

# REINFORCEMENT LEARNING FOR INFORMATION RETRIEVAL

**Alexander Kuhnle**

**Miguel Aroca-Ouellette**

**John Reid**

**Dell Zhang**

# Your Presenters



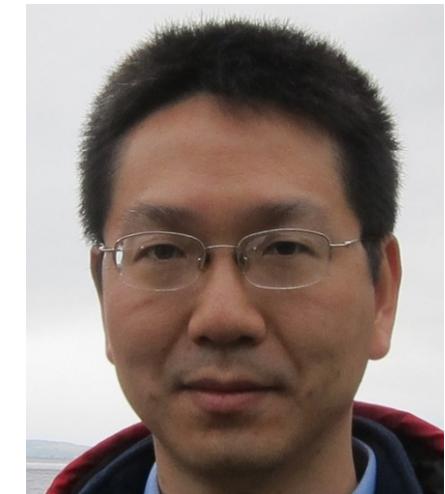
Alexander Kuhnle



Miguel Aroca-Ouellette



John Reid



Dell Zhang

{alexander.kuhnle, miguel.aroca, john.reid, dell.zhang}@blueprism.com

Tutorial Website: [rl-starterpack.github.io](https://rl-starterpack.github.io)

# Schedule

- **09:30-10:30 RL Basics and Tabular Q-Learning**
- 10:30-10:45 Coffee Break
- **10:45-11:45 Deep Q-Network (DQN)**
- 11:45-12:00 Coffee Break
- **12:00-12:30 IR applications using DQN**
- 12:30-14:00 Lunch Break
- **14:00-15:00 Policy Gradient (REINFORCE)**
- 15:00-15:15 Coffee Break
- **15:15-15:45 Actor Critic**
- 15:45-16:00 Coffee Break
- **16:00-17:00 IR applications using REINFORCE**
- **17:00-17:15 Outlook**

# Zoom Housekeeping

- **During presentation:**

- Keep yourself muted
- Do not share your screen
- Ask questions in chat

- **During tutorial exercises:**

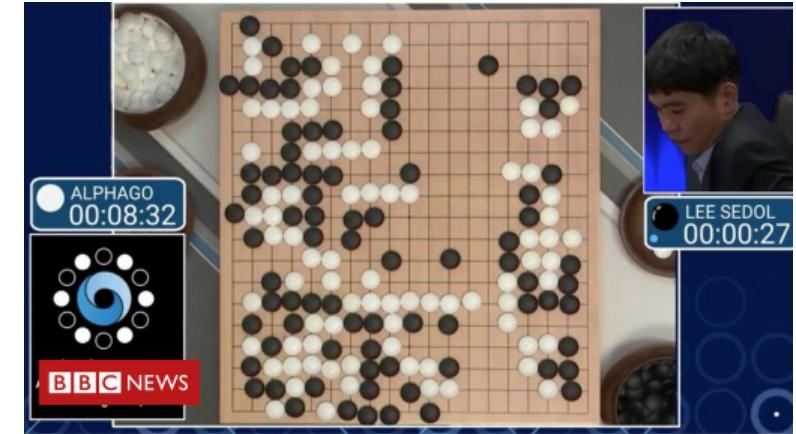
- **If you have a question:** Feel free to unmute to ask, or ask in the chat. You can share your screen as well if that helps explain
- Otherwise please keep yourself muted to keep background noise to a minimum

**Thank you!**

# Why RL for IR?

## Technology Push

- Reinforcement Learning + Deep Learning:  
*Great Successes*
  - AlphaGo, etc.
- Sequential Decision Making:  
*Beyond Independence Relevance*
  - Document Ranking Positions
  - Document Interdependencies



## Demand Pull

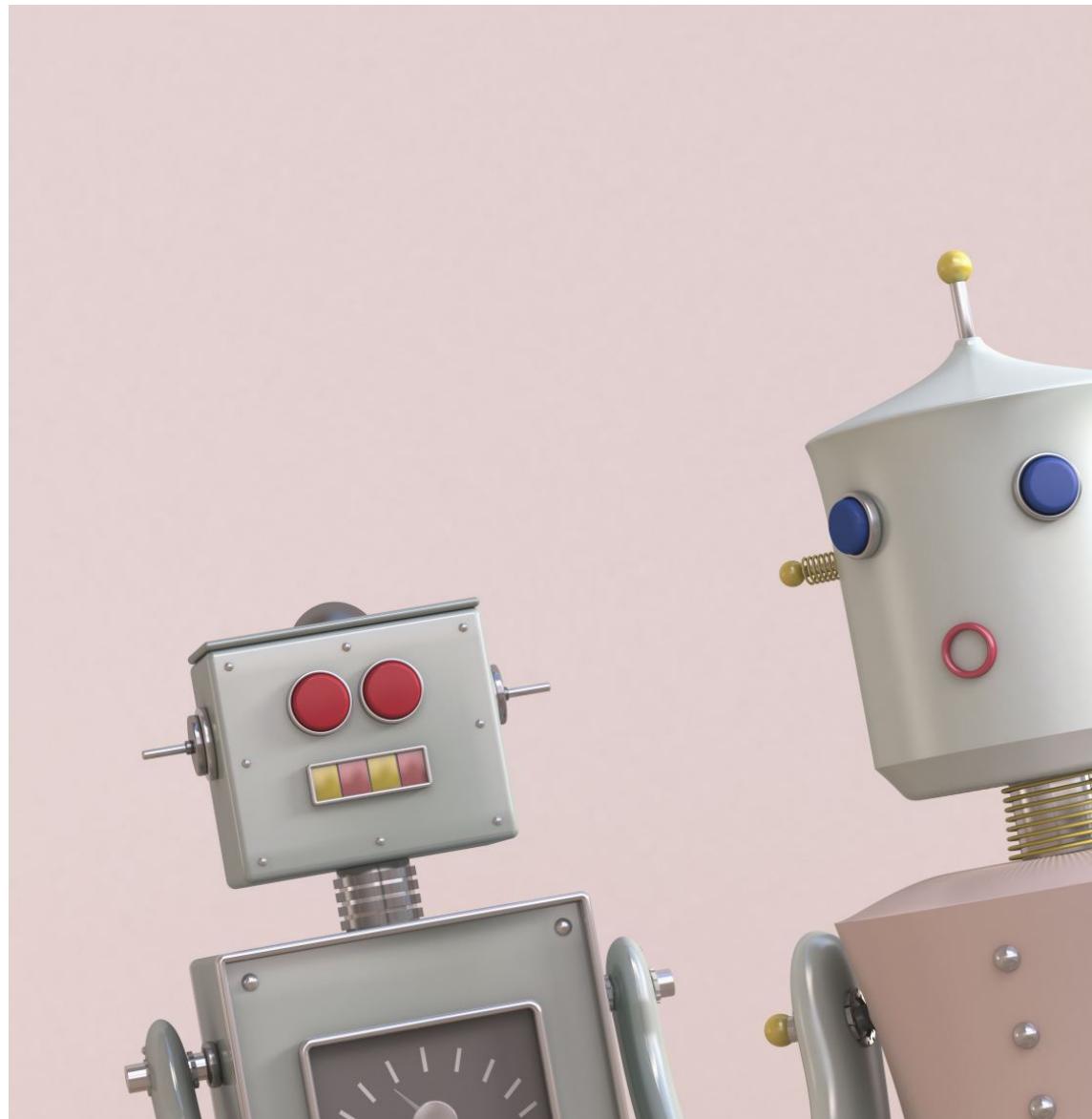
- Web and Mobile IR Applications:  
*Personalised and Interactive*
  - Static User Preferences → Evolving User Interests
  - Immediate User Satisfaction → Long-Term Engagement

"reinforcement learning" source:SIGIR

About 47 results (0.04 sec)

