off-policy deep RL

Remi Munos

DeepMind **Paris**

How to do fundamental research in deep RL?

Ingredients of a satisfying research:

1- The need

- Observe limitation of current approaches
- Identify the core problem

2- The idea

- Design an algorithm

3- The benefit

- Theoretical analysis in simplified setting
- Improved numerical performance

Off-policy deep RL

- The need
  - Limitations of DQN and A3C
  - off-policy, multi-steps RL

- The idea:
  - Truncated importance sampling while preserving contraction property
  - The algorithm: Retrace

- The benefit:
  - Convergence to optimal policy in finite state spaces
  - Practical algorithms (ACER, Reactor, MPO, Impala)

Introduction to Reinforcement Learning (RL)

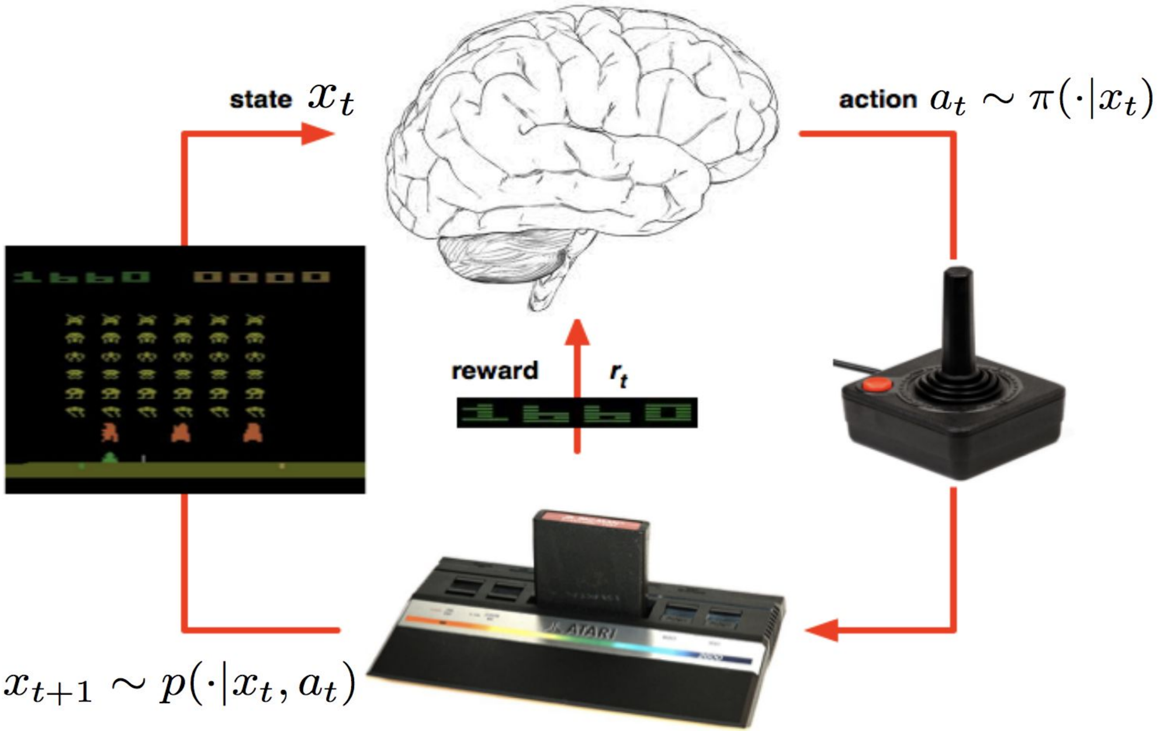
- Learn to make good decisions
- No supervision. Learn from rewards

I learned to ride with RL...

Two approaches:

- Value based ([Bellman, 1957]'s dynamic programming)
- Policy based ([Pontryagin, 1956]'s maximum principle)

The RL agent in its environment



Bellman's dynamic programming

- Define the value function Q^π of a policy $\pi(a|x)$:

$$Q^\pi(x,a) = \mathbb{E}\Big[\sum_{t\geq 0} \gamma^t r_t \Big| x, a, \pi\Big],$$

and the optimal value function:

$$Q^*(x,a) = \max_\pi Q^\pi(x,a).$$

(expected sum of future rewards if the agent plays optimally).

- Bellman equations:

$$Q^\pi(x,a) = r(x,a) + \gamma \mathbb{E}_{x'}\Big[\sum_{a'} \pi(a'|x') Q^\pi(x',a') \Big| x, a\Big]$$

$$Q^*(x,a) = r(x,a) + \gamma \mathbb{E}_{x'}\Big[\max_{a'} Q^*(x',a') \Big| x, a\Big]$$

- Optimal policy $\pi^*(x) = \arg\max_a Q^*(x,a)$

Represent Q using a neural network

- Represent value function $Q_w(x, a)$ with a neural net.
- How to train $Q_w(x, a)$? We don't have supervised values. We only know we want

$$Q_w(x, a) \approx r(x, a) + \gamma \mathbb{E}_{x'} \left[\max_{a'} Q_w(x', a') \middle| x, a \right]$$

- After a transition $x_t, a_t \to x_{t+1}$,

$$\text{train } Q_w(x_t, a_t) \text{ to predict } \underbrace{r_t + \gamma \max_a Q_w(x_{t+1}, a)}_{\text{target values}}$$

- Minimize loss $\Big(\underbrace{r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)}_{\text{temporal difference } \delta_t} \Big)^2$.
- At the end of learning, $\mathbb{E}[\delta_t] = 0$.

Deep Q-Networks (DQN) [Mnih et al. 2013, 2015]

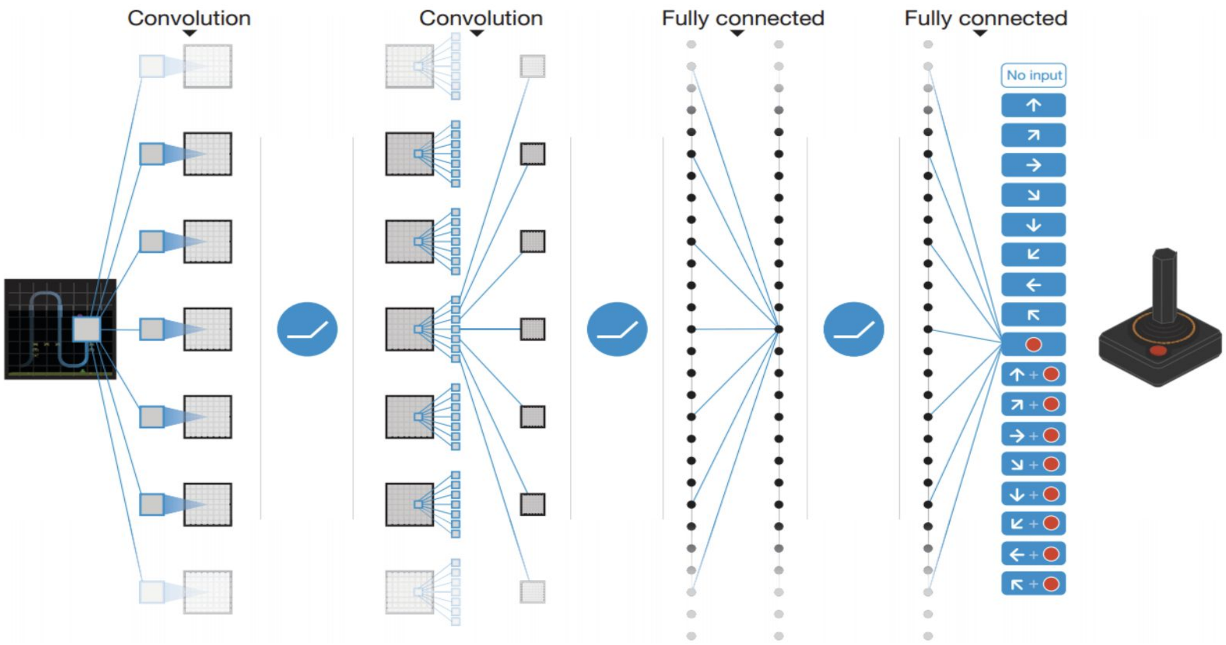
Problems: (1) data is not iid, (2) target values change
Idea: be as close as possible to supervised learning

1. Dissociate acting from learning:
   - Interact with the environments by following behavior policy
   - Store transition samples x_t, a_t, x_{t+1}, r_t into a memory replay
   - Train by replaying iid from memory
2. Use target network fixed for a while

$$\text{loss} = \left(r_t + \gamma \max_a Q_{w_{target}}(x_{t+1}, a) - Q_w(x_t, a_t) \right)^2$$

