Playing Atari games with a deep Q-network

Niklas Smedemark-Margulies *
Khoury College of Computer Science
Northeastern University
Boston, MA, 02115
smedemark-margulie.n@northeastern.edu

Ondrej Biza *
Khoury College of Computer Science
Northeastern University
Boston, MA, 02115
biza.o@northeastern.edu

1 Reinforcement Learning

Reinforcement Learning models an agent's interaction with an environment as a sequential decision making process. The agent observes the state of the environment (e.g. the current screen of a video game, joint configurations of a robot, etc.) and chooses an action to execute. Then, it receives an arbitrary scalar reward, and continues acting based on the next state of the environment. The aim is to maximize the discounted sum of rewards the agent receives over its life.

We use a formalism called the Markov Decision Process (MDP, [1]) to represent the problem: $\mathcal{M} = \langle S, A, P, R, \rho_0, \gamma \rangle$. S and A are the sets of all possible states and actions respectively. $P : S \times A \times S \to [0, 1]$ is a transition function that expresses the probability of transitioning from the current state to some next state given a selected action. The reward function $R : S \times A \to \mathbb{R}$ is the expected reward for state-action pairs. ρ_0 is a distribution over the starting states and γ is a discount factor.

The Markov assumption underlying MDPs says that the current state of the environment contains all information we need to know to make decisions. In other words, we do not need to keep track of the history of visited states and selected actions. Hence, we can represent the behavior of an agent as a policy $\pi : S \times A \to [0, 1]$, a probability distribution over actions in each state.

To evaluate the quality of a policy, we can calculate its return, i.e. the sum of its expected discounted rewards: Return $= \sum_{t=0}^{\infty} \gamma^t r_t$, where $(r_t)_{t=0}^{\infty}$ is a sequence of rewards obtained by starting in state $s_0 \sim \rho_0$ and following the policy π. A related quantity of interest is the **state-action value function** $Q_\pi(s, a)$, describing the expected discounted return after starting in state s and executing action a. This function follows a recursive relation called a Bellman equation.

$$Q_\pi(s, a) = \underbrace{R(s, a)}_{\text{Immediate reward}} + \gamma \, \mathbb{E}_{a' \sim \pi, s' \sim P(s'|a,s)} \underbrace{[Q_\pi(s', a')]}_{\text{Future reward}} \tag{1}$$

The key elements of an agent's behavior can be broadly grouped into two categories, and RL methods typically focus on one or a combination of these categories. In *model-based* methods, the agent seeks to learn a predictive model of future state of the environment $p(s_{t+1}|s_t, a_t)$ and the reward function $R(s_t, a_t)$. Conversely, a *model-free* reinforcement learning agent learns either a policy or a value function, without explicitly modeling the dynamics of the environment.

In *policy-based* methods, the agent seeks to learn the parameters θ of a policy $\pi(a|s, \theta)$ mapping the current state to a distribution over actions. Finally, in *value-based* methods, the agent seeks to model its expected future rewards; in particular, in *Q-learning* the agent tries to predict the value of actions in the current state $Q^*(a, s)$ for an optimal policy π^*.

*equal contribution

Northeastern CS 7150 course project. Summer 2020.

1.1 Q-learning

Q-learning aims to learn the state-action value function $Q^*(a, s)$ for an optimal policy π^*. Equation 1 leads us naturally to an update rule, where we must learn a state-action value function that satisfies our recursion.

$$Q_{\text{new}}(s_t, a_t) = \underbrace{Q(s_t, a_t)}_{\text{current estimate}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Bigg[\underbrace{\Big(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \Big)}_{\text{estimated optimal reward}} - Q(s_t, a_t) \Bigg] \tag{2}$$

We take the maximum over the action values of the next state s_{t+1} because an optimal policy π^* always select the action with the highest value. This leads the optimal policy to always be deterministic, which is valid as long as the state of the environment is fully observable (i.e. the Markov assumption holds).

In classical approaches, the update rule is usually applied online for each transition tuple (s_t, a_t, r_t, s_{t+1}) collected by an agent. Since a reinforcement learning agent must trade-off exploration (collecting interesting experiences) with exploitation (collecting rewards), Q-learning is often used in conjunction with an ϵ-greedy policy. With probability ϵ the policy picks a random action; otherwise, it selects the best action according to the current Q value estimate.

For toy problems with a small enough action space and environment state space, we can explicitly store a table containing the Q value of every action in every state. Linear function approximators can be used in environments with too many states (e.g. states represented as images). Both of these methods are guaranteed to converge to the true optimal Q function under some assumptions [2], but they also lack the capacity to learn complex non-linear problems, which makes their application limited.

2 DQN

2.1 Key Innovations

The key innovation in a deep Q-network [3] is to represent the state-action value function $Q(s, a; \phi)$ by a deep neural network with parameters ϕ. Deep learning usually requires a large dataset of i.i.d. samples; both of these assumptions are broken in reinforcement learning. We require the agent to learn from experience that is collected while it interact with the environment; experiences collected sequentially will be highly correlated, and the number of available experiences may be limited.

A deep Q-network combats the problem of limited and correlated data by keeping a growing experience buffer storing transitions (s_t, a_t, r_t, s_{t+1}) that the agent has experienced during training. Mini-batches sampled from the experience buffer represent many different regions of the agent's state space visited over many episodes, approximately satisfying the i.i.d. assumption.

