Playing Games with Deep Reinforcement Learning: A Survey Paper

Morgan Jones

Abstract—This paper serves as an introduction to Deep Reinforcement learning and an overview of its development using the domain of Atari 2600 as a benchmark. This survey details the background required for deep reinforcement learning and covers selected central algorithms in the field. To conclude, it describes some leading-edge research and identifies goals for future direction.

Index Terms—Computer Society, IEEE, IEEEtran, journal, LaTeX, paper, template.

1 Introduction

One of the main goals of artificial intelligence (AI) is to produce fully autonomous agents that can learn from experience the optimal behaviour for their environments. Reinforcement Learning (RL) provides a framework for such experience-driven learning and the advent of deep learning has enabled RL to tackle problems previously intractable, opening the door to amazing possibilites. The Deep Q-Learning (DQN) algorithm [1] was introduced by DeepMind to beat previous state-of-the-art agents at playing Atari 2600. This ignited the field of Deep Reinforcement Learning. Since then, growth of the field has been exponential.

Deep RL has achieved great success "a decade ahead of its time" [2] as AlphaGo [3] defeated Lee Sedol the 18-time Go world champion with a 4-1 victory in Seoul, South Korea. It was thought impossible due to the vast game-tree complexity of Go 10^{360} (chess is 10^{123} for comparison). The same authors then developed AlphaZero [4] that beat each world-champion computer program in the games of Chess, Shogi and Go. Pluribus achieved superhuman performance at 6-player no-limit Texas Holdem when competing against elite human players in a series of matches [5]. Two of the players being past 1st place winners of the poker world series. This represents a massive milestone for AI as the algorithm deals with a partially observable six-agent environment. Many real-world strategic interactions happen in similar environments. OpenAI used RL to train a robotic hand to solve a Rubiks cube then transferring what was learnt in simulation to be done on physical actuators in the real world with amazing results [6]. AlphaStar [7] achieved grandmaster level in StarCraft II, reaching top of the league and beating 99.8% of officially ranked human players.

Games like these present an interesting challenge and testbed for Deep RL but they are only the beginning. The vision behind Deep RL is of creating systems that learn and adapt in real world domains.

• *M. Jones is with the Department of Computer Science, Aberystwyth University, Wales, SY23 3FL.*
E-mail: mwj7@aber.ac.uk

Last revised May 5, 2020.

2 Background

2.1 Reinforcement Learning

RL is an iterative trail & error, reward & punishment process over time. At each time step the RL agent acts upon its environment by taking an action. The environment then provides feedback to the agent in the form of an immediate reward. The goal of reinforcement learning is to train an agent through experience to act upon its environment such that the environment yields the most reward to the agent.

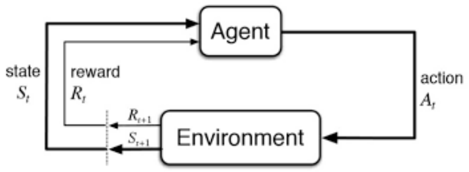


Fig. 1. Interaction between agent and environment in reinforcement learning.

RL differs to Supervised Learning [8] because the targets for learning change over time as the agent learns from interactions with the environment. This is different from supervised learning where the targets for learning are fixed from the beginning.

2.1.1 Markov Decision Process

Reinforcement Learning is traditionally modelled as a Markov Decision Process (MDP) [9]. A MDP is a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards. It is defined as a set of states, actions, a transition model, a reward function, and an initial state distribution $\langle S, A, P(s|s,a), R(s), \rho_0 \rangle$.

S	States are the set of environment states. The current state being the state of the environment the agent is currently experiencing.
A	Actions are the set of actions the agent can take. At each state only a subset of actions may be available.

$P(s'\|s,a)$	Transition model is the probability of successful transition for each state-action pair. In deterministic environments this would be 1 but the general formalism of an MDP includes stochastic environments.
$R(s)$	Reward function is the reward the environment yields to the agent for being in a specific state.
ρ_0	Initial state distribution defines the set of states the agent can start in.

At any given time-step t, an agent perceives its state s_t and selects an action a_t. The environment responds by giving the agent some reward r_t and transitioning the agent into state s_{t+1} (s'). When a terminal state is reached a run of the sequential problem is over. A sequence of states and actions between the initial state and the terminal state is called a trajectory, denoted by T.

Diagram of MDP

It is an assumption of MDPs that all transitions are **Markovian**, that is, the probability of reaching s' from s is entirely dependent on s and not any previous states. This means that the value of the future is independent of the past and can only be decided by the actions taken in the present.

