Reinforcement learning in the Atari Learning Environment

Christopher Sipola (s1667278)

19 January 2016

Abstract

This paper evaluates research on various reinforcement learning techniques and planning algorithms when applied to different Atari 2600 games. Techniques explored in this paper include those that conduct explicit feature creation (such as Basic and DISCO), those that use deep neural networks to interpret visual input and map them to actions (namely the deep Q-network algorithm), and those that combine or improve upon these techniques. This paper will also explore the applicability of these algorithms and techniques outside of the Atari environment, as well as discuss how it may influence or motivate research in other fields, such as robotics. Therefore, there will be an emphasis on the generalizability of the learning algorithms and the nature and form of the information that is being used by the learning agent. In general, the deep Q-network algorithm and its close relatives perform best and have the greatest applicability to other domains.

Contents

1 Introduction and motivation

One notable attempt to formally define machine intelligence uses the working definition: "Intelligence measures an agent's ability to achieve goals in a wide range of environments" [?]. Indeed, one of the primary goals of the field 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" [?]. This is progress toward what is usually referred to as artificial general intelligence (AGI), which is an intelligence comparable to humans that can reason and solve a variety of tasks without being programmed for each task separately. AGI is also referred to as "strong AI," "full AI," or "human-level AI." This is more sophisticated than current "narrow" or "weak" AI which is an intelligence that can only be applied to a specific task, such as identifying objects in images.

At the moment, we have not developed AGI. However, it is the implicit or explicit goal of many AI researchers to achieve this level of AI. Demis Hassabis, the CEO of DeepMind Technologies—which is a Google entity that is arguably the world leader in AI research—states that the goals of DeepMind are to "(1) [s]olve intelligence [and] (2) [u]se it to solve everything else". Implied in "solving intelligence" is achieving a general intelligence. Demis says, "More prosaically, how is it that we're going to practically do this? Well, we're going to go about doing this by trying to build the world's first general-purpose learning machine" [?].

However, developing algorithms that can perform a variety of tasks can be challenging if the nature of the inputs—as well as the way in which the algorithm interacts with its environment—differ with each task. For example, it can be difficult to design a general-purpose algorithm that can fly a helicopter and drive a car since the sensory inputs as well as the possible actions that can be taken by the learning agent differ in nearly every way. A reasonable first step, therefore, is to identify or create an environment where the inputs from the environment and the set of possible actions to act on the environment remain the same while the learning tasks vary. This way, researchers can focus on developing an algorithm than can be applied to various learning tasks without the added complexity of having the algorithm adjust itself to changing actions and inputs.

It is therefore no surprise that video games—which have a fixed set of actions (i.e., controller button and joystick inputs) and set sensory inputs (a grid of pixels)—were identified as a fruitful testbed for developing general learning algorithms. In order to harness and best interact with this learning environment, researchers developed an emulator called the Atari Learning Environment (ALE) that serves as an interface between learning agents and the hundreds of games on the Atari 2600 console. The code for the ALE has been shared and is used by many AI research teams at the time of this writing.

Research using the ALE reached peak popularity when the DeepMind team developed an algorithm that can play 41 out of 49 tested Atari games better than a human games tester [?]. Since then, there have been further advances by improving on their technique. These techniques are already being applied to fields outside of video games and have pushed the boundaries and understanding of reinforcement learning.

2 Background

2.1 Reinforcement learning

2.1.1 Overview

The seminal textbook on reinforcement learning defines the field as "one of the most active research areas in artificial intelligence" whereby "an agent tries to maximize the total amount of reward it receives when interacting with a complex, uncertain environment" [**?**]. Unlike supervised learning, the agent is not provided with labels or targets with which it is trained; rather, the agent receives a scalar (that is, single-value) feedback signal to indicate how well the agent is doing [**?**]. This feedback signal is almost always delayed—often significantly—and it is only after many timesteps that the agent knows whether its actions and the results of those actions have produced a desirable result. In order to determine which actions results in the largest total future reward, the agent must use trial-and-error, often choosing actions randomly in the first trial from the available set of options and adjusting its choices gradually given eventual reward feedback.

