Playing Atari with Deep Reinforcement Learning: a Review

Anastasia Prisacaru and Volha Taliaronak

August 7, 2020

1 Introduction

Games have always been an important part of Artificial Intelligence (AI) by enabling the researchers to measure the capacity of AI algorithms and compare it against humans' capabilities or other approaches. To quote Julian Togelius - the professor at the Department of Computer Science and Engineering at the New York University [Gershgorn, Dave, 2018], "Games are such a good place for AI to learn because they're an analogue of the real world, but with an objective".
One of the most popular methods for teaching bots to play games, which is also used by OpenAI and the AI research lab, predominantly founded by Elon Musk and Sam Altman, is called reinforcement learning (RL). Due to its main concepts such as agent, environment, actions and reward, RL provides a clear language to state general AI problems. For instance, [Mnih et al., 2013] presented a novel approach of training a deep learning model, which can successfully learn to play different Atari games, by only receiving raw pixels and reward as input, with no change of architecture or hyperparameters. The model trained with the so-called Deep Q-Network approach
The outline of this report is as follows: First, we briefly describe the historical background of RL in Section 2. Next, we discuss the five techniques the DQN approach is based on in Section 3. We give an overview about the preprocessing and training of the DQN approach in Section 4. Section 5 provides a brief summary about the results of the experiments and puts DQN into perspective with other similar approaches. In Section 6, we discuss the implementation of the DQN approach on the example of the Cart Pole game. Finally, we conclude this work in Section 7.

2 Background

The history of reinforcement learning has two main threads, both long and rich. One thread concerns learning by trial and error. It took its start in the psychology of animal learning (Throndike, 1911) and led to the revival of reinforcement learning in the early 1980s. Trial-and-error learning combines two important aspects: selectional – trying alternatives and selecting among them by comparing their consequences – and associative – meaning that the alternatives found by selection are associated with particular situations. In 1954 Minsky discussed computational models of reinforcement learning and described his construction of an analog machine composed of components (SNARCs). Also influential was the work of Donald Michie. In 1961 and 1963, he described a simple trial-and-error learning system for learning how to play tic-tac-toe called MENACE, which consistently emphasized the role of trial and error and learning as essential aspects of artificial intelligence. In 1986 John Holland introduced classifier systems, true reinforcement learning systems including association and value functions. A key component of his classifier systems was always a genetic algorithm, whose role was to evolve useful representations. Other studies showed how reinforcement learning could address important problems in neural network learning, in particular, how it could produce learning algorithms for multilayer networks (Barto, Anderson, and Sutton, 1982; Barto and Anderson, 1985; Barto and Anandan, 1985; Barto, 1985, 1986; Barto and Jordan, 1987) [Sutton and Barto, 2018].
The other thread concerns the problem of optimal control and its solution using value functions and dynamic programming and does not involve learning. This class of methods uses the concepts of dynamical system's state and of a value function ("optimal return function") to define a functional equation, today known as Bellman equation. Bellman also introduced in 1957 the discrete stochastic version of the optimal control problem known as Markovian decision processes (MDPs) [Sutton and Barto, 2018]. The solution methods of optimal control (e.g. dynamic programming) require complete knowledge of the system to be controlled. Nevertheless these methods are also considered to be RL methods, because like other RL methods, they gradually reach the correct answer through successive approximations [Sutton and Barto, 2018]. Both Bellman equation and MDPs are considered to be essential elements underlying the theory and algorithms of modern RL.
The exceptions from these two threads formed a third one, less distinct thread, concerning temporal-difference methods. These methods are distinctive in being driven by the difference between temporally successive

estimates of the same quality (e.g. of the probability of winning in the tic-tac-toe). The origins of temporal-difference learning are in part in animal learning psychology, in particular, in the notion of secondary reinforcers. A secondary reinforcer is a stimulus that has been paired with a primary reinforcer such as food or pain and, as a result, has come to take on similar reinforcing properties [Sutton and Barto, 2018]. Arthur Samuel (1959) was the first to propose and implement a learning method that included temporal-difference ideas, as part of his celebrated checkers-playing program. In 1972, Klopf brought trial-and-error learning together with an important component of temporal-difference learning. Klopf was interested in principles that would scale to learning in large systems, and thus was intrigued by notions of local reinforcement, whereby subcomponents of an overall learning system could reinforce one another. He developed the idea of "generalized reinforcement", whereby every component (nominally, every neuron) views all of its inputs in reinforcement terms: excitatory inputs as rewards and inhibitory inputs as punishments. Sutton (1978a, 1978b, 1978c) developed Klopf's ideas further.

