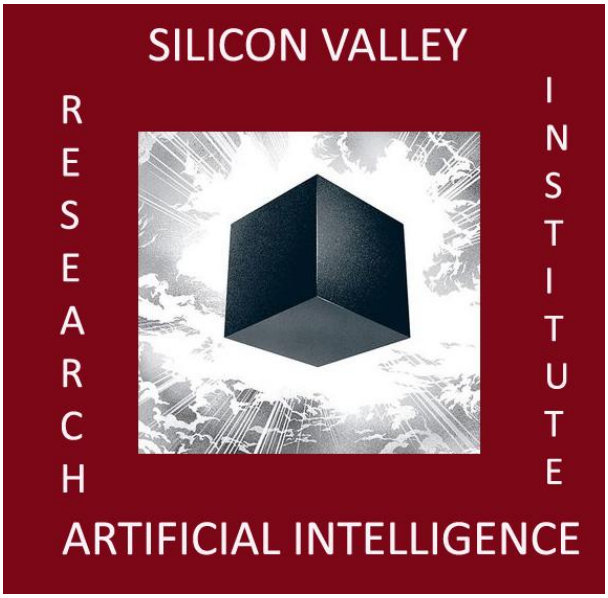

Deep Reinforcement Learning: a brief introduction

The game of go/weichi had been a favorite field of research since the birth of deep learning. In 2006 Remi Coulom in France applied Ulam's old Monte Carlo method to the problem of searching a tree data structure (one of the most common problems in computational math, but particularly difficult when the data are game moves), and thus obtained the Monte Carlo Tree Search algorithm, and then applied it to go/weichi ("Bandit Based Monte-Carlo Planning", 2006). At the same time Levente Kocsis and Csaba Szepesvari developed the UCT algorithm (Upper Confidence Bounds for Trees algorithm), an application of the bandit algorithm to the Monte Carlo search, the bandit algorithm being a problem in probability theory first studied by Herbert Robbins in 1952 at the Institute for Advanced Study ("Some aspects of the sequential design of experiments", 1952).

These algorithms dramatically improved the chances by machines to beat go masters: in 2009 Fuego Go (developed at the University of Alberta) beat Zhou Junxun, in 2010 MogoTW (developed by a French-Taiwanese team) beat Catalin Taranu, in 2012 Tencho no Igo/ Zen (developed by Yoji Ojima) beat Takemiya Masaki, in 2013 Crazy Stone (by Remi Coulom) beat Yoshio Ishida, and in 2016 AlphaGo (developed by Google's DeepMind) beat Lee Sedol. DeepMind's victory was widely advertised. DeepMind used a slightly modified Monte Carlo algorithm but, more importantly, AlphaGo taught itself by playing against itself. AlphaGo's neural network was trained with 150,000 games played by go/weichi masters. AlphaGo had played 200 million games by January 2017 when it briefly debuted online disguised under the moniker Master; and in May 2017 it beat the world's master Ke Jie.

DeepMind had previously combined convolutional networks with reinforcement learning to train a neural network to play videogames: in 2013 Volodymyr Mnih and others had trained convolutional networks with a variant of Q-learning, the "asynchronous actor-critic" algorithm or

A3C, in order to improve the policy function ("Playing Atari with Deep Reinforcement Learning", 2013). These Deep Q-Network (DQN) was the first in a progression of deep reinforcement learning methods developed by DeepMind, leading to AlphaGo and then AlphaZero.

Deep Q-learning suffers from some inherent limitations and requires a lot of computational power, which translates not only in cost but also in slow response time. DeepMind had rediscovered Lin's "experience replay" of 1992 as a remedy for the Atari-playing network. Next, DeepMind rediscovered the policy gradient methods for reinforcement learning (although different from the likelihood-ratio method of REINFORCE): for example, David Silver's "deterministic policy gradient" of 2014 and Nicolas Heess' "stochastic value gradient" of 2015. Volodymyr Mnih's team worked on asynchronous gradient-descent algorithms that run multiple agents in parallel instead of using "experience replay". In 2015 Arun Nair and others designed the General Reinforcement Learning Architecture (Gorila) that greatly improved the performance on those Atari games by performing that kind of asynchronous training of reinforcement learning agents (100 separate actor-learner processes).

In 2016 the DeepMind team also adapted "deep Q-learning" to the more general domain of continuous action ("Continuous Control with Deep Reinforcement Learning", 2016).

