# RECURRENT EXPERIENCE REPLAY IN DISTRIBUTED REINFORCEMENT LEARNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Building on the recent successes of distributed training of RL agents, in this paper we investigate the training of RNN-based RL agents from experience replay. We investigate the effects of parameter lag resulting in representational drift and recurrent state staleness and empirically derive an improved training strategy. Using a single network architecture and fixed set of hyper-parameters, the resulting agent, Recurrent Replay Distributed DQN, triples the previous state of the art on Atari-57, and surpasses the state of the art on DMLab-30. R2D2 is the first agent to exceed human-level performance in 52 of the 57 Atari games.

## 1 INTRODUCTION

Reinforcement learning (RL) has seen a rejuvenation of research interest recently due to repeated successes in solving challenging problems such as reaching human-level play on Atari 2600 games (Mnih et al., 2015), beating the world champion in the game of Go (Silver et al., 2017), and playing competitive 5-player DOTA (OpenAI, 2018b). The earliest of these successes leveraged experience replay for data efficiency and stacked a fixed number of consecutive frames to overcome the partial observability in Atari 2600 games. However, with progress towards increasingly difficult, partially observable domains, the need for more advanced memory-based representations increases, necessitating more principled solutions such as recurrent neural networks (RNNs). The use of LSTMs (Hochreiter & Schmidhuber, 1997) within RL has been widely adopted to overcome partial observability (Hausknecht & Stone, 2015; Mnih et al., 2016; Espeholt et al., 2018; Gruslys et al., 2018).

In this paper we investigate the use of recurrent neural networks with experience replay. We have two primary contributions. First, we perform an empirical study into the effects of various approaches to RNN training with experience replay and how these are affected in distributed training settings. Second, we present an agent that integrates these findings to achieve significant advances in the state-of-the-art on both Atari-57 (Bellemare et al., 2013) and DMLab-30 (Beattie et al., 2016) using a single network architecture and set of hyper-parameters.

## 2 BACKGROUND

### 2.1 REINFORCEMENT LEARNING

Our work is set within the Reinforcement Learning (RL) framework (Sutton & Barto, 1998), in which an agent interacts with an environment to maximize the sum of discounted, $\gamma \in [0, 1)$, rewards. We model the environment as a Partially Observable Markov Decision Process (POMDP) given by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, \mathcal{O})$ (Monahan, 1982; Jaakkola et al., 1995; Kaelbling et al., 1998). The underlying Markov Decision Process (MDP) is defined by $(\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, $T$ a transition function mapping state-actions to probability distributions over next states, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. Finally, $\Omega$ gives the set of observations potentially received by the agent and $\mathcal{O}$ is the observation function mapping (unobserved) states to probability distributions over observations.

Within this framework, the agent receives an observation $o \in \Omega$, which may only contain partial information about the underlying state $s \in \mathcal{S}$. When the agent takes an action $a \in \mathcal{A}$ the environment responds by transitioning to state $s' \sim T(\cdot|s, a)$ and giving the agent a new observation, $o' \sim \Omega(\cdot|s')$, and reward, $r \sim R(s, a)$.

Although there are many approaches to RL in POMDPs, we focus on using recurrent neural networks (RNNs) with backpropagation through time (BPTT) (Werbos, 1990) to learn a representation that disambiguates the true state of the POMDP.

The Deep Q-Network agent (DQN) (Mnih et al., 2015) learns to play games from the Atari-57 benchmark by using frame-stacking of $4$ consecutive frames as observations, and training a convolutional network to represent a value function with Q-learning (Watkins & Dayan, 1992), from data continuously collected in a replay buffer (Lin, 1993). Other algorithms like the A3C (Mnih et al., 2016), use an LSTM and are trained directly on the online stream of experience without using a replay buffer. Hausknecht & Stone (2015) combined DQN with an LSTM by storing sequences in replay and initializing the recurrent state to zero during training.

### 2.2 Distributed Reinforcement Learning

Recent advances in reinforcement learning have achieved significantly improved performance by leveraging distributed training architectures which separate learning from acting, collecting data from many actors running in parallel on separate environment instances (Horgan et al., 2018; Espeholt et al., 2018; Gruslys et al., 2018; OpenAI, 2018b;a; Jaderberg et al., 2018).

Distributed replay allows the Ape-X agent (Horgan et al., 2018) to decouple learning from acting, with actors feeding experience into the distributed replay buffer and the learner receiving (randomized) training batches from it. In addition to distributed replay with prioritized sampling (Schaul et al., 2016), Ape-X uses $n$-step return targets (Sutton, 1988), the double Q-learning algorithm (van Hasselt et al., 2016), the dueling DQN network architecture (Wang et al., 2016) and $4$-frame-stacking. Ape-X achieved state-of-the-art performance on Atari-57, significantly out-performing the best single-actor algorithms. It has also been used in continuous control domains and again showed state-of-the-art results, further demonstrating the performance benefits of distributed training in RL.

IMPALA (Espeholt et al., 2018) is a distributed reinforcement learning architecture which uses a first-in-first-out queue with a novel off-policy correction algorithm called V-trace, to learn sequentially from the stream of experience generated by a large number of independent actors. IMPALA stores sequences of transitions along with an initial recurrent state in the experience queue, and since experience is trained on exactly once, this data generally stays very close to the learner parameters. Espeholt et al. (2018) showed that IMPALA could achieve strong performance in the Atari-57 and DMLab-30 benchmark suites, and furthermore was able to use a single large network to learn all tasks in a benchmark simultaneously while maintaining human-level performance.

### 2.3 The Recurrent Replay Distributed DQN Agent

We propose a new agent, the Recurrent Replay Distributed DQN (R2D2), and use it to study the interplay between recurrent state, experience replay, and distributed training. 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, similarly to Gruslys et al. (2018). Instead of regular $(s, a, r, s')$ transition tuples, we store fixed-length ($m = 80$) sequences of $(s, a, r)$ in replay, with adjacent sequences overlapping each other by $40$ time steps, and never crossing episode boundaries. When training, we unroll both online and target networks (Mnih et al., 2015) on the same sequence of states to generate value estimates and targets. We leave details of our exact treatment of recurrent states in replay for the next sections.