2.1.2 Key concepts

More formally, at each timestep t, the agent executes some action a_t and receives some observation o_t and reward r_t. The environment, with which the agent interacts, receives action a_t and outputs observation o_t and reward r_t. The history $h_t = a_1, o_1, r_1, ..., a_t, o_t, r_t$ represents all events up to and including timestep t, while the state $s_t = f(h_t)$ is some function of the history that summarizes the information up to timestep t [**?**].[1]

The agent and the environment often have different states—s_t^a and s_t^e, respectively—with the agent state s_t^a often being an imperfect representation of the environment state s_t^e [**?**]. An example of this is an helicopter that uses a reinforcement learning algorithm to perform stunts; this helicopter will have sensory inputs—which may include data from a camera, a gyroscope, and/or an accelerometer—that are used to model the world and the agent's place in it. Indeed, we as humans can also only rely on limited sensory inputs to model the world.

When $o_t = s_t^a = s_t^e$, it is said that the agent has *full observability*, meaning it has access to the model that the environment uses to generate observations and rewards. To return to our analogy before, it is as if the helicopter understands the world perfectly—its exact location and orientation in the air, the wind speed at the current and all future moments, and so on—instead of relying on limited sensory input. In theory, this should be ideal for the agent since it gives the agent as much information as possible for making decisions. Situations where the agent is granted full observability are discussed later in this paper [**?**].

The agent's behavior is defined by its *policy* π, which maps each state s_t^a to an action a_t. The agent's *reward function* maps each state (or state-action pair) to a reward r_t that measures the desirability of the state at that particular timestep.[2] A *value* is total reward expected in the future given a policy; it is the goal of the agent to adjust its policy to maximize this metric. The *state-value function* $V^\pi(s)$ returns the value when starting in a state s and following a policy π. The *action-value function* $Q^\pi(s, a)$ is the value of taking action a in state s under policy π. Finally, the agent's *model* predicts how the environment will respond to an action [**?**].

[1] This summary can be imperfect in that information can be lost.

[2] Not every timestep will have a reward.

2.1.3 Q-learning

Determining value is a central problem in reinforcement learning. One recent breakthrough for doing so is the Q-learning algorithm, which approximates the optimal action-value function $Q^*(s_t, a_t)$ using the update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big) \tag{1}$$

where α is the learning rate and γ is the discount factor [?]. This paper will not get into the details of the algorithm or how the update works, but it is helpful to have some of the notation here to aid in discussion of improvements made to the algorithm.

2.2 The Atari Learning Environment (ALE)

Atari 2600 was a popular video game console created in 1977 for which hundreds of games were created, including Asteroids, Centipede, Breakout, Pac-Man, Pong, and Space Invaders. The console RAM is just 128 bytes (1024 bits). The screen is 160×210 pixels, with 128 possible colors. The joystick allows for 18 possible actions or commands from the joystick and the single button [?].

Researchers in 2012 released the Atari Learning Environment (ALE), a video game emulator and "software framework designed to make it easy to develop agents that play arbitrary Atari 2600 games" [?]. This proved to be a fruitful testbed for reinforcement learning algorithms for three reasons:

- Atari has a large collection of highly-varied games, allowing researchers to test algorithms that perform a variety of tasks in a single environment;
- Because Atari 2600 was build using very simple hardware, the ALE emulator can run reinforcement learning models much faster than actual gameplay would allow; and
- "the Atari state and action interface is simple enough for learning agents, but complex enough to control many different games" [?].

3 Approaches to creating Atari game-playing agents

3.1 Explicit feature creation: Basic, BASS, DISCO, LSH, and RAM

When creating their reinforcement learning algorithms, the researchers that created ALE favored more explicit feature construction using five methodologies: Basic, BASS, DISCO, LSH, and RAM [?].

Basic and BASS work by removing the background image and creating binary features—one for each color—for tiles of pixels. DISCO subtracts the background (as in Basic and BASS) but then labels the resulting objects so that if can then give them a position and velocity. LSH projects the image data into a smaller number of features and performs random projections, increasing the likelihood that similar images can be associated with each other. RAM creates a binary feature from the Atari's 1024 bits of memory, which is notably different from the other four methodologies in that it grants full observability (that is, $o_t = s^a_t = s^e_t$) to the agent [?].

Researchers used these extracted features as inputs into the SARSA(λ) model, which is a "traditional technique for model-free reinforcement learning" [?]. SARSA(λ) is a take on the

SARSA model, which uses the information from the elements $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ (hence the name SARSA) to determine the "transition from one state-action pair to the next" [**?**].

These methods were tested on 55 games. In 40 games, performance of BASS seemed best according to the evaluation methods. Surprisingly, RAM was not a top performer despite having full observability into the environment state. This is likely because the representation of the game state as an image carries more useful information for the learning problem than do the raw RAM bits [**?**].

