Reinforcement Learning Basics

Feryal Behbahani, Jane Wang

Volunteers: Anoop Kulkarni, Sowmya Parthiban, Matthew Sargent

neuromatch academy

Special thanks

https://github.com/Feryal/rl_mlss_2020

Section 1

Introduction to reinforcement learning

Jane Wang

neuromatch academy

What is reinforcement learning?

- Learning from reward, “trial-and-error”
- A general framework/formalism for deciding how to act in an environment
- Can be divided into 2 components
  - 1) prediction
  - 2) control

Reward/value prediction (classical conditioning)

Ivan Pavlov, 1897

Reward/value prediction (classical conditioning)

Ivan Pavlov, 1897

Reward/value prediction (classical conditioning)

Ivan Pavlov, 1897

Reward prediction error

RPE = **Actual reward** - **Expected**

Reward prediction error

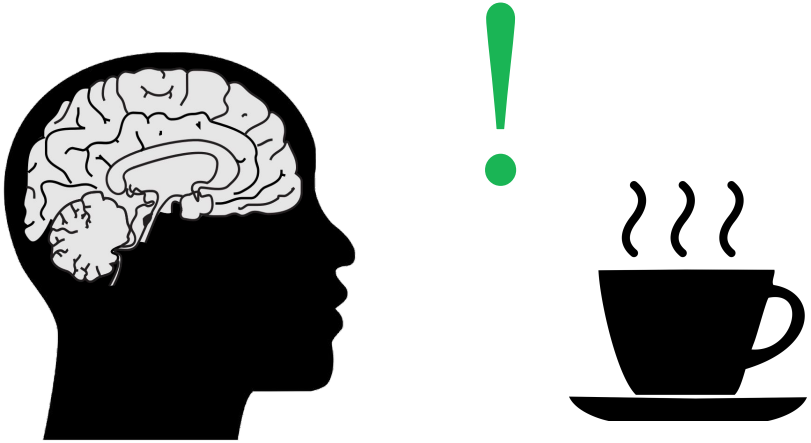
RPE = **Actual reward** - **Expected**

Learning happens when there's prediction error

$$\textbf{Actual reward} \xleftarrow{\alpha} \textbf{Expected}$$

With learning rate α

Reward prediction error



RPE = **Actual reward** - **Expected**

Learning happens when there's prediction error

$$\textbf{Actual reward} \xleftarrow{\alpha} \textbf{Expected}$$

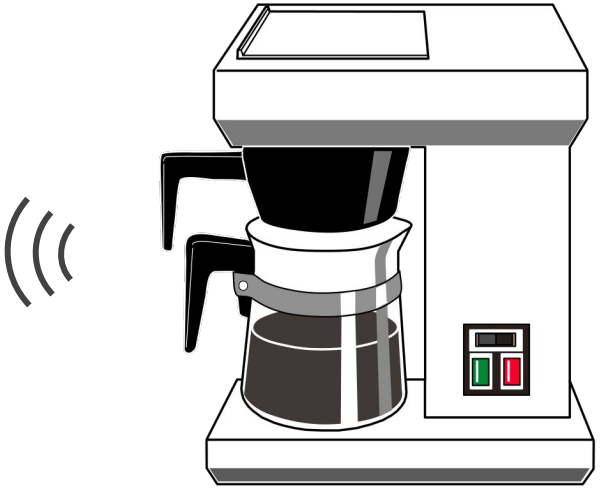
With learning rate α

Reward prediction error

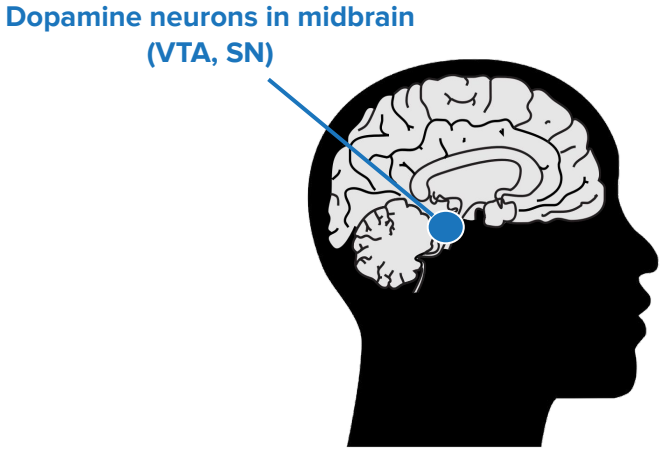
Actual reward = **Expected**

RPE = **0**

Reward prediction error

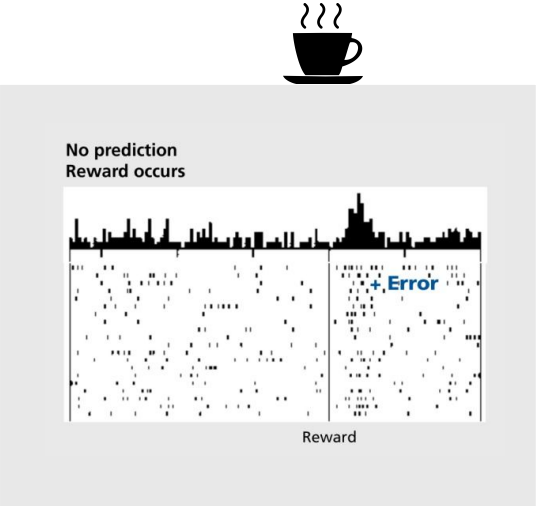


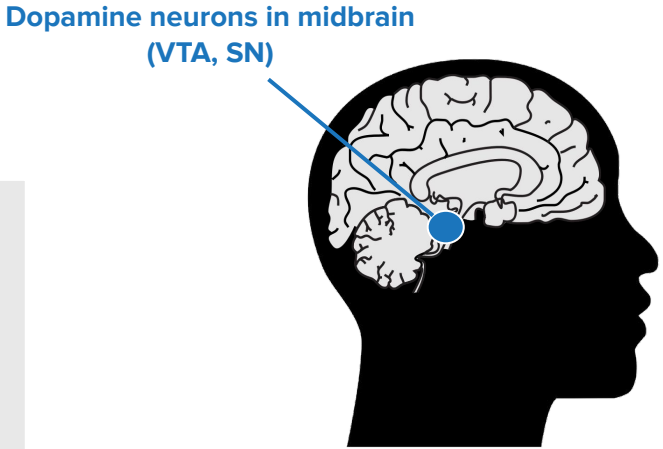
Reward prediction in the brain



Schultz, Dayan, Montague, 1997, Science.
Schultz, 2016, Dialogues Clin Neurosci.

Reward prediction in the brain

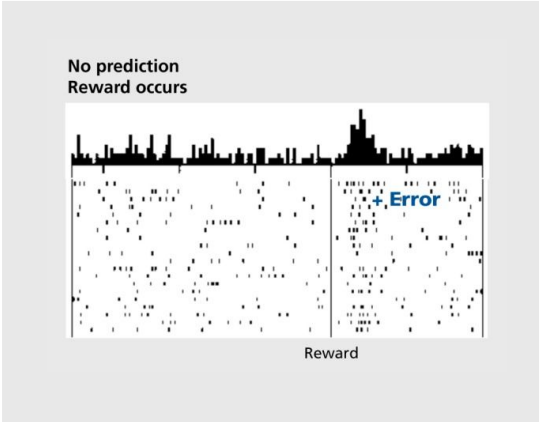




Schultz, Dayan, Montague, 1997, Science.
Schultz, 2016, Dialogues Clin Neurosci.

Reward prediction in the brain

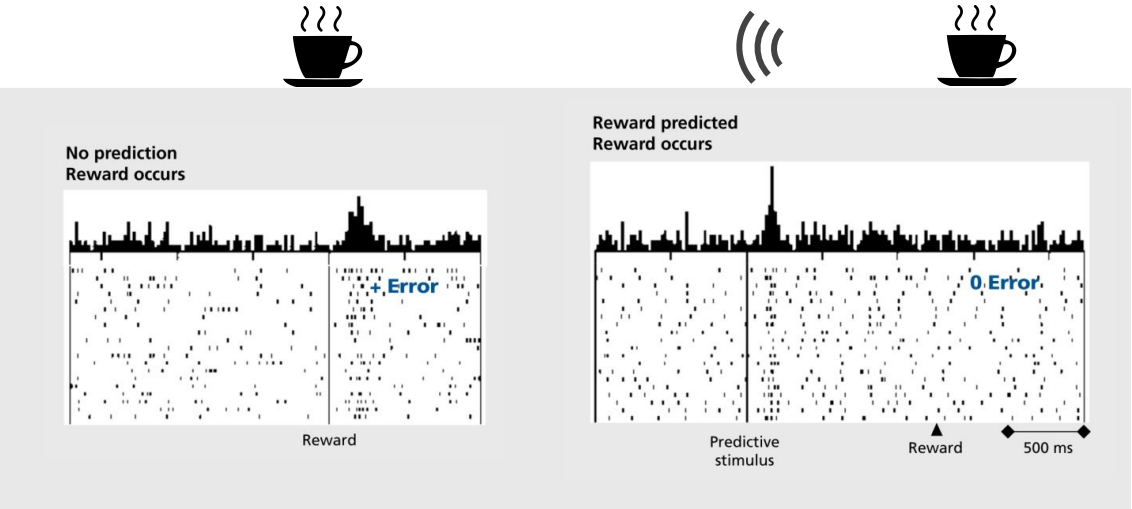
Before learning



Schultz, 2016, Dialogues Clin Neurosci.

Reward prediction in the brain

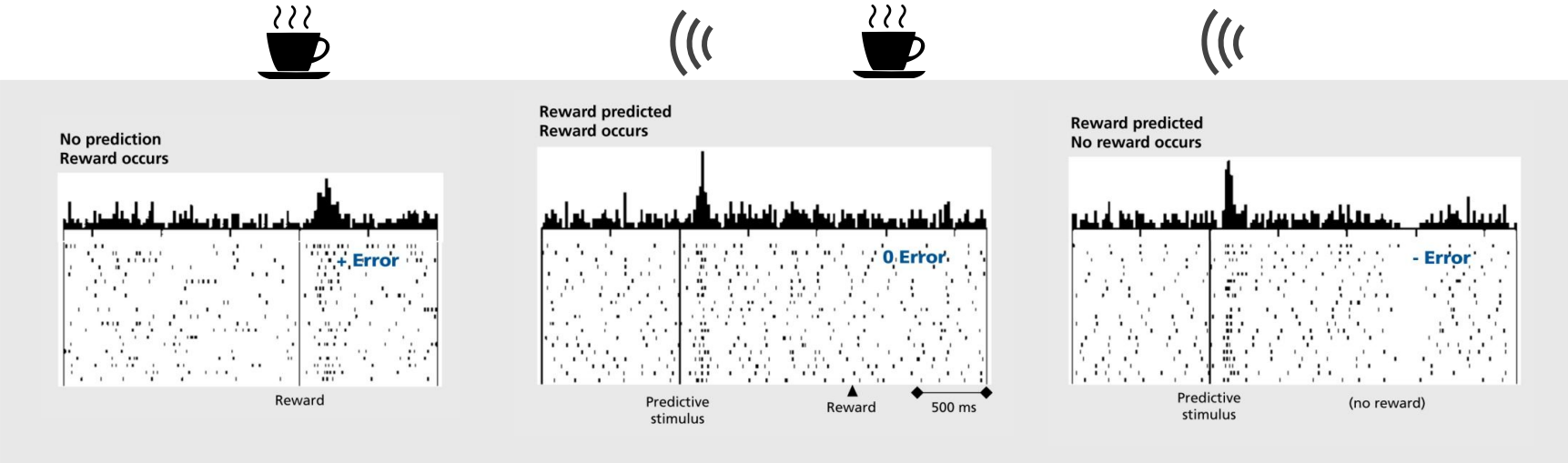
After learning



Schultz, 2016, Dialogues Clin Neurosci.

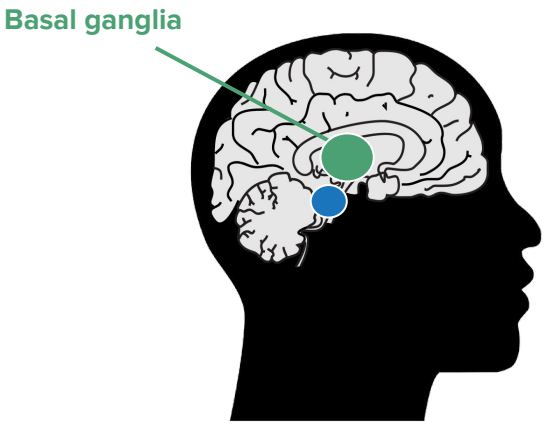
Reward prediction in the brain

Prediction incorrect - more learning



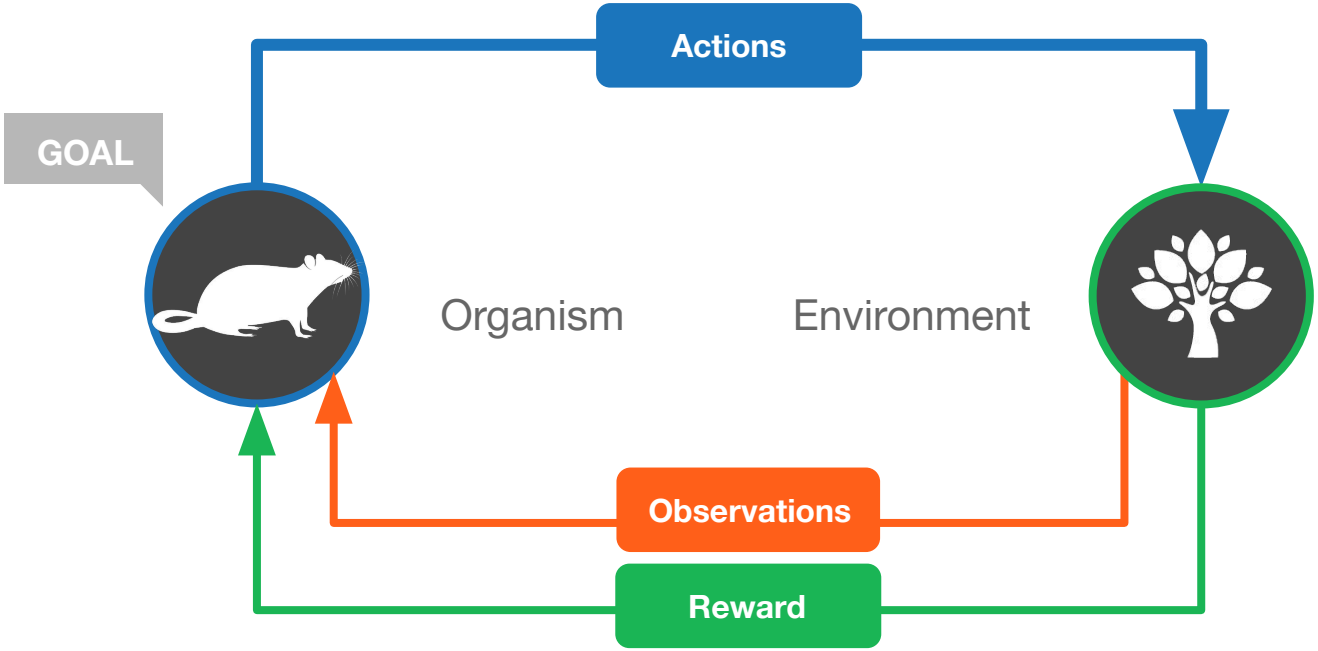
Schultz, 2016, Dialogues Clin Neurosci.

