POLITECNICO DI MILANO
Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

FAST REINFORCEMENT LEARNING USING DEEP STATE FEATURES

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Supervisor: Prof. Marcello Restelli
Co-supervisors: Dott. Carlo D'Eramo, Matteo Pirotta, Ph.D.

Master's Thesis by:
Daniele Grattarola (student ID 853101)

Academic Year 2016-2017

Contents

1 State Of The Art **3**
1.1 Value-based Deep Reinforcement Learning 3
1.2 Other approaches 5
1.2.1 Memory architectures 5
1.2.2 AlphaGo 6
1.2.3 Asynchronous Advantage Actor-Critic 7
1.3 Related work 7

References **9**

Chapter 1

State Of The Art

The integration of RL and neural networks has a long history. Early RL literature [18, 22, 2] presents *connectionist* approaches in conjunction with a variety of RL algorithms, mostly using dense ANNs as approximators for the value functions from low-dimensional (or engineered) state spaces. The recent and exciting achievements of DL, however, have caused a sort of RL *renaissance*, with DRL algorithms outperforming classic RL techinques on environments which were previously considered intractable. Much like the game of Chess was believed out of the reach of machines until IBM's *Deep Blue* [4] won against the world champion Garry Kasparov, DRL has paved the way to solve a wide spectrum of complex tasks which were previously considered a stronghold of humanity.

In this chapter we present the most important and recent results in DRL research, as well as some work related to the approach that is proposed in this thesis.

1.1 Value-based Deep Reinforcement Learning

In 2015, Mnih et al. [14] introduced the *deep Q-learning* (DQL or, more commonly, DQN) algorithm which we detailed in Section **??**, and basically ignited the field of DRL. The important contributions of DQN consisted in providing an end-to-end framework to train an agent starting from the pixel-level representation of the *Atari* environments, which proved to be more stable than previous approaches thanks to the use of *experience replay* [12]. Moreover, the same architecture was reused to solve many different games without the need for *hyperparameter tuning*, which proved the effectiveness of the method. From this work (which we could call *introductory*), many improvements have been proposed in the literature. Van Hasselt et al. (2016) proposed *Double DQN* (DDQN) [23] to solve an over-estimation issue in DQN due to the max operator used in the parameters update (see Algorithm **??**). This approach used two separate CNNs: an *online network* to select the action for the collection of samples, and a *target network* to produce the

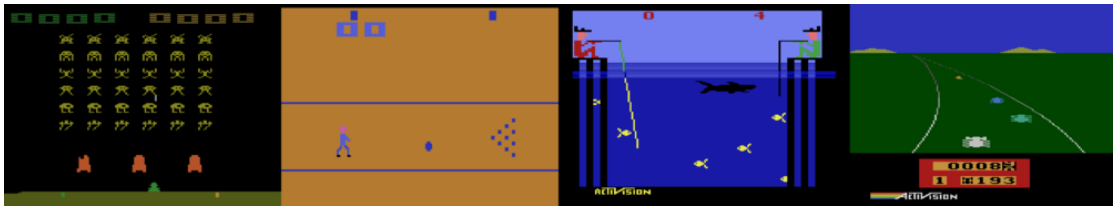

Figure 1.1: some of the games available in the Atari suite of environments.

update targets. DDQN performed better than DQN on the *Atari* games.

Schaul et al. (2016) [19] introduced the concept of *prioritized experience replay*, which replaced the uniform sampling from the replay memory of DQN with a sampling strategy weighted by the *TD errors* committed by the network. This improved the performance of both DQN and DDQN.

Wang et al. (2016) introduced a slightly different end-to-end *dueling architecture* [24], composed of two different deep estimators: one for the state-value function V and one for the *advantage function* $A : S \times A \to \mathbb{R}$ defined as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{1.1}$$

In this approach, the two networks share the same convolutional layers but use two separate dense layers. The two streams are then combined to estimate the optimal action-value function as[1]:

$$Q^\pi(s,a) = V^\pi(s) + (A^\pi(s,a) - \max_{a'} A^\pi(s,a')) \tag{1.2}$$

Several other extensions of the DQN algorithm have been proposed in recent years. Among these, we cite Osband et al. (2016) [16] who proposed a better exploration strategy; He et al. (2017) [10] added a constrained optimization approach called *optimality tightening* to propagate the reward faster and improve accuracy and convergence; Anschel et al. (2017) [1] improved the variance and instability of DQN by averaging previous Q estimates; Munos et al. (2016) [15] and Harutyunyan et al. (2016) [8] proposed to incorporate on-policy samples to the Q-learning target and seamlessly switch between off-policy and on-policy samples, which resulted in faster reward propagation and convergence.

