Journal of Artificial Intelligence Research 61 (2018) 523-562 Submitted 9/17; published 3/18

Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents

Marlos C. Machado MACHADO@UALBERTA.CA
University of Alberta, Edmonton, Canada

Marc G. Bellemare BELLEMARE@GOOGLE.COM
Google Brain, Montréal, Canada

Erik Talvitie ERIK.TALVITIE@FANDM.EDU
Franklin & Marshall College, Lancaster, USA

Joel Veness AIXI@GOOGLE.COM
DeepMind, London, United Kingdom

Matthew Hausknecht MATTHEW.HAUSKNECHT@MICROSOFT.COM
Microsoft Research, Redmond, USA

Michael Bowling MBOWLING@UALBERTA.CA
University of Alberta, Edmonton, Canada
DeepMind, Edmonton, Canada

Abstract

The Arcade Learning Environment (ALE) is an evaluation platform that poses the challenge of building AI agents with general competency across dozens of Atari 2600 games. It supports a variety of different problem settings and it has been receiving increasing attention from the scientific community, leading to some high-profile success stories such as the much publicized Deep Q-Networks (DQN). In this article we take a big picture look at how the ALE is being used by the research community. We show how diverse the evaluation methodologies in the ALE have become with time, and highlight some key concerns when evaluating agents in the ALE. We use this discussion to present some methodological best practices and provide new benchmark results using these best practices. To further the progress in the field, we introduce a new version of the ALE that supports multiple game modes and provides a form of stochasticity we call sticky actions. We conclude this big picture look by revisiting challenges posed when the ALE was introduced, summarizing the state-of-the-art in various problems and highlighting problems that remain open.

1. Introduction

The Arcade Learning Environment (ALE) is both a challenge problem and a platform for evaluating general competency in artificial intelligence (AI). Originally proposed by Bellemare, Naddaf, Veness, and Bowling (2013), the ALE makes available dozens of Atari 2600 games for agent evaluation. The agent is expected to do well in as many games as possible without game-specific information, generally perceiving the world through a video stream. Atari 2600 games are excellent environments for evaluating AI agents for three main reasons: 1) they are varied enough to provide multiple different tasks, requiring general competence, 2) they are interesting and challenging for humans, and 3) they are free of experimenter's bias, having been developed by an independent party.

The usefulness of the ALE is reflected in the amount of attention it has received from the scientific community. The number of papers using the ALE as a testbed has exploded in recent years. This has resulted in some high-profile success stories, such as the much publicized Deep Q-Networks (DQN), the first algorithm to achieve human-level control in a large fraction of Atari 2600 games (Mnih et al., 2015). This interest has also led to the first dedicated workshop on the topic, the AAAI Workshop on Learning for General Competency in Video Games (Albrecht et al., 2015). Several of the ideas presented in this article were first discussed at this workshop, such as the need for standardizing evaluation and for distinguishing open-loop behaviours from closed-loop ones.

Given the ALE's increasing importance in the AI literature, this article aims to be a "check-up" for the Arcade Learning Environment, taking a big picture look at how the ALE is being used by researchers. The primary goal is to highlight some subtle issues that are often overlooked and propose some small course corrections to maximize the scientific value of future research based on this testbed. The ALE has incentivized the AI community to build more generally competent agents. The lessons learned from that experience may help further that progress and may inform best practices as other testbeds for general competency are developed (e.g., Beattie et al., 2016; Brockman et al., 2016; Johnson, Hofmann, Hutton, & Bignell, 2016; Levine et al., 2013).

The main contributions of this article are: 1) To discuss the different evaluation methods present in the literature and to identify, for the typical reinforcement learning setting, some methodological best practices gleaned from experience with the ALE (Sections 3 and 4). 2) To address concerns regarding the deterministic dynamics of previous versions of the platform, by introducing a new version of the ALE that supports a form of stochasticity we call *sticky actions* (Section 5). 3) To provide new benchmark results in the reinforcement learning setting that ease comparison and reproducibility of experiments in the ALE. These benchmark results also encourage the development of sample efficient algorithms (Section 6). 4) To revisit challenges posed when the ALE was introduced, summarizing the state-of-the-art in various problems and highlighting problems that are currently open (Section 7). 5) To introduce a new feature to the platform that allows existent environments to be instantiated in multiple difficult levels and game modes (Section 7.4.1)

2. Background

In this section we introduce the formalism behind reinforcement learning (Sutton & Barto, 1998), as well as how it is instantiated in the Arcade Learning Environment. We also present the two most common value function representations used in reinforcement learning for Atari 2600 games: linear approximation and neural networks. As a convention, we indicate scalar-valued random variables by capital letters (e.g., S_t, R_t), vectors by bold lowercase letters (e.g., $\boldsymbol{\theta}, \boldsymbol{\phi}$), functions by non-bold lowercase letters (e.g., v, q), and sets with a calligraphic font (e.g., $\mathcal{S}, \mathcal{A}$).

2.1 Setting

We consider an agent interacting with its environment in a sequential manner, aiming to maximize cumulative reward. It is often assumed that the environment satisfies the Markov property and is modeled as a Markov decision process (MDP). An MDP is formally defined

as a 4-tuple $(\mathcal{S}, \mathcal{A}, p, r)$. Starting from state $S_0 \in \mathcal{S}$, at each step the agent takes an action $A_t \in \mathcal{A}$, to which the environment responds with a state $S_{t+1} \in \mathcal{S}$, according to a transition probability kernel $p(s' \mid s, a) \doteq \Pr(S_{t+1} = s' \mid S_t = s, A_t = a)$, and with a reward $R_{t+1} \in \mathbb{R}$, where $r(s, a)$ indicates the expected reward for a transition from state s under action a, that is, $r(s, a) \doteq \mathbb{E}[R_t \mid S_t = s, A_t = a]$.

In the context of the ALE, an action is the composition of a joystick direction and an optional button press. The agent observes a reward signal, which is typically the change in the player's score (the difference in score between the previous time step and the current time step), and an observation $O_t \in \mathcal{O}$ of the environment. This observation can be a single 210×160 image and/or the current 1024-bit RAM state. Because a single image typically does not satisfy the Markov property, we distinguish between observations and the environment state, with the RAM data being the real state of the emulator.[1] A frame (as a unit of time) corresponds to 1/60th of a second, the time interval between two consecutive images rendered to the television screen. The ALE is deterministic: given a particular emulator state, s, and a joystick input, a, there is a unique resulting next state, s'. In other words, $p(s' \mid s, a) = 1$. We will return to this important characteristic in Section 5.

Agents interact with the ALE in an episodic fashion. An episode begins by resetting the ALE to its initial configuration, and ends at a natural endpoint of a game's playthrough (this often corresponds to the player losing their last life). The primary measure of an agent's performance is the score achieved during an episode, namely the undiscounted sum of rewards for that episode. While this performance measure is quite natural, it is important to realize that score, in and of itself, is not necessarily an indicator of AI progress. In some games, agents can maximize their score by "getting stuck" in a loop of "small" rewards, ignoring what human players would consider to be the game's main goal. Nevertheless, score is currently the most common measure of agent performance so we focus on it here.

Beyond the minimal interface described above, almost all agents designed for the ALE implement some form of reward normalization. The magnitude of rewards can vary wildly across games; transforming the reward to fit into a roughly uniform scale makes it more feasible to find game-independent meta-parameter settings. For instance, some agents divide every reward by the magnitude of the first non-zero reward value encountered, implicitly assuming that the first non-zero reward is "typical" (Bellemare, Naddaf, et al., 2013). Others account only for the sign of the reward, replacing each reward value with -1, 0, or 1, accordingly (Mnih et al., 2015). Most agents also employ some form of hard-coded preprocessing to simplify the learning and acting process. We briefly review the three most common preprocessing steps as they will play a role in the subsequent discussion. 1) *Frame skipping* (Naddaf, 2010) restricts the agent's decision points by repeating a selected action for k consecutive frames. Frame skipping results in a simpler reinforcement learning problem and speeds up execution; values of $k = 4$ and $k = 5$ have been commonly used in the literature. 2) *Color averaging* (Bellemare, Naddaf, et al., 2013) and *frame pooling* (Mnih et al., 2015) are two image-based mechanisms to flatten two successive frames into a single one in order to reduce visual artifacts resulting from limitations of the Atari 2600 hardware – by leveraging the slow decay property of phosphors on 1970s televisions, objects on the screen could be displayed every other frame without compromising the game's visual aspect (Montfort

[1] The internal emulator state also includes registers and timers, but the RAM information and joystick inputs are sufficient to infer the next emulator state.

& Bogost, 2009). Effectively, color averaging and frame pooling remove the most benign form of partial observability in the ALE. Finally, 3) *frame stacking* (Mnih et al., 2015) concatenates previous frames with the most recent in order to construct a richer observation space for the agent. Frame stacking also reduces the degree of partial observability in the ALE, making it possible for the agent to detect the direction of motion in objects.

2.2 Control in the Arcade Learning Environment

The typical goal of reinforcement learning (RL) algorithms is to learn a *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maps each state to a probability distribution over actions. Ideally, following the learned policy will maximize the discounted cumulative sum of rewards.[2] Many RL algorithms accomplish this by learning an *action-value* function $q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which encodes the long-range value of taking action a in state s and then following policy π thereafter. More specifically, $q_\pi(s, a) \doteq \mathbb{E}_{\pi,p}\left[\sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i} \mid S_t = s, A_t = a\right]$, the expected discounted sum of rewards for some discount factor $\gamma \in [0, 1]$, where the expectation is with respect to both the policy π and the probability kernel p. However, in the ALE it is not feasible to learn an individual value for each state-action pair due to the large number of possible states. A common way to address this issue is to approximate the action-value function by parameterizing it with a set of weights $\boldsymbol{\theta}$. We write $\hat{q}(s, a, \boldsymbol{\theta}) \approx q_\pi(s, a)$ for the approximate value of the state-action given the weights $\boldsymbol{\theta}$. We discuss below two approaches to value function approximation that have been successfully applied to the games available in the ALE. We focus on these particular methods because they are by now well-established, well-understood, achieve a reasonable level of performance, and reflect the issues we study here.

The first approach is to design a function that, given an observation, outputs a vector, $\boldsymbol{\phi}(s, a)$, that denotes a feature-based representation of the state s when taking action a. With this approach, q_π is estimated through a linear function approximator such that $\hat{q}(s, a, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s, a)$. Sarsa($\lambda$) (Rummery & Niranjan, 1994) is a control algorithm that learns an approximate action-value function of a sequence of improving policies. As states are visited and rewards are observed, $\hat{q}$ is updated and an improved policy, which the agent then follows, is obtained from these estimates. The update equations are:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_{t+1}, A_{t+1}) - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_t, A_t), \\ \boldsymbol{e}_t &= \gamma\lambda \boldsymbol{e}_{t-1} + \boldsymbol{\phi}(S_t, A_t), \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \delta_t \boldsymbol{e}_t, \end{aligned}$$

where α denotes the step-size and $\boldsymbol{e}_t$ the eligibility trace vector ($\boldsymbol{e}_{-1} \doteq \mathbf{0}$). The first benchmarks in the ALE applied this approach with a variety of simple feature representations (e.g., Bellemare, Naddaf, et al., 2013; Bellemare, Veness, & Bowling, 2012b; Naddaf, 2010). Recently, Liang, Machado, Talvitie, and Bowling (2016) introduced a feature representation (Blob-PROST) that allows Sarsa(λ) to achieve comparable performance to DQN (described below) in several Atari 2600 games. We refer to such an approach as Sarsa(λ) + Blob-

[2] We use the *discounted* sum of rewards in our formalism because this is commonly employed by agents in the ALE. Empirical evidence has shown that agents generally perform better when maximizing the *discounted* cumulative sum of rewards, even though they are actually evaluated in the *undiscounted* case. This formulation disincentivizes agents to postpone scoring.

PROST. Recently, Martin, Sasikumar, Everitt, and Hutter (2017) combined Sarsa(λ) and the Blob-PROST features with a method for incentivizing exploration in hard games.

A recent trend in reinforcement learning is to use neural networks to estimate $q_\pi(s, a)$, substituting the requirement of a good handcrafted feature representation with the requirement of an effective network architecture and algorithm. Mnih et al. (2015) introduced Deep Q-Networks (DQN), an algorithm that learns representations in a neural network composed of three hidden convolutional layers followed by a fully-connected hidden layer. The network weights are updated through backpropagation with the following update rule:

$$\boldsymbol{\theta}_{t+1} \quad = \quad \boldsymbol{\theta}_t + \alpha\Big[R_{t+1} + \gamma \max_{a \in \mathcal{A}} q(S_{t+1}, a, \boldsymbol{\theta}_t^-) - q(S_t, A_t, \boldsymbol{\theta}_t)\Big]\nabla_{\boldsymbol{\theta}_t}\ q(S_t, A_t, \boldsymbol{\theta}_t),$$

where $\boldsymbol{\theta}_t^-$ denotes the weights of a duplicate network, which are updated less often for stability purposes:

$$\boldsymbol{\theta}_t^- = \begin{cases} \boldsymbol{\theta}_t, & \text{if } t \bmod C = 0, \\ \boldsymbol{\theta}_{t-1}^-, & \text{otherwise,} \end{cases}$$

with C being a parameter, the target network update frequency. Additional components of the algorithm include clipping the rewards (as described above) and the use of experience replay (Lin, 1993) to decorrelate observations. DQN has inspired much follow-up work combining reinforcement learning and deep neural networks (e.g., Jaderberg et al., 2017; Mnih et al., 2016; Schaul, Quan, Antonoglou, & Silver, 2016; van Hasselt, Guez, & Silver, 2016).