Control and action selection (instrumental conditioning)



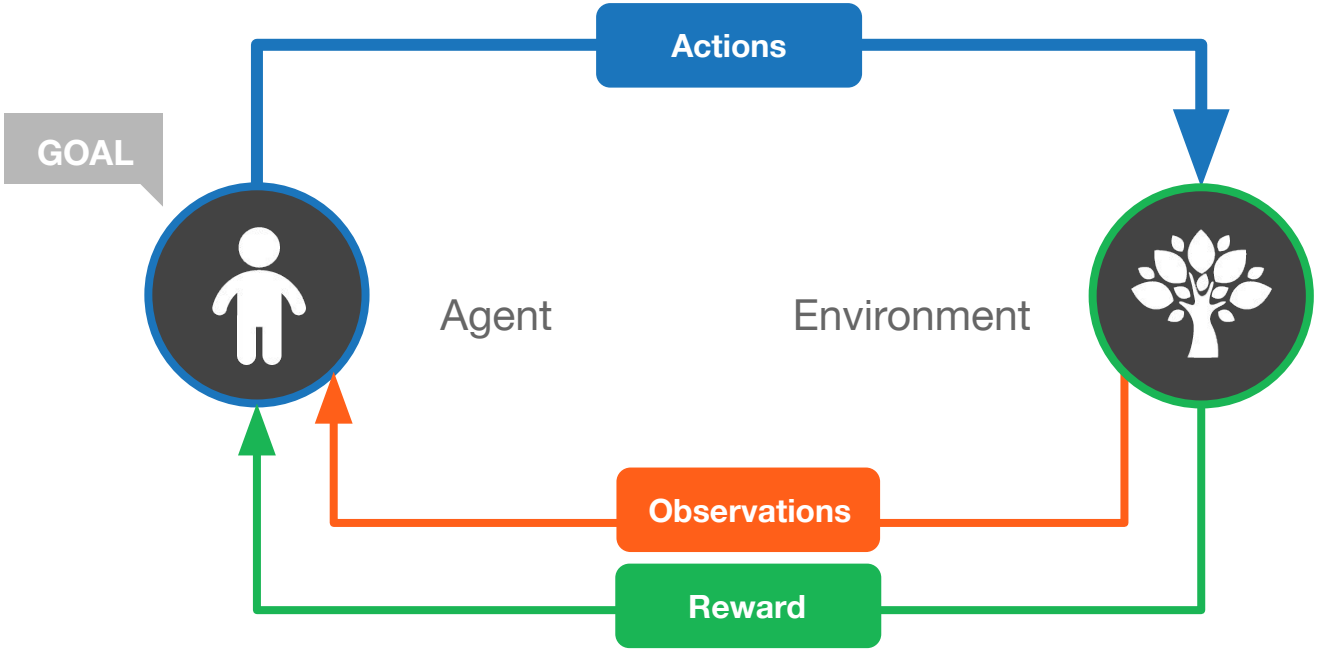
Wang et al, 2014, Neurobiol. Learn. Mem.

Reinforcement learning formalism



Learning to achieve **goals** via **trial and error** by **interacting** with an **environment**

Reinforcement learning in agents



Learning to achieve **goals** via **trial and error** by **interacting** with an **environment**

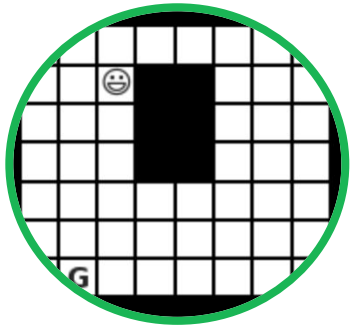
What is an environment?

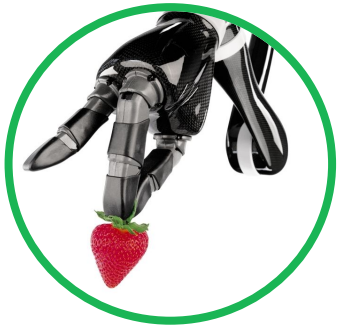
Environment receives **actions** and in turn provides **observations** and **rewards** to the agent.

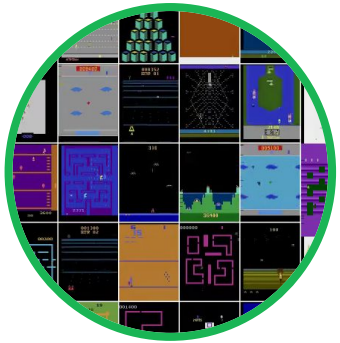
It can be stochastic, complex and not fully observable to the agent.

Learning to achieve **goals** via **trial and error** by **interacting** with an **environment**

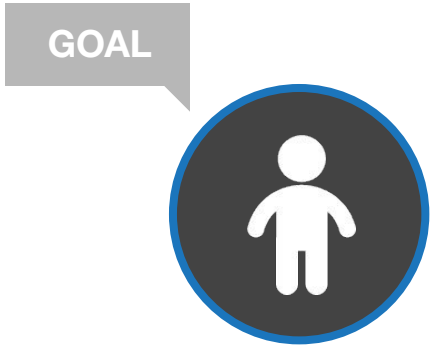
What is an environment?







What is an agent?

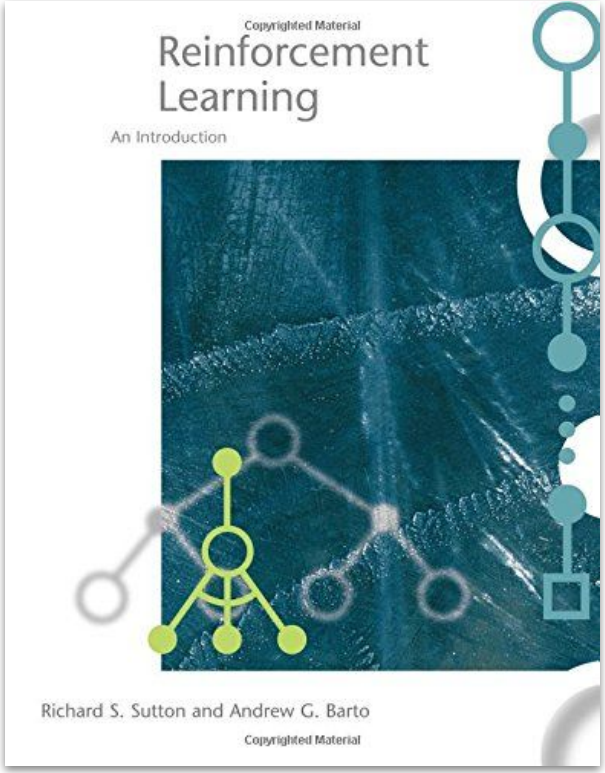


Agent learns a policy mapping **observations** to **actions**.

The agent's **goal** is to maximize its cumulative **reward**.

Learning to achieve **goals** via **trial and error** by **interacting** with an **environment**

Reinforcement Learning: An Introduction



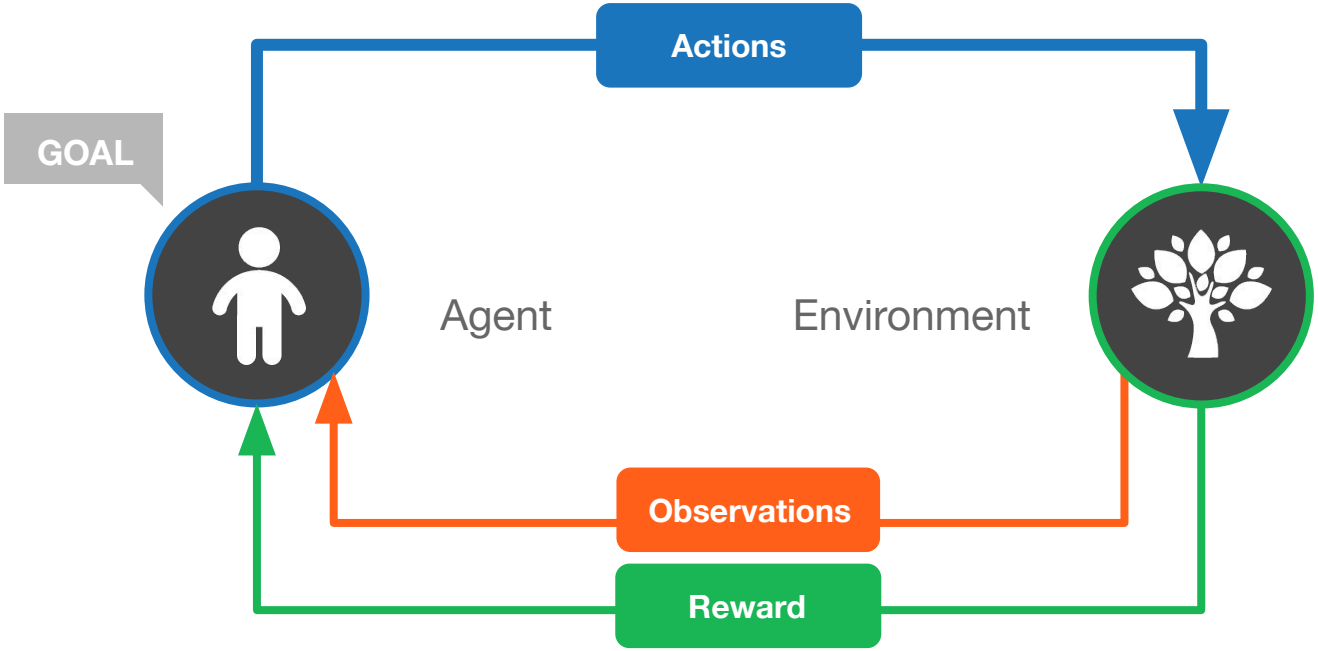
Section 2

General formulation and MDPs

Feryal Behbahani

neuromatch academy

Reinforcement learning in agents



Learning to achieve **goals** via **trial and error** by **interacting** with an **environment**

Where do rewards come from?

Reward is typically a scalar feedback signal.

- Extrinsic
  - Environmental
  - Typically hard-coded by the environment designer
- Intrinsic
  - Curiosity/Surprise/Novelty
  - Learning progress
  - Empowerment
  - Explanation
  - Compression

Why is RL hard?

RL = exploration to gather experience + learning from experience

Supervised Learning = ? + learning from experience

In **supervised learning**, data is often gathered and filtered by humans, which alleviates the **exploration problem**.

Why is RL hard?

