State of the Art Control of Atari Games Using Shallow Reinforcement Learning

Yitao Liang[†], Marlos C. Machado[‡], Erik Talvitie[†], and Michael Bowling[‡]
[†]Franklin & Marshall College
Lancaster, PA, USA
{yliang, erik.talvitie}@fandm.edu
[‡]University of Alberta
Edmonton, AB, Canada
{machado, mbowling}@ualberta.ca

ABSTRACT

The recently introduced Deep Q-Networks (DQN) algorithm has gained attention as one of the first successful combinations of deep neural networks and reinforcement learning. Its promise was demonstrated in the Arcade Learning Environment (ALE), a challenging framework composed of dozens of Atari 2600 games used to evaluate general competency in AI. It achieved dramatically better results than earlier approaches, showing that its ability to learn good representations is quite robust and general. This paper attempts to understand the principles that underlie DQN's impressive performance and to better contextualize its success. We systematically evaluate the importance of key representational biases encoded by DQN's network by proposing simple linear representations that make use of these concepts. Incorporating these characteristics, we obtain a computationally practical feature set that achieves competitive performance to DQN in the ALE. Besides offering insight into the strengths and weaknesses of DQN, we provide a generic representation for the ALE, significantly reducing the burden of learning a representation for each game. Moreover, we also provide a simple, reproducible benchmark for the sake of comparison to future work in the ALE.

Keywords

Reinforcement Learning, Function Approximation, DQN, Representation Learning, Arcade Learning Environment

1. INTRODUCTION

In the reinforcement learning (RL) problem an agent autonomously learns a behavior policy from experience in order to maximize a provided reward signal. Most successful RL approaches have relied upon the engineering of problem-specific state representations, diminishing the agent as fully autonomous and reducing its flexibility. The recent Deep Q-Network (DQN) algorithm [20] aims to tackle this problem, presenting one of the first successful combinations of RL and deep convolutional neural-networks (CNN) [13, 14], which are proving to be a powerful approach to representation learning in many areas. DQN is based upon the well-known Q-learning algorithm [31] and uses a CNN to simultaneously learn a problem-specific representation and estimate a value function.

Games have always been an important testbed for AI, frequently being used to demonstrate major contributions to the field [5, 6, 8, 23, 28]. DQN follows this tradition, demonstrating its success by achieving human-level performance in the majority of games within the Arcade Learning Environment (ALE) [2]. The ALE is a platform composed of dozens of qualitatively diverse Atari 2600 games. As pictured in Figure 1, the games in this suite include first-person perspective shooting games (e.g. Battle Zone), platforming puzzle games (e.g. Montezuma's Revenge), sports games (e.g. Ice Hockey), and many other genres. Because of this diversity, successful approaches in the ALE necessarily exhibit a degree of robustness and generality. Further, because it is based on problems designed by humans for humans, the ALE inherently encodes some of the biases that allow humans to successfully navigate the world. This makes it a potential stepping-stone to other, more complex decision-making problems, especially those with visual input.

DQN's success in the ALE rightfully attracted a great deal of attention. For the first time an artificial agent achieved performance comparable to a human player in this challenging domain, achieving by far the best results at the time and demonstrating generality across many diverse problems. It also demonstrated a successful large scale integration of deep neural networks and RL, surely an important step toward more flexible agents. That said, problematic aspects of DQN's evaluation make it difficult to fully interpret the results. As will be discussed in more detail in Section 6, the DQN experiments exploit non-standard game-specific prior information and also report only one independent trial per game, making it difficult to reproduce these results or to make principled comparisons to other methods. Independent re-evaluations have already reported different results due to the variance in the single trial performance of DQN (e.g. [12]). Furthermore, the comparisons that Mnih et al. did present were to benchmark results using far less training data. Without an adequate baseline for what is achievable using simpler techniques, it is difficult to evaluate the cost-benefit ratios of more complex methods like DQN.

In addition to these methodological concerns, the evaluation of a complicated method such as DQN often leaves open the question of which of its properties were most important to its success. While it is tempting to assume that the neural network must be discovering insightful tailored representations for each game, there is also a considerable amount of domain knowledge embedded in the very struc-

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*
Copyright © 2016, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

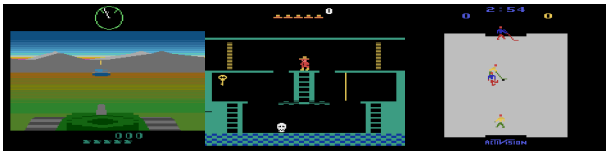

Figure 1: Examples from the ALE (left to right: Battle Zone, Montezuma's Revenge, Ice Hockey).

ture of the network. We can identify three key structural biases in DQN's representation. First, CNNs provide a form of spatial invariance not exploited in earlier work using linear function approximation. DQN also made use of multiple frames of input allowing for the representation of short-range non-Markovian value functions. Finally, small-sized convolutions are quite well suited to detecting small patterns of pixels that commonly represent objects in early video game graphics. These biases were not fully explored in earlier work in the ALE, so a natural question is whether non-linear deep network representations are key to strong performance in ALE or whether the general principles implicit in DQN's network architecture might be captured more simply.

The primary goal of this paper is to systematically investigate the sources of DQN's success in the ALE. Obtaining a deeper understanding of the core principles influencing DQN's performance and placing its impressive results in perspective should aid practitioners aiming to apply or extend it (whether in the ALE or in other domains). It should also reveal representational issues that are key to success in the ALE itself, potentially inspiring new directions of research.

We perform our investigation by elaborating upon a simple linear representation that was one of DQN's main comparison points, progressively incorporating the representational biases identified above and evaluating the impact of each one. This process ultimately yields a fixed, generic feature representation able to obtain performance competitive to DQN in the ALE, suggesting that the general form of the representations learned by DQN may in many cases be more important than the specific features it learns. Further, this feature set offers a simple and computationally practical alternative to DQN as a platform for future research, especially when the focus is not on representation learning. Finally, we are able provide an alternative benchmark that is more methodologically sound, easing reproducibility and comparison to future work.

2. BACKGROUND

In this section we introduce the reinforcement learning problem setting and describe existing results in the ALE.