Like Ape-X, we use $4$-frame-stacks and the full $18$-action set when training on Atari. On DMLab, we use single RGB frames as observations, and the same action set discretization as Hessel et al. (2018b). Following the modified Ape-X version in Pohlen et al. (2018), we do not clip rewards, but instead use an invertible value function rescaling of the form $h(x) = \mathrm{sign}(x)(\sqrt{|x| + 1} - 1) + \epsilon x$ which results in the following $n$-step targets for the Q-value function:

$$\hat{y}_t = h\left(\sum_{k=0}^{n-1} r_{t+k}\gamma^k + \gamma^n h^{-1}\left(Q(x_{t+n}, a^*; \theta)\right)\right), \quad a^* = \arg\max_a Q(x_{t+n}, a; \theta^-).$$

Here, $\theta^-$ denotes the target network parameters which are copied from the online network parameters $\theta$ every $2500$ learner steps.

Our replay prioritization differs from that of Ape-X in that we use a mixture of max and mean absolute $n$-step TD-errors $\delta_i$ over the sequence: $p = \eta \max_i \delta_i + (1 - \eta)\bar{\delta}_i$. We set $\eta$ and the priority exponent to $0.9$. This more aggressive scheme is motivated by our observation that averaging over long sequences tends to wash out large errors, thereby compressing the range of priorities and limiting the ability of prioritization to pick out useful experience. We also found no benefit from using the importance weighting that has been typically applied with prioritized replay (Schaul et al., 2016), and therefore omitted this step in R2D2.

Finally, compared to Ape-X, we used the slightly higher discount of $\gamma = 0.997$, and disabled the loss-of-life-as-episode-end heuristic that has been used in Atari agents in some of the work since (Mnih et al., 2015). A full list of hyper-parameters is provided in the appendix.

We train the R2D2 agent with a single GPU-based learner, performing approximately 5 network updates per second (each update on a mini-batch of $64$ length-$80$ sequences), and each actor performing $\sim 260$ environment steps per second on Atari ($\sim 130$ per second on DMLab).

## 3 Training Recurrent RL Agents with Experience Replay

In order to achieve good performance in a partially observed environment, an RL agent requires a state representation that encodes information about its state-action trajectory in addition to its current observation. The most common way to achieve this is by using an RNN, typically an LSTM (Hochreiter & Schmidhuber, 1997), as part of the agent's state encoding. To train an RNN from replay and enable it to learn meaningful long-term dependencies, whole state-action trajectories need to be stored in replay and used for training the network. Hausknecht & Stone (2015) compared two strategies of training an LSTM from replayed experience:

- Using a zero start state to initialize the network at the beginning of sampled sequences.
- Replaying whole episode trajectories.

The zero start state strategy's appeal lies in its simplicity, and it allows independent decorrelated sampling of relatively short sequences, which is important for robust optimization of a neural network. On the other hand, it forces the RNN to learn to recover meaningful predictions from an atypical initial recurrent state ('initial recurrent state mismatch'), which may limit its ability to fully rely on its recurrent state and learn to exploit long temporal correlations. The second strategy on the other hand avoids the problem of finding a suitable initial state, but creates a number of practical, computational, and algorithmic issues due to varying and potentially environment-dependent sequence length, and higher variance of network updates because of the highly correlated nature of states in a trajectory when compared to training on randomly sampled batches of experience tuples.

The authors observed little difference between their two strategies for the empirical agent performance on a set of Atari games, and therefore opted for the simpler zero state strategy. One possible explanation for this is that in some cases (as we will see below), an LSTM tends to converge to a more 'typical' state if allowed a certain number of 'burn-in' steps, and so recovers from a bad initial recurrent state on a sufficiently long sequence. We also hypothesize that while the zero state strategy may suffice in the largely fully observable Atari domain, it prevents a recurrent network from learning actual long-term dependencies in more memory-critical domains (e.g. on DMLab).

To fix these issues, we propose and evaluate two strategies for training a recurrent neural network from randomly sampled replay sequences, that can be used individually or in combination:

- *Storing the recurrent state in replay and using it to initialize the network at training time.* This partially remedies the weakness of the zero start state strategy, however it may suffer from the effect of 'representational drift' leading to 'recurrent state staleness', as the stored recurrent state generated by a sufficiently old network could differ significantly from a typical state produced by a more recent version.
- *Allow the network a 'burn-in period' by using a portion of the replay sequence only for unrolling the network and producing a start state, and update the network only on the*

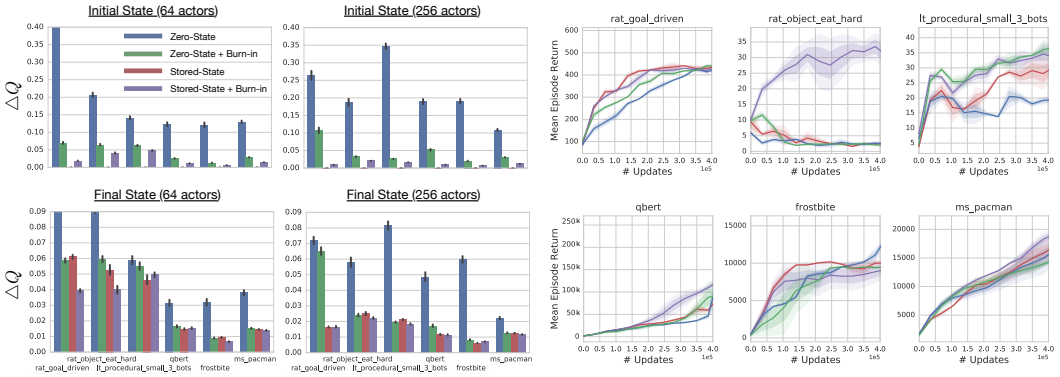


Figure 1: Left two columns: Q-value discrepancy $\Delta Q$ as a measure for recurrent state staleness, measured at first state (top) and last state (bottom) of replay sequences, for agents training on a selection of Atari and DMLab levels with different training strategies and numbers of actors. Right: Empirical agent performance of the 256-actor versions of the agents.