- **Sequential** decision making
- **Exploration** might not be trivial
- **High-dimensional** continuous states & actions
- The **environment** is often **unknown**
- **Rewards** are **unknown**, **delayed** and possibly **sparse**
- **Non-stationarity**

Markov Decision Process (MDP)

Finite set of **states**

Transition function

Discount factor

$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

Finite set of **actions**

Reward function

We often solve the **RL problem** by modelling it as an **MDP.**

Markov Decision Process (MDP)

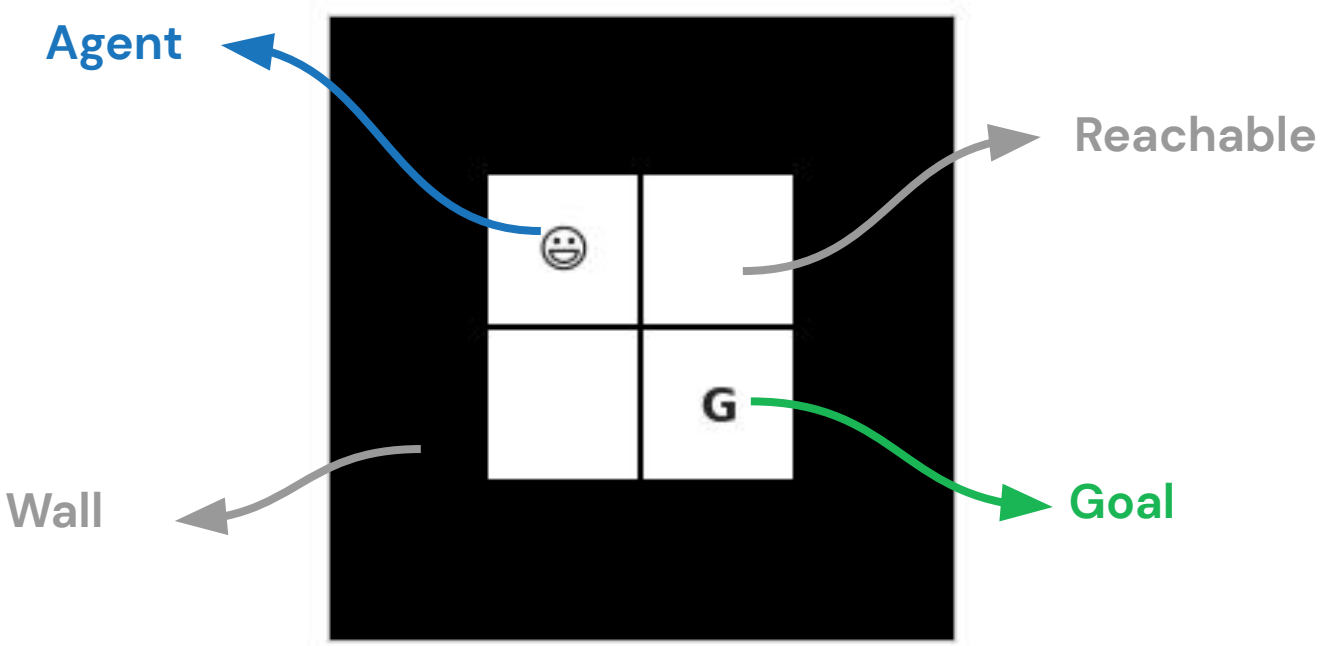
$$\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

Markov property: the next state depends only on the current state, and not all of the states that came before it.

$$\mathbf{P}(\underbrace{\mathbf{s_{t+1}}}_{\text{Future}} | \underbrace{\mathbf{s_t}, \mathbf{a_t}}_{\text{Present}}, \underbrace{\mathbf{s_{t-1}}, \mathbf{a_{t-1}}, \ldots, \mathbf{s_0}}_{\text{Past}}) = \mathbf{P}(\underbrace{\mathbf{s_{t+1}}}_{\text{Future}} | \underbrace{\mathbf{s_t}, \mathbf{a_t}}_{\text{Present}})$$

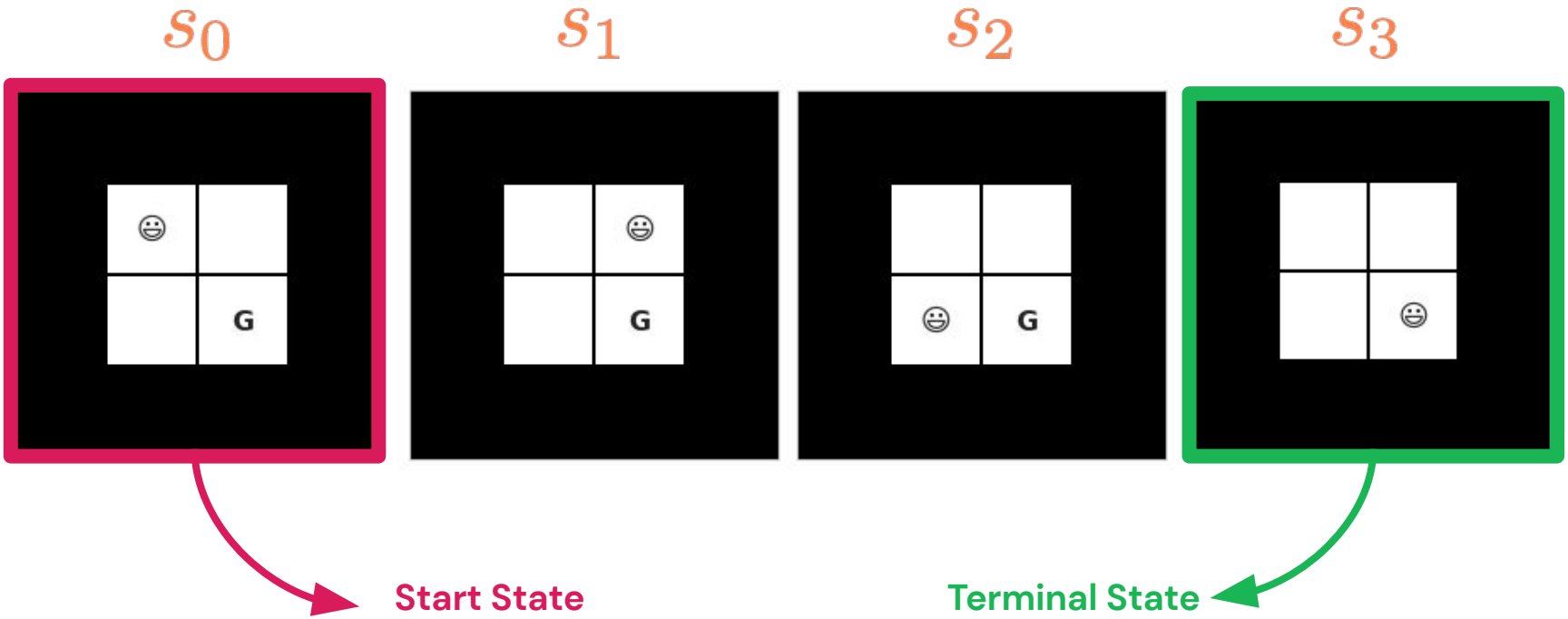
Grid World Environment

Size of the world: [2 X 2]

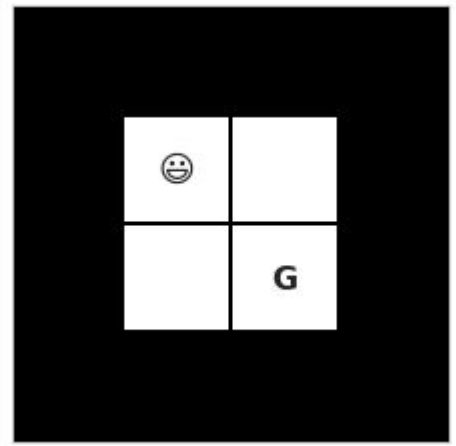


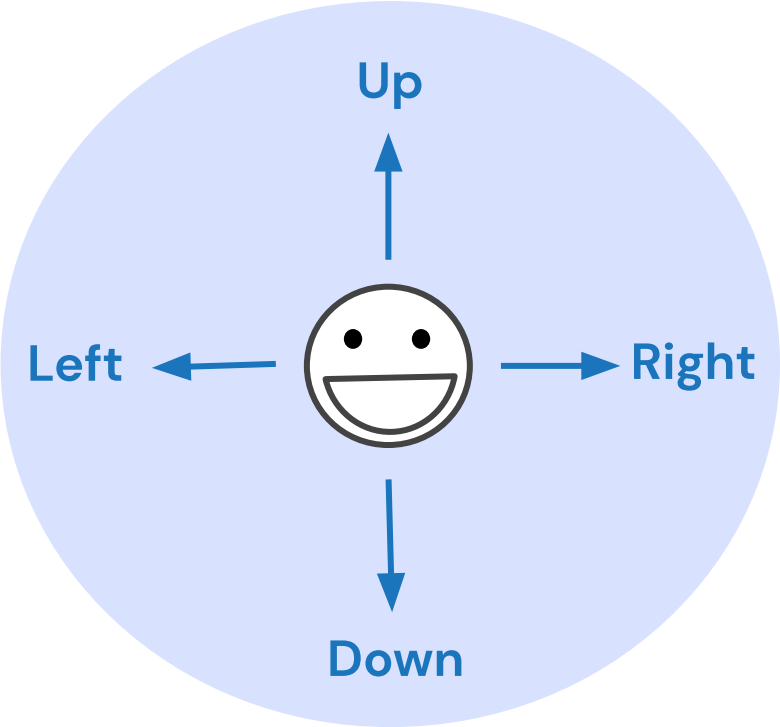
State space $\mathcal{S}$

There are **4 unique states** in this environment.