Ironically, few people noticed that in September 2015 Matthew Lai unveiled an open-source chess engine called Giraffe that used deep reinforcement learning to teach itself how to play chess (at international master level) in 72 hours. It was designed by just one person and it ran on a the humble computer of his department at Imperial College London. (Lai was hired by Google DeepMind in January 2016, two months before AlphaGo's exploit against the Go master).

In 2016 Toyota demonstrated a self-teaching car, another application of deep reinforcement learning like AlphaGo: a number of cars are left to randomly roam the territory with the only rule that they have to avoid accidents. After a while, the cars learn how to drive properly in the streets.

Poker was another game targeted by A.I. scientists, so much so that the University of Alberta even set up a Computer Poker Research Group. Here in 2007 Michael Bowling developed an algorithm called Counterfactual Regret Minimization or CFR ("Regret Minimization in Games with Incomplete Information", 2007), based on the "regret matching" algorithm invented in 2000 by Sergiu Hart and Andreu Mas-Colell at the Einstein Institute of Mathematics in Israel ("A Simple Adaptive Procedure Leading to Correlated Equilibrium", 2000). These are techniques of self-playing: given some rules describing a game, the algorithm plays against itself and develops its own strategy for playing the game better and better. It's yet another form of reinforcement learning, except that in this case reinforcement learning is used to devise the strategy from scratch, not to learn the strategy used by humans. The goal of CFR and its numerous variants is to approximate solutions for imperfect information games such as poker. CFR variants became the algorithms of choice for "poker bots" used in computer poker competitions. In 2015 Bowling's team developed Cepheus and Tuomas Sandholm at Carnegie Mellon University developed Claudico, that played the professionals at a Pittsburgh casino (Pennsylvania). Claudico lost but in 2017 Libratus, created by the same group, won. Libratus employed a new

algorithm, called CFR+, introduced in 2014 by Finnish hacker Oskari Tammelin ("Solving Large Imperfect Information Games Using CFR+, 2014) that learns much faster compared with previous versions of CFR. However, the setting was absolutely unnatural, in particular to rule out card luck. It is safe to state that no human players had ever played poker in such a setting before. But it was telling that the machine started winning when the number of players was reduced and the duration of the tournament was extended: more players in a shorter time beat Claudico, but fewer players over a longer time lost to Libratus.

Deep reinforcement learning has three main problems: it requires a lot of data, which is unnatural (animals can learn something even if they saw it only once or twice), it fails all too easily in situations that differ just slightly from the situations for which the system has been trained, and its internal workings are largely inscrutable, which makes us uncomfortable to use them on mission-critical projects.

A different strand from DeepMind's A3C harkens back to the "relational Markov decision process", introduced by Carlos Guestrin ("Planning under Uncertainty in Complex Structured Environments", 2003); and to the Object-Oriented Markov Decision Process, introduced by Micheal Littman's student Carlos Diuk at Rutgers University ("An Object-Oriented Representation for Efficient Reinforcement Learning", 2008). The relational Markov decision process blends probabilistic and relational representations with the express goal of planning action in the real world. Marta Garnelo's "deep symbolic reinforcement" at Imperial College London ("Towards Deep Symbolic Reinforcement Learning",2016) and Dileep George's "schema networks" to transfer experience from one situation to other similar situations (Schema Networks", 2017) built upon these ideas, basically wedding first-order logic representation with deep reinforcement learning.

Yet another route led to Trust Region Policy Optimization (TRPO), introduced in 2015 by Pieter Abbeel's student John Schulman at UC Berkeley as an alternative to DeepMind's policy-gradient methods (at least for continuous control tasks). DeepMind itself contributed an alternative to policy-gradient methods: ACER developed by Ziyu Wang and others of Nando de Freitas' team ("Sample Efficient Actor-Critic with Experience Replay", 2016), a variant of the Retrace method developed by their colleague Remi Munos ("Safe and Efficient Off-policy Reinforcement", 2016). John Schulman himself improved TRPO with PPO, "proximal policy optimization", released by OpenAI in 2017 ("Proximal Policy Optimization Algorithms", 2017). DQN, A3C, TRPO, ACER and PPO were just the tip of the iceberg: algorithms for optimization of reinforcement learning multipled and evolved rapidly after the success of these techniques in learning to play games.