3.2 Explicit feature creation: HyperNEAT-GPP

HyperNEAT is an extension of the Neuro Evolution of Augmenting Topologies (NEAT) algorithm [**?**] and, like a DQN, is built on a neural network. Also, "[i]n comparison to standard NEAT, HyperNEAT's encoding allows it to take advantage of geometric regularities present in many board and 2D games, such as those in Atari 2600". However, unlike DQN, there is also explicit feature extraction where the algorithm categorizes objects (”blobs” of pixels) based on a similarity score. The "self blob"—the blob carrying out the actions of the agent—is identified using a separate process [**?**].

This methodology was tested on just two games—Asterix and Freeway. While the methods outperformed others of its time in 2012 for those games (including BASS, DISCO, and RAM), it is hard to argue that the goal of a generalized algorithm that can be applied to varied tasks had been achieved [**?**].

3.3 Planning: breadth-first search and UCT algorithm

Planning is a technique that does not fall under the category of reinforcement learning but is worth mentioning as a basis of comparison with the reinforcement learning algorithms. It involves searching through state-action pairs, using the game emulator's to carry out determine the outcome at each event. In theory, if all branches could be searched—without computational limitations—then this "perfect model" should perform better than reinforcement learning techniques. However, this is computationally infeasible given that the agent can choose from 18 actions 60 times in one second, giving $18^{60} \approx 10^{75}$ branches to search to plan just one second ahead [**?**].

One planning technique tested is breadth-first search, which allows the agent to plan 1/3 of a second into the future. Another is UCT (upper confidence bounds applied to trees) algorithm, which pares down the number of branches to those most likely to give the highest value and collapsing essentially duplicate state-action branches (say, if two actions or commands result in the same state) before conducting a search [**?**].

These search methods performed better than Basic, BASS, DISCO, LSH, and RAM in most games. However, each action-step took around 15 seconds and, as mentioned earlier, the emulator itself was used for planning as opposed to an agent-created model using only observations and rewards. The games where planning performed worst were in games with rewards that were more delayed or when long-term planning and strategy were required [**?**].

3.4 Automatic feature creation: deep Q-network (DQN)

Researchers at Google DeepMind created a model termed a deep Q-network (DQN) to automatically detect features from the input pixel data and approximate the optimal action-value function $Q^*(s_t, a_t)$. This model is the combination of reinforcement learning and a deep convolutional neural network [?]. A deep convolutional neural network is a type of neural network—a machine learning model that is loosely based the functionality of neurons in the brain—that has been optimized to process two-dimensional visual data by taking advantage of the spatial relationships between pixels [?].

One typical issue with such an approach is that the calculation of $Q^*(s_t, a_t)$ is unstable when approximated by a neural network, mostly due to natural correlation in the data—that is, the observations are not independent and identically distributed like they typically are in other applications of neural networks. To remedy this, the authors decorrelate the data with a "novel variant of Q-learning" that employs two methodologies: (1) experience replay, which is a way to shuffle the data; and (2) an "iterative update that adjusts the action-values (Q) towards target values that are only periodically updated [?].

Experience replay, a technique inspired by neuroscience [?] but developed in the field of robotics, is characterized as the learning agent "remember[ing] its past experiences and repeatedly present[ing] the experiences to its learning algorithm as if the agent experienced again and again what it had experienced before" [?]. This allows for greater data efficiency in that each experience can be used for multiple updates to the weights [?]. To do this, experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored in data set $D_t = \{e_1, ..., e_t\}$, and then the learning algorithm draws uniform random samples $(s, a, r, s') \sim U(D)$,[3] using the loss function

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right] \tag{2}$$

for the Q-learning update at iteration i for parameters θ.[4] The equation is given here in case one would like to compare to equation 1 at a high level; for a more in-depth explanation of this Q-learning update, see [?].

Of the 49 games on which they were tested, DQN performed better than all other game-playing methods up to that point (which includes BASS, DISCO, HyperNEAT, and planning using breadth-first search and UCT) on 43 of the games. DQN also "performed at a level that was comparable to that of a professional games tester across the set of 49 games, achieving more than 75% of the human score on more than half of the games" [?], where the human score is computed as "the median reward achieved [by the human] after around two hours of playing each game" [?].

The larger success of this approach is that the algorithm, network architecture, and hyperparameters were kept constant when testing across games, meaning this does indeed meet the authors' goal of creating an "artificial agent that is capable of learning to excel at a diverse array of challenging tasks" [?].