*remaining part of the sequence.* We hypothesize that this allows the network to (partially) recover from a poor start state (zero, or stored but stale) and find itself in a better initial state before being required to produce accurate value outputs.

In all our experiments we will be using the proposed agent architecture from Section 2.3 with replay sequences of length $m = 80$, with an optional burn-in prefix of $l = 40$ steps. Our aim is to assess the effect representational drift and recurrent state staleness and how they are affected by the different training strategies. For that we will compare the Q-values produced by the network on sampled replay sequences when unrolled using one of these strategies and the Q-values produced when using the true stored recurrent states at each step.

More formally, let $o_t, \ldots, o_{t+m}$ and $h_t, \ldots, h_{t+m}$ denote the replay sequence of observations and stored recurrent states, and denote by $h_{t+1} = h(o_t, h_t; \theta)$ and $q(o_t, h_t; \theta)$ the recurrent state and Q-value vector $(Q(o_t, a))_{a \in \mathcal{A}}$ output by the recurrent neural network with parameter vector $\theta$, respectively. We write $\hat{h}_t$ for the start state used in the training step determined by one of the above strategies (either 0, $h_t$ or the resulting recurrent state from unrolling the network with parameters $\hat{\theta}$ on the sequence prefix $o_{t-l}, \ldots, o_{t-1}$) and $\hat{h}_{t+1}, \ldots, \hat{h}_{t+m}$ (where $\hat{h}_{i+1} = h(o_i, \hat{h}_i; \hat{\theta})$) for the sequence of recurrent network states produced by the RNN at training time. We estimate the impact of representational drift and recurrent state staleness by their effect on the Q-value estimates, by measuring *Q-value discrepancy*

$$\Delta Q = \frac{\|q(o_{t+i}, \hat{h}_{t+i}; \hat{\theta}) - q(o_{t+i}, h_{t+i}; \hat{\theta})\|_2}{|\max_a (q(o_{t+i}, \hat{h}_{t+i}; \hat{\theta}))_a|}$$

for the first ($i = 0$) and last ($i = m - 1$) states of the replay sequence. The normalization by the maximal Q-value helps comparability between different environments and training stages, as the Q-value range of an agent can vary drastically between these. Note that we are not directly comparing the Q-values produced at acting and training time, $q(o_t, h_t; \theta)$ and $q(o_t, \hat{h}_t; \hat{\theta})$, as these can naturally be expected to be distinct as the agent is being trained, but instead focus on the difference that results from applying the same network (parameterized by $\hat{\theta}$) to the distinct recurrent states.

In Figure 1 (first two columns), we are comparing agents trained with the different strategies on several Atari and DMLab environments in terms of this proposed metric. It can be seen that the zero start state heuristic results in a significantly more severe effect of recurrent state staleness on the outputs of the network. As hypothesized above, this effect is greatly reduced for the last sequence states compared to the first ones, after the RNN has had time to 'recover' from the atypical start state, but the effect of staleness is still substantially worse here for the zero state than the stored state strategy.

Interestingly, the burn-in (from zero start state) strategy mitigates the staleness problem almost as effectively as the stored state strategy (though their combination works best). This is noteworthy,

as the only difference between the pure zero state and the burn-in strategy lies in the fact that the latter unrolls the network over a prefix of states on which the network does not receive updates. In informal experiments (not shown here) we verified that this is not due to the different unroll lengths themselves (i.e., that using just the zero state strategy on sequences of length $l + m$ would perform worse). We hypothesize that the beneficial effect of burn-in lies in the fact that it prevents 'destructive updates' to the RNN parameters resulting from it producing highly inaccurate initial outputs on the first few time steps after a zero state initialization. Another potential downside of the pure zero state heuristic is that it prevents the agent from strongly relying on its recurrent state and exploit long-term temporal dependencies, see Section 6.

On the right side of Figure 1, we compare the empirical performance of the training strategies on several Atari and DMLab tasks. As expected from the above observations, the comparison shows clear under-performance of zero start state compared to the other approaches in several of the tasks, and an overall advantage of the stored state strategy with burn-in. Note that, at least on the almost fully observable Atari environments, one cannot necessarily expect the different strategies to give rise to substantially different agent performances, as these environments should not strongly require the agent to use its recurrent memory effectively.

We conclude the section with the observation that both stored state and burn-in strategy provide substantial advantages over the naive zero state training strategy, in terms of (indirect) measures of the effect of representation drift and recurrent state staleness, and empirical performance. Since they combine beneficially, we are going to use both of these strategies in the empirical evaluation of our proposed agent in Section 5.

## 4 Effect of Distributed RL Agent Training

In this section, we investigate the effects of distributed training of an agent using a recurrent neural network, where a large number of actors feed their experience into a replay buffer for a single learner.

On the one hand, the distributed setting typically presents a less severe problem of representational drift than the single-actor case, such as the one studied in (Hausknecht & Stone, 2015). This is because in relative terms, the large amount of generated experience is replayed less frequently (on average, an experience sample is replayed less than once in the Ape-X agent, compared to eight times in DQN), and so distributed agent training tends to give rise to a smaller degree of 'parameter lag' (the mean age, in parameter updates, of the network parameters used to generate an experience, at the time it is being replayed).

On the other hand, the distributed setting allows for easy scaling of computational resources according to hardware or time constraints. An ideal distributed agent should therefore be robust to changes in, e.g., the number of actors, without careful parameter re-tuning. As we have seen in the previous section, RNN training from replay is sensitive to the issue of representational drift, the severity of which can depend on exactly these parameters.

To investigate these effects, we train our proposed agent architecture with a substantially smaller number of actors. This has a direct (inversely proportional) effect on the parameter lag. Specifically, in our experiments, as the number of actors is changed from $256$ to $64$, the mean parameter lag goes from $1500$ to approximately $5500$ parameter updates, which in turn impacts the magnitude of representation drift and recurrent state staleness, as measured by $\Delta Q$ in the previous section.

The left column in Figure 1 shows an overall increase of the average $\Delta Q$ for the smaller number of actors, both for first and last states of replayed sequences. This supports the above intuitions and highlights the increased importance of an improved training strategy (compared to the zero state strategy) in the distributed training setting, if an empirical level of agent performance is to be maintained across ranges of extrinsic and potentially hardware dependent parameters.