3. Divergent Evaluation Methodologies in the ALE

The ALE has received significant attention since it was introduced as a platform to evaluate general competency in AI. Hundreds of papers have used the ALE as a testbed, employing many distinct experimental protocols for evaluating agents. Unfortunately, these different evaluation protocols are often not carefully distinguished, making direct comparisons difficult or misleading. In this section we discuss a number of methodological differences that have emerged in the literature. In subsequent sections we give special focus to two particularly important methodological issues: 1) *different metrics for summarizing agent performance*, and 2) *different mechanisms for injecting stochasticity in the environment.*

The discussion about the divergence of evaluation protocols and the need for standardizing them first took place at the AAAI Workshop on Learning for General Competency in Video Games. One of the reasons that authors compare results generated with differing experimental protocols is the high computational cost of evaluating algorithms in the ALE – it is difficult to re-evaluate existing approaches to ensure matching methodologies. For that reason it is perhaps especially important to establish a standard methodology for the ALE in order to reduce the cost of principled comparison and analysis. One of the main goals of this article is to propose such a standard, and to introduce benchmark results obtained under it for straightforward comparison to future work.

3.1 Methodological Differences

To illustrate the diversity in evaluation protocols, we discuss some methodological differences found in the literature. While these differences may be individually benign, they

are frequently ignored when comparing results, which undermines the validity of direct comparisons.

3.1.1 Episode Termination

In the initial ALE benchmark results (Bellemare, Naddaf, et al., 2013), episodes terminate when the game is over. However, in some games the player has a number of "lives" which are lost one at a time. Terminating only when the game is over often makes it difficult for agents to learn the significance of losing a life. Mnih et al. (2015) terminated training episodes when the agent lost a life, rather than when the game is over (evaluation episodes still lasted for the entire game). While this approach has the potential to teach an agent to avoid "death," Bellemare et al. (2016b) noted that it can in fact be detrimental to an agent's performance. Currently, both approaches are still common in the literature. We often see episodes terminating when the game is over (e.g., Hausknecht, Lehman, Miikkulainen, & Stone, 2014; Liang et al., 2016; Lipovetzky, Ramirez, & Geffner, 2015; Martin et al., 2017), as well as when the agent loses a life (e.g., Nair et al., 2015; Schaul et al. 2016; van Hasselt et al., 2016). Considering the ideal of minimizing the use of game-specific information and the questionable utility of termination using the "lives" signal, *we recommend that only the game over signal be used for termination.*

3.1.2 Setting of Hyperparameters

One of the primary goals of the ALE is to enable the evaluation of agents' general ability to learn in complex, high-dimensional decision-making problems. Ideally agents would be evaluated in entirely novel problems to test their generality, but this is of course impractical. With only 60 available games in the standard suite there is a risk that methods could "overfit" to the finite set of problems. In analogy to typical methodology in supervised learning, Bellemare, Naddaf, et al. (2013) split games into "training" and "test" sets, only using results from training games for the purpose of selecting hyperparameters, then fully evaluating the agent in the test games only once hyperparameters have been selected. This methodology has been inconsistently applied in subsequent work – for example, hyperparameters are sometimes selected using the entire suite of games, and in some cases hyperparameters are optimized on a per-game basis (e.g., Jaderberg et al., 2017).[3] For the sake of evaluating generality, *we advocate for a train/test game split as a way to evaluate agents in problems they were not specifically tuned for.*

3.1.3 Measuring Training Data

The first benchmarks in the ALE (Bellemare, Naddaf, et al., 2013) trained agents for a fixed number of episodes before evaluating them. This can be misleading since episode lengths differ from game to game. Worse yet, in many games the better an agent performs the longer episodes last. Thus, under this methodology, agents that learn a good policy early receive more training data overall than those that learn more slowly, potentially magnifying

[3]The methodology based on the train/test split, as well as most of the other methodologies applied to the ALE, assume that hyperparameters can be tuned without consequences. Therefore, despite being a very interesting problem, in this paper we will not discuss the setting in which the cost associated with finding hyperparameters is taken into consideration.

their differences. Recently it has become more common to measure the amount of training data in terms of the total number of frames experienced by the agent (Mnih et al., 2015), which aids reproducibility, inter-game analysis, and fair comparisons. That said, since performance is measured on a per-episode basis, it may not be advisable to end training in the middle of an episode. For example, Mnih et al. (2015) interrupt the training as soon as the maximum number of frames is reached, while Liang et al. (2016) pick a total number of training frames, and then train each agent until the end of the episode in which the total is exceeded. This typically results in a negligible number of extra frames of experience beyond the limit. Another important aspect to be taken into consideration is frame skipping, which is a common practice in the ALE but is not reported consistently in the literature. *We advocate evaluating from full training episodes from a fixed number of frames*, as was done by Liang et al. (2016), and *we advocate taking the number of skipped frames into consideration when measuring training data*, as the time scale in which the agent operates is also an algorithmic choice.

3.1.4 Summarizing Learning Performance

When evaluating an agent in 60 games, it becomes necessary to compactly summarize the agent's performance in each game in order to make the results accessible and to facilitate comparisons. Authors have employed various statistics for summarizing agent performance and this diversity makes it difficult to directly compare reported results. *We recommend reporting training performance at different intervals during learning.* We discuss this issue in more detail in Section 4.

3.1.5 Injecting Stochasticity

The original Atari 2600 console had no source of entropy for generating pseudo-random numbers. The Arcade Learning Environment is also fully deterministic – each game starts in the same state and outcomes are fully determined by the state and the action. As such, it is possible to achieve high scores by learning an open-loop policy, i.e., by simply memorizing a good action sequence, rather than learning to make good decisions in a variety of game scenarios (Bellemare, Naddaf, Veness, & Bowling, 2015). Various approaches have been developed to add forms of stochasticity to the ALE dynamics in order to encourage and evaluate robustness in agents (e.g., Brockman et al., 2016; Hausknecht & Stone, 2015; Mnih et al., 2015; Nair et al., 2015). *Our recommendation is to use* sticky actions*, implemented in the latest version of the ALE.* We discuss this issue in more detail in Section 5.

4. Summarizing Learning Performance

One traditional goal in reinforcement learning is for agents to continually improve their performance as they obtain more data (Hutter, 2005; Ring, 1997; Singh, Barto, & Chentanez, 2004; Sutton et al., 2011; Thrun & Mitchell, 1993; Wilson, 1985). Measuring the extent to which this is the case for a given agent can be a challenge, and this challenge is exacerbated in the Arcade Learning Environment, where the agent is evaluated across 60 games. When evaluating an agent in only a few problems, it is common practice to plot learning curves, which provide a rich description of the agent's performance: how quickly it

learns, the highest performance it attains, the stability of its solutions, whether it is likely to continue to improve with more data, etc.

While some have reported results in the ALE using learning curves (e.g., Mnih et al. 2016; Ostrovski, Bellemare, van den Oord, & Munos 2017; Schaul et al. 2016), it is difficult to even effectively display, let alone comprehend and compare, 60 learning curves. For the sake of comparison and compact reporting, most researchers have applied various approaches to numerically summarize an agent's performance in each game (e.g., Bellemare, Naddaf, et al., 2013; Hausknecht et al., 2014; Munos, Stepleton, Harutyunyan, & Bellemare, 2016; Nair et al., 2015). Unfortunately, the variety of different summary statistics in results tables makes direct comparison difficult. In this section we consider some common performance measures seen in the literature and ultimately identify one as being particularly in line with the continual learning goal and advocate for it as the standard for reporting learning results in the ALE.

4.1 Common Performance Measures

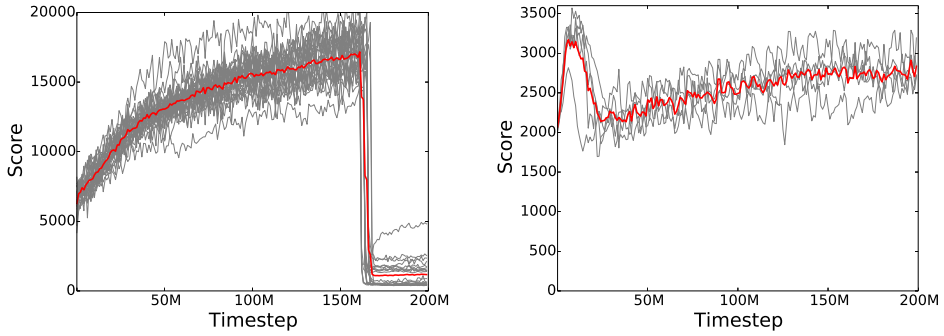
Here we discuss some common summary statistics of learning performance that have been employed in the Arcade Learning Environment in the past.

4.1.1 Evaluation after Learning

In the first ALE benchmark results, Bellemare, Naddaf, et al. (2013) trained agents for a fixed training period, then evaluated the learned policy using the average score in a number of evaluation episodes with no learning. Naturally, a number of subsequent studies used this evaluation protocol (e.g., Defazio & Graepel, 2014; Liang et al., 2016; Martin et al., 2017). One downside to this approach is that it hides issues of sample efficiency, since agents are not evaluated during the entire training period. Furthermore, an agent can receive a high score using this metric without continually improving its performance. For instance, an agent could spend its training period in a purely exploratory mode, gathering information but performing poorly, and then at evaluation time switch to an exploitative mode. While the problem of developing a good policy during an unevaluated training period is an interesting one, in reinforcement learning the agent is typically expected to continually improve with experience. Importantly, ϵ-greedy policies tend to perform better than greedy policies in the ALE (Bellemare, Naddaf, et al., 2013; Mnih et al., 2015). Therefore, this protocol does not necessarily benefit from turning off exploration during evaluation. In fact, often the reported results under this protocol do use ϵ-greedy policies during evaluation.

4.1.2 Evaluation of the Best Policy

When evaluating Deep Q-Networks, Mnih et al. (2015) also trained agents for a fixed training period. Along the way, they regularly evaluated the performance of the learned policy. At the end of the training period they evaluated the *best* policy in a number of evaluation episodes with no learning. A great deal of follow-up work has replicated this methodology (e.g., Schaul et al., 2016; van Hasselt et al., 2016). This protocol retains the downsides of evaluation after learning, and adds an additional one: it does not evaluate the *stability* of the agent's learning progress. Figure 1 illustrates the importance of this issue by showing different learning curves in the game Centipede. On one hand, Sarsa(λ)



(a) Sarsa(λ) + Blob-PROST (b) DQN

Figure 1: Comparison between learning curves of DQN and Sarsa(λ) + Blob-PROST in Centipede. Notice the y-axes are not on the same scale. Each point corresponds to the average performance over the last one hundred episodes. Grey curves depict individual trials. The red curve depicts the average over all trials.

+ Blob-PROST achieves a high score early on but then becomes unstable and fails to retain this successful policy. DQN's best score is much lower but it is also more stable (though not perfectly so). Reporting the performance of the best policy fails to recognize the plummeting behavior of both algorithms and DQN's more stable performance. Note also that the best score achieved across training is a statistically biased estimate of an agent's best performance: to avoid this bias, one should perform a second, independent evaluation of the agent at that particular point in time, as reported by Wang et al. (2016).

4.1.3 Area Under the Learning Curve

Recently, eschewing an explicit evaluation phase, Stadie, Levine, and Abbeel (2015) proposed the area under the learning curve as an evaluation metric. Intuitively, the area under the learning curve is generally proportional to how long a method achieves "good" performance, i.e., the average performance during training. Methods that only have performance spikes and methods that are unstable generally perform poorly under such metric. However, area under the learning curve does not capture the "plummeting" behavior illustrated in Figure 1. For example, in this case, Sarsa(λ) + Blob-PROST looks much better than DQN using this metric. Furthermore, area under the curve cannot distinguish a high-variance, unstable learning process from steady progress towards a good policy, even though we typically prefer the latter.

4.2 Proposal: Performance During Training

The performance metric we propose as a standard is simple and has been adopted before (e.g., Bellemare, Veness, & Bowling, 2012a). At the end of training (and ideally at other points as well) report the average performance of the last k episodes. This protocol does not use the explicit evaluation phase, thus requiring an agent to perform well while it is learning. This better aligns the performance metric with the goal of continual learning while also simplifying experimental methodology. Unstable methods that exhibit spiking and/or

plummeting learning curves will score poorly compared to those that stably and continually improve, even if they perform well during most of training.

Another advantage is that this metric is well-suited for analysis of an algorithm's sample efficiency. While the agent's performance near the end of training is typically of most interest, it is also straightforward to report the same statistic at various points during training, effectively summarizing the learning curve with a few selected points along the curve. Furthermore, if researchers make their full learning curve data publicly available, others can easily perform post-hoc analysis for the sake of comparison for any amount of training without having to fully re-evaluate existing methods. Currently, it is fairly standard to train agents for 200 million frames, in order to facilitate comparison with the DQN results reported by Mnih et al. (2015). This is equivalent to approximately 38 days of real-time gameplay and even at fast frame rates represents a significant computational expense. By reporting performance at multiple points during training, researchers can easily draw comparisons earlier in the learning process, reducing the computational burden of evaluating agents.

In accordance with this proposal, the benchmark results we present in Section 6 report the agent's average score of the last 100 episodes before the agent reaches 10, 50, 100, and 200 million frames[4] and our full learning curve data is publicly available.[5] We chose to average over 100 episodes in an attempt to obtain a reliable statistic that would not vary too much in case a small number of episodes present scores that are outliers. This number should also guarantee that the scores obtained in the oldest episodes being averaged still describe the performance of the current policy. This evaluation protocol allows us to derive insights regarding the learning rate and stability of the algorithms and will offer flexibility to researchers wishing to compare to these benchmarks in the future.

5. Determinism and Stochasticity in the Arcade Learning Environment

In almost all games, the dynamics within Stella itself (the Atari 2600 VCS emulator embedded within the ALE) are deterministic given the agent's actions. The agent always starts at the same initial state, and a given sequence of actions always leads to the same outcome. Bellemare et al. (2015) and Braylan, Hollenbeck, Meyerson, and Miikkulainen (2015) showed that this determinism can be exploited by agents that simply memorize an effective sequence of actions, attaining state-of-the-art scores while ignoring the agent's perceived state altogether. Such an approach is not likely to be successful beyond the ALE – in most problems of interest it is difficult, if not impossible, to exactly reproduce a specific state-action sequence, and closed-loop decision-making is required. An agent that relies upon the determinism of the ALE may achieve high scores, but may also be highly sensitive to small perturbations. For example, Hausknecht and Stone (2015) analyzed the role of determinism in the success of HyperNEAT-GGP (Hausknecht et al., 2014). Figure 2 shows that *memorizing-NEAT* (solid boxes) performs significantly worse under multiple forms of mild

[4] By reporting results this way we explicitly count the number of actions taken by the agent in the environment, making the timescale agents operate a parameter that does not impact the total number of interactions the agent will have with the environment. Mnih et al. (2015), in their seminal paper, reported their results with respect to the the number of decisions made by the agent. Their results obtained after 50 million agent steps, with a frame skip of 4, is equivalent to what we call 200 million frames.

