"Life shouldn't be a lifetime of waiting."

Liu Cixin, *Death's End*

CMPUT 628 Deep RL

Marlos C. Machado

Book Cover

Class 15/ 25

Reminders & Notes

- Assignment 3 is due this weekend
- Assignment 4 is due March 14
- You should also be already thinking about your Seminar and Paper Commentary

Please, interrupt me at any time!

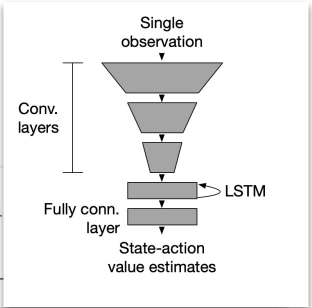
So far: DQN and much more

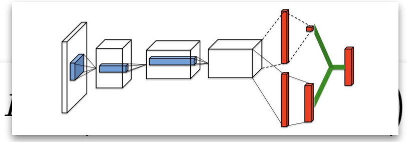
$$Y(R_{t+1}, O_{t+1}; \boldsymbol{\theta}_t) = R_{t+1} + \gamma Q(O_{t+1}, \arg\max_{a\in\mathcal{A}} Q(O_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}^-)$$

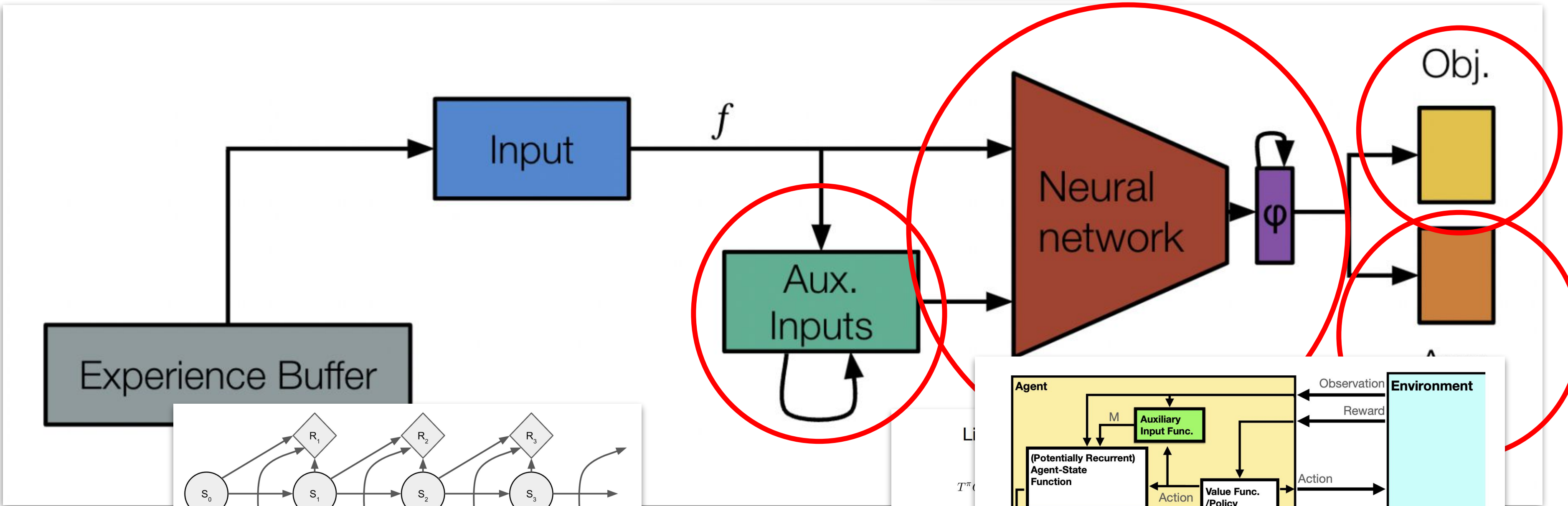
$$\mathcal{L}^{\text{DDQN}} = \mathbb{E}_{\tau\sim U(\mathcal{D})}\Big[\big(Y(R_{t+1}, O_{t+1}; \boldsymbol{\theta}_t) - Q(O_t, A_t; \boldsymbol{\theta}_t)\big)^2\Big],$$

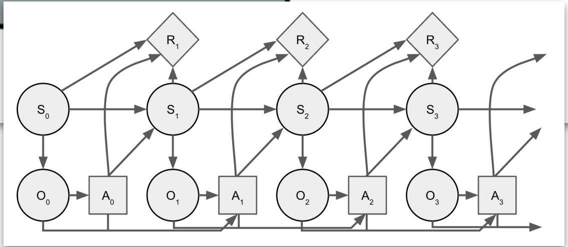
$$Y_n(R_{t+1:n}, O_{t+n}; \boldsymbol{\theta}^-) = \sum_{t=0}^{n-1} \gamma^t R_{t+1}$$

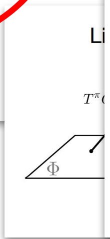
$$\mathcal{L}^{\text{DQN}}_{\text{n-step}} = \mathbb{E}_{\tau^n\sim U(\mathcal{D})}\Big[\big(Y_n(R_{t+1:n}, O_{t-}$$

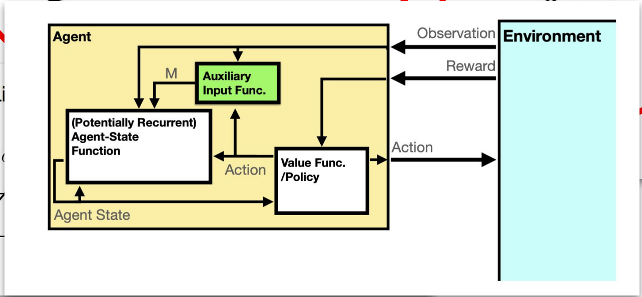




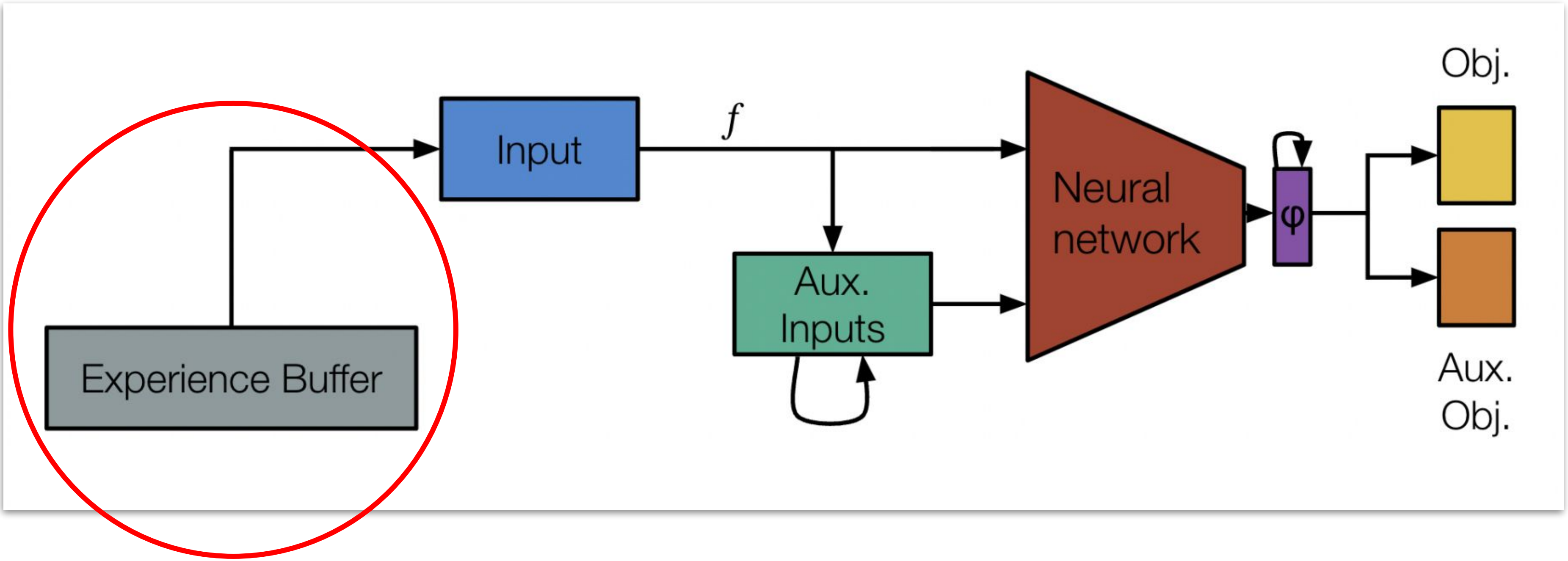


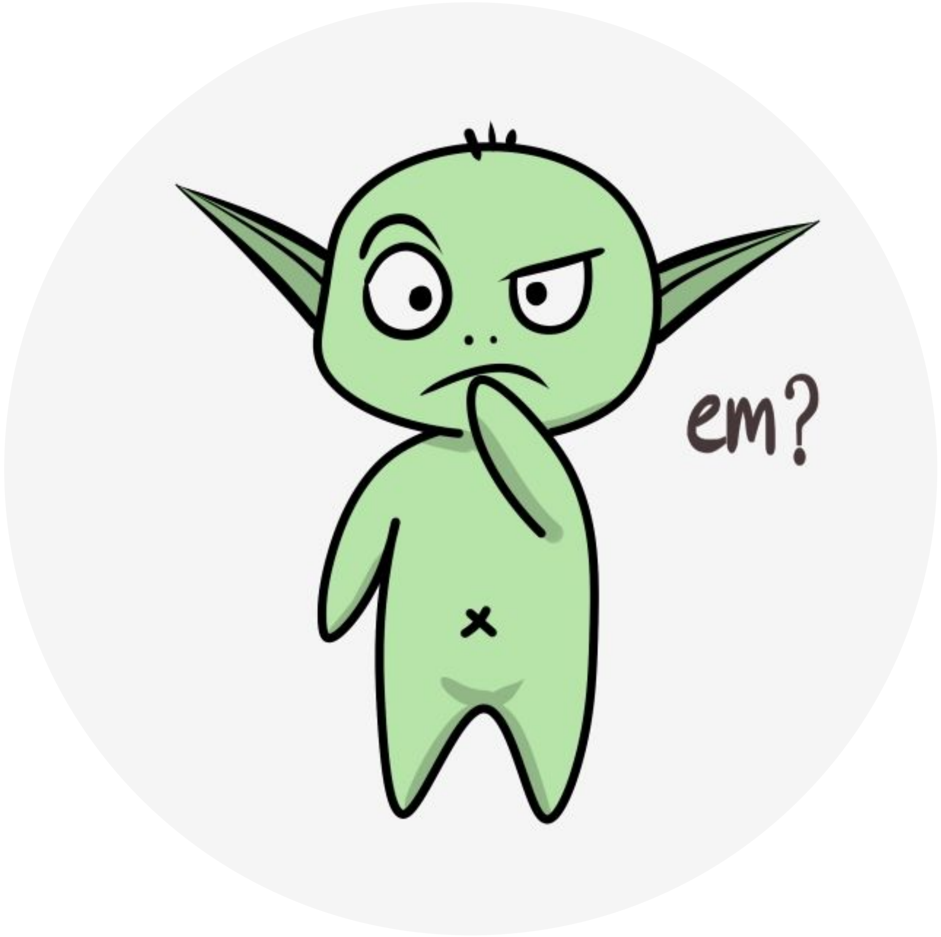






Today


Experience Buffer
Input
f
Aux. Inputs
Neural network
φ
Obj.
Aux. Obj.