2.1 Reinforcement Learning

In the *reinforcement learning* (RL) problem [26, 27] an agent interacts with an unknown environment and attempts to maximize a "reward" signal. The environment is commonly formalized as a *Markov decision process* (MDP) $\mathcal{M}$ defined as a 5-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma \rangle$. At time t the agent is in the state $s_t \in \mathcal{S}$ where it takes an action $a_t \in \mathcal{A}$ that leads to the next state $s_{t+1} \in \mathcal{S}$ according to the transition probability kernel P, which encodes $Pr(s_{t+1}|s_t, a_t)$. The agent also observes a reward $r_{t+1} \sim \mathcal{R}(s_t, a_t, s_{t+1})$. The agent's goal is to learn the *optimal policy*, a conditional distribution $\pi(a|s)$ that maximizes the *state value function* $V^\pi(s) \doteq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right]$ for all $s \in \mathcal{S}$, where $\gamma \in [0, 1)$ is known as the *discount factor*.

As a critical step toward improving a given policy π, it is common for reinforcement learning algorithms to learn a *state-action value function*, denoted:

$$Q^\pi(s, a) \doteq \mathbb{E}_\pi\left[\mathcal{R}(s_t, a_t, s_{t+1}) \; + \; \gamma V^\pi(s_{t+1}) \; \middle| s_t = s, a_t = a\right].$$

However, in large problems it may be infeasible to learn a value for each state-action pair. To tackle this issue agents often learn an approximate value function: $Q^\pi(s, a; \theta) \approx Q^\pi(s, a)$. A common approach uses linear function approximation (LFA) where $Q^\pi(s, a; \theta) = \theta^\top \phi(s, a)$, in which θ denotes the vector of weights and $\phi(s, a)$ denotes a static feature representation of the state s when taking action a. However, this can also be done through non-linear function approximation methods, including neural networks.

One of the most popular reinforcement learning algorithms is Sarsa(λ) [22]. It consists of learning an approximate action-value function while following a continually improving policy π. As the states are visited, and rewards are observed, the action-value function is updated and consequently the policy is improved since each update improves the estimative of the agent's expected return from state s, taking action a, and then following policy π afterwards, *i.e.* $Q^\pi(s, a)$. The Sarsa(λ) update equations, when using function approximation, are:

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}; \vec{\theta}_t) - Q(s_t, a_t; \vec{\theta}_t) \\ \vec{e}_t &= \gamma\lambda \vec{e}_{t-1} + \nabla Q(s_t, a_t; \vec{\theta}_t) \\ \vec{\theta}_{t+1} &= \vec{\theta}_t + \alpha \delta_t \vec{e}_t, \end{aligned}$$

where α denotes the step-size, the elements of $\vec{e}_t$ are known as the *eligibility traces*, and δ_t is the *temporal difference error*. Theoretical results suggest that, as an on-policy method, Sarsa(λ) may be more stable in the linear function approximation case than off-policy methods such as Q-learning, which are known to risk divergence [1, 10, 18]. These theoretical insights are confirmed in practice; Sarsa(λ) seems to be far less likely to diverge in the ALE than Q-learning and other off-policy methods [7].

2.2 Arcade Learning Environment

In the Arcade Learning Environment agents have access only to sensory information (160 pixels wide by 210 pixels high images) and aim to maximize the score of the game being played using the 18 actions on a standard joystick without game-specific prior information. Note that in most games a single screen does not constitute Markovian state. That said, Atari 2600 games are deterministic, so the entire history of interaction fully determines the next screen. It is most common for ALE agents to base their decisions on only the most recent few screens.

2.2.1 Linear Value Function Approximation

Early work in the ALE focused on developing new generic feature representations to be used with linear RL methods. Along with introducing the ALE itself, Bellemare et al. [2] presented an RL benchmark using four different feature sets obtained from the game screen. *Basic* features tile the screen and check if each of the available colors in the Atari 2600 are active in each tile. *BASS* features add pairwise combinations of Basic features. *DISCO* features attempt to detect

and classify objects on the screen in order to infer their positions and velocities. *LSH* simply applies Locally Sensitive Hashing [9] to raw Atari 2600 screens. Bellemare, Veness, and Bowling proposed an extension to the Basic feature set which involved identifying which parts of the screen were under the agents' direct control. This additional information is known as *contingency awareness* [3].

These feature sets were, for some time, the standard representations for Atari 2600 games, being directly used in other work [7, 16, 17] or serving as the base for more elaborate feature sets [4]. The feature representations presented in this paper follow the spirit of this early work. We use Basic features (formally described below) as a starting point, and attempt to capture pairwise spatial relationships between objects on the screen.

Basic Features: To obtain Basic features we first divide the Atari 2600 screen into 16×14 tiles of size 10×15 pixels. For every tile (c, r) and color k, where $c \in \{1, \ldots, 16\}$, $r \in \{1, \ldots, 14\}$, and $k \in \{1, \ldots, 128\}$, we check whether color k is present within the tile (c, r), generating the binary feature $\phi_{c,r,k}$. Intuitively, Basic features encode information like "there is a red pixel within this region". There are $16 \times 14 \times 128 = 28{,}672$ Basic features in total.

It can be computationally demanding to work directly with 160×210 pixel screens with a 128 color palette. To make the screen more sparse, following previous work (*e.g.* [2, 3]) we subtract the background from the screen at each frame before processing it. The background is precomputed using 18,000 samples obtained from random trajectories.

2.2.2 Non-Linear Value Function Approximation

The use of non-linear function approximation in the ALE is more recent. Veness et al. [30], for example, propose the use of compression techniques as a policy evaluation approach in RL, evaluating their method in the ALE. Soon after, the research landscape changed due to the success of using deep learning techniques to approximate value functions, demonstrated by the DQN algorithm [19, 20]. DQN achieved 75% of a human player's score in the majority of games and this inspired much more work in the ALE using deep learning (*e.g.* [11, 24, 25]).

DQN employs a deep convolutional neural network to represent the state-action value function Q. A deep convolutional network uses weight sharing to make it practical to learn a function of high-dimensional image input. Specifically, when processing the image a small network focused on a small region of the screen called a *filter* is applied at multiple positions on the screen. Its output at each position forms a new (smaller) image, which can then be processed by another filter, and so on. Multiple layers of convolution allow the network to detect patterns and relationships at progressively higher levels of abstraction. DQN's network uses three layers of convolution with multiple filters at each layer. The final result of those convolutions is then processed via a more standard fully connected feed-forward network. The convolutional aspect of the network allows the network to detect relationships like "A bullet is near an alien here," where "bullet" and "alien" can be position invariant concepts. The non-linear global layers allow it to represent whole-screen concepts such as "A bullet is near an alien somewhere." This is the inspiration of our spatially invariant features (Section 3). Further, Mnih et al. actually provide the four most recent images as input to their network, allowing it to detect relationships through time (such as movement). This motivates our short-order-Markov features (Section 4). Finally, note that Basic features are essentially a simple convolution where the filters simply detect the presence of each color in their range. DQN's more sophisticated filters motivate our improvement of object detection for base features (Section 5).