[5] `http://www.marcgbellemare.info/static/data/machado17revisiting.zip`

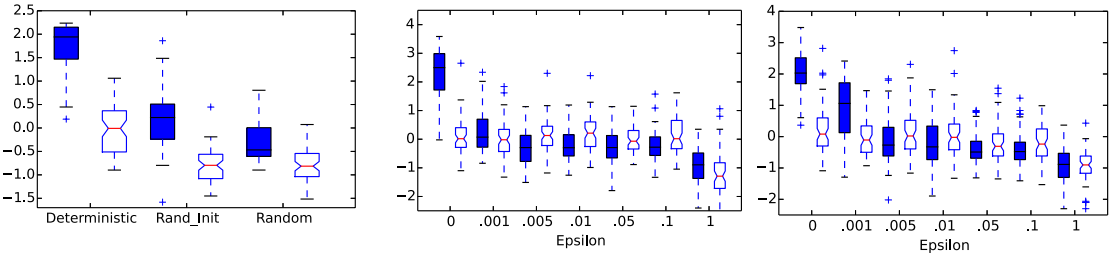


Figure 2: Final performance of a HyperNEAT agent under various models of stochasticity in ALE. Each plot corresponds to a different stochasticity model. Retangular boxplots correspond to memorizing-NEAT while hollow, pinched boxplots correspond to randomized-NEAT (Hausknecht et al., 2014). Each boxplot represents a single evaluation of 61 Atari 2600 games. Z-Score normalization is applied to normalize the per-game scores. The agent's overall performance is depicted in the y-axis while the amount of stochasticity in the environment increases along the x-axis. The first figure depicts the impact of random no-ops at the beginning of the game. Reference scores for a fully deterministic and fully random environments are provided. The second graph depicts the performance of both algorithms when forced to select actions ϵ-greedily for different values of ϵ. The third graph depicts the performance of both algorithms when forced to repeat the previous action with probability ϵ (equivalent to sticky actions). Reproduced from the work of Hausknecht and Stone (2015).

stochasticity, whereas *randomized-NEAT* (hollow, pinched boxes), which is trained with some stochastic perturbations, performs worse in the deterministic setting, but is more robust to various forms of stochasticity. As an evaluation platform, the deterministic ALE does not effectively distinguish between agents that learn robust, closed-loop policies from brittle memorization-based agents.

Recognizing this limitation in earlier versions of the ALE, many researchers have augmented the standard behavior of the ALE to evaluate the robustness of their agents and to discourage memorization (e.g., random frame skips, Brockman et al., 2016; injecting stochasticity, Hausknecht & Stone, 2015; no-ops, Mnih et al., 2015; human starts, Nair et al., 2015). Again, this wide range of experimental protocols makes direct comparison of results difficult. We believe the research community would benefit from a single standard protocol that empirically distinguishes between brittle, open-loop solutions and robust, closed-loop solutions.

In this section we discuss the Brute (first briefly introduced by Bellemare et al., 2015) as an example of an algorithm that explicitly and effectively exploits the environment's determinism. We present results in five Atari 2600 games comparing the Brute's performance with traditionally successful reinforcement learning methods. We then introduce the *sticky actions* method for injecting stochasticity into the ALE and show that it effectively distinguishes the Brute from methods that learn more robust policies. We also discuss pros and cons of several alternative experimental protocols aimed at discouraging open-loop policies, ultimately proposing *sticky actions* as a standard training and evaluation protocol, which is already incorporated in the latest versions of the Arcade Learning Environment.

Game	The Brute	Sarsa(λ) + Blob-PROST	DQN
Asterix	6,909 (1,018)	4,173 (872)	3,501 (420)
Beam Rider	1,132 (178)	2,098 (508)	4,687 (704)
Freeway	1.1 (0.4)	32.1 (0.4)	32.2 (0.1)
Seaquest	621 (192)	1,340 (245)	1,397 (215)
Space Invaders	1,432 (226)	723 (86)	673 (18)

Table 1: The Brute's performance compared to Sarsa(λ) + Blob-PROST and DQN in the deterministic Arcade Learning Environment. Standard deviation over trials is reported between parenthesis (we evaluated 24 trials for Sarsa(λ) + Blob-PROST and the Brute, and 5 trials for DQN). See text for details.

5.1 The Brute

The Brute is an algorithm designed to exploit features of the original Arcade Learning Environment. Although developed independently by some of this article's authors, it shares many similarities with the trajectory tree method of Kearns, Mansour, and Ng (1999). The Brute uses the agent's trajectory $h_t = a_1, o_1, a_2, o_2, \ldots, o_t$ as state representation, assigning individual values to each state. Because of the ALE's determinism, a single sample from each state-action pair is sufficient for a perfect estimate of the agent's return up to that point. The Brute maintains a partial history tree that contains all visited histories. Each node, associated with a history, maintains an action-conditional transition function and a reward function. The Brute estimates the value for any history-action pair using bottom-up dynamic programming. The agent follows the best trajectory found so far, with infrequent random actions used to search for better trajectories.

In order to be able to apply the Brute to stochastic environments, our implementation maintains the maximum likelihood estimate for both transition and reward functions. We provide a full description of the Brute in Appendix A.

5.1.1 Empirical Evaluation

We evaluated the performance of the Brute on the five training games proposed by Bellemare, Naddaf, et al. (2013). The average score obtained by the Brute, as well as of DQN and Sarsa(λ) + Blob-PROST, are presented in Table 1. Agents interacted with the environment for 50 million frames and the numbers reported are the average scores agents obtained in the last 100 episodes played while learning. We discuss our experimental setup in Appendix B.

The Brute is crude but we see that it leads to competitive performance in a number of games. In fact, Bellemare et al. (2015), using a different evaluation protocol, report that the Brute outperformed the best learning method at the time on 45 out of 55 Atari 2600 games. However, as we will see, this performance critically depends on the environment's determinism. In the next section we discuss how we modified the ALE to introduce a form of stochasticity we call sticky actions; and we show that the Brute fails when small random perturbations are introduced.

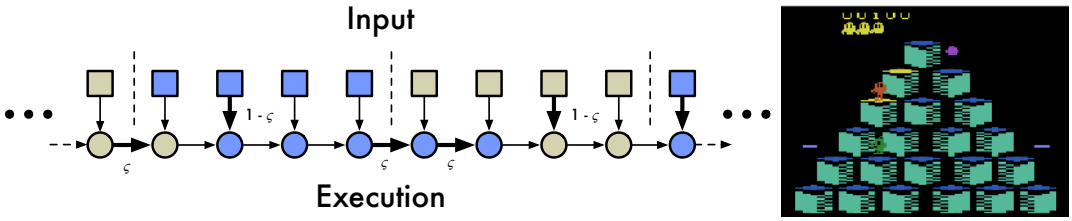


Figure 3: **Left.** Interaction between the environment's input and the action it executes. Different colors represent different actions, boldface arrows indicate time steps at which past execution and input disagree. With probability ς, the agent's input is ignored and the immediately preceding action is instead repeated. Vertical dotted lines indicate frame skipping boundaries; note that these are for illustration only, as our approach does not depend on frame skipping. **Right.** Q*BERT is one game where different stochasticity models have significantly different effects.

5.2 Sticky Actions

This section introduces *sticky actions*, our approach to injecting stochasticity into the ALE. This approach also evaluates the robustness of learned policies. Its design is based on the following desiderata:

- the stochasticity should be minimally non-Markovian with respect to the environment, i.e., the action to be executed by the emulator should be conditioned only on the action chosen by the agent and on the previous action executed by the emulator;
- the difficulty of existing tasks should not be changed, i.e., algorithms that do not rely on the environment's determinism should not have their performance hindered by the introduction of stochasticity; and
- it should be easy to implement in the ALE, not requiring changes inside the Stella emulator, but only on the framework itself.

In sticky actions there is a *stickiness* parameter ς, the probability at every time step that the environment will execute the agent's previous action again, instead of the agent's new action. More specifically, at time step t the agent decides to execute action a; however, the action A_t that the environment in fact executes is:

$$A_t = \begin{cases} a, & \text{with prob.} \quad 1-\varsigma, \\ a_{t-1}, & \text{with prob.} \quad \varsigma. \end{cases}$$

In other words, if $\varsigma = 0.25$, there is 25% chance the environment will not execute the desired action right away. Figure 3 (left) illustrates this process.

Notice that if an agent decides to select the same action for several time steps, the time it will take to have this action executed in the environment follows a geometric distribution. The probability the previous action is executed k times before the new action is executed is $\varsigma^k(1-\varsigma)$.

Sticky actions are different from random delays because, in the former, the agent can change its mind at any time by sending a new action to the emulator. To see why this matters, consider the game Q*BERT, where a single wrong action may cause the agent to jump

Game	The Brute				Sarsa(λ) + Blob-PROST				DQN			
	Determin.		Stochast.		Determin.		Stochast.		Determin.		Stochast.	
Asterix	6909	(1018)	308	(31)	4173	(872)	3411	(414)	3501	(420)	3123	(96)
Beam Rider	1132	(178)	428	(18)	2098	(508)	1851	(407)	4687	(704)	4552	(849)
Freeway	1.1	(0.4)	0.0	(0.0)	32.1	(0.4)	31.8	(0.3)	32.2	(0.1)	31.6	(0.7)
Seaquest	621	(192)	81	(7)	1340	(245)	1204	(190)	1397	(215)	1431	(162)
Space Invaders	1432	(226)	148	(11)	723	(86)	583	(31)	673	(18)	687	(37)

Table 2: The impact of stochasticity on different algorithms. We report the average over 24 trials for Sarsa(λ) + Blob-PROST and the Brute, and the average over 5 trials for DQN. Standard deviation over trials is within parentheses. The deterministic setting uses $\varsigma = 0.0$ while the stochastic setting uses $\varsigma = 0.25$. See text for details.

off the pyramid and lose a life (Figure 3, right). Under sticky actions, the agent can switch to a no-op before landing on the edge, knowing that with high probability the action will not be continued up to the point it pushes the agent off the pyramid. With random delays, the previous action will be executed until the delay is passed, even if the agent switched to a no-op before landing on the edge. This increases the likelihood the agent will be forced to continue moving once it lands on the edge, making it more likely to fall off the pyramid.

Sticky actions also interplay well with other aspects of the Arcade Learning Environment. Most Atari 2600 games are deterministic and it would be very hard to change their dynamics. Our approach only impacts which actions are sent to be executed. Sticky actions also interacts well with frame skipping (c.f. Section 2). With sticky actions, at each intermediate time step between the skipped frames there is a probability ς of executing the previous action. Obviously, this applies until the current action is executed, when the previous action taken and the current action become the same. Figure 3 depicts the process for a frame skip of 4.

5.2.1 Evaluating the Impact of Sticky Actions

We now re-evaluate the performance of the Brute, DQN and Sarsa(λ) + Blob-PROST under the sticky actions protocol. The intuition is that the Brute, which exploits the assumption that the environment is deterministic, should perform worse when stochasticity is introduced. We repeated the experiments from Section 5.1.1, but with $\varsigma = 0.25$. Table 2 depicts the algorithms' performance in both the stochastic environment and in the deterministic environment.

We can see that the Brute is the only algorithm substantially impacted by the sticky actions. These results suggest that sticky actions enable us to empirically evaluate an agent's robustness to perturbation.

5.3 Alternative Forms of Stochasticity

To conclude this section, we briefly discuss some alternatives to sticky actions, listing their pros (+) and cons (−). These alternatives fall in two broad categories: start-state methods and stochastic methods. In start-state methods, the first state of an episode is chosen randomly, but the deterministic dynamics remain unchanged. These approaches are less intrusive as the agent retains full control over its actions, but do not preclude exploiting the environment's determinism. This may be undesirable in games where the agent can exploit

game bugs by executing a perfectly timed sequence of actions, as in, for example, the game Q*bert (Z. Wang, personal communication, 2016). On the other hand, stochastic methods impact the agent's ability to control the environment uniformly throughout the episode, and thus its performance. We believe our proposed method minimizes this impact.

5.3.1 Initial No-ops

When evaluating the agent, begin the episode by taking from 0 to k no-op actions, selected uniformly at random (Mnih et al., 2015). By affecting the initial emulator state, this prevents the simplest form of open-loop control.

+ No interference with agent action selection.
− Impact varies across games. For example, initial no-ops have no effect in the game Freeway.
− The environment remains deterministic beyond the choice of starting state.
− Brute-like methods still perform well.

5.3.2 Random Human Starts

When evaluating the agent, randomly pick one of k predetermined starting states. Nair et al. (2015), for example, sampled starting states at random from a human's gameplay.

+ Allows evaluating the agent in very different situations.
− The environment remains deterministic beyond the choice of starting state.
− Brute-like methods still perform well.
− It may be difficult to provide starting states that are both meaningful and free of researcher bias. For example, scores as reported by Nair et al. (2015) are not comparable across starting states: although in a full game of Pong an agent can score 21 points, from a much later starting state this score is unachievable.

5.3.3 Uniformly Random Action Noise

With a small probability ς, the agent's selected action is replaced with another action drawn uniformly from the set of legal actions.

+ Matches the most commonly used form of exploration, ϵ-greedy.
− May significantly interfere with agent's policy, for example, when navigating a narrow cliff such as in the game Q*bert.

5.3.4 Random Frame Skips

This approach, implemented in OpenAI's Gym (Brockman et al., 2016), is closest to our method. Each action randomly lasts between k_1 and k_2 frames.

+ Does not interfere with action selection, only the timing of action execution.
− This restricts agents to using frame skip. In particular, the agent cannot react to events occurring during an action's period.

– Discounting must also be treated more carefully, as this makes the effective discount factor random.

– The agent has perfect reaction time since its actions always have an immediate effect.

