DeepMind

RL in Real World: Offline RL - Lecture 1 - Introduction to RL

Caglar Gulcehre

Based on Lecture notes by Feryal Behbahani, Matt Hoffman, Bobak Shariari and Abbas Abdolmaleki

Who am I?

Main Interest Areas:

- Offline RL and Apprenticeship Learning
- Natural Language Understanding
- Representation Learning: Causal or Self-Supervision
- Achieving Scientific Principles for Deep Learning
- Memory

Obtained PhD from MILA under Yoshua Bengio's supervision

Currently:

Senior Research Scientist @ Deepmind

DeepMind

Thursday Lecture 1 – Intro to RL, 09:00–09:45

15 mins Break

Thursday Lecture 1 – Intro to RL, 10:00–10:30

Thursday Lecture 2 – Intro to Offline RL, 15:15–16:00

15 mins Break

Thursday Lecture 2 – Intro to Offline RL, 16:15–16:45

Friday Lecture 3 – Offline RL, 11:00–11:45

15 mins Break

Friday Lecture 3 – Offline RL, 12:00–12:30

DeepMind

“Intelligence is the ability to learn to perform well over a wide range of environments”

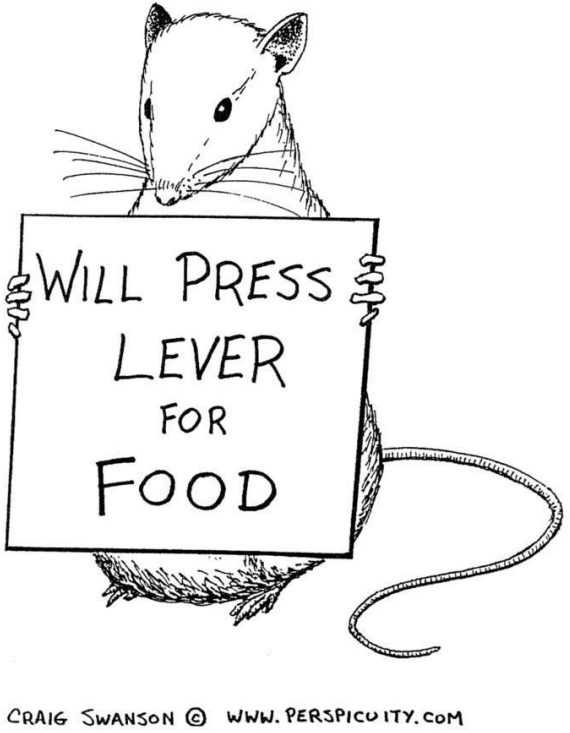
DeepMind

“Intelligence is the ability to learn to perform well over a wide range of environments”

The way to get there: **build general purpose learning systems**

Reinforcement Learning

- How do we learn to solve problems?
  - **Trial and error**



RL Example


After 15000 trials
Time 00:00.03
Reward -0.000562359414637
Cumulative -0.000562

RL Example


After 200 trials
Time 00:00.03
Reward 0.000491882729269
Cumulative 0.000492

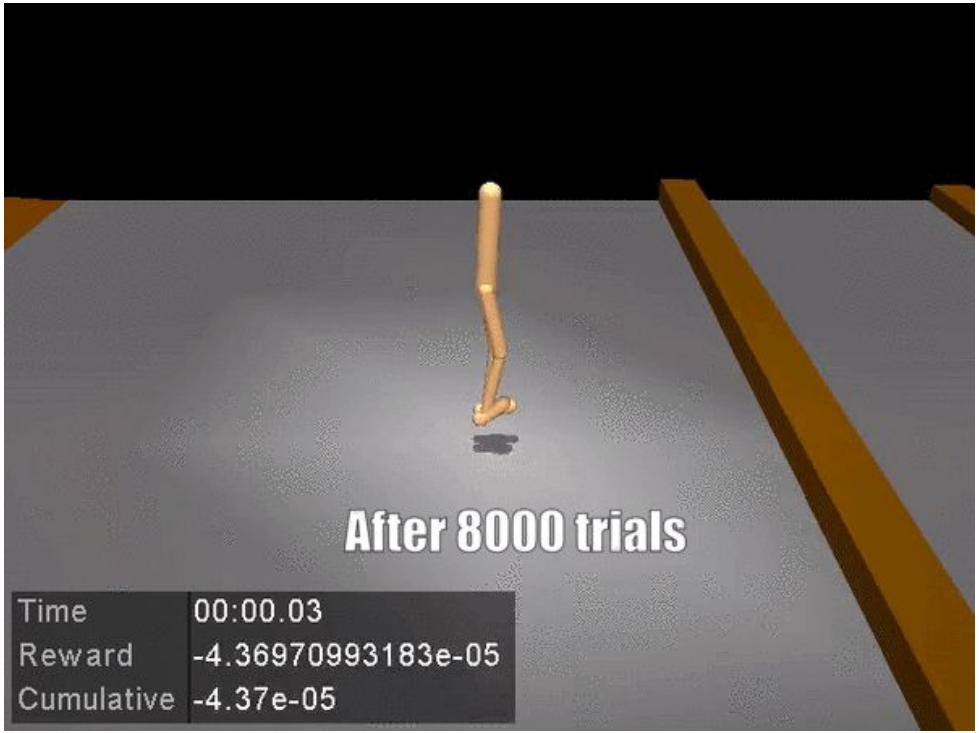
RL Example


After 2000 trials
Time 00:00.03
Reward -0.000388528895386
Cumulative -0.000389

RL Example


After 6000 trials
Time 00:00.03
Reward -0.00096222858493
Cumulative -0.000962

RL Example


After 8000 trials
Time 00:00.03
Reward -4.36970993183e-05
Cumulative -4.37e-05

RL Example


After 15000 trials
Time 00:00.03
Reward -0.000562359414637
Cumulative -0.000562

Reinforcement Learning

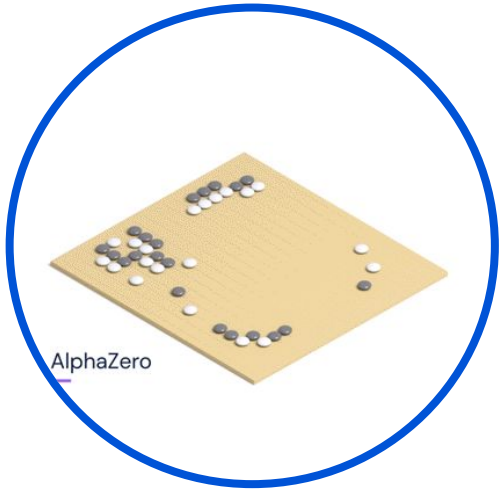
- How do we learn to solve problems?
  - **Trial and error**
- Reinforcement learning (RL) is a general framework to express how this process is performed.

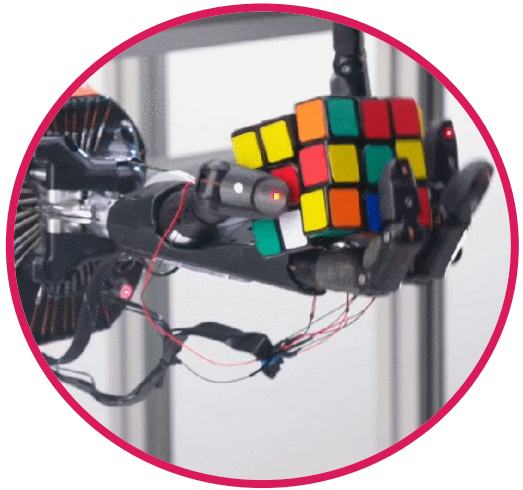
- There are two important aspects to the paradigm

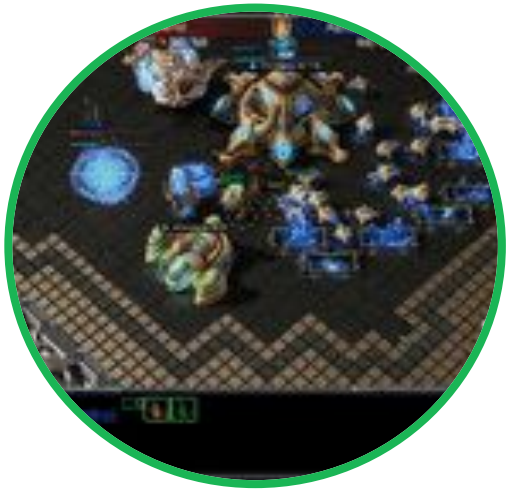
  1- It allows us to specify the goal

  2- It can deal with long-term dependencies

Successes of RL in machine learning







Why makes RL harder than supervised learning?