[1] In the original paper, the authors explicitly indicate the dependence of the estimates on different parameters (e.g. $V^\pi(s,a;\phi,\alpha)$ where ϕ is the set of parameters of the convolutional layers and α of the dense layers). For coherence in the notation of this thesis, here we report the estimates computed by the network with the same notation as the estimated functions (i.e. the network which approximates V^π is indicated as V^π, and so on...)

1.2 Other approaches

1.2.1 Memory architectures

Graves et al. (2016) [7] proposed *Differentiable Neural Computer* (DNC), an architecture in which an ANN has access to an external memory structure, and learns to read and write data by gradient descent in a goal-oriented manner. This approach outperformed normal ANNs and DNC's precursor *Neural Turing Machine* [6] on a variety of query-answering and natural language processing tasks, and was used to solve a simple *moving block* puzzle with a form of reinforcement learning in which a sequence of instructions describing a goal is coupled to a reward function that evaluates whether the goal is satisfied (a set-up that resembles an animal training protocol with a symbolic task cue).

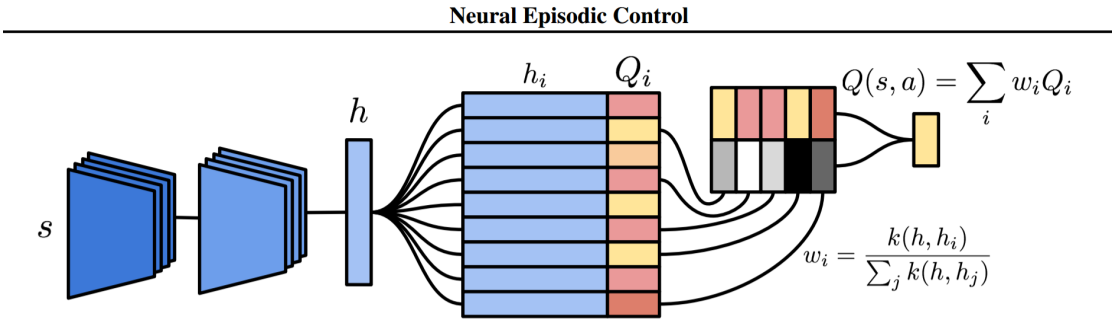

Figure 1.2: architecture of NEC.

Pritzel et al. (2017) [17] extended the concept of differentiable memory to DQN with *Neural Episodic Control* (NEC). In this apporach, the DRL agent consists of three components: a CNN which processes pixel images, a set of memory modules (one per action), and a dense ANN which converts read-outs from the action memories into action-values. The memory modules, called *differentiable neural dictionaries* (DNDs), are memory structures which resemble the dictionary data type found in computer programs. DNDs are used in NEC to associate the state embeddings computed by the CNN to a corresponding Q estimate, for each visited state: a read-out for a key consists in a weighted sum of the values in the DND, with weights given by normalized kernels between the lookup key and the corresponding key in memory (see Figure 1.2). DNDs are populated automatically by the algorithm without learning what to write (which greatly speeds up the training time with respect to DNC).

NEC outperformed every previous DRL approach on Atari games, by achieving better results using less training samples.

1.2.2 AlphaGo

Traditional board games like chess, checkers, Othello and Go are classical test benches for artificial intelligence. Since the set of rules which characterizes this type of games is fairly simple to represent in a program, the difficulty in solving these environments stems from the complexity of the state space. Among the cited games, Go was one of the last board games in which an algorithm had never beaten top human players, with its characteristic 19×19 board which allowed for approximately 250^{150} sequences of moves (number of legal moves per position elevated to the length of the game). Silver

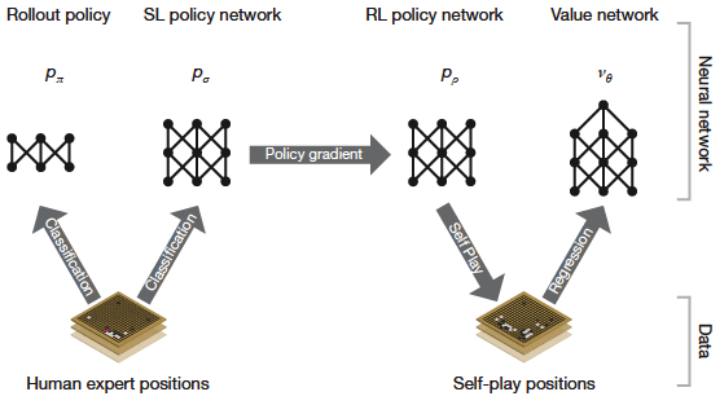

Figure 1.3: neural network training pipeline of AlphaGo.

et al. (2016) [20] introduced *AlphaGo*, a computer program based on DRL which won 5 games to 0 against the European Go champion in October 2015; soon after that, Alpha go defeated 18-time world champion Lee Sedol 4 games to 1 in March 2016, and world champion Ke Jie 3 to 0 in May 2017. After these results, Google DeepMind (the company behind AlphaGo) decided to retire the program from official competitions and released a dataset containing 50 self-play games [9].

