Playing Games with Deep Reinforcement Learning: A Survey Paper

Morgan Jones

Abstract—This paper serves as an introduction to Deep Reinforcement learning and a walkthrough of its development using the domain of video games as a benchmark. This survey details the background required for deep reinforcement learning and covers 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 with the advent of deep learning enabling RL to tackle problems previously intractable. The Deep Q-Learning (DQN) algorithm was introduced by DeepMind to beat previous state-of-the-art agents at playing Atari 2600. This ignited the field of Deep Reinforcement Learning; from then on, growth of the field has been exponential.

Deep RL has achieved great success a decade ahead of its time[REF DEEPMIND HOMEPAGE] as AlphaGo [REF] 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 [REF] that beat each world-champion computer programs 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 [REF]. 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 [REF].

AlphaStar [REF] achieved grandmaster level in StarCraft II, reaching top of the league and beating 99.8% of officially ranked human players.

Video games present an interesting challenge and testbed for Deep RL but they are not the end goal. 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

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.

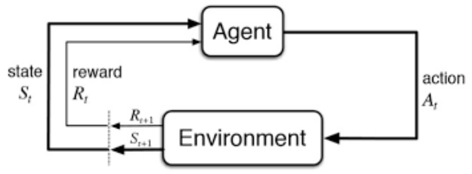

Fig. 1. Interaction between agent and environment in reinforcement learning.

RL differs to Supervised Learning 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) [REF]. 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 and consequently the highest reward from the MDP.

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

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

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 [2]. 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 that were previously intractable. Settings that have high-dimensional state and action spaces, domains where the input is visually rich or where input is required to be encoded over a long history.

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 also 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.

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 as input for the next item of the input sequence that is feed to the recurrent network.

3 Taxonomy

4 Key Algorithms

DQN, A3C, ...etc.

4.1 Deep Q-Learning with Experience Replay

4.2 Ayncrhonous Advantage Actor-Critic

5 State-of-the-art Work

AlphaZero, MuZer ..etc.

6 Future Directions

Goals for the future...

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 authors would like to thank Demis Hassabis and David Silver for being such inspirational lecturers. REMOVE THIS!

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[2] Y. LeCun, Y. Bengio and G. Hinton. *Deep learning* Nautre, 521, 436-444, 2015