RL = exploration to gather experience + learning from experience

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

Overview

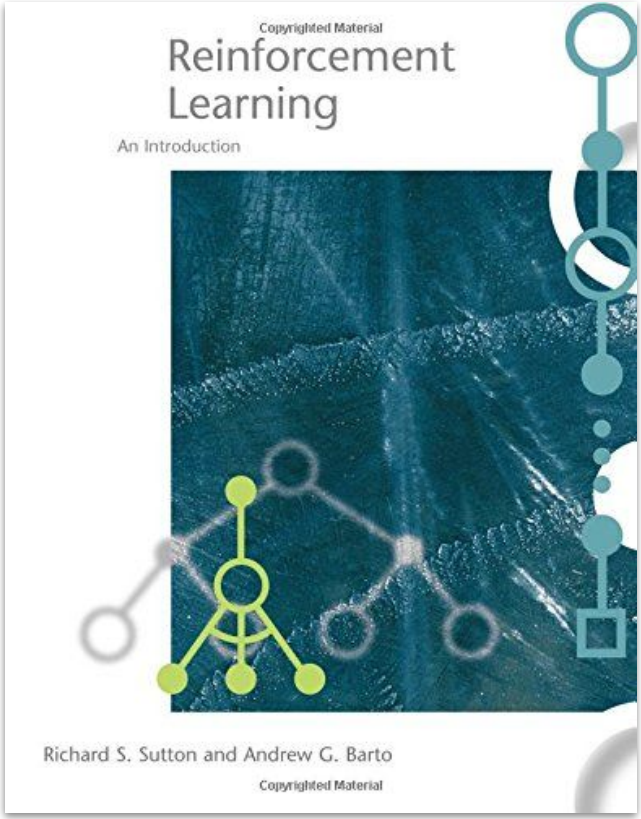
Fundamental Concepts

Value-based methods

Policy-based methods

Challenges & opportunities

Reinforcement Learning: An Introduction



Overview

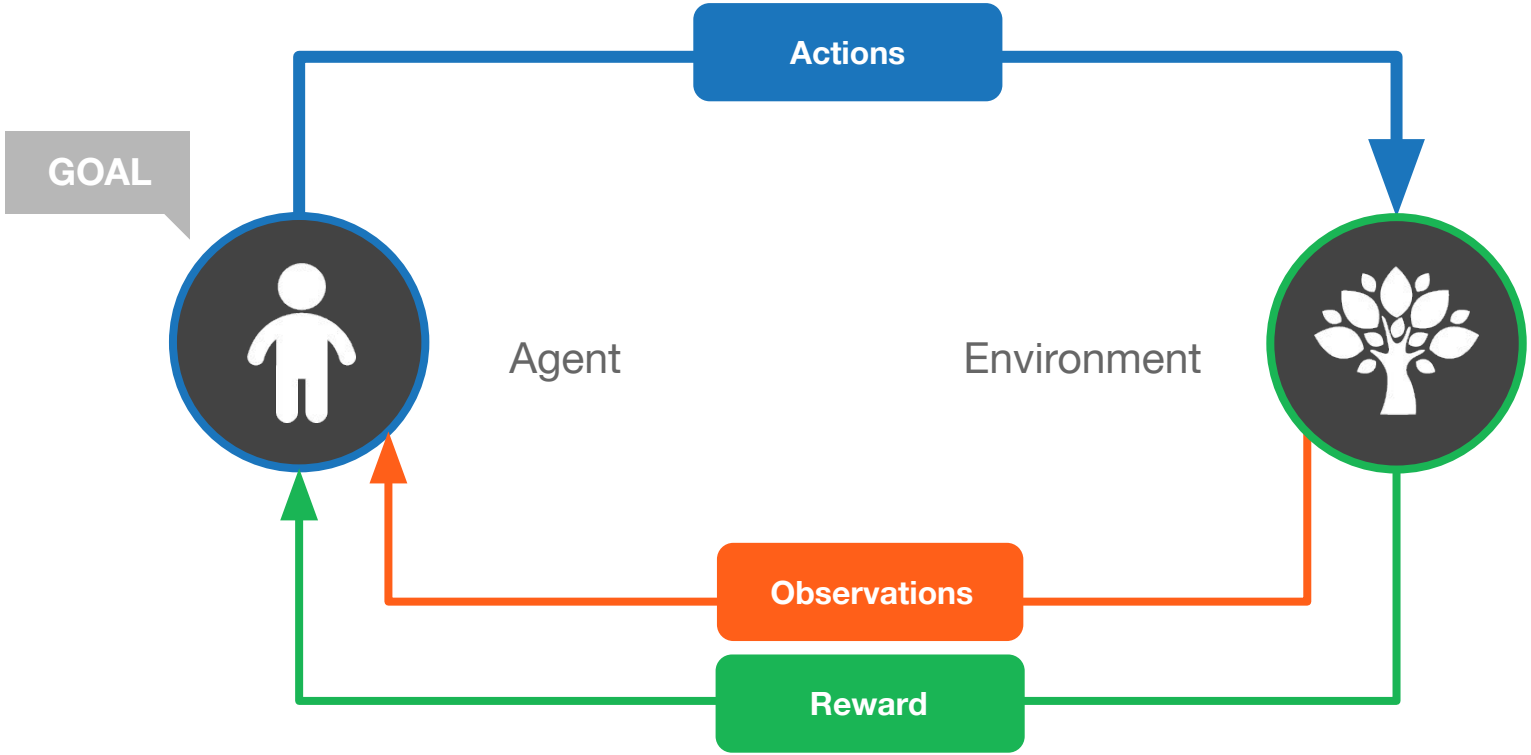
Fundamental Concepts

Value-based methods

Policy-based methods

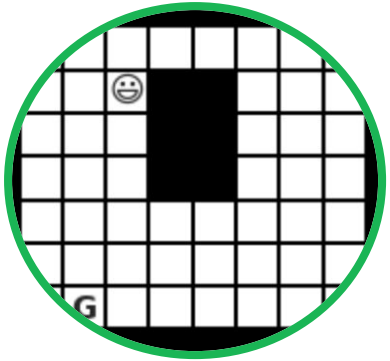
Challenges & opportunities

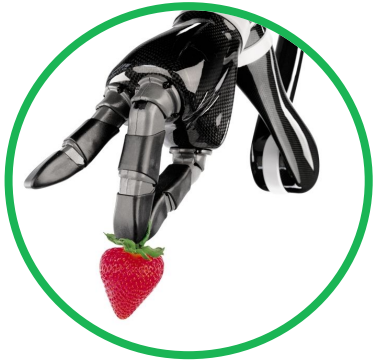
What is reinforcement learning?

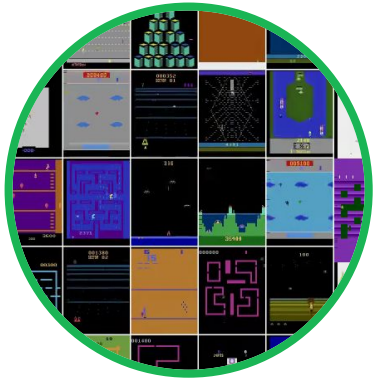


Learning to achieve **goals** via **trial and error** by **interacting** with an **environment**

What is an environment?







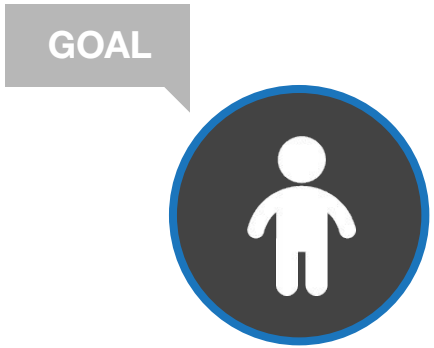
What is an environment?

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 agent?



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

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

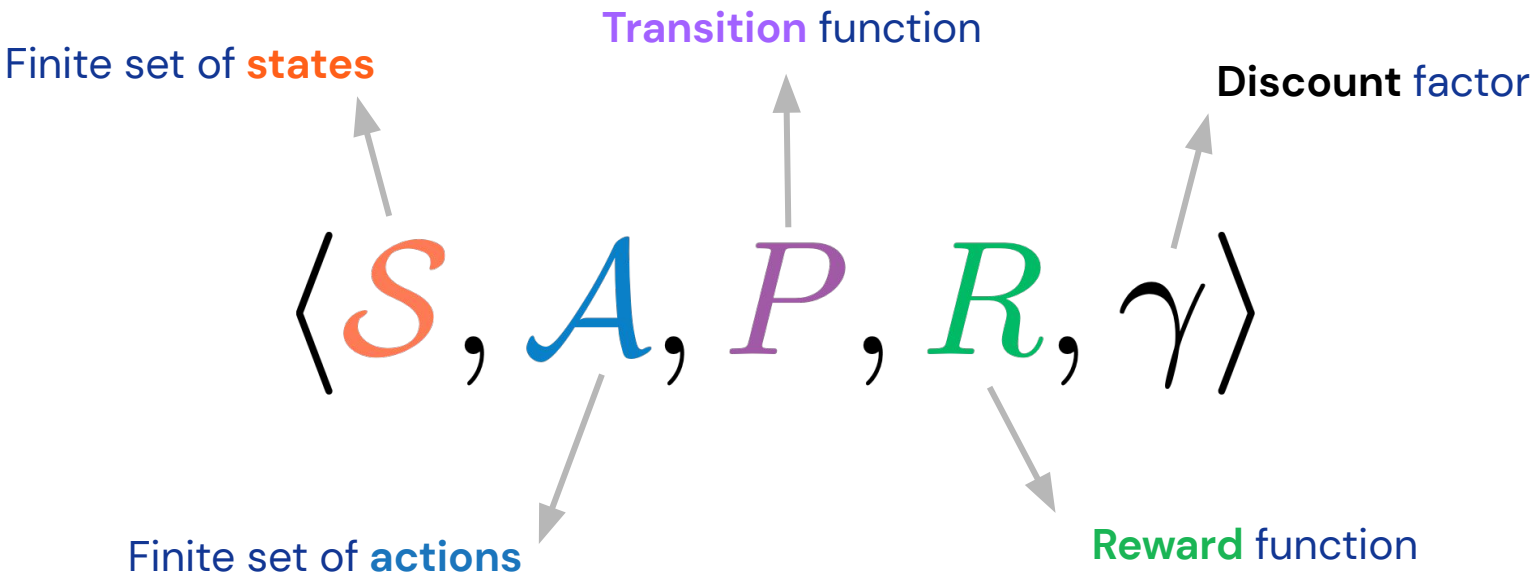
Learning to achieve **goals** via **trial and error** by **interacting** with an **environment**

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

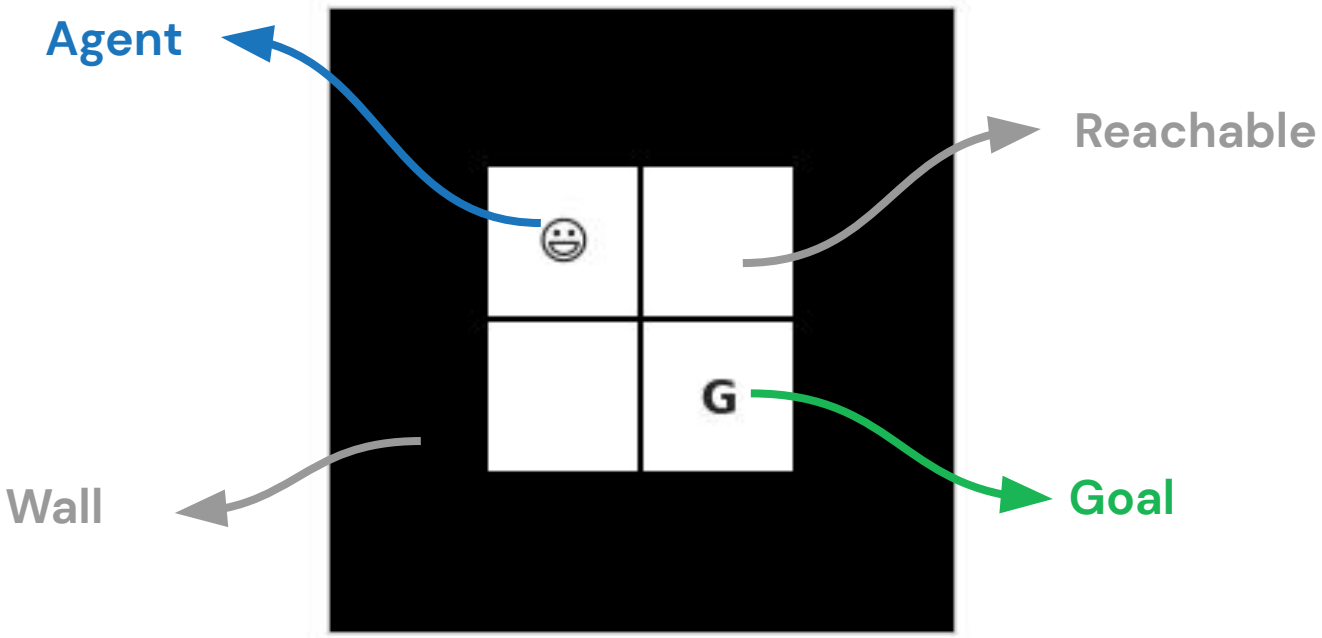
Markov Decision Process (MDP)



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

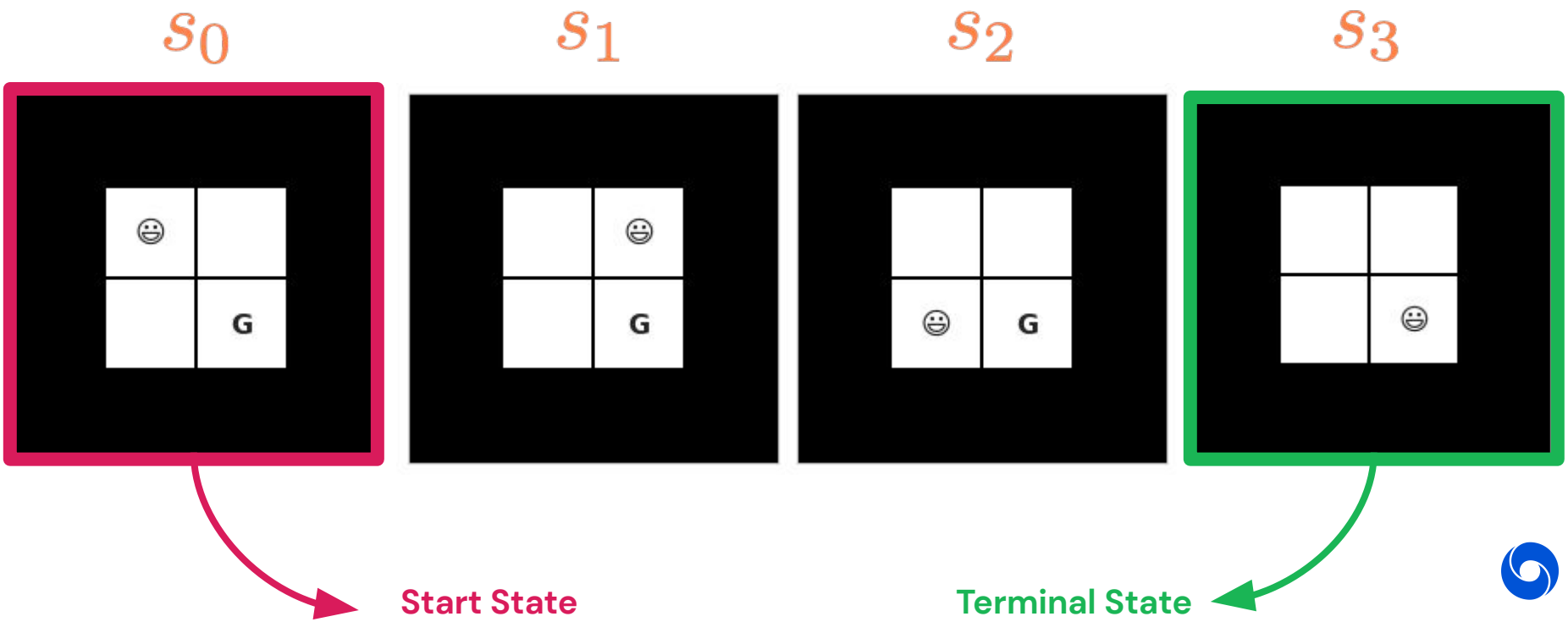
Grid World Environment

Size of the world: [2 X 2]