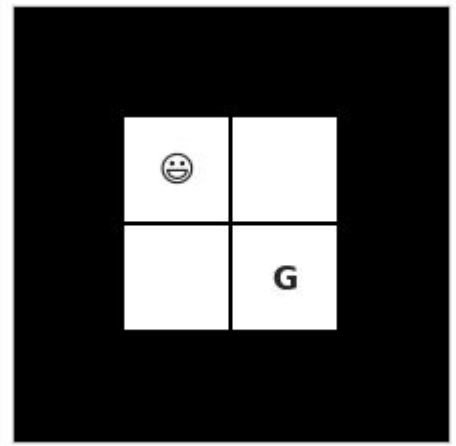


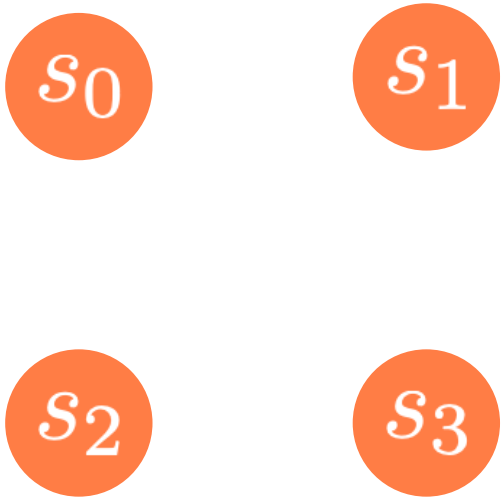
Action Space $\mathcal{A}$



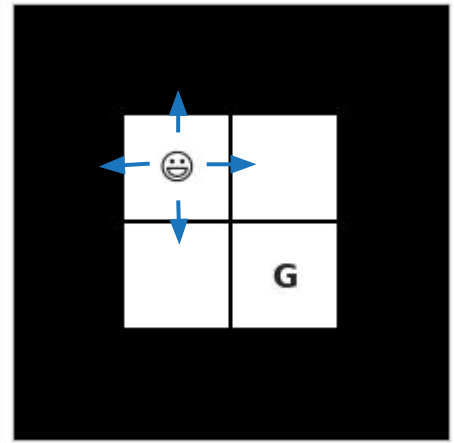


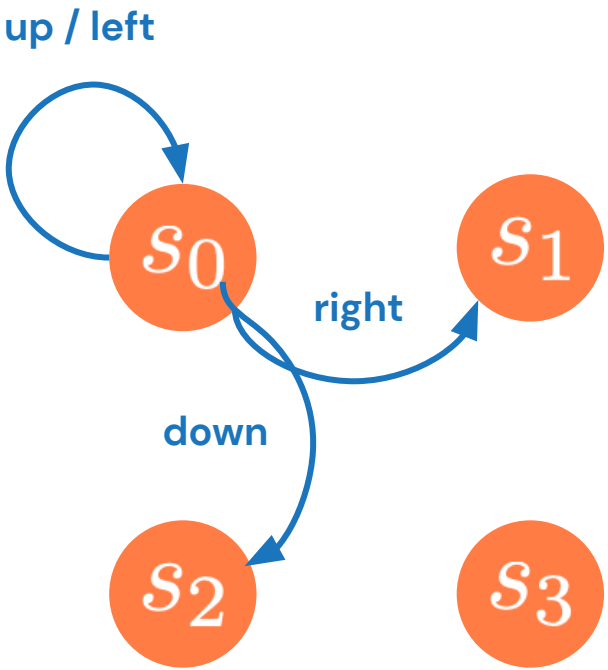
Transition model P



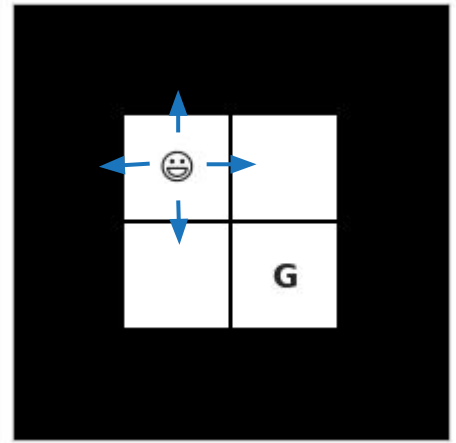


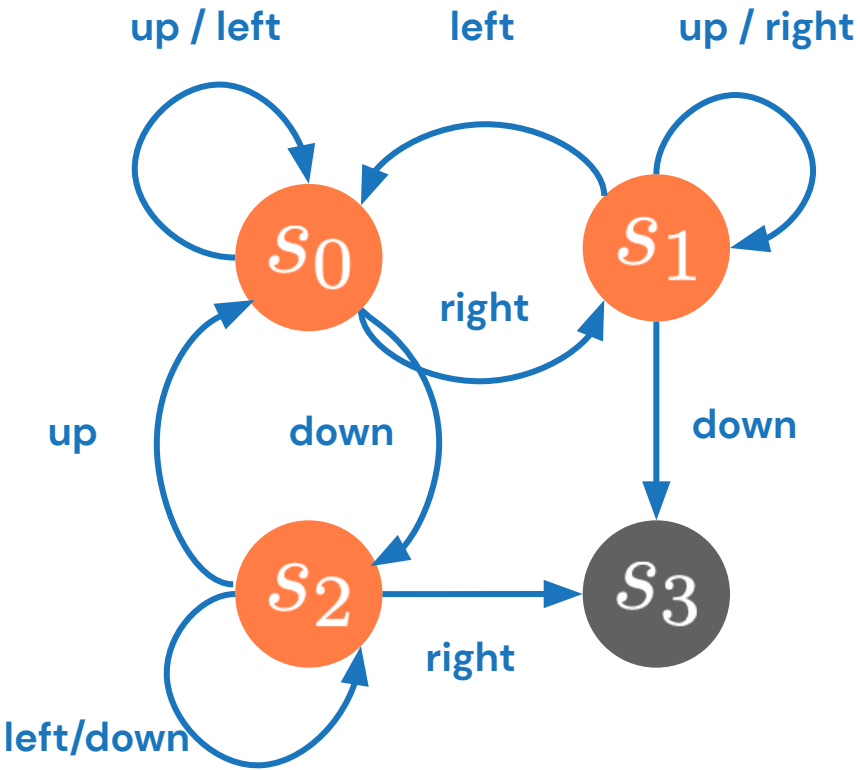
Transition model P





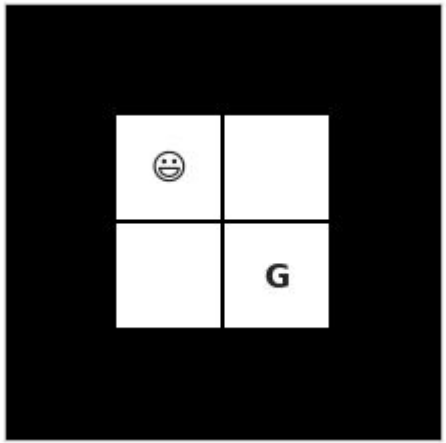
Transition model P

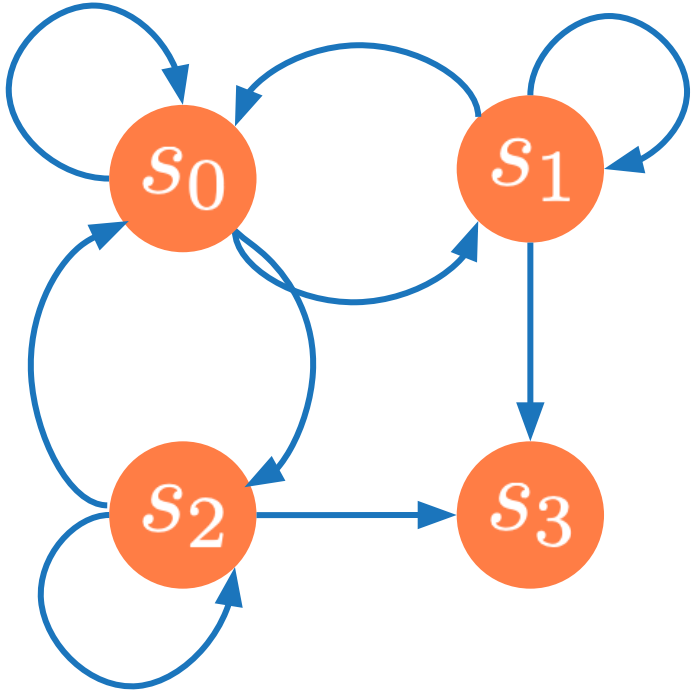




Rewards R

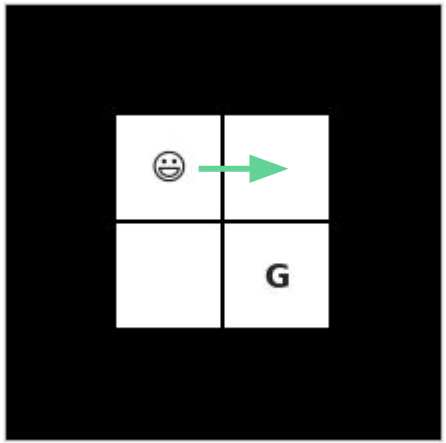
Empty cell $\mathbf{r} = 0$
Hit the wall $\mathbf{r} = -5$
Goal reached $\mathbf{r} = 10$

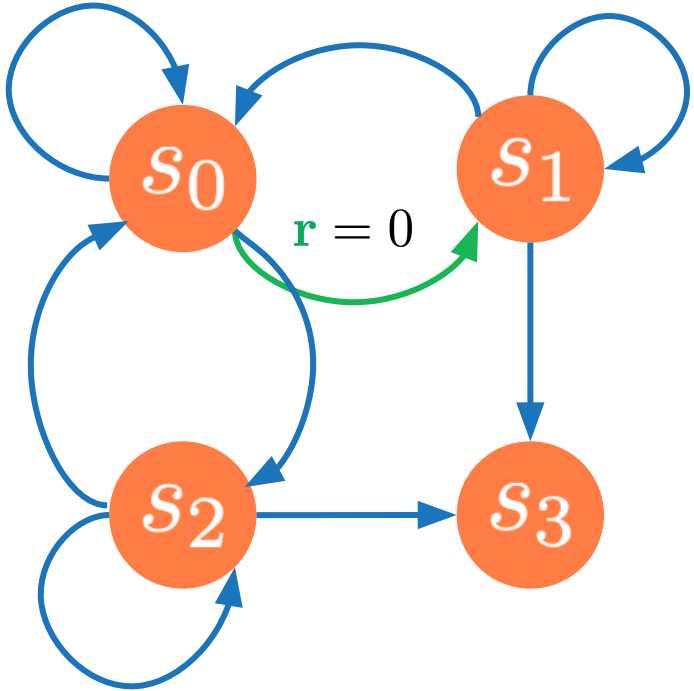




Rewards R

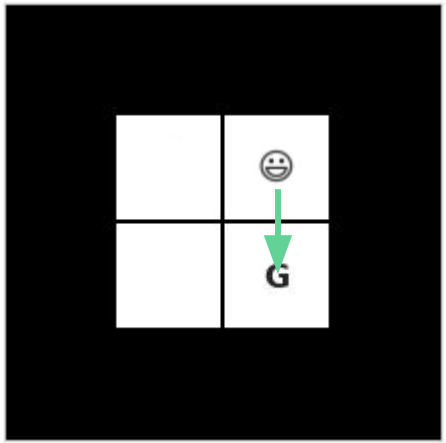
Empty cell $\mathbf{r} = 0$
Hit the wall $\mathbf{r} = -5$
Goal reached $\mathbf{r} = 10$

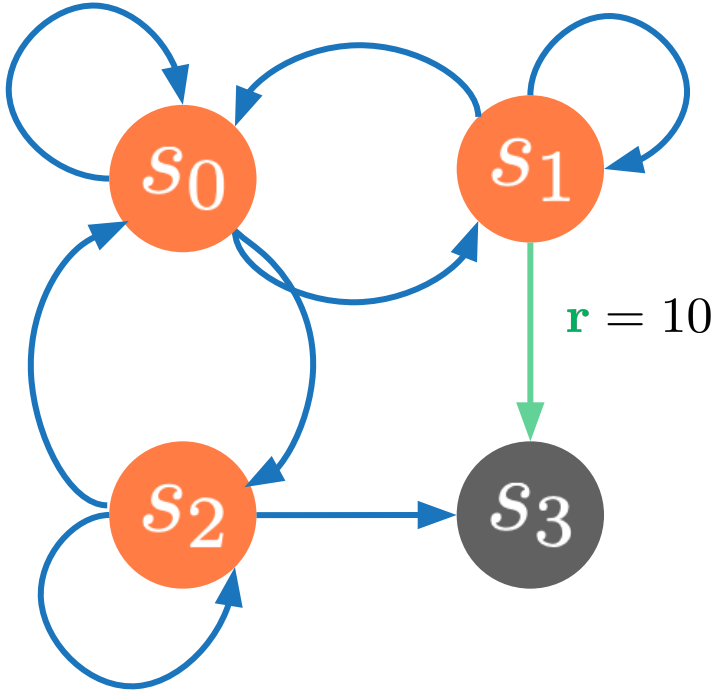




Rewards R

Empty cell $\mathbf{r} = 0$
Hit the wall $\mathbf{r} = -5$
Goal reached $\mathbf{r} = 10$





Agent's Goal

The agent's **goal** is to maximize the **discounted** **sum of future rewards**.

$$\mathbf{G_t} = \underbrace{\mathbf{r_t} + \gamma \mathbf{r_{t+1}} + \gamma^2 \mathbf{r_{t+2}} +}_{\textbf{Return}} \ldots$$

Discount γ specifies how much **future rewards** are worth compared to **immediate reward**.

Policy

Policy maps **states** to **actions**.

$$\mathbf{a_t} \sim \pi(\mathbf{a_t} | \mathbf{s_t})$$

Typically in RL we are searching for the **optimal policy**, which will maximize the **discounted** **sum of future rewards**.

Section 3

The Bellman equation

Feryal Behbahani

neuromatch academy

State Value Function

State value function maps **states** to **expected rewards**.

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t\right]$$

It is equal to **expected total discounted reward** for an agent starting from that **state** and following its **policy**. It specifies how **good** it is to be in a given **state**.

State-Action Value Function

State-action value function maps an **action** in a given **state** to **expected rewards**.

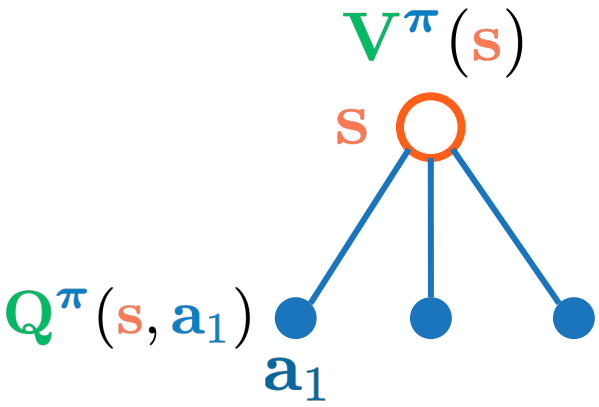
$$\mathbf{Q}^{\pi}(\mathbf{s_t}, \mathbf{a_t}) = \mathbb{E}_{\pi}\left[\mathbf{r_t} + \gamma \mathbf{r_{t+1}} + \gamma^2 \mathbf{r_{t+2}} + \ldots \middle| \mathbf{s_t}, \mathbf{a_t}\right]$$

It is equal to **expected total discounted reward** for an agent starting from state **s** and performing action **a** and following its **policy**.

State-Action Value vs State Value

The relationship between $\mathbf{V}^{\boldsymbol{\pi}}$ and $\mathbf{Q}^{\boldsymbol{\pi}}$

$$\mathbf{V}^{\boldsymbol{\pi}}(\mathbf{s}) = \sum_{\mathbf{a} \in \mathcal{A}} \boldsymbol{\pi}(\mathbf{a}|\mathbf{s}) \mathbf{Q}^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a})$$



Bellman Expectation Equation

Return at time t

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t\right]$$

Return at time t+1

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \ldots) | s_t\right]$$

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[r_t + \gamma V(s_{t+1}) | s_t\right]$$

Bellman Expectation Equation

Return at time t

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t\right]$$

Return at time t+1

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \ldots) | s_t\right]$$

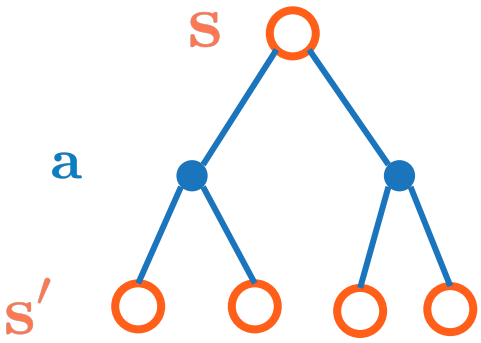
$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[r_t + \gamma V(s_{t+1}) | s_t\right]$$

The **state-action value function** can be similarly decomposed to:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\,\mathbb{E}_{s_{t+1}}\left[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t, a_t\right]$$

Bellman's principle of optimality

The **value** of a **state** under an **optimal policy** is equal to the **expected return** from the **best action** from that **state**.



$$\mathbf{V}^*(\mathbf{s}) = \max_{\mathbf{a}} \mathbf{Q}^*(\mathbf{s}, \mathbf{a})$$

$$\mathbf{Q}^*(\mathbf{s}, \mathbf{a}) = \mathbf{R}(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathbf{P}(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \max_{\mathbf{a}'} \mathbf{Q}^*(\mathbf{s}', \mathbf{a}')$$

Optimal policy

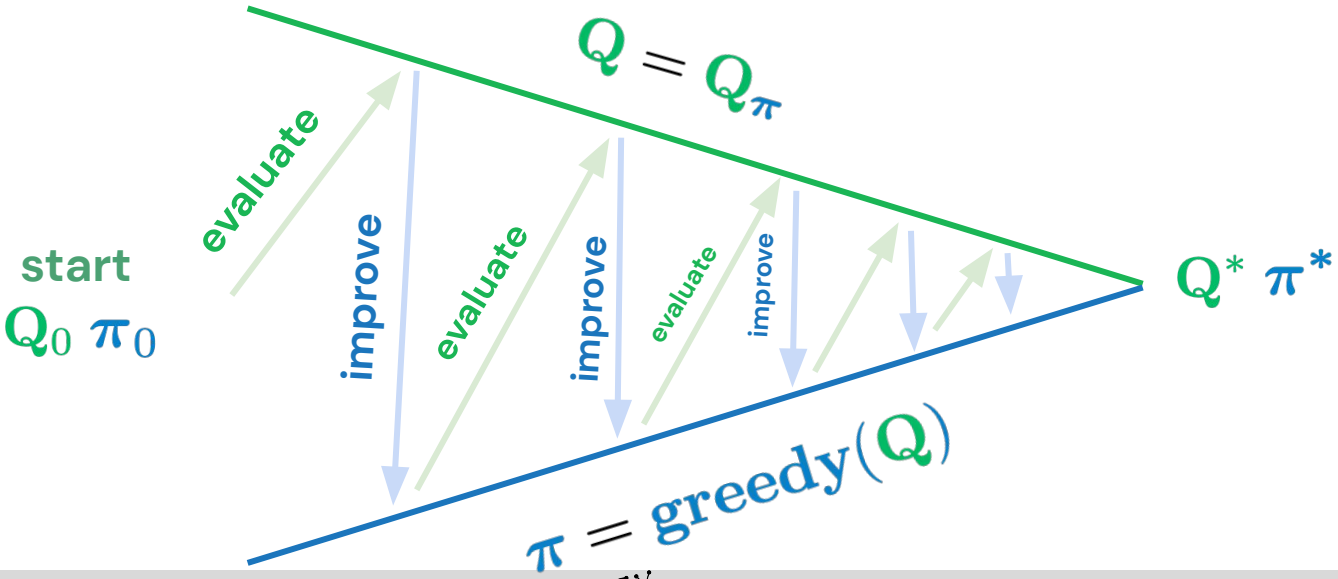
An **optimal policy** can be found by maximising over the **optimal state-action value function**. If we know the $\mathbf{Q}^*(\mathbf{s}, \mathbf{a})$ we immediately have the optimal policy $\boldsymbol{\pi}^*(\mathbf{s})$.

$$\boldsymbol{\pi}^*(\mathbf{s}) = \max_{\mathbf{a}} \mathbf{Q}^*(\mathbf{s}, \mathbf{a})$$

We say that the **optimal policy** is **greedy** with respect to the **optimal value function**.