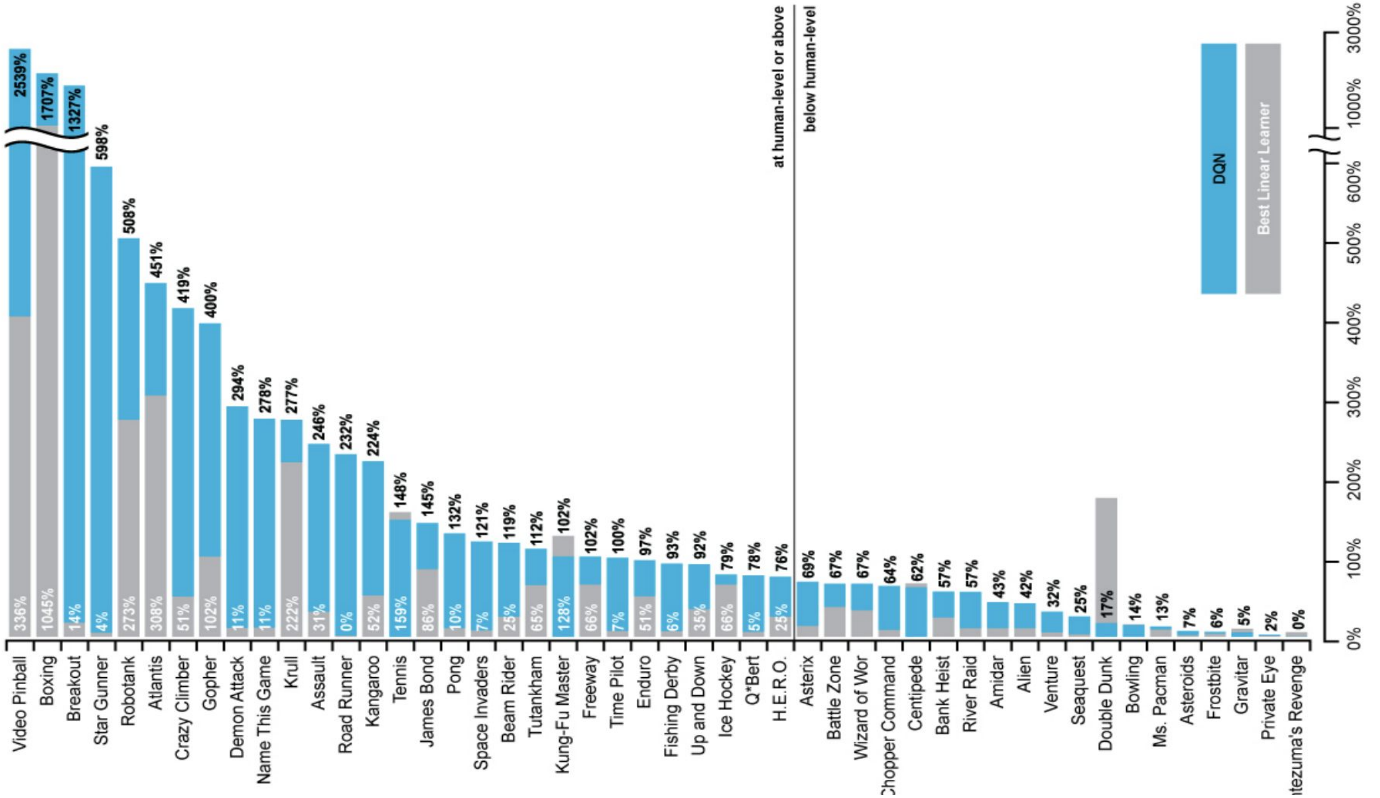
Properties: DQN is off-policy, and uses 1-step bootstrapping.

DQN network



Convolution
Convolution
Fully connected
Fully connected
No input

DQN Results in Atari



Pontryagin's maximum principle

- Parametrized policy $\pi_\theta(a|x)$
- **Policy gradient**: optimize

$$J(\theta) = \mathbb{E}\Big[\sum_{t\geq 0} \gamma^t r_t \Big| a_t \sim \pi_\theta(\cdot|x_t)\Big],$$

by gradient ascent:

$$\nabla J(\theta) = \mathbb{E}\Big[\sum_{t\geq 0} \gamma^t \nabla \log \pi_\theta(a_t|x_t)\big(\underbrace{r_t + \gamma r_{t+1} + \dots}_{Q^{\pi_\theta}(x_t,a_t)}\big)\Big]$$

Actor-critic algorithm learn both π_θ and Q_w.

Asynchronous Advantage Actor-Critic (A3C)

A3C [Mnih et al., 2016] is an asynchronous actor-critic algorithm

- Learn state-value function $V^\pi(x) = \mathbb{E}\Big[\sum_{t\geq 0}\gamma^t r_t | x, \pi\Big]$ by minimizing a n-step Temporal Difference:

$$\Big(V_w(x_t) - \big(\underbrace{r_t + \gamma r_{t+1} + ... + \gamma^{n-1} r_{t+n-1} + \gamma^n V_w(x_{t+n})}_{n-\text{steps target value}}\big)\Big)^2$$

- Policy $\pi_\theta(a_t|x_t)$ improved by following gradient ascent:

$$\nabla \log \pi_\theta(a_t|x_t)\Big(r_t + \gamma r_{t+1} + ... + \gamma^{n-1} r_{t+n-1} + \gamma^n V_w(x_{t+n}) - V_w(x_t)\Big)$$

Properties: On-policy algorithm, uses multi-steps learning.

DQN versus A3C

DQN:

- Pros: Off-policy learning $\rightarrow$ use memory replay (sample efficiency), allow any exploration strategy
- Cons: One-step learning: Slow to propagate information, accumulates errors, no RNNs

A3C:

- Pros: Multi-steps learning $\rightarrow$ fast propagation of information, possible use of RNNs
- Cons: On-policy learning: does not allow memory replay, neither exploration

The Need: off-policy, multi-steps learning

Two desired properties of a RL algorithm:

- Off-policy learning
  - use memory replay
  - do exploration
  - lag between acting and learning
- Use multi-steps learning
  - propagate rewards rapidly
  - avoid accumulation of approximation/estimation errors
  - Allow learning from sequences (RNN)

Ex: Q-learning (and DQN) is off-policy but does not use multi-steps returns
Policy gradient (and A3C) use returns but are on-policy.

Both properties are important in deepRL. **Can we have both simultaneously?**

Off-policy reinforcement learning

Behavior policy $\mu(a|x)$, **target policy** $\pi(a|x)$

Observe trajectory $\{x_0 = x, a_0 = a, r_0, \ldots, x_t, a_t, r_t, \ldots\}$

where $a_t \sim \mu(\cdot|x_t)$, $r_t = r(x_t, a_t)$ and $x_{t+1} \sim p(\cdot|x_t, a_t)$

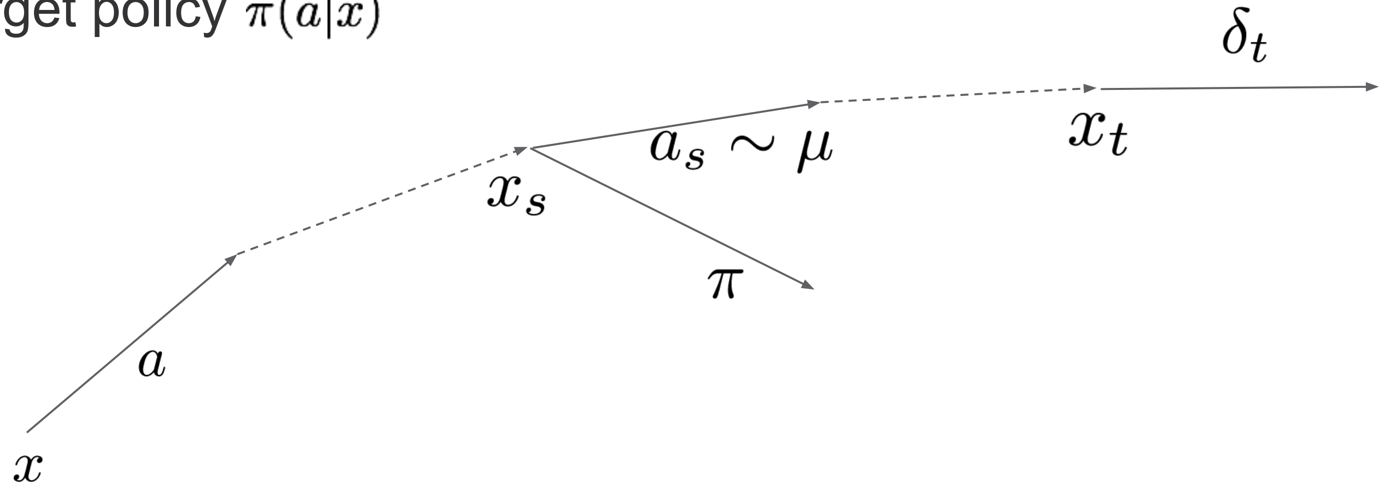
Goal:

- Policy evaluation: $Q^\pi(x,a) = \mathbb{E}\Big[\sum_{t\geq 0} \gamma^t r_t | x_0 = x, a_0 = a, \pi\Big]$
- Optimal control: $Q^*(x,a) = \max_\pi Q^\pi(x,a)$

Off-policy credit assignment problem

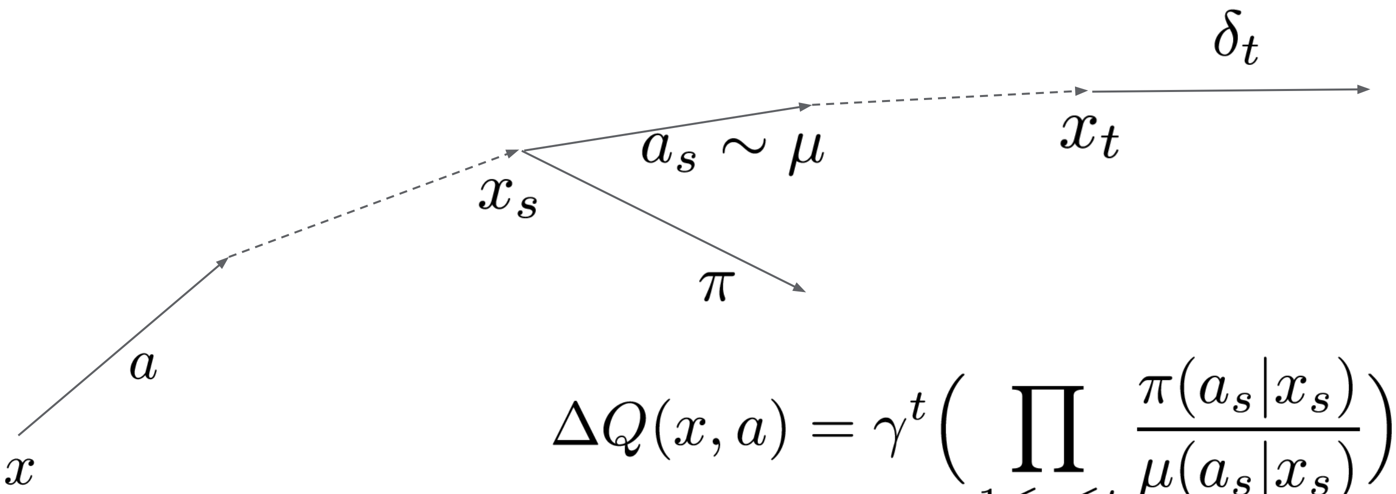
Behavior policy $\mu(a|x)$

Target policy $\pi(a|x)$



Can we use the TD δ_t to estimate $Q^\pi(x_s, a_s)$ for all $s \leq t$?