## 5 Experimental Evaluation

Based on our findings regarding RNN training in the distributed setting in the previous two sections, we chose to use the stored state strategy with burn-in period for training the R2D2 agent. In this

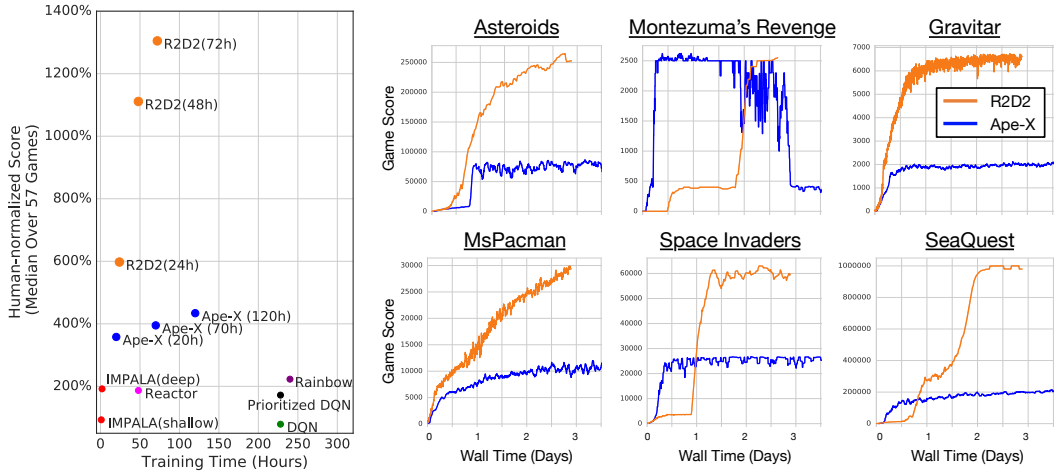


Figure 2: Atari-57 results. Left: median human-normalized scores and training times of various agent architectures. Diagram reproduced and extended from (Horgan et al., 2018). Right: Example individual learning curves of R2D2 and Ape-X.

section we evaluate the empirical performance of R2D2 on two challenging benchmark suites for deep reinforcement learning: Atari-57 (Bellemare et al., 2013) and DMLab-30 (Beattie et al., 2016).

One of the fundamental contributions of Deep Q-Networks (DQN) (Mnih et al., 2015) was to set as standard practice the use of a single network architecture and set of hyper-parameters across the entire suite of 57 Atari games. Unfortunately, expanding past Atari this standard has not been maintained and, to the best of our knowledge, at present there is no algorithm applied to both Atari-57 and DMLab-30 under this standard. In particular, we will compare performance with Ape-X and IMPALA for which hyper-parameters are tuned separately for each benchmark.

For R2D2, we use a single neural network architecture and a single set of hyper-parameters across all experiments. This demonstrates greater robustness and generality than has been previously observed in deep RL. It is also in pursuit of this generality, that we decided to disable the (Atari-specific) heuristic of treating life losses as episode ends, and did not apply reward clipping. Despite this, we observe state-of-the-art performance in both Atari and DMLab, validating the intuitions derived from our empirical study.

### 5.1 ATARI-57

The Atari-57 benchmark is built upon the Arcade Learning Environment (ALE) (Bellemare et al., 2013), and consists of 57 classic Atari 2600 video games. Initial human-level performance was achieved by DQN (Mnih et al., 2015), and since then RL agents have improved significantly through both algorithmic and architectural advances. Currently, state of the art for a single actor is achieved by the recent distributional reinforcement learning algorithms IQN (Dabney et al., 2018) and Rainbow (Hessel et al., 2018a), and for multi-actor results, Ape-X (Horgan et al., 2018).

Figure 2 (left) shows the median human-normalized scores across all games for R2D2 and related methods (see appendix for full Atari-57 scores and learning curves). R2D2 achieves an order of magnitude higher performance than all single-actor agents and triples the previous state-of-the-art performance of Ape-X using fewer actors (256 instead of 360), resulting in higher sample- and time-efficiency. Table 1 lists mean and median human-normalized scores for R2D2 and other algorithms, highlighting these improvements.

In addition to achieving state-of-the-art results on the entire task suite, R2D2 also achieves the highest ever reported agent scores on a large fraction of the individual Atari games. In Figure 2 (right) we highlight some of these individual learning curves of R2D2. As an example, notice the performance on MS.PACMAN is even greater than that of the agent reported in Van Seijen et al. (2017), which

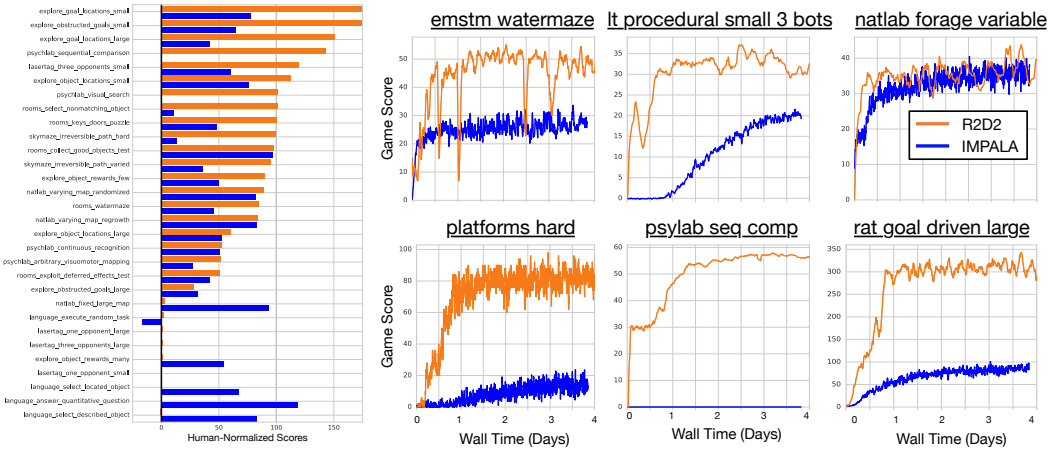


Figure 3: DMLab-30 comparison with IMPALA (left) final scores and (right) learning curves.