How to find the optimal policy? (Policy Iteration)

We can start from a random policy and random value function and find the optimal policy and value function through the following iterative process:



Why does this work? (Policy Iteration Theorem)

Given the **true** **value function** $Q_\pi(s, a)$ for any **policy** π for $s \in \mathcal{S}$ and $a \in \mathcal{A}$ it can always be **greedified** to obtain a **better policy**:

$$\pi' = \arg\max_a Q^\pi(s, a)$$

Where "**better**" means:

$$Q^{\pi'}(s, a) \geq Q^\pi(s, a)$$

Equality happens when we find the **optimal** **value function** and **policy**.

Section 4

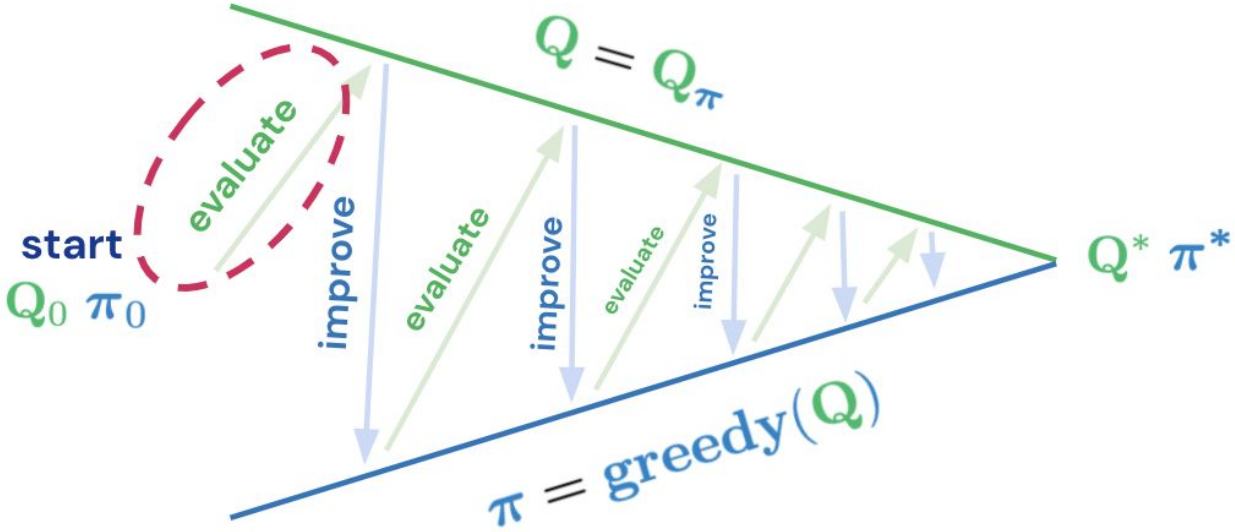
Policy evaluation

Jane Wang

neuromatch academy

Iterative policy evaluation

During policy evaluation we are making the value function consistent with the current policy, also through an **iterative process**.



Iterative policy evaluation

Evaluate a given policy π iteratively to find its **true value function**:

$$\mathbf{Q}_0 \rightarrow \mathbf{Q}_1 \rightarrow \ldots \rightarrow \mathbf{Q}^{\pi}$$

At each iteration **k:**

For all states $\mathbf{s} \in \mathcal{S}$ and $\mathbf{a} \in \mathcal{A}$

Update $\mathbf{Q}_k(\mathbf{s}, \mathbf{a})$ from $\mathbf{Q}_{k-1}(\mathbf{s}', \mathbf{a}')$

Model-based policy evaluation: **Dynamic Programming**

Dynamic programming finds the **optimal policy** given a perfect **model of the environment**.

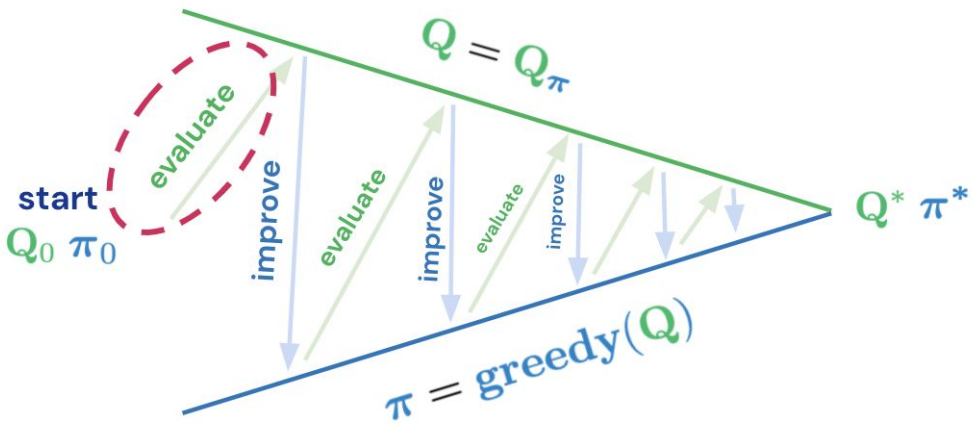
DP algorithms turn Bellman equations into update rules and iterate until convergence:

$$\mathbf{Q}_k(\mathbf{s}, \mathbf{a}) \leftarrow \mathbf{R}(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathbf{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \sum_{\mathbf{a}' \in \mathcal{A}} \boldsymbol{\pi}_{k-1}(\mathbf{a}'|\mathbf{s}') \mathbf{Q}_{k-1}(\mathbf{s}', \mathbf{a}')$$

Policy Iteration

We can start from a random policy and random value function and find the optimal policy and value function through the following iterative process:

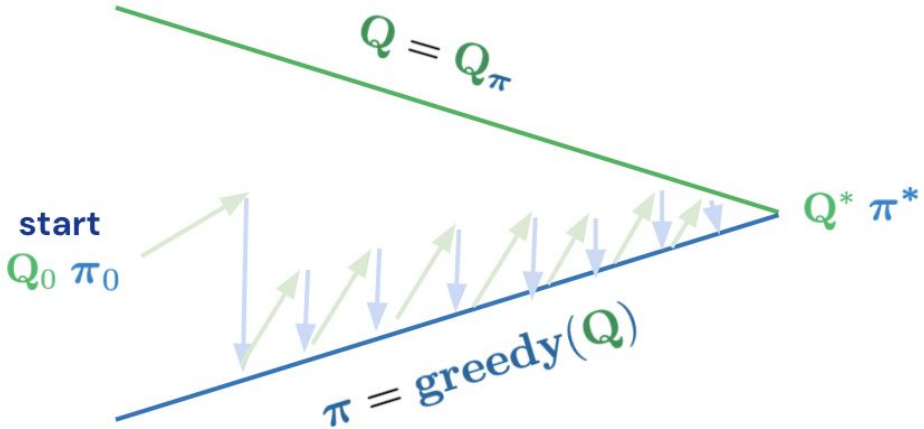
$$\mathbf{Q}_k(\mathbf{s},\mathbf{a}) \leftarrow \mathbf{R}(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathbf{P}(\mathbf{s}'|\mathbf{s},\mathbf{a}) \sum_{\mathbf{a}' \in \mathcal{A}} \boldsymbol{\pi}_{k-1}(\mathbf{a}'|\mathbf{s}')\mathbf{Q}_{k-1}(\mathbf{s}',\mathbf{a}')$$



Value Iteration

We can iteratively improve our estimation of the value function by directly using the **Bellman optimality equation**.

$$\mathbf{Q}_k(\mathbf{s},\mathbf{a}) \leftarrow \mathbf{R}(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathbf{P}(\mathbf{s}'|\mathbf{s},\mathbf{a}) \max_{\mathbf{a}'} \mathbf{Q}_{k-1}(\mathbf{s}',\mathbf{a}')$$



Model-free policy evaluation

If we **do not have access to the true model of the environment**, we can instead learn directly from episodes of experience.

- **Monte Carlo Methods**
- **Temporal Difference Learning**

Monte Carlo Policy Evaluation

The goal is to update the value function towards the **actual return**.

We have to wait until the end of the episode to start learning.

$$\mathbf{Q}_k(\mathbf{s_t}, \mathbf{a_t}) \leftarrow \mathbf{Q}_{k-1}(\mathbf{s_t}, \mathbf{a_t}) + \overset{\text{Learning rate}}{\alpha} \big(\underbrace{\mathbf{G_t}}_{\text{TD Target}} - \mathbf{Q}_{k-1}(\mathbf{s_t}, \mathbf{a_t})\big)$$

Reminder $\mathbf{G_t} = \mathbf{r_t} + \gamma \mathbf{r_{t+1}} + \ldots + \gamma^{T-1} \mathbf{r}_T$

Monte Carlo Policy Evaluation

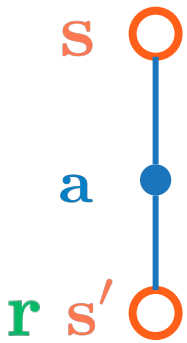
The goal is to update the value function towards the **actual return**.

We have to wait until the end of the episode to start learning.

$$\mathbf{Q}_k(\mathbf{s_t}, \mathbf{a_t}) \leftarrow \mathbf{Q}_{k-1}(\mathbf{s_t}, \mathbf{a_t}) + \alpha \overbrace{(\underbrace{\mathbf{G_t}}_{\text{TD Target}} - \mathbf{Q}_{k-1}(\mathbf{s_t}, \mathbf{a_t}))}^{\text{RPE}}$$

Reminder $\mathbf{G_t} = \mathbf{r_t} + \gamma \mathbf{r_{t+1}} + \ldots + \gamma^{T-1} \mathbf{r}_T$

Temporal Difference Learning



Instead of waiting until the end of the episode, TD can learn from incomplete episodes via **bootstrapping**, updating a guess from a guess.

$$\mathbf{Q}_k(\mathbf{s},\mathbf{a}) \leftarrow \mathbf{Q}_{k-1}(\mathbf{s},\mathbf{a}) + \alpha\,\overbrace{\Big(\underbrace{\mathbf{r} + \mathbf{Q}_{k-1}(\mathbf{s}',\boldsymbol{\pi}(\mathbf{s}'))}_{\text{TD Target}} - \mathbf{Q}_{k-1}(\mathbf{s},\mathbf{a})\Big)}^{\text{RPE}}$$

TD vs MC

- TD learns **after each step** while MC has to wait for the **end of episode** when **return** is known.
- TD can learn in **non-episodic** environments while MC cannot.

Bias/Variance Tradeoff

- **Return** is an **unbiased estimation** of the value function while TD target is a **biased estimation** (a guess from a guess).
- TD target is much **lower variance** than the **return**
  - **Return** depends on sequences of random **actions**, **transitions**, and **rewards**.
  - TD target depends on **one random action**.

Section 5

Model-free learning

Jane Wang

neuromatch academy

Model-free policy evaluation

If we **do not have access to the true model of the environment**, we can instead learn directly from episodes of experience.

- **Monte Carlo Methods**
- **Temporal Difference Learning**

Model-free learning

When we don't have access to the model of environment, we learned how to do **model-free evaluation**, now we focus on **model-free learning** where we want to learn to optimise the **value function** of an **unknown MDP**.

On-policy learning "Learning on the job"

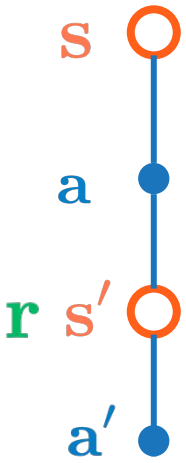
Off-policy learning "Look over someone's shoulder"

Exploration

- **ε-Greedy** is a simple **policy improvement** method that allows for continual exploration.
- **Exploit:** with probability 1-ε we select a **greedy action** wrt the **value function**.
- **Explore:** with probability ε a **random action** is uniformly selected instead.

