blueprism
AI Labs

REINFORCEMENT LEARNING FOR INFORMATION RETRIEVAL

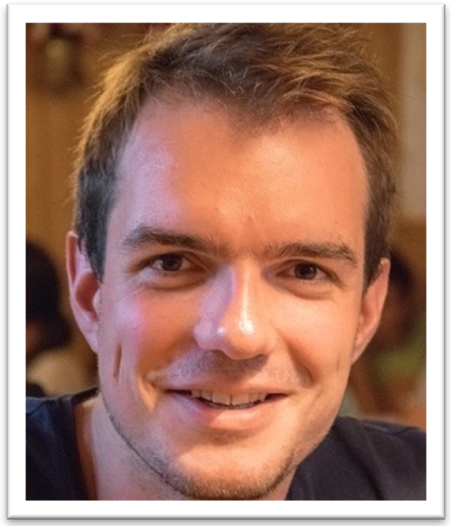
Alexander Kuhnle

Miguel Aroca-Ouellette

John Reid

Dell Zhang

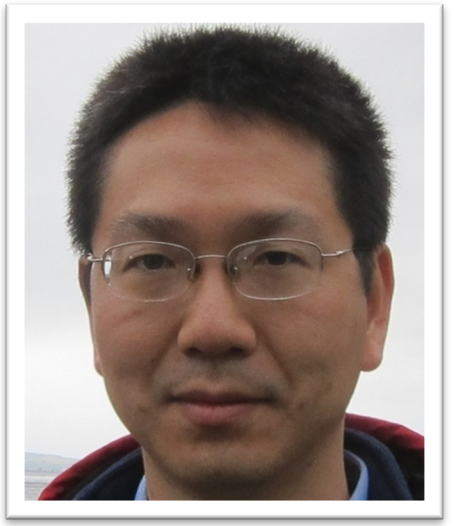
Your Presenters



Alexander Kuhnle

Miguel Aroca-Ouellette

John Reid



Dell Zhang

Murat Sensoy

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

Tutorial Website: **rl-starterpack.github.io**

Schedule

- 08:00-09:00 **RL Basics and Tabular Q-Learning**
- 09:00-09:30 **Deep Q-Network (DQN) 1/2**

- 09:30-10:00 Coffee Break

- 10:00-10:30 **Deep Q-Network (DQN) 2/2**
- 10:30-11:15 **IR Applications using DQN**

- 11:15-13:15 Lunch Break (*45 minutes extra*)

- 13:15-14:15 **Policy Gradient (REINFORCE)**
- 14:15-15:15 **IR Applications using REINFORCE**

- 15:15-15:45 Coffee Break

- 15:45-16:15 **Actor Critic**
- 16:15-16:45 **Outlook**
- 16:45-17:15 **Final Q&A**

All times are in ***BST (UTC+1)***

Zoom Housekeeping

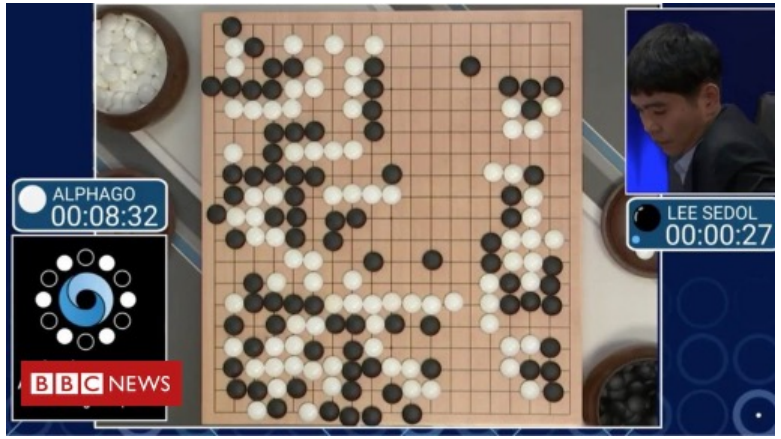
- **During presentation:**
  - Keep yourself muted
  - Do not share your screen
  - Ask questions in chat
- **During tutorial exercises:**
  - **If you have a question:** Feel free to unmute to ask, or ask in the chat. You can share your screen as well if that helps explain
  - Otherwise please keep yourself muted to keep background noise to a minimum

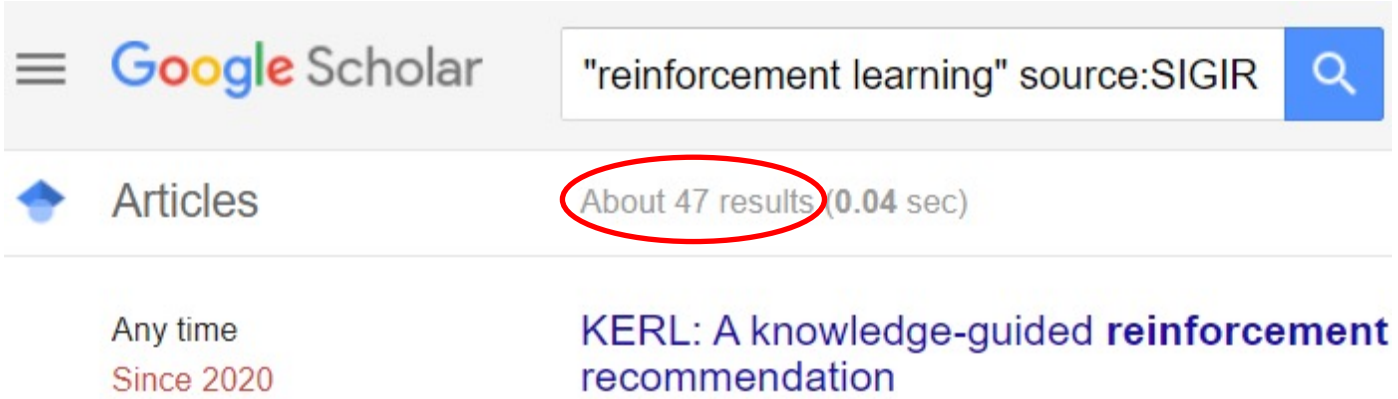
Thank you!

Why RL for IR?

Technology Push

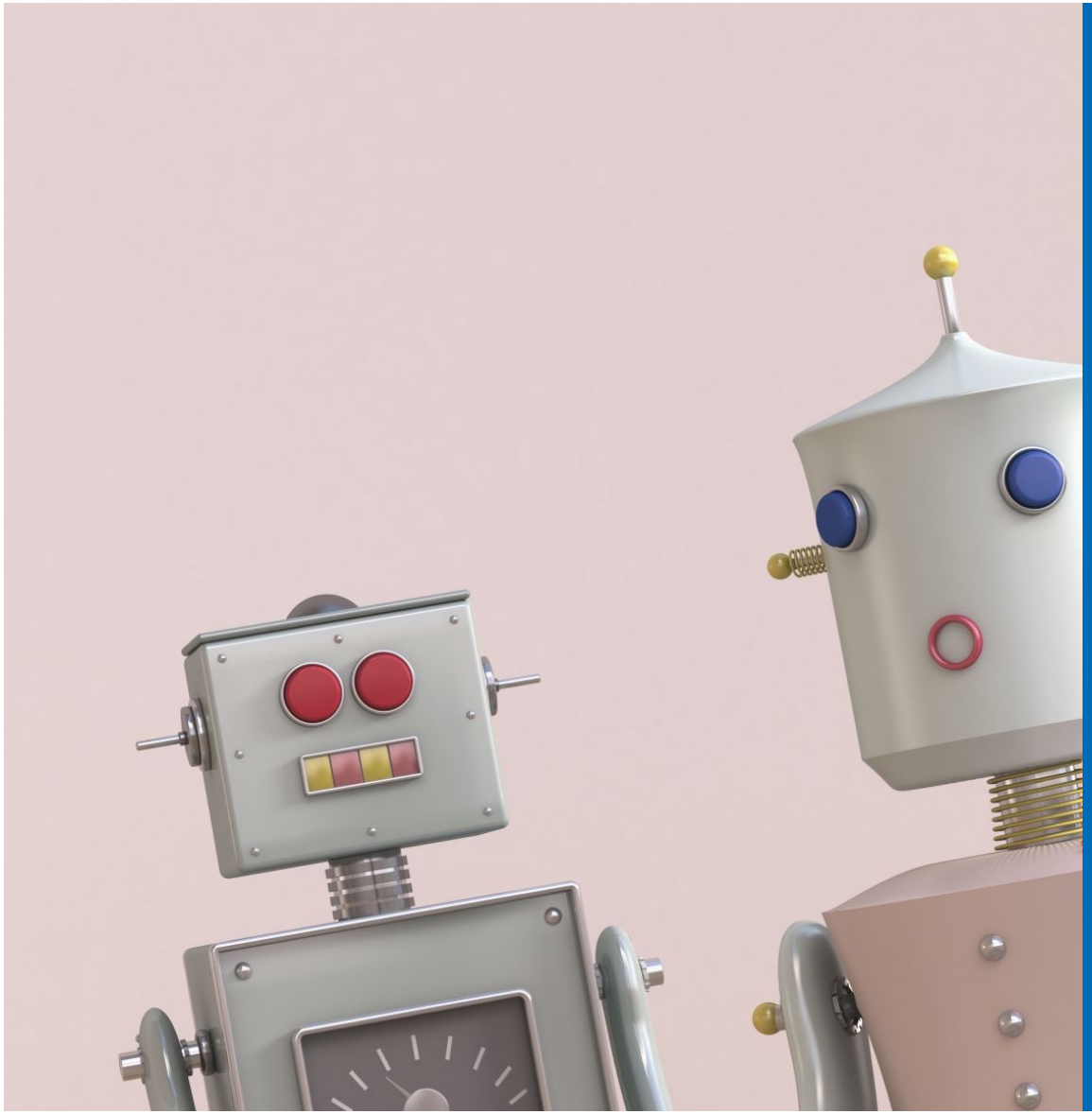
- Reinforcement Learning + Deep Learning: *Great Successes*
  - AlphaGo, etc.
- Sequential Decision Making: *Beyond Independence Relevance*
  - Document Ranking Positions
  - Document Interdependencies