The goal of a RL agent is to act in a way that gets as much reward from the environment as possible. In an MDP the goal of an agent is to maximise the expected total future discounted rewards. If an agent can act in this way it will have solved the MDP. Solving an MDP will solve the reinforcement learning problem.

$$\max[\sum_{t=0}^{T} \gamma^t R(s_t)] \tag{1}$$

We say discounted rewards because each intermediate reward is typically discounted by a factor γ $(0, 1]$. The discount factor describes the preference of an agent for current rewards over future rewards, it is the trade-off between short-term and long-term planning. When γ is close to 0 future rewards are viewed as insignificant. When γ is close to 1 future rewards are viewed as almost equivalent to current rewards.

Not all Deep RL uses an MDP as a model although most use at least some part of it. This is the difference between model-free and model-based RL algorithms and something that will be detailed later. It is my experience that being able to visualise the problem as an MDP is beneficial to reading the literature.

2.1.2 Value & Policy based Solutions

An MDP can be solved by learning an optimal policy π^*. A policy is a mapping of states to actions $\pi : S \to A$ that tells our agent what action to take at each step along its trajectory. An optimal policy π^* is a policy that yields the highest expected value from each state (2) and consequently the highest reward from the MDP (3).

$$\pi^*_s = \max_{\pi} V^{\pi}(s) \tag{2}$$

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{3}$$

Another solution is to calculate the value of each state and use the state values to select an optimal action in each state. The value of each state is defined recursively by the **Bellman Equation**.

$$V(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|, s, a) V(s') \tag{4}$$

Traditional solutions to an MDP would perform value-iteration or policy-iteration. Value-iteration uses a dynamic programming approach in the form of Bellman Updates to converge on the actual values of each state. Policy-iteration repeatedly evaluates and improves a policy. The evaluation calculates the value of each state according to the policy. The improvement calculates a new policy by considering a one-step reward look-ahead that updates each of the state values previously evaluated. Iteration terminates when policy improvement yields no change in state values.

2.2 Deep Learning

Deep Reinforcement learning is the application of Deep Learning to the Reinforcement Learning problem. Deep learning is a subfield of Machine Learning (ML) that is concerned with the use large artificial neural networks [8]. These deep networks often have many intermediate layers and thousands if not millions of parameters. Application of these networks allows RL to scale to problems with high-dimensional state and action spaces that were previously intractable.

Deep neural networks are used to approximate the solutions to an MDP therefore they approximate either the policy function or the value function. Policy networks typically output as a softmax over actions whereas value networks output a numerical value. There are three options for the value function a deep neural network can approximate.

- State value function $V^{\pi}(s)$
- State-action value function $Q^{\pi}(s, a)$
- Advantage function $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$

$V^{\pi}(s)$ is the value of being in a state s assuming the agent continues to follow policy π. $Q^{\pi}(s, a)$ is the value of being in a state s and taking action a assuming the agent continues to follow policy π. $A^{\pi}(s, a)$ describes how much better it is to take a specific action a in state s over other actions assuming the agent keeps acting according to policy π. This is useful because sometimes we do not need to describe how good an action is in an absolute sense, but only how much better it is than others on average.

2.2.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is the most common type of neural network used in Deep RL. CNNs compose non-linear modules that transform raw data into more abstract representations that can then be used for classification REF. CNNs are deep-learning methods designed to deal with data in the form of multi-dimensional arrays such as images. CNNs are often called feature extractors in the sense that each convolutional layer will extract higher level (more abstract) features. The first layer perhaps identifying lines and edges, the next layer identifying shapes, and later layers recognising whole objects within an image.

CNNs are composed of Convolutional layers, Pooling Layers and Fully-Connected Layers (also referred to as dense layers).
A **convolutional layer** contains a set of filters (also referred to as neurons or kernels). Each filter is a 3D array with a set width & height known as the receptive field and a depth equal to the depth of the input. Filters slide or convolve over the input performing local element-wise multiplication and summing the result. This weighted sum is then passed through a non-linearity, typically a ReLU activation function $f(x) = max(0, x)$. Filters typically shift one unit at a time over the input. Strides can be used to reduce the dimensionality of the output, by shifting more than one unit at a time. After sliding the filters of a convolutional layer over all the locations, the layer produces an output called an activation map or feature map. The output activation maps are then used as input to proceeding layers.
Pooling Layers merge neighbouring features into one. They perform down sampling along the spatial dimensionality of the given input. Pooling layers decrease the dimensionality of the representation and therefore the number of network parameters in later layers. They also make the feature maps more robust to small shifts and distortions of the feature image. This is known as local translation invariance. There are multiple types of pooling layers. Min Pooling, Max Pooling and Average Pooling are examples [REF].

Fully-Connected Layers are multi-layer perceptions and perform similar functions found in shallow neural networks. It is the job of the fully-connected layers to classify the output of the preceding convolutional and pooling layers.