| | | Atari-57 | | DMLab-30 | |
|---|---|---|---|---|---|
| **Human-Normalized Score** | | **Median** | **Mean** | **Median** | **Mean-Capped** |
| Ape-X | (Horgan et al., 2018) | 358.1% | 1584.2% | N/A | N/A |
| Reactor | (Gruslys et al., 2018) | 187% | N/A | N/A | N/A |
| IMPALA, deep-experts | (Espeholt et al., 2018) | 191.8% | 957.6% | 49.0% | 45.8% |
| IMPALA, PBT | (Hessel et al., 2018b) | N/A | N/A | 77.9% | **61.5**% |
| R2D2 | | **1304.9%** | **3525.4%** | **84.3%** | 59.8% |

Table 1: Comparison of Atari-57 and DMLab-30 results. Unlike the IMPALA agent from (Espeholt et al., 2018), 'IMPALA, PBT' uses population-based training and the same improved action set from (Hessel et al., 2018b) as R2D2.

was engineered specifically for this game. Furthermore, we notice that Ape-X achieves super-human performance for the same number of games as Rainbow (49), and that its improvements came from improving already strong scores. R2D2 on the other hand is super-human on 52 out of 57 games, leaving only the five most challenging exploration problems with below human-level performance: MONTEZUMA'S REVENGE, PITFALL, SKIING, SOLARIS and PRIVATE EYE. We believe progress on the exploration problem combined with an agent such as R2D2, can in the near future reach super-human performance across the entire Atari-57 suite.

### 5.2 DMLAB-30

DMLab-30 is a suite of 30 problems set in a 3D first-person game engine, testing for a wide range of different challenges (Beattie et al., 2016). While Atari can largely be approached with only frame-stacking, DMLab-30 requires long-term memory to achieve reasonable performance. Perhaps because of this, and the difficulty of integrating recurrent state with experience replay, top-performing agents have, to date, always come in the form of actor-critic algorithms trained in (near) on-policy settings. For the first time we show state-of-the-art performance on DMLab-30 using a value-function-based agent.

Figure 3 (left) compares R2D2 with IMPALA. We again see R2D2 significantly out-performing related methods, while using the same set of hyper-parameters across all domains, and a significantly smaller neural network than for example IMPALA. Indeed, Table 1 that even when compared to the population-based training (PBT) version of IMPALA, R2D2 achieves better final median performance.

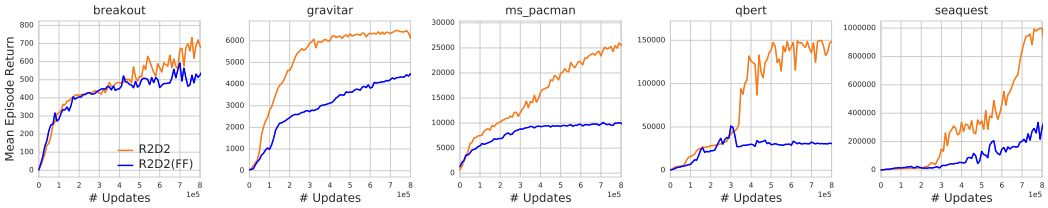


Figure 4: Ablation study: recurrent vs. feed-forward variant of R2D2.

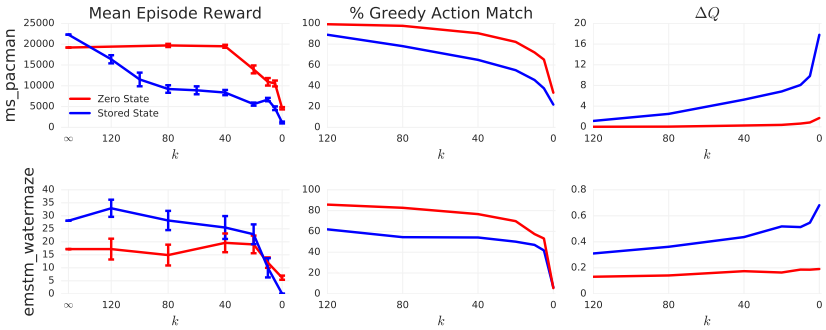


Figure 5: Effect of restricting R2D2's policy's memory on MS.PACMAN and EMSTM_WATERMAZE.

## 6 ANALYSIS OF AGENT PERFORMANCE

Atari-57 is a class of environments which are almost fully observable (given 4-frame-stack observations), and agents trained on it are not necessarily expected to benefit from a memory-augmented representation. The main algorithmic difference between R2D2 and its predecessor, Ape-X, is the use of a recurrent neural network, and it is therefore surprising by how large a margin R2D2 surpasses the previous state of the art on Atari. In this section we analyze the role of the LSTM network and our proposed training strategy for the high performance of the R2D2 agent.

Since the performance of asynchronous or distributed RL agents can depend on subtle implementational details and even factors such as precise hardware setup, it is impractical to perform a direct comparison to the Ape-X agent as reported in (Horgan et al., 2018). Instead, here we verify that the LSTM and its training strategy play a crucial role for the success of R2D2 by a comparison of the R2D2 agent with a purely feed-forward variant, all other parameters held fixed. This ablation in Figure 4 shows very clearly that the LSTM component is crucial for boosting the agent's peak performance as well as learning speed, explaining most of the performance difference to Ape-X.

In our next experiment we test to what extent the R2D2 agent relies on its memory, and how this is impacted by the different training strategies. For this we select the Atari game MS.PACMAN, on which R2D2 shows state-of-the-art performance despite the game being virtually fully observable, and the DMLab task EMSTM_WATERMAZE, which strongly requires the use of memory. We train two agents on each game, using the zero and stored state strategies, respectively. We then evaluate these agents by restricting their policy to a fixed history length: at time step $t$, their policy uses an LSTM unrolled over time steps $o_{t-k+1}, \ldots, o_t$, with the hidden state $h_{t-k}$ replaced by zero instead of the actual hidden state (note this is only done for evaluation, not at training time of the agents).