Deep neural networks has been prevailing in RL in the last several years, in games, robotics, natural language processing, etc. Deep reinforcement learning, a combination of deep neural networks and reinforcement learning, is benefiting from big data, powerful computation, new algorithmic techniques, mature software packages and architectures, and strong financial support [Li, 2017]. Deep Q-network [Mnih et al., 2013], guided policy search [Levine et al., 2015], unsupervised reinforcement and auxiliary learning [Jaderberg et al., 2016] are just several examples of the long list of exciting achievements resulting from combination of RL techniques with the deep learning approach. Deep learning and reinforcement learning, being selected as one of the MIT Technology Review 10 Breakthrough Technologies in 2013 and 2017 respectively, will play their crucial role in achieving artificial general intelligence [Li, 2017]. David Silver, the major contributor of AlphaGo [Silver et al., 2017], even came up with the following formula: artificial intelligence = reinforcement learning + deep learning [Silver, David, 2016].

In this work, we discuss a paper where one of the breakthroughs of deep RL was originally introduced, namely the Deep Q-Network (DQN) approach [Mnih et al., 2013].

3 DQN Approach

In their paper [Mnih et al., 2013] address three challenges of reinforcement learning and propose a solution to each of them. First, the input data from which the agent learns, is often sparse, noisy and delayed. For instance, the delay between actions and resulting rewards, can reach up to thousands of time steps, which makes the learning process particularly challenging. The authors propose a solution, by using a CNN trained with a variant of Q-learning. However, when combining reinforcement learning with deep neural networks, another two challenges arise. First, in contrast with deep learning, where the input data sequences are assumed to be independent from each other, reinforcement learning algorithms encounter sequences of highly correlated states. Furthermore, deep learning algorithms assume a stationary distribution; whereas in reinforcement learning, the data distribution changes as the agent evolves and learns new behaviours. These two challenges can be tackled by randomly sampling past sequences of experiences and smoothing the training distribution over many past behaviours. We will elaborate more on these challenges and according solutions in the following subsections, where we describe the five base techniques of the Deep-Q-Network approach.

3.1 Experience Replay Mechanism

The experience replay technique is used for smoothing high correlation of data and dealing with the non-stationary distribution, by saving the agent's experiences at each time-step and pooling them over many episodes into a replay memory. At the beginning, a buffer B of the length N is initialized, where N is the number of experience sequences which will be stored. At each time step t the agent performs an action and it's experience is saved as a tuple e_t as shown bellow:

$$e_t = (s_t, a_t, r_t, s_{t+1}),\ B = [e_1, e_2, ..., e_N]$$

where s_t is the current state, a_t - the chosen action, r_t - the received reward and s_{t+1} - the new state. Once the replay memory buffer is full, it is updated based on FIFO strategy. In [Mnih et al., 2013] was used a replay memory buffer of size 1,000,000, thereby making the data extracted from the replay memory much more meaningful over time. During the training process, the past N experiences are sampled randomly and uniformly from the replay memory, therefore breaking the correlations between them and reducing the

variance of the updates. Furthermore, each experience is potentially used for updating the weights, which allows for greater data efficiency.

3.2 Epsilon-Greedy Strategy

At the beginning of the training process, the learning agent has no knowledge about the environment and no policy to follow, which means that it has to explore the environment by choosing random moves from a given set of actions. Once the agent has started to get positive rewards, it will start saving the state-action pairs with the biggest reward, creating it's policy. During the rest of training process, the agent either selects a random action (exploratory step) or chooses the action with the biggest future reward according to its policy (exploitative step). Trade-off between exploration and exploitation is commonly negotiated by using the ϵ-greedy strategy. The ϵ value signifies the probability with which the agent performs a random action or chooses the best action according to its policy with the probability $1 - \epsilon$, as shown bellow:

$$a_t = \begin{cases} a_t^* & \text{, with probability } 1 - \epsilon \\ \text{random action} & \text{, with probability } \epsilon \end{cases}$$

