The Arcade Learning Environment: An Evaluation Platform For General Agents (Extended Abstract)*

Marc G. Bellemare†
University of Alberta
mg17@cs.ualberta.ca

Yavar Naddaf
Empirical Results Inc.
yavar@empiricalresults.ca

Joel Veness† and **Michael Bowling**
University of Alberta
{veness,bowling}@cs.ualberta.ca

Abstract

In this extended abstract we introduce the Arcade Learning Environment (ALE): both a challenge problem and a platform and methodology for evaluating the development of general, domain-independent AI technology. ALE provides an interface to hundreds of Atari 2600 game environments, each one different, interesting, and designed to be a challenge for human players. ALE presents significant research challenges for reinforcement learning, model learning, model-based planning, imitation learning, transfer learning, and intrinsic motivation. Most importantly, it provides a rigorous testbed for evaluating and comparing approaches to these problems. We illustrate the promise of ALE by presenting a benchmark set of domain-independent agents designed using well-established AI techniques for both reinforcement learning and planning. In doing so, we also propose an evaluation methodology made possible by ALE, reporting empirical results on over 55 different games. We conclude with a brief update on the latest ALE developments. All of the software, including the benchmark agents, is publicly available.

1 Introduction

A longstanding goal of artificial intelligence is the development of algorithms capable of general competency in a variety of tasks and domains without the need for domain-specific tailoring. To this end, different theoretical frameworks have been proposed to formalize the notion of "big" artificial intelligence [Russell, 1997; Hutter, 2005; Legg, 2008]. The growing interest in competitions such as the General Game Playing competition [Genesereth *et al.*, 2005], Reinforcement Learning competition [Whiteson *et al.*, 2010], and the International Planning competition [Coles *et al.*, 2012] also clearly reflects a desire for practical general competency.

Designing generally competent agents raises the question of how to best evaluate them. Empirically evaluating general competency on a handful of parametrized benchmark problems is, by definition, flawed. Such an evaluation is prone to method overfitting [Whiteson *et al.*, 2011] and discounts the amount of expert effort necessary to transfer the algorithm to new domains. Ideally, the algorithm should be compared across domains that are (i) *varied* enough to claim generality, (ii) each *interesting* enough to be representative of settings that might be faced in practice, and (iii) each created by an *independent* party to be free of experimenter's bias.

In this article, we introduce the Arcade Learning Environment (ALE): a new challenge problem, platform, and experimental methodology for empirically assessing agents designed for general competency. ALE is a software framework for interfacing with emulated Atari 2600 game environments. While general competency remains the long-term goal for artificial intelligence, ALE proposes an achievable stepping stone: techniques for general competency across the gamut of Atari 2600 games. We believe this represents a goal that is attainable in a short time-frame yet formidable enough to require new technological breakthroughs.

2 Arcade Learning Environment

We begin by describing our main contribution, the Arcade Learning Environment (ALE). ALE is a software framework designed to facilitate the development of agents that play arbitrary Atari 2600 games.

2.1 The Atari 2600

The Atari 2600 is a home video game console developed in 1977 and sold for over a decade [Montfort and Bogost, 2009]. It popularized the use of general purpose CPUs in game console hardware, with game code distributed through cartridges. Over 500 original games were released for the console; nearly all arcade games of the time – including PAC-MAN and SPACE INVADERS – were ported to the console.

Despite the number and variety of games developed for the Atari 2600, its hardware is relatively simple. It has a 1.19Mhz CPU and can be emulated much faster than real-time on modern hardware. The cartridge ROM (typically 2–4kB) holds the game code, while the console RAM itself only holds 128 bytes (1024 bits). A single game screen is 160 pixels wide and 210 pixels high, with a 128-colour palette; 18 "actions" can be input to the game via a digital joystick: three positions of the joystick for each axis, plus a single button. The

*This is an extended abstract of a paper published in the Journal of Artificial Intelligence Research [Bellemare *et al.*, 2013].

†These authors are now at Google DeepMind.

Figure 1: Screenshot of PITFALL!.

Atari 2600 hardware limits the possible complexity of games, which we believe strikes the perfect balance: a challenging platform offering conceivable near-term advancements in learning, modelling, and planning.