SARSA: On-policy TD Control

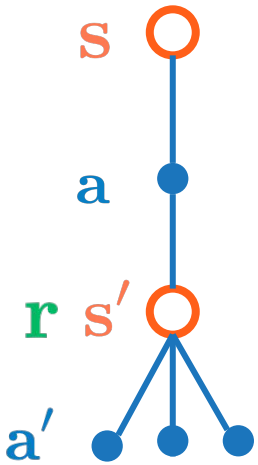
- **SARSA** learns the **policy** by executing it in the environment to gather experiences.
- We often use **ε-Greedy policy** to gather experiences and each update needs access to $(\mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}', \mathbf{a}')$.



$$\mathbf{Q}(\mathbf{s}, \mathbf{a}) \leftarrow \mathbf{Q}(\mathbf{s}, \mathbf{a}) + \alpha(\mathbf{r} + \gamma \mathbf{Q}(\mathbf{s}', \mathbf{a}') - \mathbf{Q}(\mathbf{s}, \mathbf{a}))$$

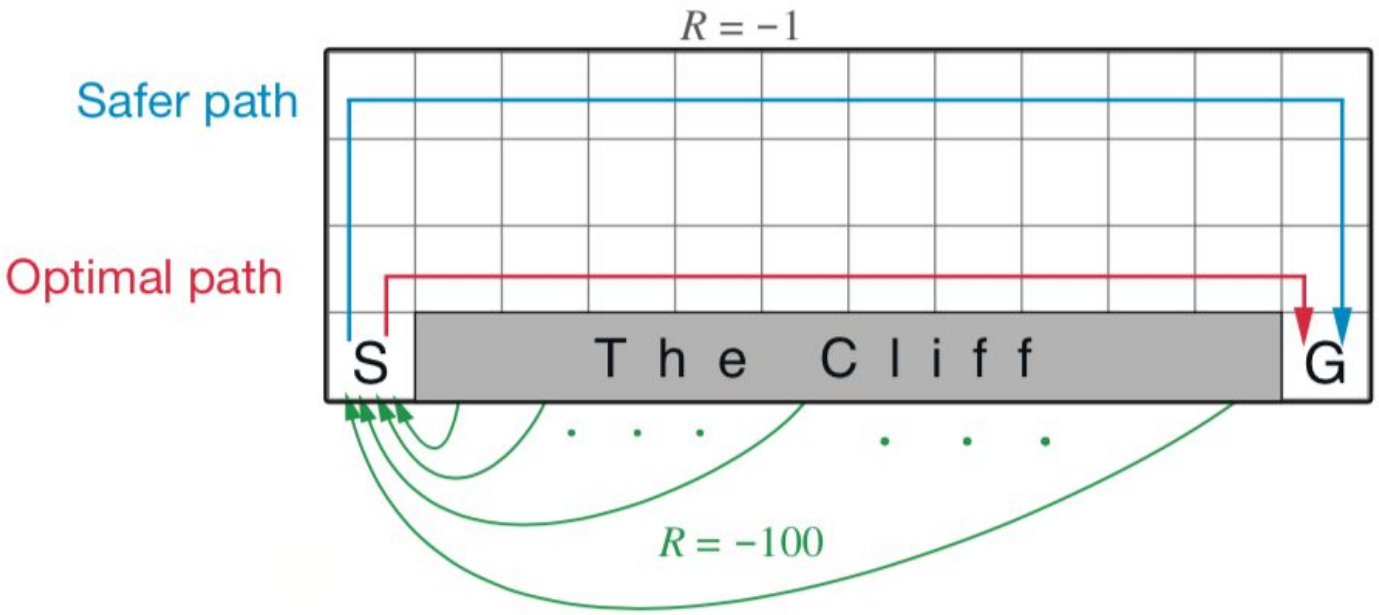
Q Learning: Off-policy TD Control

- **Q-learning** learns about the **value** of its **greedy policy** from experiences gathered by a **different policy** (e.g. **ε-Greedy**).
- We use **max** of the Q-value for $\mathbf{a}'$ instead of executing the policy for an extra step.



$$\mathbf{Q}(\mathbf{s},\mathbf{a}) \leftarrow \mathbf{Q}(\mathbf{s},\mathbf{a}) + \alpha(\mathbf{r} + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{s}',\mathbf{a}') - \mathbf{Q}(\mathbf{s},\mathbf{a}))$$

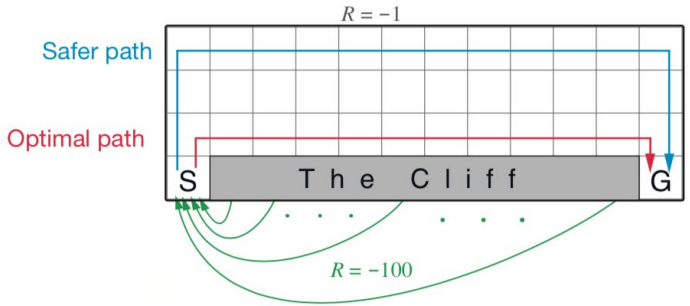
SARSA vs Q-Learning: The Cliff-walking Example

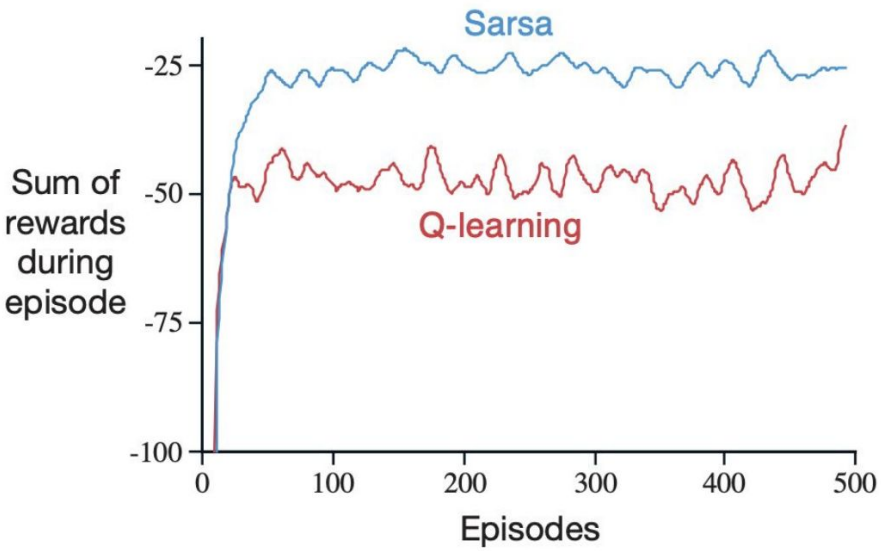


Sutton & Barto. Reinforcement Learning: An Introduction. (Chapter 6)

SARSA vs Q-Learning: The Cliff-walking Example

- **Q-learning** learns the **optimal path** while its online performance is worse than **SARSA**.
- **SARSA** learns the **safer path**.





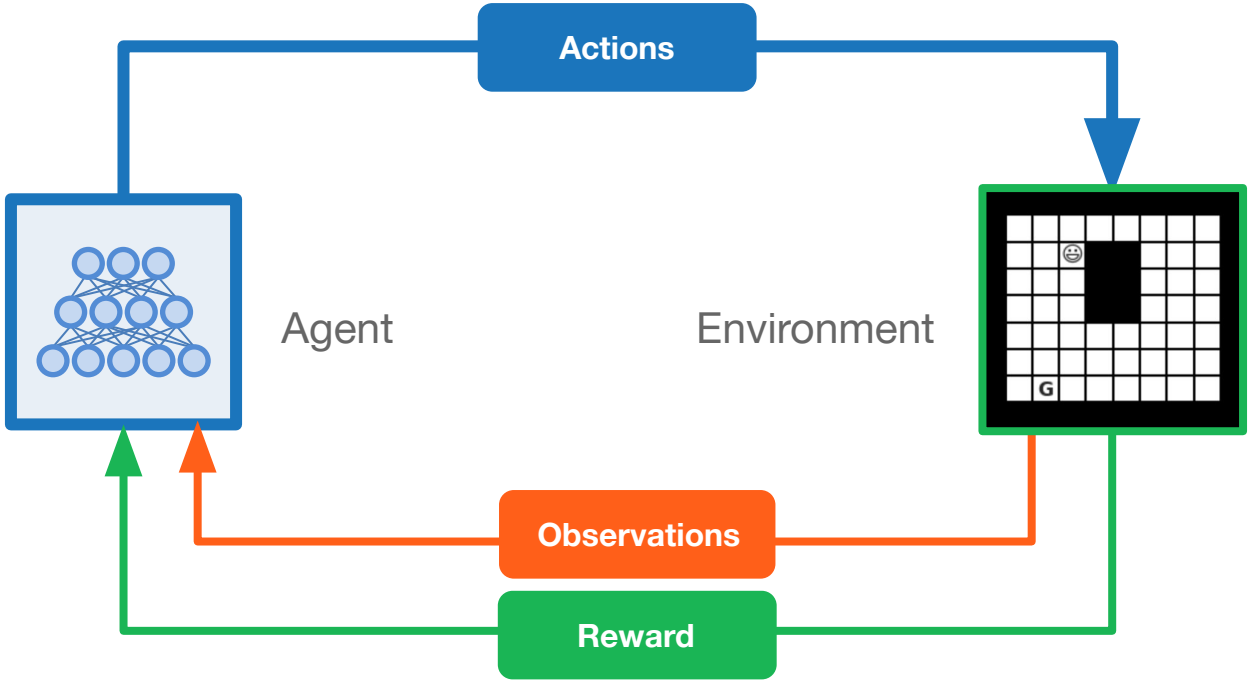
Section 6

Function approximation

Feryal Behbahani

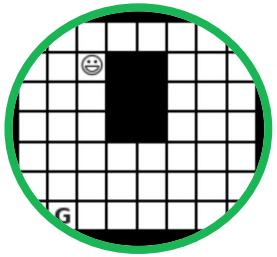
neuromatch academy

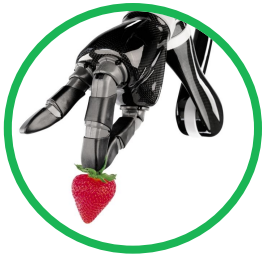
Function approximation



Why function approximation?

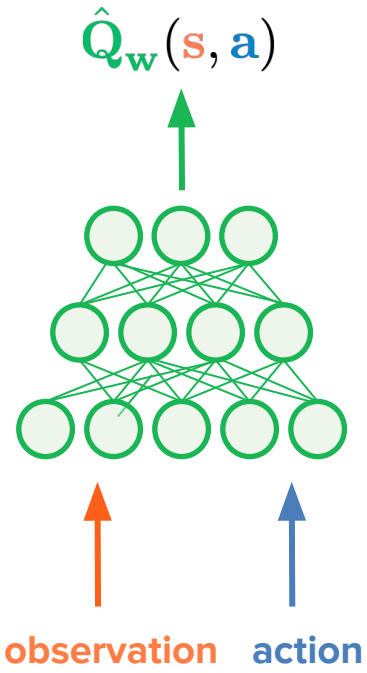
- Up to now we used lookup tables to represent the **value function**, for example each $s \in \mathcal{S}$ and $a \in \mathcal{A}$ has an entry in the **Q-table** $Q(s, a)$
- In **large MDP**s with large **action** and **state** space this becomes infeasible due to **memory requirements** and **inefficiency of learning** the **value** of each **state** in isolation.





How?

- We now have a great machinery to approximate any continuous function efficiently: **neural networks**!
- Given a **state**, use a neural network with parameters $\mathbf{w}$ to predict the **Q-values** of each **action**.
- Allows to **generalize** from **seen states** to **unseen states**.



How?

- We will use the same **TD targets** we saw earlier as a **loss function**.
- Our goal is to find the parameter vector $\mathbf{w}$ minimising the **mean-squared error** between the **approximate value function** $\hat{Q}_{\mathbf{w}}(\mathbf{s}, \mathbf{a})$ and the **true value function.**
- This can be optimized with gradient descent, using automatic differentiation.

The deadly triad

Bad news: there is possibility of **divergence** when we combine these three powerful ingredients:

Off-policy learning

Learning from experiences collected by a different behaviour policy than the target policy we are learning about (e.g. Q-learning).

+ Bootstrapping

Learning value estimates from other value estimates (e.g. in TD and DP)

+ Function approximation

Significantly generalising from experiences

Section 9

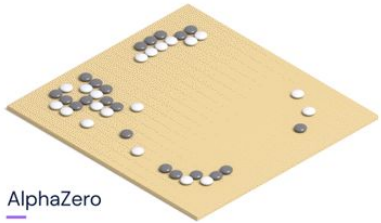
Real-world applications and ethics

Jane Wang

neuromatch academy

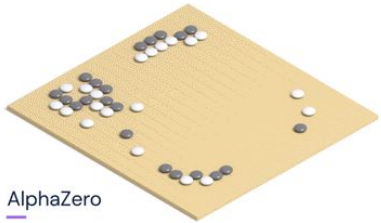
Real-world Reinforcement Learning: **Examples**

Game playing
(more on this tomorrow)



Real-world Reinforcement Learning: **Examples**

Game playing
(more on this tomorrow)

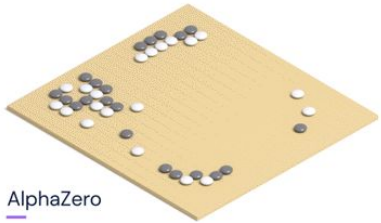


Robotics / Manipulation

Self-driving cars

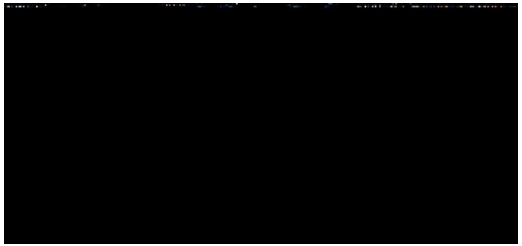
Real-world Reinforcement Learning: **Examples**

Game playing
(more on this tomorrow)



Robotics / Manipulation

User personalization



Self-driving cars

Managing energy usage

Real-world Reinforcement Learning: **Challenges**

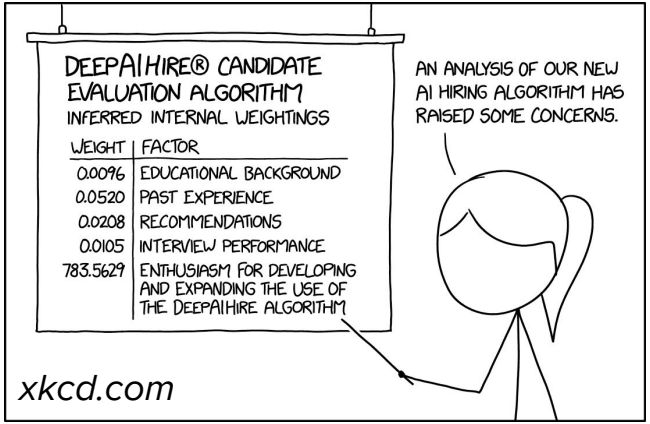
- Learning difficulty:
  - Large action and state spaces
  - Smart / continuing **exploration**
  - Requirements for **fast inference** / Generalization

Real-world Reinforcement Learning: **Challenges**

- Learning difficulty:
  - Large action and state spaces
  - Smart / continuing **exploration**
  - Requirements for **fast inference** / Generalization
- Safety concerns
  - Usually interaction with the environment is **expensive** e.g. robotics
  - Large and unknown delays in actuators, sensors and rewards
  - Reward function is **unspecified**, **risk sensitive** or **multi-objective**

