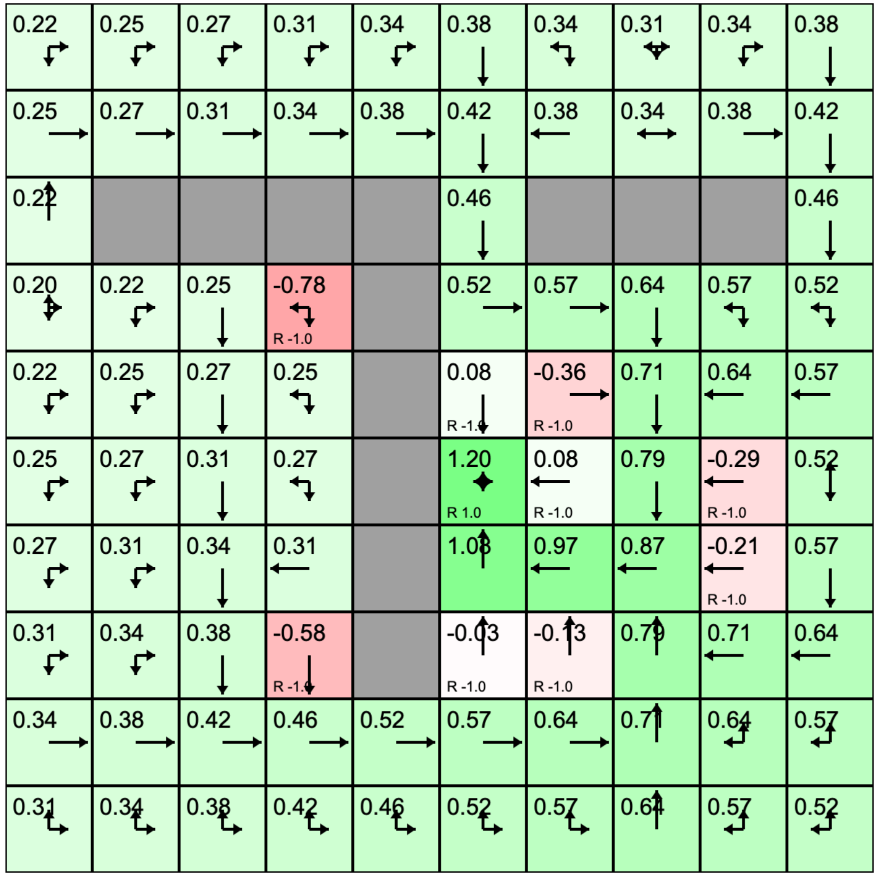
- $\gamma = 1$ a reward in the future is worth just as much as a reward now. This removes the time agency of the agent, a longer path to the reward will be just as optimal as a shorter path to the same reward. Can lead to instability, especially when episodes not time constrained
- $\gamma = 0$ agent is only concerned about the next reward, will forgo large future rewards just to get a small reward now.
- 0.95 is a good starting point
- What is the discount factor used to learn this value function?
  - 0.9, because the value decays by a factor of 0.9 for each step away from the goal

$$G = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4^4 + \ \ldots$$

$$\gamma = 0: \quad G = r_0$$

$$\gamma = 1: \quad G = r_0 + r_1 + r_2 + r_3 + r_4 + \ \ldots$$

$$\gamma = 0.95: \quad G = r_0 + 0.95 * r_1 + 0.9 * r_2 + 0.86 * r_3 + 0.81 * r_4 + \ \ldots$$