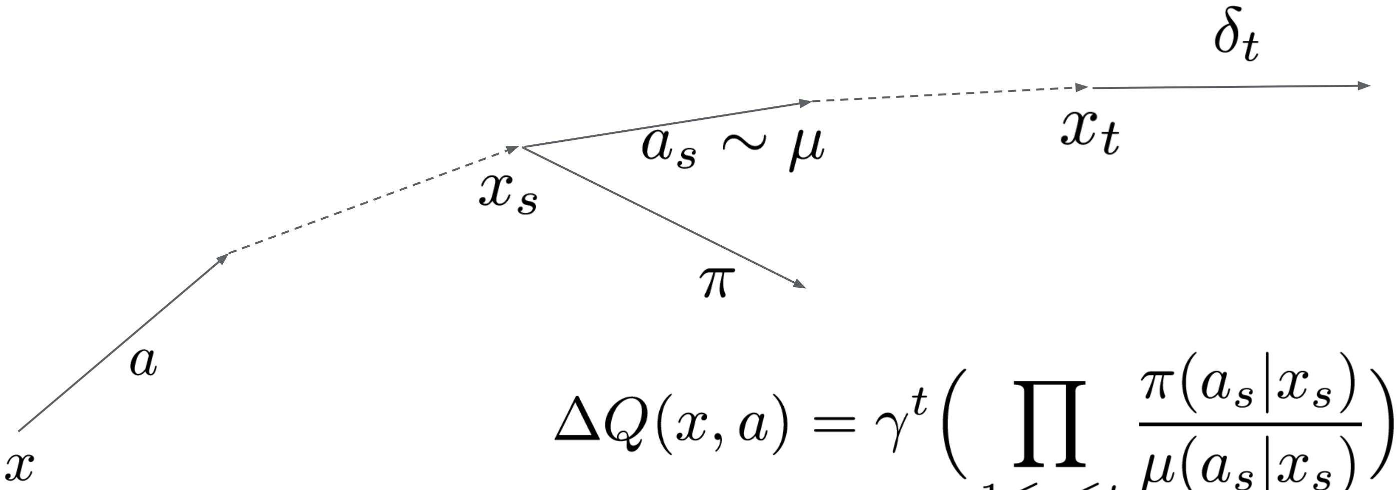
Importance sampling



$$\Delta Q(x,a) = \gamma^t \Big(\prod_{1 \le s \le t} \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \Big) \delta_t$$

Reweight the trace by the product of IS ratios

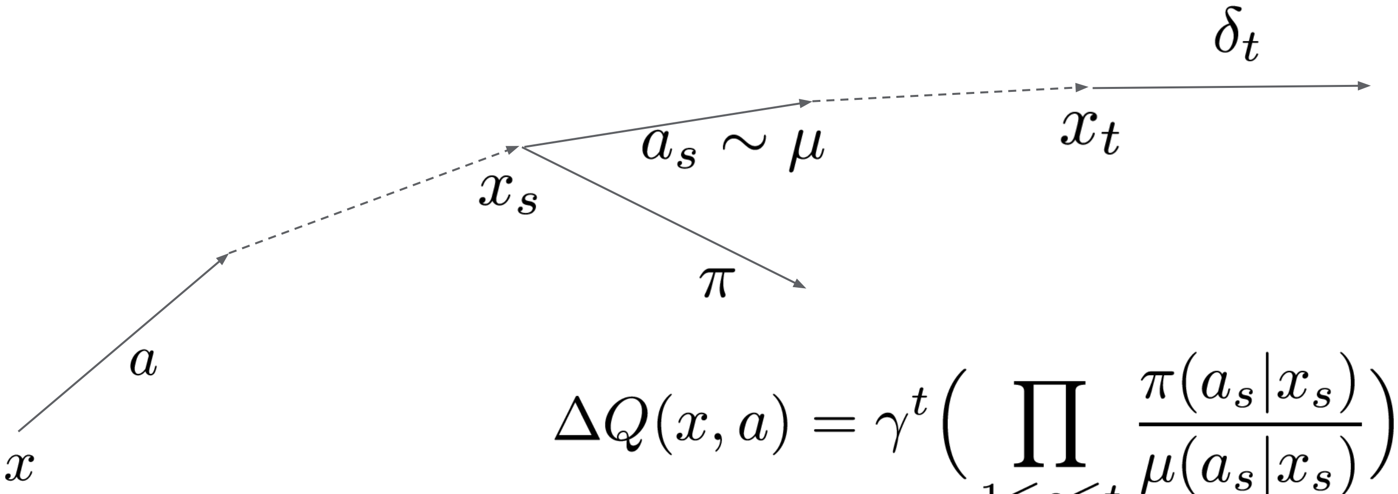
Importance sampling



$$\Delta Q(x,a) = \gamma^t \Big(\prod_{1 \le s \le t} \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \Big) \delta_t$$

Unbiased estimate of Q^π

Importance sampling



$$\Delta Q(x,a) = \gamma^t \Big(\prod_{1 \le s \le t} \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \Big) \delta_t$$

Unbiased estimate of Q^π

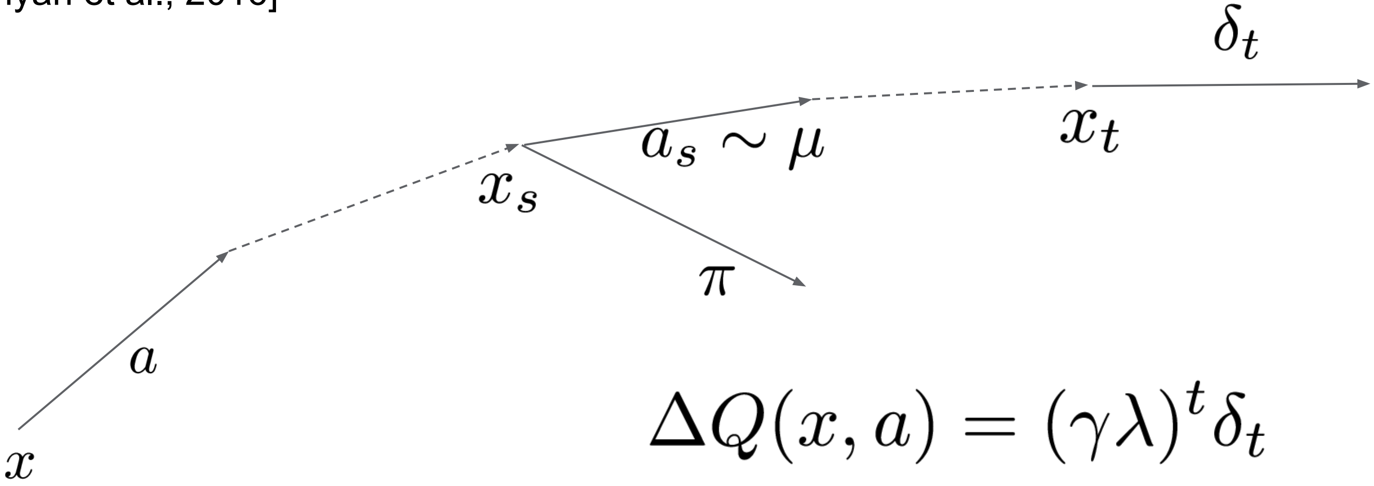
Large (possibly infinite) variance

[Precup, Sutton, Singh, 2000], [Mahmood, Yu, White, Sutton, 2015] ,...

Not stable!

$Q^\pi(\lambda)$ algorithm

[Harutyunyan et al., 2016]

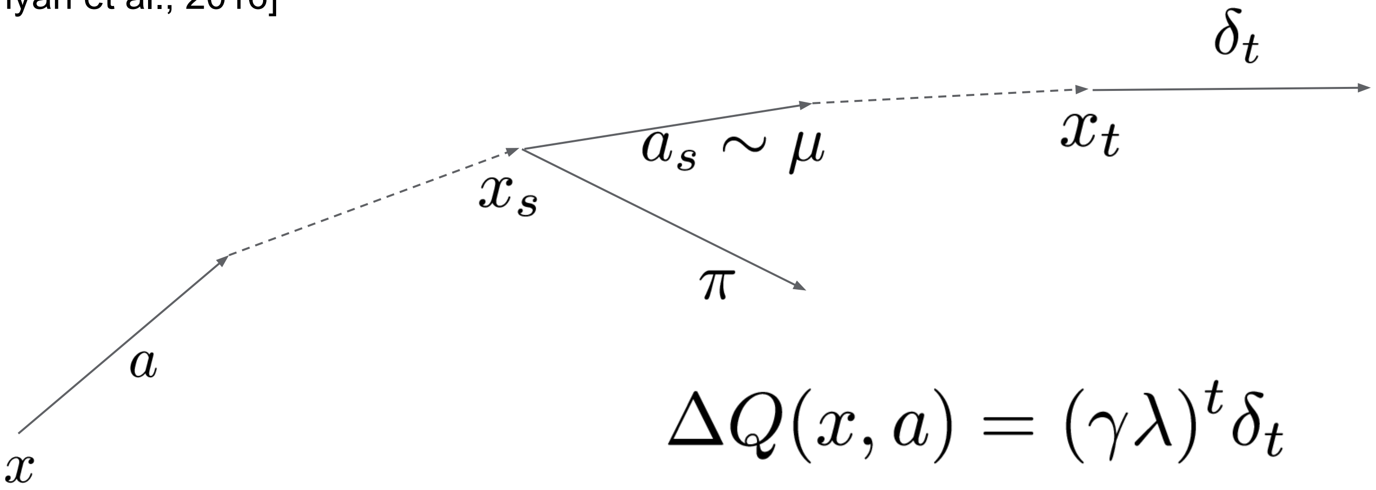


$$\Delta Q(x,a) = (\gamma\lambda)^t \delta_t$$

Cut traces by a constant λ^t

$Q^\pi(\lambda)$ algorithm

[Harutyunyan et al., 2016]