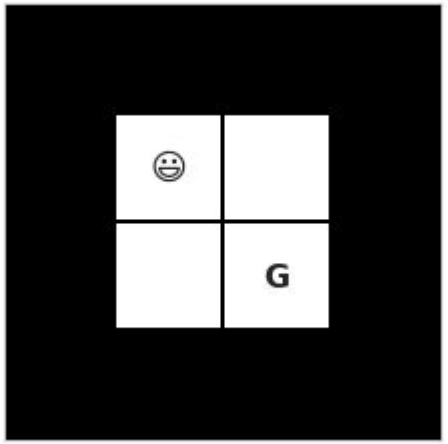


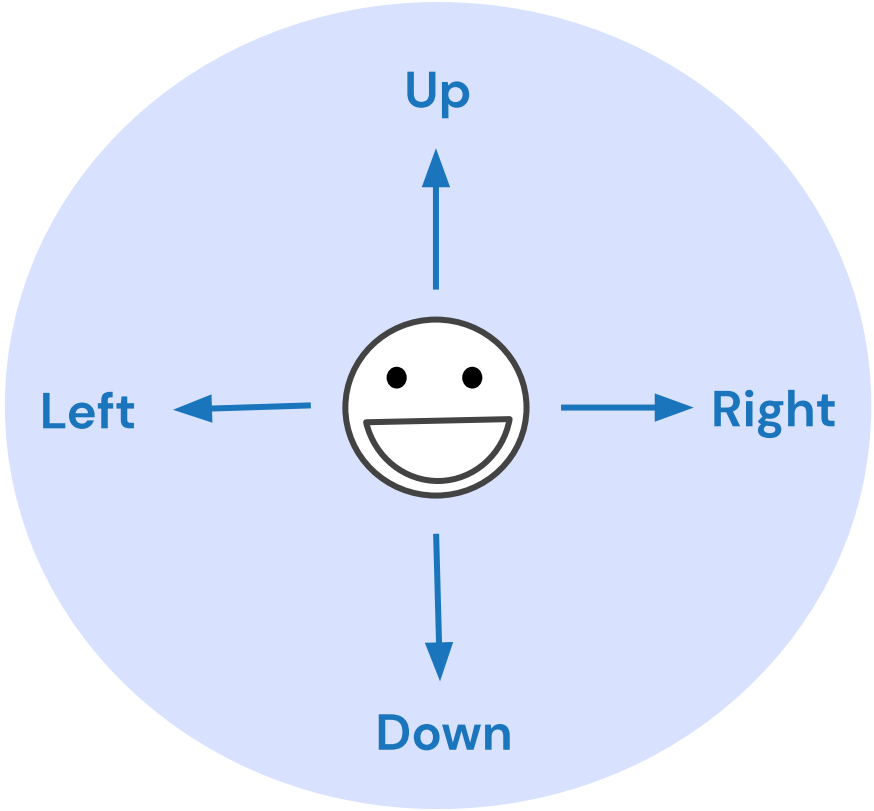
State space $\mathcal{S}$

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

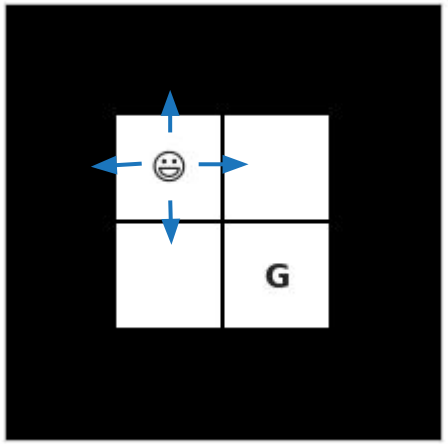


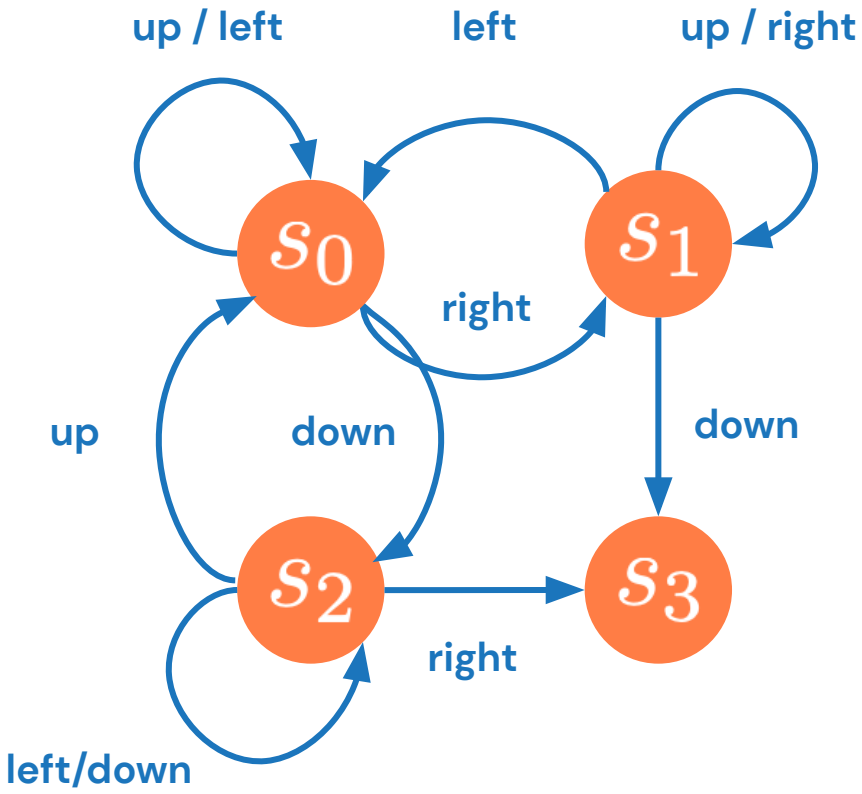
Action Space $\mathcal{A}$



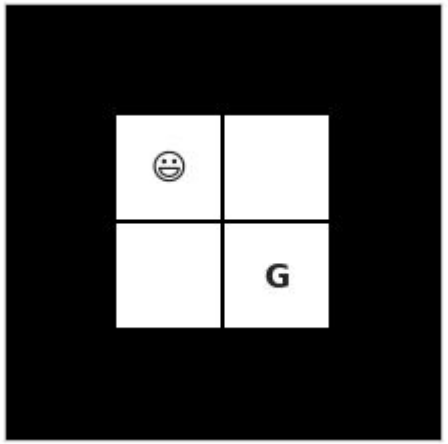


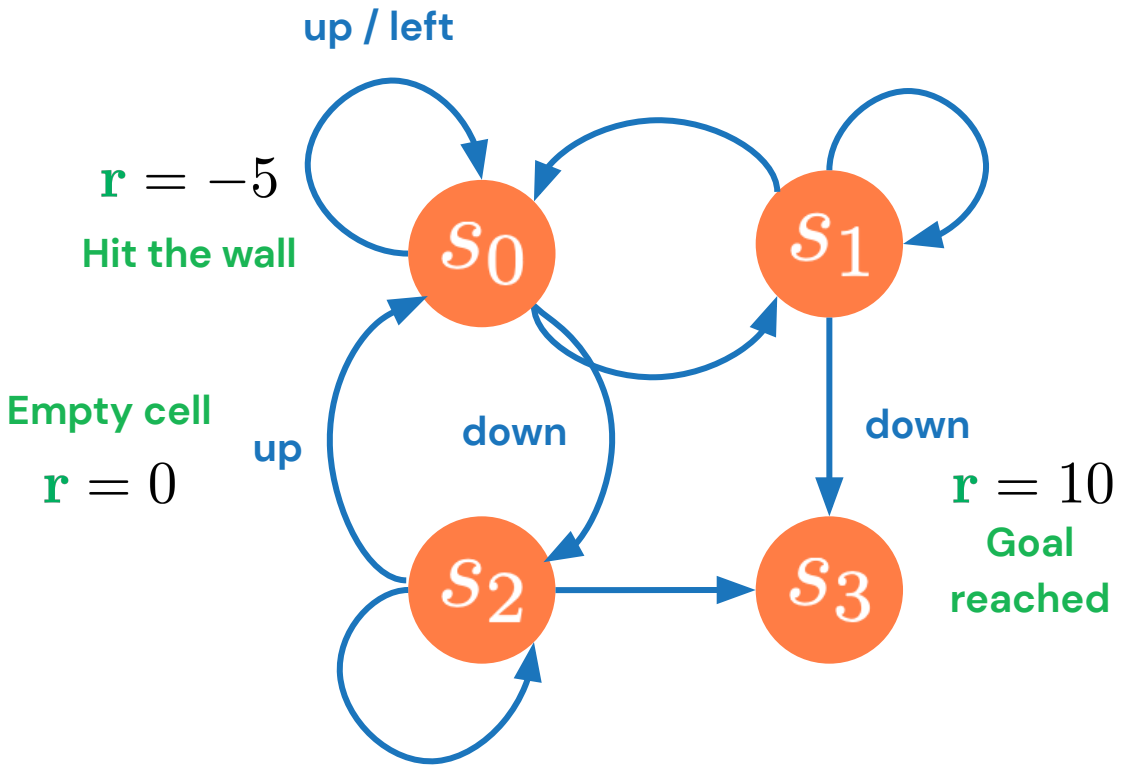
Transition model P





Rewards R


G


up / left
$r = -5$
Hit the wall
s_0
s_1
Empty cell
$r = 0$
up
down
down
$r = 10$
Goal reached
s_2
s_3

Markov Decision Process (MDP)

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

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

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

Agent's Goal

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

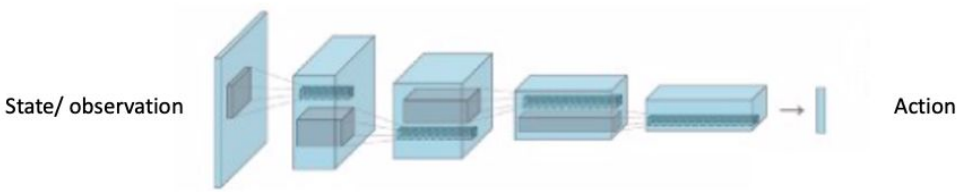
$$\mathbf{G_t} = \underbrace{\mathbf{r_t} + \gamma \mathbf{r_{t+1}} + \gamma^2 \mathbf{r_{t+2}} +}_{\textbf{Return}} \ldots$$

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

Policy

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

$$a_t \sim \pi(a_t | s_t)$$



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

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 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**.