Any time  
Since 2020

KERL: A knowledge-guided **reinforcement** recommendation

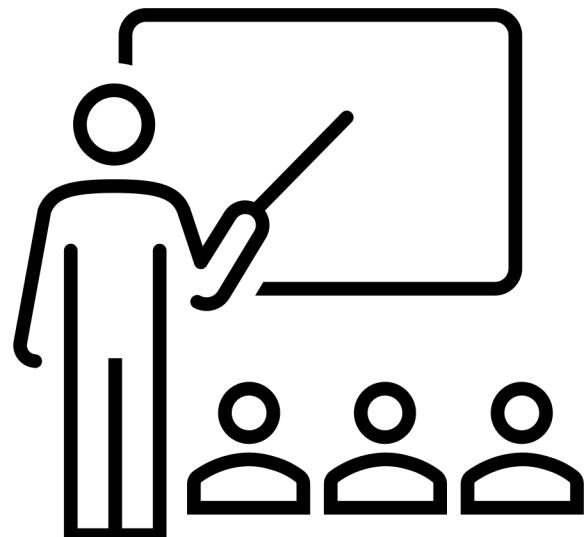


# REINFORCEMENT LEARNING BASICS

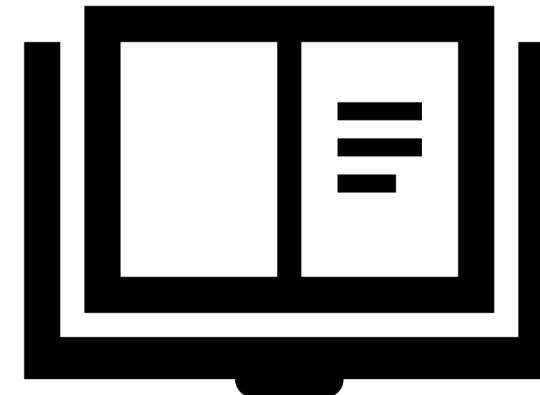
John Reid

# Machine learning

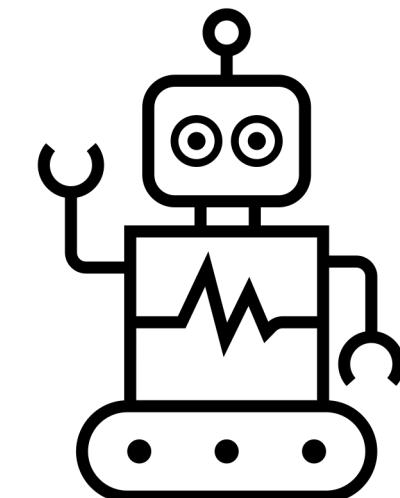
Supervised learning



Unsupervised learning

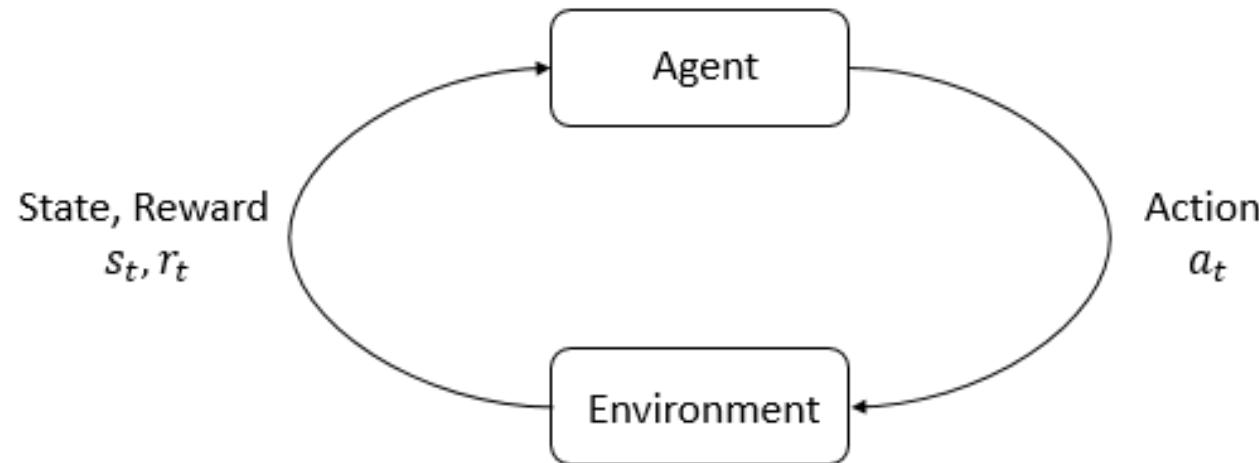


Reinforcement learning



# What is reinforcement learning?

learning what to do  
mapping situations to actions  
maximising reward



# Elements of reinforcement learning

agent

environment

policy

reward signal

value function

model of the environment\*

\* optional

# Characteristics of reinforcement learning

trial-and-error search

exploration versus exploitation

delayed reward

uncertain environment

consider whole problem

on- and off-policy

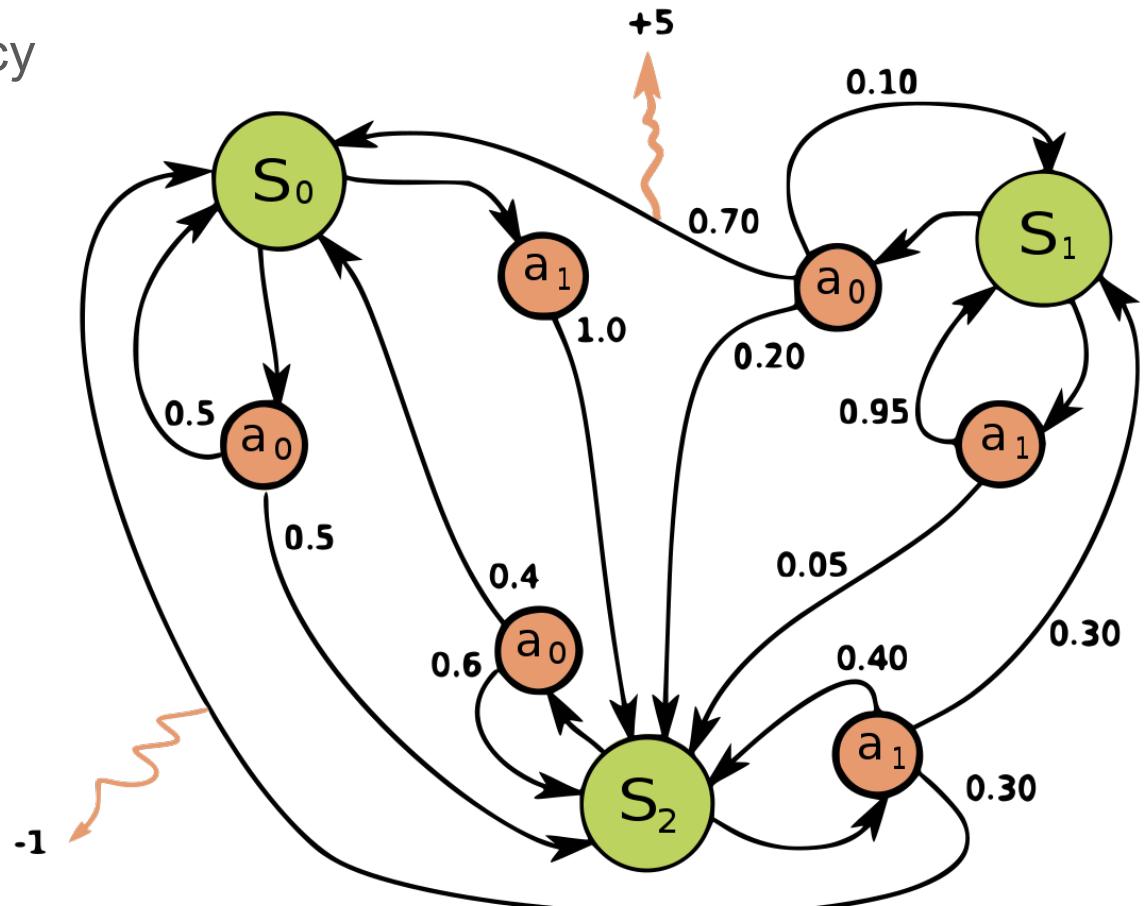
# Markov decision processes

Framework of *discrete-time stochastic control* processes

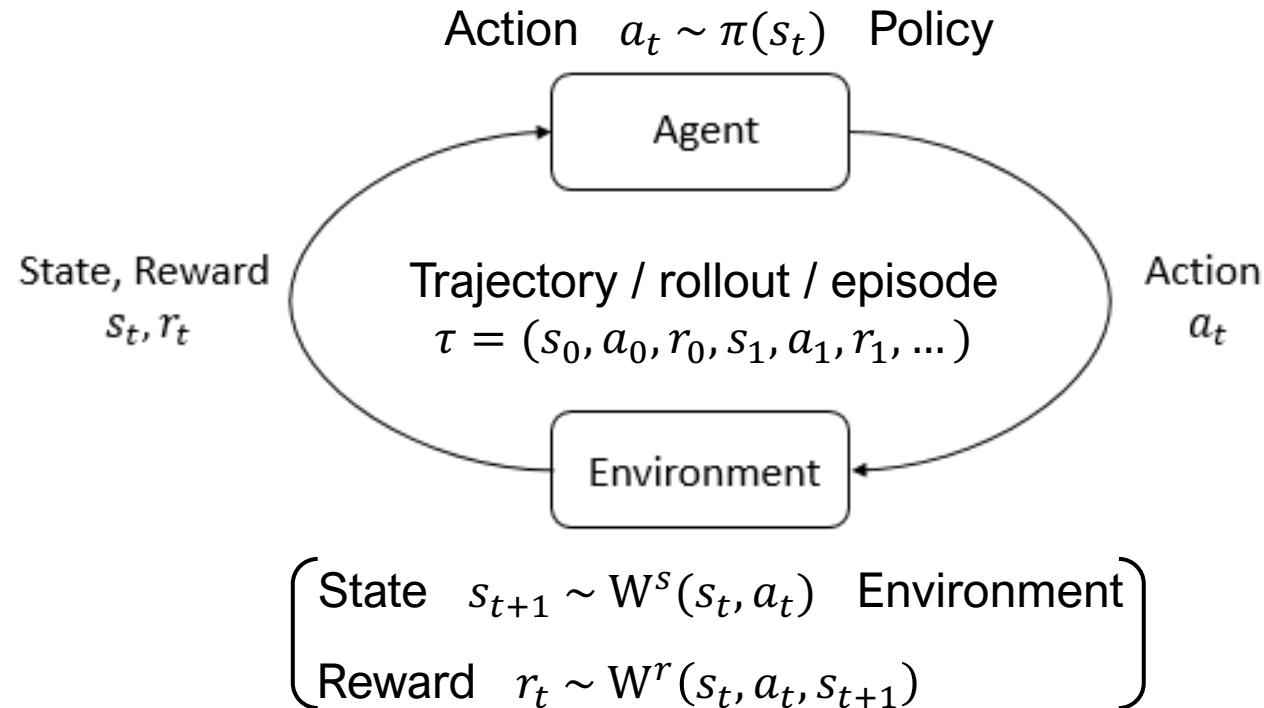
Theory focusses on how to choose a good policy

RL policy typically uses Markov assumption

Extension: *partial observability* (POMDP)



# Interactions with the environment



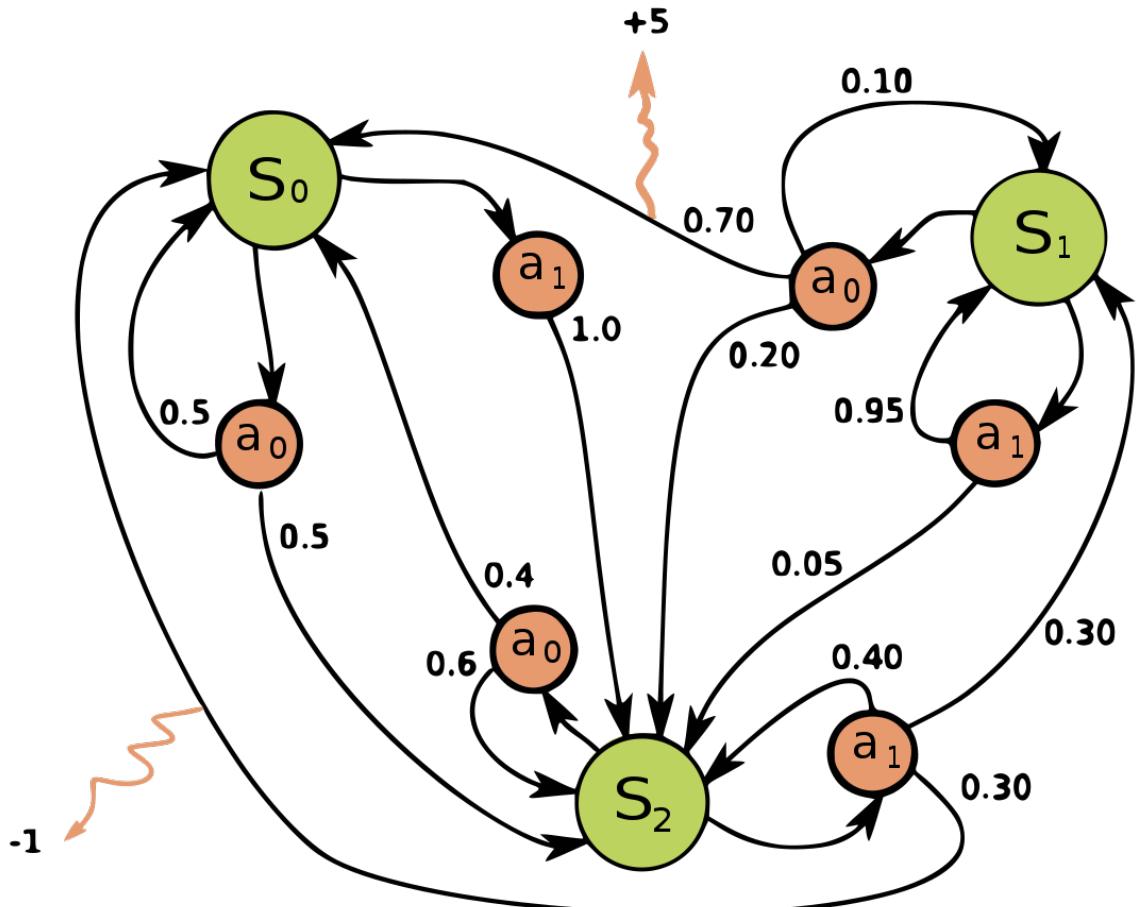
*model-based vs model-free*

# Credit assignment problem

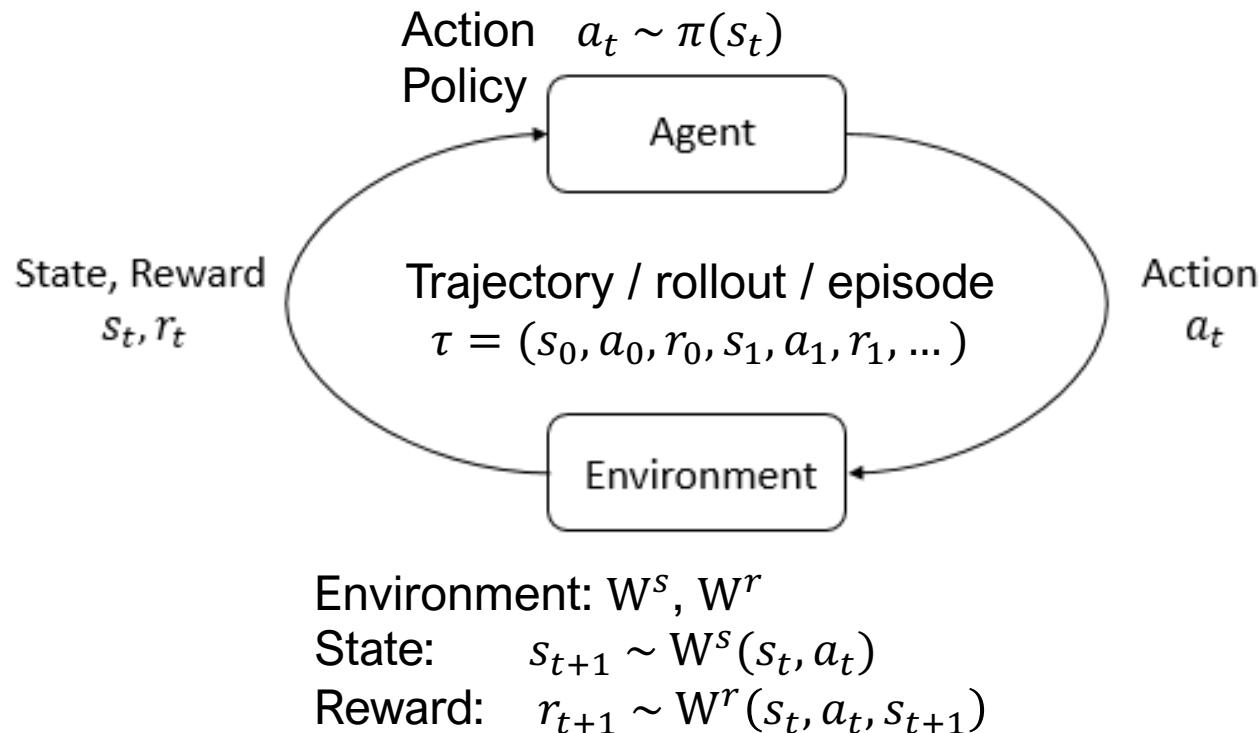
Which actions led to reward?

