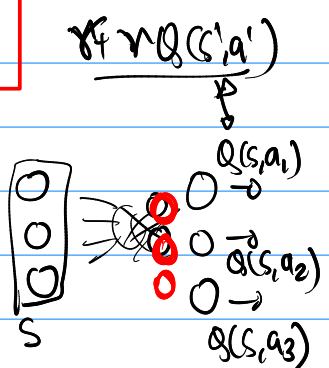
Real-time reference notes / Lecture 01

(Repeated for ready reference)

1. RL basics: States, actions, rewards, returns
2. Intuition behind value-based methods, tabular version
3. Filling up the Q-table using ϵ-greedy algorithms
4. Scalability of tabular RL
   - (a) Size of table
   - (b) Continuous states & actions
   - (c) Visiting each (s,a) pair
5. Approximation of $Q(s,a)$ using neural networks
   - (a) $s, a \rightarrow Q(s,a)$
   - (b) $s \rightarrow Q(s,a_1), Q(s,a_2), Q(s,a_3)$
6. Terminology
   - (a) Model-free & Model-based
   - (b) Online & offline — access to env → (s_t, a_t, r_t, s_{t+1})
   - (c) On-policy & off-policy
7. Bellman equation
   - ϵ-greedy using Bellman equation



8. MKS paper (ATARI)
9. Issues with basic DQN
   - Bootstrapping
   - stabilisation
   - Last layer discrimination
   - sampling memory

$$Q(s,a) = r + \gamma Q(s',a') \qquad (s,a,r,s')$$

Lecture notes, part 2 : Stabilisation methods

① Bootstrapping problem, $Q^{\theta}(s,a) = r + \gamma Q^{\bar{\theta}}(s',a')$

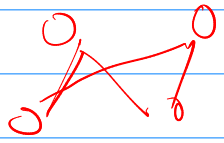
$Q^{\theta}(s,a)$ → produced by the same network?

$Q^{\bar{\theta}}(s',a')$ → use target network (also used in the 2015 DQN paper/code)

② correlated samples problem → on policy
→ off policy
↓
prioritised experience replay
hindsight experience replay

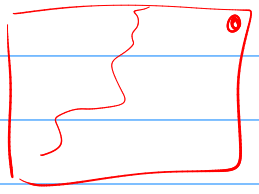
1 million samples (s,a,r,s')

③ Rainbow → use everything you have in terms of tools



Exploration-exploitation dilemma

Return to ε-greedy → problems with long episodic tasks with causal linkages



Solutions
ⓐ Directed exploration
ⓑ Frontier states
ⓒ curiosity } similar but curiosity = short term
ⓓ novelty }
ⓔ option critics → requires domain knowledge

Simplest form of novelty: exploration bonus

Problem: Even good (s,a) pairs ignored once visited several times

Better exploration bonuses: Never Give Up, Agent 57

Reshape step reward to $r_t = r_t^e + \beta r_t^i$

r_t^e: original (extrinsic)

r_t^i: new (intrinsic)

Non-curiosity based possibilities: Action persistence

ε-greedy, DEZ-greedy

Open problem at this stage:

Value based methods are challenging when outputs are continuous

↓

move to policy gradient

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

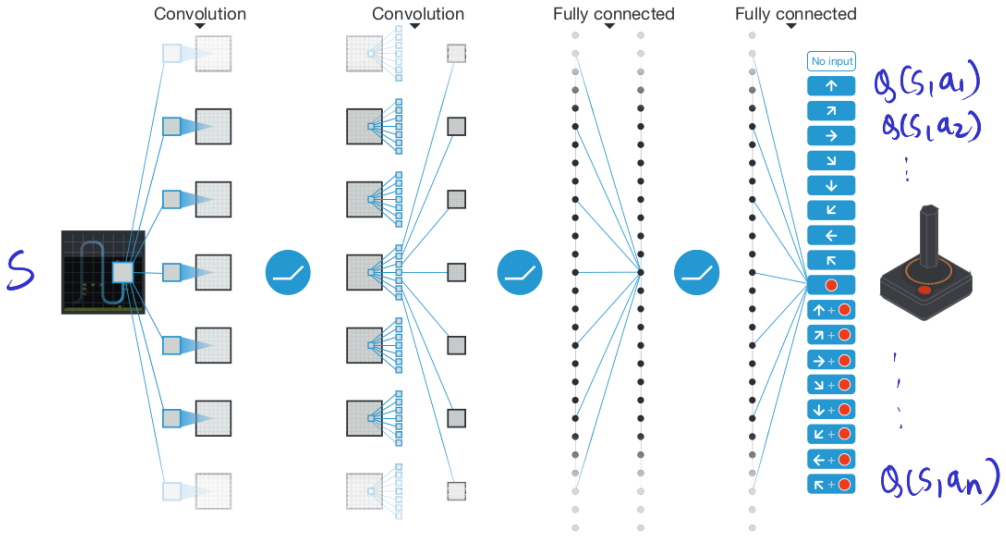
$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s,\ a_t = a,\ \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a) (see Methods)[19].

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function[20]. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values $r + \gamma \max_{a'} Q(s', a')$. We address these instabilities with a novel variant of Q-learning, which uses two key ideas. First, we used a biologically inspired mechanism termed experience replay[21–23] that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution (see below for details). Second, we used an iterative update that adjusts the action-values (Q) towards target values that are only periodically updated, thereby reducing correlations with the target.

↳ online & target networks

While other stable methods exist for training neural networks in the



PRIORITIZED EXPERIENCE REPLAY

Tom Schaul, John Quan, Ioannis Antonoglou and David Silver
Google DeepMind
{schaul,johnquan,ioannisa,davidsilver}@google.com

(s,a,r,s',Δ)

(s,a,r1,s1')
(s,a,r2,s2')

1 million samples

ABSTRACT

Experience replay lets online reinforcement learning agents remember and reuse experiences from the past. In prior work, experience transitions were uniformly sampled from a replay memory. However, this approach simply replays transitions at the same frequency that they were originally experienced, regardless of their significance. In this paper we develop a framework for prioritizing experience, so as to replay important transitions more frequently, and therefore learn more efficiently. We use prioritized experience replay in Deep Q-Networks (DQN), a reinforcement learning algorithm that achieved human-level performance across many Atari games. DQN with prioritized experience replay achieves a new state-of-the-art, outperforming DQN with uniform replay on 41 out of 49 games.

3.2 PRIORITIZING WITH TD-ERROR

The central component of prioritized replay is the criterion by which the importance of each transition is measured. One idealised criterion would be the amount the RL agent can learn from a transition in its current state (expected learning progress). While this measure is not directly accessible, a reasonable proxy is the magnitude of a transition's TD error δ, which indicates how 'surprising' or unexpected the transition is: specifically, how far the value is from its next-step bootstrap estimate (Andre et al., 1998). This is particularly suitable for incremental, online RL algorithms, such as SARSA or Q-learning, that already compute the TD-error and update the parameters in proportion to δ. The TD-error can be a poor estimate in some circumstances as well, e.g. when rewards are noisy; see Appendix A for a discussion of alternatives.

To overcome these issues, we introduce a stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling. We ensure that the probability of being sampled is monotonic in a transition's priority, while guaranteeing a non-zero probability even for the lowest-priority transition. Concretely, we define the probability of sampling transition i as

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{1}$$

where $p_i > 0$ is the priority of transition i. The exponent α determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.