If 2016 had been the year of reinforcement learning, by 2017 it was becoming an easy target for all sorts of attacks. First of all, several studies showed that other forms of machine learning could replicate the feats of DeepMind: Jeff Clune's team at Uber AI Labs ("Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning", 2017), Tim Salimans at OpenAI ("Evolution Strategies as a Scalable Alternative to Reinforcement Learning", 2017) as well as Ben Recht's students Horia Mania and Aurelia Guy at UC Berkeley ("Simple Random Search Provides a Competitive Approach to Reinforcement Learning", 2018) showed that much simpler genetic algorithms

could do as well on most tasks. Secondly, Sham Kakade's student Aravind Rajeswaran at the University of Washington showed that multi-layered neural networks are an unnecessary complication for robotic tasks of continuous control such as those of the OpenAI gym environment ("Towards Generalization and Simplicity in Continuous Control", 2017). For example, Nicolas Heess and others at Deep Mind used deep reinforcement learning to train a robot for walking, running, jumping and turning ("Emergence of Locomotion Behaviours in Rich Environments", 2017). The training required 64 CPUs running for over 100 hours. The video was impressive, but similar results had been obtained five years earlier by Emanuel Todorov's students Yuval Tassa and Tom Erez (both later hired by DeepMind) at the University of Washington, running on much smaller and slower hardware using not reinforcement learning but a humbler method, online trajectory optimization, aka model predictive control, on their physics simulator MuJoCo ("Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization", 2012).

The original reason to develop the complex algorithms of A.I. was not that it was impossible to play chess or weichi/go with simpler algorithms but rather that it was plain silly to do so: it would have required colossal amounts of computation on that slow and expensive hardware. It seems like A.I. in the age of AlphaGo reached a point where its complex algorithms do require a colossal amount of computation; which seems to contradict the whole point of doing A.I. Nobody ever proved formally that simple algorithms cannot achieve superhuman performance at chess or go/weichi or any other deterministic game. The availability of computational power reduces the raison d' être of A.I.'s sophisticated algorithms. What's the point of developing multilayer networks if simple architectures can do the same job? The only reason would be to speed up things, or run on cheaper hardware, but instead the opposite is happening: DeepMind and OpenAI have to throw more and more computational power at their algorithms. It was not surprising that, after the initial enthusiasm, A.I. scientists started revisiting the old simple algorithms that were dismissed 20 decades earlier. Some of them might indeed prove that A.I. unnecessarily complicated things because we forgot that the stumbling block was not a faulty theory but only slow hardware.

One of the biggest problems for reinforcement learning is that it requires a reward function: the reward function really represents what we want the system to learn. The problem is that designing the correct reward function is extremely difficult in the real world. If the reward is not correct, reinforcement learning acts like an uncontrollable chain reaction.

Another unsolved problem in reinforcement learning is the problem of balancing exploration and exploitation. This is a general problem of mathematical optimization called "the multi-armed bandit problem". The classic example is the dinner: you can have dinner at your favorite restaurant, where you will almost certainly get a good meal, or try a new restaurant, with a chance that you'll discover a new favorite. This doesn't sound like a big deal. But apply the same reasoning to medical care and you can probably appreciate why it is a big deal. Exploitation consists in making the best decision that maximizes immediate return. Exploration consists in investing in gathering more knowledge about the environment, which in the long run might yield a higher return. Exploration is extremely expensive in reinforcement learning so it tends to be neglected. But in some cases there is a huge price to pay for it. That's why DQN could play Atari games so well but not Montezuma Revenge, a game that requires more exploration.

Historically speaking, we have learned that the classic A.I. problems can be divided in categories based on some factors, and it turns out that Go belong to the simplest category of A.I. problems. Problems that are deterministic, fully observable and known, discrete (a limited number of possible states) and static (the rules don't change) are actually "easy" to solve when we can afford to throw virtually infinite computational power at them. There is nothing in their formulation that makes it impossible to solve them: it is "difficult" to solve them simply because the number of possible solutions is very big. In the old days of slow, expensive hardware one would program the machine to look for the solution by simulating human intuition. In the age of cheap, fast hardware, one programs the machine to use brute force and simply try all possible combinations until one turns out to be the correct solution. The game of weichi/go is one such problem: "difficult" because of the many many many possible moves, but "easy" for a machine that can try all of them.

The next step after weichi/go was the multiplayer videogame Dota 2 of 2013, played daily by over one million people. In 2018 OpenAI announced a bot, OpenAI Five, that, like AlphaGo Zero, learned the game from scratch by self-play without any knowledge of how humans play it. This problem is neither fully observable/known nor discrete nor static. OpenAI Five trains by playing 180 years worth of games against itself every day, running on 256 GPUs and 128,000 CPU cores, using a separate LSTM for each hero. OpenAI Five uses proximal policy optimization. In 2018 OpenAi used the same algorithm to teach a robotic hand, Dactyl, to manipulate cubes.