In Figure 5 (left) we decrease the history length $k$ from $\infty$ (full history) down to $0$ and show the degradation of agent performance (measured as mean score over 10 episodes) as a function of $k$. We additionally show the difference of max-Q-values and the percentage of correct greedy actions (where the unconstrained variant is taken as ground truth).

We first observe that restricting the agent's memory gradually decreases its performance, indicating its nontrivial use of memory on both domains. Crucially, while the agent trained with stored state

shows higher performance when using the full history, its performance decays much more rapidly than for the agent trained with zero start states. This is evidence that the zero start state strategy, used in past RNN-based agents with replay, limits the agent's ability to learn to make use of its memory. While this doesn't necessarily translate into a performance difference (like in MS.PACMAN), it does so whenever the task requires an effective use of memory (like EMSTM_WATERMAZE). This advantage of the stored state compared to the zero state strategy may explain the large performance difference between R2D2 and its close cousin Reactor (Gruslys et al., 2018), which trains its LSTM policy from replay with the zero state strategy.

Finally, the right and middle columns of Figure 5 show a monotonic decrease of the quality of Q-values and the resulting greedy policy as the available history length $k$ is decreased to $0$, providing a simple causal link between the constraint and the empirical agent performance.

## 7 CONCLUSIONS

Here we take a step back from evaluating performance and discuss our empirical findings in a broader context. There are two surprising findings in our results.

First, although zero state initialization was often used in previous works (Hausknecht & Stone, 2015; Gruslys et al., 2018), we have found that it leads to misestimated action-values, especially in the early states of replayed sequences. Moreover, without burn-in, updates through BPPT to these early time steps with poorly estimated outputs seem to give rise to destructive updates and hinder the network's ability to recover from suboptimal initial recurrent states. This suggests that either the context-dependent recurrent state should be stored along with the trajectory in replay, or an initial part of replayed sequences should be reserved for burn-in, to allow the RNN to rely on its recurrent state and exploit long-term temporal dependencies, and the two techniques can also be combined beneficially. We have also observed that the underlying problems of representational drift and recurrent state staleness are potentially exacerbated in the distributed setting, highlighting the importance of robustness to these effects through an adequate training strategy of the RNN.

Second, we found that the impact of LSTM training goes beyond providing the agent with memory. Instead, LSTM training also serves a role not previously studied in RL, potentially by enabling better representation learning, and thereby improves performance even on domains that are fully observable and don't obviously require memory.

Finally, taking a broader view on our empirical results, we note that scaling up of RL agents through parallelization and distributed training allows them to benefit from huge experience throughput and achieve ever-increasing results over broad simulated task suites such as Atari-57 and DMLab-30. Impressive as these results are in terms of raw performance, they come at the price of high sample complexity, consuming billions of simulated time steps in hours or days of wall-clock time. One widely open avenue for future work lies in improving the sample efficiency of these agents, to allow applications to domains that do not easily allow fast simulation at similar scales. Another remaining challenge, very apparent in our results on Atari-57, is exploration: Save for the $5$ hardest-exploration games from the Atari-57, R2D2 surpasses human-level performance on this task suite significantly, essentially 'solving' many of the games therein.

## REFERENCES

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. DeepMind Lab. *arXiv preprint arXiv:1612.03801*, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: an evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Will Dabney, Georg Ostrovski, David Silver, and Remi Munos. Implicit quantile networks for distributional reinforcement learning. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1096–1105, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Audrunas Gruslys, Will Dabney, Mohammad Gheshlaghi Azar, Bilal Piot, Marc Bellemare, and Remi Munos. The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkHVZWZAZ.

Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. *CoRR, abs/1507.06527*, 7(1), 2015.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018a.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. *arXiv preprint arXiv:1809.04474*, 2018b.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Tommi Jaakkola, Satinder P Singh, and Michael I Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in neural information processing systems*, pp. 345–352, 1995.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv preprint arXiv:1807.01281*, 2018.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

George E Monahan. State of the arta survey of partially observable markov decision processes: theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

OpenAI. Learning dexterous in-hand manipulation. *arXiv preprint: arxiv:1808.00177*, 2018a.

OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018b.

Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado Van Hasselt, John Quan, Mel Vecerik, Matteo Hessel, Remi Munos, and Olivier Pietquin. Observe and look further: Achieving consistent performance on atari. *arXiv preprint: arxiv:1805.11593*, 2018.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 5392–5402, 2017.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

# APPENDIX

## HYPER-PARAMETERS

R2D2 uses the same 3-layer convolutional network as DQN (Mnih et al., 2015), followed by an LSTM with $512$ hidden units, which feeds into the advantage and value heads of a dueling network (Wang et al., 2016), each with a hidden layer of size $512$.

| | |
|---|---|
| Number of actors | $256$ |
| Actor parameter update interval | $400$ environment steps |
| Sequence length $m$ | $80$ (+ prefix of $l = 40$ in all burn-in experiments) |
| Replay buffer size | $4 \times 10^6$ observations ($10^5$ part-overlapping sequences) |
| Priority exponent | $0.9$ |
| Discount $\gamma$ | $0.997$ |
| Minibatch size | $64$ |
| Optimizer | Adam (Kingma & Ba, 2014) |
| Optimizer settings | learning rate $= 10^{-4}$, $\varepsilon = 10^{-4}$ |
| Target network update interval | $2500$ updates |

Table 2: Hyper-parameters values used in R2D2. All missing parameters follow the ones in Ape-X (Horgan et al., 2018).

