REINFORCEMENT LEARNING ALGORITHMS FOR PLAYING ATARI ASSAULT

ARTIFICIAL INTELLIGENCE IN PROCESSES AND AUTOMATION

Kristóf Horváth
Computer Science for Autonomous Systems
Eötvös Loránd University
Budapest, Egyetem tér 1-3, 1053
hkristof03@gmail.com

ABSTRACT

Early stages of deep learning led humanity to breakthroughs in computer vision and speech recognition. It took some time until researchers were able to combine deep learning with reinforcement learning as a result of inherent differences between the two approach. In this report I present an improvement on Q-learning and two algorithms that achieve or beat the scores of a human expert player on Atari games.

1 Introduction

The first deep learning model that could learn reinforcement learning control policies directly from sensory input was a convolutional neural network. It was trained with a variant of Q-learning[1]. The method was applied to Atari 2600 games without any adjustment of the parameters. Researchers found that it outperformed previous approaches that mostly relied on hand-crafted data and also surpassed human experts on some games[1]. It was already known that Q-learning overestimates action values. Double Q-learning proved that such overestimations are common and harm performance and it led to much better performance[2].

2 Atari Assault

Assault was created by a game developer company based out of Asia, called Bomb in 1983. Bomb developed the game for the Atari 2600 home video game console, developed and produced by Atari Inc. Assault is a battle game where the player fights against an alien mother ship which continually deploys smaller ships. The ships shoot missiles at the player and also deploy land units, which attack the player from left or from right side. The player aim is to eliminate the opposition while dodging its fire. The action space of the environment consists of moving left, right, shooting left, right and up. A frame from the game is shown on Figure 1.

3 Methods

In the following section I present a side-effect that emerges from Q-learning and I discuss an algorithm that generally solves it. Afterward, I demonstrate an approach to approximate the Q function when the state space of the environment is large and table descriptors are not suitable to use. This approach serves the basis of deep reinforcement learning algorithms, which are discussed in the remaining part of the section.

3.1 Double Q-learning

Q-learning follows the greedy policy given the current action values, which means it selects the action with the maximum Q value. A significant positive bias occurs when the maximum of the over estimated values is implicitly used as an estimate for the maximum value [3]. In case when there are several actions a corresponding to a state s whose true values $q(s,a)$ are all zero but whose estimated values, $Q(s,a)$ are uncertain and thus distributed above and below zero, the maximum is positive and a positive bias occurs[4]. This is called *maximization bias*. Q-learning uses the same samples to determine the maximizing action and to estimate its value. Double Q-learning eliminates the maximization bias with two independent estimates $Q_1(A)$ and $Q_2(A)$, both are estimates for q_*. One estimate, for example Q2 is used to choose the maximizing action $A^* = \text{argmax}_a\, Q_2(a)$ and $Q1$ is used to estimate the value of the maximizing action $Q_1(A^*) = Q2(\text{argmax}_a\, Q_1(a))$. In this case, the estimated value of the maximizing action is unbiased in the sense that $\mathbb{E}[Q_1(A^*)] = q(A^*)$. This process can be reversed by swapping Q_2 and Q_1 to get a second unbiased estimate $Q_1(\text{argmax}_a\, Q_2(a))$. Double-Q learning updates only one of the estimates at each episode. The update that Double-Q learning performs is defined by (1).

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, \underset{a}{\text{argmax}}\, Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)] \tag{1}$$

There is an equal probability to update one of the estimates at each iteration. To update $Q_2(S_t, A_t)$ the two Q functions are swapped in (1). Both Q tables can be used to choose an action according to the $\epsilon - greedy$ policy by taking the average of the two action-value estimates.

3.2 Nonlinear Function Approximation

Reinforcement learning control algorithms can approximate value functions represented as a parameterized functional form with weight vector $\mathbf{w} \in \mathbb{R}^d$. An artificial neural network can approximate $Q_(S, A, \mathbf{w}) \approx Q_*(S, A)$, where $\mathbf{w}$ can be the vector of connection weights of all layers. Typically the dimension of the weight vector is much less than the number of states ($d \ll |S|$), and a change in on element of the weight vector changes the estimated value of many states. A change in one state generalizes to various other states, which is powerful but also more difficult to manage and understand.