AlhpaGo is a complex architecture which combines deep CNNs, reinforcement learning, and Monte Carlo Tree Search (MCTS) [3, 5]. The process is divided in two phases: a neural network training pipeline and MCTS. In the training pipeline, four different networks are trained: a *supervised learning* (SL) policy network trained to predict human moves; a *fast* policy network to rapidly sample actions during MC rollouts; a *reinforcement learning* policy network that improves the SL network by optimizing the final outcome of games of self-play; a *value* network that predicts the winner of games. Finally, the policy and value networks are combined in an MCTS algorithm that selects actions with a lookahead search, by building a partial search tree using the estimates computed with each network.

1.2.3 Asynchronous Advantage Actor-Critic

Actor-critic algorithms [21] are TD methods that have a separate memory structure to explicitly represent the policy independent of the value function. The policy structure is known as the actor, because it is used to select actions, and the estimated value function is known as the critic, because it criticizes the actions made by the actor. Mnih et al.

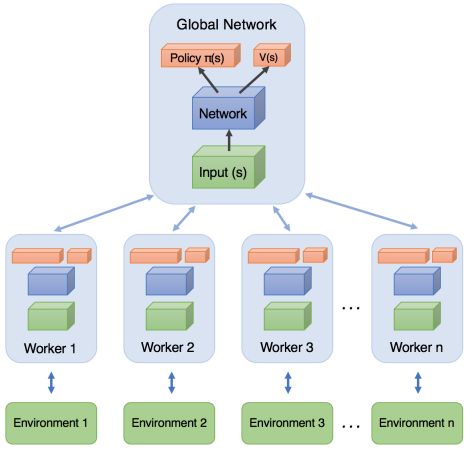

Figure 1.4: the asynchronous architecture of A3C.

(2016) [13] presented a deep variation of the actor-critic algorithm, called *Asynchronous Advantage Actor-Critic* (A3C). In this approach, different instances of actor-critic pairs are run in parallel to obtain a lower-variance estimate of the value function, without the need of a replay memory to stabilize training. Each *worker* consists in a deep CNN with a unique convolutional section and two separate dense networks on top, one for the value function and one for the policy.

This asynchronous methodology was applied to other classical RL algorithms in the same paper; we only report the actor-critic variant as it was the best performing, with substantially shorter training times and comparable performance to DQN and its variations described in Section 1.1.

1.3 Related work

In this section we give an overview of previous work with similarities to our approach.

Lange and Riedmiller (2010) [11] proposed the *Deep Fitted Q-iteration* (DFQ) algorithm, a batch RL method which used deep dense autoencoders to extract a state representation from pixel images. In this algorithm, a training set of $< s, a, r, s' >$ transitions is collected with a random exploration strategy, where s, s' are pixel images of two consecutive states. The samples are then used to train a dense autoencoder with

two neurons in the innermost layer, which is in turn used to encode all states in the training set. This encoded dataset is then passed as input to FQI, which produces an estimate for the Q function using a kernel based approximator. A new policy is then computed from the estimated Q and the encoder, and the process is repeated using the new policy until the obtained Q is considered satisfactory.

FE: Deep Auto-Encoder Neural Networks in Reinforcement Learning Predict dynamics: Faster Reinforcement Learning After Pretraining Deep Networks to Predict State Dynamics

Bibliography

[1] Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning.

[2] Dimitri P. Bertsekas and John N. Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.

[3] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[4] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[5] Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michele Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The grand challenge of computer go: Monte carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, 2012.

[6] Alex Graves and Greg Wayne. Neural turing machines. 2014.

[7] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[8] A. Harutyunyan, M. G. Bellemare, T. Stepleton, and R. Munos. Q (λ) with off-policy corrections. In *International Conference on Algorithmic Learning Theory*, pages 305–320. Springer, 2016.

[9] D. Hassabis and D. Silver. Alphago's next move (deepmind.com/blog/alphagos-next-move/), 2017.

[10] F. S. He, Y. Liu, A. G. Schwing, and J. Peng. Learning to play in a day: Faster deep reinforcement learning by optimality tightening. In *International Conference on Learning Representations (ICLR)*, 2017.

[11] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE, 2010.

[12] Long-H Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3/4):69–97, 1992.

[13] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–33, 2015.

[15] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

[16] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pages 4026–4034, 2016.

[17] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. 2017.

[18] Gavin A. Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering, 1994.

[19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2016.

[20] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[22] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[23] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[24] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Duel- ing network architectures for deep reinforcement learning. In *International Conference on Machine Learning (ICML).*, 2016.