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Lin (1991, 1992): The origins of the experience replay buffer

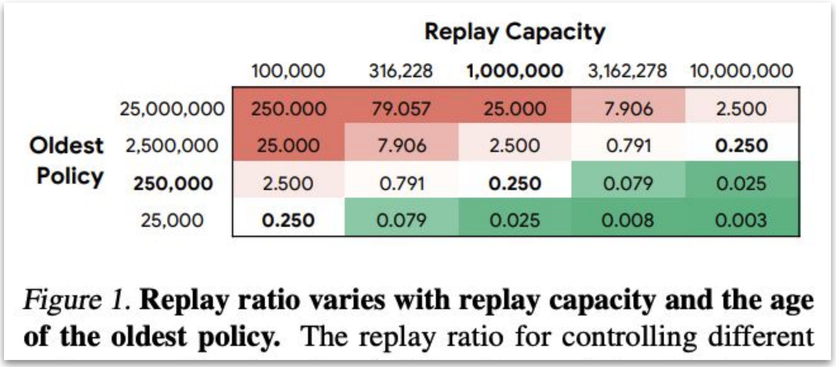
- 30+ years ago, Lin was already evaluating the pairing of Q-learning and neural networks in “moderately complex and nondeterministic” environments
- The experience replay buffer was introduced to improve learning efficiency instead of having a sample used “only once and then thrown away”
- Decorrelating data never motivated the experience replay buffer. It is even said that it can be more effective if a “sequence of experiences is replayed in temporally backward order”.
- Lin also wrote that ideally, “the agent should only replay the experiences involving actions that still follow the current policy”

Mnih et al. (2013; 2015) resuscitated the experience replay buffer

- When first mentioned, it is said that the experience replay buffer “randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution”
  - “learning directly from consecutive samples is inefficient, owing to the strong correlations between the samples; randomizing the samples breaks these correlations and therefore reduces the variance of the updates”
- Besides that, “each step of experience is potentially used in many weight updates, which allows for greater data efficiency”

Hyperparameters really matter

- The size of the experience replay buffer is a very sensitive hyperparameter
  - The size of the replay buffer obviously impacts the amount of off-policy data the agent uses
  - The same is true for the number of updates per time step (and many other hypers, actually)
- The reality is that many hyperparameters, and algorithmic choices, are deeply intertwined — the traditional assumptions about independence are wrong
- Fedus et al. (2019) has coined the term *replay ratio: "the number of gradient updates per environment transition"*, which is arguably what really matters

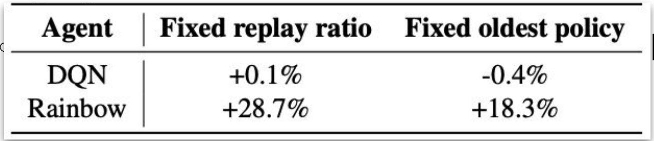


		Replay Capacity				
		100,000	316,228	**1,000,000**	3,162,278	10,000,000
	25,000,000	250.000	79.057	25.000	7.906	2.500
Oldest Policy	2,500,000	25.000	7.906	2.500	0.791	**0.250**
	250,000	2.500	0.791	**0.250**	0.079	0.025
	25,000	**0.250**	0.079	0.025	0.008	0.003

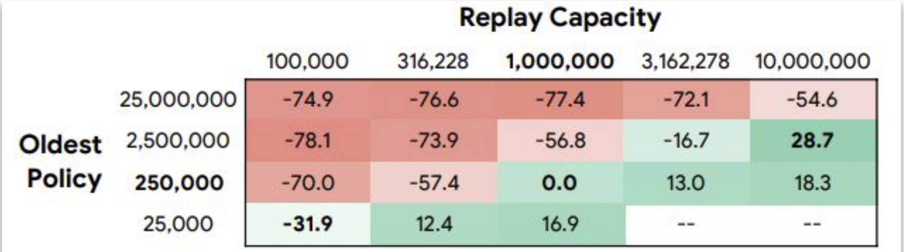
Figure 1. **Replay ratio varies with replay capacity and the age of the oldest policy.** The replay ratio for controlling different

Hyperparameters really matter (Fedus et al., 2019)

- The replay ratio can have a really big impact on performance.
- Results by Fedus et al. (2019) with Rainbow (Hessel et al., 2018) on a subset of 14 Atari 2600 games averaged over 3 seeds (using sticky actions).
- Insights (on Rainbow, not DQN):
  - Increasing replay capacity improves perf.
  - perf.
  - But "sparse-reward games" benefit from data generated by older policies



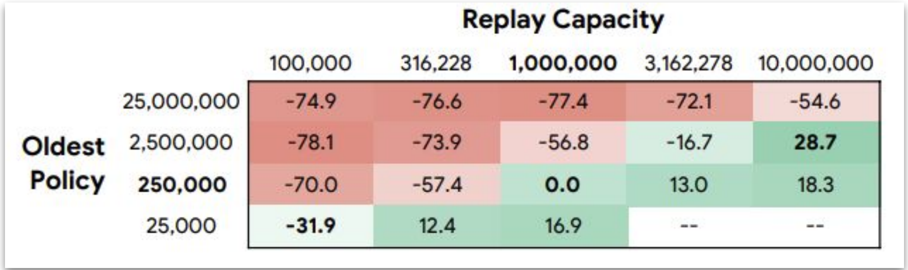
Agent	Fixed replay ratio	Fixed oldest policy
DQN	+0.1%	-0.4%
Rainbow	+28.7%	+18.3%



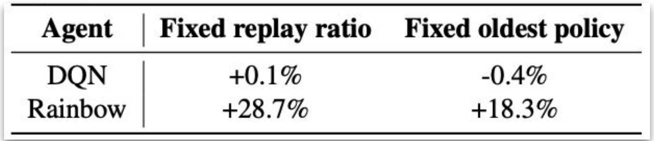
		Replay Capacity				
		100,000	316,228	**1,000,000**	3,162,278	10,000,000
Oldest Policy	25,000,000	-74.9	-76.6	-77.4	-72.1	-54.6
	2,500,000	-78.1	-73.9	-56.8	-16.7	**28.7**
	250,000	-70.0	-57.4	**0.0**	13.0	18.3
	25,000	**-31.9**	12.4	16.9	--	--

Figure 2. **Performance consistently improves with increased replay capacity and generally improves with reducing the age of the oldest policy.** Median percentage improvement over the Rainbow baseline when varying the replay capacity and age of the oldest policy in Rainbow on a 14 game subset of Atari. We do not run the two cells in the bottom-right because they are extremely expensive due to the need to collect a large number of transitions per policy.

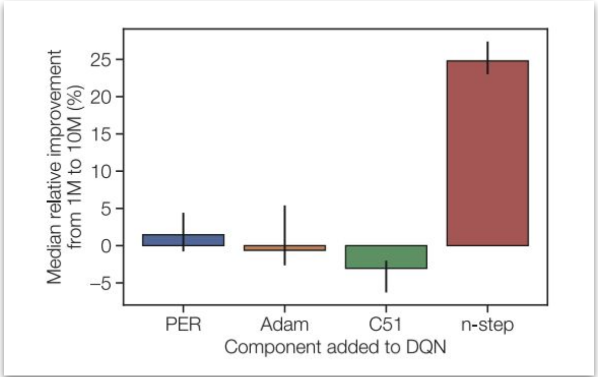
Which one of Rainbow's is the culprit? (Fedus et al., 2019)

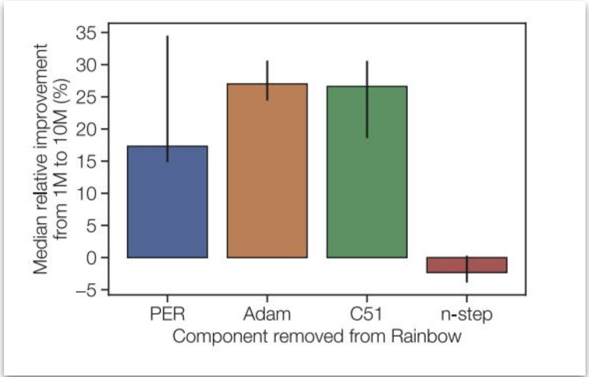


		Replay Capacity				
		100,000	316,228	**1,000,000**	3,162,278	10,000,000
Oldest Policy	25,000,000	-74.9	-76.6	-77.4	-72.1	-54.6
	2,500,000	-78.1	-73.9	-56.8	-16.7	**28.7**
	250,000	-70.0	-57.4	**0.0**	13.0	18.3
	25,000	**-31.9**	12.4	16.9	--	--