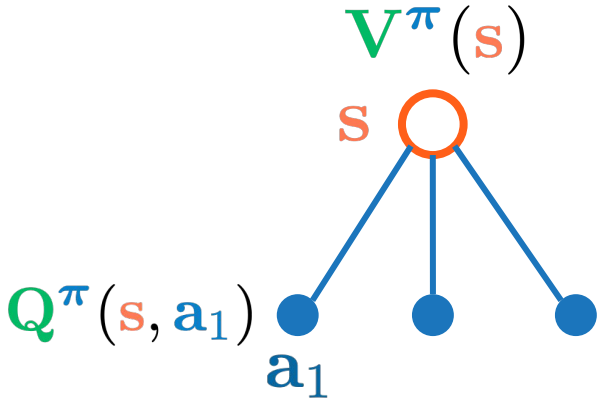
$$Q^{\pi}(s_t, a_t) = \underset{\pi}{\mathbb{E}}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t, a_t\right]$$

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 $\mathbf{V}^{\pi}$ and $\mathbf{Q}^{\pi}$

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



In summary

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

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

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

Overview

Fundamental Concepts

Value-based methods

Policy-based methods

Challenges & opportunities

What are we going to learn?

1. How to **compute** the value function?
2. How to **use it to optimise** the policy?
3. How to **learn** it, if we don’t have the model of the environment?

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

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

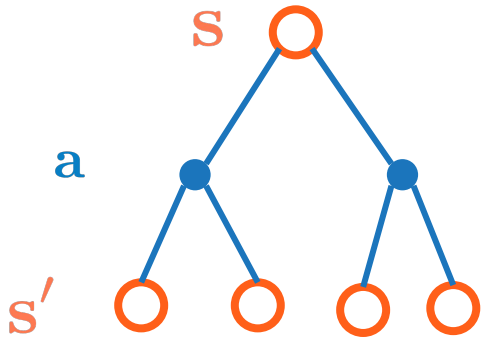
$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[r_t + \gamma V^{\pi}(s_{t+1}) | s_t\right]$$

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**.



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

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

How to improve on a policy? (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**:

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

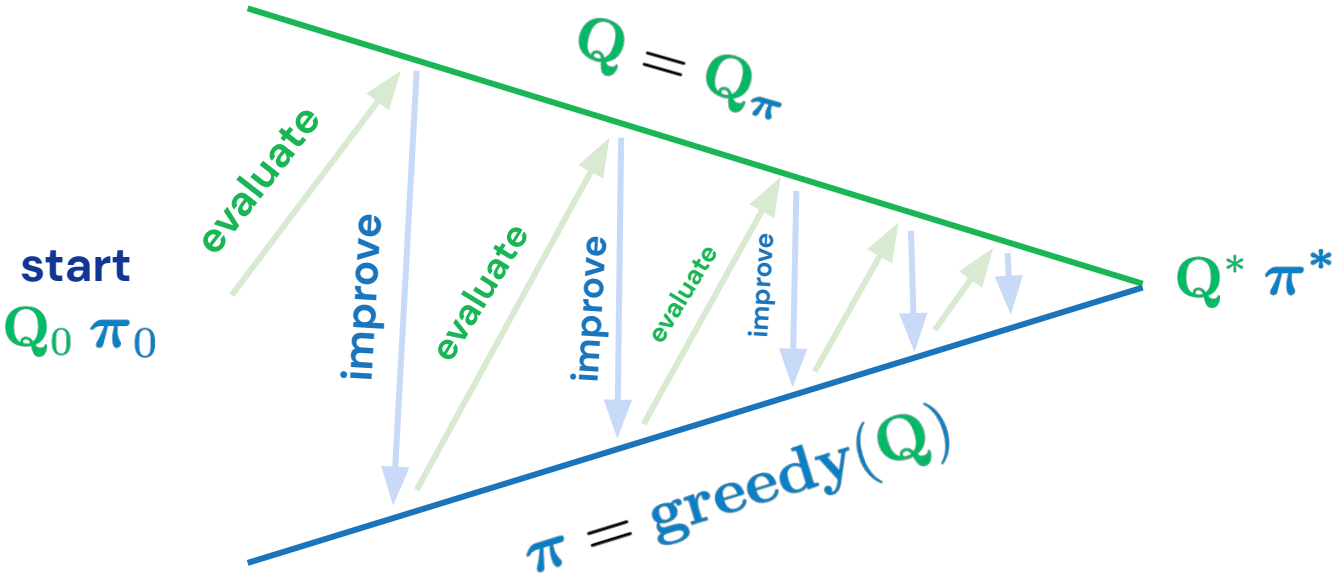
Where **better** means:

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

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

How to find the optimal policy? (Policy Iteration)

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



Policy Evaluation vs Policy Improvement

Policy evaluation:

Policy Evaluation step computes the state-value function of a state s with the policy π and the Bellman updates:

$$\mathbf{V}^{\pi}(\mathbf{s}) = \mathop{\mathbb{E}}_{\pi}[\mathbf{r}_t + \gamma \mathbf{V}^{\pi}(\mathbf{s}_{t+1})|\mathbf{s}_t] = \sum_{\mathbf{a} \in \mathcal{A}} \pi(\mathbf{a}|\mathbf{s}) \left(\mathbf{R}(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathbf{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \mathbf{V}^{\pi}(\mathbf{s}') \right)$$

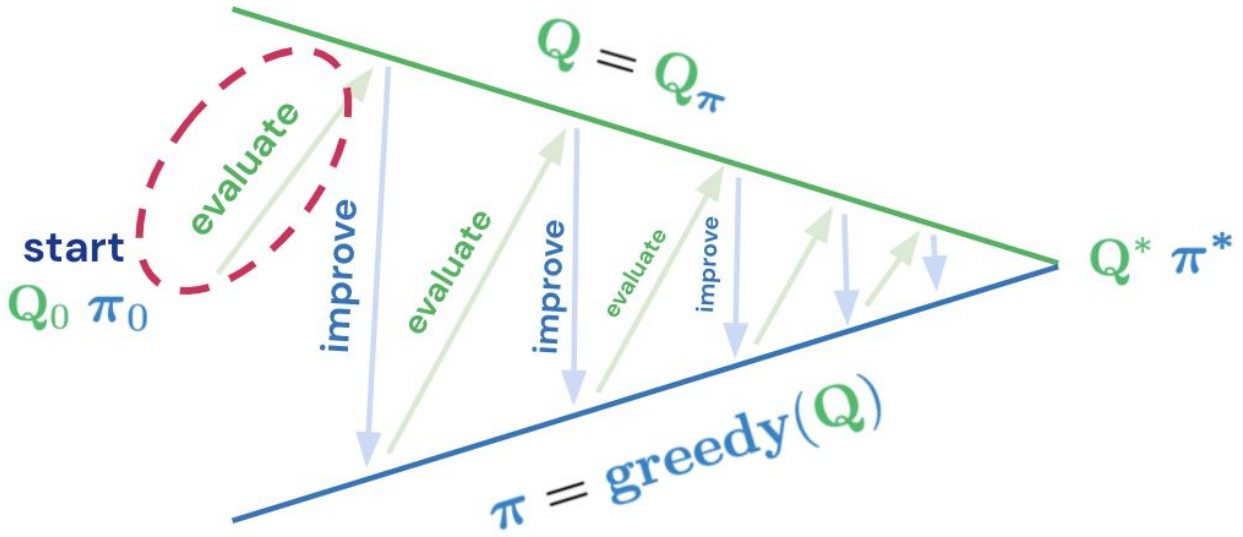
Policy Improvement:

Policy Improvement operator tries to find a better policy based on the value function, for example by using the greedy max operator:

$$\mathbf{Q}^{\pi}(\mathbf{s}, \mathbf{a}) = \mathop{\mathbb{E}}_{\mathbf{s}' \in \mathcal{S}}[\mathbf{R}(\mathbf{s}, \mathbf{a}) + \gamma \mathbf{V}^{\pi}(\mathbf{s}_{t+1})|\mathbf{s}_t] = \sum_{\mathbf{s}' \in \mathcal{S}} \left(\mathbf{R}(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathbf{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \max_{\mathbf{a}'} \mathbf{Q}^{*}(\mathbf{s}', \mathbf{a}') \right)$$

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**.

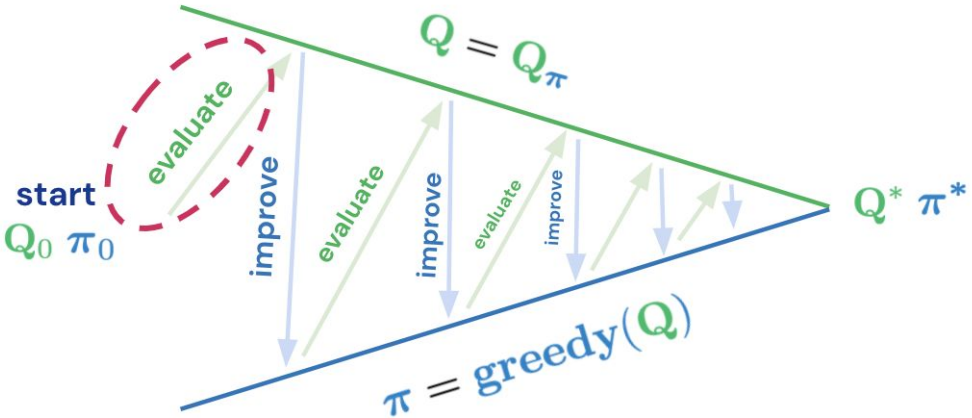
DP algorithms turn Bellman equations into update rules and iterate until convergence:

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

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:

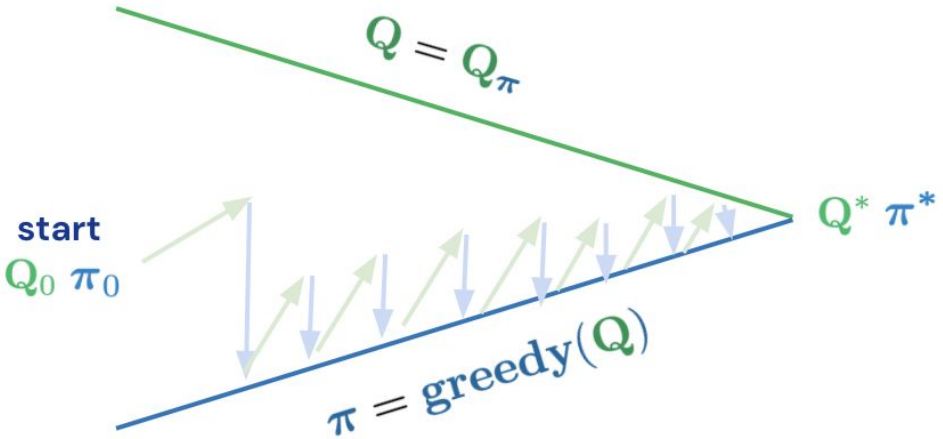
$$\mathbf{Q}_k(\mathbf{s},\mathbf{a}) \leftarrow \mathbf{R}(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathbf{P}(\mathbf{s}'|\mathbf{s},\mathbf{a}) \sum_{\mathbf{a}' \in \mathcal{A}} \boldsymbol{\pi}_{k-1}(\mathbf{a}'|\mathbf{s}')\mathbf{Q}_{k-1}(\mathbf{s}',\mathbf{a}')$$



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**

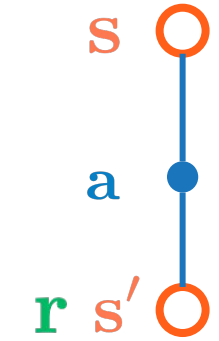
Monte Carlo Policy Evaluation

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.

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

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

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.

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

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**.

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**.