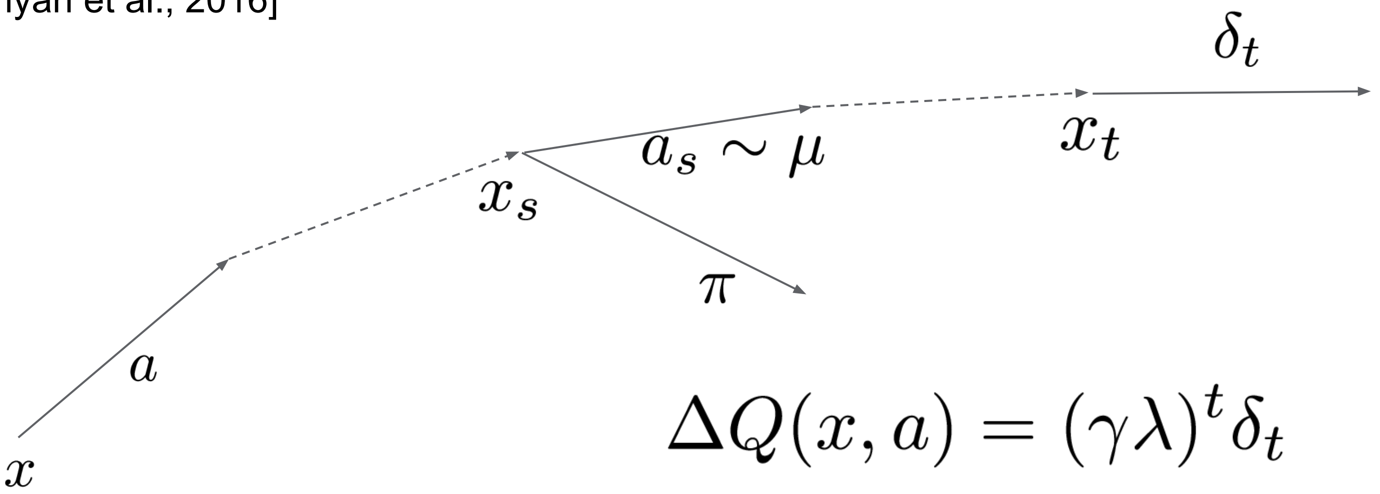


$$\Delta Q(x,a) = (\gamma\lambda)^t \delta_t$$

works if $\|\pi - \mu\|_1 \le \frac{1-\gamma}{\lambda\gamma}$

$Q^\pi(\lambda)$ algorithm

[Harutyunyan et al., 2016]



$$\Delta Q(x, a) = (\gamma\lambda)^t \delta_t$$

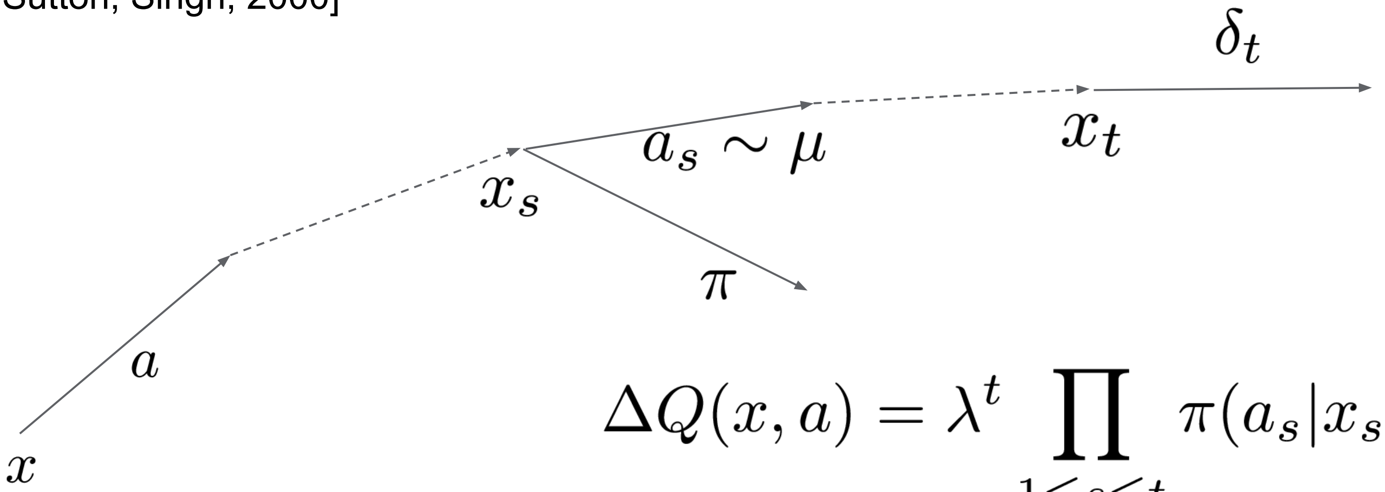
works if $\|\pi - \mu\|_1 \leq \frac{1-\gamma}{\lambda\gamma}$

may not work otherwise

No guarantee!

Tree backup TB(λ) algorithm

[Precup, Sutton, Singh, 2000]

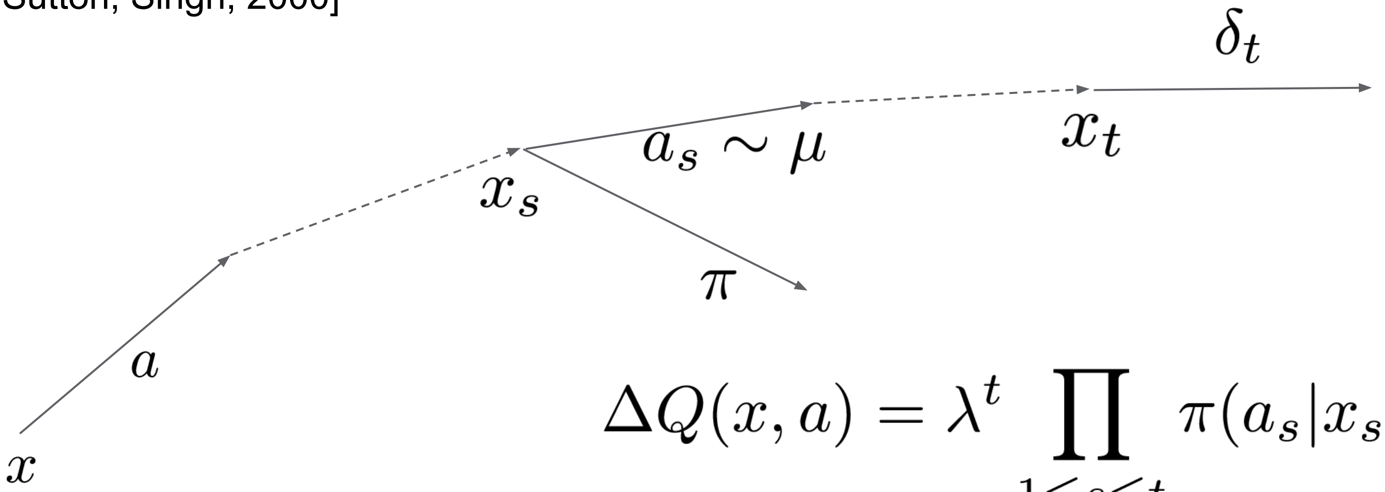


$$\Delta Q(x,a) = \lambda^t \prod_{1 \le s \le t} \pi(a_s|x_s)\delta_t$$

Reweight the traces by the product of target probabilities

Tree backup TB(λ) algorithm

[Precup, Sutton, Singh, 2000]

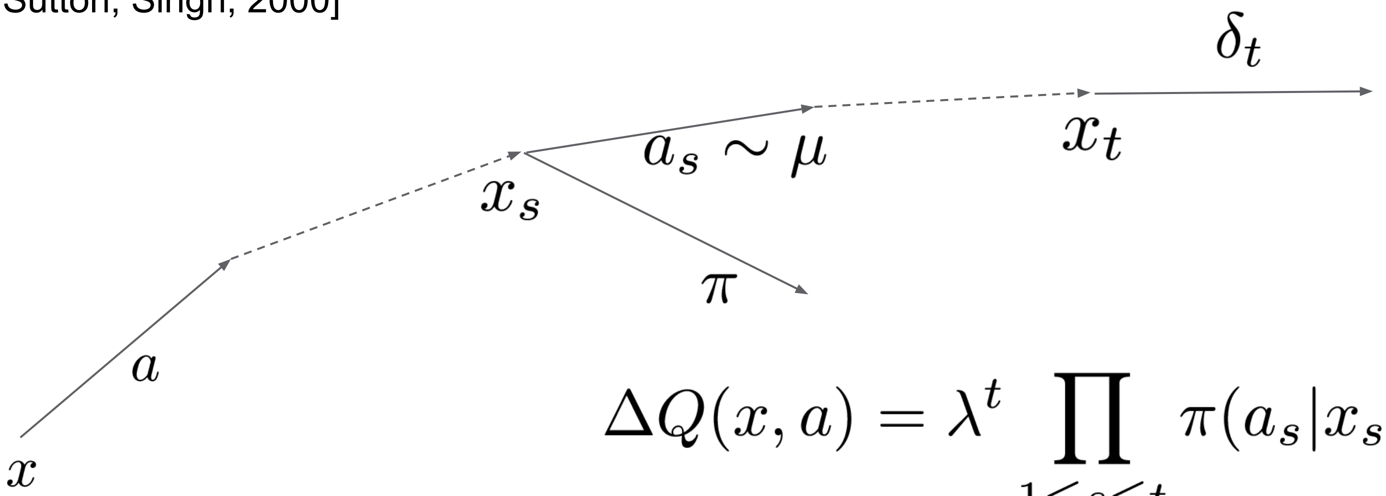


$$\Delta Q(x,a) = \lambda^t \prod_{1 \le s \le t} \pi(a_s|x_s)\delta_t$$

works for arbitrary policies π and μ

Tree backup TB(λ) algorithm

[Precup, Sutton, Singh, 2000]



$$\Delta Q(x,a) = \lambda^t \prod_{1 \le s \le t} \pi(a_s|x_s)\delta_t$$

works for arbitrary policies π and μ

cut traces unnecessarily when on-policy **Not efficient!**

General off-policy return-based algorithm:

$$\Delta Q(x,a) = \sum_{t\geq 0} \gamma^t \Big(\prod_{1\leq s\leq t} c_s \Big) \Big(\underbrace{r_t + \gamma \mathbb{E}_\pi Q(x_{t+1}, \cdot) - Q(x_t, a_t)}_{\delta_t} \Big)$$