Not obvious for delayed or sparse rewards

Trial-and-error required



# Rewards and returns



Infinite-horizon discounted return:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

Finite-horizon undiscounted return:

$$R^H(\tau) = \sum_{t=0}^H r_t$$

# Value-based methods

Expectations of returns

State value:

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi}[R(\tau) \mid s_0 = s]$$

State-action / Q-value:

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi}[R(\tau) \mid s_0 = s, a_0 = a]$$

# FrozenLake-v0

*“Winter is here. You and your friends were tossing around a frisbee at the park when you made a wild throw that left the frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you’ll fall into the freezing water. There’s an international frisbee shortage, so it’s imperative that you navigate across the lake and retrieve the disc. However, the ice is slippery, so you won’t always move in the direction you intend.”*

Start	Frozen	Frozen	Frozen
Frozen	Hole -1	Frozen	Hole -1
Frozen	Frozen	Frozen	Hole -1
Hole -1	Frozen	Frozen	Goal +1

# FrozenLake-v0: state value

Start 0.0	Frozen 0.1	Frozen 0.3	Frozen 0.1
Frozen 0.1	Hole -1.0	Frozen 0.4	Hole -1.0
Frozen 0.3	Frozen 0.5	Frozen 0.7	Hole -1.0
Hole -1.0	Frozen 0.7	Frozen 0.9	Goal 1.0

# FrozenLake-v0: state value (model-free!)

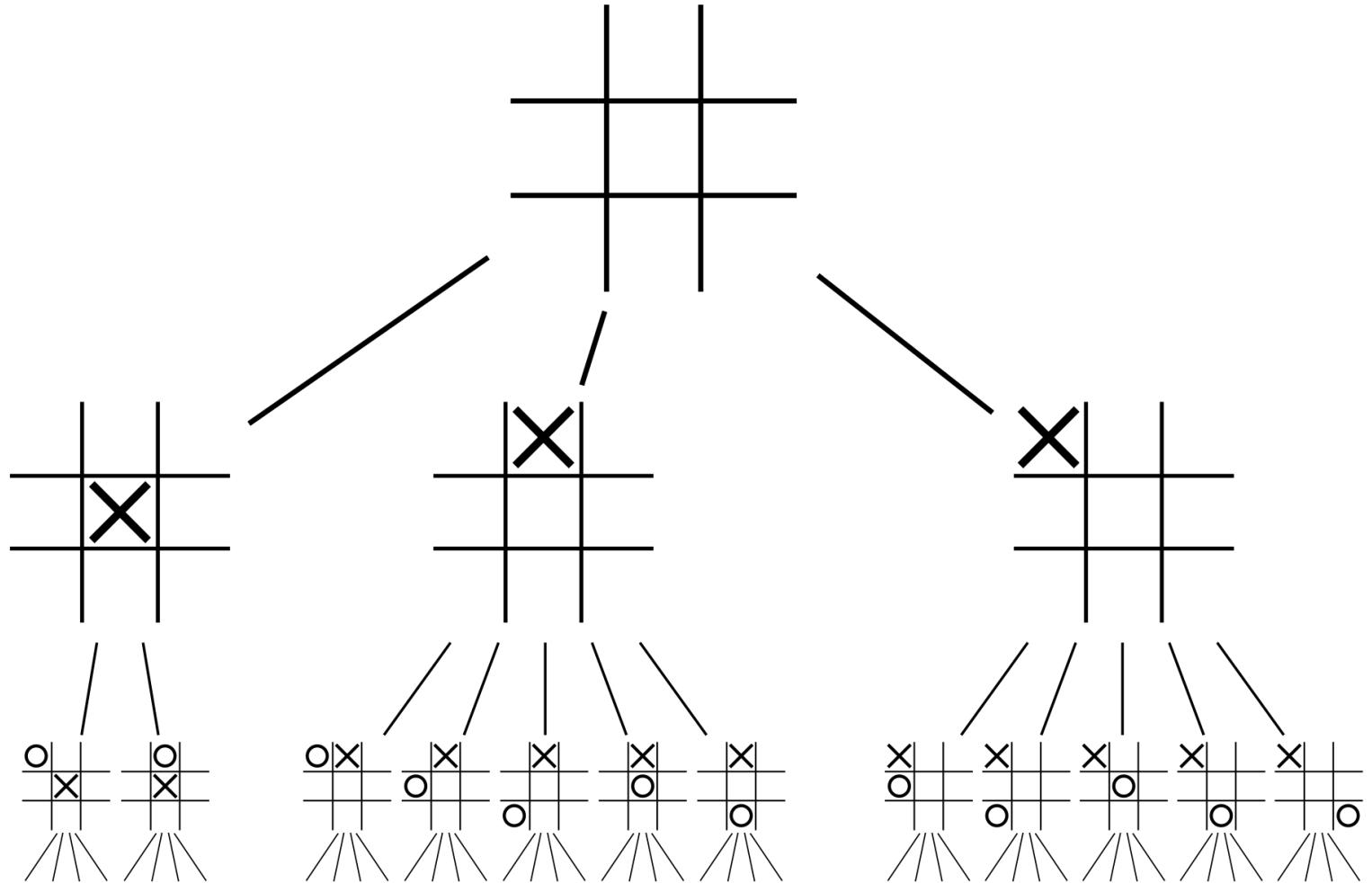
Start 0.0	Frozen 0.1	Frozen 0.3	Frozen 0.1
Frozen 0.1	Hole -1.0	??? ??? Frozen 0.4 ??? ???	Hole -1.0
Frozen 0.3	Frozen 0.5	Frozen 0.7	Hole -1.0
Hole -1.0	Frozen 0.7	Frozen 0.9	Goal 1.0

# Model-based example

Game tree

Policy plans using tree

AlphaGo



Traced by User:Stannered, original by en:User:Gdr - Traced from en:Image:Tic-tac-toe-game-tree.png, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=1877696>

# FrozenLake-v0: state-action / Q-value

0.0 0.0 Start 0.1 0.1			0.0 -0.1 Frozen 0.2 -0.9			0.3 0.1 Frozen 0.1 0.4			0.0 0.2 Frozen 0.0 -0.9		
-0.1 0.0 Frozen -0.9 0.2			Hole -1.0			0.1 -0.9 Frozen -0.9 0.5			Hole -1.0		
0.0 0.2 Frozen 0.4 -0.9			-0.9 0.2 Frozen 0.6 0.6			0.3 0.6 Frozen -0.9 0.8			Hole -1.0		
Hole -1.0			0.4 -0.9 Frozen 0.8 0.6			0.7 0.7 Frozen 1.0 0.9			Goal 1.0		

# FrozenLake-v0: state-action / Q-value

0.0 0.0 Start 0.1 0.1			-0.1 Frozen 0.2 -0.9	0.1 Frozen 0.4 0.3	0.2 Frozen -0.9 0.0
-0.1 0.0 Frozen -0.9 0.2			Hole -1.0	0.1 Frozen 0.5 -0.9	Hole -1.0
0.0 0.2 Frozen 0.4 -0.9			-0.9 Frozen 0.6 0.6	0.3 Frozen 0.8 -0.9	Hole -1.0
Hole -1.0			-0.9 Frozen 0.8 0.6	0.7 Frozen 1.0 0.9	Goal 1.0

# Q-learning

Value-based algorithm:

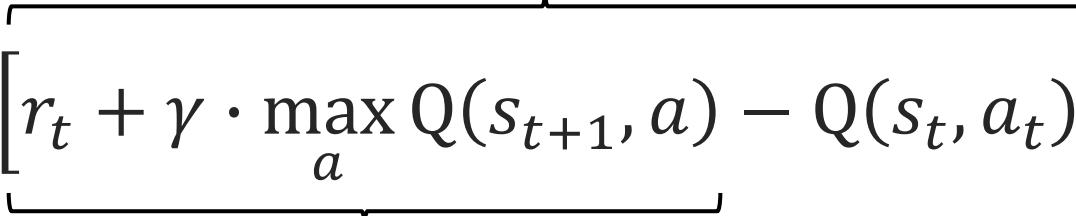
- Learn the Q-value function  $Q(s, a)$
- Define (deterministic) policy based on value function:  $\pi(s) = \arg \max_a Q(s, a)$

Assumption: discrete action space

Temporal difference learning:

$$Q(s_t, a_t) := Q(s_t, a_t) + \lambda \cdot \underbrace{\left[ r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]}_{\text{TD target}}$$

TD residual



# Bellman equation

When using infinite-horizon discounted returns

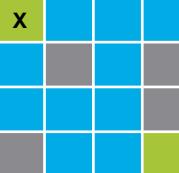
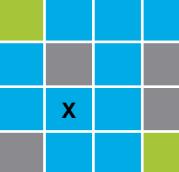
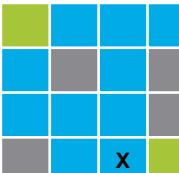
$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

the Bellman equation gives us a recursive definition of the value-function

$$\begin{aligned} V^\pi(s_0) &= E_{\tau \sim \pi(s_0)}[R(\tau)] \\ &= E_{\tau \sim \pi(s_0)}[r_0 + \gamma V^\pi(s_1)] \end{aligned}$$

So  $r_0 + \gamma V^\pi(s_1)$  is an unbiased estimator of  $V^\pi(s_0)$ , which inspires temporal difference learning.

# Tabular Q-learning

		right	down	left	up
• 1:	 A 4x4 grid world state. The agent is at position (1,1) marked with an 'x'. The goal is at (4,4). The grid contains obstacles at (1,3), (2,3), and (3,3). Action values: right = 0.1, down = 0.1, left = 0.0, up = 0.0.	0.1	0.1	0.0	0.0
•	...	...	...	...	...
• 9:	 A 4x4 grid world state. The agent is at position (1,1) marked with an 'x'. The goal is at (4,4). The grid contains obstacles at (1,3), (2,3), and (3,3). Action values: right = 0.6, down = 0.6, left = 0.2, up = -0.9.	0.6	0.6	0.2	-0.9
•	...	...	...	...	...
• 15:	 A 4x4 grid world state. The agent is at position (1,1) marked with an 'x'. The goal is at (4,4). The grid contains obstacles at (1,3), (2,3), and (3,3). Action values: right = 1.0, down = 0.9, left = 0.7, up = 0.7.	1.0	0.9	0.7	0.7

# Exploration vs exploitation

0.0 0.0 0.0	Start	0.0	0.0 Frozen -1.0	? Frozen ?	Frozen	Frozen
0.0 ? Frozen -1.0 ?	Frozen	Hole		Frozen	Hole	Hole
Frozen		Frozen		Frozen	Hole	Hole
Hole		Frozen		Frozen	Goal	Goal