Demand Pull

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



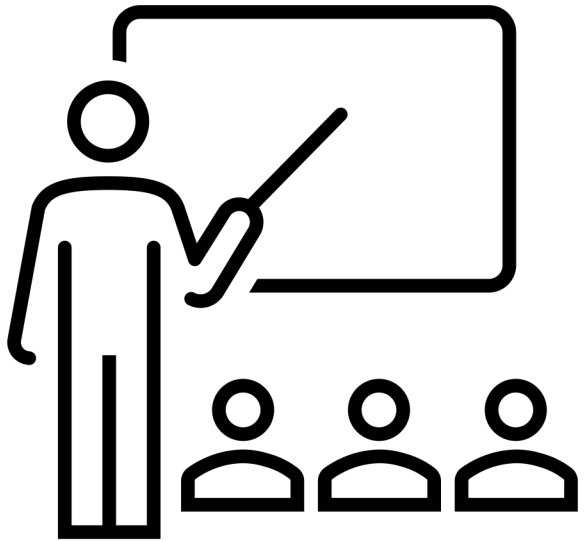
REINFORCEMENT LEARNING BASICS

John Reid

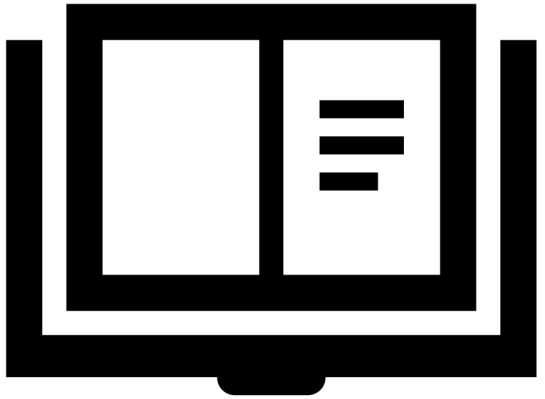
blueprism AI Labs

Machine learning

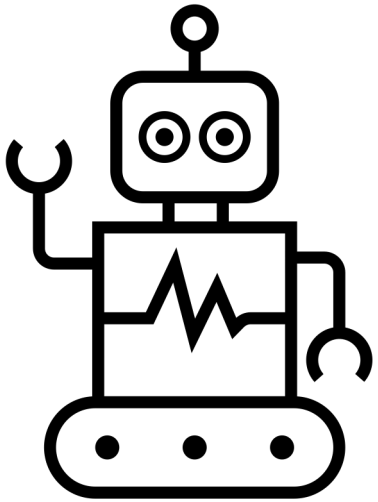
Supervised learning



Unsupervised learning



Reinforcement learning

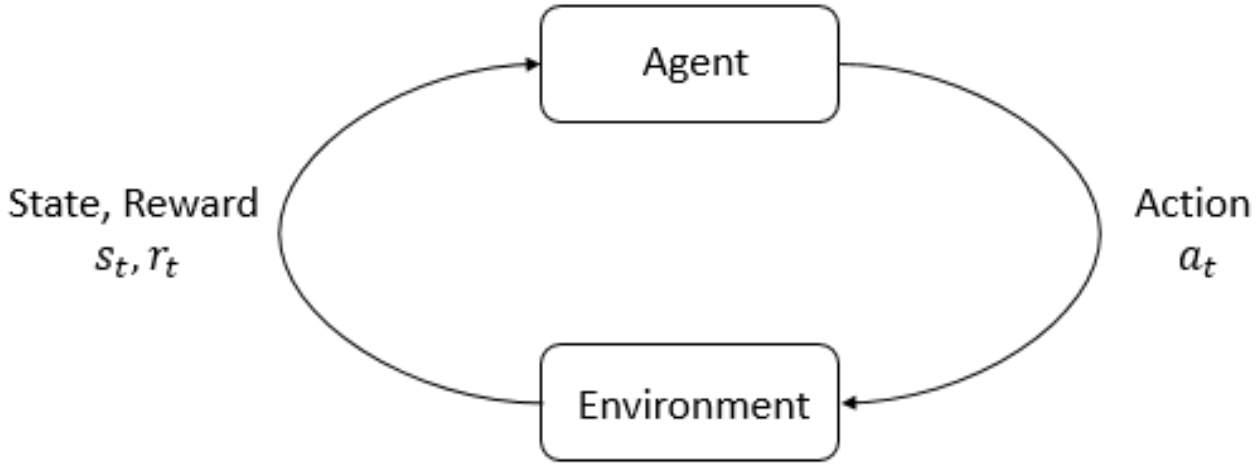


What is reinforcement learning?

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

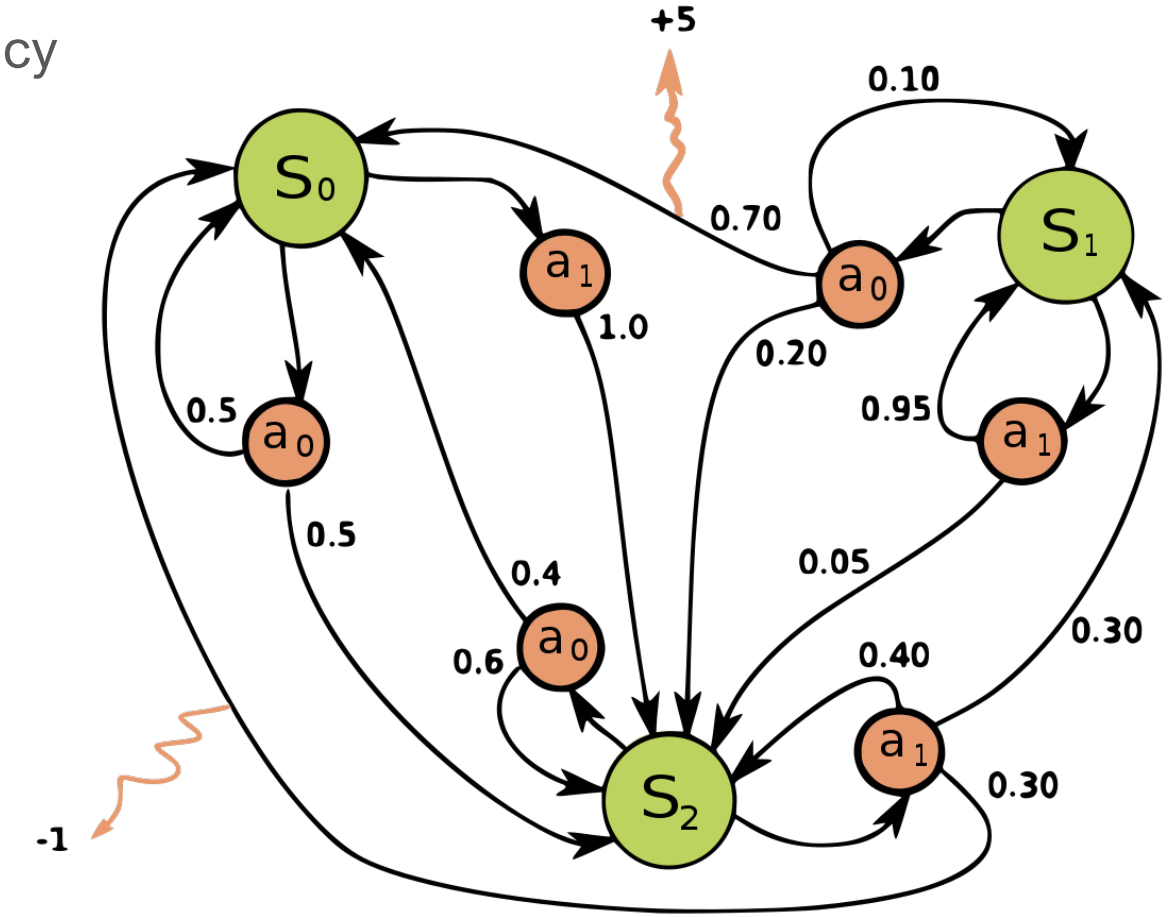
Markov decision processes

Framework of *discrete-time stochastic control processes*

Theory focusses on how to choose a good policy

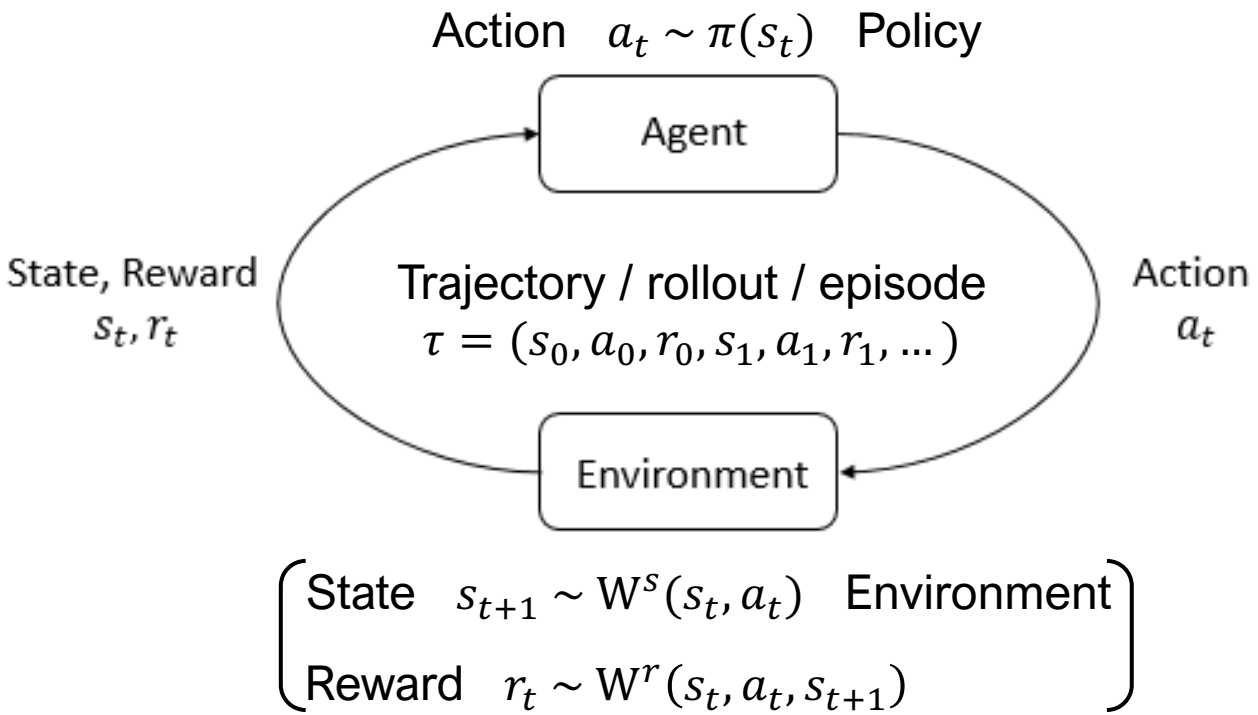
RL policy typically uses Markov assumption

Extension: *partial observability* (POMDP)



By waldoalvarez - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=59364518

Interactions with the environment



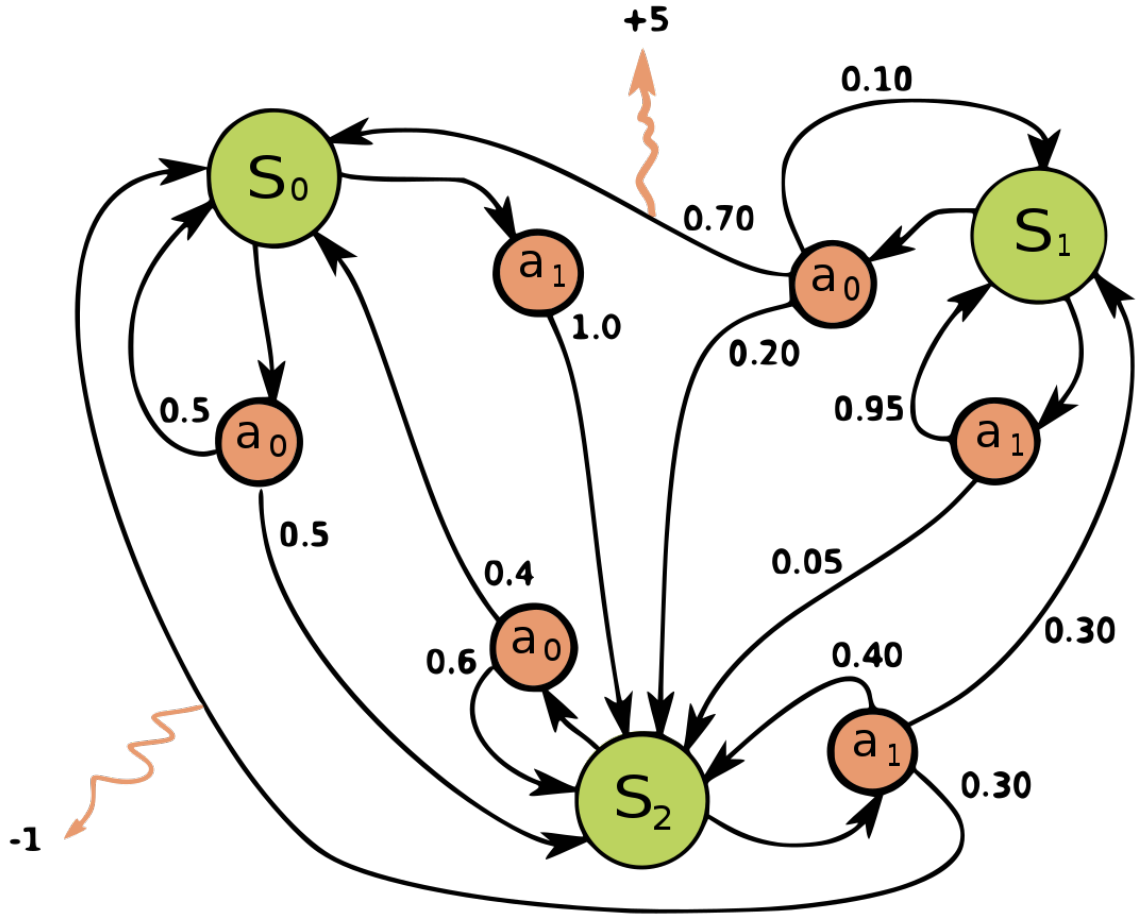
model-based vs model-free

Credit assignment problem

Which actions led to reward?