Algorithm:	Trace coefficient:	Problem:
IS	$c_s = \frac{\pi(a_s\|x_s)}{\mu(a_s\|x_s)}$	high variance
$Q^\pi(\lambda)$	$c_s = \lambda$	No guarantee
$TB(\lambda)$	$c_s = \lambda\pi(a_s\|x_s)$	not efficient

Off-policy policy evaluation:

Theorem 1: Assume finite state space. Generate trajectories according to behavior policy μ . Update all states along trajectories according to:

$$Q_{k+1}(x,a) = Q_k(x,a) + \alpha_k \sum_{t\geq 0} \gamma^t (c_1 \dots c_t)\big(r_t + \gamma \mathbb{E}_\pi Q_k(x_{t+1},\cdot) - Q_k(x_t,a_t)\big)$$

Assume all states visited infinitely often. Under usual SA assumptions,

$$\boxed{\text{If} \quad 0 \leq c_s \leq \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \quad \text{then} \quad Q_k \rightarrow Q^\pi \text{ a.s.}}$$

Sufficient conditions for a **safe** algorithm (works for any μ and π)

Off-policy return-based operator

Lemma:
Assume the traces satisfy $0 \le c_s \le \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}$.

Then the **off-policy return-based operator**:

$$\mathcal{R}Q(x,a) = Q(x,a) + \mathbb{E}_\mu\Big[\sum_{t\ge 0}\gamma^t(c_1\dots c_t)\big(r_t + \gamma\mathbb{E}_\pi Q(x_{t+1},\cdot) - Q(x_t,a_t)\big)\Big]$$

is a contraction mapping (whatever μ and π) and Q^π is its fixed point.

Proof [part 1]

$$
\begin{aligned}
\mathcal{R}Q(x,a) &= Q(x,a) + \mathbb{E}_\mu\Big[\sum_{t\geq 0}\gamma^t(c_1\dots c_t)\big(r_t + \gamma\mathbb{E}_\pi Q(x_{t+1},\cdot) - Q(x_t,a_t)\big)\Big] \\
&= \mathbb{E}_\mu\Big[\sum_{t\geq 0}\gamma^t(c_1\dots c_t)\big(r_t + \gamma\big[\mathbb{E}_\pi Q(x_{t+1},\cdot) - c_{t+1}Q(x_{t+1},a_{t+1})\big]\big)\Big]
\end{aligned}
$$

Thus

$$
\begin{aligned}
(\mathcal{R}Q_1 - \mathcal{R}Q_2)(x,a) &= \mathbb{E}_\mu\Big[\sum_{t\geq 0}\gamma^{t+1}(c_1\dots c_t)\Big(\mathbb{E}_\pi(Q_1-Q_2)(x_{t+1},\cdot) - c_{t+1}(Q_1-Q_2)(x_{t+1},a_{t+1})\Big)\Big] \\
&= \mathbb{E}_\mu\Big[\sum_{t\geq 0}\gamma^{t+1}(c_1\dots c_t)\sum_a\big(\pi(a|x_{t+1}) - \mu(a|x_{t+1})c_{t+1}(a)\big)(Q_1-Q_2)(x_{t+1},a)\Big]
\end{aligned}
$$

which is a linear combination weighted by non-negative coefficients which sum to...

Proof [part 2]

$$
\begin{aligned}
\text{Sum of the coeff.} \quad &= \mathbb{E}_\mu\Big[\sum_{t\geq 0}\gamma^{t+1}(c_1\dots c_t)\sum_a\big(\pi(a|x_{t+1})-\mu(a|x_{t+1})c_{t+1}(a)\big)\Big] \\
&= \mathbb{E}_\mu\Big[\sum_{t\geq 0}\gamma^{t+1}(c_1\dots c_t)(1-c_{t+1})\Big] \\
&= \gamma-(1-\gamma)\mathbb{E}_\mu\Big[\sum_{t\geq 1}\gamma^{t}(c_1\dots c_t)\Big] \\
&\in [0,\gamma]
\end{aligned}
$$

Thus $\|\mathcal{R}Q_1-\mathcal{R}Q_2\|_\infty \leq \gamma\|Q_1-Q_2\|_\infty$

Tradeoff for trace coefficients c_s

- **Contraction coefficient of the expected operator**

$$\eta := \gamma - (1-\gamma)\mathbb{E}_\mu\Big[\sum_{t\geq 1}\gamma^t(c_1\cdots c_t)\Big] \in [0,\gamma]$$

$\eta = \gamma$ when $c_s = 0$ (one-step Bellman update)
$\eta = 0$ when $c_s = 1$ (full Monte-Carlo rollouts)

- **Variance of the estimate** (can be infinite for $c_s = \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}$)

Large c_s uses multi-steps returns, but large variance
Small c_s low variance, but do not use multi-steps returns

Retrace(λ)

[Munos et al., 2016]

Our recommendation: $\boxed{c_s = \lambda \min\left(1, \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}\right)}$

Properties:

- Low variance since $c_s \leq 1$
- Safe (off policy): cut the traces when needed $c_s \in \left[0, \frac{\pi(a_s|x_s)}{\mu(a_a|x_s)}\right]$
- Efficient (on policy): but only when needed. Note that $c_s \geq \lambda\pi(a_s|x_s)$

Summary

Algorithm:	Trace coefficient:	Problem:
IS	$c_s = \frac{\pi(a_s\|x_s)}{\mu(a_s\|x_s)}$	high variance
$Q^\pi(\lambda)$	$c_s = \lambda$	not safe (off-policy)
$TB(\lambda)$	$c_s = \lambda\pi(a_s\|x_s)$	not efficient (on-policy)
$Retrace(\lambda)$	$c_s = \lambda \min\left(1, \frac{\pi(a_s\|x_s)}{\mu(a_s\|x_s)}\right)$	none!

Retrace(λ) for optimal control

Let (μ_k) and (π_k) sequences of behavior and target policies and

$$Q_{k+1}(x,a) = Q_k(x,a) + \alpha_k \sum_{t\geq 0} (\lambda\gamma)^t \prod_{1\leq s\leq t} \min\left(1, \frac{\pi_k(a_s|x_s)}{\mu_k(a_s|x_s)}\right)\left(r_t + \gamma \mathbb{E}_\pi Q_k(x_{t+1},\cdot) - Q_k(x_t,a_t)\right)$$

Theorem 2

Under previous assumptions (+ a technical assumption)

Assume (π_k) are "increasingly greedy" wrt (Q_k)

Then, a.s.,

$$\boxed{Q_k \rightarrow Q^*}$$

Remarks

- If (π_k) are greedy policies, then $c_s = \lambda \mathbb{I}\{a_s \in \arg\max_a Q_k(x_s, a)\}$
  $\rightarrow$ **Convergence of Watkin's Q(λ)** to Q^*
  (open problem since 1989)

- "Increasingly greedy" allows for smoother traces thus faster convergence

- The behavior policies (μ_k) do **not** need to become greedy wrt (Q_k)
  $\rightarrow$ **no GLIE assumption** (Greedy in the limit with infinite exploration)
  (first return-based algo converging to Q^* without GLIE)

Theoretical guarantees of Retrace

Under assumption of finite-state space:

- Convergence to optimal policy
- Cut traces when -and only when- needed
- Adjust the length of the backup to the “off-policy-ness” of the data

Should be useful in deep RL since it allows memory-replay, exploration, distributed acting, and learn from sequences.