2.2 Interface

ALE is built on top of Stella[1], an open-source Atari 2600 emulator. It allows the user to interface with the Atari 2600 by receiving joystick motions, sending screen and/or RAM information, and emulating the platform. ALE also provides a game-handling layer which transforms each game into a standard reinforcement learning problem by identifying the accumulated score and whether the game has ended. The action space consists of the 18 discrete actions defined by the joystick controller. When running in real-time, the simulator generates 60 frames per second, and at full speed emulates up to 6000 frames per second. The reward at each time-step is defined on a game by game basis, typically by taking the difference in score or points between frames. An episode begins on the first frame after a reset command is issued, and terminates when the game ends or after a predefined number of frames[2]. The user therefore has access to several dozen games through a single common interface, and adding support for new games is relatively straightforward.

ALE further provides the functionality to save and restore the state of the emulator. When issued a *save-state* command, ALE saves all the relevant data about the current game, including the contents of the RAM, registers, and address counters. The *restore-state* command similarly resets the game to a previously saved state. This allows the use of ALE as a generative model to study topics such as planning and model-based reinforcement learning.

2.3 Source Code

ALE is released as free, open-source software under the terms of the GNU General Public License. The latest version of the source code is publicly available at:

http://arcadelearningenvironment.org

The source code for the agents used in the benchmark experiments below is also available on the publication page for this article on the same website. While ALE itself is written in C++, a variety of interfaces are available that allow users to interact with ALE in the programming language of their choice. Support for new games is easily added by implementing a derived class representing the game's particular reward and termination functions.

[1]http://stella.sourceforge.net

[2]The latter is to prevent situations such as in TENNIS, where a degenerate agent can choose to play indefinitely by refusing to serve.

3 Benchmark Results

Planning and reinforcement learning are two different AI problem formulations that can naturally be investigated within the ALE framework. The purpose of the empirical results presented in this paper is twofold: first, to established a point of comparison with more advanced approaches; second, to illustrate our proposed methodology for empirical validation with ALE.

3.1 Reinforcement Learning

In our experiments we used SARSA(λ), a model-free reinforcement learning technique, in combination with linear function approximation and ϵ-greedy exploration [see Sutton and Barto, 1998, for details]. We compared the following feature generation methods:

Basic. A binary encoding of the presence of colours within the Atari 2600 screen.

BASS. The Basic method augmented with pairwise feature combinations. A restricted colour space (3-bit) is used to minimize the size of the resulting feature set.

DISCO. A heuristic object detection pipeline trained on offline data.

LSH. A method encoding the Atari 2600 screen into a small set of binary features via Locally Sensitive Hashing [Gionis *et al.*, 1999].

RAM. An encoding of the Atari 2600's 1024 bits of memory together with pairwise combinations.

Evaluation Methodology

Agents were designed using five training games and further evaluated on 50 testing games. The training games were also used for finding suitable parameter values for the SARSA(λ) algorithm and the feature generation methods. The testing games were chosen semi-randomly from 381 games listed on Wikipedia when ALE was first developed.

Evaluation was performed by dividing gameplay into episodes, as described in Section 2.2, with the maximum episode length set to 5 minutes of real-time play (18,000 frames). Each method was evaluated over 30 trials, with each trial composed of 5,000 training episodes followed by 500 evaluation episodes during which no learning took place. The agent's performance was measured as the average score achieved during this evaluation period.

Results

Table 1 shows illustrative results from two training games (ASTERIX and SEAQUEST) and three test games. For purposes of comparison, we also provide performance measures for three baseline agents: *Random*, *Constant*, and *ϵ-Constant*, chosen for the simplicity of their associated policies. For the results in Table 1 we also provide the performance of a beginner human player averaged over five episodes. Of note,

	Basic	BASS	DISCO	LSH	RAM	Random	Const.	ϵ-Const.	Human
ASTERIX	862	860	755	**987**	943	288	650	338	620
SEAQUEST	579	**665**	422	509	594	108	160	451	156
BOXING	-3	16	12	10	**44**	-1	-25	-10	-2
H.E.R.O.	6,053	**6,459**	2,720	3,836	3,281	712	0	148	6,087
ZAXXON	1,392	2,069	70	**3,365**	304	0	0	2	820

Table 1: Reinforcement learning results for selected games.

learning agents perform statistically significantly better than baseline agents on **40** games out of **55**.