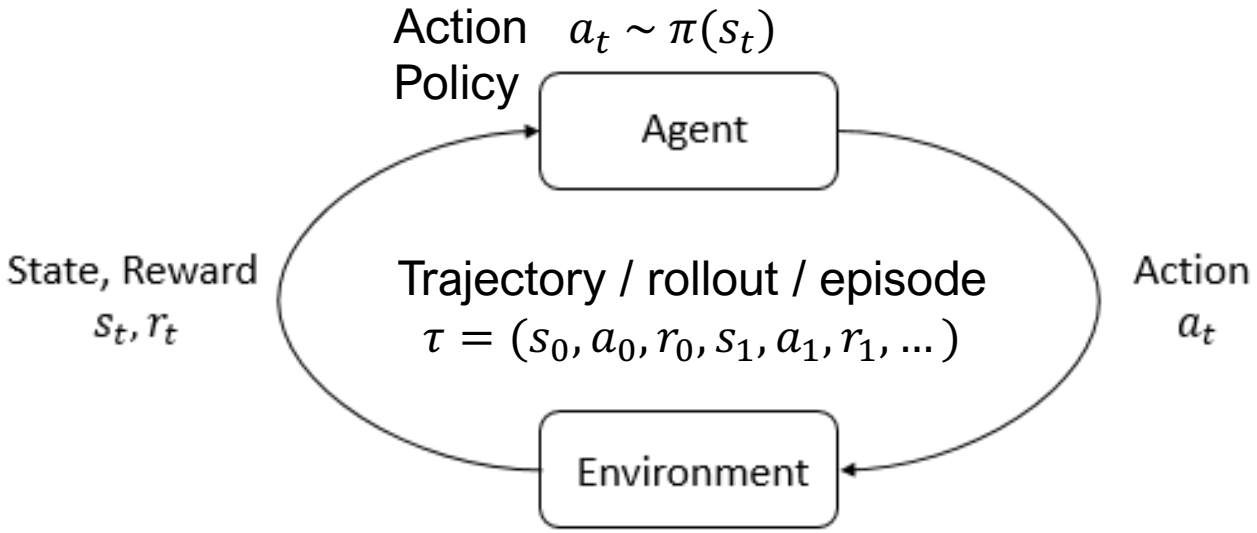
Not obvious for delayed or sparse rewards

Trial-and-error required



blueprism AI Labs

Rewards and returns



Environment: W^s, W^r
State: $s_{t+1} \sim \mathrm{W}^s(s_t, a_t)$
Reward: $r_{t+1} \sim \mathrm{W}^r(s_t, a_t, s_{t+1})$

Infinite-horizon discounted return:

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

Finite-horizon undiscounted return:

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

Value-based methods

Expectations of returns

State value:

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

State-action / Q-value:

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

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!)

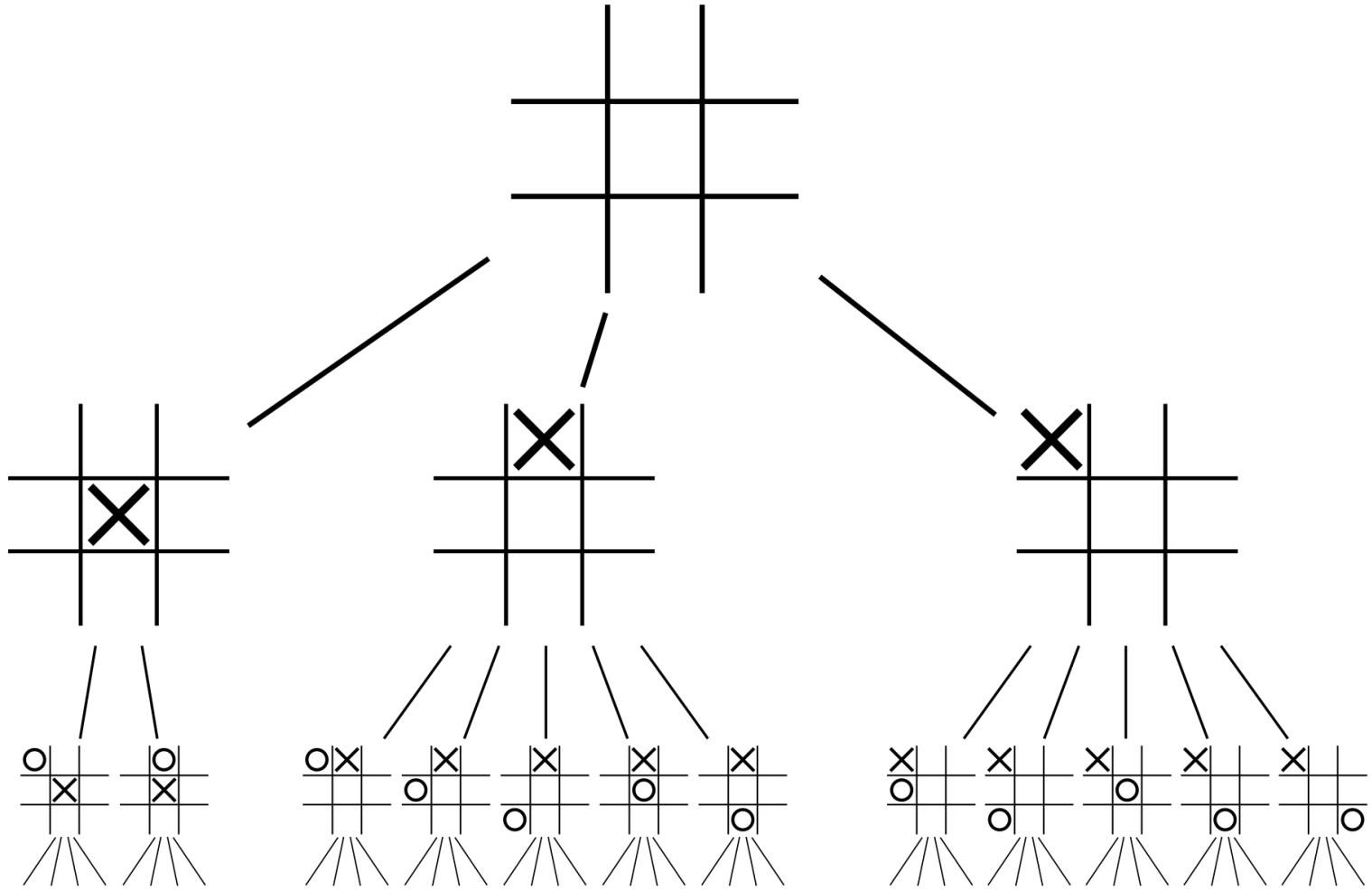
Start 0.0	Frozen 0.1	Frozen 0.3	Frozen 0.1
Frozen 0.1	Hole -1.0	??? ??? Frozen 0.4 ??? ???	Hole -1.0
Frozen 0.3	Frozen 0.5	Frozen 0.7	Hole -1.0
Hole -1.0	Frozen 0.7	Frozen 0.9	Goal 1.0

Model-based example

Game tree

Policy plans using tree

AlphaGo



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

Q-learning

Value-based algorithm:

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

Assumption: discrete action space

Temporal difference learning:

$$\mathrm{Q}(s_t, a_t) \coloneqq \mathrm{Q}(s_t, a_t) + \lambda \cdot \overbrace{\Big[\underbrace{r_t + \gamma \cdot \max_a \mathrm{Q}(s_{t+1}, a)}_{\text{TD target}} - \mathrm{Q}(s_t, a_t)\Big]}^{\text{TD residual}}$$

Bellman equation

When using infinite-horizon discounted returns

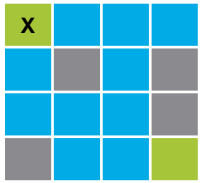
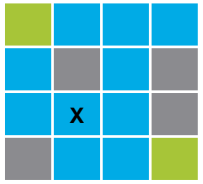
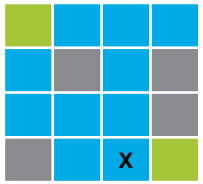
$$\mathrm{R}(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

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

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

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

Tabular Q-learning

		right	down	left	up
• 1:		0.1	0.1	0.0	0.0
•	…	…	…	…	…
• 9:		0.6	0.6	0.2	-0.9
•	…	…	…	…	…
• 15:		1.0	0.9	0.7	0.7