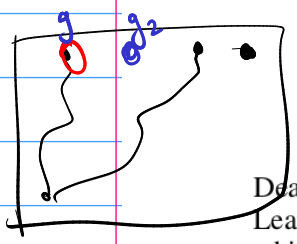
The first variant we consider is the direct, proportional prioritization where $p_i = |\delta_i| + \epsilon$, where ϵ is a small positive constant that prevents the edge-case of transitions not being revisited once their error is zero. The second variant is an indirect, rank-based prioritization where $p_i = \frac{1}{\text{rank}(i)}$, where $\text{rank}(i)$ is the rank of transition i when the replay memory is sorted according to $|\delta_i|$. In this case, P becomes a power-law distribution with exponent α. Both distributions are monotonic in $|\delta|$, but the latter is likely to be more robust, as it is insensitive to outliers. Both variants of stochastic prioritization lead to large speed-ups over the uniform baseline on the Cliffwalk task, as shown on Figure 2 (right).

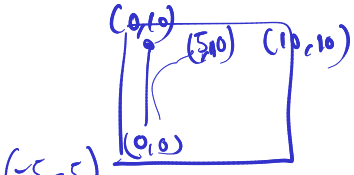
Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

Hindsight Experience Replay

Marcin Andrychowicz*, **Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel†, Wojciech Zaremba†**
OpenAI





Abstract

Dealing with sparse rewards is one of the biggest challenges in Reinforcement Learning (RL). We present a novel technique called *Hindsight Experience Replay* which allows sample-efficient learning from rewards which are sparse and binary and therefore avoid the need for complicated reward engineering. It can be combined with an arbitrary off-policy RL algorithm and may be seen as a form of implicit curriculum.

; Beach, CA, USA.

3.3 Algorithm

The idea behind Hindsight Experience Replay (HER) is very simple: after experiencing some episode $s_0, s_1, \ldots, s_T$ we store in the replay buffer every transition $s_t \rightarrow s_{t+1}$ not only with the original goal used for this episode but also with a subset of other goals. Notice that the goal being pursued influences the agent's actions but not the environment dynamics and therefore we can replay each trajectory with an arbitrary goal assuming that we use an off-policy RL algorithm like DQN (Mnih et al., 2015), DDPG (Lillicrap et al., 2015), NAF (Gu et al., 2016) or SDQN (Metz et al., 2017).

One choice which has to be made in order to use HER is the set of additional goals used for replay. In the simplest version of our algorithm we replay each trajectory with the goal $m(s_T)$, i.e. the goal which is achieved in the final state of the episode. We experimentally compare different types and quantities of additional goals for replay in Sec. 4.5. In all cases we also replay each trajectory with the original goal pursued in the episode. See Alg. 1 for a more formal description of the algorithm.

HER may be seen as a form of implicit curriculum as the goals used for replay naturally shift from ones which are simple to achieve even by a random agent to more difficult ones. However, in contrast to explicit curriculum, HER does not require having any control over the distribution of initial environment states. Not only does HER learn with extremely sparse rewards, in our experiments it also performs better with sparse rewards than with shaped ones (See Sec. 4.4). These results are indicative of the practical challenges with reward shaping, and that shaped rewards would often constitute a compromise on the metric we truly care about (such as binary success/failure).

Algorithm 1 Hindsight Experience Replay (HER)

Given:

- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

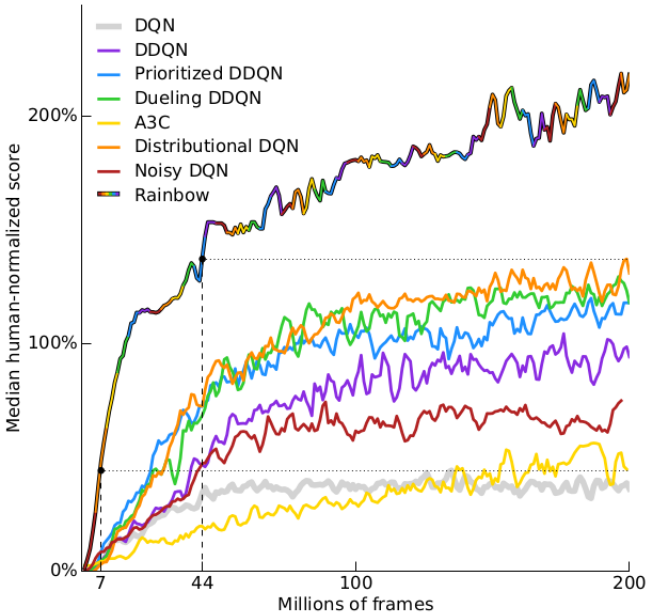
Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks
Initialize replay buffer R
for episode = $1, M$ **do**
  Sample a goal g and an initial state s_0.
  for $t = 0, T-1$ **do**
    Sample an action a_t using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ $||$ denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  end for
  for $t = 0, T-1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g,\ a_t,\ r_t,\ s_{t+1}||g)$ in R ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}$(**current episode**)
    for $g' \in G$ **do**
      $r' := r(s_t, a_t, g')$
      Store the transition $(s_t||g',\ a_t,\ r',\ s_{t+1}||g')$ in R ▷ HER
    end for
  end for
  for $t = 1, N$ **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ and minibatch B
  end for
end for

Rainbow: Combining Improvements in Deep Reinforcement Learning

Matteo Hessel DeepMind
Joseph Modayil DeepMind
Hado van Hasselt DeepMind
Tom Schaul DeepMind
Georg Ostrovski DeepMind
Will Dabney DeepMind
Dan Horgan DeepMind
Bilal Piot DeepMind
Mohammad Azar DeepMind
David Silver DeepMind

Abstract

The deep reinforcement learning community has made several independent improvements to the DQN algorithm. However, it is unclear which of these extensions are complementary and can be fruitfully combined. This paper examines six extensions to the DQN algorithm and empirically studies their combination. Our experiments show that the combination provides state-of-the-art performance on the Atari 2600 benchmark, both in terms of data efficiency and final performance. We also provide results from a detailed ablation study that shows the contribution of each component to overall performance.



Introduction

The many recent successes in scaling reinforcement learning (RL) to complex sequential decision-making problems were kick-started by the Deep Q-Networks algorithm (DQN; Mnih et al. 2013, 2015). Its combination of Q-learning with convolutional neural networks and experience replay enabled it to learn, from raw pixels, how to play many Atari

Double Q-learning. Conventional Q-learning is affected by an overestimation bias, due to the maximization step in Equation 1, and this can harm learning. Double Q-learning (van Hasselt 2010), addresses this overestimation by decoupling, in the maximization performed for the bootstrap target, the selection of the action from its evaluation. It is possible to effectively combine this with DQN (van Hasselt, Guez, and Silver 2016), using the loss

$$(R_{t+1} + \gamma_{t+1} q_{\overline{\theta}}(S_{t+1}, \operatorname*{argmax}_{a'} q_\theta(S_{t+1}, a')) - q_\theta(S_t, A_t))^2.$$

This change was shown to reduce harmful overestimations that were present for DQN, thereby improving performance.

difference from (1)?

NEVER GIVE UP: LEARNING DIRECTED EXPLORATION STRATEGIES

Adrià Puigdomènech Badia* **Pablo Sprechmann*** **Alex Vitvitskyi** **Daniel Guo**

Bilal Piot **Steven Kapturowski** **Olivier Tieleman** **Martín Arjovsky**

Alexander Pritzel **Andew Bolt** **Charles Blundell**

DeepMind {adriap, psprechmann, avlife, danielguo, piot, skapturowski, tieleman, apritzel, abolt, cblundell}@google.com