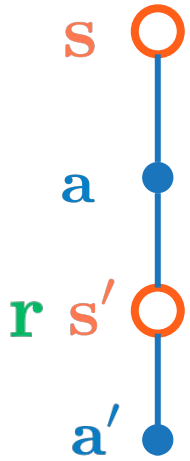
On-policy learning "Learning on the job"

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

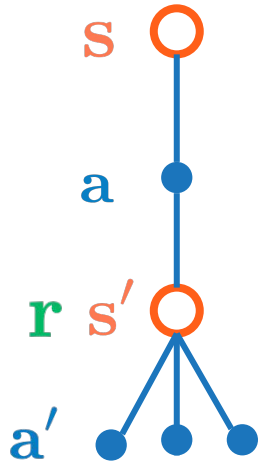


- **SARSA** learns the **policy** by executing it in the environment to gather experiences.
- We often use ε-**Greedy policy** to gather experiences and each update needs access to $(\mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}', \mathbf{a}')$.

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

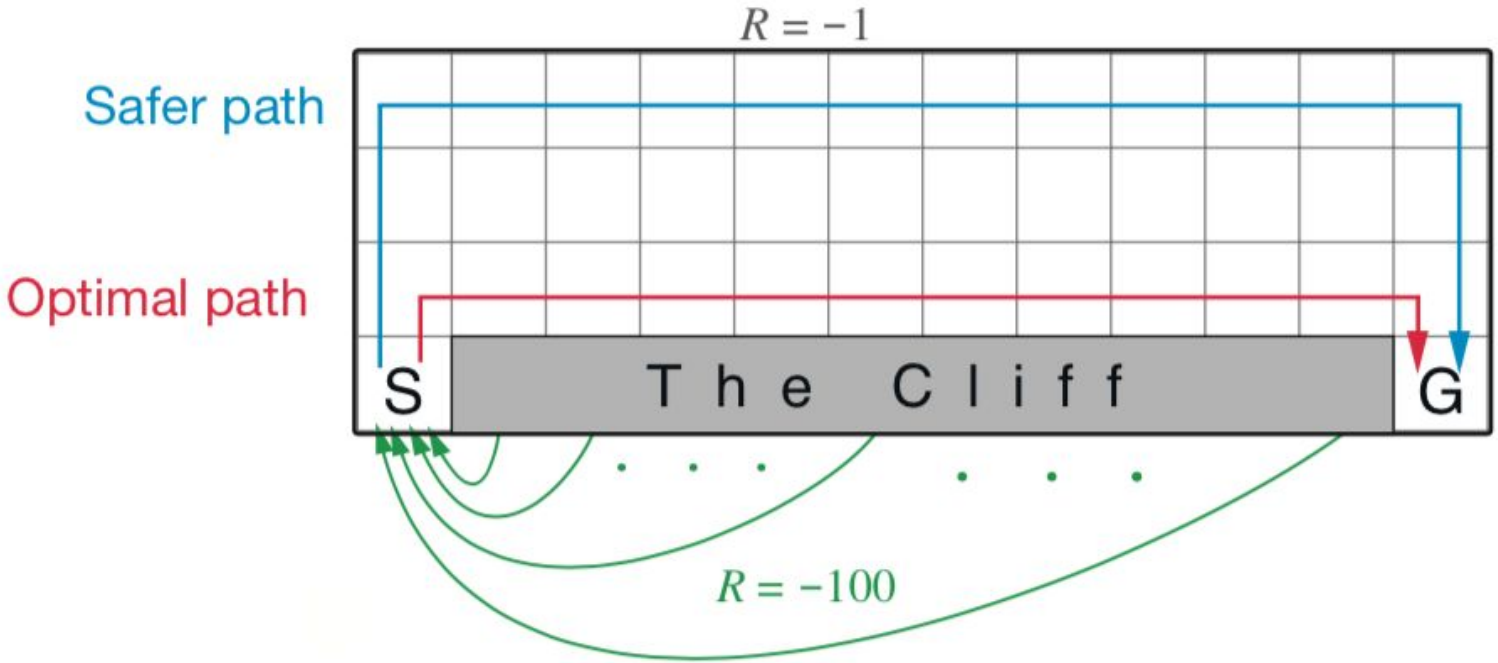
Q Learning: Off-policy TD Control

- **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.



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

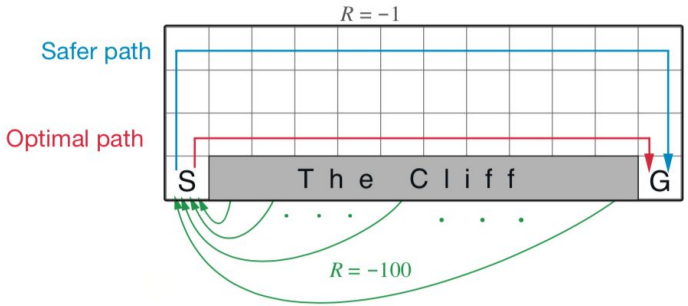
SARSA vs Q-Learning: The Cliff-walking Example

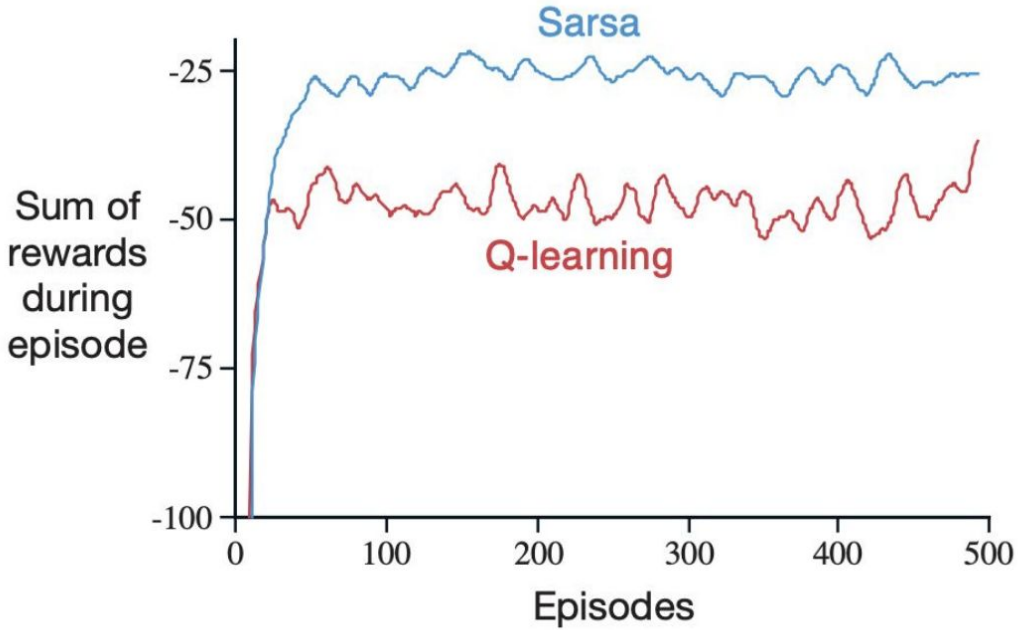


Sutton & Barto. Reinforcement Learning: An Introduction. (Chapter 6)

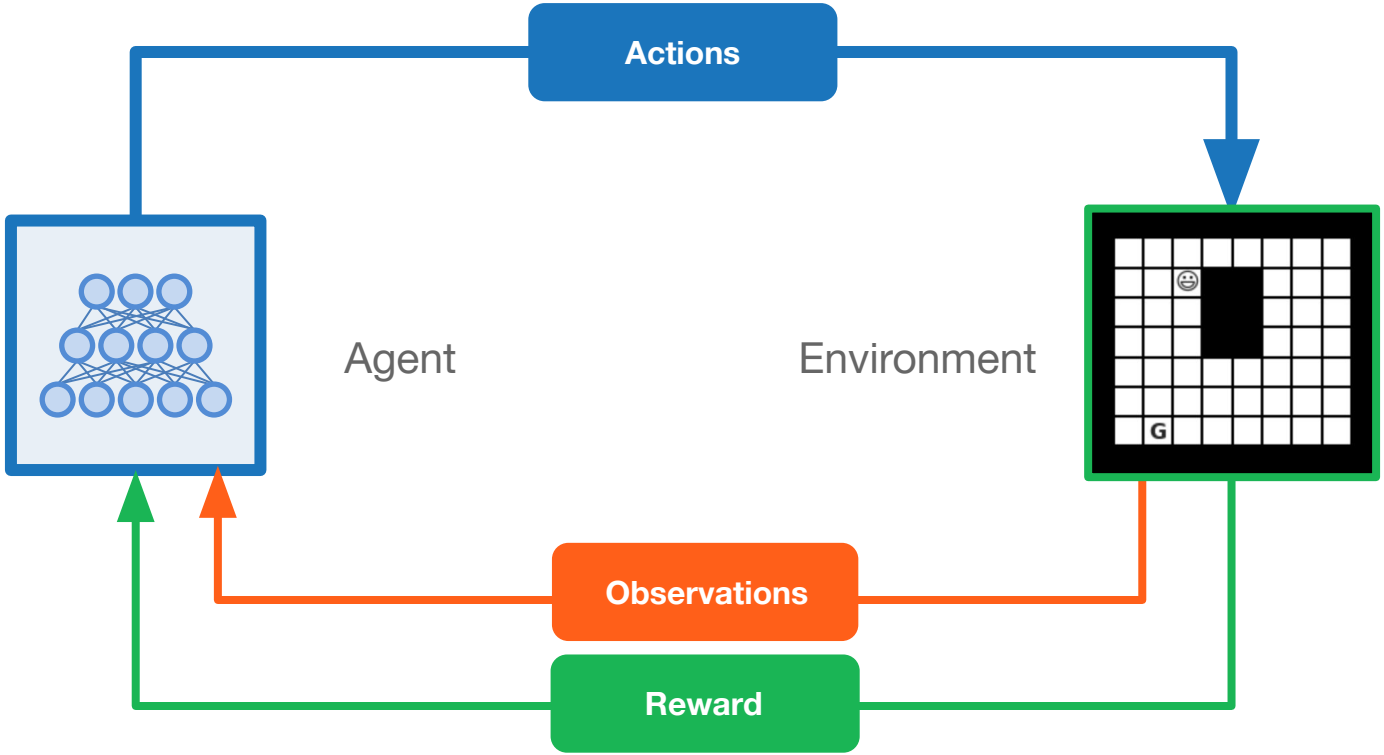
SARSA vs Q-Learning: The Cliff-walking Example

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



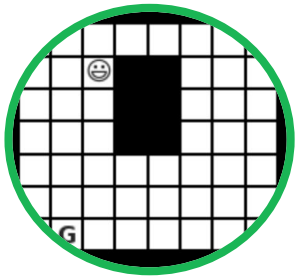


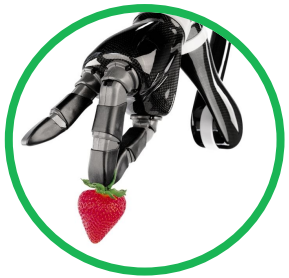
Function approximation



Why function approximation?

- Up to now we used lookup tables to represent the **value function**, for example each $\mathbf{s} \in \mathcal{S}$ and $\mathbf{a} \in \mathcal{A}$ has an entry in the **Q-table** $\mathbf{Q}(\mathbf{s}, \mathbf{a})$
- 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.



Why function approximation?

- With **function approximation**, we can tackle these **large MDP**s, by **generalizing** from **seen states** to **unseen states**.
- We can use the **TD target** we learned about earlier as a **loss function** to optimize using gradient descent.
- Our goal is to find the parameter vector $\mathbf{w}$ minimising the **mean-squared error** between the **approximate value function** $\hat{Q}_{\mathbf{w}}(s, a)$ and the **true value function**.