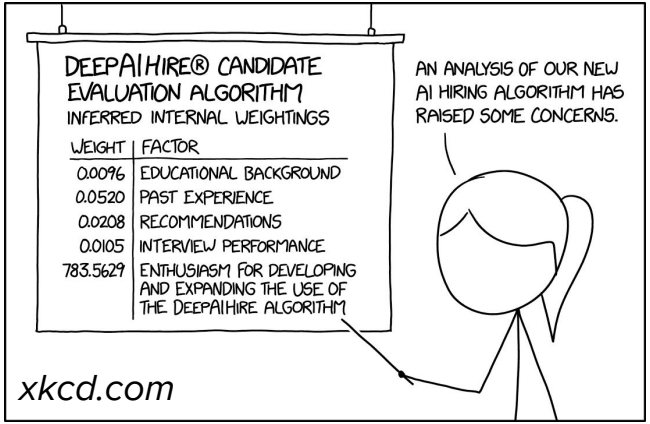
To find out more

- **Safety:** Victoria Krakovna's list of resources (vkrakovna.wordpress.com/)



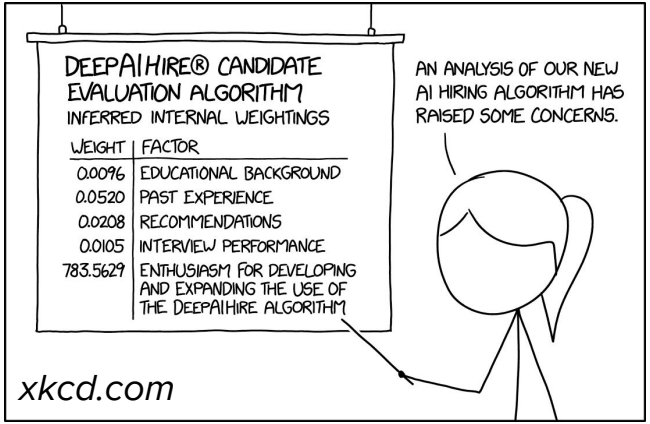
To find out more

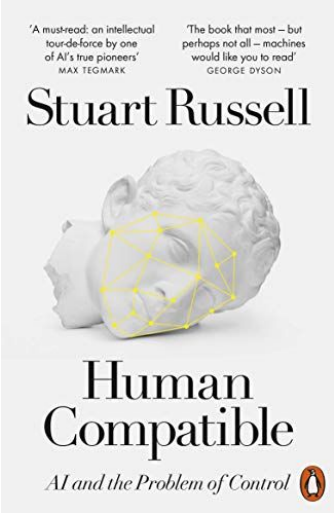
- **Safety:** Victoria Krakovna's list of resources (vkrakovna.wordpress.com/)
- **Fairness, ethics:** Timnit Gebru, Joy Buolamwini (Gender Shades)



To find out more

- **Safety:** Victoria Krakovna's list of resources (vkrakovna.wordpress.com/)
- **Fairness, ethics:** Timnit Gebru, Joy Buolamwini (Gender Shades)
- **Beneficial AI:** Stuart Russell (Human Compatible)





Ethics - discuss with a group partner or partners (3 min each):

- **Safety:** What are some safety issues that arise in RL that don't arise with e.g. supervised learning?
- **Generalization:** How should we deal with situations where our agents "go out of distribution"?
- How important do you think **interpretability** is in the ethical and safe deployment of RL agents in the real world?

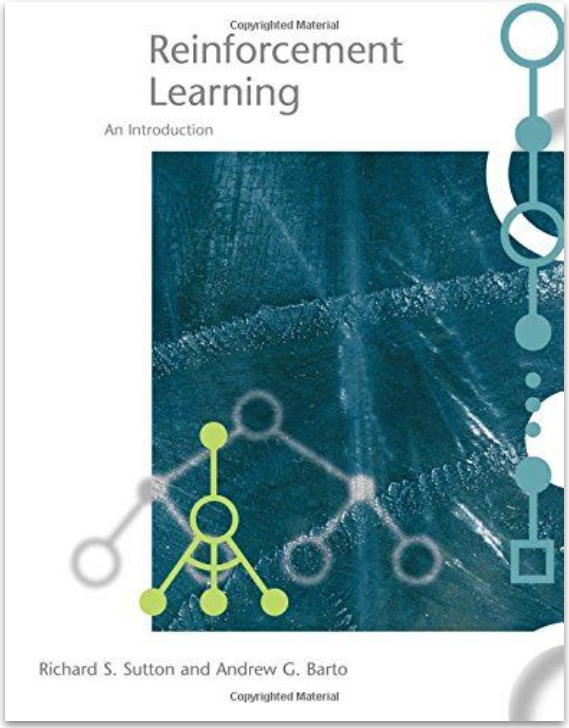
Section 10

How to learn more?

Feryal Behbahani

neuromatch academy

Reinforcement Learning: An Introduction



Want to learn more?

Books

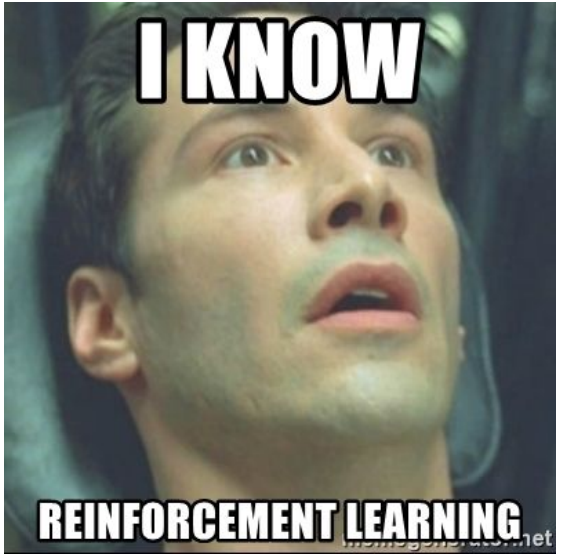
- Reinforcement Learning: an Introduction by Sutton & Barto
- Algorithms for Reinforcement Learning by Csaba Szepesvari

Lectures and course

- RL Course by David Silver
- Reinforcement Learning Course | UCL & DeepMind
- Emma Brunskill Stanford RL Course
- RL Course on Coursera by Martha White & Adam White

More practical

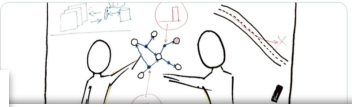
- Spinning Up in Deep RL by Josh Achiam
- Acme white paper & Colab tutorial



David Abel @dabelcs · Jun 22
Replying to @FeryalMP
I wrote a simple python library during my time at Brown that I've used in courses before, this could be helpful? Here's the GitHub:

david-abel/
simple_rl

Arjun Chandra 🇮🇳
@boelger
Replying to @boelger and @FeryalMP
I do also run a reading group, in case this can be of some use for someone:



Xavier Giró
@DocXavi
Replying to @FeryalMP
Our course on Deep & Reinforcement Learning at @UPCTelecos Barcelona:

telecombcn-dl.github.io/drl-2020/

Robin Ranjit Singh Chauhan @robinc · Jun 22
Replying to @FeryalMP

TalkRL: The Reinforcement Learning Podcast on Apple Podcasts
Technology · 2021
podcasts.apple.com

Johan S. Obando 👍
@JS_Obando
Replying to @FeryalMP
I highly recommend the Revisiting Rainbow 🌈 paper and its implementation. 📚🤩
📜Paper: arxiv.org/abs/2011.14826
🐙Code:

JohanSamir/
revisiting_rainbow
Revisiting Rainbow

2 Contributors · 0 Issues · 45 Stars · 1 Fork

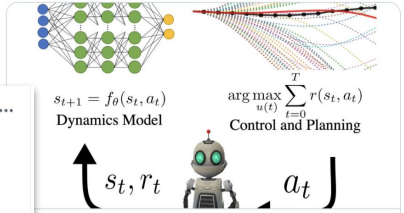
Feryal
@FeryalMP

If you have used or made any tutorials/videos/blog posts/open-source projects/libraries that helps with learning about 🧀 Reinforcement Learning 🐁 can you please share the links here?

10:40 AM · Jun 22, 2021 · Twitter for Android

61 Retweets **4** Quote Tweets **275** Likes

Nathan Lambert @natolambert · Jun 22
Replying to @FeryalMP
Some work on model-based RL:
- debugging / details tutorial: natolambert.com/writing/debugg...
- high level motivation robotic.substack.com/p/mbrl
- Hyperparameter sensitivity



Importance of Hyperparameter Optimization for Model-based ...
BAIR Blog

Ifty Mohammad Rezwan @imr165 · Jun 22
Replying to @FeryalMP
Lillian weng's p

Rohini Das @rohini__das · Jun 23
Replying to @FeryalMP
Here's the course website for Csaba's RL theory course at the UofA:
rltheory.github.io
Has all the lectures videos for the Winter 2021 term.

Home
Homepage of the course CMPUT 653: Theoretical Foundations of Reinforcement ...
rltheory.github.io

Pablo Samuel Castro
@pcastr
Replying to @FeryalMP
I wrote an intro to RL with colab and video:
psc-g.github.io/posts/mentorin...

There's a version of the video in Spanish as well (linked from the same site) 🙂

References

- Mnih et al., 2013 Playing Atari with Deep Reinforcement Learning
- Bellemare, Dabney, Munos, 2017 A Distributional Perspective on Reinforcement Learning
- van Hasselt et al., 2018 Deep Reinforcement Learning and the Deadly Triad
- Hessel et al., 2017 Rainbow: Combining Improvements in Deep Reinforcement Learning
- Puigdomènech Badia et al., 2020 Agent57: Outperforming the Atari Human Benchmark
- Sutton et al., 2000 Policy Gradient Methods for Reinforcement Learning with Function Approximation
- Lillicrap et al., 2015 Continuous control with deep reinforcement learning
- Barth-Maron et al., 2018 Distributed Distributional Deterministic Policy Gradients
- Mnih et al., 2016 Asynchronous Methods for Deep Reinforcement Learning
- OpenAI et al., 2019 Solving Rubik's Cube with a Robot Hand
- Schulman et al., 2015 Trust Region Policy Optimization
- Schulman et al., 2017 Proximal Policy Optimization Algorithms
- Espeholt et al., 2018 IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures
- Abdolmaleki et al., 2018 Maximum a Posteriori Policy Optimisation
- Wang et al., 2016 Sample Efficient Actor-Critic with Experience Replay
- Silver et al., 2017 A general reinforcement learning algorithm that masters chess, shogi and Go through self-play
- Fu et al., 2020 D4RL: Datasets for Deep Data-Driven Reinforcement Learning
- Gulcehre et al., 2020 RL Unplugged: Benchmarks for Offline Reinforcement Learning

Thank you

Special thanks to

Kelson Shilling-Scrivo, Konrad Kording, Anis Zahedifard

neuromatch
academy

Section 7

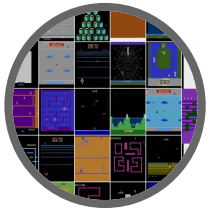
Deep Q-Networks (DQN)

Feryal Behbahani

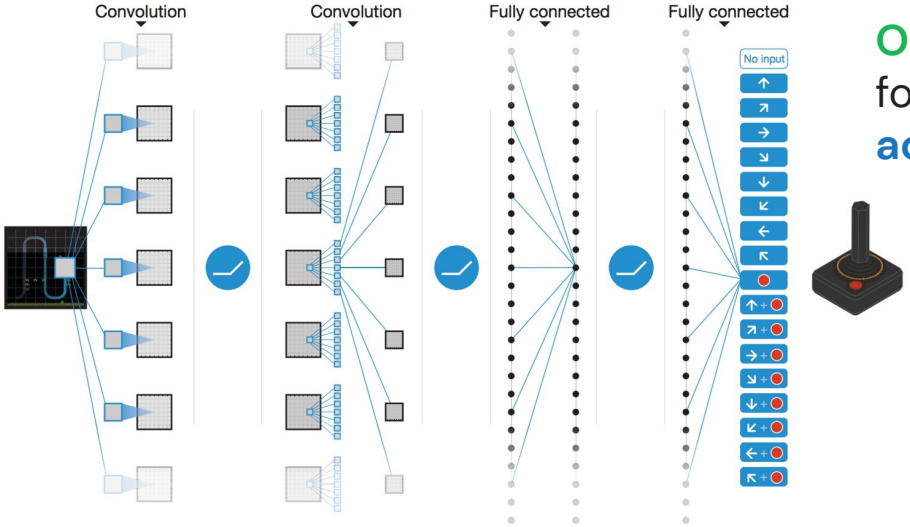
neuromatch academy

Deep Q-Networks (DQN)

End-to-end learning of the **value function** $Q(s, a)$ from **pixels**.



Input is the stack of raw pixels from the last 4 frames.



Output is the **Q-function** for the 18 possible joystick **actions**.

(Mnih et al., 2013)

Deep Q-Networks (DQN)

End-to-end learning of the **value function** $\mathbf{Q}(\mathbf{s}, \mathbf{a})$ from **pixels**.

DQN addressed main challenges with **value function** approximation

- Non-stationary targets
- Correlation between samples

Deep Q-Networks (DQN)

End-to-end learning of the **value function** $\mathbf{Q}(\mathbf{s}, \mathbf{a})$ from **pixels**.