Our comprehensive empirical results (found in the full paper) show that, while even simple learning schemes perform well in Atari 2600 games, much more work remains to be done. Different methods perform well on different games, and no single method performs best on all games. Some games, such as MONTEZUMA'S REVENGE, are particularly challenging and require high-level planning far beyond what current domain-independent methods achieve.

Our results also highlight the value of ALE as an experimental methodology. As an example, the DISCO object detection approach performs reasonably well on the training set, but suffers a dramatic reduction in performance when applied to the testing set. In turn, this suggests the method is less robust than the other methods we studied.

3.2 Planning

The Arcade Learning Environment can naturally be used to study planning techniques by using the emulator itself as a generative model. Initially it may seem that allowing the agent to plan into the future with a perfect model trivializes the problem. However, this is not the case: the size of state space in Atari 2600 games prohibits exhaustive search. Eighteen different actions are available at every frame; at 60 frames per second, looking ahead one second requires $18^{60} \approx 10^{75}$ simulation steps. Furthermore, rewards are often sparsely distributed, which causes significant horizon effects in many planning algorithms. In this paper we investigate two planning methods:

Breadth-first search. This approach builds a search tree in a uniform, breadth-first fashion until a node limit is reached.

UCT. UCT [Kocsis and Szepesvári, 2006] uses the UCB bandit algorithm to favour more promising branches in its exploration. UCT is known to perform well in many settings [Browne *et al.*, 2012].

Results

We matched the experimental setup of Section 3.1 and used the training games to determine the length of the planning horizon as well as the constant controlling the amount of exploration at internal nodes of the UCT tree. We evaluated both planning methods using 10 episodes per game. Table 2 provides a comparison of our two planning methods for a selected subset of games. For reference purposes, we also include the performance of the best learning agent and the best baseline policy (from Table 1). Together, our two search methods performed better than both learning and baseline agents on **49** out of **55** games. In most cases, UCT performs significantly better than breadth-first search. Note, however, that these results are not altogether comparable: while the learning agents all play in real-time, the planning agents typically required 15 seconds per action: effectively three minutes of search effort per second of game play.

	Full Tree	UCT	Best Learner
ASTERIX	2,136	**290,700**	987
SEAQUEST	288	**5,132**	665
BOXING	**100**	**100**	44
H.E.R.O.	1324	**12,860**	6,459
ZAXXON	0	**22,610**	3,365

Table 2: Planning results for selected games.

4 Evaluation Metrics for General Atari 2600 Agents

Applying algorithms to a large set of games as we did in Sections 3.1 and 3.2 presents difficulties when interpreting the results. While the agent's goal in all games is to maximize its score, scores for different games cannot be easily compared. Each game uses its own scale for scores, and different game mechanics make some games harder to learn than others. The challenges associated with comparing general agents has been previously highlighted by Whiteson *et al.* [2011]. Although we can always report comprehensive performance tables, as we do in the full paper, some more compact summary statistics are also desirable. We now introduce simple techniques to compare agents across a diverse set of domains such as our test set of Atari 2600 games.

4.1 Normalizing Scores

To compare agents across different games, we normalize each score according to a *score range* $[a_g, b_g]$ specific to game g, with $a_g < b_g$. If s_g is the score achieved by an agent on game g, then its normalized score is

$$z_g := \frac{s_g - a_g}{b_g - a_g}.$$

In this paper we consider three normalization schemes:

Normalization to a single reference. Here we define b_g to be the score achieved by the random player on game g, and set $a_g = 0$.

Normalization to a baseline set. We define b_g to be the maximum of the set of scores $\mathcal{S}_g := \{s_{g,i}\}$ achieved by our baseline policies, and take $a_g := \min \mathcal{S}_g$.

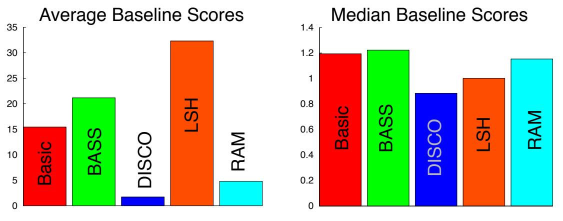

Figure 2: Aggregate normalized scores for the five reinforcement learning agents.