5.3.5 Asynchronous Environment

More complex environments might involve unpredictable communication delays between the agent and the environment. This is the case in Minecraft (Project Malmo; Johnson et al., 2016), Starcraft (Ontañón et al., 2013), and robotic RL platforms (Sutton et al., 2011).

\+ This setting naturally discourages agents relying on determinism.

– Lacks reproducibility across platforms and hardware.

– With sufficiently fast communications, reverts to a deterministic environment.

5.3.6 Overall Comparison

Our proposed solution, sticky actions, leverages some of the main benefits of other approaches without most of their drawbacks. It is free from researcher bias, it does not interfere with agent action selection, and it discourages agents from relying on memorization. The new environment is stochastic for the whole episode, generated results are reproducible, and our approach interacts naturally with frame skipping and discounting.

6. Benchmark Results in the Arcade Learning Environment

In this section we introduce new benchmark results for DQN and Sarsa(λ) + Blob-PROST in 60 different Atari 2600 games using sticky actions. It is our hope that future work will adopt the experimental methodology described in this paper, and thus be able to directly compare results with this benchmark.

6.1 Experimental Method

We evaluated DQN and Sarsa(λ) + Blob-PROST in 60 different Atari 2600 games. We report results using the sticky actions option in the new version of the ALE ($\varsigma = 0.25$), evaluating the final performance while learning, at 10, 50, 100 and 200 million frames. We computed score averages of each trial using the 100 final episodes until the specified threshold, including the episode in which the total is exceeded. We report the average over 5 trials for DQN and the average over 24 trials for Sarsa(λ) + Blob-PROST. To ease reproducibility, we listed all the relevant parameters used by Sarsa(λ) + Blob-PROST and DQN in Appendix B. We encourage researchers to present their results on the ALE in the same reproducible fashion.

6.2 Benchmark Results

We present excerpts of the obtained results for Sarsa(λ) + Blob-PROST and DQN in Tables 3 and 4. These tables report the obtained scores in the games we used for training. These games were originally proposed by Bellemare, Naddaf, et al. (2013). The complete results are available in Appendix C.

Game	10M frames		50M frames		100M frames		200M frames	
Asterix	2,088.3	(302.5)	3,411.0	(413.5)	3,768.1	(312.5)	4,395.2	(460.7)
Beam Rider	1,149.1	(235.2)	1,851.2	(406.7)	2,116.4	(516.0)	2,231.9	(470.5)
Freeway	28.7	(5.1)	31.8	(0.3)	31.9	(0.2)	31.8	(0.2)
Seaquest	747.9	(222.2)	1,204.2	(189.8)	1,327.1	(337.9)	1,403.1	(301.7)
Space Invaders	458.2	(23.8)	582.9	(30.7)	661.6	(51.4)	759.7	(43.9)

Table 3: Results on the ALE's original training set using Sarsa(λ) + Blob-PROST. Averages over 24 trials are reported and standard deviation over trials is presented between parenthesis.

Game	10M frames		50M frames		100M frames		200M frames	
Asterix	1,732.6	(314.6)	3,122.6	(96.4)	3,423.4	(213.6)	2,866.8	(1,354.6)
Beam Rider	693.9	(111.0)	4,551.5	(849.1)	4,977.2	(292.2)	5,700.5	(362.5)
Freeway	13.8	(8.1)	31.7	(0.7)	32.4	(0.3)	33.0	(0.3)
Seaquest	311.5	(36.9)	1,430.8	(162.3)	1,573.4	(561.4)	1,485.7	(740.8)
Space Invaders	211.6	(14.8)	686.6	(37.0)	787.2	(173.3)	823.6	(335.0)

Table 4: Results on the ALE's original training set using DQN. Averages over 5 trials are reported and standard deviation over trials is presented between parenthesis.

Because we report the algorithms' performance at different points in time, these results give us insights about learning progress made by each algorithm. Such analysis allows us to verify, across 60 games, how often an agent's performance plummets; as well as how often agents reach their best performance before 200 million frames.

In most games, Sarsa(λ) + Blob-PROST's performance steadily increases for the whole learning period. In only 10% of the games the scores obtained with 200 million frames are lower than the scores obtained with 100 million frames. This difference is statistically significant in only 3 games:[6] Carnival, Centipede, and Wizard of Wor. However, in most games we observe diminishing improvements in an agent's performance. In only 22 out of 60 games we observe statistically significant improvements from 100 million frames to 200 million frames.[6] In several games such as Montezuma's Revenge this stagnation is due to exploration issues; the agent is not capable of finding additional rewards in the environment.

DQN has much higher variability in the learning process and it does not seem to benefit much from additional data. DQN obtained its highest scores using 200 million frames in only 35 out of 60 games. Agents' performance at 200 million frames was statistically better than agents' performance at 100 million frames in only 18 out of 60 games.[7] In contrast, Sarsa(λ) + Blob-PROST achieves its highest scores with 200 million samples in 50 out of 60 games. We did not observe statistically significant performance decreases for DQN when comparing agents' performance at 100 and 200 million samples.[7] It is important to add a caveat that the lack of statistically significant results may be due to our sample size ($n = 5$). The t-test's power may still be too low to detect significant differences in DQN's performance. It is worth pointing out that when DQN was originally introduced, its results consisted of only one independent trial. Despite its high computational cost we evaluated it on 5 trials in an attempt to evaluate such an important algorithm more

[6] Welch's t-test ($p < 0.05$; $n = 24$).
[7] Welch's t-test ($p < 0.05$; $n = 5$).

thoroughly, addressing the methodological concerns we discussed above and offering a more reproducible and statistically comparable DQN benchmark.

We also compared the performance of both algorithms in each game to understand specific trends such as performance plummeting and absence of learning. Performance drops seem to be algorithm dependent, not game dependent. Centipede is the only game in which plummeting performance was observed for both DQN and Sarsa(λ) + Blob-PROST. The decrease in performance we observe in other games occurs only for one algorithm. On the other hand, we were able to identify some games that seem to be harder than others for both algorithms. Both algorithms fail to make much progress on games such as Asteroids, Pitfall, and Tennis. These games generally pose hard exploration tasks to the agent; or have complex dynamics, demanding better representations capable of accurately encoding value function approximations.

We can also compare our results to previously published results to verify the impact our proposed evaluation protocol has in agents' performance. This new setting does not seem to benefit a specific algorithm. Sarsa(λ) + Blob-PROST and DQN still present comparable performance, with each algorithm being better in an equal number of games, as suggested by Liang et al. (2016). As we already discussed in Section 5, using sticky actions seems to only substantially hinder the performance of the Brute agent, not having much impact in the performance of DQN and Sarsa(λ) + Blob-PROST. We observed decreased performance for DQN and Sarsa(λ) + Blob-PROST only in three games: Breakout, Gopher, and Pong.

7. Open Problems and the Current State-of-the-Art in the ALE

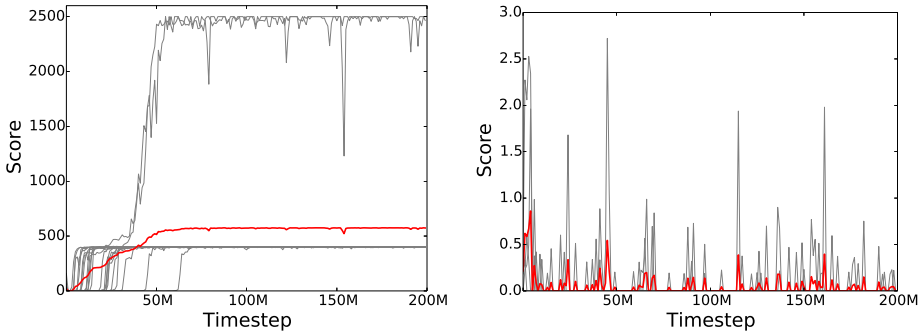
To provide a complete big picture of how the ALE is being used by the research community, it is also important to discuss the variety of research problems for which the community has used the ALE as a testbed. In the past few years we have seen several successes showcased in the ALE, with new results introduced at a rapid pace.

We list five important research directions the community has worked on using the ALE, and we use current results in the literature to argue that while there has been substantial progress these problems still remain open. These research directions are:

- representation learning,
- exploration,
- transfer learning,
- model learning, and
- off-policy learning.

7.1 Representation Learning

The ALE was originally introduced to pose the problem of general competency: expecting a single algorithm to be capable of playing dozens of Atari 2600 games. Therefore, agents must either use generic encodings capable of representing all games (e.g., Liang et al., 2016), or be able to automatically learn representations. The latter is obviously more desirable for



(a) Sarsa(λ) + Blob-PROST (b) DQN

Figure 4: Comparison between learning curves of DQN and Sarsa(λ) + Blob-PROST in Montezuma's Revenge. Notice the y-axes are not on the same scale. Each point corresponds to the average performance over the last one hundred episodes. Grey curves depict individual trials. The red curve depicts the average over all trials.

the potential of discovering better representations while alleviating the burden of having handcrafted features.

Deep Q-Networks (DQN) of Mnih et al. (2015) demonstrate it is possible to learn representations jointly with control policies. However, reinforcement learning methods based on neural networks still have a high sample complexity, requiring at least dozens of millions of samples before achieving good performance, in part due to the need for learning this representation. In the results we report, DQN's performance (Table 9) is better than Sarsa(λ) + Blob-PROST's (Table 8) in less than 20% of the games when evaluated at 10 million frames, and achieves comparable performance at 100 million frames. The high sample complexity also seems to hinder the agents' performance in specific environments, such as when non-zero rewards are very sparse. Figure 4 illustrates this point by showing how DQN sees non-zero rewards occasionally while playing Montezuma's Revenge (Figure 4b), but it does not learn to obtain non-zero rewards consistently. Recently, researchers have tried to address this issue by weighting samples differently, prioritizing those that seem to provide more information to the agent (Schaul et al., 2016). Another approach is to use auxiliary tasks that allow agents to start learning a representation before the first extrinsic reward is observed (Jaderberg et al., 2017); the distributions output by the C51 algorithm of Bellemare, Dabney, and Munos (2017) may be viewed as a particularly meaningful set of auxiliary tasks. Finally, intrinsically generated rewards (Bellemare et al., 2016b) may also provide a useful learning signal which the agent can use to build a representation.

Despite this high sample complexity, DQN and DQN-like approaches remain the best performing methods overall when compared to simple, hand-coded representations (Liang et al., 2016). However, these improvements are not as dramatic as they are in other applications (e.g., computer vision; Krizhevsky, Sutskever, & Hinton, 2012). Furthermore, this superior performance often comes at the cost of additional tuning, as recently reported by Islam, Henderson, Gomrokchi, and Precup (2017) in the context of continuous control. This suggests that there is still room for significant progress on effectively learning good representations in the ALE.

Different approaches that learn an internal representation in a sample efficient way have also been proposed (Veness, Bellemare, Hutter, Chua, & Desjardins, 2015), although they have not yet been fully explored in this setting. Other directions the research community has been looking at are the development of better visualization methods (Zahavy, Ben-Zrihem, & Mannor, 2016), the proposal of algorithms that alleviate the need for specialized hardware (Mnih et al., 2016), and genetic algorithms (Kelly & Heywood, 2017).

7.2 Planning and Model-Learning

Despite multiple successes of search algorithms in artificial intelligence (e.g., Campbell, Hoane, & Hsu, 2002; Schaeffer et al., 2007; Silver et al., 2016), planning in the Arcade Learning Environment remains rare compared to methods that learn policies or value functions (but see the papers by Bellemare, Naddaf, et al., 2013; Guo, Singh, Lee, Lewis, & Wang, 2014; Jinnai & Fukunaga, 2017; Lipovetzky et al., 2015; Shleyfman, Tuisov, & Domshlak, 2016, for published planning results in the ALE). Developing heuristics that are general enough to be successfully applied to dozens of different games is a challenging problem. The problem's branching factor and the fact that goals are sometimes thousands of steps ahead of the agent's initial state are also major difficulties.

Almost all successes of planning in the ALE use the generative model provided by the Stella emulator, and so have an exact model of the environment. Learning generative models is a very challenging task (Bellemare, Veness, & Bowling, 2013; Bellemare, Veness, & Talvitie, 2014; Chiappa, Racaniere, Wierstra, & Mohamed, 2017; Oh, Guo, Lee, Lewis, & Singh, 2015) and so far, there has been no clear demonstration of successful planning with a learned model in the ALE. Learned models tend to be accurate for a small number of time steps until errors start to compound (Talvitie, 2014). As an example, Figure 5 depicts rollouts obtained with one of the first generative models trained on the ALE (Bellemare, Veness, & Bowling, 2013). In this figure we can see how the accuracy of rollouts start to drop after a few dozen time steps. Probably the most successful example of model learning in the ALE is due to Oh et al. (2015) who designed an algorithm capable of learning multistep models that, up to one hundred time steps, appear accurate. These models are able to assist with exploration, an indication of the models' accuracy. However, because of compounding errors, the algorithm still needs to frequently restore its model to the real state of the game. More recently, Chiappa et al. (2017) showed significant improvements over this original model, including the ability to plan with the internal state. In both cases, however, the models are much slower than the emulator itself; designing a fast, accurate model remains an open problem.

A related open problem is how to plan with an imperfect model. Although an error-free model might be unattainable, there is plenty of evidence that even coarse value functions are sufficient for the model-free case (Veness et al., 2015), raising the question of how to compensate for a model's flaws. Training set augmentation (Talvitie, 2014, 2017; Venkatraman, Hebert, & Bagnell, 2015) has shown that it is possible to improve an otherwise limited model. Similarly, Farahmand, Barreto, and Nikovski (2017) showed that better planning performance could be obtained by using a value-aware loss function when training the model. We believe this to be a rich research direction.