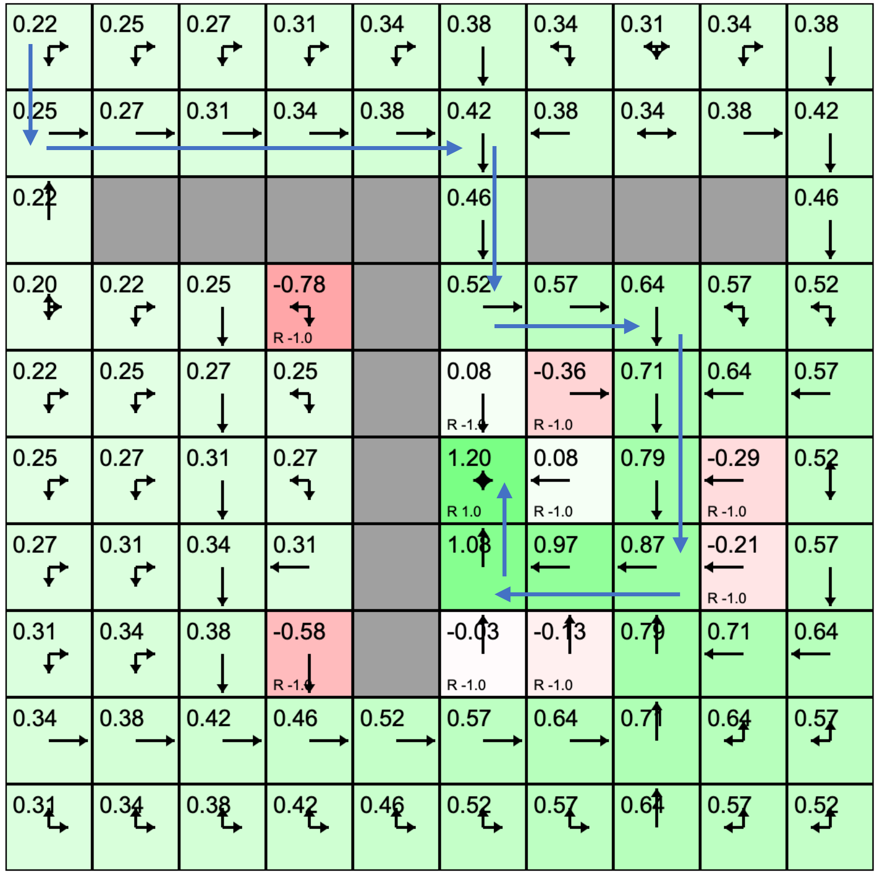


This is why the value of each position decreases as we move away from the goal

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

The policy

- The agent's overall goal is to learn the optimal behaviour **policy**
- The policy is the learned decision-making process the agent uses to select the next action, a, based on the current state, s
- The greedy policy is found by choosing the action with highest value for each state
- In grid world the learned policy is represented by the arrows
- What route does the agent take from the initial state (0,0) to the goal state (5,5) under this policy?

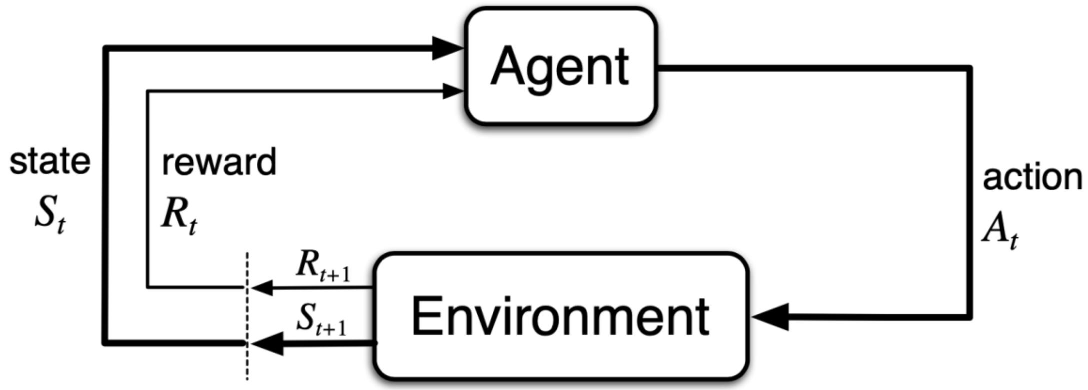


https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Terminology review

Agent learns an optimal behaviour **policy** by interacting with its **environment** to maximise its total **reward**, the **return**

A lot of RL methods learn a state-action **value function** $\boldsymbol{Q(s, a)}$**.** This is a learned estimate of the **expected discounted return** received after visiting every state-action pair