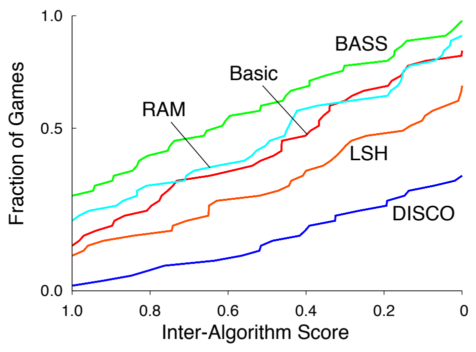

Figure 3: Score distribution over all games.

Inter-algorithm normalization. Identical to normalization to a baseline set, except that $\mathcal{S}_g$ is the set of scores achieved by the algorithms under comparison. By definition, the normalized scores are in the range $[0,1]$ and are easily compared. However, this approach has no objective interpretation, since scores depend on the choice of algorithms.

4.2 Aggregating Scores

Once normalized scores are obtained for each game, the next step is to produce a measure that reflects how well each agent performs across the set of games. In this paper we discuss how average and median normalized scores are difficult to interpret (Figure 2). We then propose the *score distribution* as a natural generalization of the median score.

The score distribution shows the fraction of games on which an algorithm achieves a certain normalized score or better. It is essentially a quantile plot or inverse empirical CDF (Figure 3). Using the baseline score distribution, we can easily determine the proportion of games for which methods perform better than the baseline policies (scores above 1). The inter-algorithm score distribution, on the other hand, effectively conveys the relative performance of each method.

An alternate evaluation metric is to perform paired tests over the raw scores. For each game, we performed a two-tailed Welsh's t-test with 99% confidence intervals on the agents' scores. These tests, like other metrics, suggest that DISCO performs worst and BASS, best.

5 New Developments: The Brute

The year following the original publication of this paper has seen a number of exciting developments, not the least Mnih *et al.*'s work on combining neural networks with reinforcement learning [Mnih *et al.*, 2015]. We also discovered a simple trajectory optimization algorithm, *the Brute*, which achieves high scores on a surprising number of Atari 2600 games. As the name implies, the Brute operates in a naive, seemingly incredibly inefficient way. Yet its good performance suggests improvements to ALE are needed to make it better suited to the evaluation of generally competent agents.

The Brute is both a planning and a learning method, and is tailored to deterministic, episodic environments. It is a planning method in that it constructs a search tree from simulations; however, it is also a learning method as each of its simulations is a full episode of interaction with the environment. Thus, the Brute does not require access to a generative model. Each node of its search tree corresponds to an action-observation sequence h. Save for some initial state aliasing, the Atari 2600 is deterministic: as such, a single transition suffices to learn $R(h,a)$, the reward for taking action a from node h, and $T(h,a)$, the transition function.

The Brute keeps track of the number of visits v_h to each node h and acts according to a modified ϵ-greedy policy π:

$$Q(h,a) = \begin{cases} -\infty & \text{if } T(h,a) \text{ is unknown} \\ R(h,a) + \max_b Q(T(h,a),b) & \text{otherwise} \end{cases}$$

$$\pi(h) := \begin{cases} \arg\max_a Q(h,a) & \text{w.p. } 1 - \frac{\epsilon}{\log(v_h+1)} \\ \text{uniformly random} & \text{w.p. } \frac{\epsilon}{\log(v_h+1)} \end{cases},$$

with terminal states always yielding a reward of 0, and $\epsilon = 0.005$. This kind of policy yields long, narrow search trees with are particularly adapted to Atari 2600 games. For similar amounts of training time, the Brute outperforms the best learning method on **45** out of **55** games, and is better than the best planning method on **11** games (Table 3 presents some particularly striking examples). These results are particularly significant since the Brute does not generalize to new situations, but simply optimizes a single trajectory. Unlike our planning methods, the Brute also acts in real-time.

	The Brute	Best Learner	Best Planner
BEAM RIDER	3,084	996	**6,625**
Q*BERT	5,850	614	**17,343**
MONTEZUMA	**2,500**	10.7	0.0
PRIVATE EYE	**14,891**	1947	100
ROAD RUNNER	15,600	900	**38,725**

Table 3: Brute results for selected games.