DQN addressed main challenges with **value function** approximation

- Non-stationary targets → **Target Network**
- Correlation between samples → **Experience Replay**

The DQN Loss

We minimize the loss between

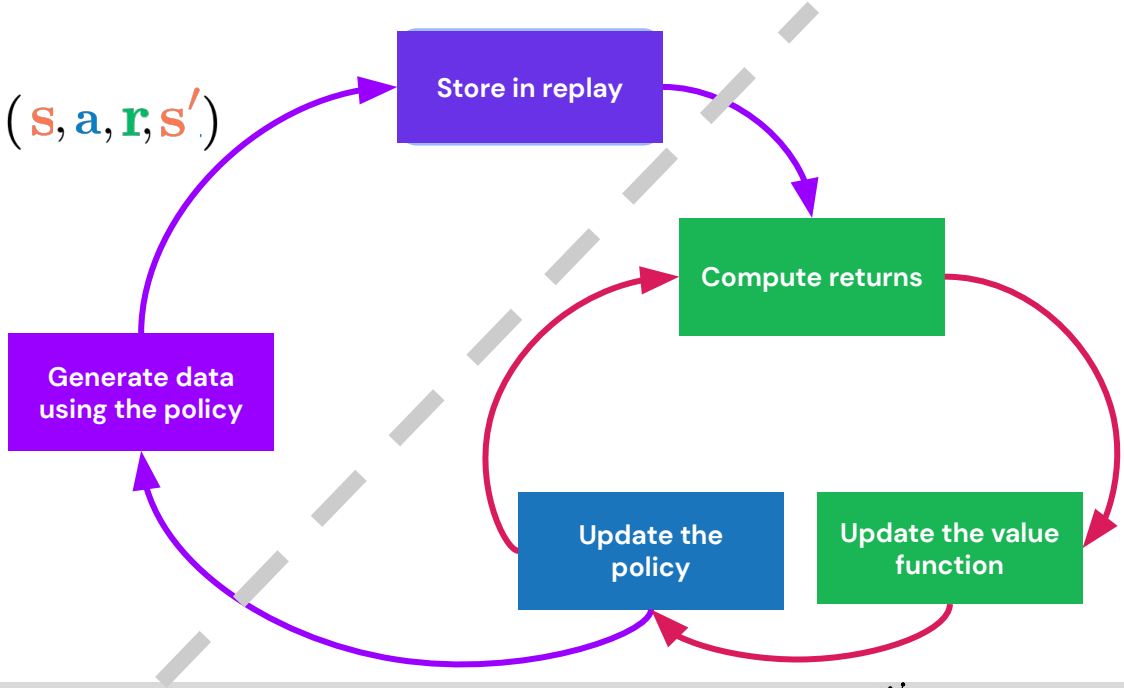
- the **TD Target**
- the **current value**

$$L(\mathbf{W}) = \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'}\left[\left(\mathbf{R}(\mathbf{s},\mathbf{a},\mathbf{s}') + \overbrace{\gamma \max_{\mathbf{a}'} \mathbf{Q}_{\mathbf{W}_{\text{old}}}(\mathbf{s}',\mathbf{a}')}^{\text{old/target value function}} - \underbrace{\mathbf{Q}_{\mathbf{W}}(\mathbf{s},\mathbf{a})}_{\text{optimized value function}}\right)^2\right]$$

Target network $\mathbf{W}_{\text{old}}$ uses a different/older set of weights than the weights being updated

Experience Replay

Remove temporal correlations by storing transitions in a **replay**



In some sense there are **two loops:**

- **Data Generation**

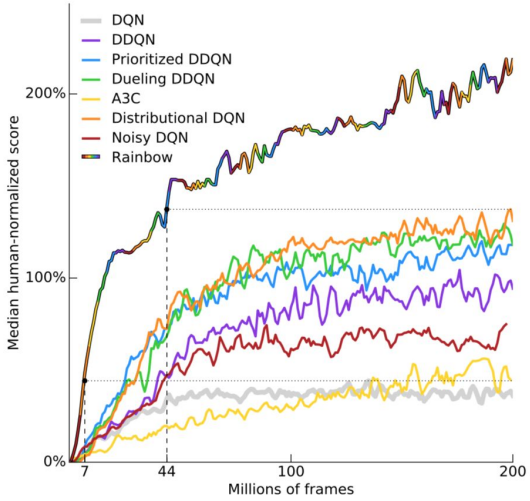
  and

- **Parameter optimization**

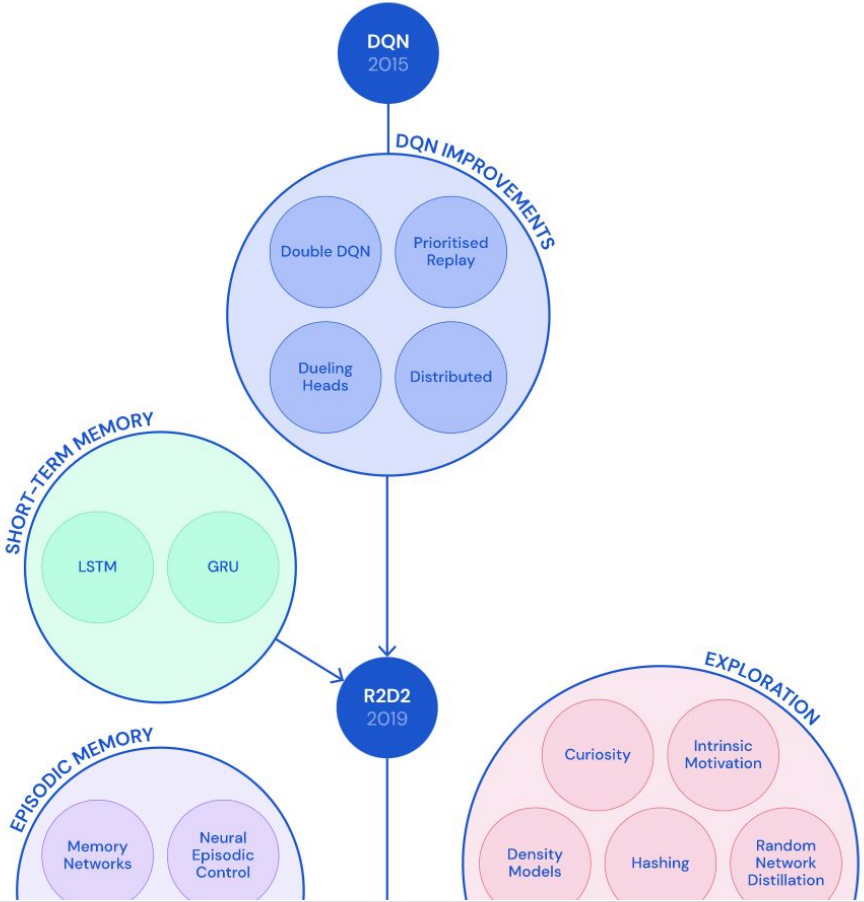
Can we decouple these?

Rainbow

Combines a collection of algorithmic improvements and large-scale training.



(Hessel et al., 2018)

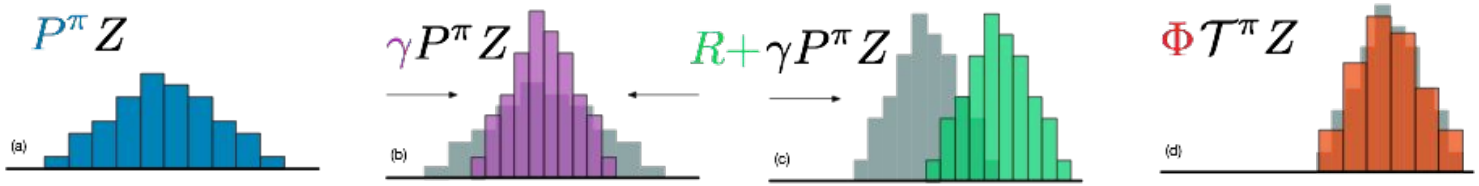


Distributional RL

Distributional RL goes further and learns a **distribution** over **returns**!

$$\mathbf{Q_w}(\mathbf{s}, \mathbf{a}) = \mathbb{E}\underbrace{\mathbf{Z_w}(\mathbf{s}, \mathbf{a})}_{\text{distribution over value}}$$

The **Bellman update** and **loss** now involve these distributions



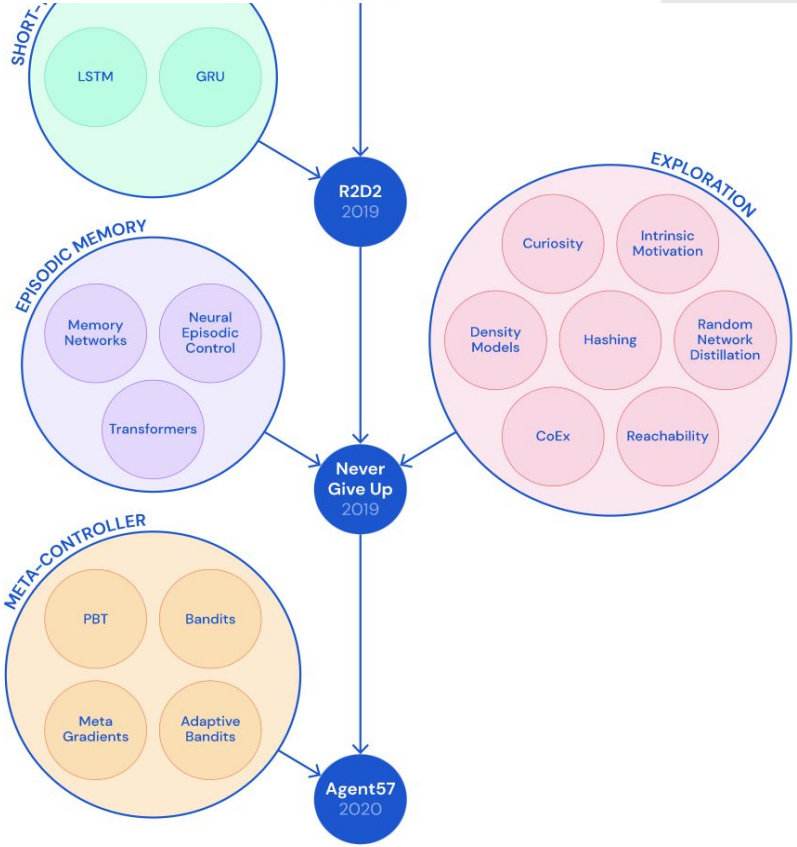
(Bellemare et al., 2017)

Agent-57

- Leverages an efficient exploration strategy which is novelty seeking

- Bandit meta-controller that adapts the long-term vs short-term behaviour of the agent.

(Puigdomènech Badia, et al., 2020)



Section 8

Other deep RL methods

Jane Wang

neuromatch academy

Challenges for value-based learning methods

- **Continuous** action space (value-based methods don't naturally handle this)
- Large action spaces - **hard to explore**
- Optimal policy is a **stochastic** one - epsilon greedy not optimal

Observations $\longrightarrow$ $Q(s, a)$ $\longrightarrow$ **Policy**

Why policy optimization?

- Learning the **policy** **directly** can sometimes be simpler than learning the **value function**
- Particularly better suited in **continuous control** and **large action spaces**.
- In **some** problems the optimal policy is **stochastic**
- But stochasticity can make **exploration** easier

What is policy optimization?

We explicitly represent the **policy** with its own parameters independent of any **value function**

$$\mathbf{a_t} \sim \boldsymbol{\pi_\theta}(\mathbf{a_t}|\mathbf{s_t}) \quad \text{vs} \quad \mathbf{a_t} = \arg\max_{\mathbf{a}} \mathbf{Q}(\mathbf{s_t}, \mathbf{a}) = \boldsymbol{\pi}(\mathbf{s})$$

We update the **policy parameters** using **gradient ascent** to **maximise expected future reward**.

Directly parameterizing the policy

We'll now turn to **parameterized policies** given by $\boldsymbol{\theta}$

$$\mathbf{a_t} \sim \pi_{\boldsymbol{\theta}}(\mathbf{a_t} | \mathbf{s_t})$$

Future rewards exactly as seen previously!

and we will write the **expected return** as a function of these parameters

$$\mathbf{J}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{s_{0:T+1}}, \mathbf{a_{0:T}}} \left[\sum_{t=0}^{T} \gamma^t \, \mathbf{R}(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}) \right]$$

$$= \sum_{\mathbf{s_{0:T+1}}, \mathbf{a_{0:T}}} \left[\sum_{t=0}^{T} \gamma^t \, \mathbf{R}(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}) \right] \Pr(\mathbf{s_{0:T+1}}, \mathbf{a_{0:T}})$$

Summing over all possible trajectories

Multiplied by their probability

High level intuition

- **Gradient of performance is in same direction as gradient of policy**
- Key takeaway:
  - Starting with some random initial policy for acting
  - Sample some actions to interact with environment
  - In proportion to the return (or estimated future value), adjust the weights to increase probability of taking those same actions

$$\nabla_{\boldsymbol{\theta}} \mathbf{J}(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \mathbf{G}_{\mathbf{t}}^{(\mathbf{i})} \nabla \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_{\mathbf{t}}^{(\mathbf{i})} | \mathbf{s}_{\mathbf{t}}^{(\mathbf{i})})$$

Advantage Actor Critic (A2C) and Baselines

The **policy gradient** can have *high variance*. **Can we reduce it?**

Baseline! Expectation zero because independent of $\mathbf{a}$

$$\nabla_\theta \mathbf{J}(\theta) = \mathbb{E}_{d^\pi(\mathbf{s})\pi(\mathbf{a}|\mathbf{s})}\left[\underbrace{\left(\mathbf{Q}^{\pi_\theta}(\mathbf{s},\mathbf{a}) - \mathbf{V}^{\pi_\theta}(\mathbf{s})\right)}_{\mathbf{A}^{\pi_\theta}(\mathbf{s},\mathbf{a})} \nabla \log \pi_\theta(\mathbf{a}|\mathbf{s})\right]$$

The **advantage** measures how much **better** $\mathbf{a}$ is than following the policy $\pi_\theta(\mathbf{s})$

A2C:

- Estimate **Q** using returns $\mathbf{G}_\mathbf{t}^{(\mathbf{i})}$
- Estimate **V** with a neural network

(Mnih, et al., 2015)

Plethora of RL methods

Value-based

vs

policy-based

Model-based

vs

Model-free

on-policy

vs

off-policy

Offline

vs

Online

Episodic

vs

Lifelong

...