3. SPATIAL INVARIANCE

Recall that Basic features detect the presence of each color in various regions of the screen. As discussed above, this can be seen as analogous to a single convolution with a crude filter. The BASS feature set, which was amongst the best performing representations before DQN's introduction encodes pairwise relationships between Basic features, still anchored at specific positions, so it is somewhat analogous to a second convolutional layer. But in many games the absolute positions of objects are not as important as their relative positions to each other. To investigate the effect of ignoring absolute position we impose a non-linearity over BASS (analogous to DQN's fully connected layers), specifically taking the max of BASS features over absolute position.

We call the resulting feature set Basic Pairwise Relative Offsets in Space (B-PROS) because it captures pairwise *relative* distances between objects in a single screen. More specifically, a B-PROS feature checks if there is a pair of Basic features with colors $k_1, k_2 \in \{1, \ldots, 128\}$ separated by an offset (i, j), where $-13 \leq i \leq 13$ and $-15 \leq j \leq 15$. If so, $\phi_{k_1,k_2,(i,j)}$ is set to 1, meaning that a pixel of color k_1 is contained within some block (c, r) and a pixel of color k_2 is contained within the block $(c + i, r + j)$. Intuitively, B-PROS features encode information like "there is a yellow pixel three tiles below a red pixel". The computational complexity of generating B-PROS features is similar to that of BASS, though ultimately fewer features are generated. Note that, as described, B-PROS contains redundant features (e.g. $\phi_{1,2,(4,0)}$ and $\phi_{2,1,(-4,0)}$), but it is straightforward to eliminate them. The complete feature set is composed of the Basic features and the B-PROS features. After redundancy is eliminated, the B-PROS feature set has 6,885,440 features in total $(28{,}672 + ((31 \times 27 \times 128^2 - 128)/2 + 128))$.

Note that there have been other attempts to represent pairwise spatial relationships between objects, for instance DISCO [2] and contingency awareness features [3]. However, these existing attempts are complicated to implement, demanding to compute, and less effective (as will be seen), most likely due to unreliable estimates of object positions.

3.1 Empirical Evaluation

Our first set of experiments compares B-PROS to the previous state of the art in linear representations for the Atari 2600 [2, 3] in order to evaluate the impact of the non-linearity applied to BASS. For the sake of comparison we follow Bellemare et al.'s methodology [2]. Specifically, in 24 independent trials the agent was trained for 5000 episodes. After the learning phase we froze the weights and evaluated the learned policy by recording its average performance over 499 episodes. We report the average evaluation score across the 24 trials. Following Bellemare et al., we defined a maximum length for each episode: 18,000 frames, *i.e.* five minutes of real-time play. Also, we used a frame-skipping technique, in which the agent selects actions and updates its

Table 1: Comparison of linear representations. Bold denotes the largest value between B-PROS and Best Linear [2]. See text for more details.

Game	Best Linear	CAF	B-PROS (std. dev.)	
Asterix	987.3	1332.0	**4078.7**	(1016.0)
Beam Rider	929.4	1742.7	**1528.7**	(504.4)
Breakout	5.2	6.1	**13.5**	(11.0)
Enduro	129.1	159.4	**240.5**	(24.2)
Freeway	16.4	19.97	**31.0**	(0.7)
Pong	-19.2	-17.4	**4.9**	(6.6)
Q*Bert	613.5	960.3	**1099.8**	(740.0)
Seaquest	664.8	722.89	**1048.5**	(321.2)
Space Invaders	250.1	267.93	**384.7**	(87.2)

value function every x frames, repeating the selected action for the next $x-1$ frames. This allows the agent to play approximately x times faster. Following Bellemare et al. we use $x = 5$ (DQN also uses a similar procedure).

We used Sarsa(λ) with replacing traces and an ϵ-greedy policy. We performed a parameter sweep over nine games, which we call "training" games. The reported results use a decay rate $\gamma = 0.99$, an exploration rate $\epsilon = 0.01$, a step-size $\alpha = 0.5$ and eligibility decay rate $\lambda = 0.9$.

Table 1 compares our agent to existing linear agents in our set of training games (note that Breakout, Enduro, Pong, and Q*Bert were not training games for the earlier methods). "Best Linear" denotes the best performance obtained among four different feature sets: Basic, BASS, DISCO and LSH [2]. For reference, in the "CAF" column we also include results obtained by contingency awareness features [3], as reported by Mnih et al. [20]. Note, that these results are not directly comparable because their agent was given 10,000 episodes of training (rather than 5000).

B-PROS' performance surpasses the original benchmarks by a large margin and, except in one game, even surpasses the performance of CAF, despite being far simpler and training with half as much data. Further, some of the improvements represent qualitative breakthroughs. For instance, in Pong, B-PROS allows the agent to win the game with a score of 20-15 on average, while previous methods rarely scored more than a few points. In Enduro, the earlier methods seem to be stymied by a sudden change in dynamics after approximately 120 points, while B-PROS is consistently able to continue beyond that point.

When comparing these features sets for all 53 games B-PROS performs better on average than all of Basic, BASS, DISCO, and LSH in 77% (41/53) of the games. Even with less training data, B-PROS performs better on average than CAF in 77% (38/49) of the games in which CAF results were reported[1]. The dramatic improvement yielded by B-PROS clearly indicates that focusing on relative spatial relationships rather than absolute positions is vital to success in the ALE (and likely in other visual domains as well).

4. NON-MARKOVIAN FEATURES

B-PROS is capable of encoding relative distances between objects but it fails to encode movement, which can be a very important aspect of Atari games. For instance, the agent may need to know whether the ball is moving toward or away from the paddle in Pong. Previous linear representations similarly relied mainly on the most recent screen. In contrast, Mnih et al. use the four most recent screens as input, allowing DQN to represent short-order-Markov features of the game screens. In this section we present an extension to B-PROS that takes a similar approach, extracting information from the two most recent screens.

