Extending DQN with Double Q-learning and Prioritized Experience Replay

Merckx, Yannick
Vrije Universiteit Brussel

ABSTRACT

Prioritized Experience Replay and Double Q-learning are separately known to improve the performance of Deep Q-networks (DQN). How do those adjustments perform separately and combined. In this paper, we tried to answer these questions by implementing both extensions on an existing DQN framework and empirically evaluating the performance on two Atari Games, Krull and Boxing. We show that BLA BLA BLA

1. INTRODUCTION

Prioritized Experience Replay [Schaul et al., 2015] and Double Q-learning [Van Hasselt et al., 2016] are separately, two known adjustements to Deep Q-networks (DQN), who are known to sometimes improve the performance. A Deep Q-network is a reinforcement learning algorithm that achieves human-level performance across many Atari Games. [Mnih et al., 2015] The goal of reinforcement learning [Sutton and Barto, 1998] is to learn good policies for sequential decision problems, by optimizing a cumulative future reward signal. One of the most popular reinforcement learning algorithms is Q-learning [Watkins and Dayan, 1992]. One of the drawbacks of Q-learning is that it is known for occasionally learning extremely high action values. This is caused by the maximization step over the estimated action values, which tends to prefer overestimated to underestimated values. DQN is using the Q-learning algorithm and also tends to be overoptimistic. Double Q-learning is a reinforcement learning algorithm that can replace the Q-learning algorithm in the DQN and reduces the overoptimistic behaviour [Van Hasselt et al., 2016]. Prioritized experience replay improves the performance in a total different way. With online reinforcement learning, updates the agent incrementally his parameters, while it is observing the stream of experiences. Reinforcement learning is known to be unstable or even to diverge when nonlinear function approximations. And since DQN is using a neural network to represent the action-value function , DQN was suffering instability. The two main reason of this instability were the close correlation between the updates and inability to remember rare and interesting experiences. [Mnih et al., 2015] resolved this issues by adding *Experience replay* [**?**] to the DQN. Prioritized replay makes this experience replay more efficient and effective by prioritizing the selection of transitions. Where with the original DQN, the experiences were uniformly selected from the memory. Now the main of question for this research is how does the DQN perform is both adjustments are implemented. We will try to answer this question by testing it empirically on two Atari games, Krull and Boxing. Before we continue with the Empirical results, we will first discus the implementation details of the adjustments in the following sections.

2. BACKGROUND

When we want to solve a sequential decision problem, we can learn to estimate the optimal value for each action. This optimal value can be described as the maximum discounted future reward. Formally we describe the true value of an action a in state s, when using a certain policy π by:

$$\mathbb{Q}_\pi(a,b) \equiv \mathbb{E}[R_1 + \gamma R_2 + ... \ |S_0 = s, A_0 = \alpha, \pi] \quad (1)$$

where $\gamma \in [0,1]$ is the discount factor, which allows us to express more importance to immediate rewards than later rewards. Then, the optimal value is $Q_*(s,a) = max_\phi Q_\phi(s,a)$ The optimal policy is a greedy policy, which selects the highest action value for each state.
Q-learning [Watkins and Dayan, 1992] allows use to learn these optimal action values. Because it is for most interesting problems too large to calculate all action values, but we can try to approximate the optimal values by a parameterized value function $(s,a;\theta_t)$. The update function of the parameters follows the standard Q-learning update:

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta_t))\nabla_{\theta_t} Q(S_t, A_t; \theta_t) \quad (2)$$

with R_{t+1} the immediate observing reward, after taken action A_t in state S_t. α is here the learning rate and Y_t^Q the target, which is defined as:

$$Y_t^Q \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t) \quad (3)$$

Note that, there is a big resembles in the update rule of θ and stochastic gradient descent. [Mnih et al., 2015] saw this resembles also during the development of the Deep Q-network and used this to implement Q-learning together with neural networks.

3. DEEP Q-NETWORKS

A Deep Q-network (DQN) is a multilayer neural network that uses end-to-end reinforcement learning [Mnih et al., 2015]. As earlier mentioned, we see a lot of similarities between equation 2 and stochastic gradient descent. In DQN is the parameter θ from equation 2 represented by the weights

of the network. Note that, Reinforcement learning is known to be unstable for non linear approximations of the action values. This is also the case for DQN, since it is a neural network. [Mnih et al., 2015] came up with two smart ingredients, which resolved the issue and dramatically improved the performance. The first ingredient is experience replay, which will be discussed in section 6. The second one is the use of a target network. The target network is just the same online network, only with older parameter values θ_-. The parameter values θ_- are periodically updated with the parameter values θ of the online network and are kept fixed for a certain amount of steps. This causes, together with the experience replay, for a decorrelation for the experiences and resolves the instability. Because of the target network is the target of DQN slightly different:

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-) \tag{4}$$

4. DOUBLE Q-LEARNING

Double Q-learning [**?**] decouples the evaluation and selection of an action. Standard Q-learning sometimes result in extremely high action value. Together with the max operation in equation 4 and 2 are the overestimated action values selected and tends the Standard Q-learning to be overoptimistic.[Van Hasselt et al., 2016]

The decoupling of the selection happens introducing two parameters θ_t and $\theta_t^{'}$. The target for Double Q-learning can defined as:

$$Y_t^{DQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \theta_t); \theta_t^{'}) \tag{5}$$

The first weights θ_t are the online weights, which are used to select the action. The second weights θ_t^t are used to evaluate the selected action more fairly. Note that, θ_t can be θ_t^t updated symmetrically by switching the roles of the two weights after a certain amount of time.