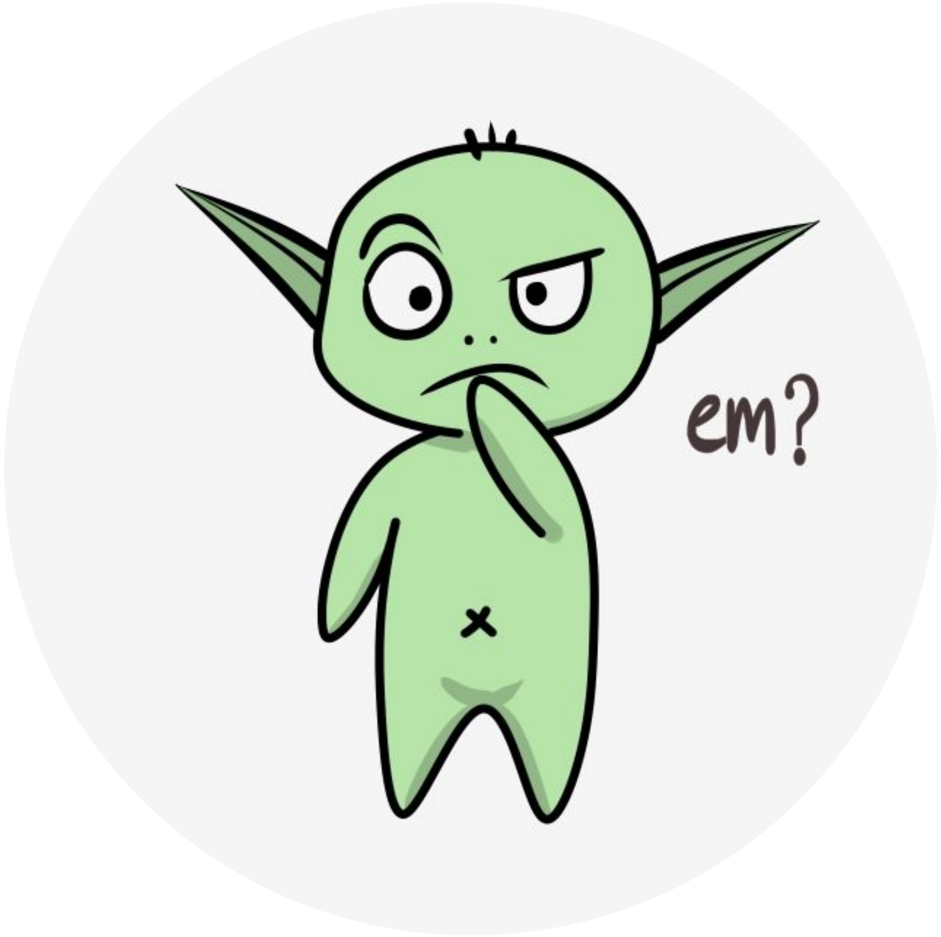


Agent	Fixed replay ratio	Fixed oldest policy
DQN	+0.1%	-0.4%
Rainbow	+28.7%	+18.3%





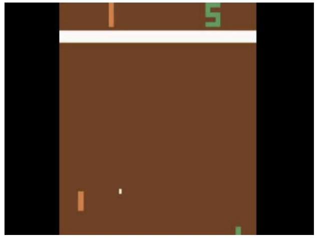
Why? Because of the deadly triad and a smaller bootstrap? Variance reduction? Unclear

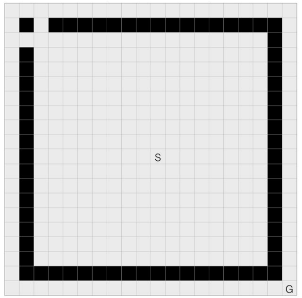


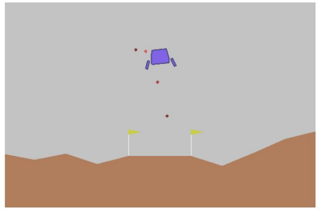
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

There are other takes on experience replay buffers

- Zhang and Sutton (2017) present a quite opinionated one
- Empirical analysis is somewhat contrived
  - Tabular and LFA are not interesting in this context
  - Non-LFA is done with 1 hidden layer; it is not surprising Fedus et al. (2016) were unable to reproduce Zhang and Sutton's (2016) findings
- The idea of incorporating the last sample into the minibatch (size 10) is interesting

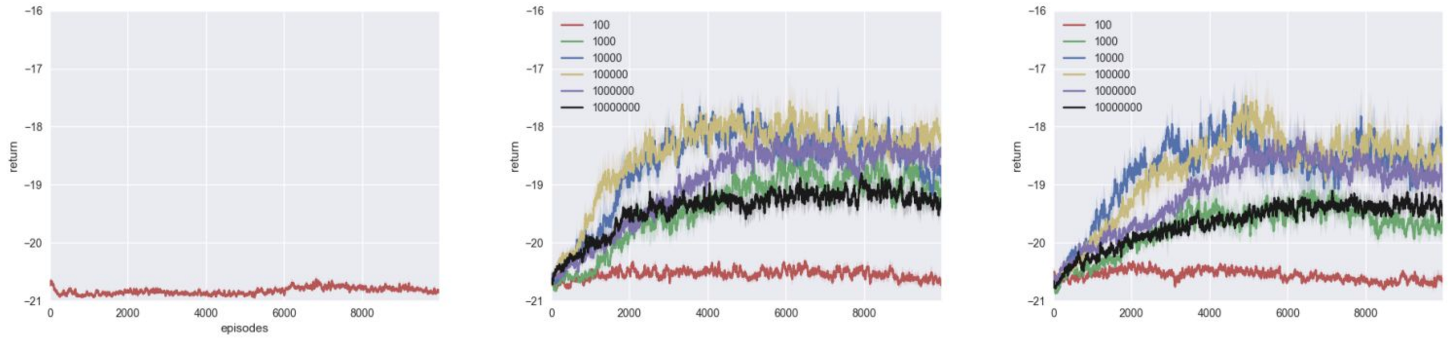






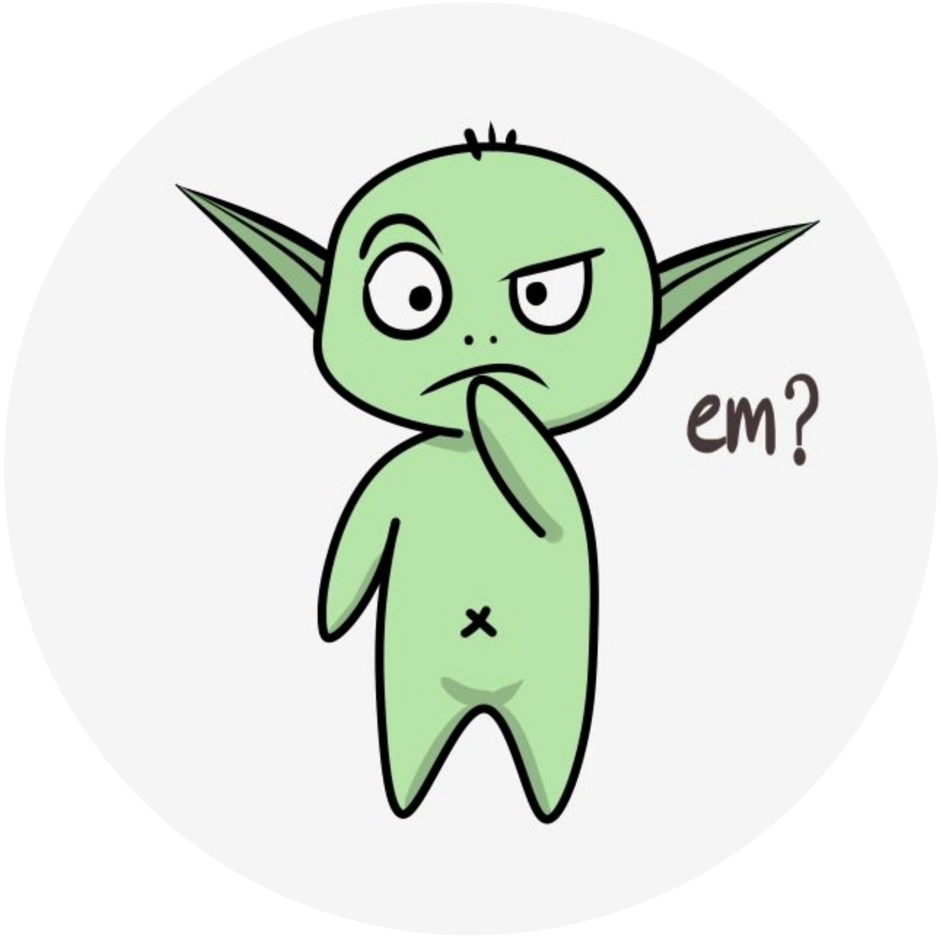
Non-LFA Results by Zhang and Sutton (2017)

RAM features



(a) Online-Q (b) **DQN** (c) **DQN + CER**

Figure 6: Training progression with non-linear function representation in the game Pong. Lines with different colors represent replay buffers with different size, and the number inside the image shows the replay buffer size. The results are averaged over 10 independent runs, and standard errors are plotted. The curves are smoothed by a sliding window of size 30. It is expected that the agent does not solve the game Pong, as it is to difficult to approximate the state-value function with a single-hidden-layer network.

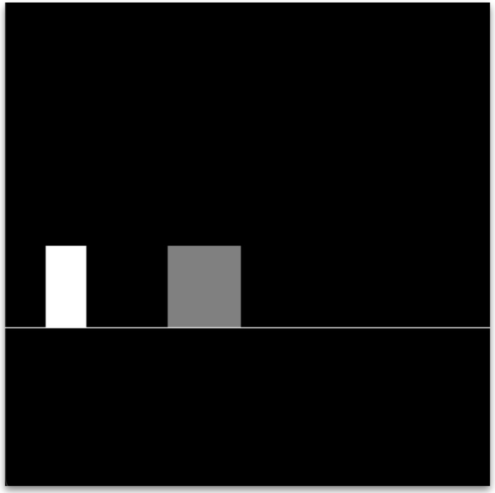


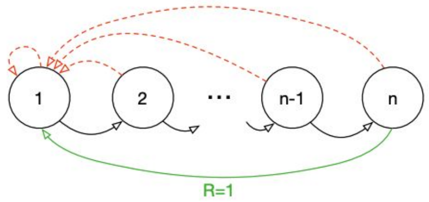
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

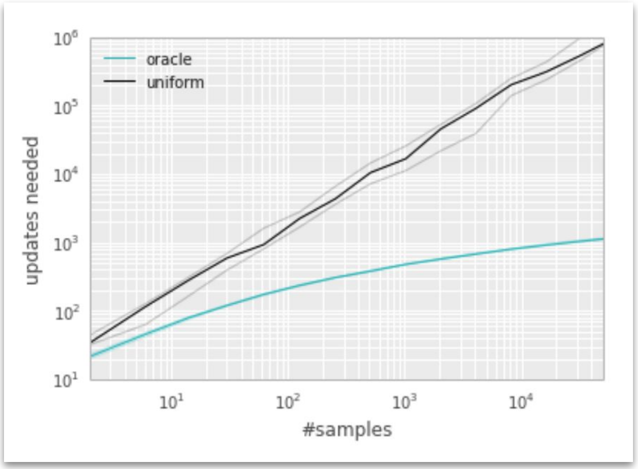
Schaul et al. (2016): Prioritized Experience Replay (PER)

- In most approaches, samples are drawn *uniformly at random*
- Are there transitions that are more informative than others?
  - At least for value learning, yes – This dates back all the way to prioritized sweeping (Moore & Atkeson; 1993)
  - And there are even some transitions that are *rare*
- Key idea: Prioritize experience according to their *usefulness* at a specific time
  - Specifically, Schaul et al. propose prioritizing transitions by the magnitude of their TD error
- We can think about which experiences to store and which experiences to sample; PER is about sampling