The epsilon-greedy strategy is used both during the training and the playing process, but usually with different ϵ values. During the training of the agent, the authors of the paper decreased the value of ϵ linearly from 1 to 0.1 over the first million frames and kept its value fixed at 0.1 thereafter. Thus the agent was acting randomly the first 1 million frames and the rest 9 million frames it was choosing the best action from the policy 90% of the time. After the training process was over, the agent was playing the games with the ϵ value set to 0.05, in order to avoid the agent getting stuck in an unknown state, but the policy will not be changed even if the random action gave a positive reward.

3.3 Target Network

For training the agent, the authors use a convolutional neural network (CNN) trained with a variant of the Q-learning algorithm, with stochastic gradient descent to update the weights. The CNN has three hidden layers with an action-value function *Q(s,a)*. The Q-function is based on the Bellman equation and is defined as the maximum expected return achievable, by following any strategy after seeing some sequence s and then taking

some action a:

$$Q(s, a) = max_{\pi}\mathbb{E}[R_t|s_t = s, a_t = a, \pi]$$

where π is the policy, which maps sequences to actions (or distributions over actions). The loss function is also based on the Bellman equation combined with a non-linear function approximator (neural network with weights). The first layer convolves 16 8x8 filters with stride 4. The second layer convolves 32 4x4 filters with stride 2. After every convolution layer a rectifier nonlinearity function (RELU) is applied. The output layer is a fully-connected linear layer with a single output for each valid action (varied between 4 and 18).

3.4 Clipping Rewards

In order to use the same neural network agent to play multiple games without changing the learning rate, the authors use the clipping rewards technique. This is necessary, because each game has a different score system, so the agent's reward will be incremented by 1 in case of a positive score, decreased by 1 if it's negative, or 0 - otherwise. However, this technique has the downside that the agent is not able to distinguish the magnitude of the reward.

3.5 Skipping Frames

Calculating an action of the agent for each game frame can be computationally expensive, that's why the agent is only observing every kth frame and performing the same action over the following k frames. The authors set the value of k to 4 for all games except the Space Invaders, where $k = 3$ was used, because some objects were not visible when skipping 4 frames.

4 Preprocessing and Training

The authors used the Atari games provided by Arcade Learning Environment, which was designed for evaluating the development of general, domain-independent AI technology [Bellemare et al., 2013]. The following Atari games were chosen for training the agent: Pong, Breakout, Space invaders, Seaquest, Beam Rider, Q*bert and Enduro. Each game frame is preprocessed by converting it to gray-scale, down-sampling it from 210×160 to 110×84 pixel and cropping an 84x84 image of the region of the screen, that roughly captures the playing area. In order to improve the learning process, the authors consider sequences of actions and observations and learn game

strategies that depend upon these sequences, otherwise it will be impossible to learn new strategies by only observing images of the current state. Therefore, the input to the neural network consists of 4 84 $\times$ 84 images. The resulting frame goes through three convolution layers, where the final output layer with a single output for each valid action.
The authors have trained the agent on each game separately for 10 million frames with no adjustment of the architecture or hyperparameters (the only adjustment was made for Space Invaders when changing the number of skipped frames to 3). They have set the replay memory buffer to 1 million frames, because the observations of the environment get more meaningful over time.

5 Evaluation and Results

The model trained with the DQN approach was tested on 7 Atari games and it achieved a better performance than an expert human on 3 of them and outperformed some of the previous approaches on all games. Some of the previous works which aimed to train a reinforcement learning agent to play Atari games are Contingency [Bellemare et al., 2012b] and Tug-of-War Hashing [Bellemare et al., 2012a]. Both approaches combine the State–action–reward–state–action algorithm (SARSA) and the ϵ-greedy strategy. In the first approach [Bellemare et al., 2012b], the authors aimed to detect contingent regions, or in other words, the parts of a game frame whose immediate future value depends on the the agent's actions. In contrast to the DQN approach, where the agent was trained and tested for each game separately, the contingency model was trained on 5 Atari games and tested on 46 different ones. However, in the contingency approach, each color was treated as a separate channel, whereas in the DQN approach the model learnt from raw black and white frames. Taking in account the fact that most Atari games have different colors for different objects, processing each color separately, makes it easier for the model to detect objects in its environment. Similarly to Contingency, in the Tug-of-War Hashing approach [Bellemare et al., 2012a], each color was treated as a separate color channel and the model was also trained on 5 games and tested on 50 other ones. The authors used a hashing method based on the Tug-of-War sketch, for approximating linear policies on several different feature sets hand-engineered for the Atari task. This hashing method has the computational advantage of saving each feature exactly once. Both methods described above were compared with the DQN approach, by running a greedy policy with $\epsilon = 0.05$ for