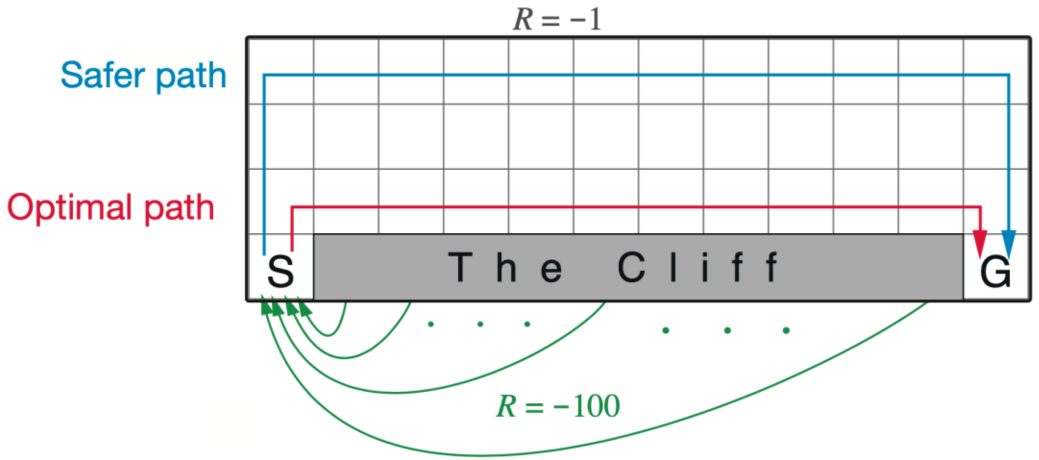
Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*

Exploration

- A key concept in RL is the exploration/exploitation trade off
- We need to explore so that we can discover new behaviours which might be optimal
- But we don't want to explore too much otherwise we will never exploit the knowledge we have gained
- A commonly used exploration policy is **ϵ-greedy**:
  - The **explore rate**, ϵ, is the probability that the agent takes a random action
  - With probability $1 - \epsilon$ the agent will take the greedy action $a = \max_a Q(s, a)$
  - The explore rate can be set to a constant small value (often 0.05) or can be set to 1 initially and decay throughout training

On vs Off policy methods

- Often, we use the ϵ-greedy policy to explore, but we want to learn the optimal greedy policy
- Meaning our **behaviour policy** (ϵ-greedy) is different to our **target policy** (greedy)
- If our behaviour policy is the same as our target policy this is called an **on-policy** method e.g. SARSA
- If our behaviour policy is different from our target policy this is called an **off-policy** method e.g. Q-learning
- When could it be advantageous to choose each type of method?



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*

Learning the value function: Monte Carlo

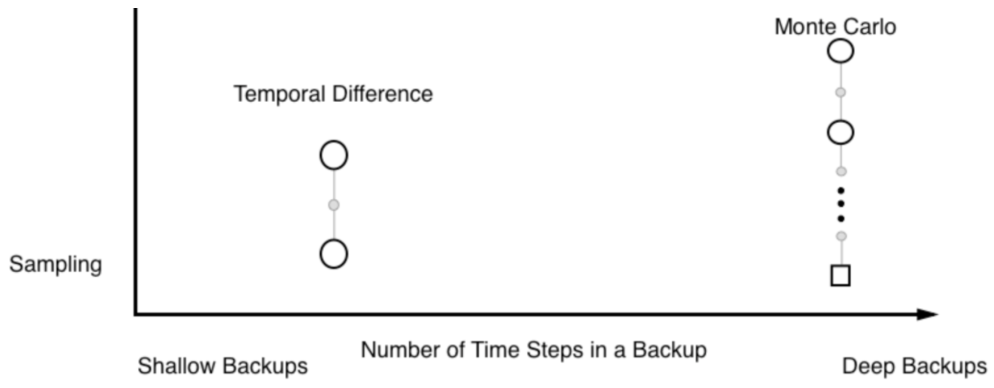
- Now all we need to do is learn the $Q(s, a)$ function

- In Monte Carlo RL we learn the mean return for each s, a pair
    - Keep track of sum of the return $S_G(s, a)$
    - And a count of high many times each s, a pair visited $C(s, a)$
    - So that $Q(s, a)$ = $S_G(s, a)$/ $C(s, a)$

- Implementation is a bit tricky, but principle is simple

- $G(s_t, a_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \gamma^T r_{t+T}$