You have faced a pedagogical example before

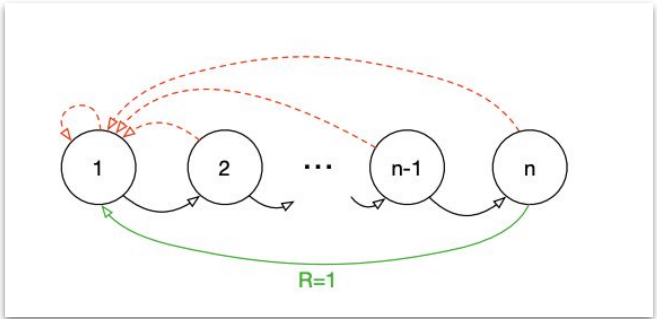


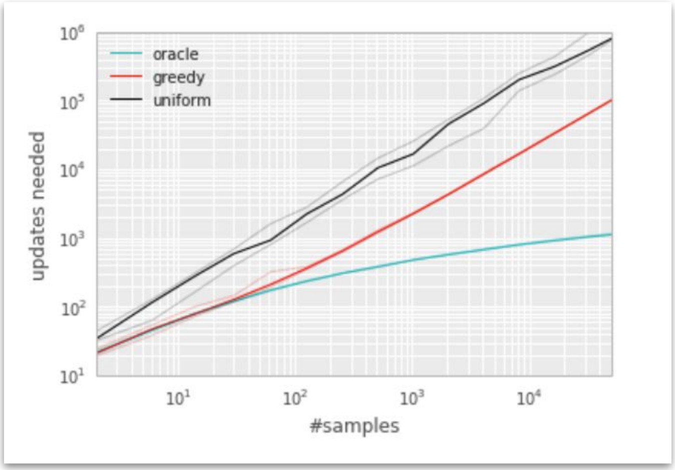




Schaul et al. (2016)

Main idea: Prioritize transitions by the magnitude of their TD error (Schaul et al., 2016), and if they've been sampled before





Implementing this naively can be quite inefficient. Schaul et al. (2016) implemented their replay buffer as a binary heap, which makes sampling O(1) and updating O(*log N*)

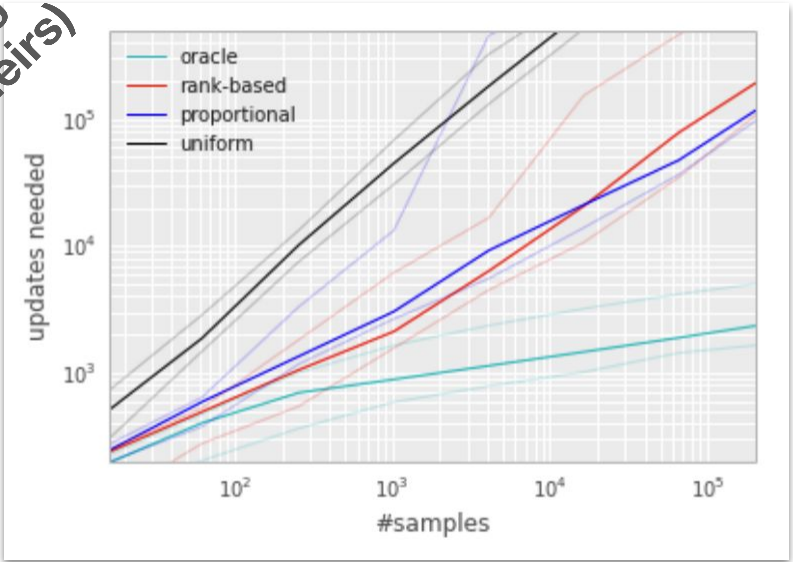
Greedy is not great, though (Schaul et al., 2016)

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

Now, you also need importance sampling (but they anneal theirs)

Proportional: $p_i = |\delta_i| + \epsilon$

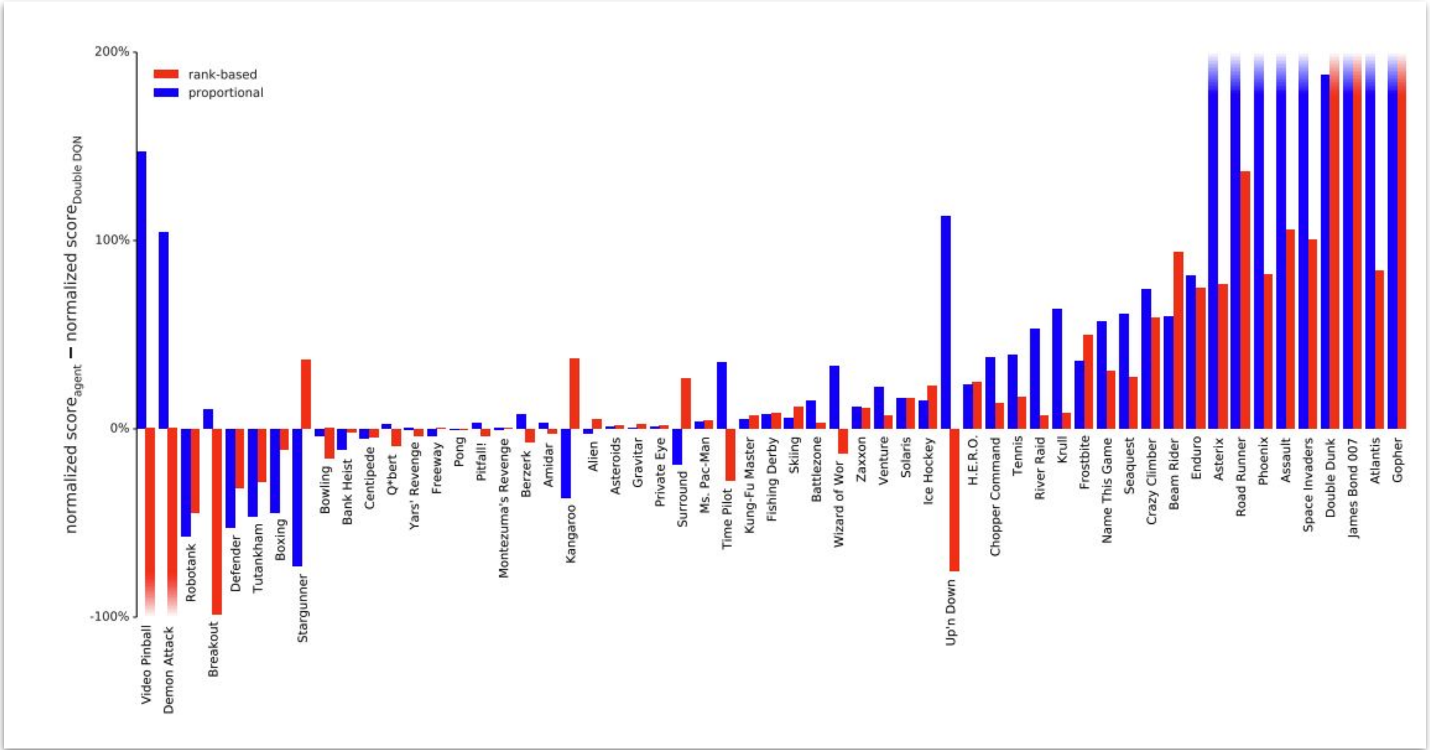
Rank-based: $p_i = \frac{1}{\text{rank}(i)}$



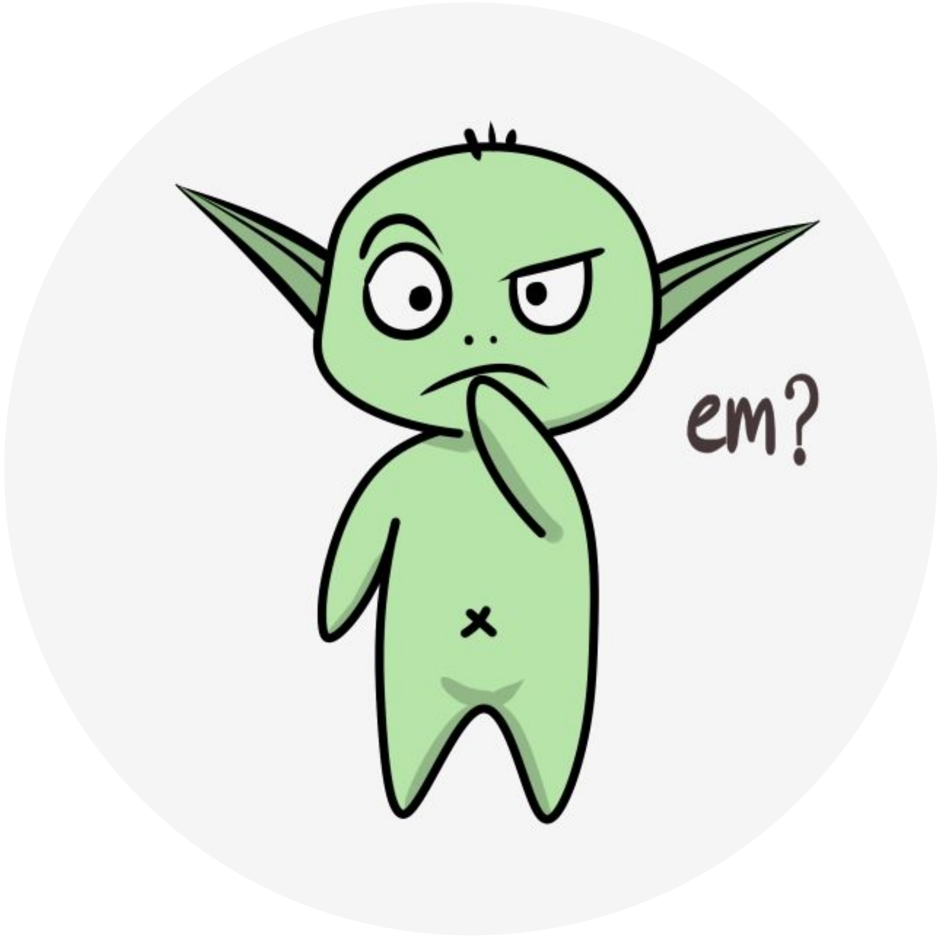
*Linear function approximation

Implementing these is even trickier. Schaul et al. (2016) discuss using a piecewise linear function with k segments for the proportional strategy, and a sum-tree for the rank-based one

Guess what? It works ¯_(ツ)_/¯ (... in Atari 2600 games)