The deadly triad

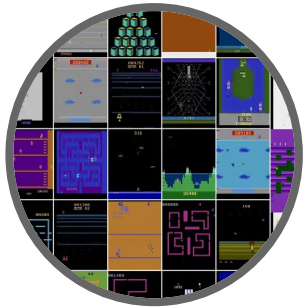
Bad news: there is possibility of **divergence** when we combine these three powerful ingredients:

Function approximation
Significantly generalising from experiences

\+ **Bootstrapping**
Learning value estimates from other value estimates (e.g. in TD and DP)

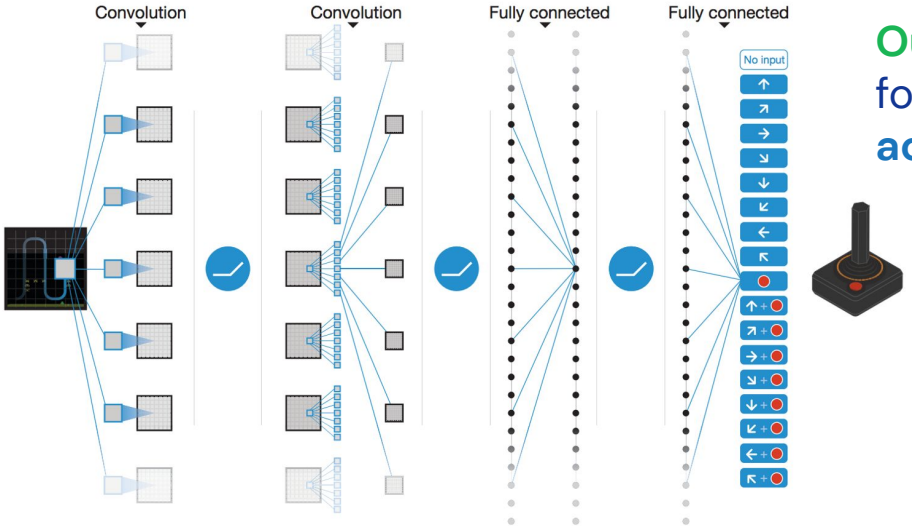
\+ **Off-policy learning**
Learning from experiences collected by a different behaviour policy than the target policy we are learning about (e.g. Q-learning).

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)

The DQN Loss

With DQN we can't even apply the Q-function update in closed form

old/updated value function

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

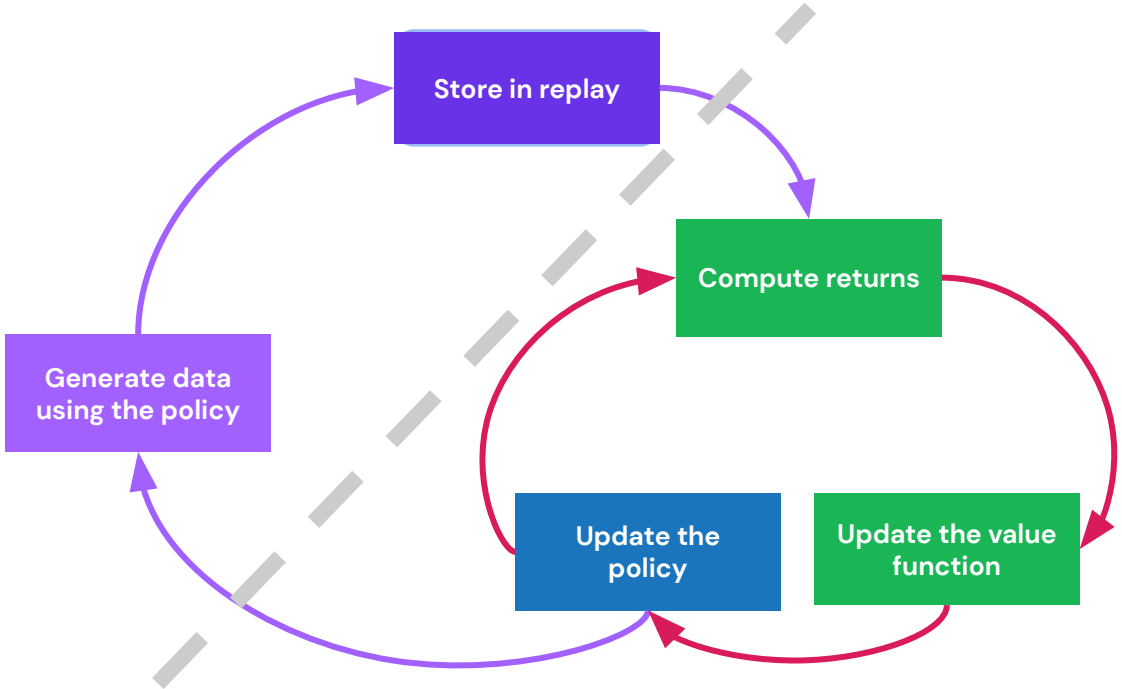
optimized value function

so instead we minimize the loss between

- an **Bellman-updated** value function and
- the **current** value

Replay

Off-policy methods can also make use of **replay** (storage of data)



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

- **Data Generation**

  and

- **Parameter optimization**

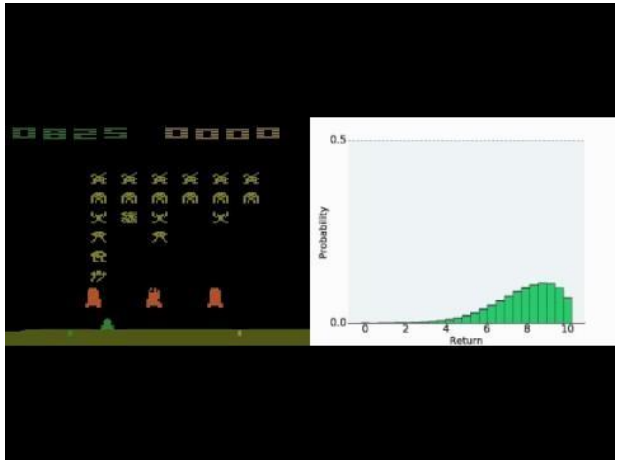
Can we split these?

Distributional RL

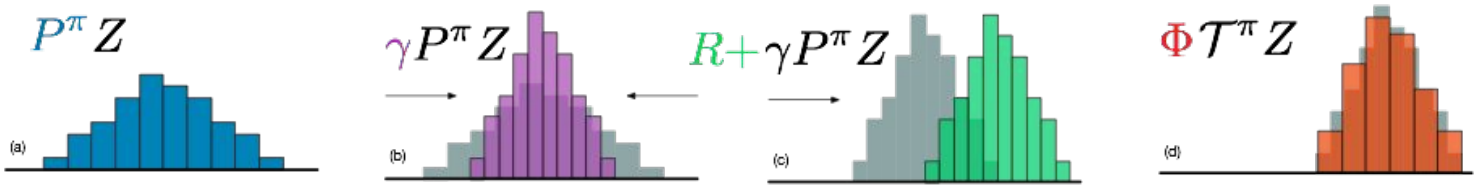
Distributional RL goes further and learns a **distribution** over **returns**!

$$Q_w(s, a) = \mathbb{E} Z_w(s, a)$$

distribution over value



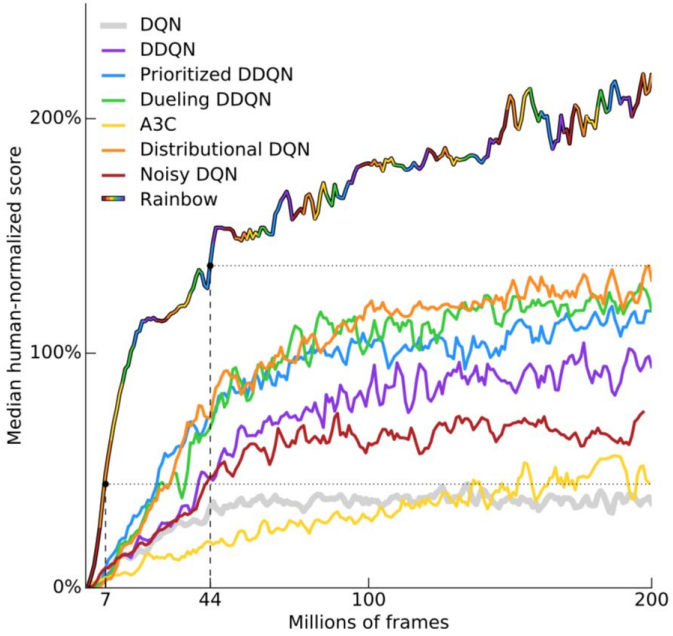
The **Bellman update** and **loss** now involve these distributions



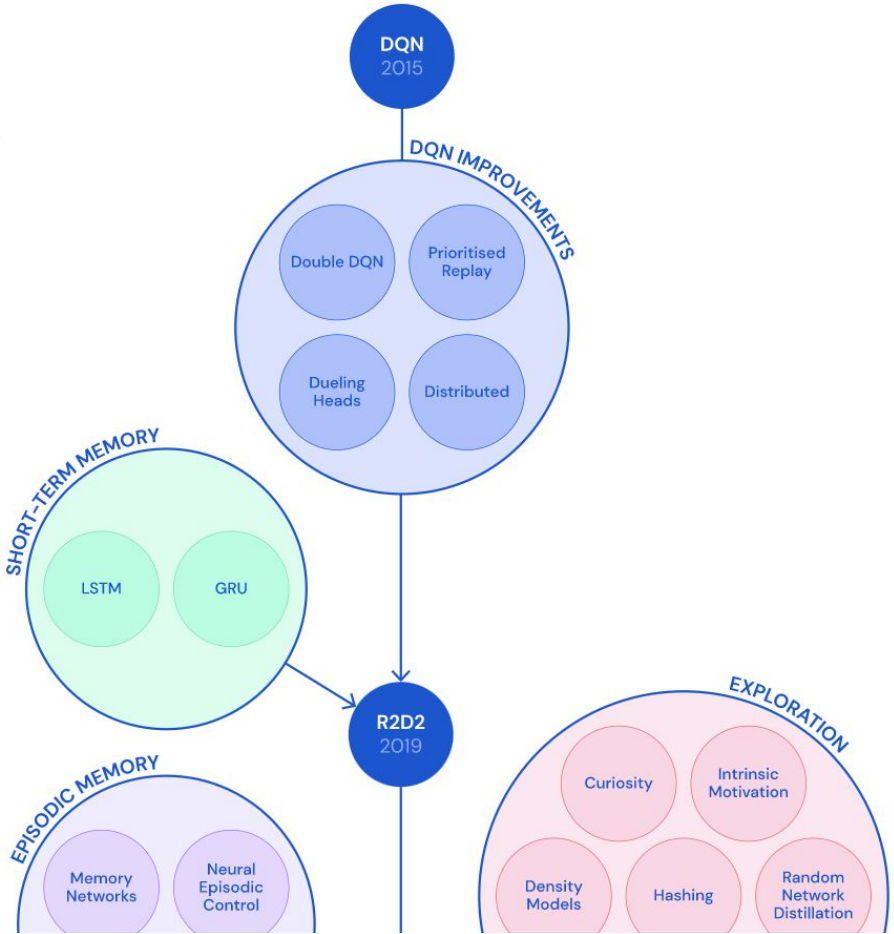
(Bellemare et al., 2017)