The Brute's good performance depends critically on the deterministic nature of ALE. However, this kind of trajectory optimization seems detrimental to ALE's purpose as an evaluation platform and methodology for general competency. As a result, we are currently incorporating a stochastic control element into ALE. We are also considering averaging performance over multiple starting positions. We believe this new, improved ALE will provide an even more interesting platform for AI research.

References

[Bellemare *et al.*, 2013] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, June 2013.

[Browne *et al.*, 2012] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[Coles *et al.*, 2012] A. Coles, A. Coles, A.G. Olaya, S. Jiménez, C.L. López, S. Sanner, and S. Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88, 2012.

[Diuk *et al.*, 2008] Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[Dowe and Hajek, 1998] D. L. Dowe and A. R. Hajek. A non-behavioural, computational extension to the Turing Test. In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications*, 1998.

[Genesereth *et al.*, 2005] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General Game Playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Databases*, 1999.

[Hausknecht *et al.*, 2012] Matthew Hausknecht, Piyush Khandelwal, Risto Miikkulainen, and Peter Stone. HyperNEAT-GGP: A HyperNEAT-based Atari general game player. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2012.

[Hernández-Orallo and Dowe, 2010] J. Hernández-Orallo and David L. Dowe. Measuring universal intelligence: Towards an anytime intelligence test. *Artificial Intelligence*, 174(18):1508 – 1539, 2010.

[Hernández-Orallo and Minaya-Collado, 1998] J. Hernández-Orallo and N. Minaya-Collado. A formal definition of intelligence based on an intensional variant of Kolmogorov complexity. In *Proceedings of the International Symposium of Engineering of Intelligent Systems*, 1998.

[Hutter, 2005] Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2005.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 15th European Conference on Machine Learning*, 2006.

[Legg and Veness, 2011] Shane Legg and Joel Veness. An approximation of the universal intelligence measure. In *Proceedings of the Ray Solomonoff Memorial Conference*, 2011.

[Legg, 2008] Shane Legg. *Machine Super Intelligence*. PhD thesis, University of Lugano, 2008.

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

[Mohan and Laird, 2009] Shiwali Mohan and John E. Laird. Learning to play Mario. Technical Report CCA-TR-2009-03, Center for Cognitive Architecture, University of Michigan, 2009.

[Monroy *et al.*, 2006] German A. Monroy, Kenneth O. Stanley, and Risto Miikkulainen. Coevolution of neural networks using a layered pareto archive. In *Proceedings of the 8th Genetic and Evolutionary Computation Conference*, 2006.

[Montfort and Bogost, 2009] Nick Montfort and Ian Bogost. *Racing the Beam: The Atari Video Computer System*. MIT Press, 2009.

[Naddaf, 2010] Yavar Naddaf. Game-Independent AI Agents for Playing Atari 2600 Console Games. Master's thesis, University of Alberta, 2010.

[Pierce and Kuipers, 1997] D. Pierce and B.J. Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92(1-2), 1997.

[Russell, 1997] Stuart J. Russell. Rationality and intelligence. *Artificial intelligence*, 94(1):57–77, 1997.

[Schaul *et al.*, 2011] Tom Schaul, Julian Togelius, and Jürgen Schmidhuber. Measuring intelligence through games. *ArXiv*, 1109.1314, 2011.

[Stober and Kuipers, 2008] Jeremy Stober and Benjamin Kuipers. From pixels to policies: A bootstrapping agent. In *Proceedings of the 7th IEEE International Conference on Development and Learning*, 2008.

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

[Sutton *et al.*, 2011] R.S. Sutton, J. Modayil, M. Delp, T. Degris, P.M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagents Systems (AAMAS)*, 2011.

[Thrun and Mitchell, 1995] Sebastian Thrun and Tom M. Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1), 1995.

[Watkins and Dayan, 1992] Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[Whiteson *et al.*, 2010] Shimon Whiteson, Brian Tanner, and Adam White. The reinforcement learning competitions. *AI Magazine*, 31(2):81–94, 2010.

[Whiteson *et al.*, 2011] Shimon Whiteson, Brian Tanner, Matthew E. Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. In *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2011.

[Wintermute, 2010] Samuel Wintermute. Using imagery to simplify perceptual abstraction in reinforcement learning agents. In *Proceedings of the the 24th Conference on Artificial Intelligence (AAAI)*, 2010.