3.3 Deep Q-Networks

Initial versions of deep learning (DL) algorithms required large amount of hand-labelled data for efficient training. Supervised learning assumes direct association between inputs and targets to be found. Most of these algorithms assume that the data is independent and has a fixed underlying distribution. However, reinforcement learning must be able to learn from experience, which consists of a scalar reward signal that is frequently sparse, noisy and delayed. The delay between actions and resulting rewards can be thousands of timesteps long, therefore it is considerably harder to find associations. The sequences of states are highly correlated and the data distribution changes over time. The first approach that utilized both reinforcement learning and deep learning in the same application solved these problems with a variant of the Q-learning algorithm, called Deep Q-Network (DQN)[1]. DQN was developed to play Atari games with the same neural network architecture and hyperparameters used for each game. DQN exploits a non-linear function approximator, a convolutional neural network called the Q-network to process raw video data. To overcome the previously mentioned differences between RL and DL, DQN uses a mechanism called *experience replay*. This technique stores the agent's experience at each time step, $e_t = (s_t, a_t, r_t, s_{t+1})$ in *replay memory*, from where random samples are drawn and Q updates are performed on them. After experience replay the agent selects and takes an action according to the ϵ-greedy policy. Traditional reinforcement learning algorithms estimate the action-value function by using the Bellman equation as an iterative update in which case Q_i converges to Q^* as the number of iterations tends to infinity. This approach is not practical, because it estimates the value-function separately for each sequence, without generalization. DQN uses the Q-network with weights θ to estimate the action-value function, $Q(s, a; \theta) \approx Q^*(s, a)$. The loss function that is used for training the Q-network changes at every iteration i, according to (2).

$$L_i(\theta_i) = \mathbb{E}_{s,a \approx \rho(.)}[(y_i - Q(s, a; \theta_i))^2] \tag{2}$$

In this equation, for iteration i the target is $y_i = \mathbb{E}_{s' \approx \epsilon}[r + \gamma\, \text{argmax}_a\, Q(s', a'; \theta_{i-1})|s, a]$ and ρ is a probability distribution over sequences s and actions a, which is called the behaviour distribution. The original DQN algorithm was improved and new versions use two separate networks with the same architecture, a target and an online network[5]. The earlier is called target because it is used to approximate the targets for DQN and the later is called online, because its weights are updated during training. The online network is used to approximate the action values while the target

network computes the target y_i. The target network weights θ^- are not affected by the steps of the algorithm and are periodically replaced by the weights of the online network at every τ steps, so that $\theta_i^- = \theta_i$. The loss function is differentiated with respect to the weights of the online network, according to (3).

$$\nabla L_i(\theta_i) = \mathbb{E}_{s,a\approx\rho(.);s'\approx\epsilon}[(r + \gamma \operatorname*{argmax}_a Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)] \tag{3}$$

Stochastic gradient descent is used to optimize the loss function instead of computing the full expectations. DQN solves reinforcement learning tasks without a model of the environment's dynamics, therefore it is model-free. It is off-policy, because it follows the behaviour distribution but learns about the greedy strategy $a = \operatorname{argmax}_a Q(s, a; \theta)$. In the Q-network architecture there is a separate output unit for each possible action. The outputs correspond to the predicted Q-values for the input state.

3.4 Double Q-learning

The Double-Q-learning (DDQN) algorithm aims to solve the maximization-bias of the DQN algorithm, which arises by using the same set of weights θ_t to select and evaluate an action. As a result, DQN overestimates the action values and learns overoptimistic policies[2]. Equation (4) describes the cause of the maximization bias that results from the DQN target term.

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \theta_t); \theta_t) \tag{4}$$

The DDQN algorithm uses two set of weights θ and θ' separately to determine the greedy policy and to determine its value. The max operation is decomposed into action selection and evaluation. The online network is used to evaluate the greedy policy and the target network estimates its value. The target update for the DDQN algorithm is defined by Equation (5).

$$Y_t^{DoubleQ} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \theta_t); \theta_t^-) \tag{5}$$

The selection of the greedy action is the result of the online weights θ_t and the second set of weights $\theta_t^{'}$ is used to evaluate the value of the current policy. The roles of the two set of weights are symmetric and therefore they can be swapped in Equation (5). Research proved that instability can be related to off-policy learning with function approximation [6–8]. The empirical results of the DDQN algorithm present reduction in the overestimations and improvement in the stability of the learning process.

4 Experiments

In the following section I demonstrate experiments that were conducted according to the original DQN paper with slight modifications in the architecture of the Q-network. Each frame of the Atari Assault game has a shape of (250,160,3) such as height, width and RGB channels. Each frame was preprocessed by cropping out useless information and converting it to grayscale. One preprocessed frame is shown on Figure 2.