FULL RESULTS

| GAMES | IMPALA(DEEP) | R2D2 |
|---|---|---|
| explore_goal_locations_large | 83.1 | **292.4** |
| explore_goal_locations_small | 209.4 | **469.2** |
| explore_object_locations_large | 37.0 | **41.48** |
| explore_object_locations_small | 57.8 | **83.36** |
| explore_object_rewards_few | 39.8 | **70.0** |
| explore_object_rewards_many | **58.7** | 3.36 |
| explore_obstructed_goals_large | **39.5** | 35.2 |
| explore_obstructed_goals_small | 135.2 | **354.4** |
| language_answer_quantitative_question | **219.4** | -0.4 |
| language_execute_random_task | -49.9 | **0.12** |
| language_select_described_object | **324.6** | -3.6 |
| language_select_located_object | **189.0** | 2.0 |
| lasertag_one_opponent_large | -0.2 | **0.0** |
| lasertag_one_opponent_small | -0.1 | **0.0** |
| lasertag_three_opponents_large | -0.1 | **0.08** |
| lasertag_three_opponents_small | 19.1 | **37.64** |
| natlab_fixed_large_map | **34.7** | 3.16 |
| natlab_varying_map_randomized | 36.1 | **38.56** |
| natlab_varying_map_regrowth | 20.7 | **20.88** |
| psychlab_arbitrary_visuomotor_mapping | 16.4 | **30.64** |
| psychlab_continuous_recognition | 29.9 | **30.72** |
| psychlab_sequential_comparison | 0.0 | **56.6** |
| psychlab_visual_search | 0.0 | **79.8** |
| rooms_collect_good_objects_test | 9.0 | **9.76** |
| rooms_exploit_deferred_effects_test | 15.6 | **48.0** |
| rooms_keys_doors_puzzle | 28.0 | **54.08** |
| rooms_select_nonmatching_object | 7.3 | **66.96** |
| rooms_watermaze | 26.9 | **46.52** |
| skymaze_irreversible_path_hard | 13.6 | **100.0** |
| skymaze_irreversible_path_varied | 45.1 | **96.0** |

Table 3: Performance of R2D2 compared to IMPALA on the DMLab-30 suite of tasks. Note that the IMPALA agent does not use the improved action set from (Hessel et al., 2018b) that R2D2 uses, but on the other hand IMPALA uses substantially larger 15-layer ResNet, and importantly a custom 'optimistic asymmetric reward clipping' which yields a big advantage in the LANGUAGE_* tasks.

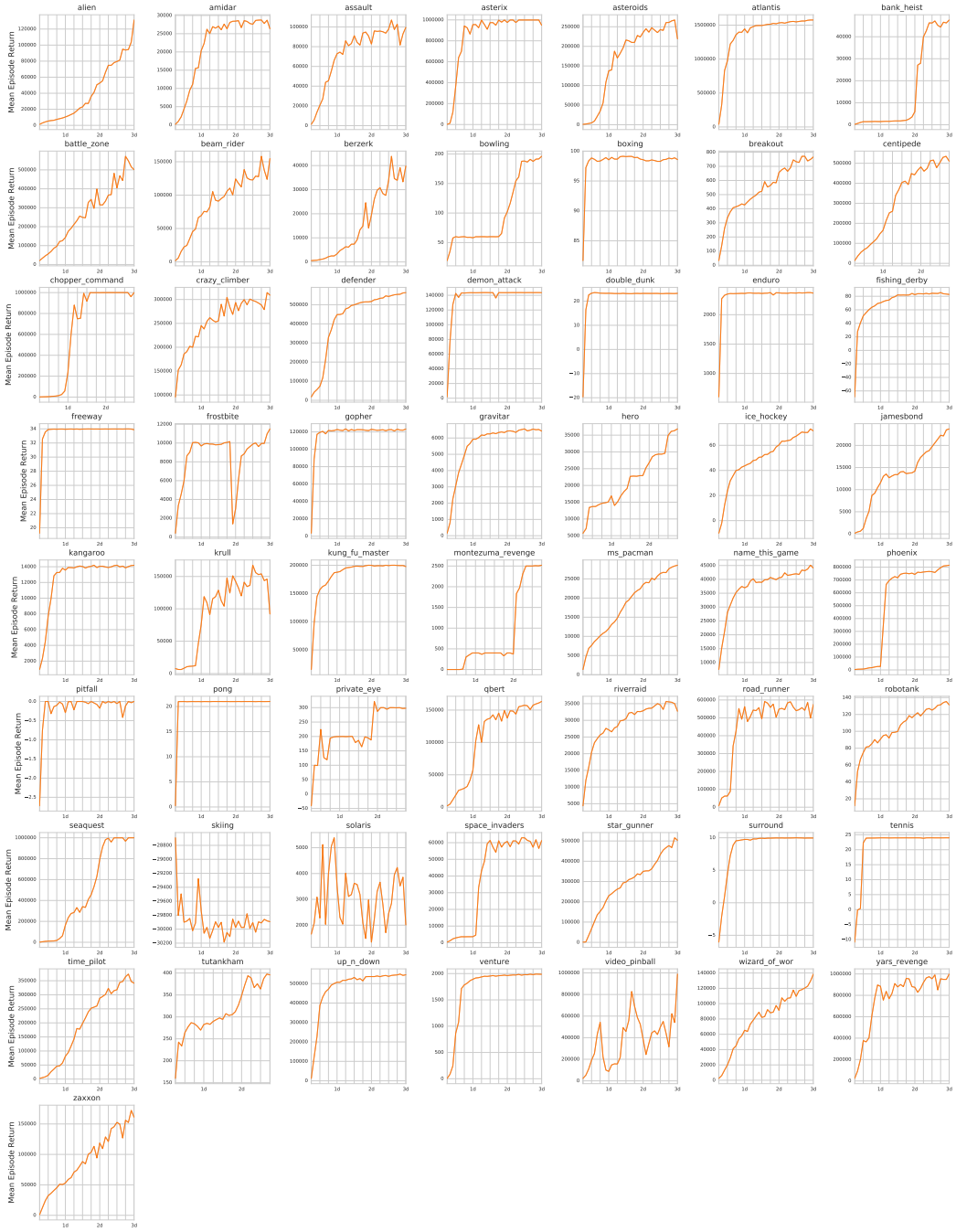


Figure 6: R2D2 learning curves on 57 Atari games.