Basic Pairwise Relative Offsets in Time (B-PROT) features represent pairwise relative offsets between Basic features obtained from the screen five frames in the past and Basic features from the current screen (*i.e.* PROT features). More specifically, for every pair of colors $k_1, k_2 \in \{1, \ldots, 128\}$ and every offset (i, j), where $-13 \leq i \leq 13$ and $-15 \leq j \leq 15$, a binary B-PROT feature $\phi_{k_1,k_2,(i,j)}$ is 1 if a pixel of color k_1 is contained within some block $(c+i, r+j)$ on the screen five frames ago and a pixel of color k_2 is contained within the block (c, r) in the current screen.

The B-PROST feature set contains Basic, B-PROS, and B-PROT features. Note that there are roughly twice as many B-PROT features as B-PROS because there are no redundant offsets. As such, B-PROST has a total of 20,598,848 sparse, binary features ($6{,}885{,}440 + 31 \times 27 \times 128^2$).

4.1 Empirical Evaluation

B-PROS outperformed all the other linear architectures in the previous experiment. Subsequent extensions will be primarily compared to DQN (see Section 6). As such, in these experiments, we adopted an evaluation protocol similar to Mnih et al.'s. Each agent was trained for 200,000,000 frames (equivalent to 40,000,000 decisions) over 24 independent trials. The learned policy in each trial was evaluated by recording its average performance in 499 episodes with no learning. We report the average evaluation score over the 24 trials. In an effort to make our results comparable to DQN's we also started each episode with a random number of "no-op" actions and restricted the agent to the minimal set of actions that have a unique effect in each game.

The first two columns of Table 2 present results using B-PROS and B-PROST in the training games. B-PROST outperforms B-PROS in all but one of the training games. One particularly dramatic improvement occurs in Pong; B-PROS wins with a score of 20-9, on average, while B-PROST rarely allows the opponent to score at all. Another result worth noting is in Enduro. The randomized initial conditions seem to have significantly harmed the performance of B-PROS in this game, but B-PROST seems to be robust to this effect. When evaluated over all 49 games evaluated by Mnih et al. the average score using B-PROST is higher than that using B-PROS in 82% of the games (40/49)[2]. This clearly indicates the critical importance of non-Markov features to success in the ALE.

Before making a final comparison with DQN, we will make one more improvement to our representation.

5. OBJECT DETECTION

Because Basic features encode the positions of individual pixels on the screen, both B-PROS and B-PROST features struggle to distinguish which pixels are part of the same object. DQN's network is capable of learning far more subtle filters. In order to measure the impact of improved low-level object detection, we consider a simple extension to Basic that exploits the fact that Atari screens often contain several contiguous regions of pixels of the same color. We call such

[1]Due to space restrictions we reserve the full table of 53 games for the longer version of this paper [15].

[2]The full results table is available in the longer paper [15].

Table 2: Comparison of relative offset features. Bold indicates the best average of the three columns. The † indicates significant differences between B-PROST and Blob-PROST.

Game	B-PROS Avg. (std. dev.)	B-PROST Avg. (std. dev.)	Blob-PROST Avg. (std. dev.)
Asterix	8194.3 (1802.9)	**8928.8**† (1714.5)	4400.8 (1201.3)
Beam Rider	1686.9 (255.2)	1808.9 (309.2)	**1902.3** (423.9)
Breakout	7.1 (1.7)	15.0 (4.7)	**46.7**† (45.7)
Enduro	34.9 (71.5)	207.7 (23.1)	**257.0**† (79.8)
Freeway	23.8 (6.7)	29.1 (6.3)	**31.5**† (1.2)
Pong	10.9 (5.2)	18.9 (1.3)	**20.1**† (0.5)
Q*Bert	3647.8 (1273.3)	3608.7 (1129.0)	**6946.8**† (3036.4)
Seaquest	1366.1 (445.9)	1636.5 (519.5)	**1664.2** (440.4)
Space Invaders	505.1 (130.4)	582.9 (139.0)	**723.1**† (101.5)

regions "blobs". Rather than directly represent coarsened positions of pixels, we first process the screen to find a list of blobs. Blob features then represent coarsened positions of blobs on the screen. Changing the "primitive" features from Basic to Blob yields the Blob-PROST feature set.

Note that blobs are a simplification; in many Atari games, as would be true in more natural images, objects consist of multiple close but separate blobs. Grouping only strictly contiguous pixels into each blob may generate redundant blobs that all represent a single object. As a simple means to address this we add a tolerance to the contiguity condition, *i.e.* we consider pixels that are in the same $s \times s$ pixel square to be contiguous. This approach has an inherent trade-off. On the one hand, with sufficiently large s we may successfully represent each object with few blobs. Also, by reducing the number of blobs, we substantially decrease the number of primitive features, making Blob-PROS and Blob-PROT features easier to compute. On the other hand, if s is too high then multiple distinct objects may be grouped together. In our experiments we set s to be 6 after an informal search using the set of training games. It is very likely that with a more systematic selection of s one can obtain better results than those reported here.

We define the position of a blob as the centroid of the blob's smallest bounding box. To generate the features, we first generate Blob features that are analogous to Basic features: we divide the screen into tiles of 4×7 pixels and then for every color k and block (c, r), where $c \in \{1, \ldots, 40\}, r \in \{1, \ldots, 30\}$ and $k \in \{1, \ldots, 128\}$, the Blob feature $\phi_{c,r,k}$ is 1 if the block (c, r) contains the centroid for some blob of color k. Note that we use a finer resolution than Basic. This is feasible only because of the extreme sparsity of blobs on the screen. The number of blobs in one frame ranged from 6 (Pong) to 412 (Battle Zone). This sparsity also makes the background subtraction step unnecessary; the background typically reduces to a handful of blobs itself.

The Blob-PROST feature set is then constructed from these primitive features in the same manner as B-PROST. There are 153,600 Blob features ($40 \times 30 \times 128$), 38,182,976 Blob-PROS features ($(79 \times 59 \times 128^2 - 128)/2 + 128$), and 76,365,824 Blob-PROT features ($79 \times 59 \times 128^2$). This yields a total of 114,702,400 Blob-PROST features. Though the number of possible features is very large, most would never be generated due to the sparsity of blobs.