Rainbow

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

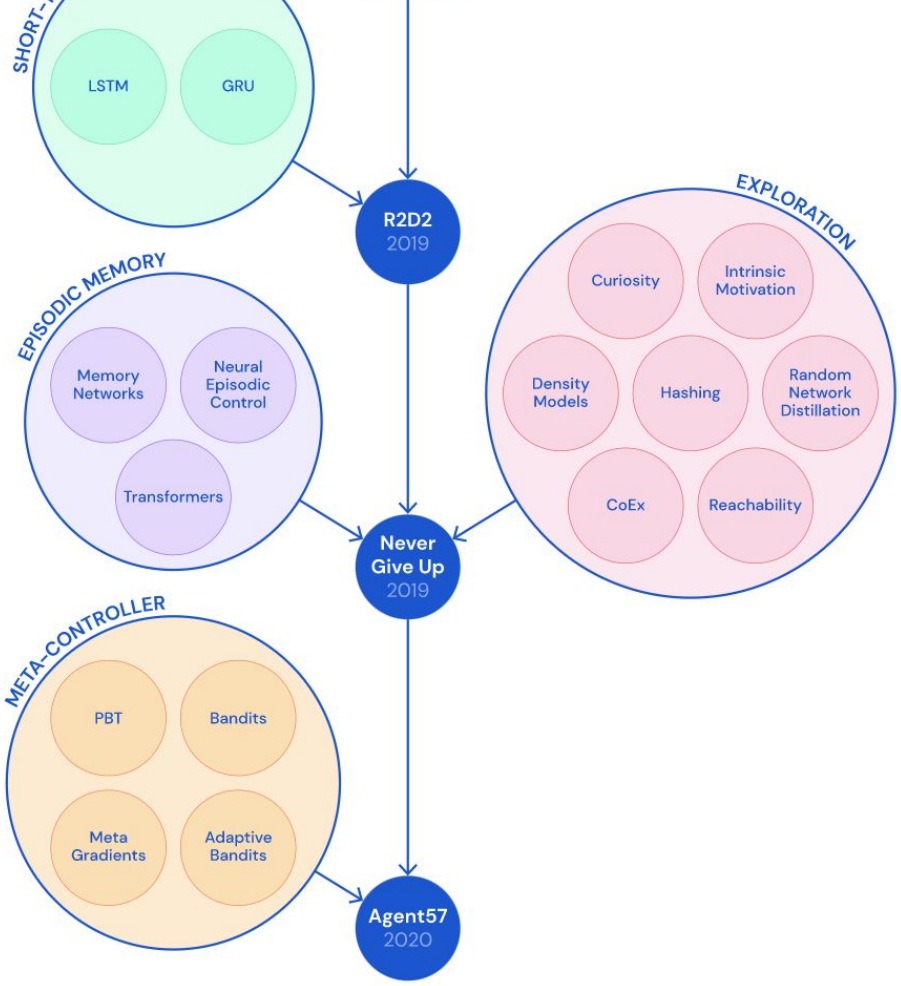


(Hessel et al., 2018)



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)

Overview

Fundamental Concepts

Value-based methods

Policy-based methods

Challenges & opportunities

What are we going to learn?

1. What is a **parameterized policy**
2. How to derive the **policy gradient**
3. How this relates to **value functions**
4. How minor variations lead to **vastly different algorithms**

What is policy optimisation?

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

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

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

Why policy optimisation?

- Learning the **policy** 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**
- Stochasticity can make **exploration** easier

Directly parameterizing the policy

Future rewards exactly as seen previously!

We'll now turn to **parameterized policies** given by θ

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

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

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

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

Summing over all possible trajectories

Multiplied by their probability

In what follows we'll show how to **compute the gradient** of this

The log-derivative trick

The gradient of our **expected return** moves inside and only applies to the distribution over trajectories

$$\nabla_\theta J(\theta) = \sum_{s_{0:T+1}, a_{0:T}} \left(\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1}) \right) \nabla_\theta \Pr(s_{0:T+1}, a_{0:T} \mid \theta)$$

$$= \sum_{s_{0:T+1}, a_{0:T}} \left(\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1}) \right) \underbrace{\left(\nabla_\theta \log \Pr(s_{0:T+1}, a_{0:T} \mid \theta) \right)}_{\text{The "log-derivative trick"}} \Pr(s_{0:T+1}, a_{0:T} \mid \theta)$$

$$\nabla f(x) = \nabla \log f(x) f(x)$$

which can be again re-written as an expectation over **trajectories**

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

Generally useful!

Gradient of a trajectory's log-probability

Recall from the **Markov property** that our probability factorizes

$$\Pr(\mathbf{s}_{0:T+1}, \mathbf{a}_{0:T} \mid \boldsymbol{\theta}) = \Pr(\mathbf{s}_0) \prod_{t=0}^{T} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\, \mathbf{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

so the gradient of its log simplifies to

independent of $\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} \log \Pr(\mathbf{s}_{0:T+1}, \mathbf{a}_{0:T} \mid \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \left[\log \Pr(\mathbf{s}_0) + \sum_{t=0}^{T} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t) + \sum_{t=0}^{T} \log \mathbf{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$= \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$$

The policy gradient (putting it all together)

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\left(\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1})\right)\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\right]$$

independent of actions a_t the expectation vanishes, see properties of scores

split sum into before t and after t

$$= \mathbb{E}\left[\sum_{t=0}^{T}\left(\sum_{n=0}^{T} \gamma^n R(s_n, a_n, s_{n+1})\right) \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

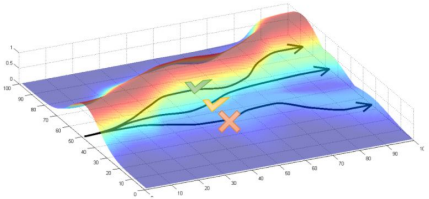
$$= \mathbb{E}\left[\sum_{t=0}^{T}\left(\sum_{n=0}^{t-1} \gamma^n R(s_n, a_n, s_{n+1}) + \sum_{n=t}^{T} \gamma^n R(s_n, a_n, s_{n+1})\right) \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

$$= \mathbb{E}\left[\sum_{t=0}^{T}\left(\sum_{n=t}^{T} \gamma^n R(s_n, a_n, s_{n+1})\right) \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

$$= \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \left(\sum_{n=t}^{T} \gamma^{n-t} R(s_n, a_n, s_{n+1})\right) \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

G_t

direction to move to make action more likely

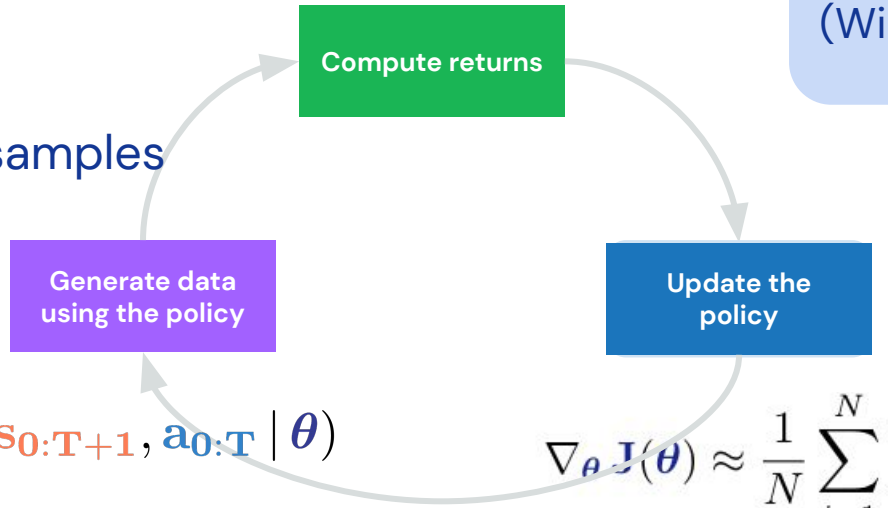


“Vanilla” policy gradients (REINFORCE)

$$G_t^{(i)} = \sum_{n=t}^{T} \gamma^{n-t} R(s_n^{(i)}, a_n^{(i)}, s_{n+1}^{(i)})$$

The basis of the **REINFORCE** algorithm (Williams, 1992)

Given N trajectory samples



$$s_{0:T+1}^{(i)}, a_{0:T}^{(i)} \sim \Pr(s_{0:T+1}, a_{0:T} \mid \theta)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \gamma^t G_t^{(i)} \nabla \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

Connection with Value-based methods

Recall the **Q-function**, evaluated at some **initial state** and **action**,

$$\mathbf{Q}^{\pi_\theta}(\mathbf{s_0}, \mathbf{a_0}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \, \mathbf{R}(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}) \,\middle|\, \mathbf{s_0}, \mathbf{a_0}\right]$$

We can replace reward trajectories with this function in our gradient, i.e.

$$\nabla_\theta \mathbf{J}(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \, \mathbf{Q}^{\pi_\theta}(\mathbf{s_t}, \mathbf{a_t}) \nabla \log \pi_\theta(\mathbf{a_t}|\mathbf{s_t})\right]$$

discounting makes dependence on **initial state** more important

future rewards

The Policy Gradient Theorem

The **policy gradient theorem** (Sutton et al., 2000) directly relates the **policy gradient** and Q

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

where $d^{\pi_\theta}(s)$ is the **stationary distribution** under the given policy*

Policy search methods often differ primarily in how **future rewards are estimated**.

*or the discounted probability of visiting at any time t.

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)

Deterministic Policy Gradients (DPG)

Usually **optimal policies** are deterministic $\mathbf{a} = \pi_{\boldsymbol{\theta}}(\mathbf{s})$

- Q-learning can do this with value-functions.
- Can we do so for policy search?

The **deterministic policy gradient theorem** (Silver et al., 2014) does exactly this:

$$\nabla_{\boldsymbol{\theta}} \mathbf{J}(\boldsymbol{\theta}) = \mathbb{E}_{d^{\pi}(\mathbf{s})} \Big[\underbrace{\nabla_{\mathbf{a}} \mathbf{Q}^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a})\big|_{\mathbf{a}=\pi_{\boldsymbol{\theta}}(\mathbf{s})}}_{\text{direction to move } \mathbf{a} \text{ to make the value higher}} \underbrace{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\mathbf{s})}_{\text{how moving } \boldsymbol{\theta} \text{ influences the direction of } \mathbf{a}} \Big]$$

Used with deep networks this is Deep DPG (**DDPG**) (Lillicrap et al., 2018)

Actor Critic Methods in General

We have *sneakily* introduced **Actor-Critic** methods that:

- Learn a **value function** (V or Q) or **advantage**
- Use this **value function** to learn the **policy**

For example **DDPG** explicitly splits these into two steps:

- Update a parameterized $\mathbf{Q_w(s, a)}$ using **TD-learning**
- Update a parameterized $\boldsymbol{\pi_\theta(s)}$ using $\mathbf{Q}$

What about the gradient step?

Sometimes **just following** the gradient step can be unstable

- Many algorithms **penalize** the gradient step
- Often based on the **entropy or KL** between the new and old policy

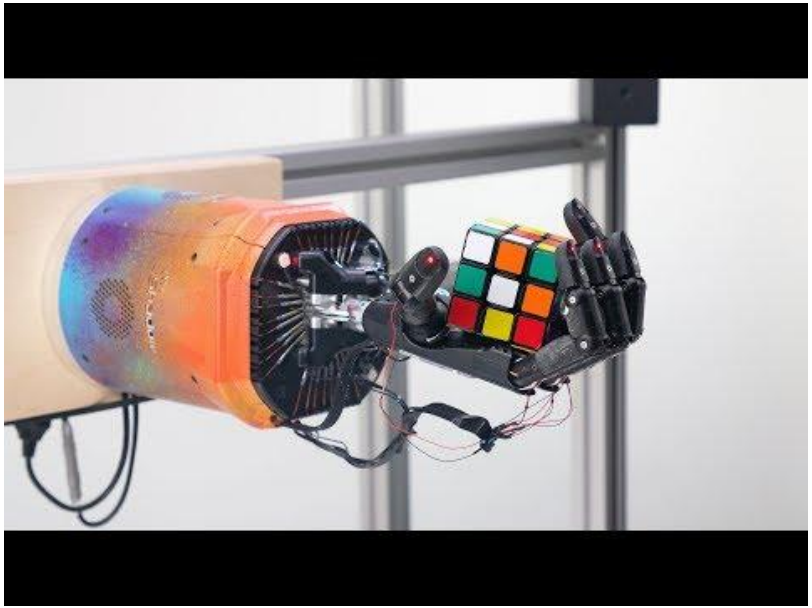
$$D_{KL}(\pi_{\theta'} \| \pi_{\theta})$$