| GAMES | HUMAN | REACTOR | IMPALA(S/D) | APE-X | R2D2 |
|---|---|---|---|---|---|
| alien | 7127.8 | 6482.1 | 1536.0/15962.1 | 40804.9 | **109038.4** |
| amidar | 1719.5 | 833.0 | 497.6/1554.8 | 8659.2 | **27751.24** |
| assault | 742.0 | 11013.5 | 12086.9/19148.5 | 24559.4 | **90526.44** |
| asterix | 8503.3 | 36238.5 | 29692.5/300732.0 | 313305.0 | **999080.0** |
| asteroids | 47388.7 | 2780.3 | 3508.1/108590.1 | 155495.1 | **265861.2** |
| atlantis | 29028.1 | 308257.5 | 773355.5/849967.5 | 944497.5 | **1576068.0** |
| bank_heist | 753.1 | 988.7 | 1200.3/1223.2 | 1716.4 | **46285.6** |
| battle_zone | 37187.5 | 61220.0 | 13015.0/20885.0 | 98895.0 | **513360.0** |
| beam_rider | 16926.5 | 8566.5 | 8219.9/32463.5 | 63305.2 | **128236.08** |
| berzerk | 2630.4 | 1641.4 | 888.3/1852.7 | **57196.7** | 34134.8 |
| bowling | 160.7 | 75.4 | 35.7/59.9 | 17.6 | **196.36** |
| boxing | 12.1 | 99.4 | 96.3/**100.0** | **100.0** | 99.16 |
| breakout | 30.5 | 518.4 | 640.4/787.3 | **800.9** | 795.36 |
| centipede | 12017.0 | 3402.8 | 5528.1/11049.8 | 12974.0 | **532921.84** |
| chopper_command | 7387.8 | 37568.0 | 5012.0/28255.0 | 721851.0 | **960648.0** |
| crazy_climber | 35829.4 | 194347.0 | 136211.5/136950.0 | **320426.0** | 312768.0 |
| defender | 18688.9 | 113127.8 | 58718.3/185203.0 | 411943.5 | **562106.0** |
| demon_attack | 1971.0 | 100188.5 | 107264.7/132827.0 | 133086.4 | **143664.6** |
| double_dunk | -16.4 | 11.4 | -0.4/-0.3 | 12.8 | **23.12** |
| enduro | 860.5 | 2230.1 | 0.0/0.0 | 2177.4 | **2376.68** |
| fishing_derby | -38.8 | 23.2 | 32.1/44.9 | 44.4 | **81.96** |
| freeway | 29.6 | 31.4 | 0.0/0.0 | 33.7 | **34.0** |
| frostbite | 4334.7 | 8042.1 | 269.6/317.8 | 9328.6 | **11238.4** |
| gopher | 2412.5 | 69135.1 | 1002.4/66782.3 | 120500.9 | **122196.0** |
| gravitar | 3351.4 | 1073.8 | 211.5/359.5 | 1598.5 | **6750.0** |
| hero | 30826.4 | 35542.2 | 33853.2/33730.6 | 31655.9 | **37030.4** |
| ice_hockey | 0.9 | 3.4 | -5.3/3.5 | 33.0 | **71.56** |
| jamesbond | 302.8 | 7869.3 | 440.0/601.5 | 21322.5 | **23266.0** |
| kangaroo | 3035.0 | 10484.5 | 47.0/1632.0 | 1416.0 | **14112.0** |
| krull | 2665.5 | 9930.9 | 9247.6/8147.4 | 11741.4 | **145284.8** |
| kung_fu_master | 22736.3 | 59799.5 | 42259.0/43375.5 | 97829.5 | **200176.0** |
| montezuma_revenge | **4753.3** | 2643.5 | 0.0/0.0 | 2500.0 | 2504.0 |
| ms_pacman | 6951.6 | 2724.3 | 6501.7/7342.3 | 11255.2 | **29928.2** |
| name_this_game | 8049.0 | 9907.1 | 6049.6/21537.2 | 25783.3 | **45214.8** |
| phoenix | 7242.6 | 40092.3 | 33068.2/210996.5 | 224491.1 | **811621.6** |
| pitfall | **6463.7** | -3.5 | -11.1/-1.7 | -0.6 | 0.0 |
| pong | 14.6 | 20.7 | 20.4/**21.0** | 20.9 | **21.0** |
| private_eye | **69571.3** | 15177.1 | 92.4/98.5 | 49.8 | 300.0 |
| qbert | 13455.0 | 22956.5 | 18901.3/**351200.1** | 302391.3 | 161000.0 |
| riverraid | 17118.0 | 16608.3 | 17401.9/29608.0 | **63864.4** | 34076.4 |
| road_runner | 7845.0 | 71168.0 | 37505.0/57121.0 | 222234.5 | **498660.0** |
| robotank | 11.9 | 68.5 | 2.3/13.0 | 73.8 | **132.4** |
| seaquest | 42054.7 | 8425.8 | 1716.9/1753.2 | 392952.3 | **999991.84** |
| skiing | **-4336.9** | -10753.4 | -29975.0/-10180.4 | -10789.9 | -29970.32 |
| solaris | **12326.7** | 2165.2 | 2368.4/2365.0 | 2892.9 | 4198.4 |
| space_invaders | 1668.7 | 2448.6 | 1726.3/43595.8 | 54681.0 | **55889.0** |
| star_gunner | 10250.0 | 70038.0 | 69139.0/200625.0 | 434342.5 | **521728.0** |
| surround | 6.5 | 6.7 | -8.1/7.6 | 7.1 | **9.96** |
| tennis | -8.3 | 23.3 | -1.9/0.5 | 23.9 | **24.0** |
| time_pilot | 5229.1 | 19401.0 | 6617.5/48481.5 | 87085.0 | **348932.0** |
| tutankham | 167.6 | 272.6 | 267.8/292.1 | 272.6 | **393.64** |
| up_n_down | 11693.2 | 64354.3 | 273058.1/332546.8 | 401884.3 | **542918.8** |
| venture | 1187.5 | 1597.5 | 0.0/0.0 | 1773.5 | **1992.0** |
| video_pinball | 17667.9 | 469365.8 | 228642.5/**572898.3** | 546197.4 | 483569.72 |
| wizard_of_wor | 4756.5 | 13170.5 | 4203.0/9157.5 | 46204.0 | **133264.0** |
| yars_revenge | 54576.9 | 102759.8 | 80530.1/84231.1 | 148594.8 | **918854.32** |
| zaxxon | 9173.3 | 25215.5 | 1148.5/32935.5 | 42285.5 | **181372.0** |

Table 4: Performance of R2D2 compared to Reactor, IMPALA (shallow and deep expert variants), and Ape-X, 30 no-op starts.