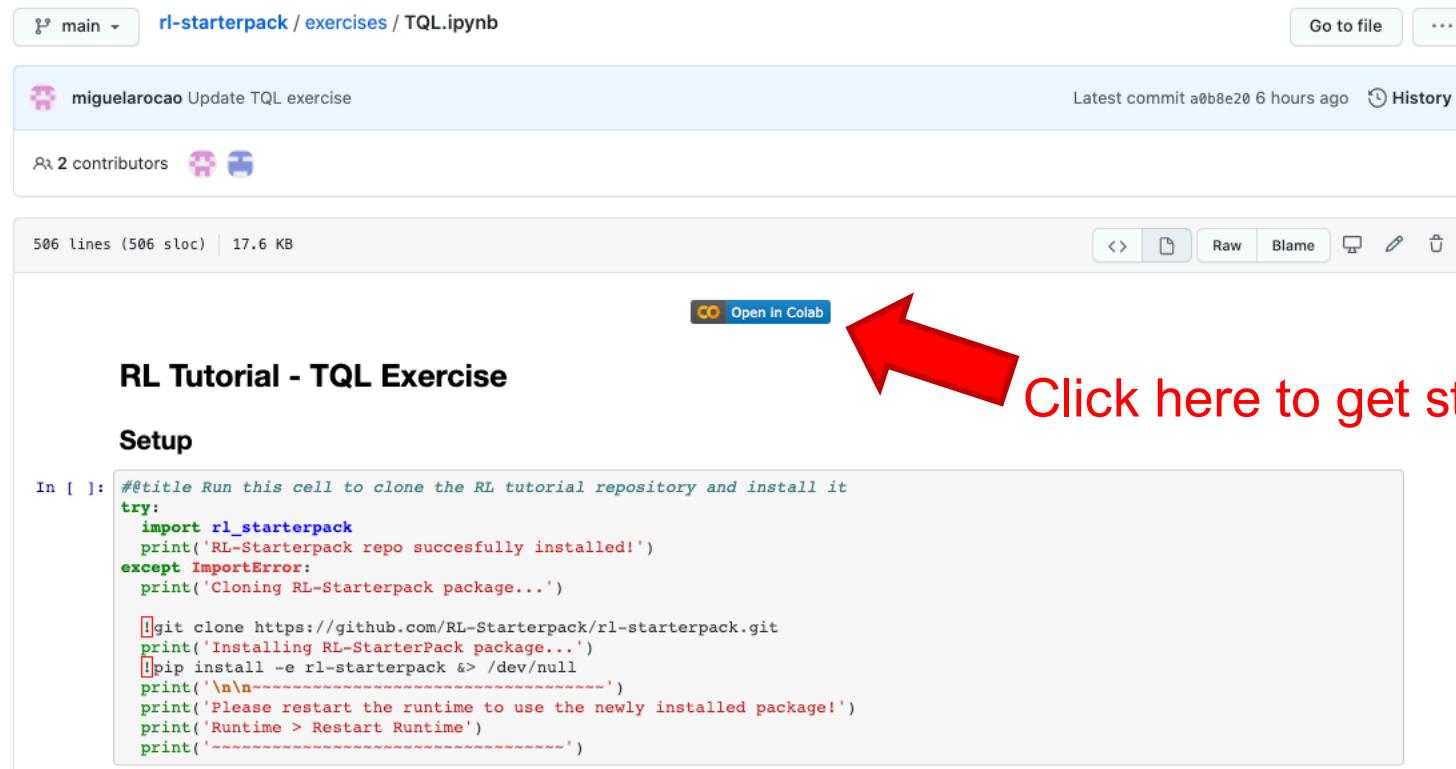
# Reward shaping

0.0 0.0 Start 0.0	?	0.0 Frozen ?	0.0 Frozen ?	0.0 Frozen ?
0.0 0.0 Frozen 0.0	Hole 0.0		Frozen	Hole 0.0
? ? Frozen 0.0	Frozen		Frozen	Hole
Hole	Frozen		Frozen	Goal

# TQL Exercise

Access from Quick Links @ [rl-starterpack.github.io/#quick-links](https://rl-starterpack.github.io/#quick-links)

Direct Link: <https://github.com/RL-Starterpack/rl-starterpack/blob/main/exercises/TQL.ipynb>



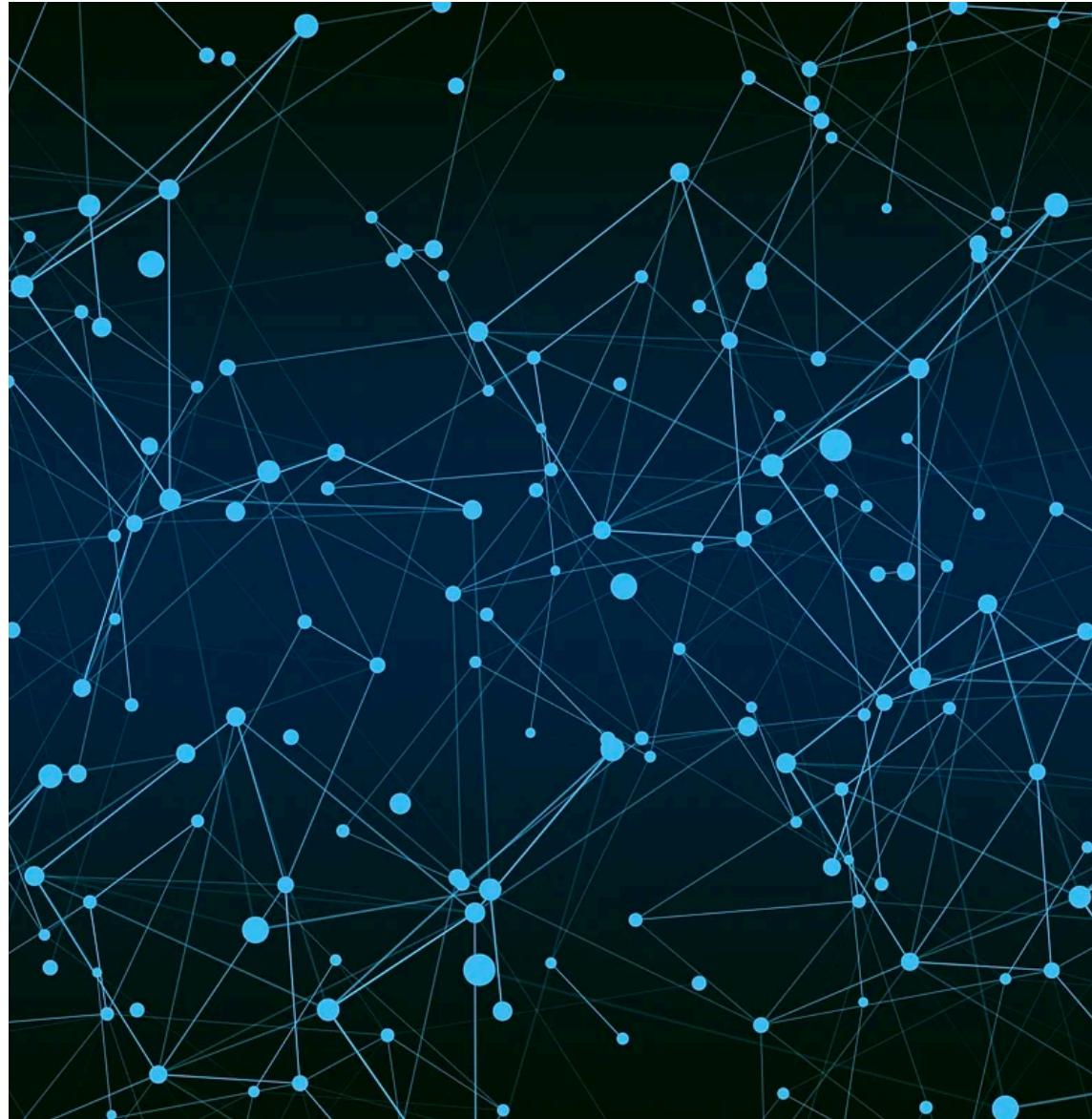
The screenshot shows a GitHub repository page for `TQL.ipynb`. The page includes a header with the repository name, a commit history showing a recent update by `miguelarocao`, contributor information, and file statistics. Below the header is a toolbar with options like `<>`, `Raw`, and `Blame`. A prominent red arrow points to the `Open in Colab` button in the toolbar. To the right of the arrow, the text `Click here to get started in Colab!` is displayed.

`RL Tutorial - TQL Exercise`

`Setup`

```
In [ ]: #@@title Run this cell to clone the RL tutorial repository and install it
try:
    import rl_starterpack
    print('RL-Starterpack repo successfully installed!')
except ImportError:
    print('Cloning RL-Starterpack package...')

    !git clone https://github.com/RL-Starterpack/rl-starterpack.git
    print('Installing RL-StarterPack package...')
    !pip install -e rl-starterpack &> /dev/null
    print('\n\n-----')
    print('Please restart the runtime to use the newly installed package!')
    print('Runtime > Restart Runtime')
    print('-----')
```

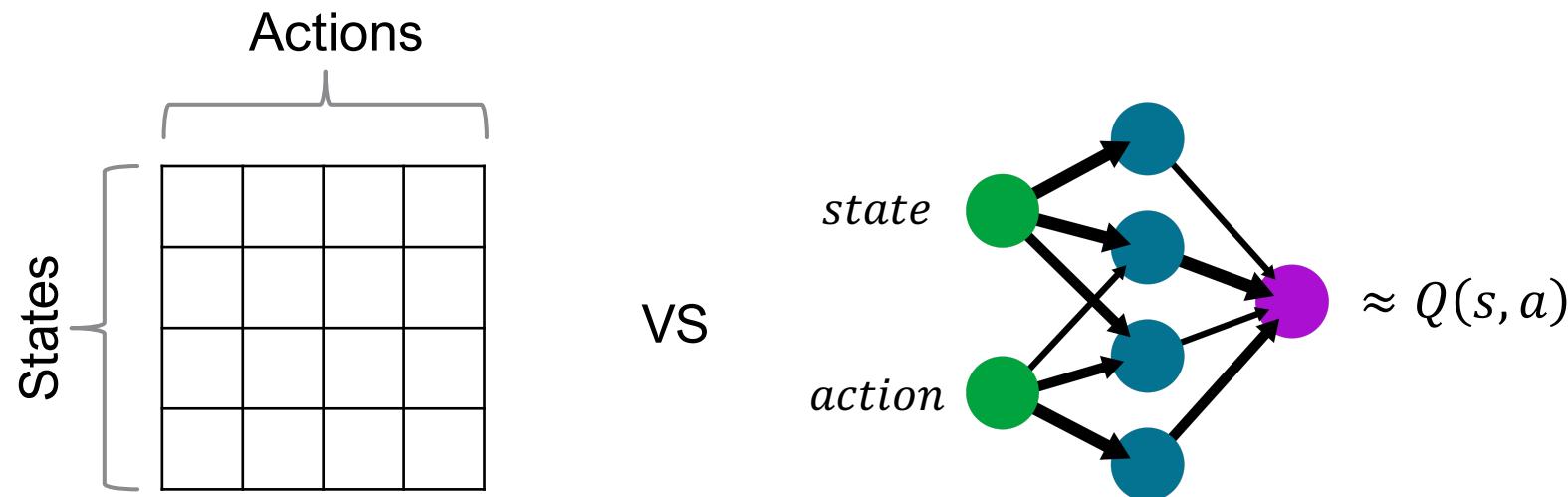


# DEEP Q-LEARNING

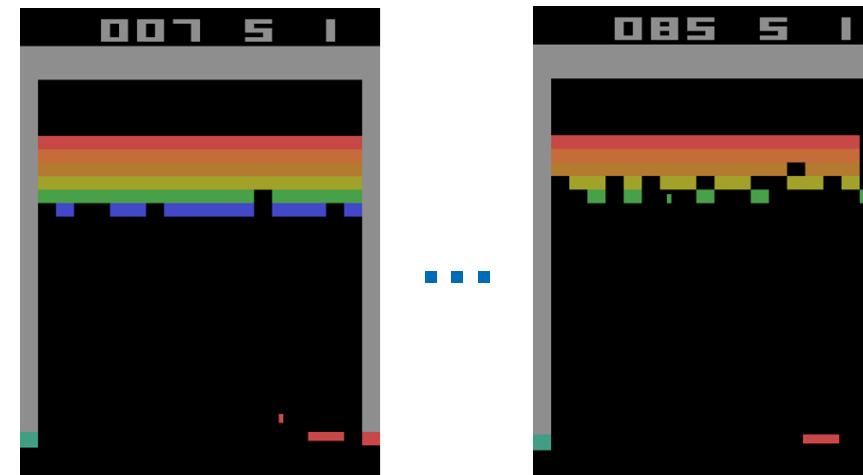
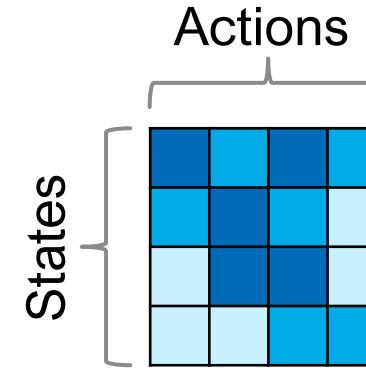
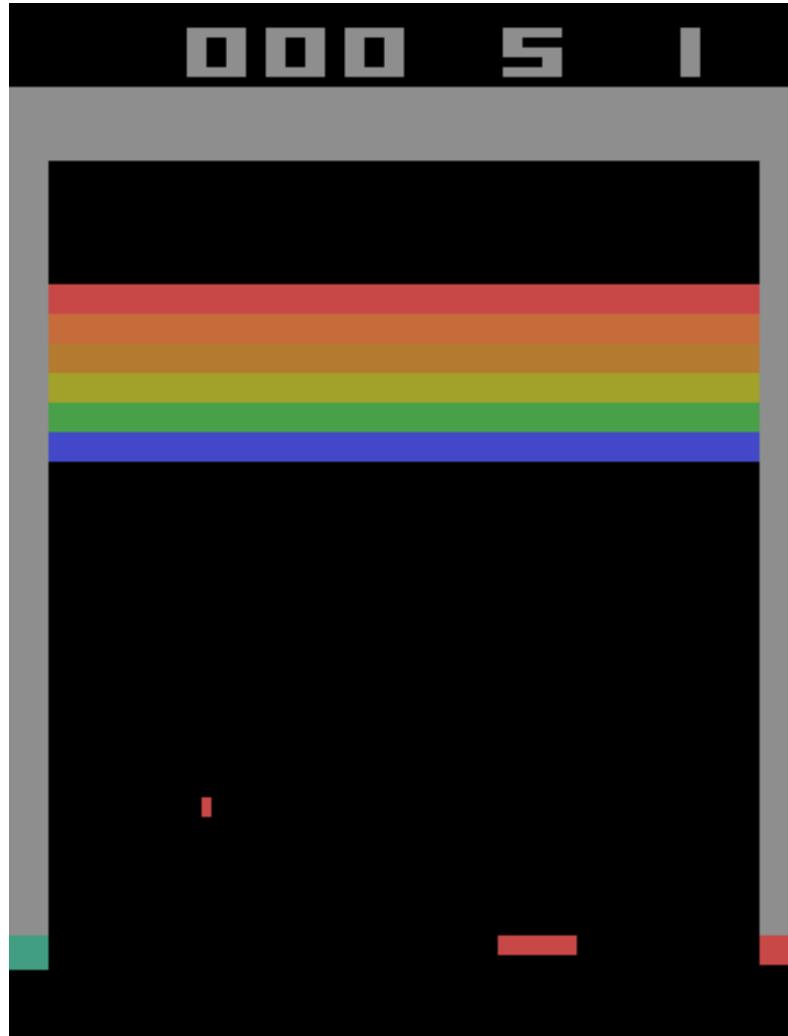
Miguel Aroca-Ouellette