Exploration vs exploitation

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

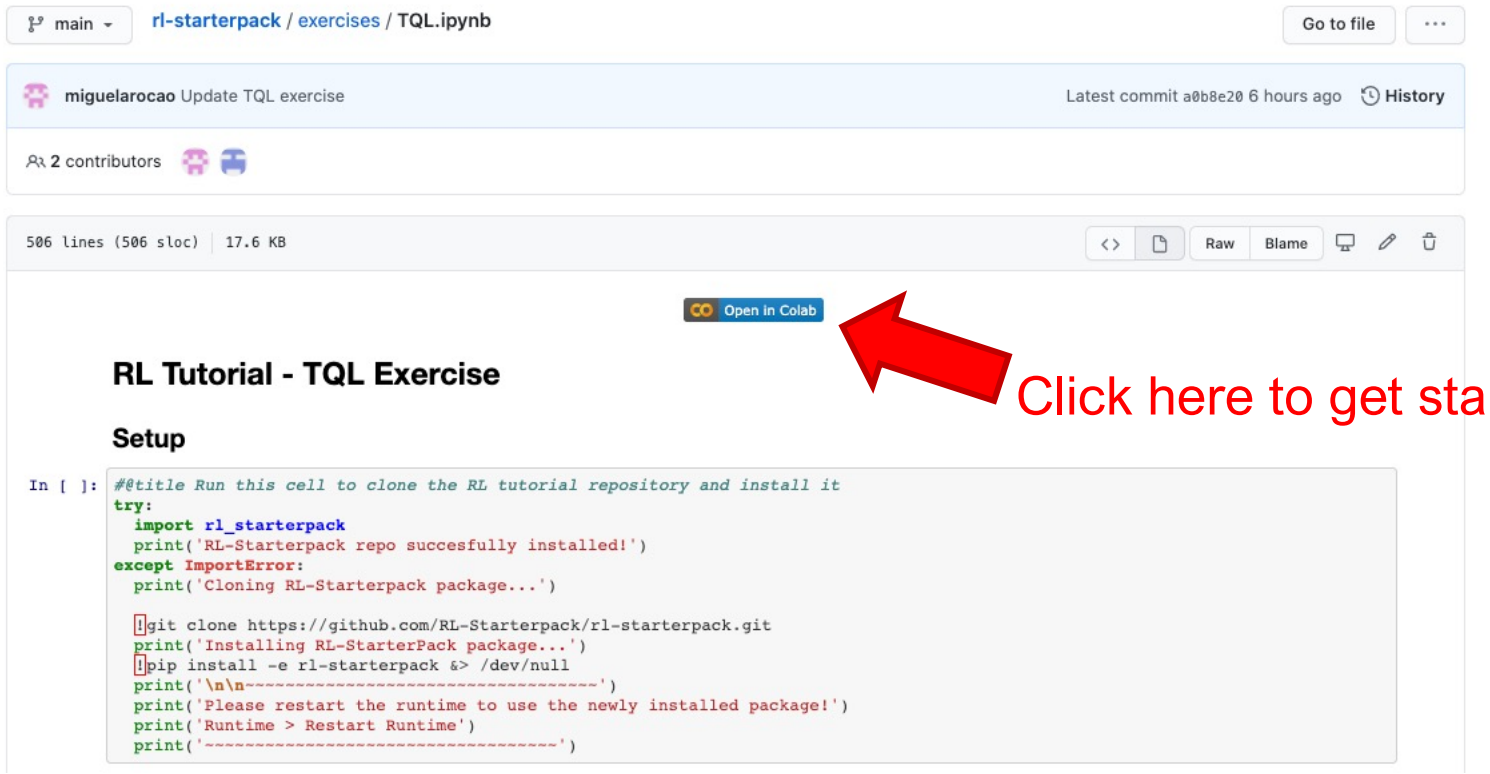
Reward shaping

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

TQL Exercise

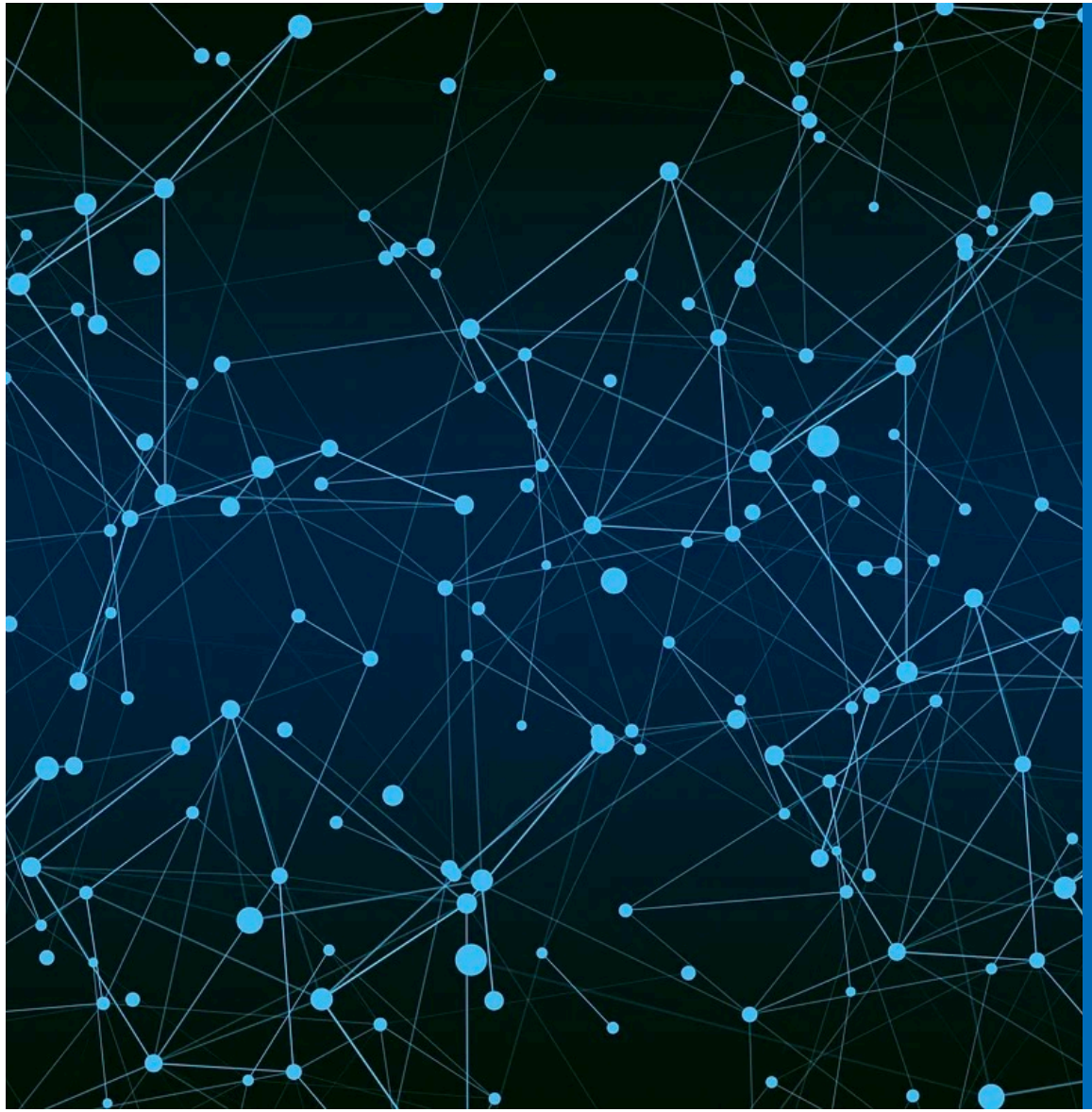
Access from **Quick Links @ rl-starterpack.github.io/#quick-links**

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



Click here to get started in Colab!

blueprism AI Labs

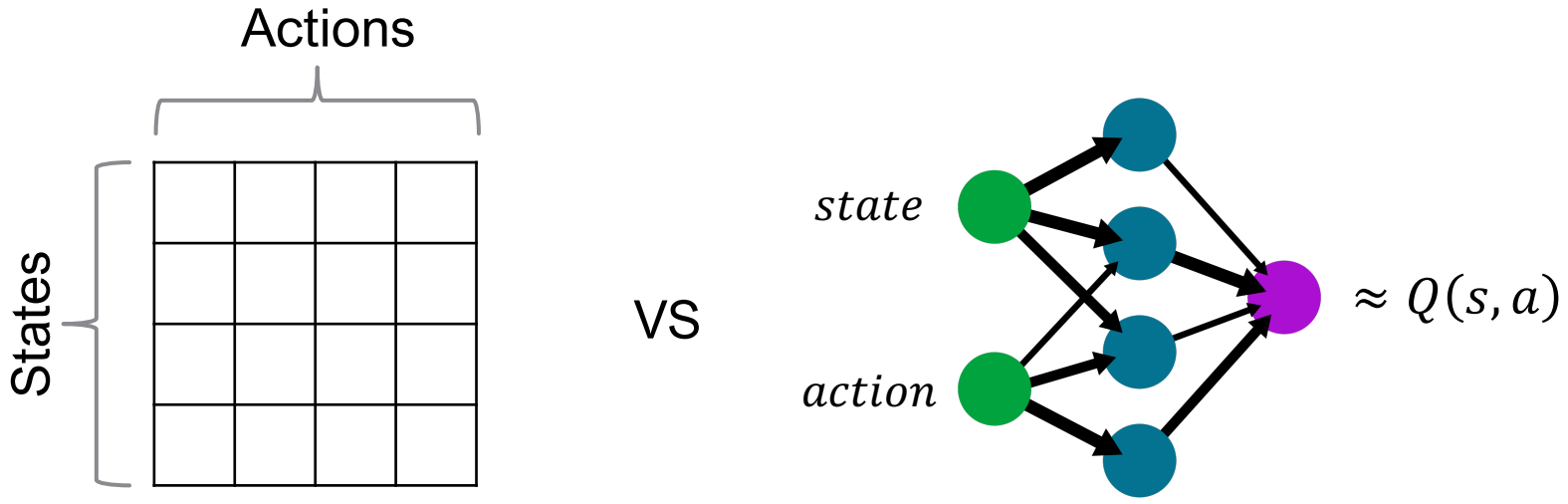


DEEP Q-LEARNING

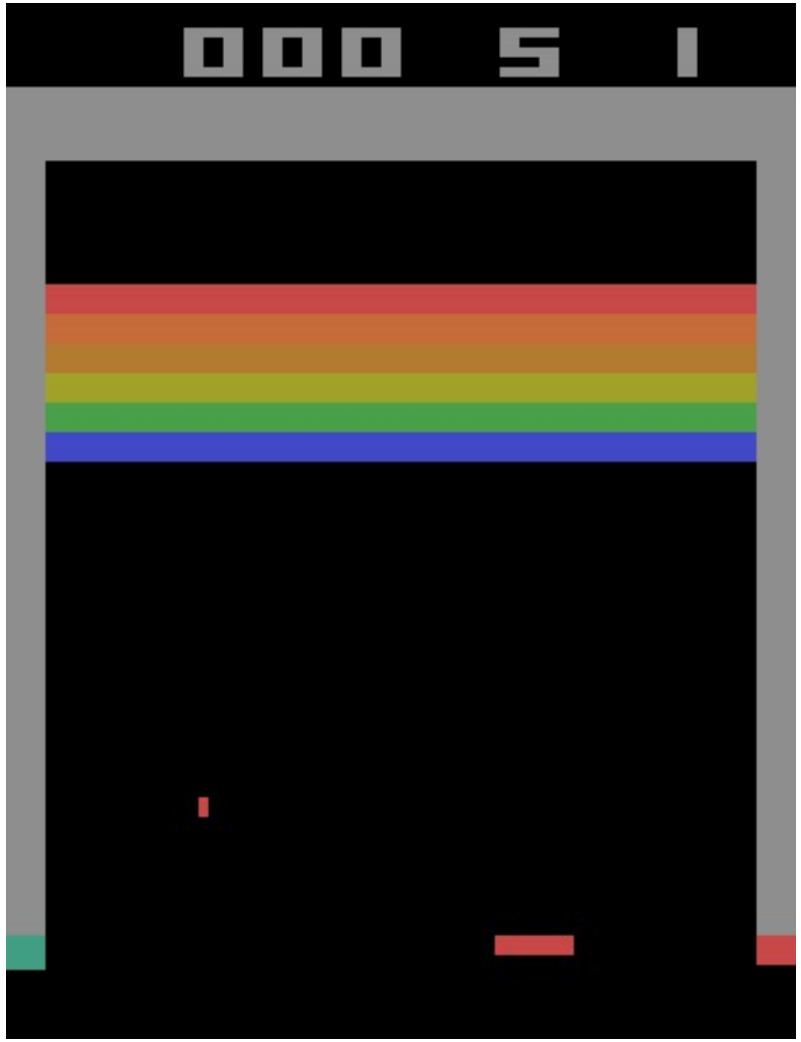
Alexander Kuhnle

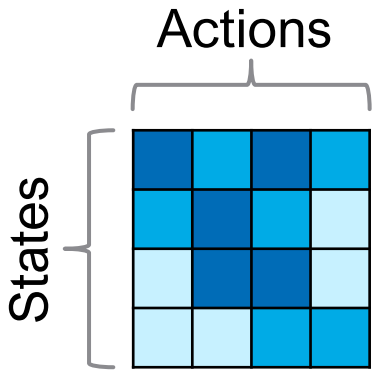
Deep Q-learning Networks Motivation

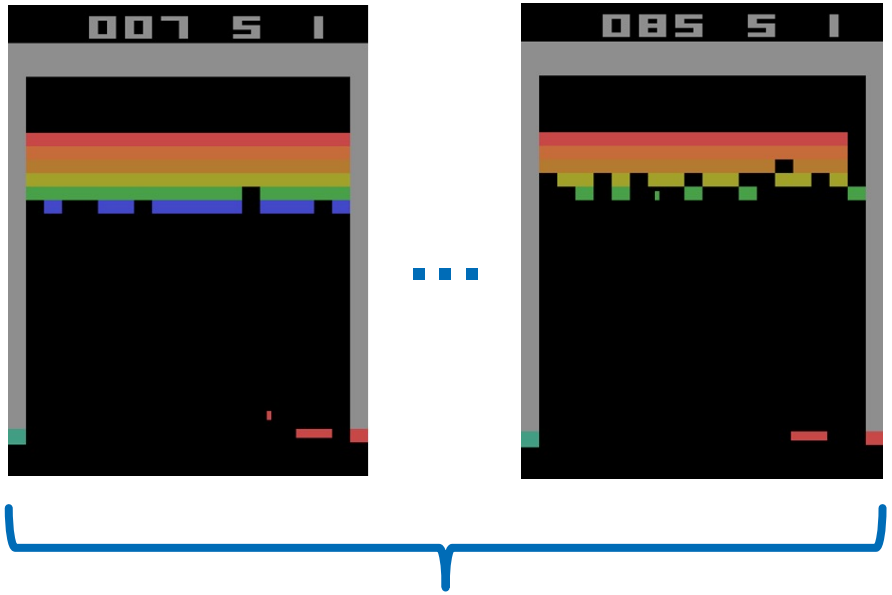
- In TQL we used a table to store the Q-values for our (state, action) pairs.
- Now imagine a scenario where we have 10,000 states and 10,000 actions per state. We would then need to maintain a **table with 100 million entries**.
- Can we instead **learn a function** which takes in the state and action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