*1 seed? No-op setting, similar to what Double DQN



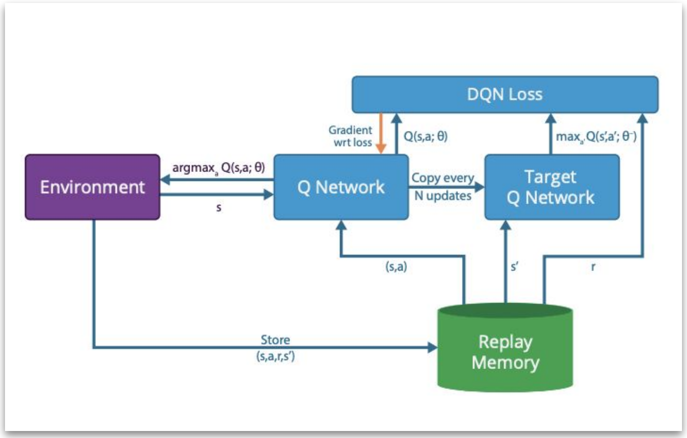
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Distributed Experience Replay Buffers

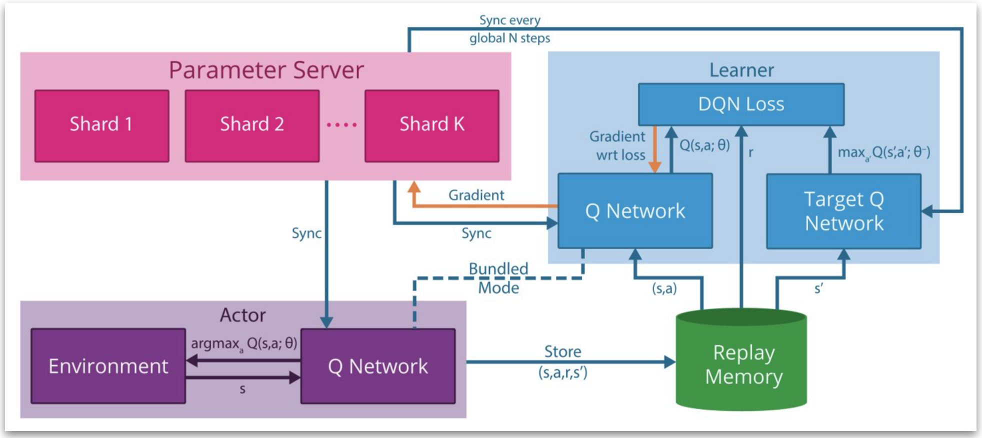
- Can we learn faster?
  - Bottleneck: Data generation. Can we parallelize that?
- Idea: Have multiple agents running in parallel while interacting with *multiple instances of the same environment*
- Before we even continue, you have to realize that going to a distributed data generating system **is changing the problem formulation and constraints**
  - *Or maybe, it is in fact just generating data faster ¯_(ツ)_/¯*
- Actors can store their own experience, leading to a distributed experience replay buffer, or can have a shared, central experience replay buffer (Nair et al., 2015)

General RL Architecture (Gorila) (Nair et al., 2015)

- Distributed training of DQN through Asynchronous Stochastic Gradient Descent (ASGD), leading to a 10× speed-up



DQN

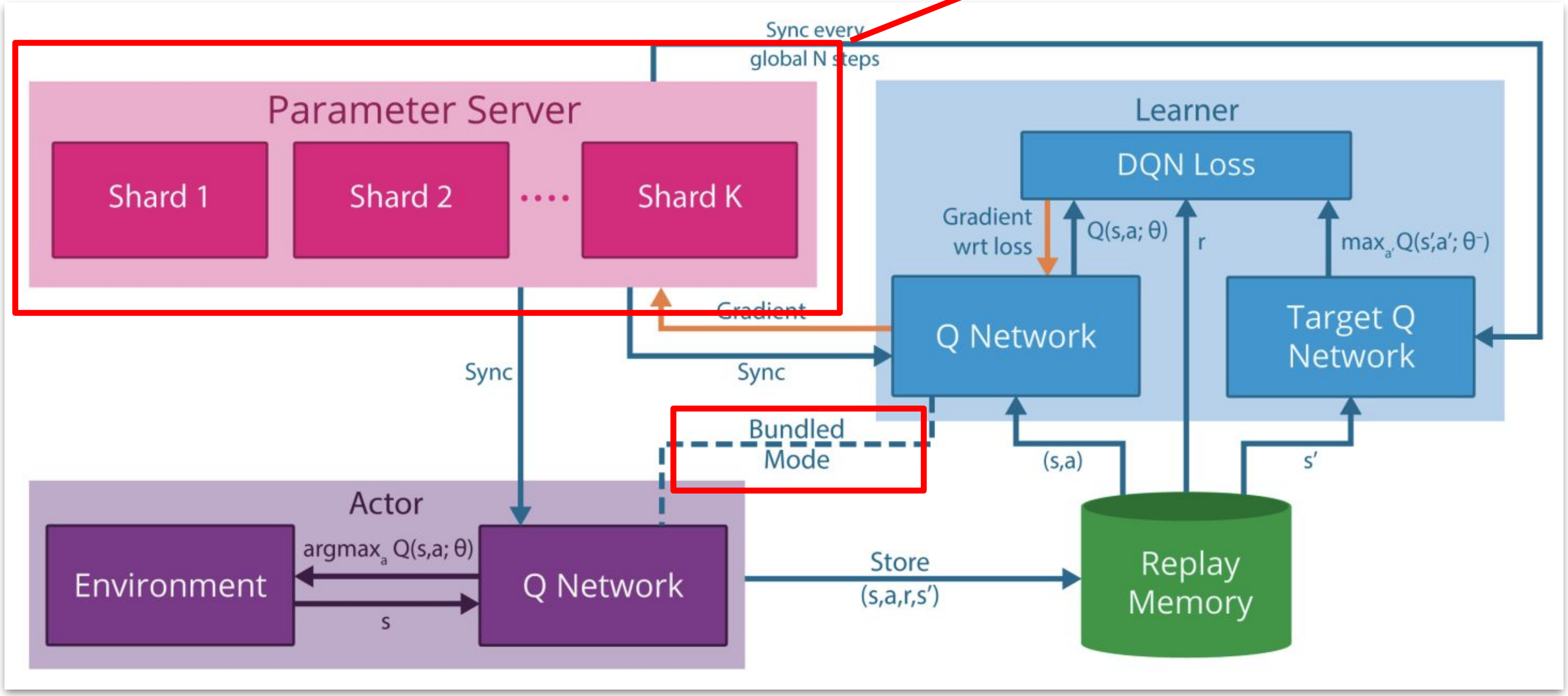


Gorila

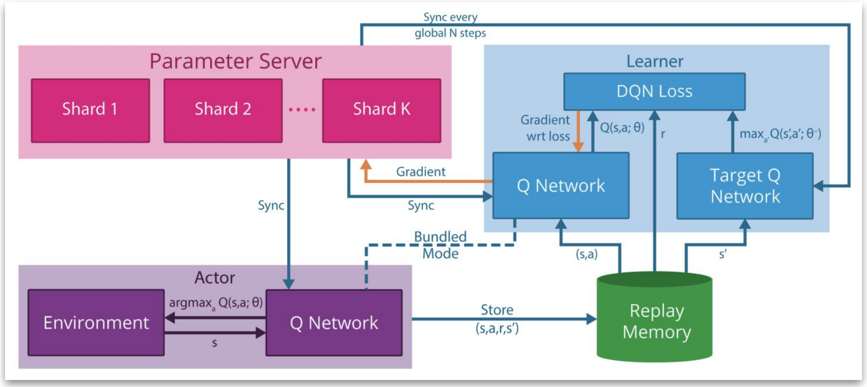
* This is obviously a new architecture as well

General RL Architecture (Gorila) (Nair et al., 2015)

Distributed Q-Network $Q(s,a; \theta^{+})$

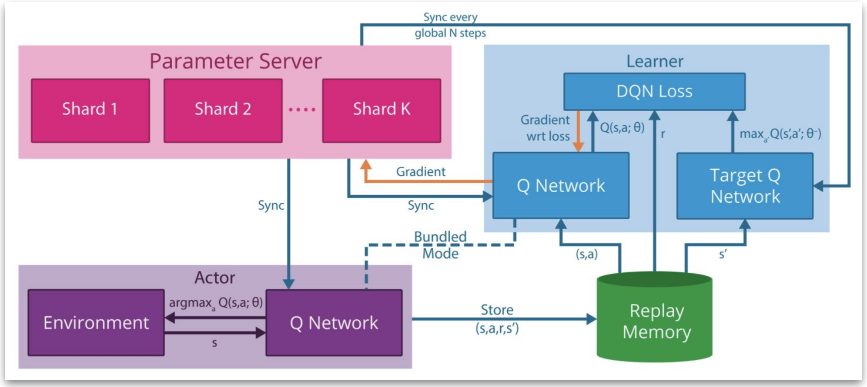


Gorila DQN (Nair et al., 2015)



- Parameter server parameters are θ^+
- Actors and Learners have replicas of the current Q-Network, $Q(s,a; \theta)$, and they update from parameter server, θ^+
- Learner has both the local, current Q-Network, $Q(s,a; \theta)$, a target net $Q(s,a; \theta^-)$ The learner sends the gradient updates to the parameter server
- Actor and learner networks, $Q(s,a; \theta)$, are synchronized before *every acting* step
- "The learner's target network is updated from the parameter server θ^+ after every *N* gradient updates in the central parameter server."
- "gradients are not applied directly, but instead communicated to the (...) server"

Gorila DQN (Nair et al., 2015)



- A couple of stability tricks:
  - "All gradients older than the threshold are discarded by the parameter server"
  - "each actor/learner keeps a running average and standard deviation of the absolute DQN loss for the data it sees and discards gradients with absolute loss higher than the mean plus several standard deviations"
  - "we used the AdaGrad update rule (Duchi et al., 2011)."