# Deep Q-learning Networks Motivation

- In TQL we used a table to store the Q-values for our (state, action) pairs.
- Now imagine the scenario where we have 10,000 states and 10,000 actions per state. We would then need to **maintain a table with 100 million entries**.
- Can we instead **learn a function** which takes in the state and action and outputs the corresponding Q-value?
- Deep Q-learning Networks (**DQN**): use **deep neural networks** for this mapping function

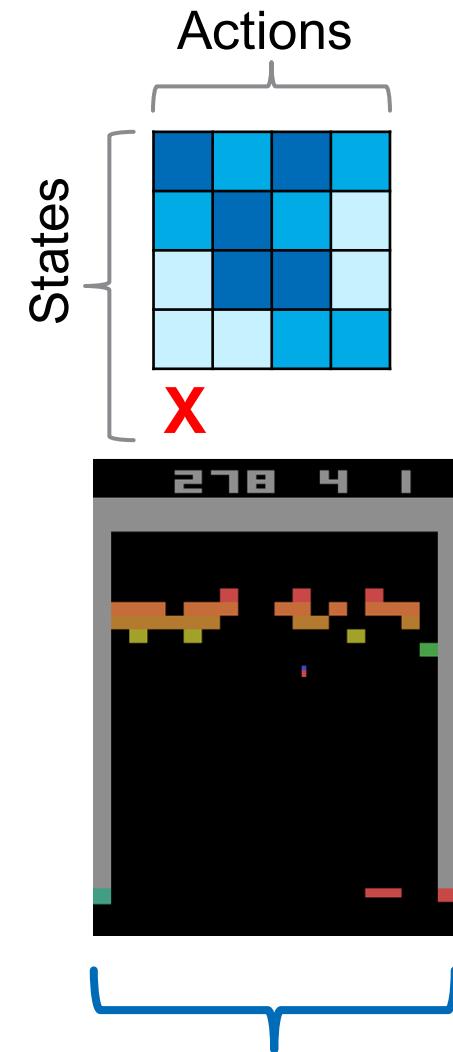
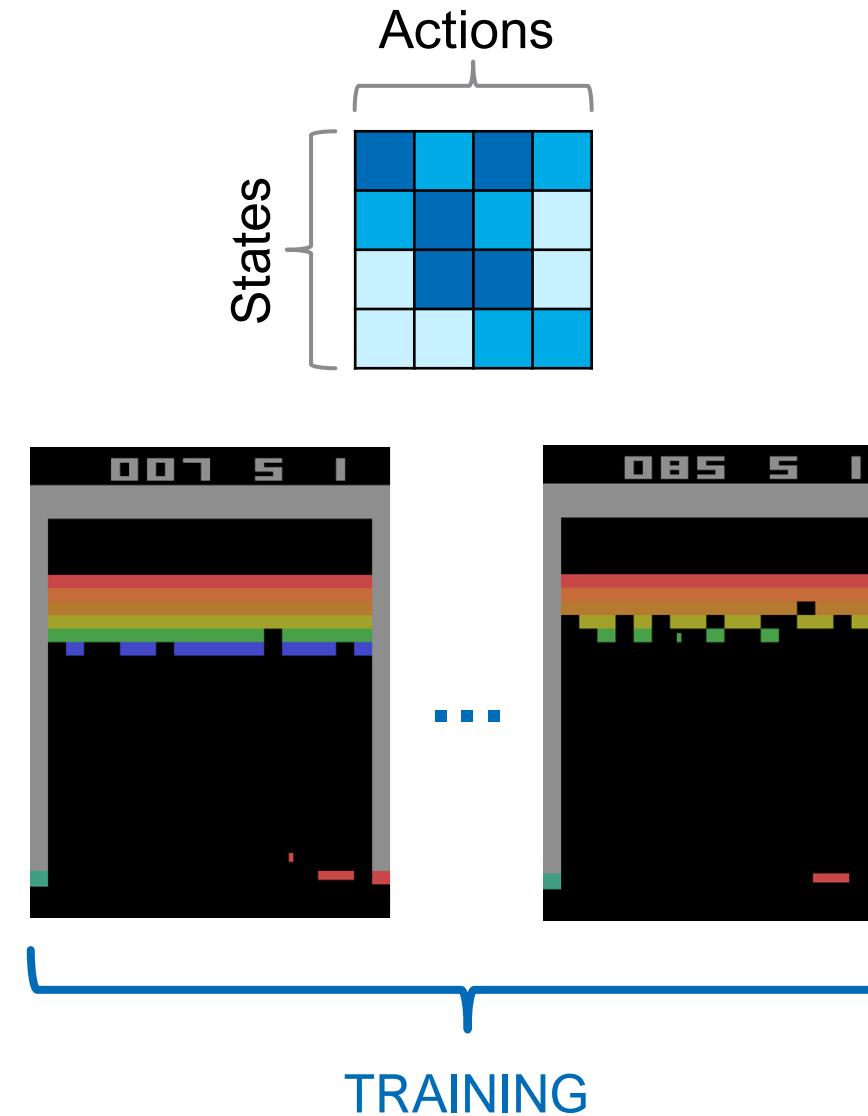
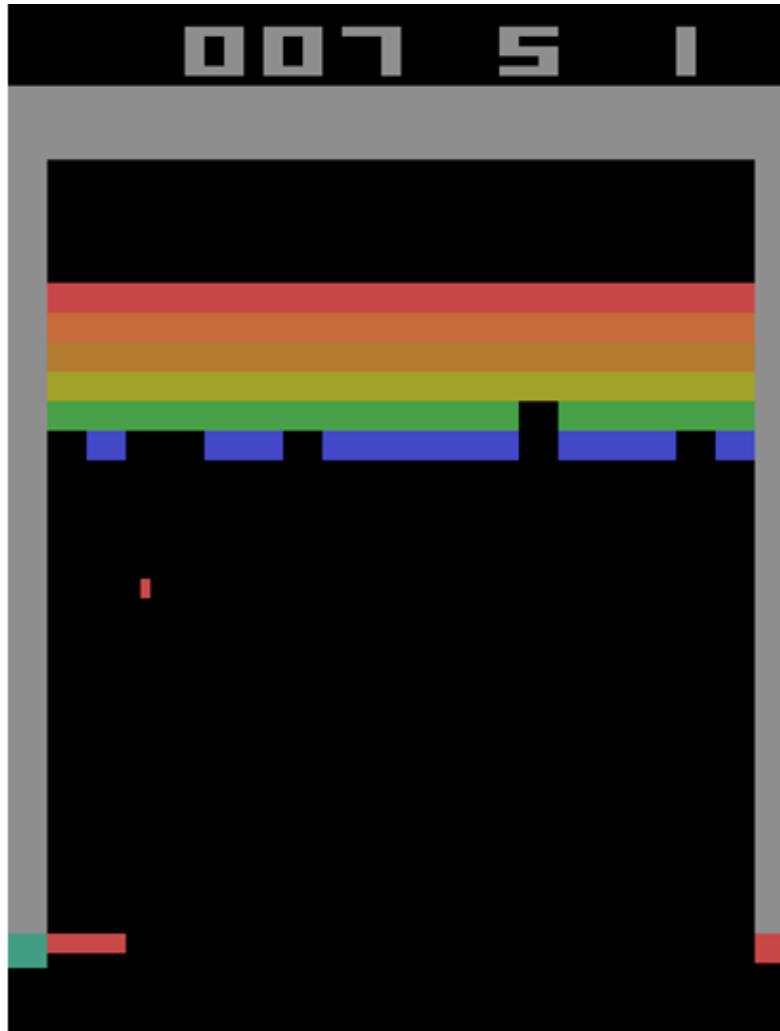


# Deep Q-learning Networks Motivation



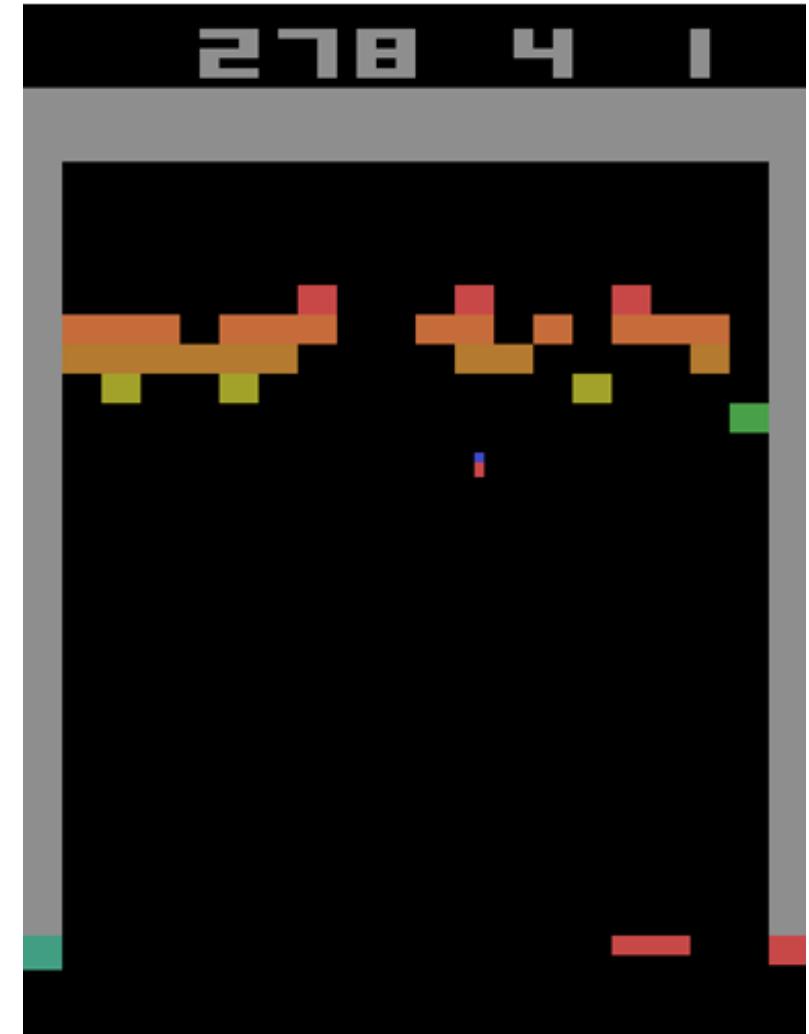
TRAINING

# Deep Q-learning Networks Motivation



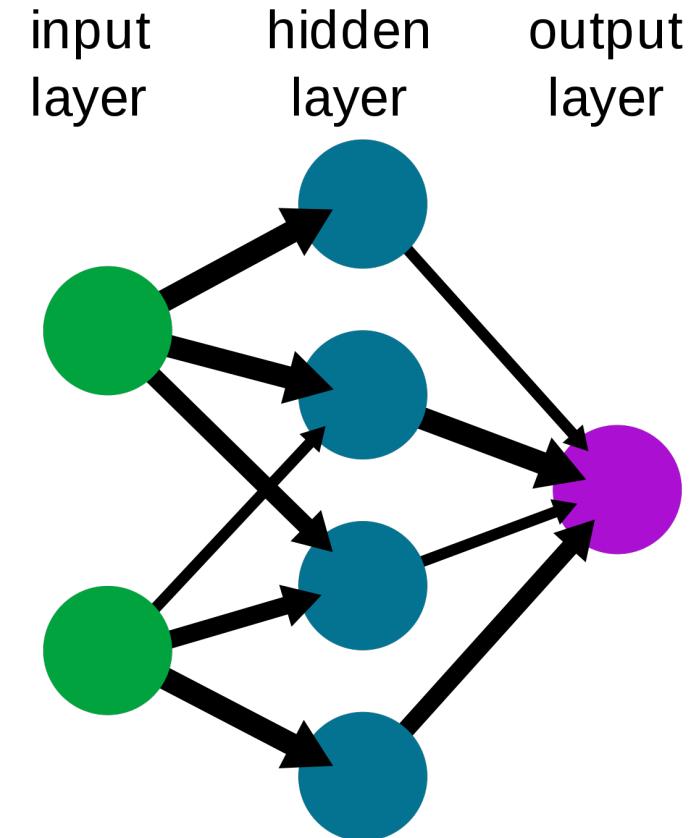
# Deep Q-learning Networks Motivation

- DQNs also allow us to handle states which are not easily represented as table entries, such as images
- By leveraging neural networks in our DQN we can hope to:
  - Handle **previously unseen states**
  - Handle **more complex states**
  - Take advantage of similarities between states