- $Q(s_t, a_t) = S_G(s_t, a_t)/\ C(s_t, a_t)$

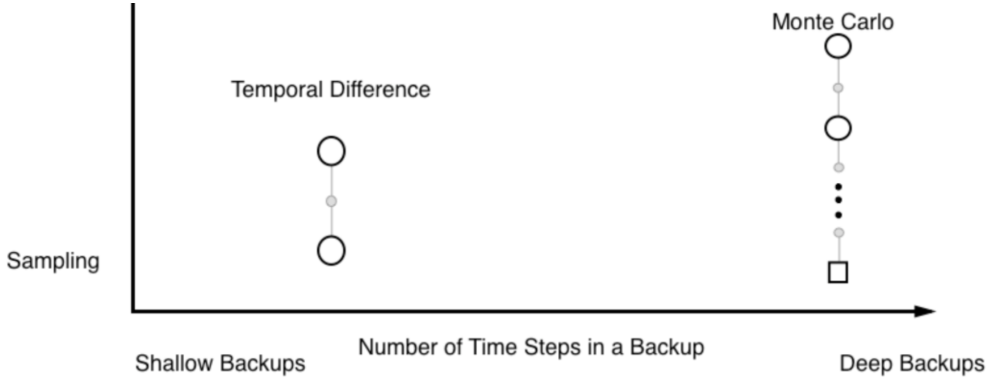


- We are restricted to updating only after an episode has finished, because we need to know the full sequence of states, actions and rewards
    - Won't work for non-episodic tasks and can cause problems even for episodic ones

- Off policy learning is more difficult (importance sampling in Sutton and Barto)

Temporal difference methods

- Temporal difference methods replace the observed future return with an estimate of the future return from the current value function $Q(s_t, a_t)$
  - This is called bootstrapping
  - This means we don't have to wait until the end of the episode to learn
  - Much easier to do off-policy learning

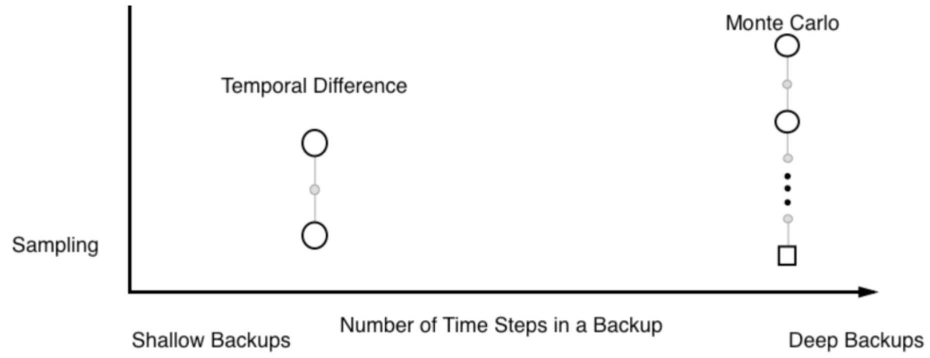


Learning the value function: on policy temporal difference (SARSA)

- Instead of using observed return:
  - $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \gamma^T r_{t+T}$

- We can bootstrap off current $Q(s, a)$ function to estimate the expected future return:
  - $\underbrace{r_t}_{\text{Reward from time} = t} + \underbrace{\gamma Q(s_{t+1}, a_{t+1})}_{\text{Expected return from time } t+1 \text{ onwards}}$

- We iteratively update the value function with learning rate α
  - $Q(s_t, a_t) = Q(s_t, a_t) + \alpha\,(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$

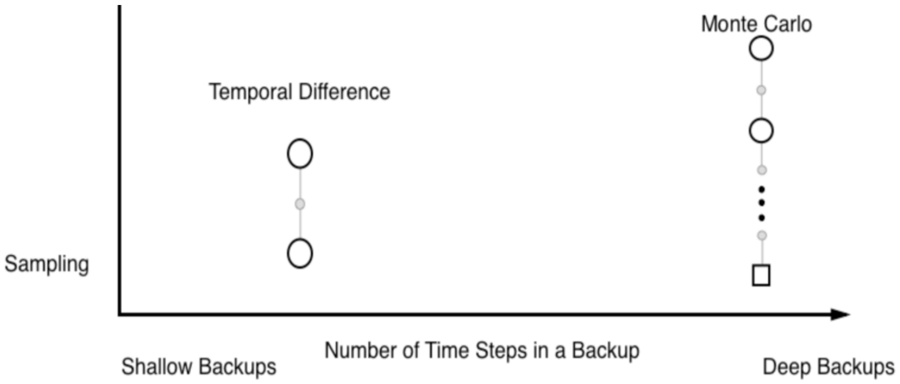


Learning the value function: off policy temporal difference (Q-learning)

- Instead of using observed return:
  - $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \gamma^T r_{t+T}$