Gorila DQN in Atari 2600 Games (Nair et al., 2015)

- “In all experiments, Gorila DQN used: $N_{param} = 31$ and $N_{learn} = N_{act} = 100$. We use the bundled mode. Replay memory size D = 1 million frames and used ε-greedy as the behaviour policy with annealed from 1 to 0.1 over the first one million global updates. Each learner syncs the parameters θ^{-} of its target network after every 60K parameter updates performed in the parameter server.”
    - 100 actors and learners running in parallel, with the parameter vector split across 31 machines

Gorila DQN in Atari 2600 Games (Nair et al., 2015)

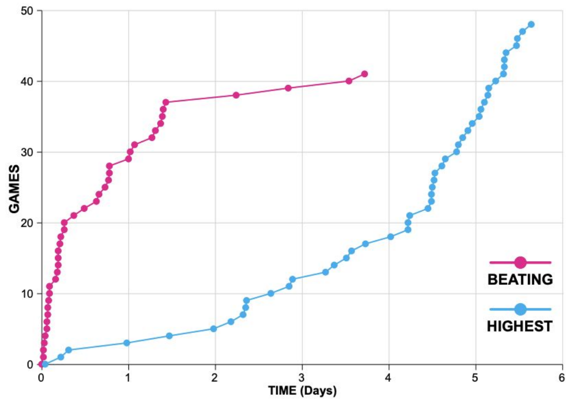
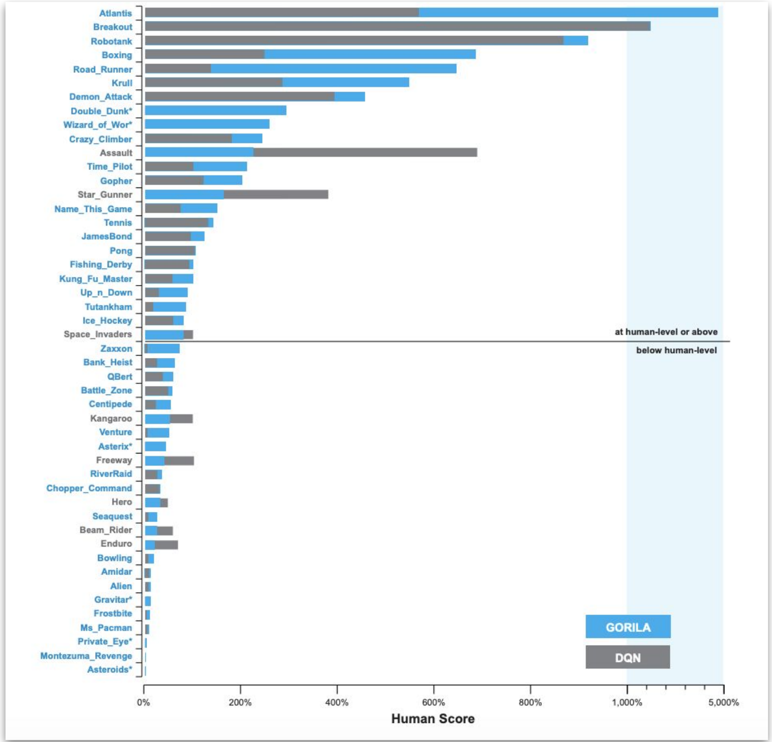
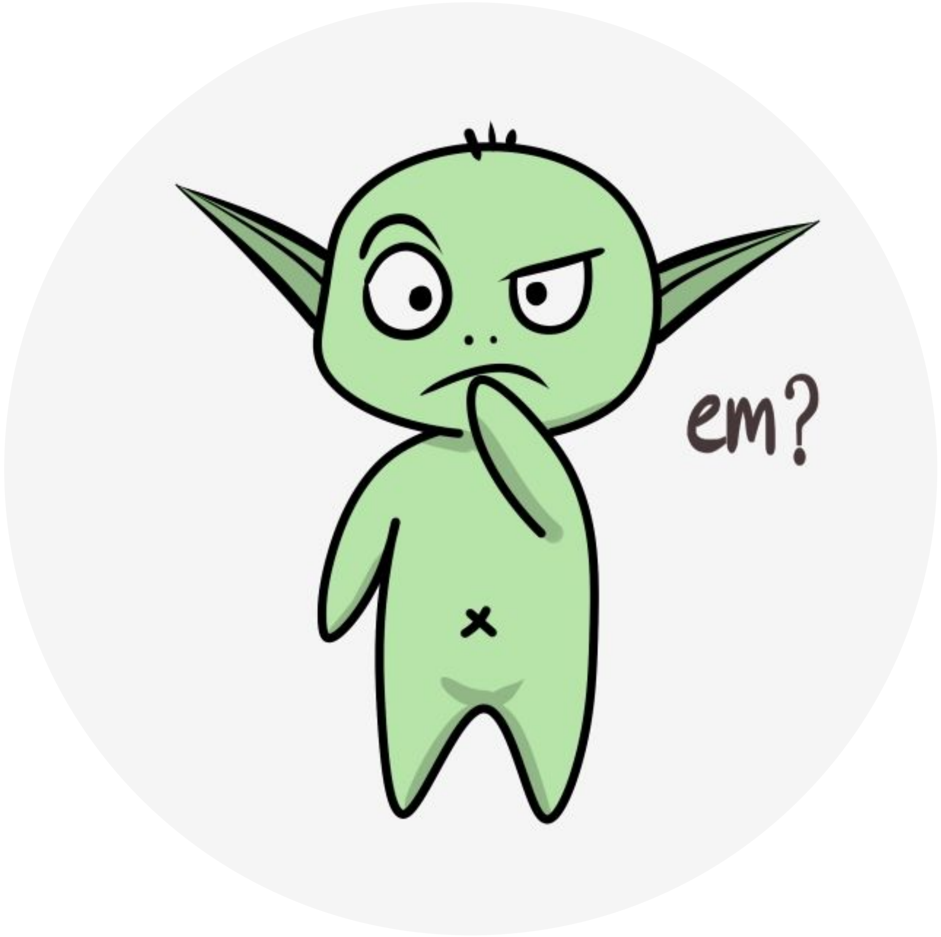


Figure 5. The time required by Gorila DQN to surpass single DQN performance (red curve) and to reach its peak performance (blue curve).

vs 12-14 days



but seeing 100x more data



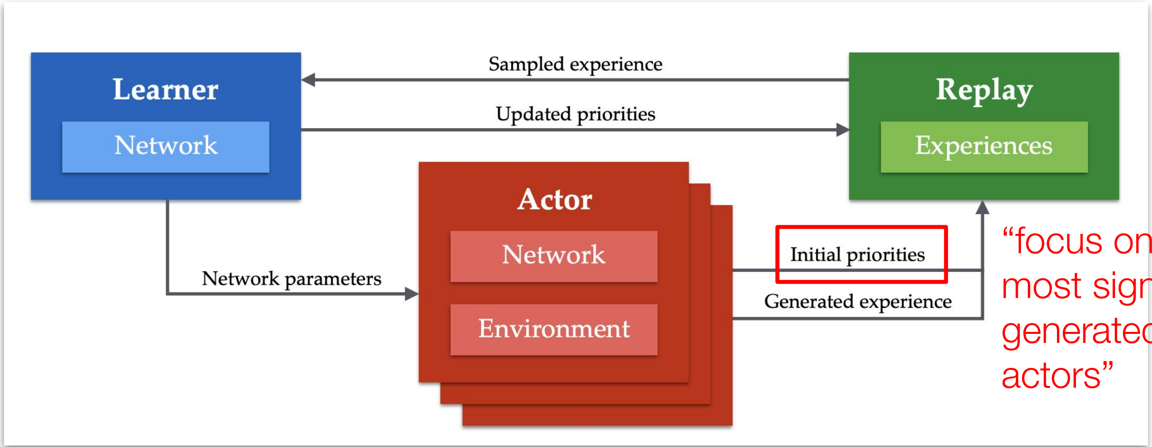
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Distributed Prioritized Experience Replay (Horgan et al., 2018)

- Motivation: “Sharing experiences has certain advantages compared to sharing gradients. Low latency communication is not as important as in distributed SGD, because experience data becomes outdated less rapidly than gradients, provided the learning algorithm is robust to off-policy data.”
- Idea: to decouple acting from learning, focusing on distributed generation of experience (instead of parallelizing the computation of gradients)

Distributed Prioritized Experience Replay (Horgan et al., 2018)

- Idea: to decouple acting from learning, focusing on distributed generation of experience (instead of parallelizing the computation of gradients)
    - Specifically: "the actors interact with their own instances of the environment by selecting actions according to a shared neural network, and accumulate the resulting experience in a *shared experience replay memory*; the learner replays samples of experience and updates the neural network"



"focus only on the most significant data generated by the actors"

Distributed Prioritized Experience Replay (Horgan et al., 2018)

- “In our experiments, hundreds of actors run on CPUs to generate data, and a single learner running on a GPU samples the most useful experiences”
  - Uses a shared replay buffer with *shared* priorities (instead of a local one uniformly sampled)
- “We use 360 actor machines (each using one CPU core) to feed data into the replay memory as fast as they can generate it; approximately 139 frames per second (FPS) each, for a total of ~50K FPS, which corresponds to ~12.5K transitions (because of a fixed action repeat of 4) (...) gradients are computed for ~9.7K transitions per second on average”
- They actually considered Double DQN with n-step returns and duelling networks (with prioritized experience replay); not really DQN. Actors used ε-greedy exploration, but with different values of ε across different actors

Distributed Prioritized Experience Replay (Horgan et al., 2018)