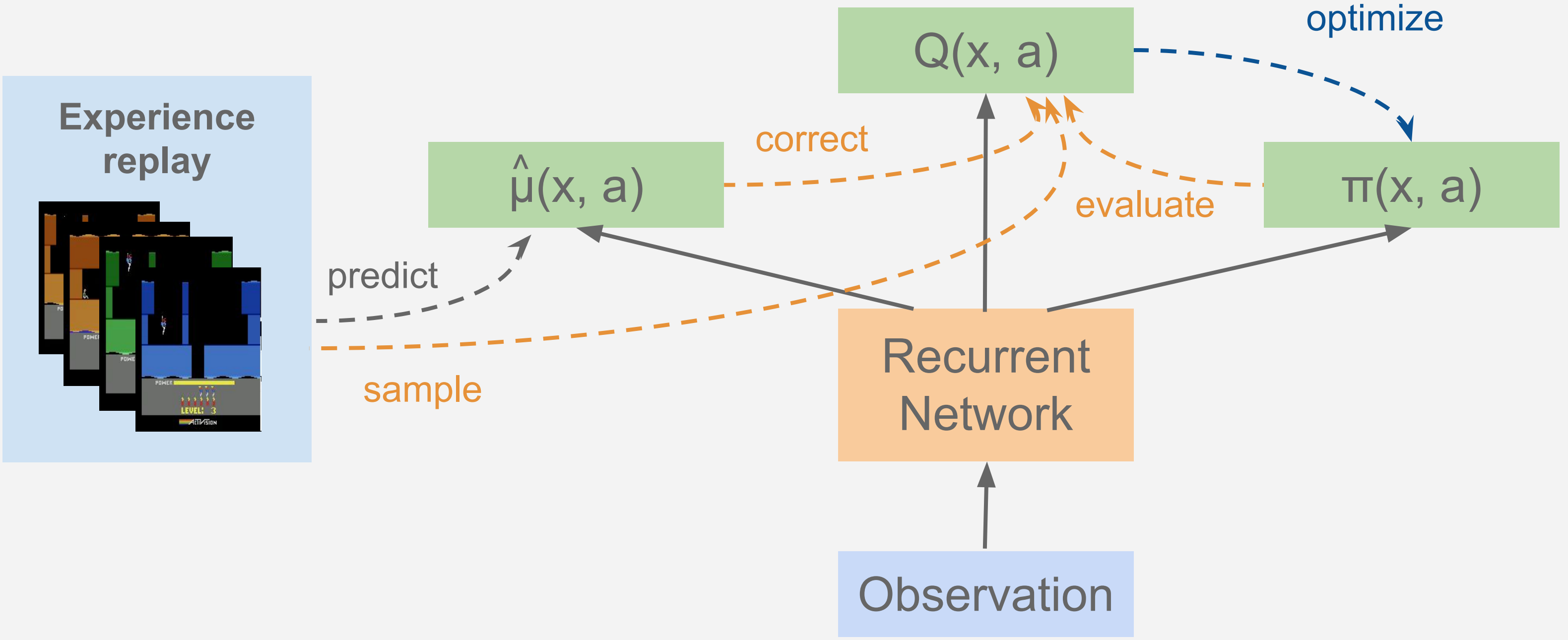
Now, does in work in practice?

Retrace for deepRL

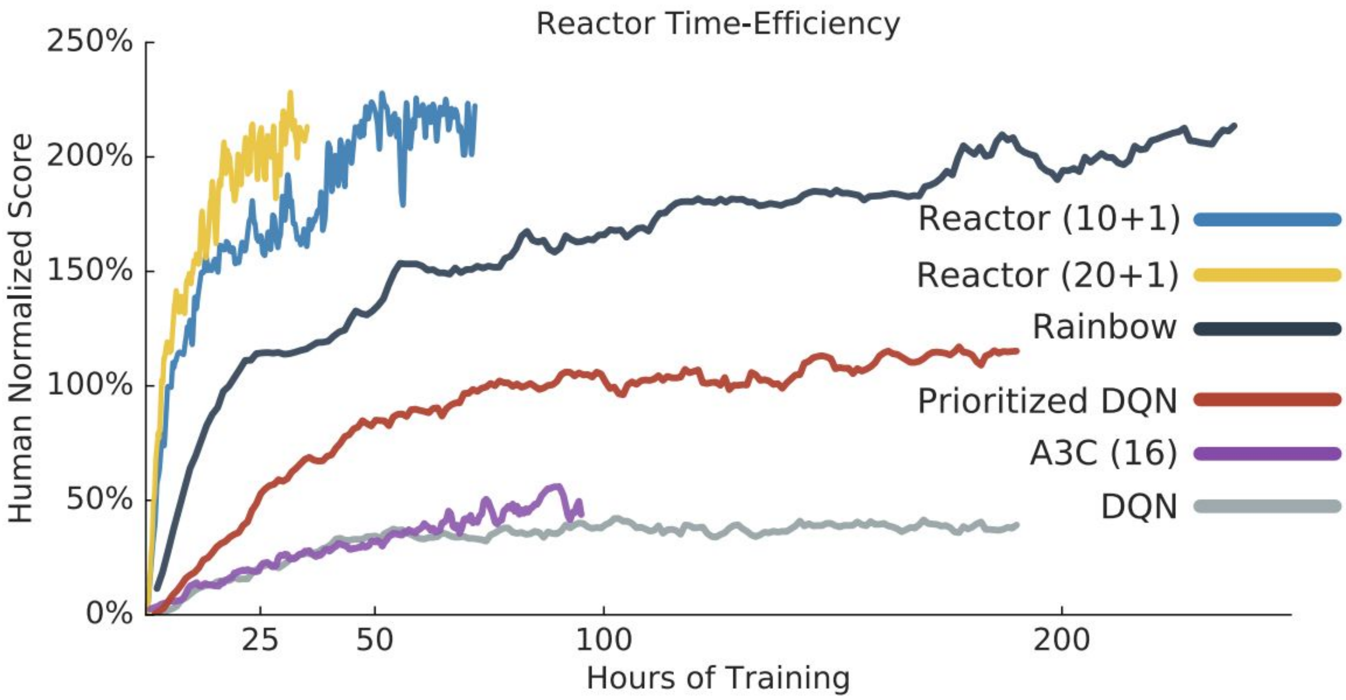
Several actor-critic architectures at DeepMind:

- **ACER** (Actor-Critic for Experience Replay) [Wang et al., 2017]. Policy gradient. Works for continuous actions.
- **Reactor** (Retrace-actor) [Gruesly et al., 2018]. Use beta-LOO to update policy. Use LSTM.
- **MPO** (Maximum a posteriori Policy Optimization) [Abdolmaleki et al., 2018] Soft (KL-regularized) policy improvement.
- **IMPALA** (IMPortance Weighted Actor-Learner Architecture) [Espeholt et al., 2018]. Heavily distributed agent. Uses V-trace.

Reactor [Gruesly et al., 2018]



Reactor performances on Atari



Reactor performances on Atari

Algorithm	Normalized scores	Mean rank	Elo
Random	0.00	11.65	-563
Human	1.00	6.82	0
DQN	0.69	9.05	-172
DDQN	1.11	7.63	-58
Duel	1.17	6.35	32
Prior	1.13	6.63	13
Prior. Duel.	1.15	6.25	40
A3C LSTM	1.13	6.30	37
Rainbow	1.53	4.18	186
Reactor ND [5]	1.51	4.98	126
Reactor	1.65	4.58	156
Reactor 500M	**1.82**	**3.65**	**227**

Table 1: Random human starts

Algorithm	Normalized scores	Mean rank	Elo
Random	0.00	10.93	-673
Human	1.00	6.89	0
DQN	0.79	8.65	-167
DDQN	1.18	7.28	-27
Duel	1.51	5.19	143
Prior	1.24	6.11	70
Prior. Duel.	1.72	5.44	126
ACER[6] 500M	1.9	-	-
Rainbow	**2.31**	3.63	270
Reactor ND [5]	1.80	4.53	195
Reactor	1.87	4.46	196
Reactor 500M	2.30	**3.47**	**280**

Table 2: 30 random no-op starts.

Control suite with MPO

MPO (Maximum a posteriori Policy Optimization)

[Abdolmaleki et al., 2018]

on the DeepMind control suite

(set of continuous control tasks intended to serve as performance benchmarks for RL agents)

See: https://www.youtube.com/watch?v=he_BPw32PwU

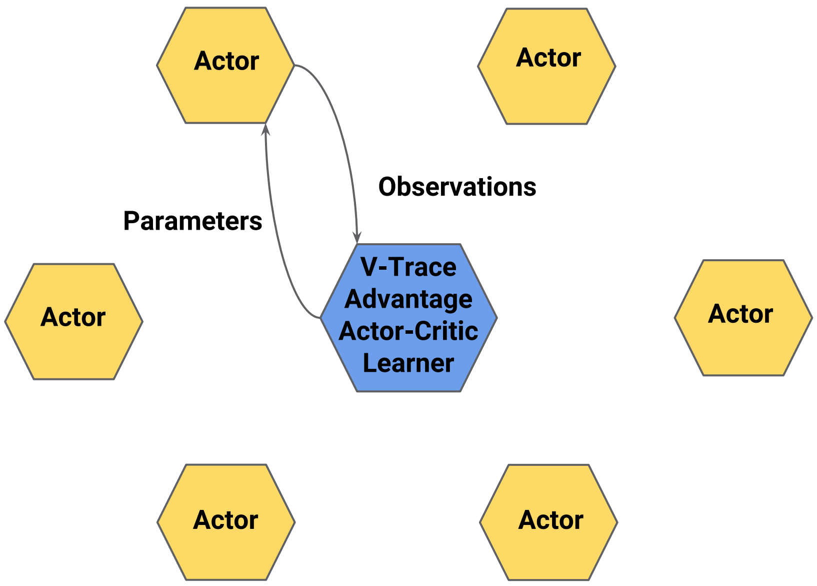
IMPALA [Espeholt et al., 2018]

IMPortance Weighted **A**ctor-**L**earner **A**rchitecture

- Heavily distributed architecture
- Many actors (CPU),
- one (or more) learner (GPU)

- Actors generate trajectories and place them into a queue.
- Learner dequeues and performs parameter updates.

Stale experience
→ requires off-policy learning: V-trace



V-trace: off-policy algorithm using V-values

V-trace = Modified version of Retrace where we learn V instead of Q

- The V-Trace corrected estimate for the value $V(x_s)$ is:

$$v_s \stackrel{\text{def}}{=} V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left(\prod_{i=s}^{t-1} c_i \right) \underbrace{\rho_t \left(r_t + \gamma V(x_{t+1}) - V(x_t) \right)}_{\delta_t V}$$

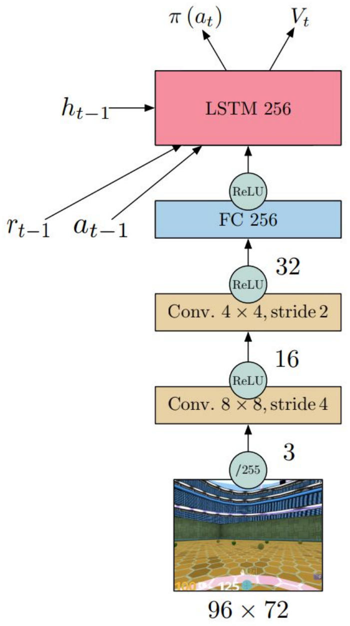
- where $\rho_i \stackrel{\text{def}}{=} \min\left(\bar{\rho}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}\right)$ and $c_i \stackrel{\text{def}}{=} \min\left(\bar{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}\right)$.