Every experiment used the same parameters for the ϵ-greedy policy as illustrated on Figure 3. The probability to select a random action exponentially decays from 0.99 and its minimum value is fixed at 0.01. As it is reflected, the probability of exploration significantly reduced at half of the episodes and the agent is required to exploit its environment. I used the same network architecture that was proposed by the DQN paper, once with additional Batch Normalization(BN) layers after each Convolutional and Dense layer except the last one. BN aims to reduce the so called *internal covariance shift* and tries to fix the distribution of inputs by rescaling the data batches to have a mean of zero and a standard deviation of one [9]. Generally BN accelerates the training and improves its stability. According to the DQN paper the averaged reward plots are noisy and they give the wrong impression that the DQN algorithm does not make a steady progress, which I confirm on Figure 4. The estimated action-value function Q, which provides an estimate of how much discounted reward the agent can obtain by following its policy from any given state is a more stable metric. For this reason, at the beginning of each experiment I saved a fixed set of states generated by ϵ-greedy policy with maximum randomness. At the end of each episode, I took the average of the maximum Q value predicted by the Q-network for each frame. Figure 5 shows stable improvements regarding the Q values for both configurations. However, according to the DDQN paper these are highly overestimated values. Papers [1, 2] state that DQN and DDQN were trained on 10.000.000 and 50.000.000 frames, respectively. Based on the rough average of 800 frames per episode I approximate that it took at least 10 days to train even DQN, which is considerably less training time that I had.

Figure 1: A frame from Atari Assault-v0.

.

Figure 2: Preprocessed frame of Atari Assault-v0. Useless information is cropped out and the frame is converted to grayscale.

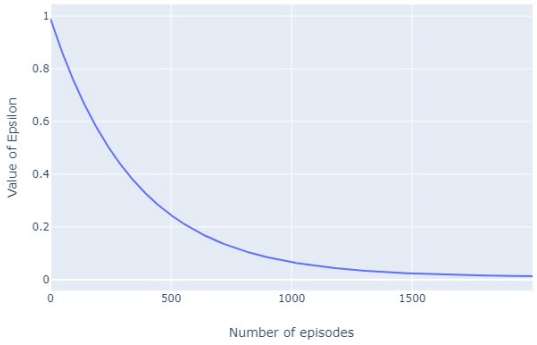

Figure 3: Epsilon greedy policy

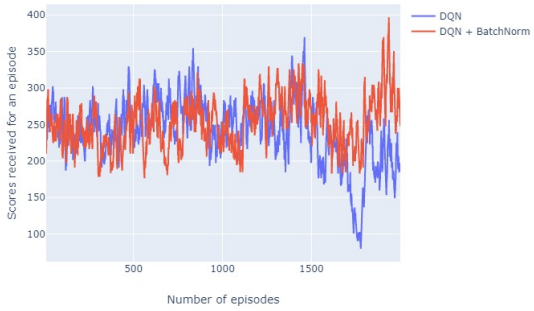

Figure 4: Moving average of the cumulative rewards the agent receives.

.

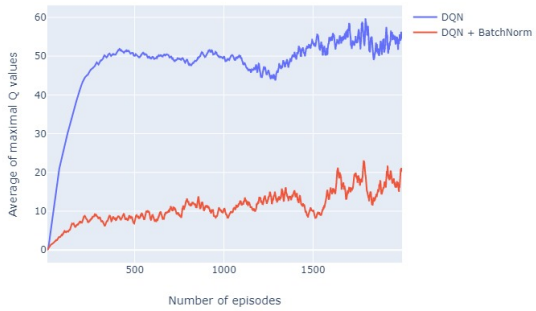

Figure 5: Average of the Q values predicted by the Q-network on a fixed set of states after each iteration.

5 Conclusion

In this report I presented an improvement on Q-learning that generally eliminates the detrimental consequences of the maximization bias. Deep Q-learning connects deep learning and reinforcements learning by solving the emerging hurdles with experience replay. The DQN algorithm reached human level scores on several Atari games alone. The DDQN algorithm revealed that DQN suffers from overoptimistic policies and it provides a great improvement by building upon the Double Q-learning algorithm. The experiments I ran on the Atari Assault game confirm the original DQN results.

References

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

(2) Van Hasselt, H.; Guez, A.; Silver, D. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016; Vol. 30.

(3) Hasselt, H. Double Q-learning. *Advances in neural information processing systems* **2010**, *23*, 2613–2621.

(4) Sutton, R. S.; Barto, A. G., *Reinforcement learning: An introduction*; MIT press: 2018.

(5) Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature* **2015**, *518*, 529–533.

(6) Baird, L. In *Machine Learning Proceedings 1995*; Elsevier: 1995, pp 30–37.

(7) Sutton, R. S.; Szepesvári, C.; Maei, H. R. In *NIPS*, 2008.

(8) Sutton, R. S.; Mahmood, A. R.; White, M. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research* **2016**, *17*, 2603–2631.

(9) Ioffe, S.; Szegedy, C. In *International conference on machine learning*, 2015, pp 448–456.