1 INTRODUCTION

The problem of exploration remains one of the major challenges in deep reinforcement learning. In general, methods that guarantee finding an optimal policy require the number of visits to each state–action pair to approach infinity. Strategies that become greedy after a finite number of steps may never learn to act optimally; they may converge prematurely to suboptimal policies, and never gather the data they need to learn. Ensuring that all state-action pairs are encountered infinitely often is the general problem of maintaining exploration (François-Lavet et al., 2018; Sutton & Barto, 2018). The simplest approach for tackling this problem is to consider stochastic policies with a non-zero probability of selecting all actions in each state, e.g. ϵ-greedy or Boltzmann exploration. While these techniques will eventually learn the optimal policy in the tabular setting, they are very inefficient and the steps they require grow exponentially with the size of the state space. Despite these shortcomings, they can perform remarkably well in dense reward scenarios (Mnih et al., 2015). In sparse reward settings, however, they can completely fail to learn, as temporally-extended exploration (also called deep exploration) is crucial to even find the very few rewarding states (Osband et al., 2016).

need infinite visits to learn q values with guarantees

purely random actions in each step is not workable

*Equal contribution.

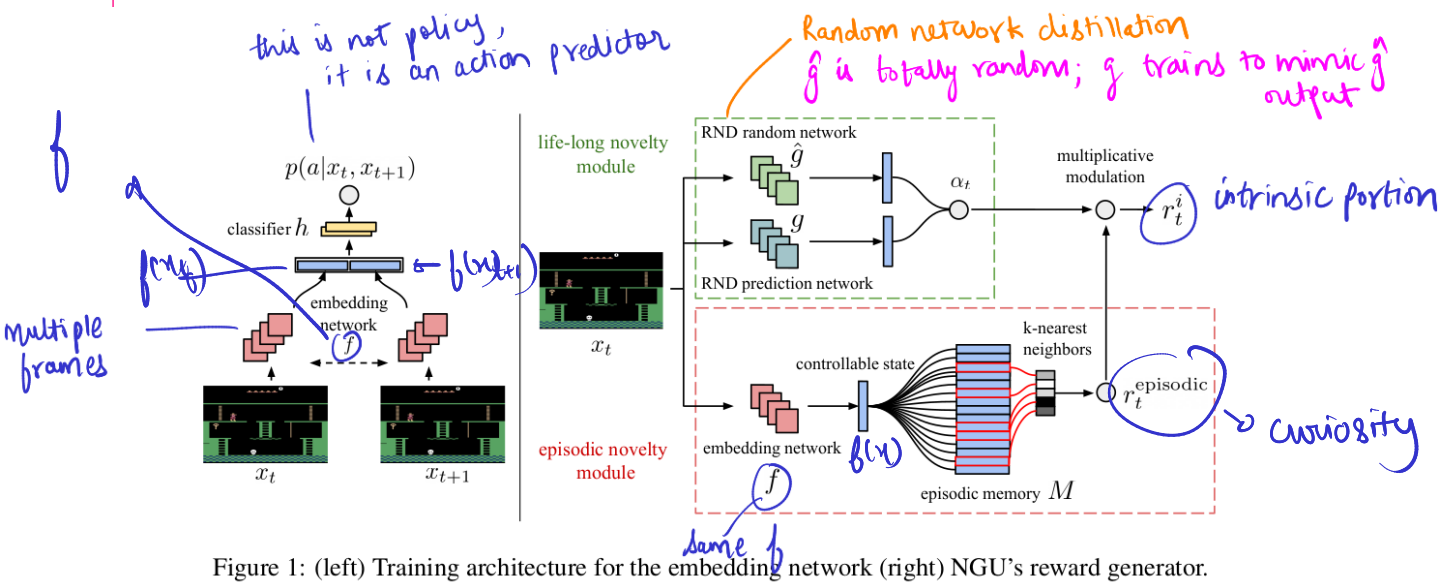


Figure 1: (left) Training architecture for the embedding network (right) NGU's reward generator.

$$r = \left[r_t^e + \beta r_t^i\right] = r_t^{\text{episodic}} \cdot \min\left\{\max\left\{\alpha_t, 1\right\}, L\right\}$$

This is a follow-up of NGU

Agent57: Outperforming the Atari Human Benchmark

Adrià Puigdomènech Badia [*1] **Bilal Piot** [*1] **Steven Kapturowski** [*1] **Pablo Sprechmann** [*1] **Alex Vitvitskyi** [1]
Daniel Guo [1] **Charles Blundell** [1]