- Converges to $\dfrac{\min\left(\bar{\rho}\mu(a|x), \pi(a|x)\right)}{\sum_b \min\left(\bar{\rho}\mu(b|x), \pi(b|x)\right)}$

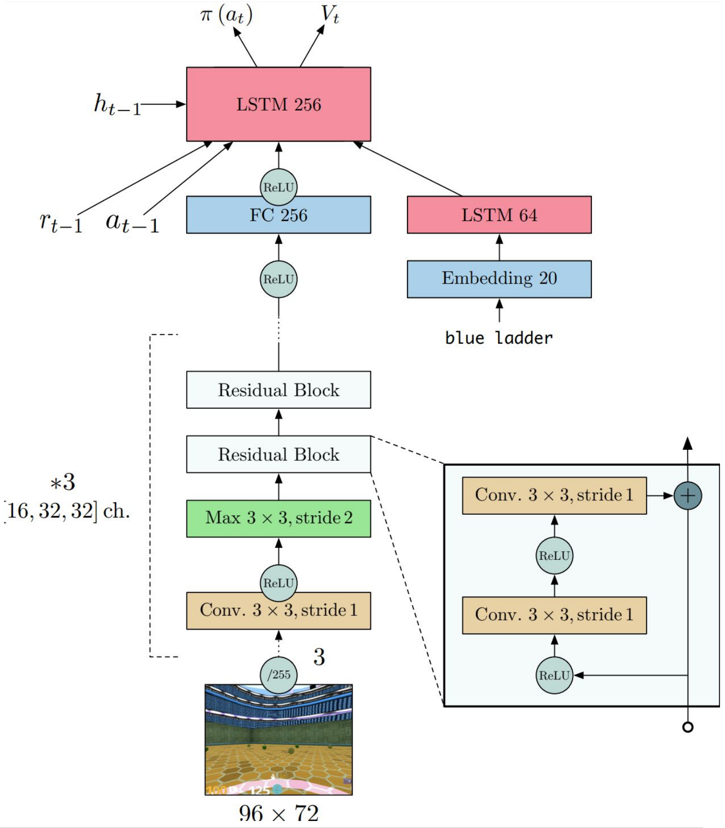
- The V-Trace update for the value function is: $(v_s - V(x_s))\nabla V(x_s)$
- The V-Trace update for the policy is: $\rho_s \nabla \log \pi(a_s|x_s)(r_s + \gamma v_{s+1} - V(x_s))$

Impala Architectures

Small CNN-LSTM



Deep ResNet CNN-LSTM

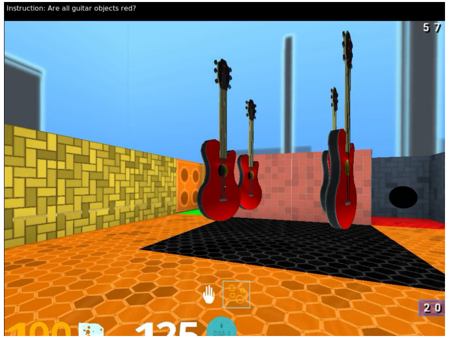


DMLab-30 Task Set

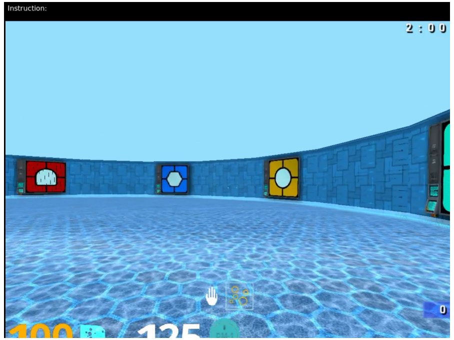
DeepMind Lab

- Set of 30 cognitive tasks in DeepMind Lab 3D environment.
- Many of the tasks are procedurally generated.