This leads to novel algorithms:

- KL + REINFORCE = **TRPO** or **PPO**
- Entropy + A2C = **ACER** or **IMPALA**
- KL + Actor-Critic = **MPO**

(Schulman et al., 2015; 2017)
(Wang et al., 2016)
(Espeholt et al., 2018)
(Abdolmaleki et al., 2018)

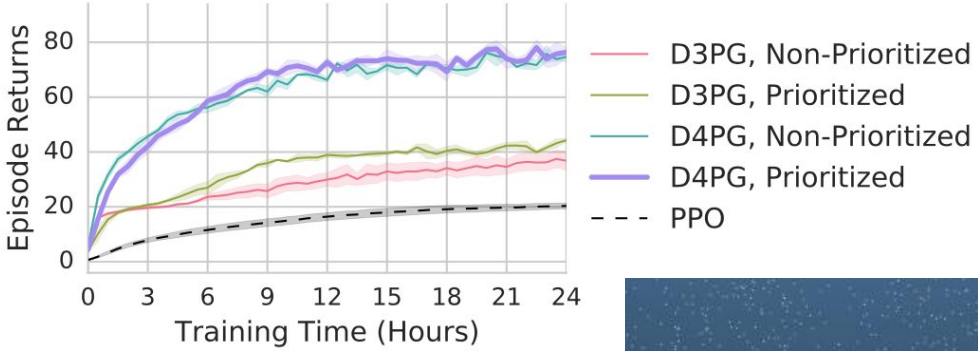
On-policy methods like PPO shine in simulation



(OpenAI et al., 2019)

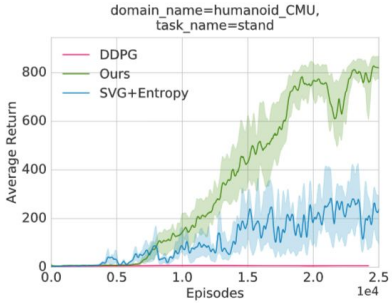
Distributed Distributional DDPG (D4PG)

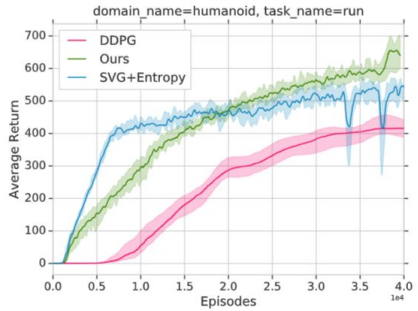
Adds **Distributed** and **Distributional** to DDPG.

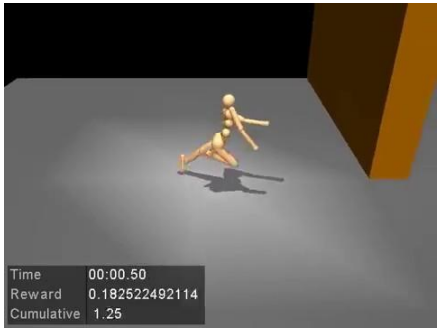


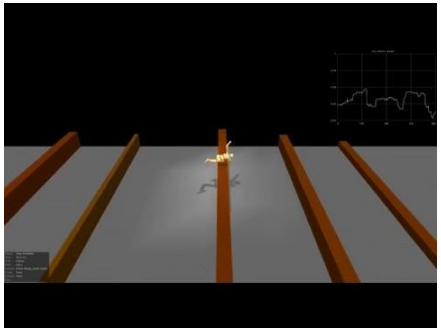
(Barth-Maron et al., 2018)

Maximum a Posteriori Policy Optimisation (MPO)





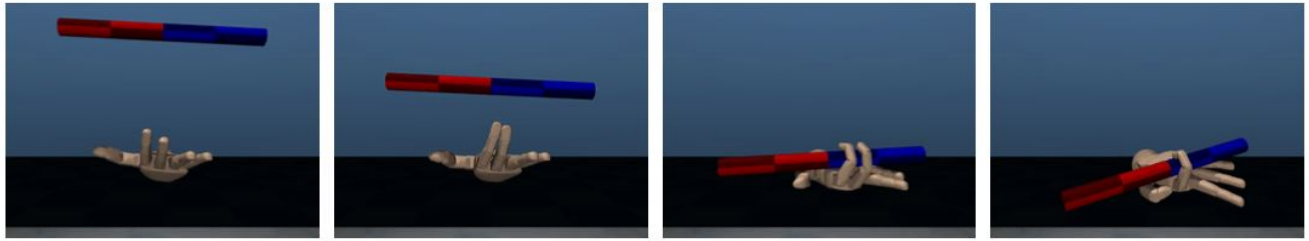




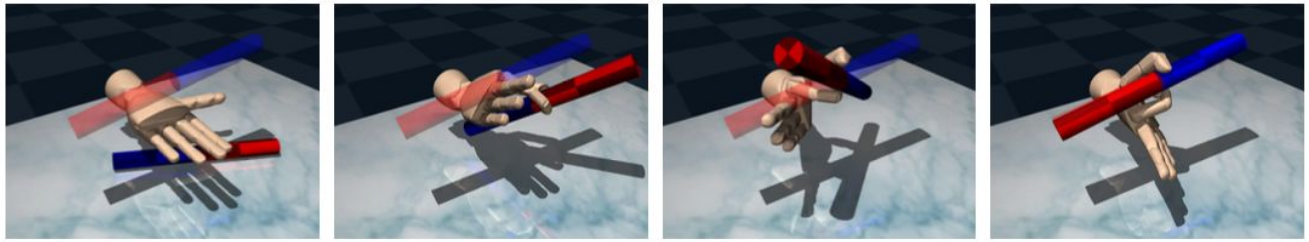
(Abdolmaleki et al., 2018)

Manipulation results

Catch:



Match moving target:



Pickup and Orient:

Overview

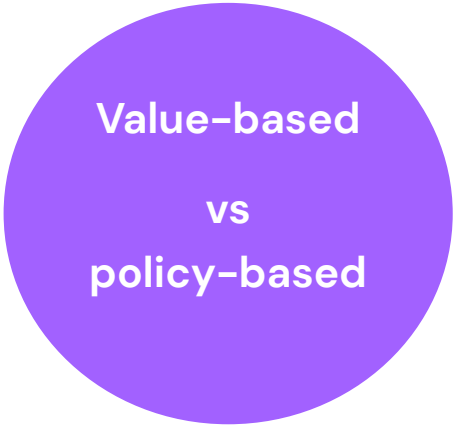
Fundamental Concepts

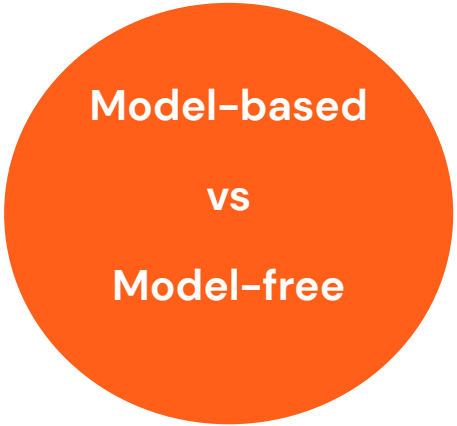
Value-based methods

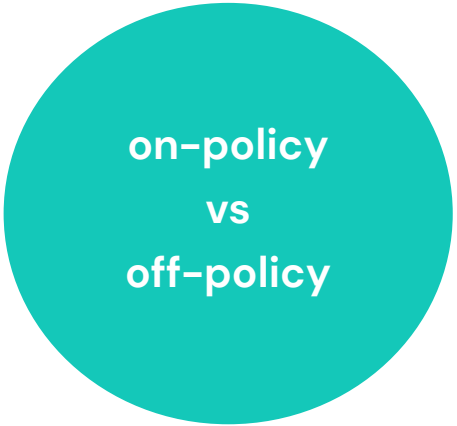
Policy-based methods

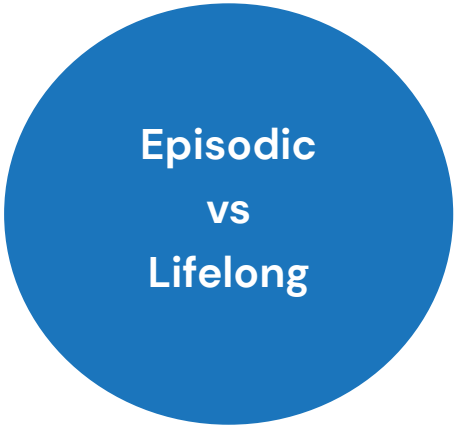
Challenges & opportunities

Plethora of RL methods









...

Real world Reinforcement Learning

Challenges

- Large action and state spaces
- Usually interaction with the environment is **expensive** e.g. robotics
- Reward function is unspecified, risk sensitive or multi-objective
- Large and unknown delays in actuators, sensors and rewards
- Requirements for fast inference
- Exploration
- Generalization

Off-policy vs On-policy RL

- **On-policy RL** is when you assume the data you are learning from
  - (s, a, r, s') are the distribution that you are learning.
- **Off-policy RL** learn from a behavioral distribution that is different from yours.
  - (s, a, r, s') that learn a VF from are from a different behavioral distribution.

Real world Reinforcement Learning

Opportunities

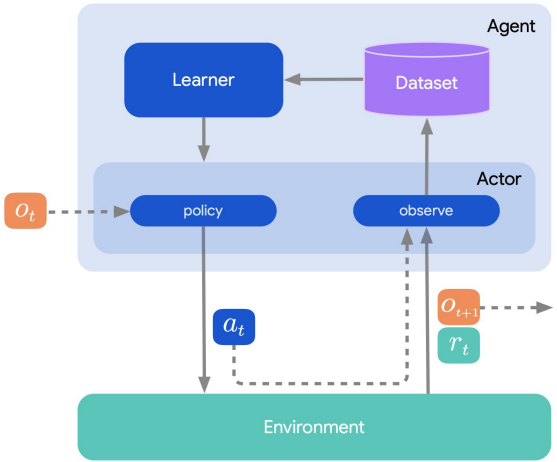
- Offline-RL (e.g. D4RL, RL Unplugged)
- RL Frameworks (e.g. Acme, Dopamine, RLLib, SEED)
- Representation learning (e.g. object based, contrastive losses)
- Temporal abstractions (e.g. options)
- Memory (e.g. transformers, LSTMs, NTM)

Distributed RL

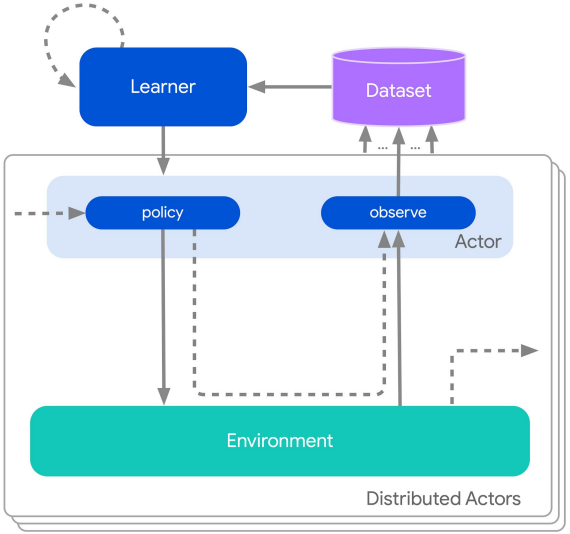
There are often **two loops** in RL: data collection and optimization

- We can exploit this by splitting the loops and running in parallel!

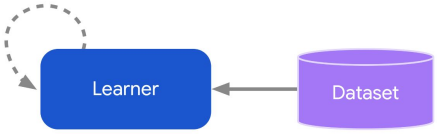
Single-process



Distributed



Batch



DeepMind

@caglarml

Thank you

Special thanks to

Bobak Shahriari

Nando de Freitas

Feryal Behbahani

Matt Hoffman

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