5.1 Empirical Evaluation

The last column of Table 2 presents results using Blob-PROST in the training games. In all but one of the training games, Blob-PROST outperformed both B-PROS and B-PROST on average[3]. In 6 of the 9, Blob-PROST's average performance was statistically significantly better than B-PROST (using Welch's t-test with $p < 0.05$). Asterix is a game in which Blob-PROST notably fails; perhaps blob detection lumps together distinct objects in a harmful way. Q*Bert is a notable success; the scores of B-PROS and B-PROST indicate that they rarely complete the first level, while Blob-PROST's score indicates that it consistently clears the first level. Out of the 49 games evaluated by Mnih et al., Blob-PROST's average performance was higher than that of both B-PROS and B-PROST in 59% (29/49) of the games. Its performance was statistically significantly higher than that of B-PROST in 47% (23/49) of the games, showing the clear contribution of even simple improvements in object detection. Though object detection itself is not a feature, it make all features based upon it carry more meaningful information that helps our agents better "interpret" the screen.

6. COMPARISON WITH DQN

The next set of experiments compare Blob-PROST to DQN, showing that the three enhancements investigated above largely explain the performance gap between DQN and earlier LFA approaches. In performing this comparison, we necessarily confront problematic aspects of the original DQN evaluation (some of which we adopt for comparison's sake). In Section 7 we present a more methodologically sound ALE benchmark for comparison to future work.

6.1 DQN Evaluation Methodology

In each game, Mnih et al. trained a DQN agent once for 200,000,000 frames. Rather than evaluating the learned policy after the whole learning phase completed (as in earlier work), they evaluated their agent's performance every 4,000,000 frames and selected the best performing weights. They reported the average and standard deviation of the scores of this best learned policy in 30 evaluation episodes.

Mnih et al. also restricted their agent to the minimal set of actions that have a unique effect in each game, game-specific prior information not exploited by earlier work in the ALE. Since in most games the minimal action set contains fewer actions than the full action set, agents with access to the minimal set may benefit from a faster effective learning rate (this is discussed further in Section 7).

To avoid overfitting to the Atari's determinism, Mnih et al. randomized the initial state of each episode by taking a random number of "no-op" actions. The number of "no-ops" was uniformly randomly selected from $\{1, \ldots, 30\}$.

Mnih et al. also modified when episodes ended. In many Atari games the player has a number of "lives" and the game ends when they are exhausted. DQN employed the same episode termination criteria as Bellemare et al. [2], *i.e.* the end of the game or expiration of the 5 minute time limit, but during training also terminated episodes when the agent lost a life (another form of game-specific prior information).We have not specifically evaluated the impact of this mechanism, but one might speculate that in games that have "lives" (e.g. Breakout, Space Invaders) this modification could more easily generate agents with an "avoid death" policy.

[3]The implementation of the three feature sets we introduced, as well as the code used in our experiments is available at `https://github.com/mcmachado/b-pro`.

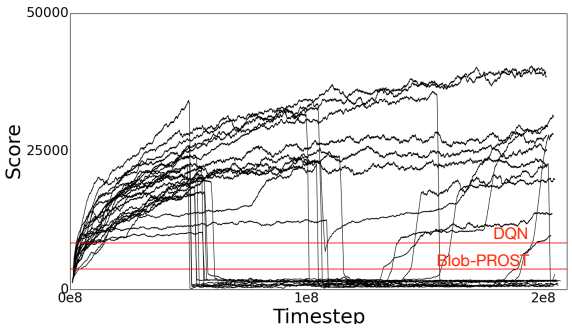

Figure 2: Learning curves for Blob-PROST on the game Up 'n Down, smoothed over 500 episodes.

The DQN experimental methodology possesses three main problematic flaws. First, one cannot make statistical comparisons from a single trial. While Mnih et al. report a standard deviation, this only represents the amount of variation observed when executing the one chosen policy, and not variation observed over different independent executions of the inherently stochastic algorithm. DQN's consistently high performance across games suggests it does indeed regularly perform well, but the reported performance numbers on any particular game may not be representative of their expected performance. Second, selecting the best performing parameterization over the whole learning period is a significant deviation from typical RL methodology. Such a choice may be masking instability issues where learning performance is initially strong but later goes awry. This effect can occur even using comparatively stable linear function approximators. For example, Figure 2 shows 24 independent learning trials from the game Up 'n Down using our linear Blob-PROST agent. Many learning trials in this game exhibit an agent steadily improving its performance before a sudden plummet from which it does not recover. The reported performance of DQN and Blob-PROST is also depicted, showing the impact of such instability: most trials perform much better than the reported average at some point. We contend that general competency should include consistent and stable improvement over time, best measured by an agent's performance at the end of training. Third, using designer-provided game-specific prior knowledge in the form of the minimal action set and termination on "death" is counter to the goal of evaluating general competency.

Some of these concerns have been addressed in emerging follow-up work to DQN [21], which uses the final network weights for evaluation and reports the average score over five independent trials.

6.2 Comparing Blob-PROST and DQN

As discussed in Section 4, for comparison's sake, we adopted a similar methodology. We did use the minimal action set, and did add random "no-ops" to the beginning of episodes, but we opted not to utilize the life counter.

6.2.1 Computational Cost

We found that Blob-PROST is far more computationally practical than DQN. We compared the algorithms' resource needs on 3.2GHz Intel Core i7-4790S CPUs. We ran them for 24 hours (to allow resource usage to stabilize), then measured their runtime and memory use.

The computational cost of our implementation of Blob-PROST can vary from game to game. The runtime of our implementation ranged from 56 decisions per second (Alien) to 300 decisions per second (Star Gunner), that is 280-1500 frames per second (many times real time speed). The memory utilization of our implementation ranged from 50MB (Pong) to 9GB (Battle Zone). Note that Battle Zone was an outlier; the next most memory intensive game (Star Gunner) used only 3.7GB of memory. Furthermore, the memory utilization of Blob-PROST can likely be effectively controlled by simplifying the color palette or through the use of feature hashing (*e.g.* [4]).

In contrast, we found DQN's computational cost to be quite consistent across games, running at a speed of approximately 5 decisions per second (*i.e.* 20 frames per second, 3 times slower than real time), and requiring approximately 9.8GB of memory. DQN already uses a reduced color palette and is not immediately amenable to feature hashing to control memory usage. On the other hand, it is amenable to GPU acceleration to improve runtime. Mnih et al. do not report DQN's runtime but in recent follow-up work the GPU accelerated speed has been reported to be approximately 330 frames per second [29], still slower than the Blob-PROST agent in most games. Further note that obtaining enough GPUs to support multiple independent trials in all the games (necessary for statistical comparisons) is, in itself, a prohibitive cost. Emerging work [21] shows that a related approach can be accelerated via CPU parallelism.