3.5 Improvements to DQN

DQN now seems to be the gold standard for Atari game playing algorithms. There have been advances to DQN as presented by the Google DeepMind team in their 2013 and 2015 papers, but these improved techniques still rely on DQN at their core. Some examples of these are double Q-learning, combining DQN and planning algorithms, using a massively distributed architecture,

[3]The apostrophe symbol represents the time period following that of the other variables. The change is terminology is because we are no longer at time t.

[4]The Q-network parameters θ_i are included in the notation here to distinguish between the usual parameters θ_i and the ones that are kept constant between updates θ_i^-.

and performing the Q-learning update using more significant experiences instead of choosing them uniformly randomly.

3.5.1 Double Q-learning

Q-learning has the problem of overestimating action-values, resulting in policies that are not ideal according to the data. This overestimation comes from preferring overestimated action-values by using the max term in equation 1. Researchers have resolved this issue with an adaptation to the DQN—called double Q-learning—which "not only reduces the observed overestimations, as hypothesized, but that this also leads to much better performance on several games." It does this by using different sets of weights for updating the policy and for determining value, therefore separating the selection of an action and its evaluation [**?**].

3.5.2 Combining DQN and planning

Models that combine planning and DQN perfom much better than DQN alone, but as mentioned earlier, planning "exploit[s] information that is not available to human players, and [is] orders of magnitude slower than needed for real-time play"[**?**]. This is interesting but is perhaps not very useful for applications outside of situations where the agent has full observability.

3.5.3 Parallelization

Researchers have adapted DQN to run in a massively distributed architecture. The "architecture uses four main components: parallel actors that generate new behaviour; parallel learners that are trained from stored experience; a distributed neural network to represent the value function or behaviour policy; and a distributed store of experience." [**?**]. This parallelization reduced run time for most games by an order of magnitude while also improving performance on 41 of the 49 games.

3.5.4 Prioritized experience replay

In equation 2, experiences are drawn uniformly from a bank of experiences in order to perform Q-learning update. However, this ignores the significance of the experiences. Researchers have found sampling experiences that are more likely to result in significant learning can lead to better policy updates. This sampling scheme outperforms uniform experience sampling in 41 of the 49 games [**?**].

4 Impact on reinforcement learning and other AI fields

It is perhaps unreasonable when assessing the importance of research conducted using the ALE to expect each breakthrough to translate directly to a "real-world" application. Even without direct applicability, researchers can (and have) learned a great deal about reinforcement learning and other involved algorithms through the ALE.

Despite this, the algorithms developed in the ALE *are* being adopted directly in research aiming to solve "real-world" problems. Some notable ones are novel applications of visualization techniques to exploring the "brain" of learning agents, the prediction of video output, and using DQN to solve simulated physical tasks—all discussed below.

4.1 Visualization of state-actions

In an effort to understand representations of state-actions learned by the DQN algorithm—and how these states are similar or dissimilar in terms of reward and value—researchers used the a technique called t-SNE, which places each point of a high-dimensional data set onto a two- or three-dimensional map [**?**]. This technique allows researchers to view game states that are visually dissimilar but produce similar rewards, or vice versa [**?**]. For example, in Space Invaders, there are destructible bunkers behind which the player can hide until they are destroyed; in late game, these bunkers provide little value, so visually dissimilar states—such as when there are and are not bunkers still in the game—produce similar rewards. This is a novel application of the relatively new t-SNE technique which can be used to increase our understanding of the inner workings of reinforcement learning algorithms in the future.

4.2 Predicting video input

Researchers have used convolutional neural networks and recurrent neural networks[5] to generate realistic images for up to 100 "action-conditional" timesteps into the future. Action-conditional means that if the agent is at timestep t and wants to predict the image produced by the game at timestep $t+s$, then it must assume an action at each timestep between t and $t+s$. When generating predictions, the agent follows its policy and chooses what it believes to be the best action at each timestep between t and $t+s$. [**?**].

This algorithm was "the first to make and evaluate long-term predictions on high-dimensional video conditioned by control inputs." Since the algorithm is not domain-specific, the authors expect that it will generalize to other vision problems in reinforcement learning [**?**].

4.3 Applications to robotic control

The DQN algorithm had already adapted to solve more than 20 simulated physics tasks, including "classic problems such as cartpole swing-up, dexterous manipulation, legged locomotion and car driving" [**?**]. The performance of the learning agent that received raw pixels as input performed nearly as well as a planning agent that was granted full observability. This has obvious implications in the field of robotics, where an agent receives rewards and state data from sensory inputs while having a limited number of actions at its disposal.