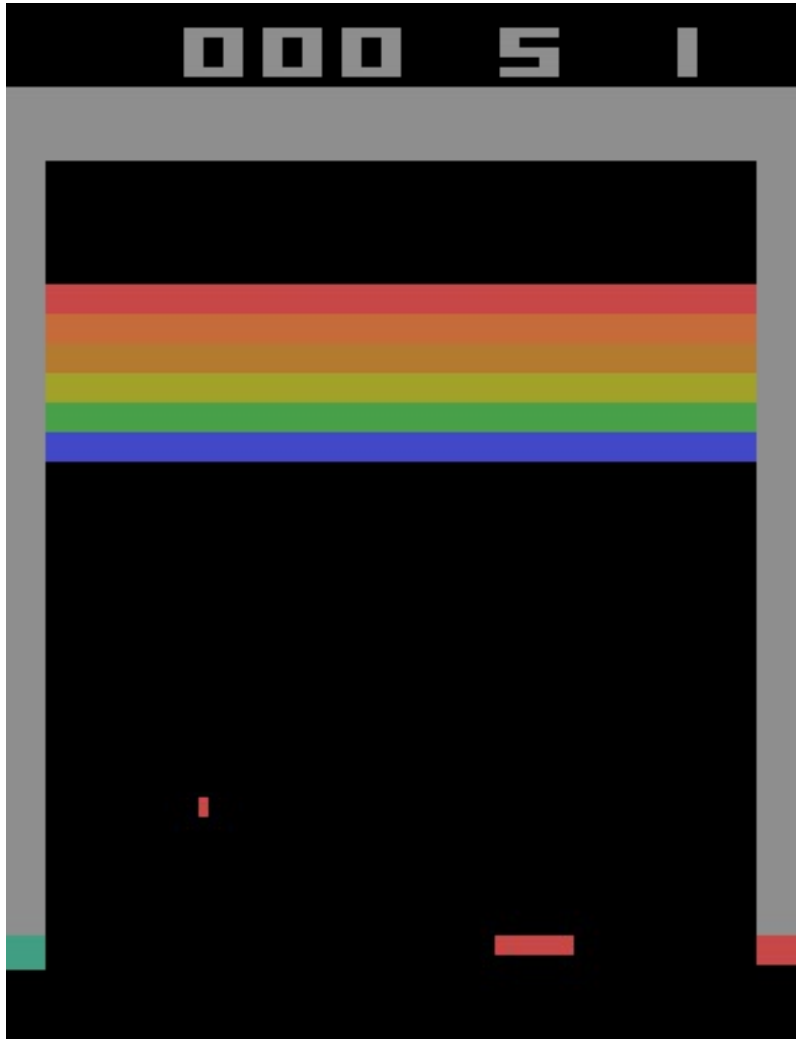
Deep Q-learning Networks Motivation

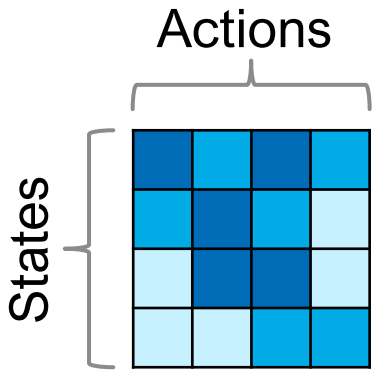


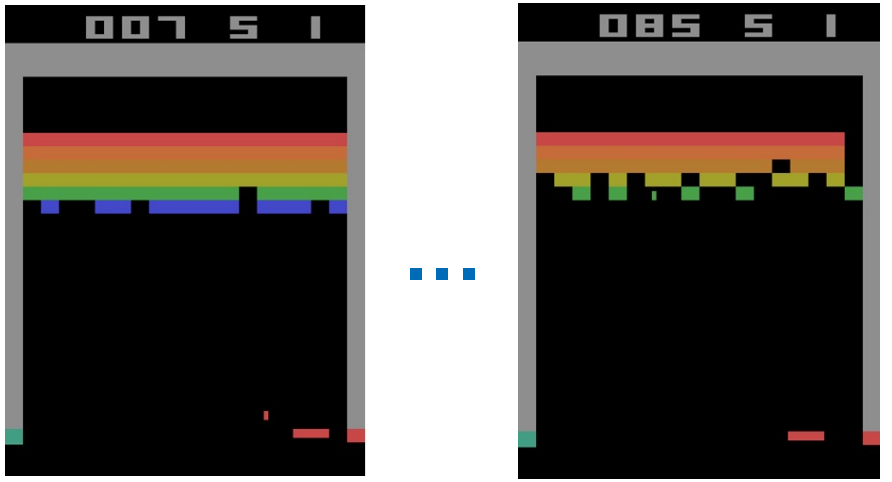




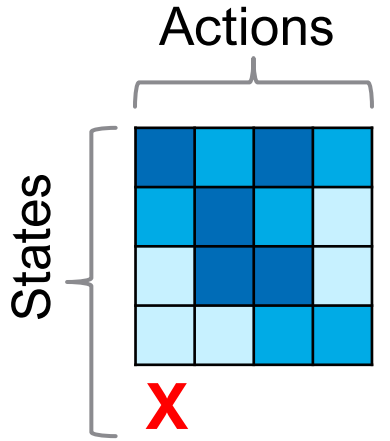
Deep Q-learning Networks Motivation

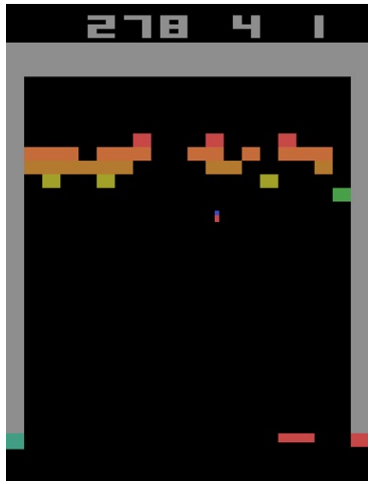






TRAINING





INFERENCE

blueprism AI Labs

Deep Q-learning Networks Motivation

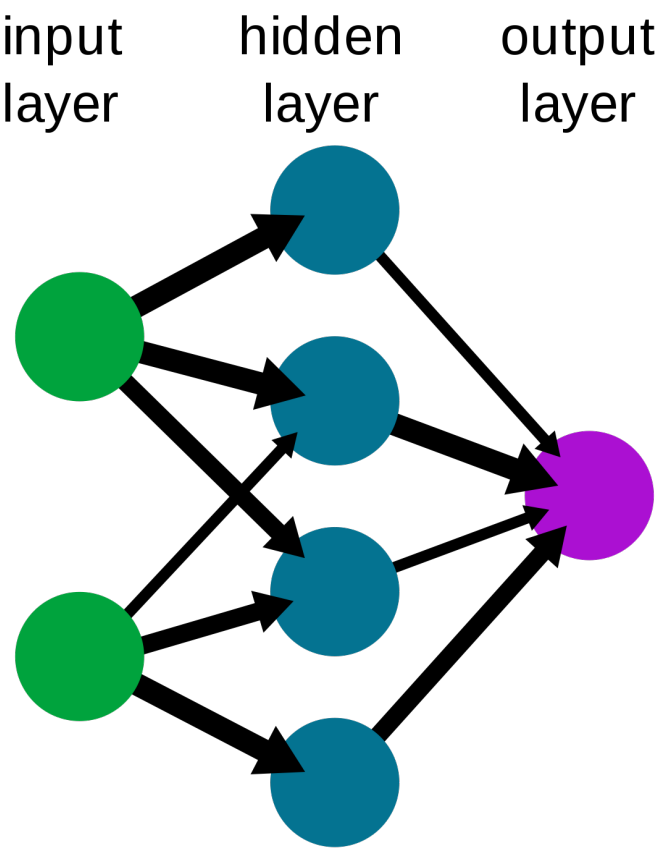
- By leveraging neural networks in DQN we can hope to:
  - Handle **more complex states**, like images, which are not easily represented as table entries.
  - Take advantage of **similarities between states** to learn as much as possible from each observation.
  - Be able to react to **previously unseen states** based on experience of similar situations.

Single-slide recap: Neural Networks

- Information flows from left to right via nodes ("neurons") and connections ("synapses").
- The output of a node y_j is computed as weighted sum of its inputs x_i with subsequent non-linear activation function σ.

$$y_j = \sigma\left(\sum_i w_{ij} \cdot x_i\right)$$

- The weights in the network are **optimized via gradient descent** to **minimize a loss function** based on a **training dataset** of input-output pairs.
- When trained with enough data, a neural network **can represent a wide variety of functions**.



DQN – In Practice

- In practice, instead of learning a function to estimate $\mathrm{Q}(s, a)$ for a given state-value pair, we instead learn a function which **estimates the Q-values for all actions of a given state**:

$$f(s) = [\mathrm{Q}(s, a_1), \ldots, \mathrm{Q}(s, a_n)]$$

- This is more efficient since we need to compute the **network output only once per state** (instead of once per state-action pair) when determining $(\arg)\max \mathrm{Q}(s, a)$.
- **During inference**, we perform action i corresponding to the **maximum $\mathrm{Q}(s, a_i)$** in this vector. If there are multiple maxima, we randomly break ties.
- **During training**, we use an **ϵ-greedy algorithm to balance exploration and exploitation**:
  - With probability (ϵ) select an action at random.
  - With probability $(1 - \epsilon)$ select the optimal action (as during inference).

DQN Training

- DQN is trained via **temporal-difference learning**. Recall:

$$\mathrm{Q}(s_t, a_t) \coloneqq \mathrm{Q}(s_t, a_t) + \lambda \cdot \overbrace{\Big[\underbrace{r_t + \gamma \cdot \max_a \mathrm{Q}(s_{t+1}, a)}_{\text{TD target}} - \mathrm{Q}(s_t, a_t)\Big]}^{\text{TD residual}}$$

- DQN simply uses **gradient descent to minimize the TD-residual**:

$$\Big[\Big(r_t + \gamma \cdot \max_a \mathrm{Q}(s_{t+1}, a)\Big) - \mathrm{Q}(s_t, a_t)\Big]^2$$

DQN Pros and Cons

Pros

- Compresses Q-table to handle high-dimensional state spaces.
- Can handle states not seen during training.
- Q-network acts as feature extractor which enables generalization of Q-values.
- Can leverage both on- and off-policy updates for sample-efficiency.

Cons

- Cannot directly be applied to continuous action spaces (would need to discretize).
- Deterministic, cannot learn stochastic policies.
- Need to separately add ϵ-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.

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.

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

DQN Extensions

Double DQN

- Use value estimate from target network, but action from online network.

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

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

DQN Extensions

Double DQN

- Use value estimate from target network, but action from online network.

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

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

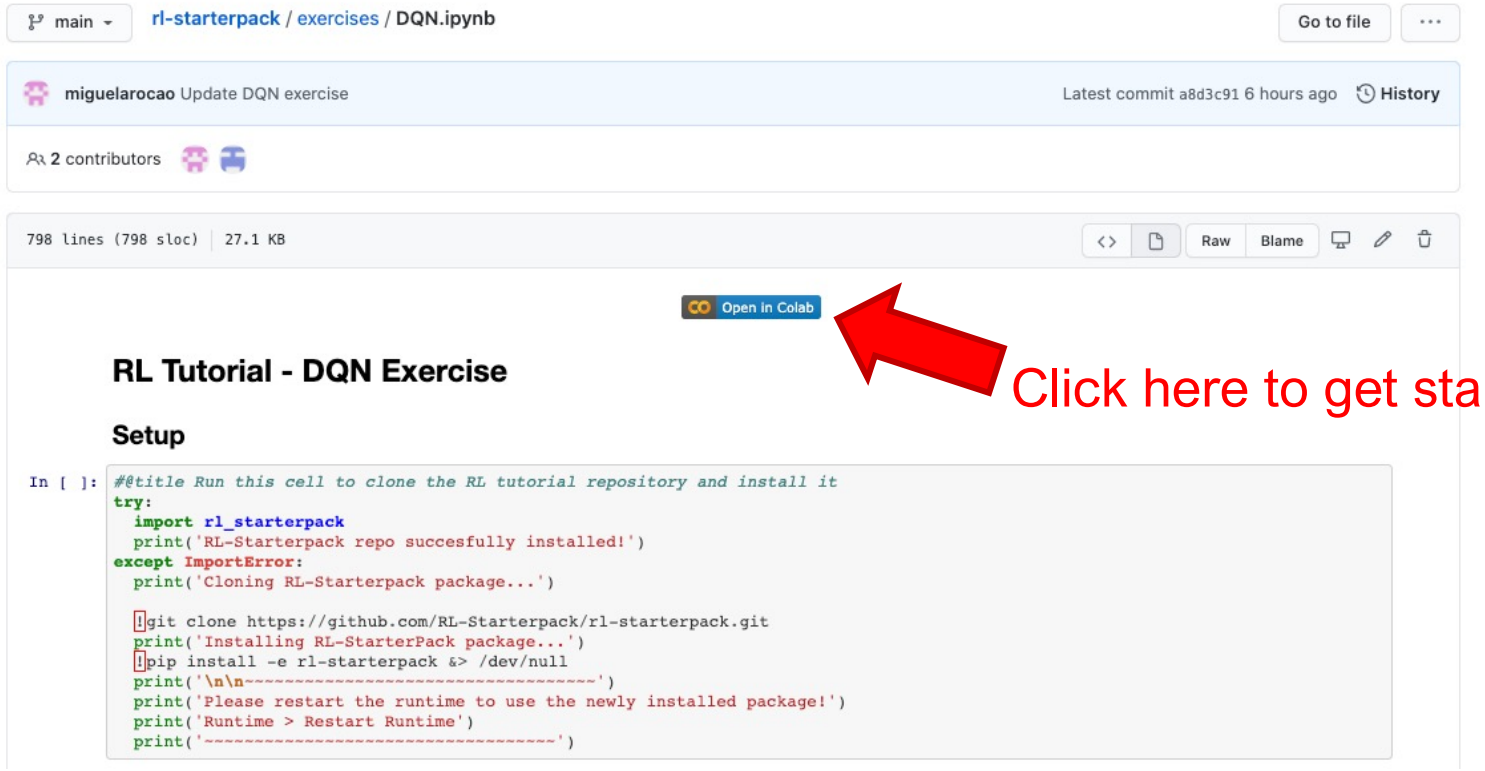
And more...