- We can bootstrap off current $Q(s, a)$ function to estimate the expected future return:
  - $\underbrace{r_t}_{\text{Reward from time} = t} + \gamma \underbrace{\max_a Q(s_{t+1}, a)}_{\text{Expected return from time } t+1 \text{ onwards}}$

- We iteratively update the value function with learning rate α
  - $Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$



Q-learning vs SARSA

SARSA (on policy)

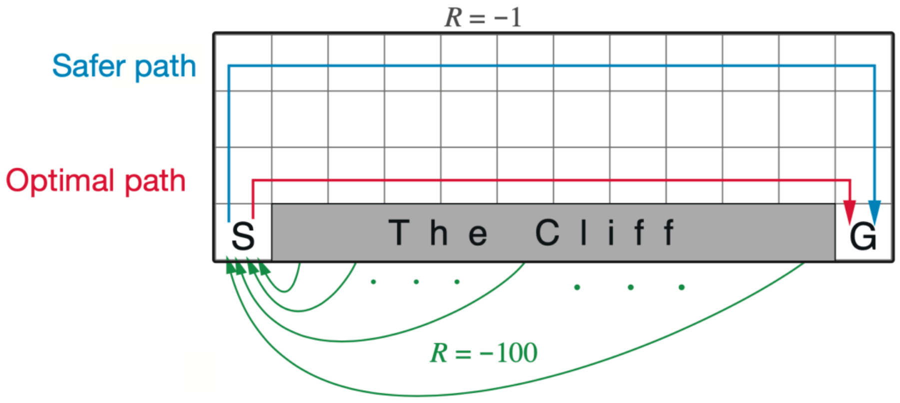
- $Q(s_t, a_t) = Q(s_t, a_t) + \alpha\,(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$

Update according to the action taken by the behaviour policy

Q-learning (off policy)

- $Q(s_t, a_t) = Q(s_t, a_t) + \alpha\,(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$

Update according to the greedy policy



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*

Transitions

- $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right)$
  - Q learning transition: (s_t, a_t, r_t, s_{t+1})
- $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$
  - SARSA transition: $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$
- $Q(s_t, a_t) = S_G(s_t, a_t) / C(s_t, a_t)$
  - Monte Carlo learns episodically so uses a sequence of transitions:
    - $(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_N, a_N, r_N)$

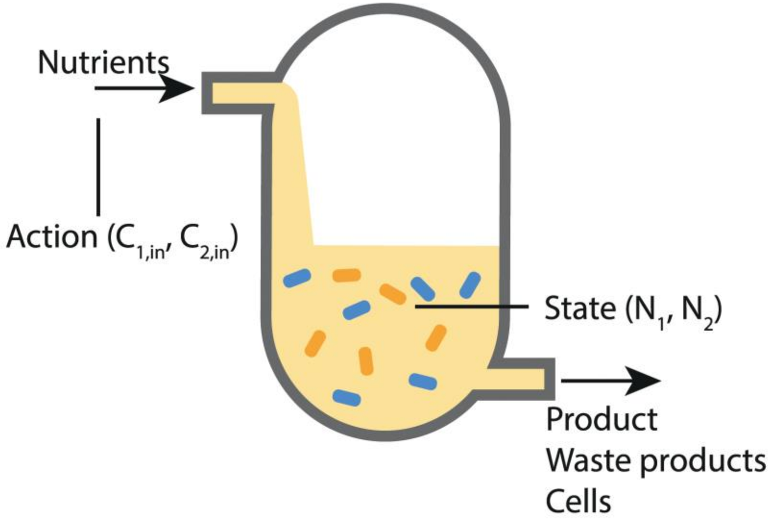
Pseudocode overview – a TD learning agent