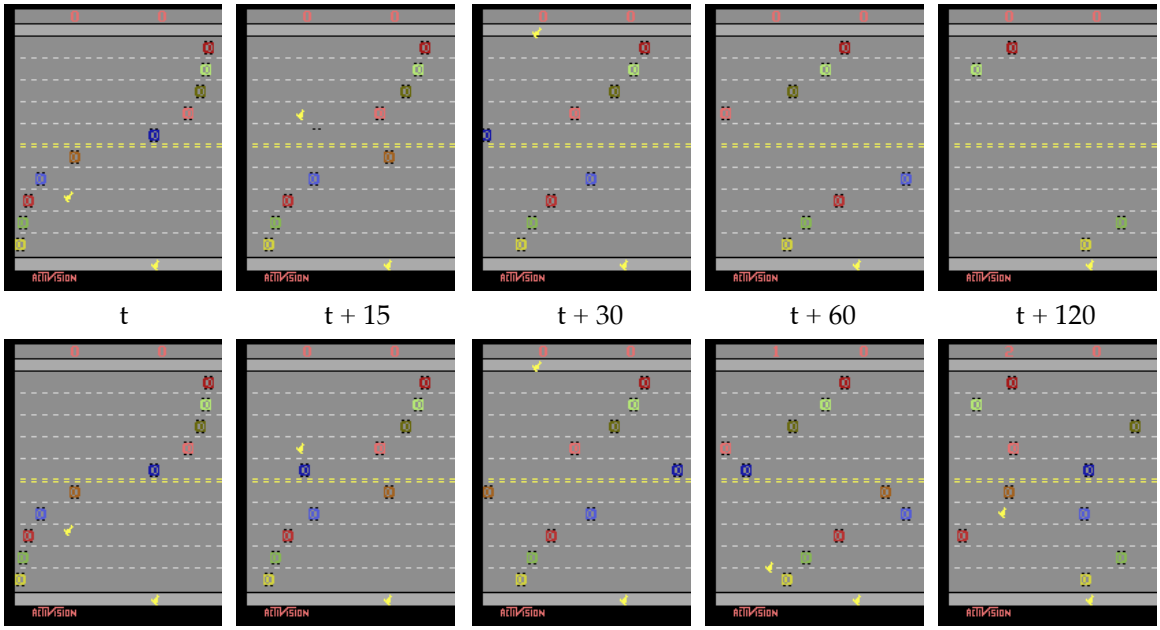
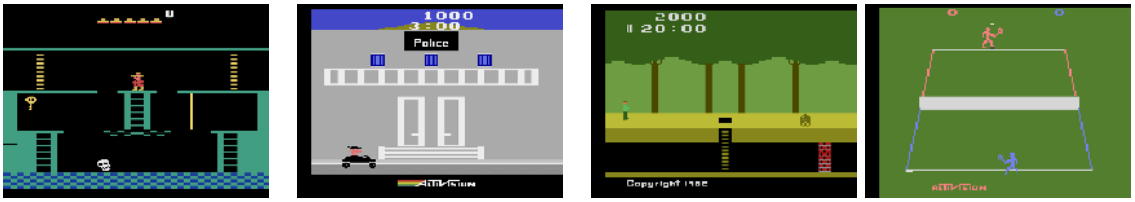


Figure 5: **Top row:** Rollout obtained with a learned model of the game Freeway. **Bottom row:** Ground truth. Small errors can be noticed ($t + 15$) but major errors are observed only when the chicken crosses the street ($t + 30$), as depicted in frame $t + 60$. The score is not updated and the chicken does not respawn at the bottom of the screen. Later, cars start to disappear, as shown in the frame $t + 120$. This model was learned using quad-tree factorization (Bellemare, Veness, & Bowling, 2013).

7.3 Exploration

Most approaches for exploration focus on the tabular case and generally learn models of the environment (e.g., Brafman & Tennenholtz, 2002; Kearns & Singh, 2002; Strehl & Littman, 2008). The community is just beginning to investigate exploration strategies in model-free settings when function approximation is required (e.g., Bellemare et al., 2016b; Machado, Bellemare, & Bowling, 2017; Martin et al., 2017; Osband, Blundell, Pritzel, & Roy, 2016; Ostrovski et al., 2017; Vezhnevets et al., 2017). This is the setting in which the ALE lies. Visiting every state does not seem to be a feasible strategy given the large number of possible states in a game (potentially 2^{1024} different states since the Atari 2600 has 1024 bits of RAM memory). In several games such as Montezuma's Revenge and Private Eye (see Figure 6) even obtaining any feedback is difficult because thousands of actions may be required before a first positive reward is seen. Given the usual sample constraints (200 million frames), random exploration is highly unlikely to guide the agent towards positive rewards. In fact, some games such as Pitfall! and Tennis (see Figure 6) pose an even harder challenge: random exploration is more likely to yield negative rewards than positive ones. In consequence, many simpler agents learn that staying put is the myopically best policy, although recent state-of-the-art agents (e.g., Bellemare et al., 2017; Jaderberg et al., 2017) can sometimes overcome this negative reward gradient.



(a) Mont. Revenge (b) Private Eye (c) Pitfall! (d) Tennis

Figure 6: Challenging games in the ALE due to poor exploration.

Some researchers recently started trying to address the exploration problem in the ALE. Machado, Srinivasan, and Bowling (2015) extended optimistic initialization to function approximation. Oh et al. (2015) and Stadie et al. (2015) learned models to predict which actions lead the agent to frames observed least often, or with more uncertainty. Bellemare et al. (2016b), Martin et al. (2017) and Ostrovski et al. (2017) extended state visitation counters to the case of function approximation. Osband et al. (2016) used randomized value functions to better explore the environment. Machado et al. (2017) and Vezhnevets et al. (2017) proposed the use of options to generate decisive agents, avoiding the dithering commonly observed in random walks. However, despite successes in individual games, such as Bellemare et al.'s (2016b) success in Montezuma's Revenge, none of these approaches has been able to improve, in a meaningful way, agents' performance in games such as Pitfall!, where the only successes to date involve some form of apprenticeship (e.g., Hester et al., 2017).

There is still much to be done to narrow the gap between solutions applicable to the tabular case and solutions applicable to the ALE. An aspect that still seems to be missing are agents capable of committing to a decision for extended periods of time, exploring in a different level of abstraction, something that humans frequently do. Maybe agents should not be exploring in terms of joystick movements, but in terms of object configurations and game levels. Finally, for intrinsically difficult games, agents may need some form of intrinsic motivation (Barto, 2013; Oudeyer, Kaplan, & Hafner, 2007) to keep playing despite the apparent impossibility of scoring in the game.

7.4 Transfer Learning

Most work in the ALE involves training agents separately in each game, but many Atari 2600 games have similar dynamics. We can expect knowledge transfer to reduce the required number of samples needed to learn to play similar games. As an example, Space Invaders and Demon Attack (Figure 7) are two similar games in which the agent is represented by a spaceship at the bottom of the screen and it is expected to shoot incoming enemies. A more ambitious research question is how to leverage general video game experience, sharing knowledge across games that are not directly analogous. In this case, more abstract concepts could be learned, such as "sometimes new screens are seen when the avatar goes to the edge of the current screen".

There are attempts to apply transfer learning in the ALE (Parisotto, Ba, & Salakhutdinov, 2016; Rusu et al., 2016). Such attempts are restricted to a dozen games that tend

(a) Space Invaders (b) Demon Attack

Figure 7: Very similar games in the ALE.

to be similar and generally require an "expert" network first, instead of learning how to play all games concurrently. Taylor and Stone (2009) have shown one can face *negative transfer* depending on the similarity between the tasks being used. It is not clear how this should be addressed in the ALE. Ideally one would like to have an algorithm automatically deciding which games are helpful and which ones are not. Finally, current approaches are only based on the use of neural networks to perform transfer, conflating representation and policy transfer. It may be interesting to investigate how to transfer each one of these entities independently. To help explore these issues, the most recent version of the ALE supports game modes and difficulty settings.

7.4.1 Modes and Difficulties in the Arcade Learning Environment

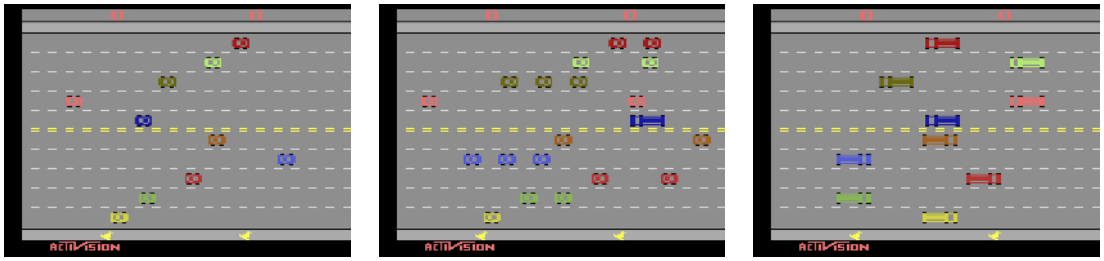
Originally, many Atari 2600 games had a default game mode and difficulty level that could be changed by changing physical switches on the console. These mode/difficulty switches had different consequences such as changing the game dynamics or introducing new actions (see Figure 8). Until recently, the ALE allowed agents to play games only in their default mode and difficulty. The newest version of the ALE allows one to select among all different game modes and difficulties that are single player games. We call each mode-difficulty pair a *flavor*.

This new feature opens up research avenues by introducing dozens of new environments that are very similar. Because the underlying state representations across different flavors are probably highly related, we believe negative transfer is less likely, giving an easier setup for transfer. The list of such games the ALE will initially support, and their number of flavors, is available in Appendix D.

7.5 Off-Policy Learning

Off-policy learning algorithms seem to be brittle when applied to the ALE. Defazio and Graepel (2014) have reported divergence when using algorithms such as GQ(λ), without the projection step, and Q-learning.

Besides the proposal of new algorithms that are theoretically better behaved (e.g., Maei & Sutton, 2010), attempts to reduce divergence in off-policy learning currently consist of heuristics that try to decorrelate observations, such as the use of an experience replay buffer and the use of a target network in DQN (Mnih et al., 2015). Recent papers introduce



(a) Mode 1 (b) Mode 2 (c) Mode 4

Figure 8: Different modes of the game Freeway.

changes in the update rules of Q-Learning to reduce overestimation of value functions (van Hasselt et al., 2016), new operators that increase the action-gap of value function estimates (Bellemare, Ostrovski, Guez, Thomas, & Munos, 2016a), and more robust off-policy multi-step algorithms (Harutyunyan, Bellemare, Stepleton, & Munos, 2016; Munos et al., 2016). However, besides a better theoretical understanding about convergence, stable (and practical) off-policy learning algorithms with function approximation are still an incomplete piece in the literature. So far, the best empirical results reported in the ALE were obtained with algorithms whose performance is not completely explained by current theoretical results. A thorough empirical evaluation of recent off-policy algorithms, such as GTD (Sutton, Szepesvári, & Maei, 2008), remains lacking.

Addressing the aforementioned issues, either through a convincing demonstration of the efficacy of the current theoretically sound algorithms for off-policy learning, or through some of the improvements described above may free us from the increased complexity of using experience replay and/or target networks. Also, this would allow us to better reuse samples from policies that are very different from the one being learned.

8. Conclusion

In this article we took a big picture look at how the Arcade Learning Environment is being used by the research community. We discussed the different evaluation methodologies that have been employed and how they have been frequently conflated in the literature. To further the progress in the field, we presented some methodological best practices and a new version of the Arcade Learning Environment that supports stochasticity and multiple game modes. We hope such methodological practices, with the new ALE, allow one to clearly distinguish between the different evaluation protocols. Also, we provide benchmark results following these methodological best practices that may serve as a point of comparison for future work in the ALE. We evaluated reinforcement learning algorithms that use linear and non-linear function approximation, and we hope to have promoted the discussion about sample efficiency by reporting algorithms' performance at different moments of the learning period. In the final part of this paper we concluded the big picture look we took by revisiting the challenges posed in the ALE's original article. We summarized the current state-of-the-art and we highlighted five problems we consider to remain open: representation learning, planning and model-learning, exploration, transfer learning, and off-policy learning.

Acknowledgements

The authors would like to thank David Silver and Tom Schaul for their thorough feedback on an earlier draft, and Rémi Munos, Will Dabney, Mohammad Azar, Hector Geffner, Jean Harb, and Pierre-Luc Bacon for useful discussions. We thank the anonymous reviewers for their feedback, which improved the clarity of the paper. We would also like to thank the several contributors to the Arcade Learning Environment GitHub repository, specially Nicolas Carion for implementing most of the mode and difficult selection and Ben Goodrich for providing a Python interface to the ALE. Yitao Liang implemented, with Marlos C. Machado, the Blob-PROST features. This work was supported by grants from Alberta Innovates – Technology Futures (AITF), through the Alberta Machine Intelligence Institute (Amii), and by the NSF grant IIS-1552533. Computing resources were provided by Compute Canada through CalculQuébec. Marc G. Bellemare performed this work while at DeepMind.

Appendix A. The Brute

The Brute is an algorithm designed to exploit features of the original Arcade Learning Environment. Although developed independently by some of the authors, it shares many similarities with the trajectory tree method by Kearns et al. (1999). The Brute relies on the following observations:

- The ALE is deterministic, episodic, and guarantees a unique starting state, and
- in most Atari 2600 games, *purpose* matters more than individual actions, i.e., most Atari 2600 games have important high-level goals, but individual actions have little impact.

This algorithm is crude but leads to competitive performance in a number of games.

A.1 Determinism and Starting Configurations

A history is a sequence of actions and observations $h_t = a_1, o_1, a_2, o_2, \ldots, o_t$, with the reward r_t included in the observation o_t.[8] Histories describe sequential interactions between an agent and its environment. Although most of reinforcement learning focuses on a Markov state, a sufficient statistic of the history, we may also reason directly about this history. This approach is particularly convenient when the environment is partially observable (Even-Dar, Kakade, & Mansour, 2005; Kearns et al., 1999) or non-Markov (Hutter, 2005). Given a history h_t, the transition function for an action a and subsequent observation o is

$$\Pr(H_{t+1} = h_t, a, o \,|\, H_t = h_t, A_t = a) = \Pr(O_{t+1} = o \,|\, H_t = h_t, A_t = a).$$

This transition function induces a Markov decision process over histories. This MDP is an infinite *history tree* (Figure 9) whose states correspond to distinct histories.

An environment is deterministic if taking action a from history h always produces the same observation. It is episodic when we have zero-valued, absorbing states called terminal states. In the episodic setting learning proceeds by means of resets to one or many start

[8] In the interest of legibility and symmetry, we follow here the convention of beginning histories with an action. Note, however, that the ALE provides the agent with an initial frame o_0.