- **Prioritized replay memory**: instead of randomly replaying experience.
- **Dueling DQN:** Split estimating $\mathrm{Q}(s_t, a_t)$ into estimating $\mathrm{V}(s_t) + \mathrm{A}(s_t, a_t)$.
- etc...

DQN Exercise

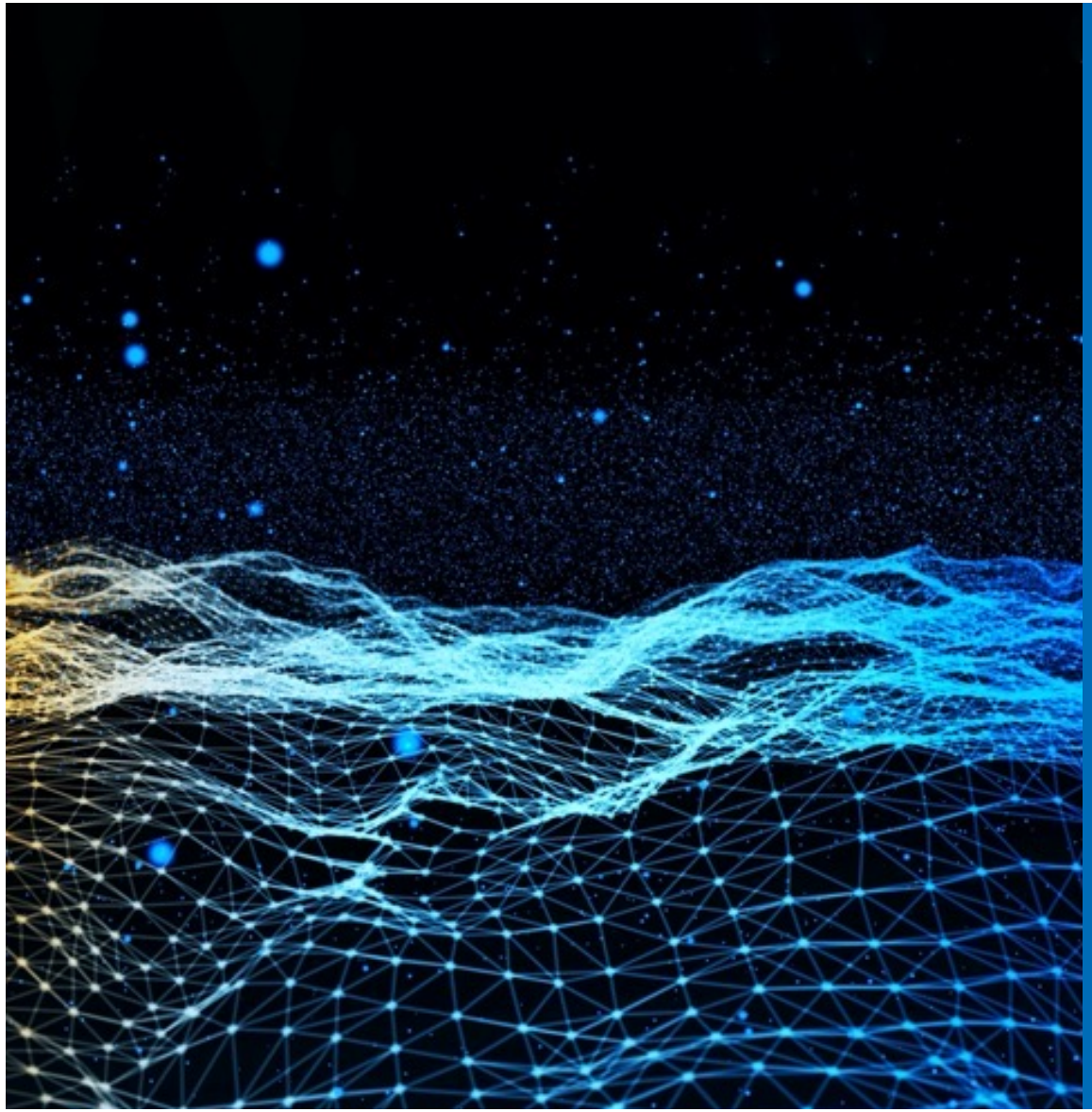
Access from **Quick Links @ rl-starterpack.github.io/#quick-links**

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



Click here to get started in Colab!

blueprism AI Labs



POLICY GRADIENT METHODS

Alexander Kuhnle

Policy Gradient Motivation

- Q-learning **cannot handle continuous action spaces**.
  - Due to computing $\arg\max_a \mathrm{Q}(s_t, a_t)$ to select an optimal action.

- Q-learning **is not compatible with a stochastic policy model**.
  - This would enable us to balance exploration and exploitation based on uncertainty.

- Q-learning **does not directly optimize the policy**.
  - Actions are inferred from the learned Q-values by a greedy policy.

Let's try to optimize the policy directly instead!

Policy Gradient

- In Policy Gradient methods, we **directly model a parametrized distribution** as the policy:

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

- We train the policy by optimizing a **maximization problem**. We want to maximize a measure of the agent's performance $J(\theta)$:

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

- Typically, for $J(\theta)$ we use the **expected discounted episode return** given the current policy:

$$J(\theta) = \mathrm{E}_{\tau\sim\pi_\theta}[\mathrm{R}(\tau)] \quad \text{where} \quad \mathrm{R}(\tau) = \sum_{t=0}^{\infty} \gamma^t \cdot r_t$$

Policy Gradient

- Maximize $\mathrm{E}_{\tau\sim\pi_\theta}[R(\tau)]$ with gradient descent:

$$\nabla_\theta \mathrm{E}_{\tau\sim\pi_\theta}[\mathrm{R}(\tau)] \;=\; \nabla_\theta \int \mathrm{R}(\tau)\cdot\pi_\theta(\tau)\,\mathrm{d}\tau$$

- Computing the gradient of the expectation is tricky because it depends on both:
  - The action selection (directly depends on π_θ)
  - The trajectory rewards (indirectly depends on π_θ)

- Fortunately, the **Policy Gradient Theorem** tells us that expectation and gradient can be swapped as follows:

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

Policy Gradient: REINFORCE

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

- 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 θ at random.
2. Generate one, or multiple, episode trajectories using π_θ (on-policy).
3. Compute the discounted episode return $R(\tau)$.
4. For each timestep in that trajectory ($t = 1, \ldots, T$):
   a) Update the policy parameters: $\theta \leftarrow \theta + \alpha \cdot \mathrm{R}(\tau) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)$.
5. Continue with 2. until convergence.

Policy Gradient – In Practice

- We can **model our policy with a neural network** which takes in the current state as 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:

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

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

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

Policy Gradient – In Practice

- By sampling from the policy during training, we **naturally balance exploration and exploitation**:
  - As training progresses, the policy learns to assign more probability mass to high-value actions. Consequently, we implicitly learn to exploit more and explore less.
- Note that **each training update impacts all actions for a given state**, unlike Q-learning where each update impacts only a single state-action pair.
  - Shifting probability mass for the target action based on the experienced return implicitly shifts all action probabilities.
- **During inference**, we usually take the **most likely action** according to the action-distribution.

Policy Gradient Pros and Cons

Pros

- Handles stochastic policies, which enables intrinsic exploration vs exploitation trade-off.
- Handles continuous action spaces.
- Optimizes directly for the goal of maximizing returns, instead of surrogate Q-values.

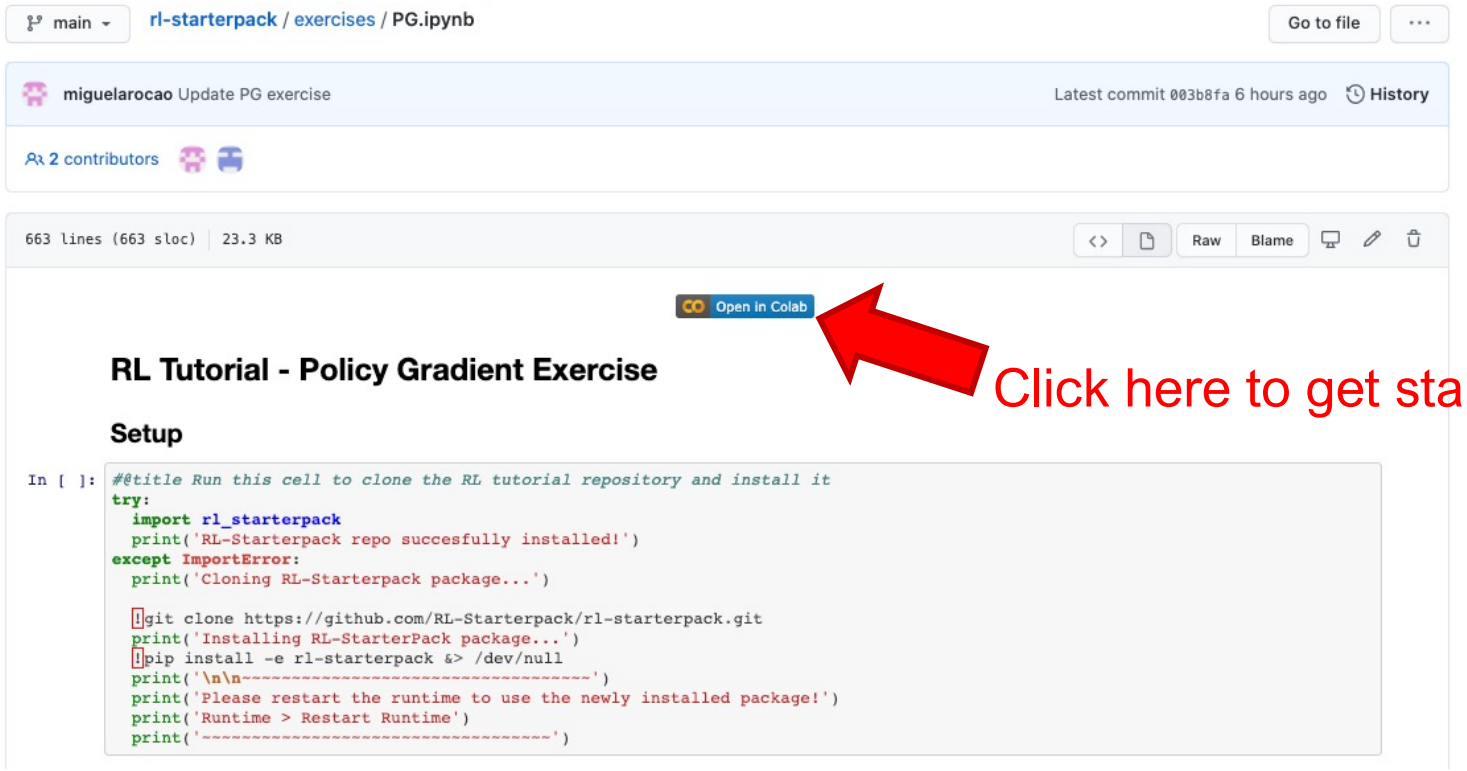
Cons

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

Policy Gradient Exercise

Access from **Quick Links @ rl-starterpack.github.io/#quick-links**

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



Click here to get started in Colab!