2.2.2 Recurrent Networks

Recurrent neural networks are used in deep RL and often trailed as alternative architectures. The Deep Recurrent Q-Network (DRQN) algorithm uses a recurrent architecture with DQN to play a partially observable version of Atari [10].
Recurrent networks are neural networks with memory that specialise in processing sequential data. Output of hidden layers are stored in memory neurons. The memory neurons are passed back to the hidden layer along with the next input from the sequence as it is feed to the network.
The two most common types of recurrent network are Gated Recurrent Unit (GRU) [11] and Long Short-Term Memory (LSTM) [11]. Recurrence can be used in conjunction with CNNs as a single hybrid architecture.

3 Taxonomy

3.1 Model-Based & Model-Free

Some agents have access to (or learn) a model of the environment. This allows agents to plan, by thinking time steps ahead, what the result of the environment would be for future actions they could take. Algorithms using an environment model are called model-based, those that do not are called model-free.

3.1.1 Model Given & Model Learned

Sometimes a model of the environment is given to the agent in the form of a simulator, sometimes it has to be learnt by the agent from scratch. Often a ground-truth model of the environment is unavailable because the true environment dynamics are unknown. AlphaZero [4] is a famous example of a model-based agent that learns the model from scratch, because at the beginning of training it does not know any of the game rules.

3.2 On-Policy & Off-Policy

Off-policy RL is when the behavioural policy is different from the target policy. On-policy RL is when the behavioural policy is that same as the target policy. The behavioural policy is the policy the agent uses when acting on the environment to generate new experiences. The target policy is the policy that is updated when learning from past experiences.

3.3 Value-Based or Policy-Based

If the agent is trying to solve the problem by learning an optimal policy function the algorithm is policy-based. If the agent is trying to learn an optimal value function the algorithm is value-based. Actor-critic methods are hybrid algorithms where an agent is trying to learn both the policy and value functions. The actor (policy) acts in the environment and the critic (value) evaluates the actor's actions and dictates improvements.

4 Key Algorithms

4.1 Deep Q-Learning with Experience Replay

Deep Q-Learning with Experience Replay (DQN) [1] substantially launched the field of Deep RL. DQN uses a RL framework and deep learning to play Atari 2600 games implemented in the Arcade Learning Environment [12]. Instead of using traditional value iteration (5) algorithms the authors used a CNN to approximate the optimal state-action value function Q*(s,a) (6). The parameters of the network are represented by θ.

$$Q^*(s,a) = \mathbf{E}_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q^*(s',a')|s,a] \tag{5}$$

$$Q^*(s,a) \approx Q(s,a;\theta) \tag{6}$$

The CNN takes in raw visual input of the Atari games as states and outputs a Q-value for each possible action. The agent then plays Atari by selecting actions according to the predictions of the network. The game state is inputted to the CNN as 4 stacked histories each grey-scaled, down sampled and cropped into an 84x84 region. Multiple histories are used because a state cannot be fully represented in a single frame, for example, direction and velocity of the ball in Pong cannot be shown by a single frame.
At each time step DQN stores experiences in a replay buffer to learn from later. This is significant because the batching and sampling of past experience enables stable training of the deep CNN. The network is trained by minimising a sequence of loss functions. Each function's target for learning y_i comprises the next state's Q-value predicted by the previous iterations network and an immediate reward returned from the environment (8). Stochastic Gradient

Descent (SGD) is used when updating the weights of the network according to the loss (7).

$$L_i(\theta_i) = \mathbf{E}_{s,a\sim p(.)}[(y_i - Q(s,a;\theta_i))^2] \tag{7}$$

$$y_i = \mathbf{E}_{s'\sim\varepsilon}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a] \tag{8}$$

The network is trained for 10 million frames with a replay memory of the 1 million most recent frames. During training an ε-greedy policy is used for action selection with greediness being annealed linearly from 1 to 0.1 over the first million frames. ε-greedy selection is a trade-off between exploration and exploitation selecting the highest Q-value action with probability $1-\varepsilon$ or an alternative action uniformly at random. In other words, the greater the value of ε the more exploration and visa-versa.

Q-value increases over training showing deep networks can be stably trained to solve RL problems (Fig. 3). The results against previous art and human players show DQN outperforms any traditional RL approach and even human scores on 3 of the 7 games tested.

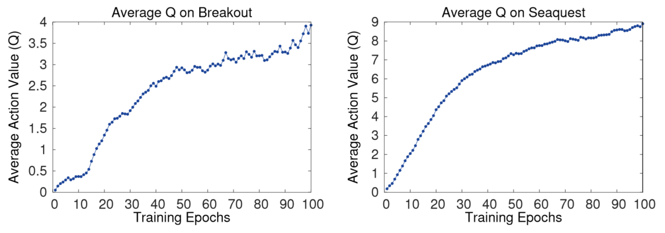


Fig. 2. State-Action value learning progress of DQN on two Atari 2600 games, Breakout (left) and Seaquest (right).