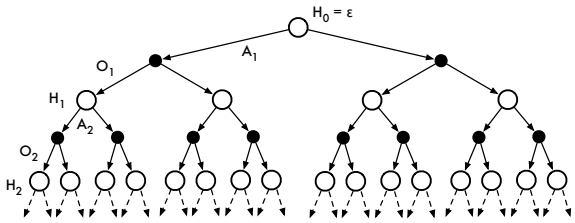


Figure 9: History tree representation of an environment.

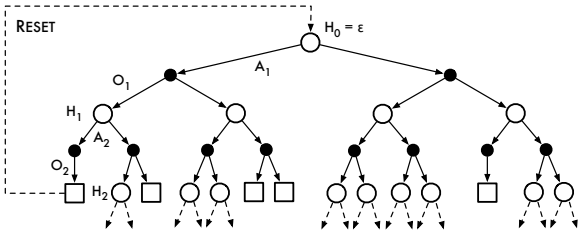


Figure 10: In the episodic setting, the agent is reset after reaching a terminal state (represented by a square). We equate this reset with the empty history.

states. Since the agent is informed of this reset, we equate it with the empty history ϵ (Figure 10). The Stella emulator is deterministic and, by the nature of Atari 2600 games, defines an episodic problem.

Depending on the game, both software and hardware resets of the emulator may leave the system in a number of initial configurations. These different configurations arise from changing timer values, registers, and memory contents at reset. However, these effects are game-dependent and difficult to control. In fact, the ALE contains code to avoid these effects and guarantee a unique starting configuration. We will use the term *reproducible* to describe an environment like the ALE that is deterministic, episodic, and has a unique starting configuration.

Determinism simplifies the learning of an environment's transition model: a single sample from each state-action pair is sufficient. Reproducibility allows us to effectively perform experiments on the history tree, answering questions of the form "what would happen if I performed this exact sequence of actions?" Not unlike Monte-Carlo tree search in a deterministic domain, each experiment begins at the root of the history tree and selects actions until a terminal state is reached, observing rewards along the way. Although it is possible to do the same in any episodic environment, learning stochastic transitions and reward functions is harder, not only because they require more samples but also because the probability of reaching a particular state (i.e., a history) is exponentially small in its length.

A.2 Value Estimation in a History Tree

According to Bellman's optimality equation (Bellman, 1957), the optimal value of executing action a in state s is

$$q_*(s,a) = r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{b \in \mathcal{A}} q_*(s',b).$$

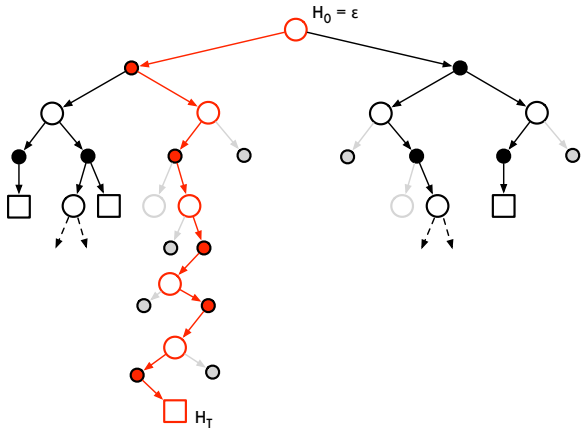


Figure 11: A partially known history tree. Filled gray circles represent actions not yet taken by the agent, large gray circles unseen observations (when the environment is stochastic). The most recent episode is highlighted in red. The lower bound $\hat{q}(h, a)$ is updated by starting at H_T and following the path back to the root.

Given a full history tree of finite depth, estimating the value for any history-action pair is simply a matter of bottom-up dynamic programming, since all states (i.e., histories) are transient. We can in fact leverage an important property of history trees: Consider a partially known history tree for a deterministic environment and define $\hat{q}(h, a) = -\infty$ for any unknown history-action pair. Then the equation

$$\hat{q}(h, a) = r(h, a) + \gamma \sum_{h'} p(h' \mid h, a) \max_{b \in \mathcal{A}} \hat{q}(h', b)$$

defines a lower bound on $q_*(h, a)$.

When learning proceeds in episodes, we can update the lower bound $\hat{q}(h, a)$ iteratively. We begin at the terminal node h_T corresponding to the episode just played. We then follow the episode steps $a_{T-1}, h_{T-1}, a_{T-2}, h_{T-2}, \ldots$ in reverse, updating $\hat{q}(h_t, a_t)$ along this path, up to and including the starting history-action pair (ϵ, a_1). Since no information has been gathered outside of this path, all other action-values must remain unchanged, and this procedure is correct. If $\pi(h) \doteq \arg\max_{a \in \mathcal{A}} \hat{q}(h, a)$ is stored at each node, then updating one episode requires time $O(T)$. Figure 11 illustrates the inclusion of a new episode into a partial history tree.

The Brute maintains a partial history tree that contains all visited histories. Each node, associated with a history, maintains an action-conditional transition function and reward function. Our implementation maintains the maximum likelihood estimate for both functions. This allows us to apply the Brute to stochastic environments, although $\hat{q}(h, a)$ is only guaranteed to be a proper lower bound if the subtree rooted at h is fully deterministic. This allowed us to apply the exact same algorithm in the context of sticky actions (Section 5). The value $\hat{q}(h, a)$ is maintained at each node and updated from the maximum likelihood estimates at the end of each episode, as described above.

A.3 Narrow Exploration

In Atari 2600 games, most actions have little individual effect. An agent can thus be more efficient if it focuses on a few narrow, promising trajectories rather than explore every detail of its environment. We may think of this focus as emphasizing purpose, i.e., achieving specific goals. The sequence of actions which maximizes the lower bound $\hat{q}(h, a)$ at each node is one such purposeful path. Since exploration is less relevant at nodes which have been visited often, we also progressively reduce the rate of exploration in the upper parts of the history tree.

To encourage the exploration of the most promising trajectory, the Brute's policy is an ϵ-greedy policy over $\hat{q}(h, a)$: with probability $1 - \epsilon$, we choose one of the maximum-valued actions (breaking ties uniformly at random), and with probability ϵ we select an action uniformly at random. To encourage the exploration of narrow paths, ϵ is decreased with the number of visits $n(h)$ to a particular node in the history tree. Specifically,

$$\epsilon(h) = \min \left\{ \frac{0.05}{\log(n(h) + 1)}, 1.0 \right\}.$$

Appendix B. Experimental Setup

We used the same evaluation protocol, and parameters, in all experiments discussed in this article. In the next section we list the parameters used when defining the task in the Arcade Learning Environment. Later we discuss the parameters used by the Brute, Sarsa(λ) + Blob-PROST, and DQN.

B.1 Evaluation Protocol and Arcade Learning Environment Parameters

We report our results aiming at evaluating the robustness of the learned policy and of the learning algorithm. All results we report for the Brute and for Sarsa(λ) + Blob-PROST are averaged over 24 trials, and all results we report for DQN are averaged over 5 trials. We evaluated DQN fewer times because its empirical validation is more expensive due to its requirement for specialized hardware (i.e., GPUs). We obtained the result of each trial by averaging over the last 100 episodes that led the agent to observe a total of k frames. Along this article we reported results for k equals to 10, 50, 100, and 200 million frames.

The unique parameter in the Arcade Learning Environment that is not fixed across all sections in this article is ς, i.e., the amount of stochasticity present in the environment. We set ς to 0.0 in Section 5.1.1 while we set ς to 0.25 in the rest of the article. We do not use game-specific information. Episodes terminate after 5 minutes of gameplay or when the agent has lost all of its lives. Agents have access to all 18 primitive actions available in the ALE, not knowing if specific actions have any effect in the environment. Finally, all algorithms used a frame skip equals to 5 when playing the games. We summarize all parameters that are shared across all methods in Table 5.

B.2 Parameters used by the Brute

The Brute has only two parameters to be set: γ and ϵ. We defined $\gamma = 1.0$ and $\epsilon = 0.005/\log(n_i + 2)$, where n_i denotes the number of times we have seen the history h_i (see

Hyperparameter	Value	Description
Action set	Full	18 actions are always available to the agent.
Max. episode length	18,000	Each episode lasts, at most, 5 minutes (18,000 frames).
Frame skip	5	Each action lasts 5 time steps. See Section 2 for details.
Stochasticity (ς)	0.0 or 0.25	We used $\varsigma = 0.25$ for all experiments, except for those in Section 5.1.1, in which we used $\varsigma = 0.0$.
Lives signal used	False	We did not use the game-specific information about the number of lives the agent has at each time step.
Number of episodes used for evaluation	100	In the continual learning setting, we report the average score obtained in the last 100 episodes used for learning.
Number of frames used for learning	10, 50, 100 and 200 million	We report scores obtained after each one of these four milestones.
Number of trials ran	24 or 5	The Brute and Sarsa(λ) + Blob-PROST were evaluated in 24 trials, DQN was evaluated in 5 trials.

Table 5: Parameters used to evaluate the Brute, DQN, and Sarsa(λ) + Blob-PROST.

Appendix A for details). An important implementation detail is that we used Spooky Hash[9] as our hashing function. We do not average current and previous ALE screens as other methods do.

B.3 Parameters used by DQN

DQN was ran using the same parameters used in its original paper (Mnih et al., 2015), with the exception of the frame skip, which we set to 5 after preliminary experiments, and ϵ, which we set to 0.01 due to the absence of an evaluation phase. Also, we did not use game-specific information and we evaluated DQN in the continual learning setting, as discussed in Section B.1. Table 6 lists the values of all DQN parameters used throughout this article.

We report the parameters the same way Mnih et al. (2015) reported, with an agent-centric perspective. Thus, the value of the parameters *Final expl. frame*, *Replay memory*, and *Replay start size* are being reported in terms of the rate at which the agent operates, regardless of the frame skip value. We use the rate at which the environment operates (i.e., 200 million frames) to count the number of interactions the agent had with the environment to allow a fair comparison across algorithms.

B.4 Parameters used by Sarsa(λ) + Blob-PROST

We evaluated Sarsa(λ) + Blob-PROST using $\alpha = 0.5$, $\lambda = 0.9$, and $\gamma = 0.99$. Agents followed an ϵ-greedy policy ($\epsilon = 0.01$). We did not sweep most of the parameters, using the parameters reported by Liang et al. (2016). However, we did verify, in preliminary experiments, the impact different values of frame skip have in this algorithm. We also verified whether color averaging impacts agents' performance. We decide to use a frame skip of 5 and to average colors. For most games, averaging screen colors significantly improves the results, while the impact of different number of frames to skip varies across games. Table 7 summarizes, for Sarsa(λ) + Blob-PROST, all the parameters we use throughout this article.

[9] `http://burtleburtle.net/bob/hash/spooky.html`

Hyperparameter	Value	Description
Step-size (α)	0.00025	Step-size used by RMSProp.
Gradient momentum	0.95	Gradient momentum used by RMSProp.
Squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
Min squared gradient	0.01	Constant added to the denominator of the RMSProp update.
Discount factor (γ)	0.99	Discount factor used in Q-Learning update rule. Rewards are discounted by how far they are in time.
Initial expl. rate (ϵ)	1.0	Probability a random action will be taken at each time step.
Final expl. rate (ϵ)	0.01	Probability a random action will be taken at each time step.
Final expl. frame	1,000,000	Number of steps over which ϵ is linearly annealead.
Minibatch size	32	Number of samples over which each update is computed.
Replay memory	1,000,000	The samples used in the algorithm's updates are drawn from the last 1 million recent frames.
Replay start size	50,000	Number of steps over which a random policy is executed to first populate the replay memory.
Agent history length	4	Number of most recent frames the agent observed that are given as input to the network.
Update frequency	4	Frequency with which the target network is updated.
Update frequency of target network	10,000	Number of actions the agent selects between successive updates.
Frame pooling	True	The observation received consists of the maximum pixel value between the previous and the current frame.
Number of different colors	8	NTSC is the color palette in which each screen is encoded but at the end only the luminance channel is used.

Table 6: Parameters used in the experiments evaluating DQN (Mnih et al., 2015). We report the parameters the same way Mnih et al. (2015) reported, with an agent-centric perspective. Thus, the value of the parameters *Final expl. frame*, *Replay memory*, and *Replay start size* are being reported in terms of the rate at which the agent operates, regardless of the frame skip value. See reference for more details about the parameters listed below.

Appendix C. Complete Benchmark Results

We extend the results presented in Section 6 (Tables 3 and 4) by reporting algorithms' performance in 60 games supported by the ALE. We used the evaluation protocol described in Appendix B when generating the results below. Table 8 summarizes the performance of Sarsa(λ) + Blob-PROST and Table 9 summarizes DQN's performance. The games originally used as training games by each method are highlighted with the † symbol. In Table 8, the list of games we used for training Sarsa(λ) + Blob-PROST is longer than the one in Table 9 because we are reporting the training games used by Liang et al. (2016), which was the setting we initially replicated. We put an asterisk on the results of Sarsa(λ) + Blob-PROST in the game Journey Escape because it is the average over 23 trials instead of the regular 24 trials. One of our executions crashed and this particular result might be slightly biased in the event that the crash was correlated with the agent's performance in the episode.

Hyperparameter	**Value**	**Description**
Step-size (α)	0.50	Step-size used in Sarsa(λ) update rule. At every time step we divide α by the largest number of active features we have seen so far. This reduces the step-size, avoiding divergence, while ensuring the step-size will never increase.
Discount factor (γ)	0.99	Discount factor used in Sarsa(λ) update rule. Rewards are discounted by how far they are in time.
Exploration rate (ϵ)	0.01	Probability a random action will be taken at each time step.
Eligibility traces decay rate (λ)	0.90	Used in Sarsa(λ) update rule. Encodes the trade-off between bias and variance.
Eligibility threshold	0.01	We set to 0 any value in the eligibility trace vector that becomes smaller than this threshold.
Feature set	Blob-PROST	Originally introduced by Liang et al. (2016), Blob-PROST stands for Blob Pairwise Relative Offsets in Space and Time.
Color Averaging	True	The observation received is the average between the previous and the current frame.
Grid width	4	Each row of the game screen is divided into 40 tiles that are 4 pixels wide each.
Grid height	7	Each column of the game screen is divided into 30 tiles that are 7 pixels high each.
Neighborhood size	6	Tolerance used to detect blobs.
Number of different colors	128	NTSC is the color palette in which each screen is encoded.