ACTOR-CRITICS

Miguel Aroca-Ouellette

Problems with REINFORCE

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

Problems with REINFORCE

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

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

Problems with REINFORCE

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

- 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:

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

Actor-critic I

- After some opaque maths:

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

- *"Reward to go"*:

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

Actor-critic I

- After some opaque maths:

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

- *"Reward to go"*:

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

- Actor-critic I:

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

Actor-critic I

- Actor-critic I:

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

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

Actor-critic I

- Actor-critic I:

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

- We train two models simultaneously:
  - *"Actor"* policy $\pi_\theta(a|s)$ using above policy gradient formulation
  - *"Critic"* Q-value function $\mathrm{Q}(s,a)$ to approximate $\mathrm{Q}^{\pi_\theta}(s,a)$ 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 $b(s)$:

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

Actor-critic II

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

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

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

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

Actor-critic II

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

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

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

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

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

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

Actor-critic II

- Actor-critic II:

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

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

Actor-critic II

- Actor-critic II:

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

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

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

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:

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

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

$$\hat{g}_{\mathrm{REINFORCE}} = 0.2 \cdot 6 \cdot \nabla_\theta \log \pi_\theta(1|s) + 0.5 \cdot 7 \cdot \nabla_\theta \log \pi_\theta(2|s) + 0.3 \cdot 8 \cdot \nabla_\theta \log \pi_\theta(3|s)$$

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:

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

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

$$\hat{g}_{\mathrm{REINFORCE}} = 0.2 \cdot 6 \cdot \nabla_\theta \log \pi_\theta(1|s) + 0.5 \cdot 7 \cdot \nabla_\theta \log \pi_\theta(2|s) + 0.3 \cdot 8 \cdot \nabla_\theta \log \pi_\theta(3|s)$$

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

$$b(s) = 7 \approx \mathrm{V}^{\pi_\theta}(s) \implies \hat{g}_{\mathrm{Baseline}} = -0.2 \cdot \nabla_\theta \log \pi_\theta(1|s) + 0.3 \cdot \nabla_\theta \log \pi_\theta(3|s)$$

Actor-critic policy gradient versions

- Actor-critic I: *"Q actor-critic"* (AC)

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

Actor-critic policy gradient versions

- Actor-critic I: *"Q actor-critic"* (AC)

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

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

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

Actor-critic policy gradient versions

- Actor-critic I: *"Q actor-critic"* (AC)

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

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

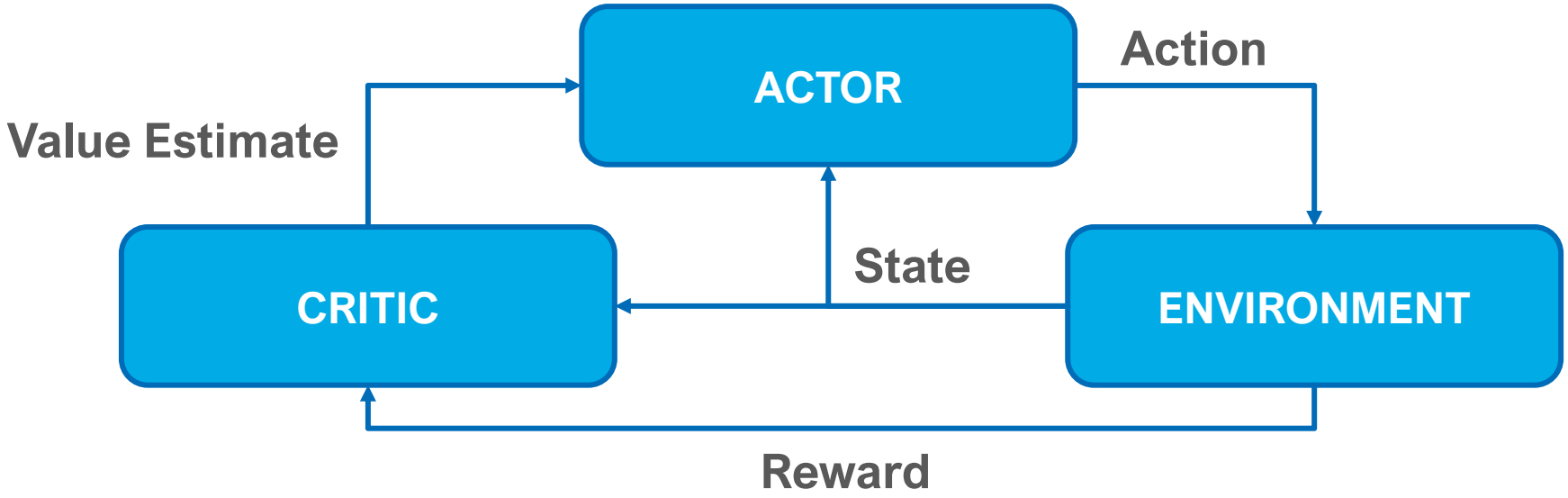
$$\mathrm{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\underbrace{\left(\mathrm{R}(\tau[t:])-\mathrm{V}^{\pi_\theta}(s)\right)}_{\text{Advantage }\mathrm{A}^{\pi_\theta}(s,a)}\cdot\nabla_\theta\log\pi_\theta(a_t|s_t)\right]$$

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

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

Actor-critic in practice

- Both actor and critic are randomly initialized at the beginning of training
- 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**
- What does training both these models look like in practice?



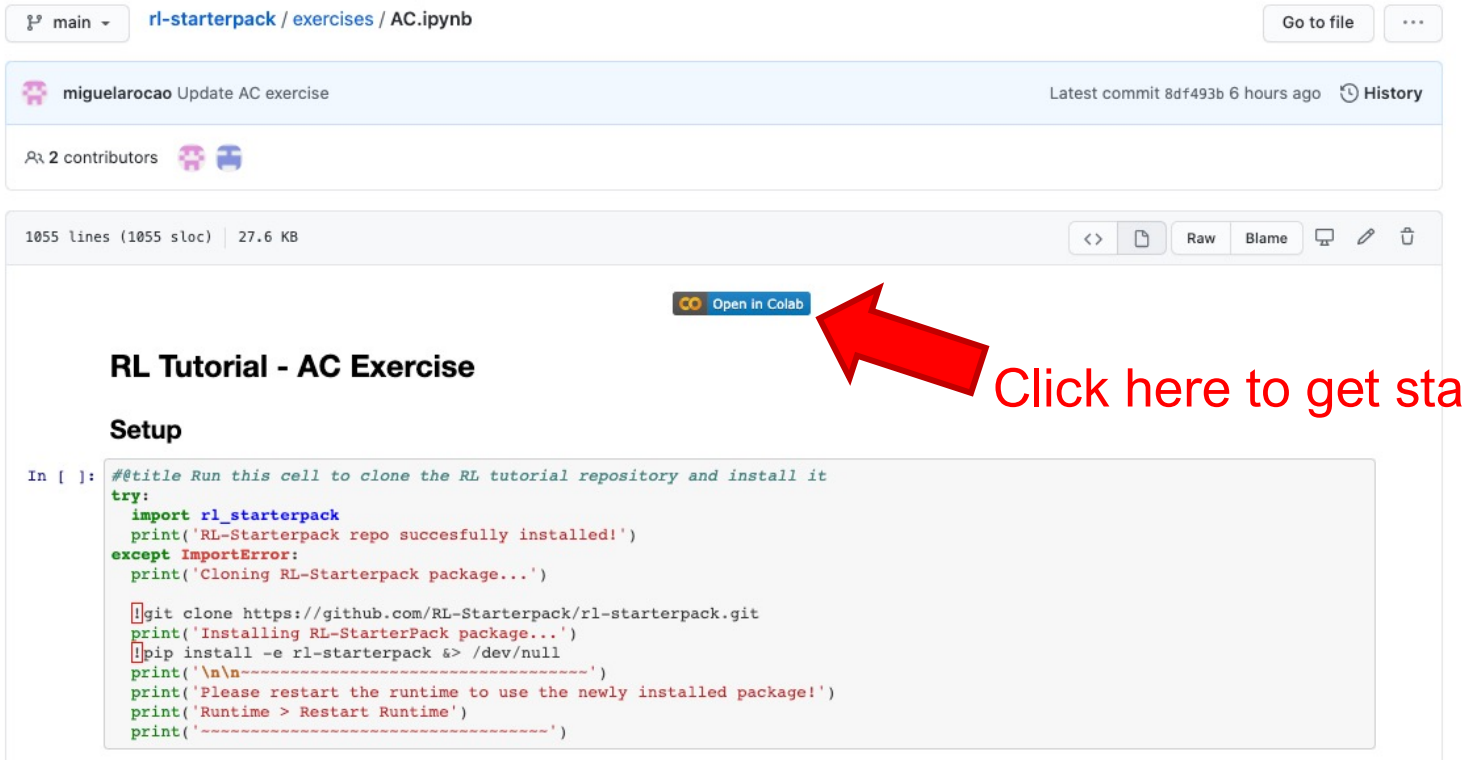
Actor-critic in practice

- Let's take training these models to both extremes:
  - **Only train actor:** A very certain actor means no useful experience data for critic to improve value function
  - **Only train critic:** A very accurate critic means close-to-zero advantage reward for actor (in A2C)
    - ➢ Desired behavior if task is solved, but undesired behavior if actor is stuck prematurely
- Thus, critic and actor need to **evolve in tandem**
- In practice both are updated simultaneously, however variations exist. For example:
  - Only update the actor when critic is "sufficiently" accurate
  - Update the critic more often than actor

Actor-critic Exercise

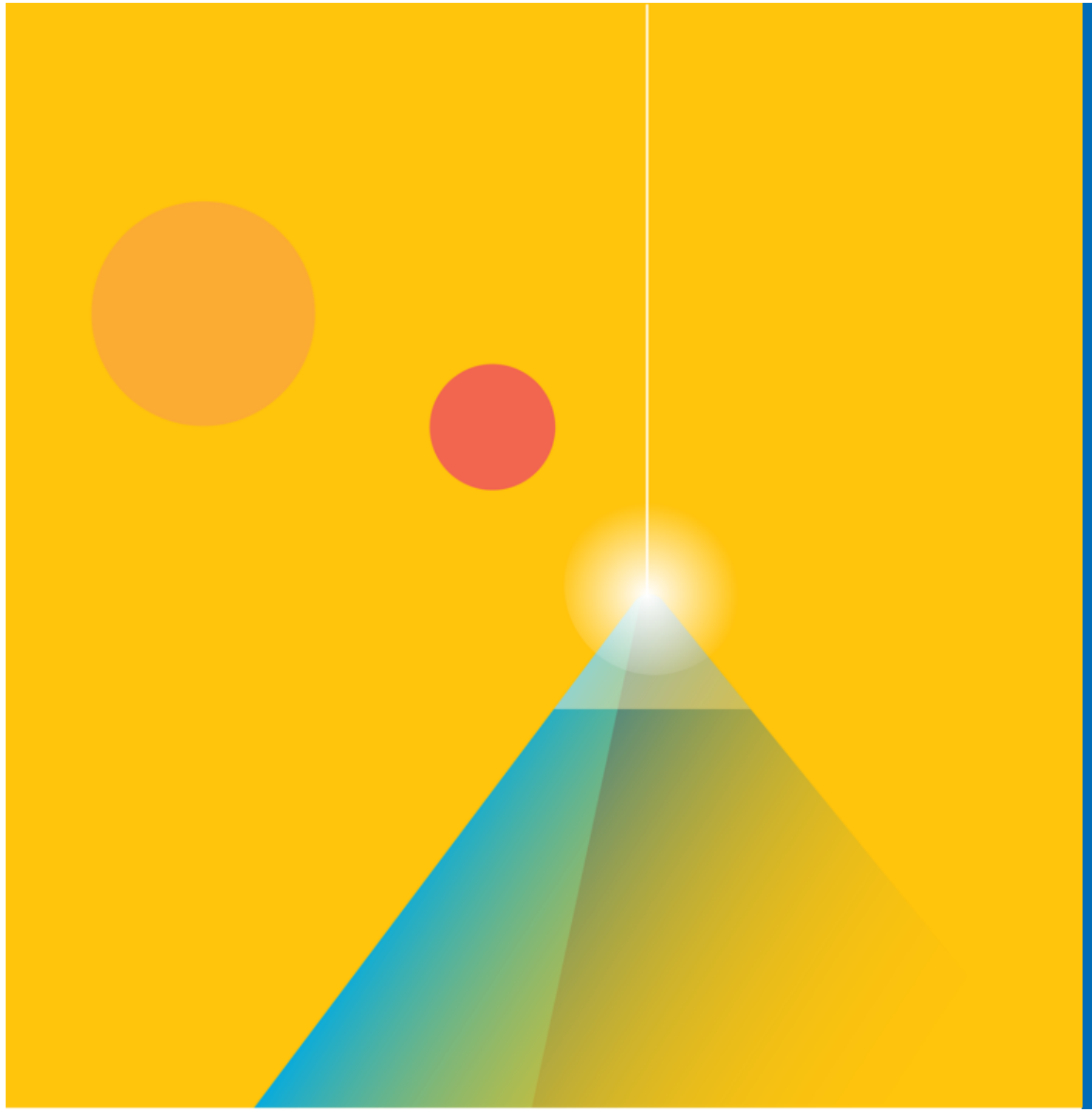
Access from **Quick Links** @ **rl-starterpack.github.io/#quick-links**