A second problem the deep Q-network faces is that both terms of its loss function, the prediction $Q(s_t, a_t; \phi)$ and the target $r_t + \gamma \max_a Q(s_{t+1}, a; \phi)$, involve the neural net. It is a departure from the supervised learning framework, where the targets are part of the dataset and fixed throughout training. In practice, having the same parameters in both the prediction and the target will lead to feedback loops that amplify mistakes made by the Q-network and lead to divergence. The authors fix the problem by keeping a separate set of parameters ϕ', called the target network, that are synchronized with the learned parameters ϕ, called a policy network, every N steps. The benefit of improved stability comes at the cost of a slow training speed. Hence, the value of the hyper-parameter N is important.

2.2 Atari 2600

A crucial part of the impact of DQN is its application to a problem domain with a huge state space that is challenging for human players. The Arcade Learning Environment is a suite of 55 games with an API ready to be used to test planning or learning systems [4]. Atari games pose challenges on the perception front–an agent needs to recognize the player character, obstacles, enemies, etc. from pixels–as well as in terms of the reward assignment problem. For example, the gap between

	Freeway
DQN (ours)	31.35
Random (ours)	0
DQN [3]	30.3
Human [3]	29.6
Random [3]	0

Table 1: Average game scores over 100 episodes.

selecting a good action in Breakout (hitting the ball) and receiving a reward (breaking a brick) could be around 100 time steps. A successful reinforcement learning agent needs to propagate the rewards back through time to identify successful actions.

3 Implementation Details

In our implementation, we make use of OpenAI Gym [5], a Python wrapper for the Arcade Learning Environment [6]. Gym provides a simple python interface for agents to interact with environments, and also provides wrappers that automate several of the preprocessing steps necessary to faithfully reproduce the results of the original DQN paper.

Specifically, each Atari video frame is transformed into a grayscale, 84x84 pixel image. The agent will choose an action only once per 4 frames, repeating the previous action at other frames. Finally, each agent state is a tensor of 4 consecutive frames.

We create a simple ring buffer to store "experiences," with each experience containing a source state, an action, a resulting state, a reward, and a boolean flag of whether the resulting state was terminal.

We use a Huber loss with a $\delta = 2$:

$$\text{Huber}(x, y, \delta) = \begin{cases} \frac{1}{2}(y-x)^2 & \text{if } |y-x| \leq \delta \\ \delta \cdot |y-x| - \frac{1}{2} \cdot \delta^2, & \text{otherwise} \end{cases}$$

For the purposes of debugging our implementation and tracking the progress of agent training, we use Tensorboard to track metrics such as the total reward per episode, episode length, and temporal difference error. We view agent play using Jupyter or MoviePy.

4 Implementation Difficulties

As with some other deep learning applications, one of the main difficulties of training DQN is the tight coupling between model components, which makes it difficult to test model components in isolation. Furthermore, the DQN model's training is highly sensitive to hyperparameters, requiring significant trial-and-error for debugging.

5 Results

We provide our re-implementation of a subset of the results from Extended Data Table 2 in [3]. Here, we compare the average episode rewards of our trained method against authors' previously published values.

6 Related work

Prior to the DQN paper, there had been few successful applications of reinforcement learning to real-world problems. Three notable exceptions are performing aerobatics with an RC helicopter [7], playing the game of Backgammon [8] and winning the RoboCup competition [9] (robots playing soccer). Even though the Backgammon projects and other subsequent approaches used neural networks linear function approximators were more popular due to their convergence guarantees [10].

After the deep Q-network paper, the combination of deep neural nets and reinforcement learning has been successfully applied to many different areas, such as robotics (e.g. [11, 12]) and conversational agents (e.g. [13]). Perhaps most notably, a deep reinforcement learning method defeated the world champion in the game of Go, which was previously considered a grand AI challenge [14].

There have been many improvements to the vanilla deep Q-network that speed-up its training. To name a few, a prioritized replay buffer biases the replay buffer sampler towards experiences the deep Q-network makes mistakes on [15], double Q-learning mitigates value overestimation error by using the policy network to select actions for the Bellman equation update [16], and dueling networks increase the accuracy of the estimated Q value by decomposing it into a state value and a state-action advantage function [17].

References

[1] Richard Bellman. "A Markovian Decision Process". In: *Journal of Mathematics and Mechanics* 6.5 (1957), pp. 679–684. ISSN: 00959057, 19435274.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN: 978-0-262-19398-6. URL: `https://www.worldcat.org/oclc/37293240`.

[3] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[4] Marc G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents (Extended Abstract)". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. 2015, pp. 4148–4152.

[5] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[6] M. G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47 (June 2013), pp. 253–279.

[7] Pieter Abbeel et al. "An Application of Reinforcement Learning to Aerobatic Helicopter Flight". In: *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*. 2006, pp. 1–8.

[8] Gerald Tesauro. "Temporal Difference Learning and TD-Gammon". In: *Commun. ACM* 38.3 (1995), pp. 58–68.

[9] Yong Duan, Qiang Liu, and XinHe Xu. "Application of reinforcement learning in robot soccer". In: *Engineering Applications of Artificial Intelligence* 20.7 (2007), pp. 936–950. ISSN: 0952-1976.

[10] J. N. Tsitsiklis and B. Van Roy. "An analysis of temporal-difference learning with function approximation". In: *IEEE Transactions on Automatic Control* 42.5 (1997), pp. 674–690.

[11] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2016).

[12] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436.

[13] Jiwei Li et al. "Deep Reinforcement Learning for Dialogue Generation". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*. 2016, pp. 1192–1202.

[14] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nat.* 529.7587 (2016), pp. 484–489.

[15] Tom Schaul et al. "Prioritized Experience Replay". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016.

[16] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100.

[17] Ziyu Wang et al. "Dueling Network Architectures for Deep Reinforcement Learning". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 2016, pp. 1995–2003.