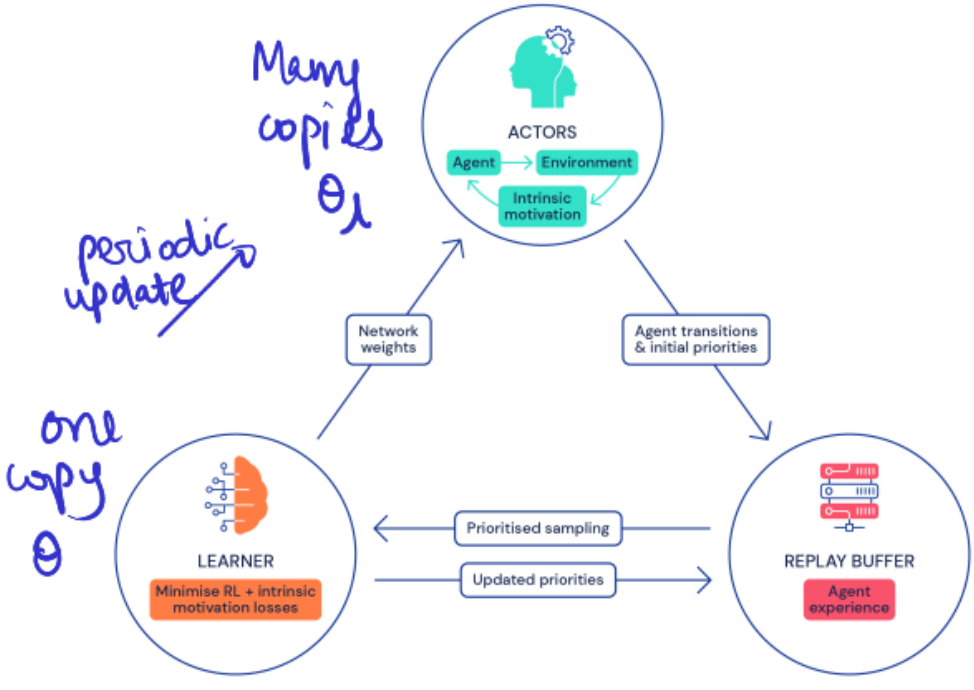


Figure 2. A schematic depiction of a distributed deep RL agent

3. Improvements to NGU

3.1. State-Action Value Function Parameterization

The proposed architectural improvement consists in splitting the state-action value function in the following way:

Recall: NGU had single Q function

$$Q(x, a, j; \theta) = Q(x, a, j; \theta^e) + \beta_j Q(x, a, j; \theta^i),$$

where $Q(x, a, j; \theta^e)$ and $Q(x, a, j; \theta^i)$ are the extrinsic and intrinsic components of $Q(x, a, j; \theta)$ respectively. The sets of weights θ^e and θ^i separately parameterize two neural networks with identical architecture and $\theta = \theta^i \cup \theta^e$. Both $Q(x, a, j; \theta^e)$ and $Q(x, a, j; \theta^i)$ are optimized separately in the learner with rewards r^e and r^i respectively, but with the same target policy $\pi(x) = \arg\max_{a \in \mathcal{A}} Q(x, a, j; \theta)$. More precisely, to train the weights θ^e and θ^i, we use the same sequence of transitions sampled from the replay, but with two different transformed Retrace loss functions (Munos et al., 2016). For $Q(x, a, j; \theta^e)$ we compute an extrinsic transformed Retrace loss on the sequence transitions with rewards r^e and target policy π, whereas for $Q(x, a, j; \theta^i)$ we compute an intrinsic transformed Retrace loss on the same sequence of transitions but with rewards r^i and target policy π. A reminder of how to compute a transformed Retrace loss on a sequence of transitions with rewards r and target policy π is provided in App. C.