6.2.2 ALE Performance

Because only one trial of DQN was reported in each game, and because of the prohibitively high computational cost of independently re-evaluating DQN, a principled comparison between our method and DQN is essentially impossible. Instead, we used a number of ad hoc measures aimed at forming an intuitive understanding of the relative capabilities of the two algorithms, based on available evidence.

First, for each game we record how many trials of Blob-PROST obtained evaluation scores greater than DQN's single trial. If we were to compare an algorithm to itself in this way, we would expect roughly 50% of the trials to have greater performance. In Table 3 (available in the Appendix), the column marked "% trials > DQN" reports the results. The average percentage of trials better than DQN, across all games, was 41%. That is, if you select a game uniformly randomly and run Blob-PROST, there is an estimated 41% chance that it will surpass DQN's reported score.

Second, in each game we compared Blob-PROST's middle trial (12th best) to the single reported DQN trial. If Blob-PROST's evaluation score compares favorably to DQN in this trial then this suggests that scores like DQN's are typical for Blob-PROST. Again, if this method were used to compare an algorithm to itself, one would expect the middle trial to be better in roughly half of the games. In Table 3 the column marked "Middle trial" reports the results. Blob-PROST's middle trial evaluation score was better than DQN's in 43% (21/49) of the games. In 3 of the games in which it performed worse, Blob-PROST's score was not stastically significantly different than DQN's. So in 49% of the games Blob-PROST's middle trial was either not different from or better than DQN's single reported trial.

Third, in each game we compared Blob-PROST's best trial to the single DQN trial. Since it is possible that some of DQN's trials are high-performing outliers, this is intended to give a sense of the level of performance Blob-PROST is capable of reaching, regardless of whether it can do so consistently. In 65% (32/49) of the games, Blob-PROST's best trial's performance exceeded that of the DQN trial. Again, in three games in which Blob-PROST did worse, the difference was not statistically significant. So in 71% of the games Blob-PROST's best trial was either not different from or better than DQN's single reported trial.

Finally, DQN's main result was achieving at least 75% of a human's performance in over half of the games (29/49). The best trial of Blob-PROST crossed this threshold in 34/49 games. The middle trial did so in only 20/49 games.

The Blob-PROST representation was generated using simple and natural enhancements to BASS (one of DQN's main comparison points) aimed at incorporating similar structural biases to those encoded by DQN's designer-provided network architecture. All told, these results seem to indicate that this fixed representation is of comparable quality to DQN's learned representation across many games.

7. ALE BENCHMARK

As discussed above, besides better understanding the core issues underlying DQN's success in the ALE, it is important to present a reproducible benchmark without the problematic flaws discussed in Section 6 (*i.e.* one that reports more than one trial, evaluates the final set of weights, and eschews game-specific prior knowledge). Moreover, establishing this benchmark is important for the ALE's continued use as an evaluation platform. The principled evaluation of complicated approaches depends upon the existence of a sound baseline for what can be achieved with simple approaches.

To this end we also present the performance of Blob-PROST using the full action set. Our agents' average performance after 24 trials using the full action set is reported in the first column of Table 3 in the Appendix. We recommend that future comparisons to Blob-PROST use these results. Surprisingly, we found that when using the full action set Blob-PROST performed slightly better in comparison to DQN using the measures described above than when using the minimal action set. It is not entirely clear why this would be; one possible explanation is that the presence of redundant actions may have a positive impact on exploration in some games.

8. CONCLUSIONS AND FUTURE WORK

While it is difficult to draw firm conclusions, the results indicate the Blob-PROST's performance in the ALE is competitive to DQN's, albeit likely slightly worse overall. Most importantly, these results indicate we may have been able to capture some of the key features of DQN in a practical, fixed linear representation. We saw progressive and dramatic improvements by respectively incorporating relative distances between objects, non-Markov features, and more sophisticated object detection. This illuminates some important representational issues that likely underly DQN's success. It also suggests that the general properties of the representations learned by DQN may be more important to its success in ALE than the specific features it learns in each game. This deeper understanding of the specific strengths of DQN should aid practitioners in determining when its benefits are likely to outweigh its costs.

It is important to note that we do not intend to diminish the importance of DQN as a seemingly stable combination of deep neural networks and RL. This is a promising and exciting research direction with potential applications in many domains. That said, taking into account the fact that DQN's performance may be inflated to an unknown degree as a result of evaluating the best performing set of weights, the fact of DQN's extreme computational cost, and the methodological issues underlying the reported DQN results, we conclude that Blob-PROST is a strong alternative candidate to DQN both as an ALE performance benchmark and as a platform on which to build future ALE agents when representation learning is not the topic being studied. Our results also suggest in general that fixed representations inspired by the principles underlying convolutional networks may yield competitive, lighter-weight alternatives.

As future work, it may be interesting to investigate the remaining discrepancies between our results and DQN's. In some games like Gravitar, Frostbite and Krull Blob-PROST's performance was several times higher than DQN's, while the opposite was true in games such as Star Gunner, Breakout and Atlantis. In general, DQN seems to excel in shooting games (a large part of the games supported by the ALE), maybe because it is able to easily encode object velocities and to predict objects' future positions. DQN also excels on games that require a more holistic view of the whole screen (e.g. Breakout, Space Invaders), something pairwise features struggle with. On the other hand, some of the games in which Blob-PROST surpasses DQN are games where it is fairly easy to die, maybe because DQN interrupts the learning process after the agent loses a life. Other games Blob-PROST succeeds in are games where the return is very sparse (e.g. Tennis, Montezuma's Revenge). These games may generate very small gradients for the network, making it harder for an algorithm to learn both the representation and a good policy. Alternatively DQN's process of sampling from the experience memory may not be effective to draw the "interesting" samples.

These conjectures suggest directions for future work. Adaptive representation methods may potentially benefit from these insights by building in stronger bias toward the types of features we have investigated here. This would allow them to quickly perform well, and then focus energy on learning exceptions to the rule. In the realm of linear methods, these results suggest that there may still be simple, generic enhancements that yield dramatic improvements. More sophisticated object detection may be beneficial, as might features that encode more holistic views of the screen.

Acknowledgements

This research was supported by Alberta Innovates Technology Futures and the Alberta Innovates Centre for Machine Learning. Computing resources were provided by Compute Canada through CalculQuébec.