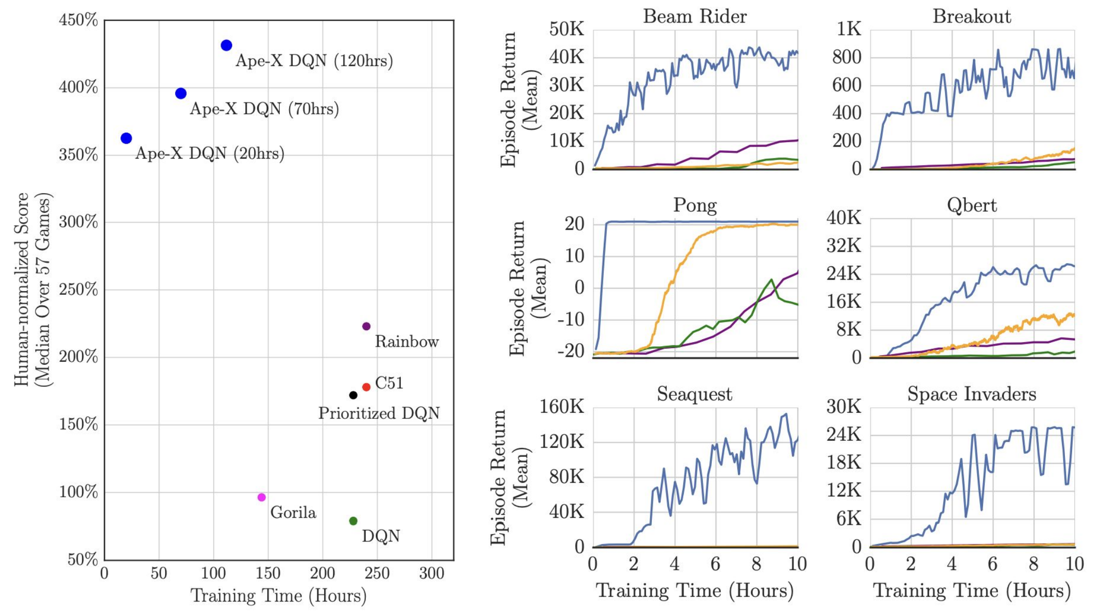
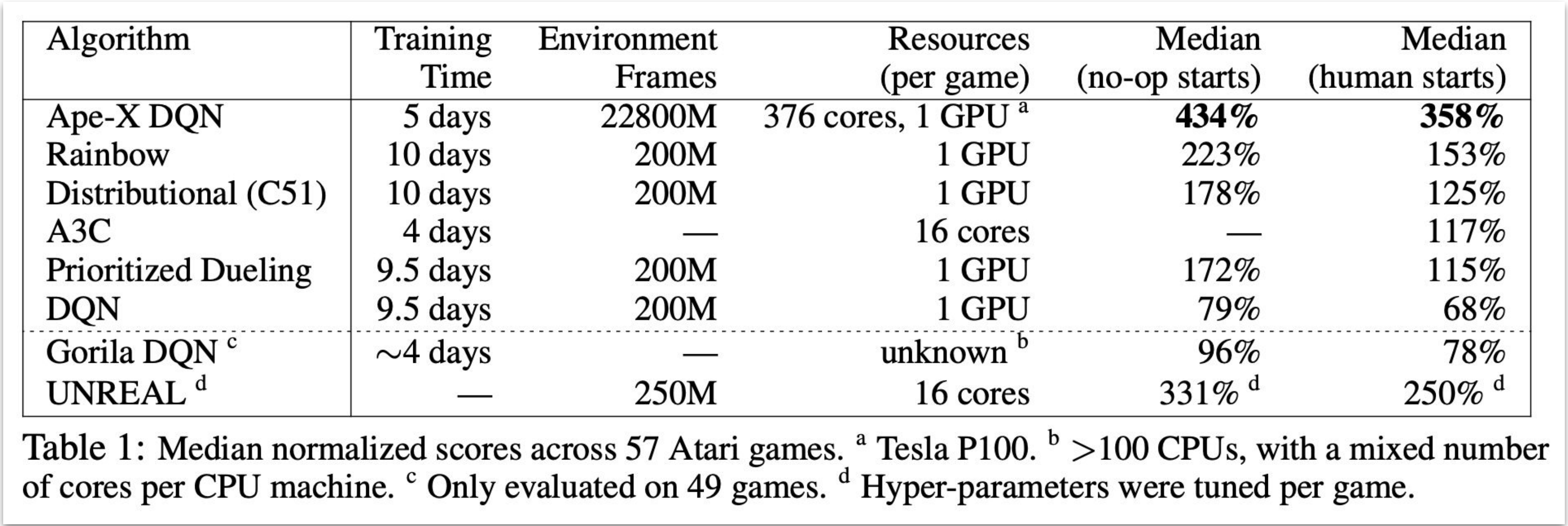


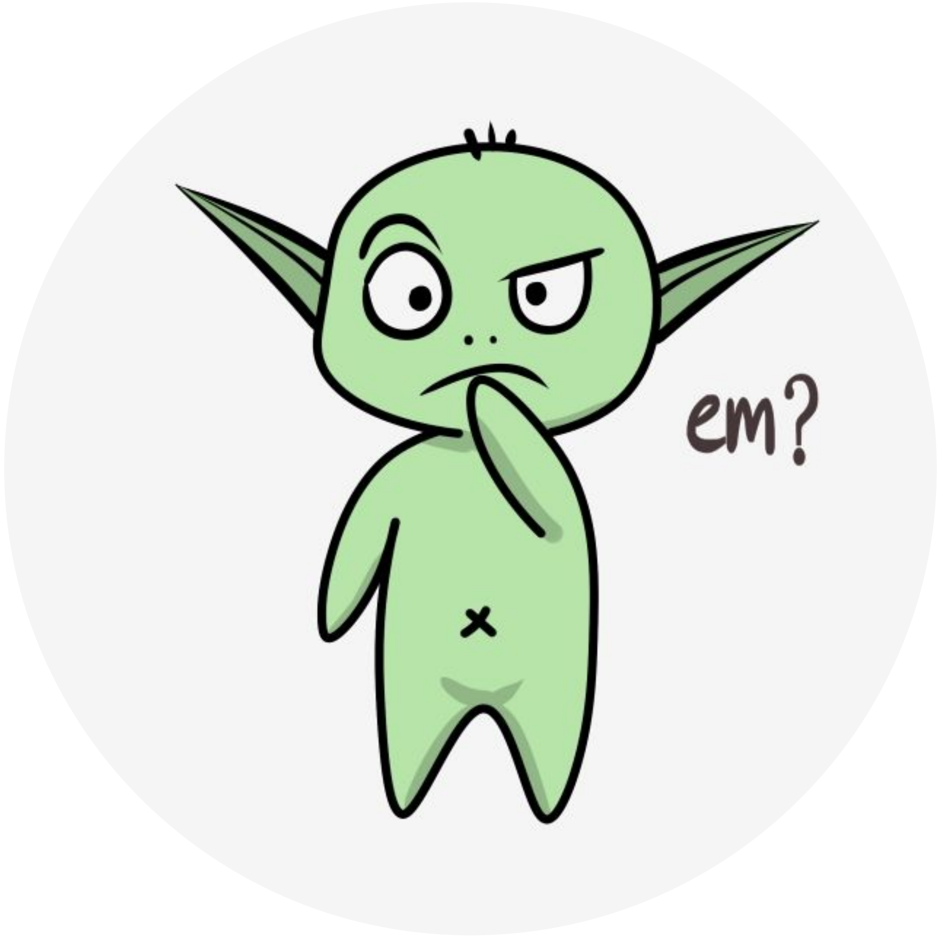
Figure 2: Left: Atari results aggregated across 57 games, evaluated from random no-op starts. Right: Atari training curves for selected games, against baselines. Blue: Ape-X DQN with 360 actors; Orange: A3C; Purple: Rainbow; Green: DQN. See appendix for longer runs over all games.

Distributed Prioritized Experience Replay (Horgan et al., 2018)



Algorithm	Training Time	Environment Frames	Resources (per game)	Median (no-op starts)	Median (human starts)
Ape-X DQN	5 days	22800M	376 cores, 1 GPU [a]	**434%**	**358%**
Rainbow	10 days	200M	1 GPU	223%	153%
Distributional (C51)	10 days	200M	1 GPU	178%	125%
A3C	4 days	—	16 cores	—	117%
Prioritized Dueling	9.5 days	200M	1 GPU	172%	115%
DQN	9.5 days	200M	1 GPU	79%	68%
Gorila DQN [c]	~4 days	—	unknown [b]	96%	78%
UNREAL [d]	—	250M	16 cores	331% [d]	250% [d]

Table 1: Median normalized scores across 57 Atari games. [a] Tesla P100. [b] >100 CPUs, with a mixed number of cores per CPU machine. [c] Only evaluated on 49 games. [d] Hyper-parameters were tuned per game.



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Training RNN-based RL agents from Ape-X (Kapturowski et al., 2019)

- Recurrent Replay Distributed DQN (R2D2) "is most similar to Ape-X, built upon prioritized distributed replay and n-step double Q-learning (with n = 5), generating experience by a large number of actors (typically 256) and learning from batches of replayed experience by a single learner. Like Ape-X, we use the dueling network architecture of Wang et al. (2016), but provide an LSTM layer after the convolutional stack"
- R2D2 has many many details and implementation choices; I won't cover them here, I'll focus on the high-level intuition of what they did with their replay buffer

Training RNN-based RL agents from Ape-X (Kapturowski et al., 2019)

- Kapturowski et al., 2019 argued that Hausknecht and Stone’s (2015) approach might have worked just because their problems were too simple
- Instead, they evaluated
  - Storing the recurrent state of the RNN in the experience replay buffer
  - Allowing the network a burn-in period to get a start state for the LSTM (they consider a sequence of length 80, with a burn-in period of 20 or 40)
- That’s it! The key insights about the distributed architecture come from Ape-X, but they carefully analyzed a couple of options on how to deal with RNNs

Training RNN-based RL agents from Ape-X (Kapturowski et al., 2019)

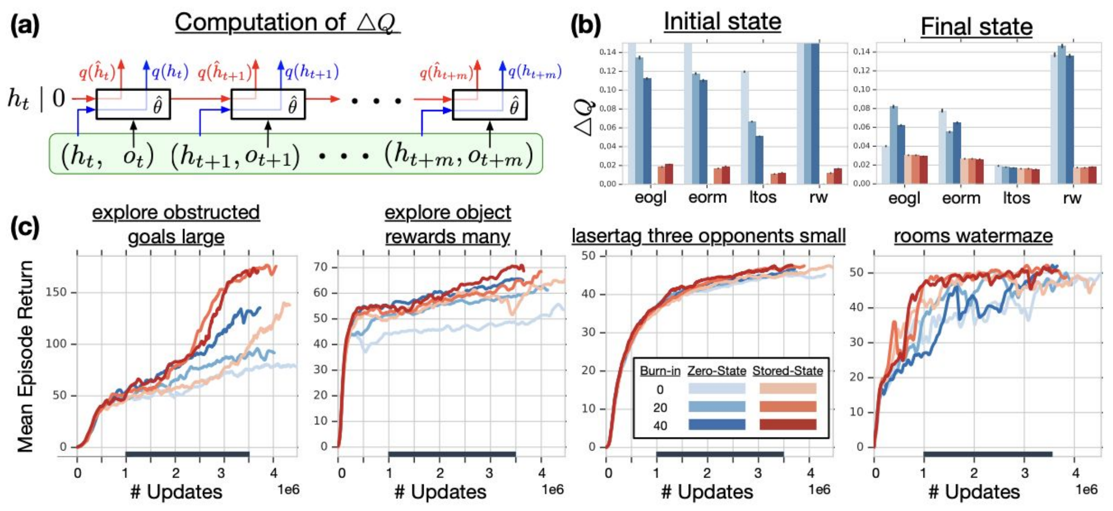


Figure 1: Top row shows Q-value discrepancy ΔQ as a measure for recurrent state staleness. **(a)** Diagram of how ΔQ is computed, with green box indicating a whole sequence sampled from replay. For simplicity, $l = 0$ (no burn-in). **(b)** ΔQ measured at first state and last state of replay sequences, for agents training on a selection of DMLab levels (indicated by initials) with different training strategies. Bars are averages over seeds and through time indicated by bold line on x-axis in bottom row. **(c)** Learning curves on the same levels, varying the training strategy, and averaged over 3 seeds.