5 Conclusion and suggestions for future research

Using the ALE, researchers have developed reinforcement learning algorithms that can outperform humans on the majority of games tested on the Atari 2600. The Google DeepMind team claims in one paper that they have created "the first artificial agent that is capable of learning to excel at a diverse array of tasks," and this does not seem to be far from the truth: the learning agent beat human scores in the vast majority of tested games, which was something that was not achieved by agents developed in prior research [**?**]. In creating their DQN algorithm, they have pushed the boundaries of reinforcement learning and AI.

[5]A recurrent neural network is a neural network that simulates memory by allowing for directed cycles between units in the network.

However, there is still much room for future research. To start, it may be most fruitful for researchers to only focus on developing learning algorithms that do *not* have full observability. Full observability is not useful for many applications outside of video games, and it is not interesting for the purpose of developing AGI since any AGI will often not have a perfect model of environment with which it is interacting. This may mean slowly shifting focus away from pure planning algorithms or algorithms that can observe the environment state directly (e.g., RAM).

In the short term, it may be useful to use audio data from the games as an additional sensory input. Seasoned players of video games understand that some information about the game state is expressed through audio cues. For example, significant game events are often paired with specific sounds or musical motifs. In some games and in some situations, information about the game state is *only* expressed through audio cues. For example, in some games, danger is represented by an increase in the speed of the music, without any visual indicator. Developing learning agents that can learn from audio cues may help to push the reinforcement learning research further by exploring how learning agents use multiple types of sensory inputs—and whether there are any interesting interactions and benefits from doing so.

Since perception systems in robotics rely on image inputs that are based on three-dimensional space, it could be useful to learn from three-dimensional games. DeepMind is already exploring this through their creation of DeepMind Lab, which is a virtual environment resembling "a blockish 3-D first-person shooter computer game" where "an AI agent takes the form of a floating orb that can perceive its surroundings, move around, and perform simple actions" [**?**]. The researcher can program tasks to be learned by the agent, thereby granting much more control over the learning problem than with pre-made games. DeepMind Lab has been released for the use of any AI researcher.

The DeepMind team is also planning on applying learning to more complex video games, announcing a collaboration with the video game developer Blizzard Entertainment to develop StarCraft II as an AI research environment—much in the same way Atari 2600 was developed into an environment for the use of any AI researchers [**?**]. StartCraft II is a real-time strategy (RTS) video game where players command units to gather resources, which are then spent on buildings and military units that are used to attack and eliminate another enemy on the map. This game is played competitively, and professional players often make 200 to 300 actions per minute, where actions may include telling units to move or attack, buildings to create units, or workers to construct buildings. In this game, there is incomplete information due to a fog-of-war mechanic, exponentially more possible actions than in an Atari game, and much more complex strategy—often through the use of deception and the application of game theory through an understanding of a "metagame".[6] Research using the StarCraft II learning environment will at the very least reveal the strengths and the limitations of reinforcement learning, while the most optimistic result is the creation of a learning agent that can apply very-high level strategy from choosing from an extremely large number of possible actions in a short timeframe.

A more ambitious goal is for researchers not to "aim for a single architecture that can, disjointly, learn an array of separate tasks, but for one that can incrementally build on skills learned on previous tasks to perform more complex ones" [**?**]. This expands the working definition of machine intelligence mentioned at the beginning of this paper and is a notably different problem

[6]The "metagame" describes the evolving set of popular strategies and those used to counter them. For example, if strategy A is popular and expected, then a player may commit to using a strategy B which counters strategy A. Over time, if strategy B becomes expected, a different strategy C may be used, which may itself be countered by either a prior strategy like A or a new strategy D.

addressed by the reinforcement learning algorithms described here. If using this expanded definition of intelligence, researchers may have to use different tasks to develop their agents and change the benchmarks they use to evaluate success. For example, outside of the nature of the tasks, researchers may want to train their agents on tasks that are of a lower complexity than the tasks they use to evaluate the agents.

AI researchers, through their creation of the DQN model using the Atari Learning Environment, have created learning agents that are capable of performing varied tasks at a level comparable to a human—with no change to the underlying algorithm, architecture, or hyperparameters. The speed of advancement in this research shows no signs of slowing down, as evidenced by the recently released DeepMind Lab and the announcement of the StarCraft II AI research environment. The goal of creating learning agents capable of playing complex 3-D games or games with vastly more possible actions than Atari games may seem audacious, but it is exactly through the setting of audacious goals that AlphaGo and other breakthrough AI have been created. Furthermore, the payoff from this research is already evident in applications to robotics and to other types of problems within reinforcement learning.

References