APPENDIX

The following table reports the data used to compare Blob-PROST to DQN's reported results (see Section 6) as well as our final ALE benchmark (see Section 7). A longer version of this paper presents full tables for all experiments [15].

Table 3: The first column presents the average performance of Blob-PROST using the full action set for comparison to future work (see Section 7). The standard deviation reported represents the variability over the 24 independent learning trials. The rest of the columns present our comparison between Blob-PROST and DQN (see Section 6 for details). In these results the Blob-PROST agent uses the minimal action set, for the sake of comparison. The standard deviation reported for the best trial as well as for the median trial is the standard deviation while evaluating for 499 episodes. We use bold face to denote values that are higher than DQN's and † to denote Blob-PROST averages that are statistically significantly different than DQN's average. To do the comparison for each game we used Welch's t-test (p<0.025, which accounts for the usage of DQN results in two tests, for an overall significance level of 0.05).

Game	Avg. (std. dev.) – Full		Best trial (std. dev.)		Middle trial (std. dev.)		% trials > DQN	DQN		Human	Random
Asterix	3996.6	(743.9)	**6921.5**	(4501.2)	4472.9†	(2514.8)	8.3	6012.0	(1744.0)	8503.0	210.0
Beam Rider	2367.3	(815.0)	2965.5†	(1376.1)	1807.6†	(657.0)	0.0	6846.0	(1619.0)	5775.0	363.9
Breakout	52.9	(38.0)	190.3†	(179.7)	33.2†	(21.3)	0.0	401.2	(26.9)	31.8	1.7
Enduro	296.7	(7.8)	299.1	(26.9)	279.2†	(28.2)	0.0	301.8	(24.6)	309.6	0.0
Freeway	32.3	(0.5)	**32.6**†	(1.0)	**31.7**†	(1.0)	95.8	30.3	(0.7)	29.6	0.0
Pong	20.2	(0.4)	**20.5**†	(0.8)	**20.2**†	(1.4)	95.8	18.9	(1.3)	9.3	-20.7
Q*Bert	8072.4	(2210.5)	**14449.4**†	(4111.7)	5881.4†	(1069.0)	12.5	10596.0	(3294.0)	13455.0	163.9
Seaquest	1664.2	(440.4)	2278.0†	(294.1)	1845.1†	(469.8)	0.0	5286.0	(1310.0)	20182.0	68.4
Space Invaders	844.8	(144.9)	889.8†	(374.5)	712.9†	(250.3)	0.0	1976.0	(893.0)	1652.0	148.0
Alien	4154.8	(532.5)	**4886.6**†	(1151.9)	**4219.1**†	(635.4)	95.8	3069.0	(1093.0)	6875.0	227.8
Amidar	408.4	(177.5)	**825.4**	(225.3)	522.3	(153.1)	12.5	739.5	(3024.0)	1676.0	5.8
Assault	1107.9	(207.7)	1829.3†	(702.9)	1380.8†	(449.9)	0.0	3359.0	(775.0)	1496.0	224.4
Asteroids	1759.5	(182.1)	**2229.9**†	(937.0)	**1635.1**	(617.9)	50.0	1629.0	(542.0)	13157.0	719.1
Atlantis	37428.5	(11599.7)	42937.7†	(13763.1)	19983.2†	(5830.4)	0.0	85641.0	(17600.0)	29028.0	12850.0
Bank Heist	463.4	(93.6)	**793.6**†	(102.9)	**455.1**	(100.9)	62.5	429.7	(650.0)	734.4	14.2
Battle Zone	26222.8	(4070.0)	**37850.0**†	(7445.0)	25836.0	(6616.3)	41.7	26300.0	(7725.0)	37800.0	2360.0
Bowling	65.9	(14.2)	**91.1**†	(13.6)	**62.0**	(2.5)	91.7	42.4	(88.0)	154.8	23.1
Boxing	89.4	(16.5)	**98.3**†	(3.3)	**95.8**†	(5.5)	87.5	71.8	(8.4)	4.3	0.1
Carnival	4322.0	(3705.0)	-	-	-	-	-	-	-	-	-
Centipede	3903.3	(6838.8)	**21137.0**†	(6722.8)	426.7†	(313.8)	20.8	8309.0	(5237.0)	11963.0	2091.0
Chopper Command	3006.6	(782.0)	4898.9†	(2511.2)	2998.0†	(1197.7)	0.0	6687.0	(2916.0)	9882.0	811.0
Crazy Climber	73241.5	(10467.9)	81016.0†	(26529.2)	59848.9†	(27960.7)	0.0	114103.0	(22797.0)	35411.0	10781.0
Demon Attack	1441.8	(184.8)	2166.0†	(1327.5)	1774.2†	(979.8)	0.0	9711.0	(2406.0)	3401.0	152.1
Double Dunk	-6.4	(0.9)	**-4.1**†	(2.3)	**-6.2**†	(2.8)	100.0	-18.1	(2.6)	-15.5	-18.6
Fishing Derby	-58.8	(12.0)	-28.7†	(19.0)	-57.4†	(18.3)	0.0	-0.8	(19.0)	5.5	-91.7
Frostbite	3389.7	(743.6)	**4534.0**†	(1496.8)	**3557.5**†	(864.3)	100.0	328.3	(250.5)	4335.0	65.2
Gopher	6823.4	(1022.5)	7451.1†	(3146.1)	5006.9†	(3118.9)	0.0	8520.0	(32.8)	2321.0	257.6
Gravitar	1231.8	(423.4)	**1709.7**†	(669.7)	**1390.6**†	(752.9)	91.7	306.7	(223.9)	2672.0	173.0
H.E.R.O.	13690.3	(4291.2)	**20273.1**†	(1155.1)	13642.1†	(48.4)	12.5	19950.0	(158.0)	25673.0	1027.0
Ice Hockey	14.5	(3.4)	**22.8**†	(6.5)	**14.5**†	(5.5)	100.0	-1.6	(2.5)	0.9	-11.2
James Bond	636.3	(192.1)	**1030.5**†	(947.1)	**587.8**	(99.5)	50.0	576.7	(175.5)	406.7	29.0
Kangaroo	3800.3	(2211.0)	**9492.8**†	(2918.2)	3839.1†	(1601.1)	8.3	6740.0	(2959.0)	3035.0	52.0
Krull	8333.9	(5599.5)	**33263.4**†	(15403.3)	**7463.9**†	(1719.7)	95.8	3805.0	(1033.0)	2395.0	1598.0
Kung-Fu Master	33868.5	(6247.5)	**51007.6**†	(9131.9)	**33232.1**†	(7904.5)	91.7	23270.0	(5955.0)	22736.0	258.5
Montezuma's Revenge	778.1	(789.8)	**2508.4**†	(27.8)	**400.0**†	(0.0)	100.0	0.0	(0.0)	4367.0	0.0
Ms. Pac-Man	4625.6	(774.3)	**5917.9**†	(1984.0)	**4667.4**†	(2126.0)	100.0	2311.0	(525.0)	15693.0	307.3
Name This Game	6580.1	(1773.0)	**7787.0**†	(744.7)	6387.1†	(1046.7)	20.8	7257.0	(547.0)	4076.0	2292.0
Pooyan	2228.1	(274.5)	-	-	-	-	-	-	-	-	-
Private Eye	33.0	(47.6)	100.3	(4.0)	0.0	(0.0)	0.0	1788.0	(5473.0)	69571.0	24.9
River Raid	10629.1	(2110.5)	**14583.3**†	(3078.3)	**9540.4**†	(1053.8)	91.7	8316.0	(1049.0)	13513.0	1339.0
Road Runner	24558.3	(12369.2)	**41828.0**†	(9408.1)	**28848.5**†	(6857.4)	75.0	18257.0	(4268.0)	7845.0	11.5
Robotank	28.3	(2.7)	34.4†	(7.7)	28.5†	(8.0)	0.0	51.6	(4.7)	11.9	2.2
Skiing	-29842.6	(19.8)	-	-	-	-	-	-	-	-	-
Star Gunner	1227.7	(165.1)	1651.6†	(435.4)	1180.0†	(85.2)	0.0	57997.0	(3152.0)	10250.0	664.0
Tennis	0.0	(0.0)	**0.0**†	(0.2)	**0.0**†	(0.2)	100.0	-2.5	(1.9)	-8.9	-23.8
Time Pilot	3972.0	(878.3)	5429.5	(1659.7)	4069.3†	(647.7)	0.0	5947.0	(1600.0)	5925.0	3568.0
Tutankham	81.4	(50.5)	**217.7**†	(33.9)	41.7†	(4.2)	8.3	186.7	(41.9)	167.7	11.4
Up and Down	19533.0	(18733.6)	**41257.8**†	(12061.8)	3716.7†	(1050.8)	45.8	8456.0	(3162.0)	9082.0	533.4
Venture	244.5	(490.4)	**1397.0**†	(294.4)	0.0†	(0.0)	20.8	380.0	(238.6)	1188.0	0.0
Video Pinball	9783.9	(3043.9)	21313.0†	(14916.4)	9480.7†	(6563.1)	0.0	42684.0	(16287.0)	17298.0	16257.0
Wizard of Wor	2733.6	(1259.0)	**5681.2**†	(4573.2)	**3442.9**	(2582.0)	50.0	3393.0	(2019.0)	4757.0	563.5
Zaxxon	8204.4	(2845.3)	**11721.8**†	(4009.7)	**9280.0**†	(2251.4)	87.5	4977.0	(1235.0)	9173.0	32.5
Times > DQN	N/A		32/49		21/49						
Times statist. ≥ DQN	N/A		35/49		24/49						