It works in Atari 2600 games ¯_(ツ)_/¯ (Kapturowski et al., 2019)

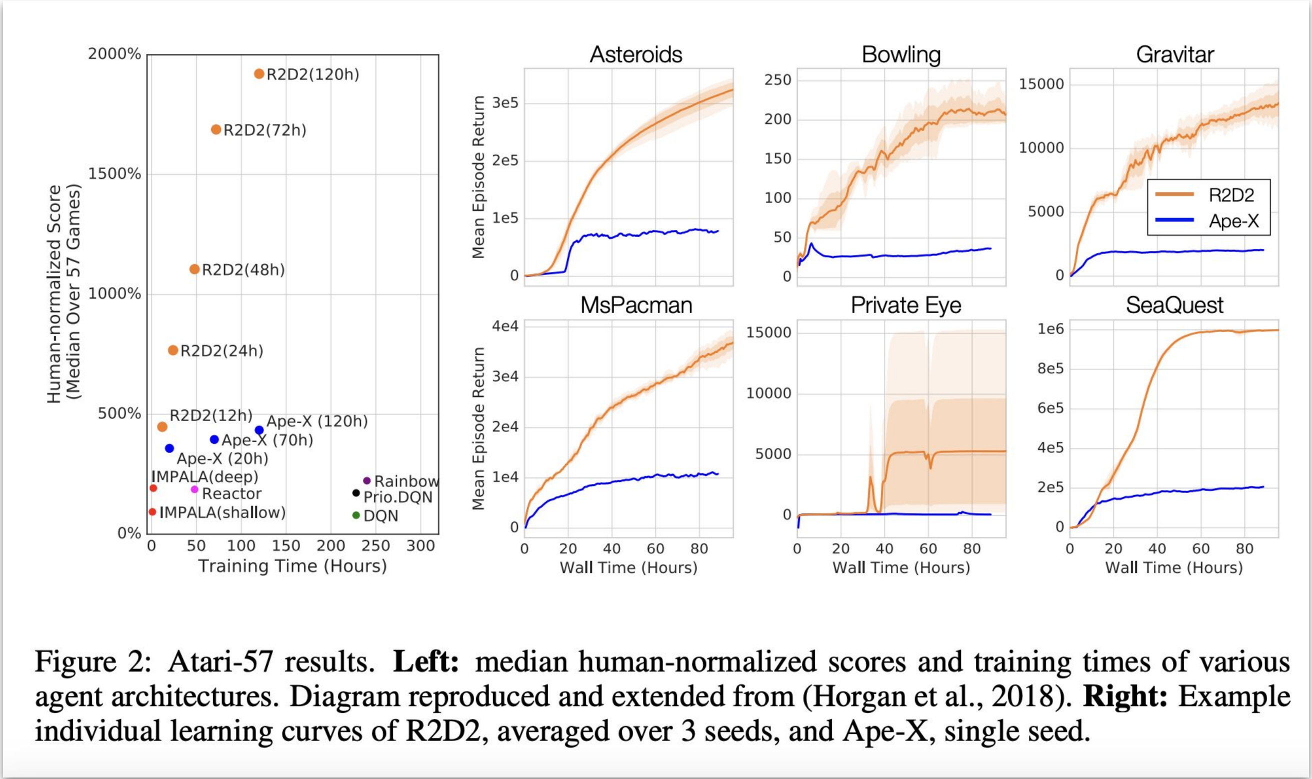


Figure 2: Atari-57 results. **Left:** median human-normalized scores and training times of various agent architectures. Diagram reproduced and extended from (Horgan et al., 2018). **Right:** Example individual learning curves of R2D2, averaged over 3 seeds, and Ape-X, single seed.

50B frames?

And in DM-Lab30 ¯_(ツ)_/¯ (Kapturowski et al., 2019)

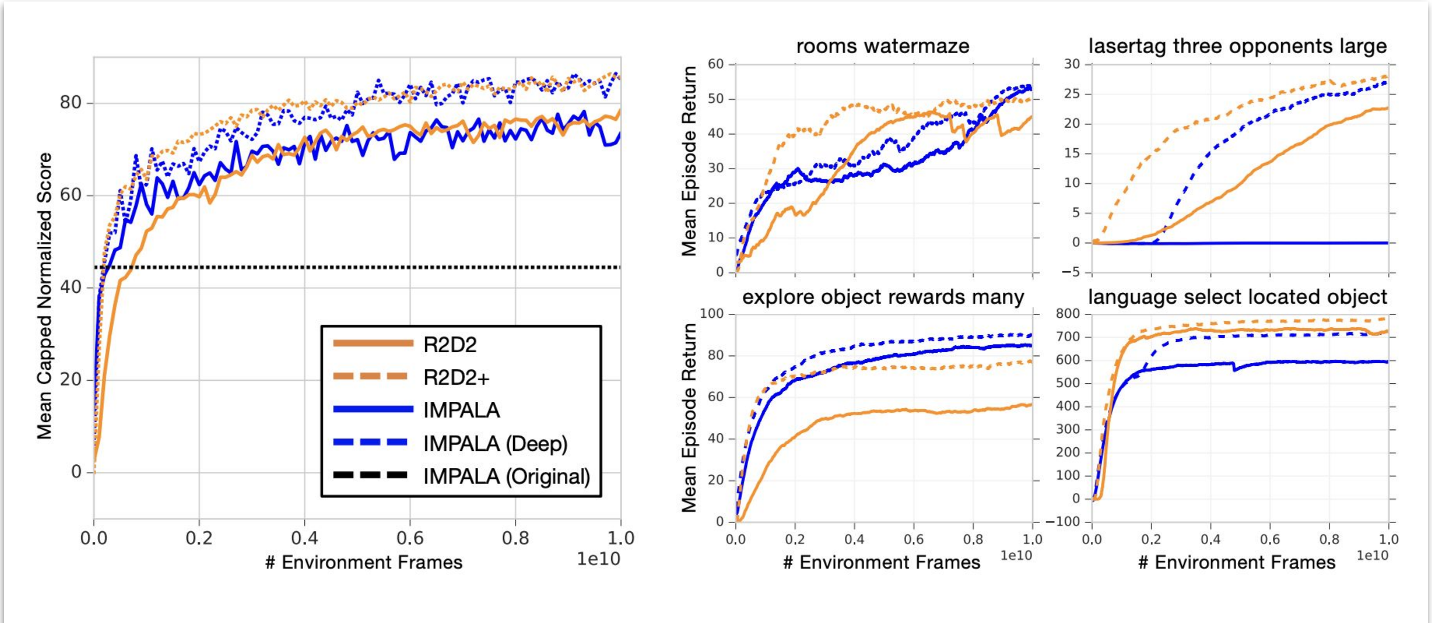
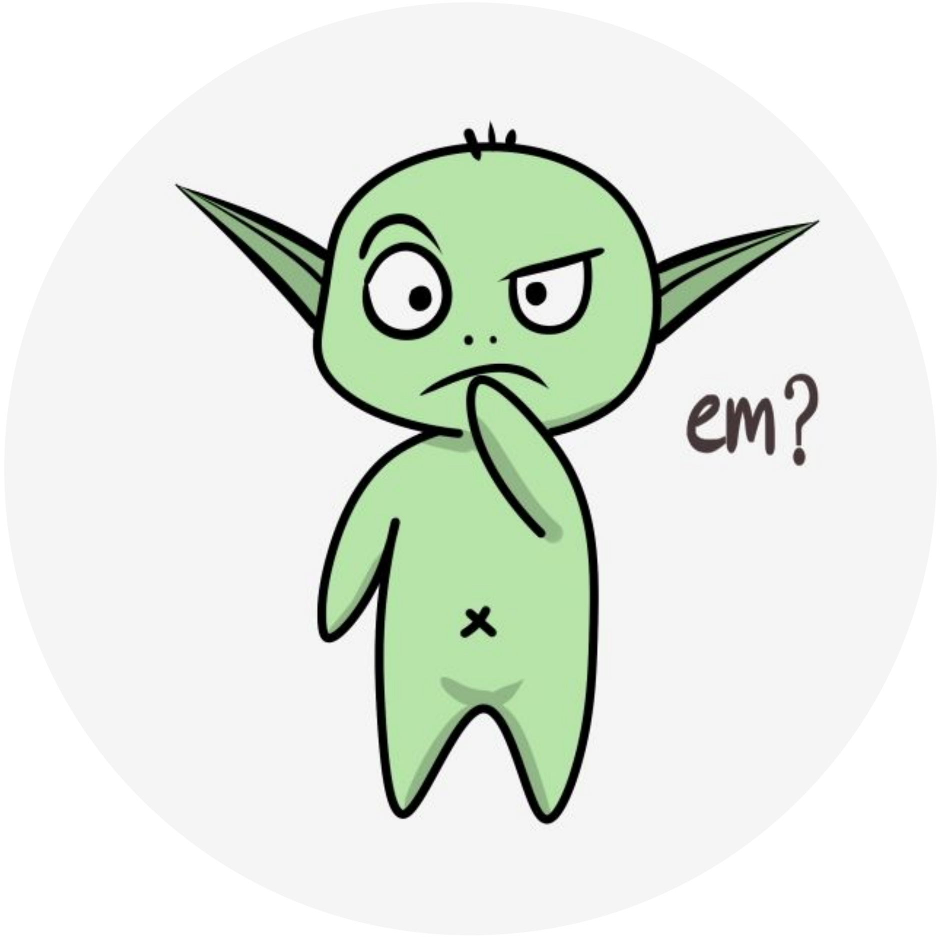


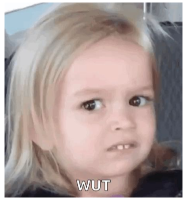
Figure 3: DMLab-30 comparison of R2D2 and R2D2+ with our re-run of IMPALA shallow and deep in terms of mean-capped human-normalized score (Espeholt et al., 2018).



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Distributed training is very appealing

- Who doesn't want to get results faster?
  Even if that means not making apples-to-apples comparisons? ←sarcasm
- Sometimes it just works, your problem is conducive to it and all you care about is to get good results (e.g., controlling balloons in the stratosphere)
- Sometimes data generation in parallel (the problem) is even introduced as a solution to limitations of solution methods



- We will talk more about this in the context of policy gradient methods
  - APE-X, for example, was also evaluated with DDPG

Next class

- What I plan to do:
    - Next next class: I'll start talking about Policy Gradient Methods
- What I recommend YOU to do for next classes:
    - Read
        - *V. Mnih et al.: Asynchronous Methods for Deep Reinforcement Learning. ICML 2016*
        - *T. P. Lillicrap et al.: Continuous Control with Deep Reinforcement Learning. ICLR 2016*
        - *S. Fujimoto, H. van Hoof, D. Meger: Addressing Function Approximation Error in Actor-Critic Methods. ICML 2018*
        - *T. Haarnoja, A. Zhou, P. Abbeel, S. Levine: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. ICML 2018*