There have been many improvements and alterations to DQN since its publication. Deep Recurrent Q-Networks [10], Double DQN [13], Prioritized DQN [14], Duelling DQN [15], Distributional DQN [16] and Noisy DQN [17] are a few examples. Publications have successfully combined improvements to DQN. RAINBOW [18] is an algorithm that integrates many of the previously mentioned DQN variants into a single hybrid algorithm resulting in better Atari performance.

4.1.1 *DQN is value-based, model-free and off-policy*

This algorithm is a value-based algorithm because it attempts to learn the optimal state-action value function. It does not concern itself with learning the optimal policy, this would be role of a policy-based algorithm. DQN is model-free RL because it only predicts state-action values and does not learn an MDP model of the environment. DQN is off-policy RL because the behavioural policy is simply an ε-greedy selection.

4.2 Ayncrhonous Advantage Actor-Critic

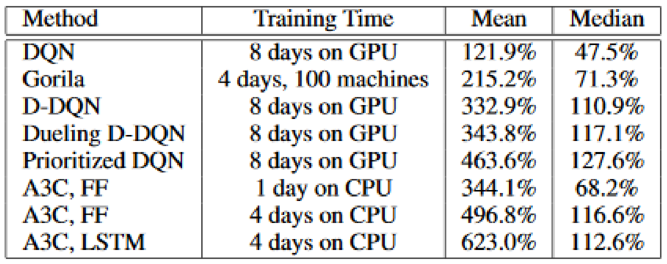
The Asynchronous Advantage Actor-Critic (A3C) algorithm was introduced in a paper by Mnih et al. [19] that presented a general asynchronous framework for training deep RL agents. The framework used parallelism in the form of simultaneous actor-learner agents running on different CPU cores. Each actor-learner explores environment states using different exploration policies and, as they learn, update a global parameter set that is then be propagated to all other actor-learners. The framework reduced the hardware requirements for training Deep RL agents, provided a stabilising role to the learning that was previously done exclusively through experience replay, and enabled a large range of RL algorithms to be used effectively in Deep RL.
To test their framework the authors implemented asynchronous variants of four popular RL algorithms. The most successful was A3C an on-policy actor-critic policy gradient method. A3C uses an advantage function (9) as a critic computed by calculating feed forward n-step returns (10). The n-step returns take into account n immediate rewards reutrned from the environment and state evaluations predicted by the network $V(s_t;\theta_v)$. A3C maintains a policy $\pi(a_t|s_t;\theta)$ and an estimate of the value function $V(s_t;\theta_v)$. A single CNN approximates both policy and value with a separate output head for each. The agents policy and value function are updated after a set number of actions are taken or a terminal state is reached. The update of the network performs gradient ascent using the **Policy Gradient Theorem** (11) [REF]. The authors use RMSProp [20] for optimising updates.

$$A(s,a) = Q(s,a) - V(s) \tag{9}$$

$$A(s,a) \approx (\sum_{i=0}^{k-1} \gamma^i r_{t+i}) + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v) \tag{10}$$

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta') A(s_t,a_t;\theta,\theta_v) \tag{11}$$

A3C beat previous art at playing Atari 2600, including DQN. A3C outperformed all agents in half the training time while being trained using 16 CPU cores, other agents were trained using an Nvidia K40 GPU.



Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Fig. 3. Performance of A3C against previous Atari agents. Table shows Mean and Median human-normalised scores on 57 Atari games. Two architectures are compared for A3C, a feed forward network (FF) and long-short term memory network (LSTM).

5 State-of-the-art Work

5.1 MuZero

6 Future Directions

Goals for the future...

6.1 Interpretability & Explainable AI

Model Extraction. Model Visualisation.

6.2 Real World Domains

7 Conclusion

We need a conclusion.

Appendix A
Proof of the First Zonklar Equation

Appendix one text goes here.

Appendix B

Appendix two text goes here.

Acknowledgments

The author would like to thank David Silver for his fantastic, freely available, lecture series [REF] on Deep RL and OpenAI for their "Spinning Up in Deep RL" guide [REF] that presents a great introduction to the field and collation of key papers for researchers just starting out.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] DeepMind, "Alphago page," https://deepmind.com/research/case-studies/alphago-the-story-so-far, accessed on 2020-05-08.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[5] N. Brown and T. Sandholm, "Superhuman ai for multiplayer poker," *Science*, vol. 365, no. 6456, pp. 885–890, 2019.

[6] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[7] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[9] S. Russel and P. Norvig, *Artifical Intelligence A Modern Approach*. Pearson Education, 2010, pp. 645–648.

[10] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *AAAI Fall Symposium Series*, 2015.

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[12] 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*, vol. 47, pp. 253–279, 2013.

[13] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[14] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[15] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[16] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 449–458.

[17] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin *et al.*, "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[18] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[20] T. Tieleman and G. Hinton, "Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude," 2012.