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



Click here to get started in Colab!

blueprism AI Labs



OUTLOOK

Miguel Aroca-Ouellette

blueprism AI Labs

Data Collection

- Simultaneously collecting data and training agents can be very slow. How to improve?
- Assuming we have simulated environment available, we could:
  - Parallelize environment interaction
  - Distribute model training

Massively Parallel Methods for Deep Reinforcement Learning

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, David Silver
{ARUNSNAIR, PRAV, BLACKWELLS, CAGDASALCICEK, RORYF, ADEMARIA, DARTHVEDA, MUSTAFASUL, CBEATTIE, SVP, LEGG, VMNIH, KORAYK, DAVIDSILVER @GOOGLE.COM }
Google DeepMind, London

https://arxiv.org/pdf/1507.04296.pdf

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM
[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

https://arxiv.org/pdf/1602.01783.pdf

Learning from demonstrations

- What if you already have data?
- **Imitation learning:** training policy to imitate what I see in demonstrations
- **Inverse reinforcement learning:** learning reward function from demonstrations
- Useful in real-world problems (i.e. autonomous vehicles)

Generative Adversarial Imitation Learning

Jonathan Ho
Stanford University
hoj@cs.stanford.edu

Stefano Ermon
Stanford University
ermon@cs.stanford.edu

https://arxiv.org/pdf/1606.03476.pdf

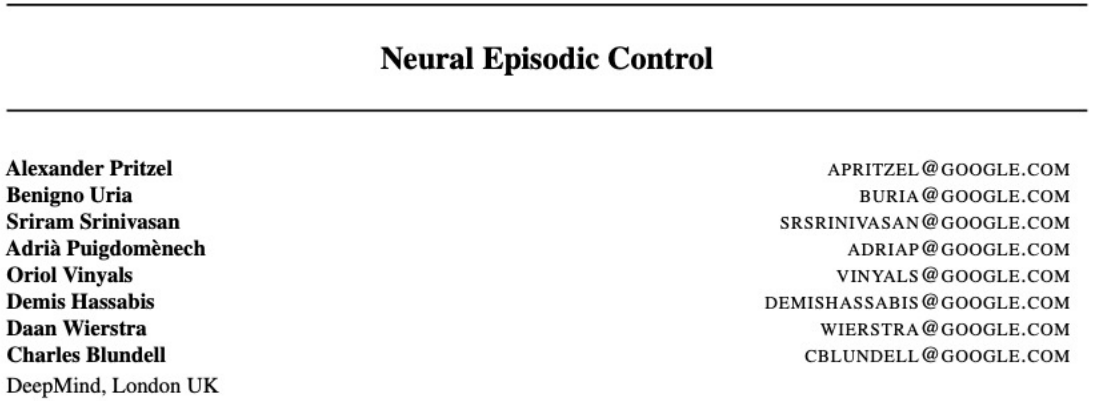
Algorithms for Inverse Reinforcement Learning

Andrew Y. Ng ANG@CS.BERKELEY.EDU
Stuart Russell RUSSELL@CS.BERKELEY.EDU
Computer Science Division, U.C. Berkeley, Berkeley, CA 94720 USA

https://ai.stanford.edu/~ang/papers/icml00-irl.pdf

Internal State

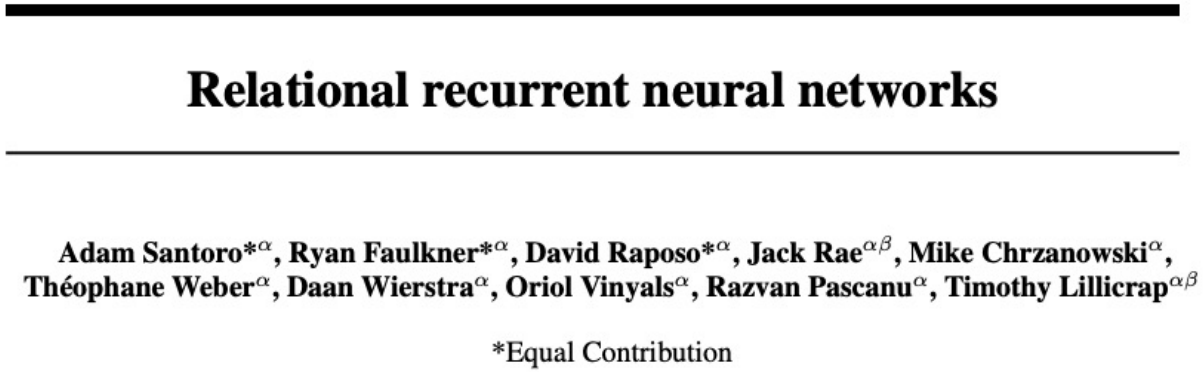
- So far only looked at “reactive” agents: yields action considering only current state
- What if we designed agents with memory?
- Could use RNN based models to allow **state that evolves over time**
- Potentially allows for **long-term planning**, but complicates training



Neural Episodic Control

Alexander Pritzel APRITZEL@GOOGLE.COM
Benigno Uria BURIA@GOOGLE.COM
Sriram Srinivasan SRSRINIVASAN@GOOGLE.COM
Adrià Puigdomènech ADRIAP@GOOGLE.COM
Oriol Vinyals VINYALS@GOOGLE.COM
Demis Hassabis DEMISHASSABIS@GOOGLE.COM
Daan Wierstra WIERSTRA@GOOGLE.COM
Charles Blundell CBLUNDELL@GOOGLE.COM
DeepMind, London UK

https://arxiv.org/pdf/1703.01988.pdf



Relational recurrent neural networks

Adam Santoro*[α], Ryan Faulkner*[α], David Raposo*[α], Jack Rae[αβ], Mike Chrzanowski[α], Théophane Weber[α], Daan Wierstra[α], Oriol Vinyals[α], Razvan Pascanu[α], Timothy Lillicrap[αβ]

*Equal Contribution

https://arxiv.org/pdf/1806.01822.pdf

blueprism AI Labs

Predicting the future

- Add auxiliary loss based on agent's prediction of future
- Useful when reward is sparse
- Provides agents with a kind of **intrinsic motivation** or **curiosity**
- Allows for more meaningful exploration vs exploitation trade-off

Curiosity-driven Exploration by Self-supervised Prediction

Deepak Pathak [1] Pulkit Agrawal [1] Alexei A. Efros [1] Trevor Darrell [1]

https://arxiv.org/pdf/1705.05363.pdf

EXPLORATION BY RANDOM NETWORK DISTILLATION

Yuri Burda* OpenAI
Harrison Edwards* OpenAI
Amos Storkey Univ. of Edinburgh
Oleg Klimov OpenAI

https://arxiv.org/pdf/1810.12894.pdf

Goal-parametrized policies

- What if we want our agent to be able to handle multiple goals?
- For example: tell agent ”pick up the red cube” or “pick up the green sphere”:
- We could modify our agent to incorporate the goal in its policy

Unsupervised Predictive Memory in a Goal-Directed Agent

Greg Wayne*,[1], Chia-Chun Hung*,[1], David Amos*,[1], Mehdi Mirza[1], Arun Ahuja[1], Agnieszka Grabska-Barwińska[1], Jack Rae[1], Piotr Mirowski[1], Joel Z. Leibo[1], Adam Santoro[1], Mevlana Gemici[1], Malcolm Reynolds[1], Tim Harley[1], Josh Abramson[1], Shakir Mohamed[1], Danilo Rezende[1], David Saxton[1], Adam Cain[1], Chloe Hillier[1], David Silver[1], Koray Kavukcuoglu[1], Matt Botvinick[1], Demis Hassabis[1], Timothy Lillicrap[1].

[1]DeepMind, 5 New Street Square, London EC4A 3TW, UK.

*These authors contributed equally to this work.

https://arxiv.org/pdf/1803.10760.pdf

Planning with Goal-Conditioned Policies

Soroush Nasiriany,* Vitchyr H. Pong,* Steven Lin, Sergey Levine
University of California, Berkeley
{snasiriany,vitchyr,stevenlin598,svlevine@berkeley.edu}

https://arxiv.org/pdf/1911.08453.pdf

Model-based influences

- Can we combine model-based methods with model-free?
- One approach is to learn a “world model”
- Can (exclusively) train using a model of the environment

World Models

David Ha [1] **Jürgen Schmidhuber** [2 3]

https://arxiv.org/pdf/1803.10122.pdf

DREAM TO CONTROL: LEARNING BEHAVIORS BY LATENT IMAGINATION

Danijar Hafner *
University of Toronto
Google Brain

Timothy Lillicrap
DeepMind

Jimmy Ba
University of Toronto

Mohammad Norouzi
Google Brain

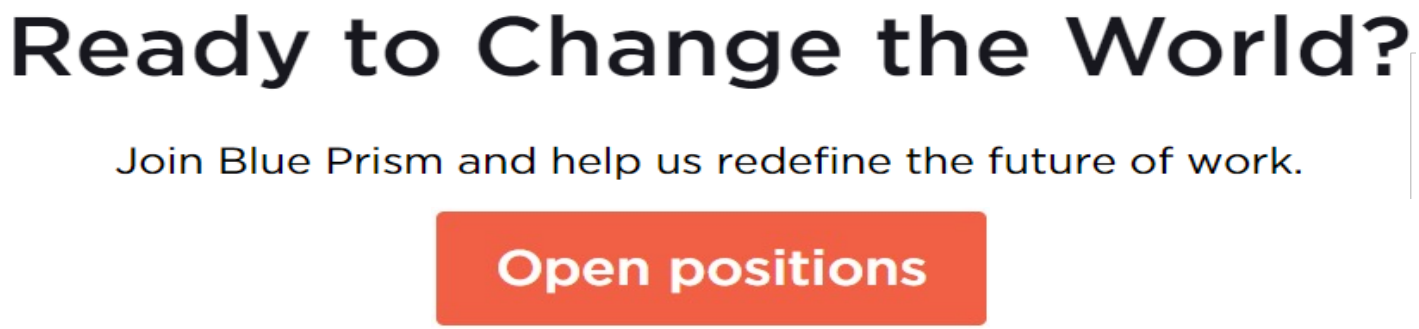
https://arxiv.org/pdf/1912.01603.pdf

Outlook Summary

- **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
- …and more

We Are Hiring

- **Research Engineer** (AI Labs)
- **Python Engineer** (AI Labs)
- **Senior Python Engineer** (AI Labs)



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, murat.sensoy}@blueprism.com

Thanks & Acknowledgements

blueprism
AI Labs

Feedback

Please help us improve this tutorial!

We have a short 2min feedback form:

https://forms.gle/hJBUkYR1sMQE8RkC9

Your feedback is greatly appreciated!