# Single-slide recap: Neural Networks

- Information flows from left to right via nodes (neurons) and connections.
- The output of a node is computed as a weighted sum of its inputs which is then passed through a non-linearity
- When trained with enough data **can represent a wide variety of functions**
- The weights in the network are **trained to minimize a loss function** on its output. This is done via **gradient descent**



# DQN – In Practice

- In practice, instead of using our function to estimate  $Q(s, a)$  for a single state-value pair, we instead have the function **estimate  $Q(s, a)$  for all actions of a given state**

$$f(s) = [Q(s, a_1), \dots, Q(s, a_n)]$$

- This allows us to call our network only once per state (instead of once per state-action pair)
- During **inference** we perform the **action  $i$  corresponding to the maximum  $Q(s, a_i)$**  in this vector. If there are multiple maxima, we randomly break ties
- During **training** we use an  **$\epsilon$ -greedy algorithm to balance exploration and exploitation**
  - With probability ( $\epsilon$ ) select an action at random
  - With probability ( $1 - \epsilon$ ) select the optimal action (as during inference)

# DQN Training

- To train our DQN we use temporal-difference training. Recall:

$$Q(s_t, a_t) := Q(s_t, a_t) + \lambda \cdot \underbrace{\left[ r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]}_{\text{TD target}}$$

TD residual

The diagram illustrates the temporal-difference residual. It shows a horizontal bracket under the equation, with its top end labeled "TD residual" and its bottom end labeled "TD target". The bracket encloses the term  $r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ .

- To train our network we simply use **gradient descent to minimize the TD-residual**
- Intuition: The TD-target is a better estimate of our Q-value since it accounts for the reward and is closer to the terminal state (where the Q-value is known)

# DQN Pros and Cons

## Pros

- Compresses Q-table for data
- Can handle states not seen during training
- Q-networks act as feature extractors which allows for generalization of Q-values
- Can leverage both on- and off-policy updates for sample-efficiency

## Cons

- Deterministic: cannot learn stochastic policies
- Cannot directly be applied to continuous action spaces (need to discretize)
- Need to separately add  $\epsilon$ -greedy algorithm to balance exploration vs exploitation

# DQN Extensions

## Experience replay

- Decouple update batches from on-policy experience stream to reduce correlation
- Save transitions  $(s_t, a_t, r_t, s_{t+1})$  to buffer
- Randomly sample from replay buffer and apply Q-learning update

## Target network

- Use a separate Q-network to estimate TD-target
- Target network is synced infrequently with main network
- Reduce correlation between Q-value and TD-target

$$Q(s_t, a_t) := Q(s_t, a_t) + \lambda \cdot \underbrace{\left[ r_t + \gamma \cdot \max_a Q^{\text{target}}(s_{t+1}, a) - Q(s_t, a_t) \right]}_{\text{TD target}}$$

# DQN Extensions

## Double DQN

- Unlike Target Network approach use action from online network, but value from target network

$$Q(s_t, a_t) := Q(s_t, a_t) + \lambda \cdot [r_t + \gamma \cdot V^{\text{target}} - Q(s_t, a_t)]$$

$$V^{\text{target}} = Q^{\text{target}} \left( s_{t+1}, \arg \max_a Q(s_{t+1}, a) \right)$$

## And More!

- Prioritized instead of random experience replay memory
- Dueling DQN: Split estimating  $Q(s_t, a_t)$  into estimating  $V(s_t) + A(s_t, a_t)$
- and many more...

# DQN Exercise

Access from Quick Links @ [rl-starterpack.github.io/#quick-links](https://rl-starterpack.github.io/#quick-links)

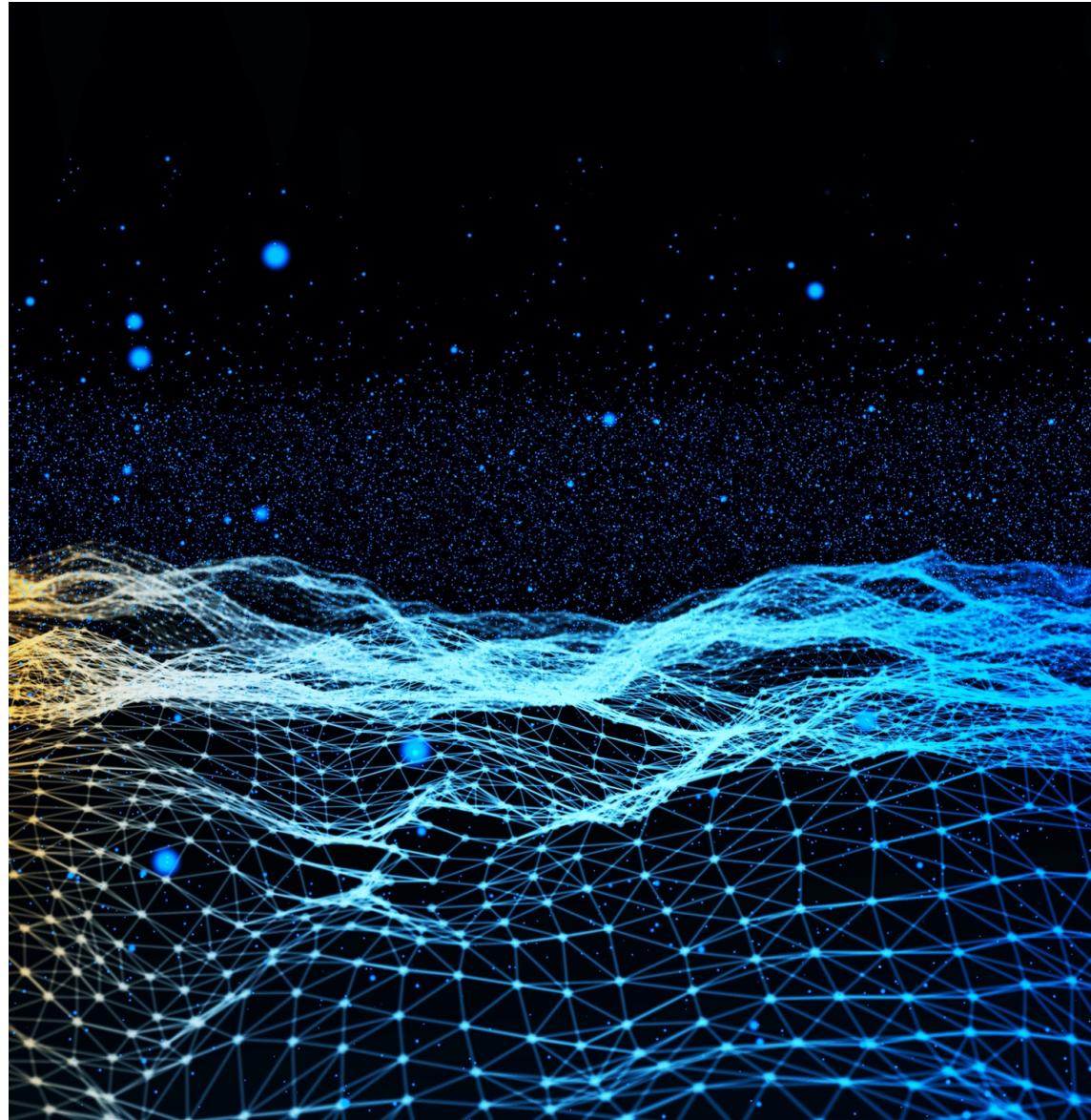
Direct Link: <https://github.com/RL-Starterpack/rl-starterpack/blob/main/exercises/DQN.ipynb>

The screenshot shows a GitHub repository page for 'rl-starterpack / exercises / DQN.ipynb'. At the top, there's a navigation bar with 'main' and a dropdown, followed by the repository path and file name. To the right are 'Go to file' and '...' buttons. Below the header, there's a commit history section with one commit by 'miguelarocao' that says 'Update DQN exercise'. It shows '2 contributors'. The main content area displays the Jupyter notebook code. A red arrow points to a blue 'Open in Colab' button located at the top right of the code editor. The code itself is a setup script for cloning the repository and installing it in Google Colab.

```
In [ ]: #@title Run this cell to clone the RL tutorial repository and install it
try:
    import rl_starterpack
    print('RL-Starterpack repo successfully installed!')
except ImportError:
    print('Cloning RL-Starterpack package...')

!git clone https://github.com/RL-Starterpack/rl-starterpack.git
print('Installing RL-StarterPack package...')
!pip install -e rl-starterpack &> /dev/null
print('\n\n-----')
print('Please restart the runtime to use the newly installed package!')
print('Runtime > Restart Runtime')
print('-----')
```

Click here to get started in Colab!



# POLICY GRADIENT METHODS

Miguel Aroca-Ouellette

# Policy Gradient Motivation

- Q-learning **cannot handle continuous action spaces**
  - Due to needing  $\underset{a}{\operatorname{argmax}} Q(s_t, a_t)$  to select an optimal action
- Q-learning **does not allow for a stochastic policy**
  - This would allow us to balance exploration and exploitation based on our uncertainty
- Q-learning **does not directly optimize the policy**
  - The learned Q-values are used within a greedy policy

**Let's try to optimize the policy directly instead!**

# Policy Gradient

- In Policy Gradient methods we **directly use a parametrized function** as the policy:

$$\pi(a|s, \theta) \equiv \pi_\theta(a|s)$$

- We train the policy by optimising a maximisation problem. We want to maximise a measure of the performance  $J(\theta)$

$$\theta_{n+1} = \theta_n + \alpha \nabla J(\theta_n)$$

- For example,  $J(\theta)$  could be our expected return given the current policy parameters

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

# Policy Gradient

- Maximize  $E_{\tau \sim \pi_\theta}[R(\tau)]$  with  $R(\tau) = \sum_{t=0}^{\infty} \gamma^t \cdot r_t$

$$\nabla_\theta E_{\tau \sim \pi_\theta}[R(\tau)] = \nabla_\theta \int R(\tau) \cdot \pi_\theta(\tau) d\tau$$

- Computing the gradient is tricky because it depends on both:
  - The action selection (directly depends on  $\pi_\theta$ )
  - The trajectory rewards (indirectly depends on  $\pi_\theta$ )
- Fortunately, the **Policy gradient theorem** tells us:

$$\nabla_\theta E_{\tau \sim \pi_\theta}[R(\tau)] \propto E_{\tau \sim \pi_\theta}[R(\tau) \cdot \nabla_\theta \log \pi_\theta(a|s)]$$

# Policy Gradient: REINFORCE

$$\nabla_{\theta} E_{\tau \sim \pi_{\theta}}[R(\tau)] \propto E_{\tau \sim \pi_{\theta}}[R(\tau) \cdot \nabla_{\theta} \log \pi_{\theta}(a|s)]$$

- Now how do we estimate this expectation for a given policy?
- We can use a Monte Carlo (MC) estimate based on episode rollouts. This is called the **REINFORCE algorithm**:

1. Initialize  $\theta$  at random
2. Generate a trajectory on-policy  $\pi_{\theta}$
3. For each timestep in that trajectory ( $t = 1, \dots, T$ ):
  1. Get the episode return  $R(\tau)$  or use “reward to go”:  $\sum_{k=t}^T \gamma^{k-t} R(k)$
  2. Update the policy parameters:  $\theta \leftarrow \theta + \alpha R(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t|s_t)$