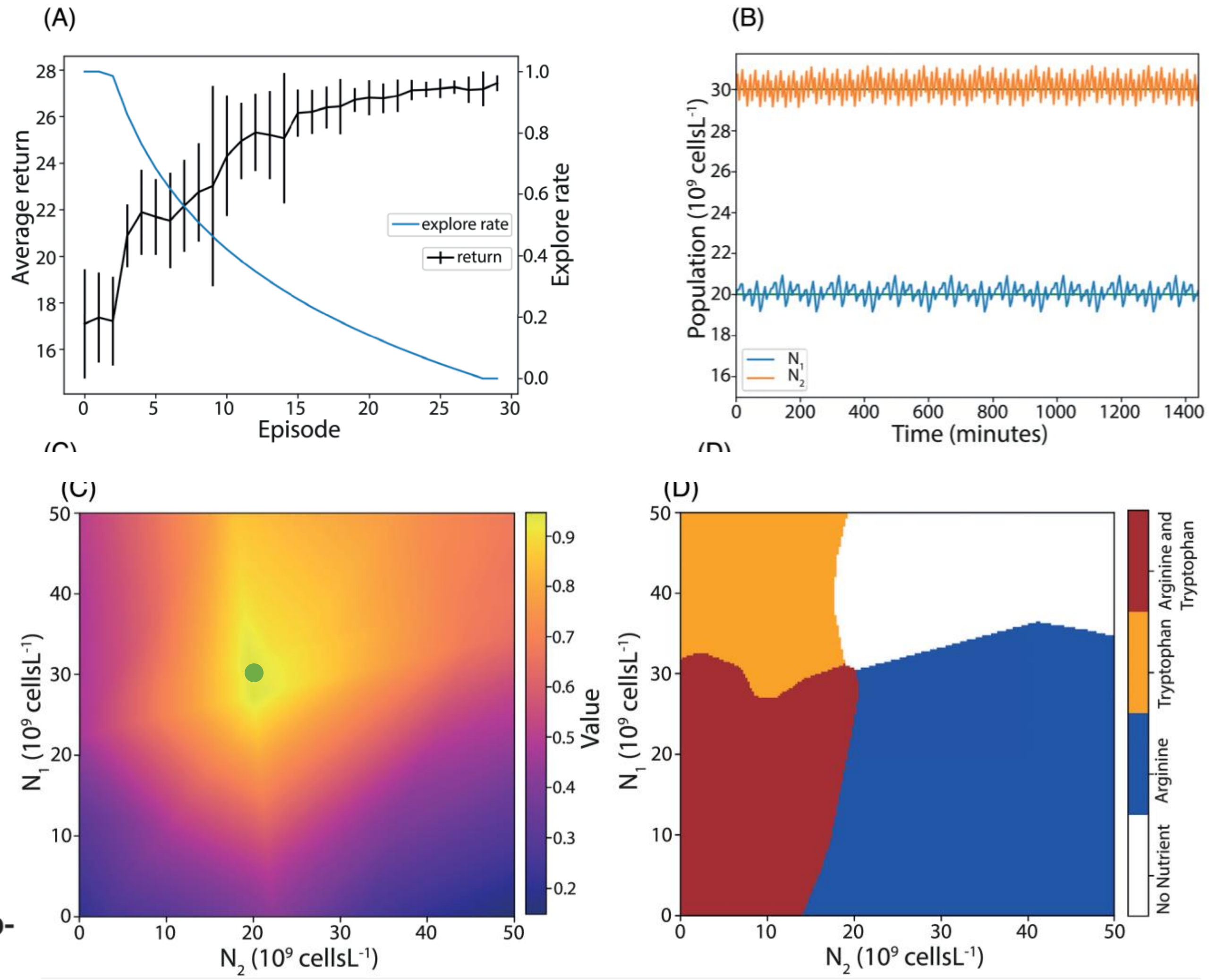
Q learning agent:
- Q_function
  - This is the function that maps state-action pairs to learned values
  - In theory can be any function, we will look at using lookup tables and neural networks
- Policy
  - The agents policy maps a state observed from the environment to an action to take
  - We will use epsilon greedy policies to incorporate exploration
- Update_Q_function
  - This function takes a transition and updates our current value function using the Q learning or SARSA update rules

Pseudocode overview – training loop

```
For each episode:

        while the episode has not reached a terminal state:
                get an action from the agent's policy
                apply that action to the environment to update the current state

                if we are training a temporal difference agent update the value function

        if we are training a Monte Carlo agent update the value function
```

Example training output





Deep reinforcement learning for the control of microbial co-cultures in bioreactors

Neythen J. Treloar, Alex J. H. Fedorec, Brian Ingalls, Chris P. Barnes

Summary

- RL can be used to solve control problems
- Often this is done by learning a value function from a sequence of states, actions and rewards
- This can be done episodically using Monte Carlo techniques
- Or online using temporal difference
  - Q-learning (off policy)
  - SARSA (on policy)
- Temporal difference methods allow us to update the value function during an episode and easily do off policy learning