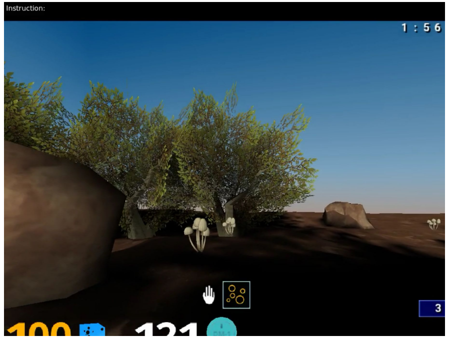
Grounded Language



Memory



Outdoor Foraging

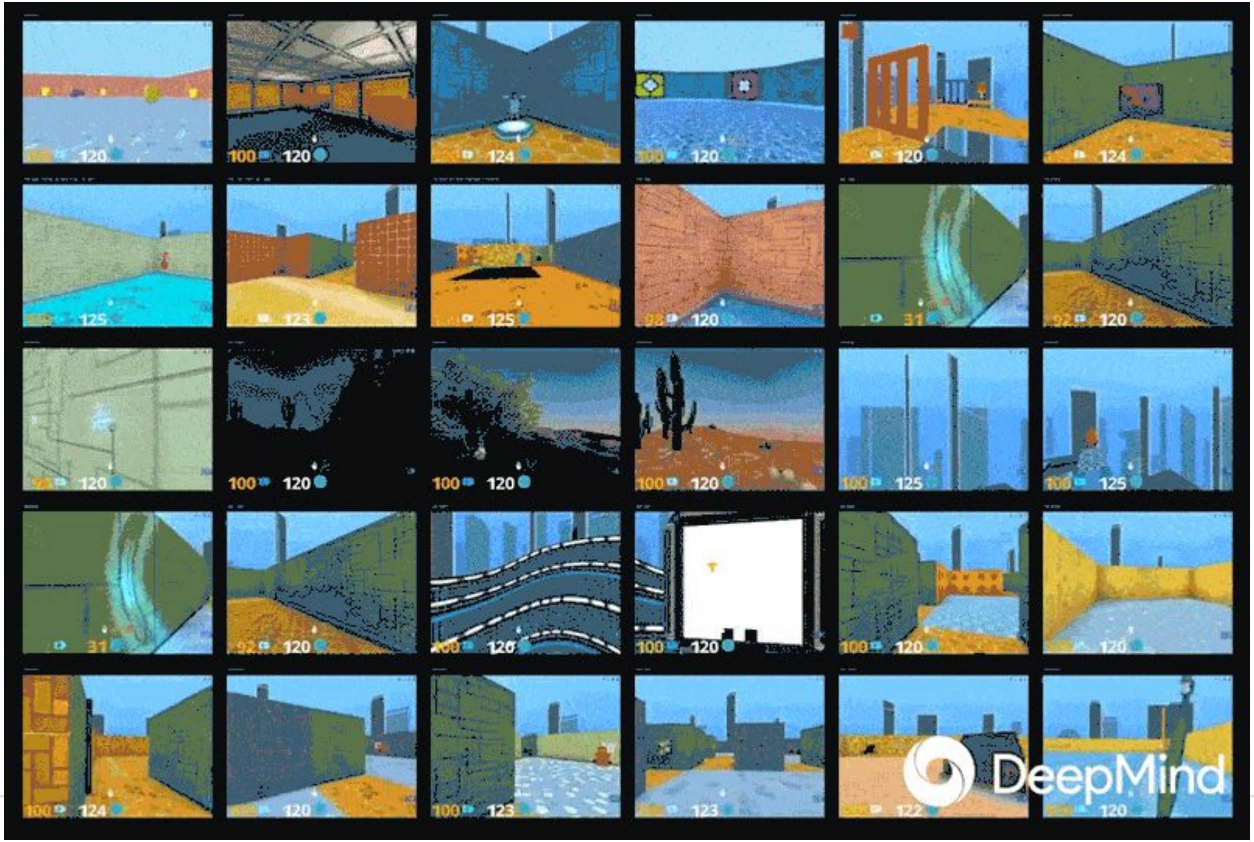


Navigation

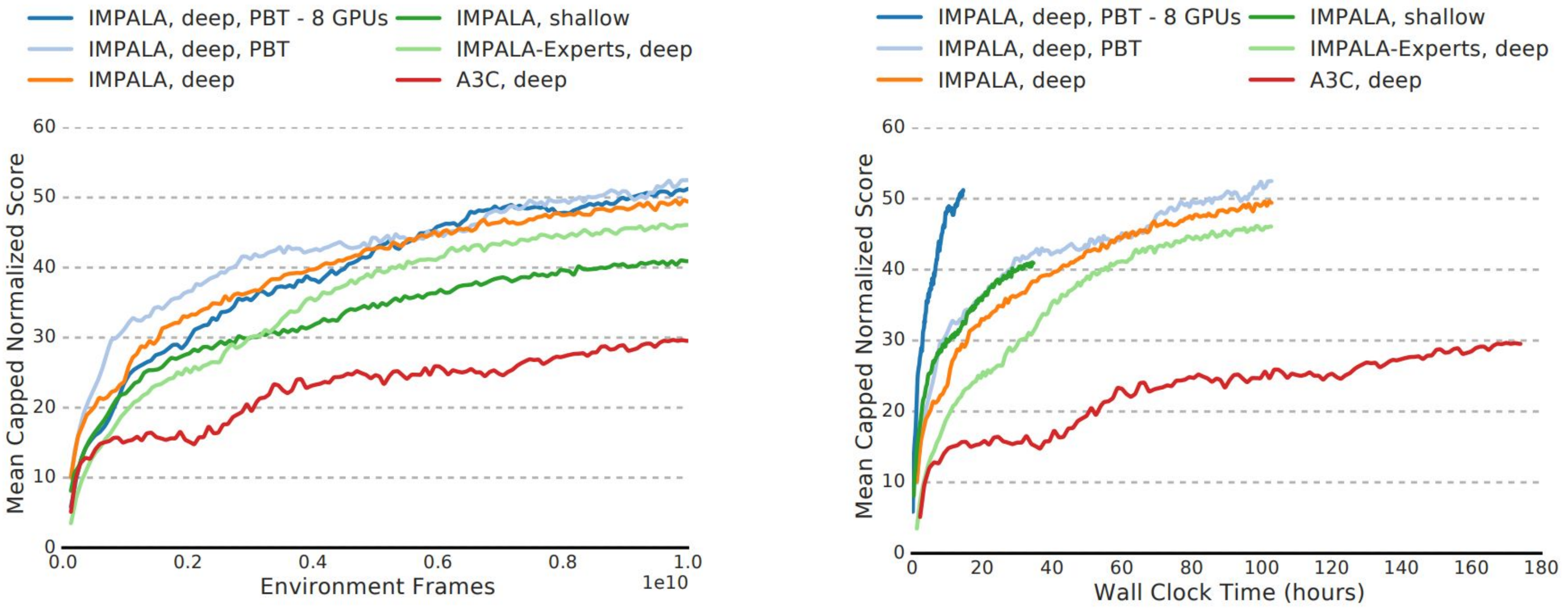
DMLab-30 Task Set

Individual task:
An agent trained per task

Multi-tasks:
A single agent trained on all tasks simultaneously



Performance of Impala on DMLab-30



- IMPALA outperforms A3C (10x more data efficient, 2x overall final performance)
- **Positive transfer** in multi-task training

IMPALA Videos - Mushroom foraging task

See: https://github.com/deepmind/lab/tree/master/game_scripts/levels/contributed/dmlab30

Mushroom foraging task. The agent must collect mushrooms within a naturalistic terrain environment to maximise score. The mushrooms do not regrow. The map is randomly generated. The spawn location is randomized for each episode. **Foraging task**

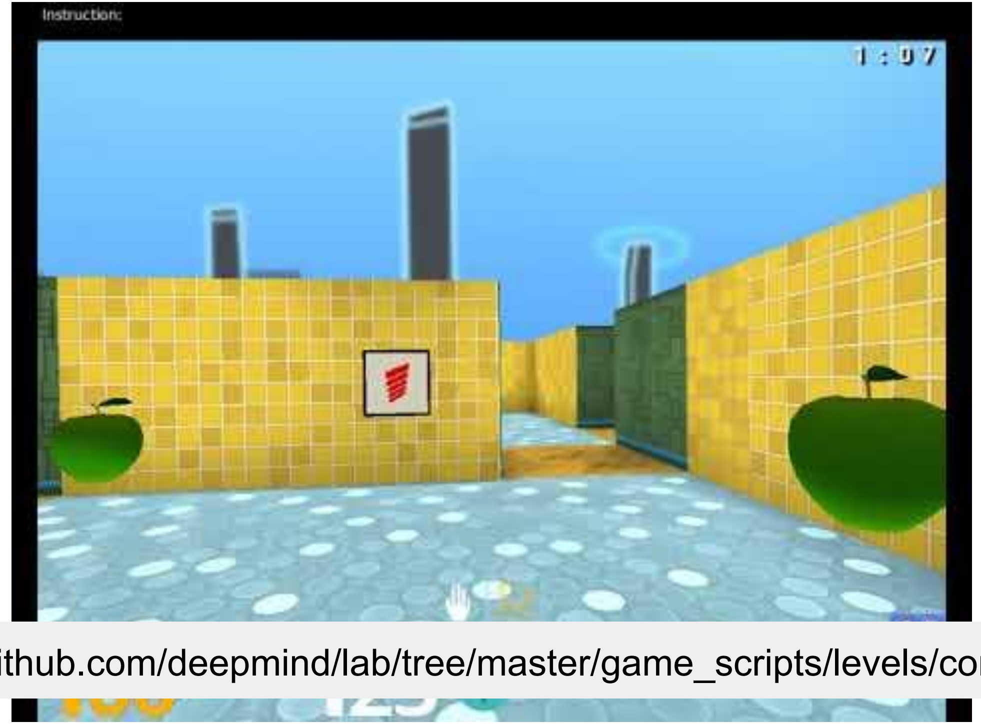
IMPALA Videos - Select Located Object task

See: https://github.com/deepmind/lab/tree/master/game_scripts/levels/contributed/dmlab30

The agent is asked to collect a specified coloured object in a specified coloured room. Example: “Pick the red object in the blue room.” **Language task**

DeepMind

IMPALA Videos - Object Locations



See: https://github.com/deepmind/lab/tree/master/game_scripts/levels/contributed/dmlab30

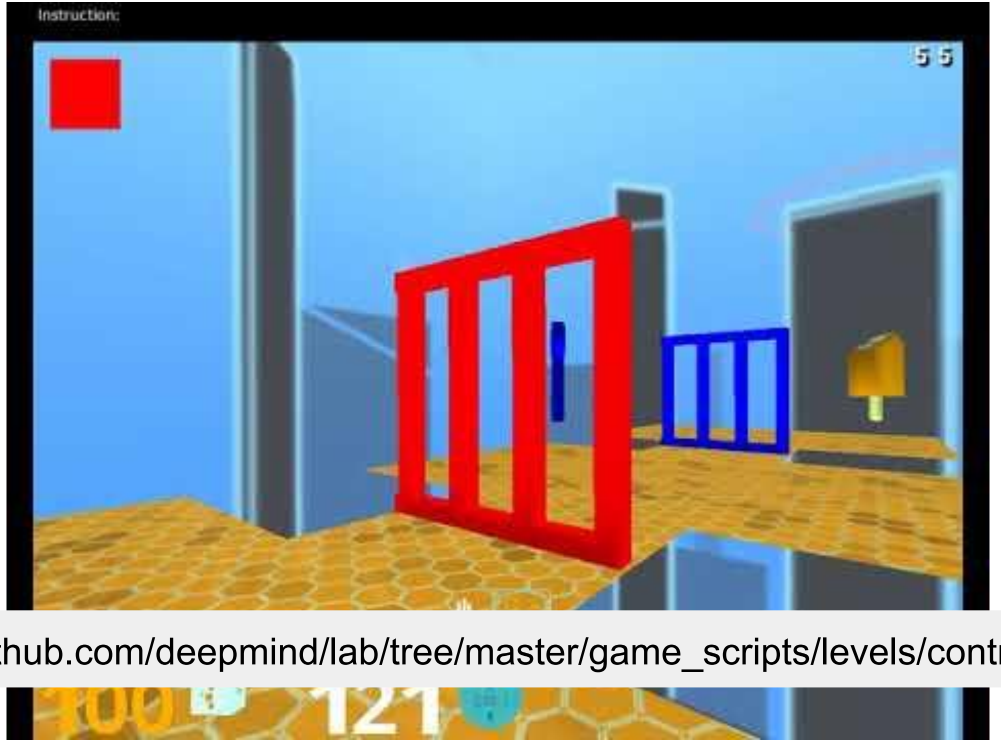
The agent must collect as many apples as possible before the episode ends to maximise their score. Upon collecting all of the apples, the level will reset, repeating until the episode ends. Apple locations, level layout and theme are randomized per episode. **Navigation task**

IMPALA Videos - Obstructed Goals

See: https://github.com/deepmind/lab/tree/master/game_scripts/levels/contributed/dmlab30

Agents are required to find the goal as fast as possible, but now with randomly opened and closed doors. **Navigation task**

IMPALA Videos - Keys Doors Puzzle



See: https://github.com/deepmind/lab/tree/master/game_scripts/levels/contributed/dmlab30

A procedural planning puzzle. The agent must reach the goal object, located in a position that is blocked by a series of coloured doors. Coloured keys can be used to open matching doors once. Collecting keys in the wrong order can make the goal unreachable. **Requires planning**

IMPALA Videos - Select Non-matching Object

See: https://github.com/deepmind/lab/tree/master/game_scripts/levels/contributed/dmlab30

The agent must choose an object that is different from the one it has seen before. The agent is placed into a first room containing an object and a teleport pad. Touching the pad teleports the agent to a second room containing two objects, one of which matches the object in the previous room. **Requires memory**

IMPALA Videos - Watermaze

See: https://github.com/deepmind/lab/tree/master/game_scripts/levels/contributed/dmlab30

The agent must find a hidden platform which, when found, generates a reward. This is difficult to find the first time, but in subsequent trials the agent should try to remember where it is and go straight back to this place. Tests episodic memory and navigation ability. **Requires episodic memory**

We need more fundamental research in deep RL!

DeepRL is a super exciting research topic.

We need new idea, new algorithms, new theories!

Please join the fun!

References (used in the slides):

- [Pontryagin, 1956] See: *Optimal Processes of Regulation,* 1960 for English version.
- [Bellman, 1957] *Dynamic Programming*
- [Sutton, 1988] *Learning to predict by the methods of temporal differences*
- [Watkins, 1989] *Learning From Delayed Rewards*
- [Precup, Sutton, Singh, 2000] *Eligibility traces for off-policy policy evaluation*
- [Mnih et al., 2015] *Human Level Control Through Deep Reinforcement Learning*
- [Mnih et al., 2016] *Asynchronous Methods for Deep Reinforcement Learning*
- [Mahmood, Yu, White, Sutton, 2015] *Emphatic Temporal-Difference Learning*
- [Harutyunyan et al., 2016] *Q(λ) with Off-Policy Corrections*
- [Munos et al., 2016] *Safe and Efficient Off-Policy Reinforcement Learning*
- [Wang et al., 2017] Sample Efficient Actor-Critic with Experience Replay
- [Gruesly et al., 2018] The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning
- [Abdolmaleki et al., 2018] Maximum a Posteriori Policy Optimization
- [Espeholt et al., 2018] IMPortance Weighted Actor-Learner Architecture