# Policy Gradient – In Practice

- We can **model our policy with a neural network** which takes in the current state as the input and outputs a distribution over actions.
- If we are modelling a continuous action space, we often assume the distribution over actions is a Gaussian and reparametrize accordingly

$$\pi_{\theta}(a|s) = \mathcal{N}(a|\mu_{\theta}, \sigma_{\theta}) \text{ where } f_{\theta}(s) \rightarrow [\mu_{\theta}, \sigma_{\theta}]$$

- Instead of maximizing our objective, we minimize a surrogate loss which gives us the desired gradient:

$$L(s, a) = -R(\tau) \cdot \log \pi_{\theta}(a|s)$$

$$\theta_{n+1} = \theta_n - \alpha \nabla L(s, a)$$

$$\theta \leftarrow \theta + \alpha R(\tau) \nabla_{\theta} \log \pi_{\theta}(a|s)$$

# Policy Gradient – In Practice

- We then follow our **REINFORCE** algorithm:

1. Collect episode(s) by sampling from our current policy
2. Iterate over each timestep and compute the loss by using the (discounted) reward and log probability of the action in the rollout
3. Update the policy neural network based on the mean of these computed losses

# Policy Gradient – In Practice

- By sampling from our policy during training we **naturally balance exploration and exploitation.**
- As training progresses, the policy favors high-value actions and we exploit more and explore less.
- During inference we usually **take the maximum of our action-distribution.**
- Note that **each training update impacts all the actions for a given state**, unlike Q-learning where each update impacts only a single state-action pair

# Policy Gradient Pros and Cons

## Pros

- Handles stochastic policies, which leads to intrinsic exploration VS exploitation
- Handles continuous action spaces
- Optimizes for our goal directly, instead of surrogate Q-values

## Cons

- REINFORCE estimates of  $\nabla_{\theta} E_{\tau \sim \pi_{\theta}}[R(\tau)]$  can have high variance, leading to instability
- Can only use on-policy samples, thus has worse sample efficiency than DQN
- Prone to local optima

# Policy Gradient Exercise

Access from Quick Links @ [rl-starterpack.github.io/#quick-links](https://rl-starterpack.github.io/#quick-links)

Direct Link: <https://github.com/RL-Starterpack/rl-starterpack/blob/main/exercises/PG.ipynb>

rl-starterpack / exercises / PG.ipynb

miguelarocao Update PG exercise

Latest commit 003b8fa 6 hours ago History

2 contributors

663 lines (663 sloc) | 23.3 KB

Open in Colab

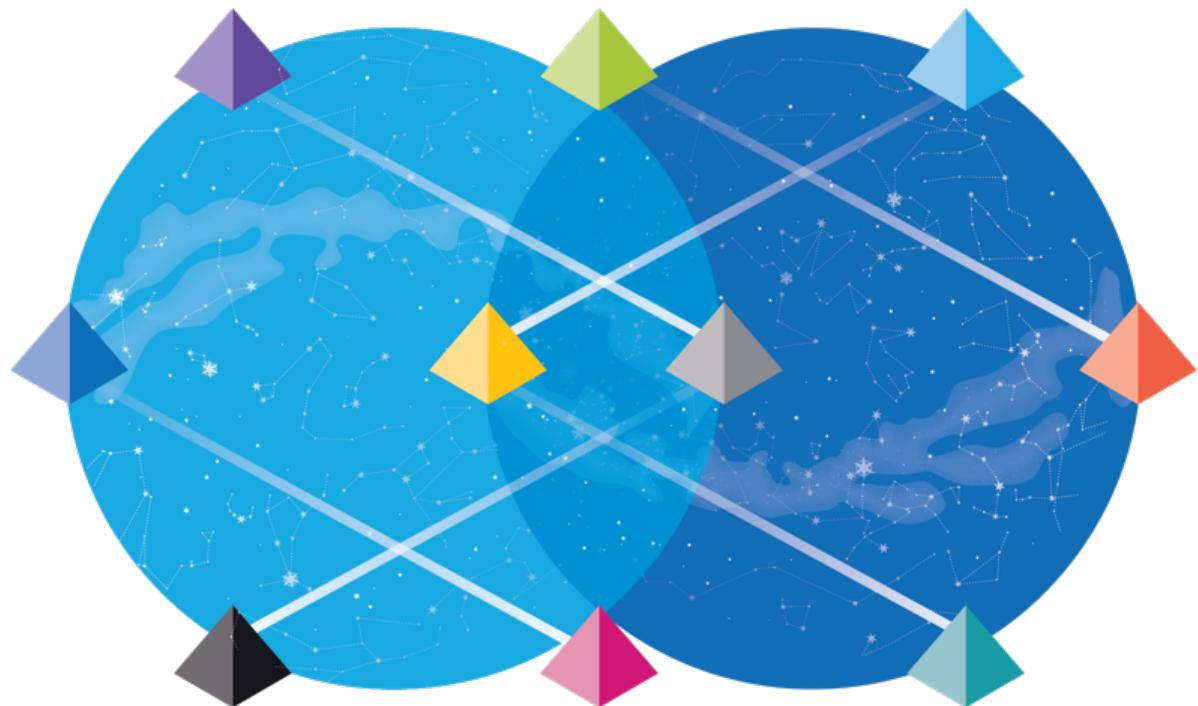
RL Tutorial - Policy Gradient Exercise

Setup

```
In [ ]: #@title Run this cell to clone the RL tutorial repository and install it
try:
    import rl_starterpack
    print('RL-Starterpack repo successfully installed!')
except ImportError:
    print('Cloning RL-Starterpack package...')

!git clone https://github.com/RL-Starterpack/rl-starterpack.git
print('Installing RL-StarterPack package...')
!pip install -e rl-starterpack &> /dev/null
print('\n-----')
print('Please restart the runtime to use the newly installed package!')
print('Runtime > Restart Runtime')
print('-----')
```

Click here to get started in Colab!



# ACTOR-CRITICS

Alexander Kuhnle

# Problems with REINFORCE

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T R(\tau) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

# Problems with REINFORCE

$$\nabla_{\theta} E_{\tau \sim \pi_{\theta}}[R(\tau)] = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T R(\tau) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- Data-inefficient: for each update, we need to collect at least one, ideally multiple episode rollouts

# Problems with REINFORCE

$$\nabla_{\theta} E_{\tau \sim \pi_{\theta}}[R(\tau)] = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T R(\tau) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- Data-inefficient: for each update, we need to collect at least one, ideally multiple episode rollouts
- Imprecise reward assignment: all actions within an episode get the same return, even if later actions cannot influence earlier ones

# Actor-critic I

- After some opaque maths:

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T R(\tau[t:]) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

# Actor-critic I

- After some opaque maths:

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T R(\tau[t:]) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- “Reward to go”:

$$\mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau[t:])] = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau) \mid s_0 = s_t, a_0 = a_t] = Q^{\pi_{\theta}}(s_t, a_t)$$

# Actor-critic I

- After some opaque maths:

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T R(\tau[t:]) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- “Reward to go”:

$$\mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau[t:])] = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau) \mid s_0 = s_t, a_0 = a_t] = Q^{\pi_{\theta}}(s_t, a_t)$$

- Actor-critic I:

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{s, a \sim \pi_{\theta}} [Q^{\pi_{\theta}}(s, a) \cdot \nabla_{\theta} \log \pi_{\theta}(a | s)]$$

# Actor-critic I

- Actor-critic I:

$$\nabla_{\theta} E_{\tau \sim \pi_{\theta}}[R(\tau)] = E_{s,a \sim \pi_{\theta}}[Q^{\pi_{\theta}}(s,a) \cdot \nabla_{\theta} \log \pi_{\theta}(a|s)]$$

- We train two models simultaneously:
  - “Actor” policy  $\pi_{\theta}(a|s)$  using above policy gradient formulation
  - “Critic” Q-value function  $Q(s,a)$  to approximate  $Q^{\pi_{\theta}}(s,a)$  via TD learning (Q-learning)

# Actor-critic I

- Actor-critic I:

$$\nabla_{\theta} E_{\tau \sim \pi_{\theta}}[R(\tau)] = E_{s,a \sim \pi_{\theta}}[Q^{\pi_{\theta}}(s,a) \cdot \nabla_{\theta} \log \pi_{\theta}(a|s)]$$

- We train two models simultaneously:
  - “Actor” policy  $\pi_{\theta}(a|s)$  using above policy gradient formulation
  - “Critic” Q-value function  $Q(s,a)$  to approximate  $Q^{\pi_{\theta}}(s,a)$  via TD learning (Q-learning)
- Advantage:
  - Updates are based on timesteps, not episode rollouts
  - More data-efficient, more frequent updates

# Actor-critic II

- It can be shown that for any state-dependent “*baseline*” function  $b(s)$ :

$$\mathbb{E}_{s,a \sim \pi_\theta} [b(s) \cdot \nabla_\theta \log \pi_\theta(a|s)] = 0$$

# Actor-critic II

- It can be shown that for any state-dependent “*baseline*” function  $b(s)$ :

$$\mathbb{E}_{s,a \sim \pi_\theta} [b(s) \cdot \nabla_\theta \log \pi_\theta(a|s)] = 0$$

- Hence the following is mean-preserving but, for an appropriate baseline, variance-reducing:

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} [\mathcal{R}(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T (\mathcal{R}(\tau[t: ]) - b(s)) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

# Actor-critic II

- It can be shown that for any state-dependent “*baseline*” function  $b(s)$ :

$$\mathbb{E}_{s,a \sim \pi_\theta} [b(s) \cdot \nabla_\theta \log \pi_\theta(a|s)] = 0$$

- Hence the following is mean-preserving but, for an appropriate baseline, variance-reducing:

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} [\mathcal{R}(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T (\mathcal{R}(\tau[t: ]) - b(s)) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

- It can further be shown that the choice  $b(s) = V^{\pi_\theta}(s)$  is optimal w.r.t. variance reduction, Actor-critic II:

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} [\mathcal{R}(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T (\mathcal{R}(\tau[t: ]) - V^{\pi_\theta}(s)) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

# Actor-critic II

- Actor-critic II:

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T (R(\tau[t: ]) - V^{\pi_{\theta}}(s)) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- We again train two models simultaneously:
  - “Actor” policy  $\pi_{\theta}(a|s)$  using above policy gradient formulation
  - “Critic” state-value function  $V(s)$  to approximate  $V^{\pi_{\theta}}(s)$  via TD learning

# Actor-critic II

- Actor-critic II:

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T (R(\tau[t: ]) - V^{\pi_{\theta}}(s)) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- We again train two models simultaneously:
  - “Actor” policy  $\pi_{\theta}(a|s)$  using above policy gradient formulation
  - “Critic” state-value function  $V(s)$  to approximate  $V^{\pi_{\theta}}(s)$  via TD learning
- Advantage:
  - Gradient estimator based on this formulation is lower-variance
  - Relative instead of absolute perspective: how much better than expected is this rollout?