5. DOUBLE DQN

Double DQN solves the issues of overoptimistic behaviour by using DQN with Double Q learning. In section 4, we saw that decoupling the selection and evaluation of the action value requires two separate weights. Further did we see in section 3, that DQN is already in the possession of a second set of weights, namely the target network. This makes is very easy to integrate double Q-learning in DQN. We define the target $Y_t^{DoubleDQN}$ as:

$$Y_t^{DoubleDQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \theta_t); \theta^-) \tag{6}$$

The second weights $\theta_t^{'}$ of equation 5 are replaced by the weights of the target network θ^-. Note that, θ^- still gets periodically updated by the online weights (as in the original DQN) and not symmetrically, as suggested in section 4.

6. PRIORITIZED EXPERIENCE REPLAY

Prioritized Experience Replay is a different approach on fetching the experiences from the replay memory. DQN uses experience replay to break the correlation between the experiences. This is realised by saving all transitions and fetching them uniformly from the replay memory. The problem with this approach is that rare experience are forgotten and a lot of non-informative transitions are replayed.With Prioritized Experience replay we prioritize the transitions and replay them based on their prior. [Schaul et al., 2015] discovered that prioritizing the transitions, makes the experience replay most of the time more efficient and effective.

6.1 Prioritizing with TD-error

The measure for prioritizing every transition is essential to let this method work well. As a measurement to calculate the priors, we choice the temporal difference error (TD-error). The idea behind taking this measurement is the fact that the experiences with the biggest TD-error are to most 'interesting' to replay. The TD-error δ_j for the transition j of a vanilla DQN can be described as:

$$\delta_j = R_j + \gamma_j \max_a Q(S_t, a) - Q(S_{t-1}, A_{t-1}) \tag{7}$$

As we can see is this a very convenient measure, since we already need to calculate it for the DQN. Although, there are some issues when using the TD-error. First of all updating the TD-errors. To update the TD-error, we need to sweep the entire replay memory. As a solution, we suggest to update only the TD-error, when the experience is replayed. Note that, as a consequence experiences with a low TD-error may take a long time before they get replayed. Another issue, slow convergence and the lack of diversity , when taking a greedy replay selection.
As a solution for these issues, we use Stochastic Prioritization.

6.2 Stochastic Prioritization

Stochastic Prioritization is an alternative selection method for the greedy selection method. This is also the sampling method, we will use during our experiments. Based on a certain on a certain probability, each sample will be picked from the replay memory. We define the probalibity for sample i by:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{8}$$

with p_i the priority of sample i and α determining the importance of the priority.

The priority can be represented in many ways. We represent it rank-based:

$$p_i = \frac{1}{rank(i)} \tag{9}$$

where $rank(i)$ is the rank of instance i in the sorted replay memory, according to $|\delta|$.

Important to note is that these stochastic updates introduce bias. We correct this by introducing importance-sampling (IS) weights. [Schaul et al., 2015]

$$w_i = (\frac{1}{N} . \frac{1}{P(i)})^\beta \tag{10}$$

This correction is applied in the Q-learning update rule, where we use weighted deltas $w_i\delta_i$ instead of regular delta's $delta_i$

Now, we know how Prioritized Experience Replay and Double Q learning are implemented in a DQN structure, we can continue with the experiments.

7. ATARI EXPERIMENTS

The main question of the paper is how do these adjustments influence the performance of the DQN. We will answer this question by empirically evaluating the adjustments separately and combined on two Atari Games. We take as a baseline the performance of the vanilla DQN on the Atari games.

7.1 Experimental Setup

The two Atari games, that will be learned are Boxing and Krull. All experiments are ran 5 times on the Hydra cluster with 16gb for each run. The network architecture for all experiments is the same and is identical to the network used in [Mnih et al., 2015].

7.1.1 Hyper-parameters

In all the experiments are the same hyper-parameters used. The parameter values will be based mainly on the hyper-parameters of the tuned DQN in [Mnih et al., 2015]. The discount will be set to $\gamma = 0.99$, and the learning rate to $\alpha = 0.00025$. The number of steps between target network updates will be $\tau = 10000$. The training steps will be set to 20 million. The agent will be evaluated every X steps. The size of the experience replay memory will be 100000 transitions. The minibatch size is 32 and the history length and action step both 4 steps. At last we have ϵ, which is at the start 1 and is linearly decreasing to 0.1.

7.2 Empirical Results of Double DQN

7.3 DQN with Prioritized Experience Replay

7.4 Empirical Results of Prioritized Double DQN

7.5 Side Note

We know that the experimental setup is not ideal. 5 runs for each experiment and low amount of total steps makes the results less accurate. The reason is the limited time frame., which caused a lot of trouble. The initial idea was to conduct the experiments with the hyper-parameters of the original paper by [Mnih et al., 2015]. Soon it became clear that an experiment with a memory-size of 1 million and 50 million steps, would not finish in time. As suggested, we tried to reduce the frame size to 42x42, but this resulted in almost no earning for several Atari games. Eventually after a lot of side-experiment, we ended up with the configuration suggested in 7.1.1 and the two Atari games Krull and Boxing. Both of these games are selected based on their fast learning rate and good performance in a limited configuration. The limited time frame for this project had also an impact on the amount of runs. Ideally we liked to conduct 10 runs or more for each experiment. This meant that we had to run 80 runs at the same time on the hydracluster. Unfortunately, this seemed to be not possible, because or the UserRAM-Limit. Also balancing the runs a bit more and only assign 16gb to experiments that needed it, dit not help. This resulted in several runs getting randomly killed after a few days. Eventually we succeed to create a stable experimental environment by doing only 5 runs per experiment.

8. DISCUSSION

9. REFERENCES

[Mnih et al., 2015] 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. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

[Schaul et al., 2015] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[Van Hasselt et al., 2016] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100.

[Watkins and Dayan, 1992] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.