a fixed number of steps. All three methods performed better than the agent which makes only random moves and as stated before, the DQN approach outperformed them in all 7 games.

6 Implementation

In order to reproduce the results presented in the paper significant resources would be required. The computational power of a conventional laptop wouldn't be enough to train an agent to play Atari games. Therefore, we decided to choose a game with less complexity, such as Cartpole. The cartpole game, also known as the inverted pendulum, is essentially a pole attached by an un-actuated joint to a cart with the center of gravity above its pivot point, which can move either left or right along a frictionless track. The pendulum starts upright and the goal of the game is to keep the pole balanced as long as possible. The game is over when the pole is more than 15 degrees from vertical or if the cart has moved more than 3 units from the center.
As a base for our code, we used the implementation provided in a tutorial about the DQN approach applied for training an agent to play the cartpole game [Adam Paszke, 2019]. The author used the environment "Cartpole-v0" provided by the OpenAI Gym, which is a toolkit for developing and comparing reinforcement learning algorithms. As a machine learning library for training the neural networks, the PyTorch library is used. Due to the reduced complexity of the game, the buffer of the replay memory which stores the experiences of the agent is set to 10.000. The epsilon value starts at 0.9 and it is exponentially reduced to 0.05 during training. Similarly to the paper, the model is a convolution neural network with 3 hidden layers and with ReLu used as activation function. We trained the model for 500 epochs and by the end of the training, the agent reached the score of over 200 points. However, when training the agent for a longer time, the model seemed to overfit and to deliver worse scores.

7 Conclusion

In this short overview we discussed [Mnih et al., 2013], where a novel approach of training a reinforcement learning agent by using deep neural networks was introduced. We started with the history of reinforcement learning and discussed the importance of the junction where RL meets deep neural networks. Next, we described the core techniques of the DQN approach

presented in the paper in detail. We elaborated on how the DQN approach overcomes well-known challenges of RL combined with deep neural networks, such as noisy and delayed input data, highly correlated states and non-stationary data distribution. The model trained with the DQN approach received as input raw pixel data and the reward. As a result, the agent has learnt complex control policies for 7 different Atari games with no change of architecture or hyperparameters. The model outperformed previous RL approaches on all games and even performed better than an expert human player on three of them. We evaluated the DQN approach on the Cartpole game and received promising results after a fairly short training period.

References

[Adam Paszke, 2019] Adam Paszke (2019). Reinforcement learning (dqn) tutorial. `https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html`. [Accessed: 15-March-2020].

[Bellemare et al., 2012a] Bellemare, M., Veness, J., and Bowling, M. (2012a). Sketch-based linear value function approximation. In *Advances in Neural Information Processing Systems*, pages 2213–2221.

[Bellemare et al., 2013] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

[Bellemare et al., 2012b] Bellemare, M. G., Veness, J., and Bowling, M. (2012b). Investigating contingency awareness using atari 2600 games. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

[Gershgorn, Dave, 2018] Gershgorn, Dave (2018). Why are ai researchers so obsessed with games?. `https://qz.com/1348177/why-are-ai-researchers-so-obsessed-with-games/`. [Accessed: 01-March-2020].

[Jaderberg et al., 2016] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks.

[Levine et al., 2015] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2015). End-to-end training of deep visuomotor policies.

[Li, 2017] Li, Y. (2017). Deep reinforcement learning: An overview.

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

[Silver et al., 2017] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

[Silver, David, 2016] Silver, David (2016). Tutorial: Deep reinforcement learning. `https://arxiv.org/pdf/1701.07274.pdf`. [Accessed: 08-March-2020].

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.