# Why does the baseline reduce variance?

- Distribution:

$$\pi_{\theta}(1|s) = 0.2, \quad \pi_{\theta}(2|s) = 0.5, \quad \pi_{\theta}(3|s) = 0.3$$

- Q-values:

$$Q^{\pi_{\theta}}(s, 1) = 6, \quad Q^{\pi_{\theta}}(s, 2) = 7, \quad Q^{\pi_{\theta}}(s, 3) = 8$$

# Why does the baseline reduce variance?

- Distribution:

$$\pi_\theta(1|s) = 0.2, \quad \pi_\theta(2|s) = 0.5, \quad \pi_\theta(3|s) = 0.3$$

- Q-values:

$$Q^{\pi_\theta}(s, 1) = 6, \quad Q^{\pi_\theta}(s, 2) = 7, \quad Q^{\pi_\theta}(s, 3) = 8$$

- Estimator without baseline:  $E_{s,a \sim \pi_\theta}[Q^{\pi_\theta}(s, a) \cdot \nabla_\theta \log \pi_\theta(a|s)]$

$$\hat{g}_{\text{REINFORCE}} = 6 \cdot \nabla_\theta \pi_\theta(1|s) + 7 \cdot \nabla_\theta \pi_\theta(2|s) + 8 \cdot \nabla_\theta \pi_\theta(3|s)$$

# Why does the baseline reduce variance?

- Distribution:

$$\pi_\theta(1|s) = 0.2, \quad \pi_\theta(2|s) = 0.5, \quad \pi_\theta(3|s) = 0.3$$

- Q-values:

$$Q^{\pi_\theta}(s, 1) = 6, \quad Q^{\pi_\theta}(s, 2) = 7, \quad Q^{\pi_\theta}(s, 3) = 8$$

- Estimator without baseline:  $E_{s,a \sim \pi_\theta}[Q^{\pi_\theta}(s, a) \cdot \nabla_\theta \log \pi_\theta(a|s)]$

$$\hat{g}_{\text{REINFORCE}} = 6 \cdot \nabla_\theta \pi_\theta(1|s) + 7 \cdot \nabla_\theta \pi_\theta(2|s) + 8 \cdot \nabla_\theta \pi_\theta(3|s)$$

- Estimator with baseline:  $E_{s,a \sim \pi_\theta}[(Q^{\pi_\theta}(s, a) - b(s)) \cdot \nabla_\theta \log \pi_\theta(a|s)]$

$$b(s) = 7 \approx V^{\pi_\theta}(s) \implies \hat{g}_{\text{Baseline}} = -\nabla_\theta \pi_\theta(1|s) + \nabla_\theta \pi_\theta(3|s)$$

# Actor-critic policy gradient versions

- Actor-critic I: “Q *actor-critic*” (AC)

$$\mathbb{E}_{s,a \sim \pi_\theta} [Q^{\pi_\theta}(s, a) \cdot \nabla_\theta \log \pi_\theta(a|s)]$$

# Actor-critic policy gradient versions

- Actor-critic I: “Q *actor-critic*” (AC)

$$\mathbb{E}_{s,a \sim \pi_\theta} [Q^{\pi_\theta}(s, a) \cdot \nabla_\theta \log \pi_\theta(a|s)]$$

- Actor-critic II: "Advantage *actor-critic*" (A2C)

$$\mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T \underbrace{(R(\tau[t:])-V^{\pi_\theta}(s))}_{\text{Advantage } A^{\pi_\theta}(s, a)} \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

# Actor-critic policy gradient versions

- Actor-critic I: “Q actor-critic” (AC)

$$\mathbb{E}_{s,a \sim \pi_\theta} [Q^{\pi_\theta}(s, a) \cdot \nabla_\theta \log \pi_\theta(a|s)]$$

- Actor-critic II: "Advantage actor-critic" (A2C)

$$\mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T \underbrace{(R(\tau[t:])-V^{\pi_\theta}(s))}_{\text{Advantage } A^{\pi_\theta}(s, a)} \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

- Actor-critic I + II: "TD actor-critic" (A2C)

$$\mathbb{E}_{s,a,s' \sim \pi_\theta} \left[ \underbrace{\left( (r + \gamma \cdot V^{\pi_\theta}(s')) - V^{\pi_\theta}(s) \right)}_{\text{TD Advantage } A^{\pi_\theta}(s, a)} \cdot \nabla_\theta \log \pi_\theta(a|s) \right]$$

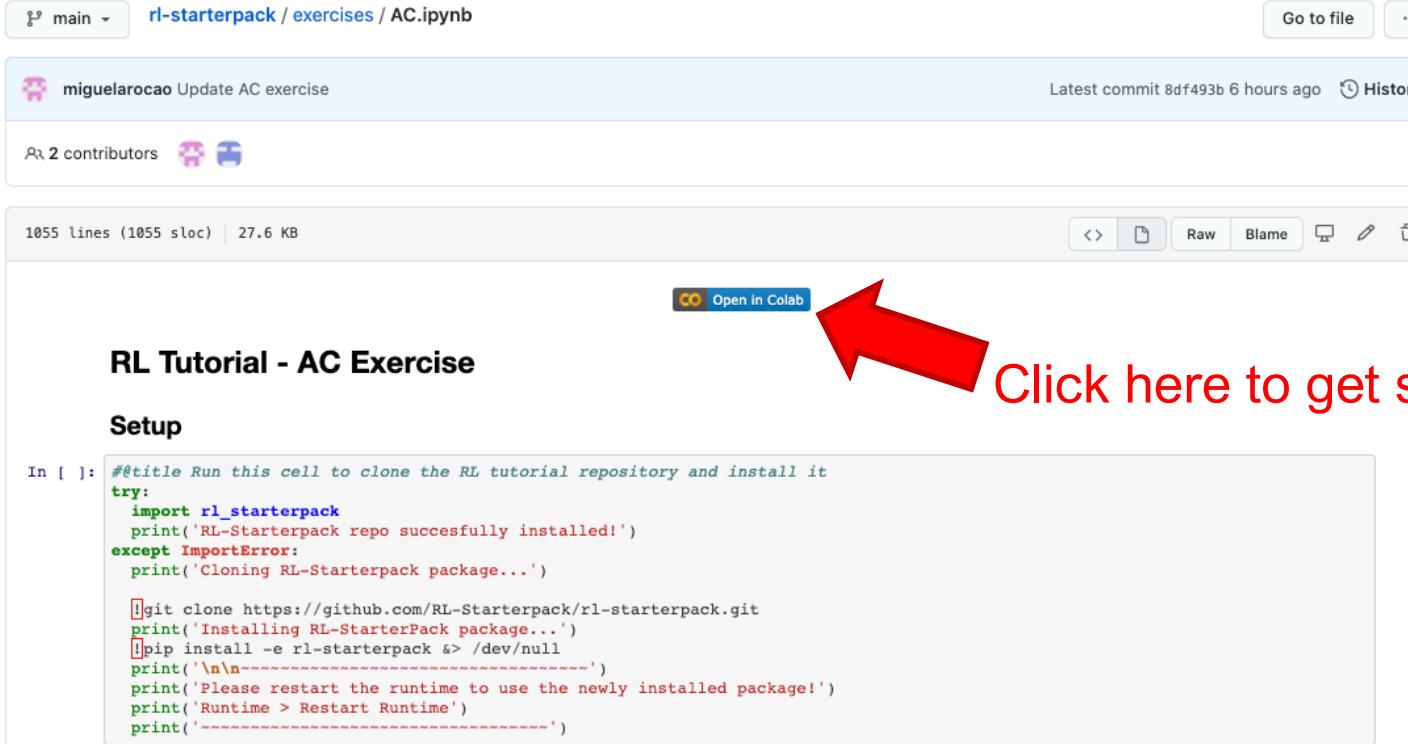
# Actor-critic in practice

- The reward landscape is indirectly propagated to the actor via the critic's value estimates
- The training data for both actor and critic is determined by the actor's decisions
- Both actor and critic are randomly initialized at the beginning of training
- The critic needs to evolve in tandem with the actor policy
  - A very certain actor means no useful experience data for critic to improve value function
  - In the case of A2C, a very accurate critic means close-to-zero advantage reward for actor
    - Desired behavior if task is solved
    - Undesired behavior if actor is stuck prematurely

# Actor-critic Exercise

Access from Quick Links @ [rl-starterpack.github.io/#quick-links](https://rl-starterpack.github.io/#quick-links)

Direct Link: <https://github.com/RL-Starterpack/rl-starterpack/blob/main/exercises/AC.ipynb>



A screenshot of a GitHub repository page for 'AC.ipynb'. The page shows basic repository information: 'miguelarocao' updated it, there are 2 contributors, and the file has 1055 lines (1055 sloc) and is 27.6 KB. At the bottom, there's a code editor with a cell containing Python code for cloning the repository and installing it. A prominent red arrow points to the 'Open in Colab' button, which is located just above the code editor area.

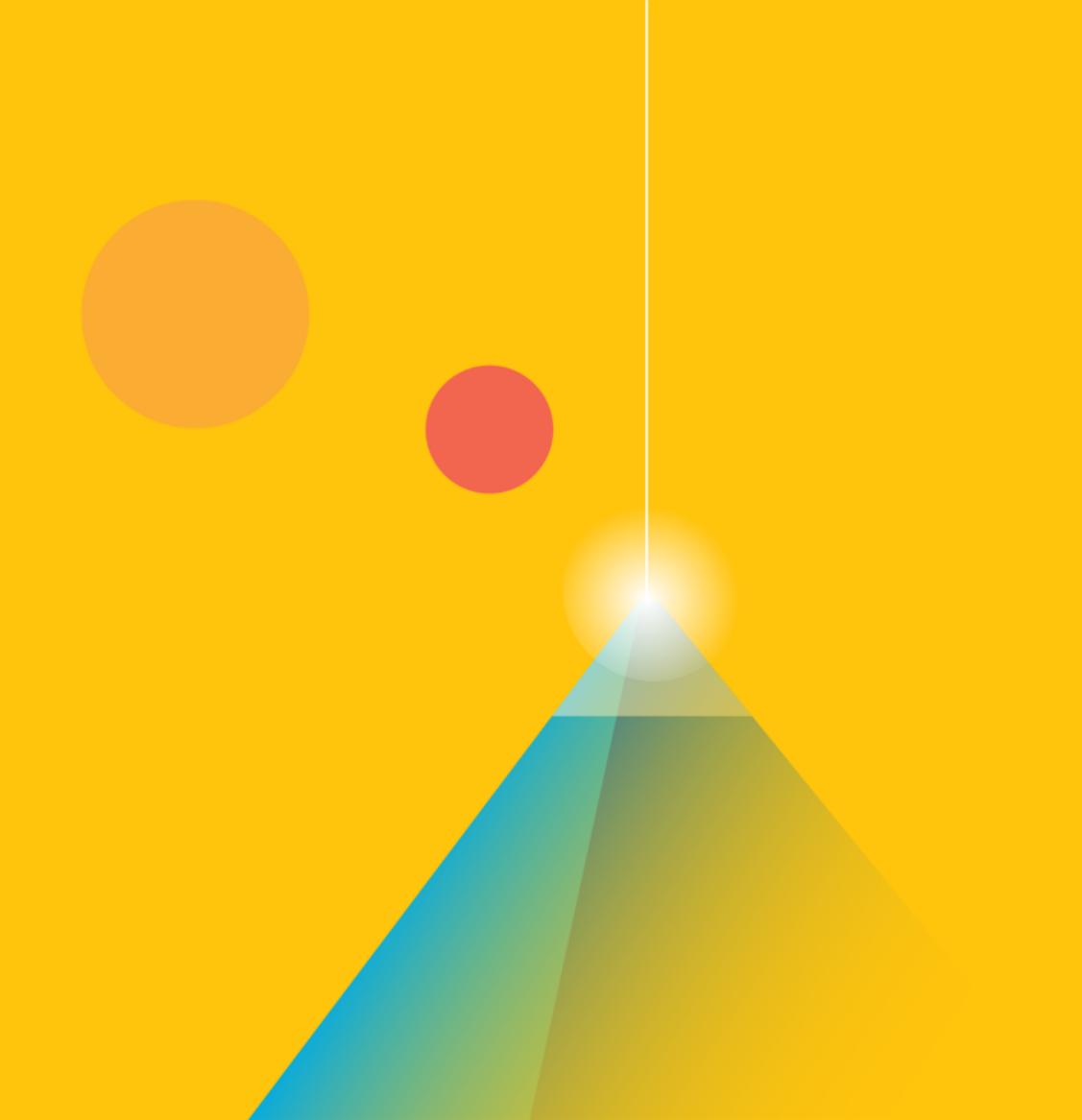
RL Tutorial - AC Exercise

Setup

```
In [ ]: #@title Run this cell to clone the RL tutorial repository and install it
try:
    import rl_starterpack
    print('RL-Starterpack repo successfully installed!')
except ImportError:
    print('Cloning RL-Starterpack package...')

!git clone https://github.com/RL-Starterpack/rl-starterpack.git
print('Installing RL-StarterPack package...')
!pip install -e rl-starterpack &> /dev/null
print('\n\n')
print('Please restart the runtime to use the newly installed package!')
print('Runtime > Restart Runtime')
print('-----')
```

Click here to get started in Colab!



# OUTLOOK

Alexander Kuhnle

# Outlook

- **Data collection:** parallelized environment interaction, distributed model training
- **Learning from demonstrations:** imitation learning, inverse reinforcement learning
- **Internal state:** re-active vs evolving decision making, planning capacity
- **Predicting the future:** auxiliary loss, curiosity (particularly with sparse rewards)
- **Goal-parametrized policies:** multiple objectives, varying prioritization
- **Model-based influences:** infuse environment knowledge, learn world model

# We Are Hiring

- Senior AI Lead
- Research Engineer
- Software Engineer

## Ready to Change the World?

Join Blue Prism and help us redefine the future of work.

**Open positions**

<https://www.blueprism.com/who-we-are/culture-and-careers/>

Apply online or **reach out to one of the organisers directly to apply!**

{alexander.kuhnle, miguel.aroca, john.reid, dell.zhang}@blueprism.com

# Thanks & Acknowledgements



The Chartered Institute for IT  
Information Retrieval Specialist Group

Search Solutions 2020



Tutorial Chair  
**Dr Haiming Liu**

# Feedback

**Please help us improve this tutorial!**

We have a short 2min feedback form:

<https://forms.gle/xXdjcrETVd9kD8Nn7>

Your feedback is greatly appreciated!