Table 7: Parameters used in the experiments evaluating Sarsa(λ) + Blob-PROST (Liang et al., 2016). See reference for more details about the parameters listed below.

Game	10M frames		50M frames		100M frames		200M frames	
Alien	1,910.2	(557.4)	3,255.3	(562.8)	3,753.5	(712.0)	4,272.7	(773.2)
Amidar	210.4	(42.6)	332.3	(64.6)	414.6	(84.2)	411.4	(177.4)
Assault	435.9	(94.8)	651.9	(148.7)	851.7	(185.4)	1,049.4	(182.7)
Asterix†	2,146.8	(364.8)	3,417.8	(445.3)	3,767.8	(354.9)	4,358.0	(431.6)
Asteroids	1,350.1	(259.5)	1,378.1	(233.0)	1,443.4	(218.1)	1,524.1	(191.2)
Atlantis	39,731.2	(8,187.9)	41,833.5	(23,356.0)	36,289.2	(8,868.5)	38,057.5	(8,455.2)
Bank Heist	256.2	(66.6)	357.6	(72.1)	394.8	(64.8)	419.7	(60.5)
Battle Zone	11,009.2	(4,417.2)	19,178.3	(3,293.4)	22,419.2	(4,204.4)	25,089.6	(4,845.9)
Beam Rider†	1,200.2	(242.9)	1,859.2	(391.9)	2,126.1	(523.7)	2,234.0	(471.5)
Berzerk	473.5	(82.1)	542.5	(84.4)	572.1	(70.2)	622.3	(70.1)
Bowling	62.2	(5.7)	61.7	(3.5)	62.9	(3.5)	64.4	(4.3)
Boxing	34.8	(13.4)	70.1	(15.2)	79.1	(9.7)	78.1	(16.5)
Breakout†	12.3	(1.5)	16.8	(1.5)	18.6	(1.7)	20.2	(1.9)
Carnival	2,206.2	(855.7)	4,207.5	(857.7)	4,959.7	(935.9)	3,489.8	(2,621.5)
Centipede	8,226.7	(950.4)	12,968.2	(1,492.9)	15,599.6	(1,341.1)	1,189.3	(1,040.4)
Chopper Comm.	1,647.5	(389.2)	2,080.9	(562.3)	2,319.8	(725.7)	2,402.8	(806.5)
Crazy Climber	32,518.8	(3,868.1)	49,041.2	(5,015.8)	55,184.2	(5,559.2)	60,471.0	(5,534.9)
Defender	5,775.3	(890.9)	7,343.6	(1,607.2)	8,863.3	(1,380.2)	10,778.6	(1,509.0)
Demon Attack	385.4	(144.8)	628.9	(96.9)	921.5	(91.5)	1,272.2	(253.6)
Double Dunk	-10.6	(1.4)	-8.5	(0.8)	-7.6	(0.6)	-6.9	(0.5)
Elevator Action	3,228.9	(4,415.4)	8,797.3	(5,832.1)	9,981.8	(5,310.2)	11,147.8	(4,291.3)
Enduro†	120.3	(49.8)	241.3	(28.6)	275.1	(13.0)	294.0	(8.0)
Fishing Derby	-87.4	(4.9)	-76.5	(6.3)	-73.2	(6.7)	-69.2	(8.9)
Freeway†	29.9	(1.6)	31.8	(0.3)	31.9	(0.3)	31.8	(0.3)
Frostbite	1,375.0	(939.1)	2,470.7	(1,241.6)	2,815.3	(1,218.0)	3,207.2	(1,040.4)
Gopher	2,961.1	(495.3)	4,631.9	(454.0)	5,259.9	(535.2)	5,555.4	(594.1)
Gravitar	629.8	(201.5)	863.5	(255.6)	979.5	(340.1)	1,150.0	(397.5)
H.E.R.O.	9,452.6	(2,433.1)	12,909.6	(2,686.0)	14,072.7	(3,382.5)	14,910.2	(3,887.6)
Ice Hockey	-2.2	(1.2)	3.5	(2.1)	8.2	(3.1)	12.6	(3.5)
James Bond	461.1	(187.4)	599.1	(230.2)	659.1	(243.5)	719.8	(292.0)
Journey Escape	-5,592.9*	(1,253.2)*	-5,121.6*	(5,952.6)*	-4,654.0*	(5,446.3)*	-2,338.9*	(952.8)*
Kangaroo	1,305.6	(555.7)	2,442.5	(1,282.4)	3,152.8	(1,546.3)	4,225.8	(2,046.9)
Krull	4,922.1	(1,703.7)	6,762.2	(5,168.9)	7,491.5	(5,823.9)	8,894.9	(8,482.7)
Kung-Fu Master	20,679.3	(2,246.8)	23,548.3	(2,926.8)	26,745.6	(3,281.5)	29,915.5	(3,647.5)
Mont. Revenge	117.1	(175.3)	520.8	(486.7)	567.7	(588.9)	574.2	(590.1)
Ms. Pac-Man	2,626.7	(521.1)	3,446.0	(462.9)	3,916.6	(542.5)	4,440.5	(616.4)
Name This Game	4,626.2	(284.3)	6,164.2	(357.0)	6,219.8	(1,821.2)	6,750.3	(1,376.5)
Phoenix	2,319.4	(534.0)	4,579.9	(303.9)	4,247.3	(1,360.1)	5,197.0	(374.5)
Pitfall!	-0.3	(1.2)	-0.1	(0.3)	0.0	(0.0)	0.0	(0.0)
Pong†	1.8	(3.9)	10.9	(3.3)	12.6	(2.8)	14.5	(2.0)
Pooyan	1,347.2	(121.5)	1,820.5	(107.5)	2,006.7	(159.4)	2,197.8	(133.7)
Private Eye	36.7	(46.3)	36.2	(49.2)	27.9	(44.9)	44.2	(49.3)
Q*bert†	3,535.9	(745.2)	4,605.7	(567.3)	5,931.9	(1,174.4)	6,992.9	(1,479.0)
River Raid	4,141.9	(574.4)	7,399.5	(492.5)	8,988.3	(1,154.3)	10,639.2	(1,882.6)
Road Runner	18,258.0	(2,876.9)	23,380.2	(5,940.3)	28,453.4	(3,227.4)	31,493.9	(4,160.7)
Robotank	21.3	(3.4)	25.0	(2.8)	26.3	(2.7)	27.3	(2.3)
Seaquest†	788.2	(225.2)	1,201.6	(178.4)	1,319.2	(356.5)	1,402.9	(328.3)
Skiing	-29,965.4	(59.9)	-29,955.6	(50.7)	-29,955.9	(57.7)	-29,940.2	(102.2)
Solaris	480.9	(185.1)	585.5	(213.3)	704.2	(264.3)	807.3	(216.2)
Space Invaders†	466.2	(30.4)	579.1	(36.5)	656.7	(67.6)	759.4	(58.7)
Star Gunner	1,002.1	(64.6)	1,014.1	(72.7)	1,058.1	(101.0)	1,107.6	(125.6)
Tennis	-0.1	(0.1)	-0.1	(0.0)	-0.1	(0.0)	-0.1	(0.0)
Time Pilot	3,439.5	(503.4)	3,997.8	(436.4)	4,112.0	(289.4)	4,221.5	(402.1)
Tutankham	122.2	(3.5)	152.7	(16.0)	88.9	(64.1)	91.5	(63.3)
Up and Down	10,580.2	(3,446.4)	10,049.1	(9,340.7)	11,514.5	(11,988.8)	15,400.1	(14,864.6)
Venture	0.0	(0.0)	0.0	(0.0)	53.8	(263.7)	139.3	(323.2)
Video Pinball	11,271.1	(1,142.7)	13,259.3	(1,327.2)	14,334.7	(1,097.4)	13,398.0	(3,643.7)
Wizard of Wor	1,975.9	(471.4)	2,738.8	(613.3)	3,247.5	(713.0)	2,043.5	(801.3)
Yar's Revenge	4,961.2	(1,200.2)	5,460.2	(1,145.0)	6,073.9	(1,052.9)	7,257.8	(1,884.8)
Zaxxon	1,180.9	(618.8)	4,539.6	(1,401.0)	6,701.4	(1,974.3)	8,166.8	(3,979.8)

Table 8: Sarsa(λ) + Blob-PROST results across 60 games. See Appendix B for details.

Game	10M frames		50M frames		100M frames		200M frames	
Alien	600.5	(23.6)	1,426.6	(81.6)	1,952.6	(216.0)	2,742.0	(357.5)
Amidar	91.6	(10.5)	414.2	(53.6)	621.6	(92.6)	792.6	(220.4)
Assault	688.9	(16.0)	1,327.5	(83.9)	1,433.9	(126.6)	1,424.6	(106.8)
Asterix†	1,732.6	(314.6)	3,122.6	(96.4)	3,423.4	(213.6)	2,866.8	(1,354.6)
Asteroids	301.4	(14.3)	458.1	(28.5)	458.0	(18.9)	528.5	(37.0)
Atlantis	6,639.4	(208.4)	51,324.4	(8,681.7)	291,134.7	(31,575.2)	232,442.9	(128,678.4)
Bank Heist	32.3	(6.5)	448.2	(104.8)	740.7	(130.6)	760.0	(82.3)
Battle Zone	2,428.3	(200.4)	10,838.4	(1,807.6)	15,048.5	(2,372.0)	20,547.5	(1,843.0)
Beam Rider †	693.9	(111.0)	4,551.5	(849.1)	4,977.2	(292.2)	5,700.5	(362.5)
Berzerk	434.5	(51.2)	457.5	(9.4)	470.0	(24.5)	487.2	(29.9)
Bowling	28.7	(0.8)	29.4	(1.8)	32.8	(3.6)	33.6	(2.7)
Boxing	18.6	(3.8)	71.7	(2.7)	77.9	(0.5)	72.7	(4.9)
Breakout	14.2	(1.2)	75.1	(4.3)	57.9	(14.6)	35.1	(22.6)
Carnival	588.5	(47.0)	2,131.6	(534.3)	4,621.9	(191.0)	4,803.8	(189.0)
Centipede	3,075.2	(381.1)	2,280.0	(184.2)	2,555.2	(195.1)	2,838.9	(225.3)
Chopper Comm.	841.4	(144.3)	2,104.8	(327.7)	3,288.1	(339.2)	4,399.6	(401.5)
Crazy Climber	43,716.6	(2,571.2)	80,599.6	(4,209.8)	64,807.3	(26,100.0)	78,352.1	(1,967.3)
Defender	2,409.9	(78.6)	2,525.7	(124.0)	2,711.6	(96.8)	2,941.3	(106.2)
Demon Attack	154.8	(11.5)	3,744.6	(688.9)	4,556.5	(947.2)	5,182.0	(778.0)
Double Dunk	-20.9	(0.3)	-18.4	(1.2)	-15.6	(1.6)	-8.7	(4.5)
Elevator Action	6.7	(13.3)	4.5	(9.0)	4.7	(9.4)	6.0	(10.4)
Enduro	473.2	(22.3)	578.0	(79.6)	597.4	(153.1)	688.2	(32.4)
Fishing Derby	-63.1	(7.8)	7.5	(4.1)	12.2	(1.4)	10.2	(1.9)
Freeway†	13.8	(8.1)	31.7	(0.7)	32.4	(0.3)	33.0	(0.3)
Frostbite	241.8	(30.8)	292.5	(28.8)	274.3	(8.8)	279.6	(13.9)
Gopher	679.6	(35.2)	2,233.7	(123.1)	2,988.8	(514.4)	3,925.5	(521.4)
Gravitar	79.5	(8.0)	109.3	(3.1)	118.5	(22.0)	154.9	(17.7)
H.E.R.O.	1,667.9	(1,107.8)	11,564.0	(3,722.4)	14,684.7	(1,840.6)	18,843.3	(2,234.9)
Ice Hockey	-15.1	(0.3)	-8.9	(1.7)	-4.4	(2.0)	-3.8	(4.7)
James Bond	30.7	(6.0)	191.4	(144.9)	517.2	(35.8)	581.0	(21.3)
Journey Escape	-2,220.0	(176.1)	-2,409.7	(341.2)	-2,959.0	(383.9)	-3,503.0	(488.5)
Kangaroo	298.6	(56.1)	8,878.8	(2,886.1)	12,846.9	(688.3)	12,291.7	(1,115.9)
Krull	4,424.7	(492.7)	6,035.6	(248.6)	6,589.8	(264.4)	6,416.0	(128.5)
Kung-Fu Master	9,468.1	(1,975.9)	17,537.4	(1,128.8)	17,772.3	(3,423.3)	16,472.7	(2,892.7)
Mont. Revenge	0.2	(0.4)	0.2	(0.4)	0.0	(0.0)	0.0	(0.0)
Ms. Pac-Man	1,675.5	(41.9)	2,626.1	(139.8)	2,964.9	(100.8)	3,116.2	(141.2)
Name This Game	2,265.6	(171.0)	4,105.4	(932.3)	4,105.6	(653.5)	3,925.2	(660.2)
Phoenix	1,501.2	(278.1)	3,174.0	(543.5)	2,607.1	(644.1)	2,831.0	(581.0)
Pitfall!	-24.9	(14.8)	-28.2	(13.0)	-23.3	(9.6)	-21.4	(3.2)
Pong	-15.9	(1.0)	12.2	(1.0)	15.2	(0.7)	15.1	(1.0)
Pooyan	2,278.9	(273.7)	3,528.9	(256.3)	3,387.8	(182.8)	3,700.4	(349.5)
Private Eye	81.6	(15.6)	60.4	(92.4)	1,447.4	(2,567.9)	3,967.5	(5,540.6)
Q*bert	674.7	(53.6)	3,142.1	(1,238.7)	7,585.4	(2,787.4)	9,875.5	(1,385.3)
River Raid	3,166.2	(125.2)	8,738.1	(500.0)	10,733.1	(229.9)	10,210.4	(435.0)
Road Runner	14,742.2	(1,553.4)	37,271.7	(1,234.5)	41,918.4	(1,762.5)	42,028.3	(1,492.0)
Robotank	4.1	(0.3)	28.4	(1.4)	38.0	(1.6)	58.0	(6.4)
Seaquest†	311.5	(36.9)	1,430.8	(162.3)	1,573.4	(561.4)	1,485.7	(740.8)
Skiing	-20,837.5	(1,550.2)	-17,545.5	(4,041.5)	-13,365.1	(800.7)	-12,446.6	(1,257.9)
Solaris	1,030.2	(40.3)	977.7	(112.5)	783.4	(55.3)	1,210.0	(148.3)
Space Invaders†	211.6	(14.8)	686.6	(37.0)	787.2	(173.3)	823.6	(335.0)
Star Gunner	603.0	(28.0)	1,492.3	(79.7)	11,590.5	(4,658.9)	39,269.9	(5,298.8)
Tennis	-23.8	(0.1)	-23.9	(0.1)	-23.9	(0.0)	-23.9	(0.0)
Time Pilot	1,078.8	(60.3)	1,068.1	(138.8)	1,330.7	(177.1)	2,061.8	(228.8)
Tutankham	56.5	(10.0)	64.9	(12.6)	65.1	(11.9)	60.0	(12.7)
Up and Down	4,378.4	(172.5)	6,718.3	(671.2)	5,962.8	(618.7)	4,750.7	(1,007.5)
Venture	24.4	(46.9)	21.4	(15.1)	4.4	(5.4)	3.2	(4.7)
Video Pinball	4,009.3	(271.9)	7,817.0	(1,884.4)	16,626.2	(3,740.6)	15,398.5	(2,126.1)
Wizard of Wor	184.2	(22.0)	1,377.4	(71.0)	1,440.6	(237.3)	2,231.1	(820.8)
Yar's Revenge	7,261.4	(777.1)	10,344.8	(452.4)	10,312.3	(528.9)	13,073.4	(1,961.8)
Zaxxon	53.5	(51.0)	672.3	(748.5)	1,638.2	(784.0)	3,852.1	(1,120.7)