REFERENCES

[1] L. C. Baird III. Residual Algorithms: Reinforcement Learning with Function Approximation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 30–37, 1995.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.

[3] M. G. Bellemare, J. Veness, and M. Bowling. Investigating Contingency Awareness using Atari 2600 Games. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI)*, pages 864–871, 2012.

[4] M. G. Bellemare, J. Veness, and M. Bowling. Sketch-Based Linear Value Function Approximation. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pages 2222–2230, 2012.

[5] M. Bowling, N. Burch, M. Johanson, and O. Tammelin. Heads-up Limit Hold'em Poker is Solved. *Science*, 347(6218):145–149, January 2015.

[6] M. Campbell, A. J. H. Jr., and F.-H. Hsu. Deep Blue. *Artificial Intelligence*, 134(1âĂŞ2):57 – 83, 2002.

[7] A. Defazio and T. Graepel. A Comparison of Learning Algorithms on the Arcade Learning Environment. *CoRR*, abs/1410.8620, 2014.

[8] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty. Building Watson: An Overview of the DeepQA Project. *AI Magazine*, 31(3):59–79, 2010.

[9] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 518–529, 1999.

[10] G. J. Gordon. Reinforcement Learning with Function Approximation Converges to a Region. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pages 1040–1046, 2000.

[11] X. Guo, S. P. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pages 3338–3346, 2014.

[12] M. J. Hausknecht and P. Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. *CoRR*, abs/1507.06527, 2015.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pages 1106–1114, 2012.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[15] Y. Liang, M. C. Machado, E. Talvitie, and M. H. Bowling. State of the Art Control of Atari Games Using Shallow Reinforcement Learning. *CoRR*, abs/1512.01563, 2015.

[16] N. Lipovetzky, M. Ramirez, and H. Geffner. Classical Planning with Simulators: Results on the Atari Video Games. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[17] M. C. Machado, S. Srinivasan, and M. Bowling. Domain-Independent Optimistic Initialization for Reinforcement Learning. In *AAAI Workshop on Learning for General Competency in Video Games*, 2015.

[18] F. S. Melo, S. P. Meyn, and M. I. Ribeiro. An Analysis of Reinforcement Learning with Function Approximation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 664–671, 2008.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari With Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*, 2013.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 02 2015.

[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *CoRR*, abs/1602.01783, 2016.

[22] G. A. Rummery and M. Niranjan. On-line Q-Learning using Connectionist Systems. CUED/F-INFENG/TR 166, Cambridge University Engineering Dept., September 1994.

[23] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is Solved. *Science*, 317(5844):1518–1522, 2007.

[24] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust Region Policy Optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[25] N. Sprague. Parameter Selection for the Deep Q-learning Algorithm ((Extended Abstract)). In *Proceedings of the Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2015.

[26] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[27] C. Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2010.

[28] G. Tesauro. Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, Mar. 1995.

[29] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. *CoRR*, abs/1509.06461, 2015.

[30] J. Veness, M. G. Bellemare, M. Hutter, A. Chua, and G. Desjardins. Compress and Control. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2015.

[31] C. J. C. H. Watkins and P. Dayan. Technical Note: Q-Learning. *Machine Learning*, 8(3-4), May 1992.