Table 9: DQN results across 60 games. See Appendix B for details.

Appendix D. Number of Game Modes and Difficulties in the Games Supported by the Arcade Learning Environment

Game	# Modes	# Diffic.	Game	# Modes	# Diffic.
Alien	4	4	Journey Escape	1	2
Amidar	1	2	Kangaroo	2	1
Assault	1	1	Krull	4	1
Asterix	1	1	Kung Fu Master	1	1
Asteroids	33	2	Montezuma Revenge	1	1
Atlantis	4	1	Ms. Pac-Man	4	1
Bank Heist	8	4	Name this Game	3	2
Battle Zone	3	1	Phoenix	1	1
Beam Rider	1	2	Pitfall	1	1
Berzerk	12	1	Pong	2	2
Bowling	3	2	Pooyan	4	1
Boxing	1	4	Private Eye	5	4
Breakout	12	2	Q*bert	1	2
Carnival	1	1	River Raid	1	2
Centipede	2	1	Road Runner	1	1
Chopper Command	2	2	Robot Tank	1	1
Crazy Climber	4	2	Seaquest	1	2
Defender	10	2	Skiing	10	1
Demon Attack	4	2	Solaris	1	1
Double Dunk	16	1	Space Invader	16	2
Elevator Action	1	1	Star Gunner	4	1
Enduro	1	1	Tennis	2	4
Fishing Derby	1	4	Time Pilot	1	3
Freeway	8	2	Tutankham	4	1
Frostbite	2	1	UpNDown	1	4
Gopher	2	2	Venture	1	4
Gravitar	5	1	Video Pinball	2	2
Hero	5	1	Wizard Of Wor	1	2
Ice Hockey	2	4	Yar's Revenge	4	2
James Bond	2	1	Zaxxon	4	1

Table 10: Atari 2600 games supported by the Arcade Learning Environment and the respective number of modes and difficulties available in each game. Modes only playable by two-players have been excluded.

References

Albrecht, S. V., L., J. C., Buckeridge, D. L., Botea, A., Caragea, C., Chi, C., Damoulas, T., Dilkina, B. N., Eaton, E., Fazli, P., Ganzfried, S., Lindauer, M. T., Machado, M. C., Malitsky, Y., Marcus, G., Meijer, S., Rossi, F., Shaban-Nejad, A., Thiebaux, S., Veloso, M. M., Walsh, T., Wang, C., Zhang, J., & Zheng, Y. (2015). Reports on the 2015 AAAI Workshop Program. *AI Magazine*, *36*(2), 90–101.

Barto, A. G. (2013). Intrinsic Motivation and Reinforcement Learning. In *Intrinsically Motivated Learning in Natural and Artificial Systems*, pp. 17–47. Springer.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., & Petersen, S. (2016). DeepMind Lab. *CoRR*, *abs/1612.03801*.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A Distributional Perspective on Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 449–458.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2015). The Arcade Learning Environment: An Evaluation Platform for General Agents (Extended Abstract). In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4148–4152.

Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P. S., & Munos, R. (2016a). Increasing the Action Gap: New Operators for Reinforcement Learning. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 1476–1483.

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., & Munos, R. (2016b). Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1471–1479.

Bellemare, M. G., Veness, J., & Bowling, M. (2012a). Investigating Contingency Awareness using Atari 2600 Games. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 864–871.

Bellemare, M. G., Veness, J., & Bowling, M. (2012b). Sketch-Based Linear Value Function Approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2222–2230.

Bellemare, M. G., Veness, J., & Bowling, M. (2013). Bayesian Learning of Recursively Factored Environments. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1211–1219.

Bellemare, M. G., Veness, J., & Talvitie, E. (2014). Skip Context Tree Switching. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1458–1466.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, *3*, 213–231.

Braylan, A., Hollenbeck, M., Meyerson, E., & Miikkulainen, R. (2015). Frame Skip is a Powerful Parameter for Learning to Play Atari. In *AAAI Workshop on Learning for General Competency in Video Games*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *CoRR*, *abs/1507.04296*.

Campbell, M., Hoane, Jr., A. J., & Hsu, F.-h. (2002). Deep Blue. *Artificial Intelligence*, *134*(1-2), 57–83.

Chiappa, S., Racaniere, S., Wierstra, D., & Mohamed, S. (2017). Recurrent Environment Simulators. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Defazio, A., & Graepel, T. (2014). A Comparison of Learning Algorithms on the Arcade Learning Environment. *CoRR*, *abs/1410.8620*.

Even-Dar, E., Kakade, S. M., & Mansour, Y. (2005). Reinforcement Learning in POMDPs Without Resets. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 690–695.

Farahmand, A.-M., Barreto, A., & Nikovski, D. (2017). Value-Aware Loss Function for Model-based Reinforcement Learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1486–1494.

Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3338–3346.

Harutyunyan, A., Bellemare, M. G., Stepleton, T., & Munos, R. (2016). Q(λ) with Off-Policy Corrections. In *Proceedings of the International Conference on Algorithmic Learning Theory (ALT)*, pp. 305–320.

Hausknecht, M., & Stone, P. (2015). The Impact of Determinism on Learning Atari 2600 Games. In *AAAI Workshop on Learning for General Competency in Video Games*.

Hausknecht, M. J., Lehman, J., Miikkulainen, R., & Stone, P. (2014). A Neuroevolution Approach to General Atari Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games*, *6*(4), 355–366.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J., & Gruslys, A. (2018). Deep Q-learning from Demonstrations. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*.

Hutter, M. (2005). *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer.

Islam, R., Henderson, P., Gomrokchi, M., & Precup, D. (2017). Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. In *ICML Workshop on Reproducibility in Machine Learning*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2017). Reinforcement Learning with Unsupervised Auxiliary Tasks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Jinnai, Y., & Fukunaga, A. (2017). Learning to Prune Dominated Action Sequences in Online Black-Box Planning. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 839–845.

Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The Malmo Platform for Artificial Intelligence Experimentation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4246–4247.

Kearns, M. J., Mansour, Y., & Ng, A. Y. (1999). Approximate Planning in Large POMDPs via Reusable Trajectories. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1001–1007.

Kearns, M. J., & Singh, S. P. (2002). Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, *49*(2-3), 209–232.

Kelly, S., & Heywood, M. I. (2017). Emergent Tangled Graph Representations for Atari Game Playing Agents. In *European Conference on Genetic Programming (EuroGP)*, pp. 64–79.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1106–1114.

Levine, J., Congdon, C. B., Ebner, M., Kendall, G., Lucas, S. M., Miikkulainen, R., Schaul, T., & Thompson, T. (2013). General Video Game Playing. In *Dagstuhl Follow-Ups*.

Liang, Y., Machado, M. C., Talvitie, E., & Bowling, M. H. (2016). State of the Art Control of Atari Games Using Shallow Reinforcement Learning. In *Proceedings of the International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pp. 485–493.

Lin, L.-J. (1993). Reinforcement Learning for Robots Using Neural Networks. Tech. rep., Carnegie Mellon University, School of Computer Science.

Lipovetzky, N., Ramirez, M., & Geffner, H. (2015). Classical Planning with Simulators: Results on the Atari Video Games. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1610–1616.

Machado, M. C., Bellemare, M. G., & Bowling, M. (2017). A Laplacian Framework for Option Discovery in Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2295–2304.

Machado, M. C., Srinivasan, S., & Bowling, M. (2015). Domain-Independent Optimistic Initialization for Reinforcement Learning. In *AAAI Workshop on Learning for General Competency in Video Games*.

Maei, H. R., & Sutton, R. S. (2010). GQ(λ): A General Gradient Algorithm for Temporal-Difference Prediction Learning with Eligibility Traces. In *Proceedings of the Conference on Artificial General Intelligence (AGI)*, pp. 719–726.

Martin, J., Sasikumar, S. N., Everitt, T., & Hutter, M. (2017). Count-Based Exploration in Feature Space for Reinforcement Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2471–2478.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level Control through Deep Reinforcement Learning. *Nature*, *518*(7540), 529–533.

Montfort, N., & Bogost, I. (2009). *Racing the Beam: The Atari Video Computer System*. MIT Press.

Munos, R., Stepleton, T., Harutyunyan, A., & Bellemare, M. G. (2016). Safe and Efficient Off-Policy Reinforcement Learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1046–1054.

Naddaf, Y. (2010). Game-independent AI Agents for Playing Atari 2600 Console Games. Master's thesis, University of Alberta.

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., Maria, A. D., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., & Silver, D. (2015). Massively Parallel Methods for Deep Reinforcement Learning. In *ICML Deep Learning Workshop*.

Oh, J., Guo, X., Lee, H., Lewis, R. L., & Singh, S. P. (2015). Action-Conditional Video Prediction using Deep Networks in Atari Games. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2863–2871.

Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., & Preuss, M. (2013). A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games*, *5*, 293–311.

Osband, I., Blundell, C., Pritzel, A., & Roy, B. V. (2016). Deep Exploration via Bootstrapped DQN. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 4026–4034.

Ostrovski, G., Bellemare, M. G., van den Oord, A., & Munos, R. (2017). Count-Based Exploration with Neural Density Models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2721–2730.

Oudeyer, P., Kaplan, F., & Hafner, V. (2007). Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation*, *11*(2), 265–286.

Parisotto, E., Ba, L. J., & Salakhutdinov, R. (2016). Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Ring, M. (1997). CHILD: A First Step Towards Continual Learning. *Machine Learning*, *28*(1), 77–104.

Rummery, G. A., & Niranjan, M. (1994). On-line Q-Learning using Connectionist Systems. CUED/F-INFENG/TR 166, Cambridge University Engineering Department.

Rusu, A. A., Colmenarejo, S. G., Gucehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., & Hadsell, R. (2016). Policy Distillation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Schaeffer, J., Burch, N., Bjrnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., & Sutphen, S. (2007). Checkers is Solved. *Science*, *317*(5844), 1518–1522.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized Experience Replay. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Shleyfman, A., Tuisov, A., & Domshlak, C. (2016). Blind Search for Atari-Like Online Planning Revisited. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3251–3257.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, *529*, 484–503.

Singh, S., Barto, A. G., & Chentanez, N. (2004). Intrinsically Motivated Reinforcement Learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1281–1288.

Stadie, B. C., Levine, S., & Abbeel, P. (2015). Incentivizing Exploration in Reinforcement Learning With Deep Predictive Models. *CoRR*, *abs/1507.00814*.

Strehl, A. L., & Littman, M. L. (2008). An Analysis of Model-Based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, *74*(8), 1309–1331.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., Szepesvári, C., & Maei, H. R. (2008). A Convergent O(n) Temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1609–1616.

Sutton, R., Modayil, J., Delp, M., Degris, T., Pilarski, P., White, A., & Precup, D. (2011). Horde: A Scalable Real-Time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *Proceedings of the International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pp. 761–768.

Talvitie, E. (2014). Model Regularization for Stable Sample Rollouts. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 780–789.

Talvitie, E. (2017). Self-Correcting Models for Model-Based Reinforcement Learning. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 2597–2603.

Taylor, M. E., & Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, *10*, 1633–1685.

Thrun, S., & Mitchell, T. M. (1993). Lifelong Robot Learning. In *Robotics and Autonomous Systems*.

van Hasselt, H., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 2094–2100.

Veness, J., Bellemare, M. G., Hutter, M., Chua, A., & Desjardins, G. (2015). Compress and Control. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 3016–3023.

Venkatraman, A., Hebert, M., & Bagnell, J. A. (2015). Improving Multi-Step Prediction of Learned Time Series Models. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 3024–3030.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., & Kavukcuoglu, K. (2017). FeUdal Networks for Hierarchical Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 3540–3549.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & De Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1995–2003.

Wilson, S. (1985). Knowledge Growth in an Artificial Animal. In *Proceedings of the International Conference on Genetic Algorithms (ICGA)*, pp. 16–23.

Zahavy, T., Ben-Zrihem, N., & Mannor, S. (2016). Graying the Black Box: Understanding DQNs. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1899–1908.